Bioinformatics tool development with a focus on structural bioinformatics and the analysis of genetic variation in humans

A thesis submitted in fulfilment of the requirement for the degree

of

DOCTOR OF PHILOSOPHY
IN BIOINFORMATICS

of

RHODES UNIVERSITY, SOUTH AFRICA

Department of Biochemistry and Microbiology
Faculty of Science

by

David K. Brown

September 2017

Abstract

This thesis is divided into three parts, united under the general theme of bioinformatics tool development and variation analysis. Part 1 describes the design and development of the Job Management System (JMS), a workflow management system for high performance computing (HPC). HPC has become an integral part of bioinformatics. Computational methods for molecular dynamics and next generation sequencing (NGS) analysis, which require complex calculations on large datasets, are not yet feasible on desktop computers. As such, powerful computer clusters have been employed to perform these calculations. However, making use of these HPC clusters requires familiarity with command line interfaces. This excludes a large number of researchers from taking advantage of these resources. JMS was developed as a tool to make it easier for researchers without a computer science background to make use of HPC. Additionally, JMS can be used to host computational tools and pipelines and generates both web-based interfaces and RESTful APIs for those tools. The web-based interfaces can be used to quickly and easily submit jobs to the underlying cluster. The RESTful web API, on the other hand, allows JMS to provided backend functionality for external tools and web servers that want to run jobs on the cluster. Numerous tools and workflows have already been added to JMS, several of which have been incorporated into external web servers.

One such web server is the Human Mutation Analysis (HUMA) web server and database. HUMA, the topic of part 2 of this thesis, is a platform for the analysis of genetic variation in humans. HUMA aggregates data from various existing databases into a single, connected and related database. The advantages of this are realized in the powerful querying abilities that it provides. HUMA includes protein, gene, disease, and variation data and can be searched from the angle of any one of these categories. For example, searching for a protein will return the protein data (*e.g.*

protein sequences, structures, domains and families, and other meta-data). However, the related nature of the database means that genes, diseases, variation, and literature related to the protein will also be returned, giving users a powerful and holistic view of all data associated with the protein. HUMA also provides links to the original sources of the data, allowing users to follow the links to find additional details.

HUMA aims to be a platform for the analysis of genetic variation. As such, it also provides tools to visualize and analyse the data (several of which run on the underlying cluster, via JMS). These tools include alignment and 3D structure visualization, homology modeling, variant analysis, and the ability to upload custom variation datasets and map them to proteins, genes and diseases. HUMA also provides collaboration features, allowing users to share and discuss datasets and job results.

Finally, part 3 of this thesis focused on the development of a suite of tools, MD-TASK, to analyse genetic variation at the protein structure level via network analysis of molecular dynamics simulations. The use of MD-TASK in combination with the tools developed in the previous parts of this thesis is showcased via the analysis of variation in the renin-angiotensinogen complex, a vital part of the renin-angiotensin system.

Declaration

I declare that this thesis is my own, unaided work, unless otherwise stated. It is being submitted for the degree of Doctor of Philosophy at Rhodes University. It has not been submitted before for any degree or examination in any other university.

David Brown

[illegible]

This 9th day of September 2017

Acknowledgements

I would like to acknowledge the following people for their contributions to this work:

First and foremost, I would like to acknowledge my supervisor, Prof. Özlem Taştan Bishop, who made this work possible. Without her guidance, encouragement, and strong leadership, this would surely never have completed.

Secondly, to the members of the RUBi lab, especially Rowan Hatherley, who helped develop the 'bio' side of my bioinformatics skill set, and David Penkler, for the numerous discussions and 'collaborations'.

Thirdly, to everyone who contributed to the various publications that make up this thesis. Without these contributions, there would be no thesis.

And lastly, to Canan Atilgan and the members of her lab in Istanbul, thank you for hosting me and sharing your knowledge.

Personal acknowledgements:

I would like to acknowledge the support provided by my family and friends. Although not necessarily academic in nature, it was just as important.

Thank you to my parents, Ken and Lynne Brown, as well as my brothers, Graeme and Bruce, for their support, but also for constantly badgering me to finish up. It served as great motivation!

Secondly, I need to thank my friends for their support and comradery in the lab and at the Rat & Parrot, when times were tough (special mention to Rowan Hatherley).

Lastly, and most importantly, I would like to thank my girlfriend, Tayla Waterworth, who kept me motivated right up and until the end. Without her love, support, and tea, this thesis my never have been completed.

Acknowledgment of funding:

This work is supported by the National Institutes of Health Common Fund under grant number U41HG006941 to H3ABioNet.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Description
ACL	Access Control
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CADD	Computer-Aided Drug Design
CGAS	Candidate Gene Association Studies
CWL	Common Workflow Language
CLI	Command-Line Interface
CPU	Central Processing Unit
CRUD	Create, Read, Update, and Delete
CSS	Cascading Style Sheet
DBMS	Database Management System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
GWAS	Genome Wide Association Studies
H3Africa	Human Heredity and Health in Africa
H3ABioNet	H3Africa Bioinformatics Network
HPC	High Performance Computing
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HUMA	Human Mutation Analysis web server and database
IaaS	Infrastructure as a Service
JMS	Job Management System
JSON	JavaScript Object Notation
MD	Molecular Dynamics
MSA	Multiple Sequence Alignment
MTV	Model-Template-View
MVC	Model-View-Controller
NGS	Next Generation Sequencing
NHGRI	National Human Genome Research Institute
ORM	Object Relational Mapper
OS	Operating System
PaaS	Platform as a Service
PCA	Principal Component Analysis
PV	Protein Viewer
PBS	Portable Batch System
PDB	Protein Data Bank
RAM	Random Access Memory
REST	Representational State Transfer

RPM	Redhat Package Manager
RUBi	Research Unit in Bioinformatics
SaaS	Software as a Service
SLURM	Simple Linux Utility for Resource Management
SNV	Single Nucleotide Variant
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WMS	Workflow Management System
WWW	World Wide Web
XML	Extensible Markup Language
YAWL	Yet Another Workflow Language

Research Outputs

Conference Poster Presentations

1. David Brown, Rowan Hatherley, and Özlem Taştan Bishop. "HUMA: a web server and database for the analysis of genetic variations in humans". ECCB, France, 2014.
2. David Brown, David Penkler, Thommas Musyoka, and Özlem Taştan Bishop. "JMS: a workflow management system for high performance computing (HPC) clusters". ISCB-ECCB, Dublin, 2015.
3. David Brown, Rowan Hatherley, and Özlem Taştan Bishop. "HUMA: a web server and database for mining and analysing variation in humans". ISCB-ECCB, Dublin, 2015.
4. David Brown and Özlem Taştan Bishop. "JMS: Creating and running complex computational pipelines on high performance computer clusters". Biophysics in the Understanding, Diagnosis and Treatment of Infectious Diseases, South Africa, 2015.
5. David Brown and Özlem Taştan Bishop. "HUMA: a platform for the analysis of genetic variation in humans". H3Africa 9th Consortium Meeting, Mauritius, 2016.
    - Placed 2nd in the poster competition

Conference Oral Presentations

1. David Brown, Rowan Hatherley, and Özlem Taştan Bishop. "HUMA: A platform for the analysis of genetic variation in humans". SASBI-SAGS, South Africa, 2014.
2. David Brown and Özlem Taştan Bishop. "HUMA: a platform for the analysis of genetic variation in humans". H3ABioNet AGM, South Africa, 2015.
3. David Brown and Özlem Tastan Bishop. "HUMA: a platform for the analysis of genetic variation in humans". SASBMB, South Africa, 2016.

- Won the BSP prize for the best bioinformatics talk

4. <u>David Brown</u> and Özlem Tastan Bishop. "HUMA: a platform for the analysis of genetic variation in humans". H3Africa 9th Consortium Meeting, Mauritius, 2016.
5. <u>David Brown</u> and Özlem Tastan Bishop. "JMS: Creating and running complex computational pipelines on HPC clusters". CHPC National Meeting, South Africa, 2016.

Publications

1. Hatherley, R., <u>Brown, DK</u>., Musyoka, T., Penkler, D., Faya, N., Lobb, KA., and Taştan Bishop, Ö. **SANCDB: A South African Natural Compound Database**. *Journal of Cheminformatics*, **7:29**, 2015.
2. <u>Brown, DK</u>., Penkler, DL., Musyoka, T., and Taştan Bishop, Ö. **JMS: An Open Source Workflow Management System and Web-Based Cluster Front-End for High Performance Computing**. *PLoS One,* **10 (8): e0134273**, 2015.
3. Hatherley, R., <u>Brown, DK</u>., Glenister, M., Taştan Bishop, Ö. **PRIMO: An Interactive Homology Modeling Pipeline.** *PLoS One,* **11 (11): e0166698**, 2016.
4. Mulder, N., *et al.* **H3ABioNet, a sustainable pan-African bioinformatics network for human heredity and health in Africa.** *Genome Research,* **26 (2): 271-277**, 2016.
5. <u>Brown, DK</u>., and Taştan Bishop, Ö**. Role of Structural Bioinformatics in Drug Discovery by Computational SNP Analysis**. *Global Heart,* **12 (2): 151-161**, (2017).
6. <u>Brown, DK</u>., Amamuddy, OS., and Taştan Bishop, Ö. **Structure-based analysis of single nucleotide variants in the renin-angiotensinogen complex**. *Global Heart,* **12 (2): 121-132**, (2017).
7. Mulder, N., *et al.* **Development of Bioinformatics Infrastructure for Genomics Research in H3Africa**. *Global Heart,* **12 (2): 271-277**, (2017).

8. Brown, DK., Penkler, DL., Amamuddy, OS., Ross, C., Atilgan, AR., Atilgan, C., and Taştan Bishop, Ö. **MD-TASK: a software suite for analyzing molecular dynamics trajectories**. *Bioinformatics,* **33 (17): 2768-2771**, (2017).
9. Brown, DK., and Taştan Bishop, Ö**. HUMA: a platform for the analysis of genetic variation in humans**. *Human Mutation*, **in press,** (2017).

Thesis Motivation and Overview

The motivation behind this thesis was to develop tools and services that enable bioinformatics research, specifically research into the effects of genetic variation on protein structure and function. Additionally, this work was motivated by the need to enable this kind of work in under-resourced, African countries where it was previously not feasible. With the advent of big data in bioinformatics, biologists are drowning in ever-larger quantities of data, and new tools must be developed to cater for this. The project aims to cater for this need, especially in the context of these low-resource countries, where universities may not be able to afford powerful high-performance computing centers. This was done by providing tools that; 1) enable software developers to more easily develop and release their own tools; 2) enable researchers to use these tools from anywhere in the world with an internet connection; and 3) collect and aggregate data in meaningful ways so that it can be used to provide insight into protein function and how it relates to disease.

This thesis is made up of 10 chapters divided into four parts. The research conducted for this thesis is covered in the first three parts, each of which builds on top of the next, while part 4 simply consists of the conclusion and references.

Part 1 of the thesis is made up of chapters 1 – 3 and covers the design and development of the Job Management System (JMS), a workflow management system and cluster front-end for HPC computing. Additionally, chapter 3 covers the development of tools and workflows that are housed in and executed via JMS, including a homology modeling pipeline and variant analysis workflow.

Part 2 is made up of chapters 4 – 6 and describes the Human Mutation Analysis (HUMA) web server and database. HUMA aggregates data from various existing databases and provides tools to visualize and analyze this data. To execute tools on the underlying HPC infrastructure, HUMA makes use of JMS, the workflow management system described in part 1.

Part 3 consists of two chapters. Chapter 7 focuses on the development of MD-TASK, a tool suite to analyze molecular dynamics simulations using network analysis techniques, perturbation response scanning, and dynamic cross-correlation. Chapter 8 follows this up by providing a case study in which the tools developed in parts 1 and 2 of this thesis are used in conjunction with MD-TASK to analyze the renin-angiotensinogen protein complex, which plays a vital role in the renin-angiotensin system.

All three parts of this thesis are united under the common theme of bioinformatics tools development with a focus on the analysis of variation. JMS provides tools and workflows to this end, HUMA combines these tools and workflows with data to provide a complete analysis platform, and MD-TASK can be used in conjunction with these tools to carry out a complete project based around the analysis of variation at the protein structure level.

Part 1: Development of a web-based cluster front-end and workflow management system

1. Bioinformatics in the modern age of computing

1.1. High Performance Computing

High Performance Computing (HPC) has become a staple of modern computational science. Computational science, also referred to as scientific computing, can be defined as a field at the intersection of modeling scientific processes and using computers to produce quantitative results from these models [1]. It is used in a broad range of fields, including, but not limited to, cosmology and astronomy, meteorology, seismology, computational chemistry, and of course, bioinformatics. Amongst other things, it involves the development of algorithms and software for big data analysis, computer simulations, and the modeling of scientific processes to make predictions about future situations. These sorts of calculations often require a large amount of computational power and, to be practical, are carried out on supercomputers or computer clusters [2].

The term, HPC, refers to the use of powerful computers and efficient parallel processing techniques to execute tasks in a far shorter time period than would normally be expected. There are three main models for achieving HPC capability, namely, cluster computing, grid computing, and cloud computing [3–7]. All of these models have their own advantages, but essentially provide the user with more computing power in terms of the number of Central Processing Unit (CPU) cores, Random Access Memory (RAM), and storage available. Simply put, more CPU cores means that more processes can be executed in parallel, more RAM means that more data can be accessed at an increased speed, and more storage means that more data from computational experiments can be stored and backed up.

1.1.1. Models for HPC

Cluster computing is arguably the most common form of HPC. A computer cluster is created by networking a collection of machines together so that they can work in tandem to carry out computationally intensive operations *i.e.* the workload for a job is shared amongst computers in the cluster. Individual machines in a cluster, called nodes, are hidden from the user and their resources are pooled to create what is in essence a single powerful machine [3–5]. A cluster can offer an affordable alternative to purchasing a dedicated supercomputer as small-scale clusters can be created from standard desktop computers and laptops. Clusters are located at a single geographic location and are usually owned by a single organization. They are tightly coupled systems, running homogeneous hardware and software. Job management and scheduling as well as resource management is centralized. They have high network bandwidth and low latency and, as such, are well suited to handle jobs that require a large amount of inter-process communication [4].

Grid computing aggregates the resources of multiple, geographically distributed computers [4]. As such, a grid can be made up of multiple computer clusters. Grids can be formed between multiple organizations who pool their computing resources. They are loosely coupled systems with distributed job management and scheduling and heterogeneous hardware and software. Different locations in the grid are usually linked via the Internet. As such, a grid environment has high latency and low bandwidth, making it unsuitable for jobs that require high inter-process communication. On the other hand, grid computing is well-suited to large jobs that can be broken down into small, standalone chunks [4].

A major advantage of grid computing is that it can be made up of a heterogeneous array of machines that may already exist in an organization, thereby negating the need to purchase new hardware. Grid environments are also extremely modular, with fewer points of failure. A side effect of this is that upgrading parts of the grid can be done without requiring any downtime [4]. Grids can also grow very large as they can be made up of the pooled resources of multiple organizations.

On the other hand, having geographically distributed nodes and different administrative domains makes grid environment more difficult to manage. Heterogeneous hardware and software adds to this burden. Geographical distance between nodes is also one of the main reasons for the high latency, low-bandwidth connections mentioned earlier, which means slow communication speeds between nodes [5]. As such, the strengths of grid computing – its distributed nature and heterogeneous environment – are also behind its weaknesses.

Cloud computing is the newest of the HPC computing paradigms. Generally, when we refer to cloud computing, we are referring to applications that are being provided as a service via the Internet as well as the hardware and software platforms that are providing these services [3]. Cloud computing services are accessed across the Internet without regard for the underlying infrastructure *i.e.* the data centers that host these services. Cloud service providers charge consumers based on usage of their services and can scale the resources provided to the user up or down as required. In this model, computing can be seen as the 5th utility and is delivered in a similar way to more traditional utilities such as water, electricity, gas and telephony, where consumers are charged based on usage and do not, themselves, own the underlying infrastructure [3,8]. Cloud computing can be broken down into three types of services [3]:

1. Software as a Service (SaaS) – users make use of applications hosted by the service provider.

2. Platform as a Service (PaaS) – service providers provide an integrated environment where users can build test and deploy their own applications.
3. Infrastructure as a Service (IaaS) – users make use of processing power, storage, networking, and other computing resources to set up their own environments in the cloud.

Advantages of cloud computing include enormous cost savings in terms of both time and money. The consumer is not required to purchase and set up the expensive infrastructure required to create their own data centers. They also do not have to worry about maintaining and upgrading the infrastructure over time. This is all handled by the service provider while the consumer simply pays for usage. Clouds also eliminate the complexity of setting up and managing the infrastructure and services. In addition to this, cloud computing provides data redundancy and sharing and the convenience of being accessible from any machine, anywhere in the world via the Internet [3,4].

Disadvantages of cloud computing include the need for a constant and fast Internet connection, the fact that many cloud-based applications do not have all of the features of their traditional counterparts, and the loss of direct control of infrastructure and services. In addition to this, there are legal and regulatory issues that must be taken into consideration. As governments become more interested in cloud computing, some are working to develop regulations to monitor and ensure privacy and security of data. Depending on where a user's data is being stored, it may fall under different jurisdictions making it difficult to know what regulations and laws apply. In addition, users may not know where their data is being stored [9].

1.1.2. Use of HPC in bioinformatics

As it has with other computational fields, HPC has become a crucial part of bioinformatics. Advances in the fields of biochemistry and molecular biology coupled with improvements in HPC and computational modeling have facilitated discoveries in a number of areas including drug discovery, systems biology and genome research [10]. As such, a large amount of effort has been devoted to designing and developing new techniques and algorithms that take advantage of available HPC resources.

In structural bioinformatics, computationally intensive tasks such as homology modeling, molecular docking, and molecular dynamics can be executed in parallel and scale nicely with increasing numbers of CPU cores. As such, they are well suited to being executed on a cluster, grid, or in the cloud.

Homology modeling, a structural bioinformatics technique, allows users to model the structure of a protein for which there is no experimentally determined structure. Obtaining accurate predictions can be a challenging process, both intellectually and computationally. Over the years, a number of tools have been developed to tackle this challenge including MODELLER [11,12], Robetta [13], SWISS-MODEL [14–16] and YASARA. In the case of MODELLER, a user-selected number of models are generated for a given protein sequence. Each one of these models is generated independently and, as such, can be generated in parallel. If each model is generated in parallel on a separate core, the time taken to generate all the models can be described by:

Overall time = M*T/C

Where M is the number of models to generate, T is the average time taken to generate a single model, and C is the number of cores used to execute the job in parallel. From this equation, it can be seen that the more cores available, the faster the job will be executed. Thus, the benefit of performing homology modeling on a cluster, where there are potentially hundreds or even thousands of CPU cores, becomes obvious. To illustrate this, both Robetta and YASARA make use of distributed computing platforms, Robetta@Home and Models@Home[17] respectively. By installing this software, any user in the world can become part of a massively distributed network of computers. When the user's computer is idle for a certain amount of time and the screensaver is displayed, it becomes active in the network and can start receiving tasks from Robetta or YASARA. Using this model, these tools have been able to perform an amount of work that would otherwise have been unfeasible.

Molecular docking simulations are used to predict the conformation of a receptor-ligand complex [18]. Simulations are most commonly used for virtual screening during structure-based drug discovery. In such a scenario, the receptor is usually a protein and the ligand is usually a small molecule. Molecular docking involves repeatedly docking the ligand to the receptor to find the conformation with the lowest binding energy. This conformation will hopefully mimic the natural interaction between the receptor and ligand. During virtual screening, a library of compounds is docked against one or more receptors in order to find compounds with the highest binding affinity [19]. Compound libraries can consist of thousands or even millions of compounds. As such, HPC platforms are required to screen these libraries in a reasonable time frame. High throughput implementations of docking tools such as AutoDock [20] have been developed to cater for this. Although AutoDock is a single-threaded application, there have been a number of attempts to parallelize it [21–23]. The lab responsible for developing AutoDock eventually released their own parallel version, AutoDock Vina [24], which produced a twofold speed up, while retaining accuracy [25]. Frameworks such as DOVIS [26] and MOLA [27] for running AutoDock easily and efficiently on Linux clusters have also been developed. All of this effort serves to show the perceived importance of using HPC to speed up the virtual screening process.

Molecular dynamics (MD) simulations are probably one of the most demanding tasks in computational biology in terms of CPU usage [10] and are generally not viable on a desktop computer using CPUs. As such, cluster computing has been successfully employed to speed up these simulations. Recently, work has been done to execute molecular dynamics simulations on clusters that use graphical processing units (GPU) as opposed to CPUs to perform calculations [28–30]. Implementations of MD applications, AMBER [28] and GROMACS [31], with GPU acceleration have produced speed ups in excess of fivefold what can be achieved with CPUs alone.

Outside of structural bioinformatics, HPC has also been employed to speed up Next Generation Sequencing (NGS) analyzes such as variant calling pipelines, Genome Wide Association Studies (GWAS), and genome annotation [10,32]. The sequencing of dozens of genomes or hundreds of exomes produces large datasets and analyzing these datasets on normal hardware is not plausible. To cater for these large datasets, tools such as MegaSeq [33] use the MapReduce [34] programming paradigm. Using MapReduce, data is split into manageable chunks that can be processed in parallel. The results of each parallel process are then aggregated at the end to produce the overall result. As such, MapReduce has been widely used to analyze large data sets on clusters [35] and is particularly well suited to bioinformatics [36].

The above examples are only a small sample of the uses of HPC in bioinformatics. Other uses could even include speeding up less computationally intensive applications so that they could provide users with results in real-time [10].

1.1.3. Challenges of HPC in bioinformatics

In the previous sections, the benefits of HPC and its uses in bioinformatics were made clear. Unfortunately, the vast majority of researchers are still not able to take advantage of these benefits.

There are two main reasons for this. Firstly, the costs involved in setting up a computer cluster are still too high. Although small-scale clusters can be set up that take advantage of existing lab machines, these cluster may not have enough computational power to perform more demanding tasks such as molecular dynamics or NGS analysis. Purchasing more powerful hardware to set up a dedicated cluster is expensive and possibly inefficient, especially for smaller labs that might not be able to provide a sustained workload for the cluster. Fortunately, cloud computing may be able to offer a solution in this regard, as users are able to rent computational power in the cloud and only pay for the amount used [37].

The second challenge is the steep learning curve associated with using HPC resources. Computer clusters are normally accessed via a command-line interface (CLI). Resource managers or Job Schedulers such as the Portable Batch System (PBS), Torque or the Simple Linux Utility for Resource Management (SLURM) are used to submit, manage, and schedule jobs on the cluster. These tools are made up of a suite of command-line utilities that users must master to take advantage of HPC. To do this, users must first become familiar with the CLI, which itself provides a steep learning curve. For biologists, biochemists, or any non-computational scientists, this can be an intimidating task to undertake and a serious barrier to entry for HPC. That many CLI programs written for science are not user-friendly also does not help to improve this situation [38]. In some cases, even scientists who have some CLI experience may feel too intimidated to attempt learning the cluster. Aside from learning the command-line utilities, users must learn how to set up job scripts, which contain the commands that will be submitted to the cluster along with certain directives for requesting resources and setting up job meta-data. Users must also understand certain concepts. For example, when submitting jobs, their job will not be running on the machine that they are submitting from and so will not have access to files on the local filesystem. Jobs on the cluster will usually only be able to access files and data stored in shared storage. As such, users must prepare the data before submitting the job to ensure it is available across all nodes on the cluster. Another example of this is that software that is available on the local node may not be available on the cluster or needs to be loaded via the use of environment modules. All these factors provide stumbling blocks for users who are using a cluster for the first time and may frustrate even experienced computational scientists. Busy scientists often do not want to dedicate the required time to teach themselves these concepts and end up forgoing HPC, which, ironically, would likely have saved them time in the long run.

1.2. Workflow Management Systems

Computational pipelines or workflows are made up of a series of tasks, which run sequentially or in parallel and whose order is determined by data dependencies [39]. Each task consumes data from input files or previous tasks and produces data for follow-on tasks [40]. Workflows have emerged as an effective and increasingly popular tool in science for automating large-scale data analysis and calculations [41,42]. Workflows define the analysis steps to be taken out, manage the execution of those steps (often on distributed resources), and collect the outputs. They also allow analysis to be reproduced by eliminating human error that could crop up in intermediate steps [43].

Workflow Management Systems (WMS), also called Workflow Composition Systems, allow users to assemble components (software tools) into computational pipelines and can be either user-directed or automatic [44].

User-directed WMSs require users to directly design and edit the workflow themselves either via a graphical user interface (GUI), as is the case with Kepler [45] and Galaxy [46], or via workflow languages such as Extensible Markup Language (XML), the Common Workflow Language

(CWL) [47], Nextflow [48], and Yet Another Workflow Language (YAWL) [49]. Graphical interfaces are more user-friendly compared to workflow languages, but often offer less flexibility. Workflows composed via a GUI must also be converted to a workflow language to be transferred or stored [44].

Automatic WMSs are more challenging to develop as they automatically compose workflows based on high-level user requirements such as what input the user will provide and what the expected output is at the end of the workflow [44]. These systems are ideal for large workflows where manual composition is time-consuming.

1.2.1. Use in bioinformatics

Over the past two decades, the field of bioinformatics has embraced WMSs wholeheartedly. With workflows being part of almost every kind of bioinformatics analysis, bioinformaticians have been a driving force behind the advancement of WMSs. In this time, two main categories of WMSs have emerged. The first of these is based on building workflows from remotely located web or grid services. One example of this is Taverna (although Taverna is also able to execute local scripts) [50,51]. Taverna links to thousands of web services provided by a number of suppliers including the EBI, NCBI and BioMart [52]. Users can pick from predefined workflows or create their own by combining services on a drag-and-drop workflow design panel. Other WMSs in this category include Discovery Net [53], Biowep [54], and Triana [55]. The main advantage of using distributed services is that most of the computation is performed remotely. This means that tools and databases do not need to be installed on the local machine and expensive hardware does not need to be purchased to be able to run these tools. Disadvantages include the need for a stable and fast Internet connection, lack of control over the remote services, and inherent unreliability.

The second category consists of systems that execute tools and workflows locally on the machine or cluster that they are set up on. These systems usually come prepackaged with several tools, while also allowing users to add more by installing the tool on the local server or cluster and writing a configuration file describing how the tool should be executed. Adding new tools to these systems can be challenging. Examples of this type of system include Galaxy [46], Kepler [45], Anduril [56], and Ergatis [57].

The first of these systems, Galaxy, was initially built for the analysis of genomics data and is arguably the most popular bioinformatics WMS. As with Taverna, Galaxy provides a drag-and-drop workflow design panel. More interestingly, however, Galaxy allows users to produce workflows based on their job history [58]. Users analyze a given dataset tool by tool via the Galaxy interface. While they do this, Galaxy stores a comprehensive job history including all the input files, parameters, and outputs of each tool. Once the user has finished analyzing the dataset, they can generate a workflow from the stored job history. The generated workflow can then be used to reproduce results of the analysis at a later stage as well as perform the analysis on different datasets. This focus on reproducibility has set Galaxy apart from other systems.

There are also WMSs that fall outside of these categories. Tavaxy [59] is able to incorporate workflows from both Galaxy and Taverna. The result is an integrated environment where users can take advantage of both WMS categories to develop hybrid workflows. Other WMSs such as COSMOS [60], a python library focused on the parallelization of tasks in a workflow, sacrifice ease of use for more power and efficiency. Workflows are defined using Python functions that support the MapReduce programming paradigm.

The importance of WMSs in bioinformatics is evidenced by the myriad of solutions available, some of which have been mentioned above. However, despite these efforts, there remains room for improvement.

1.2.2. Limitations of existing systems

The problem of creating workflows from component tools has largely been solved. Tools like Galaxy and Taverna provide GUI-based workflow editors that allow users to design workflows by dragging tool components onto a canvas and assigning data dependencies. On the other hand, the issue of adding tools to the WMS in the first place is still an area that can be improved upon. For example, with Galaxy and Ergatis, users are required to write complicated configuration files in XML describing how the tool would be run from the command line. These configuration files are not intuitive to create, requiring a certain degree of system administration expertise. In addition, adding tools will generally require administrator privileges. As such, users who may simply want to add their own custom tools to the system must request an administrators assistance.

Secondly, although WMSs often run in an HPC environment, they provide no visible means of interacting with the underlying resource manager. When a job is submitted, users cannot see where in the queue the job is or how busy the underlying cluster is. This information can be useful to users who want to gain an idea about when their jobs will be completed.

Lastly, researchers who develop workflows often want to make those workflows public via their own websites. Current WMSs do not offer any easy way of making tools and workflows public via external interfaces. Although some provide web services that could technically be used for this purpose, they do not always expose all the required information. For example, a WMS might expose a service that allows the user to submit input parameters for a tool in order to execute it.

An interface made for this tool may initially work, but could be broken by an update that changed what parameters the tool required. To cater for this, the developer of the external interface would have to update their interface each time the tool was updated.

1.3. Research motivation

HPC is an important resource for conducting any form of computational science, including bioinformatics. Developments in the field of HPC have facilitated advances in areas including drug discovery, systems biology and genome research. With time, the need for accessible HPC resources will become even greater as huge amounts of data are generated from NGS projects around the world. As the amount of data grows, the computational power required to analyze this data increases as well. Unfortunately, HPC resources are difficult to use for non-expert users and, as such, remain out of reach for most scientists. To cater for this, several WMSs have been built that offer graphical interfaces, usually web-based, and abstract away the complexity of using the underlying HPC resources. These systems allow users to run tools and create computational pipelines to analyze their data from start to end on a cluster or in the cloud. Unfortunately, they still have limitations.

In some cases, the level of abstraction has become too high, and users are unable to interact directly with the underlying HPC resources of the cluster. For example, WMSs often do not offer a means of viewing where a user's job is in the queue of the underlying job scheduler. Additionally, a user who simply wants to submit a custom script to the cluster has no way to do this via the WMS, and must instead deal with the complexities of the CLI. Administrative users are also unable to manage the underlying resource manager settings or configure queues and nodes. As such, existing WMSs offer an easy way of running tools and workflows on an HPC cluster, but no means of managing or monitoring the status of the cluster.

Over and above this, most WMSs lack flexibility when it comes to adding custom tools as they require administrator permissions, the writing of complicated configuration files, or both. This results in a barrier to entry for researchers that may want to add their own tools and scripts to the system.

Lastly, existing WMSs provide no easy means for making tools and workflows publicly accessible via external interfaces. For example, researchers often want to make their tools available via their own custom web interfaces. Although some WMSs provide web services, which could technically allow a user to access the tool or workflow via an external interface, these services have not been expressly designed for this purpose and can be cumbersome to use.

To overcome these issues, it would be greatly beneficial to have a hybrid system, which combines the features of a WMS with those of an HPC cluster front-end. Such a system would overcome the limitations of existing WMSs and expose the full power of HPC to non-expert users.

1.4. Research aims & objectives

The focus of this work is to develop software for biologists and bioinformaticians that will ease the burden of using and managing HPC resources. The project will combine the functionality seen in existing WMSs with cluster monitoring and management functionality to create a complete and unique solution for executing tools and workflows on an HPC cluster.

The specific objectives for this work are as follows:

1. Develop a web-based front-end for an HPC cluster that is compatible with any job scheduler or resource manager.
2. Incorporate workflow management functionality, including functionality to create, upload and edit tools directly via the web interface.

3. Provide access control for tools, workflows, jobs, and HPC resources so that they can be shared between users.
4. Develop functionality to allow semi-automated external interfaces to be developed for tools and workflows.
5. Populate the system with bioinformatics tools and workflows.

2. JMS design & development

2.1. Introduction

The Job Management System (JMS) has been developed as a web-based cluster front-end and workflow management system. Initially built as a module within the Human Mutation Analysis (HUMA) web server [61], which is the topic of part 2 of this thesis, JMS was designed with the simple purpose of receiving job requests from the HUMA interface, executing them on the cluster, and returning the results. It soon became apparent that JMS was a useful system for allowing jobs to be executed on our cluster via a web interface, and that this system could be used in any web server, rather than just HUMA. As such, it was decided that the JMS module would be split into its own project, whose purpose it would be to service job requests from any of our web servers as well as any web servers that our group might develop in the future. By doing this, we would negate the need to "reinvent the wheel" each time a new web server was developed that required access to the cluster. The JMS module was, thus, extracted from the HUMA project and further developed as a standalone web server. It is, however, still used by HUMA as well as number of our other bioinformatics web servers, which will be discussed in later chapters.

The rest of this chapter is dedicated to describing the design, development and features of JMS. The work presented in this chapter has been published in a research article [62] and a second manuscript is planned.

2.2. Design rationale

When splitting JMS from HUMA, we were confronted with two options on how to proceed. As JMS was initially developed as a module within HUMA, we could continue to develop it as a module, which would then simply be copied into new web servers to give them access to JMS functionality. The second option was to convert it into a standalone web server that could expose the JMS functionality via a set of web services. This option had several advantages over the former. Firstly, the JMS module was developed using Django [63], a Python web framework. As such, it would only be compatible with other Django-based web servers. By setting JMS up as a stand-alone web server and exposing its functionality via a set of web services, it became language and framework agnostic, which means that any web server written using any technology or language can interact with it.

Secondly, having the JMS code in a central location makes it easier to affect changes, updates, and bug fixes. If each web server that requires JMS functionality were to have its own JMS module, each and every module would need to be updated when, for example, a bug is discovered or a new feature is added. By setting up a centralized JMS web server, updates only need to be made at a single location. This also prevents different web servers from running different versions of JMS.

Lastly, separating JMS into a stand-alone web server meant that we could also develop a user interface for it. Where interaction with the module was entirely programmatic, the JMS web server allows users to login via a web interface to manage and run jobs. This also means that, in addition to JMS being a tool to manage jobs submitted from other servers, it could become a fully-fledged WMS, where users could add their own tools and scripts, formulate workflows, and execute and manage jobs. Functionality allowing administrators to manage, monitor, and configure the cluster was also built into the interface.

Due to the advantages discussed above, it was decided that JMS would be developed as a standalone web server. This configuration means that JMS caters to four groups of people. Firstly, it caters to developers of outside web servers, who want to allow their web servers to execute tools or workflows on a cluster. This was the initial purpose of JMS and means that these developers do not need to "reinvent the wheel" by building their own systems for submitting and managing jobs on a cluster.

Secondly, JMS caters to tool and workflow *developers* by providing an interface to easily create and manage tools and organize them into complex workflows. These tools and workflows can be executed via the JMS interface or incorporated into external web servers as mentioned above. Tool and workflow management is discussed again in section 2.7.

Thirdly, JMS caters to tool and workflow *users*. This group of users require no programming, scripting, or command-line expertise. Instead, JMS generates a web interface for tools and workflows created by developers. Users simply need to select the tool or workflow of their choice and fill in the required inputs via the web interface. In our case, these users would be biologists or even bioinformaticians with little or no programming or CLI experience. JMS allows these users, who previously may have been intimidated by the prospect of using HPC, to submit jobs to a cluster as easily as filling in details on a web form.

Lastly, JMS caters to system administrators by providing a simple means for them to manage and monitor the underlying cluster. Cluster management is discussed in more detail in section 2.7.5.

As far as we are aware, JMS is the only WMS that caters to such a diverse group of users. In addition, it is the only WMS designed with the goal of allowing tools and workflows to be executed via external servers.

2.3. Implementation details

JMS was built using a number of technologies and frameworks including the Django web framework [63], the Django REST Framework [64], and Twisted [65], an event-driven Python networking library. Using a web framework such as Django means that JMS is compatible with any Database Management System (DBMS) that is supported by the Django Object Relational Mapping (ORM) tool. During development, JMS was tested with both SQLite [66] and MySQL [67]. In our production environment, MySQL is being used.

2.3.1. Django

Django is a Python web framework that facilitates the development of web-based applications by providing methods to create, edit, and access data in a database and dynamically generate web pages [68]. Django implements the Model-Template-View (MTV) design paradigm, which is similar to the Model-View-Controller (MVC) [69] architecture, and separates the application into three independent components, namely, the model, view, and template layers.

2.3.1.1. Model

Applications built with Django consist of several models, which define the structure of the underlying database. Each Django application contains a *models.py* module that may contain any number of classes, which inherit from Django's base Model class. Each of the model classes maps to a table in the database while the attributes of the model classes map to columns in those tables. These classes are used by the Django ORM to setup the database as well as add, update and delete data in the tables.

2.3.1.2. View

The view layer of a Django application contains the application logic. It receives a request from a user and performs some action based on that request. Practically speaking, a Django application will contain a *views.py* file which contains a set of Python functions (or classes depending on the design). These functions are mapped to Uniform Resource Locators (URLs) and are executed when a user sends a request to one of the mapped URLs (*e.g.* when a user enters a URL into their browsers address bar). Views can store and access data by communicating with the model layer of the Django application and can return web pages via the template layer.

2.3.1.3. Template

The template layer describes what and how data should be displayed. In other words, Django templates provide the layout for dynamically generating web pages. Put together, the functions in the view layer access data via the model classes and use templates to generate the web pages that will display that data.

2.3.2. Django REST framework

Representational State Transfer (REST) is an architectural style used by the World Wide Web (WWW) [70]. The Django REST Framework makes use of this style to expose the functionality of a Django application via a web-based Application Programming Interface (API). Web API's provide programmatic access to services located anywhere on the WWW. Developers access these services by sending Hypertext Transfer Protocol (HTTP) requests to a relevant URL. HTTP is the protocol used for communication on the WWW [71]. It will not be discussed in detail here, but there are a few aspects that need to be explained:

2.3.2.1. HTTP address

All HTTP requests need to be sent to an address on the WWW. The address comes in the form of a URL, which makes up part of the first line of an HTTP message, and is used to locate a resource on the WWW. When using the Django REST framework, this URL will point to one of the classes in the view layer of the Django application.

2.3.2.2. HTTP methods

Along with the resource address, the first line of an HTTP message also contains a method token, which describes what action should be performed on the resource located by the address. In practice, the most commonly used HTTP methods used by web developers are POST, GET, PUT, and DELETE. By convention, these methods should map to Create, Read, Update, and Delete (CRUD) operations respectively.

The POST request method *should* be used to store data on the web server. Data can be sent with the POST request in the body of the message.

The GET request method should be used to retrieve data from the web server. Parameters can be sent in the URL to specify search terms to filter what data should be returned.

The PUT request method should be used to update a resource on the web server. The resource is located based on parameters in the URL and is updated with data sent as part of the request body.

The DELETE method, as its name suggests, should be used to delete a resource on the server. As with the PUT request, the resource to delete can be located via parameters in the URL.

The above descriptions of HTTP methods are simplistic and may not cover all the nuances of the protocol, but they are sufficient for describing how these HTTP methods are used within the JMS application. Other methods such as OPTIONS and HEAD also exist, but are not used explicitly in the application code.

The Django REST framework facilitates the use of HTTP methods by allowing developers to create a function for each HTTP method within the classes in the views layer. As such, the address of an HTTP request points to a class in the views layer and the HTTP method points to a specific function within that class that will be executed when the request is received. These functions accept an HTTP request object as a parameter, which allows the developer to access the body of the request in the function. Parameters defined in the URL are passed to the function as arguments.

2.3.3. Twisted

Twisted is an event-driven Python framework used for developing network applications. It provides libraries for communicating via various network protocols such as HTTP, Transmission Control Protocol (TCP), and User Datagram Protocol (UDP).

The Twisted framework was used in JMS to build the Impersonator (discussed in more detail in section 2.6.1). Twisted was used to set up a lightweight, standalone web server that listens for requests on a single URL and spawns processes as a particular user, either to authenticate that user or to perform a task on the master node of the cluster.

2.3.4. SQLite

SQLite is a lightweight relational DBMS. It is file-based, which makes it portable and cross-platform. A SQLite database can be copied between different machines by simply copying and pasting the database file from one machine to another. This portability and the fact that it is operating system (OS) independent and lightweight has made it popular on mobile platforms such as Android and iOS. SQLite support is also built into Python via the standard libraries, which makes it easy to incorporate into Python frameworks such as Django.

SQLite was used in the initial stages of testing and development of JMS. It had the advantages of being able to quickly and easily deploy a database without having to go through the cumbersome process of setting up a more heavyweight DBMS such as MySQL. Unfortunately, the Django ORM for SQLite does not support updating the database schema after the database's initial creation. As such, the database had to be deleted and recreated each time the schema was updated. During testing and development, this was not an issue, but in production this would result in data loss.

Additionally, SQLite performance does not match process-based systems such as MySQL. However, in a system such as JMS where high levels of concurrency in the database are not required, this was not an issue.

2.3.5. MySQL

MySQL is a process-based relational DBMS, that provides improved performance in comparison to SQLite. As opposed to file-based DBMSs, MySQL runs as a background service that applications can connect to from across a network. As such, a MySQL database does not need to be on the same machine as the application it is servicing.

MySQL is being used as the production database for JMS. Aside from the improved performance and the fact that a MySQL database can be hosted on a separate machine, the Django ORM is able to update the schema of a MySQL database without needing to recreate the database. This means that the database can be modified without losing the data that is already contained within it, which is vital when it comes to installing updated versions of JMS.

2.4. Architecture

The JMS architecture can be described in two ways. Firstly, the *system architecture* (Fig. 2.1 A) describes how JMS fits into the broader HPC environment. Secondly, the *software architecture* (Fig. 2.1 B) describes the actual design of the application.

2.4.1. System architecture

JMS has been designed to fit into an organization's existing HPC environment. JMS is installed as a separate component on the master node of the cluster (Fig. 2.1 A). As such, JMS does not interfere with an existing cluster setup. It communicates with the cluster resource manager using plugins. This means that for JMS to work with a specific resource manager, such as Torque or SLURM, a plugin must be created for it. This plugin architecture can potentially allow JMS to work with any resource manager by allowing developers to create custom plugins (see section 2.4.2.1.2).

JMS can communicate with external interfaces such as desktop or mobile applications or even other web servers via its RESTful web API. Tools and workflows that have been developed and/or stored in JMS can be accessed by external interfaces via this API. In fact, JMS's own interface, described in section 2.4.2.3, makes use of the RESTful web API, which means that all functionality

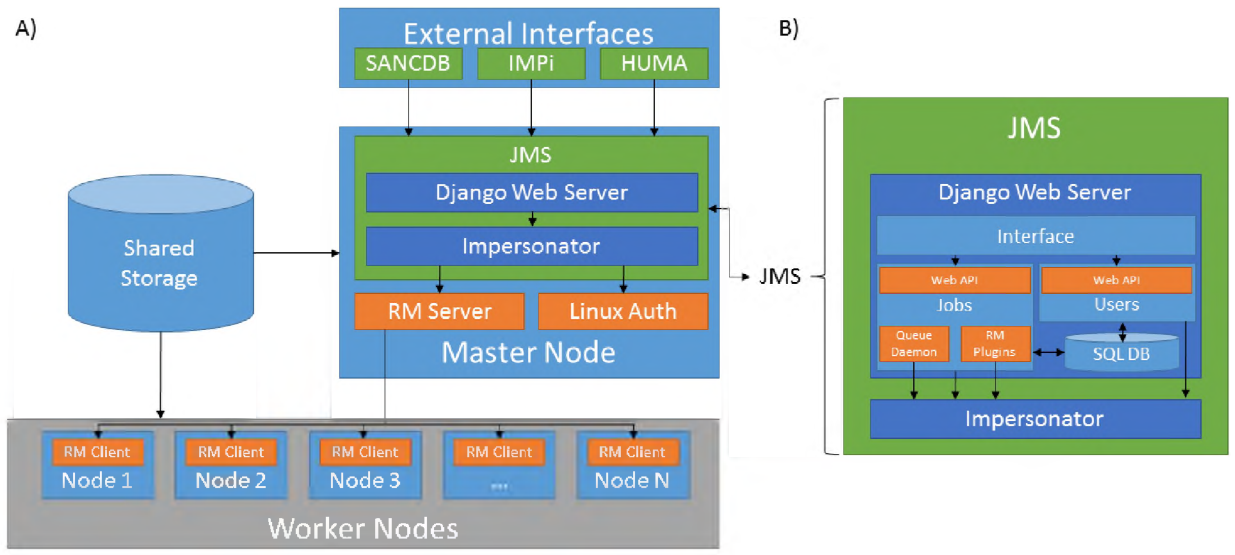


***Fig. 2.1. A)** System architecture - JMS fits into the existing architecture of an HPC cluster. It is installed on the master node of the cluster and communicates with the underlying resource manager via the resource manager plugin and Impersonator. **B)** Software Architecture – The JMS software architecture consists of a Django web server and the Impersonator server. The Django web server consists of 3 modules (the Interface, Jobs, and Users modules) and uses a SQL database for data storage. The Interface communicates with the Jobs and Users modules via their respective web APIs. The Jobs module also contains the queue daemon, for keeping the database up-to-date, and resource manager plugins for interacting with the underlying resource manager.*

provided by the JMS interface is also available to external interfaces, given that they have valid credentials for authentication.

2.4.2. Software architecture

JMS has been developed using the Django web framework. The project consists of three Django apps, namely, the Jobs app, Users app, and Interface app. Each of these modules adopts the traditional three-tiered architecture commonly used in web development consisting of a presentation tier (the template layer), application tier (the view layer), and data tier (the model layer) [72]. The Jobs and Users apps expose their functionality via a RESTful web API, which the Interface app accesses. The Jobs app also includes a background daemon, used to keep the job history up-to-date, as well as the resource manage plugins required to communicate with the underlying cluster. Outside of the Django project, JMS includes what we have termed the "Impersonator server", which is discussed further in section 2.6.1. The project relies on a relational database to store user, job and system details. The software architecture is illustrated in Fig. 2.1 B.

2.4.2.1. Jobs app

The Jobs app is the biggest of the Django apps and contains almost all the functionality specific to JMS. This includes configuring and querying underlying HPC resources to get status information, submitting and managing jobs on the cluster, adding, storing and managing tools and workflows, and access control and collaboration features. Due to its size – the Jobs app is almost 6000 lines of Python code and climbing, compared to the other apps, which are all less than 1000 lines – careful attention had to be paid to the design and structure of the code. Clean, well-structured code is important for future maintenance and development of the system. In addition, it is easier for new developers to join a project if the code is well-structured and easy to understand. As such, a hierarchical structure was created for the Jobs app where the 6000 lines of code were split into approximately 40 cohesive modules. In this context, cohesion refers to the degree to which the elements in these modules belong together. In other words, code that was logically related was placed in modules together. For example, all code that related to adding, updating and managing a tool can be found in the *Tools* module. This logical structure prevented individual modules and classes from becoming too large and difficult to manage. It also resulted in more searchable, readable, and understandable code.

The structure of the Jobs app is illustrated in Fig. 2.2. At the top level, the views module provides web-based access to the Jobs app via a RESTful web API. The views module contains no application logic, simply handling HTTP requests by calling functions in the next layer, receiving the result and returning it in a response to the request. The views module also provides access control (ACL) functionality by accepting or denying requests based on whether the requesting user is authenticated and whether they have access to the resource in question.

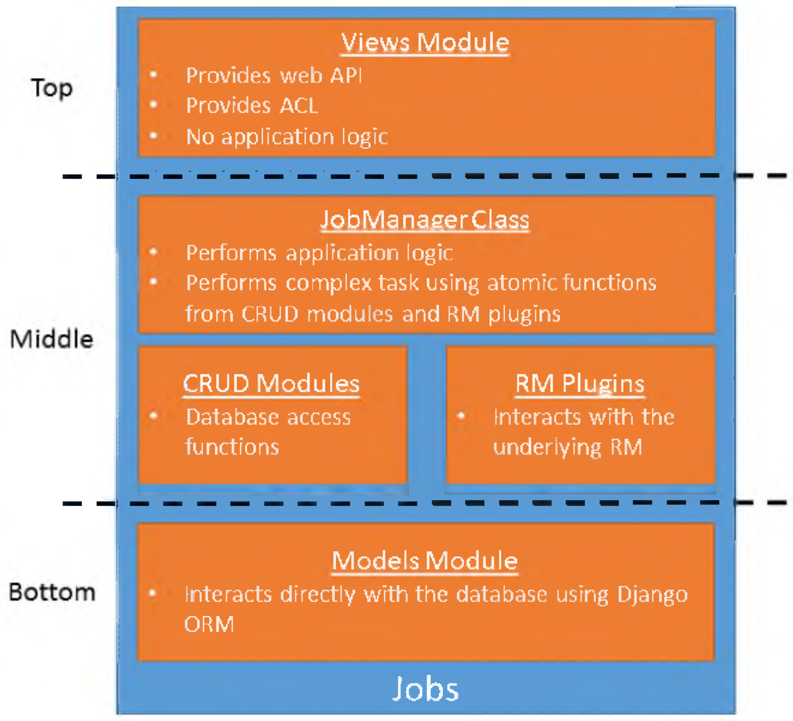

Top
Views Module
Provides web API
Provides ACL
No application logic
Middle
JobManager Class
Performs application logic
Performs complex task using atomic functions from CRUD modules and RM plugins
CRUD Modules
Database access functions
RM Plugins
Interacts with the underlying RM
Bottom
Models Module
Interacts directly with the database using Django ORM
Jobs

Fig. 2.2. The Jobs app – this module consists of three layers. The top layer provides the access points for the module i.e. the web API. The middle layer is responsible for the business logic of the applications. The bottom layer is responsible for interacting the underlying database to store and edit tools and workflows.

The middle level consists of two sub-levels. The top sub-level is home to the JobManager module. This module contains the application logic for the Jobs app and is responsible for receiving requests from the views module and performing a complex task based on the request details. Complex tasks are performed by calling multiple smaller functions from the bottom sub-level.

The bottom sub-level consists of several modules made up of what we will refer to as "atomic functions" - functions that cannot be broken down into smaller components. These functions are either part of the resource manager plugins or the data access/CRUD modules. Resource manager plugins are discussed in detail in section 2.4.2.1.2, but are essentially used to communicate with and query the underlying HPC resources. For example, they contain functions to submit jobs to the cluster or check the scheduler queue. On the other hand, the CRUD modules are used to add, update, delete or read data from the database. To do this, they communicate with the models module, used by Django to interface with the database.

As an example of how these levels work together, consider the scenario of a user running a job on a cluster. The user submits a request from the JMS web interface to run a tool. The request contains the details for the job including a job name and description, the input parameters required by the tool, and any files required by the tool. The views module receives this request and determines whether the submitted user is authenticated and whether they have permission to submit jobs. If the security check is successful, the submitted details are extracted from the request and sent to the *RunToolJob* function of the JobManager module. To perform the complex task of creating and running the tool job, the *RunToolJob* function calls several atomic functions from the resource manager plugin and CRUD modules. Firstly, it sends the job details to the *AddJob* function in the Jobs CRUD module to create an entry for the job in the database. Jobs are made up of one or more stages. Although a workflow job may consist of more than one stage, tool jobs will only ever have one stage. As such, the next atomic function that is called is the *AddJobStage* function of the JobStages CRUD module. Next, the job directory hierarchy is set up on the cluster and the uploaded files are saved in this hierarchy. The input parameters are then saved to the database by calling the *AddParameter* function of the Parameters CRUD module. Lastly, the command required to run the job on the cluster is generated and the *ExecuteJobScript* function of the resource manager plugin is called to run the job. Once the job is running, the job ID is returned to the view module, which in turn returns it to the web interface.

This example illustrates the different purposes of the three levels. The first level acts as a landing platform for requests, the second level contains the application logic, and the bottom level contains the atomic functions used to perform the complex application logic tasks. This architecture aligns closely with the MVC design paradigm.

The Jobs app contains two subsections, which should be described in more detail. The first of these is the queue daemon, which is responsible for keeping job details in the database up-to-date, and the second is the resource manager plugin architecture that has been developed for JMS.

2.4.2.1.1. Queue daemon

Despite technically being a part of the Jobs app, the queue daemon runs as a separate service to the Django web server and can be seen as a separate component. The queue daemon is implemented in the form of a Django management command. Django management commands are accessible via the command line. This allows the queue daemon to be started and stopped via the CLI.

The queue daemon is a simple service that runs in the background to fulfill two purposes. Firstly, it continuously polls the job queue of the resource manager to keep the job details in the database up-to-date. For example, it will check the queue every five seconds and determine whether any jobs have finished, whether they completed successfully or not, and whether any dependent jobs can now be set to run. All the details it picks up are stored in the JMS database so that users can access them via the job history page. In future, these details will also be used to generate graphs and reports detailing how the underlying cluster is being used.

The second function of the queue daemon is to send notifications when jobs complete. These notifications can be sent to users via e-mail or to an external web server in the form of an HTTP request (the use-case for the latter is discussed in section 3.3).

The queue daemon polls the job queue by running a command that returns the status of all jobs currently in the queue. This command is run at user-specified intervals, for example, every 15 seconds, and is dependent on the underlying resource manager. For example, if the Torque resource manager is being used, `` `qstat -x` `` is executed to check the queue. The output of this command is then parsed and used to update the database.

2.4.2.1.2. Resource manager plugins

Resource manager plugins are used to connect JMS to the underlying cluster's resource manager. This gives JMS the ability to interact with the cluster and perform tasks such as submitting jobs, monitoring the job queue, and configuring cluster nodes, queues, and settings.

A resource manager plugin is created by developing a Python wrapper for a cluster's underlying resource manager. By default, JMS uses a plugin for the Torque resource manager, however, plugins can be created for any resource manager. To develop a plugin, three rules must be followed:

1) Plugins must inherit from the *BaseResourceManager* class

The *BaseResourceManager* class provides a standard API that allows JMS to interact with it and any classes that inherit from it. As such, all plugins must inherit from this class.

2) Plugins must override certain functions of the *BaseResourceManager* class

Simply inheriting from the *BaseResourceManager* class is not enough. Plugins must also override certain functions that are specific to the underlying resource manager. These functions generally wrap the command line clients of the resource manager so that they can be used to interact with the rest of JMS.

3) All functions must return a specified object type

Several predefined objects have been created so that JMS can expect a standard format return value from plugin functions. Functions that are overridden because of rule number 2 must still return the same type of object. This is so that JMS is always working with a consistent set of objects, no matter what plugin is being used.

By following these three rules, JMS can work with any plugin without needing a code change. As all plugins make use of a standard API, all their functions have the same signature *i.e.* the same function name, inputs, and outputs. For example, one of the base functions that needs to be overridden is the *ExecuteJobScript* function. This function accepts a string containing the path to the script that must be executed on the cluster, submits the job, and returns the ID of the job if submission was successful. All plugins must override this function, keeping the input and output types the same (the path to the script and the job ID, respectively), but implementing the function specifically for the relevant resource manager. In the case of Torque, the function executes the `*qsub*` command, which takes the path to the script as an argument. In the case of SLURM, the overriding function in the plugin would execute the `*sbatch*` command. However, because both functions would have the same signature, this would not matter to JMS.

The predefined objects that are used for the return values of the overriding functions are essentially generic data structures that can contain any sort of data. Although the data structures are consistent across all plugins, depending on the resource manager, the content being stored in the structures may differ. JMS has been designed to expect these data structures to be returned from the plugins, and does not worry about the content being stored in it. The data structures are then converted to JavaScript Object Notation (JSON), before being returned to the JMS interface, where they are used to populate the interface with content. The data structures can also be used to generate unique interfaces for different resource managers as they allow the data fields, themselves, to be defined. Thanks to this plugin architecture, the JMS interface can adapt to suit which ever resource manager plugin it is using. This is useful, as different resource managers will almost always have different and unique settings and options, which would require unique interfaces to manipulate.

2.4.2.2. Users app

Although small compared to the Jobs app, the Users app is the second biggest JMS app. It is responsible for user creation, management and authentication. As with the Jobs app, the Users app can be accessed via a RESTful web API, exposed by the Users app's own views module. Because of the diminutive size of the Users app, there was no need to create sub-layers as was done with the Jobs app. As such, the application logic was written directly in the views module, which communicates directly with the models module to access the database. This is the standard Django approach and eliminates the need for the middle layer used in the Jobs app.

To authenticate users, a custom authentication backend was developed. This authentication backend communicates with the Impersonator server (see section 2.6.1) to allow users to log in using their SSH credentials *i.e.* users can log in to JMS using the same username and password that they use to log in to the master node of the cluster. This process is discussed in further detail in section 2.6.1.

2.4.2.3. Interface app

The Interface app is responsible for the JMS web interface. As such, it operates at the view and template levels of the MTV design paradigm (as opposed to the other apps, which operated at the model and view levels). Where the other apps respond to requests with data in JSON, the Interface app responds with traditional web pages. These web pages then make use of the functionality of the Jobs and Users apps by interacting with their respective web APIs (Fig. 2.3).

The design philosophy used when building JMS was to minimize the required number of page loads when navigating the interface without allowing any one page to become too large and cumbersome. Where traditional websites need to reload the page every time they send requests to the server, Single Page Applications (SPA) make use of Asynchronous JavaScript and XML (AJAX) requests to dynamically fetch content and update the current page *i.e.*

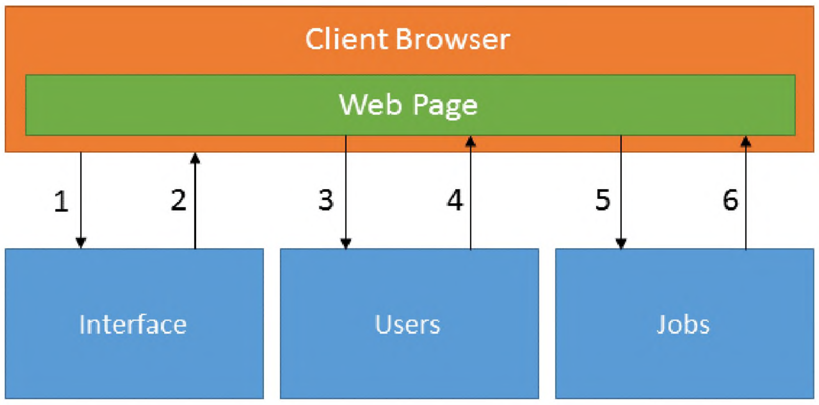


*Fig. 2.3. The Interface app - **1)** The client browser sends a request to the Interface app when the user first attempts to go to the website. **2)** The Interface app responds with a web page. **3)** When fetching user details, the web page loaded by the interface sends an AJAX request to the Users app. **4)** The Users app responds with the requested details. **5)** When fetching job, tool or workflow details, an AJAX request is sent to the Jobs app requesting the relevant details. **6)** The Jobs app responds with the requested details. Once a particular page is loaded, the Interface app is not needed again unless the user wants to move to another page.*

AJAX allows HTTP requests to be sent using JavaScript and without reloading the page. This technique can be used to give the illusion of a multi-paged website, whereas in actual fact, there is only one page that is constantly being updated by fetching, showing and hiding content. In other words, the page configuration is changed to make it appear as if another page has loaded. We will refer to these different configurations as “virtual pages”.

Advantages of SPAs include shorter page load times and lower bandwidth consumption. This is because, instead of fetching the entire webpage, the AJAX requests only fetch the *data* that needs to be displayed on the page. Computational load on the server is also reduced, as web pages do not need to be dynamically generated (using Django templates) with every request. SPAs also provide a better, less jarring user experience as they can display loading animations while they wait for responses from the server. As such, they behave more like mobile or desktop applications than websites.

On the other hand, the initial page load for an SPA can be quite large, as much of the content that would normally be spread out over multiple pages is now contained in a single page. This can also make the page quite cumbersome to work with as a developer. Several techniques have been devised to deal with this, however, including only loading virtual pages when they are required. This results in a slower loading time the first time a virtual page is required, but once that section is loaded, it does not need to be loaded again.

The second issue is that any bugs that crop up during a session could affect the entire application as opposed to a single page. JavaScript errors that are not handled properly may even go so far as to make a webpage unusable. In the case of an SPA, this would render the entire application unusable. Such errors would only arise as the result of a programming error and, as such, are technically not a result of SPA architecture. That said, SPAs can accentuate programming errors by allowing them to affect an entire application as opposed to a single page of the application.

When building JMS, a hybrid approach was chosen where SPA techniques were used in conjunction with a traditional multi-paged website to reduce the overall number of pages without introducing a single point of failure or allowing any one page to become too large and cumbersome. The resultant interface is made up of six actual web pages:

- Login page
- Dashboard
- Tools page
- Workflows page
- Job History page
- Cluster Management page

Of those pages, the dashboard, tools and workflows pages each make use of AJAX and SPA techniques to create the illusion of additional pages. The dashboard, for example, has one virtual page for monitoring the cluster and a second virtual page for monitoring a specific job. Similarly, the tools and workflows pages are made up of three virtual pages each – one displaying a list of all the tools/workflows, one to execute a tool/workflow, and one to edit a tool/workflow. As such, although there are six actual pages, there are 11 virtual pages.

The Interface app is only responsible for loading the web pages. The functions of the JMS system are still performed by the Jobs and Users apps. To perform these functions, the web pages send AJAX requests to the RESTful web APIs of these apps. For example, when a user wants to run a tool, s/he must click on the "Run" button for that tool. When this happens, an AJAX request is sent to the Jobs app to fetch the content required to generate an input page for the tool. While waiting for a response to this request, the webpage will hide the currently displayed *virtual* page, which in this case is the list of tools, and show a loading animation. When the response is received from the server, a virtual page is generated that allows the user to provide inputs for the tool. The loading animation is then hidden and replaced by this new virtual page. To the user, it appears as if a new page has loaded, but the transition is smoother thanks to the loading animation.

All JMS pages, even those that do not make use of virtual pages, use AJAX to interact with the server. For example, when settings are updated on the Cluster Management page, these updated settings are sent to the server via an AJAX request. Without using AJAX request, this would require a page reload. As such, AJAX provides performance improvements even when virtual pages are not involved.

Due to the use of SPA techniques, most of the computational load for the interface is on the client-side. As such, a large amount of JavaScript code was written – over 8000 lines of custom JavaScript, not including 3rd party plugins. In order to keep this code manageable, the Knockout framework [73] was used. Knockout is a pure JavaScript library that uses the Model-View-View Model (MVVM) design pattern to simplify the process of creating and maintaining JavaScript user interfaces (UI). Knockout minimizes the amount and complexity of JavaScript you need to write to keep your UI in-sync with data received from the server.

Finally, being web-based, JMS can be accessed from any device. As mobile devices have gained considerable market share, it was considered important to design the interface in such a way that it would be accessible on these smaller devices. As such, the Bootstrap framework [74] was employed. Like Knockout, Bootstrap is a JavaScript framework designed to help build user UIs.

However, where Knockout focuses on handling data received from the server and using it to update content displayed on a webpage, Bootstrap is focused on the actual design of the page. In other words, it is a combination of JavaScript, Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) used to build reusable UI components. In addition, Bootstrap components are completely responsive [75], which means that they adapt to the size of the screen they are being viewed on. As such, a website designed using the Bootstrap framework will automatically optimize itself for the device it is being used on, whether that device is a smartphone, tablet, or desktop PC.

2.5. Database

The JMS database was designed and built using the Django ORM. As such, JMS is compatible with any DBMS that the Django framework supports. For testing and development, we used SQLite and MySQL.

2.5.1. Design

The JMS database was designed to store the details required to execute tools and workflows on the cluster as well as to keep a detailed history of all jobs that have been run. The database schema is illustrated by Fig. 2.4. Below we discuss some of the decisions that went into this design.

2.5.1.1. Tools

Adding tools to WMSs such as Galaxy requires users to create XML configuration files describing how the tool can be executed. JMS allows users to input these details via an intuitive web interface and stores them in a relational database rather than in an XML configuration file. To store these details requires several tables (Fig. 2.5).

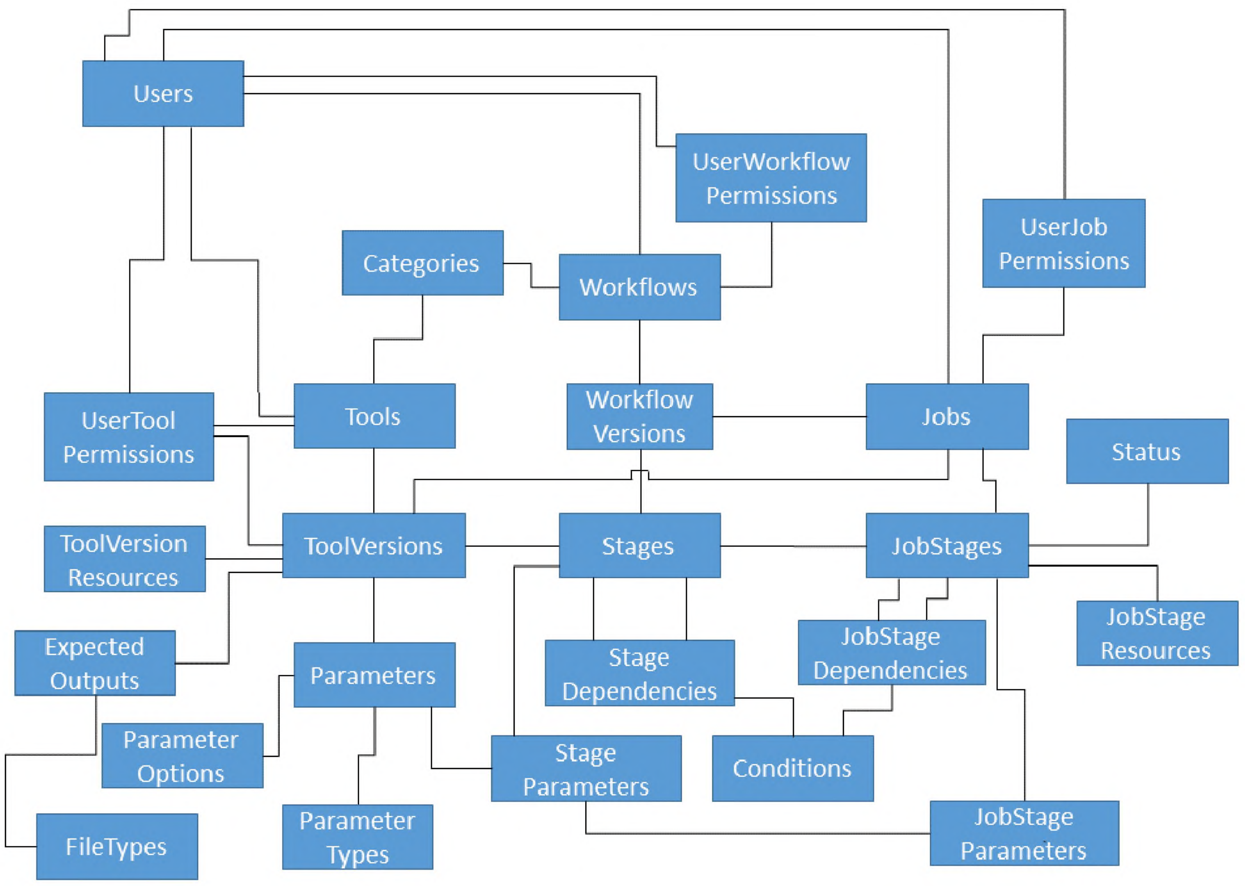


Fig. 2.4. Database – simplified entity relationship diagram depicting the relationships between tables in the database. Fields have been emitted from the diagram for the sake of simplicity.

When users add a tool to JMS, an entry is created in the *Tools* table. This entry stores data such as the tool name and description, which category it falls under, and which user created it. At the same time as a tool is added, an entry is added to the *ToolVersions* table. This table was added to JMS to allow for a single tool to have multiple versions. On tool creation, a version with a *ToolVersionNum* of 'dev' is created for that tool. This is the development version of the tool. When users edit a tool, they are really editing the development version of the tool. As such, most of the other tables in this section of the database relate to the *ToolVersions* table. Together with the *ToolVersions* table, these additional tables store information on how to execute the tool. For example, the *ToolVersions* table has a field called *Command*. This field stores the command that would be used to execute the tool from the command line. However, most command line tools also

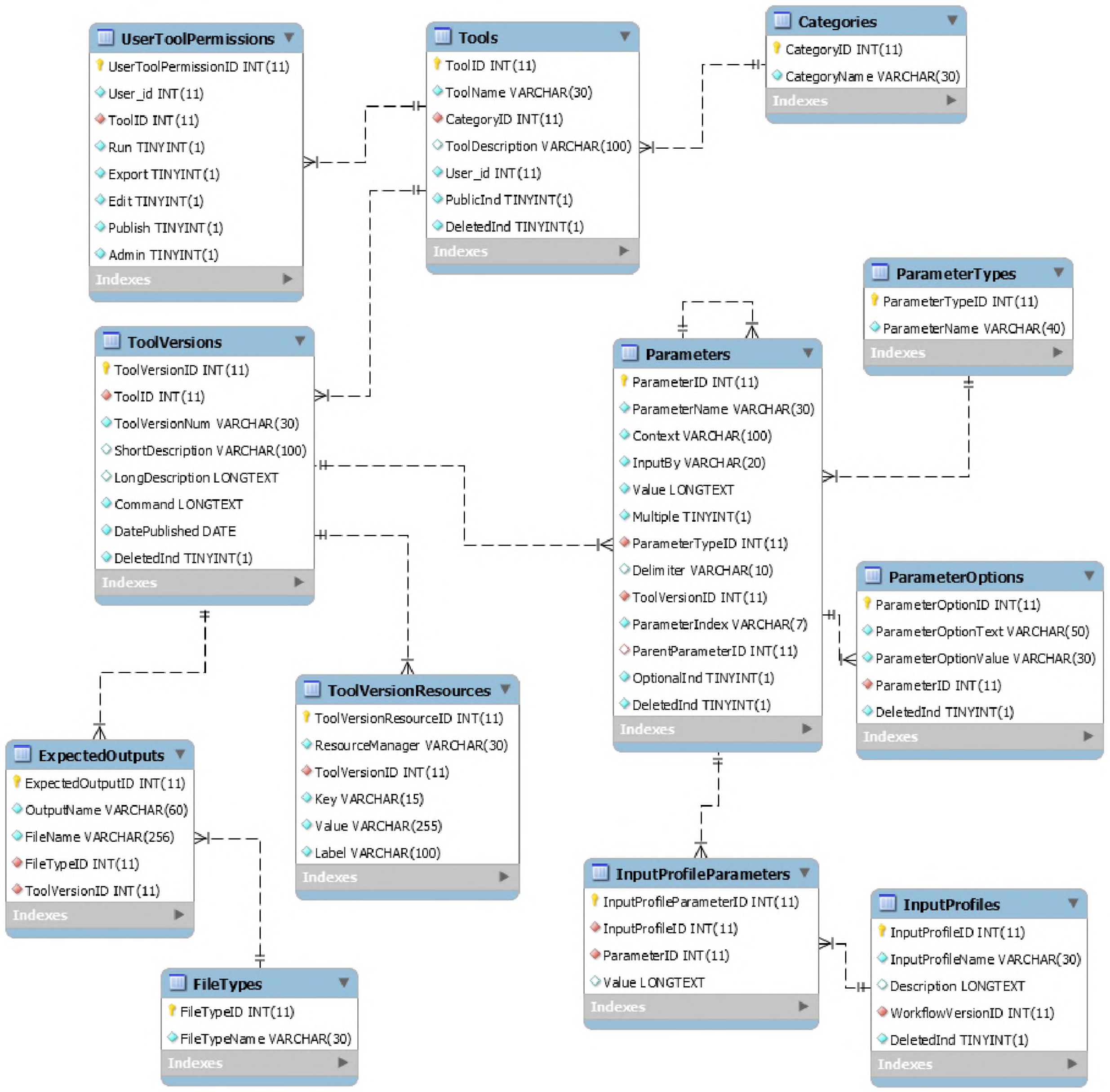


Fig. 2.5. Tool tables – entity relationship diagram generated by MySQL Workbench depicting the tables in the database required to manage tools. Fields with a gold key next to them are primary keys. Fields with red diamonds next to them are foreign keys.

require arguments, or parameters, to execute correctly. The *Parameters* table stores data about the parameters that the user needs to input for the tool. This includes data that is used to generate the interface for the tool as well as data that is used to generate the job script that is submitted to the cluster. For example, when generating the interface, the *ParameterName* field is used to give the input field a label so that the user knows what sort of data to input. In addition, the *ParameterTypeID* field is used to describe what type of input box should be generated in the

Parameter Type	**Input Control**
Text	Textbox
Number	Textbox accepting only numbers
True/False	Checkbox
Options	Dropdown menu
Complex object	Custom built input interface allowing users to enter hierarchical data (input is converted to and stored as JSON)
Related parameter	Reference to another parameter – useful in conjunction with complex parameters

Table 2.1. Parameter types supported by JMS

interface. For example, it could be a textbox, a dropdown menu, or a checkbox. The full list of parameter types is described in Table 2.1.

Important to note in Table 2.1 are complex objects. These are unique to JMS and allow JMS to generate advanced interfaces consisting of popup dialogs and hierarchical data. A complex object is converted to a JSON string before being sent through to the server. For example, if a tool accepts a list of authors as a parameter, these can be input as multiple complex objects, each consisting of the authors name, surname and title. Complex objects themselves are made up of list of parameters, which could also be complex objects. As such, JMS allows complex objects to be nested within each other. Using the authors example from before, a fourth parameter could be a list of publications that the author has published, where each publication would be a nested complex object consisting of the publication title, date, and the journal it was published in.

The second unique parameter type is the "related parameter", which allows users to relate parameters that were input earlier to a new parameter. Using the authors example above, authors and publications could be input as two separate lists of complex objects instead on nesting publications within the author objects. A related parameter could then be added to the publications objects to relate a publication with a subset of the authors input via the authors parameter. This would mean that, as opposed to the example above, if authors co-authored a publication, the publication wouldn't need to be input multiple times for each author (see section 3.2 for an example of this).

The database also allows users to create input profiles. Input profiles are essentially sets of default inputs for a tool and come in handy when a user needs to repeat a job several times and does not want to enter in the same inputs repeatedly. In addition, it is often the case where a number of parameters for a tool can be kept constant. Input profiles allow defaults to be set for these parameters.

Permission Type	**Description**
Run	Users are allowed to execute the tool
Edit	Users are allowed to update the tools *e.g.* change the command, parameters and scripts used to execute the tool
Export	Users are allowed to export the tool and accompanying scripts to compressed file
Publish	Users are allowed to publish new versions of the tool
Administrate	Users are allowed to delete the tool as well as share the tool with other users and assign access permissions to those users

Table 2.2. Per user access permissions that can be assigned for a tool

In addition to storing the inputs required to run the tool, the database stores the outputs expected from the tool. This is useful for creating workflows as it allows the user to specify which outputs of the tool should become the inputs of the next tool in the workflow. This information is also used on the *Job History* page to allow users to download certain outputs once the job is complete.

As opposed to the above tables, which store details on how a tool is run, the *ToolVersionResources* table stores details on what compute resources, such as memory, processing cores, and wall-time, are required by the tool. This table includes the *ResourceManager* field, which allows different resources to be assigned depending on which resource manager is being used.

Lastly, the *UserToolPermissions* table stores access control information for each tool. This table allows *per user* access permissions to be assigned to tools. Permissions that can be granted to users are described in Table 2.2.

2.5.1.2. Workflows

Workflows are created in JMS by pipelining existing tools. The JMS provides a workflow design interface (discussed in section 2.7.3.1), which allows users to drag-and-drop tools onto a canvas and assign dependencies between those tools. These dependencies include conditions determining when/if a tool in the pipeline should be run and what data should be passed from a previous tool to the next tool. These details are stored across several tables in the JMS database (Fig. 2.6).

There are a number of similarities between the tools and workflows parts of the database. As with tools, a versioning system has been developed for workflows. In addition, workflows have the same user permissions system as tools and can also be assigned to the same categories. This is where the similarities end, however. Workflow versions are made up of one or more stages. A stage in a workflow represents a version of a tool that is executed as part of that stage. The *Stages* table also includes a *SubWorkflowVersionID* field. In future, this field will be used to allow workflows to be made up of other workflows in addition to tools.

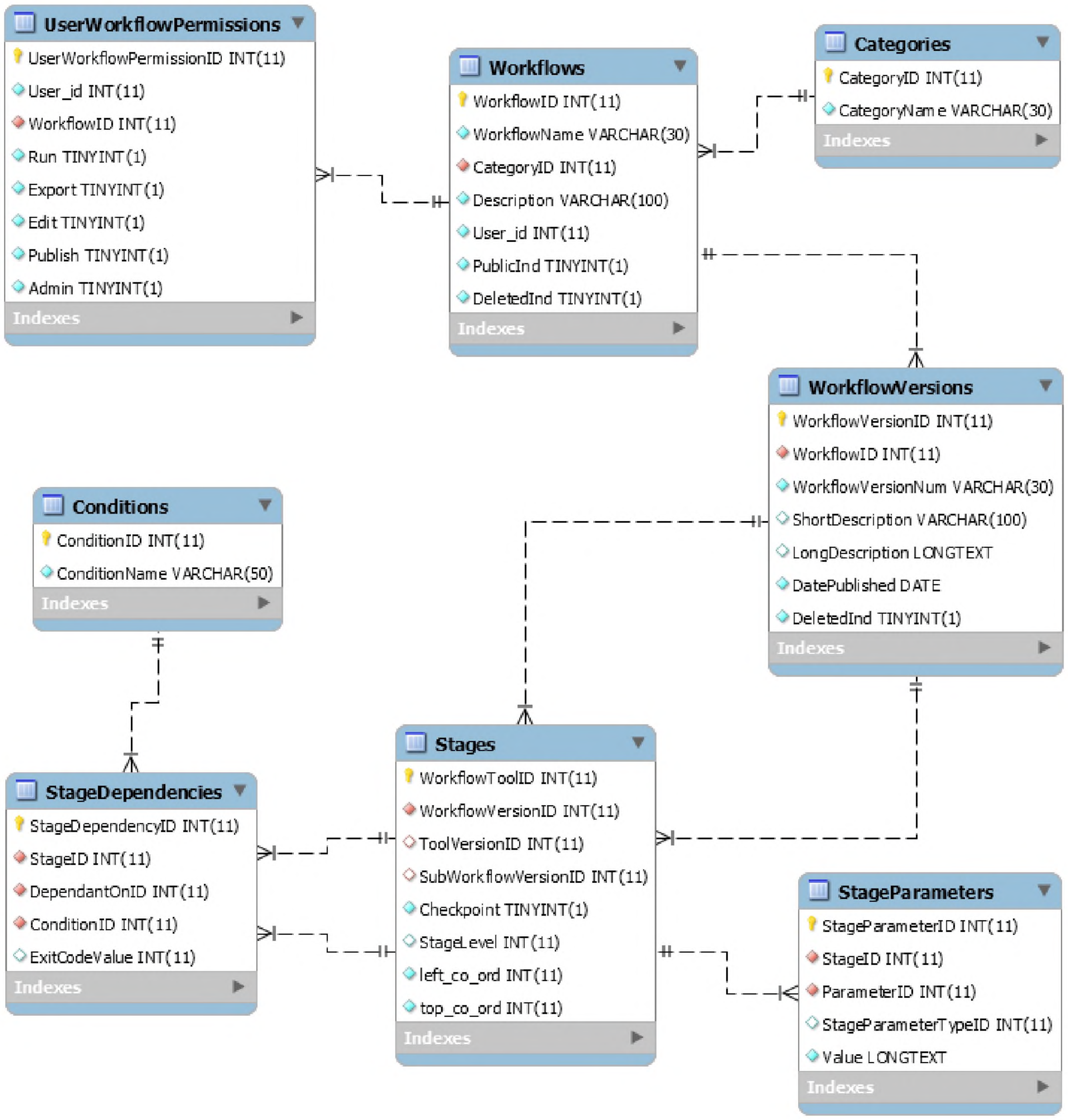


Fig. 2.6. Workflow tables – entity relationship diagram generated by MySQL Workbench depicting the tables in the database required to manage workflows. Fields with a gold key next to them are primary keys. Fields with red diamonds next to them are foreign keys.

The workflows section of the database also includes the *StageDependencies* table. This table stores information on which stages depend on one another and what conditions must be satisfied for them to execute. For example, stage B may depend on stage A completing successfully, while stage C may depend on stage A failing. In this example, both stage B and C depend on stage A, but with different dependency conditions. The *Conditions* table stores the list of possible conditions (Table 2.3).

The *StageParameters* table stores the parameters of the different tools in the workflow that should be populated automatically by JMS. For example, if one of the parameters of stage B in the workflow needs to be populated by the output of stage A in the workflow, this information is stored in the *StageParameters* table.

2.5.1.3. Jobs

Where the previous two sections described the parts of the database that are required to create, store and edit tools and workflows, the following section describes the parts of the database needed to track and store job details when a tool or workflow is executed (Fig. 2.7). All details required to execute the job are stored, allowing jobs to be accurately repeated in the future if necessary.

Condition	**Description**
Stage completed	The stage will execute once the stage it depends on completes
Stage completed successfully	The stage will execute only if the stage it depends on completes successfully
Stage failed	The stage will execute only if the stage it depends on fails
Exit code	The stage will execute only if the stage it depends on exits with a certain code

Table 2.3. Stage dependency conditions

As with tools and workflows, jobs can be shared with other users. Different access permissions, such as the ability to repeat, view and delete jobs, can be granted to these users. These permissions are stored in the *UserJobPermissions* Table.

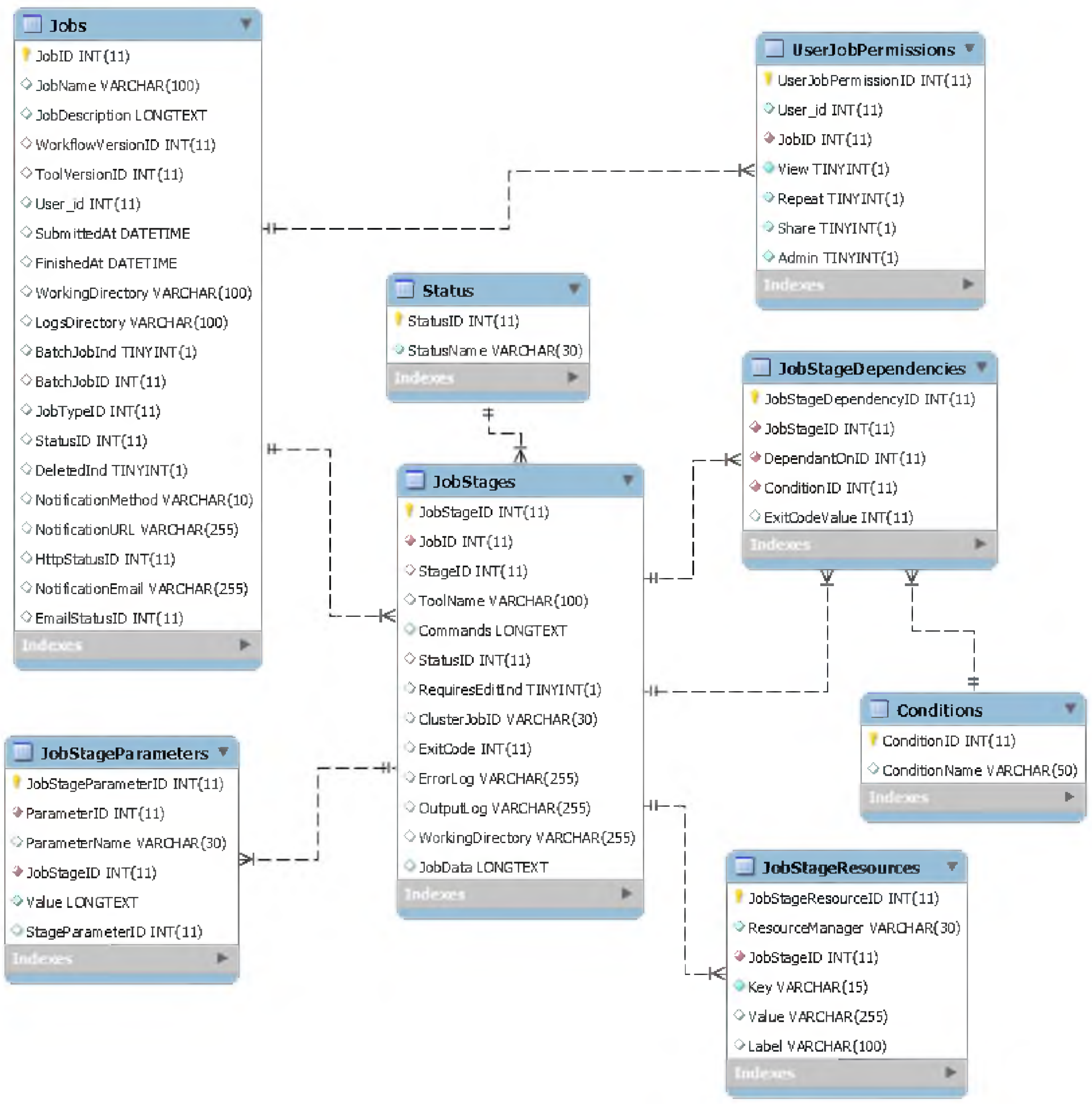


Fig. 2.7. Job tables – entity relationship diagram generated by MySQL Workbench depicting the tables in the database required to manage jobs. Fields with a gold key next to them are primary keys. Fields with red diamonds next to them are foreign keys.

When a tool or workflow is executed, an entry is created in the *Jobs* table. As such, a job is essentially an instance of a tool or workflow that has been, or is being, executed. Details stored for jobs include the job name and description, which tool or workflow version is being executed, the user that submitted the job, and details for sending notifications when the job completes.

As a job can be an instance of a workflow, it can be made up of many stages. Where the *Jobs* table stores the overall job data, the *JobStages* table keeps track of the execution details of each stage of a job *e.g.* which tool is being executed, the command used to execute the tool, the status of the stage, the job ID for the job on the cluster, and where the working directory and job output and error logs can be found on the server. The *JobData* field stores status information for the job that comes directly from the underlying resource manager. As such, the data stored in this field will be dependent on which resource manager is being used and is accessed via a call to the resource manager plugin, which returns the data in the predefined *DataSection* data structure. This object is then converted to JSON before being stored in the database.

The *JobStageParameters* table stores the values input by a user for each of a tool's parameters. This is necessary to repeat the job at a later stage. For the same reason, the *JobStageResources* table stores the details of what computational resources were used to execute the job on the cluster.

Lastly, the *JobStageDependencies* table is similar to the *StageDependencies* table in that it stores details on the dependencies for each stage of a job. Each time a stage completes, this table is checked to see which of the remaining stages have had all their dependencies satisfied and can, thus, begin executing.

2.5.1.4. Summary

The JMS database provides a central location for all data regarding tools, workflows, and jobs. It stores data on how to execute tools and workflows, and allows the creation of multiple versions of these tools and workflows. It also stores details regarding the *types* of inputs required by tools, which allows JMS to dynamically generate web interfaces for those tools and the workflows that use them.

When a user executes a tool or workflow, JMS stores the details on how the tool or workflow was executed. These details include what the user inputs were, what version of the tool or workflow was executed, and what resources were used to execute it. With these details stored, the job can be repeated at any point in the future. In addition, while the job is running, JMS gets status information from the underlying resource manager. This data is also stored in the database and can be used for debugging and reporting purposes.

2.6. Security

JMS has unique needs when compared to most web servers. To submit and manage jobs on a cluster as well as view and retrieve the results of these jobs, JMS needs to be able to impersonate users on the underlying cluster. For example, when hosting a website on Ubuntu using Apache 2, the website will, by default, run as the user “www-data”. As such, any process that is spawned by the web server will be owned by this user. This is fine for most websites, but JMS has special needs. When a user, “john”, submits a job to be executed on the cluster, the process that is spawned must be owned by “john”, and not by “www-data”. Similarly, if another user, “wendy”, submits a job, the processes that are spawned must be owned by “wendy”. In other words, JMS, which is running under the user “www-data”, must be able to impersonate multiple users on the cluster. There are two main reasons for this requirement:

1) Filesystem access

If “john” submits a job, he expects the scripts or tools that run to be able to access the same files and directories on the cluster that he can. For example, the task that he is executing may need to read data from his home directory and as such, the process will need to have permission to access that directory. If the job is being run as “www-data”, it won’t have access to “john’s” home directory.

2) Accounting

Cluster administrators often want to know, which users are running jobs and what type of jobs they are running. This information is also useful for reporting purposes and for finding users who may be abusing the cluster. If all jobs submitted by JMS run under the “www-data” user, it may obscure this information.

JMS uses its own custom authentication backend to allow it to impersonate users on the cluster as well as provide access control for tools, workflows, and jobs.

2.6.1. Impersonator

The Impersonator is a lightweight, standalone server that runs alongside JMS on the master node of the cluster. It has been developed using Twisted [65], an event-driven networking engine written entirely in Python. The Impersonator runs as the root user on the localhost. It is not accessible from any other host and all communication between it and JMS is encrypted using public-key encryption. The Impersonator is used for two tasks, namely, authenticating users and impersonating users.

2.6.1.1. Authentication

Authentication is another area where JMS differentiates itself from most web servers. Where a normal web server will have a backend database containing user details that it authenticates against, JMS authenticates against the underlying Linux operating system. This means that users can log into JMS using the same username and password that they would use to SSH into the

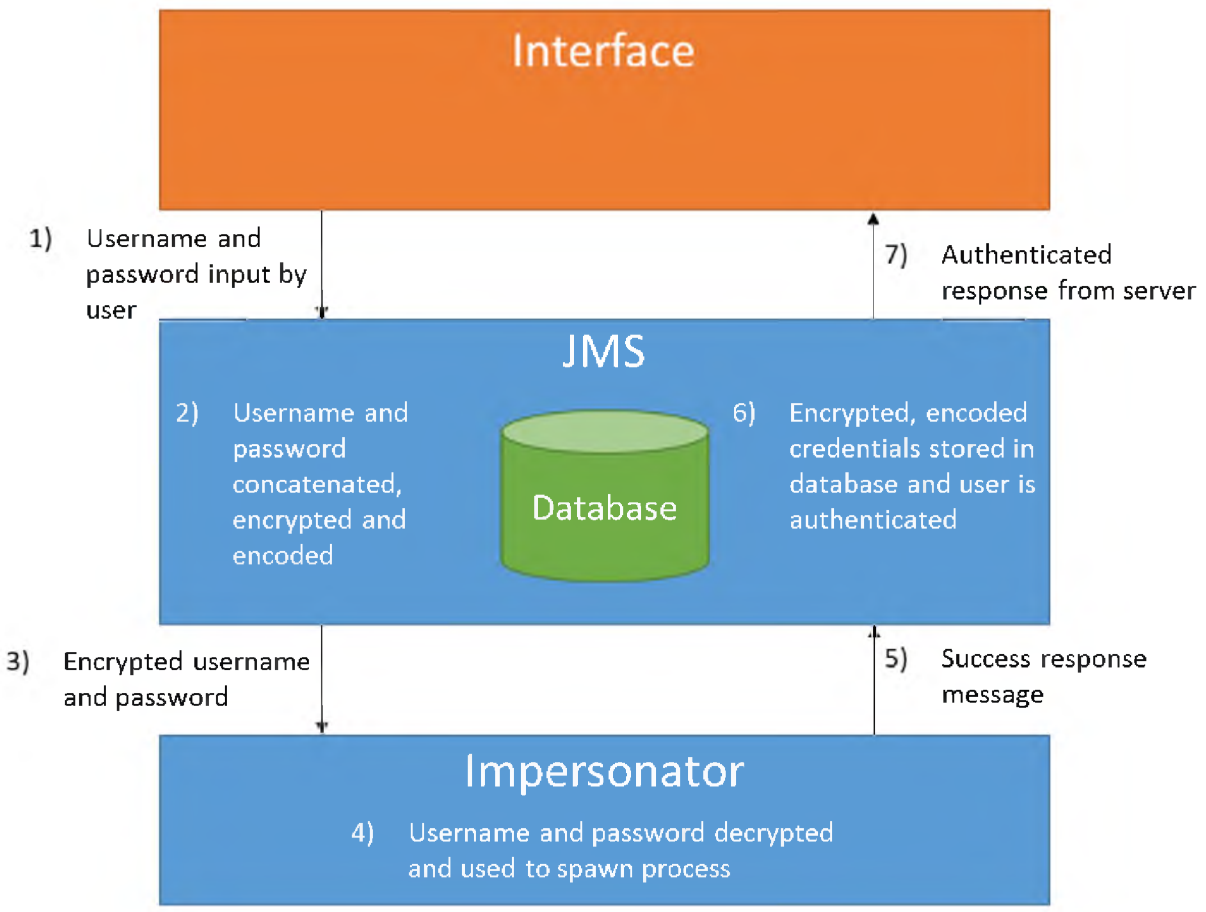

Interface
1) Username and password input by user
7) Authenticated response from server
JMS
2) Username and password concatenated, encrypted and encoded
Database
6) Encrypted, encoded credentials stored in database and user is authenticated
3) Encrypted username and password
5) Success response message
Impersonator
4) Username and password decrypted and used to spawn process

Fig. 2.8. Authentication process – JMS authenticates against the underlying Linux operating system by spawning a process as a particular user using the user input username and password. If the username and password combination can successfully spawn a process, the credentials must be correct and the user is logged in. The encrypted credentials are then stored in the database so they can be used later for impersonating the user.

master node of the cluster. As such, all users who have an account on the master node of the cluster will automatically have access to JMS.

The Django framework has a built-in authentication backend, which authenticates a username and password against an entry in the database. To authenticate against the underlying Linux accounts, a custom backend was developed to override the built-in Django authentication method. The custom authentication process is illustrated in Fig. 2.8. The method accepts a username and password input by a user. When JMS receives the credentials on the server-side, the username and password are concatenated, but separated by a colon. This concatenated string is encrypted using the Impersonator's public key. Before sending it to the Impersonator, the encrypted string is encoded using base64, to ensure that it has all legal characters. The encoded string is then sent to the Impersonator, which decodes it and uses the private key to decrypt it. The resultant string is the concatenated username and password. This string is split into two separate strings, the username and password, based on the position of the inserted colon. The Impersonator uses the username and password to spawn a process and run a simple command. If this fails, the status of the HTTP response is set to 403 (*i.e.* the forbidden response code used by HTTP) and an error message is returned to JMS. If it is successful, the credentials are stored in a timed dictionary - the entry is removed from the dictionary if no requests are received for 10 minutes. The status code of the HTTP response is set to 200 and the output of the command that was run is returned to JMS. On receiving the response, JMS checks to see if the authentication request was successful or not based on these status codes. If the status code is 200, the encrypted and encoded credential string of the user is stored in the database. If this is the first time the user is logging into JMS, an entry in the database must be created for the user. If the user has logged in before, the previously created entry is simply updated with the latest credentials. The encrypted credentials are stored so that JMS can impersonate the user at a later stage without requiring the user to enter their username and password again. Once these details are stored, the user is logged in and may proceed to use JMS.

2.6.1.2. Impersonation

Impersonating users on the master node of the cluster is the main purpose of the Impersonator server. Certain tasks, such as submitting jobs to the cluster, reading files and traversing directories in the filesystem require knowledge of the permissions of the particular user. As such, processes that perform these tasks need to run as the user in question. JMS impersonates users by communicating with the Impersonator server. This process is illustrated in Fig. 2.9.

When a user submits a request from the interface (*e.g.* a request to run a job on the cluster) it is received by JMS and, depending on the details received, a command is formulated to be submitted to the cluster. To run the command as the user, JMS must send the encrypted user credentials along with the command to the Impersonator. The credentials, which were stored after successful authentication, are retrieved from the database and concatenated with the formulated command. On receiving the request, the Impersonator decrypts the user credentials and checks the timed dictionary to see whether the credentials match those stored for the user. If the credentials do not match those stored in the dictionary, the request fails and an error message is returned to JMS

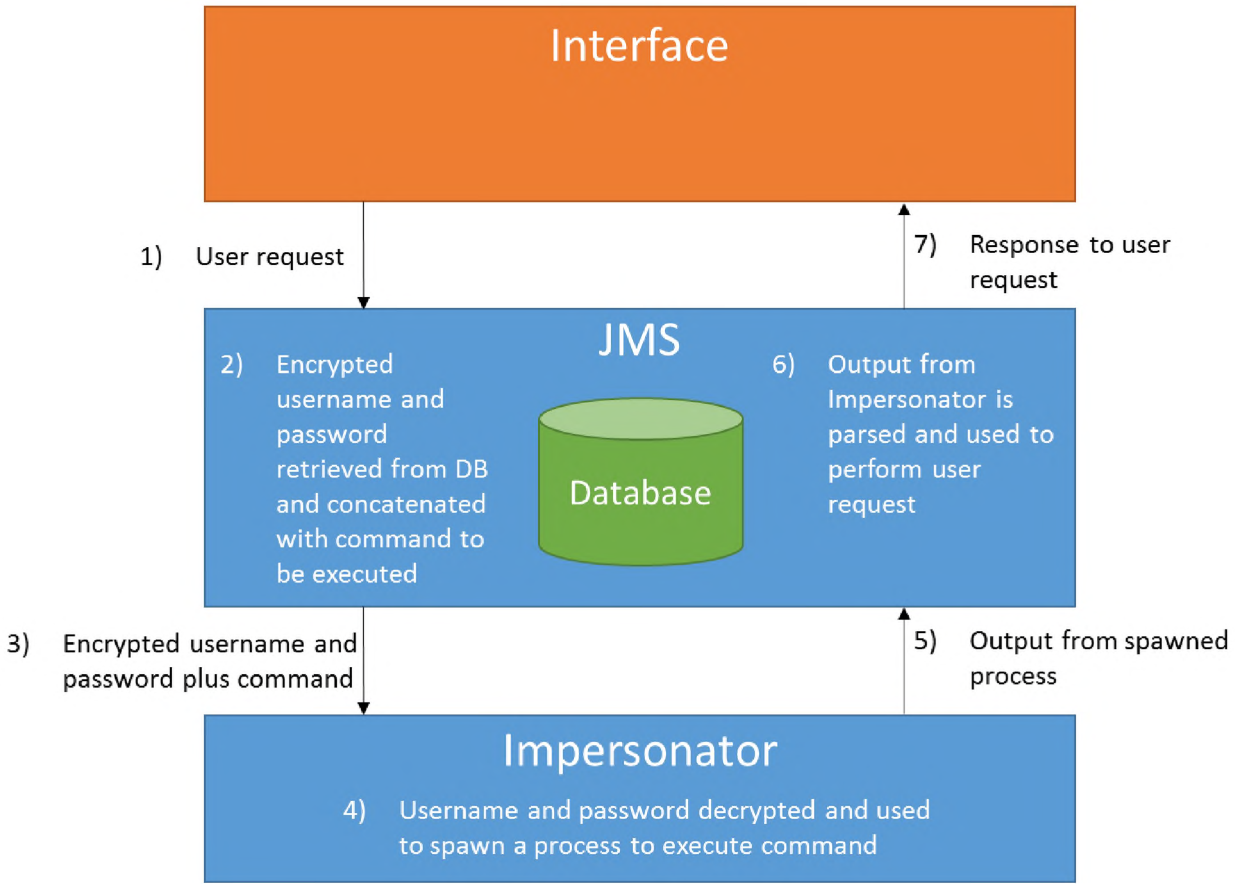

Interface
1) User request
7) Response to user request
JMS
2) Encrypted username and password retrieved from DB and concatenated with command to be executed
Database
6) Output from Impersonator is parsed and used to perform user request
3) Encrypted username and password plus command
5) Output from spawned process
Impersonator
4) Username and password decrypted and used to spawn a process to execute command

Fig. 2.9. *Impersonation – JMS must impersonate users on the cluster when submitting jobs and accessing files. To do this, processes that perform these tasks are spawned by the Impersonator using the username and password that was stored in the database when the user logged in. These processes will, therefore, run as that user and have the same permissions etc.*

along with a 403 status code. If the credentials do match, a process is spawned to execute the command and the output of the process is returned to JMS. If the credentials stored in the dictionary have expired, the authentication process is run again, before the command is executed.

2.6.2. Access control

As alluded to in previous sections, JMS allows users to share tools, workflows, and jobs with other users on the system. Over and above simple sharing, JMS provides users with the ability to grant specific permissions to these users. For example, if a user creates a tool, he can grant certain permissions to other users such as the ability to run, edit, export, publish versions of, or administrate the tool (Table 2.2). In addition, the tools and workflows can be made publicly available, which essentially grants the "Run" permission to all users on the system.

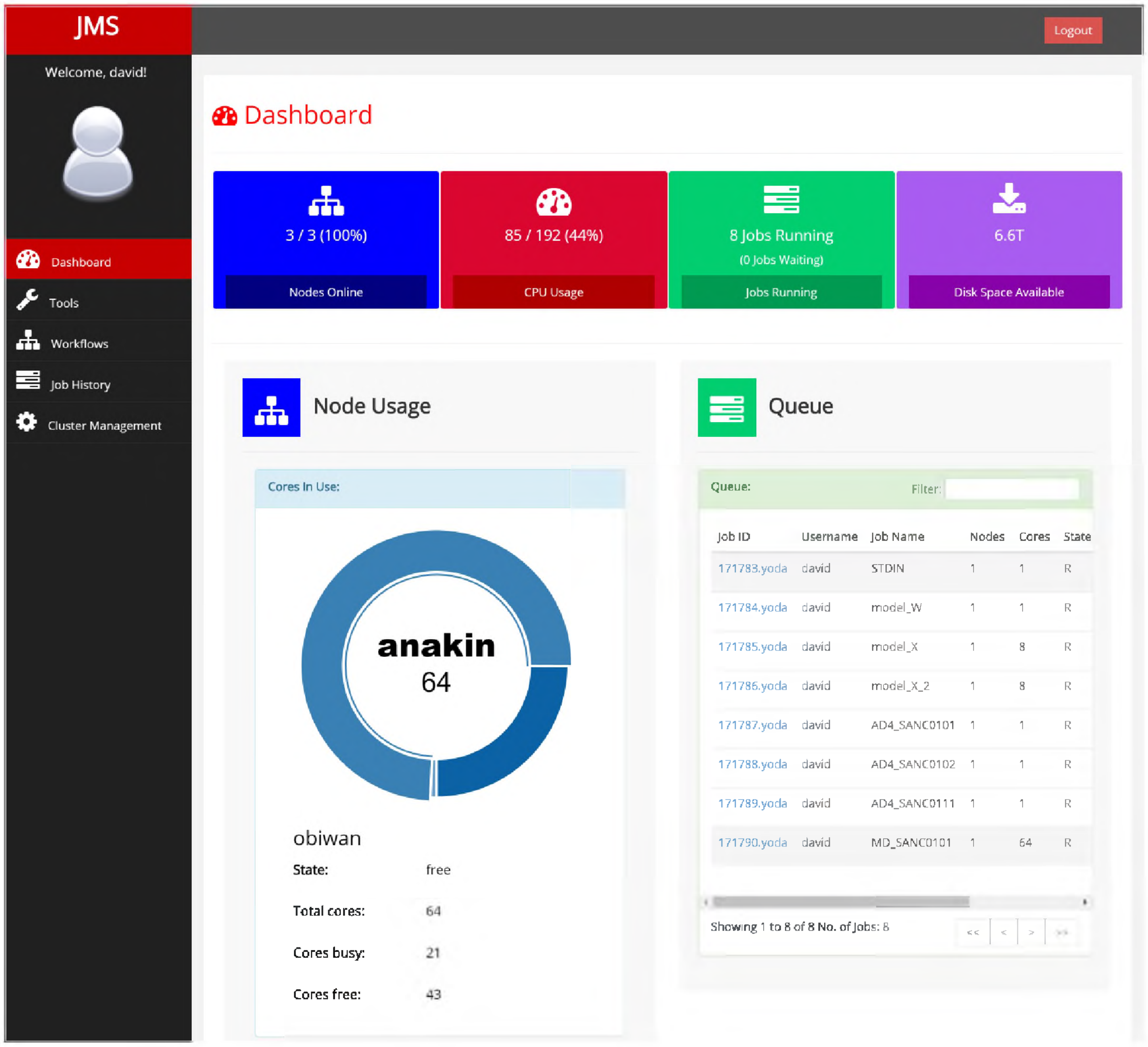
Access control functionality works via the *UserToolPermissions*, *UserWorkflowPermissions*, and *UserJobPermissions* tables in the database. When a tool, workflow, or job is shared with another user, an entry is created in the relevant table for the user, which includes the permissions that have been granted to the user. When a user attempts to access an item that has been shared with them, these tables are first checked to see whether the user has the necessary permissions. If they don't, a permission denied error is returned.

2.7. Results & discussion

The previous sections have discussed some important concepts regarding how JMS was designed and developed. In this section, the results that have been achieved through this work are discussed. Specifically, the features provided by JMS and their implementation within the interface are analyzed. These features include a dashboard for monitoring the status of the cluster and the jobs running on it, tool and workflow management, job management and monitoring, and cluster management and configuration.

2.7.1. Dashboard

The dashboard is the JMS home page (Fig. 2.10). It displays status information for the overall cluster, individual nodes in the cluster, and the job queue. The four blocks at the top of the interface provide the overall status of the cluster at a glance. These blocks allow users to quickly see if any



obiwan

State:	free
Total cores:	64
Cores busy:	21
Cores free:	43

Job ID	Username	Job Name	Nodes	Cores	State
171783.yoda	david	STDIN	1	1	R
171784.yoda	david	model_W	1	1	R
171785.yoda	david	model_X	1	8	R
171786.yoda	david	model_X_2	1	8	R
171787.yoda	david	AD4_SANC0101	1	1	R
171788.yoda	david	AD4_SANC0102	1	1	R
171789.yoda	david	AD4_SANC0111	1	1	R
171790.yoda	david	MD_SANC0101	1	64	R

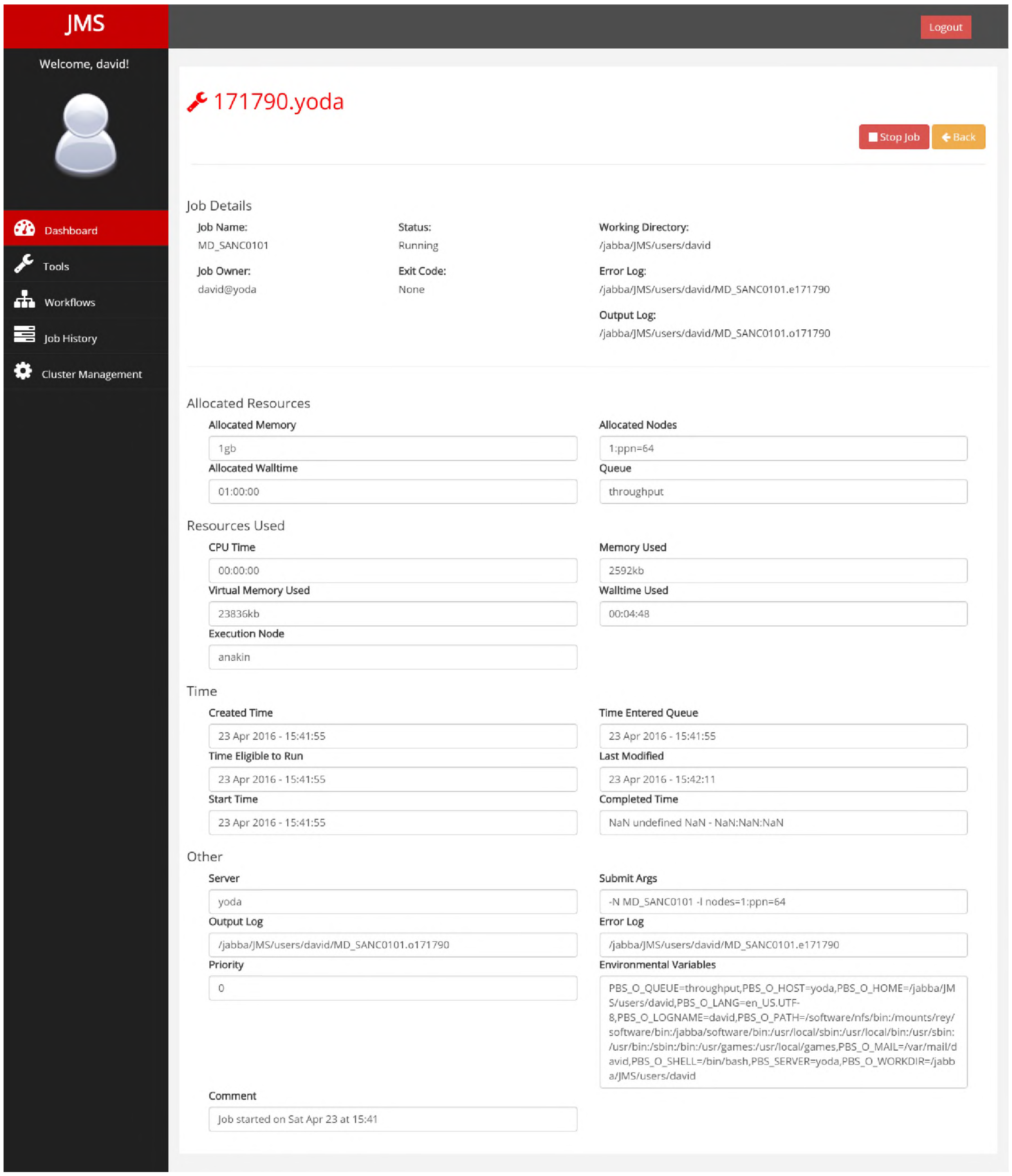
Showing 1 to 8 of 8 No. of Jobs: 8

Fig. 2.10. JMS dashboard – the dashboard contains status information for the underlying cluster and allows users to monitor their jobs in the queue.

nodes are offline, if any CPU cores are free for job submission, how many jobs are running, how long the job queue is, and how much disk space is still available on the server. They provide a good example of functions that need to be overridden by resource manager plugins. Data used to populate the first three blocks is dependent on the underlying resource manager. As such, this data is obtained via functions that have been overridden by the respective resource manager plugin. On the other hand, obtaining the disk space available on the server is not in any way dependent on the resource manager. As such, this function is implemented in the *BaseResourceManager* class and is inherited by all resource manager plugins. It does not need to be overridden.

The 'Node Usage' section of the dashboard provides detailed status information for individual nodes. This information includes the current state of the selected node *i.e.* whether it is free to accept new jobs, busy, or offline. The total number of cores available on the node and how many of those cores are free or busy is also displayed here. How this information is obtained is also dependent on the underlying resource manager and, as such, this data is retrieved via overridden functions in the resource manager plugin.

The 'Queue' section provides a table for checking the job queue of the cluster. It allows users to see how many jobs are being run, who is running these jobs, and what resources each job is using. The job queue is updated every 15 seconds by calling an overridden function in the resource manager plugin. To improve the loading time when there are thousands of jobs in the queue, the table is paged, meaning that it only shows a portion of the jobs at a time. Users can use the buttons at the bottom-right of the table to move between pages. Jobs can also be cancelled from the job table by clicking on the red "trash can" button in the last column of the table. Lastly, by clicking on the job ID in the first column of the table, users are presented with a detailed status page for the selected job (Fig. 2.11).



171790.yoda

Job Details

Job Name: MD_SANC0101	Status: Running	Working Directory: /jabba/JMS/users/david
Job Owner: david@yoda	Exit Code: None	Error Log: /jabba/JMS/users/david/MD_SANC0101.e171790
		Output Log: /jabba/JMS/users/david/MD_SANC0101.o171790

Allocated Resources

Allocated Memory: 1gb	Allocated Nodes: 1:ppn=64
Allocated Walltime: 01:00:00	Queue: throughput

Resources Used

CPU Time: 00:00:00	Memory Used: 2592kb
Virtual Memory Used: 23836kb	Walltime Used: 00:04:48
Execution Node: anakin	

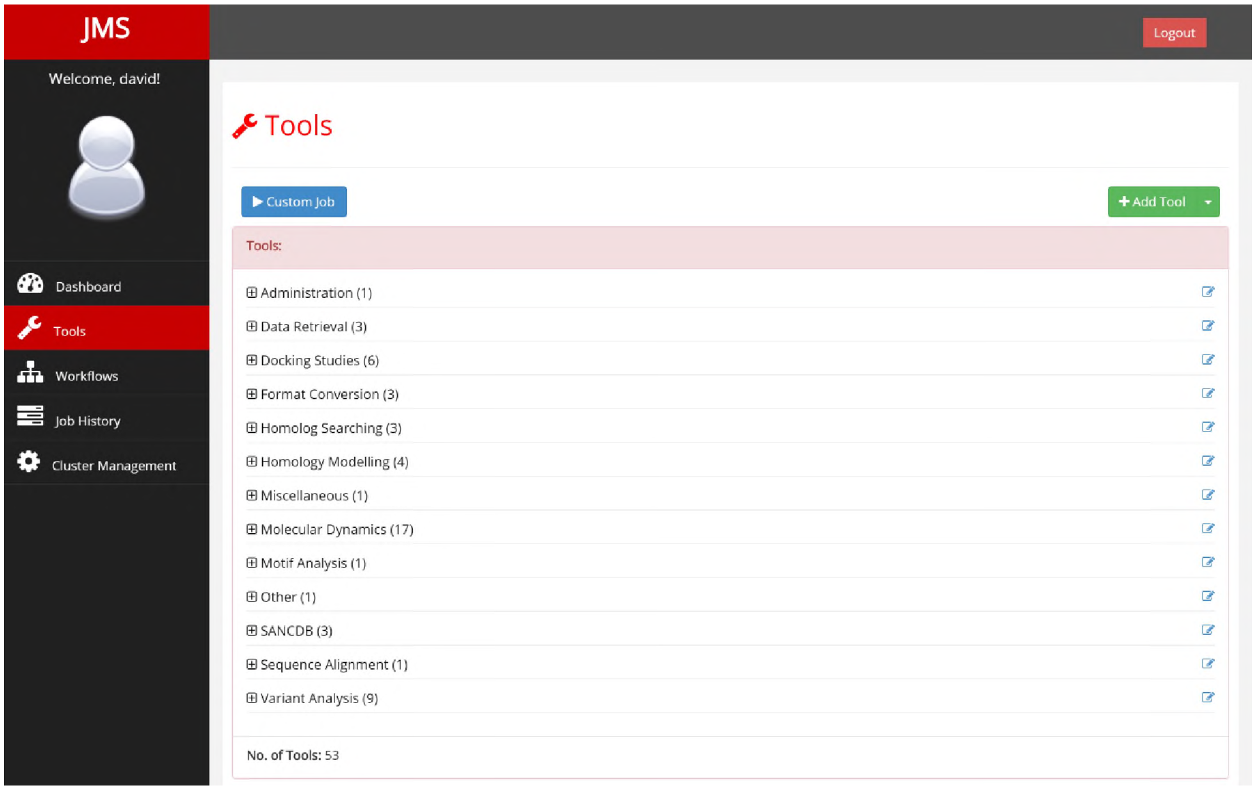
Time

Created Time: 23 Apr 2016 - 15:41:55	Time Entered Queue: 23 Apr 2016 - 15:41:55
Time Eligible to Run: 23 Apr 2016 - 15:41:55	Last Modified: 23 Apr 2016 - 15:42:11
Start Time: 23 Apr 2016 - 15:41:55	Completed Time: NaN undefined NaN - NaN:NaN:NaN

Other

Server: yoda	Submit Args: -N MD_SANC0101 -l nodes=1:ppn=64
Output Log: /jabba/JMS/users/david/MD_SANC0101.o171790	Error Log: /jabba/JMS/users/david/MD_SANC0101.e171790
Priority: 0	Environmental Variables: PBS_O_QUEUE=throughput,PBS_O_HOST=yoda,PBS_O_HOME=/jabba/JMS/users/david,PBS_O_LANG=en_US.UTF-8,PBS_O_LOGNAME=david,PBS_O_PATH=/software/nfs/bin:/mounts/rey/software/bin:/jabba/software/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games,PBS_O_MAIL=/var/mail/david,PBS_O_SHELL=/bin/bash,PBS_O_SERVER=yoda,PBS_O_WORKDIR=/jabba/JMS/users/david
Comment: Job started on Sat Apr 23 at 15:41	

Fig. 2.11. Job details virtual page – accessed by clicking on a job in the queue on the dashboard page. Information displayed on this page will depend on the underlying resource manager plugin being used, but will usually include detailed information about the job such the resources allocated and used, timing information, environmental variables, arguments, and output logs.



Tools:

⊞ Administration (1)
⊞ Data Retrieval (3)
⊞ Docking Studies (6)
⊞ Format Conversion (3)
⊞ Homolog Searching (3)
⊞ Homology Modelling (4)
⊞ Miscellaneous (1)
⊞ Molecular Dynamics (17)
⊞ Motif Analysis (1)
⊞ Other (1)
⊞ SANCDB (3)
⊞ Sequence Alignment (1)
⊞ Variant Analysis (9)

No. of Tools: 53

Fig. 2.12. Tools page – from this page users can add, edit, delete, share, and run tools. Users can also run custom jobs from this page.

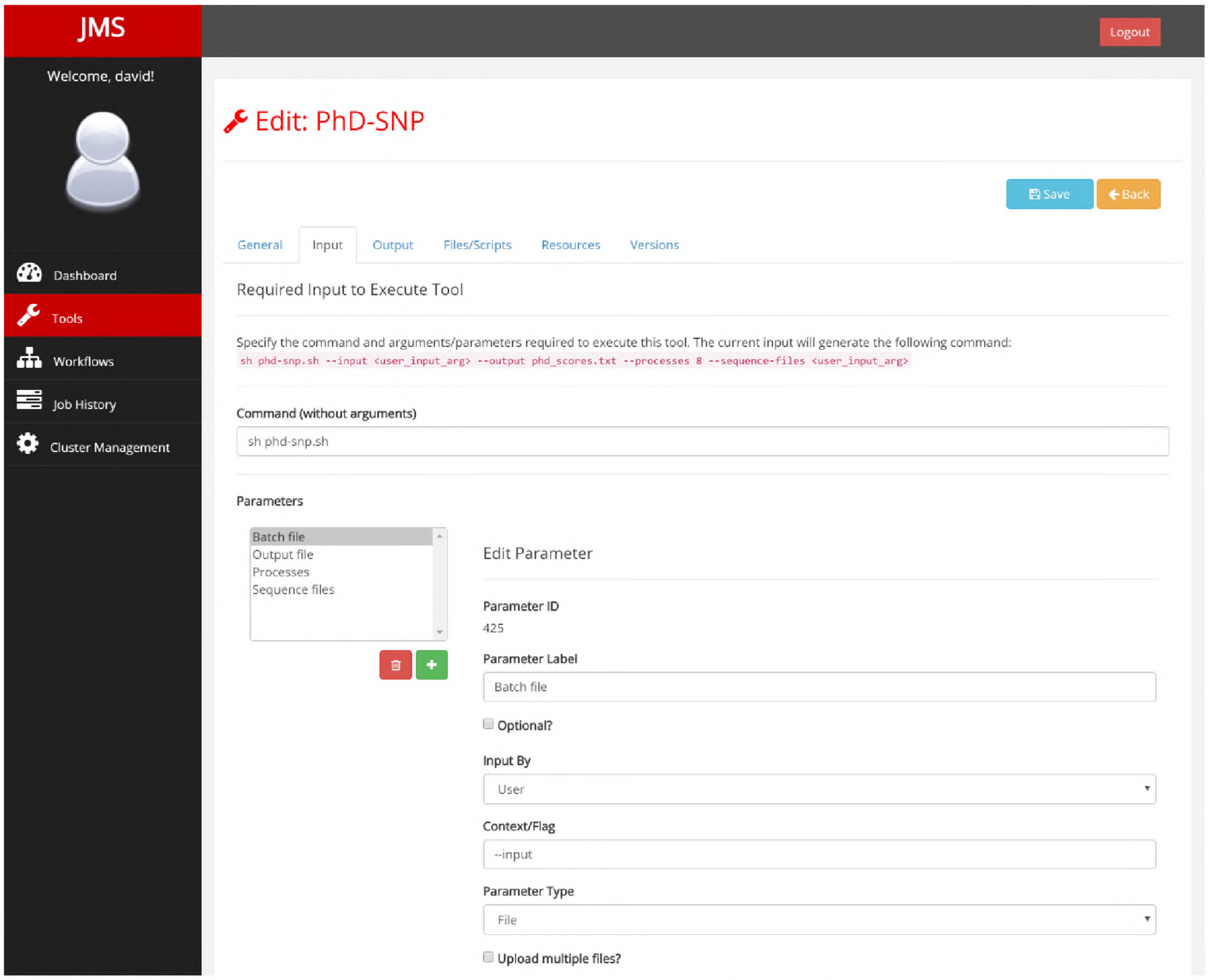
2.7.2. Tool management

JMS allows users to add and create tools directly via the web interface. This contrasts with other similar WMSs, which require users to add tools by writing complex configuration files via a text editor or the CLI. By creating a web interface for this purpose, the entire process of building custom workflows with one's own tools and scripts is made easier.

The "Tools" page provides a list of all the tools that have been added to the system, split into user-assigned categories (Fig. 2.12). From this page, users can run custom commands on the cluster, add tools, and run, edit and share existing tools.

2.7.2.1. Creating/editing tools

JMS can run any tool or script that can be executed from the command line. There are two ways in which tools can be added to JMS.



Edit: PhD-SNP

Required Input to Execute Tool

Specify the command and arguments/parameters required to execute this tool. The current input will generate the following command:

```
sh phd-snp.sh --input <user_input_arg> --output phd_scores.txt --processes 8 --sequence-files <user_input_arg>
```

Command (without arguments): sh phd-snp.sh

Parameters

- Batch file
- Output file
- Processes
- Sequence files

Edit Parameter

Parameter ID: 425

Parameter Label: Batch file

Optional? [ ]

Input By: User

Context/Flag: --input

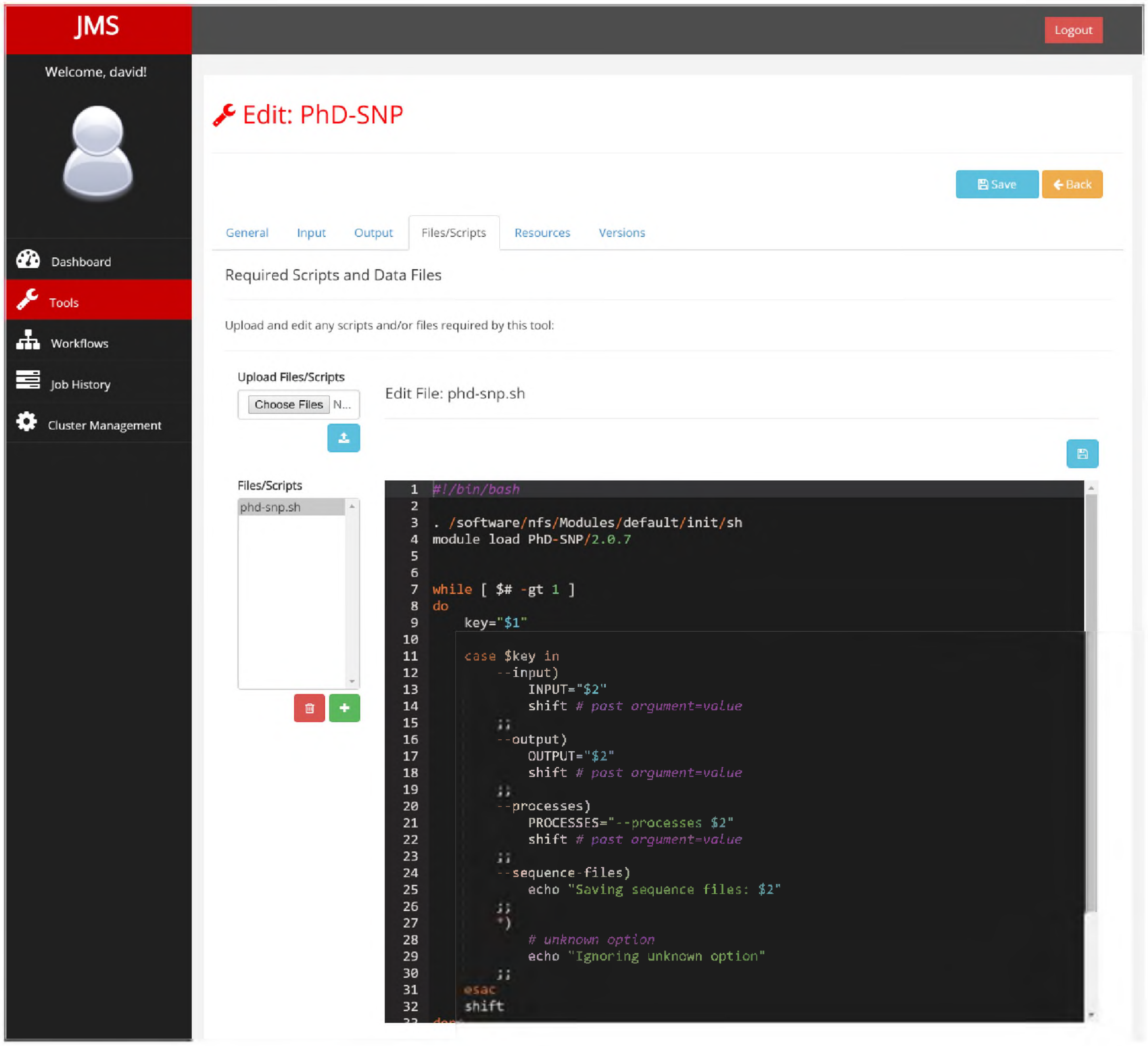
Parameter Type: File

Upload multiple files? [ ]

Fig. 2.13. Tool inputs – JMS provides and interface for describing how a tool is to be run. This includes providing the command required to run the tool as well as the parameters that the command accepts.

Firstly, a command-line utility that is already installed on the cluster can be added to JMS by specifying the command and parameters that are required to run it. This information is specified directly via the web interface (Fig. 2.13). As the tool is already available on the cluster, there is no need to upload anything.

Secondly, a custom script or executable that has not already been installed on the cluster can be uploaded via the JMS interface (Fig. 2.14). Once uploaded, the user specifies how the script can be executed in the same way as above. When running an uploaded tool, the script is copied into the working directory for the job. As such, when specifying how to run the tool, the user can



Required Scripts and Data Files

Upload and edit any scripts and/or files required by this tool:

Edit File: phd-snp.sh

```
#!/bin/bash

. /software/nfs/Modules/default/init/sh
module load PhD-SNP/2.0.7


while [ $# -gt 1 ]
do
    key="$1"

    case $key in
        --input)
            INPUT="$2"
            shift # past argument=value
        ;;
        --output)
            OUTPUT="$2"
            shift # past argument=value
        ;;
        --processes)
            PROCESSES="--processes $2"
            shift # past argument=value
        ;;
        --sequence-files)
            echo "Saving sequence files: $2"
        ;;
        *)
            # unknown option
            echo "Ignoring unknown option"
        ;;
    esac
    shift
```

Fig. 2.14. Code editor – JMS provides a code editor that allows users to upload, create and edit their own scripts via the web interface. This lets developers quickly fix bugs in their scripts and test them without needing to upload a new script each time.

assume that any uploaded scripts can be accessed by simply specifying the filename *i.e.* no absolute path to the script is required.

When specifying how to run the tool, the interface provides a textbox to enter in the command that would be used to run the script. For example, if a Python script called *script.py* is uploaded, the command to run it could be `` `python script.py` ``. If the script accepts arguments, they should not be included here.

Clicking the green plus below the select box adds a parameter (Fig. 2.13). Parameters allow users to input values for the command-line arguments of a tool via the web interface. When a parameter is added (or selected from the list of parameters), an "Edit Parameter" area becomes visible. Here, users can enter a label for the parameter, specify whether it is optional, specify whether the parameter value is input via a user or the system, provide the argument flag or context, and specify what type of parameter it is.

The parameter label is the label that is used on the input page for the tool. As previously mentioned, JMS automatically generates custom web interfaces for each tool. These interfaces provide input controls that allow users to input values for each parameter. The parameter label is used to name each of the input controls on the page so that users know what values to enter.

JMS allows certain parameter values to be input automatically by the system. In certain cases, one or more of the arguments for a script may always remain constant. In these scenarios, it makes sense to allow the system to enter these values automatically. If a parameter is set to be entered by the system, an input control will not be provided on the generated input page.

The "Context/Flag" field is usually used to specify the argument's flag within the command. For example, if the command is `python script.py -i arg1`, the flag for the first parameter would be `-i`. If the argument has no flag, this field is left blank. In addition to simply specifying argument flags, more complex argument contexts can be specified. For example, if the command to be run is `python script.py --input="arg1"`, the context would be `--input="$VALUE"`. In this example, `$VALUE` acts as a placeholder specifying where the user input value should be inserted in the generated command.

The last way in which a parameter can be customized is by specifying a "type". The parameter type chosen determines the type of input control that will be provided on the generated input page. By default, the parameter type is set to "Text". In such a case, a textbox will be provided on the input page for the user to enter a value. The full list of parameter types was provided in Table 2.1.

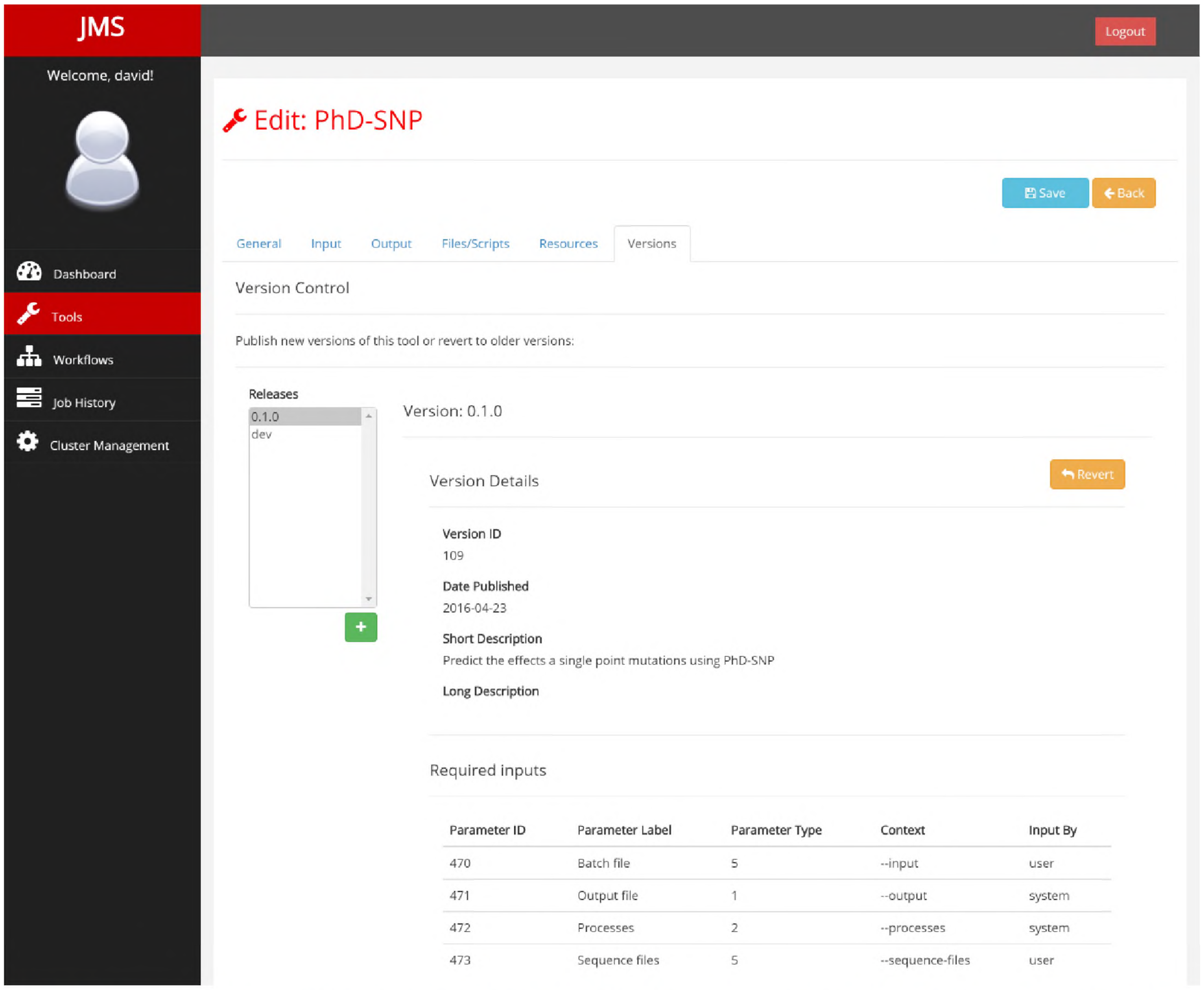
In addition to specifying the requirements to run the tool *i.e.* the command and parameters, JMS lets users specify the expected outputs of the tool. These outputs are the files produced by the tool when it is executed. Specifying what the expected outputs of a tool are is useful when creating a workflow, as will be discussed in the following section. The expected outputs are also used on the "Job History" page to create download links.

JMS also allows users to specify what computational resources should be allocated to a tool when it is executed on the cluster. This can be used to prevent users from abusing the cluster by requesting large amounts of computational power to execute tools that don't need it.

Once a tool has been created, it can be edited via the same interface. Tools can be edited by the owner (the user that created the tool), or any user that has been given the "Edit" permission.

2.7.2.2. Version control

JMS has been designed with built-in version control for tools and workflows. New versions can be published from the 'Versions' tab when editing a tool (Fig. 2.15). When this is done, the user is asked to enter a version number for the new version. An exact copy of the development tool version in its current state is created in the database *i.e.* a duplicate of the 'dev' record for the tool is created with the version number set to the user entered value. Duplicates of all the related parameters, expected outputs, and resource entries for the development version are also created and related to the new version. Lastly, any uploaded tools and scripts are duplicated and stored



Publish new versions of this tool or revert to older versions:

Version: 0.1.0

Version Details

Version ID
109

Date Published
2016-04-23

Short Description
Predict the effects a single point mutations using PhD-SNP

Long Description

Required inputs

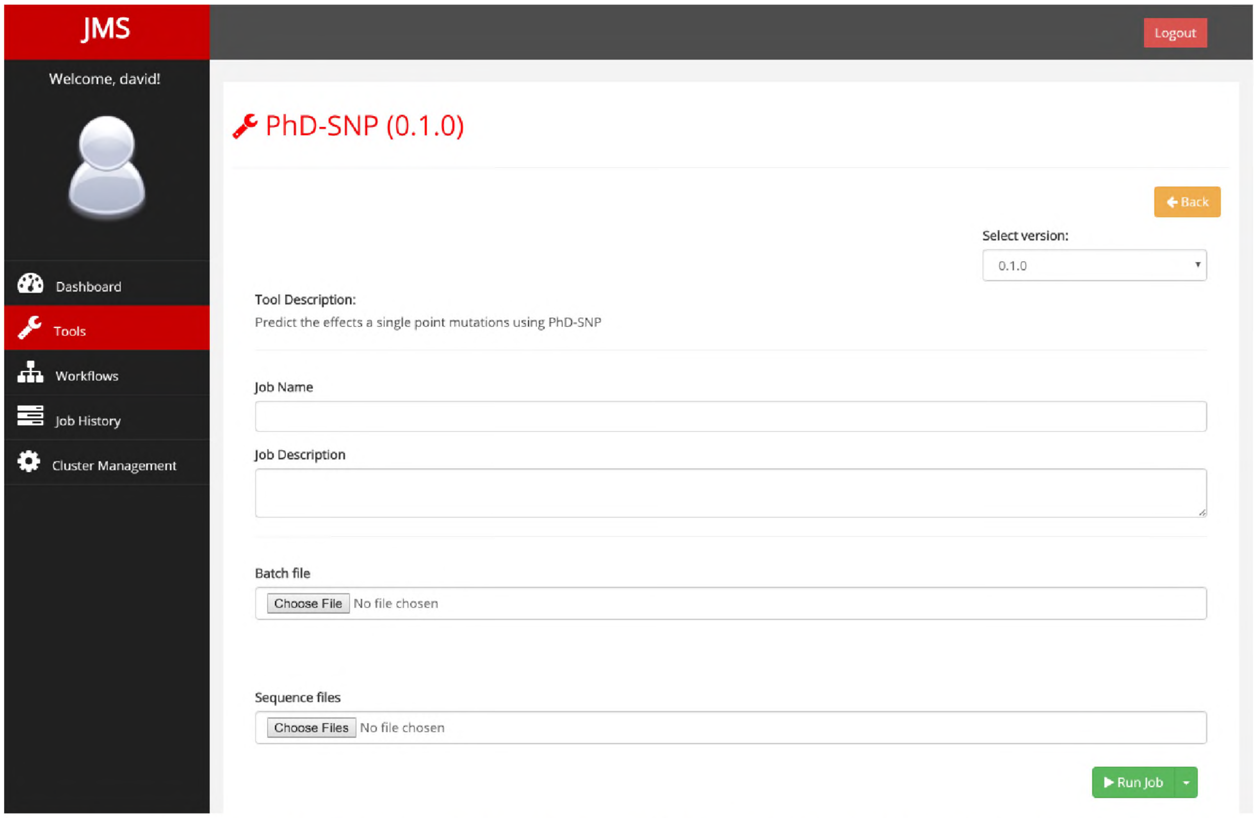
Parameter ID	Parameter Label	Parameter Type	Context	Input By
470	Batch file	5	--input	user
471	Output file	1	--output	system
472	Processes	2	--processes	system
473	Sequence files	5	--sequence-files	user

Fig. 2.15. Version control – a version of a tool can be published by clicking the green "+" in the above screenshot and entering a version number. This will create a snapshot of the development version at the current point in time. The development can be reverted to an older version by selecting the version in the "Releases" list on the left and then clicking on the "Revert" button.

separately to the development version. As such, the new version is a snapshot of the development version at a point in time. Once published, the new version can never be edited again. The development version can instantly be reverted to an old release at any time, however.

2.7.2.3. *Running tools*

To run tools, JMS use the parameter details that were entered when the tool was created to generate a web interface (Fig. 2.16). This interface allows users to enter values for each parameter and submit a job. If JMS is able to successfully run the tool with the supplied parameters, the user is



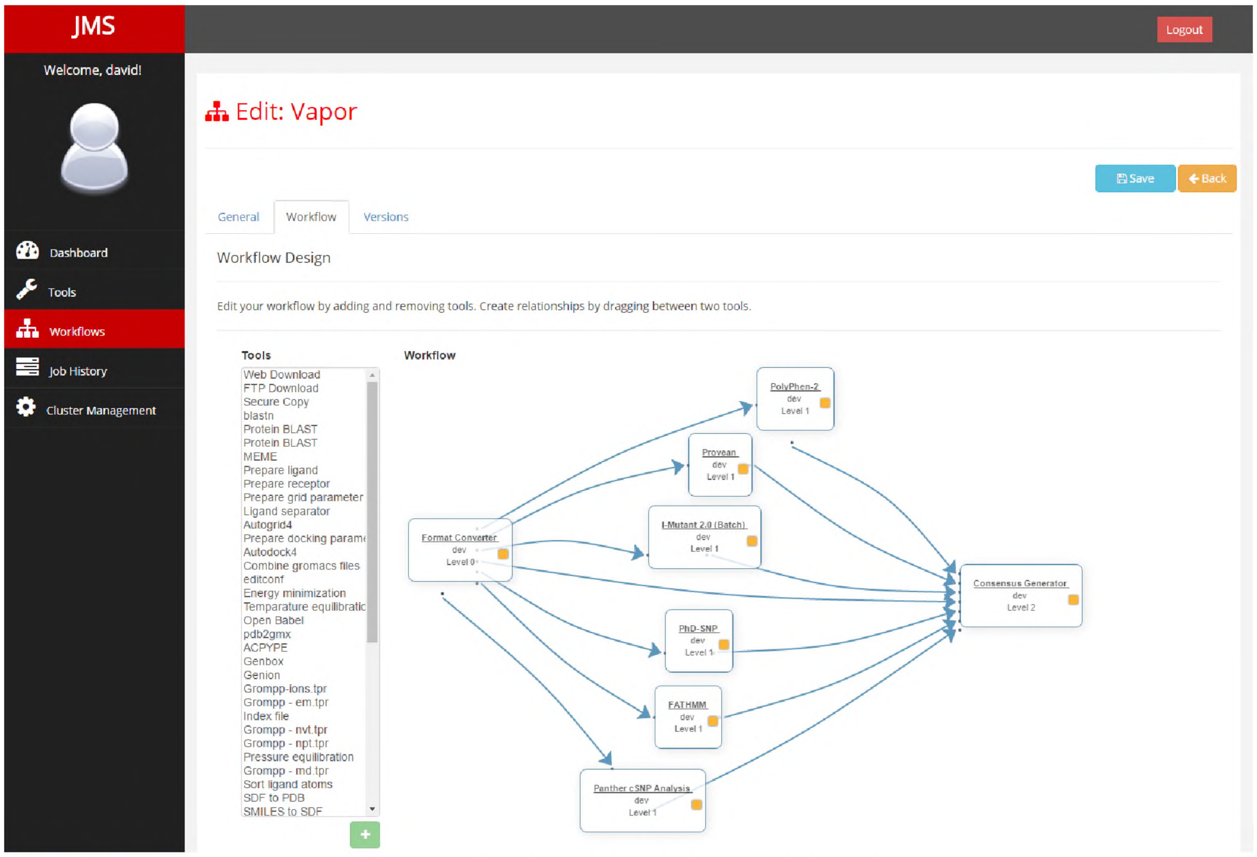
Tool Description:
Predict the effects a single point mutations using PhD-SNP

Fig 2.16. Tool interface – JMS generates an interface for all tools in the system. The interface allows you to enter in a job name and description as well as values for any parameters that are required by the command (unless the value for the parameter is input automatically by the system). The user can also select the version of the tool that they wish to run.

given the option to go to the "Job History" page to monitor the job (discussed in section 2.7.4) or to remain on the input page to run the tool again. The interface also allows the user to enter a job name, which can be used to keep track of jobs, and a job description, which can be used to remind the user of the purpose of the job at a later stage.

2.7.3. Workflow management

The "Workflows" page provides a list of all the workflows that have been added to JMS. These workflows are sorted into several user-defined categories. From this page, users can add new workflows and edit, share, and run existing workflows. As such, the "Workflows" page is modeled closely after the "Tools" page.



Edit your workflow by adding and removing tools. Create relationships by dragging between two tools.

Fig. 2.17. Workflow editor – JMS provides a drag-and-drop workflow editor. Users can add tools to the canvas by double-clicking on the tool in the list on the left-hand side. Tools can be moved around the canvas and dependencies can be created between tools by dragging and dropping.

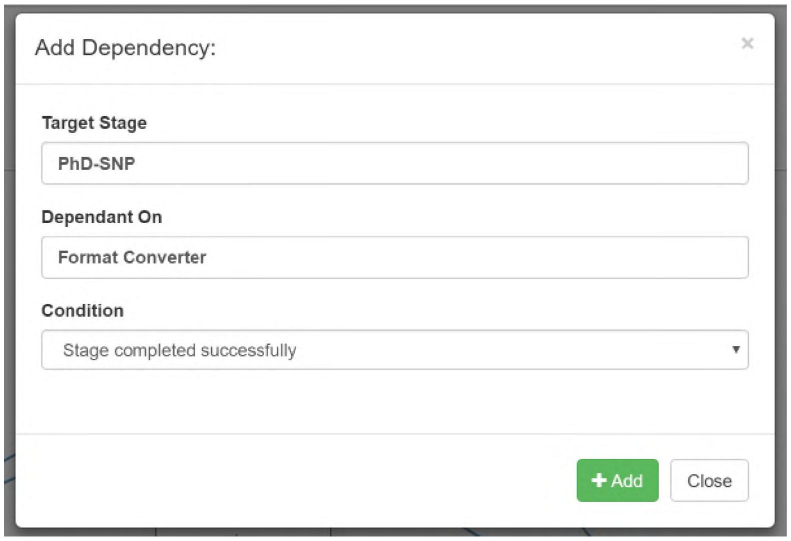
2.7.3.1. Creating/editing workflows

Workflows are made up of one or more tools. They are created by combining tools into pipelines, where the outputs of earlier tools become the inputs of later tools. JMS provides an intuitive drag-and-drop interface, not unlike the one provided by Galaxy, for building workflows (Fig. 2.17). The interface has a select box on the left-hand side, which contains a list of all the tools that the user has permission to run. To the right of this box is the workflow canvas. A tool can be added to the canvas by double clicking it in the list. In addition, multiple tools can be added at once by holding in the *Ctrl* key and selecting the required tools in the list, then clicking on the green "plus" button at the bottom-right of the list.

Once added, the tools will appear in the top-right hand corner of the canvas. On the canvas, a tool appears as a blue, rectangular box with its name and a small, yellow square on the inside. Tools can be arranged on the canvas by dragging them to the desired location.

Once the tools have been arranged on the canvas, the dependencies between the tools must be specified. This can be done by hovering over the yellow block in the first tool and dragging to the second tool. When this is done, the "Add Dependency" dialog box (Fig. 2.18) will appear, which will allow the user to assign a condition to the dependency. When running the workflow, this condition must be satisfied before the second stage begins execution. By default, the condition is simply that the first stage completes successfully.

Double-clicking on one of the tools on the canvas will bring up the "Edit Stage" dialog (Fig. 2.19). This dialog allows the user to specify which version of the tool will be used in the workflow as well as whether this tool can be used as a checkpoint. In addition, parameters that should be

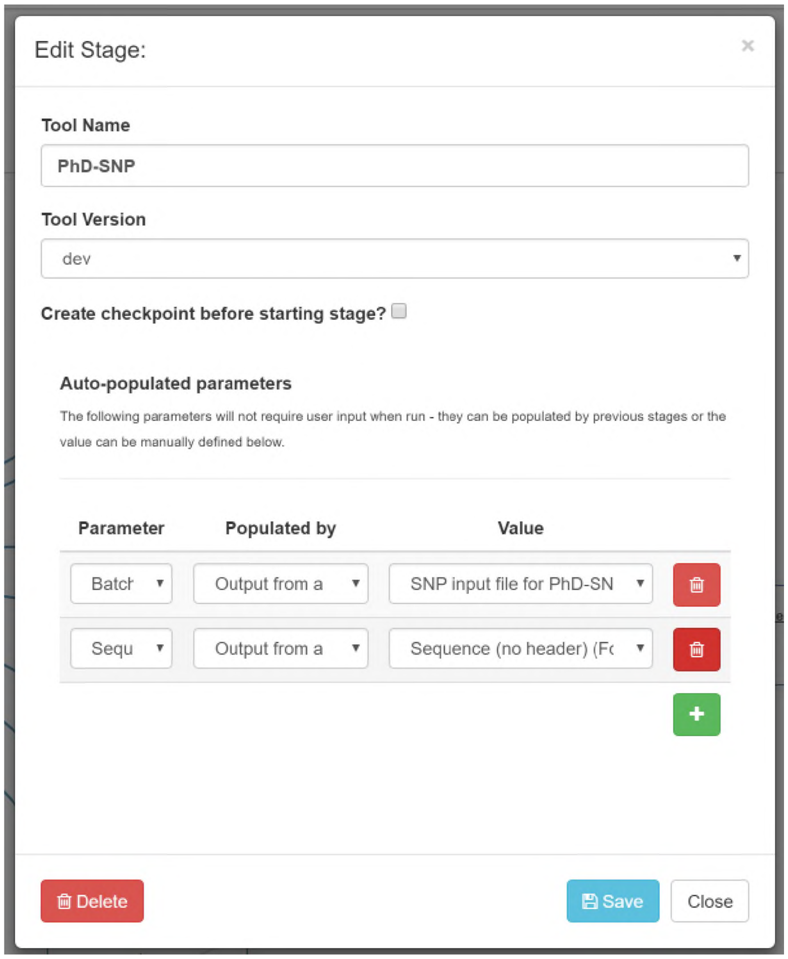

Add Dependency:

Target Stage: PhD-SNP

Dependant On: Format Converter

Condition: Stage completed successfully

Fig. 2.18. Setting dependencies – the dependency dialog allows users to create dependencies between stages by setting conditions on when a stage should or shouldn't run


Edit Stage:

×

Tool Name

PhD-SNP

Tool Version

dev

Create checkpoint before starting stage? ☐

Auto-populated parameters

The following parameters will not require user input when run - they can be populated by previous stages or the value can be manually defined below.

Parameter	Populated by	Value
Batch	Output from a	SNP input file for PhD-SN
Sequ	Output from a	Sequence (no header) (Fc

Fig. 2.19. Editing a stage – the stage dialog allows a user to specify which outputs from a previous stage should be used as inputs to the current stage. It also lets users specify which version of the tool should be used.

automatically populated by the system are specified here. For example, if one of the parameters of the tool should be populated by the output of the previous tool, it can be specified here. Using these options, complex workflows, with multiple stages and routes to completion, can easily be created from just a single interface.

2.7.3.2. Running workflows

As with tools, JMS generates an interface that can be used to execute workflows. This interface is generated from the component tools of the workflow. Where a tool interface simply looks at the parameters of a single tool to generate the interface, the workflows interface must look at the parameters of each of its component tools. Parameters that are automatically populated by the outputs of previous stages must also be considered - no input controls are generated for these parameters as they do not need to be input by the user.

2.7.4. Job management and monitoring

There are two places that provide job management and monitoring functionality in JMS. The first of these places, the Dashboard (Fig. 2.10 & Fig 2.11), provides job monitoring and management functionality via the job queue. As described earlier, users can monitor and cancel jobs via this queue. They can also get more detailed status information for a job by clicking on its job ID in the queue. The dashboard only provides basic job monitoring, however. Its purpose is to provide a high-level overview of activity on the cluster.

On the other hand, the "Job History" page provides in-depth details on all the jobs that the user has ever executed. One of the more unique features of JMS, in comparison to other WMSs, is that it can track jobs that have been submitted to the cluster via other sources. For example, if a user submits a job to the cluster via the command line or even via another WMS such as Galaxy or Ergatis, JMS can pick it up and store it in its own job history. This is achieved via the queue daemon (see section 2.4.2.1.1). As previously explained, the queue daemon polls the job queue of the underlying resource manager. Polling the queue returns the status information for all jobs running on the cluster. This includes jobs that were submitted via other sources. JMS parses all this data and stores it in the database. If it finds a job that has been submitted via another source, there won't be a record of this job in the JMS database. To cater for this, it will create an entry for that job, and from then on, update the entry each time the queue is polled until the job completes.

JMS, tracks four types of jobs. These job types and the differences between them and how they are stored in the database are describe as follows:

1) Tool job

When a tool that is housed in JMS is executed, this job is stored in the database as a “tool job”. Tool jobs consist of a single job stage, but, because the job stage is not part of a predefined workflow, the entry in the *JobStages* table does not link to the *Stages* table. The entry for the job in the *Jobs* table does relate to a tool in the *Tools* table, however.

2) Workflow job

A workflow job is created when a predefined JMS workflow is submitted to the cluster. Workflow jobs are made up of one or more job stages. Each job stage entry links to an entry in the *Stages* table. In addition, the entry for the job in the *Jobs* table relates to a workflow in the *Workflows* table.

3) Custom job

In addition to being able to submit tools and workflows to the cluster, JMS allows users to submit custom commands and scripts. This functionality is useful if a user simply wants to run a quick job on the cluster and doesn’t want to create a whole new tool for it. As such, custom jobs are normally used for once-off, unique jobs. Like tool jobs, custom jobs consist of a single job stage, which does not relate to an entry in the *Stages* table. Unlike tool and workflow jobs, they do not relate to entries in either of the *Tools* or *Workflows* tables. As such, meta data such as the parameters used and the expected outputs of the job are not available for these types of jobs.

4) External job

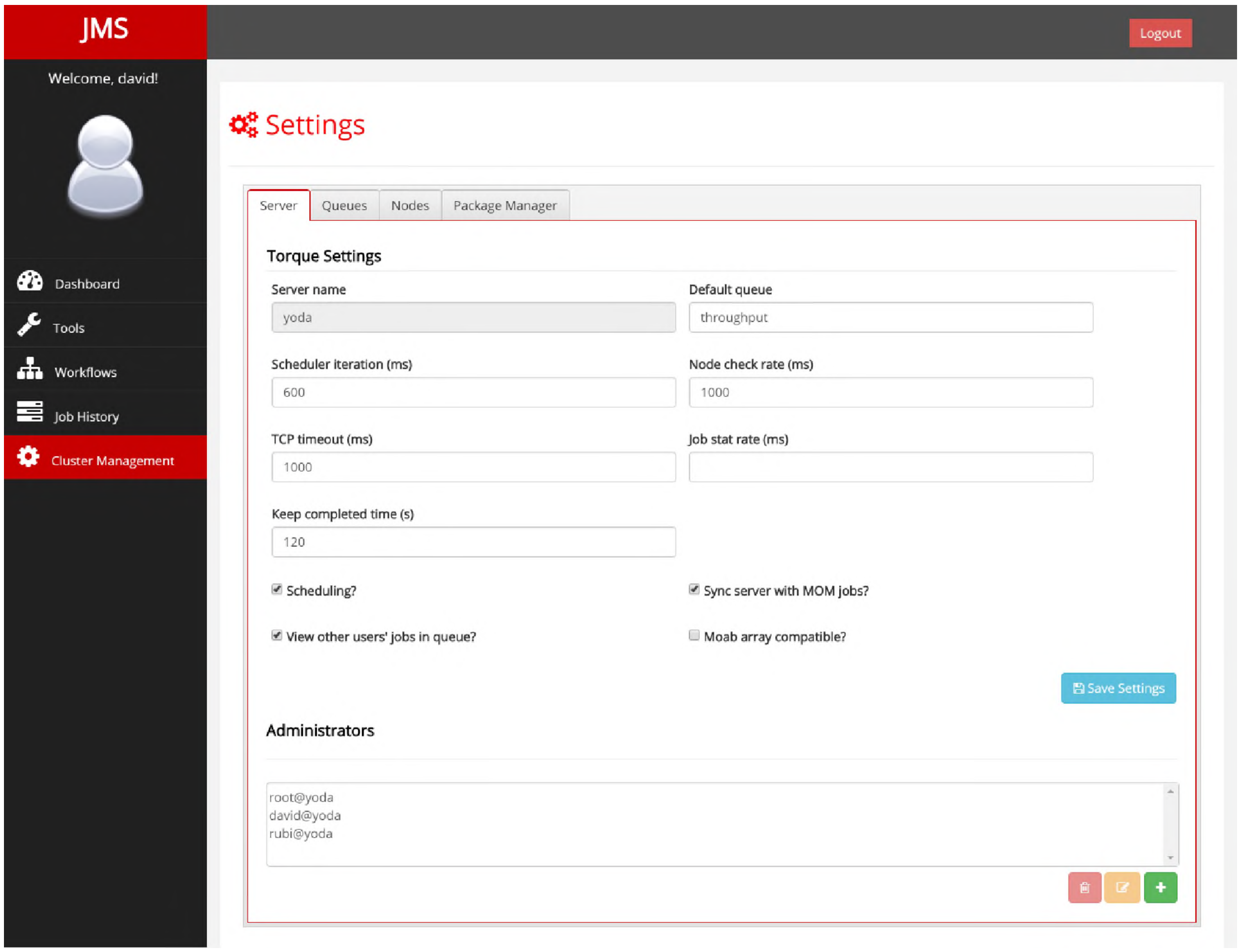
As discussed above, external jobs are jobs that have been submitted by an external source *i.e.* a source other than JMS. These jobs could be submitted via another WMS or even directly via the

CLI. JMS picks these jobs up by polling the job queue. External jobs are stored in a similar way to custom jobs in that they do not relate to either a tool or a workflow. They also consist of only a single job stage, which does not relate to the *Stages* table. External jobs differ to custom jobs in that there is no sure way to determine what commands or scripts were submitted to the cluster. Additionally, with the three previous job types, JMS manages the execution of the jobs on the cluster. This includes the directory hierarchy for the job as well as output and error logs. JMS *does* automatically try to detect where the working directory and logs are for external jobs, however, this depends on the resource manager plugin being used.

On the job history page, JMS displays as many details as possible for each job type. For example, for tools, it will list the expected outputs for the tool as well as the input parameters. For a workflow, it will display these details for each of the tools that made up the workflow. For custom and external jobs, input parameters and expected outputs are not available, and so only the standard information that is available for all job types is displayed. This includes the output and error logs (possibly blank for external jobs), and advanced job details from the resource manager such as computing resources allocated and used, environmental variables, and date and time information. A file manager has also been included in this page, which allows users to browse the working directory and any subdirectories and open and view any text files that were generated as part of the job. Users are also able to download the entire working directory from this page, as well as repeat, share and delete the job from the job history.

2.7.5. HPC cluster management

One of the areas where JMS sets itself apart from other WMSs is its ability to configure and manage the underlying cluster. This is achieved via the resource manager plugins discussed in



Torque Settings

Server name: yoda	Default queue: throughput
Scheduler iteration (ms): 600	Node check rate (ms): 1000
TCP timeout (ms): 1000	Job stat rate (ms):
Keep completed time (s): 120	
[✓] Scheduling?	[✓] Sync server with MOM jobs?
[✓] View other users' jobs in queue?	[ ] Moab array compatible?

Administrators

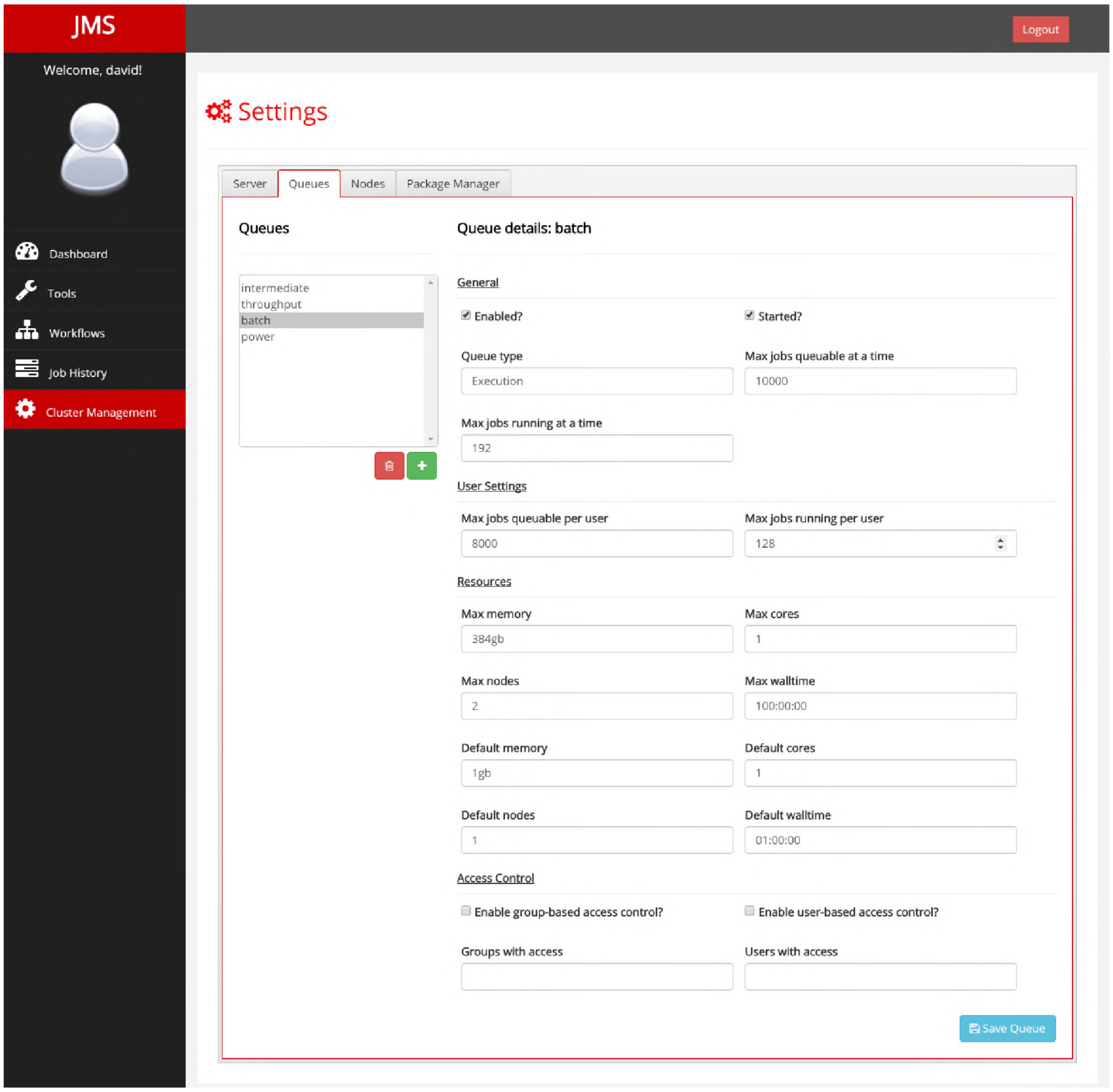
root@yoda
david@yoda
rubi@yoda

Fig. 2.20. Resource manager settings – JMS allows administrators to manage a cluster by tweaking settings on the cluster management page. The settings available to be tweaked on this page are dependent on the underlying resource manager plugin.

sectiom 2.4.2.1.2. As such, the specific cluster management features offered by JMS are dependent on the plugin being used. These features fall into three categories.

Firstly, JMS offers configuration options for the underlying resource manager itself (Fig. 2.20). Using Torque as an example, these settings include options such as who the cluster administrators are, what the default queue is, how long jobs should be displayed in the queue after they have completed, and more technical options such as the scheduler iteration interval and node check rate. Essentially, these are high level settings for the cluster as a whole.

The second category contains configuration and management functionality for queues (Fig. 2.21). Administrators can create, edit and delete queues via the cluster interface. The configuration options for each queue will again be dependent on the underlying resource manager, but will generally include resource allocation settings such as the maximum number of nodes, processing cores, RAM, and wall-time available to jobs submitted to the queue, as well as access control settings (which users or groups have permission to submit jobs to the queue), and more general settings such as the maximum number of jobs that can be running or queued at a time.



Queues

intermediate
throughput
batch
power

Queue details: batch

General

☑ Enabled? ☑ Started?

Queue type: Execution	Max jobs queuable at a time: 10000
Max jobs running at a time: 192	

User Settings

Max jobs queuable per user: 8000	Max jobs running per user: 128

Resources

Max memory: 384gb	Max cores: 1
Max nodes: 2	Max walltime: 100:00:00
Default memory: 1gb	Default cores: 1
Default nodes: 1	Default walltime: 01:00:00

Access Control

☐ Enable group-based access control? ☐ Enable user-based access control?

Groups with access:	Users with access:

Fig. 2.21. Queue configuration – Administrators can create and configure queues from this page. The exact functionality available will again be dependent on the underlying resource manager.

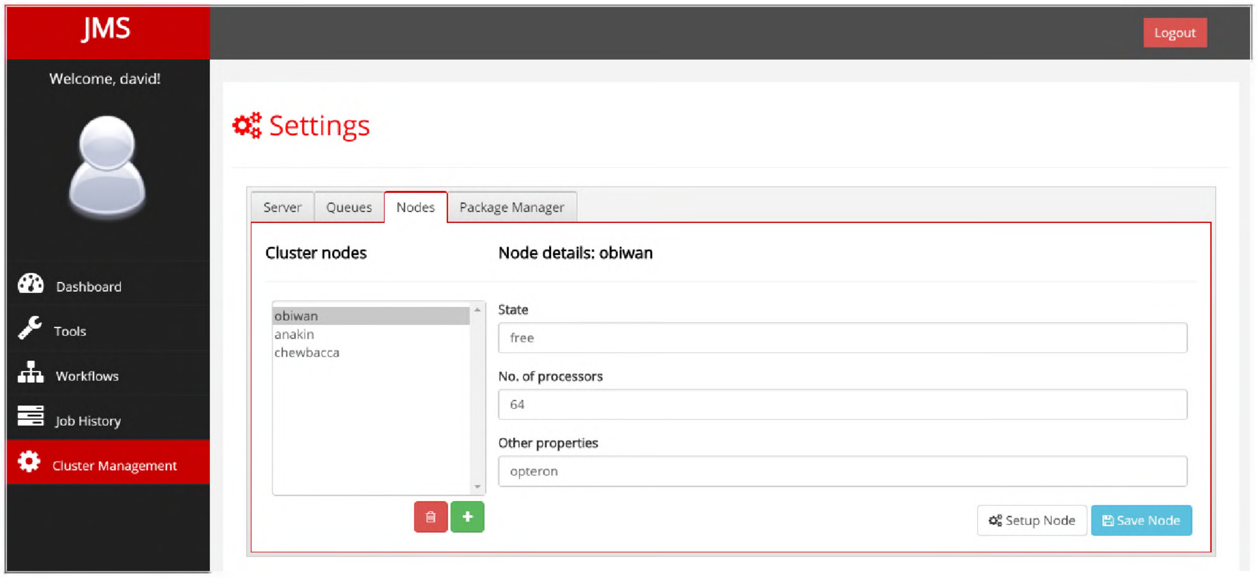


Fig. 2.22. Node configuration – the Nodes tab allows administrators to add, remove and configure nodes. Once again, the exact functionality here is dependent on the underlying resource manager plugin.

The final category focusses on node configuration (Fig. 2.22). Here users can add, edit and delete nodes. As per the other categories, the configuration options available will depend on the resource manager plugin. Our Torque plugin simply allows administrators to specify how many processing cores are available on each node (there is a "Properties" field as well, however, its purpose is only descriptive).

The final tab on the 'Cluster Management' page is the 'Package Management' tab. This is still a work-in-progress and won't be discussed in detail here other than to say that it will allow administrators to install and manage packages on the nodes of the cluster using either Ansible [76], Conda [77], or a combination of the two technologies.

2.8. Comparison with similar software

JMS has been compared to five popular, existing WMSs, namely, Galaxy [46], Ergatis [57], Taverna[50], WImpiBLAST [78], and Yabi [79]. During these comparisons, five features were noted that distinguished JMS.

Firstly, JMS incorporates a web-based code editor, which gives developers the ability to create and edit tools and scripts directly in their web browser. The code editor, an open source JavaScript plugin called Ace [80], provides a number of useful features including automatic indentation, syntax highlighting, and word completion. It is incredibly useful for troubleshooting and testing scripts, as it allows developers to test and then tweak their scripts without ever needing to leave JMS. This can save a considerable amount of time during development and has proved useful during the development of our own tools and workflows (see chapter 3).

The second distinguishing feature is JMS input profiles. Input profiles allow users to create default sets of inputs for tools, that can be used to auto-fill certain fields. When considering whether systems supported input profiles, we required that the users be allowed to create multiple sets of default inputs. Simply allowing a user to set the default inputs for a tool or workflow did not qualify in this regard.

Thirdly, a major feature of JMS and one of the original reasons for developing it is its ability to easily and quickly make tools and workflows available via external interfaces. The same code used to generate the interface for a tool within JMS can be used to generate an interface for an external web server.

Fourth, JMS provides administrator users with the ability to manage the cluster from the web interface. This includes configuring server and queue settings as well as adding additional nodes to the configuration. This functionality is dependent on the resource manager plugin. In this regard, JMS can be used web-based cluster front-end.

Lastly, JMS was the only one of the systems tested that provided a comprehensive dashboard. Although most WMSs provided users with some sort of job queue, JMS is the only system that displays detailed information about the status of the underlying cluster.

A summarized comparison of JMS to other similar and freely available systems is provided in Table 2.4. Because the aims of JMS do not necessarily align exactly with those of other systems, it is difficult to compare features without seeming slightly biased. For example, in Table 2.4, JMS is the only system to have cluster configuration features and a dashboard. This hails from the fact that JMS is a combination of a cluster front-end and a WMS. A system such as Galaxy is a focused WMS and, as such, one would not expect it to have these features. Galaxy does, however, have more extensive workflow features. Notably, it can convert job history into a workflow. This provides non-IT experts with an extremely easy means of creating workflows and is something that sets Galaxy apart from other systems. For the most part, shortcomings of JMS amount from the fact that it is a new system. As such, it has only a small user base and library of tools and workflows. As the system matures, these facets will be rectified. That said, one of the main goals of JMS is to be a platform for developers to create and house new tools and workflows that will be made public via external web servers. In this regard, JMS stands out from the rest.

Features	**Galaxy**	**Ergatis**	**Taverna**	**WImpiBLAST**	**Yabi**	**JMS**
Job management	Yes	Yes	Yes	Yes	Yes	Yes
Workflow management	Yes	Yes	Yes	No	Yes	Yes
File upload/download/view	Yes	Yes	Yes	Yes	Yes	Yes
Development tools	No	No	No	No	No	Yes
Support for multiple resource managers	Yes	No	n/a	No	Yes	No
REST API	Yes	No	Yes	No	Yes	Yes
Input profiles	No	No	No	No	No	Yes
Batch jobs	Yes	No	No	No	No	Yes

Cluster configuration	No	No	No	No	No	Yes
Dashboard	No	No	No	No	No	Yes
Make tools public via external web interfaces	No	No	No	No	No	Yes
Job history to workflow	Yes	No	No	No	No	No
Existing user base and external testing	Yes	Yes	Yes	Yes	Yes	No
Library of existing tools and workflows	Yes	Yes	Yes	Yes	Yes	No

Table 2.4. Comparison between JMS and similar software (table taken from JMS publication[62])

As can be seen in Table 2.4, JMS has both advantages and disadvantages. The lack of a large base of tools and workflows may put off end-users, but the incorporation of development features may attract developers. There is, however, no reason that JMS cannot be run alongside other systems. In fact, one attractive option may be to run a WMS such as Galaxy alongside JMS on the same cluster. Galaxy is established with many existing tools and workflows and a large user base. However, jobs run on the cluster via Galaxy will still be picked by JMS and included in the JMS job history. The JMS job history stores cluster usage details that can be used to generate reports and statistics that may be useful for purchasing and funding purposes. In addition, while users who are more comfortable using Galaxy to run jobs can continue to use Galaxy, developers can use JMS to build tools and workflows and make them public via external web interfaces. In this way, JMS can be seen to complement existing systems.

2.9. Maintenance

To maintain a software project over the long term, one of two things are required. Either, the project must obtain significant funding that will pay developer and administrator salaries, or a large community must be built around the project. Since long term funding is rare in academia, most academic software relies on developer communities. The main way to create such a community is by open sourcing the software, and this is the route that was taken with JMS.

JMS was open-sourced by uploading the code to Github. In the long run, we hope to attract new developers to maintain and further develop the project. To do this, the project will first be made more visible, by deploying it to public platforms, such as the CHPC in Cape Town, where hundreds of researchers will be able to use it. Work is already in progress to do this.

In addition, there are plans to deploy JMS to HPC centers all around Africa. This will further increase the visibility of the platform and attract new developers and funding.

Work is also under way to properly document the JMS software architecture and development process. This will make it easier for new developers to join and contribute.

2.10. Summary & future work

JMS is a web-based workflow management system and HPC cluster front-end. It provides several useful features including the ability to quickly and easily add tools and workflows directly via the web interface, make tools and workflows public via external interfaces, manage and monitor jobs running on the cluster, and manage and monitor the cluster itself. Due to this extensive and broad range of features, JMS uniquely caters to four groups of users including system administrators, tool and workflow developers, end users, and finally, developers of external web servers, who want a quick and easy way to add tools and workflows to their own interfaces.

There are several features that make JMS unique. As mentioned before, JMS caters to a broad range of users. To our knowledge, there is no other system designed to cater to as broad a range of users. JMS combines extensive workflow management features with cluster management and monitoring features to ease the burden of using HPC resources. Additionally, JMS provides a unique means of adding tools – a process that can be done entirely via the web interface – which greatly increases the speed at which tools and workflows can be developed and debugged. A code editor has been incorporated so that tools and script can be updated directly via the JMS interface. After running a tool, output logs are also made available which developers can use for debugging purposes. Finally, the JMS web API has been developed from the ground up with the specific goal of allowing tools and workflows to be made public via external interfaces. This means that data can be obtained from the JMS API, which can be used to automatically generate an interface for a specific version of a tool. To our knowledge, JMS is the only system that offers such a service.

A unique security system has also been developed for JMS. This system has been called the Impersonator and allows users to authenticate using their SSH credentials. The main purpose of this system, however, is to allow JMS to impersonate users on the system. The Impersonator allows JMS to spawn a process as the logged in user, meaning that jobs can be submitted to the cluster using the correct credentials and with the correct user permissions.

Although JMS offers several unique features, there are numerous other systems in the WMS space that also provide attractive features. Although JMS has been built as an independent system, it can also work well alongside existing systems. This is because JMS is able to track all jobs submitted to the cluster, including those submitted via external sources. As such, the JMS job history is extensive and offers powerful reporting potential.

As JMS is a new system, there is still a lot of work to be done. Future plans include building in Ansible [76] support for improved node management as well as Conda [77] support as part of a tool/package management feature. This will allow us to develop an online repository of tools that can be downloaded and installed into any JMS instance at the click of a button. It will also facilitate the sharing of tools and workflows between researchers from different institutions.

Additionally, we are looking at improving the installation process. Development of a Docker [81] image for JMS is being considered for this. Alternative options would be to create Debian and/or Redhat Package Manager (RPM) packages so that JMS could be installed with a simple command.

The job history page is also being redesigned to provide a greater emphasis on the results returned by tools and workflows. This will improve the user experience and usability of the system.

Finally, improved data provenance and tracking needs to be added to JMS. Currently, if a user would like to use the result of a previous job as input to another job, they must download the result from the job history page first, before uploading it again on the input page for a new job. This is an inefficient and wasteful process. Future work will allow JMS to keep track of data files from previous jobs and allow them to be used as input files to future jobs without needing to re-upload them.

3. JMS bioinformatics applications

3.1. Introduction

The previous chapter described the design and development of JMS as a platform to house and execute tools and workflows on a HPC cluster. These tools and workflows can target any form of computational science. One of the aims of this project, however, was the development of bioinformatics-related applications, specifically structural bioinformatics and variation analysis tools. In this chapter, we describe several bioinformatics tools and workflows that we have developed and housed within JMS. Much of the work in this chapter has been published [61,82,83].

3.2. SANCDB submission tool

The South African Natural Compounds Database (SANCDB) [82] was designed and developed by Rowan Hatherley, a post-doctoral student from the Research Unit in Bioinformatics (RUBi) at Rhodes University. SANCDB stores compounds that have been isolated from organisms found in South Africa. Data was manually extracted from literature in the form of theses and peer-reviewed publications. The result was a database consisting of information about compounds including the names, formulas, references (*i.e.* literature in which the compound was isolated), both anecdotal and confirmed uses, and activity (*e.g.* anti-cancer, anti-bacterial). In addition, structures for the compounds were generated and made available for download in various formats including SDF, MOL2, PDB, and SMILES. SANCDB makes all this data available via a user-friendly web server.

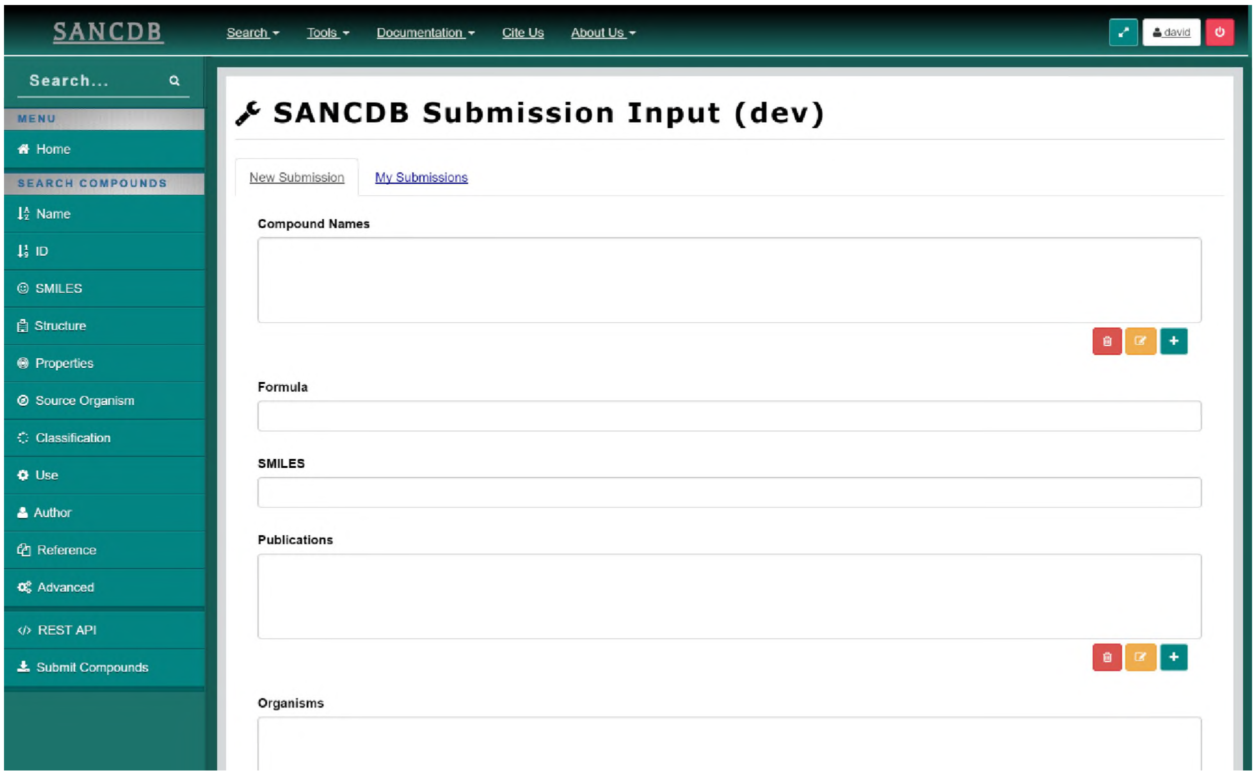


Fig. 3.1. SANCDB submission input form – input form generated on the SANCDB website using the same JavaScript code used when generating JMS interfaces. Data submitted via this interface will be in the perfect format to be received by JMS, but will lack JMS credentials.

One of the features provided by the SANCDB web server is a curated user submission pipeline (Fig. 3.1), which allows users to upload their own compounds to the SANCDB database, giving their work more exposure and providing SANCDB with a second source of compounds (over-and-above actively searching literature). This pipeline consists of a series of Python scripts that have been uploaded to JMS and used to make JMS tools.

3.2.1. Backend scripts

The submission pipeline consists of several Python scripts. The first script is simply used to accept user input and write it to a file. This file will eventually be examined by curators for mistakes in the user input. The second script accepts a compound in SMILES format and uses CORINA [84] to generate a structure in SDF format. The third and fourth scripts take a compound in SDF format as a parameter and uses Open Babel [85] to convert the compound to PDB and MOL2 format respectively. The last script uses GAMESS [86] to minimize a PDB file.

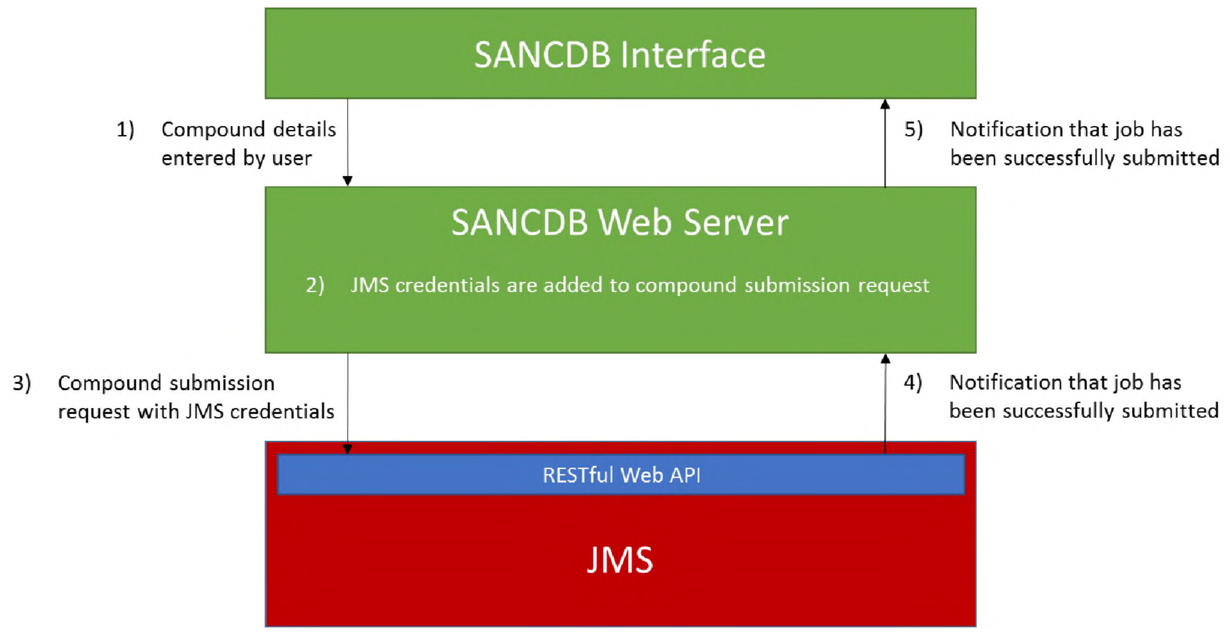
3.2.2. JMS implementation

For each of the scripts described above, a tool was created in JMS. The scripts were uploaded to the respective JMS tools and the details required to execute the scripts were entered via the JMS interface. The tools were then combined via the workflow creator/editor interface to form a JMS workflow. The final outputs of this workflow are the SDF structure generated by CORINA, the PDB and MOL2 files generated by Open Babel, and the minimized PDB file generated by GAMESS.

3.2.3. Interface

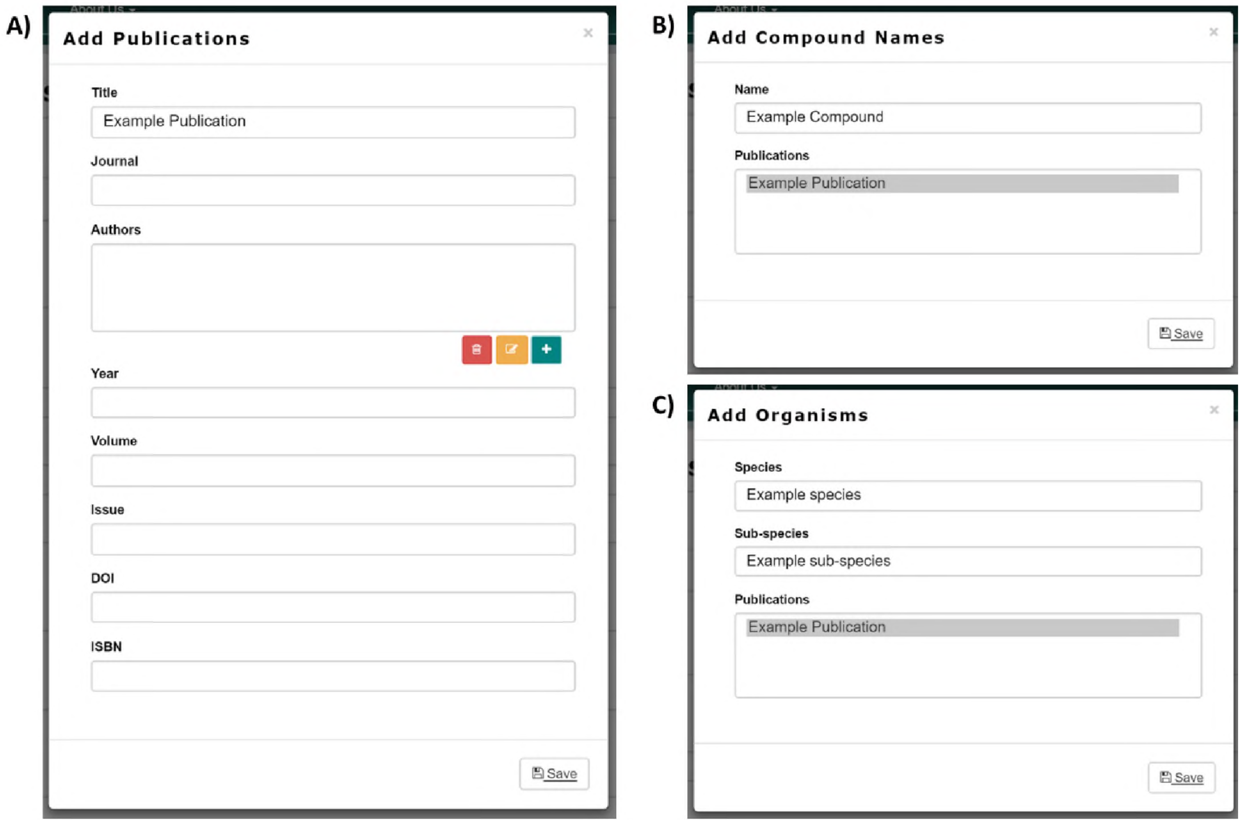
To generate the interface for the submission pipeline in the SANCDB website (Fig. 3.1), the same JavaScript code that is used to generate interfaces in the JMS website was used. The JavaScript code queries the JMS web API to obtain information on what parameters are required by the particular version of the tool being used. This information includes the type of each parameter (Table 2.1), which is used to decide what sort of input controls should be used. Currently, this still requires some manual work on the external interface, but in future, an entirely automated JavaScript plugin will be developed to simplify this process even further.

When a compound is submitted to SANCDB, the job details are submitted in a format that is compatible with the JMS web API. This is because SANCDB interface has been generated using the same code that is used within JMS itself. However, when submitting jobs to JMS, the submitting user must be authenticated with a JMS username and password. Storing these


SANCDB Interface
1) Compound details entered by user
5) Notification that job has been successfully submitted
SANCDB Web Server
2) JMS credentials are added to compound submission request
3) Compound submission request with JMS credentials
4) Notification that job has been successfully submitted
RESTful Web API
JMS
Fig. 3.2. *JMS submission process – the compound details are submitted to the SANCDB web server where the required authorization credentials are added to the request. The request is then forwarded to JMS, which starts executing the pipeline before sending a notification back to the SANCDB web server that the job has been successfully submitted. The SANCDB user is then notified of this.*

credentials on the client-side of SANCDB (*i.e.* in the JavaScript code) would be insecure as they would be visible to anyone. Therefore, when a compound is submitted, the request is routed via the SANCDB server before being forwarded on to JMS. JMS credentials are stored securely on the server-side of SANCDB and are added to the submission request before it is forwarded to JMS. JMS then executes the workflow on the cluster. This process is illustrated in Fig. 3.2.

The SANCDB interface provides an example of how complex objects and related parameters are used. The *Compound Names*, *Publications*, and *Organisms* parameters are complex objects. A *Publication* complex object also accepts an *Author* complex object as a nested object (Fig. 3.3. A), while the *Compound Names* (Fig. 3.3. B) and *Organisms* (Fig. 3.3. C) objects each use a related parameter to relate the respective compound names and the organisms that the compounds were found in back to one or more publications.



Add Publications

Title: Example Publication

Journal:

Authors:

Year:

Volume:

Issue:

DOI:

ISBN:

Save

Add Compound Names

Name: Example Compound

Publications: Example Publication

Save

Add Organisms

Species: Example species

Sub-species: Example sub-species

Publications: Example Publication

Save

*Fig 3.3. Complex objects and related parameters. **A)** 'Publications' are complex objects that contain a number of fields including the title of the publication, the journal it was published in, and the year it was published in. Complex objects can also include nested complex objects – in this case, the 'Authors' field of a 'Publication' is a list of Author complex objects. **B)** The 'Compound Name' complex object contains a 'Name' field and a 'Publications' field. The 'Publications' field is an example of a related parameter as it allows users to select 'Publications' that were input in A. **C)** Similarly to 'Compound Names', 'Organisms' contain a 'Publications' field, which is a related parameter.*

3.3. PRIMO

Homology modeling is a structural bioinformatics technique which allows researchers to predict the structure of a protein sequence, referred to as the "target sequence", based on one or more "template structures". A template structure is a protein that is a homolog of the target protein and whose structure has been solved experimentally. Homology modeling uses the known structure of the template to predict the structure of the target and is based on the idea that a protein's structure is more conserved than its sequence [87].

The Protein Interactive Modeling (PRIMO) pipeline [83] was developed in conjunction with Rowan Hatherley and Michael Glenister of RUBi. PRIMO is an interactive, web-based homology modeling pipeline that was established with two aims in mind:

1) Make homology modeling easy for users with little or no experience
2) Give experienced users sufficient interactive capabilities to customize modeling runs

Existing homology modeling servers often provide users with a "black box". For example, using the Phyre2 web server [88], the user supplies a protein sequence to the server and the server performs some calculation and produces a model of the protein structure. No other user interaction is required. Although this may be useful for users without any homology modeling experience, it leaves the user blind to the actual work that occurs on the server. The user must simply trust that the server is correct. There is also no way for the user to customize the modeling run and affect the quality of the resultant models.

Other servers, such as HHPred [89] and SWISS-MODEL [16], provide users with limited interactive ability, such as the ability to select which templates that get used for modeling. HHPred, which uses MODELLER [11] for the final stage of the modeling process, also allows users to edit the PIR file required by MODELLER.

Without performing homology modeling manually (by downloading and installing the required software on a local machine), users of existing web-based software are left with very little say in how their models are produced. As such, we realized that there was a gap in the homology modeling server space for a homology modeling platform – a server that provides all the required tools to perform homology modeling and allows the user to run those tools in whichever way they prefer to obtain their end result.

3.3.1. Backend scripts

The PRIMO backend scripts were written by Rowan Hatherley. They are essentially a set of Python scripts, which cater to each stage of the homology modeling process. Currently, the PRIMO homology modeling platform consists of three stages:

1) Template identification

The PRIMO template identification script gives users the option of uploading their own templates, specifying templates using their PDB ID and chain (*e.g.* 4HHB:A), or choosing between HHPred and BLAST [90] to automatically identify suitable templates. Future plans involve adding Fugue [91], and PSI-BLAST [92] as alternatives to HHPred and BLAST for automatically identifying templates.

2) Target-template alignment

The PRIMO alignment script gives users the option of selecting an alignment program as well as the mode to run the alignment program in. Currently, it gives users the option of running T-COFFEE [93], MAFFT [94], MUSCLE [95] and CLUSTAL Omega [96] alignment programs. Additionally, the alignment generated by HHPred or BLAST (depending on which program was chosen during the template identification stage) can be used. When using T-COFFEE, users can also select to run in 3D-COFFEE [97] mode, which takes structural information into account when aligning sequences. When using MAFFT, users can select to run in Psuedo-homologs mode, which is an in-house implementation of MAFFT-Homologs – essentially 50 homologs are fetched for each of the target and template sequences before performing a multiple sequence alignment.

3) Modeling

PRIMO uses MODELLER to perform the actual modeling calculations. Before this can be done, a PIR file is generated from the alignment. PRIMO automatically trims the alignment to optimize it for modeling. It then also generates the actual modeling script. Users can select how many models should be generated, the refinement level, and whether the generated models should be superposed afterwards.

3.3.2. JMS implementation

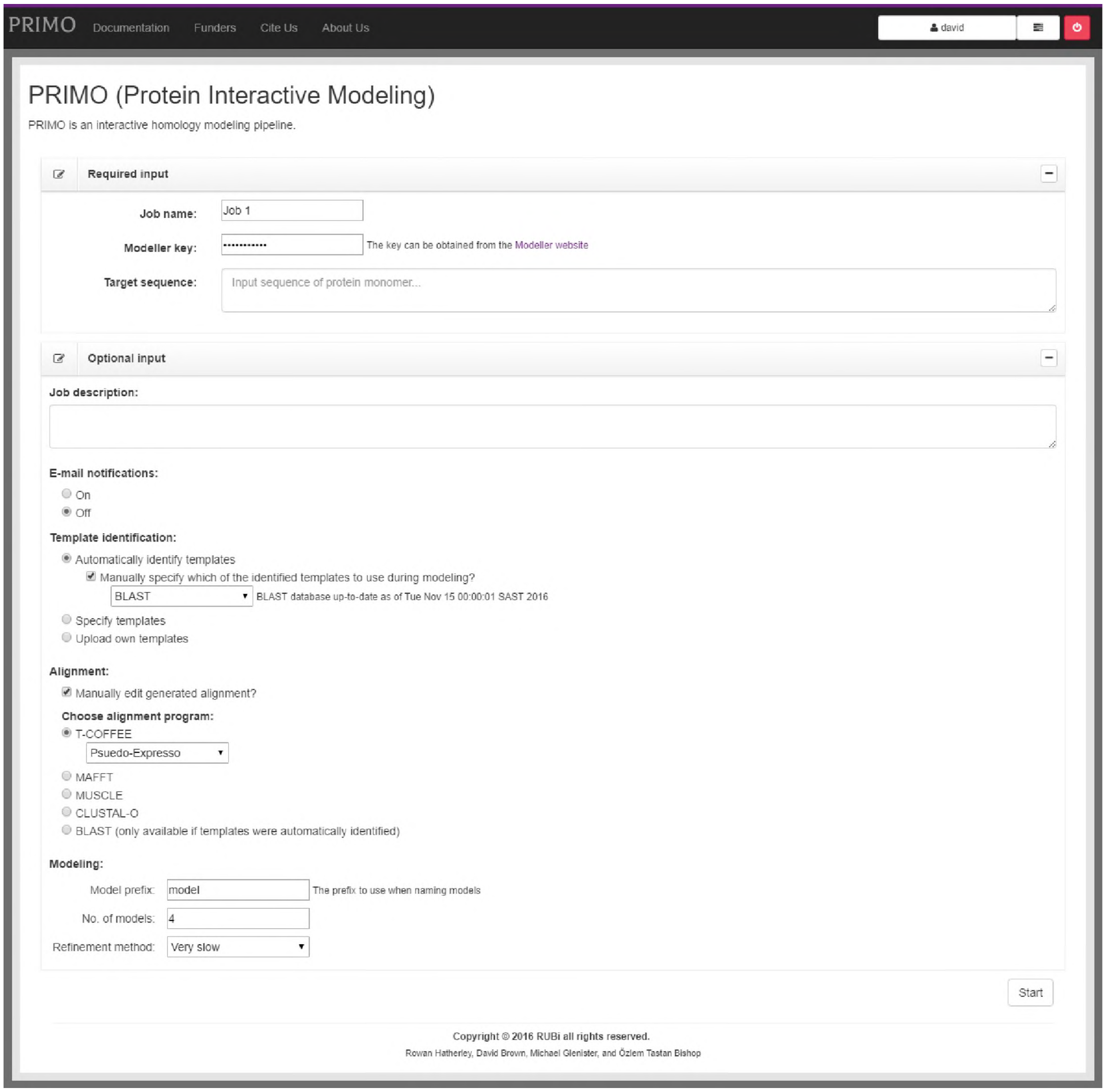
PRIMO was added to JMS as three separate tools representing the three homology modeling stages. Each of the backend scripts was uploaded to the respective JMS tools and the relevant information on how to run the scripts was entered via the JMS interface. The three tools were not combined into a JMS workflow, however, as we wanted PRIMO to pause between each stage. This was to let users view and edit the results of each stage before continuing to the next stage.

3.3.3. Interface

Unlike the SANCDB submission pipeline, which used the JMS web API to generate an interface, a custom interface was built from scratch for PRIMO (Fig. 3.4). Jobs were still submitted to JMS via the web API, but by manually building the interface, we gained greater flexibility in the design. The PRIMO interface was developed with the help of Michael Glenister and can be accessed at https://primo.rubi.ru.ac.za.

As with SANCDB, the PRIMO website does not access JMS directly for security reasons. Instead, when a job is submitted, the request is first sent to the PRIMO server, where the job details are stored. Because a custom interface was built for PRIMO, the data is sent to the PRIMO web server is a slightly different format than what JMS expects. In order to make it compatible with the JSON API, the data is reformatted on the PRIMO server. This involves adding the JMS credentials to the request, as with SANCDB, and mapping the input values received from the interface to the correct JMS parameter IDs.

To display modeled structures and their sequence alignment to the templates, we built the PV-MSA JavaScript plugin. PV-MSA is a JavaScript wrapper that combines Protein Viewer (PV) [98], a web-based molecular viewer, and the BioJS Multiple Sequence Alignment (MSA) Viewer [99], a



PRIMO (Protein Interactive Modeling)

PRIMO is an interactive homology modeling pipeline.

Required input

Job name: Job 1

Modeller key: ••••••••••• The key can be obtained from the Modeller website

Target sequence: Input sequence of protein monomer...

Optional input

Job description:

E-mail notifications:
On [✓ Off]

Template identification:
[✓ Automatically identify templates]
[✓ Manually specify which of the identified templates to use during modeling?]
BLAST BLAST database up-to-date as of Tue Nov 15 00:00:01 SAST 2016
Specify templates
Upload own templates

Alignment:
[✓ Manually edit generated alignment?]
Choose alignment program:
[✓ T-COFFEE]
Psuedo-Expresso
MAFFT
MUSCLE
CLUSTAL-O
BLAST (only available if templates were automatically identified)

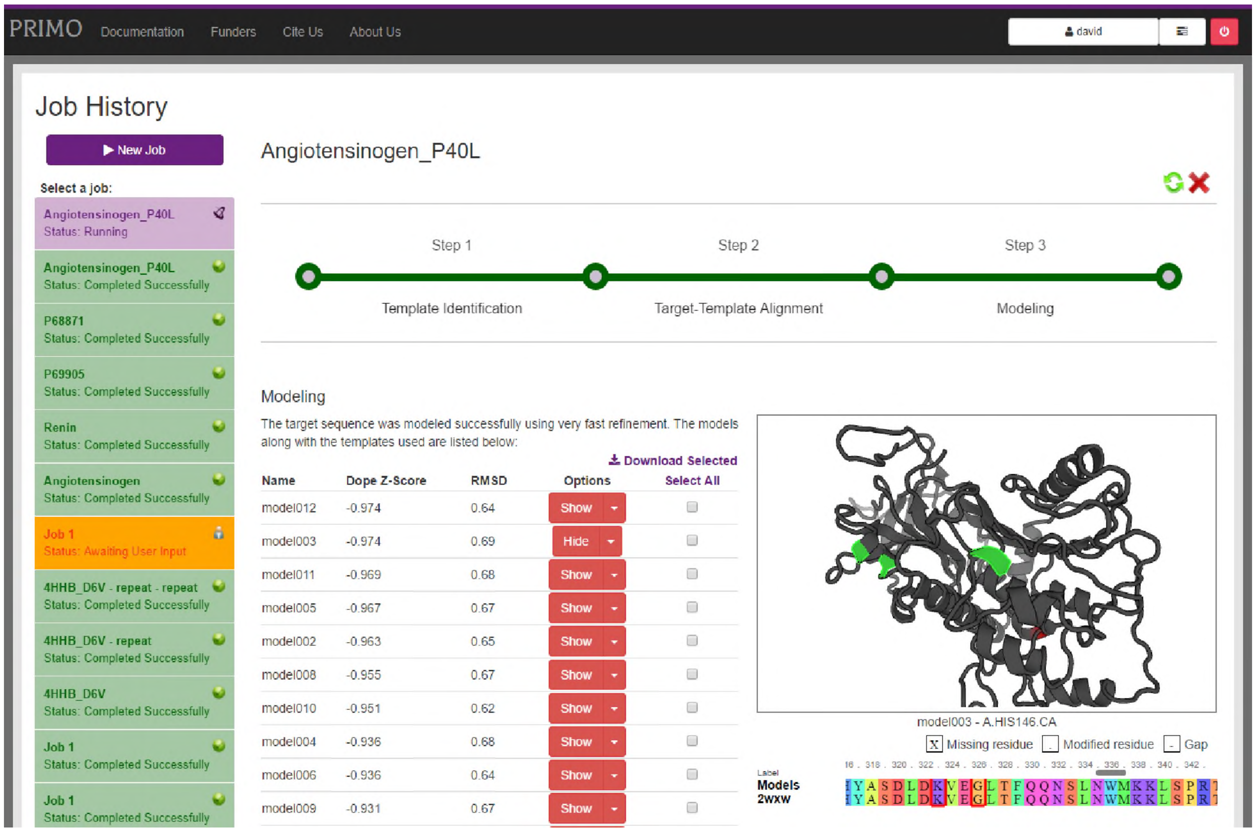
Modeling:
Model prefix: model The prefix to use when naming models
No. of models: 4
Refinement method: Very slow

Start

Copyright © 2016 RUBi all rights reserved.
Rowan Hatherley, David Brown, Michael Glenister, and Özlem Tastan Bishop

Fig. 3.4. *PRIMO submission interface – the custom interface that was built to submit PRIMO jobs. This interface will not submit data in the format required by JMS. The request is therefore reformulated on the PRIMO web server before being sent to JMS.*

web-based sequence alignment viewer, into a single, functional tool. PV-MSA allows developers to link sequences in the alignment to structures in the molecular viewer. Users can then either select residues in the alignment, at which point the residue will be highlighted in the structure, or select residues in the structure, at which point they will be highlighted in the alignment. PV-MSA also provides a far easier API to use and can be set up in a web page with only a few lines of code. It provides easy functions to add, remove, show, and hide structures and alignments and allows linked structures and alignments to be shown and hidden independently of each other. These functions make PV-MSA much easier to use for developers than if they were using the PV and MSA plugins independently. However, the new functions do not currently cover all the functionality provided by PV and MSA. Fortunately, the PV and MSA objects that are being wrapped by the PV-MSA plugin are easily accessible via the PV-MSA API and can be used to



Modeling

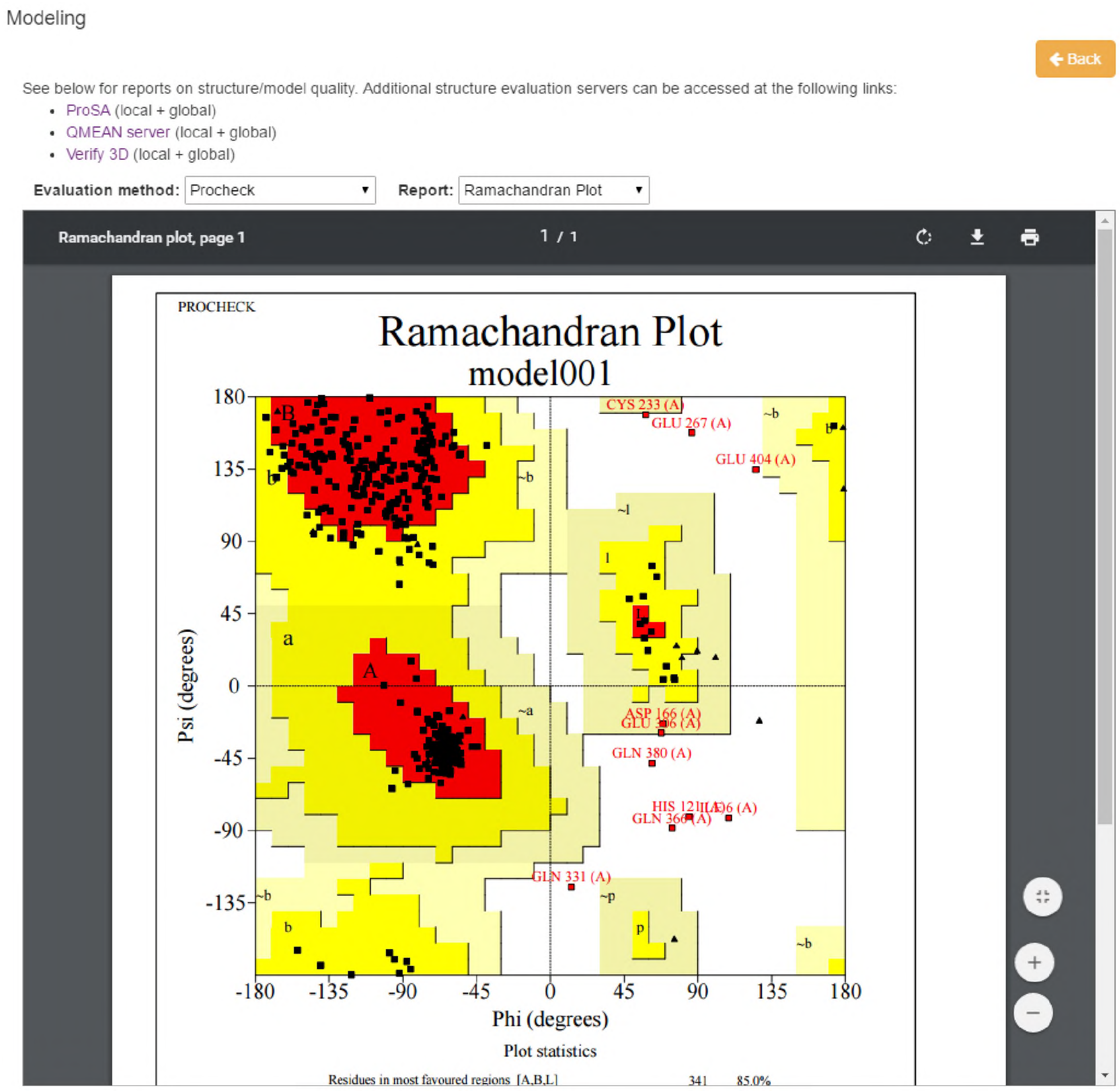
The target sequence was modeled successfully using very fast refinement. The models along with the templates used are listed below:

Download Selected

Name	Dope Z-Score	RMSD	Options	Select All
model012	-0.974	0.64	Show	☐
model003	-0.974	0.69	Hide	☐
model011	-0.969	0.68	Show	☐
model005	-0.967	0.67	Show	☐
model002	-0.963	0.65	Show	☐
model008	-0.955	0.67	Show	☐
model010	-0.951	0.62	Show	☐
model004	-0.936	0.68	Show	☐
model006	-0.936	0.64	Show	☐
model009	-0.931	0.67	Show	☐

```
Models  IYASDLDKVEGLTFQQNSLNWMKKLSPRT
2wxw    IYASDLDKVEGLTFQQNSLNWMKKLSPRT
```

Fig 3.5. PV-MSA – used by PRIMO to display the templates used and models generated by the process as well as the alignment between the templates and models. Selecting a residue highlights that residue in green on the structure and surrounds the residue in a red box in the alignment.



Modeling

Back

See below for reports on structure/model quality. Additional structure evaluation servers can be accessed at the following links:

- ProSA (local + global)
- QMEAN server (local + global)
- Verify 3D (local + global)

Evaluation method: Procheck **Report:** Ramachandran Plot

Ramachandran plot, page 1 1 / 1

Plot statistics

Residues in most favoured regions [A,B,L] 341 85.0%

Fig. 3.6. Procheck Ramachandran plot - Procheck produces a number of different charts depicting model quality. Different charts can be viewed by selecting them in the "Report" dropdown menu.

implement advanced features. The PV-MSA wrapper has been open-sourced and made available at https://github.com/davidbrownza/PV-MSA. PV-MSA's use in the PRIMO interface is illustrated in Fig. 3.5.

3.3.4. Model evaluation

The PRIMO results table depicted in Fig. 3.5 provides the Dope Z-score of the model and the root mean square deviation (RMSD) between the model and the templates. These measures by themselves are not sufficient. As such, the ability to run Procheck [100] to evaluate models was built into the web server. This can be done by clicking the dropdown menu arrow on the "Show" button (visible in Fig. 3.5) and clicking on the "Structure Evaluation" button.

On the Procheck results page (see Fig. 3.6), there are also links to other evaluation tools that can be used to further validate the quality of the model. These tools include ProSA [101], QMEAN [102], and Verify3D [103].

3.4. VAPOR

In recent years, numerous tools have been developed to predict the possible effects of variants on protein stability and function. On their own, these tools have varying degrees of accuracy, with benchmarks showing that they seldom get more than 75% of predictions correct [104,105]. To gain a greater degree of certainty, researchers usually run a number of these tools and combine the resulting predictions. This can be a time-consuming process, as each tool must be run individually for all of the variants or sequences that need to be analyzed. To cater for this, tools such Meta-SNP [104] and PredictSNP [105], which automatically run and combine the results of various prediction tools, have been developed. These tools have shown that generating a consensus from the results of multiple tools using machine learning techniques can further improve accuracy.

Meta-SNP combines the results of four existing tools, PANTHER [106], SIFT [107], PhD-SNP [108], and SNAP [109], and uses a novel machine-learning algorithm that can more accurately predict whether a variant is deleterious or not. By doing this, it can correctly predict the effects of 79% of variants in their test datasets, which is 3% higher than the best performing component tool.

Similarly, PredictSNP is a consensus classifier that combines the results of six existing tools, namely, MAPP [110], PhD-SNP, PolyPhen-1 [111], PolyPhen-2 [112], SIFT, and SNAP. PANTHER and nsSNPAnalyzer [113] were also initially evaluated for inclusion into PredictSNP, however, their performance, in terms of both accuracy and the percentage of the test dataset that they were able to evaluate, was found to be unsatisfactory. Benchmarks using the PMD-Uniprot dataset (a subset of the Protein Mutant Database (PMD) containing only variants for which there are associated sequences in the UniProt database [114]) and MMP dataset (a dataset consisting of 13 massively mutated proteins) found that PredictSNP outperformed Meta-SNP by approximately 3%.

Both Meta-SNP and PredictSNP are available in the form of publicly accessible websites. A standalone version of the PredictSNP consensus classifier is also available, however, this requires that the component tools be run manually before feeding the results to the PredictSNP classifier. As such, these tools cannot be installed on a local machine to analyze thousands of sequences and millions of variants. Part 2 of this thesis describes the establishment of a database containing all known human Single Nucleotide Variants (SNV). This database was made accessible via a user-friendly website. As part of this website, we wanted to provide a tool to predict whether non-synonymous SNVs in the database were damaging or had destabilizing effects. As with PredictSNP and Meta-SNP, we also wanted to calculate consensus scores using multiple existing tools. Thus, we developed the Variant Analysis Portal (VAPOR). VAPOR is a workflow that combines the predictions of Provean [115], PolyPhen-2 [116], PANTHER [117], PhD-SNP [118], FATHMM [119], I-Mutant 2.0 [120] and MuPRO [121].

3.4.1. Backend scripts

VAPOR consists of eight backend scripts and tools. Firstly, the six variant analysis tools mentioned above were installed on the server. Wrapper scripts were created for four of the six tools, PolyPhen-2, PhD-SNP, PANTHER, and I-Mutant 2.0, to make them easier to use. A format conversion script was then developed to convert a sequence and a list of variants to the formats required by all six of the tools. Finally, a consensus generator script was created that merged the results of the respective tools into a single table

3.4.1.1. PolyPhen-2

Running PolyPhen-2 usually requires that the user set up a scratch directory and then run two independent scripts. Firstly, the `run_pph.pl` script, which extracts annotations for several structural and sequence databases and calculates a number of evolutionary conservation scores, must be executed. This script takes a list of variants as input and can also be supplied with the path to the scratch directory and sequence file. The output of this step is then supplied as input to the PolyPhen-2 probabilistic classification tool (`run_weka.pl`), which produces the final predictions.

A small bash script, `polyphen2.sh` (Appendix A), was created, which wraps the above scripts into a single command. This new script accepts a path to the input file, output file, sequences file, and scratch directory. When the script is executed, it first checks to see if the scratch directory exists. If the scratch directory does not exist, it is created along with the required sub-directories. Following this, `run_pph.pl` is executed. If `run_pph.pl` executes successfully, `run_weka.pl` is executed and the result is written to the output file. If either of the PolyPhen-2 scripts fails, an appropriate error message is written to the terminal.

3.4.1.2. PANTHER

As with PolyPhen-2, running PANTHER consists of more than one tool. Firstly, the `pantherScore.pl` script must be run to classify user input sequences against the PANTHER database. Next, the `snp_analysis.pl` script is run to classify the user input variants. The `snp_analysis.pl` script takes the output of the `pantherScore.pl` as one of its arguments.

To run this pipeline, a bash script, `panther.sh` (Appendix A), was created, which accepts a path to a Fasta file, variants file and output file. The PANTHER commands require a temporary directory to be made to store intermediate files. The `panther.sh` script creates the temporary directory and the executes the two commands described above, writing the final result to the specified output file.

3.4.1.3. PhD-SNP

PolyPhen-2 and PANTHER both accept a list of variants as input. This allows a large number of variants to be analyzed all at once. Similarly, Provean and FATHMM also accept a list of SNPs. On the other hand, PhD-SNP accepts a single SNP at a time. As such, given a protein sequence and a list of 40 variants to analyze, PhD-SNP would need to be manually run 40 times. To cater for this, a wrapper script was written, which accepts a list of variants and automatically runs PhD-SNP for each variant in the list. The script, `PhD-SNP_batch.py` (Appendix A), was written in Python and accepts a path to an output file as well as a number indicating how many instances of PhD-SNP should run in parallel. Parallelism is achieved by spawning a user-defined number of processes to run PhD-SNP and then continuously monitoring each of the spawned processes to detect when they complete. As soon as a process completes, a new process is spawned to analyze the next variant in the list. This continues until all variants have been analyzed.

3.4.1.4. I-Mutant 2.0

As with PhD-SNP, the I-Mutant 2.0 script analyzes a single variant at a time. To cater for this, the same process as is described above was followed to create a batch script, `i-mutant2.0_batch.py` (Appendix A), which could analyze a list of variants in parallel using I-Mutant 2.0.

3.4.1.5. Format conversion

Each of the six tools that make up VAPOR accept a protein sequence and a list of variants as input. Unfortunately, they also each expect different formats for the variants and sometimes even the sequence. To cater for this, a Python script, `format_converter.py` (Appendix A), was created that accepts a protein sequence, in Fasta format, and a list of variants, in HGVS format using one-letter amino acid codes, as input. A new variant file is created for each of the six tools containing all the variants from the input file in the respective formats. Only one other sequence file format is required, which is the sequence without the Fasta header. This format is used by I-Mutant 2.0 and PhD-SNP.

3.4.1.6. Consensus generation

Each of the six tools that make up VAPOR output their results to a specified output file. A Python script, `consensus_genrator.py` (Appendix A), was written that reads in the contents of these output files and merges the results into a single, tab-delimited file. This part of the VAPOR pipeline is not yet complete as future work will be done to develop a consensus classifier like PredictSNP.

3.4.2. JMS implementation

Each of the variant analysis tools that were installed as part of VAPOR can be used with the help of environment modules. Environment modules modify environmental variables on a server to allow certain tools or versions of tools to be usable. For example, when the Provean environment module is loaded, it prepends the Provean `bin/` directory, which contains all the Provean executables, to the PATH environment variable. As such, the Provean executables can be used from anywhere without providing an absolute or relative path to the executable.

Each of the wrapper scripts described in the previous sections were saved to the /bin directories of the respective tools. This means that they are accessible when the environment module for that tool is loaded. To use these scripts from JMS, a JMS tool was created for each of the analysis tools.

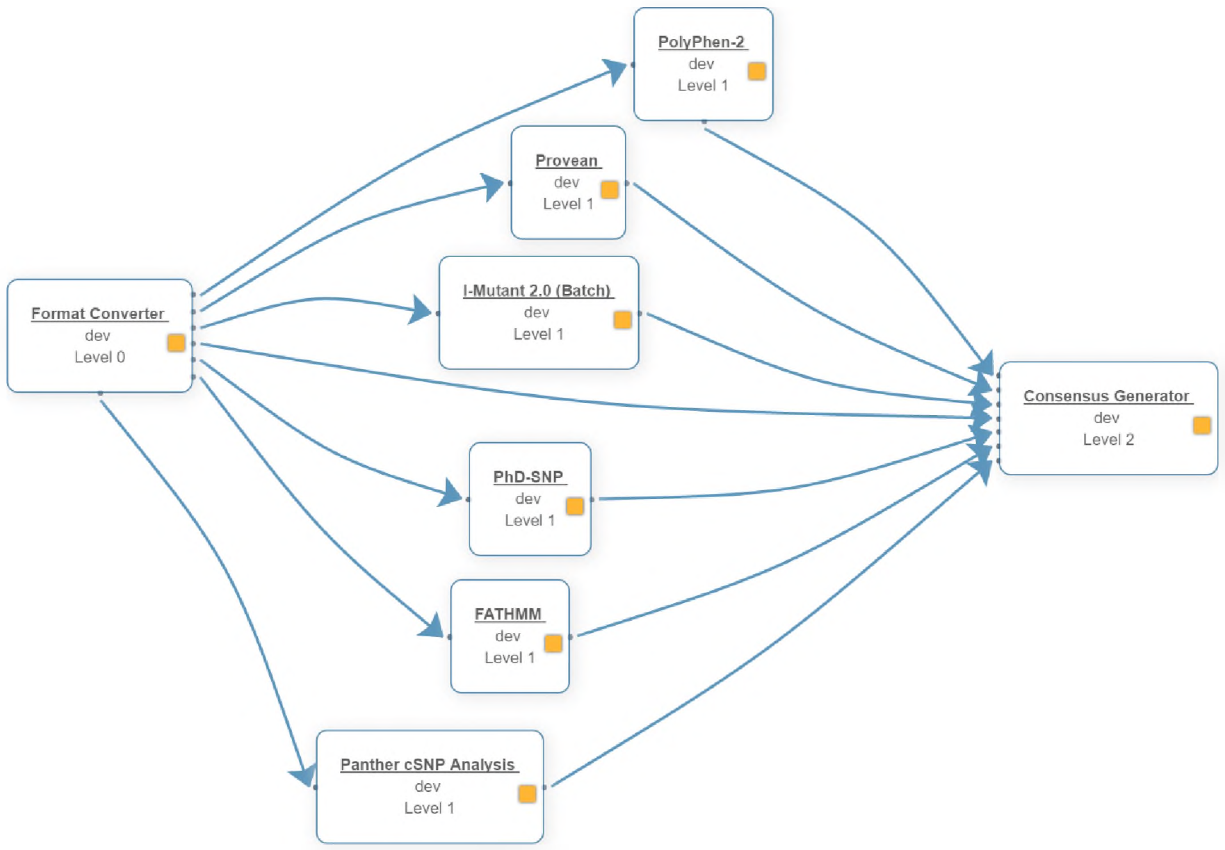


Fig. 3.7. VAPOR workflow – screenshot of the VAPOR workflow within JMS. The workflow consists of a preparation stage (Format Converter), and execution stage, and consensus stage (Consensus Generator).

A simple script was then created within each JMS tool that accepts the arguments required by its respective analysis tool, loads the required environment module to put the necessary scripts in the PATH, and then executes the analysis tool (or the wrapper for the analysis tool) by passing the user-input arguments to it. The commands and parameters required to execute these scripts were entered for each tool via the JMS interface.

JMS tools were also created for both the `format_converter.py` and `consensus_generator.py` scripts. In fact, these scripts were created directly within JMS using the provided code editor.

A JMS workflow was then created consisting of three stages (Fig. 3.7.). The first stage consists of only the Format Converter tool, the second stage consists of the six variant analysis tools, and the final stage consists of the Consensus Generator tool. When the Format Converter tool runs, it produces the required inputs for the variant analysis tools. Once it completes, these inputs are passed to the respective tools, which can all run in parallel. The Consensus Generator tool waits for all the analysis tools to complete before merging the results.

3.4.3. Interface

The VAPOR workflow has been incorporated into the HUMA web server, the topic of section two of this thesis. As such, the interface will be described there.

3.5. Tools & workflows

Over and above the tools and workflows described already, which are used by our external web servers (SANCDB, PRIMO, and HUMA), several additional tools and workflows have been built and housed within JMS. These include a docking workflow and a molecular dynamics workflow, which has produced results that have since been published [122].

3.5.1. Molecular docking with AutoDock

The molecular docking pipeline was added to JMS by David Penkler of RUBi. It was built by pipelining various scripts and binaries from AutoDock4 [123] and AutoDock Tools (ADT). Protein-ligand docking using AutoDock consists of six tools/scripts.

3.5.1.1. Workflow description

The first two stages, *receptor preparation* and *ligand preparation*, can be run in parallel as they are independent of one another. Both stages accept a PDB file as input. These PDB files represent the protein receptor and ligand, respectively. The prepared receptor and ligand are output from these stages in pdbqt format and used as input to the next stage, *grid map parameterization*. Three additional user input parameters must be supplied to this stage, namely, grid center coordinates, grid spacing, and number of points. The output from this stage, a grid parameter file, is used as input to the fourth stage, *grid map generation*. This stage generates various grid map files, which are used as input for the final stage. Before the last stage can be run, however, a fifth stage, *docking parameterization*, must be run to create a docking parameter file, which is also used as input to the

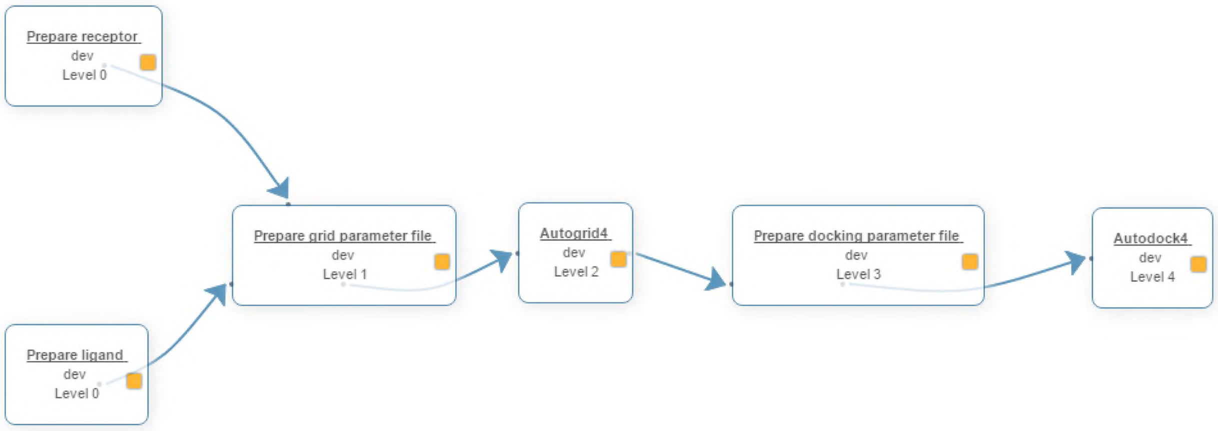


Fig. 3.8. *Molecular docking workflow – a workflow made up of various AutoDock scripts and utilities to perform protein-ligand docking.*

final stage. The final stage, the actual *docking run*, can then be executed. This stage can take hours to complete, but will produce a docking log file containing the final docking results.

3.5.1.2. JMS implementation

Each tool/script that forms part of the AutoDock workflow was used to create a standalone tool in JMS. As per usual, the details required to execute the scripts as well as the required parameters and expected outputs were entered. The workflow creator/editor interface was then used to create a JMS workflow from the standalone tools (Fig. 3.8.).

3.5.2. Molecular dynamics with GROMACS

Thommas Musyoka, a PhD student at RUBi, used GROMACS 4.5.5 [124] to design and build a molecular dynamics pipeline to be used with protein-ligand complexes or protein-only structures. This workflow could be used in conjunction with the docking pipeline described above to analyze promising docking results in greater detail.

3.5.2.1. Workflow description

As an initial input, the workflow accepts a PDB file. If there is a ligand present in the PDB file, the ligand and protein are first separated into separate PDB files. The pdb2gmx GROMACS utility was run on the protein PDB file and ACPYPE [125] was used on the ligand PDB file to generate a force field parameter file, topology file and GROMACS co-ordinate file for each. The co-ordinate files for the protein and ligand are then combined again. The combined co-ordinate file is solvated in a "box" using the editconf and genbox GROMACS utilities. The genion GROMACS utility is then used to neutralize the entire system. Finally, the mdrun and grompp tools areused to minimize the system based on a user-supplied parameter file. These tools are again used to equilibrate the system and then again to perform the final molecular dynamics simulation.

3.5.2.2. JMS implementation

The molecular dynamics workflow described was implemented in JMS in a similar way to the docking workflow. The Python script used for separating the protein from the ligand as well as the Perl script that was used to combine the co-ordinate files were uploaded to JMS. For the rest of the tools, the command field was simply supplied with the absolute path to the utilities installed on the cluster. Once all the individual tools were tested and found to be working, they were combined into a JMS workflow (Fig. 3.9.).

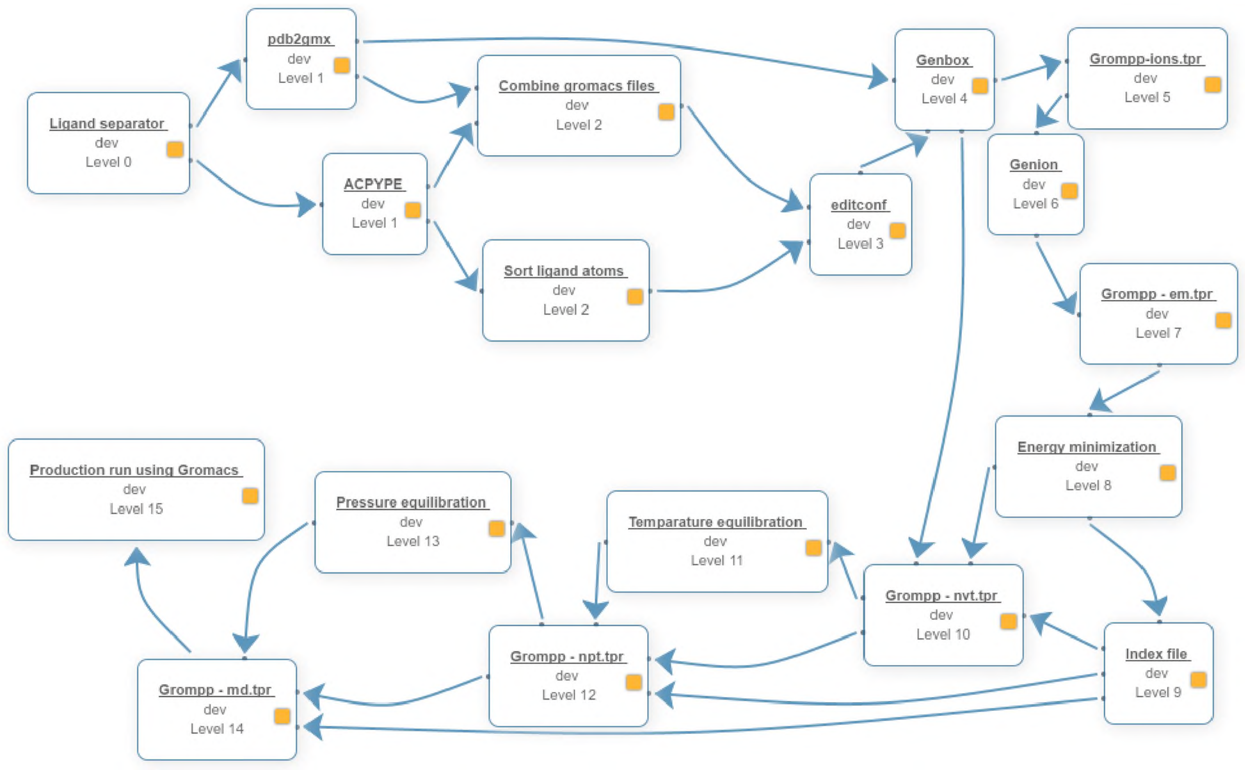

Ligand separator dev Level 0
pdb2gmx dev Level 1
ACPYPE dev Level 1
Combine gromacs files dev Level 2
Sort ligand atoms dev Level 2
editconf dev Level 3
Genbox dev Level 4
Grompp-ions.tpr dev Level 5
Genion dev Level 6
Grompp - em.tpr dev Level 7
Energy minimization dev Level 8
Index file dev Level 9
Grompp - nvt.tpr dev Level 10
Temparature equilibration dev Level 11
Grompp - npt.tpr dev Level 12
Pressure equilibration dev Level 13
Grompp - md.tpr dev Level 14
Production run using Gromacs dev Level 15

Fig. 3.9. *Molecular dynamics workflow – a workflow made up of scripts and utilities from various packages to perform molecular dynamics.*

3.5.3. All tools

Other than workflows, JMS has been populated with several bioinformatics and other tools. Table 3.1 provides a list of all the tools currently housed in JMS, segmented into categories, and including the tools that were used to create the previously described workflows.

Category: Administration	
E-mail	Send an e-mail to a list of e-mail addresses
Category: Data Retrieval	
Web Download	Download data with an HTTP request
FTP Download	Download data with using the File Transfer Protocol (FTP)
Secure Copy	Fetch a file using the scp utility
Category: Docking Studies	
Prepare ligand	Generate PDBQT ligand file with Autodock v4.2.6 scripts
Prepare receptor	Generate PDBQT receptor file with Autodock v4.2.6 scripts
Prepare grid parameter file	Generate GPF file with Autodock v4.2.6 scripts
Autogrid4 (v4.2.6)	Run Autogrid software to generate docking grid
Prepare docking parameter file	Generate DPF file with Autodock v4.2.6 scripts
Autodock4 (v4.2.6)	Run Autodock simulation
Category: Format Conversion	
SDF to PDB	Convert from SDF to PDB using Open Babel v2.3.2
SMILES to SDF	Convert from SMILES to SDF using Corina v3.60
SDF to MOL2	Convert from SDF to MOL2 using Open Babel v2.3.2
Category: Homolog Searching	
Protein BLAST (v2.2.31)	Search for sequence homologs using blastp
Nucleotide BLAST (v2.2.31)	Search for sequence homologs using blastn
Category: Homology Modeling	
Template Identification	Identify templates using HHPred v3.0.1
Target-template Alignment	Align sequences using various alignment programs
Modeling	Model protein using MODELLER v9.15
Category: Miscellaneous	
GAMESS Minimization	Minimize a PDB structure with GAMESS (01 May 2013)
Category: Molecular Dynamics	
Ligand Separator	Separate protein and ligand into separate PDB files
Combine Gromacs files	Combine co-ordinate files into a single file
editconf (v4.5.5)	Create infinite, neutral system around protein and ligand
mdrun (v4.5.5)	Run molecular dynamics simulation
grompp (v4.5.5)	Generates various file required for molecular dynamics
ACPYPE (2014-08-27)	Prepare ligand force field

pdb2gmx (v4.5.5)	Convert PDB to Gromacs co-ordinate file
Genbox (v4.5.5)	Generate box and add solvent
Genion (v4.5.5)	Neutralize the system
Category: SANCDB	
DerivatizeME	Generate derivatives of a compound
SANCDB Submission	Accept compounds input by user
Category: Sequence Alignment	
MAFFT (v7.245)	Align sequences with MAFFT
Category: Variant Analysis	
PolyPhen-2 (v2.2.2)	Analyze a list of variants with PolyPhen-2
Provean (v1.1.5)	Analyze a list of variants with Provean
PANTHER (v1.03)	Analyze a list of variants with PANTHER
PhD-SNP (v2.0.7)	Analyze a list of variants with PhD-SNP
I-Mutant 2.0 (v2.0.7)	Analyze a single variant with I-Mutant 2.0
I-Mutant 2.0 (batch)	Analyze a list of variants with I-Mutant 2.0
FATHMM	Analyze a list of variants with FATHMM
Format conversion	Convert input variants to the formats required by other tools
Consensus generator	Merge results of multiple variant analysis tools

Table 3.1. List of all tools currently housed in JMS

3.6. Conclusion

In this chapter, the bioinformatics applications of JMS have been described by way of example. The workflows and tools that have been described are real-world systems, some of which have already been published, used to produce published results, or are currently under review.

Firstly, the SANCDB submission pipeline provides a means for researchers to upload their own isolated compounds to SANCDB. This pipeline was published as part of the original SANCDB paper [82].

PRIMO provides a user-friendly web server, which can be used to quickly and easily produces protein structure models by using homology modeling. The PRIMO web server was recently published [83].

VAPOR is a variant analysis workflow, housed in JMS, which aggregates results from multiple variant analysis tools. VAPOR has been incorporated into the HUMA web server (see part 2). The HUMA paper has been submitted for publication.

Additionally, tools and workflows within JMS, such as the molecular dynamics workflow have been used to produce results, which have now been published [122]. JMS is also commonly used in the RUBi lab to monitor jobs that are running on the cluster, find useful information from log files on the job history page, and store and version in-house tools and scripts. It is also used by lab members to look up useful information such as server settings and queue resource limits on the cluster management page.

Considering that JMS is still new in comparison to existing systems, the general utility of the system is impressive. To our knowledge, it is the only system that has been designed to be used in such a wide variety of ways.

Part 2: Establishment of a database and web server for the analysis of genetic variation in humans

4. Analyzing genetic variation

4.1. Human Heredity and Health in Africa (H3Africa)

Since the conclusion of the Human Genome Project in 2003, the rate of generation of new data in the biological sciences has been increasing at an exponential rate. Next Generation Sequencing (NGS) technologies have allowed genomes to be sequenced faster, more accurately, and at a fraction of the cost. This has resulted in an explosion of biological data and the revitalization of the field of bioinformatics [126].

Two undertakings in particular, the 1000 Genomes Project [127] and the International HapMap Project [128], have resulted in enormous amounts of data being generated. These projects focused on sequencing large numbers of genomes from various populations to uncover links between disease and variation. Knowledge gained from large scale sequencing projects can help to understand susceptibility and resistance to disease in a given population and facilitate the advent of personalized medicine, where treatments are tailored to individual patients [129]. Unfortunately, for various reasons, African populations have been somewhat neglected in the datasets produced. This means that Africans are in danger of missing out on the benefits of the global, genomic revolution.

One of the reasons that African populations have been neglected is that, up until recently, sequencing projects were generally undertaken by non-African research groups. African institutions lacked the skills and expertise, as well as the funding, to undertake such projects. To address this issue, the Human Heredity and Health in Africa (H3Africa) Initiative was founded with the aim of building research capacity on the African continent [130]. An important goal of this project was that research would be performed in Africa, by Africans, and that data generated would be stored and analyzed in Africa first. As such, numerous projects were launched not only to start sequencing African genomes, but also to train students and researchers, purchase and build infrastructure, and develop the necessary skills and tools to analyze the data when it arrived.

Additionally, H3Africa aims to create networks of African researchers. The initiative consists of groups (or "nodes") from universities and research organizations, both public and private, all around Africa. These nodes are encouraged to collaborate and share ideas, data, and expertise.

A significant portion of the work done by the H3Africa initiative is bioinformatics work. This includes, but is not limited to, the development of variant calling pipelines, Genome Wide Association Studies (GWAS), and the creation of data transfer and archiving systems. The H3Africa Bioinformatics Network (H3ABioNet) was established to handle this work.

As is the case with H3Africa, H3ABioNet is made up of various nodes from around Africa. The Research Unit in Bioinformatics (RUBi) at Rhodes University is one of these nodes. As part of H3ABioNet, RUBi was tasked with establishing a database and web server to store an analyze novel genetic variation uncovered via the H3Africa sequencing projects. While other groups in H3ABioNet are predominantly focused on sequence level analysis of variation, RUBi is a structural bioinformatics group. Structural bioinformatics is focused on the structure, movement and interaction of biological macromolecules, such as proteins, in three-dimensional space. As such, our goal was to focus on the downstream analysis of variation *i.e.* the analysis of variation at the protein level. The investigation, design, and development of this web server and database, which has been named the Human Mutation Analysis (HUMA) web server and database, is the topic of this section of this thesis.

We previously published a paper [131] in which we discussed the role of structural bioinformatics with regards to Single Nucleotide Variant (SNV) analysis and drug discovery. As part of this paper, we reviewed existing variation databases and SNV analysis tools and proposed a protocol for analyzing SNVs using structural bioinformatics. The remainder of this chapter will be dedicated to summarizing the discussions from this paper, but in the context of the design of HUMA.

4.2. Associating variation with disease

One of the main reasons for analyzing genomes is to associate phenotypes, such as diseases, with variation. As part of the H3Africa project, thousands of individuals from various populations around Africa are being sequenced with the aim of identifying SNPs linked to disease in African populations. To identify SNPs associated with disease, various sequence-level techniques can be employed, including GWAS and Candidate Gene Association Studies (CGAS). These techniques associate variation with diseases (or other phenotypic traits) by comparing the genomes (in the case of GWAS) or selected genes (in the case of CGAS) of healthy patients with those of unhealthy patients. If a SNP occurs at a significantly higher rate in the unhealthy patients, it can be said to be associated with a disease (or phenotype).

4.3. Retrieving SNV datasets

The purpose of this chapter is to discuss the importance of structural bioinformatics when it come to the analysis of SNVs. Before analysis of SNVs can be carried out, however, variant datasets must be retrieved from somewhere and filtered to the point where structural studies become feasible. Below, we introduce several existing, widely used variation databases and discuss the shortcomings of these databases with regards to structural studies.

4.3.1. Variation databases

One of the challenges of bioinformatics is storing the enormous amounts of data being generated by NGS projects. In line with this, various databases have been developed to store variation

Database	Description	Link	Reference
Category: Observed variants in different populations			
COSMIC	Cancer-associated variants	http://cancer.sanger.ac.uk/cosmic	132
dbSNP	Short variation	http://www.ncbi.nlm.nih.gov/projects/SNP/	133
EGA	Private variation archive	https://www.ebi.ac.uk/ega/home	134
EVA	Public variation archive	http://www.ebi.ac.uk/eva/	n/a
ExAC	Aggregated exome data from numerous sequencing projects	http://exac.broadinstitute.org/	135
gnomAD	Aggregated genome data from numerous sequencing projects	http://gnomad.broadinstitute.org/	135
HGVD	Japanese genetic variation	http://www.genome.med.kyoto-u.ac.jp/SnpDB/	136
NHGRI-EBI Catalog	Manually-curated database of published genome-wide association studies	http://www.ebi.ac.uk/gwas/home	137
TCGA	Cancer-associated variants	http://cancergenome.nih.gov/	138
Category: Structural variation			
dbVAR	Structural variation	http://www.ncbi.nlm.nih.gov/dbvar/	139
DGVa	Structural variation	http://www.ebi.ac.uk/dgva	139
Category: Databases containing functional predictions			
ClinVar	Clinical significance of variation	http://www.ncbi.nlm.nih.gov/clinvar/	140
dbGaP	Database of genotypes and phenotypes	http://www.ncbi.nlm.nih.gov/gap/	141
dbNSFP	Functional predictions and annotations of non-synonymous SNPs	https://sites.google.com/site/jpopgen/dbNSFP	142–144
LS-SNP/PDB	Non-synonymous SNPs likely to affect biological function	http://ls-snp.icm.jhu.edu/ls-snp-pdb/	145
PinSnps	Protein-protein interaction networks	http://fraternalilab.kcl.ac.uk/PinSnps/	146
SNPeffect	Characterization and annotation of SNPs	http://snpeffect.switchlab.org/	147
SNPs3D	Functional effects of non-synonymous SNPs	http://www.snps3d.org/	148
Category: Gene and protein databases			
Ensembl	Comprehensive biological database including variation	http://www.ensembl.org/	149
HGMD	Disease related gene lesions	http://www.hgmd.cf.ac.uk/	150
HUMA	Comprehensive biological database including variation	https://huma.rubi.ru.ac.za	61

OMIM	Human genes and genetic disorders	http://www.omim.org/	151
Uniprot	Protein database including non-synonymous SNPs	http://www.uniprot.org/	114
VnD	Variation and drugs	http://vnd.kobic.re.kr/	152

Table 4.1. Variation databases

identified in these projects (Table 4.1). The most well-known of these databases is probably dbSNP [133], a database created and managed by the National Center for Biotechnology Information (NCBI) as a central repository for all known short variation. The dbSNP database incorporates data from projects such as 1000 Genomes and HapMap as well many others.

The NCBI hosts a range of additional variation databases, many of which mirror databases hosted by the European Bioinformatics Institute (EBI). NCBI databases include dbVAR [139], dbGaP [141], and ClinVar [140], while EBI databases include the European Variation Archive (EVA), Database of Genomic Variants archive (DGVa) [139] and the European Genome-phenome Archive (EGA) [134]. Variation data is regularly exchanged between EVA, DGVa, dbVAR and dbSNP.

Like dbSNP, ClinVar is arguably the most widely used database when it comes to associating clinical significance with variation. Another widely used database in this category is HGMD [150], although both of these databases suffer from poor reliability scores, which limit their usefulness to an extent.

EVA is a public archive of variation data. Groups are able deposit variation uncovered in sequencing projects and make that data publicly available. EGA is similar, however, data deposited to EGA is stored privately and users must obtain permission to access these datasets from the relevant Data Access Committee. Private and secure storage of genetic data has become an important part of variation databases, not only because it is critical for patient privacy, but also because researchers want to be able to be able to analyze their own data first, before releasing it to the public.

The ExAC [135] and gnomAD are two, relatively new, databases that focus on aggregating exome and genome data, respectively, from a growing number of sequencing projects. These tools have fast become the most widely used datasets for obtaining allele frequencies.

The remainder of the databases depicted in Table 4.1 fill a wide range of categories, from manually-curated databases such as the NHGRI-EBI GWAS catalog [137] (a collaboration between the EBI and the National Human Genome Research Institute (NHGRI)) to disease specific databases such as COSMIC [132] and The Cancer Genome Atlas (TCGA) [138], which focus on variation related to cancer.

While these databases focus on variation, databases such as Ensembl [149] provide a comprehensive set of data including genes, transcripts, proteins, and exons, to which it links phenotypes and variation. Ensembl is an aggregator of data, which means that it collects data from various other sources including many of the databases discussed above. All this data is stored within a single, relational database, allowing users to perform powerful queries that link different kinds of data.

Databases that map variation to protein structures are also included in Table 4.1 *e.g.* PinSnps [146] LS-SNP/PDB [145]. These databases let users query the database for a protein or variant and then visualize the variant or variants in the protein structure.

4.3.2. Shortcomings of existing databases

The databases discussed above are mostly focused on the analysis of variation at the sequence level. The focus of our work is the analysis of variation at the protein level and, specifically, at the protein structure level. As such, although these databases and web servers provide a good source of raw data, they do not give the insights into the structural impacts of variants and are, thus, insufficient for our purposes.

The handful of databases that do include structural information, such as PinSnps and LS-SNP/PDB, are few and far between and are often limited in functionality and usefulness. For example, although LS-SNP/PDB allows users to search by protein, it only allows a single variant to be visualized in the protein structure at a time. This makes it difficult to see where variants are relative to each other. Additionally, the visualizations are static images and are, thus, not interactive.

PinSnps, on the other hand, offers interactive visualization of SNPs in the protein structure using JSmol [153]. As JSmol is implemented in pure JavaScript without making use of WebGL, rendering and interaction can be quite slow. PinSnps also only focuses on protein SNPs and neglects gene data, as well as disease data, to a degree. PinSnps does, however, include predictions from tools such as PolyPhen-2 [116] and PROVEAN [115], which provide an indication of the functional affects of the SNPs.

When designing HUMA, we wanted a tool that would provide as much biological data as possible, including genes, diseases, variation, and protein data, and we wanted to provide users with tools to analyze this data via the HUMA web server. Although a wide variety of data was to be included, the focus of HUMA would be on proteins and protein structure data. This focus was chosen as we believe it to be one of the areas of bioinformatics that is still most lacking as far as tooling is concerned, especially web-based tooling. As such, providing web-based tools in this area also satisfies the broader goal of this thesis, which is to enable research at low-resource institutions.

Structural bioinformatics is a resource intensive field. As such, the lack of web-based tools in this field means that structural bioinformatics is often out of the reach of poorer institutions.

HUMA would, therefore, provide a similar data source to Ensembl, which provides a large and diverse dataset, but with interactive structure visualization, similar to that provided by PinSnps. We also wanted to provide built-in tools to analyze variation. As such, HUMA would be an amalgamation of the best parts of the above databases with added tools.

4.4. Filtering SNV datasets for use in structural studies

Techniques such as GWAS and CGAS are used to analyze variation at the DNA level. Unfortunately, these techniques only provide an association between a variant and a phenotype. They do not give the mechanism or explain how and why the variant is associated with the phenotype.

On the other hand, structural bioinformatics techniques provide a means to explain how variation affects protein function. Where GWAS and CGAS analyze variation at the DNA sequence level, structural techniques provide a means for the downstream analysis of variation *i.e.* the analysis of variation at the protein sequence and structure level. Common structural techniques (discussed further in section 4.5) include homology modeling, molecular docking, molecular dynamics, and Residue Interaction Network (RIN) analysis. For non-synonymous SNVs, structural bioinformatics techniques let researchers form hypotheses on what effects SNPs have on protein structure, stability, and inter- and intra-protein interactions *i.e.* these techniques can be used to determine the mechanisms behind SNV-phenotype associations. Unfortunately, structural bioinformatics techniques are computationally expensive. As such, variant datasets are significantly filtered before performing the structural analysis. This can be done using tools that attempt to predict the effect of SNVs on protein function and stability, often using machine learning techniques. These tools can be used to whittle variation datasets down to manageable sizes.

4.4.1. Predicting damaging missense variants

One of the main challenges of computational SNP analysis at the sequence level is determining whether SNPs are likely to be damaging. As discussed previously, GWAS and CGAS are statistical techniques used at the DNA level to associate SNPs with diseases or phenotypes. At the protein level, several tools have been developed that predict whether diseases will be damaging. These tools have been reviewed in detail in previous papers including our own [131,154] and are summarized in Table 4.2. From the description column in this table, it can be seen that these tools usually fit into two categories *i.e.* tools that make predictions based on the sequence alone, and tools that take both the sequence and structure of the protein into account when making predictions.

Tool	Description	Link	Reference
Auto-Mute 2.0	Sequence- and structure-based	http://binf2.gmu.edu/automute/	155
CADD	Consensus classifier	http://cadd.gs.washington.edu/	156
Ensembl Variant Effect Predictor	Sequence-based + consensus classifier	http://www.ensembl.org/vep	157
FATHMM	Sequence-based	http://fathmm.biocompute.org.uk/	119
MAPP	Sequence-based	http://mendel.stanford.edu/SidowLab/downloads/MAPP/index.html	110
Meta-SNP	Consensus classifier	http://snps.biofold.org/meta-snp/	104
MetaLR	Consensus classifier	https://sites.google.com/site/jpopgen/dbNSFP	144
MetaSVM	Consensus classifier	https://sites.google.com/site/jpopgen/dbNSFP	144
MuD	Sequence- and structure-based	http://mud.tau.ac.il/	158
MutationAssessor	Sequence-based	http://mutationassessor.org/r3/	159

MutationTaster	Sequence-based	http://www.mutationtaster.org/	160
MutPred	Sequence-based	http://mutpred.mutdb.org/	161
PANTHER-PSEP	Sequence-based	http://www.pantherdb.org/tools/csnpScoreForm.jsp	117
Parepro	Sequence-based	http://www.mobioinfor.cn/parepro/	118
PolyPhen-2	Sequence- and structure-based	http://genetics.bwh.harvard.edu/pph2/	116
PredictSNP	Consensus classifier	http://loschmidt.chemi.muni.cz/predictsnp/	105
PROVEAN	Sequence- and structure-based	http://provean.jcvi.org/index.php	115
REVEL	Consensus classifier	https://sites.google.com/site/revelgenomics/	162
SIFT	Sequence-based	http://provean.jcvi.org/index.php	107
SNAP	Sequence-based	http://www.bio-sof.com/snap	163
SNPs&GO	Sequence- and structure-based	http://snps.biofold.org/snps-and-go/snps-and-go.html	164
VAPOR	Consensus classifier	https://huma.rubi.ru.ac.za/#vapor	61

Table 4.2. Tools for predicting the functional effects of non-synonymous SNVs

Benchmarking shows that the prediction tools mentioned in Table 4.2 seldom achieve an accuracy of much higher than 75%. As such, they have a significant margin for error. Because the different tools use different algorithms and scoring methods to make their predictions, they also often produce differing predictions. It is, therefore, good practice to gain a consensus from multiple tools when analyzing variants. Consensus classifiers such has Meta-SNP [104] and PredictSNP [105], which automatically run multiple prediction tools and combine the results of those tools to make their own predictions, have been shown to produce more accurate predictions.

Like the above consensus classifiers, VAPOR, our JMS workflow, which was discussed in section 3.4, combines the results of multiple prediction tools. Unlike Meta-SNP and PredictSNP, VAPOR does not attempt to make a prediction using these combined results and, instead, simply merges the results into a single table. In future, functionality will be added to generate a consensus score from these predictions. That being said, VAPOR remains as a useful tool for quickly obtaining

Tool	Description	Link	Reference
Auto-Mute 2.0	Sequence- and structure-based	http://binf2.gmu.edu/automute/	155
CUPSAT	Structure-based	http://cupsat.tu-bs.de/	166
Eris	Structure-based	http://troll.med.unc.edu/eris/login.php	167
I-Mutant2.0	Sequence- and structure-based	http://folding.biofold.org/i-mutant/i-mutant2.0.html	120
MuPro	Sequence- and structure-based	http://mupro.proteomics.ics.uci.edu/	121
NeEMO	Residue interaction networks	http://protein.bio.unipd.it/neemo/help.html	168
PoPMuSiC 2.1	Structure-based	https://soft.dezyme.com/query/create/pop	169

Table 4.3. Tools for predicting changes in stability due to non-synonymous SNPs

results from multiple SNP analysis methods and has been integrated into the HUMA web server. VAPOR is discussed further in chapter 6.

4.4.2. Predicting changes in protein stability due to variants

Predicting the impact that SNVs will have on protein stability, as opposed to function, can also be used to filter out potentially insignificant SNPs. Non-synonymous SNPs can affect the internal energy of a protein. This is usually calculated by determining the difference in Gibbs free energy between a wild type and variant protein [165]. As with functional predictions, predicting changes in stability is not an exact science and existing tools have significant error rates. As such, it is best practice to get a consensus from multiple tools.

Both increases and decreases in protein stability can result in a damaging effect on protein function. As such, when looking at the results of stability prediction tools, we are not as interested in the sign of the change in stability (*i.e.* positive or negative), but rather the magnitude of the change. A large change in either direction can result in damaging effects.

Several tools have been developed to predict the change in protein stability due to a variant (Table 4.3). These tools have been discussed in further detail in our review [131]. Unlinke Meta-SNP and PredictSNP, VAPOR also includes two stability prediction tools, I-Mutant2.0 [120] and MuPro [121]. This adds an extra dimension to the results generated by VAPOR.

4.5. Importance of structural bioinformatics in SNV analysis

Variants have long been associated with drug resistance in diseases such as influenza, tuberculosis, HIV and cancer [170–174]. Variants have also been linked to drug sensitivity - where patients respond better than expected [175]. The idea that variable genetics results in variable response to treatments opens the door to personalized medicines. Knowing which variants result in altered drug responses, combined with low-cost sequencing that could be used to determine what variants a patient has, would allow for treatments to be tailored to individual patients [176,177]. An important part of designing drugs to target these differences is understanding the structural changes caused by variants [152].

Computer-Aided Drug Design (CADD) refers to the use of computational techniques that aid in the drug discovery and design process [178]. Structural bioinformatics has been used to aid in every stage of this process [178–182], and can often replace expensive and time-consuming experimental techniques [183–185]. Examples of this include protein structure prediction techniques, such as homology modeling, which provide an alternative to X-ray crystallography and NMR techniques, while virtual screening and molecular dynamics simulations can complement or replace High-Throughput Screening (HTS).

4.5.1. Protein structure prediction

As a result of advances in NGS technologies, there is an abundance of protein sequence data available. Unfortunately, structural techniques have not been able to keep up, as experimentally solving the structures of these proteins remains a slow an expensive process. This has resulted in a large gap forming between known protein sequences and solved protein structures. To illustrate this, as of March 2017, the Protein Data Bank [186] contained structures for 40 544 distinct proteins sequence, while the Uniprot contained over 80 million distinct protein sequences.

To counter the growing sequence-structure gap, protein structure prediction software allows researchers to model the structure of proteins. There are two main ways this can be done. Comparative modeling, also known as homology modeling, was previously discussed in section 1.1.2. Homology modeling can produce decent quality models for roughly two-thirds of unsolved proteins [187–189].

Where homology modeling attempts to predict the structure of a protein based on the structure of a similar protein *i.e.* a template, *ab initio* modeling attempts to construct a model of a protein based exclusively on its amino acid sequence. As these methods are computationally expensive, they can currently only be used for small molecules [190]. In addition, their accuracy cannot yet compete with homology-based approaches [189].

Having access to the three-dimensional structure of a protein gives researchers insight into the molecular function of the protein, which, in turn, opens up the door to drug design and discovery [182,191]. In the context of SNP analysis, protein structure prediction can also be used to model SNPs into a structure. The mutated structure can then be compared to the wild type in various ways (*e.g.*

using the techniques that will be discussed in the following sections) to determine the effects that the SNPs might have on protein function and stability.

Homology modeling is commonly used in drug discovery to understand protein function and mechanisms [192], analyze of the effects of variants in the binding sites of receptor proteins [193], identify druggable pockets [194], and in numerous virtual screening studies [195–198].

4.5.2. Molecular docking and virtual screening

Molecular docking is a technique used to predict the conformation of a protein-ligand complex and was also briefly discussed in a section 1.1.2. In the context of structure-based drug design, it is used to study biomolecular interactions [199]. In a process called virtual screening, libraries containing thousands of compounds are repeatedly docked against a receptor protein with the aim of finding the compounds that bind to the receptor with the greatest affinity *i.e.* potential drug candidates [200–202]. These compounds are selected for further study. As such, virtual screening can greatly reduce the number of compounds that would have to be tested experimentally in the lab.

In the context of SNP analysis, molecular docking can be used in conjunction with protein structure prediction to predict the effect that variants will have on drug response [193]. This can be done by modeling variants into the binding site of a wild type protein structure and performing docking studies on both the wild type and variant structures. Significant changes in binding affinities between the wild type results and variant results could indicate that the variants have an impact on drug response.

Virtual screening has become a routine part of the drug discovery as it offers a cheaper and faster alternative to HTS [203]. High quality compound libraries are an important part of virtual screening and, as such, several of these libraries, including ZINC [204], ChemSpider [205], the Traditional Chinese Medicine Database@Taiwan [206] and SANCDB [82], have been made available online.

4.5.3. Molecular dynamics (MD) simulations

Where protein structure prediction and molecular docking provide a snapshot in time of a protein structure and protein-ligand complex, respectively, MD simulates the movements of the atoms in these structures or complexes over a period time. In other words, while the above two techniques produce a static picture, MD produces a "movie". This can be used to determine whether a protein structure remains stable after the introduction of one or more SNPs or whether a compound will stay docked over a period of time [207].

MD simulations are a computationally expensive operation and are one of the main reasons why variant datasets should be filtered significantly before starting structural studies. They are often used in combination with homology modeling and virtual screening to verify that the predicted structure or complex remains stable over a period of time [207,208]. In the context of SNP analysis, MD can be used to determine whether introducing a SNP will destabilize a protein or perhaps cause the protein to move or fold in a different way [209].

4.5.4. Intra- and inter-protein interactions

Interactions between residues within the same protein are known as intra-protein interactions. Because proteins fold, residues that may be far apart in the protein sequence could be located next to each other in three-dimensional space. Intra-protein interactions between these residues play an important role in helping the protein adopt the correct structural conformation [210]. Consequently, it stands to reason that anything that disrupts these interactions, such as a variant, could have a destabilizing effect on the protein and result in a loss of function.

Similarly, inter-protein interactions, also known as protein-protein interactions (PPIs) are interactions between residues from different proteins (*e.g.* in a protein complex). Proteins interact to perform functions all around the body. A variant that occurs at an interacting position in a protein complex could destabilize the complex or negatively affect the interaction, resulting in a loss of function.

Knowing which residues are import for protein inter- and intra- interactions is useful in drug design as it gives researchers an idea of areas of a protein that can be targeted to disrupt the function of the protein. In the context of SNP analysis, a variant that occurs at an interacting position is more likely to have a damaging effect than if it were to occur at a non-interacting position. Important interactions can be uncovered by analyzing the types of bonds (*e.g.* hydrogen bonds, di-sulphide bonds, *etc.*) that occur between residues.

RINs can also be used to determine the effects that variants have on intra- and inter-protein interactions. Analyzing RINs in the context of SNP analysis is the topic of section 3 of this thesis, however, and will not be discussed further here.

4.6. Protocol for analyzing SNVs using structural bioinformatics

In this chapter, we have discussed the role that structural bioinformatics plays in the drug discovery process, specifically with regards to analyzing the functional impacts of SNVs. We have also reviewed existing variation databases and tools that can be used to predict the effects of variation on protein function and stability. To bring this all together, we now suggest a protocol for analyzing variation using structural bioinformatics (Fig. 4.1). This protocol was published as part of our review paper [131].

The first step in any type of analysis is retrieving the necessary data. In our case, we require the protein sequence and structure of our protein of interest and the non-synonymous SNVs that occur in that protein. Variants can be obtained from any of the numerous databases provided in Table 4.1.

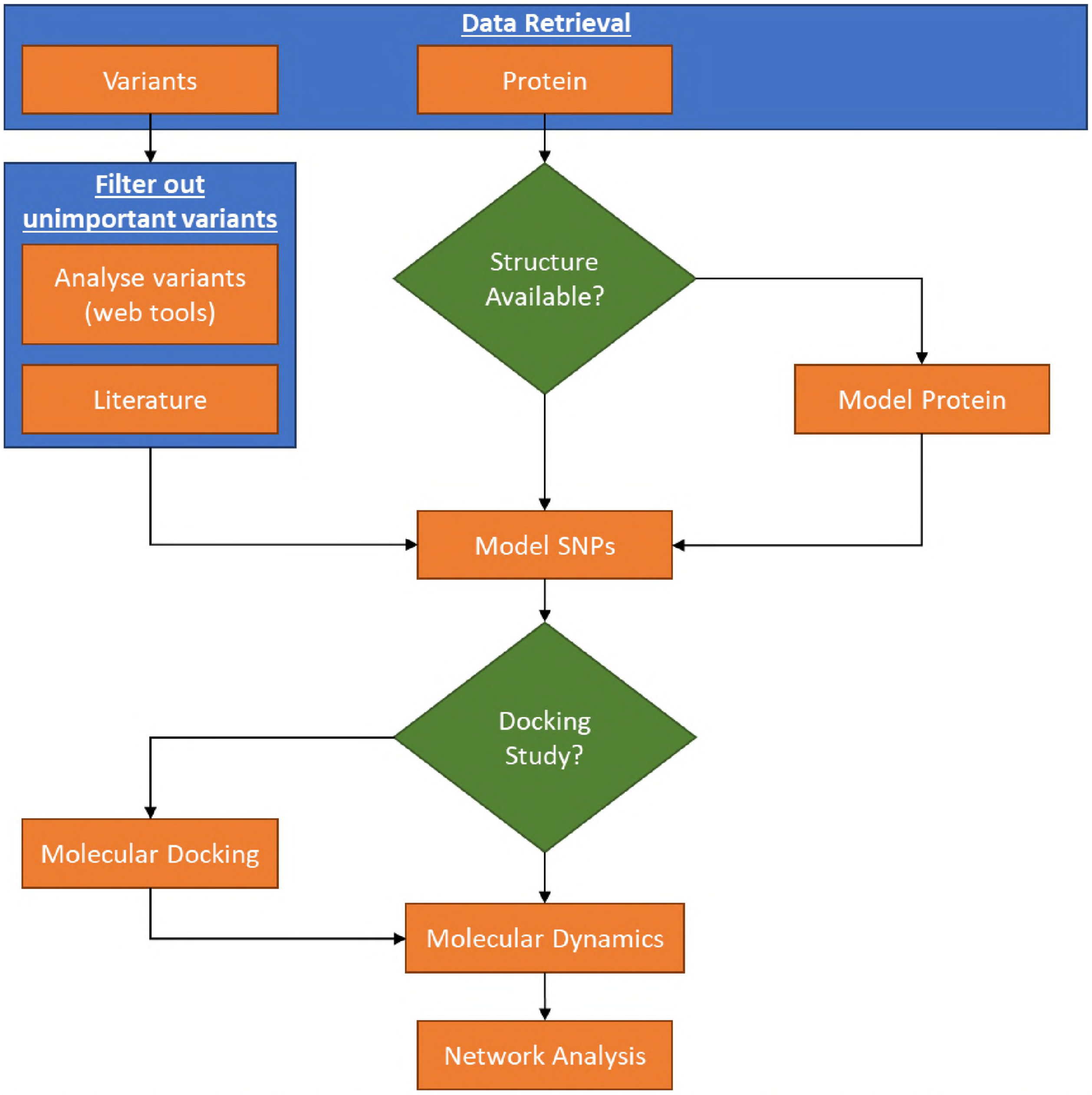


Fig. 4.1. *Protocol for analyzing SNVs – this process can be broken up into three parts: 1) Data retrieval and filtering, where the necessary sequence, structure, and variants are obtained; 2) Data creation, where the mutated structures are obtained and ligands are introduced (optional); and 3) Data analysis, where the mutated structures/complexes are analyzed using MD and network analysis.*

Protein sequences can be obtained from databases such as Uniprot or Ensembl, while protein structures, if they are available, can be obtained from the Protein Data Bank (PDB). Of these databases, Ensembl is arguably the most useful, as it provides both the variation data and sequence data and links to available structures.

If there is no structure available in the PDB, or if the structure has missing residues, it must be modeled. This can be done using online tools such as HHPred [89], SWISS-MODEL [14], I-TASSER [211], or our own PRIMO [83].

At this point, you should have the protein sequence and structure and a variant dataset from one of the databases listed in Table 4.1. The computationally expensive nature of structural bioinformatics means that the variant dataset cannot be too large. Variants that are likely to be uninteresting can be filtered out using the prediction tools mentioned in Tables 4.2 and 4.3. If the consensus gained from several of these tools points toward the variant being neutral, it can be removed from the dataset. The quickest way to do this is by using tools such as PredictSNP [105], Meta-SNP [104], or our own workflow, VAPOR, which execute multiple tools as part of a single submission.

Prediction tools are not infallible and there are times where SNVs should be retained in the dataset, despite the consensus from the tools being that the SNV is non-damaging/neutral. As such, to complement the above filtering, literature linking variants to disease can also be used as part of the filtering. In other words, if a tool predicts that a variant will have a neutral effect, but there is a known link to a disease in literature, do not remove the variant from the dataset. Literature should always trump predictions.

The next step is to look for important interacting residues in the structure of the protein. This can be done using the web-based tools such as PIC [212], COCOMAPS [213], InterProSurf [214], PDBParam [215], and PDBSum [216]. As discussed earlier, residues that take part in inter- and intra-protein interactions can be important for protein function and stability. SNVs that occur around these residues should also be retained, if possible.

With the necessary data now obtained and filtered, the variants can be modeled into the structure of the protein. Each variant should be modeled into the structure of the protein individually *i.e.* if the dataset contains 10 SNVs, 10 variant models should be produced, each containing one of the SNVs. Additional models can then be produced containing combinations of SNVs. This can be done at the researcher's discretion. For example, we previously created models containing combinations of SNVs where those SNVs occurred in interacting positions [217].

At this point, the protocol allows for an optional step depending on the purpose of the study. If the purpose is to determine whether variants affect the binding affinity of a drug or compound, a molecular docking study should be carried out, where the drugs or compounds are targeted at the mutated structures.

Regardless of whether a docking study was conducted, the next step is to run MD simulations on the mutated proteins (or protein-ligand complexes, if a docking study was carried out). MD simulations improve the reliability of docking results by showing that the ligand remains docked over a period of time, rather than detaching and floating away. MD can also be used to analyze the stability of mutated proteins. A mutated protein may have been destabilized by the variant, which impairs its function. The MD simulation essentially allows us to determine whether the variant has affected the way the protein or complex moves.

Once MD simulations are complete, they must be analyzed. The output of an MD simulation is a trajectory file, which describes the trajectories of the atoms in the molecule over the course of the simulation. This file is often analyzed using methods such as RMSD, which describe global movement, or root mean square fluctuation (RMSF), which describes the local movement of residues in the molecule. Another interesting method is to analyze the RIN of the molecule to determine how it is affected over the course of the simulation [218]. Section 3 of this thesis describes a tool suite for analyzing trajectories using network analysis.

Analysis of the MD trajectory if the final step in the protocol. At this point, the RMSD, RMSF, and network analysis of the trajectories should have provided enough data to determine, which variants are interesting to study further using experimental techniques. By using this protocol, a dataset of potentially hundreds of SNVs can be filtered down to a few dozen and then analyzed using structural bioinformatics. The results of this analysis can be used to filter variants even further for experimental studies, meaning that the dataset taken to the lab will likely to contain the most significant SNVs.

4.7. Research motivation

With the ever-increasing rate at which variation data is being generated, there is more of a need than ever for databases and tools to store and analyze this data. In Africa, this need is being accelerated with the advent of the H3Africa initiative. To this end, H3ABioNet, a bioinformatics subnetwork of H3Africa, was established with a goal of building tools and infrastructure to aid in the analysis of genetic variation uncovered by H3Africa sequencing projects.

The lack of existing tools to analyze variation at the structural level also poses a problem. Most databases and tools are focused on variation data at the sequence level. As discussed above, understanding how variation impacts protein structure can give insights into how and why this variation results in, for example, loss of function. Structural studies also provide insight into the mechanisms behind drug resistance and sensitivity – an area which has been under-studied in African populations.

Additionally, most variation databases are focused solely on variation and do not provide much data on the proteins and genes that the variants occur in and the diseases that the variants are associated with. For example, dbSNP provides the position that a variant occurs at in a protein, but you need to go to a different web server all together to look up any meaningful information about that protein.

Considering the previously described protocol, as well as the above concerns, it was concluded that a suite of tools that could facilitate the structural analysis of genetic variation would aid the objectives of the H3Africa initiative.

4.8. Research aims and objectives

The focus of this body of work is to develop a database and tools to store and analyze variation from H3Africa projects. The project will integrate data from public sources, as well as functionality from existing tools, into a unique and comprehensive solution for storing and analyzing variation. The project will also focus on the analysis of variation at the structural level, as this area has been neglected in existing tools.

The specific objectives for this work are as follows:

1. Develop a comprehensive biological database that links variation to genes, diseases, proteins, and protein structures.
2. Develop tools to analyze genetic variation using the previously described protocol.

3. Develop a web server to make the data and tools available online.
4. Develop collaboration tools to allow H3Africa groups to share data and discuss results.

5. HUMA Databases

5.1. Introduction

The Human Mutation Analysis (HUMA) database was born out of a foreseen need to store and analyze unique variation data coming from H3Africa sequencing projects. As development on HUMA began before data from H3Africa projects became available, it was initially designed using publicly available datasets, with the aim of adding H3Africa data at a later stage.

One of the stated goals of the H3Africa initiative is that the analysis of data being produced be handled by African scientists and institutions. Because of this, H3Africa data cannot be made public immediately, as the groups producing it need time to analyze it first. HUMA was designed to cater for this by splitting publicly available data (*i.e.* data obtained from public data sources) and privately uploaded data (data uploaded by H3Africa researchers or other users) into two separate databases. This allows researchers to upload and analyze variation data without fear of it being accessed by unauthorized users.

One of the aims of the HUMA project was also to facilitate collaboration between different research groups in H3Africa. The database was, thus, designed to allow datasets to be shared with other users. As part of the HUMA website, which is discussed in chapter 6, users can also run tools to analyze data and share the results with other users. Discussion forums were also provided to allow users to discuss datasets and analysis results. As such, the database had to be designed to cater for this functionality.

The remainder of this chapter is dedicated to describing the design and development of the HUMA database. A manuscript describing HUMA has been submitted for publication and is currently under review.

5.2. Data sources

In its attempt to be a comprehensive biological database, HUMA aggregates data from several public sources including dbSNP [133], ClinVar [140], PDB [186], Ensembl [149], Uniprot [114], and HUGO Gene Nomenclature Committee (HGNC) [219] into a single, connected database. This provides a great deal of power to the user as it allows them to search the database based on various names and identifiers. In addition, the result pages link back to the original source of the data.

The focus of the HUMA database is to link variation to genes, proteins, protein structures, and diseases. As such, data is broken up into four, related categories – proteins, genes, diseases, and variants. Searching based on one category will also return related data from the other categories. In other word, searching for a protein will locate details about the protein, the gene or genes that code the protein, diseases linked to the protein, and of course, variants in the protein sequence.

5.2.1. Genes

Gene data was retrieved from the Ensembl and HGNC databases. Data from Ensembl was obtained using the Biomart [52] data mining tool available on the Ensembl website. Biomart allows users to run highly customizable queries on a range of different publicly available datasets. It was used to run several queries against the Ensembl database to obtain the required gene data.

Firstly, a Fasta file containing all available gene sequences was fetched. Biomart was used to add the following fields to the headers of the Fasta sequences:

- Ensembl gene ID
- Description
- Chromosome
- Start position on the chromosome

- End position on the chromosome
- Strand
- Associated gene source
- Associated gene name

A second Biomart query was used to obtain a tab-separated values (TSV) file containing the following external IDs for the genes:

- Entrez ID (NCBI)
- Online Mendelian Inheritance in Man (OMIM) ID
- HGNC Symbol

Lastly, a TSV file was downloaded from the HGNC database containing the approved symbols and names, previous symbols and names, synonyms, and PubMed IDs for all genes. HGNC data was linked to Ensembl data based on the HGNC symbols obtained in the second Biomart query.

5.2.2. Proteins

Protein data was retrieved from Uniprot, Ensembl and the PDB. This data included protein sequences and structures, exons and coding sequences (CDSs), and links to literature.

Firstly, the SwissProt (manually annotated and reviewed) and TrEMBL (automatically annotated and not review) `.dat` files were downloaded from the Uniprot FTP site. Theses files contained details such as the identifiers, accession numbers, names and descriptions, and features, such as secondary structure and binding sites, for all known human proteins. Additionally, where available, the SwissProt file contained links to literature concerning the proteins.

Secondly, a Fasta file containing the sequences for the above proteins was downloaded from Uniprot. Although the sequences are also available in the `.dat` files, it was quicker and safer to extract them from a Fasta file.

All human protein structures were than downloaded from the PDB in PDB format. Chains in these PDB files that represented proteins were linked to Uniprot proteins based on the Uniprot accession numbers in the PDB files.

Protein domains and families were retrieved from the Pfam database [220]. This data included the Pfam identifier, name, type (family/domain), description, and residue co-ordinates.

Lastly, protein sequences were also obtained from Ensembl, as well as "supporting sequences" *i.e.* exons and CDSs. The exon data was obtained in the form of a Fasta file. Biomart was used to add the following fields to the headers of the Fasta sequences:

- Ensembl Transcript ID
- Chromosome
- Ensembl Exon ID
- Start position on the chromosome
- End position on the chromosome

Protein sequences and CDSs were also obtained in the form of Fasta files, but were downloaded directly from the Ensembl FTP site.

5.2.3. Variation

Variation data was downloaded from dbSNP. As one of the most well-known variation databases, dbSNP host a comprehensive set of all known human variation. Data was download the dbSNP

FTP site in Variant Calling Format (VCF). Data extracted from this file included the chromosome co-ordinates, dbSNP ID, and allele change.

5.2.4. Diseases

Disease data was obtained from ClinVar, Uniprot, and Ensembl. Once again, a Biomart query was used to obtain a TSV file from Ensembl, this time containing the phenotypes linked to the genes. The following fields were included in this file:

- Ensembl gene ID
- Phenotype description
- Phenotype source name
- Phenotype study external reference

The ClinVar database contains variation data with clinical significance. The `variant_summary.txt` file was downloaded from the ClinVar FTP site and diseases were extracted. This data could also be used to link our variation data to diseases.

Similarly, the humsavar.txt file from Uniprot contains non-synonymous SNVs linked to diseases. Data in the file includes the dbSNP IDs of the variants, the OMIM IDs of the diseases, and the Uniprot accessions of the proteins. This allowed us to link diseases with both variation and protein.

5.3. Database design

Two databases were developed for HUMA. The first database was built to house biological data obtained from public sources *i.e.* the data discussed above. Data in this database is accessible to any user and requires no authentication or authorization.

The second database was built to house private data *i.e.* data provided by users. This includes user account details such as names, email addresses, and authentication details such as usernames and password hashes. Additionally, data linked to user accounts is stored in this database. This includes user uploaded variant datasets and results generated by running analysis tools.

As this is the first iteration of the HUMA software, these databases are still fairly simplistic, but a lot of effort has been expended on ensuring that the design is extensible. This will allow the database to be easily extended in future.

These databases were separated for two reasons. Firstly, we decided that it was prudent to store private and public data separately to reduce the chance that an arbitrary coding error would result in exposing private data.

Secondly, separating the databases made it easier to back up and ensure redundancy of the private database. Backing up this database was considered more vital than backing up the public database, as the public database can be recreated from the raw data files. If user data were to be lost, we would not be able to recreate it. Having the much smaller, private database separate meant that we could run regular backups on this database without affecting the performance of the public database by hosting them on different machines.

5.3.1. Public database

The design of the public database is depicted Fig. 5.1. The database can be split up into four sections corresponding to the four categories mention earlier *i.e.* genes, proteins, variation, and diseases. We will now discuss the design of each of these sections in detail.

5.3.1.1. Genes

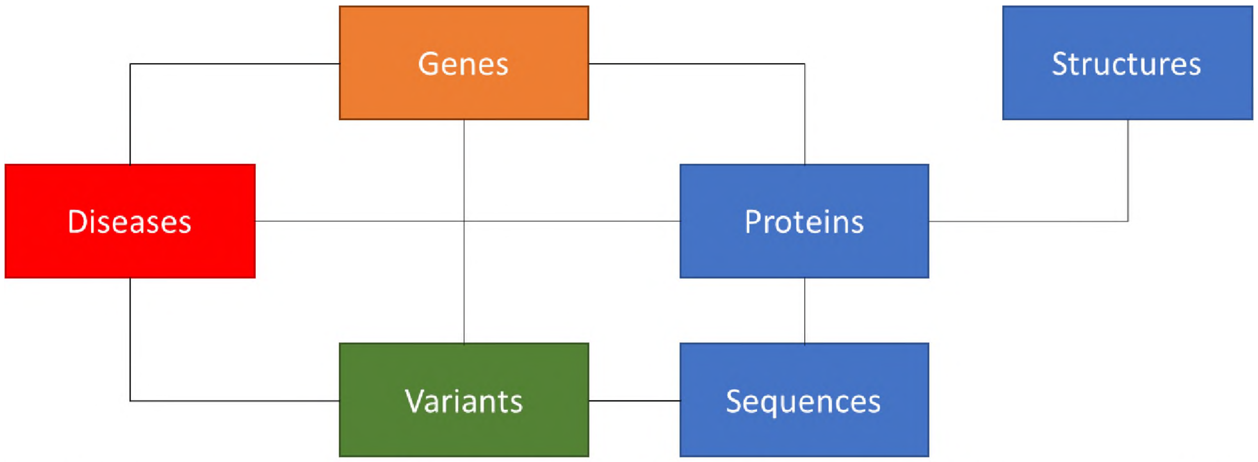


Fig. 5.1. *Public database – a simplified depiction of the public database. The sections of the database representing the four categories are color-coded. Links between the blocks depict how each section links to the other.*

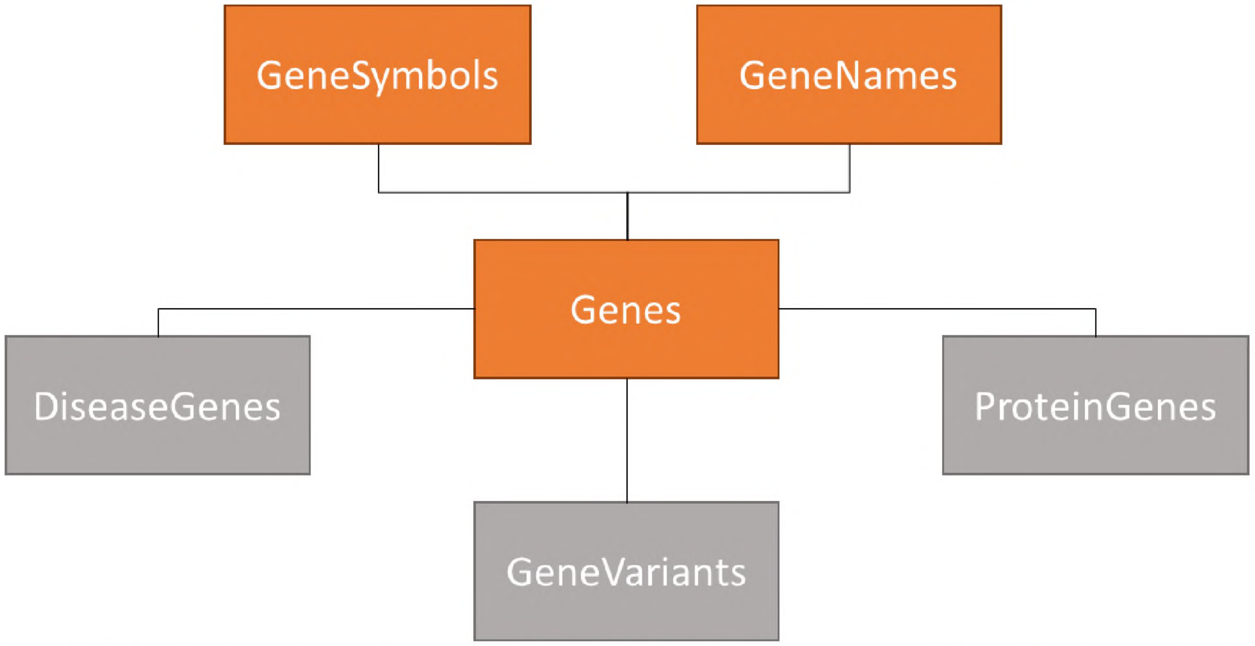


Fig. 5.2. *Gene tables – a simplified representation of the tables containing gene data (orange) and the tables linking genes to other data in the public database (gray).*

The *genes* section of the database is depicted in Fig. 5.2. The main table is the *Genes* table. This table holds most of the details about the gene including the Ensembl gene ID, name, chromosomal co-ordinates, and external IDs. Because gene sequences can be very long, the sequences are saved to files on the filesystem. The *Sequence* column then stores the path to the file rather than the sequence itself.

The remaining two tables in the *genes* section are the *GeneSymbols* and *GeneNames* tables. These tables hold alternative symbols and names used to refer to a given gene. Storing these details in addition to the standard symbol and name makes it much easier to find genes when searching.

The *genes* section is linked to the *proteins*, *variation*, and *diseases* sections via the *ProteinGenes*, *GeneVariants*, and *DiseaseGenes* tables respectively.

5.3.1.2. Proteins

The *proteins* section of the database is depicted in Fig 5.3. As the focus of this project is on the downstream analysis of variation *i.e.* analysis of variation at the protein sequence and structure level, the *proteins* section of the database is, by far, the most detailed and complex.

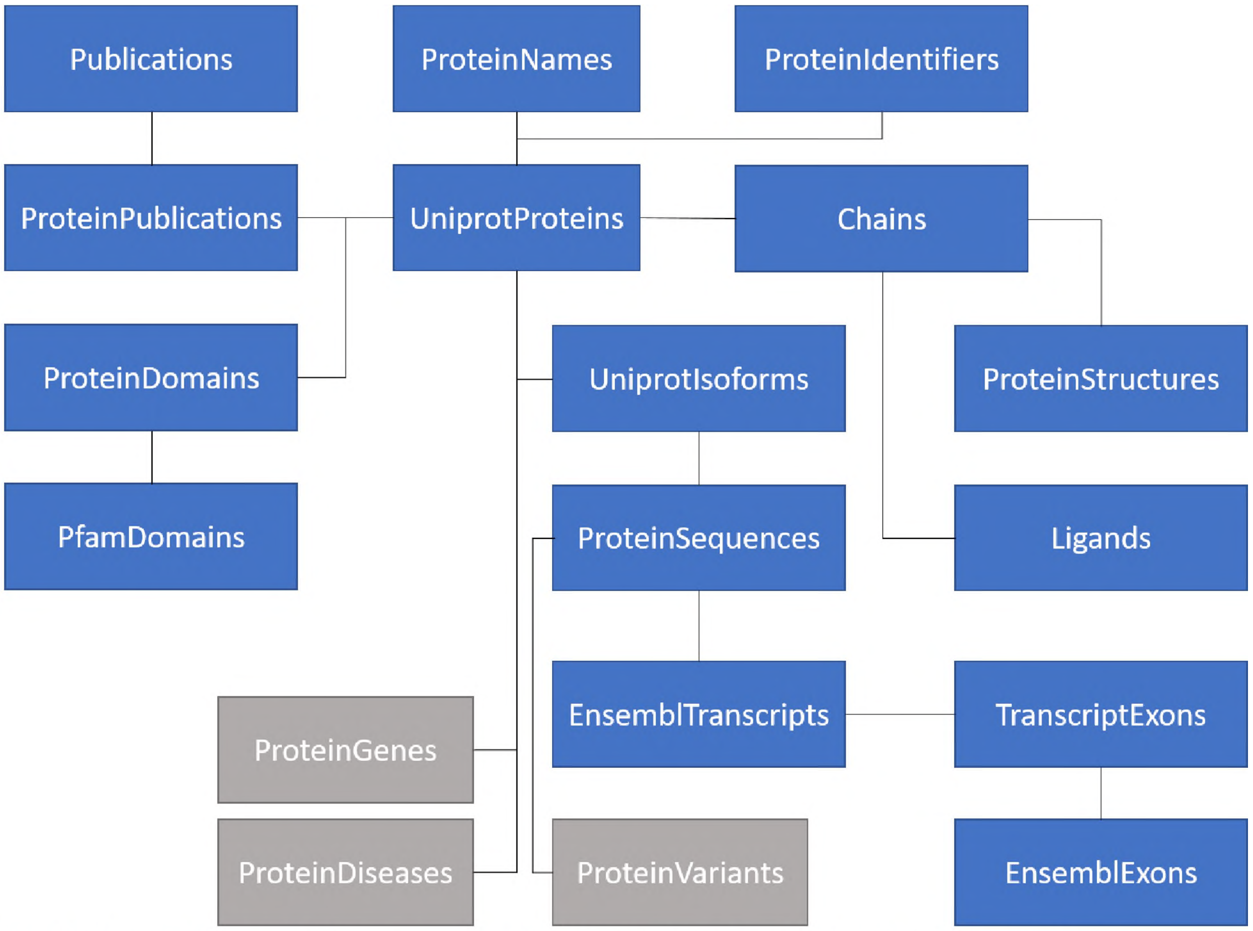


Fig. 5.3. Protein tables - a simplified representation of the tables containing protein data (blue) and the tables linking proteins to other data in the public database (gray).

The central table in the proteins section is the *UniprotProteins* table. It stores the Uniprot accession and ID, names for the protein, the function of the protein, and whether it came from the SwissProt or TrEMBL `.dat` file.

Literature extracted from the SwissProt `.dat` file is stored in the *Publications* table and linked to the *UniprotProteins* table via the *ProteinPublications* table. Similarly, additional protein names and identifiers from the `.dat` files are stored in the *ProteinNames* and *ProteinIdentifiers* tables, respectively. As with genes, the additional protein names and identifiers improve the searchability of the database.

The Uniprot database stores multiple sequences for each protein. These additional sequences are known as isoforms. As such, for each entry in the *UniprotProteins* table, there can be one or more entries in the *UniprotIsoforms* table. The actual sequences themselves are not stored in the *UniprotIsoforms* table, however. Instead, they are stored separately in the *ProteinSequences* table. This is so that sequences obtained from Ensembl can be stored in the same table to reduce redundancy.

As alluded to above, the protein sequences from Ensembl are also stored in the *ProteinSequences* table. To avoid storing duplicates of Uniprot sequences, the primary key in this table is a hash of the sequence. This means that each time an attempt is made to add an Ensembl sequence that has a matching Uniprot sequence (which would already have been added to the table), the attempt will fail, as the primary key field will clash. As such, the end result will be a table containing all the unique sequences from Uniprot and Ensembl.

Exons downloaded from Ensembl are stored in the *EnsemblExons* table. This data also includes the chromosome co-ordinates of the exons. Ensembl CDSs, along with their chromosome co-ordinates, are stored in the *EnsemblTranscripts* table. The CDSs are linked to exons via the *TransciptExons* table and are used in combination with the exons when mapping variants to protein sequences. The CDSs can easily be linked to the Ensembl protein sequence as they come from the same data source.

Unique protein domains and families from Pfam are stored in the *PfamDomains* table. These are linked to the *UniprotProteins* table in a many-to-many relationship via the *ProteinDomains* table.

Protein structure details are stored across the *ProteinStructures*, *Chains*, and *Ligands* tables. The *ProteinStructures* table stores details about the PDB file such as the PDB ID, structure title, and resolution. The PDB files are stored on the filesystem, rather than in the database.

Each of the structures in the database can have one or more chains. Chains that represent proteins have Uniprot accession numbers associated with them, which are used to link the *Chains* table to the *UniprotProteins* table.

Lastly, each chain can have multiple ligands bound to it. These are stored in the *Ligands* table.

The *proteins* section of the database is linked to the *genes*, *variation*, and *diseases* sections of the database via the *ProteinDiseases*, *ProteinVariants*, and *ProteinGenes* tables.

5.3.1.3. Variation

The *variation* section of the database is depicted in Fig 5.4. This section consists of five tables. The *VariantStore* table is a staging table where variants are stored before they are mapped to genes and proteins. This process is discussed in section 5.5.

The *Variants* table is the main table in this section. It stores the base details about all variation in the database once that variation has been mapped to genes and proteins. This includes details like the dbSNP ID, chromosomal co-ordinates, and allele change.

The process of mapping variants to proteins generates new data. The includes the position that the variant occurs in the protein sequence, the amino acid change that results from the variant, the codon, and the position of the variant within the codon. These details are stored in the *ProteinVariants* table. This table also links the *variation* section to the *proteins* section of the database.

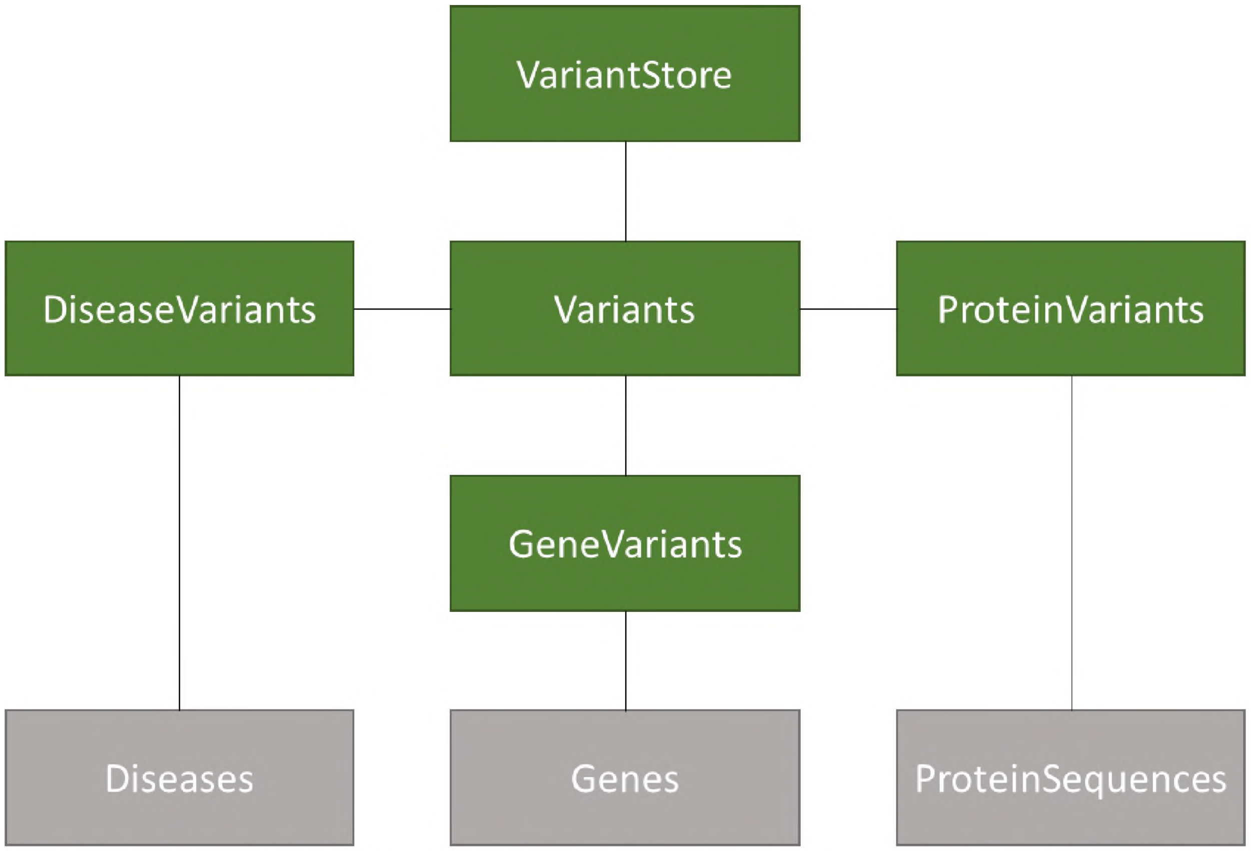


Fig 5.4. *Variant tables - a simplified representation of the tables containing variant data (green) and the tables linking variants to other data in the public database (gray).*

As previously mentioned, the *GeneVariants* table is used to link the *variation* section to the *genes* section of the database. Additionally, when variants are mapped to genes, the position that the variant occurs in the gene is calculated. This position is stored in the *GeneVariants* table.

Lastly, the *DiseaseVariants* table is used to link the *variation* section to the *diseases* section of the database. As previously stated, variants are mapped to diseases based on data from ClinVar, Uniprot, and Ensembl. Where the literature source is available, this is also stored in the *DiseaseVariants* table.

5.3.1.4. Diseases

The *diseases* section of the database is depicted in Fig 5.5. This is the simplest section of the database and consists of a single table, the *Diseases* table, which simply stores the name of the disease and the source that the disease data was retrieved from.

The *diseases* section of the database links to the *genes*, *proteins*, and *variants* sections via the *DiseaseGenes*, *DiseaseVariants*, and *ProteinDiseases* tables respectively.

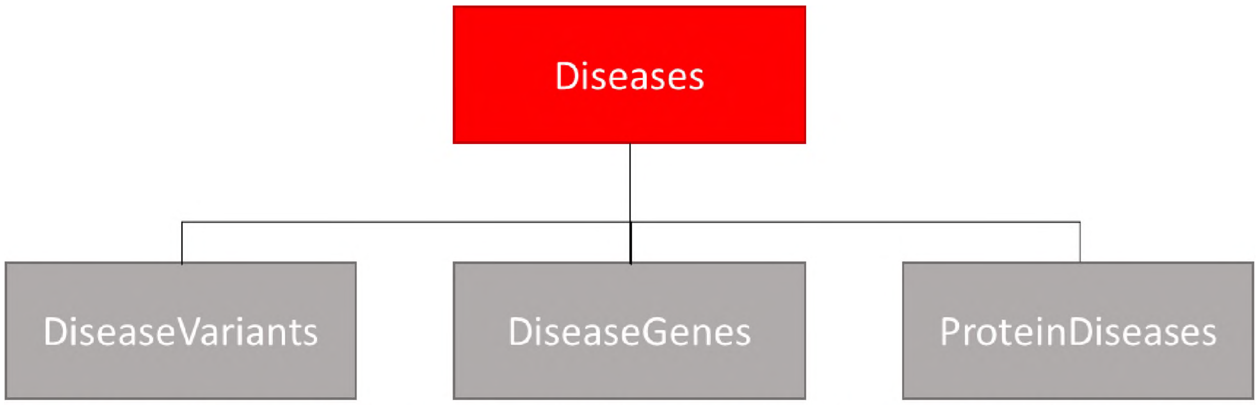


Fig. 5.5. *Disease tables - a simplified representation of the tables containing disease data (red) and the tables linking diseases to other data in the public database (gray).*

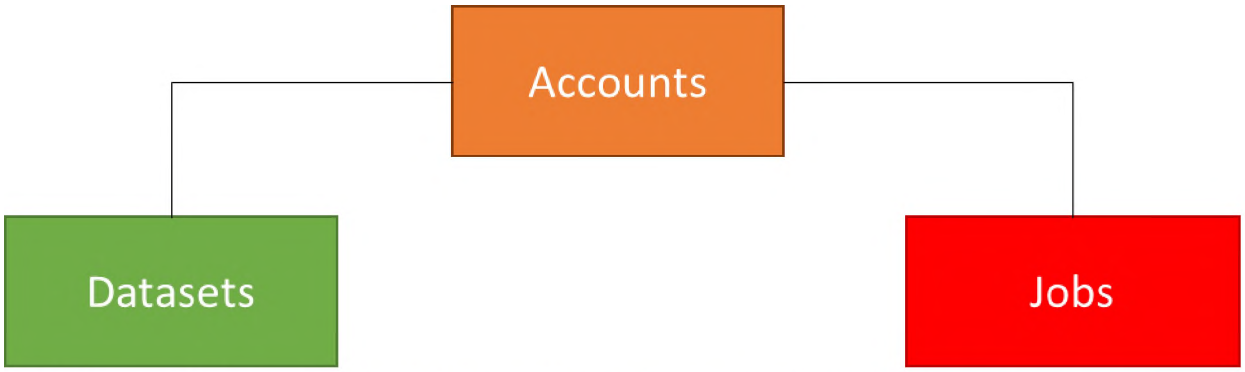


Fig. 5.6. Private database – the private database is made up of three sections; 1) the Accounts section stores details about users and groups; 2) the Datasets section store user uploaded data; and 3) the Jobs section stores the results and meta-data around user-submitted jobs.

5.3.2. Private database

The design of the private database is depicted in Fig. 5.6. The private database can be broken up into three sections. The first of these sections contains user account details. Everything in the private database links back to a user account. The *accounts* section also stores group details and controls the sharing of data and analysis results.

The second section of the private database stores job details. When a user executes a tool via the HUMA web server, a "job" is created in the database. The *jobs* section stores all the details about the job, including input parameters and outputs generated by the tools.

The final section is responsible for storing user uploaded variation datasets. The *datasets* section allows for these variants to be mapped to public data as well, despite that data being kept in a different location.

5.3.2.1. Accounts

The *accounts* section of the database is depicted in Fig. 5.7. This section consists of 12 tables, mostly orientated around group management. When a user is created, and entry is created in both the *Users* table and the *UserProfiles* table. The *Users* table is a built-in Django table that stores

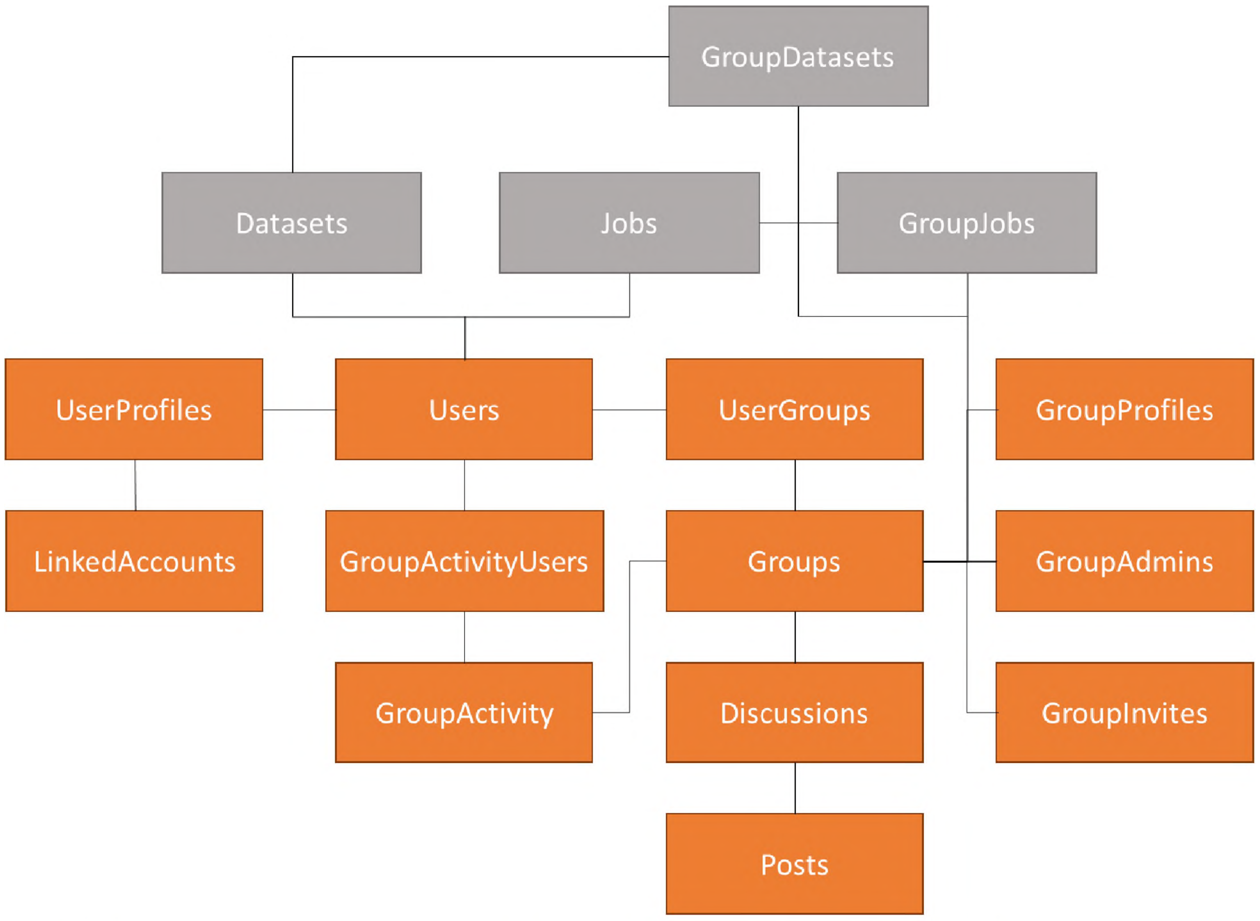


Fig. 5.7. *Account tables – a simplified representation of tables storing all user and group data (orange) and the tables linking users and groups to jobs and datasets (gray).*

base user details such as the login credentials, name and surname, and e-mail address. The *UserProfiles* table stores additional details such as the user's institution, department, and country of origin, as well as the details required to reset a forgotten password.

The *LinkedAccounts* table is used to store details of other accounts that HUMA has been linked to. For example, the HUMA web server allows users to run homology modeling jobs via the PRIMO web server (discussed in the next chapter). To allow this, a user must create an account on both web servers and link them.

The *Groups* table is a built-in Django table that simply stores the ID and name of a group. The *GroupProfiles* table was created to store additional group details, such as a description. Users are added to groups by creating an entry in the *UserGroups* table.

HUMA lets users be made administrators of groups. This is achieved by adding them to the *GroupAdmins* table.

When an administrator attempts to add a user to a group, an invitation is sent to the user. This invitation is created in the *GroupInvites* table.

Any activity that takes place in a group is recorded for accounting purposes. These recordings are stored in the *GroupActivity* table and associated with users via the *GroupActivityUsers* table.

Lastly, HUMA provides forums for users in a group where they can discuss any topic. Users can create a discussion with any topic and post comments to it. The discussion details are stored in the *Discussions* table and each comment (or “post”) is stored in the *Posts* table.

5.3.2.2. Jobs

The *jobs* section of the database is depicted in Fig. 5.8. Although analysis tools play an important role in the HUMA web server, this section of the databases is fairly simple and only consists of four tables.

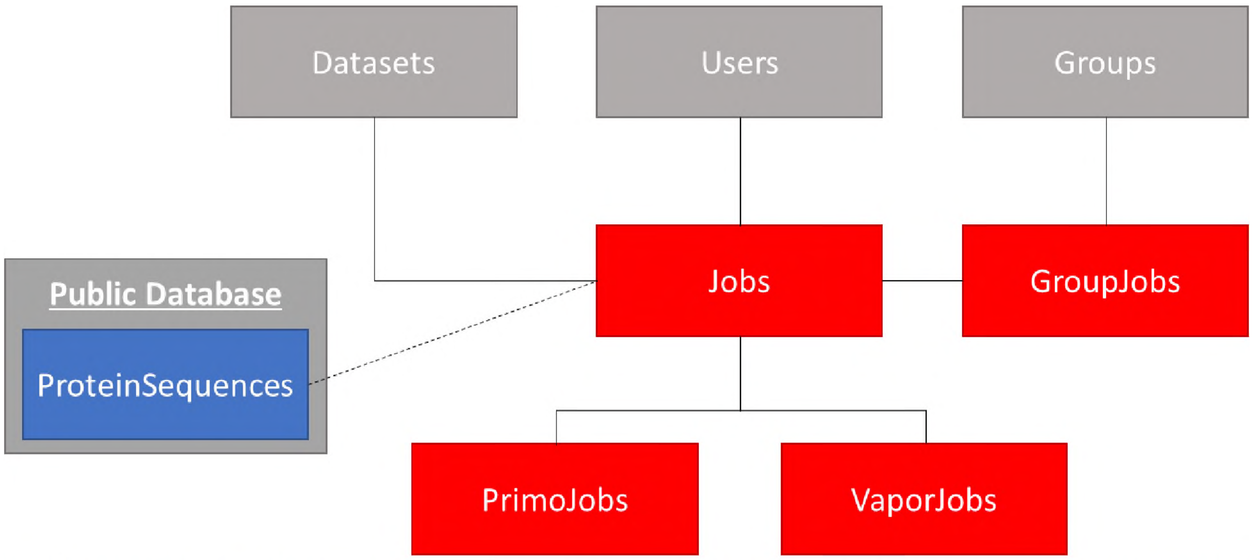


Fig. 5.8. Job tables- simplified representation of the tables containing job details (red) and the tables storing data linked to jobs (gray). A job is owned by a user and can be shared with many groups. Jobs can also be linked to datasets as well as protein sequences in the public database.

The main table is the *Jobs* table. This stores the base details about the job including the job name and description, status, timing information, and the type of job being run. To share a job with a group, and entry can be added to the *GroupJobs* table with the corresponding group and job IDs.

Currently, HUMA allows two types of job submissions, PRIMO and VAPOR. These tools accept different input parameters and produce different output files. To store these details efficiently, the *PrimoJobs* and *VaporJobs* tables were created. Where details that are common to both types of jobs are stored in the Jobs table, these two tables store details that are specific to the job type.

Another interesting thing to note is that jobs can be linked to datasets and protein sequences via the `Dataset_ID` and `Protein_Sequence_ID` fields. This allows the jobs to be displayed on the relevant dataset and protein pages (described in chapter 6). Because the *ProteinSequences* table is in the public database, there is no actual database relationship between the *Jobs* table and the *ProteinSequences* table, but the `Protein_Sequence_ID` can be used to quickly lookup the sequence in the public database.

5.3.2.3. Datasets

The *datasets* section of the database is depicted in Fig. 5.9. This section consists of seven tables.

The main table in the *datasets* section of the database is the *Datasets* table. This holds basic details such as the name, description, and the status of the dataset. Like the *Jobs* table from the previous section, the *Datasets* table links to the *GroupDatasets* table to allow the sharing of datasets with groups.

Variants can be uploaded to a dataset in VCF format. These uploads are tracked in the *Uploads* tables, which monitors them while they are being processed and added to the other tables in the section.

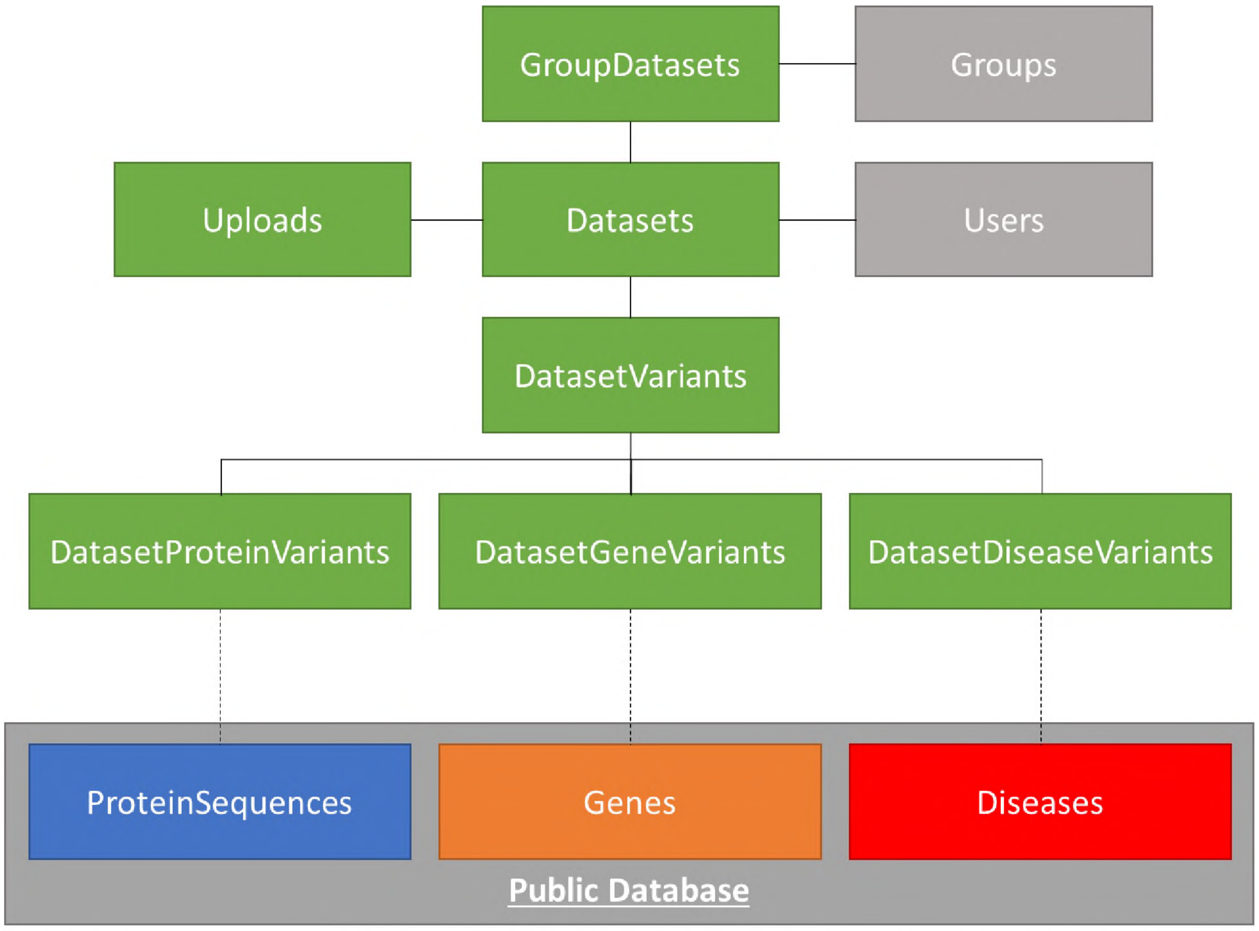


Fig. 5.9. Dataset tables – a simplified representation of the tables containing user-submitted datasets (green) and how those tables link to other data in the private and public databases.

When variants are uploaded, they are added to the *DatasetVariants* table before being processed in a similar manner to the public data. This means that they are mapped to genes, proteins, and diseases based on their chromosomal co-ordinates. When variants are successfully mapped to a gene, protein, or disease, the mapping details are stored in the *DatasetGeneVariants*, *DatasetProteinVariants*, and *DatasetDiseaseVariants* tables. Because the public data is in a different database, no actual relationship between these tables and the corresponding tables in the public database exists. However, the identifiers of the linked genes, proteins, and diseases are stored in the private tables so that these details can be looked up quickly.

5.4. CORMAP

With the data downloaded from the various sources discussed in section 5.2 and the database designed as discussed in section 5.3, the next step was to populate the database with the data from the source files. Initially, we attempted to do this using Python scripts that parsed the files and extracted the relevant data. This worked well for some of the smaller files, especially if they were in TSV format. However, for some of the larger and more complex files, such as the Uniprot `.dat` files, Python proved to be too slow. This was especially evident with the dbSNP VCF file, which contained over 150 million variants, all of which had to undergo additional processing to map them to genes, proteins, and diseases.

As we could not achieve the necessary performance to parse and insert these files into the database using Python, we attempted to write the scripts using C++. The scripts were structured in such a way that they would extract records from a source file and build up batch Structured Query Language (SQL) statements. Initially, for each different database table that we needed to insert data into, the code that generated the SQL statement had to be rewritten as the tables had different columns. However, much of the code for inserting batches remained the same across the different scripts. As such, we decided to generalize this code into a library that was dedicated to database interactions and could be reused in each of the parser scripts to insert data into the database. Thus, the C++ Object Relational Mapper (CORMAP) was born.

5.4.1. Object Relational Mapping

Object Relational Mapping (ORM) tools are programming libraries that can be used to map database tables to programmatic classes. These ORMs allow developers to interact with the database without having to write SQL. Usually, developers simply create a class that corresponds to a table in the database with attributes that correspond to columns in the table. Inserting a record into the database is then as simple as creating an instance of that class (*i.e.* an object) and using a save function provided by the ORM.

Similarly, functions are provided that can be used to read, edit and delete data from the database, all without requiring the developer to write any SQL or code that interacts directly with the database. Some ORMs even allow developers to first define the classes that represent tables and then use these definitions to generate a brand-new database from scratch. The Django ORM is one such ORM that provides functionality to generate databases from scratch off a set of classes.

5.4.2. CORMAP classes

CORMAP was designed to provide Django developers with a tool that feels like the Django ORM in C++. The tool is still a work in progress and functionality is very limited, but the ability to perform batch inserts, which is not available in many ORMs, gives CORMAP a large performance bonus. CORMAP has been open-sourced and can be downloaded from https://github.com/davidbrownza/CORMAP.

The CORMAP source code is split into three main classes (Fig. 5.10). The first class is the *DBConnection* class. It is the lowest level class, responsible for connecting to the database and sending communications. It provides functions to open and close connections to the database and prepare and execute SQL statements.

The second class is the *Model* class. This class is used as a base class, which developers must inherit from when they create their own classes that represent tables in the database. The *Model* class currently provides functions to insert single records as well as batches of records. It also has a truncate function, which deletes all records in a table. Additionally, it provides functions that allow developers to create the attributes in their classes that correspond to the columns in the database table. Currently, only integer, float, and text fields are supported.

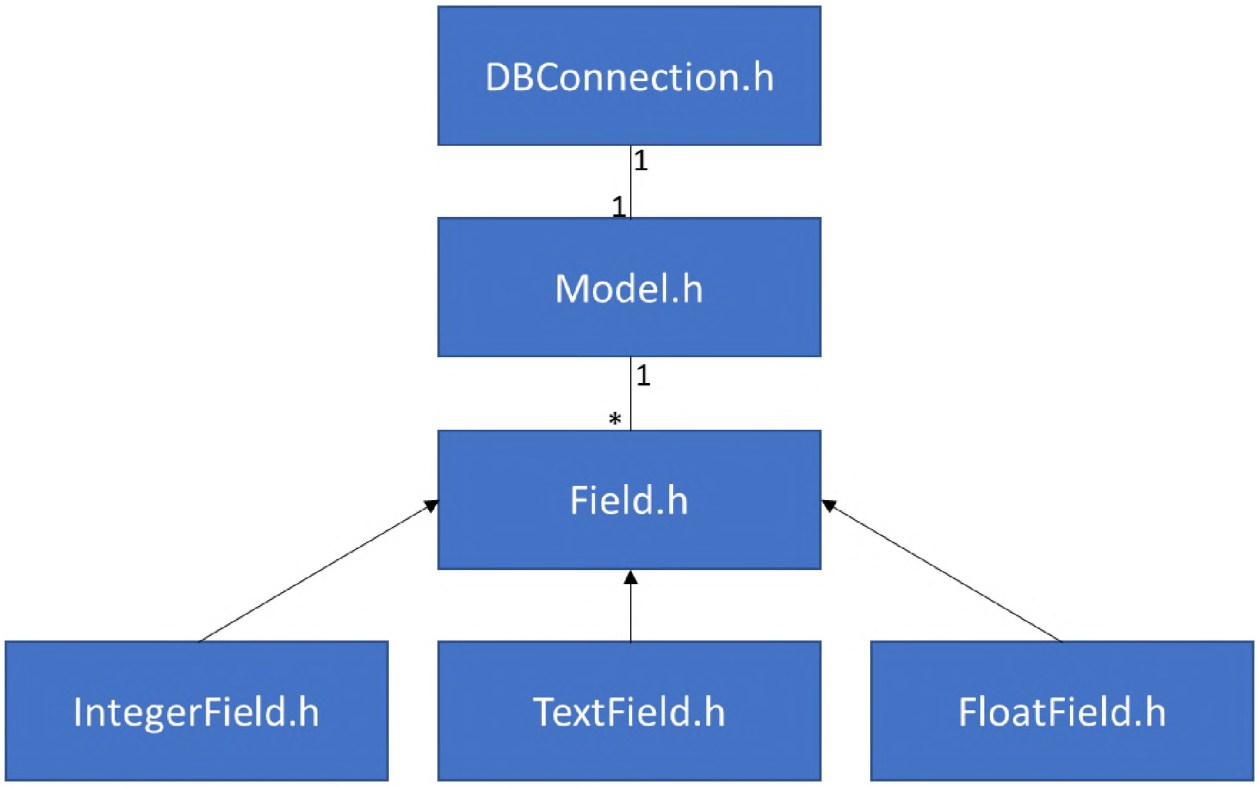
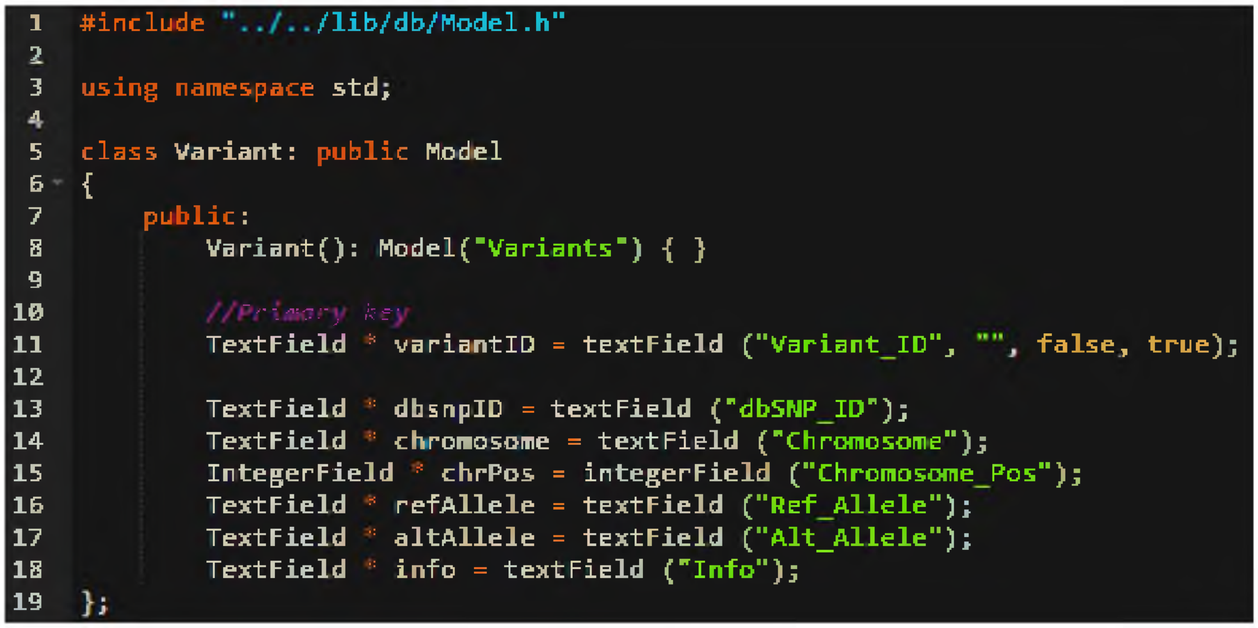


Fig. 5.10. *CORMAP classes – a class diagram depicting how the design of CORMAP.*

The third class is the *Field* class. This class is used to create the fields that represent the columns in the database. It contains functions to get the column name and type and to check whether the column is a primary key, unique, or nullable. There are three additional classes that extend the *Field* class. These three classes correspond to the three supported column types *i.e.* the *IntergerField*, *FloatField*, and *TextField* classes. These classes each contain functions to get and set the value in the field. In order to add support for additional field types, new classes that extend the *Field* class and provide methods to get and set the value must be created.

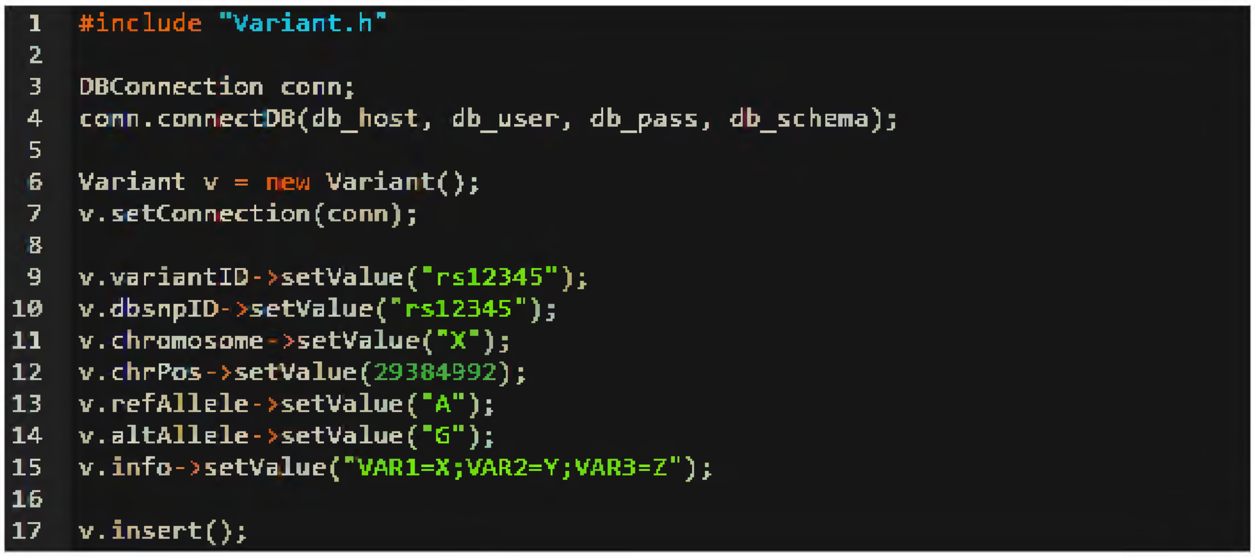
5.4.3. Example usage

The best way to describe how CORMAP works is to provide a real-world example. Fig 5.11 shows the class written to correspond to the Variants table in the public HUMA database. The class has been named *Variant* and inherits from the *Model* class. The *Variant* constructor provides the name of the table to the super class. The attributes of the class represent the columns in the table and use the *textField* and *integerField* functions provided by the *Model* class to create objects of the *TextField* and *IntegerField* classes. These functions take the names of the columns as arguments.



```
#include "../../lib/db/Model.h"

using namespace std;

class Variant: public Model
{
    public:
        Variant(): Model("Variants") { }

        //Primary key
        TextField * variantID = textField ("Variant_ID", "", false, true);

        TextField * dbsnpID = textField ("dbSNP_ID");
        TextField * chromosome = textField ("Chromosome");
        IntegerField * chrPos = integerField ("Chromosome_Pos");
        TextField * refAllele = textField ("Ref_Allele");
        TextField * altAllele = textField ("Alt_Allele");
        TextField * info = textField ("Info");
};
```

Fig. 5.11. Variant.h model class – corresponds to the Variants table in the public HUMA database.



```
#include "Variant.h"

DBConnection conn;
conn.connectDB(db_host, db_user, db_pass, db_schema);

Variant v = new Variant();
v.setConnection(conn);

v.variantID->setValue("rs12345");
v.dbsnpID->setValue("rs12345");
v.chromosome->setValue("X");
v.chrPos->setValue(29384992);
v.refAllele->setValue("A");
v.altAllele->setValue("G");
v.info->setValue("VAR1=X;VAR2=Y;VAR3=Z");

v.insert();
```

Fig. 5.12. Variant.h usage – example of how the Variant class could be used to insert a record into the database.

Fig 5.12 shows how this class can be used to insert a record into the database. In 17 lines of code, CORMAP lets the developer connect to the database and insert a record without needing to write a line of SQL. The *insert* method generates all the required SQL without the developer ever needing to think about it.

5.4.4. Future work

CORMAP is far from finished and currently has limited use cases. It was designed to rapidly insert data into the database, which it achieves, but it lacks functionality that is necessary for it to be a true ORM, such as reading, deleting and manipulating records. Future work will focus on adding this functionality as well as building in support for additional field types.

5.5. Populating the database

The public HUMA databases were populated using a semi-automated pipeline made up of Python and C++ scripts (using CORMAP). Python was used when possible, as it was far quicker to write the parser scripts using Python. C++ was used when the data files were large and complex and extra performance was required. This pipeline will be re-used in future to release updated versions of the database when new data becomes available (*e.g.* when a new version of the human reference genome is released).

Fig. 5.13 depicts the workflow used to populate the database. Each block in the figure represents a script used to populate one or more tables. Grey blocks represent scripts written in C++ while the white blocks represent Python scripts. The lines between the blocks represent dependencies. Red lines mean that manual intervention is required at this point *i.e.* the next script in the pipeline cannot be run automatically.

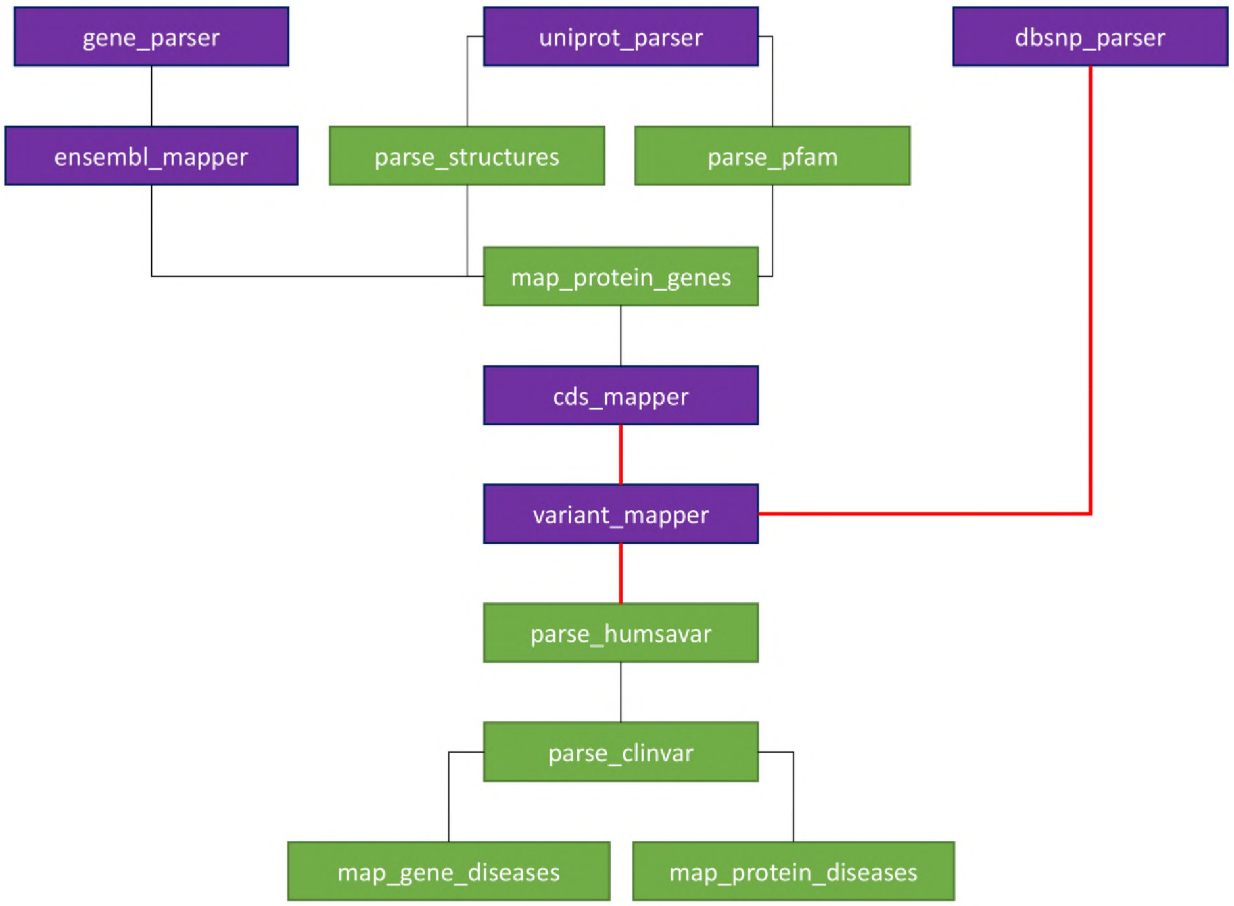


Fig. 5.13. *Populating the database – this workflow depicts the process that was followed when populating the public HUMA database. Purple blocks represent custom-built C++ utilities, while green blocks represent Python scripts. Red lines depict areas that required manual intervention. This figure was re-used from our paper in Human Mutation.*

The top three scripts in Fig. 5.13 can all be run in parallel as they do not depend on one another. The *dbsnp_parser* script is responsible for parsing the dbSNP VCF file and inserting all the variants into the *VariantStore* table.

The *gene_parser* script is used to parse gene data from Ensembl and HGNC and insert it into the *Genes*, *GeneNames*, and *GeneSymbols* tables. It also inserts data mapping genes to diseases, thus, adding data to the *Diseases* and *DiseaseGenes* tables.

The *uniprot_parser* script is responsible for parsing the two `.dat` files and the Fasta file from Uniprot and using the extracted data to populate the *UniprotProteins*, *ProteinIdentifiers*, *ProteinFeatures*, *ProteinPublications*, *Publications*, *UniprotIsoforms*, and *ProteinSequences* tables.

Once the *gene_parser* script has executed, the *ensemble_mapper* script can be run. This script inserts the protein, exon and CDS data from Ensembl and is responsible for mapping the exons to proteins and proteins to genes. As such, this script populates the *EnsemblTranscripts*, *EnsemblExons*, *TranscriptExons*, and *ProteinSequences* tables.

Once the *uniprot_parser* has finished executing, the *parse_structures* script is used to add the PDB data to the database. This script populates the *ProteinStructures*, *Chains*, and *Ligands* tables.

The *map_protein_genes* script links Uniprot proteins to the Ensembl genes by first using an ID mapping file that can be obtained from the Uniprot FTP site and then by looking for matching Ensembl proteins, which are already mapped to Ensembl genes.

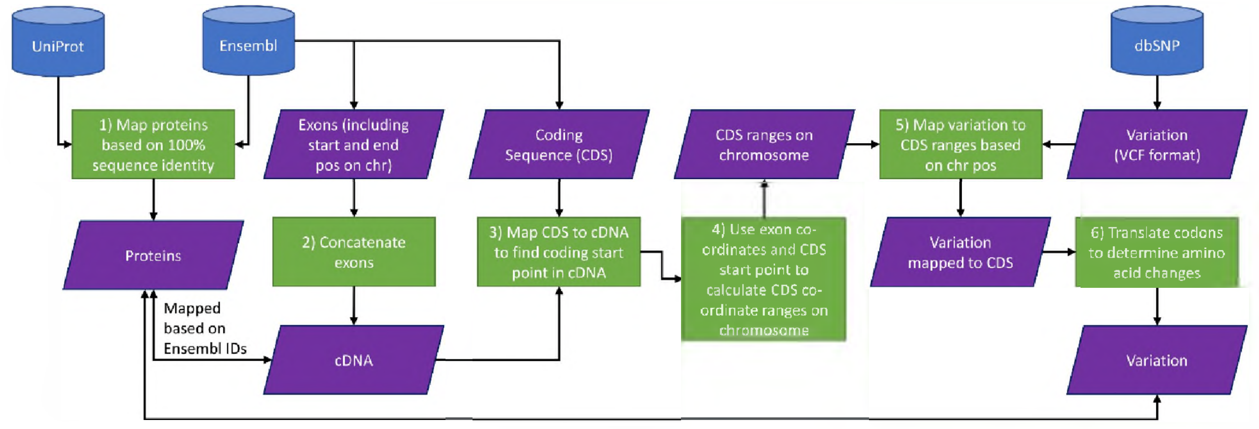

UniProt
Ensembl
dbSNP
1) Map proteins based on 100% sequence identity
Exons (including start and end pos on chr)
Coding Sequence (CDS)
CDS ranges on chromosome
5) Map variation to CDS ranges based on chr pos
Variation (VCF format)
Proteins
2) Concatenate exons
3) Map CDS to cDNA to find coding start point in cDNA
4) Use exon co-ordinates and CDS start point to calculate CDS co-ordinate ranges on chromosome
Variation mapped to CDS
6) Translate codons to determine amino acid changes
Mapped based on Ensembl IDs
cDNA
Variation

Fig. 5.14. *Mapping variants to proteins – a depiction of the process used to map variants from dbSNP to protein sequences from Uniprot and Ensembl.*

The *cds_mapper* script calculates CDS ranges. The process by which it does this is depicted in Fig. 5.14. For each protein, the exon sequences linked to that protein are stored in the *EnsemblExons* table along with the chromosomal co-ordinates. The *cds_mapper* script fetches the exons for a protein and concatenates the sequences to form the coding DNA (cDNA). Next, the CDS (stored in the *EnsemblTranscripts* table) is mapped to the cDNA so that chromosomal co-ordinates of the CDS can calculated based on the chromosomal co-ordinates of the exons. The outputs of this is a set of CDS ranges for each protein that are then stored in the *CDSRanges* table.

The next step is to execute the *variant_mapper* script, which maps variation to genes and proteins. Because of the sheer number of variants that are added to this table, the primary key in the Variants table and the foreign keys linking other tables to the Variants table must be removed before the script is executed. Indices decrease the speed at, which data can be inserted into a table as the table gains more records – a quirk of MySQL's InnoDB engine. Once the *variant_mapper* script has been executed, the keys can be reintroduced. These are the manual steps represented by the red lines in Fig. 5.13. In future, this will also be automated.

Variants are mapped to genes based on chromosomal co-ordinates and the position of the variant in the gene is calculated and used to populate the *GeneVariants* table. Mapping variants to proteins is slightly more complicated. Each variant is mapped to a CDS range. All the CDS ranges for a protein are then concatenated to form the CDS with the variant. The mutated CDS is then translated to a protein sequence with a potentially mutated residue (it could be a synonymous SNV). This process allows us to gather the details required to populate *ProteinVariants* table. Any variant that is mapped to a gene or protein is first added to the *Variants* table.

The last few scripts are quick to execute. The *parse_clinvar* script uses data from ClinVar to map variants to diseases and, thus, populate the *DiseaseVariants* table.

The *parse_humsavar* script maps diseases to variants and proteins. It is used to add data to the *DiseaseVariants* and *ProteinDisease* tables.

Lastly, the *map_gene_diseases* and *map_protein_diseases* scripts map diseases directly to genes and proteins respectively by checking if the disease is associated with a variant that has been mapped to the disease or protein.

5.6. Summary

In this chapter, we have discussed the design, development, and population of the HUMA databases. HUMA consists of two databases. The public database contains data that has been aggregated from various public sources including dbSNP, ClinVar, HGNC, Ensembl, Uniprot, Pfam and the PDB. Data from these databases is inserted into the database and mapped to each other using a semi-automated workflow containing several Python and C++ scripts and tools. A simple ORM, CORMAP, was developed to assist with database interactions when developing the C++ parser tools.

The private database is made up of all the tables that require user authentication and authorization to access. This includes account and group details, analysis results, and private datasets. Functionality has been built into this database that facilitates collaboration. This functionality includes the ability to create and join groups and share data with those groups. Additionally, the private database allows some interaction with the public database by allowing results from analysis tools, as well as variants that have been uploaded, to be linked to public data by storing the relevant identifiers in the private database.

There is still further development that can be done on the HUMA database. Currently, four categories of data (genes, proteins, variation, and diseases) are stored. In future, we will add additional data from public sources. This will include the following:

- Known drugs that target proteins (from Drugbank [221])
- Protein interaction networks (from IntAct [222])
- Pathways (from Reactome [223])

Similarly to the VnD [152] database, we will also calculate and store the physicochemical changes that occur in a protein as the result of a variant.

6. HUMA web server

6.1. Introduction

The HUMA web server was developed to provide web-based access to the databases discussed in chapter 5. Additionally, it provides access to tools that allow users to visualize, analyze, share, and upload data. As such, the HUMA web server plays a vital role in facilitating the use of the HUMA database. From here on out, we will refer to the HUMA web server simply as HUMA. HUMA is freely accessible at https://huma.rubi.ru.ac.za.

HUMA can be accessed in two ways. Firstly, a user-friendly website (see section 6.3) has been developed that provides users with click-based access to all the functionality provided by the web server.

Secondly, a RESTful web API (see section 6.2.2) has been developed that provides programmatic access to the server. As with JMS, the web interface has been developed on top of the RESTful web API. This means that all data and functionality that is available via the web interface is also available via the API. It also ensures that updates made to the interface will always be reflected in the API, meaning that the API will never fall behind in terms of functionality.

As mentioned before, various tools have been integrated into HUMA to facilitate research and collaboration. Some of these tools, such as the visualization, data uploading, and data sharing tools are built directly into HUMA. Other tools are integrated, directly or indirectly, via JMS.

Although HUMA was built specifically with the H3Africa initiative in mind, its provides a useful resource for researchers worldwide. Over time, HUMA will integrate more and more data and tools, making it a platform for the analysis of genetic variation in humans, rather than simply a database and web server.

6.2. Web server

All websites adhere to a client-server architecture, where the client (*i.e.* the webpage in the user's browser) sends requests to the server, and the server responds based on the request. As such, the server, also known as the backend, is where the business logic of an application resides. Business logic refers to the code that performs the actual tasks of the application.

As was the case with JMS, the HUMA web server was developed with the Python Django web framework, while the RESTful web API was developed using the Django REST Framework. The design of these components will be discussed below.

6.2.1. Modular server design

HUMA makes use of a modular design (Fig. 6.1), where components are broken up into logical units that can be reused. HUMA currently consists of seven modules, although this will grow as

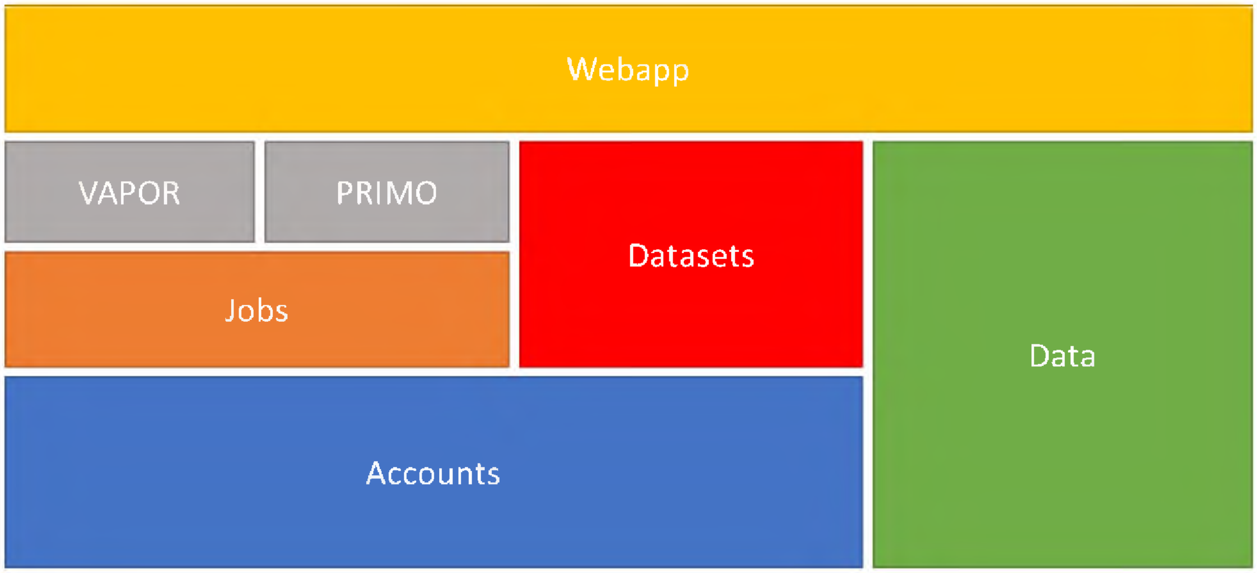


Fig. 6.1. Modular server design – the HUMA web server makes use of a modular design, where each module performs a deignated task. These modules are designed to be re-usable, and can be switched out for new modules if needed. The design keeps the server code clean and manageable.

additional tools are added to the server. These modules all form part of the same Django application, but could theoretically be broken into separate services and hosted on completely different servers. This would let HUMA scale infinitely, should the need arise (and given the resources).

Barring the *webapp* module, all HUMA modules provide RESTful web APIs that permit programmatic access. The seven existing modules are discussed below.

6.2.1.1. Accounts

The *accounts* module is responsible is responsible for handling user creation, authentication, and authorization, as well as group creation and management. This includes creating and posting to discussions. Essentially, the *accounts* module manages all interactions with the *accounts* section of the private database.

The *jobs* and *datasets* modules depend on the *accounts* module to manage ownership and sharing of jobs and datasets respectively. As such, the *primo* and *vapor* modules also depend on the *accounts* module through their dependence on the *jobs* module.

6.2.1.2. Data

The *data* module is responsible for providing access to the public database. Access to the public database is unrestricted, which means that users do not need to be authenticated to access it. As such, the *data* module is the only module that does not depend on the *accounts* module.

The *data* module only provides users with read access. This means that users cannot write to the public database *i.e.* users cannot create new records, or update or delete existing records in the public database.

Given a search term, the web API of the *data* module allows users to search for data from any of the four categories discussed in the previous chapter *i.e.* genes, proteins, variation, or disease. It also allows users to fetch a given entry from one of these categories given the entry's identifier.

6.2.1.3. Datasets

The *datasets* module allows users to upload custom datasets to the private database. This module is dependent on the *accounts* module to manage ownership and sharing of datasets.

The datasets module lets users upload VCF files of up to 100MB at a time. This corresponds to roughly 500 000 variants per file. The module also provides a First-In-First-Out (FIFO) queuing service that manages the processing of files. Users who want to upload large files must break their files up into 100MB chunks and then upload them one-by-one. These files will then enter the queue and be processed in order, along with other users' uploads. Although this may frustrate some users, it is a necessary evil to prevent a single user from uploading a massive file and blocking the queue for a prolonged period.

The VCF processing service is a workflow consisting of a several sequential stages (Fig. 6.2). Firstly, it parses the uploaded VCF file and inserts the variants into the *DatasetVariants* table in the private database.

Next, the uploaded variants are mapped to variants in the public database. When a dataset variant is mapped to a public variant, any diseases that have been associated with the public variant are then automatically associated with the dataset variants.

Dataset variants are then mapped to genes and proteins based on chromosomal co-ordinates. This is done in the same way as described for the public database.

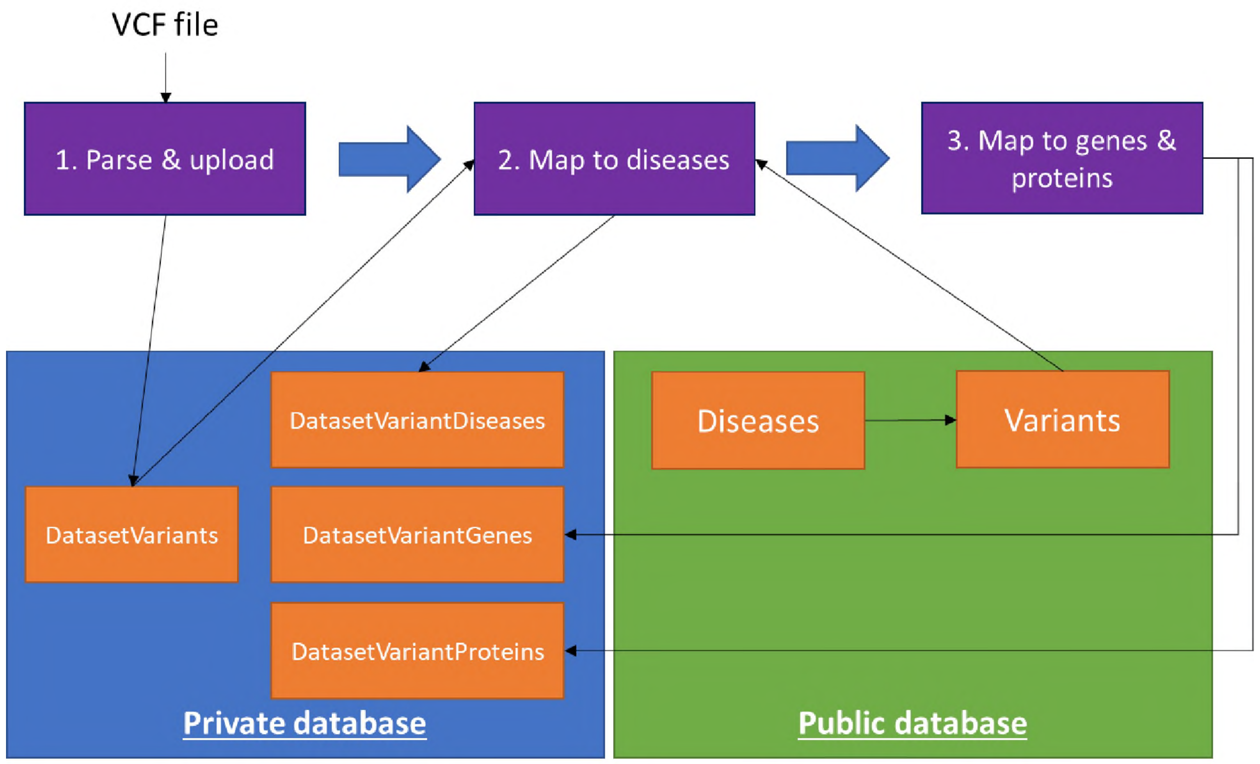


Fig. 6.2. Uploading datasets – the workflow used when populating uploading data consists of 3 steps: ***1)*** *the VCF file is parsed and variants are added to the DatasetVariants table;* ***2)*** *uploaded variants are compared to the variants in the public datasbase and, if a match is found with associated diseases, those diseases are also linked to the uploaded variant; and* ***3)*** *uploaded variants are mapped to genes and protein sequences based on their chromosomal co-ordinates.*

The end result of this process is that all uploaded variants are stored in the private database and mapped to disease, genes, and proteins in the public database. As will be discussed later, these variants can then be visualized in the protein structure and analyzed using the integrated HUMA tools such as VAPOR and PRIMO.

6.2.1.4. Jobs

The *jobs* module allows users to submit jobs to the cluster. It provides generic functions that are used across all tools to store, download, update, delete, and share job information. It also provides helper functions that make it easy for integrated tools to create job records, submit jobs to JMS, fetch files from JMS, and retrieve information about the status of jobs running on the cluster.

6.2.1.5. VAPOR

Although the *jobs* module takes care of generic job functionality such as submitting and sharing jobs and downloading job results, functionality specific to a given tool must still be catered for. The *vapor* module is responsible for functionality specific to VAPOR. To do this, it provides three web API endpoints.

The first endpoint receives user input to submit a VAPOR job. The *vapor* module provides a helper function to convert the input into a JSON structure compatible with JMS. A *jobs* module helper function is then used to submit the job to JMS.

The second endpoint receives a notification from JMS when the VAPOR workflow finishes running. It then uses helper functions from the *jobs* module to retrieve the result files from JMS.

The final endpoint is used by the web interface to retrieve the results from the server and display them on a webpage.

VAPOR integration into HUMA is discussed further in section 6.5.1.

6.2.1.6. PRIMO

Like VAPOR, functionality specific to running PRIMO is handled by the *primo* module. As such, similar web API endpoints to submit jobs, receive notifications, and return results to the web interface are provided by the module. These endpoints differ slightly from those provided by the *vapor* module, as jobs are submitted via the PRIMO web server, rather than directly to JMS.

Because PRIMO is a standalone web server, integrating PRIMO into HUMA requires users to link their HUMA accounts with a valid PRIMO account. This allows jobs to be submitted to PRIMO via the HUMA web interface. As a result of this, two additional endpoints are provided by the *primo* module, which allow a user to link and unlink their account to a PRIMO account. The technical details of this process are discussed further in section 6.5.2.

The *primo* module also includes helper functions that convert user input into the format required by the PRIMO web server as well as functions that convert PRIMO job status identifiers to HUMA identifiers and fetch results from the PRIMO web server.

6.2.1.7. Webapp

The *webapp* module serves the web pages that interact with the RESTful web APIs of the other modules. Since the HUMA website is a single page application (SPA), the *webapp* module only serves one fully-formed HTML page. However, to reduce the initial loading time, this page does not contain all the content available to a user. Instead, as the user interacts with the interface, additional HTML components are loaded as needed. The *webapp* is also responsible for serving these HTML components.

The HUMA website is large and offers many features. As such, loading the entire site at once would come with a big performance hit. By serving smaller HTML "chunks" as they are needed, HUMA greatly reduces the bandwidth requirements of any given request. This is important for African groups that may not necessarily have the same level of internet connectivity as their western and eastern compatriots. To further reduce bandwidth requirements, once a component has been loaded the first time, it is stored on the client-side and does not need to be fetched again.

A list of components served by the *webapp* module is described in Table 6.1.

Relative URL	**Description**
/	Returns the fully-formed web page that serves as the skeleton/layout for the SPA
/account_partial	If the user is logged in, returns the user menu component displayed in the top right hand corner of the screen. If the user is not logged in,

	a “Sign In” button is displayed.
/login	The HTML component that allows users to enter their username and password to log in.
/reset	The HTML component that allows users to reset a forgotten password.
/groups	The HTML content that is loaded when viewing the page displaying the list of groups that a user is in.
/group_detail/<id>	The HTML content that is loaded after selecting a group.
/register	The HTML component that allows users to enter their details to register.
/home	The HTML content that is displayed on the home page.
/genes	The HTML content that is loaded when browsing to the “Genes” page.
/proteins	The HTML content that is loaded when browsing to the “Proteins” page.
/variants	The HTML content that is loaded when browsing to the “Variants” page.
/diseases	The HTML content that is loaded when browsing to the “Diseases” page.
/vapor	The HTML content that is loaded when browsing to the “VAPOR” page. This allows users to provide the input to submit a VAPOR job.
/primo	The HTML content that is loaded when browsing to the “PRIMO” page. This allows users to provide the input to submit a PRIMO job.
/data	The HTML content that displays user datasets and lets users upload new data.
/profile	The HTML content that displays user profile details and allows the user to update these details.
/jobs	The HTML content displayed when viewing jobs.
/dataset_protein	The HTML component that lets dataset variants be displayed in the protein sequence and structure.

Table 6.1. HTML endpoints provided by the *webapp* module

6.2.2. RESTful web API

The Django REST framework [64] was used to create the RESTful web API for HUMA. This framework was previously described in section 2.3.2 and will not be discussed again here.

The HUMA web API is documented at https://huma.rubi.ru.ac.za/docs. Documentation is generated by Django REST Swagger [224], an opensource tool for automatically documenting a Django REST API using the Swagger UI.

6.2.3. Search

When providing access to a large amount of data, it is also important to provide a meaningful way of searching through that data. As such, search is an important part of the HUMA web server.

6.2.3.1. MySQL as a search engine

Initially, search was performed by querying the MySQL database using the Django ORM or raw SQL for more advanced searches. Although this technically worked, it had several caveats.

Firstly, performing searches across multiple tables in a relational database requires those tables to be “joined”. For smaller databases with few tables, this is not an issue. However, this process doesn’t scale to massive databases. As such, searches were initially limited to searching through a small subset of the available data. For example, performing a search via the *Proteins* page would only search based on protein accession numbers and protein names. If the user entered a disease name or gene accession number, results would only be returned if those terms happened to occur in the protein name. Other important fields that were left out of the search included protein domains and families and PDB structures and chains.

Secondly, when performing a SQL query, results are returned in the order they are found in the database rather than being ranked by relevance. For example, searching for “hemoglobin” will return all entries in the database that have “hemoglobin” in the name. The entry “P69905” for “Hemoglobin subunit alpha” is a reviewed protein sequence from SwissProt that has multiple publications and structures associated with it. On the other hand, the entry “G3V1N2”, is an unreviewed hemoglobin alpha sequence with no structures associated with it. In this example, “P69905” is clearly the more interesting entry and should appear higher in the search results, but because the SQL query simply tries to match the search term to the query, the is no intelligence that goes into ranking the results. Via the SQL query, the developer is technically able to order the results via other means (*e.g.* by reviewed/unreviewed, number of related structures, *etc.*), but this still does not solve the issue in a satisfactory manner. For example, sometimes a match without related structures *will* be more relevant due to some other reason such as the closeness of the match to some other field. As such, a more intelligent search mechanism is required.

Lastly, SQL queries fail to perform meaningful searches in the presence of typos or misspelt words. In biological sciences, where names can be long and complex, this poses a problem. Additionally, curtained names can be spelt in different way: hemoglobin vs haemoglobin.

MySQL *does* offer a limited way of catering for the latter two issues via full-text searches. However, performance remains an issue here. Instead, it is better to use something that has been designed for this purpose.

6.2.3.2. Elasticsearch

Elasticsearch [225] is a state-of-the-art, open source search engine, written in Java and based on Apache Lucene [226]. It is a NoSQL document store *i.e.* Elasticsearch is not a relational database. Data is stored in an index (comparable to a database schema in an RDBMS). An index can be broken up into shards (comparable to partitions in an RDBMS) and distributed across machines. Each shard can be replicated multiple times. An index can consist of many types (comparable to tables in an RDBMS) consisting of documents (comparable to rows in an RDBMS). A document is essentially a JSON blob. The fields in the JSON are comparable to columns in an RDBMS. A type, therefore, consists of a list of documents (rows) made up of fields (columns). Unlike in a relational database, where the schema is consistent for a table, each document (row) can have different fields (columns).

Unlike with a relational database, every field in an Elasticsearch document can be indexed, allowing for fast search across all fields. Replicating shards speeds up searches even further by allowing searches to occur in parallel, but only if the replica is on a different host. In the long term, this allows a system using Elasticsearch to continue scaling as usage increases. Replicas also provide failover support *i.e.* if the primary shard dies, one of the replicas is promoted to primary.

Elasticsearch also performs fuzzy queries using the Levenshtein edit distance algorithm [227], which measures the similarity between two strings. The edit distance between two strings is equivalent to the minimum number of transformations required to transform the first string into the second. As such, the higher the edit distance, the greater the difference between the two strings. Elasticsearch makes use of this concept to match entries to search terms even when there are spelling mistakes or typos.

Being a search engine, Elasticsearch, by default, ranks results based on their similarity to the search term. However, Elasticsearch also goes one step further by allowing users to specify weightings for individual fields when searching through a document. In other words, it allows users to raise the relevance of a match in a particular field so that it contributes more to the overall ranking of the document in the search results. This proves useful in our case, as it allows us to place more emphasis on a match in the "Protein_Name" field, for example, as opposed to a match to the "Function" field, which could contain the names of multiple interacting proteins.

The combination of high speed searches using fuzzy querying and receiving ranked results made Elasticsearch a match for our needs and as such, was implemented in HUMA. The "django-elasticsearch-dsl" Python library (https://github.com/sabricot/django-elasticsearch-dsl) was used to create three separate Elasticsearch indices; one each for proteins, genes, and diseases. Each index contained only one "type". The fields that were indexed for each of these indices are depicted in Table 6.2. Searches are performed across all fields for a respective index.

Index	Field (weighting)	Description
Proteins	Uniprot_Acc	Uniprot accession number
	Uniprot_ID (x5)	Uniprot ID
	Long_Name (x2)	Long name (from Uniprot)
	Short_Name	Short name (from Uniprot)
	Submitted_Name	Submitted name (from Uniprot)
	Function	Description of protein function
	Genes (x5)	List of HGNC symbols
	Diseases (x4)	List of disease names
	Alternative_Names	List of alternative names from Uniprot
	Alternative_Accessions	List of alternative accessions from Uniprot
	Domains	List of protein domains/families from Pfam
	PDB_IDs	List of PDB IDs
	Is_Reviewed	Boolean – true if the protein is from SwissProt
	Has_Function	Boolean – true if there is text in function field
	Number_Of_Structures	Integer – count of the number of PDB structures
Genes	Ensembl_Gene_ID	Ensembl gene ID
	Gene_Name (x5)	Gene name from Ensembl
	HGNC_ID (x2)	HGNC ID
	HGNC_Symbol (x2)	Gene symbol from HGNC
	Entrez_Gene_ID (x2)	NCBI gene ID
	MIM_ID	OMIM gene ID
	Proteins (x5)	List of Uniprot accessions
	Diseases (x4)	List of disease names
	Alternative_Names (x3)	List of alternative names from HGNC
	Alternative_Symbols (x2)	List of alternative symbols from HGNC
Diseases	Disease_ID	Disease ID calculated by HUMA based on source ID
	Disease_Name (x2)	Disease name obtained from external source
	Proteins	List of Uniprot accessions
	Genes	List of HGNC symbols
	Is_OMIM	Boolean – true if OMIM is external source
	gene_count	Integer – count of genes linked to disease
	protein_count	Integer – count of proteins linked to disease
	variant_count	Integer – count of variants linked to disease

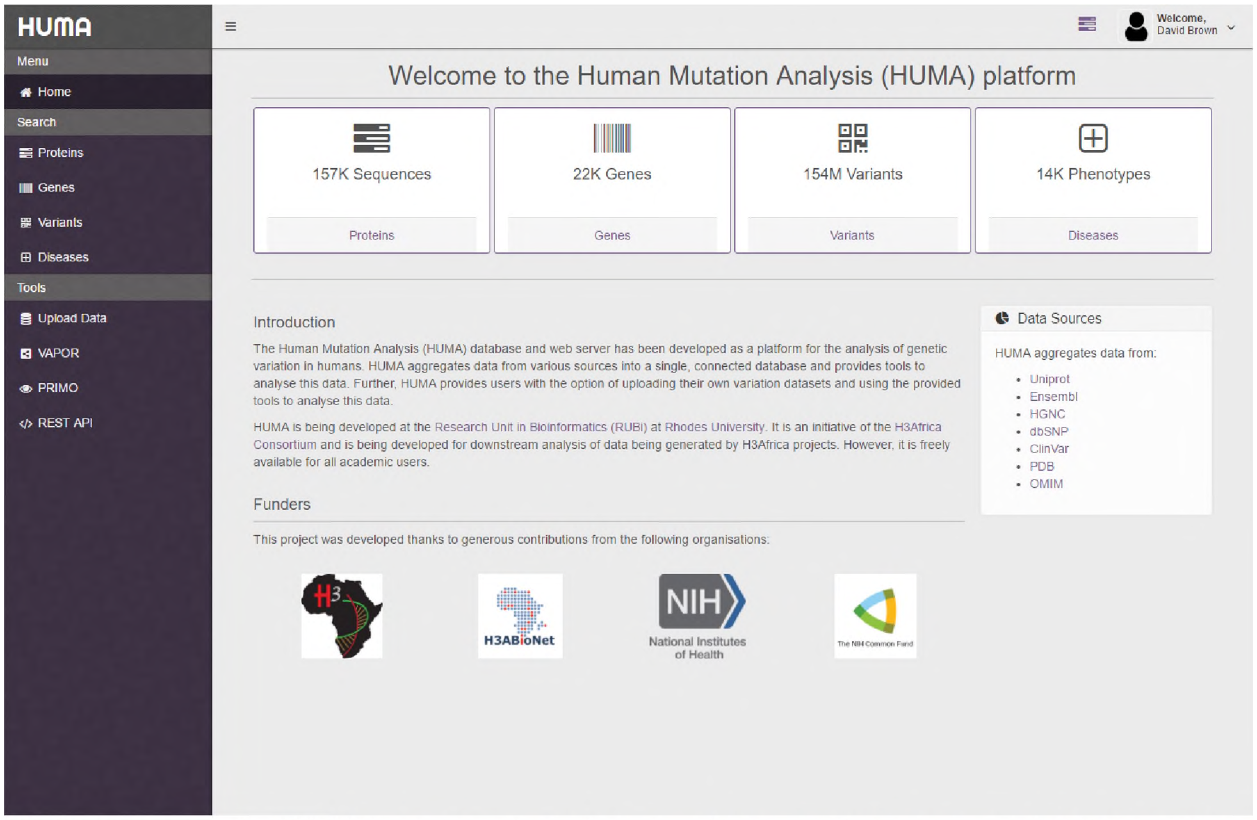
Table 6.2. Elasticsearch indices. Search performed on a particular index will search across all fields within the index, with the exception of the count fields in the Diseases index, which are simply used on the results page, and the "Number_Of_Structures" field in the Proteins index, which is used to increase the relevance of a hit. The numbers in the brackets represent the level that each field is weighted at.

6.3. Web client

A web-based interface was developed for HUMA to serve data to users (Fig. 6.3). Several techniques have been used to lower the bandwidth requirements of this website. These techniques were alluded to in the previous section and include the use of an SPA architecture, the loading of content on demand, and the caching of content on the client side to avoid unnecessary reloading.

6.3.1. Single Page Application

In contrast to JMS, which combined the use of SPA techniques with multiple webpages to improve the user experience, HUMA is a fully implemented SPA. As mentioned previously, this means



Welcome to the Human Mutation Analysis (HUMA) platform

157K Sequences	22K Genes	154M Variants	14K Phenotypes
Proteins	Genes	Variants	Diseases

Introduction

The Human Mutation Analysis (HUMA) database and web server has been developed as a platform for the analysis of genetic variation in humans. HUMA aggregates data from various sources into a single, connected database and provides tools to analyse this data. Further, HUMA provides users with the option of uploading their own variation datasets and using the provided tools to analyse this data.

HUMA is being developed at the Research Unit in Bioinformatics (RUBi) at Rhodes University. It is an initiative of the H3Africa Consortium and is being developed for downstream analysis of data being generated by H3Africa projects. However, it is freely available for all academic users.

Funders

This project was developed thanks to generous contributions from the following organisations:

Data Sources

HUMA aggregates data from:

- Uniprot
- Ensembl
- HGNC
- dbSNP
- ClinVar
- PDB
- OMIM

Fig. 6.3. Home page – the HUMA home page contains statistics around the numbers in the database, links to data sources and funders, and an introduction to the site.

that the entire website is made up of a single webpage. The illusion of multiple pages is created by hiding and showing different content when, for example, a link or a button is pressed.

One of the reasons that JMS was not implemented as an SPA was to avoid making any single page become too large and unmanageable for the developer. Another reason was that loading all the HTML and JavaScript code at once would have produced a poor user experience on the initial page load *i.e.* users with slower internet connections would have experienced a long loading time when first browsing to the website.

In HUMA, the above issues were solved with the same solution. Although HUMA is an SPA, the HTML was written in several different files. The main HTML file was the *master.html* file, which contained the layout or "scaffolding" for the entire website. This is HTML that is common to every "virtual page" on the website and include things such as the top navigation bar, the side navigation bar, the content area, and anything else that remains constant on the site (Fig. 6.4).

The account partial, which is displayed in the top right-hand corner of the HUMA interface was written in a separate file called *account_partial.html*. This HTML component is fetched with an AJAX request and loaded into the interface when a user logs in. It displays the username and a dropdown menu with various user options.

Similarly, the HTML content for each "virtual page" is loaded into the content area when the user visits the relevant URL (*e.g.* by clicking one of the options in the sidebar). For example, if the user loads the page by typing the base URL (https://huma.rubi.ru.ac.za) into the address bar, the *master.html* page will first be loaded and the *home.html* page will be loaded into the content area via an AJAX request. If the user then clicks on the "Proteins" option in the side navigation bar, the *home.html* content is hidden and the *proteins.html* content is fetched and loaded into the content

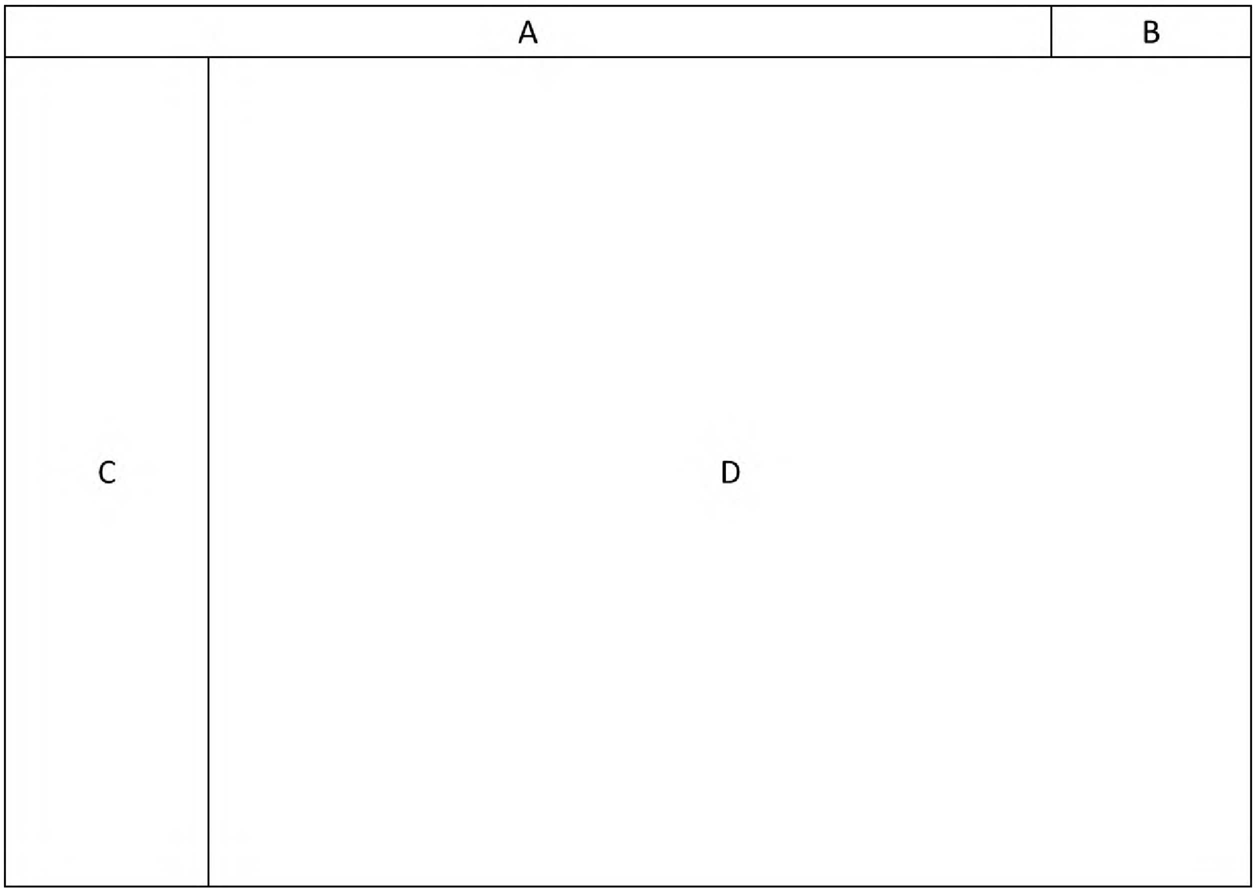


Fig. 6.4. Page structure – webpages in HUMA are made up of four sections: ***A)*** *the top navigation bar includes the page title as well as the account partial;* ***B)*** *the account partial, which is technically part of the top navigation bar, provides a "sign in" link if the user is not authenticated, or provides a menu with links to the user's profile, jobs, datasets, and groups pages;* ***C)*** *the side navigation bar provides navigation links to the data and tool pages; and* ***D)*** *the content area, where content is loaded on demand.*

area via an AJAX request. If the user decides to return to the "Home" page, the *proteins.html* content is hidden and the *home.html* content is shown again. This does not require an AJAX request to the server as the *home.html* content was only hidden and not destroyed when browsing to the "Proteins" page. As such, content for a given option only needs to be loaded once, thereby saving bandwidth.

Big content pages, such as the *proteins.html* page, are further broken up into sections. For example, the HTML content for each data block on the "Proteins" page is stored in a separate file *e.g. protein_analysis.html*, *protein_genes.html*, *protein_diseases.html*, *sequences.html*, *publications.html*. Separating theses sections from the *proteins.html* file does not improve loading time, as these sections are combined back into one before being served and are thus served as a single page. However, splitting these sections into their own files makes the code much more manageable and maintainable for the developer, who is now able to find specific code much faster.

An additional benefit of splitting the HTML into separate files and loading the content on demand is that the JavaScript files required for those sections can be loaded on demand as well, further reducing the amount of data loaded unnecessarily.

Using this architecture, we could avoid the issues encountered when building JMS. Despite being an SPA, HTML files in HUMA are kept at a maintainable size by splitting them into logical chunks and loading times were reduced by loading content as it was needed, rather than on the initial page load.

6.3.2. PV-MSA

An important part of HUMA is the analysis of protein variation at the structure level. As such, a high quality molecular visualizer was required. To this end, PV-MSA was used to display protein structures and alignments on the “Proteins” and “Datasets” pages. PV-MSA was discussed in greater detail in section 3.3.3 and can be downloaded at https://github.com/davidbrownza/PV-MSA.

6.4. Accessing the public database

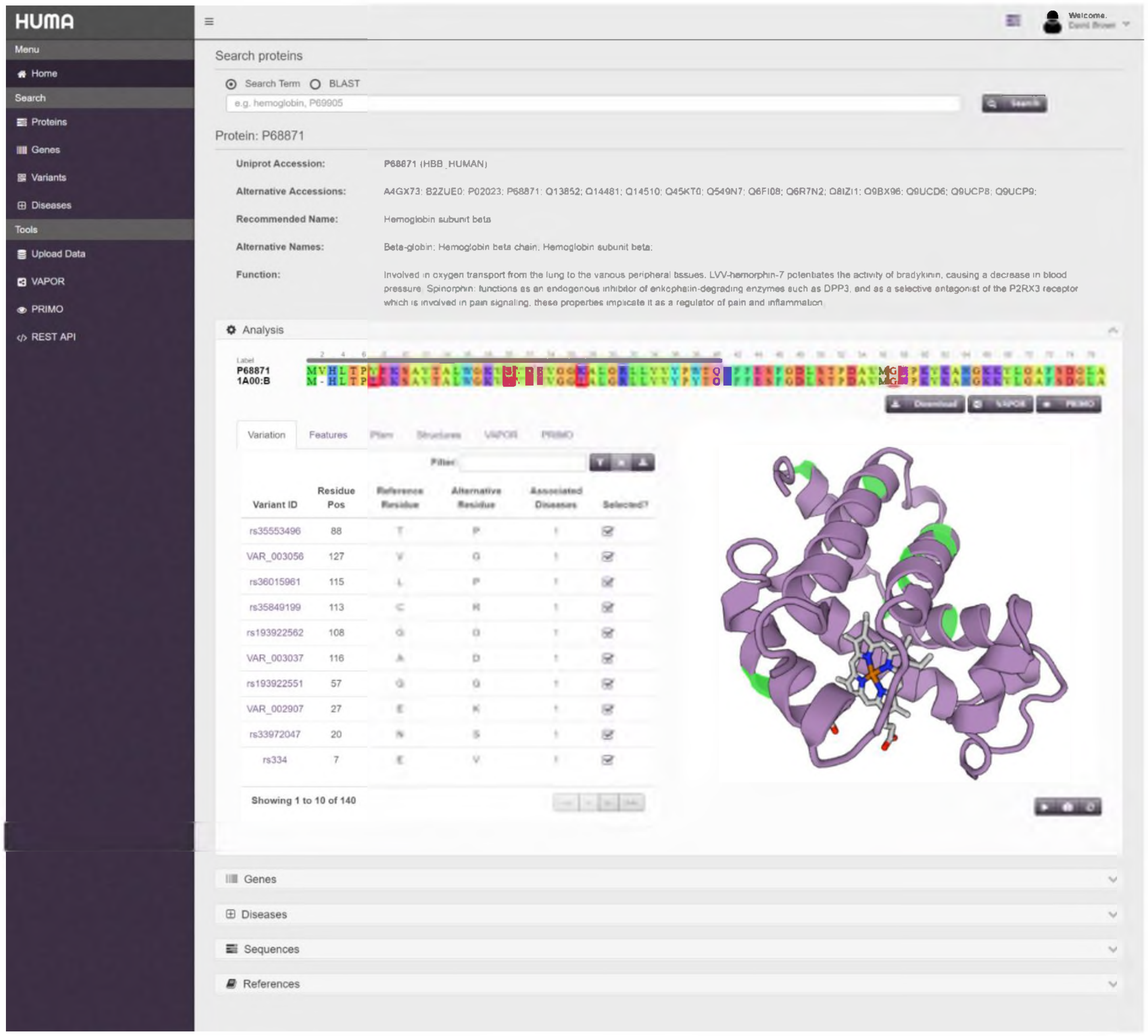
The HUMA interface provides clicked based access to the public database. There are four main data pages designed around the four data categories: proteins; genes; variants; and diseases.

6.4.1. Proteins

The ‘Proteins’ page lets users search for a given protein of interest by entering a Uniprot accession number, the protein name, or by using BLAST to search based on sequence similarity.

Once the protein of interest has been found, the user is presented with the results (Fig. 6.5). These results include the protein name and description, alternative names and symbols, the protein sequence and all available structures for the protein that could be extracted from the PDB, and the supporting sequences (*i.e.* exons, CDS, and cDNA).

Additionally, related data such as the genes that code the protein, diseases that have been associated with the protein, and variants that occur in the protein sequence, are also displayed on



Protein: P68871

Uniprot Accession:	P68871 (HBB_HUMAN)
Alternative Accessions:	A4GX73; B2ZUE0; P02023; P68871; Q13852; Q14481; Q14510; Q45KT0; Q549N7; Q6FI08; Q6R7N2; Q8IZI1; Q9BX96; Q9UCD6; Q9UCP8; Q9UCP9;
Recommended Name:	Hemoglobin subunit beta
Alternative Names:	Beta-globin; Hemoglobin beta chain; Hemoglobin subunit beta;
Function:	Involved in oxygen transport from the lung to the various peripheral tissues. LVV-hemorphin-7 potentiates the activity of bradykinin, causing a decrease in blood pressure. Spinorphin: functions as an endogenous inhibitor of enkephalin-degrading enzymes such as DPP3, and as a selective antagonist of the P2RX3 receptor which is involved in pain signaling, these properties implicate it as a regulator of pain and inflammation.

Variant ID	Residue Pos	Reference Residue	Alternative Residue	Associated Diseases	Selected?
rs35553496	88	T	P	1	✓
VAR_003056	127	V	G	1	✓
rs36015961	115	L	P	1	✓
rs35849199	113	C	R	1	✓
rs193922562	108	G	D	1	✓
VAR_003037	116	A	D	1	✓
rs193922551	57	G	Q	1	✓
VAR_002907	27	E	K	1	✓
rs33972047	20	N	S	1	✓
rs334	7	E	V	1	✓

Showing 1 to 10 of 140

Fig. 6.5. Protein results page – the results page after searching for 'P68871'.

the page. Literature that references the protein is also included on the page. This related data is arranged into five distinct data blocks on the 'Proteins' result page.

6.4.1.1. Analysis block

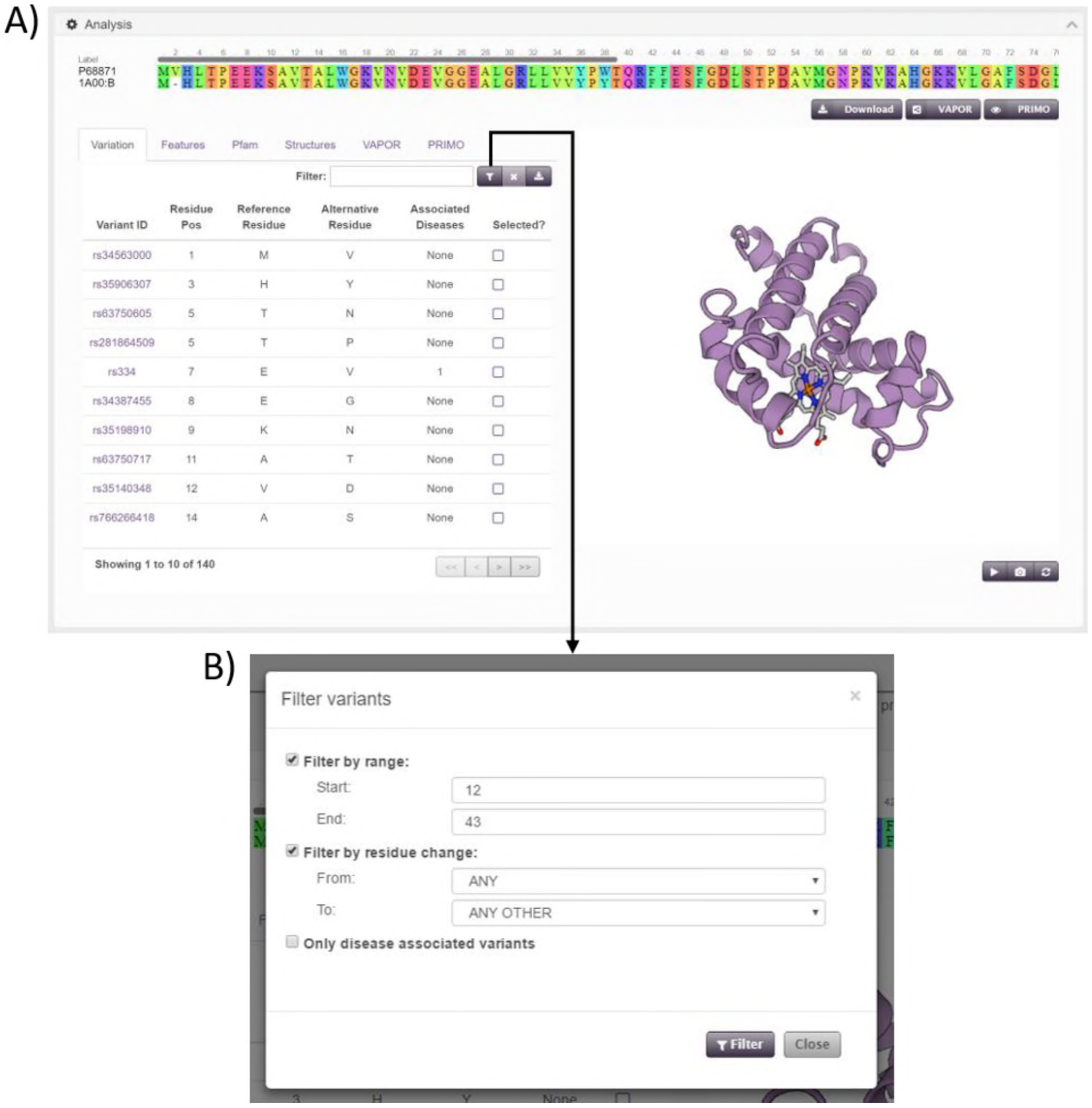
The 'Analysis' block is depicted in Fig. 6.6 A. At the top of the, the protein sequence, aligned to the sequence of the selected protein structure (if one is available), is displayed using PV-MSA. PV-MSA is also used to display the protein structure in the right hand of the graph. As such, the alignment and structure are linked. Thus, selecting a residue in the alignment will select it in the structure and vice versa.

On the left of the protein structure is a tabbed area consisting of three tabs. In the Variants tab, all variants that have been mapped to the protein sequence are tabulated. Selecting a variant will highlight it in the table. Additionally, HUMA provides some useful filtering features to filter out unwanted variants (Fig. 6.6 B). Variants can then be downloaded in various formats including VCF, dbSNP IDs, one-letter representations (*e.g.* A12V), or three-letter representations (*e.g.* ALA12VAL).

The Features tab is like the Variants tab in that it tabulates features such as binding sites, chains, and active sites. Similarly, selecting a feature will highlight its location in the protein structure and sequence.

The Structures tab displays a list of all available PDB structures for a given protein sequence. Selecting on of these structures will swap out the existing structure in the PV-MSA structure and alignment viewers for the newly selected structure.

PRIMO and VAPOR can also be run from this block. Selecting variants in the variants table (by clicking on the checkbox in the "Selected?" column) substitutes them into the sequence. Clicking on the PRIMO button in the top-right of the Analysis block will automatically populate the PRIMO input page (discussed in section 6.6.2) with the mutated sequence, job name and short description.


A)
Analysis
Label
P68871 MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGL
1A00:B M-HLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPYTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGL
Download
VAPOR
PRIMO
Variation
Features
Pfam
Structures
VAPOR
PRIMO
Filter:
B)

Variant ID	Residue Pos	Reference Residue	Alternative Residue	Associated Diseases	Selected?
rs34563000	1	M	V	None	☐
rs35906307	3	H	Y	None	☐
rs63750605	5	T	N	None	☐
rs281864509	5	T	P	None	☐
rs334	7	E	V	1	☐
rs34387455	8	E	G	None	☐
rs35198910	9	K	N	None	☐
rs63750717	11	A	T	None	☐
rs35140348	12	V	D	None	☐
rs766266418	14	A	S	None	☐

Showing 1 to 10 of 140

Filter variants

✓ Filter by range:
Start: 12
End: 43
✓ Filter by residue change:
From: ANY
To: ANY OTHER
☐ Only disease associated variants

Filter Close

*Fig. 6.6. Analysis block – contains visualization tools for alignment and structure that allows the locations of variants Pfam families and domains, and features to be highlighted in the protein sequence and structure. **A)** The Analysis block is the main feature of the Protein result page. **B)** The filter dialog provides useful ways to filter the list of variants mapped to the protein.*

Similarly, clicking the VAPOR button will populate the relevant input controls on the VAPOR input page (discussed in section 6.6.1) with the sequence, selected variants, and job name and description.

Lastly, the Analysis block provides a Download button, which can be used to download the Uniprot sequence, structure sequence, the alignment between the Uniprot and structure sequences, or the PDB file.

6.4.1.2. Genes block

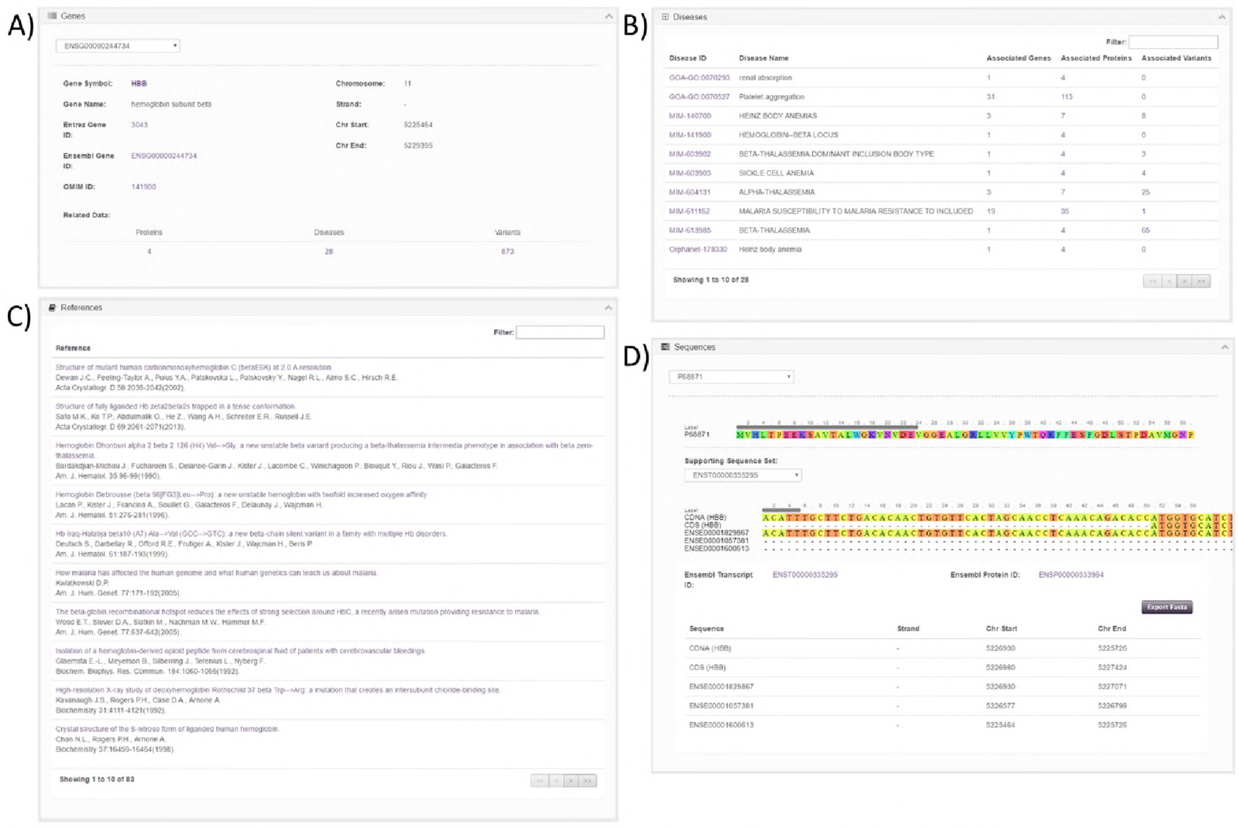
The Genes block (Fig. 6.7 A) provides a dropdown menu that holds the gene identifiers of any gene sequences that are known to code the protein sequence. Selecting one of these identifiers will display some details about that gene and will also provide a link to the 'Genes' page where the gene can be analyzed in further detail.

6.4.1.3. Diseases block

The Diseases block (Fig. 6.7 B) provides a tabulated list of diseases that have been associated with the protein. This table includes links to the 'Diseases' page where each disease can be examined in more detail.

6.4.1.4. References block

The References block (Fig. 6.7 C) displays the tabulated list of literature that has been linked to the protein. Literature that includes a DOI is linked to so that users can browse directly to the paper.



Genes

ENSG00000244734

Gene Symbol:	HBB	Chromosome:	11
Gene Name:	hemoglobin subunit beta	Strand:	-
Entrez Gene ID:	3043	Chr Start:	5225464
Ensembl Gene ID:	ENSG00000244734	Chr End:	5229395
OMIM ID:	141900		

Related Data:

Proteins	Diseases	Variants
4	28	873

Diseases

Filter:

Disease ID	Disease Name	Associated Genes	Associated Proteins	Associated Variants
GOA-GO:0070293	renal absorption	1	4	0
GOA-GO:0070527	Platelet aggregation	31	113	0
MIM-140700	HEINZ BODY ANEMIAS	3	7	8
MIM-141900	HEMOGLOBIN--BETA LOCUS	1	4	0
MIM-603902	BETA-THALASSEMIA DOMINANT INCLUSION BODY TYPE	1	4	3
MIM-603903	SICKLE CELL ANEMIA	1	4	4
MIM-604131	ALPHA-THALASSEMIA	3	7	25
MIM-611162	MALARIA SUSCEPTIBILITY TO MALARIA RESISTANCE TO INCLUDED	19	35	1
MIM-613985	BETA-THALASSEMIA	1	4	65
Orphanet-178330	Heinz body anemia	1	4	0

Showing 1 to 10 of 28

References

Filter:

Reference
Structure of mutant human carbonmonoxyhemoglobin C (betaE6K) at 2.0 A resolution. Dewan J.C., Feeling-Taylor A., Puius Y.A., Patskovska L., Patskovsky Y., Nagel R.L., Almo S.C., Hirsch R.E. Acta Crystallogr. D 58:2038-2042(2002).
Structure of fully liganded Hb zeta2beta2s trapped in a tense conformation. Safo M.K., Ko T.P., Abdulmalik O., He Z., Wang A.H., Schreiter E.R., Russell J.E. Acta Crystallogr. D 69:2061-2071(2013).
Hemoglobin Dhonburi alpha 2 beta 2 126 (H4) Val-->Gly: a new unstable beta variant producing a beta-thalassemia intermedia phenotype in association with beta zero-thalassemia. Bardakdjian-Michau J., Fucharoen S., Delanoe-Garin J., Kister J., Lacombe C., Winichagoon P., Blouquit Y., Riou J., Wasi P., Galacteros F. Am. J. Hematol. 35:96-99(1990).
Hemoglobin Debrousse (beta 96[FG3]Leu-->Pro): a new unstable hemoglobin with twofold increased oxygen affinity. Lacan P., Kister J., Francina A., Souillet G., Galacteros F., Delaunay J., Wajcman H. Am. J. Hematol. 51:276-281(1996).
Hb Iraq-Halabja beta10 (A7) Ala-->Val (GCC-->GTC): a new beta-chain silent variant in a family with multiple Hb disorders. Deutsch S., Darbellay R., Offord R.E., Frutiger A., Kister J., Wajcman H., Beris P. Am. J. Hematol. 61:187-193(1999).
How malaria has affected the human genome and what human genetics can teach us about malaria. Kwiatkowski D.P. Am. J. Hum. Genet. 77:171-192(2005).
The beta-globin recombinational hotspot reduces the effects of strong selection around HbC, a recently arisen mutation providing resistance to malaria. Wood E.T., Stover D.A., Slatkin M., Nachman M.W., Hammer M.F. Am. J. Hum. Genet. 77:637-642(2005).
Isolation of a hemoglobin-derived opioid peptide from cerebrospinal fluid of patients with cerebrovascular bleedings. Glaemsta E.-L., Meyerson B., Silberring J., Terenius L., Nyberg F. Biochem. Biophys. Res. Commun. 184:1060-1066(1992).
High-resolution X-ray study of deoxyhemoglobin Rothschild 37 beta Trp-->Arg: a mutation that creates an intersubunit chloride-binding site. Kavanaugh J.S., Rogers P.H., Case D.A., Arnone A. Biochemistry 31:4111-4121(1992).
Crystal structure of the S-nitroso form of liganded human hemoglobin. Chan N.L., Rogers P.H., Arnone A. Biochemistry 37:16459-16464(1998).

Showing 1 to 10 of 83

Sequences

P68871

```
Label
P68871           MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNP
```

Supporting Sequence Set:

ENST00000335295

```
Label
CDNA (HBB)       ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATC
CDS (HBB)        ---------------------------------------------------ATGGTGCATC
ENSE00001829867  ACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAGACACCATGGTGCATC
ENSE00001057381  ............................................................
ENSE00001600613  ............................................................
```

Ensembl Transcript ID:	ENST00000335295	Ensembl Protein ID:	ENSP00000333994

Export Fasta

Sequence	Strand	Chr Start	Chr End
CDNA (HBB)	-	5226930	5225726
CDS (HBB)	-	5226980	5227424
ENSE00001829867	-	5226930	5227071
ENSE00001057381	-	5226577	5226799
ENSE00001600613	-	5225464	5225726

Fig. 6.7. *Protein data blocks –* ***A)*** *Genes block;* ***B)*** *Diseases block;* ***C)*** *References block;* ***D)*** *Sequences block*

6.4.1.5. Sequences block

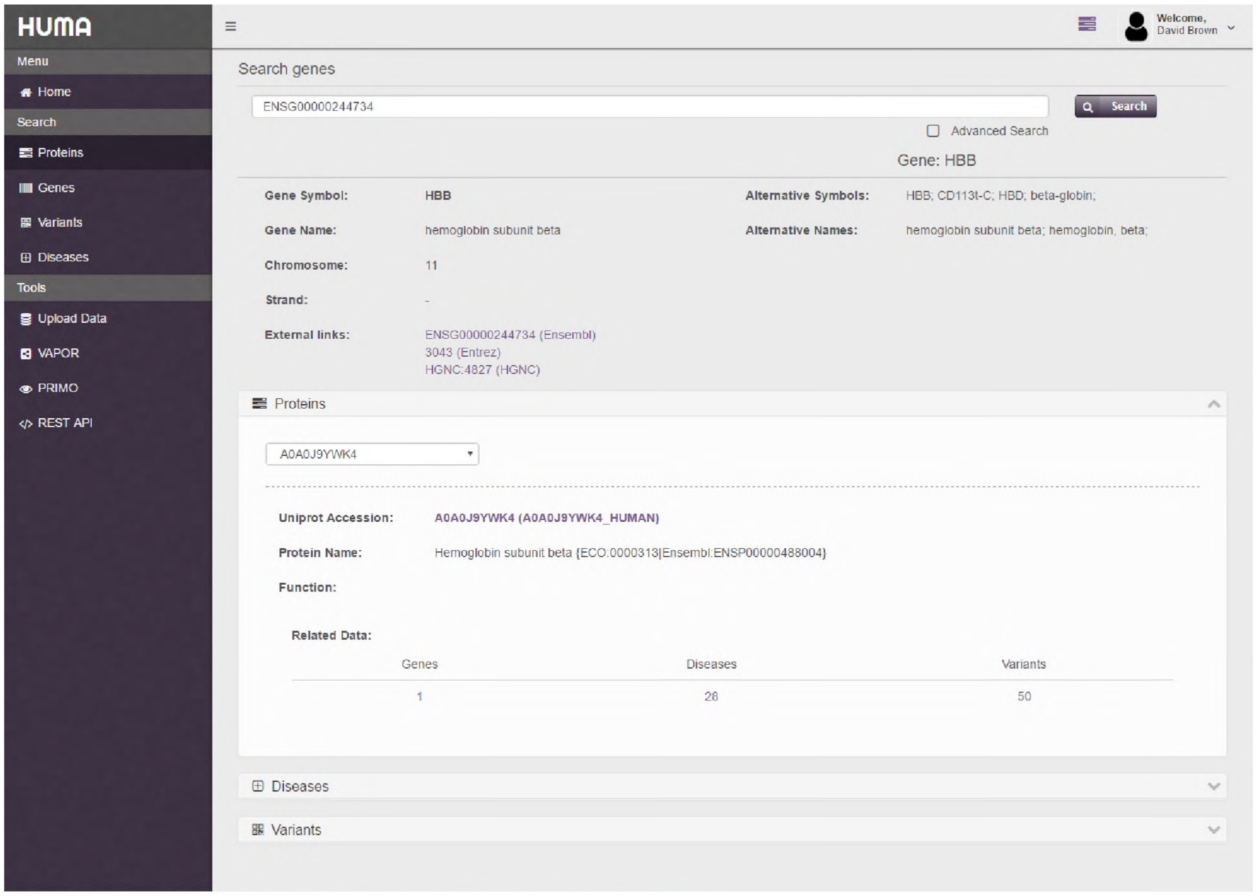
The Sequences block makes use of PV-MSA to display the aligned supporting sequences for the protein (Fig. 6.7 D). This includes the cDNA, CDS, and exons, all of which can be downloaded in Fasta format. If multiple isoforms are stored for a given protein, these different isoforms can be selected here.

6.4.2. Genes

The 'Genes' page (Fig. 6.7) follows a similar design to the 'Proteins' page *i.e.* gene details are displayed at the top of the result page and related data is displayed via three data blocks.

6.4.2.1. Proteins block

The Proteins block has a similar design to the Genes block described in section 6.4.1.2. Proteins that are coded by the gene can be selected via the dropdown menu at the top of the block. Selecting



Search genes

ENSG00000244734

Gene: HBB

Gene Symbol:	HBB	Alternative Symbols:	HBB; CD113t-C; HBD; beta-globin;
Gene Name:	hemoglobin subunit beta	Alternative Names:	hemoglobin subunit beta; hemoglobin, beta;
Chromosome:	11		
Strand:	-		
External links:	ENSG00000244734 (Ensembl) 3043 (Entrez) HGNC:4827 (HGNC)		

Proteins

A0A0J9YWK4

Uniprot Accession:	A0A0J9YWK4 (A0A0J9YWK4_HUMAN)
Protein Name:	Hemoglobin subunit beta {ECO:0000313|Ensembl:ENSP00000488004}
Function:	

Related Data:

Genes	Diseases	Variants
1	28	50

Diseases

Variants

Fig. 6.8. Gene results page – the results page after searching for 'HBB'.

a protein from this dropdown menu will display the protein details as well as provide a link back to the 'Proteins' page.

6.4.2.2. Diseases block

The Diseases block on the 'Genes' page follows the exact same design as the one described for the 'Proteins' page. See section 6.4.1.3 for details.

6.4.2.3. Variants block

The Variants block is the final data block on the 'Genes' result page and displays a tabulated list of variants that have been mapped to the gene sequence. As with variants mapped to a protein sequence, this list can be filtered and downloaded in various formats.

6.4.3. Variants

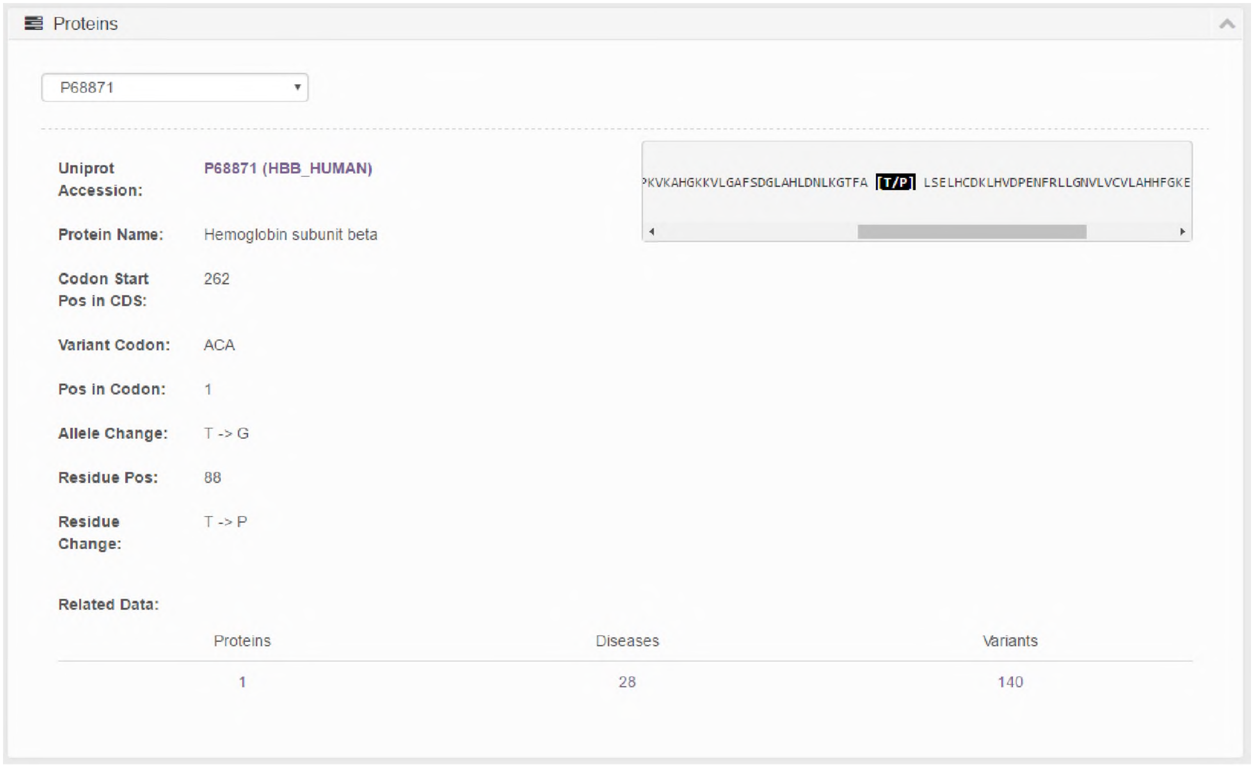
The ‘Variants’ page is the third data page and, similarly, consists of a Genes block, Proteins block, and Diseases block. Variant details displayed at the top of the page include the

6.4.3.1. Genes block

The Genes block follows the same design as the one described for the ‘Proteins’ page. See section 6.4.1.2 for details. One diverging point is that on the ‘Variants’ page, the Genes block also includes the position of the variant in the gene and the allele change.

6.4.3.2. Proteins block

The Proteins block on the ‘Variants’ page is unique (Fig. 6.9). It is designed specifically around the variant’s effect on the protein. It shows details around the codon that the variant occurs in, such as the start position of the codon in the CDS, the position of the variant in the codon, and the



Proteins

P68871

Uniprot Accession:	P68871 (HBB_HUMAN)
Protein Name:	Hemoglobin subunit beta
Codon Start Pos in CDS:	262
Variant Codon:	ACA
Pos in Codon:	1
Allele Change:	T -> G
Residue Pos:	88
Residue Change:	T -> P

PKVKAHGKKVLGAFSDGLAHLDNLKGTFA [T/P] LSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKE

Related Data:

Proteins	Diseases	Variants
1	28	140

Fig. 6.9. Proteins block on Variant result page – displays all the details around the variants position in the protein sequence.

mutated sequence of the codon. Additionally, the position of the residue, as well as the residue change caused by the variant, are displayed in this block. Lastly, the protein sequence is also displayed with the variant highlighted.

6.4.3.3. Diseases block

The Diseases block is designed in a similar manner to the Genes block on the 'Proteins' page (Fig. 6.7. A). A dropdown menu is provided at the top of the block that contains the names of any of the diseases linked to the given variant. Selecting the disease results in the disease details being displayed beneath. This contrasts with the Diseases blocks on the 'Proteins' and 'Genes' pages, which display related diseases in a tabulated list. The reason for using the tabulated list on these pages is that there can be a large number of diseases associated with a given gene or protein. On the other hand, there are seldom more than one or two diseases associated with a given variant. The tabulated list provides a better way of displaying a large number of diseases in a small space, however, when there are only a few diseases, providing a dropdown menu to display a single disease at a time allows us to provide more details about the selected disease.

6.4.4. Diseases

The final data page is the 'Diseases' page. From this page, the database can be searched for a disease of interest. Once found, the disease details are displayed as well as details about related genes, proteins, and associated variants.

6.4.4.1. Genes block

The Genes block the same design as discussed in section 6.4.1.2. The dropdown menu is, however, populated with genes that have been associated with the disease of interest.

6.4.4.2. Proteins block

The Proteins block follows the same design as discussed in section 6.4.2.1. As described for the Genes block, the dropdown menu is populated with proteins that have been associated with the disease of interest.

6.4.4.3. Variants block

Lastly, the Variants block displays a tabulated list of all variants that have been associated with a disease of interest. This block is identical to the Variants block on the 'Genes' page and provides options to filter and download variants.

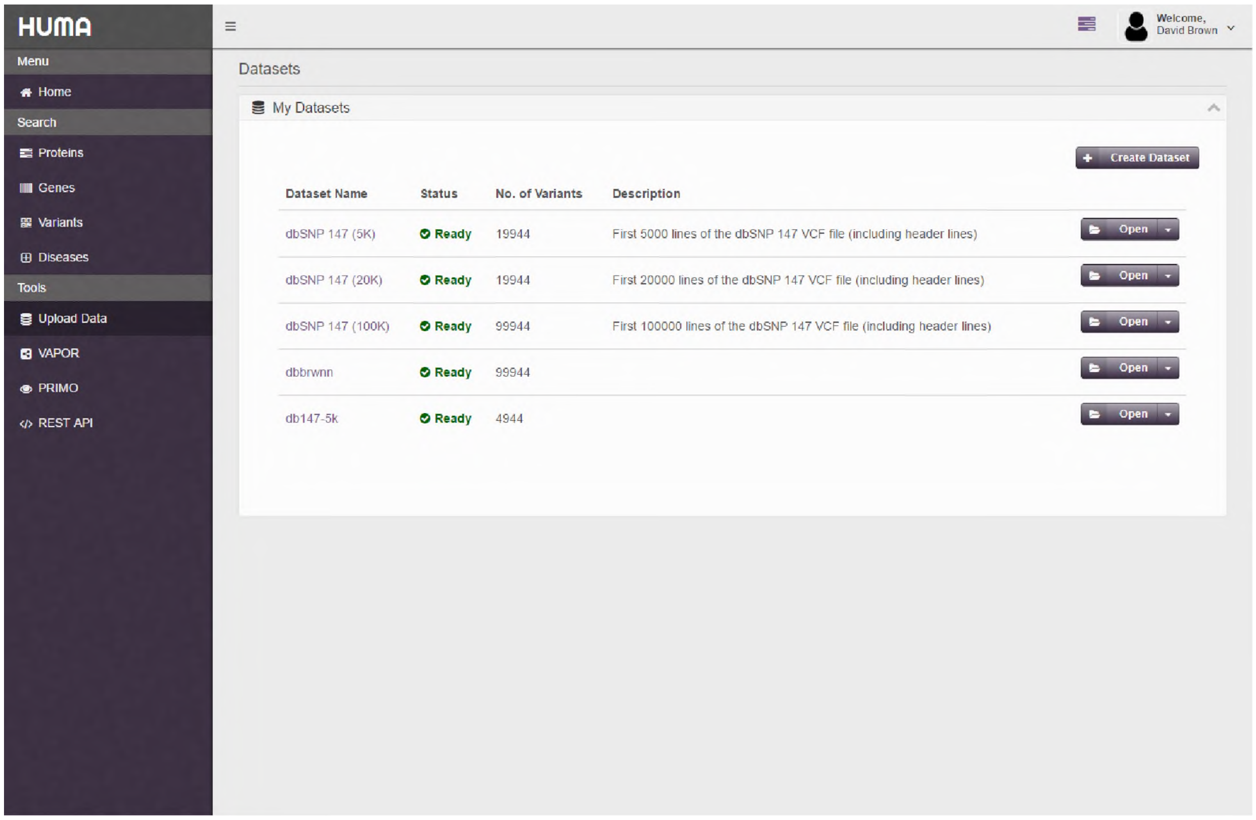
6.5. Private datasets

HUMA lets users upload their own variant datasets via the "Datasets" page (Fig. 6.10), which can be reached by clicking on the "Upload Data" button under the "Tools" section of the side bar. On the "Datasets" page, users can create, edit (*e.g.* rename the dataset or add new data to it), delete and share custom datasets.

Once created, datasets are presented in a table with columns that display the dataset name, status, number of variants in the dataset, and a description for the dataset. The final column in the table provides options to open a dataset, as well as edit, delete and share the dataset. Opening the dataset takes the user to a detailed view of the dataset, which we will refer to as the "Dataset Detail" page. This page consists of four "views".

6.5.1. Variants view

The Variants view displays a basic view of all the variants in the dataset. Variants are presented in a table (which uses server-side paging due to the potentially large number of variants in a dataset)



Datasets

My Datasets

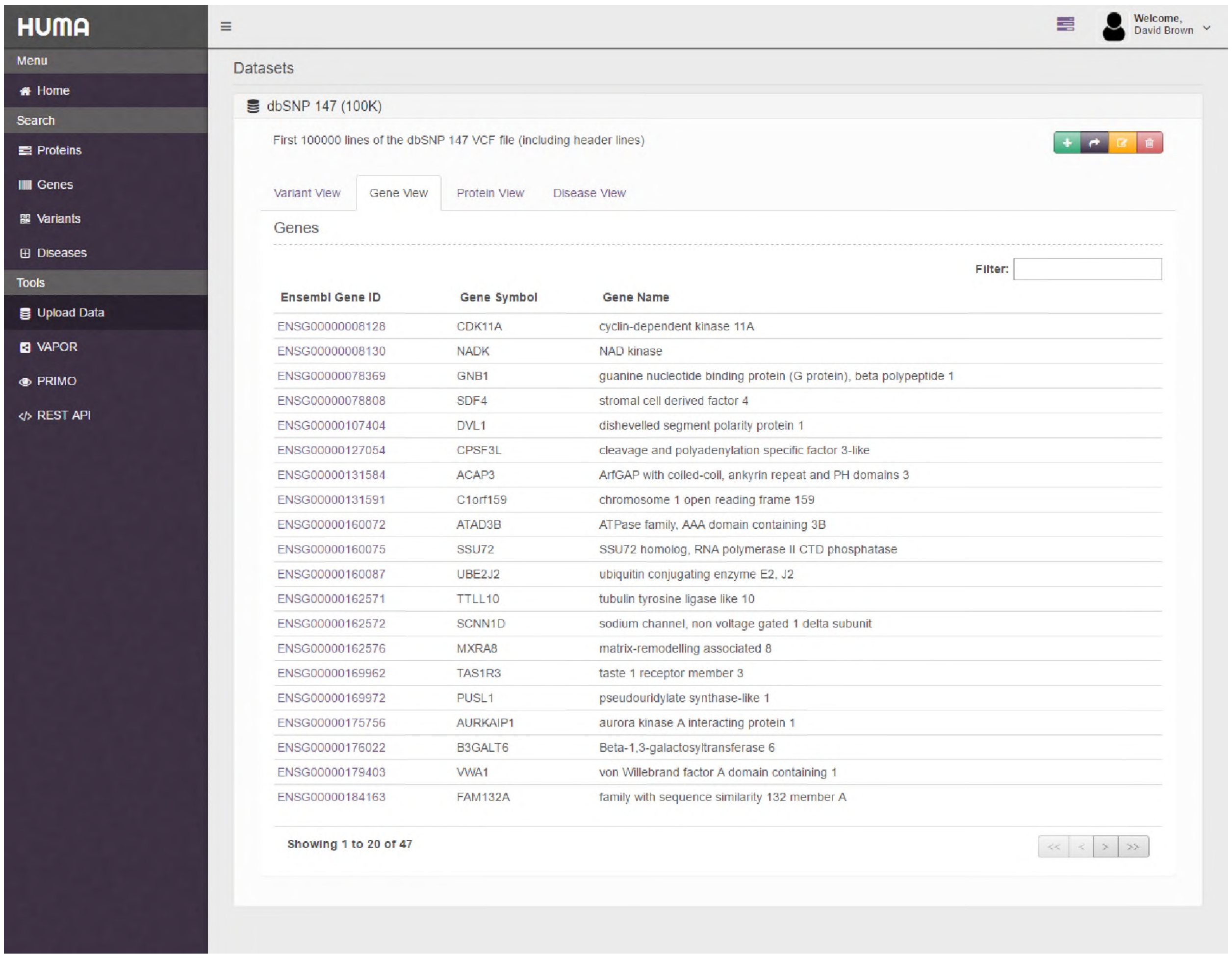
Dataset Name	Status	No. of Variants	Description
dbSNP 147 (5K)	Ready	19944	First 5000 lines of the dbSNP 147 VCF file (including header lines)
dbSNP 147 (20K)	Ready	19944	First 20000 lines of the dbSNP 147 VCF file (including header lines)
dbSNP 147 (100K)	Ready	99944	First 100000 lines of the dbSNP 147 VCF file (including header lines)
dbbrwnn	Ready	99944	
db147-5k	Ready	4944	

Fig. 6.10. *Datasets page – a list of all the datasets owned by the logged in user with options to create, edit, share, delete, and open datasets.*

that displays the variant ID, chromosome co-ordinates, allele change, and provides an option to remove the variant from the dataset.

6.5.2. Genes view

The Genes view (Fig. 6.11) provides a table with all genes that variants in the dataset have been mapped to *i.e.* if a gene is included here, it means that there is at least one variant in the dataset that has been mapped to this gene. Selecting a gene in this table takes the user to a detailed view of that gene, where all the variants that map to the gene are displayed in a table. This table also includes the position that the variant occurs in the gene.



dbSNP 147 (100K)

First 100000 lines of the dbSNP 147 VCF file (including header lines)

Genes

Ensembl Gene ID	Gene Symbol	Gene Name
ENSG00000008128	CDK11A	cyclin-dependent kinase 11A
ENSG00000008130	NADK	NAD kinase
ENSG00000078369	GNB1	guanine nucleotide binding protein (G protein), beta polypeptide 1
ENSG00000078808	SDF4	stromal cell derived factor 4
ENSG00000107404	DVL1	dishevelled segment polarity protein 1
ENSG00000127054	CPSF3L	cleavage and polyadenylation specific factor 3-like
ENSG00000131584	ACAP3	ArfGAP with coiled-coil, ankyrin repeat and PH domains 3
ENSG00000131591	C1orf159	chromosome 1 open reading frame 159
ENSG00000160072	ATAD3B	ATPase family, AAA domain containing 3B
ENSG00000160075	SSU72	SSU72 homolog, RNA polymerase II CTD phosphatase
ENSG00000160087	UBE2J2	ubiquitin conjugating enzyme E2, J2
ENSG00000162571	TTLL10	tubulin tyrosine ligase like 10
ENSG00000162572	SCNN1D	sodium channel, non voltage gated 1 delta subunit
ENSG00000162576	MXRA8	matrix-remodelling associated 8
ENSG00000169962	TAS1R3	taste 1 receptor member 3
ENSG00000169972	PUSL1	pseudouridylate synthase-like 1
ENSG00000175756	AURKAIP1	aurora kinase A interacting protein 1
ENSG00000176022	B3GALT6	Beta-1,3-galactosyltransferase 6
ENSG00000179403	VWA1	von Willebrand factor A domain containing 1
ENSG00000184163	FAM132A	family with sequence similarity 132 member A

Showing 1 to 20 of 47

Fig. 6.11. *Genes view – a list of all the genes that variants in this dataset have mapped to. Selecting a gene will take the user to a detailed view of the gene where they can see which variants mapped to it.*

6.5.3. Proteins view

Like the Genes view, the Proteins view displays a table containing the proteins that variants in the dataset have mapped to. Selecting a protein will direct the user to a detailed view of the protein in the context of the dataset (like the Analysis block in Fig. 6.6). With the initial focus of HUMA being on protein and structure level analysis, the detailed protein view is far more fleshed out than the others. As with the "Proteins" data page, the structure of the protein is rendered on the right-hand side of the page with the alignment between the Uniprot sequence and the structure sequence being displayed above. On the left of the rendered structure, the variants from the dataset that have mapped to the protein are displayed in the table.

VAPOR and PRIMO can be run from the detailed protein view to analyze dataset variants. As describe for the “Proteins” result page (section 6.4.1.1), variants can be selected and the VAPOR and PRIMO buttons can be clicked to populate the relevant pages.

6.5.4. Diseases view

The Disease view follows the same design as the Genes view. Diseases that are associated with variants in the dataset are listed in a table. Selecting a disease will display which variants have been mapped to that disease.

6.6. Integrating tools via JMS

Server-side tools, PRIMO and VAPOR, have been integrated into HUMA via JMS (Fig. 6.12). As previously discussed, the scripts that make up the respective workflows have been added to JMS as individual tools. These tools can then be accessed via the JMS web API.

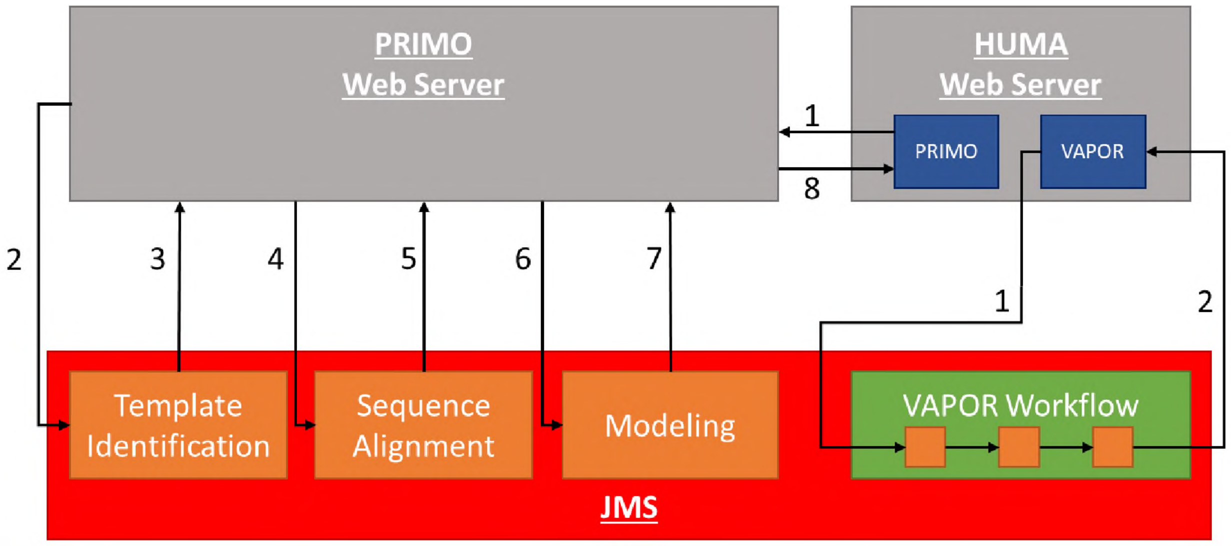
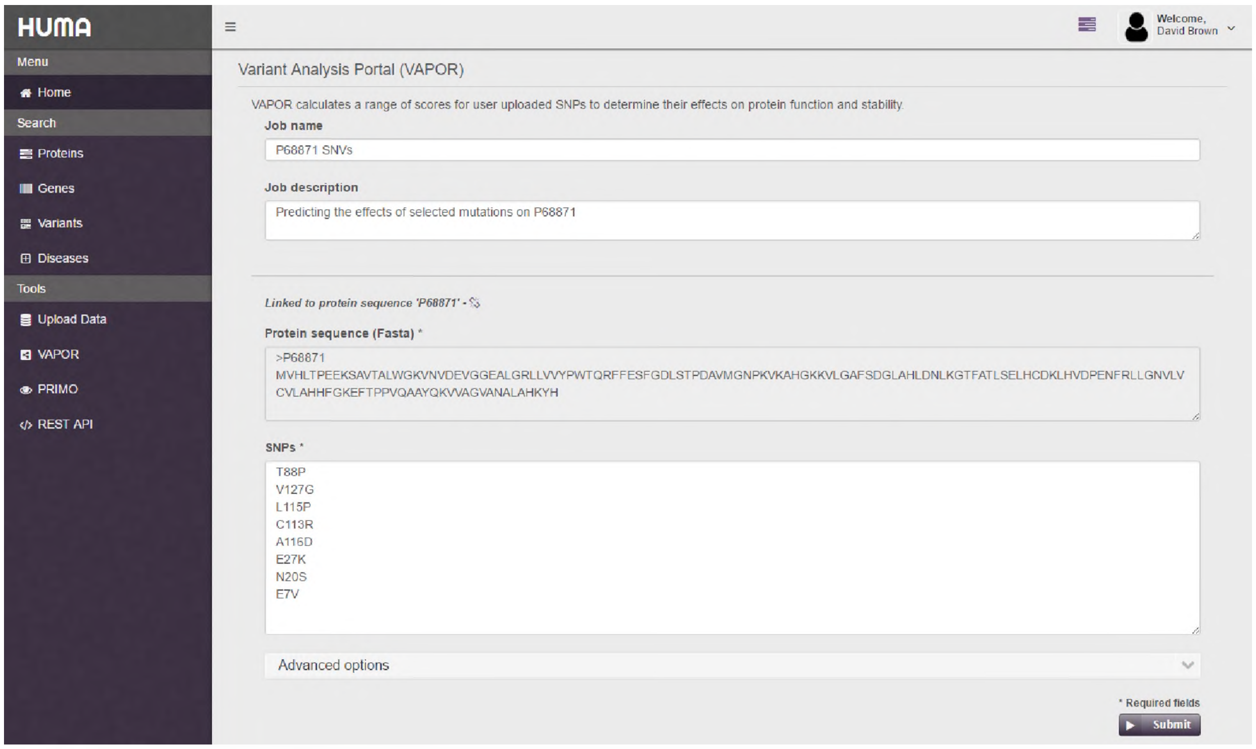


Fig. 6.12. Integrating PRIMO and VAPOR – diagram depicting how PRIMO and VAPOR have been integrated into HUMA. Numbered lines indicate inter-webserver communication. Since VAPOR is executed as a workflow within JMS, VAPOR integration into HUMA simply require HUMA to send the input to the workflow and receive the output. Since PRIMO requires user intervention between each stage, there is far more communication going back and forth between the PRIMO web server and JMS.



Variant Analysis Portal (VAPOR)

VAPOR calculates a range of scores for user uploaded SNPs to determine their effects on protein function and stability.

Job name

P68871 SNVs

Job description

Predicting the effects of selected mutations on P68871

Linked to protein sequence 'P68871' -

Protein sequence (Fasta) *

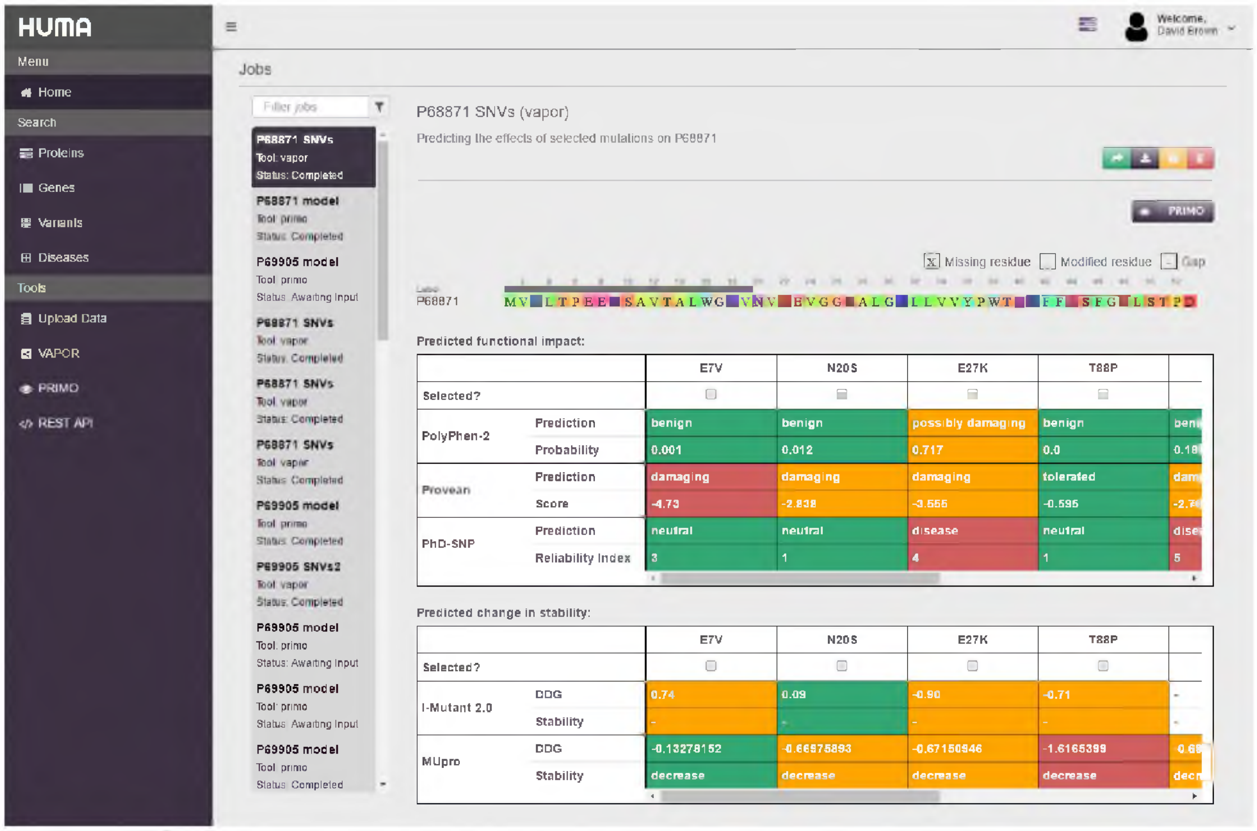
```
>P68871
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLV
CVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
```

SNPs *

```
T88P
V127G
L115P
C113R
A115D
E27K
N20S
E7V
```

* Required fields

Fig. 6.13. VAPOR input page – VAPOR requires users to input a protein sequence and a list of variants.



Jobs

P68871 SNVs
Tool: vapor
Status: Completed

P68871 model
Tool: primo
Status: Completed

P69905 model
Tool: primo
Status: Awaiting Input

P68871 SNVs
Tool: vapor
Status: Completed

P68871 SNVs
Tool: vapor
Status: Completed

P68871 SNVs
Tool: vapor
Status: Completed

P69905 model
Tool: primo
Status: Completed

P69905 SNVs2
Tool: vapor
Status: Completed

P69905 model
Tool: primo
Status: Awaiting Input

P69905 model
Tool: primo
Status: Awaiting Input

P69905 model
Tool: primo
Status: Completed

P68871 SNVs (vapor)

Predicting the effects of selected mutations on P68871

Predicted functional impact:

		E7V	N20S	E27K	T88P	
Selected?		☐	☐	☐	☐	
PolyPhen-2	Prediction	benign	benign	possibly damaging	benign	beni
	Probability	0.001	0.012	0.717	0.0	0.18
Provean	Prediction	damaging	damaging	damaging	tolerated	dam
	Score	-4.73	-2.838	-3.555	-0.595	-2.7
PhD-SNP	Prediction	neutral	neutral	disease	neutral	dise
	Reliability Index	3	1	4	1	5

Predicted change in stability:

		E7V	N20S	E27K	T88P	
Selected?		☐	☐	☐	☐	
I-Mutant 2.0	DDG	0.74	0.09	-0.90	-0.71	-
	Stability	-	-	-	-	-
MUpro	DDG	-0.13278152	-0.66975893	-0.67150946	-1.6165399	-0.6
	Stability	decrease	decrease	decrease	decrease	decr

Fig. 6.14. VAPOR results page – results from the VAPOR workflow are separated into two tables. The top table consists of predictions made by tools that predict the function impact of variants on a protein. The bottom table includes tools that predict the changes in stability caused by variants in a protein sequence.

6.6.1. VAPOR

JMS provides an API call that lets users execute a workflow given the workflow version ID and a set of input values. As described in section 3.4, VAPOR is implemented in JMS as a workflow consisting of 8 individual tools.

To execute the VAPOR workflow via HUMA, HUMA provides a custom interface where users can input a sequence, variants, and some additional advanced options (Fig. 6.13). On submission, these inputs are sent to the HUMA web server, which compiles them into a JSON data structure that is compatible with the JMS API before forwarding the request to JMS.

JMS then uses these details to execute the VAPOR workflow. Once the workflow has run to completion, JMS sends a notification to the HUMA web server, which contains the status of the finished job. If the workflow executed successfully, HUMA fetches the result file (a tab-delimited file of the predictions from each program in the workflow) from JMS and displays it on the job results page (Fig. 6.14).

To submit jobs to JMS, a JMS account was set up for HUMA by creating an account on the underlying Linux host. The login credentials were then added to the HUMA configuration file, where they could be accessed whenever communication with JMS was required.

6.6.2. PRIMO

The process for integrating PRIMO into HUMA is slightly different. Although PRIMO tools are also hosted in JMS, HUMA doesn't interact directly with JMS to run them. Instead, it links to PRIMO to execute the tools. This negates the need to manage the logic of running the PRIMO workflow in two different places and means that HUMA can benefit from updates to the PRIMO interface.

In its current form, the HUMA interface is only able to submit PRIMO jobs and monitor the status of the running job. Any further interaction with the job must be done via the PRIMO interface. After submitting a PRIMO job, HUMA redirects to the job page where the user can monitor the status of the job and view the parameters used to run the job. This page also links to the job in PRIMO, so that users can quickly navigate to the PRIMO interface to continue the modeling run.

Requiring the user to jump between PRIMO and HUMA to finish the modeling run is not an ideal experience. In future update, PRIMO will be more deeply integrated into HUMA so that the entire process can be managed from the HUMA interface.

Additionally, for users to submit jobs to PRIMO via HUMA, they must first link their accounts to an existing PRIMO account. If they haven't used PRIMO before, this means that they must first create an account via the PRIMO web server before coming back to HUMA and linking that account to their HUMA account. Although the linking process in HUMA has been designed to be as simple as possible, we plan to remove this requirement in future, either by using Single Sign-On (SSO) across all our web servers, or simply by using a centralized user database so that the same accounts can be used across each web server.

6.7. Collaboration

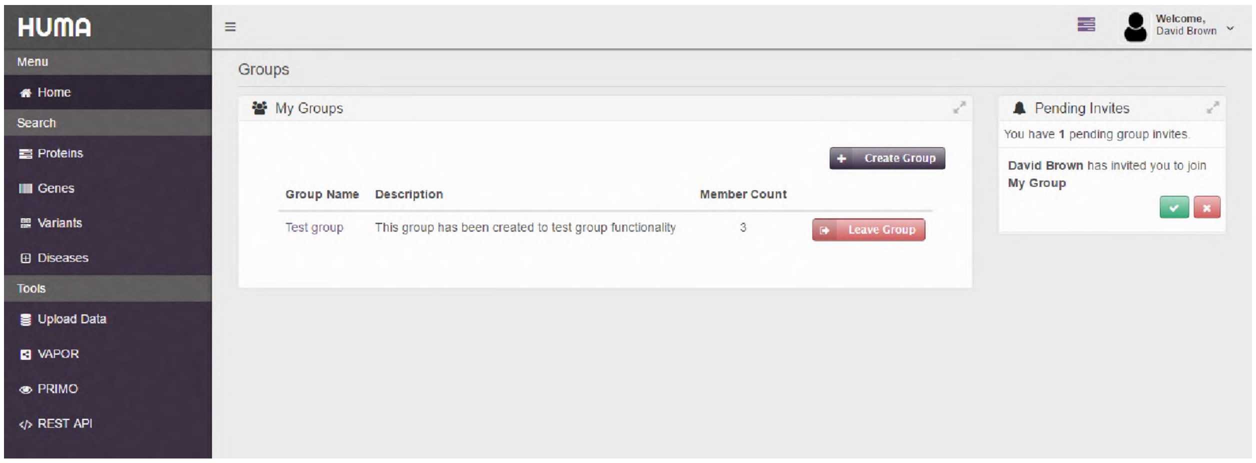
HUMA was built with the goal of enabling researchers, specifically African researchers that are part of the H3Africa consortium, to analyze their data and collaborate with one another. So far, we have discussed how HUMA can be used to analyze variation data *i.e.* through uploading VCF datasets, mapping them to public data, visualizing variants in the protein structure, and analyzing variation using PRIMO and VAPOR. Here, we will discuss how HUMA can be used to facilitate collaboration.

To enable communication, collaboration, and sharing of data, a simplistic social networking feature, groups, has been built into HUMA. The "group" concept in HUMA lets a user create a group and invite other users (*e.g.* their collaborators) into the group. They can then share datasets, analysis results, and hold discussions within the group.

6.7.1. Creating groups & inviting collaborators

A group can be created from the "Groups" page, accessed via the dropdown menu in the top-righthand corner of the HUMA interface, by clicking the "Create group" button. The user will be presented with a dialog requesting a name for the group. Creating the group is as simple as entering in a name and clicking the "Save" button. The group creator will automatically be added to the newly created group as an administrator and cannot leave the group without first assigning administrator privileges to another user.

Invitations inviting other users to join the group can be sent from within the group by clicking the "Invite User" button under the "Members" tab and entering in a semi-colon delimited list of the e-mail addresses of the users to invite. Invited users can accept or decline the invites via the "Groups"



Group Name	Description	Member Count
Test group	This group has been created to test group functionality	3

You have 1 pending group invites.

David Brown has invited you to join My Group

Fig. 6.15. Groups page – users are able to create new groups, enter groups they already belong to, and accept invites to new groups (box on the right) from this page.

page (Fig. 6.15). Currently, users must already have created an account and must log in to accept the invite. In future, users will be able to accept invites via a link sent in an e-mail.

6.7.2. Sharing datasets

Once a user is in a group, they can share their datasets with the group. As mentioned previously, on the “Datasets” page, users are provided with a button to share a given dataset. Clicking on this

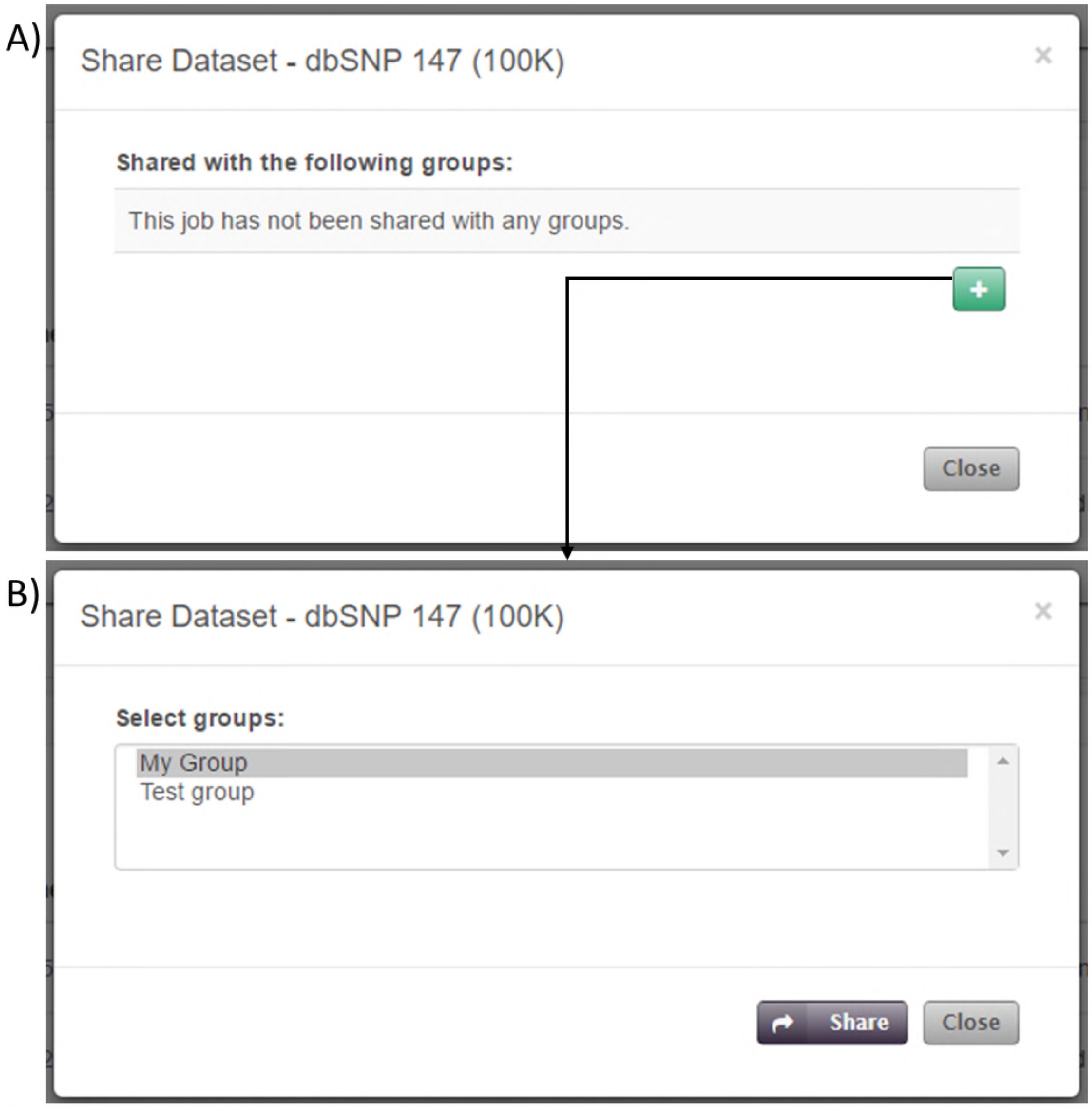

A)
Share Dataset - dbSNP 147 (100K)
×
Shared with the following groups:
This job has not been shared with any groups.
+
Close
B)
Share Dataset - dbSNP 147 (100K)
×
Select groups:
My Group
Test group
Share
Close

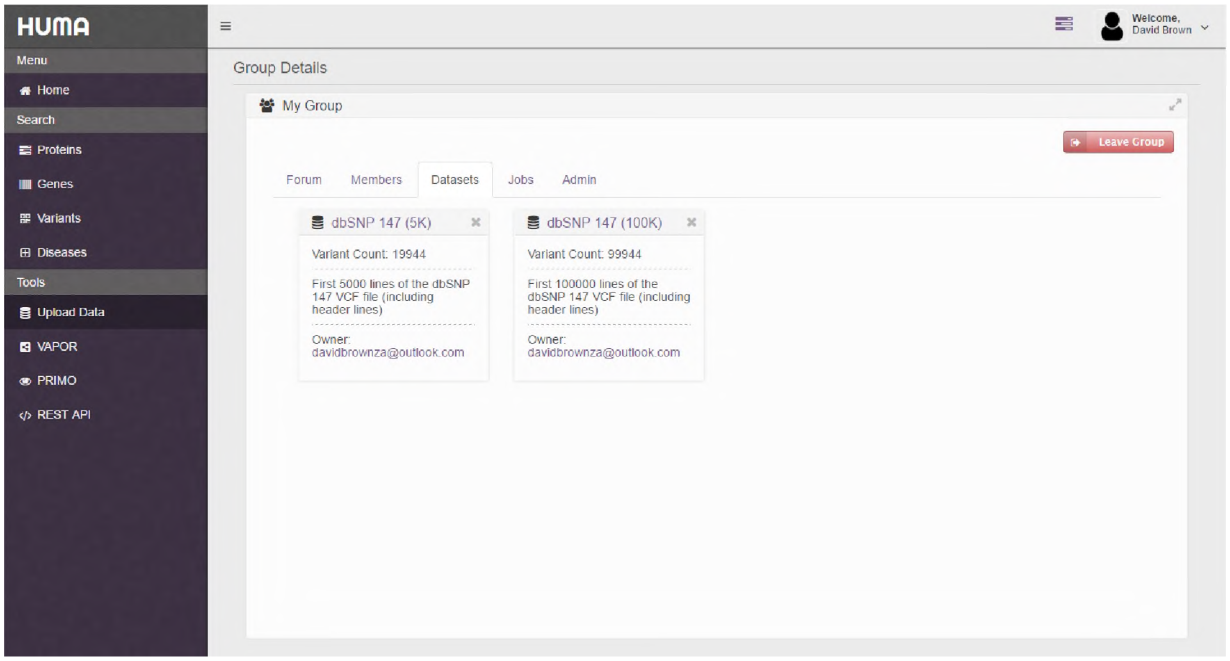
*Fig. 6.16. Sharing a dataset – HUMA provides a simlple to use sharing dialog. **A)** Clicking the “share” button on the dataset page brings up a dialog that lists all the groups that the dataset has been shared with. **B)** Clicking the “+” button shows a dialog where a user can select the groups they would like to share the dataset with.*

button will bring up a dialog with a table displaying the groups that the dataset has already been shared with (Fig. 6.16 A). In the last row of this table, the user is provided with a button to add new groups to the "shared" table. Clicking on this button brings up a list of all the groups that the user belongs to and which the given dataset has not already been shared with (Fig. 6.16 B). From here, the user can simply select the additional groups they want to share the data with and click on the "Share" button.

After a dataset has been shared with a group, it will appear under the "Datasets" tab of the given group (Fig. 6.17). Any user belonging to the group will now be able to access the dataset via this page.

6.7.3. Sharing job results

Jobs are shared with a group in an almost identical manner as datasets. From the job results page, a user can click on the "Share" button (also found by right-clicking on the job in the job list). They will then be presented with the same interface described in Fig. 6.16, but tailored towards jobs.



dbSNP 147 (5K)
Variant Count: 19944
First 5000 lines of the dbSNP 147 VCF file (including header lines)
Owner: davidbrownza@outlook.com

dbSNP 147 (100K)
Variant Count: 99944
First 100000 lines of the dbSNP 147 VCF file (including header lines)
Owner: davidbrownza@outlook.com

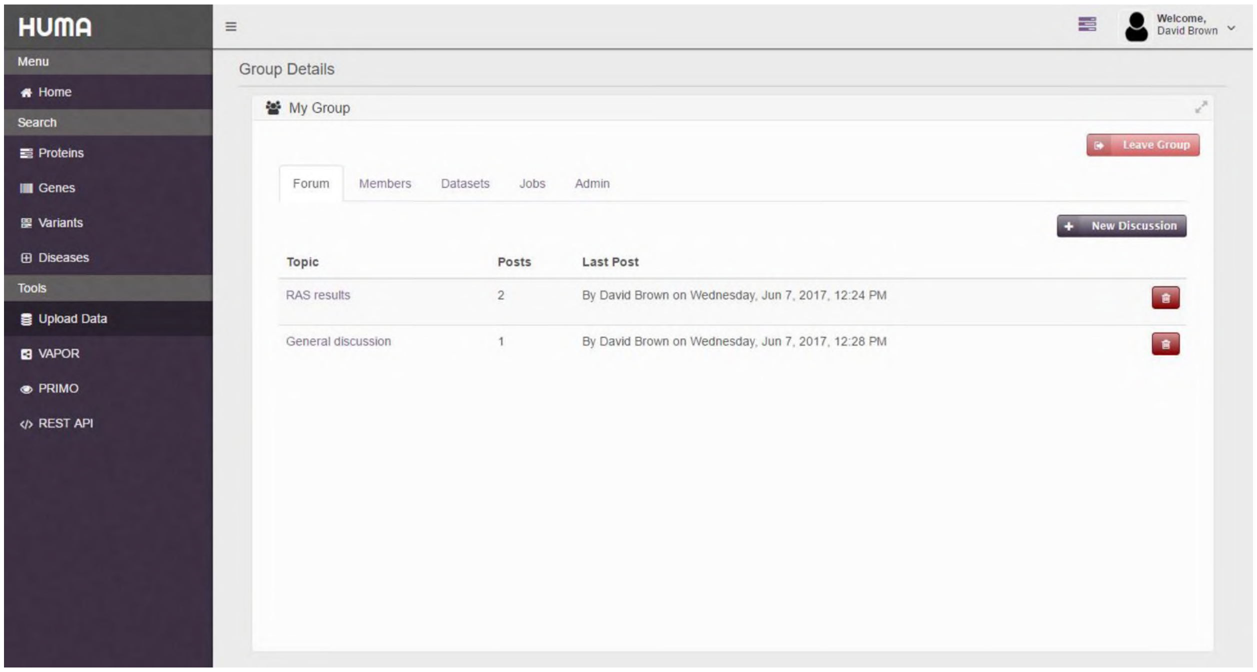
Fig. 6.17. Group datasets – datasets that have been shared with a group can be accessed by any member of the group via the "Datasets" tab on the group page.

Once shared with a group, the job can be located under the “Jobs” tab on the given group’s page. Any user belonging to the group will then be able to access this job, view the results and repeat the job with altered parameters.

6.7.4. Discussion boards

Discussion boards can be accessed via the “Forum” tab on a group’s page (Fig. 6.18). Any user can start a new discussion on this page by clicking on the “New Discussion” button and providing a topic in the resulting dialog.

As with traditional forums, users can then discuss the topic by adding posts to the discussion. These discussions could, for example, relate to datasets or analysis results that have been shared with the group. In future, we will allow users to attach items, such as files, links to items in a dataset, or job results, to their posts. These attachments could be used to give a given post more context.



Topic	Posts	Last Post
RAS results	2	By David Brown on Wednesday, Jun 7, 2017, 12:24 PM
General discussion	1	By David Brown on Wednesday, Jun 7, 2017, 12:28 PM

Fig. 6.18. Group forum – from this page, users can create new discussions and post to existing discussions. This allows members of a group to discuss results of jobs, datasets, and other interesting topics. In future, members will be able to share files and other items with one another here as well.

6.8. Maintenance

Software projects must be maintained over the course of their life time. In the case of HUMA, this means keeping the database populated with the latest available data, keeping web server up and running, and continuing to develop new and useful features.

As mentioned in previously (section 5.5), a pipeline was developed to populate the database with data from existing, public web servers. This same pipeline will be used to keep this data up-to-date. Over time, it can also be extended to include additional data from other databases.

The processes behind deploying and maintaining the HUMA web server and dependencies, such as Elasticsearch, are currently being documented. This documentation will allow future administrators and developers to easily maintain and further develop the web server.

Lastly, funding has been allocated to further develop and maintain HUMA for H3ABioNet purposes. This will help to ensure interest in the project in the medium to long term.

6.9. Summary

The HUMA web server has been developed to allow web-based access, visualization, and analysis of data in the HUMA database. This functionality is available via a user-friendly web interface as well as a logically designed web API.

The HUMA web server was developed using the Django web framework. It makes use of a modular architecture, which will allow the service to scale with usage. Each module focuses on logically related functionality such as account creation and management, data access, dataset creation and access, and job submissions and management.

The HUMA web interface has been developed as an SPA. This means that it consists of a single web page and interacts with the server using lightweight AJAX requests. To save time on the initial page load, only the sections of the page that are needed at that time are initially loaded. As the user interacts with the website, additional sections are loaded on demand. This model saves bandwidth and allows makes the user experience more dynamic and fluid.

Although numerous biological databases with web-based access exist, HUMA is unique in that it is also a platform for the analysis of this biological data. It aggregates data from numerous existing sources into a single connected database and then provides tools, such as PV-MSA, PRIMO, and VAPOR, to visualize and analyze this data.

Additionally, HUMA is unique in that it allows researchers to upload their own private datasets to the server and use the built-in tools to visualize and analyze the data. HUMA also facilitates collaboration by allowing researchers to share and discuss their datasets and the results of analyses. Although developed with H3Africa researchers in mind, HUMA is a useful tool for any researcher.

Currently, HUMA is focused on the analysis of variation at the protein structure level. In future, HUMA will provide tools to analyze non-coding variation as well. This is a natural evolution for the HUMA web server, since non-coding variants are already stored in the database and can, thus, already be found via searches on the *Variants* and *Genes* pages.

Part 3: Analyzing residue interaction networks to determine the effects of genetic variation

7. MD-TASK

7.1. Introduction

As discussed in chapter 4 (sections 4.4 – 4.6, particularly), structural bioinformatics can play an important and informative role in SNV analysis. Where sequence based techniques, such as GWAS and CGAS, allow us to associate variation with disease and phenotype traits, structural bioinformatics techniques, such as homology modeling, docking, and MD simulations allow us to understand *how* SNVs affect protein function and stability.

With advancing computational power, MD simulations have become an increasingly common method for analyzing protein structures. Traditionally, MD simulations have been analyzed using methods such as Root Mean Square Deviation (RMSD), Root Mean Square Fluctuation (RMSF), Radius of Gyration (RG), and energy-based approaches like MM/PBSA or MM/GBSA [228].

In section 4.5.1, the use of RINs in the structural analysis of proteins was alluded to. In section 4.6, a protocol for the analysis of SNVs using structural bioinformatics was described. As part of this protocol, we suggested the use of RINs to analyze MD trajectories. In this chapter, we discuss this idea in more detail and present MD-TASK, a novel tool suite for analyzing MD trajectories using network analysis techniques, as well as Perturbation Response Scanning (PRS), and Dynamic Cross-Correlation (DCC). The work presented in this chapter has been published [229] and MD-TASK has been open-sourced and made freely available (https://github.com/RUBi-ZA/MD-TASK).

7.1.1. Residue Interaction Networks

A Residue Interaction Network (RIN) can be defined as a graph, where the vertices (or nodes) correspond to residues in the protein and edges exists between the vertices if an interaction occurs between the respective residues [230]. An interaction is considered to occur if the residues are within a user-defined threshold distance of each other (usually 6.5 – 7.5 Å)[231].

By definition, since a RIN is a graph, RINs can be analyzed using a branch of mathematics known as graph theory. This means that various network measures can be used to analyze RINs, but we will focus on two methods in particular.

7.1.1.1. Average Shortest Path (L)

The shortest path between two nodes, i and j, in a network is defined as the least number of edges that must be traversed to reach i from j. Due to the protein backbone, it is safe to assume that all nodes in a network are always reachable from all other nodes in the network (assuming the cut-off threshold suggested above). The average shortest path (L) of a node i can, therefore, be calculated by summing the shortest paths between i and all other nodes and dividing the result by N-1, where N is the number of nodes in the RIN. In simpler terms, it is the average of all the shortest paths to i.

L represents the accessibility of a residue within a protein structure. Previous studies have suggested that residues with high L values help to steer conformational change [232].

7.1.1.2. Betweenness Centrality (BC)

Betweenness centrality (BC) of a node, i, can be defined as the number of shortest paths, between all other nodes in the network, that run through i. As such, it is a measure of how central a node is for communication in the network. Previous studies have suggested that residues with high BC values are located at positions that are critical to inter- and intra-protein domain communication [232].

7.1.2. Use of RINs in structural bioinformatics

The analysis of the RINs of protein structures is an area of structural bioinformatics that has been gaining traction in recent years, particularly with regards to variant analysis.

Previously, the Mutation-Minimization (MuMi) method was proposed as a means of performing Alanine scanning [232]. MuMi is a *in silico* method where each residue in a protein is independently mutated to Alanine. The resulting structures are then minimized and the changes in L (ΔL) and BC (ΔBC) between residues in the mutated structures and the wild-type structure are measured. Variants that result in significant fluctuations indicate that the mutated residue could play an important role in the protein.

RINs have also been used in cancer research. Aier *et al* [233] used RINs in combination with MD simulations, principal component analysis (PCA), and free energy landscape analysis to understand the molecular mechanism behind changes in the interactions of zeste homolog 2. RIN analysis provided useful data depicting the overall conformational changes induced by the variants.

Similarly, Anwar and Choi [234] analyzed the RINs of wild-type and variant Toll-like receptor 4 (TLR4) proteins to investigate the signalling mechanisms associated with the variants. Once again, MD simulations and PCA were also carried out as part of the study.

As can be seen from the above studies, the combination of RIN analysis and MD analysis is popular. However, in these studies, RINs were analyzed independently of the MD simulations *i.e.* RINs were calculated for the wild-type and variant structures, not the trajectory.

Bakat *et al* [235] took a step towards analyzing the RINs of MD trajectories when they used the average structure from an MD trajectory to calculate the RIN. In this work, the RIN of HIV-1 reverse transcriptase was analyzed, in combination with binding free energy analysis and PCA, to report the impact of the M184V variant on drug resistance. In this case, interacting residues were identified using PROBE [236] and RINs were constructed using RING [237].

Similarly, Xue *et al* [238] analyzed the RIN derived from the average structure of the last 10ns of MD trajectories to investigate the mechanism of cross-resistance to HIV-1 integrase strand transfer inhibitors and suggested that the combination of MD simulations and RINs can be useful in for investigating drug resistance in any biomolecular system.

Demonstrating this general applicability, in another study, Xue et al [239] used a similar methodology when analyzing Hepatitis C virus (HCV) NS5B protein variants to investigate the mechanism behind drug resistance to two inhibitors, VX-222 and ANA598.

More recently, groups have started analyzing the changes in RINs over the course of an MD trajectory. We will refer to RINs that are monitored over the course of a trajectory as Dynamic Residue Networks (DRN).

Although still a relatively new concept, DRNs have been used to provide informative data on several proteins and diseases. For example, Karamzadeh *et al* [240] calculated the DRN (although they referred to it as the Dynamic Residue Interaction Network, or DRIN) while analyzing human protein disulphide isomerase (hPDI). Comparisons of the DRNs of hPDI in oxidized and reduced states revealed potential allosteric paths between the catalytic and ligand binding sites of hPDI.

Additionally, chapter 8 of this thesis describes our use of DRNs to investigate the impact of variants in the Renin-Angiotensinogen complex [217]. In this work, DRNs revealed the mechanisms behind a potentially destabilising variant.

7.1.3. Research motivation

Although effort is being made to address this, tools to analyze the effects of variation at the structural level remain rare. An area, which shows promise in this regard is the analysis of RINs, with the aim of comparing RINs of mutated proteins with the RIN of the wild-type.

As discussed in the previous section, RIN analysis has been successfully employed in various studies to analyze the mechanisms behind disease and resistance. In these studies, tools such as RING [237], RINerator and RINalyzer [241] were used to generate and visualize RINs given a protein structure. Lately, the analysis of DRNs has been gaining popularity. However, to our knowledge, no tools currently exist for generating, visualizing, and analyzing DRNs. This has resulted in groups having to develop their own custom tools and scripts for this purpose – a time-consuming and error-prone process.

With this in mind, and considering the protocol described in section 4.6, it was concluded that a suite of tools for analyzing the effects of variation on MD simulations using network analysis would provide value to our group, the H3Africa consortium, and the scientific community as a whole.

7.1.4. Research aims and objectives

The aim of this project is to provide a suite of tools aimed at analyzing the effects of genetic variation via MD simulations. The project will make use of network analysis techniques to compare the trajectories of wild-type proteins with those from variant proteins. In addition, the tool suite will incorporate tools developed by other members of the lab, also to aimed at analyzing MD trajectories. These tools include a script to perform PRS analysis, developed by David Penkler, a script to generate residue contact maps, developed by Olivier Sheik Amamuddy, and a script to perform DCC, developed by Caroline Ross.

The specific objectives for this work are as follows:

1. Develop a suite of tools, MD-TASK, to analyze *BC* and *L* over the course of a trajectory.
2. Incorporate a tool, supplied by David Penkler, for performing PRS into the tool suite.
3. Incorporate a tool, supplied by Olivier Sheik Amamuddy, for generating weighted residue contact maps into the tool suite.
4. Incorporate a tool, supplied by Caroline Ross, for calculating DCC into the tool suite.
5. Ensure that all tools in the suite conform to a standard and consistent design and CLI

7.2. Implementation details

7.2.1. Platforms

MD-TASK was developed using Python and has been tested on Ubuntu Linux, Red Hat Enterprise Linux, MacOS, and Windows 10, although it is only officially supported on Linux-based systems.

7.2.2. 3rd party libraries

During development, various non-standard Python libraries were used. These included NumPy, which was used for matrix manipulations, SciPy, which was used for statistical calculations during PRS, Matplotlib [242], which was used for plotting results, MDTraj [243], which was used to read in trajectories in a range of formats, and NetworkX , which was used to calculate and analyze the RINs.

Additionally, the script provided by Olivier Sheik Amamuddy made use of the igraph package for R to generate residue contact maps. This is not ideal, as it adds a dependency on R. Fortunately, a Python igraph package exists and future work is planned to migrate off R.

7.2.3. Documentation

Detailed documentation was written for MD-TASK using in reStructuredText, a readable, what-you-see-is-what-you-get (WYSIWYG), plain text, mark-up language. Documentation was uploaded to GitHub as part of the MD-TASK repository.

Read the Docs (https://readthedocs.org/) (RtD) was used to host the documentation in website form. RtD support web hooks, which allow it to hook into GitHub repositories. This means that whenever the MD-TASK documentation is updated and committed to GitHub, RtD automatically pulls and hosts the updated documentation. This ensures that the hosted documentation is always consistent with the GitHub repository. RtD documentation for MD-TASK is located at: http://md-task.readthedocs.io.

7.3. Methodology & Results

7.3.1. Network Analysis

7.3.1.1. Change in Delta Average Shortest Path ($\Delta\Delta L$)

As discussed in section 7.1.1.1, L refers to the average shortest path to a given node from all other nodes. Previously , the MuMi [232] method performed Alanine scanning by mutating residues in a protein to Alanine and calculating the change in L (ΔL) due to the variant. MD-TASK takes this method a step further by first calculating ΔL for a wild type and variant protein over the course of an MD simulation, and then calculating $\Delta\Delta L$ by getting the difference of ΔL for the wild type and

ΔL for the variant. Where MuMi was developed for Alanine scanning, the suggested use for MD-TASK is SNV analysis. Given MD trajectories for a wild type and variant proteins, the method can be broken up as follows:

1. Calculate the DRN for the wild type protein

Calculating the DRN is done by calculating the RINs at given time intervals throughout the trajectory (*e.g.* every 100th frame). As mentioned before, when calculating a RIN, an interaction exists when two residues are within a certain cut-off distance from one another. By default, this distance is 7Å in MD-TASK, but users can specify a custom cut-off. When measuring the distance between residues, MD-TASK measures the distance between the Carbon-Beta atoms of the residues (Carbon-Alpha in the case of Glycine).

Each resulting RIN is essentially and N x N adjacency matrix, where N is the number of residues in the protein. In an adjacency matrix, the elements in the matrix, which can be either 1 or 0, indicate whether the vertices are adjacent or not. In a RIN, the vertices are residues and edges existing between the vertices depict residue interactions. As such, a 1 at position (5, 18) indicates that residue 5 interacts with residue 18.

Given a 100ns trajectory, if we calculated the DRN using a time interval of 1ns, we would end up with 101 RINs, including the RIN at time = 0. The DRN of the protein refers to the combination of these RINs in time order, and can be used to examine the change in the RINs over the course of the trajectory.

2. Calculate *ΔL* for the wild type DRN

Firstly, for each RIN in the DRN, all vs all shortest paths must be calculated. MD-TASK uses a custom algorithm implemented via the `all_pairs_shortest_path_length` function [244] in the NetworkX library. This results in an N x N matrix for each RIN where the elements in the matrix represent the number of paths that must be traversed to reach one residue from another. For example, a 4 at position (4, 12) means that 4 edges must be traversed to reach residue 12 from residue 4.

Given an N x N matrix such as this, where elements represent the shortest paths to a residue, a column represents all the shortest paths to a single residue. As such, the average shortest path (*L*)

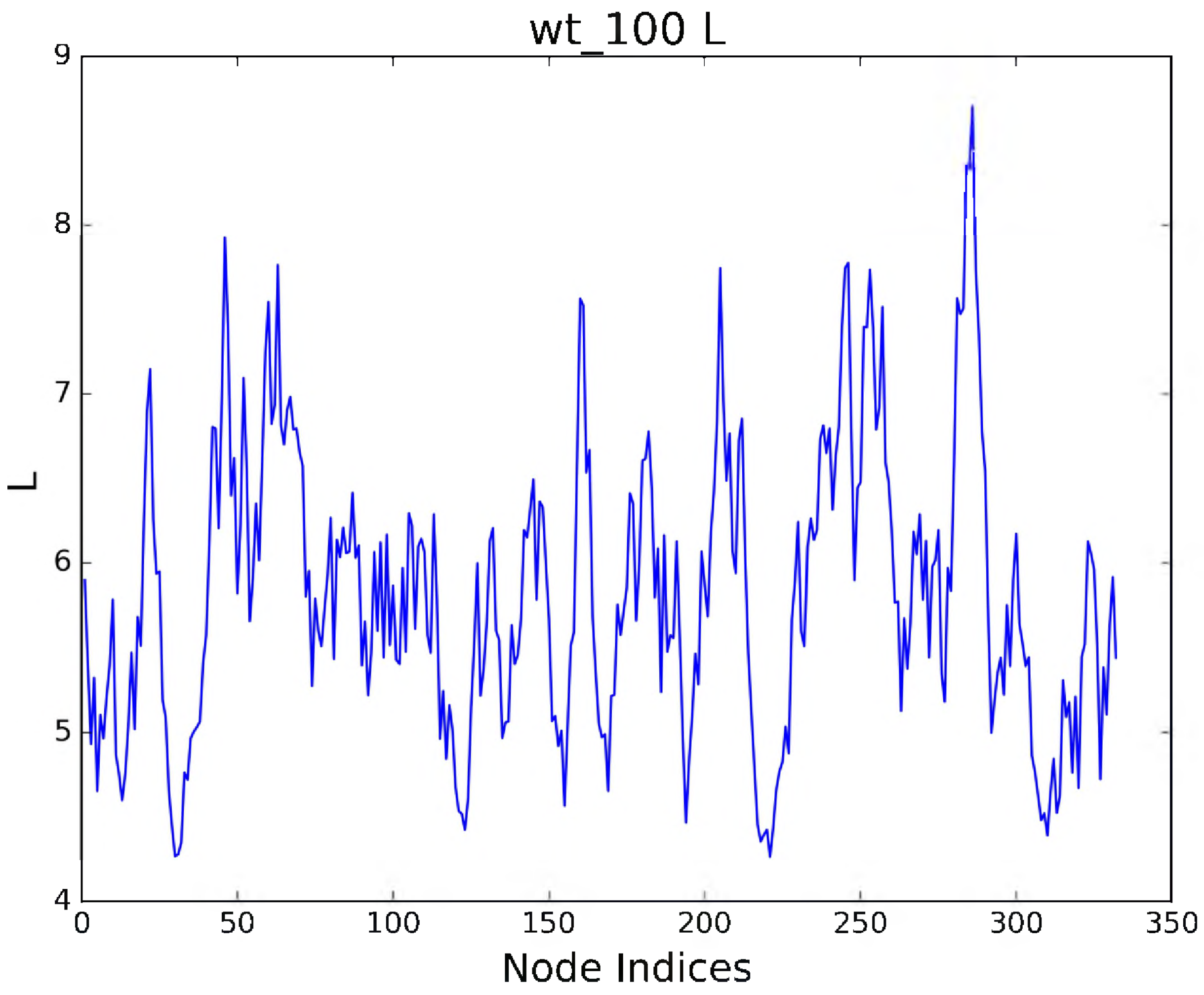


Fig. 7.1. *L* plot – plot of the average shortest path (*L*) to residues in an example protein.

to a residue can be worked out by summing all the values in the column for that residue and dividing by (N – 1), where N is the number of residues.

MD-TASK performs this calculation for all residues in all RINs in the DRN. Given the example in step 1, this results in 101 vectors (N x 1 matrix), where the values in the vector represent the *L* for the respective residues (Fig. 7.1).

Next, MD-TASK calculates the difference between the reference *L* vector (*L* at time = 0) and each of the 100 latter vectors, resulting in 100 *ΔL* matrices. These matrices are graphed using the matplotlib Python library (Fig. 7.2).

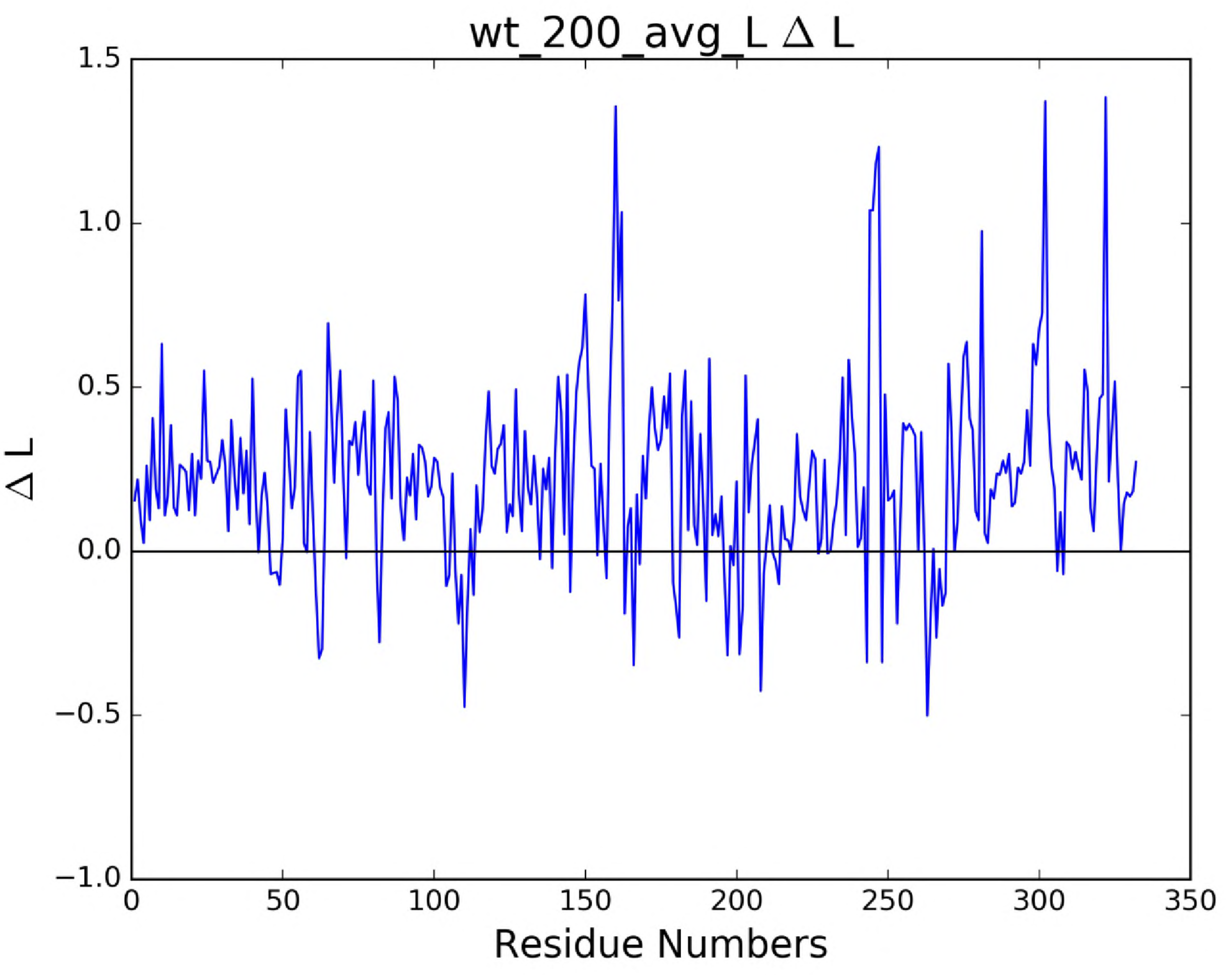


Fig. 7.2. *ΔL* plot – plot depicting the difference between *L* of the reference frame (first frame in the trajectory) and *L* of the 200th frame. The values are normalized (*ΔL*/*L*).

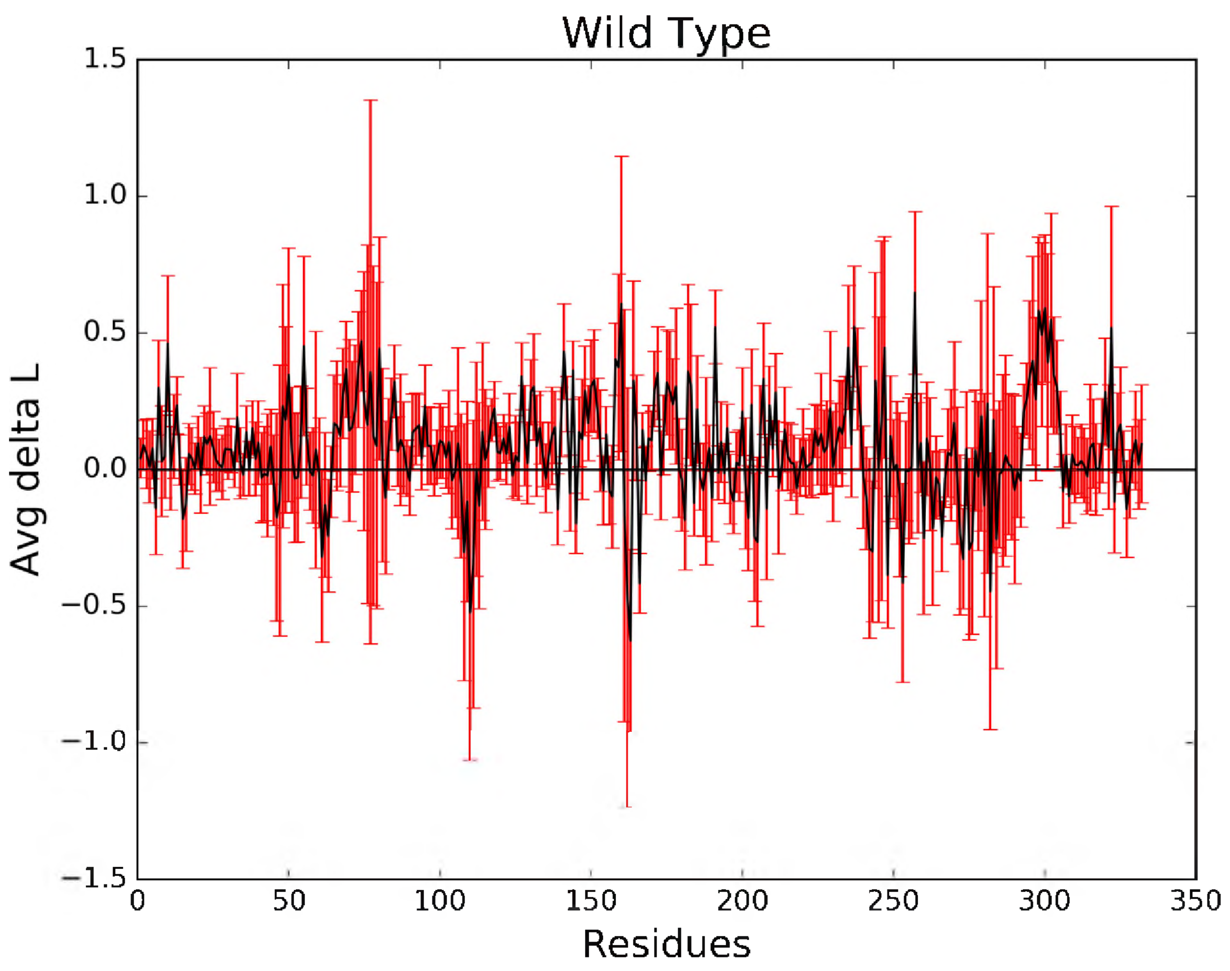


Fig. 7.3. *ΔL* of DRN – average *ΔL* over the course of the trajectory is plotted via the black line, while the standard deviation of *ΔL* over the course of the trajectory is represented by the red error bars.

3. Calculate average and standard deviation of *ΔL* for the wild type

Viewing 100 *ΔL* plots for every trajectory is not useful. To condense all this information down to a single plot, MD-TASK calculates the average *ΔL* and standard deviation of *ΔL* for each residue from all the *ΔL* plots. The results are stored in two vectors of length N. The average *ΔL* and standard deviation of *ΔL* can then be plotted on a single graph, with the average represented by a line and the standard deviation represented by error bars (Fig. 7.3).

The average *ΔL* represents the average change in *ΔL* from the reference frame. If the network changed abruptly at the beginning of the simulation, but stabilized thereafter, average *ΔL* would be very high. However, this doesn't give an indication of how much the network fluctuates over the course of the trajectory. For that, the error bars representing standard deviation provide useful information. Large error bars indicate that the network at the respective position fluctuated greatly over the course of the trajectory.

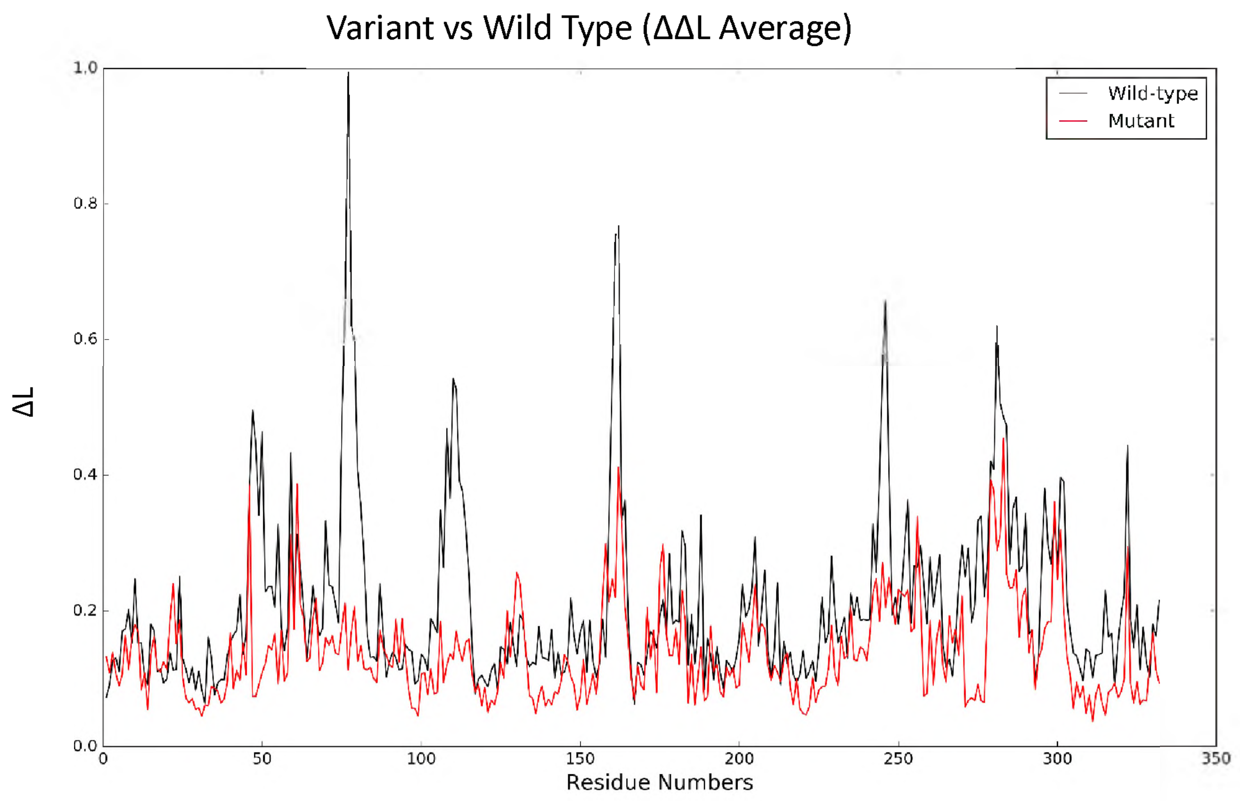
4. Repeat steps 1 through 3 for the variant trajectory

Steps 1 through 3 calculated and plotted the DRN for the wild type trajectory. This process should be repeated for the variant trajectory.

5. Calculate change in average and standard deviation of ΔL ($\Delta\Delta L$) between the wild type and variant trajectories

At this point, we have 4 vectors representing the average and standard deviation of ΔL for the wild type and variant protein respectively, which have been plotted on to two graphs. These graphs can be compared to determine areas that have changed due to the variant. However, comparing these two plots via the naked eye can be difficult. As such, MD-TASK provides a utility that calculates and plots the differences between two vectors. These $\Delta\Delta L$ plots make it easy to see areas in the variant protein where the ΔL has changed in comparison to the wild type (Fig. 7.4 and 7.5).

When analyzing large numbers of SNVs, the result of an MD-TASK network analysis will be $\Delta\Delta L$ graphs for each of standard deviation and average ΔL for each variant trajectory. If analyzing 20 variant trajectories, for example, this would result in 20 standard deviation $\Delta\Delta L$ plots and 20 average $\Delta\Delta L$ plots. This data can be condensed into two heat maps, where each row in the heat map represents a $\Delta\Delta L$ plot. MD-TASK provides a utility for creating such heat maps, an example of which is given in the following chapter.



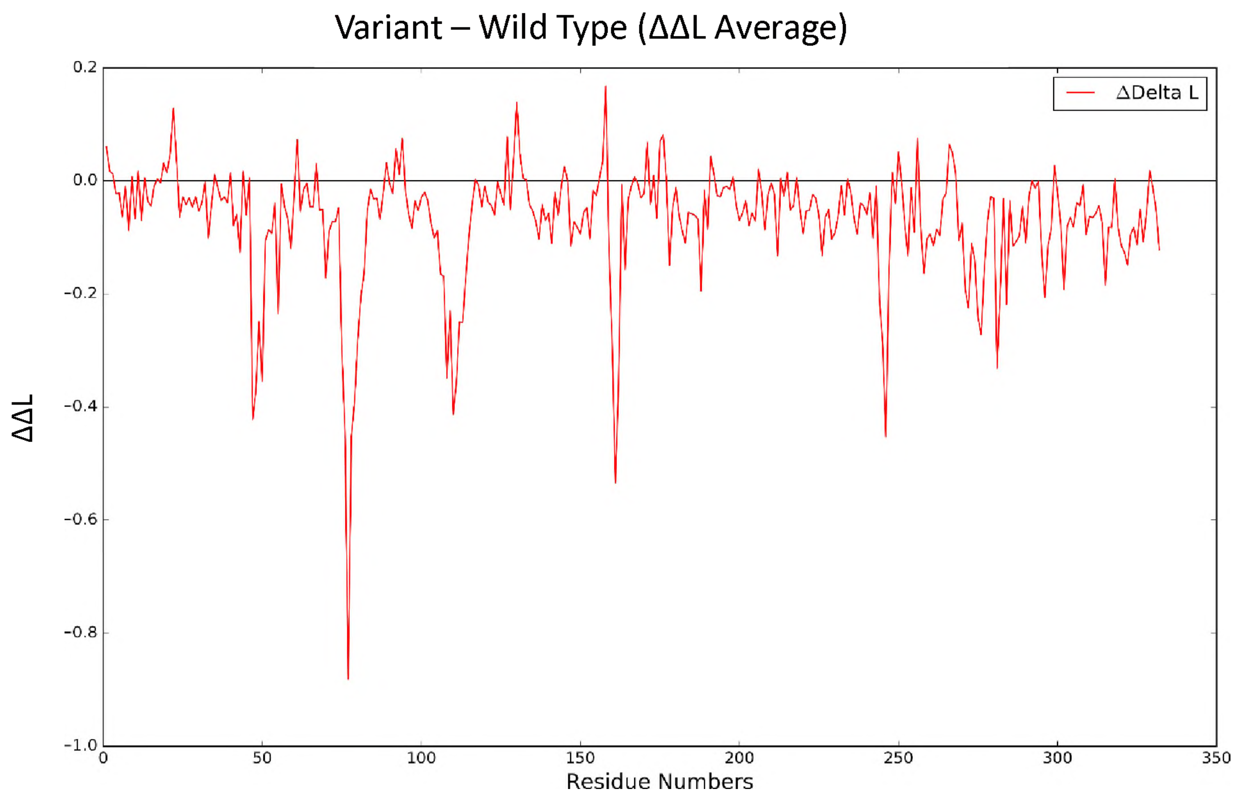
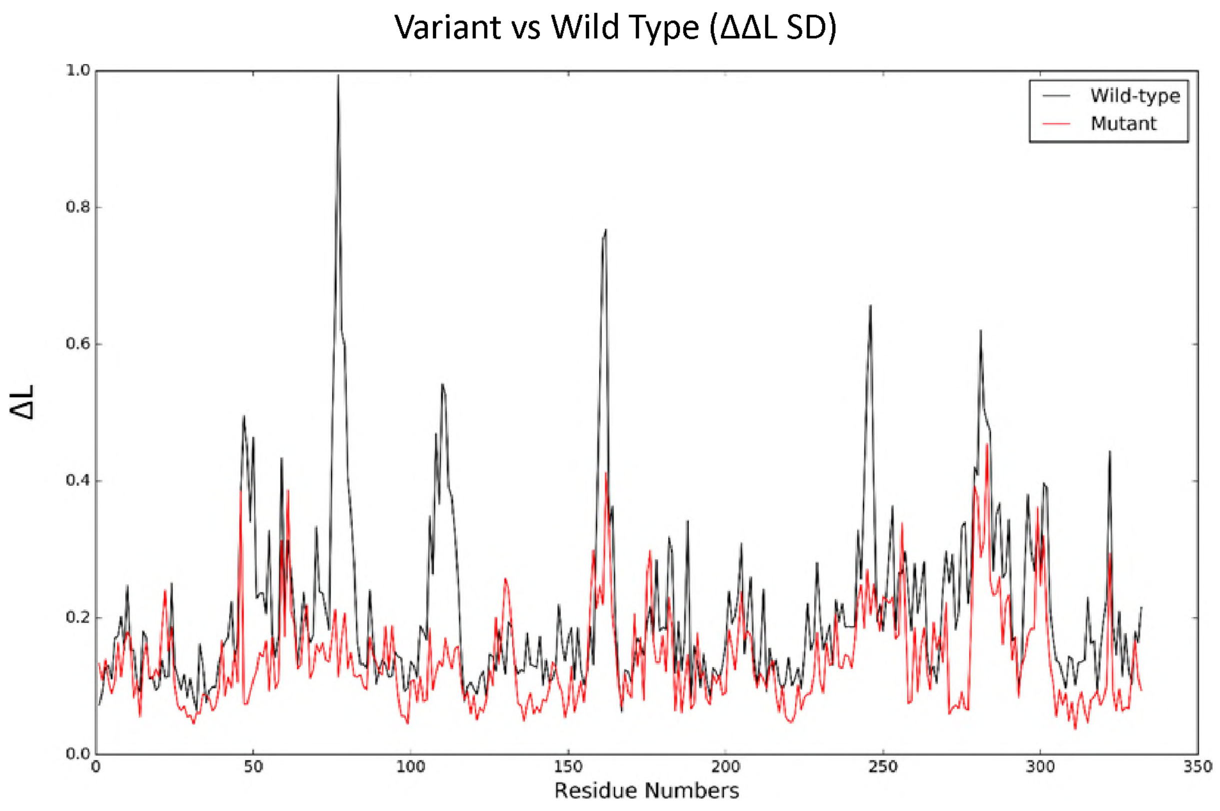


Fig. 7.4. *ΔΔL* Average –**A)** Average ΔL of the wild type (black) and variant (red) proteins plotted on the same set of axes. **B)** The difference between average ΔL of the variant and average ΔL of the wild type (variant – wild type).



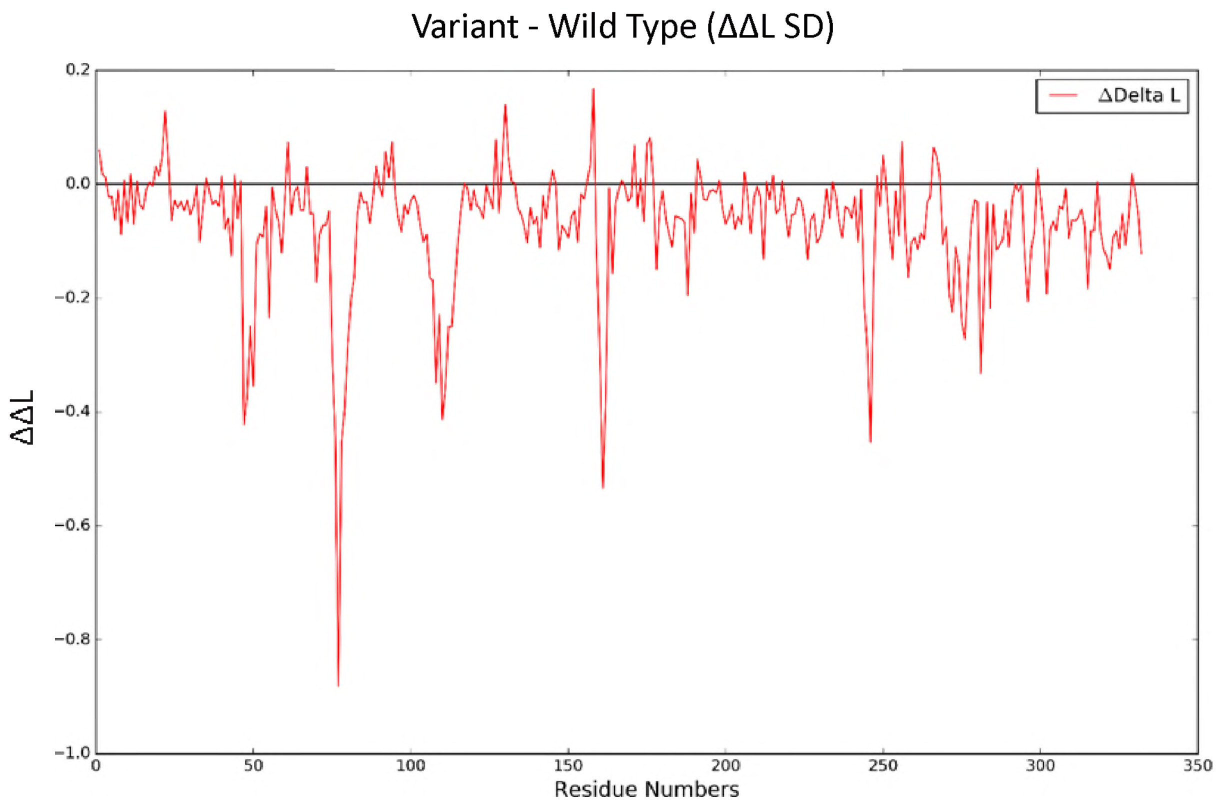


Fig. 7.5. *ΔΔL* Standard deviation (SD) –**A)** SD ΔL of the wild type (black) and variant (red) proteins plotted on the same set of axes. **B)** The difference between SD ΔL of the variant and SD ΔL of the wild type (variant – wild type).

7.3.1.2. Change in Delta Betweenness Centrality (*ΔΔBC*)

As discussed in section 7.1.1.2, *BC* refers to how central a node is for communication in a network, or, put another way, how many shortest paths run through a given node. MD-TASK follows almost the exact same steps for calculating *ΔΔBC* as for *ΔΔL*. The only difference comes in step 2, where, instead of calculating the all vs all shortest path matrix and then collapsing it into the *L* vector, MD-TASK makes use of an implementation of Brandes algorithm [245] in the NetworkX Python library to calculate a vector of length N, where each value in the vector represents *BC* for the residue at that index.

From then, since in both cases we are simply working with a set of vectors, the same methodology as used when working with *L* can be used to calculate ΔBC, the average and standard deviation of *ΔBC*, and the *ΔΔBC* plots. As such, we will not repeat this methodology here.

7.3.1.3. Residue contact maps

The tool to generate the residue contact maps was written by Olivier Sheik Amamuddy, a PhD student at Rhodes University. The initial script was refactored and cleaned up so that it conformed to the standards of the existing MD-TASK scripts. It is the only tool in the suite that make use of the statistical language, R. This is due to its dependence on the R igraph library [246]. However, an igraph package exists for Python, and there are plans to convert this tool to Python in future. This will decrease dependencies, improve maintainability, and make it easier to install.

In MD-TASK, residue contact maps are diagrams that, essentially, represent a column in an adjacency matrix *i.e.* all the interactions for a given residue. The given residue is represented as a circle in the middle of the diagram. Residues that interact with the given residue (*i.e.* have a value of 1 in the column in the adjacency matrix) are represented by circles arranged around the given

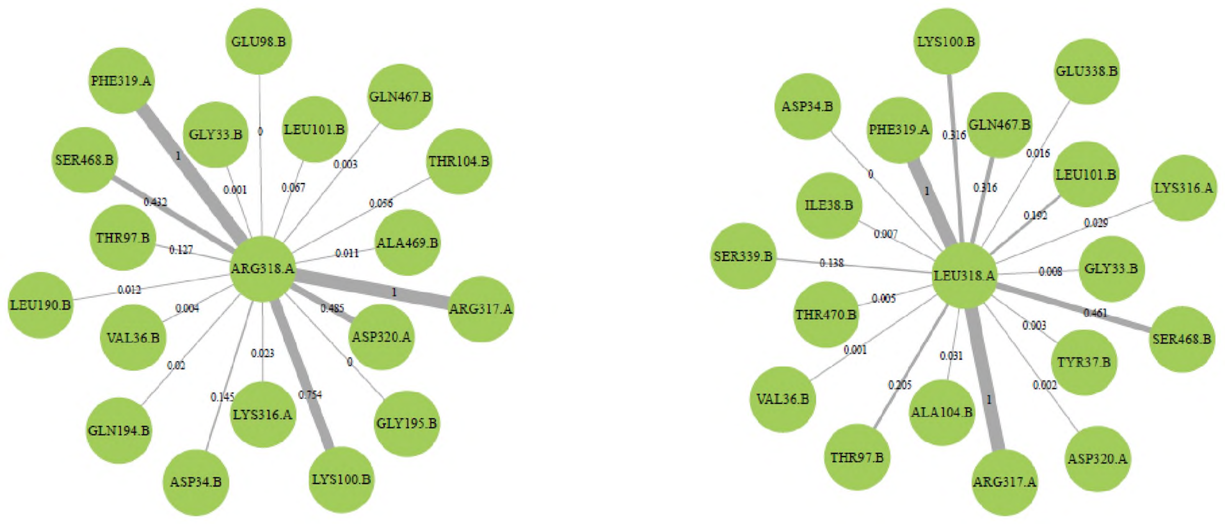


Fig. 7.6. Residue contact maps – residue contact maps for position 318 in a wild type (left) and variant (right) protein, where position 318 has mutated from Arginine to Leucine.

residue and connected to it by a simple line (an edge in the graph). Since MD-TASK works with DRNs, these edges are weighted based on how often the residues interacted thought the MD simulation (Fig. 7.6).

7.3.2. Perturbation Response Scanning (PRS)

The PRS script was contributed to the MD-TASK suite by David Penkler, another PhD student at Rhodes University. As with the residue contact map scripts, the initial script was refactored and cleaned up to be better in line with the existing MD-TASK scripts.

PRS is a computational technique that is useful for determining residues that play an important role in the conformational changes of a protein. The initial and target conformations of a protein are provided as input to the tool. Each residue in the initial conformation is then sequentially perturbed *i.e.* multiple random forces are exerted on the residues. A variance-covariance matrix produced from a suitable length MD trajectory from the initial structure can then be used to calculate the set of residues and forces that resulted in conformational changes closest to the target structure. The quality of the predicted displacements is then assessed by correlating the predicted and experimental displacements, averaged over all affected residues. The result is a correlation coefficient for each residue in the protein, output in CSV format, where a value close to 1 implies good agreement with the experimental change. Further details on the PRS technique can be obtained from David Penkler's work [247].

7.3.3. Dynamic Cross-Correlation (DCC)

DCC is a commonly used method for determining the correlation coefficients of the motions of atoms in MD simulations [248]. The DCC between two residues, *i* and *j*, can be describes as follows:

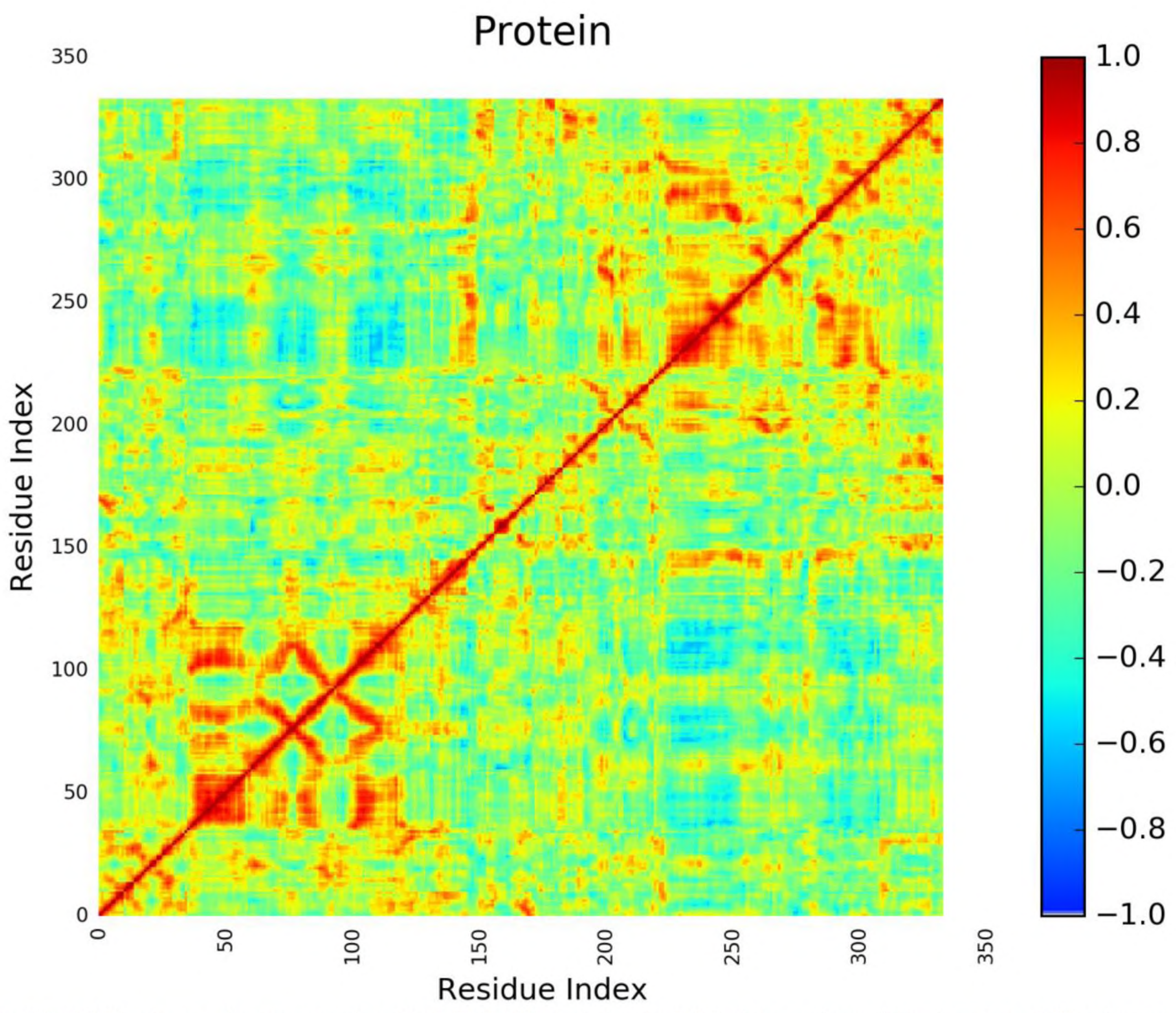


Fig. 7.7. DCC heat map – heat map produced by MD-TASK depicting the DCC of the motions of the atoms in an MD trajectory.

$$C_{ij} = \frac{\langle \Delta r_i \cdot \Delta r_j \rangle}{\sqrt{\langle \Delta r_i^2 \rangle} \cdot \sqrt{\langle \Delta r_j^2 \rangle}}$$

where Δr_i is the displacement from the average position of atom i and $\langle\rangle$ means the time average over the whole trajectory [249].

The DCC script in MD-TASK was contributed by Caroline Ross, another PhD student at Rhodes University, and produces a heat map as output (Fig. 7.7). As with the previous two scripts, it was refactored to conform to the standards of the existing MD-TASK scripts.

7.4. Conclusion

MD-TASK is a tool suite dedicated to analyzing MD simulations. To the best of our knowledge, it is currently the only downloadable tool kit that that can be used to analyze MD trajectories using *ΔΔBC*, *ΔΔL,* and PRS. These techniques have already been used in published work [217,247] and have proven to produce useful insights that correlate well with results from other methods.

Future work to turn MD-TASK into a web server is planned. The MD-TASK scripts will be housed in JMS and a custom web interface will be built for the various analysis scripts. Additionally, the dependence of MD-TASK on R will be removed by using the Python version of the igraph library to produce residue contact maps.

There is also work under way to produce new analysis scripts that use normal mode analysis and principal component analysis to analyze MD trajectories. Although these tools won't be added to the MD-TASK tool suite, they will be added to the web server.

8. Case study: the renin-angiotensin system

8.1. Introduction

The renin-angiotensin system (RAS) is a hormone system involved in regulating blood pressure and sodium levels in the blood [250]. Also known as the renin-angiotensin-aldosterone system (RAAS), it consists of several different proteins that convert the inactive angiotensinogen protein into the highly active angiotensin II octa-peptide [251]. Angiotensin II then acts in various ways to stimulate vasoconstriction in blood vessels and the reabsorption of salt and water in the kidneys, both of which lead to an increase in blood pressure [251].

Research presented in this chapter investigates the impacts of SNVs on certain proteins involved in RAS and has previously been published [217]. All figures and tables that have been re-used from this publication were done so with permission from the journal.

8.1.1. RAS and blood pressure control

8.1.1.1. Renin release

Juxtaglomerular (JG) cells are specialized smooth muscle cells located mostly in the walls of afferent arterioles in the kidney. These cells play an important role in RAS as they are responsible for secreting renin in response to three main triggers [252,253].

The first trigger is low blood pressure [254], which the JG cells are able to detect via the baroreceptors in the vascular walls. This mechanism is related to the activation of the sympathetic nervous system [255].

The second trigger is the sympathetic nerve cells that end right next to the JG cells and fire due to any major stressor, such as being in a fight, or being chased. When these nerves fire, they stimulate the release of Renin from the JG cells [255].

Lastly, the macula densa cells, which are located in the distal convoluted tubule of the nephron in the kidney, have the ability to sense the sodium in the fluid passing through the nephron. When blood pressure is low, more salt is reabsorbed into the body. The macula densa cells can sense the reduced level of sodium in the fluid passing by, which results in them signalling the JG cells via prostaglandins, thus stimulating the release of Renin [256,257].

Renin is a hormone that plays a central role in the regulation of blood pressure, specifically with regards to increasing blood pressure by cleaving a peptide, angiotensin I, from the large protein, angiotensinogen [255]. Angiotensin I is later converted to angiotensin II, a highly active peptide that acts to increase pressure in a number of ways (Fig. 8.1), as described in section 8.1.1.4.

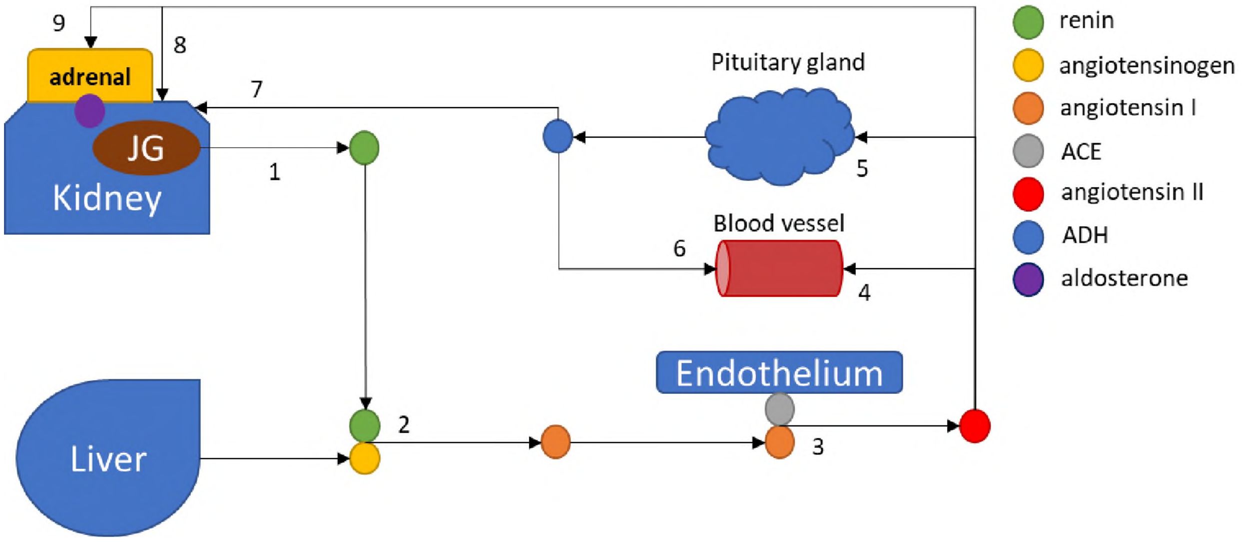


*Fig. 8.1. RAS – **1)** Renin is secreted by the JG cells in the kidney; **2)** Renin cleaves angiotensin 1 from angiotensinogen; **3)** ACE cleaves a further two residues from angiotensin I to form angiotensin II; **4)** angiotensin II causes vasoconstriction of the blood vessels; **5)** angiotensin II acts on the pituitary gland to produce ADH; **6)** ADH causes vasoconstriction of the blood vessels; **7)** ADH acts on the kidney and causes increased water absorption; **8)** angiotensin II also acts directly on the kidney and causes increased sodium absorption; **9)** angiotensin II acts on the adrenal gland causing it to produce aldosterone, which acts on the kidneys resulting in increased sodium absorption.*

8.1.1.2. Angiotensinogen

Liver cells produce the precursor hormone, angiotensinogen. When renin is released into the blood stream, it encounters and interacts with angiotensinogen throughout the body and cleaves a small peptide, angiotensin I, off angiotensinogen [252].

8.1.1.3. Angiotensin Converting Enzyme (ACE)

Angiotensin I is a 10 amino acid long peptide. After being cleaved from angiotensinogen, it continues its journey through the body until it encounters the endothelial cells in the capillaries mostly in the lungs, but also in other locations throughout the body. These endothelial cells have enzymes that sit on their surface called Angiotensin Converting Enzyme (ACE). When angiotensin interacts with ACE, a further 2 amino acids are cleaved from the peptide, resulting in the octapeptide, angiotensin II [252].

8.1.1.4. Activities of angiotensin II

Angiotensin is a highly active enzyme that acts on four main targets (Fig. 8.1). Firstly, it acts on the smooth muscle cells in the walls of blood vessels to get them to constrict. This is called vasoconstriction, which increases resistance in the blood vessels and, thus, results in an increase in blood pressure [252].

Secondly, angiotensin II has a direct effect on the proximal convoluted tubules in the kidney, which results in increased sodium reabsorption. Sodium is reabsorbed into the blood, which also results in water being reabsorbed into the blood. This increases stroke volume (SV), which is directly related to blood pressure [258].

Thirdly, angiotensin II acts on the Pituitary gland, causing it to secrete antidiuretic hormone (ADH) [259]. Like angiotensin II, ADH causes vasoconstriction of the blood vessels. In the kidneys, however, ADH causes water reabsorption by creating "channels" for water to pass through from the distal convoluted tubule (where the cell membranes are not usually permeable to water) back into the blood. As before, and increased water reabsorption results in increased SV, and thus, increased blood pressure.

Lastly, angiotensin II acts on the adrenal gland, which sits on the kidneys, and causes it to produce the hormone, aldosterone [260]. Like angiotensin II, aldosterone acts on the kidneys to cause increased sodium reabsorption, and thus, increased SV.

8.1.1.5. Lowering blood pressure

RAS controls blood pressure via a feedback loop. Low blood pressure and low levels of sodium in the blood signal the release of renin, ultimately leading to the production of angiotensin II, which causes an increase in blood pressure and reabsorption of salt, which in turn, kills the signals that resulted in renin secretion [254].

8.1.2. Hyperactivity of RAS

High blood pressure, also referred to as hypertension, is a medical condition where blood pressure in the arteries is consistently elevated. Hypertension can be caused by the overactivity of RAS [250,261–263].

Additionally, hyperactivity of RAS has been linked to congestive heart failure [264–266], kidney disease [263], and diabetes [258]. Conversely, modulation of RAS has been shown to slow and even reverse the negative vascular affects related to diabetes mellitus [254].

Hyperactivity in RAS could be related to genetic variation. Previous studies have found that polymorphisms in RAS may increase risk of hypertension, cardiomyopathy, and renal failure [267–270]. In addition, varying responses to antihypertensive drugs among individuals could also be related to genetic variation [271].

8.1.3. Inhibiting RAS

Due to the importance of RAS in diseases such as hypertension, heart failure, and diabetes, it has often been a target for drug development. Inhibiting this system can result in lower blood pressure, reduced risk of heart and renal failure, as well as reduced risk of developing diabetes [272]. As such, three classes of RAS inhibitors have surfaced, namely, ACE inhibitors, angiotensin receptor blockers (ARBs), and direct renin inhibitors [273].

As described previously, ACE is responsible for converting angiotensin I to angiotensin II by cleaving two residues off the C-terminal of the angiotensin I peptide. ACE inhibitors reduce the activity of RAS by blocking this conversion [272].

ARBs, on the other hand, reduce the activity of RAS by blocking angiotensin II from binding to angiotensin II receptors, type 1 (AT_1) [272]. They are primarily used for patients who are intolerant to ACE inhibitors. ARBs are also often used in *combination* with ACE inhibitors [272]. Since ACE inhibitors don't completely block the formation of angiotensin II (blockage is dose dependant and angiotensin can be produced via a non-ACE pathways), ARBs can be used to prevent the angiotensin II that does form from interacting with AT_1. As such, combining ARBs and ACE inhibitors can be more potent than either type on its own.

The final class of RAS inhibitors are renin inhibitors. Renin inhibitors reduce RAS activity by preventing renin from cleaving angiotensin I from angiotensinogen [274]. Renin inhibitors show promise, since they inhibit the first and rate-limiting step in RAS [251,275], which could lead to more complete inhibition of RAS than is provided by existing drugs. They have remained difficult to produce however, with early generations having poor bioavailability and potency. Third generation renin inhibitors solved many of these issues, however, with aliskiren becoming the first drug in this class to reach the market [276,277].

8.1.4. Research motivation

RAS plays an important role in modulating blood pressure and sodium levels in the body. Hyperactivity of RAS results in hypertension, has been linked to increased risk of renal and heart failure, and is closely tied to diabetes. The seriousness of these conditions has resulted in RAS receiving considerable attention from researchers. This has lead to the development of three types of RAS inhibitors. ACE inhibitors prevent the formation of angiotensin II from angiotensin I, ARBs prevent angiotensin II from interacting with AT_1 receptors, and direct renin inhibitors prevent angiotensin I from being cleaved from angiotensinogen.

While several ARBs and ACE inhibitors are available on the market, renin inhibitors remain difficult to produce. Currently, only a single renin inhibitor, aliskiren, has reached the market. However, direct renin inhibitors remaining a promising concept due to their mechanism. Since renin inhibitors operate at the rate-limiting, first step of the RAS feedback loop, they are potentially able to more completely inhibit the activity of RAS in comparison to other inhibitors.

Since renin inhibitors act to prevent renin from cleaving angiotensin I from the angiotensinogen, it is necessary to properly understand the interaction between renin and angiotensinogen. This includes discovering residues in either protein that are important for the interaction, as well as determining the effects of variation on the interaction. The renin-angiotensinogen complex, therefore, provides a useful and relevant case study on which to test the tools developed throughout this thesis.

8.1.5. Research aims & objectives

The purpose of this case study is to use the tools developed throughout the course of this thesis to analyze the effects of variation in the renin-angiotensinogen complex to determine potentially damaging SNVs and residues that are important for protein function.

The specific objectives of this work are as follows:

1. Obtain a dataset of all known SNVs in renin and angiotensinogen from HUMA
2. Determine potentially damaging SNVs in dataset using VAPOR
3. Model damaging SNVs into the respective proteins
4. Subject the wild type and variant structures to 100ns MD simulations
5. Analyze resultant trajectories using MD-TASK

8.2. Methodology

8.2.1. Data retrieval

Appropriate protein sequences and structures for renin and angiotensinogen, as well as all known SNVs in those proteins, were identified via a search of the HUMA database. SNVs were then downloaded from HUMA, while the protein sequences and structures were downloaded from Uniprot [114] and the PDB [186], respectively. Structure suitability was based on coverage of the Uniprot sequence as well as the validation metrics available on the PDB website.

8.2.2. Variant filtering

Two SNV datasets were obtained from HUMA, one each for renin and angiotensinogen. These datasets contained all variation from dbSNP that could be mapped to the respective proteins. Both datasets were initially filtered by removing nonsense and synonymous SNVs.

Due to the size of the datasets, they were further filtered by submitting them to the VAPOR pipeline on the HUMA website. Unless there was a disease association in HUMA for the variant, if more than one of the programs that make up the VAPOR pipeline predicted that a variant was benign, it was removed from the dataset.

Lastly, the Protein Interactions Calculator (PIC) [212] was used to identify interacting residues in the complex. All remaining SNVs that were not at the interface between the two proteins were removed. This was done to reduce the SNV datasets to a size that could feasibly be analyzed using computationally expensive techniques like MD.

8.2.3. Homology modeling

8.2.3.1. Wild type

The wild type monomers had to be modeled to account for missing residues. Modeling was done manually using MODELLER [11], as PRIMO is unable to model complexes.

The complex has previously been solved in the PDB structure, 2X0B. As such, it was used as the main template for modeling. Chain A of this structure covers renin, while chain B covers angiotensinogen. Three additional structures, 2WXY, 2WXW, and 2WXZ were used to cover missing residues in angiotensinogen.

PROMALS3D was used to align the templates to the target structure. After aligning, the first 73 residues of the renin sequence and the last 3 residues of the angiotensinogen sequence were trimmed as they were not covered by the templates.

8.2.3.2. Variants

Each SNV in the two datasets was independently introduced into the respective protein sequences and modeled using the same methodology as described for the wild type. This resulted in a model of the complex for each SNV in both datasets.

Additional models were generated where SNVs interacted with each other. In these cases, both SNVs were introduced into the protein sequences and models were generated using the methodology described above. This was to determine whether certain SNVs may compensate for one another when co-occurring.

8.2.3.3. Model evaluation

100 models of the wild type and each of the variant proteins were produced. The top 3 wild type models were selected based on their DOPE z-score [278]. These models were further evaluated using PROCHECK [100], PROSA [101], and VERIFY3D [103]. The top model was then chosen based on the combination of these results.

The top variant models were chosen solely based on their DOPE z-scores as the process of submitting the top three models for each variant to each evaluation server would have been too time-consuming. Since each variant was only a single residue change, it is unlikely that variants would have produced significantly different results as compared to the wild types when being analyzed with PROCHECK, VERIFY3D, and PROSA. As such, it was deemed unnecessary.

8.2.4. MD simulations

MD simulations were executed by Olivier Sheik Amamuddy. In order to run the MD simulations in reasonable time, calculations were performed on the CHPC cluster in Cape Town, South Africa, using GROMACS 5.1 [279] on 480 CPU cores. The following process was followed for each model:

- The AMBER03 force field was used for topology generation and energy calculations.
- The system was solvated and neutralized in a triclinic periodic box, before being minimized using the method of steepest decent.
- The short-range interaction cut-offs were set a 1nm each and long range electrostatics were handled by the Ewald algorithm [280].
- Temperature was equilibrated at 310K over a 100ps and pressure was equilibrated at 1 bar using the modified Berendsen thermostat [281] and Parrinello-Rahman barostat [282], respectively.
- The production MD run was then executed over 100ns with time steps of 2fs. Co-ordinates were written to file every 5000 steps.
- The LINCS algorithm [283] was used to handle rotational band lengthening.

8.2.5. MD analysis

8.2.5.1. Standard analysis

As has become the standard in MD, the MD trajectories were analyzed using RMSD and RMSF. RMSD provides a holistic view of the overall stability of a protein and was computed from the Carbon-Alpha atoms. RMSF describes the movement of individual residues in the protein and was computed from the averaged motion of all the atoms in the residue. RMSD and RMSF calculations were performed by Olivier Sheik Amamuddy.

8.2.5.2. Residue contact maps

Residue contact maps were generated for each MD simulation by Olivier Sheik Amamuddy using the `contact_map.py` script in the MD-TASK tool kit. A cut-off distance of 6.7Å was used.

8.2.5.3. Change in L (ΔL)

MD-TASK was used to calculate and plot the average and standard deviation of *ΔL* for all the MD trajectories. As above, a cut-off of 6.7Å was used. Additionally, values were normalized using the `--normalize` flag provided by the MD-TASK tool kit (*ΔL/L*). Each variant trajectory was plotted against the wild type to identify variants that resulted in significant changes.

8.2.5.4. Change in BC (ΔBC)

As above, MD-TASK was used to calculate and plot the average and standard deviation of *Δ*BC for each trajectory using a cut-off threshold of 6.7Å. Variant plots were then compared to the wild type plots to identify significant changes.

8.3. Results & discussion

8.3.1. Variant filtering

317 SNVs from renin and 212 SNVs from angiotensinogen were downloaded from HUMA. After the filtering described in the methodology section was applied, only nine angiotensinogen SNVs (Table 8.1) and six renin SNVs (Table 8.2) remained.

dbSNP ID	Residue Change	Location	Reason for inclusion
rs539231427	H39R	Interface	• Highly damaging prediction by VAPOR • Interacts with position in renin where SNV occurs
rs746613821	P40L	Interface	• Highly damaging prediction by VAPOR • Interacts with position in renin where SNV occurs

rs41271499	L43F	Interface	• Highly damaging prediction by VAPOR • Interacts with position in renin where SNV occurs
rs760531325	E48K	Interface	• Interacts with position in renin where SNV occurs
rs751752211	S49G	Interface	• Interacts with position in renin where SNV occurs
rs377047370	S49N	Interface	• Highly damaging prediction by VAPOR • Interacts with position in renin where SNV occurs
rs201406560	A104T	Interface	• Interacts with position in renin where SNV occurs
rs767370325	M105V	Interface	• Interacts with position in renin where SNV occurs
rs756744141	D168Y	Interface	• Highly damaging prediction by VAPOR • Interacts with position in renin where SNV occurs

Table 8.1. SNV dataset for angiotensinogen **(table taken from RAS publication [217])**

dbSNP ID	**Residue Change**	**Location**	**Reason for inclusion**
rs868694193	D104N	Interface	• Highly damaging prediction by VAPOR • Interacts with position in angiotensinogen where SNV occurs
rs191049685	R148C	Interface	• Interacts with position in angiotensinogen where SNV occurs
rs371704012	R148H	Interface	• Interacts with position in angiotensinogen where SNV occurs
rs770190833	A188V	Interface	• Highly damaging prediction by VAPOR
rs752426689	L318R	Interface	• Interacts with position in angiotensinogen where SNV occurs
rs201922371	F319V	Interface	• Interacts with position in angiotensinogen where SNV occurs

Table 8.2. SNV dataset for renin **(table taken from RAS publication [217])**

8.3.2. Homology modeling

Overall, 26 high quality models were produced. The top model for the wild type structure had a DOPE z-score of -1.20. PROSA results (Fig. 8.2) were also good, as were the PROCHECK (90.2%

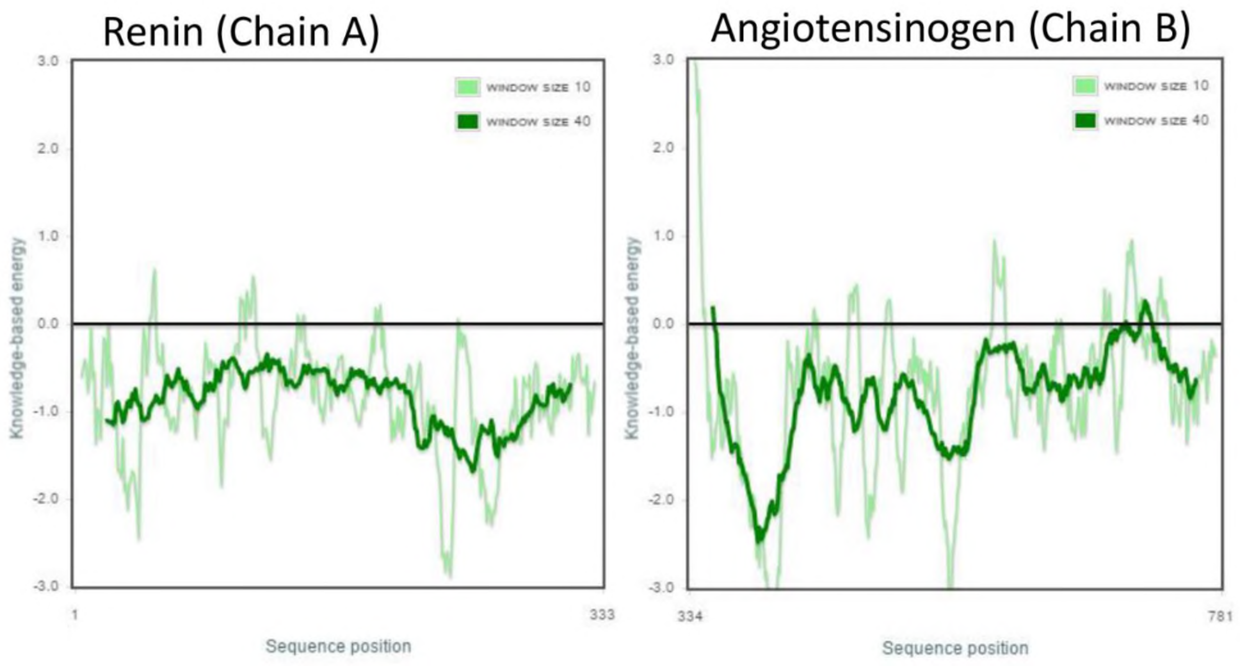


Fig. 8.2. PROSA results (re-used from RAS publication [217]) – results generated by PROSA for the wild type complex. Below the horizontal line is indicative of good quality.

of residues were found in the most favourable regions) and VERIFY3D (80.8% of residues 3-dimensional – 1-dimensional score of over 0.2) results.

The top variant models (Table 8.3) were chosen based on their DOPE z-scores. In all, six models containing a variant in renin, nine models containing a variant in angiotensinogen, and ten models containing variants at interacting positions in renin and angiotensinogen were produced, for a total of 25 variant models.

Model	**DOPE Z-Score**	**Renin (chain A) variants**	**Angiotensinogen (chain B) variants**
ren_D104N	-1.20	rs868694193	
ren_R148C	-1.19	rs191049685	
ren_R148H	-1.20	rs371704012	
ren_A188V	-1.19	rs770190833	
ren_L318R	-1.18	rs752426689	
ren_F319V	-1.17	rs201922371	
ang_H39R	-1.18		rs539231427
ang_P40L	-1.18		rs746613821
ang_L43F	-1.18		rs41271499
ang_E48K	-1.18		rs760531325

ang_S49G	-1.21		rs751752211
ang_S49N	-1.21		rs377047370
ang_A104T	-1.20		rs201406560
ang_M105V	-1.18		rs767370325
ang_D168Y	-1.19		rs756744141
ren_D104N_ang_L43F	-1.20	rs868694193	rs41271499
ren_R148C_ang_E48K	-1.17	rs191049685	rs760531325
ren_R148C_ang_S49G	-1.19	rs191049685	rs751752211
ren_R148C_ang_S49N	-1.20	rs191049685	rs377047370
ren_R148H_ang_E48K	-1.18	rs371704012	rs760531325
ren_R148H_ang_S49G	-1.20	rs371704012	rs751752211
ren_R148H_ang_S49N	-1.20	rs371704012	rs377047370
ren_L318R_ang_A104T	-1.20	rs752426689	rs201406560
ren_F319V_ang_A104T	-1.19	rs201922371	rs201406560
ren_F319V_ang_M105V	-1.20	rs201922371	rs767370325

Table 8.3. Variant models of the renin-angiotensinogen complex (table from RAS publication [217])

8.3.3. MD simulations

All 26 models were subjected to 100ns MD simulations using GROMACS. Of the 26 MD runs, the model ren_L318R failed due to a bad contact with water. Since L318R was only included, because it was at a position that interacts with a damaging SNV in angiotensinogen, and due to the large amount of data already available to us, we decided not to resolve this.

8.3.4. MD analysis

8.3.4.1. RMSD

After calculating the RMSD for the 25 successful MD simulations, only the complex containing the variant, P40L in angiotensinogen (ang_P40L), did not stabilize (Fig. 8.3). On closer investigation of the structure, it was found the P40L occurs in the region of angiotensinogen. This is the region that is cleaved by renin to become angiotensin I and instability due to a variant here may indicate that P40L could play an important role in the binding of renin to angiotensinogen and the cleaving of angiotensin I from angiotensinogen.

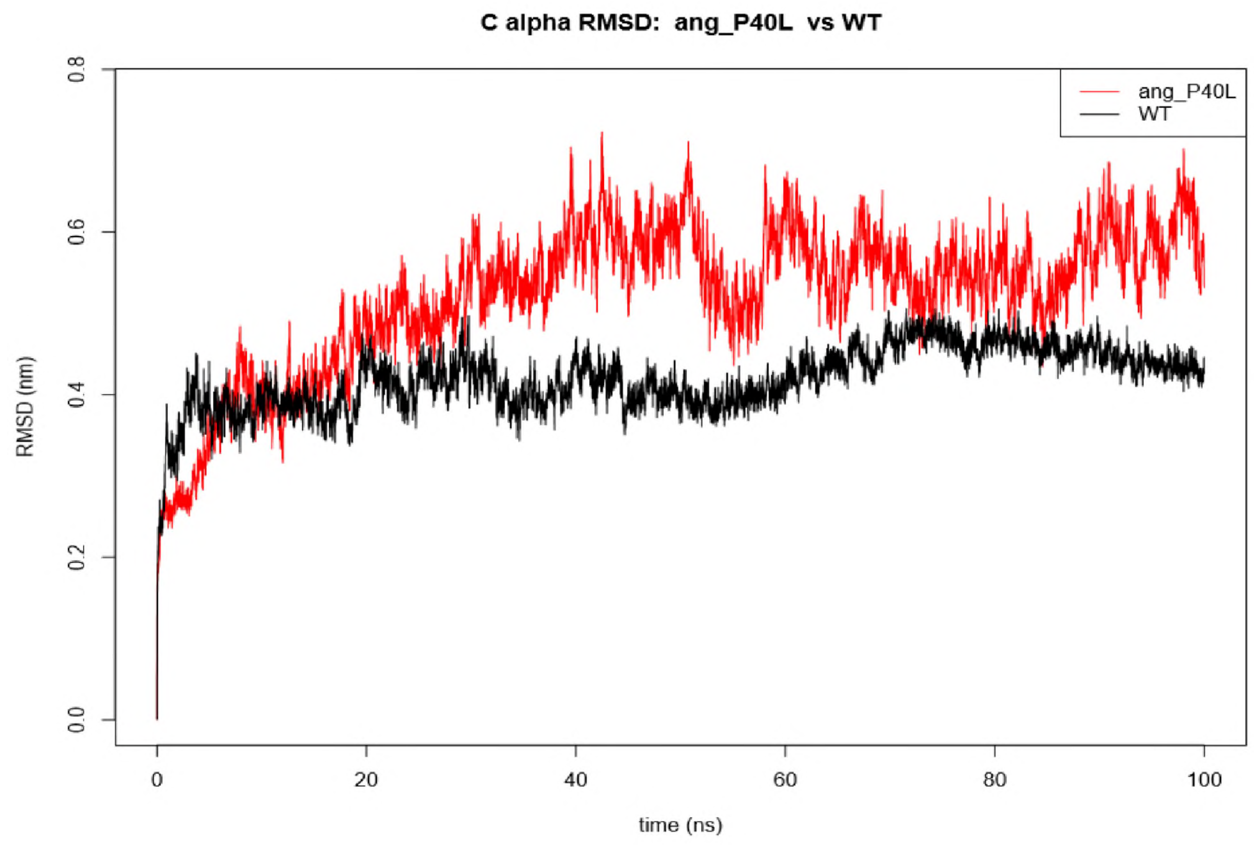


Fig. 8.3. *ang_P40L RMSD (re-used from RAS publication* [217]*) – RMSD for complex with variant, P40L, in angiotensinogen (red) vs wild type complex (black). The variant complex doesn't stabilize over the course of the 100ns trajectory.*

8.3.4.2. RMSF

RMSF revealed an overall trend towards increased rigidity in variant complexes, especially in chain A (renin), as opposed to the wild type complex. This was noticeable in the complex containing the variant, D104N in renin (Fig. 8.4), which is associated with renal tubular dysgenesis in HUMA. It was less noticeable in the unstable complex, ang_P40L, which did not stabilize over the 100ns. In this case, fluctuation was noticeable in chain B (angiotensinogen) of the complex (Fig. 8.5). The complexes containing the variants, A188V in renin (ren_A188V), and the combination of R148C in renin and E48K in angiotensinogen (ren_R148C_ang_E48K) showed similar fluctuation patterns in chain B.

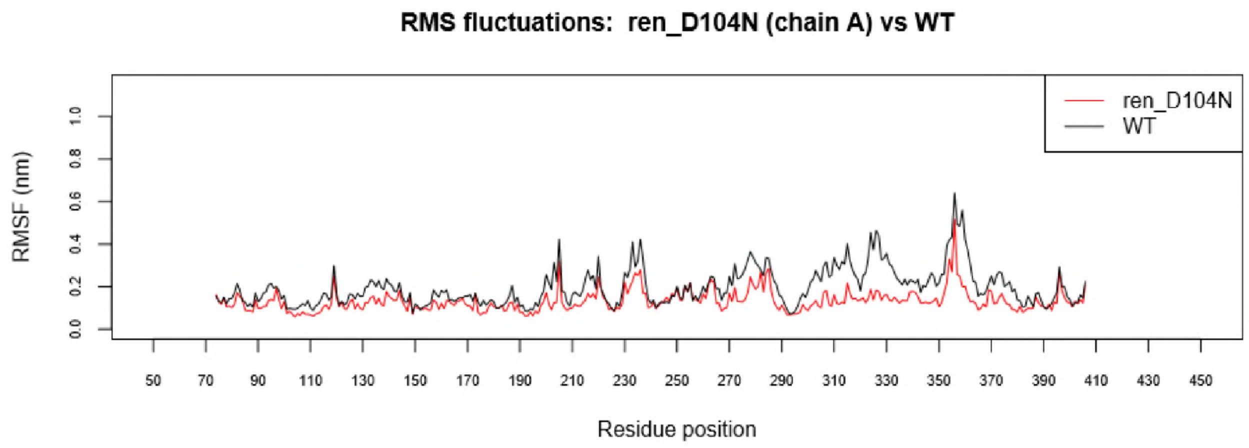


Fig. 8.4. ren_D104N RMSF (re-used from RAS publication [217]) – RMSF of chain A ren_D104N (red) shows increased rigidity when compared to the wild type (black)

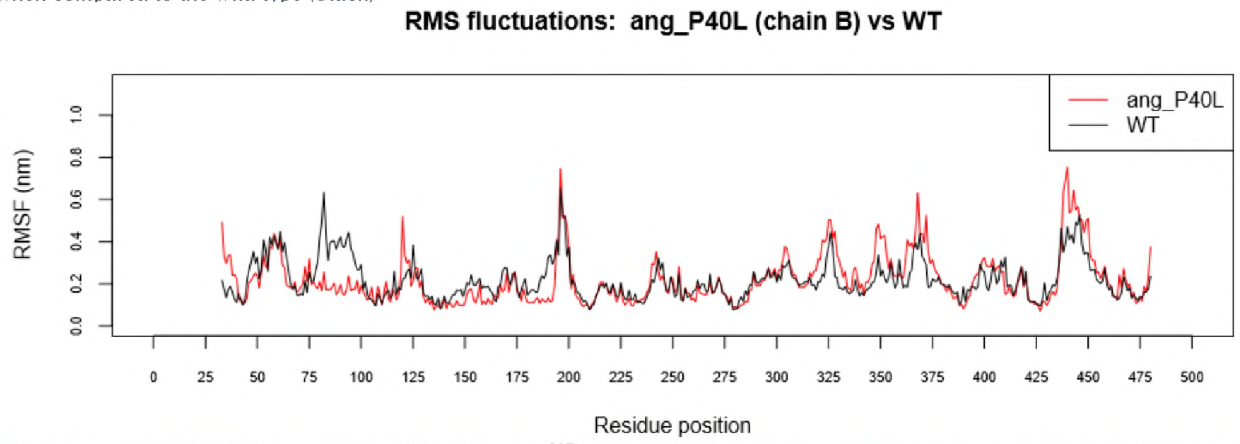


Fig. 8.5. ang_P40L RMSF (re-used from RAS publication [217]) – RMSF of chain B of ang_P40L complex (red) shows slight increase in fluctuation vs wild type (black)

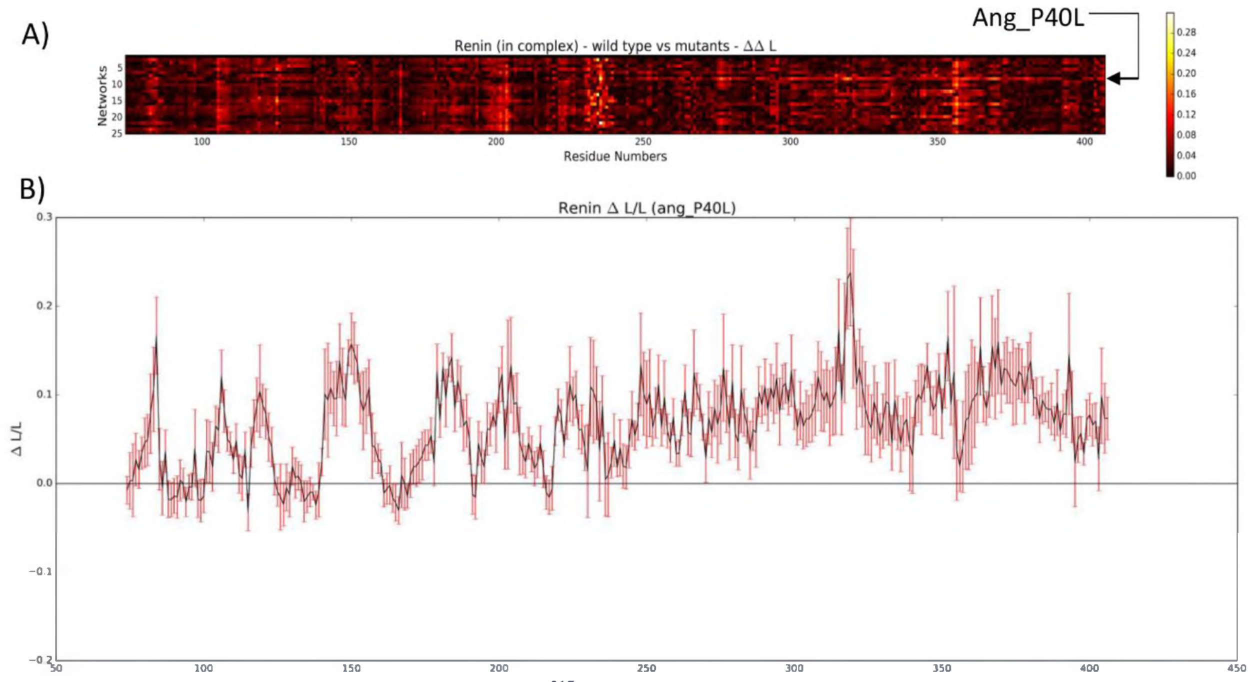


Fig. 8.6. ΔL/L ang_P40L (re-used from RAS publication [217]) – renin shows a significant increase in L/L vs the other variants.

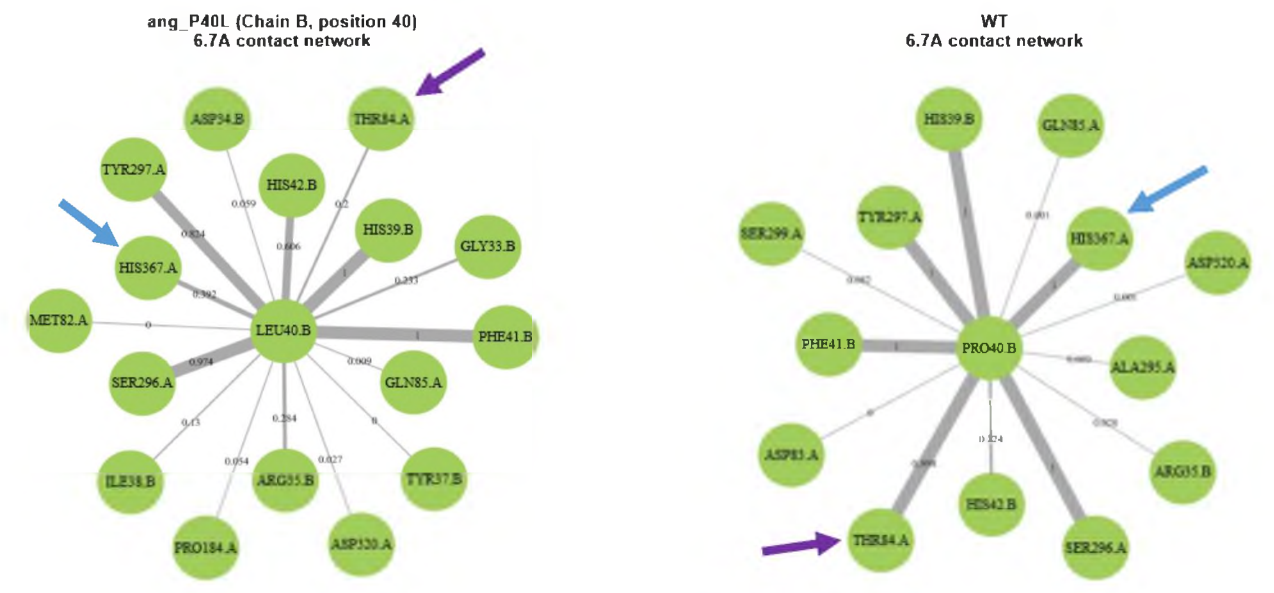


Fig. 8.7. ang_P40L residue contact map (re-used from RAS publication [217]*) – position 40 in angiotensinogen in variant complex shows decreased interaction with HIS367 (blue arrows) and THR84 (purple arrows) in renin.*

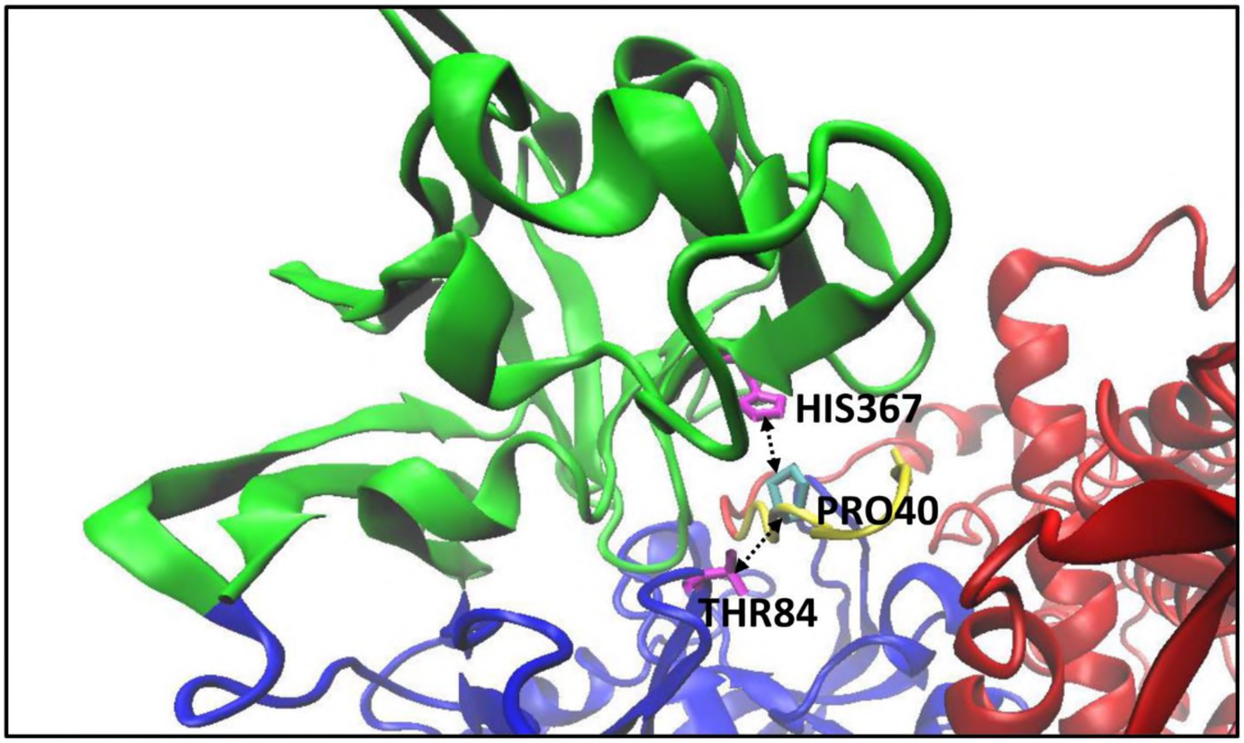


Fig. 8.8. Important interactions (re-used from RAS publication [217]*) – PRO40 in angiotensinogen (red) interacts with HIS367 and THR84 in renin (blue). When PRO40 is mutated to leucine, this interaction is lost and accessibility of a large portion of renin (green) is lost.*

the residue contact map for position 40 in the wild type vs the same position in the variant (Fig. 8.7), it can be seen that, in the variant, contact with residues HIS367 and THR84 is significantly reduced. The loss of these contacts is what results in the increase in $\Delta L/L$ for the back half of renin *i.e.* the back half of renin becomes less accessible (Fig. 8.8). Proline can interact favourably with aromatic residues, such as Histidine, due to both the hydrophobic effect and the interaction between the negatively charged π faces of the aromatic side chains and the positively charged face of the proline ring [284]. Mutating Proline to Leucine clearly has a negative effect on this interaction.

After examining the ang_P40L MD simulation in VMD, it can be seen that over the course of the simulation, the complex begins to break up, with renin and angiotensinogen starting to drift apart (Fig. 8.9). As such, it appears that the interaction between PRO40 in angiotensinogen and HIS367 and THR84 is important for the stability of the complex.

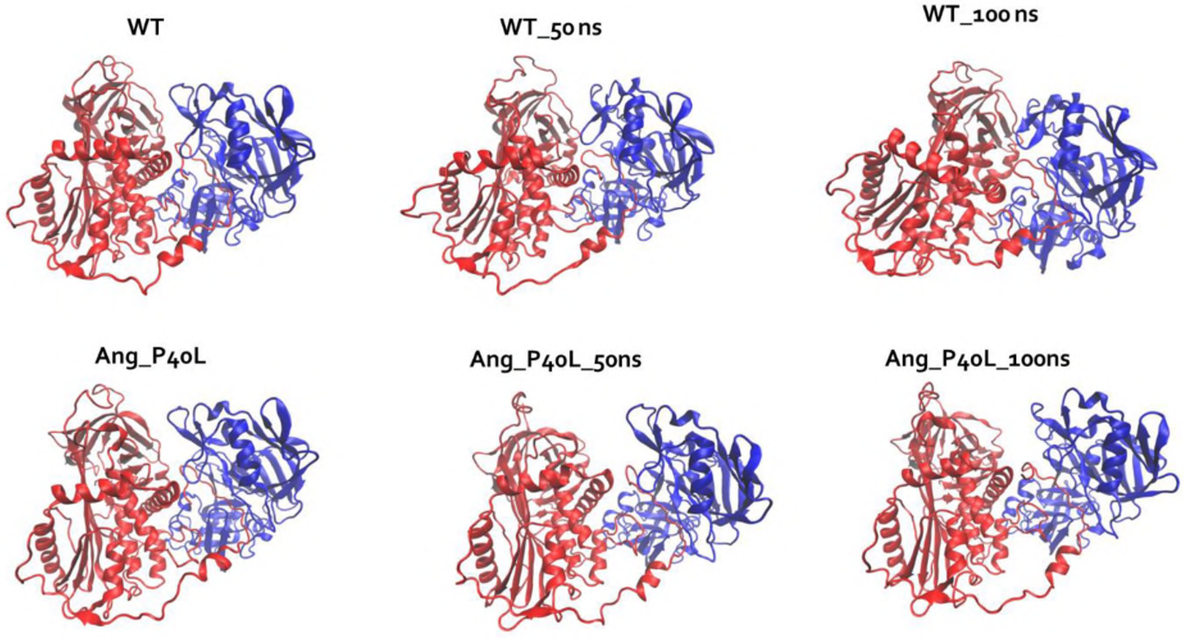


Fig. 8.9. WT vs ang_P40L MD simulation – during the simulation, the complex containing the variant, P40L in angiotensinogen, begins to break apart. This is in contrast with the wild type complex, which remains tightly bound throughout the entire simulation.

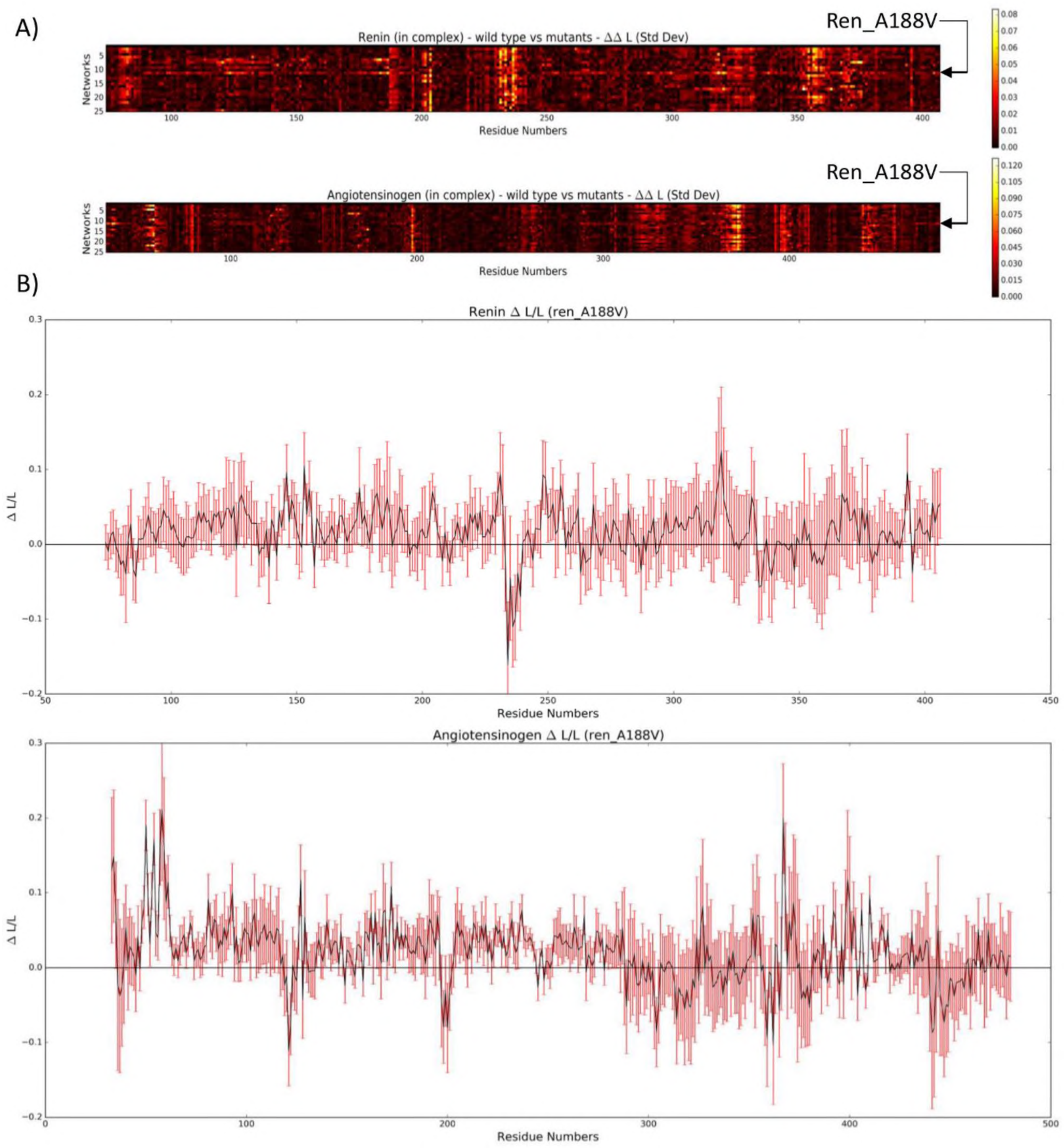


Fig. 8.10. *SD of ΔL/L for ren_A188V (re-used from RAS publication* [217]*) – ren_188V showed significant fluctuation of ΔL/L* throughout the trajectory

The standard deviation of *ΔL/L* in ren_A188V also proved interesting (Fig. 8.10). Despite the average *ΔL/L* for ren_A188V not signalling anything significant, the standard deviation of *ΔL/L* was significantly higher than that of the other variant complexes and the wild type, indicating that the network fluctuated throughout the simulation. Examining the residue contact map for position

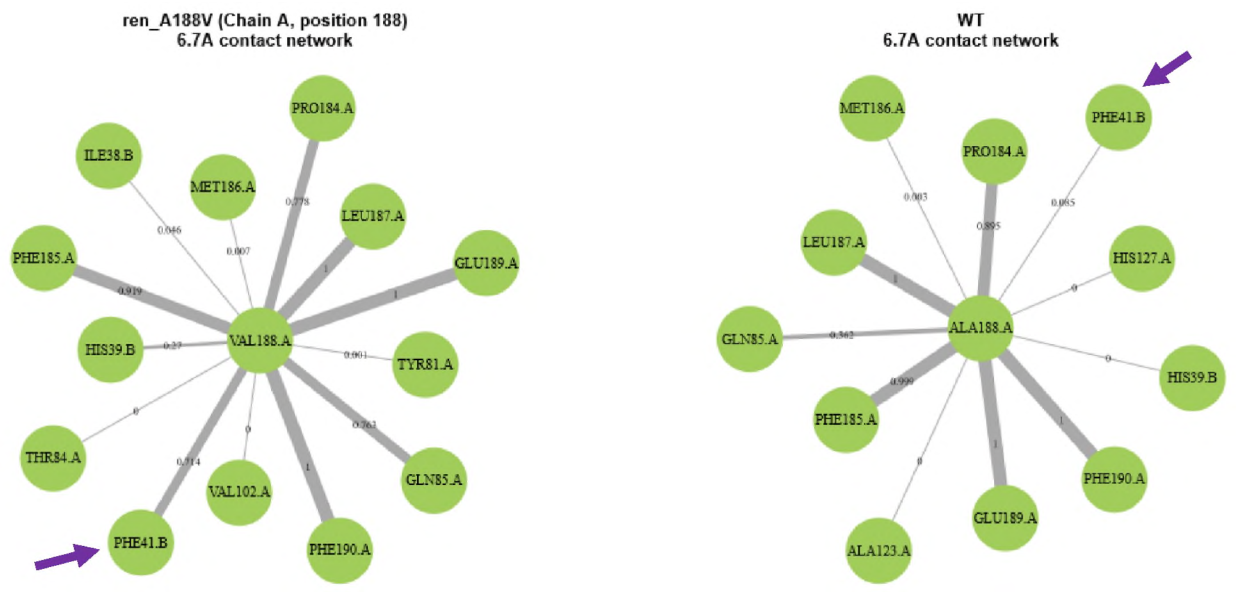


Fig. 8.11. Residue contact map for ren_A188V (re-used from RAS publication [217]) – position 188 in renin in the variant complex shows increased interaction with PHE41 in angiotensinogen.

188 in ren_A188V versus the wild type (Fig. 8.11), it appears that in the variant, there is 62.9% increase in interaction between VAL188 in renin and PHE41 in angiotensinogen, when compared to ALA188 in renin in the wild type. VAL188 interacts with PHE41 71.4% of the time. That this percentage is not very high also indicates that the contact was not particularly consistent, which could explain the fluctuation in *ΔL/L*.

8.3.4.4. Change in BC (ΔBC)

The average and standard deviation for *Δ*BC were also calculated for each MD simulation using MD-TASK. Results were relatively consistent across all variants *e.g.* in all variant complexes, significant network changes were limited to seven regions in renin and 5 regions in angiotensinogen.

8.4. Conclusion

RAS play an important role in the regulation of pressure and sodium levels in the blood. It has been implicated to play a role in various conditions including hypertension, heart and kidney failure, and diabetes. As such, it has been the target of considerable research in the past, resulting in the development of various classes of RAS inhibitors. One such class of inhibitors, direct renin inhibitors, are a promising prospect for treating hyperactivity of RAS. Unfortunately, they have been difficult to produce and, as always, further work is required. In this chapter, we used the tools developed throughout the course of this thesis to analysis the interaction between renin and angiotensinogen and as well as the effects of genetic variation on these interactions. Through this analysis, we found three residues, PRO40 in renin, and HIS367 and THR84 in angiotensinogen that appear to play an important role in the binding of renin to angiotensinogen. When the variant, P40L was introduced into renin, the interactions between position 40 in renin and HIS367 and THR84 were broken, resulting in the destabilizing of the complex. As a result, over the course of the simulation, renin and angiotensinogen began to drift apart.

This work also proved to be a good case study to showcase the use of HUMA, VAPOR, and MD-TASK. The *ΔL/L* network analysis could detect the changes in the network in renin that resulted in us further analyzing the ren_P40L complex and uncovering the important residues and interactions discussed above. The residue contact map was also useful in identifying the specific interactions that were affected. On the other hand, the RMSF analysis did not pick up any interesting results for these residues and, were we relying solely on RMSD and RMSF, we would not have been able to gain the detailed insight that we did.

Although *ΔBC* did not produce particularly useful insights in this study, we still believe it is an important measurement and that, in future studies, it will produce more useful insights. Unfortunately, there is no valid tool that will always produce interesting results.

Lastly, PRS and DCC were not used in this study as, at the time that this analysis was performed, they were not yet part of the tool kit. However, David Penkler, the PRS tool developer, has since published a study using PRS [247] and DCC is a commonly used measurement in MD studies.

Part 4: Conclusion & references

9. Conclusion

The thesis was divided into 3 main parts. In part 1, JMS, a web-based workflow management system and HPC cluster front-end, was described. JMS was developed to cater to four groups of users. Firstly, by exposing all functionality via a RESTful web API, it caters to developers who are building their own web servers. It does this by allowing developers to leverage the JMS API to run tools and workflows on the underlying cluster, without needing to "reinvent the wheel" by building their own job submission systems. This was the initial purpose of JMS and is being put into practice by PRIMO, SANCDB, HUMA and, in future, the MD-TASK web server.

Secondly, JMS caters to tools and workflow developers. JMS provides a user-friendly interface, including a built-in code editor and version management, for tool developers, making it easy for them to create and edit tools. Workflow developers, on the other hand, are users who would like to pipeline various existing tools to perform an analysis. HUMA caters to these users by allowing them to create workflows via a drag-and-drop interface. JMS will then generate an interface for the workflow and manage its execution on the underlying cluster. All of this can be done without requiring any programming from the user. VAPOR is an example of this kind of workflow.

Thirdly, JMS caters to tool and workflow *users* by automatically generating web-based interfaces to tools to allow them to be easily submitted to the cluster. No CLI experience is required.

And lastly, JMS caters to system administrators by allowing them monitor and manage (to a degree) the underlying cluster.

Future work on JMS is currently focused on building up a large library of tools and workflows, as well to make these tools and workflows more portable *i.e.* allow them to be transferred easily from one JMS instance to another, running on a different cluster. Additionally, a JMS instance will be set up at the CHPC, where it will be accessible to users all over South Africa.

Part 2 of this thesis was focused on the development of the HUMA database and web server. HUMA aggregates data from various existing databases into a single, connected and related database. It also provides a user-friendly web interface from which users can query this data. The advantage of having all this data in a single database is realized via this querying. Users can, for example, search for a protein. The data returned from this search will include the protein details, sequences and structures, domains and families, variation, genes that code the protein, diseases that have been linked to the protein, and literature related to the protein. All the original sources of this data are also linked to, allowing users to follow the links back to the original source to get more details.

HUMA also provides tools to visualize and analyze this data. Sequences and structures can be visualized via the PV-MSA JavaScript plugin, VAPOR has been integrated into the interface to analyze variation, PRIMO has been integrated to provide homology modeling functionality, and Protein BLAST can be used to search the database. Additionally, users can upload their own variation datasets, which automatically get mapped to proteins, genes, and diseases. They can then analyze this variation using the above-mentioned tools.

The HUMA web server was developed with the H3Africa consortium in mind. To this end, collaboration features were also added. Users can create groups and invite other users to those groups, as well as share datasets and job results with those groups. A discussion forum is also provided where group members can hold discussions on certain topics. As such, HUMA is more than simply a database and web server, but rather, it is a platform for the analysis of genetic variation in humans.

Part 3 of this thesis described MD-TASK and provided a case study for how it, along with HUMA and VAPOR, could be used to analyze variation at the protein structure level. MD-TASK is a tool suite for analyzing MD trajectories using network analysis, PRS and DCC. In the case study, the renin-angiotensinogen complex, a vital part of RAS, was analyzed by retrieving the relevant sequences, structures and variation from HUMA, filtering the initial variation using, amongst other things, VAPOR, modeling that variants from the filtered dataset into the protein structure, and then subjecting the wild type and variant complexes to 100ns MD simulations. Finally, the MD trajectories were analyzed using RMSD, RMSF, and, from MD-TASK, $\Delta L/L$, ΔBC, and residue contact maps. Aside from identifying some potentially important residues and variants, the case study showed that the measurements tracked by MD-TASK can provided useful insights over-and-above those obtained from traditional measures such as RMSD and RMSF.

The main aim of this thesis was to provide tools to enable bioinformatics research in Africa. Although the thesis mostly focused on structural bioinformatics, many of the tools developed have applications far beyond that. For example, JMS provides a platform for researchers to develop any type of tool and host it on HPC infrastructure. Additionally, although HUMA is currently focused on structural bioinformatics, it stores data about genes as well as non-coding variants. In future, this will allow the uses of the web server to expand beyond structural bioinformatics.

10. References

1. Eijkhout, V., Chow, E. & van de Geijn, R. *Introduction to High Performance Scientific Computing*. (2014).

2. Hazelhurst, S. Scientific computing using virtual high-performance computing : a case study using the Amazon Elastic Computing Cloud. *Structure* **338,** 94–103 (2008).

3. Sadashiv, N. & Kumar, S. M. D. Cluster, grid and cloud computing: A detailed comparison. *ICCSE 2011 - 6th Int. Conf. Comput. Sci. Educ. Final Progr. Proc.* 477–482 (2011). doi:10.1109/ICCSE.2011.6028683

4. Omer, S. M. I., Mustafa, A. B. A. & Alghali, F. A. E. Comparative study between Cluster, Grid, Utility, Cloud and Autonomic computing. *IOSR J. Electr. Electron. Eng.* **9,** 61–67 (2014).

5. Kaur, K. & Rai, A. K. A Comparative Analysis : Grid, Cluster and Cloud Computing. *Int. J. Adv. Res. Comput. Commun. Eng.* **3,** 5730–5734 (2014).

6. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. & Brandic, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Futur. Gener. Comput. Syst.* **25,** 599–616 (2009).

7. Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J. & Ghalsasi, A. Cloud computing — The business perspective. *Decis. Support Syst.* **51,** 176–189 (2011).

8. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. & Brandic, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility.

Futur. Gener. Comput. Syst. **25,** 599–616 (2009).

9. Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J. & Ghalsasi, A. Cloud computing — The business perspective. *Decis. Support Syst.* **51,** 176–189 (2011).

10. Perez-Sanchez, H., Cecilia, J. M. & Merelli, I. The role of High Performance Computing in Bioinformatics. in *International Work-Conference on Bioinformatics and Biomedical Engineering* 494–506 (2014).

11. Sali, A. & Blundell, T. L. Comparative protein modelling by satisfaction of spatial restraints. *J. Mol. Biol.* **234,** 779–815 (1993).

12. Eswar, N. *et al.* Comparative protein structure modeling using MODELLER. *Curr. Protoc. Protein Sci.* **Chapter 2,** Unit 2.9 (2007).

13. Song, Y. *et al.* High-resolution comparative modeling with RosettaCM. *Structure* **21,** 1735–1742 (2013).

14. Schwede, T., Kopp, J., Guex, N. & Peitsch, M. C. SWISS-MODEL: An automated protein homology-modeling server. *Nucleic Acids Res.* **31,** 3381–5 (2003).

15. Guex, N. & Peitsch, M. C. SWISS-MODEL and the Swiss-PdbViewer: an environment for comparative protein modeling. *Electrophoresis* **18,** 2714–23 (1997).

16. Biasini, M. *et al.* SWISS-MODEL: Modelling protein tertiary and quaternary structure using evolutionary information. *Nucleic Acids Res.* **42,** 252–258 (2014).

17. Krieger, E. & Vriend, G. Models@Home: distributed computing in bioinformatics using a screensaver based approach. *Bioinformatics* **18,** 315–318 (2002).

18. Alejandra Hernández-Santoyo, Aldo Yair Tenorio-Barajas, Victor Altuzar, H. V.-C. and C. M.-B. Protein-Protein and Protein-Ligand Docking. *Protein-Protein and Protein-Ligand Docking* 64–81 (2013). doi:http://dx.doi.org/10.5772/56376

19. Kitchen, D. B., Decornez, H., Furr, J. R. & Bajorath, J. Docking and scoring in virtual screening for drug discovery: methods and applications. *Nat. Rev. Drug Discov.* **3,** 935–949 (2004).

20. Morris, G. M. *et al.* AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *J. Comput. Chem.* **30,** 2785–2791 (2009).

21. Norgan, A. P., Coffman, P. K., Kocher, J. P. a, Katzmann, D. J. & Sosa, C. P. Multilevel parallelization of autodock 4.2. *J. Cheminform.* **3,** 1–9 (2011).

22. Khodade, P., Prabhu, R., Chandra, N., Raha, S. & Govindarajan, R. Parallel implementation of AutoDock. *J. Appl. Crystallogr.* **40,** 598–599 (2007).

23. Collignon, B., Schulz, R., Smith, J. C. & Baudry, J. Task-parallel message passing interface implementation of Autodock4 for docking of very large databases of compounds using high-performance super-computers. *J. Comput. Chem.* **32,** 1202–9 (2011).

24. Trott, O. & Olson, A. J. AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *J. Comput. Chem.* **31,** 455–461 (2010).

25. Chang, M. W., Ayeni, C., Breuer, S. & Torbett, B. E. Virtual screening for HIV protease inhibitors: A comparison of AutoDock 4 and Vina. *PLoS One* **5,** 1–9 (2010).

26. Zhang, S., Kumar, K., Jiang, X., Wallqvist, A. & Reifman, J. DOVIS: an implementation

for high-throughput virtual screening using AutoDock. *BMC Bioinformatics* **9,** 126 (2008).

27. Abreu, R. M. V, Froufe, H. J. C., Queiroz, M. J. R. P. & Ferreira, I. C. F. R. MOLA: A bootable, self-configuring system for virtual screening using AutoDock4/Vina on computer clusters. *J. Cheminform.* **2,** 10 (2010).

28. Gotz, A. W. *et al.* Routine microsecond molecular dynamics simulations with amber - part i: Generalized born. *J. Chem. Theory Comput.* **8,** 1542–1555. (2012).

29. Trott, C. R., Winterfeld, L. & Crozier, P. S. General-purpose molecular dynamics simulations on GPU-based clusters. *Cell* **1009,** 12 (2010).

30. Baker, J. a. & Hirst, J. D. Molecular dynamics simulations using graphics processing units. *Mol. Inform.* **30,** 498–504 (2011).

31. Abraham, M. J. *et al.* GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **1,** 19–25 (2015).

32. Kawalia, A. *et al.* Leveraging the power of high performance computing for next generation sequencing data analysis: tricks and twists from a high throughput exome workflow. *PLoS One* **10,** e0126321 (2015).

33. Puckelwartz, M. J. *et al.* Supercomputing for the parallelization of whole genome analysis. *Bioinformatics* **30,** 1508–1513 (2014).

34. Dean, J. & Ghemawat, S. MapReduce: Simplied Data Processing on Large Clusters. *Proc. 6th Symp. Oper. Syst. Des. Implement.* 137–149 (2004). doi:10.1145/1327452.1327492

35. Neves, M. V., Ferreto, T. & De Rose, C. Scheduling MapReduce jobs in HPC clusters. *Lect.*

Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) **7484 LNCS,** 179–190 (2012).

36. Leo, S., Santoni, F. & Zanetti, G. Biodoop: Bioinformatics on hadoop. *Proc. Int. Conf. Parallel Process. Work.* 415–422 (2009). doi:10.1109/ICPPW.2009.37

37. Hazelhurst, S. Scientific computing using virtual high-performance computing : a case study using the Amazon Elastic Computing Cloud. *Structure* **338,** 94–103 (2008).

38. Seemann, T. Ten recommendations for creating usable bioinformatics command line software. *Gigascience* **2,** 15 (2013).

39. Gil, Y. *et al.* Examining the challenges of scientific workflows. *Computer (Long. Beach. Calif).* **40,** 24–32 (2007).

40. Bux, M. & Leser, U. Parallelization in Scientific Workflow Management Systems. *arXiv Prepr. arXiv1303.7195* 24 (2013).

41. Gil, Y., González-Calero, P. A. & Deelman, E. On the black art of designing computational workflows. *High Perform. Distrib. Comput.* (2007). doi:10.1145/1273360.1273370

42. Taylor, I., Deelman, E., Gannon, D. & Shields, M. S. *Workflows for e-Science: Scientific Workflows for Grids. Workflows for e-Science: Scientific Workflows for Grids* (2007). doi:10.1007/978-1-84628-757-2

43. Taylor, I., Deelman, E., Gannon, D. & Shields, M. S. *Workflows for e-Science: Scientific Workflows for Grids. Workflows for e-Science: Scientific Workflows for Grids* (2007). doi:10.1007/978-1-84628-757-2

44. Yu, J. & Buyya, R. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Rec.* **34,** 44 (2005).

45. Altintas, I. *et al.* Kepler: an extensible system for design and execution of scientific workflows. in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on* 423–424 (2004). doi:10.1109/SSDM.2004.1311241

46. Giardine, B. *et al.* Galaxy: A platform for interactive large-scale genome analysis. *Genome Res.* **15,** 1451–1455 (2005).

47. Amstutz, P. & Tijanić, N. Common Workflow Language, Draft 2. (2015).

48. Di Tommaso, P., Chatzou, M., Prieto, P., Palumbo, E. & Notredame, C. Nextflow: A tool for deploying reproducible computational pipelines. *F1000Research* **4,** 430 (2015).

49. Van Der Aalst, W. M. P. & Ter Hofstede, a. H. M. YAWL: Yet another workflow language. *Inf. Syst.* **30,** 245–275 (2005).

50. Hull, D. *et al.* Taverna: a tool for building and running workflows of services. *Nucleic Acids Res.* **34,** W729-32 (2006).

51. Wolstencroft, K. *et al.* The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Res.* **41,** (2013).

52. Smedley, D. *et al.* BioMart--biological queries made easy. *BMC Genomics* **10,** 22 (2009).

53. Curcin, V. *et al.* Discovery Net: Towards a Grid of Knowledge Discovery. *Knowl. Discov.* 658–663 (2002). doi:http://doi.acm.org/10.1145/775047.775145

54. Romano, P. *et al.* Biowep: a workflow enactment portal for bioinformatics applications.

BMC Bioinformatics **8 Suppl 1,** S19 (2007).

55. Taylor, I., Shields, M., Wang, I. & Harrison, A. Visual Grid Workflow in Triana. *J. Grid Comput.* **3,** 153–169 (2006).

56. Ovaska, K. *et al.* Large-scale data integration framework provides a comprehensive view on glioblastoma multiforme. *Genome Med.* **2,** 65 (2010).

57. Orvis, J. *et al.* Ergatis: A web interface and scalable software system for bioinformatics workflows. *Bioinformatics* **26,** 1488–1492 (2010).

58. Goecks, J., Nekrutenko, A. & Taylor, J. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.* **11,** R86 (2010).

59. Abouelhoda, M., Issa, S. a & Ghanem, M. Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. *BMC Bioinformatics* **13,** 77 (2012).

60. Gafni, E. *et al.* COSMOS: Python library for massively parallel workflows. *Bioinformatics* **30,** 1–3 (2014).

61. Brown, D. K. & Tastan Bishop, Ö. HUMA: a platform for the analysis of genetic variation in humans. *Hum. Mutat.* (2017). doi:10.1002/humu.23334

62. Brown, D. K., Penkler, D. L., Musyoka, T. M. & Bishop, Ö. T. JMS: An Open Source Workflow Management System and Web-Based Cluster Front-End for High Performance Computing. *PLoS One* **10,** e0134273 (2015).

63. The Web framework for perfectionists with deadlines | Django. (2016). Available at:

https://www.djangoproject.com/.

64. Django REST Framework. (2016). Available at: http://www.django-rest-framework.org.

65. Kinder, K. Event-driven programming with Twisted and Python. *Linux J.* **2005,** 6 (2005).

66. SQLite Home Page. (2016). Available at: https://www.sqlite.org.

67. MySQL. (2016). Available at: https://www.mysql.com.

68. Holovaty, A. & Kaplan-Moss, J. *The Definitive Guide to Django: Web Development Done Right. Development* (2009). doi:10.1093/intimm/dxu027

69. Grove, R. F. & Ozkan, E. The MVC-web design pattern. *WEBIST 2011 - Proc. 7th Int. Conf. Web Inf. Syst. Technol.* 127–130 (2011).

70. Gorlick, M. M. & Taylor, R. N. *REST: Advanced Research Topics and Practical Applications. REST: Advanced Research Topics and Practical Applications* (2014). doi:10.1007/978-1-4614-9299-3

71. Fielding, R. *et al.* RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1. *Society* 1–114 (1999). doi:http://www.ietf.org/rfc/rfc2616.txt

72. Eckerson, W. W. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. in *Open Information Systems 10* (1995).

73. Knockout.js. (2015). Available at: http://knockoutjs.com/.

74. Bootstrap. (2015). Available at: http://getbootstrap.com/.

75. Frain, B. *Responsive Web Design with HTML5 and CSS3. Internetdocument* (2012). doi:10.11635/2319-9954/1/1/18

76. Ansible. Ansible is Simple IT Automation. (2015). Available at: http://www.ansible.com/.

77. Conda - Conda documentation. (2016). Available at: http://conda.pydata.org/docs/.

78. Sharma, P. & Mantri, S. S. WImpiBLAST: Web interface for mpiBLAST to help biologists perform large-scale annotation using high performance computing. *PLoS One* **9,** (2014).

79. Hunter, A. A., Macgregor, A. B., Szabo, T. O., Wellington, C. A. & Bellgard, M. I. Yabi: An online research environment for grid, high performance and cloud computing. *Source Code for Biology and Medicine* **7,** 1 (2012).

80. Ace - The High Performance Code Editor for the Web. (2016). Available at: https://ace.c9.io.

81. Merkel, D. Docker: lightweight Linux containers for consistent development and deployment. *Linux J.* **2014,** 2 (2014).

82. Hatherley, R. *et al.* SANCDB: a South African natural compound database. *J. Cheminform.* **7,** 29 (2015).

83. Hatherley, R., Brown, D. K., Glenister, M. & Tastan Bishop, Ö. PRIMO: An Interactive Homology Modeling Pipeline. *PLoS One* **11,** e0166698 (2016).

84. Gasteiger, J., Rudolph, C. & Sadowski, J. Automatic generation of 3D-atomic coordinates for organic molecules. *Tetrahedron Comput. Methodol.* **3,** 537–547 (1990).

85. O'Boyle, N. M. *et al.* Open Babel: An Open chemical toolbox. *J. Cheminform.* **3,** (2011).

86. Schmidt, M. W. *et al.* General atomic and molecular electronic structure system. *J. Comput. Chem.* **14,** 1347–1363 (1993).

87. Illergård, K., Ardell, D. H. & Elofsson, A. Structure is three to ten times more conserved than sequence--a study of structural response in protein cores. *Proteins* **77,** 499–508 (2009).

88. Kelley, L. A., Mezulis, S., Yates, C. M., Wass, M. N. & Sternberg, M. J. E. The Phyre2 web portal for protein modeling, prediction and analysis. *Nat. Protoc.* **10,** 845–858 (2015).

89. Söding, J., Biegert, A. & Lupas, A. N. The HHpred interactive server for protein homology detection and structure prediction. *Nucleic Acids Res.* **33,** (2005).

90. Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *J. Mol. Biol.* **215,** 403–410 (1990).

91. Shi, J., Blundell, T. L. & Mizuguchi, K. FUGUE: sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *J. Mol. Biol.* **310,** 243–257 (2001).

92. Altschul, S. F. *et al.* Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research* **25,** 3389–3402 (1997).

93. Notredame, C., Higgins, D. G. & Heringa, J. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.* **302,** 205–217 (2000).

94. Katoh, K. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res.* **30,** 3059–3066 (2002).

95. Edgar, R. C. & Edgar, R. C. MUSCLE: multiple sequence alignment with high accuracy

and high throughput. *Nucleic Acids Res.* **32,** 1792–7 (2004).

96. Sievers, F. *et al.* Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology* **7,** (2011).

97. O'Sullivan, O., Suhre, K., Abergel, C., Higgins, D. G. & Notredame, C. 3DCoffee: Combining protein sequences and structures within multiple sequence alignments. *J. Mol. Biol.* **340,** 385–395 (2004).

98. Biasini, M. pv: v1.8.1. *Zenodo* (2015). doi:10.5281/zenodo.20980

99. BioJS MSA Viewer. *2016* Available at: http://msa.biojs.net/.

100. Laskowski, R. A., MacArthur, M. W., Moss, D. S. & Thornton, J. M. PROCHECK: a program to check the stereochemical quality of protein structures. *Journal of Applied Crystallography* **26,** 283–291 (1993).

101. Wiederstein, M. & Sippl, M. J. ProSA-web: Interactive web service for the recognition of errors in three-dimensional structures of proteins. *Nucleic Acids Res.* **35,** (2007).

102. Benkert, P., Tosatto, S. C. E. & Schomburg, D. QMEAN: A comprehensive scoring function for model quality assessment. *Proteins Struct. Funct. Bioinforma.* **71,** 261–277 (2008).

103. Eisenberg, D., Lüthy, R. & Bowie, J. U. VERIFY3D: Assessment of protein models with three-dimensional profiles. *Methods Enzymol.* **277,** 396–406 (1997).

104. Capriotti, E., Altman, R. B. & Bromberg, Y. Collective judgment predicts disease-associated single nucleotide variants. *BMC Genomics* **14 Suppl 3,** S2 (2013).

105. Bendl, J. *et al.* PredictSNP: Robust and Accurate Consensus Classifier for Prediction of Disease-Related Mutations. *PLoS Comput. Biol.* **10,** 1–11 (2014).

106. Thomas, P. D. & Kejariwal, A. Coding single-nucleotide polymorphisms associated with complex vs. Mendelian disease: evolutionary evidence for differences in molecular effects. *Proc. Natl. Acad. Sci. U. S. A.* **101,** 15398–403 (2004).

107. Ng, P. C. & Henikoff, S. SIFT: Predicting amino acid changes that affect protein function. *Nucleic Acids Res.* **31,** 3812–3814 (2003).

108. Capriotti, E., Calabrese, R. & Casadio, R. Predicting the insurgence of human genetic diseases associated to single point protein mutations with support vector machines and evolutionary information. *Bioinformatics* **22,** 2729–2734 (2006).

109. Johnson, A. D. *et al.* SNAP: A web-based tool for identification and annotation of proxy SNPs using HapMap. *Bioinformatics* **24,** 2938–2939 (2008).

110. Stone, E. A. & Sidow, A. Physicochemical constraint violation by missense substitutions mediates impairment of protein function and disease severity. *Genome Res.* **15,** 978–986 (2005).

111. Ramensky, V., Bork, P. & Sunyaev, S. Human non-synonymous SNPs: server and survey. *Nucleic Acids Res.* **30,** 3894–3900 (2002).

112. Adzhubei, I., Jordan, D. M. & Sunyaev, S. R. Predicting functional effect of human missense mutations using PolyPhen-2. *Curr. Protoc. Hum. Genet.* (2013). doi:10.1002/0471142905.hg0720s76

113. Bao, L., Zhou, M. & Cui, Y. nsSNPAnalyzer: Identifying disease-associated

nonsynonymous single nucleotide polymorphisms. *Nucleic Acids Res.* **33,** (2005).

114. Apweiler, R. UniProt: the Universal Protein knowledgebase. *Nucleic Acids Res.* **32,** 115D–119 (2004).

115. Choi, Y., Sims, G. E., Murphy, S., Miller, J. R. & Chan, A. P. Predicting the Functional Effect of Amino Acid Substitutions and Indels. *PLoS One* **7,** e46688 (2012).

116. Adzhubei, I. A. *et al.* A method and server for predicting damaging missense mutations. *Nat. Methods* **7,** 248–249 (2010).

117. Tang, H. & Thomas, P. D. PANTHER-PSEP: predicting disease-causing genetic variants using position-specific evolutionary preservation. *Bioinformatics* 1–3 (2016). doi:10.1093/bioinformatics/btw222

118. Tian, J. *et al.* Predicting the phenotypic effects of non-synonymous single nucleotide polymorphisms based on support vector machines. *BMC Bioinformatics* **8,** 450 (2007).

119. Shihab, H. A. *et al.* Predicting the Functional, Molecular, and Phenotypic Consequences of Amino Acid Substitutions using Hidden Markov Models. *Hum. Mutat.* **34,** 57–65 (2013).

120. Capriotti, E., Fariselli, P. & Casadio, R. I-Mutant2.0: predicting stability changes upon mutation from the protein sequence or structure. *Nucleic Acids Res.* **33,** W306–W310 (2005).

121. Cheng, J., Randall, A. & Baldi, P. Prediction of protein stability changes for single-site mutations using support vector machines. *Proteins* **62,** 1125–32 (2006).

122. Musyoka, T. M., Kanzi, A. M., Lobb, K. A. & Tastan Bishop, Ö. Analysis of Non-Peptidic

Compounds as Potential Malarial Inhibitors against *Plasmodial* Cysteine Proteases via Integrated Virtual Screening Workflow. *J. Biomol. Struct. Dyn.* **1102,** 1–72 (2015).

123. Morris, G. M. *et al.* AutoDock4 and AutoDockTools4: automated docking with selective receptor flexibility. *J. Comput. Chem.* **30,** 2785–2791 (2009).

124. Pronk, S. *et al.* GROMACS 4.5: A high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* **29,** 845–854 (2013).

125. Sousa da Silva, A. W. & Vranken, W. F. ACPYPE - AnteChamber PYthon Parser interfacE. *BMC Res. Notes* **5,** 367 (2012).

126. Mardis, E. R. A decade's perspective on DNA sequencing technology. *Nature* **470,** 198–203 (2011).

127. The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature* **526,** 68–74 (2015).

128. The International Hapmap Consortium. The International HapMap Project. *Nature* **426,** 789–796 (2003).

129. Gonzalez-Garay, M. L. The road from next-generation sequencing to personalized medicine. *Per Med* **11,** 523–544 (2014).

130. The H3Africa Consortium. Research capacity. Enabling the genomic revolution in Africa. *Science* **344,** 1346–8 (2014).

131. Brown, D. K. & Tastan Bishop, Ö. Role of Structural Bioinformatics in Drug Discovery by Computational SNP Analysis. *Glob. Heart* **12,** 151–161 (2017).

132. Bamford, S. *et al.* The COSMIC (Catalogue of Somatic Mutations in Cancer) database and website. *Br. J. Cancer* **91,** 355–8 (2004).

133. Sherry, S. T. *et al.* dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res.* **29,** 308–311 (2001).

134. Lappalainen, I. *et al.* The European Genome-phenome Archive of human data consented for biomedical research. *Nat. Genet.* **47,** 692–695 (2015).

135. Lek, M. *et al.* Analysis of protein-coding genetic variation in 60,706 humans. *Nature* **536,** 285–291 (2016).

136. Higasa, K. *et al.* Human genetic variation database, a reference database of genetic variations in the Japanese population. *J. Hum. Genet.* **61,** 547–553 (2016).

137. Welter, D. *et al.* The NHGRI GWAS Catalog, a curated resource of SNP-trait associations. *Nucleic Acids Res.* **42,** D1001–D1006 (2014).

138. Cancer, T. & Atlas, G. Comprehensive genomic characterization defines human glioblastoma genes and core pathways. *Nature* **455,** 1061–8 (2008).

139. Lappalainen, I. *et al.* dbVar and DGVa: public archives for genomic structural variation. *Nucleic Acids Res.* **41,** D936–D941 (2013).

140. Landrum, M. J. *et al.* ClinVar: public archive of relationships among sequence variation and human phenotype. *Nucleic Acids Res.* **42,** D980–D985 (2014).

141. Mailman, M. D. *et al.* The NCBI dbGaP database of genotypes and phenotypes. *Nat. Genet.* **39,** 1181–6 (2007).

142. Liu, X., Jian, X. & Boerwinkle, E. dbNSFP: A lightweight database of human nonsynonymous SNPs and their functional predictions. *Hum. Mutat.* **32,** 894–899 (2011).

143. Liu, X., Jian, X. & Boerwinkle, E. dbNSFP v2.0: A database of human non-synonymous SNVs and their functional predictions and annotations. *Hum. Mutat.* **34,** (2013).

144. Liu, X., Wu, C., Li, C. & Boerwinkle, E. dbNSFP v3.0: A One-Stop Database of Functional Predictions and Annotations for Human Nonsynonymous and Splice-Site SNVs. *Hum. Mutat.* **37,** 235–241 (2016).

145. Ryan, M., Diekhans, M., Lien, S., Liu, Y. & Karchin, R. LS-SNP/PDB: Annotated non-synonymous SNPs mapped to Protein Data Bank structures. *Bioinformatics* **25,** 1431–1432 (2009).

146. Lu, H.-C., Herrera Braga, J. & Fraternali, F. PinSnps: structural and functional analysis of SNPs in the context of protein interaction networks. *Bioinformatics* **32,** 2534–2536 (2016).

147. Reumers, J. *et al.* SNPeffect: a database mapping molecular phenotypic effects of human non-synonymous coding SNPs. *Nucleic Acids Res.* **33,** D527–D532 (2005).

148. Yue, P., Melamud, E. & Moult, J. SNPs3D: candidate gene and SNP selection for association studies. *BMC Bioinformatics* **7,** 166 (2006).

149. Hubbard, T. *et al.* The Ensembl genome database project. *Nucleic Acids Res.* **30,** 38–41 (2002).

150. Stenson, P. D. *et al.* The Human Gene Mutation Database: building a comprehensive mutation repository for clinical and molecular genetics, diagnostic testing and personalized genomic medicine. *Hum. Genet.* **133,** 1–9 (2014).

151. Hamosh, A., Scott, A. F., Amberger, J., Valle, D. & McKusick, V. A. Online Mendelian Inheritance in Man (OMIM). *Hum. Mutat.* **15,** 57–61 (2000).

152. Yang, J. O. *et al.* VnD: a structure-centric database of disease-related SNPs and drugs. *Nucleic Acids Res.* **39,** D939–D944 (2011).

153. Hanson, R. M., Prilusky, J., Renjian, Z., Nakane, T. & Sussman, J. L. JSmol and the next-generation web-based representation of 3D molecular structure as applied to proteopedia. *Israel Journal of Chemistry* **53,** 207–216 (2013).

154. Mah, J. T. L., Low, E. S. H. & Lee, E. In silico SNP analysis and bioinformatics tools: A review of the state of the art to aid drug discovery. *Drug Discovery Today* **16,** 800–809 (2011).

155. Masso, M. & Vaisman, I. I. AUTO-MUTE 2.0: A portable framework with enhanced capabilities for predicting protein functional consequences upon mutation. *Adv. Bioinformatics* **2014,** (2014).

156. Kircher, M. *et al.* A general framework for estimating the relative pathogenicity of human genetic variants. *Nat. Genet.* **46,** 310–315 (2014).

157. McLaren, W. *et al.* The Ensembl Variant Effect Predictor. *Genome Biol.* **17,** 122 (2016).

158. Wainreb, G. *et al.* MuD: an interactive web server for the prediction of non-neutral substitutions using protein structural data. *Nucleic Acids Res.* **38,** W523–W528 (2010).

159. Reva, B., Antipin, Y. & Sander, C. Predicting the functional impact of protein mutations: application to cancer genomics. *Nucleic Acids Res.* **39,** e118–e118 (2011).

160. Schwarz, J. M., Cooper, D. N., Schuelke, M. & Seelow, D. Mutationtaster2: Mutation prediction for the deep-sequencing age. *Nature Methods* **11,** 361–362 (2014).

161. Li, B. *et al.* Automated inference of molecular mechanisms of disease from amino acid substitutions. *Bioinformatics* **25,** 2744–2750 (2009).

162. Ioannidis, N. M. *et al.* REVEL: An Ensemble Method for Predicting the Pathogenicity of Rare Missense Variants. *Am. J. Hum. Genet.* **99,** 877–885 (2016).

163. Bromberg, Y. & Rost, B. SNAP: Predict effect of non-synonymous polymorphisms on function. *Nucleic Acids Res.* **35,** 3823–3835 (2007).

164. Calabrese, R., Capriotti, E., Fariselli, P., Martelli, P. L. & Casadio, R. Functional annotations improve the predictive score of human disease-related mutations in proteins. *Hum. Mutat.* **30,** 1237–1244 (2009).

165. Thiltgen, G. & Goldstein, R. A. Assessing Predictors of Changes in Protein Stability upon Mutation Using Self-Consistency. *PLoS One* **7,** (2012).

166. Parthiban, V., Gromiha, M. M. & Schomburg, D. CUPSAT: prediction of protein stability upon point mutations. *Nucleic Acids Res.* **34,** W239–W242 (2006).

167. Yin, S., Ding, F. & Dokholyan, N. V. Eris: an automated estimator of protein stability. *Nat. Methods* **4,** 466–467 (2007).

168. Giollo, M., Martin, A. J., Walsh, I., Ferrari, C. & Tosatto, S. C. NeEMO: a method using residue interaction networks to improve prediction of protein stability upon mutation. *BMC Genomics* **15,** S7 (2014).

169. Dehouck, Y., Kwasigroch, J. M., Gilis, D. & Rooman, M. PoPMuSiC 2.1: a web server for the estimation of protein stability changes upon mutation and sequence optimality. *BMC Bioinformatics* **12,** 151 (2011).

170. Sim, S., Kacevska, M. & Ingelman-Sundberg, M. Pharmacogenomics of drug-metabolizing enzymes: a recent update on clinical implications and endogenous effects. *Pharmacogenomics J.* **13,** 1–11 (2012).

171. Casali, N. *et al.* Evolution and transmission of drug-resistant tuberculosis in a Russian population. *Nat. Genet.* **46,** 279–86 (2014).

172. Gottesman, M. M. Mechanisms of Cancer Drug Resistance. *Annu. Rev. Med.* **53,** 615–627 (2002).

173. LI, J. *et al.* Sensitive sentinel mutation screening reveals differential underestimation of transmitted HIV drug resistance among demographic groups. *Aids* 1 (2016). doi:10.1097/QAD.0000000000001099

174. Pielak, R. M., Schnell, J. R. & Chou, J. J. Mechanism of drug inhibition and drug resistance of influenza A M2 channel. *Proc. Natl. Acad. Sci. U. S. A.* **106,** 7379–84 (2009).

175. Garnett, M. J. *et al.* Systematic identification of genomic markers of drug sensitivity in cancer cells. *Nature* **483,** 570–575 (2012).

176. Kumar, R. D., Chang, L. W., Ellis, M. J. & Bose, R. Prioritizing Potentially Druggable Mutations with dGene: An Annotation Tool for Cancer Genome Sequencing Data. *PLoS One* **8,** (2013).

177. Niu, B. *et al.* Protein-structure-guided discovery of functional mutations across 19 cancer

types. *Nat Genet* **48,** 827–837 (2016).

178. Kapetanovic, I. M. Computer-aided drug discovery and development (CADDD): In silico-chemico-biological approach. *Chem. Biol. Interact.* **171,** 165–176 (2008).

179. Chou, K.-C. Impacts of bioinformatics to medicinal chemistry. *Med. Chem. (Los. Angeles).* **11,** 218–34 (2015).

180. Blundell, T. L. *et al.* Structural biology and bioinformatics in drug design: opportunities and challenges for target identification and lead discovery. *Philos. Trans. R. Soc. Lond. B. Biol. Sci.* **361,** 413–23 (2006).

181. Taboureau, O., Baell, J. B., Fernández-Recio, J. & Villoutreix, B. O. Established and emerging trends in computational drug discovery in the structural genomics era. *Chemistry and Biology* **19,** 29–41 (2012).

182. Cavasotto, C. N. & Phatak, S. S. Homology modeling in drug discovery: current trends and applications. *Drug Discovery Today* **14,** 676–683 (2009).

183. Scapin, G. Structural biology and drug discovery. *Curr. Pharm. Des.* **12,** 2087–2097 (2006).

184. Congreve, M., Murray, C. W. & Blundell, T. L. Structural biology and drug discovery. *Drug Discov. Today* **10,** 895–907 (2005).

185. Durrant, J. D. & McCammon, J. A. Molecular dynamics simulations and drug discovery. *BMC Biol.* **9,** 71 (2011).

186. Berman, H. M. *et al.* The Protein Data Bank. *Nucleic Acids Res.* **28,** 235–242 (2000).

187. Jacobson, M. & Sali, A. Comparative Protein Structure Modeling and its Applications to

Drug Discovery. *Annual Reports in Medicinal Chemistry* **39,** 259–276 (2004).

188. Ma, J., Wang, S., Zhao, F. & Xu, J. Protein threading using context-specific alignment potential. in *Bioinformatics* **29,** (2013).

189. Moult, J., Fidelis, K., Kryshtafovych, A., Schwede, T. & Tramontano, A. Critical assessment of methods of protein structure prediction: Progress and new directions in round XI. *Proteins Struct. Funct. Bioinforma.* n/a-n/a (2016). doi:10.1002/prot.25064

190. Chen, M., Lin, X., Zheng, W., Onuchic, J. N. & Wolynes, P. G. Protein Folding and Structure Prediction from the Ground Up: The Atomistic Associative Memory, Water Mediated, Structure and Energy Model. *J. Phys. Chem. B* **120,** 8557–8565 (2016).

191. Kantardjieff, K. & Rupp, B. Structural bioinformatic approaches to the discovery of new antimycobacterial drugs. *Curr. Pharm. Des.* **10,** 3195–3211 (2004).

192. Petrey, D. *et al.* Template-based prediction of protein function. *Curr. Opin. Struct. Biol.* **32,** 33–38 (2015).

193. Blair, J. M. A. *et al.* AcrB drug-binding pocket substitution confers clinically relevant resistance and altered substrate specificity. *Proc. Natl. Acad. Sci.* **112,** 3511–3516 (2015).

194. Vyas, V. K., Ghate, M., Patel, K., Qureshi, G. & Shah, S. Homology modeling, binding site identification and docking study of human angiotensin II type I (Ang II-AT1) receptor. *Biomed. Pharmacother.* **74,** 42–48 (2015).

195. Messaoudi, A., Belguith, H. & Ben Hamida, J. Homology modeling and virtual screening approaches to identify potent inhibitors of VEB-1 β-lactamase. *Theor. Biol. Med. Model.* **10,** 22 (2013).

196. Ung, P. M.-U. *et al.* Inhibitor Discovery for the Human GLUT1 from Homology Modeling and Virtual Screening. *ACS Chem. Biol.* **11,** 1908–1916 (2016).

197. Morya, V. K., Dung, N. H., Singh, B. K., Lee, H.-B. & Kim, E. Homology modelling and virtual screening of P-protein in a quest for novel antimelanogenic agent and In vitro assessments. *Exp. Dermatol.* **23,** 838–842 (2014).

198. Fazi, R. *et al.* Homology Model-Based Virtual Screening for the Identification of Human Helicase DDX3 Inhibitors. *J. Chem. Inf. Model.* **55,** 2443–2454 (2015).

199. Forli, S. *et al.* Computational protein-ligand docking and virtual drug screening with the AutoDock suite. *Nat. Protoc.* **11,** 905–919 (2016).

200. Irwin, J. J. & Shoichet, B. K. Docking Screens for Novel Ligands Conferring New Biology. *J. Med. Chem.* **59,** 4103–4120 (2016).

201. Pyzer-Knapp, E. O., Suh, C., Gómez-Bombarelli, R., Aguilera-Iparraguirre, J. & Aspuru-Guzik, A. What Is High-Throughput Virtual Screening? A Perspective from Organic Materials Discovery. *Annu. Rev. Mater. Res.* **45,** 195–216 (2015).

202. Kumar, V., Krishna, S. & Siddiqi, M. I. Virtual screening strategies: Recent advances in the identification and design of anti-cancer agents. *Methods* **71,** 64–70 (2015).

203. Lyne, P. D. Structure-based virtual screening: an overview. *Drug Discov. Today* **7,** 1047–1055 (2002).

204. Irwin, J. J. & Shoichet, B. K. ZINC - A free database of commercially available compounds for virtual screening. *J. Chem. Inf. Model.* **45,** 177–182 (2005).

205. Pence, H. E. & Williams, A. ChemSpider: An Online Chemical Information Resource. *J. Chem. Educ.* **87,** 1123–1124 (2010).

206. Chen, C. Y.-C. TCM Database@Taiwan: The World's Largest Traditional Chinese Medicine Database for Drug Screening *In Silico*. *PLoS One* **6,** e15939 (2011).

207. Musyoka, T. M., Kanzi, A. M., Lobb, K. A. & Tastan Bishop, Ö. Analysis of non-peptidic compounds as potential malarial inhibitors against Plasmodial cysteine proteases via integrated virtual screening workflow. *J. Biomol. Struct. Dyn.* **34,** 2084–2101 (2016).

208. Musyoka, T. M., Kanzi, A. M., Lobb, K. A. & Tastan Bishop, Ö. Structure Based Docking and Molecular Dynamic Studies of Plasmodial Cysteine Proteases against a South African Natural Compound and its Analogs. *Sci. Rep.* **6,** 23690 (2016).

209. Kumar, A. & Purohit, R. Use of Long Term Molecular Dynamics Simulation in Predicting Cancer Associated SNPs. *PLoS Comput. Biol.* **10,** (2014).

210. Gromiha, M. M. & Selvaraj, S. Inter-residue interactions in protein folding and stability. *Prog. Biophys. Mol. Biol.* **86,** 235–277 (2004).

211. Roy, A., Kucukural, A. & Zhang, Y. I-TASSER: a unified platform for automated protein structure and function prediction. *Nat. Protoc.* **5,** 725–738 (2010).

212. Tina, K. G., Bhadra, R. & Srinivasan, N. PIC: Protein Interactions Calculator. *Nucleic Acids Res.* **35,** (2007).

213. Vangone, A., Spinelli, R., Scarano, V., Cavallo, L. & Oliva, R. COCOMAPS: a web application to analyze and visualize contacts at the interface of biomolecular complexes. *Bioinforma.* **27,** 2915–2916 (2011).

214. Negi, S. S., Schein, C. H., Oezguen, N., Power, T. D. & Braun, W. InterProSurf: a web server for predicting interacting sites on protein surfaces. *Bioinformatics* **23,** 3397–3399 (2007).

215. Nagarajan, R. *et al.* PDBparam: Online Resource for Computing Structural Parameters of Proteins. *Bioinform. Biol. Insights* **10,** 73–80 (2016).

216. Laskowski, R. A. PDBsum: summaries and analyses of PDB structures. *Nucleic Acids Res.* **29,** 221–222 (2001).

217. Brown, D. K., Sheik Amamuddy, O. & Tastan Bishop, Ö. Structure-Based Analysis of Single Nucleotide Variants in the Renin-Angiotensinogen Complex. *Glob. Heart* **12,** 121–132 (2017).

218. Doshi, U., Holliday, M. J., Eisenmesser, E. Z. & Hamelberg, D. Dynamical network of residue–residue contacts reveals coupled allosteric effects in recognition, catalysis, and mutation. *Proc. Natl. Acad. Sci.* **113,** 4735–4740 (2016).

219. Gray, K. A., Yates, B., Seal, R. L., Wright, M. W. & Bruford, E. A. Genenames.org: The HGNC resources in 2015. *Nucleic Acids Res.* **43,** D1079–D1085 (2015).

220. Finn, R. D. *et al.* Pfam: the protein families database. *Nucleic Acids Res.* **42,** D222–D230 (2014).

221. Wishart, D. S. *et al.* DrugBank: A knowledgebase for drugs, drug actions and drug targets. *Nucleic Acids Res.* **36,** (2008).

222. Orchard, S. *et al.* The MIntAct project - IntAct as a common curation platform for 11 molecular interaction databases. *Nucleic Acids Res.* **42,** (2014).

223. Fabregat, A. *et al.* The Reactome pathway knowledgebase. *Nucleic Acids Res.* **44,** D481–D487 (2016).

224. Gibbons, M. Django REST Swagger. (2017). Available at: https://github.com/marcgibbons/django-rest-swagger.

225. Elastic. Elasticsearch: RESTful, Distributed Search & Analytics. *Online* (2017). Available at: https://www.elastic.co/products/elasticsearch.

226. Apache Software Foundation. Apache Lucene. (2017).

227. Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* **10,** 707–710 (1966).

228. Kollman, P. A. *et al.* Calculating structures and free energies of complex molecules: Combining molecular mechanics and continuum models. *Acc. Chem. Res.* **33,** 889–897 (2000).

229. Brown, D. K. *et al.* MD-TASK: a software suite for analyzing molecular dynamics trajectories. *Bioinformatics* **33,** 2768–2771 (2017).

230. Grewal, R. K. & Roy, S. Modeling Proteins as Residue Interaction Networks. *Protein Pept. Lett.* **22,** 923–933 (2015).

231. Atilgan, A. R., Turgut, D. & Atilgan, C. Screened Nonbonded Interactions in Native Proteins Manipulate Optimal Paths for Robust Residue Communication. *Biophys. J.* **92,** 3052–3062 (2007).

232. Ozbaykal, G., Rana Atilgan, A. & Atilgan, C. In silico mutational studies of Hsp70 disclose

sites with distinct functional attributes. *Proteins Struct. Funct. Bioinforma.* **83,** 2077–2090 (2015).

233. Aier, I., Varadwaj, P. K. & Raj, U. Structural insights into conformational stability of both wild-type and mutant EZH2 receptor. *Sci. Rep.* **6,** 34984 (2016).

234. Anwar, M. A. & Choi, S. Structure-Activity Relationship in TLR4 Mutations: Atomistic Molecular Dynamics Simulations and Residue Interaction Network Analysis. *Sci. Rep.* **7,** 43807 (2017).

235. Bhakat, S., Martin, A. J. M. & Soliman, M. E. S. An integrated molecular dynamics, principal component analysis and residue interaction network approach reveals the impact of M184V mutation on HIV reverse transcriptase resistance to lamivudine. *Mol. Biosyst.* **10,** 2215–2228 (2014).

236. Harvey, M. J. & De Fabritiis, G. An implementation of the smooth particle mesh Ewald method on GPU hardware. *J. Chem. Theory Comput.* **5,** 2371–2377 (2009).

237. Martin, A. J. M. *et al.* RING: networking interacting residues, evolutionary information and energetics in protein structures. *Bioinformatics* **27,** 2003–2005 (2011).

238. Xue, W. *et al.* Exploring the molecular mechanism of cross-resistance to HIV-1 integrase strand transfer inhibitors by molecular dynamics simulation and residue interaction network analysis. *J. Chem. Inf. Model.* **53,** 210–222 (2013).

239. Xue, W., Jiao, P., Liu, H. & Yao, X. Molecular modeling and residue interaction network studies on the mechanism of binding and resistance of the HCV NS5B polymerase mutants to VX-222 and ANA598. *Antiviral Res.* **104,** 40–51 (2014).

240. Karamzadeh, R. *et al.* Machine Learning and Network Analysis of Molecular Dynamics Trajectories Reveal Two Chains of Red/Ox-specific Residue Interactions in Human Protein Disulfide Isomerase. *Sci. Rep.* **7,** (2017).

241. Doncheva, N. T., Klein, K., Domingues, F. S. & Albrecht, M. Analyzing and visualizing residue networks of protein structures. *Trends in Biochemical Sciences* **36,** 179–182 (2011).

242. Hunter, J. D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **9,** 90–95 (2007).

243. McGibbon, R. T. *et al.* MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophys. J.* **109,** 1528–1532 (2015).

244. NetworkX Developers. NetworkX all_pairs_shortest_path_length function. (2017). Available at: https://networkx.github.io/documentation/networkx-1.10/_modules/networkx/algorithms/shortest_paths/unweighted.html#all_pairs_shortest_path_length. (Accessed: 25th March 2017)

245. Brandes, U. A faster algorithm for betweenness centrality*. *J. Math. Sociol.* **25,** 163–177 (2001).

246. Csárdi, G. & Nepusz, T. The igraph software package for complex network research. *InterJournal Complex Syst.* 1695 (2006).

247. Penkler, D., Sensoy, O., Atilgan, C. & Tastan Bishop, O. Perturbation Response Scanning Reveals Key Residues for Allosteric Control in Hsp70. *J. Chem. Inf. Model.* acs.jcim.6b00775 (2017). doi:10.1021/acs.jcim.6b00775

248. Kasahara, K., Fukuda, I. & Nakamura, H. A novel approach of dynamic cross correlation analysis on molecular dynamics simulations and its application to Ets1 dimer-DNA

complex. *PLoS One* **9,** (2014).

249. Di Marino, D., D'Annessa, I., Coletta, A., Via, A. & Tramontano, A. Characterization of the differences in the cyclopiazonic acid binding mode to mammalian and P. Falciparum Ca2+ pumps: A computational study. *Proteins Struct. Funct. Bioinforma.* **83,** 564–574 (2015).

250. Carey, R. M. The intrarenal renin-angiotensin system in hypertension. *Advances in Chronic Kidney Disease* **22,** 204–210 (2015).

251. Heras, M. M., Rodríguez, N. D. C. & González, J. F. N. The Renin-Angiotensin-Aldosterone System in Renal and Cardiovascular Disease and the Effects of its Pharmacological Blockade. *J. Diabetes Metab.* **3,** 1–24 (2012).

252. Yim, H. E. & Yoo, K. H. Renin-Angiotensin System - Considerations for Hypertension and Kidney. *Electrolytes Blood Press. E BP* **6,** 42–50 (2008).

253. Persson, A. E. G., Ollerstam, A., Liu, R. & Brown, R. Mechanisms for macula densa cell release of renin. in *Acta Physiologica Scandinavica* **181,** 471–474 (2004).

254. Hsueh, W. A. & Wyne, K. Renin-Angiotensin-Aldosterone System in Diabetes and Hypertension. *Journal of Clinical Hypertension* **13,** 224–237 (2011).

255. Persson, P. B. Renin: origin, secretion and synthesis. *J. Physiol.* **552,** 667–671 (2003).

256. Komlosi, P., Fintha, A. & Bell, P. D. Current mechanisms of macula densa cell signalling. in *Acta Physiologica Scandinavica* **181,** 463–469 (2004).

257. Peti-Peterdi, J. & Harris, R. C. Macula Densa Sensing and Signaling Mechanisms of Renin

Release. *J. Am. Soc. Nephrol.* **21,** 1093–1096 (2010).

258. Ribeiro-Oliveira, A. *et al.* The renin-angiotensin system and diabetes: An update. *Vascular Health and Risk Management* **4,** 787–803 (2008).

259. Usberti, M. *et al.* Effects of angiotensin II on plasma ADH, prostaglandin synthesis, and water excretion in normal humans. *Am. J. Physiol.* **248,** F254-9 (1985).

260. Patel, B. M. & Mehta, A. A. Aldosterone and angiotensin: Role in diabetes and cardiovascular diseases. *European Journal of Pharmacology* **697,** 1–12 (2012).

261. Te Riet, L., Van Esch, J. H. M., Roks, A. J. M., Van Den Meiracker, A. H. & Danser, A. H. J. Hypertension: Renin-Angiotensin-Aldosterone System Alterations. *Circulation Research* **116,** 960–975 (2015).

262. Kobori, H., Nangaku, M., Navar, L. G. & Nishiyama, A. The intrarenal renin-angiotensin system: from physiology to the pathobiology of hypertension and kidney disease. *Pharmacol. Rev.* **59,** 251–287 (2007).

263. Santos, P. C. J. L., Krieger, J. E. & Pereira, A. C. Renin-angiotensin system, hypertension, and chronic kidney disease: pharmacogenetic implications. *J. Pharmacol. Sci.* **120,** 77–88 (2012).

264. Jackson, G., Gibbs, C. R., Davies, M. K. & Lip, G. Y. H. Pathophysiology. *BMJ Br. Med. J.* **320,** 167–170 (2000).

265. Sayer, G. & Bhat, G. The Renin-Angiotensin-Aldosterone System and Heart Failure. *Cardiology Clinics* **32,** 21–32 (2014).

266. Ferrario, C. M. Cardiac remodelling and RAS inhibition. *Ther. Adv. Cardiovasc. Dis.* 1–10 (2016). doi:10.1177/1753944716642677

267. Henderson, S. O., Haiman, C. a & Mack, W. Multiple Polymorphisms in the renin-angiotensin-aldosterone system (ACE, CYP11B2, AGTR1) and their contribution to hypertension in African Americans and Latinos in the multiethnic cohort. *Am. J. Med. Sci.* **328,** 266–73 (2004).

268. Fabris, B. *et al.* Genetic polymorphisms of the renin-angiotensin-aldosterone system and renal insufficiency in essential hypertension. *J. Hypertens.* **23,** 309–16 (2005).

269. Orenes-Piñero, E. *et al.* Impact of polymorphisms in the renin-angiotensin-aldosterone system on hypertrophic cardiomyopathy. *J. Renin. Angiotensin. Aldosterone. Syst.* **12,** 521–30 (2011).

270. Ortlepp, J. R. *et al.* Genetic polymorphisms in the renin-angiotensin-aldosterone system associated with expression of left ventricular hypertrophy in hypertrophic cardiomyopathy: a study of five polymorphic genes in a family with a disease causing mutation in the myosin bindin. *Heart* **87,** 270–275 (2002).

271. Taverne, K., de Groot, M., de Boer, A. & Klungel, O. Genetic polymorphisms related to the renin-angiotensin-aldosterone system and response to antihypertensive drugs. *Expert Opin. Drug Metab. Toxicol.* **6,** 439–60 (2010).

272. Ibrahim, M. M. RAS inhibition in hypertension. *J. Hum. Hypertens.* **20,** 101–108 (2006).

273. Cagnoni, F. *et al.* Blocking the RAAS at different levels: An update on the use of the direct renin inhibitors alone and in combination. *Vascular Health and Risk Management* **6,** 549–

559 (2010).

274. Israili, Z. H. Renin inhibitors as antihypertensive agents. *Revista Latinoamericana de Hipertension* **3,** 98–112 (2008).

275. Shafiq, M. M., Menon, D. V. & Victor, R. G. Oral Direct Renin Inhibition: Premise, Promise, and Potential Limitations of a New Antihypertensive Drug. *American Journal of Medicine* **121,** 265–271 (2008).

276. Gradman, A. H. *et al.* Aliskiren, a novel orally effective renin inhibitor, provides dose-dependent antihypertensive efficacy and placebo-like tolerability in hypertensive patients. *Circulation* **111,** 1012–1018 (2005).

277. Jensen, C., Herold, P. & Brunner, H. R. Aliskiren: the first renin inhibitor for clinical treatment. *Nat. Rev. Drug Discov.* **7,** 399–410 (2008).

278. Shen, M.-Y. & Sali, A. Statistical potential for assessment and prediction of protein structures. *Protein Sci.* **15,** 2507–2524 (2006).

279. Abraham, M. J. *et al.* GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **1,** 19–25 (2015).

280. Darden, T., York, D. & Pedersen, L. Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems. *J. Chem. Phys.* **98,** 10089 (1993).

281. Berendsen, H. J. C., Postma, J. P. M., van Gunsteren, W. F., DiNola, a & Haak, J. R. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.* **81,** 3684–3690 (1984).

282. Parrinello, M. & Rahman, A. Polymorphic transitions in single crystals: A new molecular dynamics method. *J. Appl. Phys.* **52,** 7182–7190 (1981).

283. Hess, B., Bekker, H., Berendsen, H. J. C. & Fraaije, J. G. E. M. LINCS: A linear constraint solver for molecular simulations. *J. Comput. Chem.* **18,** 1463–1472 (1997).

284. Zondlo, N. J. Aromatic-Proline Interactions: Electronically Tunable CH/pi Interactions. *Acc. Chem. Res.* **46,** 1039–1049 (2013).