Faculty of Science

KROAK: A METADATA COLLECTION SYSTEM FOR LONG TERM MICROBIAL COMMUNITY MONITORING

2017 | LEE HAMMOND BERGSTRAND

B.Sc. Honours thesis - Biology

THOMPSON RIVERS UNIVERSITY

KROAK: A METADATA COLLECTION SYSTEM FOR LONG TERM MICROBIAL COMMUNITY MONITORING.

by

LEE HAMMOND BERGSTRAND

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE (HONS.)

in the

Department of Biological Sciences

(Biology)

This thesis has been accepted as conforming to the required standards by:

Jonathan D. Van Hamme (Ph.D.), Thesis Supervisor, Dept. Biological Sciences

Mila Kwiatkowska (Ph.D.), Examining Committee member, Dept. Computing Science

Don Nelson (Ph.D.), Examining Committee member, Dept. Biological Sciences

Dated this 31th day of May 2017, in Kamloops, British Columbia, Canada

© Lee Hammond Bergstrand, 2017

ABSTRACT

Amplytica is start-up company whose software, called the Amplytica Cloud Platform, helps organizations determine how microbes influence their bioprocesses. Examples of such bioprocesses include anaerobic digestion, wastewater treatment and mine site reclamation. The Amplytica Cloud Platform does this by integrating and analyzing metagenomically derived microbial community data (species composition, diversity, and abundance) and industrial bioprocess data (e.g. temperature, pH, nutrients). To achieve data integration, industrial bioprocess data is considered metadata to the microbial community information and describes the environmental conditions where the microbial community is found. The capture of this industrial metadata requires a robust metadata capture system. Kroak is a metadata capture system for the Amplytica Cloud Platform that facilitates tagging per-sample microbial community information with industrial environmental metadata. It uses a modern web interface for easy deployment, Office Open XML Workbook (XLSX) template files for easy metadata capture, and metadata classes to ensure data consistency and type identification for follow on automated statistics and machine learning. Kroak is a functional metadata capture system which will be iteratively improved upon by Amplytica. Potential improvements include changes to Kroak's data model, increasing the reliability of its metadata parsing and the expansion of its existing web application programming interface.

Thesis Supervisor: Associate Professor Jonathan Van Hamme

ACKNOWLEDGMENTS

I would like to thank Dr. Van Hamme for his continued support of my academic and entrepreneurial endeavors. I am also grateful to the members of my committee for their patience and support in overcoming the numerous obstacles that I have faced throughout my research. I would also like to thank Kamloops Innovation and Amplytica Inc. for their support of this research project.

TABLE OF CONTENTS

LIST OF FIGURES

1. BACKGROUND

In various industries, mixed microbial culture bioreactors are used to perform operations such as wastewater treatment (Henze et al. 2001), mineral extraction (Okibe et al. 2003) and biogas generation from biomass sources such a farm manure or sewage sludge (Mata-Alvarez, Mace & Llabres 2000). In these bioprocesses, various microbial populations either compete for resources or cooperate to achieve some metabolic outcome (Mata-Alvarez, Mace & Llabres 2000, Henze et al. 2001, Okibe et al. 2003). Fostering desirable microbial communities is of utmost importance as the ratio of "good" to "bad" microbes in a reactor is what dictates bioprocess efficiency and ultimately, cost (Chen, Cheng & Creamer 2008). The composition of these communities is driven primarily by the environmental conditions in the reactor vessel such as temperature, pH, organics and available nutrients (Chen, Cheng & Creamer 2008).

Amplytica Inc. is a bioinformatics company that specializes in the development of software for exploring microbial community dynamics via the analysis of high throughput sequencing data. The current rendition of this software is called the Amplytica Cloud Platform (ACP). A future commercial application of ACP is the automated optimization of mixed culture bioreactors, such as those discussed above, through the integrated analysis of microbial community dynamics and bioprocess environmental conditions.

Specifically, the goal is to allow an operator to routinely take samples for amplicon-based metagenomic sequencing while collecting process data such as bioreactor inputs (e.g. nutrients) and outputs (e.g. methane and H_2S in the case of biogas generators). Such a dataset would be then loaded into the ACP. In the context of the ACP, this process data is considered metadata to the genomics sample and describes the environmental conditions when the sample was taken from

the reactor. Inside the ACP, the amplicon-based metagenomic sequencing data is converted into microbial community composition, abundance and diversity using the QIIME microbial community analysis pipeline (Caporaso et al. 2010). When combined with process metadata this microbial community information creates a time series multidimensional dataset detailing the shift in reactor conditions and the resulting microbial community response. To predict optimal bioreactor conditions, the dataset would then be further processed by statistical and/or machine learning algorithms. For example, in anaerobic digestion this would be an optimization of reactor conditions for increased biogas production while maintaining low levels of contaminating gases such as H_2S.

2. INTRODUCTION

The type of environmental metadata available for the above optimization analysis varies by bioprocess. For example, an operator of a tank-based gold tailings bioleaching reactor would collect different process data than and an operator of a Biological Nutrient Removal (BNR) Wastewater Treatment Plant (WWTP). The former is concerned about how much gold is being leached whereas the latter is concerned with how much excess nitrogen and phosphorous is being discharged into the environment. Additionally, different operators of the same bioprocess may not have the same sensors for their bioreactors and thus will be collecting different types of process metadata. Therefore, the ACP must be configurable to capture different types of metadata on a project by project basis.

A future goal of the ACP is for it to be able to automatically identify the type of statistical or machine learning technique to use for an optimization analysis depending on the data available. The type and requirements of these techniques are yet to determined, however, early candidates

include Temporal Gaussian Process Model for Compositional Data Analysis (TGP-CODA) (Aijo, Mueller & Bonneau 2016) and Bayesian algorithms such as Bayesian Adaptive Penalized Counts Splines (BAPCS), Bayesian adaptive lasso (BAL) and Bayesian variable selection (BVS) as implemented in MDSINE (Bucci et al. 2016). Nonetheless, retaining information about the type of environmental data captured, such as whether it is an integer number, floating point number or categorical will be useful for the future development of the ACP.

Previous metadata capture systems for microbial community information have a primarily focus on storing specific types of metadata. For example, Metazen (Bischof et al. 2014) only allows certain types of pre-approved metadata to be captured. Each category of metadata must be found in a controlled vocabulary from BioOntology (Whetzel et al. 2011). Integrating ontologically organized metadata into ACP, would be overly time consuming and may limit the type of metadata which could be captured.

The subject of this thesis is Kroak, a metadata capture system for the ACP which addresses the concerns raised above. It is a lightweight system that facilitates tagging per-sample microbial community information with industrial environmental metadata to assist in the analysis of how microbial communities influence industrial processes. It uses a web interface for easy deployment, Office Open XML Workbook (XLSX) template files (ISO/IEC 29500-1:2016) for easy metadata capture, and metadata classes to ensure data consistency and type identification for follow on automated statistics and machine learning.

3. DESIGN

3.1. DATA MODEL

Kroak's current data model consists of several object-oriented programming (OOP) classes (Cox 1986) which layout a blueprint for the various objects used in the system such as objects representing individual projects, samples, metadata, and data points. Henceforth, objects and classes representing projects, samples, metadata, and data points will be capitalized. For example, the word "Sample" refers to an object which represents a single physical sample which is being analyzed.

3.1.1. PROJECT

A Project represents a collection of samples which were put through a single bioinformatics analysis (Fig. 1). Each Project has a unique identifier and has an optional human readable name and description for easy manual identification. A Project contains one or more Samples (Fig. 1).

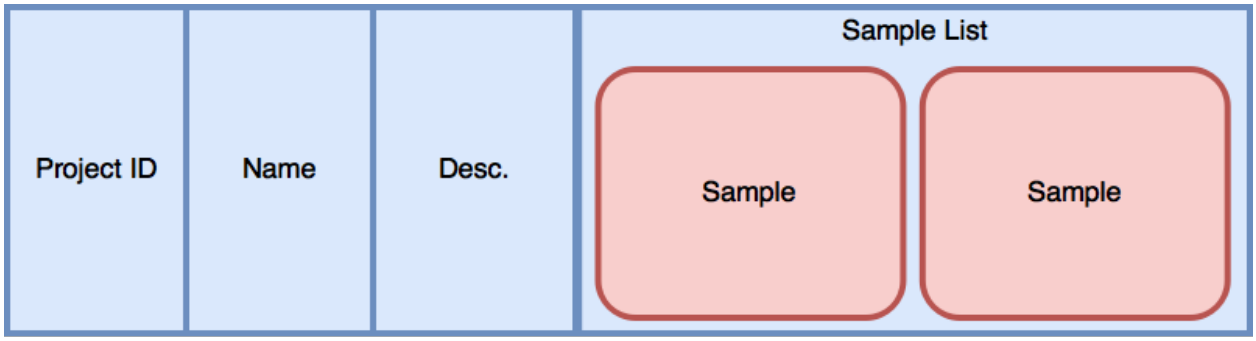

Figure 1: An example of a Project object which contains multiple Samples.

3.1.2. SAMPLE

A Sample represents a unique piece of genomic data along with its associated metadata taken during collection (Fig. 2). Like a Project, each Sample contains a unique identifier, name and

description (Fig. 2). A Sample possesses both bioinformatics analysis results and metadata in the form of a single Metadata object (Fig. 2).

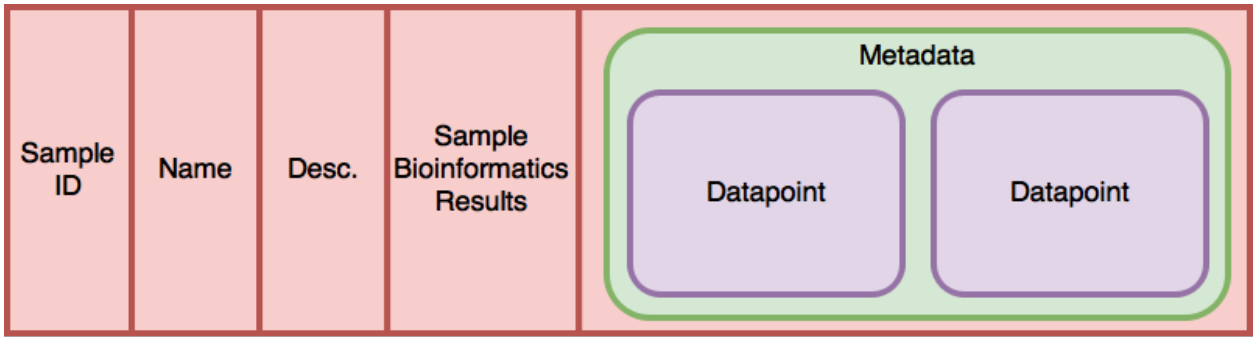

Figure 2: An example of a Sample object which contains a Metadata object with multiple Datapoints.

3.1.3. METADATA AND DATAPOINT

A Metadata object represents the collection of individual data points which belong to a single sample (Fig. 2). The Metadata object contains one or more Datapoint objects (Fig. 2). Each Datapoint represents a single piece of metadata for a given sample. It contains not only the value of the metadata but also its name, type and units. These three properties are referred to the as the data point's metadata class. For example, the piece of metadata 13 °C would have a value of 13 and a metadata class consisting of the name "Temperature", type "Float", and unit "°C". Current metadata types include Floating point number ("Float"), Integer number ("Integer"), String and Date.

3.2. INFORMATION ARCHITECTURE

3.2.1. METADATA CLASSES

Metadata classes are used in Kroak's user interface. Each data point logically belongs to a metadata class which is shared between similar data points found across samples in a project

(Fig. 3). For example, all temperature data points which have the same type and units would be in the same metadata class (Fig. 3). The aggregation of all unique metadata classes in a project is called a metadata class set. Changing a property of a metadata class iteratively changes that same property in each Datapoint belonging to that class. Properties which can be changed include the data point name, type and units.

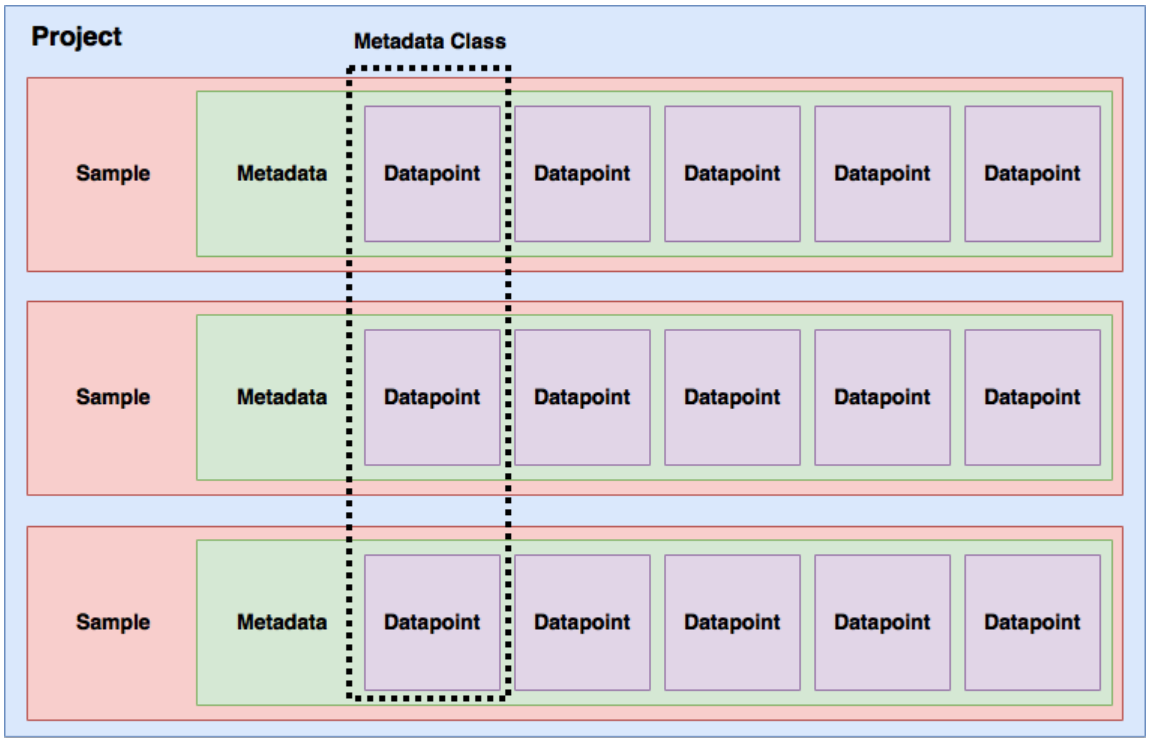

Figure 3: An overview of the classes used in Kroak. Samples contain Metadata. Metadata is composed of Datapoints. Datapoints are members of metadata classes. A metadata class is a subset of data points across samples in a project which contain the same type of data.

3.2.2. DATA FLOW

Template files are used to add metadata to Samples. Adding metadata to a project takes multiple steps. First, a user creates or edits a set of metadata classes for a project (Fig. 4, Fig. 5). These classes define the properties of a project's metadata. The user then downloads an XLSX metadata template file which contains all pre-existing project metadata and metadata classes (Fig. 4). The user then adds metadata to the template file using spreadsheet software such as Microsoft Excel.

The template is then saved and uploaded to Kroak where it is then parsed and the metadata inside is placed into a database for safe keeping (Fig. 4).

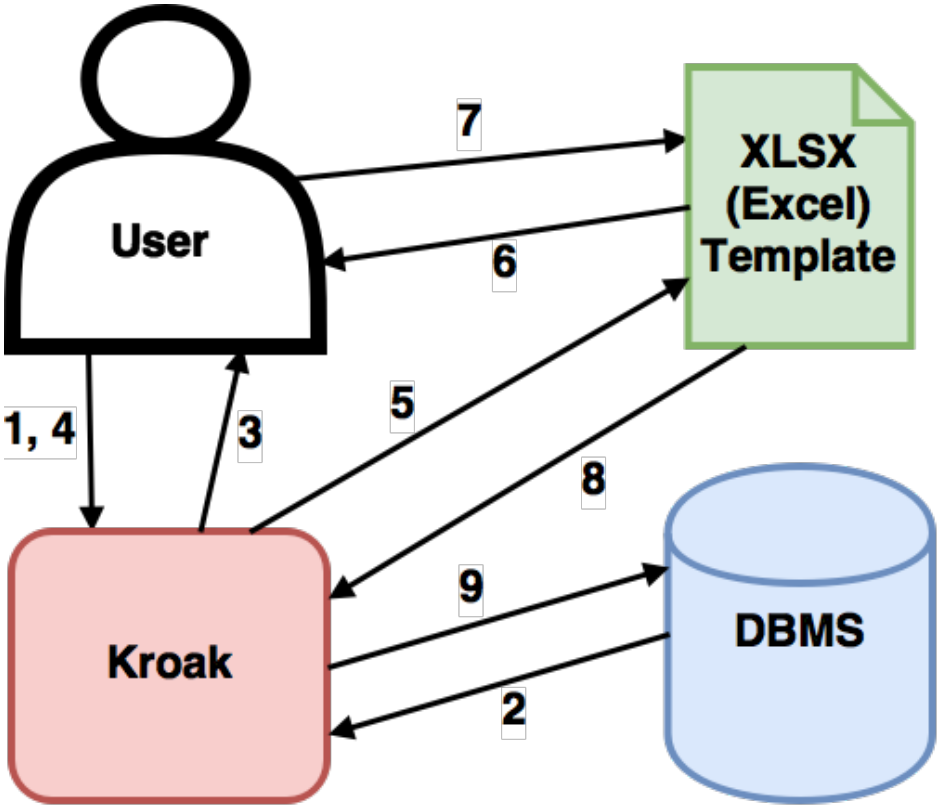

Figure 4: The standard kroak work flow consists of the following steps: 1) The user connects to Kroak, 2) Existing metadata and metadata classes are pulled from the database, 3) They are then sent to the User, 4) The user then sends updated metadata classes to Kroak, 5) These along with existing metadata are used to create an XLSX template file, 6) This template is then sent the user, 7) The user then adds metadata to the template, 8) This completed template is uploaded to Kroak, 9) Finally, the metadata is parsed from the XLSX template and placed in the database.

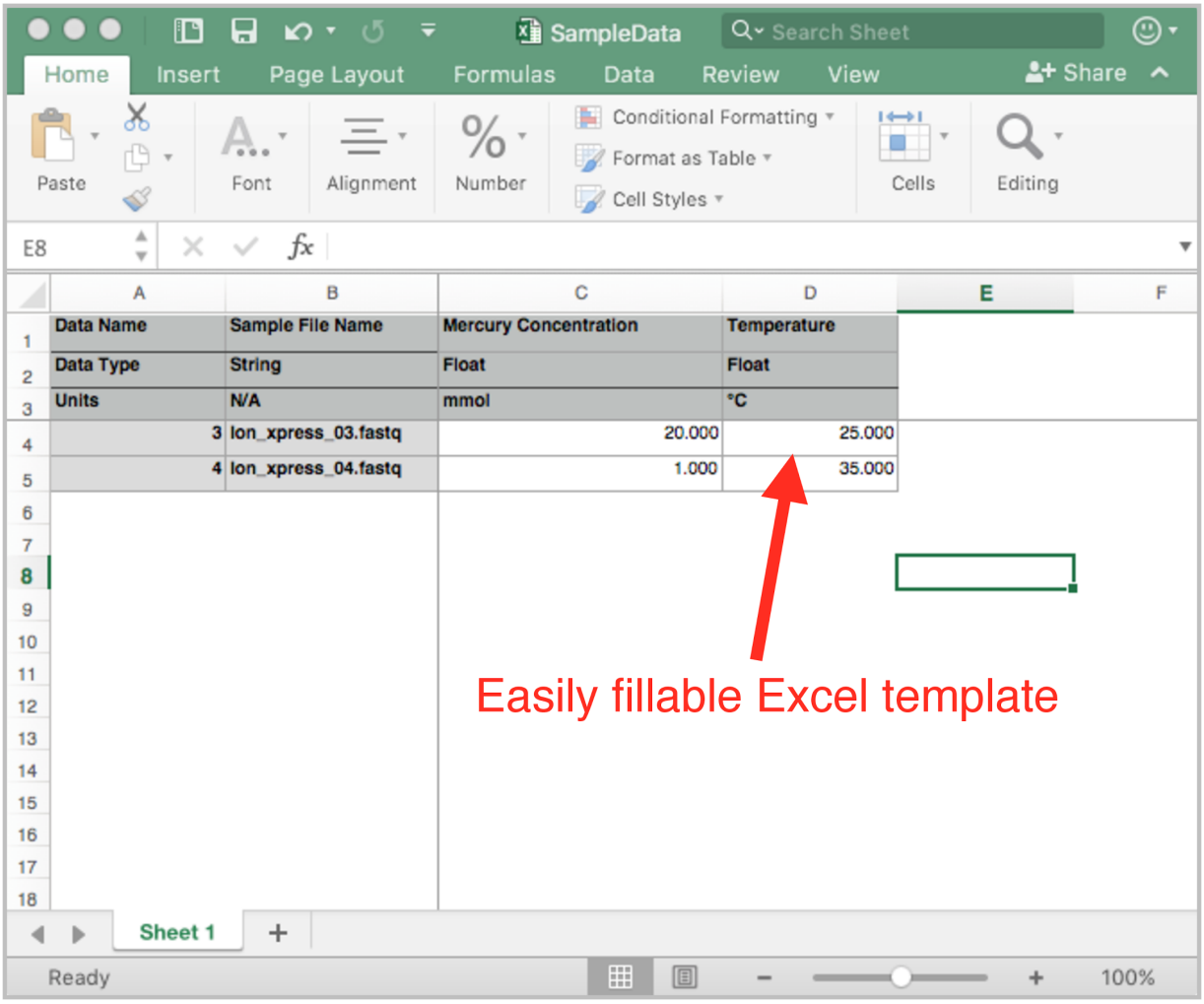

Figure 5: Editing a template XLSX file in Microsoft Excel.

3.2.3. VIEWS

Kroak's web client consists of two views. One view consisting of a page used for assigning and editing metadata classes and viewing changes to the project's metadata structure (Fig. 6). A second view is used for uploading metadata templates for specific projects via a drag and drop user interface (Fig. 7). These two views will henceforth be known as the metadata view and the upload view, respectively.

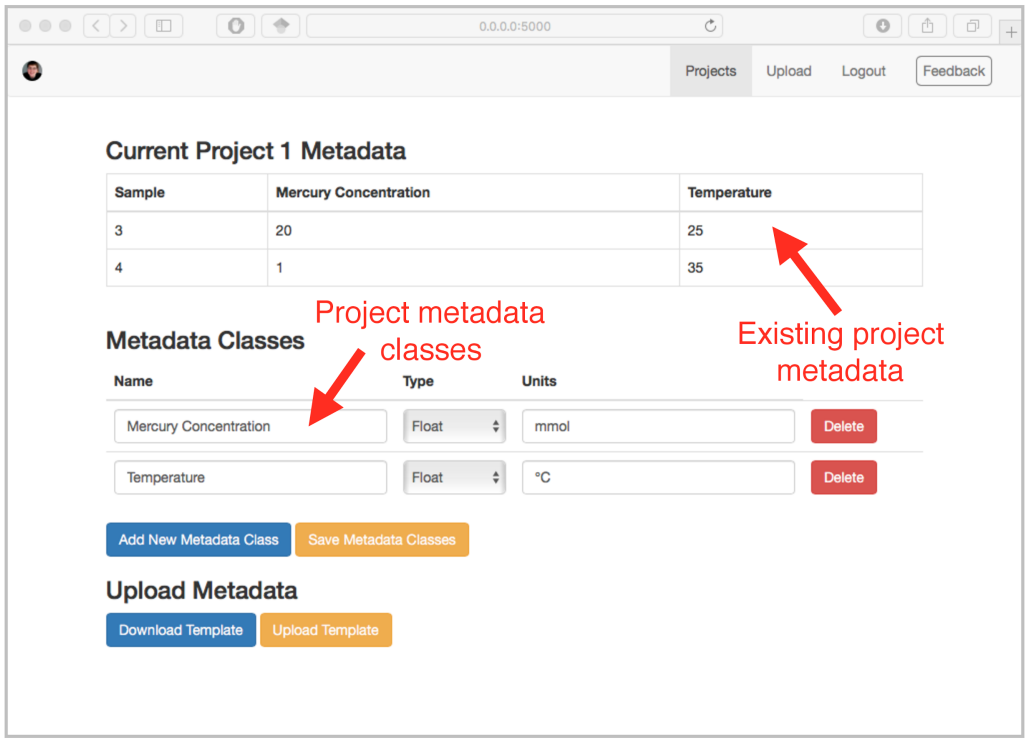

Figure 6: Kroak's user interface for view one which allows end users to declare and modify metadata classes and view existing metadata.

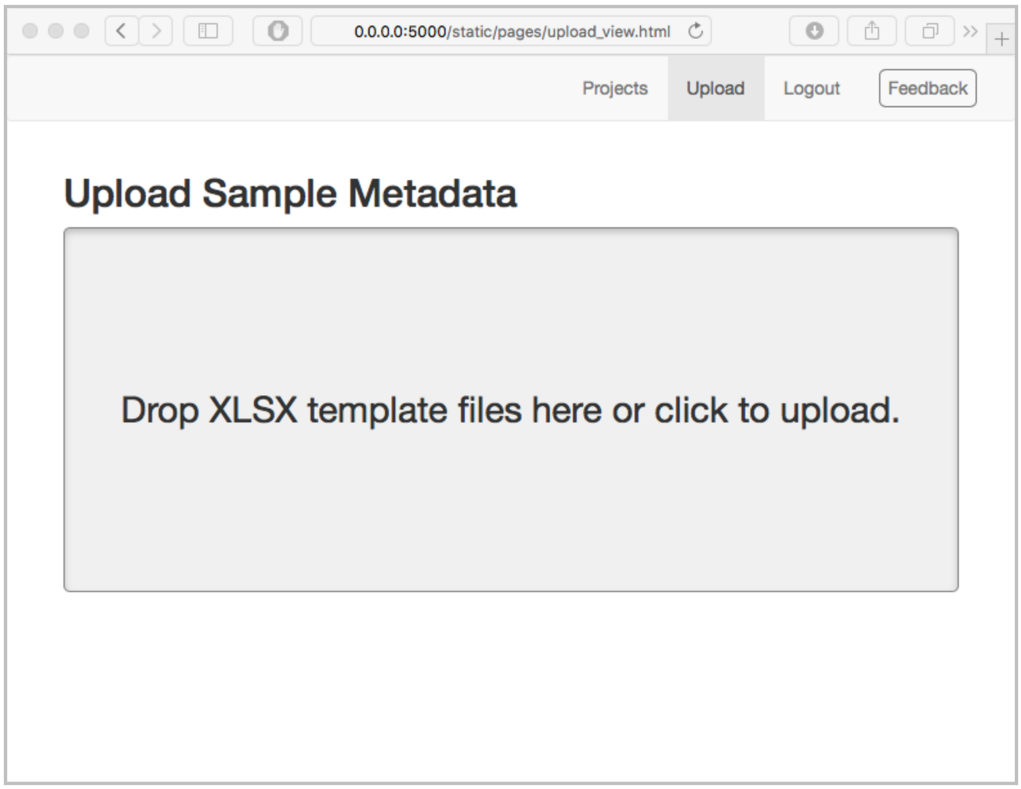

Figure 7: Kroak's user interface for view two which allows users to upload metadata templates.

4. IMPLEMENTATION

Kroak was developed using web technologies. A summary of these technologies used can be found in Supplementary Table ST1 in Appendix A.

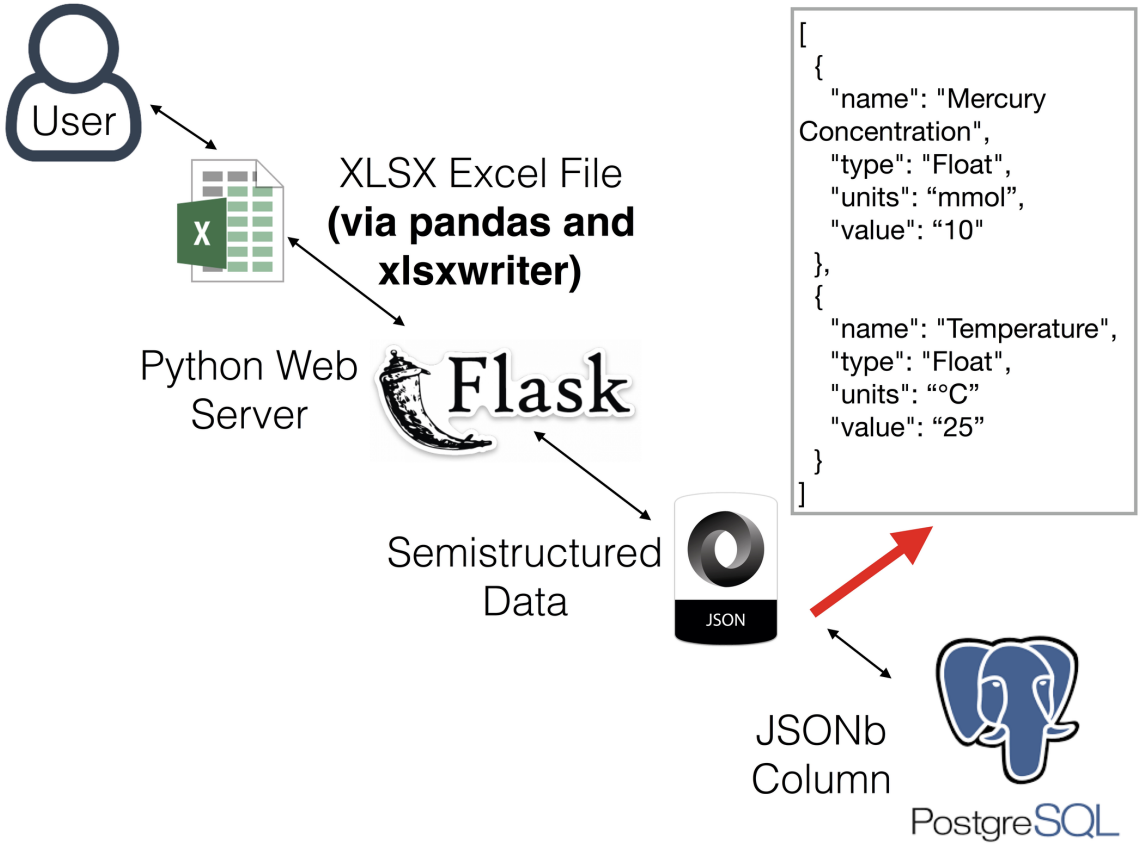

Figure 8: Kroak uses metadata templates and web technologies to input per-sample metadata into a PostgreSQL database.

4.1. USER INTERFACE

The user interacts with Kroak via a web interface consisting of two HTML5 pages representing the views above (Fig. 6, Fig. 7) (Hickson & Hyatt 2011). In the metadata view, the project's current metadata, metadata classes and metadata types are loaded asynchronously from the web server after initial web page load via Asynchronous JavaScript and XML (AJAX) (Mesbah & Van Deursen 2007). In the upload view, drag and drop file uploads are facilitated by Dropzone.js

(Meno). Both views use JQuery (Bibeault & Kats 2008) and Bootstrap (Lerner 2012). The jQuery JavaScript library was used to make AJAX requests and perform other client side scripting such as updating tables (Bibeault & Kats 2008). The Bootstrap 3.0 CSS framework was used to perform page styling and allows for the user interface to be useful on both mobile and desktop browsers (Fig. 6, Fig. 7) (Lerner 2012).

4.2. WEB SERVER

The Kroak web server supplies both web pages and data to the client. It was written using the Flask web framework (Grinberg 2014) for the Python programing language (Van Rossum & Drake 2003). The web server consists of a series of endpoints that serve specific content to the client web browser when it sends a request for a specific URL (Richardson & Ruby 2008). In addition to an endpoint which provides each of the client pages mentioned above and an endpoint for selecting a project, the web server also provides six data endpoints. Three endpoints (get_metadata, get_metadata_classes, get_metadata_types) return JavaScript Object Notation (JSON) (Crockford 2006) formatted data while one receives JSON (set_metadata_classes). Additionally, another endpoint sends the user an XLSX file (generate_template) while a final one accepts an XLSX file upload (upload_template). The get_metadata endpoint returns the entirety of all metadata for a given project. A get_metadata_classes endpoint returns a list of all metadata classes for a project. Additionally, a get_metadata_types endpoint returns a list of available metadata types. The set_metadata_classes endpoint accepts a list of updated metadata classes sent from the client. The generate_template endpoint generates a metadata template XLSX file from existing project metadata and metadata classes and sends it to the client (Fig. 4, Fig. 8). Finally, an upload_template endpoint receives this template XLSX file from the client (Fig. 4, Fig. 8).

4.3. METADATA TEMPLATE GENERATION

Existing metadata and metadata classes are integrated into an XLSX template file which is sent to the user as a file download (Fig. 4, Fig. 8). To generate this template, the system iterates through each sample's metadata and generates a combined XLSX file. This XLSX files is generated using the XLSXWriter Python library (McNamara 2017). In the resulting file, fields which should be edited are labeled grey whereas fields which can be edited are labeled white (Fig. 5).

4.4. METADATA TEMPLATE PARSING

After the user fills in the template file it can be uploaded to the server via a drag and drop interface. Once the template file is uploaded, it is immediately parsed via the Pandas data analysis library (McKinney 2010) and turned into a data frame. Pandas interprets the appropriate data type for each data point in the XLSX file in a heuristic fashion using the xlrd library (Machin 2013, IO Tools). Each row in the data frame (all data points for a given sample) is converted to JSON and stored in each Sample.

4.5. DATA REPRESENTATION AND STORAGE

The SQLAlchemy Object Relational Mapper (ORM) (Bayer 2012) was used to map in memory objects from the data model to individual relational database tables for long term storage. Project and Sample objects are mapped to the database whereas Metadata and Datapoint objects are instantiated dynamically from data contained in a parent Sample object. Each set of data points for a given sample are stored as semi-structured JSON in the parent Sample object (Fig. 9). When a data point needs to be accessed, for example to change its value, it is converted to a Datapoint object dynamically from this JSON (Fig. 9). After modification, this Datapoint is converted back to JSON and stored once again in its parent Sample. This metadata JSON is mapped to a JSONB

(Lerner 2014) column in a PostgreSQL database (Momjian 2001) along with the parent sample's other data via SQLAlchemy.

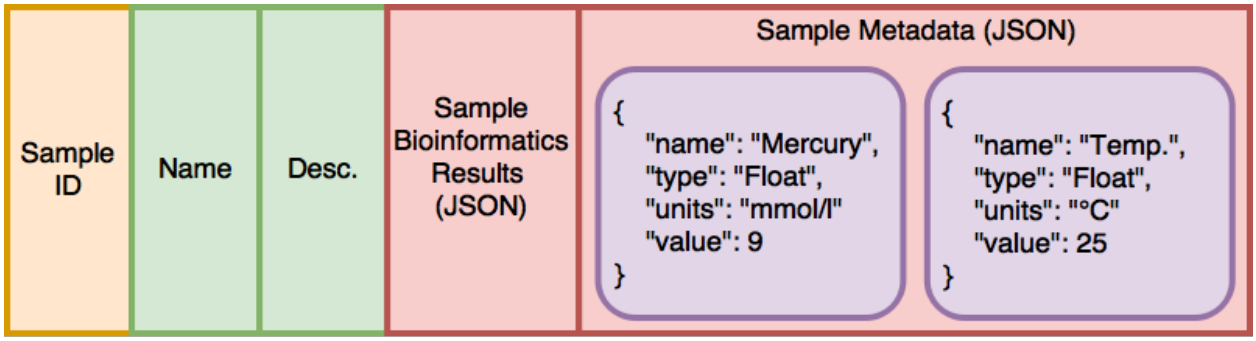

Figure 9: The representation of data points within a Sample object.

5. RESULTS

Kroak was tested with simulated metadata which included Floating point numbers, Integer numbers, Strings and Dates. It was found that Kroak stored this dataset properly and generated metadata templates which were defect free.

6. DISCUSSION

Several decisions were made during the development of Kroak. These include the use of asynchronous web technologies, metadata classes, per-sample metadata storage, XLSX template files, and the PostgreSQL database. The reasons for these design choices are found below.

6.1. STATIC PAGES AND ASYNCHRONOUS CLIENT DATA LOADING

The web client is written in a manner where it is a static web page which dynamically requests data from the server after initial page loading (Garrett 2005, Mesbah & Van Deursen 2007). This contrasts with traditional methods where the server injects content into the page before it is sent to the user. Having all rendering performed in the client's browser has several advantages

(Mesbah & Van Deursen 2007). Firstly, it allows for the initial static client to be deployed via a Content Delivery Network (CDN). CDNs are web services which cache web content across a global network of servers on different continents (Leighton & Lewin 2003). When a user requests a page from the website it is loaded from the nearest geographically located server which drastically decreases page load times (Leighton & Lewin 2003). Examples of CDNs are Cloudflare (The Web Performance & Security Company) and Amazon Cloudfront (Amazon CloudFront – Content Delivery Network (CDN)). It also pushes the cost of rendering the page to the client allowing for more cost-effective server hardware to be used (Mesbah & Van Deursen 2007). From a development perspective, having a client page which is static allows the backend server to be completely rewritten and replaced without an effect on the client (Mesbah & Van Deursen 2007). This makes it easier for the client web application to be tested independently of the server web application. For example, during automated testing, a mock up web server which provides the same content every time and does not require a database can be used as a surrogate backend (Humble & Farley 2010, Taneja, Zhang & Xie 2010). This allows testing to be faster as less setup time is required (Humble & Farley 2010, Taneja, Zhang & Xie 2010). Additionally, since the server application itself is just a series of endpoints it can also be tested independently during automated testing (Mesbah & Van Deursen 2009).

6.2. METADATA CLASSES

The use of Metadata Classes is especially important for follow on operations where knowing the type of each data point is essential. For example, in situations where statistical operations can only accept certain types of data such as categorical or numerical. Knowing the type of the available data would allow for automated systems to determine what type of statistical test to use

rather than relying on the end user to select the appropriate test. This will be useful once Kroak is fully integrated into the rest the Amplytica Cloud Platform.

6.3. PER-SAMPLE METADATA STORAGE

Storing metadata at a sample rather than project level was done to make it easier for only subsets of the data to be used at one time. Potentially, this will allow for projects to be merged and analyses to be performed on data points, with common metadata classes, originating from different projects.

6.4. XLSX FILES

The Office Open XML Workbook format is a common spreadsheet format created by Microsoft and was chosen for a variety of reasons. Internally it stores data in Extensible Markup Language (XML) format and is thus parsable by third party spreadsheet programs other than Microsoft Excel (Kosek 2008). It has been standardized by the International Organization for Standardization / International Electrotechnical Commission (ISO/IEC 29500-1:2016) and the European Computer Manufacturers Association (ECMA-376). A benefit of using this format is that Excel is commonly used in both academia and industry. For example, the author performed a small survey in which two microbial ecology researchers were asked what would be their preferred method for imputing metadata into the ACP. Both said that they would prefer to be able to upload the metadata in spreadsheet form and preferably as an Excel file. Additionally, existing data sources are often already found in XLSX format and can be copy-pasted into a Kroak metadata template. XLSX has several advantages over more open spreadsheet file formats such as Comma-separated values (CSV) (Shafranovich 2005). For example, XLSX provides facilities for colour coding cells and has data protection features which can be used to protect tables and

lock cells to prevent unnecessary changes to the sheet's structure (Harvey 2016). XLSX can also be used to enforce data stringency via the ability for cells to perform data validation (Harvey 2016). Other than colour coding, these features are not currently implemented but may be useful in the future.

6.5. POSTGRESQL

PostgreSQL was chosen as the backing store for Kroak primarily because it is a free and open source database management system (DBMS) that can be hosted on one's own servers with no licensing fees (Momjian 2001). Alternatively, various providers, such as Heroku (Middleton & Schneeman 2013), provide PostgreSQL as a remotely hosted Database as a Service (DaaS) (Hacigumus, Iyer, & Mehrotra 2002) for a low or even no fee. Additionally, PostgreSQL can act as a document database with the ability to store and query semistructured data stored in JSON columns (Lerner 2014). In Kroak, metadata is currently stored in a JSONB column which is a special column type where JSON strings are stored in a binary format (Fig. 8, Fig. 9) (Lerner 2014). The data within the JSON is parsed by data type. For example, floating point numbers in the JSON will be parsed into a binary float value and stored in the database. This saves significant database space versus storing the JSON in a purely textual column (Lerner 2014).

7. CONCLUSIONS AND FUTURE DIRECTIONS

Although Kroak is a functional metadata capture system, several opportunities for improvements and future work items were identified during its development and are detailed below.

7.1. FURTHER PLATFORM INTEGRATION

Kroak is intended to be the metadata capture component of the Amplytica Cloud platform and will be integrated soon. The ACP currently contains endpoints for handling the creation, retrieval, update and delete (CRUD) of User, Project and Sample objects. The project and sample classes from Kroak overlap with those of ACP's current data model. The two models could be joined together in a united data model allowing other ACP components to access per-sample metadata.

7.2. IMPROVED DATA MODEL

In the current implementation, each sample is stored in its own database row and all data points of a sample are stored in a single JSON cell (Fig. 8, Fig. 9). Each data point in the JSON contains not only its value but also its metadata class information such as its name, type and units (Fig. 9). If one makes a mistake in the creation of a metadata class, for example using Fahrenheit rather than Celsius as the class units, and goes to correct this mistake then Kroak currently must load each sample's metadata from the database. The JSON inside each Sample needs to be parsed and instantiated into Datapoint objects. These objects need to be searched to determine which ones belong to metadata classes that need to be changed. Then only the objects that belong to that class need to be updated. Afterwards, the whole process is reversed by converting these Datapoint objects back to JSON and then finally saving them back to their respective Sample's metadata column in the database. This is quite inefficient. This problem results from storing each data point's metadata class information alongside its actual value. This problem could be solved by normalizing the database schema further (Fig. 10) (Beeri, Bernstein, & Goodman 1978).

A more efficient solution would be to add a separate Datapoint table and Metadata_Class table (Fig. 10). This would allow for the storage of metadata class information independent of each

data point. Each Datapoint object would have a link to the Metadata_Class and Sample for which it is associated (Fig. 10). A Metadata_Class is linked to all data points across all samples which possess its class (Fig. 10). Since metadata classes are linked rather than being stored inside each data point, a single Metadata_Class object can be changed to change the properties of multiple data points across a project.

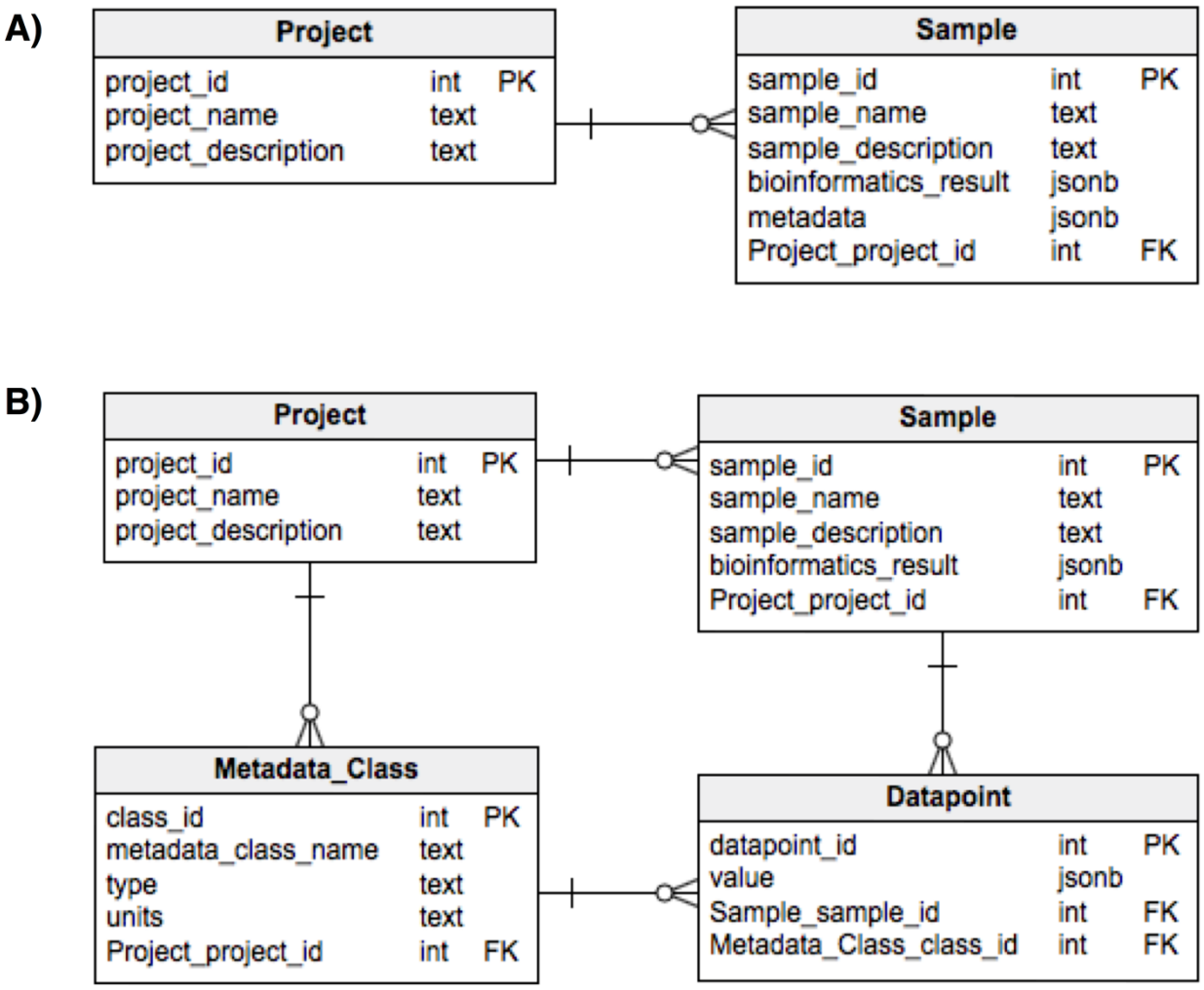

Figure 10: A) The current Kroak database schema. B) An improved Kroak database schema with new tables for metadata classes and data points.

7.3. IMPROVED METADATA PARSING

Pandas is used to parse metadata, however, some data types such as dates are simply parsed to a number rather than a native Python object. The type information found in a data point's metadata class could be used to assist in the parsing of an uploaded template file's metadata. For example, if a data point has a type of "date", then this information could be used parse the date in the template file into an object of Python's native date class (Hellmann 2011). Improved parsing and the use of native Python objects could be helpful during the future development of statistical analysis components.

7.4. EXPANDED WEB APPLICATION PROGRAMMING INTERFACE (API)

Currently our web Application Programming Interface (API) has facilities for setting and getting metadata classes, sending metadata types, retrieving project metadata, template creation and template upload. Additional endpoints could be created for CRUD operations on individual data points. Other components of the ACP have endpoints for project and sample CRUD. Together this expanded set of endpoints would create an API for manipulating all data components. Not only could this expanded API be used in a web development context but it could also be used to collect sensor data. Many bioreactors have built in sensors for measuring values such as O_2 concentration (Bambot et al. 1994), pH (Jeevarajan et al. 2002) and temperature. If these sensors are hooked up to the internet they could potentially be used to send data in real-time to Kroak via the above expanded API over HTTP (Fielding et al. 1999).

7.5. CREATION OF AUTOMATED TESTS

The creation and collection of metadata is complex and has many opportunities for errors. A suite of automated tests should be created to test both template creation and parsing. These tests would

be performed using various types of metadata to ensure that they are all compatible with the various operations performed by the codebase and that no errors or mistakes occur.

LITERATURE CITED

Aijo, T., Mueller, C. L., & Bonneau, R. (2016). Temporal probabilistic modeling of bacterial compositions derived from 16S rRNA sequencing. bioRxiv, 076836.

Amazon CloudFront – Content Delivery Network (CDN). (n.d.). Retrieved April 28, 2017, from https://aws.amazon.com/cloudfront/

Bambot, S. B., Holavanahali, R., Lakowicz, J. R., Carter, G. M., & Rao, G. (1994). Phase fluorometric sterilizable optical oxygen sensor. Biotechnology and bioengineering, 43(11), 1139-1145.

Bayer, M. (2012). Sqlalchemy-the database toolkit for python. URL http://www. sqlalchemy. org/. Accessed on the 13th of November.

Beeri, C., Bernstein, P. A., & Goodman, N. (1978, September). A sophisticate's introduction to database normalization theory. In Proceedings of the fourth international conference on Very Large Data Bases-Volume 4 (pp. 113-124). VLDB Endowment.

Bibeault, B., & Kats, Y. (2008). jQuery in Action. Dreamtech Press.

Bischof, J., Harrison, T., Paczian, T., Glass, E., Wilke, A., & Meyer, F. (2014). Metazen–metadata capture for metagenomes. Standards in genomic sciences, 9(1), 18.

Bucci, V., Tzen, B., Li, N., Simmons, M., Tanoue, T., Bogart, E., ... & Olle, B. (2016). MDSINE: Microbial Dynamical Systems INference Engine for microbiome time-series analyses. Genome biology, 17(1), 121.

Caporaso, J. G., Kuczynski, J., Stombaugh, J., Bittinger, K., Bushman, F. D., Costello, E. K., ... & Huttley, G. A. (2010). QIIME allows analysis of high-throughput community sequencing data. *Nature methods*, *7*(5), 335-336.

Chen, Y., Cheng, J. J., & Creamer, K. S. (2008). Inhibition of anaerobic digestion process: a review. Bioresource technology, 99(10), 4044-4064.

Cox, B. (1986). Object-oriented programming: an evolutionary approach.

Crockford, D. (2006). The application/json media type for javascript object notation (json). Internet Engineering Task Force Request for Comments 4627

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext transfer protocol--HTTP/1.1* (No. RFC 2616).

Garrett, J. J. (2005). Ajax: A new approach to web applications. Adaptive path.

Grinberg, M. (2014). Flask web development: developing web applications with python. " O'Reilly Media, Inc.".

Hacigumus, H., Iyer, B., & Mehrotra, S. (2002). Providing database as a service. In Data Engineering, 2002. Proceedings. 18th International Conference on (pp. 29-38). IEEE.

Harvey, G. (2016). Excel 2016 all-in-one for dummies. Hoboken, NJ: For Dummies, a Wiley brand.

Hellmann, D. (2011). The Python standard library by example. Addison-Wesley Professional.

Henze, M., Harremoes, P., la Cour Jansen, J., & Arvin, E. (2001). Wastewater treatment: biological and chemical processes. Springer Science & Business Media.

Hickson, I., & Hyatt, D. (2011). Html5. W3C Working Draft WD-html5-20110525, May.

Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader). Pearson Education.

International, E. (2011). Standard ECMA-376 - Office Open XML File Formats.

IO Tools (Text, CSV, HDF5, ...)¶. (n.d.). Retrieved April 28, 2017, from http://pandas.pydata.org/pandas-docs/stable/io.html#excel-files

ISO/IEC, (2016). ISO/IEC 29500-1:2016 Information technology -- Document description and processing languages -- Office Open XML File Formats -- Part 1: Fundamentals and Markup Language Reference. Geneva, Switzerland: ISO/IEC.

Jeevarajan, A. S., Vani, S., Taylor, T. D., & Anderson, M. M. (2002). Continuous pH monitoring in a perfused bioreactor system using an optical pH sensor. Biotechnology and bioengineering, 78(4), 467-472.

Kosek, J. (2008). From the office document format battlefield. IT Professional, 10(3).

Leighton, F. T., & Lewin, D. M. (2003). U.S. Patent No. 6,553,413. Washington, DC: U.S. Patent and Trademark Office.

Lerner, R. M. (2012). At the forge: twitter bootstrap. Linux Journal, 2012(218), 6.

Lerner, R. M. (2014). At the forge: PostgreSQL, the NoSQL database. Linux Journal, 2014(247), 5.

Machin, J. (2013, April) xlrd 0.9. 0. Python Package Index, [Online]. Available: https://pypi. python. org/pypi/xlrd. [Accessed 5 March 2013].

Mata-Alvarez, J., Mace, S., & Llabres, P. (2000). Anaerobic digestion of organic solid wastes. An overview of research achievements and perspectives. Bioresource technology, 74(1), 3-16.

McKinney, W. (2010, June). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56). van der Voort S, Millman J.

McNamara, J. (n.d.). Creating Excel files with Python and XlsxWriter. Retrieved May 01, 2017, from http://xlsxwriter.readthedocs.io/

Meno, M. (n.d.). Dropzone.js. Retrieved May 01, 2017, from http://www.dropzonejs.com/

Mesbah, A., & Van Deursen, A. (2007, March). Migrating multi-page web applications to single-page Ajax interfaces. In Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on (pp. 181-190). IEEE.

Mesbah, A., & Van Deursen, A. (2009, May). Invariant-based automatic testing of AJAX user interfaces. In Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on (pp. 210-220). IEEE.

Middleton, N., & Schneeman, R. (2013). Heroku: Up and Running. " O'Reilly Media, Inc.".

Momjian, B. (2001). PostgreSQL: introduction and concepts (Vol. 192). New York: Addison-Wesley.

Okibe, N., Gericke, M., Hallberg, K. B., & Johnson, D. B. (2003). Enumeration and characterization of acidophilic microorganisms isolated from a pilot plant stirred-tank bioleaching operation. Applied and environmental microbiology, 69(4), 1936-1943.

Richardson, L., & Ruby, S. (2008). RESTful web services. " O'Reilly Media, Inc.".

Shafranovich, Y. (2005). Common format and MIME type for comma-separated values (CSV) files. Internet Engineering Task Force Request for Comments 4180

Taneja, K., Zhang, Y., & Xie, T. (2010, September). MODA: Automated test generation for database applications via mock objects. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 289-292). ACM.

The Web Performance & Security Company. (n.d.). Retrieved April 28, 2017, from https://www.cloudflare.com/

Van Rossum, G., & Drake, F. L. (2003). Python language reference manual (p. 144). Network Theory.

Whetzel, P. L., Noy, N. F., Shah, N. H., Alexander, P. R., Nyulas, C., Tudorache, T., & Musen, M. A. (2011). BioPortal: enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications. Nucleic acids research, 39(suppl 2), W541-W545.

APPENDIX A

Supplementary Table ST1: A summary table of all software packages and technologies used to create Kroak, their roles, and citations.

Software Package / Technology	Description	Citation	Website
HTML5	HTML (HyperText Markup Language) describes and defines the content of a webpage.	Hickson & Hyatt 2011	https://www.w3.org/TR/html5/
AJAX	Ajax (Asynchronous JavaScript and XML) is a set of Web development techniques using many Web technologies on the client side to create asynchronous Web applications.	Mesbah & Van Deursen 2007	N/A
Dropzone.js	DropzoneJS is an open source library that provides drag'n'drop file uploads with image previews.	Meno	http://www.dropzonejs.com
JQuery	jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.	Bibeault & Kats 2008	https://jquery.com

Bootstrap 3.0	Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.	Lerner 2012	http://getbootstrap.com
Python 3	Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.	Van Rossum & Drake 2003	https://www.python.org
Flask	Flask is a micro web framework written in Python	Grinberg 2014	http://flask.pocoo.org
JSON	JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language,	Crockford 2006	http://www.json.org
XLSXWriter	XlsxWriter is a Python module for creating Excel XLSX files.	McNamara	http://xlsxwriter.readthedocs.io
Pandas	pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.	McKinney 2010	http://pandas.pydata.org
xlrd	A Python module for extracting data from Excel		https://pypi.python.org/pypi/xlrd

	spreadsheet files.		
SQLAlchemy	SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.	Bayer 2012	https://www.sqlalchemy.org
PostgreSQL	PostgreSQL, often simply Postgres, is a free and open source object-relational database (ORDBMS)	Momjian 2001	https://www.postgresql.org
JSONB	JSONB (‘’jsonb”) is a column type of PostgreSQL. Rather than being stored as text, the json in jsonb column is stored in a decomposed binary format that makes it slightly slower to input due to added conversion overhead, but significantly faster to process, since no reparsing is needed. jsonb also supports indexing, which can be a significant advantage.	Lerner 2014	https://www.postgresql.org/docs/9.4/static/datatype-json.html