Nature's Palette - An Open Access Repository of Spectral Data

by

©Rabeya Akhter

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

August 2020

St. John's Newfoundland

Abstract

Light environment, coloration, and color vision play an important role in nature by affecting animal behavior and ecology. Projects that study these topics often require thousands of spectral measurements from light, animal/plant integuments, and medium (air, water). However, substantial resources are wasted on reproducing the same data for different research projects because very few of these data are made publicly available to other researchers. Our proposed open-access repository allows researchers to share, search, and download spectral data. It offers a curatorial pipeline which will help to promote the digital preservation for easy discovery, access, and usability of spectral data and their metadata. The repository will provide an incredible resource of spectral measurements and encourage large-scale projects through the reuse of existing data across an international scientific community.

Acknowledgements

First and foremost, I would like to express my gratitude to almighty Allah for his greatness and for giving me the strength and courage to complete this thesis successfully. I would like to take the opportunity to thank Memorial University of Newfoundland for providing me such an opportunity to pursue the degree of Master of Science in Computer Science here.

I am extremely grateful to my supervisors Dr. Adrian Fiech, Associate Professor of Department of Computer Science for encouraging my research and helping me with important suggestions, comments, and guidance and Dr. Pierre-Paul Bitton, Assistant Professor of Psychology for his valuable time, co-operation, and generosity which set this work possible. Throughout my thesis, they both have supported me a lot with their patience and expertise. I also wish to show my appreciation to everybody who was important to the successful realization of this thesis.

Finally, my deep and sincere gratitude to my family and friends for their continuous support and encouragement throughout my years of study. It has been an amazing experience for me. This achievement would not have been possible without them.

Table of Contents

Abstract **ii**

Acknowledgments **iii**

Table of Contents **viii**

List of Figures **x**

List of Abbreviations and Symbols **x**

1 Introduction **1**

1.1 Problem description 1

1.2 Goals 2

1.3 Background and Motivation 2

1.4 Structure of the Thesis 4

2 Literature Review **5**

2.1 Open-access digital repository overview 5

2.2 Data preservation 6

2.3 Metadata 8

2.3.1 Choosing Metadata 8

2.3.1.1 Dublin Core 9
2.3.1.2 Darwin Core 9
2.3.2 Using Metadata 9
2.4 Color measurement in Visual communication 14
2.4.1 Spectral data 14
2.4.2 Spectral data file 15
2.4.3 *pavo*: an R package for the analysis, visualization and organization of spectral data 16

3 About the Project **19**
3.1 Proposed open access digital repository 19
3.1.1 Upload data 19
3.1.1.1 Data validation and analysis 20
3.1.2 Query repository 21
3.1.3 Download Data 21
3.2 Technical overview 22
3.3 Related works 23

4 Application Development **24**
4.1 Software development life cycle (SDLC) 25
4.1.1 How SDLC Works 25
4.1.2 Selecting a methodology 26
4.2 Agile Methodology 27
4.2.1 The Agile Iteration Workflow 28
4.2.1.1 Planning and Requirement Analysis 28
4.2.1.2 System Design 28
4.2.1.3 Implementation 29

4.2.1.4 Testing 29
4.2.1.5 Delivery and Feedback 29
4.2.1.6 Maintenance 29
4.2.2 Agile Model - Advantages 30

5 Requirements Analysis 31

5.1 Requirements specification 31
5.1.1 Functional requirements 32
5.1.2 Non-Functional requirements 32
5.1.3 Target environment 33
5.1.4 Use Case Diagram 33
5.1.5 Use Case Description 35
5.1.5.1 Identify User 36
5.1.5.2 Register Researcher 36
5.1.5.3 Upload Data 37
5.1.5.4 Query Repository 39
5.1.5.5 Download Data 40
5.1.5.6 Update Search Terms 41
5.1.5.7 Request Data Modification 42
5.2 Domain Modeling 44
5.3 Class Diagram Analysis 45
5.4 Sequence Diagram 46

6 Design and Prototyping 50

6.1 Design Goals 50
6.1.1 Ease of operation 50
6.1.2 Flexibility 51

6.1.3 Scalability 52
6.2 System Design and Architecture 53
6.2.1 Logical Design 53
6.2.2 Physical Architecture 53

7 Implementation and coding 56

7.1 Implementation issues 56
7.2 Version Management 57
7.3 Development platform 58
7.3.1 Prerequisites 58
7.3.1.1 NodeJS and NPM 58
7.3.1.2 MongoDB 59
7.3.1.3 R language 59
7.3.2 Project Installation steps 60
7.4 Project Structure 60
7.5 Work Environment and Demo 63

8 Hosting and Deploying 64

8.1 Application Deployment 64
8.1.1 Cloud resources 65
8.1.2 Setting up the Server 65
8.2 Hosting our application 69
8.2.1 File permission: 69

9 Application Overview 72

9.1 Using Nature's Palette 72
9.1.1 Authentication 73
9.1.2 Submit Data 76

9.1.2.1 Submission Form 77
9.1.3 Search 79
9.1.4 Download 80
9.1.5 Contact Form 81
9.1.6 About and Help 81

10 Feedback and Implications 84

10.1 Researcher/public use of Nature's Palette 84
10.1.1 Case 1 84
10.1.2 Case 2 85
10.1.3 Case 3 85
10.1.4 Case 4 86
10.1.5 Case 5 86
10.1.6 Case 6 87

11 Discussion and Conclusion 88

11.1 Future work 88
11.2 Conclusions 89

Bibliography 89

A Detail description of sequence diagram 94

A.1 Boundary, entity, and control classes 94
A.1.1 Boundary Objects: 94
A.1.2 Entity Objects 96
A.1.3 Control Objects: 97

List of Figures

2.1 Male peacock exhibits a visual display as a part of its courtship rituals 14

2.2 Spectral reflectance of specific color patches on ***Choerodon fasciatus***. 15

2.3 There are two examples (right and left) and that in each there is metadata followed by spectral data 16

2.4 Example pavo workflow, highlighting its main functions and plotting capabilities 17

3.1 Metadata file and raw files format 20

3.2 Plots describing the curation proccess 21

3.3 'Systems architecture,' a diagram demonstrating the different systems involved. 22

4.1 Agile Model 27

5.1 Use Case Diagram: two types of actors have extensive access to all functions 34

5.2 The UML class diagram of Nature's Palette application 45

5.3 UploadFiles Sequence Diagram 47

5.4 SearchData Sequence Diagram 48

5.5 Download Sequence Diagram 49

6.1 Shows a detailed view on the system's architecture, layering, and components 54

7.1 Project Structure 61

9.1 Home page 73
9.2 Login/Registration page 74
9.3 ORCID Login 75
9.4 User Profile 75
9.5 Submission page 76
9.6 Submission Step-1 77
9.7 Submission Step-2 78
9.8 Submission Step-3 79
9.9 Search filters 80
9.10 Search result 80
9.11 Contact Us Page 81
9.12 About Page 82
9.13 Help Section 82

Chapter 1

Introduction

1.1 Problem description

Light affects aspects of almost all animals' life. The coloration of plumage or body, growth, reproduction, migration, diapause and color vision intrinsically rely on light. Different species respond to light at different portions of the electromagnetic spectrum, which raises questions about the behavioral uses of spectral information in nature. In biology, the measurement of color has become increasingly popular. These measurements are found in studies of color vision research, communication, signaling, camouflage, evolution and the examination of light environment.

The availability and rising popularity of portable spectrometers have greatly advanced the study of animal coloration. These studies typically require thousands of spectral measurements, yet not all data are publicly available to other researchers. Furthermore, the datasets that are available through repositories are not curated in a way that makes them accessible and discoverable. Poor and incomplete metadata collection or lack of quality data can limit the opportunities for large scale studies.

As a result, substantial resources are wasted on duplicating data for similar research projects. Understanding the importance and analysing the target audience, we aim to develop a repository with samples of services that can be provided by the repository specific for the analysis and extraction of spectral data.

1.2 Goals

The main objective of this project is to develop a proof-of-concept open-access spectral data repository that will promote the digital preservation for easy discovery, access, and usability of spectral data across an international community of users in biology, ecology, and environmental sciences. Specifically, the target will be:

- Developing a curatorial pipeline that will allow researchers to easily supply their spectral data without going through arduous data processing, a major barrier to data sharing.
- Developing a pilot online repository of spectral data with advanced discovery functions through enhanced metadata.
- Initiating the development of new database services that would be unique to spectral data, but also advance analytical services offered by databases in general.
- Making the repository easily accessible and scalable with data growth.

1.3 Background and Motivation

Due to lack of a cohesive framework for working with and analyzing spectral data from ambient light and reflectance measurements, in 2013 Dr. Pierre-Paul Bitton and two colleagues published a package for the R programming language, called ‘pavo’, which allows the import, exploration, processing, and analysis of spectral colour data under a variety of user-defined models [1]. This package promoted protocol standardization across laboratories, batch processing, automation of workflows, and increased repeatability of colour and colour vision research. This package is now the go-to software for handling spectral data in visual ecology (415 citations since 2013; Google Scholar April 20th 2020) [1], with an increasing user base and no real competitors.

The projects for which ‘pavo’ is used typically require lots of spectral measurements. These measurements usually come from ambient natural or artificial light sources (irradiance), transmittance properties of the medium (air, water), and reflective properties of animal integuments and the substrates in their environment. Reflective properties can be acquired directly from the wild or from museum specimens. Dr. Bitton and his colleagues estimate that comparative studies in birds, for example, have characterised the plumage colours for over 40 % of described species (4000+ taxa). Yet, a survey of the papers that cite pavo demonstrates that less than 5 % of these data are publicly available to other researchers.

Furthermore, the datasets that are available through repositories are not curated in a way that makes them searchable and/or does not provide the metadata that would easily allow their inclusion in other studies. As a consequence, the opportunities for large scale studies are limited, and substantial resources (time and money) are wasted on reproducing similar data for different research projects [2] [3], [4], [5] [6].

1.4 Structure of the Thesis

Each chapter presents a brief review of related literature, followed by a description of how each stage has been realized. This thesis is organized as follows;

- In Chapter 2, we provide the review of literature, related to the project.
- In chapter 3, project is described
- In Chapter 4, the Application development is described.
- In Chapter 5, the requirements analysis stage is described.
- In Chapter 6, we describe design and prototyping stage.
- In Chapter 7, the implementation and coding stage is described.
- In Chapter 8, we present hosting and deployment stage of the project.
- In Chapter 9, we described user feedback and their implications.
- In Chapter 10, we provide an overview of the application.
- Finally, some discussion about the future work and a conclusion.

Chapter 2

Literature Review

2.1 Open-access digital repository overview

An open-access repository or open archive is a digital platform that holds research output and provides free, immediate, and permanent access to research results for anyone to use, download and distribute [7]. Digital repositories can fulfill multiple purposes. The main goal is to provide open access to research data, articles, dissertations and support scholarly communication. In addition, they can be thought of as a digital platform where,

- The content creator, owner or third party can share content.
- Content and metadata will be managed by repository architecture.
- Minimum set of basic services are offered e.g., search, access, control, share.
- The safeguarding of data will be supported and well-managed [8].

Repositories can take a variety of forms such as learning object repositories, e-print repositories, institutional repositories, data repositories and support a range of purposes such as publishing, research, learning, records management and preservation.

Thus repositories provide a number of targeted services depending on the purpose of the repository, including:

- Easy access to resources
- new methods of peer review and publication
- corporate information management
- data sharing, including re-use of learning objects, re-use of research data
- preservation of digital resources

Developing a repository application can be patchy, difficult and very hard to maintain. There are some well maintained open access repository software that exist in the market and among them the most popular repository applications are 'DSpace' [9], a turnkey institutional repository application, and 'Fedora' [10], a robust, flexible, open-source repository platform. These software help to expand the amount and diversity of scholarly material that is collected and preserved. But both of the repository software do not offer any data curation process or tool that will help researchers generate, categorize, find, analyse, and share spectral data.

2.2 Data preservation

Data can be described as the elements or units in which knowledge and information are created and metadata are the summarizing subsets of the elements of data; or the data about the data [11]. To prolong and maintain the existence and authenticity of data and its metadata, several strategies can be followed such as digital preservation, archives, catalogs, portals, and repositories. Repositories help archiving data and make sure that all protocols and requirements of storing and holdings are being met to ensure data preservation and user access. Data preservation is a process of maintaining, protecting the integrity and safety of data. It protects data from being lost or destroyed and ensures persistent access to data by planning back-up and recovery tactics, prior to the event of failure and technological change.

Policies and regulations govern formal activities that are used for data preservation. An initiative of the stakeholders within the research process including academics, industry, funders, and scholarly publishers aimed to design and implement a set of principles that are called the FAIR (Findable, Accessible, Inter-operable, and Reusable) Data Principles for preserving data. FAIR data is all about the reuse of data and emphasizes the ability of computers to find and use data. The FAIR Principles ensure that all data be Findable, Accessible, Interoperable, and Reusable. This involves data management the proper collection, notation, and storing of data but also preservation into the future of valuable digital assets [12].

- "Findable means that the data should be able to be found by an appropriate person at an appropriate time.
- Accessible means that the data is accessible either internally through a license or publicly available.
- Interoperable means that the data is formatted in a manner that is standardized and annotated.
- Reusable means that the data has clear usage licenses and is useable by both people and machines" [13].

Our proposed open-access spectral data repository strictly follows FAIR data principles which help digital preservation of spectral data with good data management.It will produce high-quality digital content and can facilitate easy data sharing which simplifies the process of data discovery, reuse, and evaluation.

FAIR data principles make sure that data collected during the research process will be publicly available where possible and if not, such restrictions should be justifiable. Data users are expected to acknowledge the source when they use others' data and acknowledge the right of the data creator to reasonable first use.

2.3 Metadata

Metadata is data that describes other data. Metadata consists of properties, which describe each dataset's entities, and their values [14].

- Entities are particular resources with UUIDs, such as files, cases, samples, and cell lines. These can be the subject of your query.
- Properties can either describe an entity or relate that entity to another entity. For instance, properties include an entity's vital status, gender, data format, or experimental strategy.

Metadata can be used on the Platform to browse and query datasets. It can be compared to effective cataloging, which includes identifying resources, defining them by criteria, bringing similar resources together, and distinguishing among those that are dissimilar. Metadata also facilitates digital identification via standard numbers that uniquely identify the resource the metadata defines [15].

2.3.1 Choosing Metadata

Metadata is an important way to protect resources and their future accessibility. So analyzing the metadata standard that best reflects the content of repository's collection and the needs of the community that will use the data, we have used Dublin core and Darwin core standard in our application. These metadata will help to describe dataset's entities, and their values such as the data source, how it was captured, and what it represents.

2.3.1.1 Dublin Core

The Dublin Core^{TM}, also known as the Dublin Core^{TM} Metadata Element Set, is a set of fifteen "core" elements for describing resources and offer expanded cataloging information and improved document indexing for search engine programs [16].

The 15 metadata elements used by Dublin Core are: title (the name given the resource), creator (the person or organization responsible for the content), subject (the topic covered), description (a textual outline of the content), publisher (those responsible for making the resource available), contributor (those who added to the content), date (when the resource was made available), type (a category for the content), format (how the resource is presented), identifier (numerical identifier for the content such as a URL), source (where the content originally derived from), language (in what language the content is written), relation (how the content relates to other resources, for instance, if it is a chapter in a book), coverage (where the resource is physically located), and rights (a link to a copyright notice).

2.3.1.2 Darwin Core

Darwin Core (often abbreviated to DwC) is an extension of Dublin Core for biodiversity informatics [17]. It is meant to provide a stable standard reference for sharing information on biological diversity. The Darwin Core is primarily based on taxa, their occurrence in nature as documented by observations, specimens, and samples, and related information.

2.3.2 Using Metadata

In order to make datasets available in Nature's Palette repository, it will be curated in a way that will enable them searchable, also storing related metadata will allow their inclusion in other studies. The data will be saved in Nature's Palette repository in two-parts: one is the metadata, and another one is the raw data. The metadata must include a key Darwin's core terms and Dublin core terms.

Terms that start with a lower case letter (e.g., genus) are defined in the Darwin Core; for more information see (https://dwc.tdwg.org/terms/). These terms are standardized and allow our database to crawl museum data to fill the metadata not supplied by the user. Terms that are flagged with an asterix (*) are mandatory. Only 10 fields are mandatory: FileName, UniqueID, genus, specificEpithet,Patch, LightAngle1, LightAngle2, ProbeAngle1, ProbeAngle2, Replicate. The following explains the terms, and offers guidance on what is considered an appropriate entry in Nature's Palette repository.

- FileName*: The name of the file containing the raw spectral data. This name should not contain the extension format (i.e., do not include '.Master.Transmission', '.jaz', '.ProcSpec', '.ttt'). For now, submissions are asked to include only a single file type.
- UniqueID*: A unique identifier for the specimen measured. Can include band or permanent marking number, species info, location, etc... E.g., TRES1015.12345, Buttercup1, VigoCarduelis1
- class: The full scientific name of the class in which the taxon is classified. E.g., Mammalia, Hepaticopsida.
- order: The full scientific name of the order in which the taxon is classified. E.g., Carnivora, Monocleales.
- family: The full scientific name of the family in which the taxon is classified. E.g., Felidae, Monocleaceae.
- genus*: The full scientific name of the genus in which the taxon is classified.

E.g., Puma, Monoclea.

- specificEpithet*: The name of the first or species epithet of the scientific name (the second part of a species name in binomial nomenclature.) E.g., concolor, gottschei.
- infraspecificEpithet: The name of the lowest or terminal infraspecific epithet of the scientific name, excluding any rank designation. For many species, this would be the subspecies-level designation.
- sex: The sex of the biological individual. Use full word (not first letter) e.g., Female - NOT 'F'.
- lifeStage: The age class or life stage of the biological individual at the time of collection. E.g., egg, eft, juvenile, adult, second-year.
- country: The name of the country or major administrative unit in which the specimen was collected. Please use the Getty Thesaurus of Geographic Names-for proper spelling. The list of countries can be found at http://www.getty.edu/vow/TGNNationPopup
- locality: The specific description of the place. E.g., 'Bariloche, 25 km NNE via Ruta Nacional 40 (=Ruta 237).'
- decimalLatitude: The geographic latitude (in decimal degrees, using the spatial reference system given in geodeticDatum, see below) of the geographic center of a Location. Positive values are north of the Equator, negative values are south of it. Legal values lie between -90 and 90, inclusive. If possible, please use WGS84 coordinates.
- decimalLongitude:The geographic longitude (in decimal degrees, using the spatial reference system given in geodeticDatum) of the geographic center of a Location. Positive values are east of the Greenwich Meridian, negative values are west of it. Legal values lie between -180 and 180, inclusive. If possible,

please use WGS84 coordinates.

- geodeticDatum: The ellipsoid, geodetic datum, or spatial reference system (SRS) upon which the geographic coordinates given in decimalLatitude and decimalLongitude as based. E.g., EPSG:4326, WGS84, NAD27. If possible, please use WGS84 coordinates.
- verbatimElevation: The original description of the elevation (altitude, usually above sea level) of the location. Please use a single value without the unit (m). If the speciment indicates a range of values (e.g., between 300 and 400 m), indicate the middle point i.e., 350 m.
- eventDate: The date when the specimen was collected. Recommended best practice is to use a date that conforms to ISO 8601:2004(E). As such, please adhere to the following format: 1809-02-12 (some time during 12 February 1809). 1906-06 (some time in June 1906). 1971 (some time in the year 1971).
- measurementDeterminedDate: The date when the spectral measurement was taken. Recommended best practice is to use a date that conforms to ISO 8601:2004(E). As such, please adhere to the following format: 1809-02-12 (some time during 12 February 1809). 1906-06 (some time in June 1906). 1971 (some time in the year 1971).
- Patch*: The location on the animal that was measured. Some class of animals have standardized morphological descriptions (e.g., the International Committee on Avian Anatomical Nomenclature has published the Handbook on Avian Anatomy). These terms are described in Latin, which is almost never used by non-vetenarians. Therefore, we highly recommend the use of the English names derived from these standardized treaties. In birds, for example, most terms can be found in Chapter 3 of Proctor and Lynch, Manual of Ornithology: Avian Structure and Function. Keep in mind that searches based on the

term 'Patch' are possible.

- LightAngle1*: The angle of the light source in relation to normal (0) in the MEDIAN PLANE with positive values (up to 90) towards the head, and negative values (down to -90) towards the rear of the animal. If measured on a plant or circular animal, values should be explained in the comment section. Measurements taken with a bifurcated probe should indicate '0'.
- LightAngle2*: The angle of the light source in relation to normal (0) in the TRANSVERSE PLANE with positive values (up to 90) towards the right, and negative values (down to -90) towards the left of the animal. If measured on a plant or circular animal, values should be explained in the comment section. Measurements taken with a bifurcated probe should indicate '0'.
- ProbeAngle1*: The angle of the probe in relation to normal (0) in the MEDIAN PLANE with positive values (up to 90) towards the head, and negative values (down to -90) towards the rear of the animal. If measured on a plant or circular animal, values should be explained in the comment section. Measurements taken with a bifurcated probe should indicate '0'.
- ProbeAngle2*: The angle of the PROBE in relation to normal (0) in the TRANSVERSE PLANE with positive values (up to 90) towards the right, and negative values (down to -90) towards the left of the animal. If measured on a plant or circular animal, values should be explained in the comment section. Measurements taken with a bifurcated probe should indicate '0'.
- Replicate*: A value representing different measurement instances of the same patch on the same specimen. If only one measurement was obtained per patch fill the column with the value '1'.
- Comments: Add comments spcific to the area measured if required only. E.g., 'Colour showing sign of fading'

2.4 Color measurement in Visual communication

Communication is an adaptation that helps animals to survive. It can be visual, tactile, auditory or chemical. Animals use communication to identify themselves, attract mates, mark territory, warn off predators.

Many of the decisions in social interactions, mate choice, and intrasexual competition rely on information transfer in visual communication between senders and receivers [18] [19].

2.4.1 Spectral data

Color measurements are often used to address ecological and evolutionary questions. These measurements are found in studies of communication, signaling, color vision research, camouflage, evolution and behavior, and in the examination of environmental, artificial, and biogenic light [20].

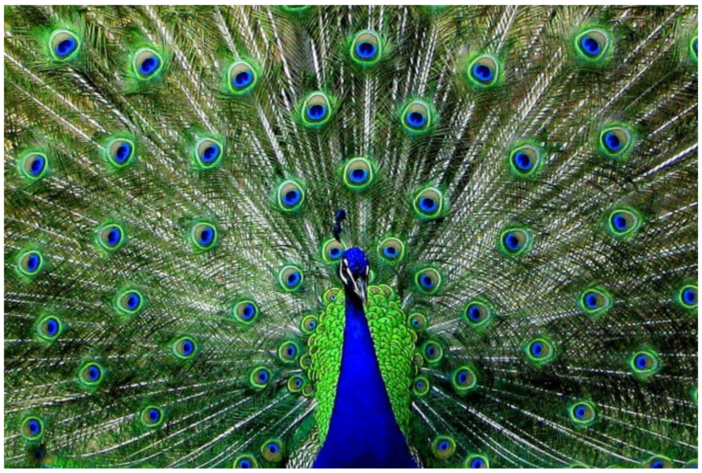


Figure 2.1: Male peacock exhibits a visual display as a part of its courtship rituals

The measurement of color, is mainly measuring the variation in optical properties such as intensity, reflectance and transmittance over a spectral range, the ultraviolet or human-visible portions of the electromagnetic spectrum. The resulting data are used to create an estimate of color as perceived by humans or other animals by integrating into one of the various models of color vision.

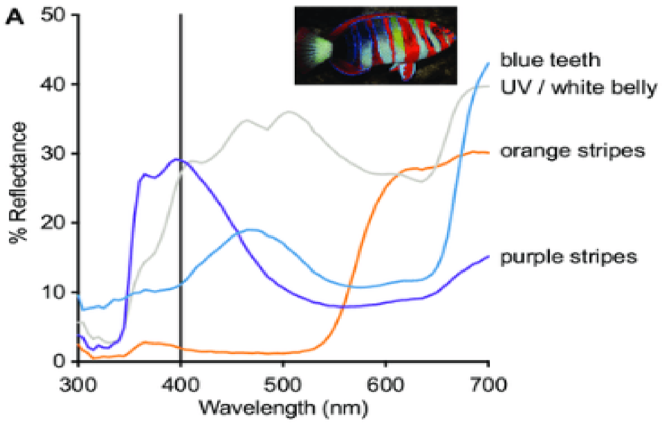


Figure 2.2: Spectral reflectance of specific color patches on ***Choerodon fasciatus***.

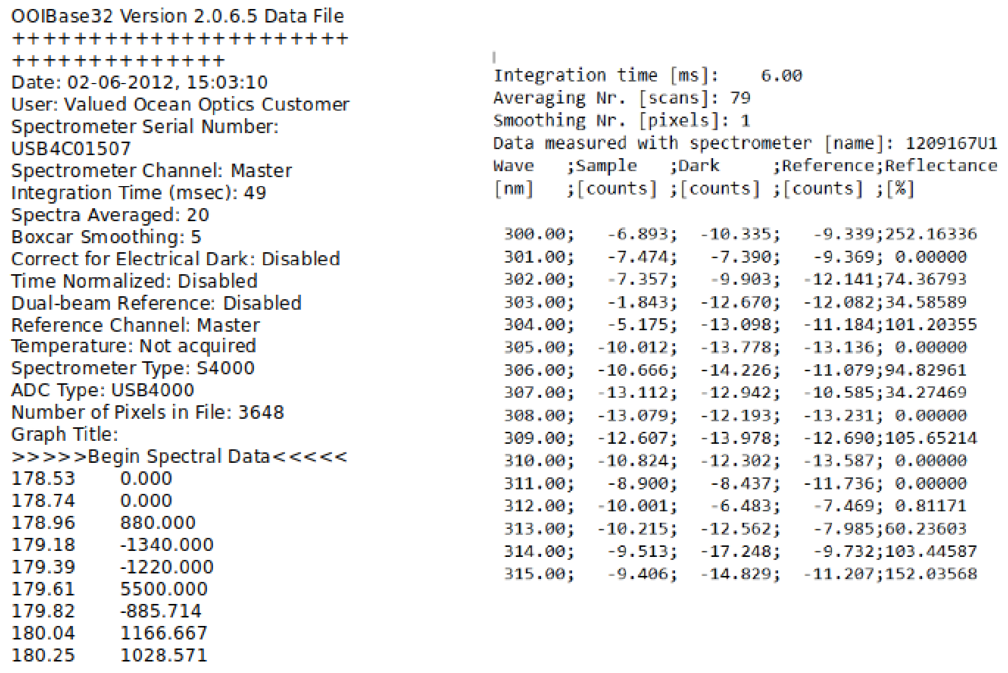
The recent improvements in spectrometers and cameras with a merger of biology and physics make measurements of light and color in biology more common. The growth of the field of visual ecology advanced the understanding of perception and processing of colour measurement and have allowed analysis of reflectance data using visual models that estimate how animals see and differentiate these colours.

2.4.2 Spectral data file

Portable spectrometers have greatly advanced the study of animal coloration. Spectrophotometry generates a lot of spectral data from multiple measures like individuals, patches, within patch. An average research project contains 1000's of raw files and a metadata file (see figure 2.3) that contains details about these raw files.

Spectra files mainly contains two columns X and Y.

- X: Wavelength
- Y: Reflectance or Transmission or Irradiance value



```
OOIBase32 Version 2.0.6.5 Data File
+++++++++++++++++++++
++++++++++++++
Date: 02-06-2012, 15:03:10
User: Valued Ocean Optics Customer
Spectrometer Serial Number:
USB4C01507
Spectrometer Channel: Master
Integration Time (msec): 49
Spectra Averaged: 20
Boxcar Smoothing: 5
Correct for Electrical Dark: Disabled
Time Normalized: Disabled
Dual-beam Reference: Disabled
Reference Channel: Master
Temperature: Not acquired
Spectrometer Type: S4000
ADC Type: USB4000
Number of Pixels in File: 3648
Graph Title:
>>>>>Begin Spectral Data<<<<<
178.53      0.000
178.74      0.000
178.96      880.000
179.18      -1340.000
179.39      -1220.000
179.61      5500.000
179.82      -885.714
180.04      1166.667
180.25      1028.571
```

```
Integration time [ms]:    6.00
Averaging Nr. [scans]: 79
Smoothing Nr. [pixels]: 1
Data measured with spectrometer [name]: 1209167U1
Wave   ;Sample   ;Dark     ;Reference;Reflectance
[nm]   ;[counts] ;[counts] ;[counts] ;[%]

 300.00;   -6.893;  -10.335;   -9.339;252.16336
 301.00;   -7.474;   -7.390;   -9.369; 0.00000
 302.00;   -7.357;   -9.903;  -12.141;74.36793
 303.00;   -1.843;  -12.670;  -12.082;34.58589
 304.00;   -5.175;  -13.098;  -11.184;101.20355
 305.00;  -10.012;  -13.778;  -13.136; 0.00000
 306.00;  -10.666;  -14.226;  -11.079;94.82961
 307.00;  -13.112;  -12.942;  -10.585;34.27469
 308.00;  -13.079;  -12.193;  -13.231; 0.00000
 309.00;  -12.607;  -13.978;  -12.690;105.65214
 310.00;  -10.824;  -12.302;  -13.587; 0.00000
 311.00;   -8.900;   -8.437;  -11.736; 0.00000
 312.00;  -10.001;   -6.483;   -7.469; 0.81171
 313.00;  -10.215;  -12.562;   -7.985;60.23603
 314.00;   -9.513;  -17.248;   -9.732;103.44587
 315.00;   -9.406;  -14.829;  -11.207;152.03568
```

Figure 2.3: There are two examples (right and left) and that in each there is meta-data followed by spectral data

Wavelength (X) includes spectral range usually a few broad spectral regions, the ultraviolet or human-visible portions of the electromagnetic spectrum. The value (Y) usually measured from ambient natural or artificial light sources (irradiance), transmittance properties of the medium (air, water), and reflective properties of animal integuments and the substrates in their environment.

2.4.3 *pavo*: an R package for the analysis, visualization and organization of spectral data

pavo: an R package provides a replicable framework to organize, and analyse spectral data. One of its strength is its ease with which it can help understand how animals perceive these colors, providing important insights into ecological and evolutionary aspects of animal visual communication [see figure 2.4].

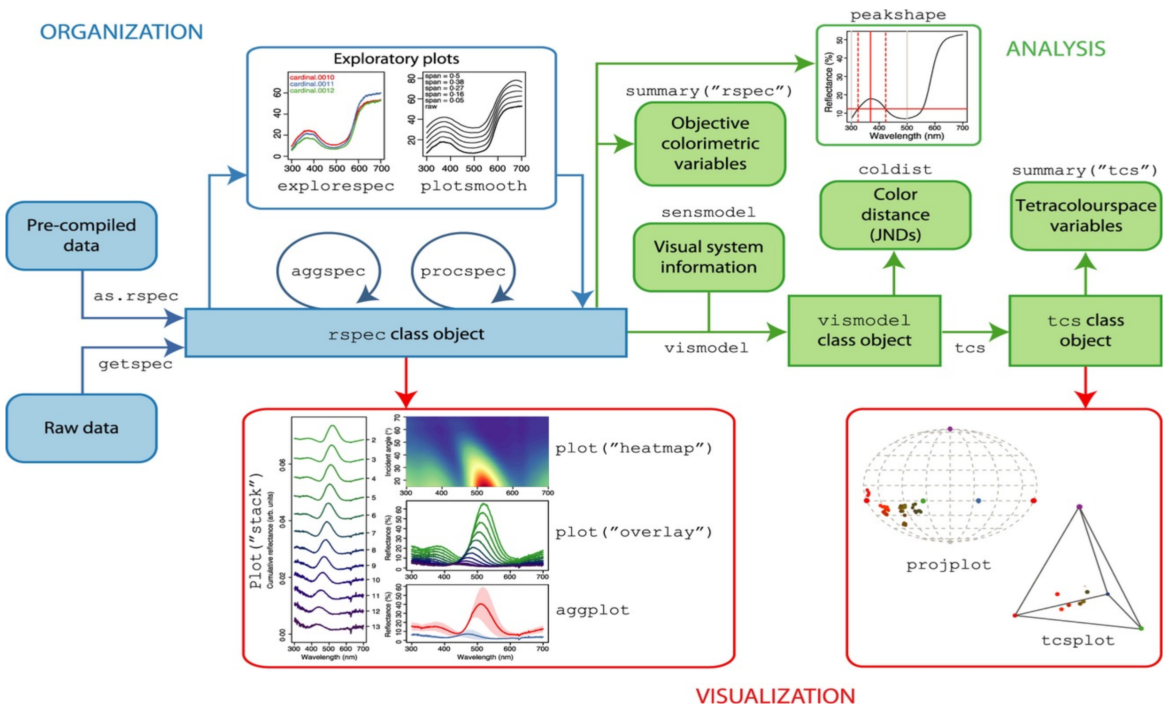


Figure 2.4: Example pavo workflow, highlighting its main functions and plotting capabilities. Adapted from Methods in Ecology and Evolution, Volume: 4, Issue: 10, Pages: 906-913, First published: 22 May 2013, DOI: (10.1111/2041-210X.12069)

pavo is highly flexible and allow users to:

- Organize and manipulate data from a variety of sources.
- Visualize data using R's state-of-the-art graphics capabilities.
- Analyse data using spectral curve shape properties and visual system modelling for a broad range of taxa.

pavo The package is developed for working with spectral and spatial colour data with the goal of establishing a flexible and integrated workflow. It includes functions that take advantage of new data classes to work seamlessly from importing raw spectra and images, to visualisation and analysis [1]. Projects that use *pavo* typically contain huge spectral measurements but not all datasets are easily accessible.

Poor and incomplete metadata collection or lack of quality data some times limit the opportunities for large scale studies. To make these spectral measurements publicly available and use them in large scale studies, we have proposed an open access repository for spectral data.

Chapter 3

About the Project

3.1 Proposed open access digital repository

Our plan is to develop an open access repository that will provide resources of spectral measurements which will encourage large-scale projects through the reuse of existing data. The repository will permit researchers to:

- Upload their data
  - Data will be curated and preserved
- Retrieve data based on complex queries
  - Darwin Core terms
  - Georeferencing
  - Position in a colour space
- Download data related to the query

3.1.1 Upload data

Researcher can upload:

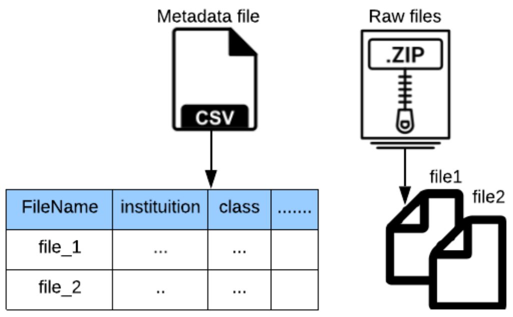


Figure 3.1: Metadata file and raw files format

- A ‘csv’ file with metadata that follows Darwin Core (Darwin Core is an extension of Dublin Core for biodiversity informatics and it is meant to provide a stable standard reference for sharing information on biological diversity) definitions and some database specific terms also e.g., units, angle of measurement, as well as the name of the raw data file it is associated with.
- A ‘zip’ file containing the raw spectral data files (from 1 file to maximum file allowed by process, see figure 3.1).

3.1.1.1 Data validation and analysis

During the submission process, a validation process will confirm that,

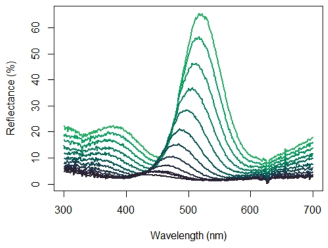
- Column names in the metadata file matches those deemed mandatory by our group.
- File names in metadata file are exact match with raw files name.
- Raw files are not corrupted.

When data submission is completed, curatorial pipeline will initiate the data analysis and metrics calculation from spectral curves using multiple functions from pavo. The curatorial pipeline includes,

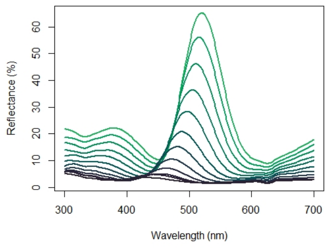
- The extraction of usable spectral data from raw files using ‘getspec’ function.

- The elimination small negative values using ‘procspec’ function.
- Smoothing out electrical noise using ‘procspec’ function.
- The generation of visual model metrics thorugh the ’vismodel’ function and ‘colspace’ function.
- Extract metrics form visual models.

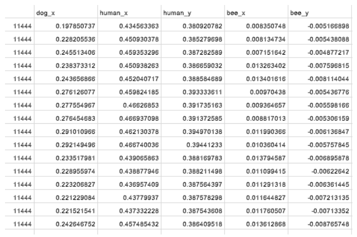
After the curation process, metrics calculated from raw files will be stored and preserved in the database.



(a) Plot-1 using getspec



(b) Plot-2 using procspec



	dog_x	human_x	human_y	bee_x	bee_y
11444	0.197850737	0.434563363	0.380920782	0.008350748	-0.005166898
11444	0.228205536	0.450930378	0.385279698	0.008134734	-0.005438088
11444	0.245513406	0.459353296	0.387282589	0.007151642	-0.004877217
11444	0.238373312	0.450938263	0.386659032	0.013263402	-0.007596815
11444	0.243656866	0.452040717	0.388584689	0.013401616	-0.008114044
11444	0.276126077	0.459824185	0.393333611	0.00970438	-0.005436776
11444	0.277554967	0.46626853	0.391735163	0.009364657	-0.005598166
11444	0.276454683	0.466937098	0.391372585	0.008817013	-0.005306159
11444	0.291010966	0.462130378	0.394970138	0.011990366	-0.006136847
11444	0.292149496	0.466740036	0.39441233	0.010360414	-0.005757845
11444	0.233517981	0.439065863	0.388169783	0.013794587	-0.006895878
11444	0.228955974	0.438877946	0.388211498	0.011099415	-0.00622642
11444	0.223206827	0.436957409	0.387564397	0.011291318	-0.006361445
11444	0.221229084	0.43779937	0.387578298	0.011644827	-0.007213135
11444	0.221521541	0.437332228	0.387543608	0.011760507	-0.00713352
11444	0.242646752	0.457485432	0.386409518	0.013612868	-0.008765748

(c) Calculated metrics

Figure 3.2: Plots describing the curation proccess

3.1.2 Query repository

Researchers can request to query the repository.

- Researchers can search the repository using the Darwin Core search terms or terms specific to database.
- Based on search query, metadata of filtered raw files will be shown
- In display table, specific metadata fields will be displayed.

3.1.3 Download Data

The filtered result can be downloaded, if the researcher selects Download. Retrieval of data has to supply the metadata file along with raw files for query results only.

Download will generate a package file consisting of:

- One file in tabular format containing all the metadata of the raw files relevant to query result.
- All raw data files identified by the query.
- A file with all the submission information.

3.2 Technical overview

Digital repository systems have similar technical structure. The database is the heart of a digital repository. The repositories built on databases ensure long term support, flexibility, and easy migration process.

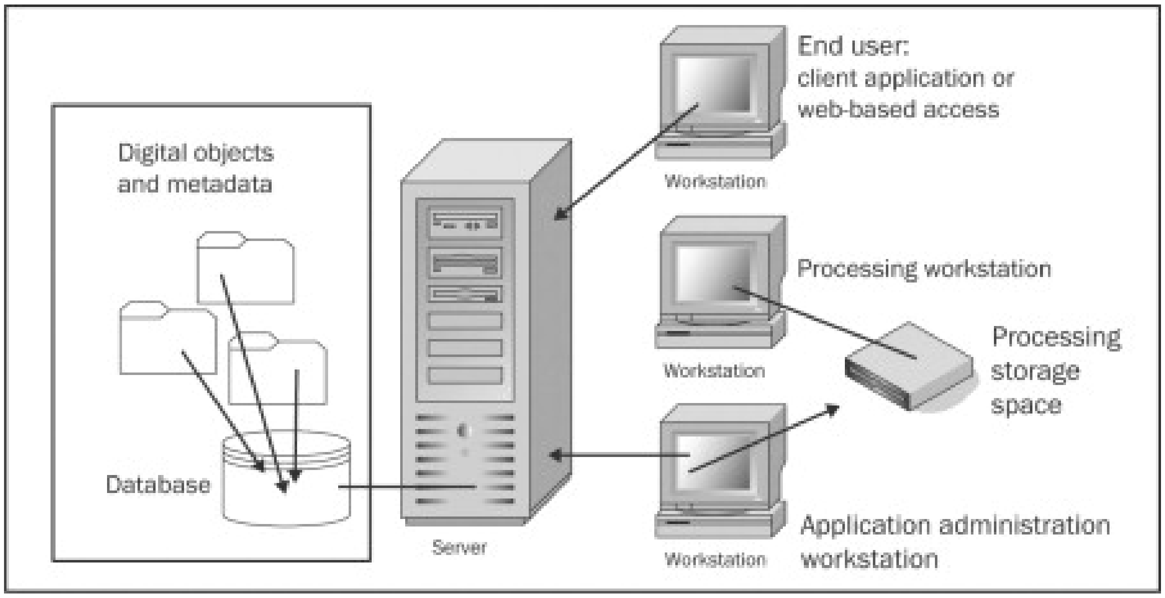

Digital objects and metadata
Database
Server
End user: client application or web-based access
Workstation
Processing workstation
Workstation
Processing storage space
Application administration workstation
Workstation

Figure 3.3: 'Systems architecture,' a diagram demonstrating the different systems involved.

Data needs to be accessible and presentable to the user when necessary. So a user interface is used to retrieve information from the database through the application server. The database server could be different from the application server which depends on the local environment (figure 3.3). It is recommended to use at least two installations, one running on a production server and another on a test server. The end-users will interact with the production server whereas testing and development will occur on the testing server. Any new update or fix should be tested on a testing server before releasing it for production. Instead of a physical machine, virtual machines can be used to keep the cost down.

3.3 Related works

We are aware of a single repository that shares similarities with the proposed project. FReD, the Floral Reflectance Database (see http://www.reflectance.co.uk/)., provides spectral reflectance data of the same nature as those that would be hosted on Nature's Palette. However, FReD contains 2231 spectra from 200 species of plants after more than 6 years of existence. We anticipate that our repository will hold 500,000 spectra from 500+ species by the end of the pilot phase. Within a few years, this repository will hold millions of measurements. Furthermore, FReD only archives plant reflectance data whereas our repository will accept reflectance, transmittance, irradiance, and radiance data from all plant, animal, medium (air, water), and light environments. This will increase the appeal and value of the repository. Finally, FReD does not offer a curation service and only provides one analytical search function; users can select spectra in the database that span a certain area of the bee colour space e.g., all spectra that a bee would perceive as red. In contrast, Nature's Palette will provide a curatorial pipeline thus removing the largest barrier to spectral data sharing, and offer search filters that will allow the use of spectral data in novel ways.

Chapter 4

Application Development

Our target is developing a pilot digital repository software and for that, we must follow certain procedures while developing the application. Software development is the process of creating, designing, deploying and supporting applications that involves a set of computer science activities. It is a set of instructions or program that executes specific commands and tells the computer what to do. As it is independent of hardware, it makes computer programmable. We can categorize software in four basic types:

System Software: Usually provides core functionalities such as hardware management, utilities, disk management, operating systems, and other operational necessities.

Programming Software: These are tools such as compilers, text editors, debuggers, linkers and others provided to the programmers to create programs.

Application Software: This can be referred to as web, or mobile application and help users to perform particular tasks, such as data management software, media players, office productivity suites.

Embedded Software: This type of software control machines and devices like telecommunication networks, industrial machine, and robots, cars. These devices, and their software, can be connected as part of the Internet of Things (IoT) [21].

Our open access digital repository of spectral data is a web based application software, which allows researchers to upload their spectral data, retrieve data based on complex queries using Darwin Core terms. The primary goal of a digital repository is to ingest, store, manage, preserve, and provide access to digital content.

Development of such kind of application includes processes such as initial research, data flow design, process flow design, flow charts, technical documentation, software testing, debugging and other architecture techniques (SDLC). This is known as the software development life cycle [22].

4.1 Software development life cycle (SDLC)

The software development life cycle aims to produce the highest quality software with the lowest cost over the shortest time. SDLC [23] ensures a detailed plan for software development like how to develop, update, maintain and test a software system to guarantee quality. In SDLC there are several distinct stages like planning, requirement gathering, and analysis, design, implementing, testing and deployment. There are various SDLC models that follow the variation of these stages.

4.1.1 How SDLC Works

SDLC helps to remove the typical difficulty of software development by following a specific plan to achieve divergent goals. In the initial stage, the plan starts by assessing the existing system for deficiencies. In the next stage, the requirement of the new system will be created. After analyzing the requirement, the software is developed through the stages of design, implementation, testing, deployment. SDLC can eliminate redundant rework and fixes errors by anticipating costly mistakes.

Typical software engineering phases are:

- Requirements analysis
- Design and Prototyping
- Implementation and coding
- Testing and Deployment
- Maintenance

By following the above list or a combination of these ensures the software development process works in an efficient productive and smooth way.

4.1.2 Selecting a methodology

SDLC model helps to establish a framework in which the steps of software development are applied. It describes an overall plan and work process for the project. Some factors to consider when deciding the model include:

- The size of the project and team
- Time-frame and deadlines to meet
- How complex the execution will be
- Clients, their availability, and their relationship with them

Based on those factors, we have used the Agile model [24]. The whole team is responsible for the whole process during the project and worked closely with each other. The client was involved during each step of the project and communicated feedback accordingly. In our case we leveraged a class and client was active in the guidance process.

4.2 Agile Methodology

The agile model is a blend of incremental and iterative process model. Its main focus is customer satisfaction and process adaptability by rapid delivery of working software products. The product is broken down with small incremental builds which results in small iterative releases with each release building on previous functionality. Each release is tested to maintain software quality. Typically each iteration can last from one to three weeks and involves simultaneous work on various areas like planning, requirements analysis, design, implementation, testing, and deploying.

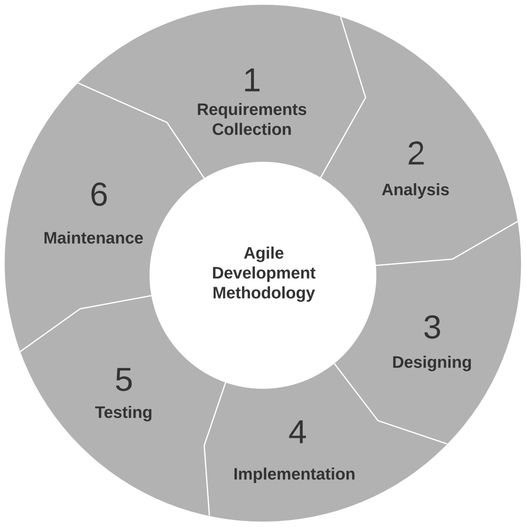


Figure 4.1: Agile Model

4.2.1 The Agile Iteration Workflow

The iterative process dominates the agile software development life cycle. Each iteration delivers working builds and supporting elements. Each iteration has fixed completion time and due to this multiple iterations will take place during the life cycle and each follows its own workflow. It is important that the stakeholders provide feedback to ensure that the features meet their needs during each release. Based on our needs we have followed the following iteration process:

4.2.1.1 Planning and Requirement Analysis

Our plan was to deliver the prototype within a short period of time. So identification of the risk associated with the project and quality assurance requirement is also done in the planning stage. With the technical feasibility study, we successfully defined various technical approaches which helped us to implement the project with minimum risk. Then we defined the requirements for the iteration based on the product backlog, stakeholder feedback. During this phase, we methodically analyzed the potential requirements of the application. Once the system is analyzed next step is to properly generate the models and business logic that will be used in our application.

4.2.1.2 System Design

In the system design stage, we worked with technical requirements such as system architecture, application platform, programming language, services, etc. A design specification was created that shows how the domain logic defined in analysis will be technically implemented.

4.2.1.3 Implementation

The actual source code is finally written in this stage. We implemented all models, service interactions, and business logic which were all specified in the prior stages. We have used version control system GitLab to record changes to a file or set of files. Versioning helps to keep track of application builds and identifies which version is currently in development, QA, and production.

4.2.1.4 Testing

After testing each unit, all units that were developed in the previous phase are integrated into the system. To find out any faults and failures, the post-integration of the entire system was tested. It is common at this stage to repeat and rework on some previous coding phases to fix the bugs found in the testing process.

4.2.1.5 Delivery and Feedback

Once the testing process is done, we integrated and delivered the working iteration into production. During this stage stakeholder usually gives the feedback to work into the requirement of the next iteration.

4.2.1.6 Maintenance

Maintenance is required to keep the application functional and up-to-date. There can be some issues that usually come up in the client environment. To fix those issues, patches are released. Maintenance is done during the post-production period to deliver the changes in the customer environment.

4.2.2 Agile Model - Advantages

During our product development process, the main advantage of using agile methodology was the ability to feed additional features into the product backlog as the rest of the process is a matter of repeating the steps again and over until product backlog is cleared and all items in the requirement stack are fulfilled.

The agile process uses an adaptive approach and is suitable for feature-driven development. With each iteration, we adapted to the changing product requirements. Through the release of iterations and minimizing the risk, the application was tested very frequently to avoid major failures in the future.

Our application development demands more involvement of the stakeholder during the whole process and agile is the perfect model for that. This model allows teams to work in close collaboration.

In the next chapters, we have described in detail the important phases of the SDLC model involved in the development of our Nature's Palette application. Each chapter demonstrates different phases of the development life cycle and covers the detailed plan for building, deploying, and maintaining the application.

Chapter 5

Requirements Analysis

Our first step towards developing the application was to define the expectations of stakeholders on this project. After researching and discovering the requirements of a system from stakeholders, the next step was the requirement analysis which is a significant and essential activity. To make it consistent and unambiguous, we analyzed, refined, and scrutinized the gathered requirements. Once the analysis was done, it provided a graphical view of the entire system which improved project understandability significantly.

5.1 Requirements specification

Software requirement needs to be implemented first in the system. There are two types of requirements, functional and non-functional requirements.

5.1.1 Functional requirements

Functional requirement defines a service that the system must offer to the user. For example, the functional requirement in context to our application will be when the customer selects "Search" they must be able to see their search result. The functional requirements for our project are:

- Allow users to upload raw measurement files and the metadata associated with these files.
- Verify that the proper metadata are available and match raw data files.
- Compute visual modelling metrics from the raw files.
- Allow users to query the database using three approaches. Through:
  - Metadata (subset of Dublin Core and Darwin Core terms), using Boolean logical operators.
  - Georeferencing of specimen provenance using polygon limits on a map.
  - Regions within the color-spaces of key models, made possible through the calculation of visual model metrics.
- Retrieval of data has to supply metadata along with raw files and also related submission information for query results only.

5.1.2 Non-Functional requirements

Software requirement can also be a non-functional, it describes the operational requirement. For example, a non-functional requirement is how responsive the web application is. We identified the following non-functional requirements for our project:

- Ease of operation: The interface will be intuitive.
- Findability: Data and supplementary materials will have sufficiently rich metadata and a unique and persistent identifier.

- Flexibility: We have a current idea of search terms and functions users want to use but these currently preferred options may change over time. We need to consider the future addition of search terms and visual model metrics – even those that would be user-defined.
- Accessibility: Metadata and data are understandable to humans and machines. Data will be deposited in a trusted repository.
- Interoperability: Metadata uses a formal, accessible, shared and broadly applicable language for knowledge representation.
- Reusability: Data and collections will have a clear usage license and provide accurate information on provenance.
- Scalability: We estimate that 1 million files will be uploaded over the first year of service but could then support an additional 1-3 million files a year.
- Data Integrity: Completeness, accuracy and consistency of the data.

5.1.3 Target environment

- All users should be able to upload, download and search data with a web browser.
- Also, administration functions (e.g., granting access, adding new search terms) should be available through the web.

5.1.4 Use Case Diagram

Once we have the functional and non-functional expectations, we need to define the list of actions or event steps with help of use cases. A use case diagram is a structure for documenting all the requirements for a system. It summarizes some of the connections and relationships between use cases, actors, and systems. The order is not shown in the diagram in which steps are followed to achieve each use case goal.

Typical use case contains,

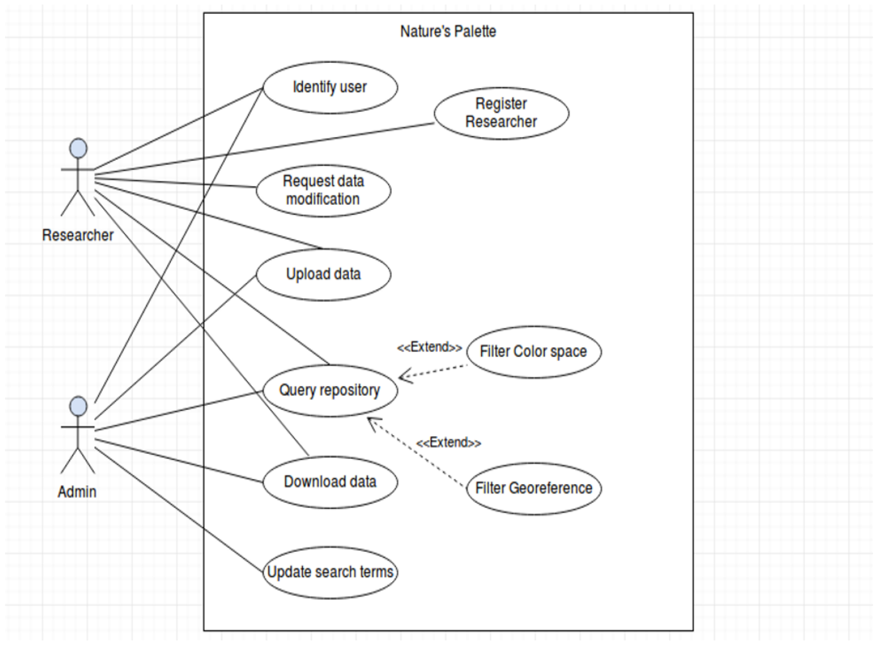


Figure 5.1: Use Case Diagram: two types of actors have extensive access to all functions

Actor: An actor is a person, organization, or external system that is an entity that interacts with the system.

Use case: A use case represents a distinct functionality of a system.

Connection: Use case diagram shows connection and relationships between use cases, actors, and systems. An association exists whenever an actor is involved with an interaction described by a use case. Extending use cases are conditional.

In general, use case diagrams are used for analyzing the requirements and capturing the functionalities of a system.

In our use case diagram, we have two actors, Administrators (admin) and Researchers. There are a total of nine use cases that represent the specific functionality of our repository system. Each actor interacts with particular use cases. A researcher actor can log in or register them-self into the repository, upload data. Once data is submitted, researcher can request for data modification, and can query repository for spectral data using filters. Further, the query can be extended using color space and georeference. Researcher can download data related to the query.

Researcher can perform only these interactions with the system even though other use cases are remaining in the system. It is not necessary that each actor should interact with all the use cases, but it can happen. The second actor named admin can interact with all the functionalities or use cases of the system. This actor can also update the search terms of the repository.

In summary, use case modeling helps us design a system from the end user's perspective. It describes the interactions between the users and the system, the user's goals, and the system's behavior in satisfying these goals.

5.1.5 Use Case Description

A use case description depicts how users will perform tasks on a website. To identify, clarify and organize system requirements each use case is represented as a sequence of steps. It begins with a user goal and ends with the goal being met. It provides a set of possible scenarios that describe the interaction between systems and users in specific environments to achieve a particular goal.

Use cases typically describe a combination of the following elements:

- Actor – anyone or anything that initiates an interaction with system.
- Entry Condition – The cause that triggers the event to initiates the use case.
- Main success scenarios [Flow of Events] – use case in which nothing goes wrong.
- Exit Condition – It is the condition to be satisfied to complete the use case.
- Alternative paths [Alternative Flow] – these paths are a variation on the main

theme. When things go wrong at the system level these exceptions are executed.

In next section we will be providing our main use cases which will describe possible interaction between the user and the system.

5.1.5.1 Identify User

Name:	Identify User
Participating actor:	Researcher
Entry condition:	Researcher is a member of the online repository.
Flow of events:	1. Researcher selects ‘Log in’. 2. System displays the log in screen. 3. Researcher enters their identification details. 4. System validates entered information. 5. The System authorizes the researcher
Exit condition:	Researcher is identified.
Alternative flows:	4a. System finds errors because of invalid information. - System informs Researcher about errors - Researcher acknowledges the error message. - System reverts to step 3.

Alternative sign in or registration using ORCID: Enabling researcher to register or sign into the system using their ORCID credentials. This can save their time and effort also they don’t have to keep track of multiple usernames and passwords.

5.1.5.2 Register Researcher

Name:	Register Researcher
Participating actor:	Researcher
Entry condition:	Researcher is not a member of the online repository.
Flow of events:	1. Researcher selects 'Register'. 2. System displays the registration screen. 3. Researcher enters their identification details. 4. System validates entered information. (See registration validation rule RVR-1) 5. Systems stores the details and creates a new account. 6. System notifies Researcher about successful registration.
Exit condition:	Researcher is registered.
Alternative flows:	4a. System finds errors because of invalid information. - System informs Researcher about errors - Researcher acknowledges the error message. - System reverts to step 3.

5.1.5.3 Upload Data

Name:	Upload Data
Participating actor:	Researcher
Entry condition:	Researcher must be identified.

Flow of events:	1. Researcher requests to upload data. 2. System presents submission instructions and conditions with metadata file templates for researcher. 3. Researcher familiarizes themselves with the instructions and accepts the conditions. 4. The System prompts for information about the submission. 5. Researcher fills the basic information (mostly Dublin core) related to the submission. 6. System validates entered information. (See validation rule UVR-1) 7. System requests Researcher to select files to be submitted 8. Researcher provides metadata file (template is provided) and raw files (archive format) and submits the data. 9. System validates files. (See validation rule UVR-2) 10. System uploads the data to the repository (without releasing them) 11. System notifies Researcher about successful submission. 12. System computes metrics for uploaded raw files (See validation rule UVR-3). 13. System stores the calculated metrics in the repository and releases the data.
Exit condition:	Files are released to the research community for search and download.

Alternative flows:	6a. System finds errors during validation: • System informs Researcher about specific errors • Researcher acknowledges the error message. • System reverts to step 4. 9a. System finds errors during metadata validation. • System informs Researcher about specific errors • Researcher acknowledges the error message. • System reverts to step 7. 12a. System finds errors during metric calculations: • System informs Researcher about specific errors • Files without errors: System continues to 13 • Files with errors are not released -System informs Researcher that ‘Request data modification’ needs to be performed

5.1.5.4 Query Repository

Name:	Query Repository
Participating actor:	Researcher
Entry condition:	Researcher requested to query the Repository.

Flow of events:	1. System presents an advanced search interface. 2. Researcher enters search terms (Darwin Core) for the query and submits the request. 3. System performs the search. (See query rule QR-1) 4. System retrieves and returns the metadata that matches the query. 5. Researcher selects advanced search criteria to further refine the results. NOT IMPLEMENTED FOR NOW. 6. System provides the refined metadata for the raw files that match the filters. NOT IMPLEMENTED FOR NOW.
Exit condition:	Metadata of raw files that match Researcher's query are displayed.
Alternative flows:	4a. Search results not found based on search terms - System displays a message to the Researcher that no matching results were found. - Researcher returns to step 1.

5.1.5.5 Download Data

Name:	Download Data
Participating actor:	Researcher
Entry condition:	Researcher performed successful search query and related metadata for the files is displayed.

Flow of events:	1. Researcher initiates download. 2. System retrieves the relevant raw files. 3. System generates metadata file containing all the metadata values for each raw file. 4. System collects metadata file and all raw files into a single package. 5. System initiates the download process and provides the single package file to the Researcher.
Exit condition:	Package file is downloaded.
Alternative flows:	5a. Download interrupted - System informs Researcher that download was unsuccessful. - System offers to retry the download - Researcher confirms the retry. - System returns to step 5.

5.1.5.6 Update Search Terms

Name:	Update Search Terms
Participating actor:	Researcher
Entry condition:	None

Flow of events:	1. Admin initiates update search terms. 2. System presents all the search terms. 3. Admin selects/ unselects the terms to enable or disable for filter search. 4. System validates search terms. (See search terms update rule UR-1) 5. Admin submit modified search terms to the system. 6. System updates the repository. 7. System provides confirmation message.
Exit condition:	Search terms are updated for filter search.
Alternative flows:	4a. System finds errors and notifies Admin. - System informs Admin that update was unsuccessful. - System offers to retry the update - Admin confirms the retry. - System returns to step 3.

5.1.5.7 Request Data Modification

Name:	Request Data Modification.
Participating actor:	Researcher
Entry condition:	Researcher already has the submission in the repository for which researcher wants to request data modification.

Flow of events:	1. Researcher requests data modification. 2. System presents modification instructions and conditions. 3. System also provides customized metadata file template. (See Data modification example package) 4. Researcher familiarizes themselves with the instructions and accepts the conditions. 5. System presents previous submission list. 6. Researcher selects specific submission that they want to modify. 7. System asks for the package that has the modified files. 8. Researcher provides the package which includes - New metadata file. - New raw files (if needed). (See Data Modification README file) 9. Researcher uploads the package 10. System validates the package. (See data modification rule DMR-1,2,3,4) 11. System notifies Researcher about successful modification. 12. System computes metrics for uploaded raw files (Same as UVR-3). 13. System stores the calculated metrics in the repository and releases the data.
Exit condition:	Researcher is registered.

Alternative flows:	4a. System finds errors and notifies Admin. - System informs Admin that update was unsuccessful. - System offers to retry the update. - Admin confirms the retry. - System returns to step 3.

5.2 Domain Modeling

Domain model provides an overview of our application domain by describing the objects and classes inside the domain and the relationships between them and the operations and attributes of the classes. The Unified Modeling Language (UML) Class diagram is a graphical notation used to construct and visualize object-oriented systems. A class diagram in the Unified Modeling Language (UML) [14] is a type of static structure diagram that describes the structure of a system by showing the system's:

- classes,
- their attributes,
- operations (or methods),
- and the relationships among objects.

An object is any person, place, thing, concept, event, screen, or report applicable to your system. A class is a representation of an object and, it is simply a template from which objects are created.

5.3 Class Diagram Analysis

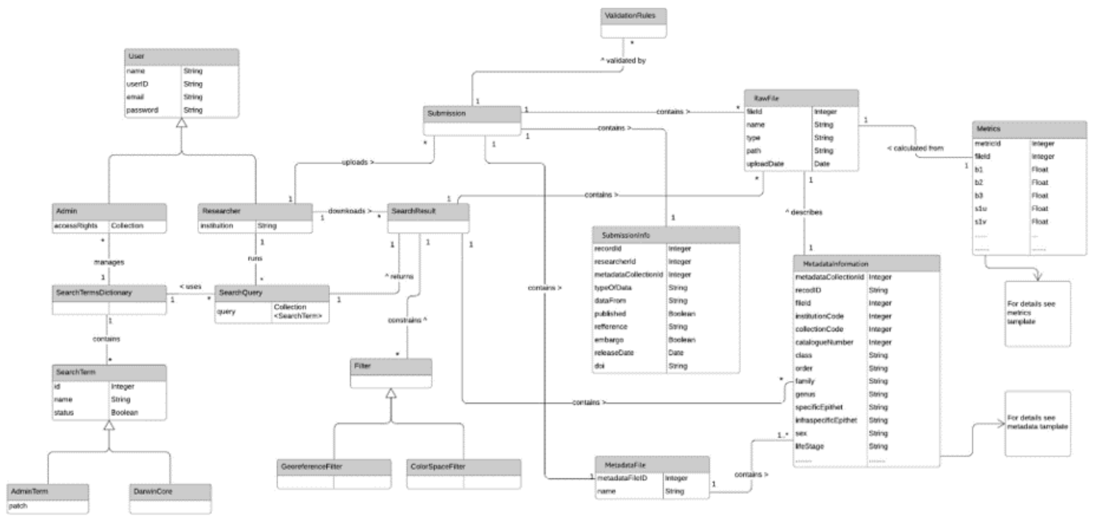


Figure 5.2: The UML class diagram of Nature's Palette application

In our domain model, we have an Admin class and Researcher class. So the User class captures the similarities between these two classes and both classes inherit their properties from user class. Admin and researchers are associated with other classes. Admin can manage SearchTermDictionary which contains SearchTerm that captures the similarities between admin defined terms and Darwin core terms.

We can see Researcher class has an association with Submission, SearchResult, SearchQuery and has one to many cardinalities with them. When researcher submits data we can see, Submission contains Rawfile, SubmissionInfo, MetadataFile and has one to many cardinalities with these cases. Metrics class is associated with Rawfile with one to one cardinality. That means metrics will be calculated when raw files is submitted. The main purpose of the diagram is to show and explain repository concepts, objects and their relationships (see figure 5.2).

5.4 Sequence Diagram

UML sequence diagrams help us to document and validate our logic by modeling the flow of logic within our system in a visual manner, and are commonly used for both analysis and design purposes.

Sequence diagrams illustrate the interactions between objects in a single-use case. Every use case has a corresponding boundary, entity, and control classes. When a particular use case is executed, the sequence diagram demonstrates how the different parts of the system interact to carry out a function and the order in which the interactions occur. Figures 5.3, 5.4, 5.5 illustrate three main sequence diagram of our system. For details please see Appendix A.

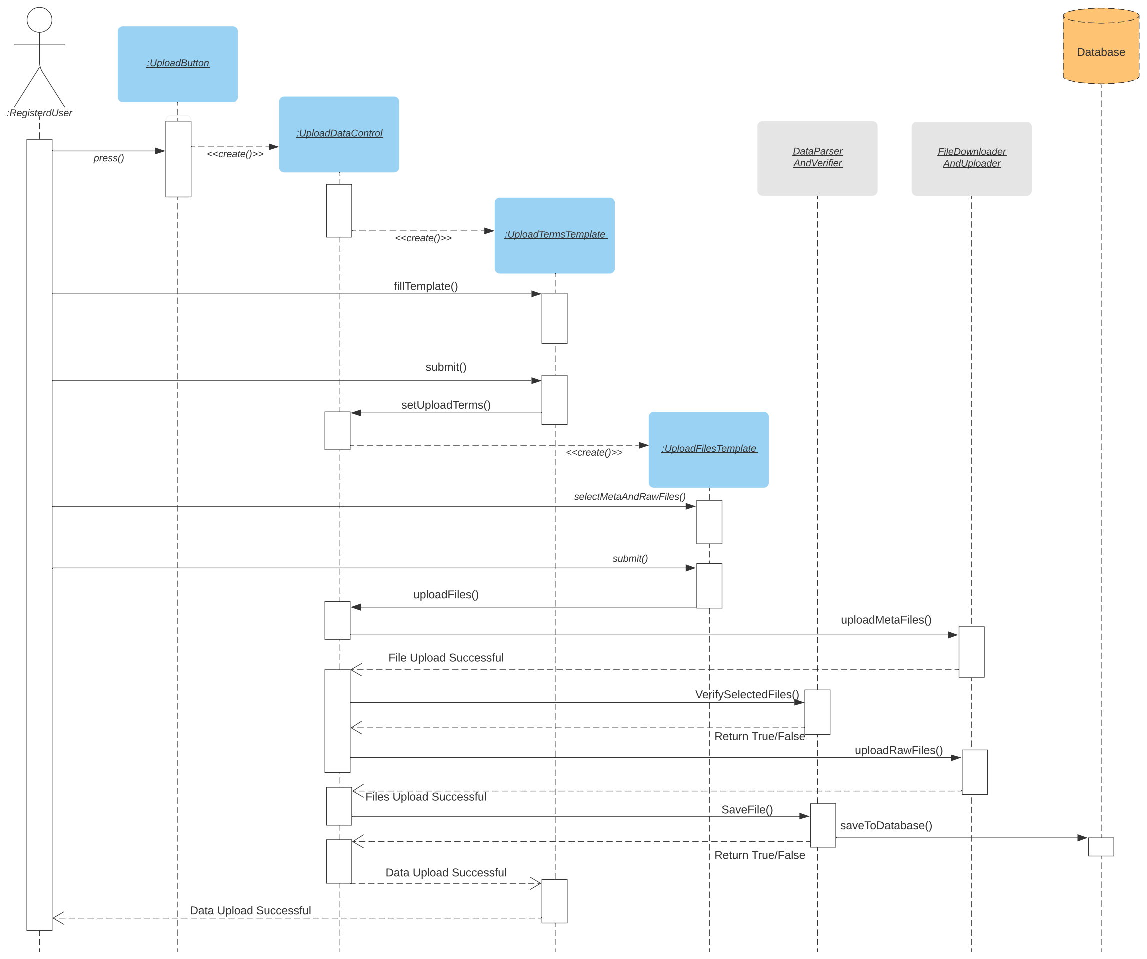


Figure 5.3: UploadFiles Sequence Diagram

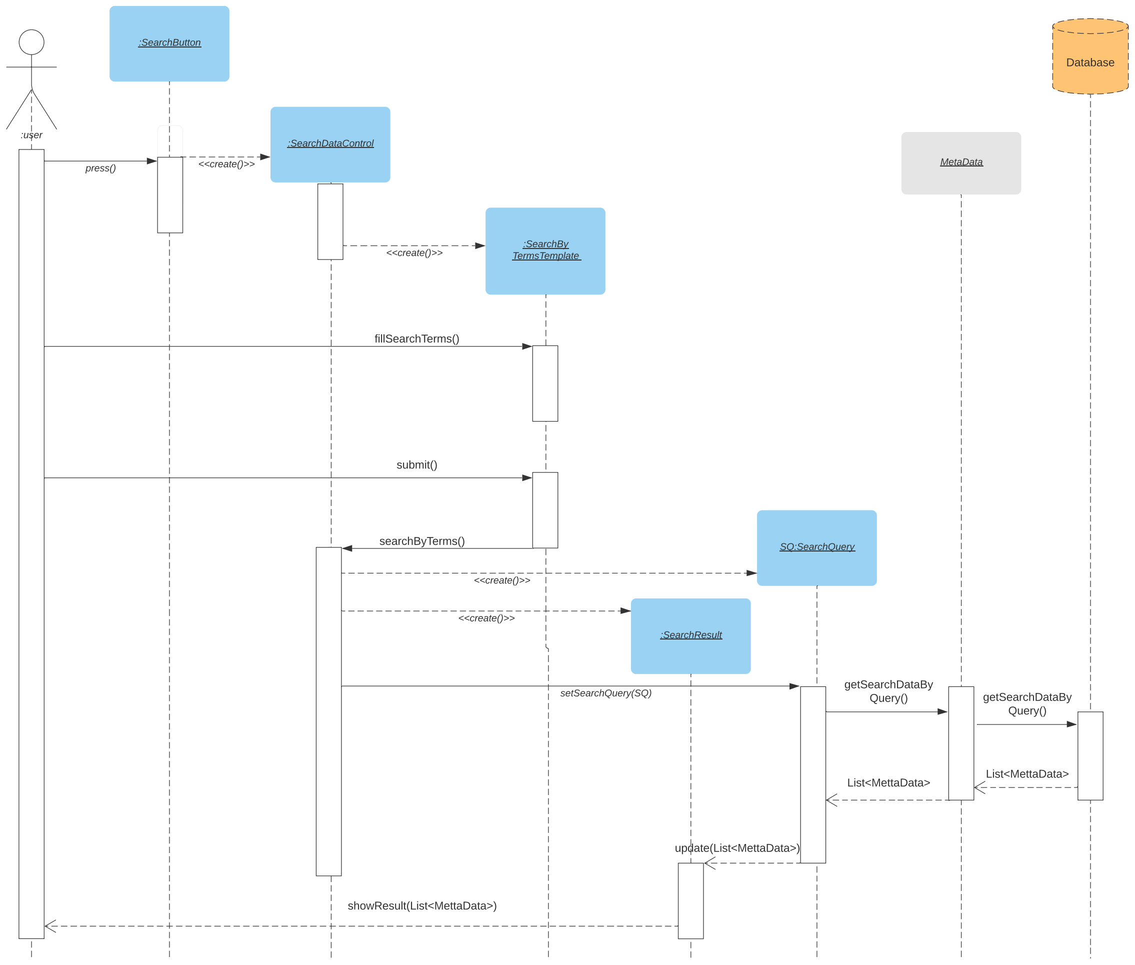


Figure 5.4: SearchData Sequence Diagram

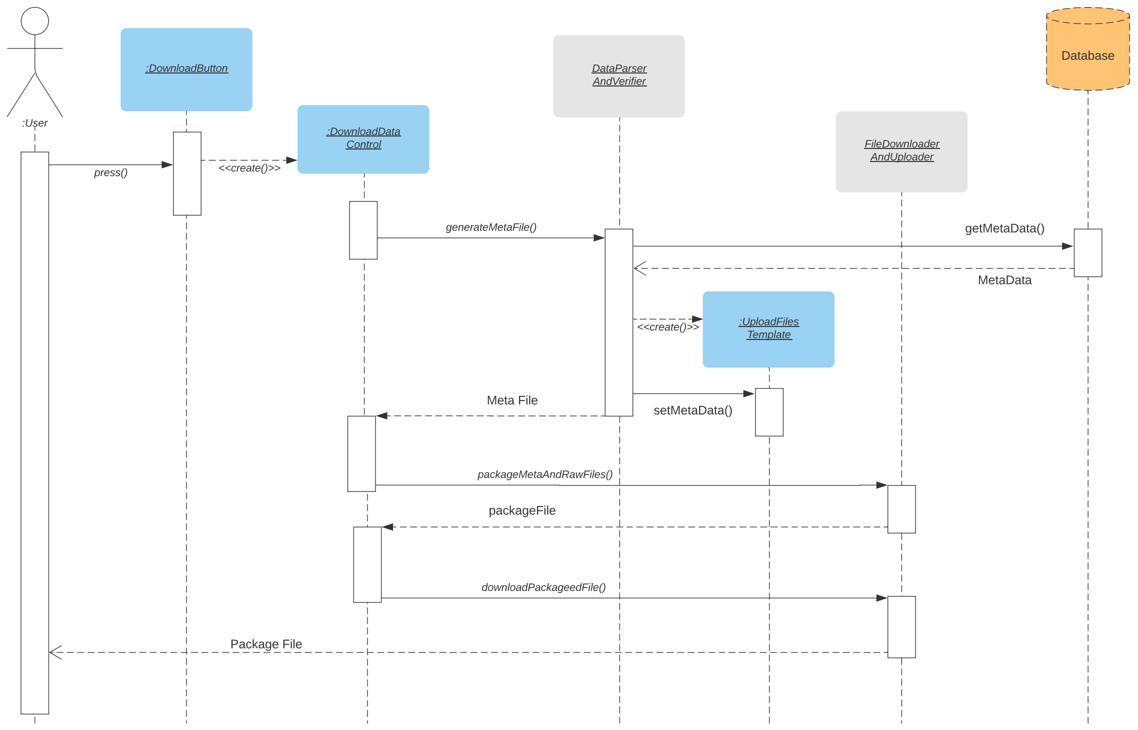


Figure 5.5: Download Sequence Diagram

Chapter 6

Design and Prototyping

After gathering the requirements and analyzing the use cases and domain model, software architecture is derived and used for implementing our application. During this phase, we ensure the design goals are meet and application is able to deliver each requirement efficiently.

6.1 Design Goals

In the initial stage, we defined our design goals to help us stay focused on the driving principles of our project. Design goals give clear direction, purpose, intent, and serve as a quality check by making sure the designs met the intended goals. The following are the main design goals of Nature's Palette project.

6.1.1 Ease of operation

We have used a simple interface for all of the system functions; the main screen will give the user simple access to the system main functionalities such as the search, upload and registration functions. User interface is friendly and familiar; we used Bootstrap, which produces simple, intuitive, responsive interfaces, and is one of the most popular front-end framework, so the system's look and feel very familiar to most users.

Long processes such as file uploading and downloading is provided to users with the ability to cancel at any time. All long processes such as uploading, downloading or parsing data are done asynchronously within the system in order to maintain responsiveness of user interface (UI) at all times. Most system operations is accessible without any form of registration, users will be able to search and download any data right away.

6.1.2 Flexibility

When designing the system special care was taken. To increase flexibility, modifiability and maintainability of our system, we decoupled the user interface from the business logic. The View (Client) run on the user's system, it sends requests to the controller (Server) using http requests such as a GET or POST request.

Our core system functionalities reside in the Controller and Models. We have used Object Oriented style to create the various modules and functionalities within our system. This reduces system complexity by mapping real world objects to system objects, offer easier testing, maintainability, troubleshooting, and error handling.

Whenever possible, high cohesion was kept to maximum within classes and modules, so that each class/module has a specific purpose and any extra responsibilities is delegated to other classes/modules. For example a special module is developed to handle all operations relating to data parsing and verification, another one for upload/download processes. This makes future modifications to the system simpler.

Another example is the creation of a special class to calculate the visual data metrics, this can be easily extended with new metrics definitions and gives the ability to allow the addition of user-specified visual metrics to system in the future. Low coupling was also an important issue, we made use of the Publish-Subscribe architectural style to propagate change events from the Model to the Controller and View, so that the model does not directly know or depend on neither the Controller nor the View, offering high modifiability.

We have provided an interface for the admin to easily add new search terms to the system, all Meta Terms will be defined and stored in a special class that will also handle the verification of search and upload terms. Such setup will make it very simple for the administrator to add, remove or change any mandatory search or upload terms, or specify a set of accepted terms within they system in just a few clicks and the effects will be applied immediately.

6.1.3 Scalability

Scalability affected many of the design decisions in the system architecture. We will use the node.js platform to develop the system, which allows for horizontal scaling right out of the box by running multiple instances of the application on different cores or different servers. We can scale the application very efficiently in the future depending on load demands. This works by creating a master (load balancer) node that creates multiple instances of the application on multiple cores, receives incoming requests and distributes them on these instances. It also allows to add another level of scaling if needed in the future by running the system on multiple machines with a load balancing node distributing the load between them.

Another design decision to facilitate scalability was to keep the database decoupled from the application, this is why we chose a three layered system, where our database will be deployed on separate servers which will allow us to scale the application and the database separately as loads demand.

6.2 System Design and Architecture

After analyzing the use cases and class diagram, the system was carefully designed to meet the design goals, functional and non functional requirements.

6.2.1 Logical Design

Functional requirements are met in logical design. There are 3 logical components; Presentation (view), Controllers, and Models, plus the Database. All User Interface responsibilities are assigned to the Views, all user interaction responsibilities are assigned to the controllers, and all data and business logic are assigned to the models. Each of our component, we have an explicit interface so that other component can send requests through this API, this will allow us to change each of their implementation or create new implementations without affecting the others.

The controller works as an intermediary that handles all requests between Views and Models. It handles all requests from the view and sends appropriate requests to the model, so that the view and model do not directly communicate, and no direct dependency is needed. The controller binds the View and the Model at run-time, so that multiple views can be used for the same model.

6.2.2 Physical Architecture

We have designed the physical layout of our system and its components by using three-layers architecture (see figure 6.1). The layers are Client Machine (Web-

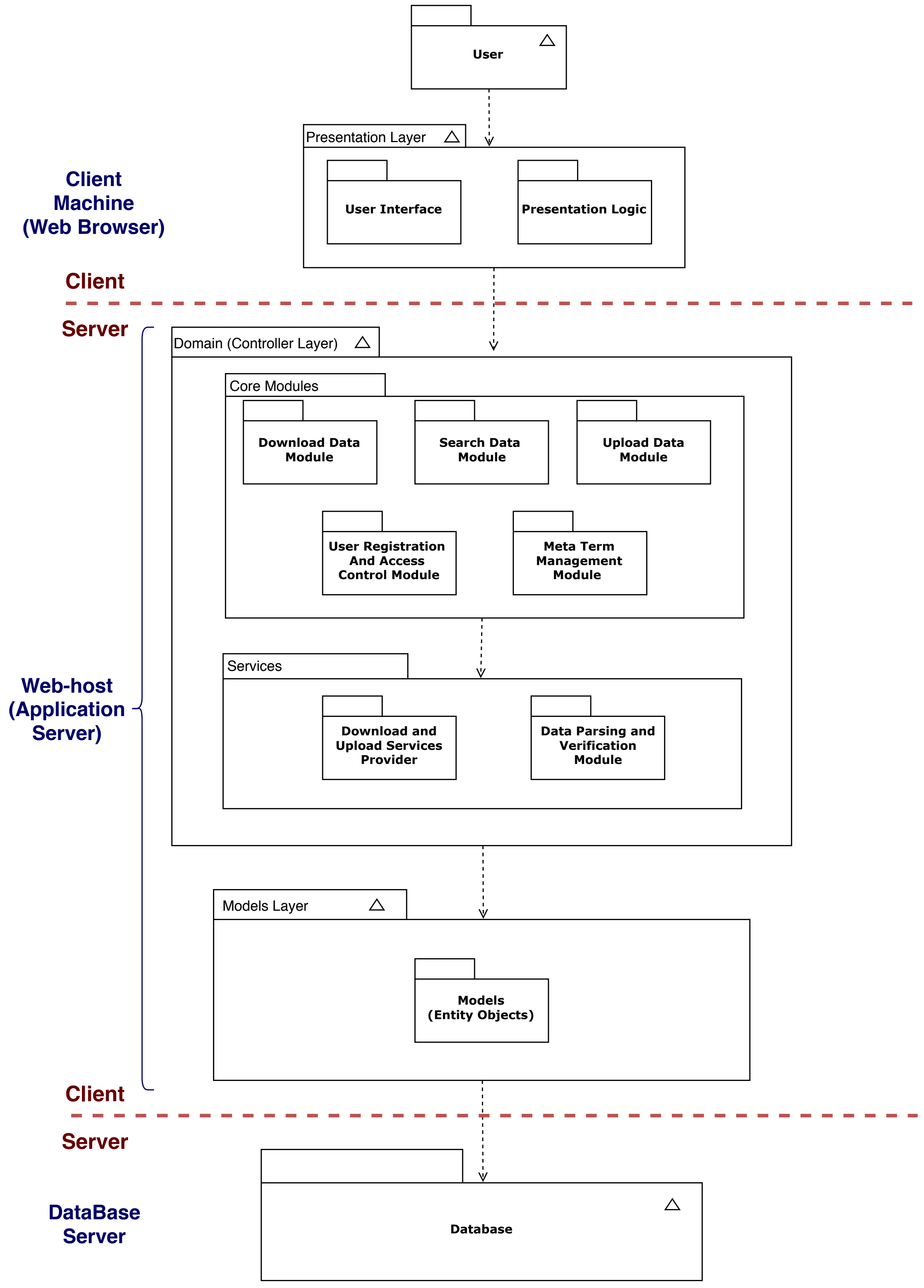

User
Presentation Layer
User Interface
Presentation Logic
Client Machine (Web Browser)
Client
Server
Domain (Controller Layer)
Core Modules
Download Data Module
Search Data Module
Upload Data Module
User Registration And Access Control Module
Meta Term Management Module
Services
Download and Upload Services Provider
Data Parsing and Verification Module
Web-host (Application Server)
Models Layer
Models (Entity Objects)
Client
Server
DataBase Server
Database

Figure 6.1: Shows a detailed view on the system's architecture, layering, and components

Browser), Web Host (Application Server), and Database Server. Each two consecutive layers will communicate through a client-server architecture through internet protocols such as HTTP. Our Main logical layers and their mapping to physical layers are:

View: UI part of the application, it sends requests to the controller, and subscribes to events in the model, it will run on the Client Machine (WebBrowser).

Controller: Handles user requests and invokes appropriate actions in models, and also handles model events and requests needed changes in View, it will run on the Web Host (Application Server).

Model: Represents the data and handles business logic, it notifies registered controllers and views of data changes, it will run on the Web Host (Application Server).

Database: The Database runs on the Database Server. It returns the information to the application server which in turn sends it to the client machine to view and edit it.

Good architecture design supports better software development process. It is costly in terms of time, effort, and money to improve software architecture at an advanced stage, so proper designing is required at the initial stages of the project. It is always a better approach to think for the long term and make the architecture flexible enough to function the system efficiently.

Chapter 7

Implementation and coding

The third phase of the SDLC process is the implementation and coding phase. After the project team obtains the customer's requirements for the project, the second phase starts, where the team designs the software. The project team then uses the design to start the implementation and coding phase.

7.1 Implementation issues

System implementation is a crucial stage of software development, where one creates an executable version of the software. But during the implementation process there are some aspects that are particularly important and which are language-independent:

Reuse Most modern software is constructed by reusing existing components or systems. While developing software, one should make as much use as possible of existing code.

Configuration management In the configuration management system, we have to keep track of the many different versions of each software component during the development process.

Host-target development Software development environment is different from Production environment because production software executes on different compute. We usually develop software in one compute (host system) and execute it on a separate computer(target system).

7.2 Version Management

To keep track of different versions of the software, we need to use a version management system. The systems include facilities to coordinate development by several programmers.

Git can be used to synchronize changes in code. During development, we used git to store changes on "master" repository. We can work with git in a lot of ways. One easy workflow is to first set up an account on GitLab, create a new repository there and then clone it to your local machine.

To set up a git repository first visit https://gitlab.com/ and create an account. To clone or download we have to select the "clone or download" button and copy the text inside the dialog box. Should be something like:

```
https://github.com/<your{\_}git{\_}user{\_}id>/Nature's_
Palette.git).
```

We can clone it on our local computer. To install git in local computer, open a command prompt/terminal and clone the repository using the URL copied above.

```
git clone https://github.com/<your{\_}git{\_}user{\_}id>/
Nature's_Palette.git
```

7.3 Development platform

Once we set up our local repository we need to set up our local development environment. In most cases, the host and target are different. We develop on one computer but we deploy on a separate machine. So we need two kinds of platform, development platform and execution platform. A platform is more than just hardware. It comes with an installed operating system. Now depending on the platform, we may need to install other supporting software like database system or interactive development environment (IDE). There can be different architecture and installed software differences between the development platform and the target platform.

7.3.1 Prerequisites

Our prerequisites for developing Nature's Palette, a web application, we need a local development machine or server running Ubuntu 18.04, along with a non-root user with sudo privileges and an active firewall. Our web application also requires Node.js and npm installed on our local machine or server, following these instructions on installing with the PPA managed by NodeSource. We have chosen Express, most popular web framework for our application framework and MongoDB installed on local machine or server, following How To Install MongoDB in Ubuntu 18.04 as our database system. For data validation and metric calculation we will be using R scripts. So our system also need R language installed.

7.3.1.1 NodeJS and NPM

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js package ecosystem, npm, is the largest ecosystem of open source libraries in the world [25]. There is a huge community that is helping built libraries, so most of the generic problems can be solved. To make development faster and more efficient npm (Node package manager) has packages we can use in our application. With the help of npm we can load lots of third party software like Mongoose, Express easily.

7.3.1.2 MongoDB

In our application, we are integrating MongoDB database with our existing Node application. When data requirements include scalability and flexibility NoSQL like MongoDB is very useful. NodeJS is designed to work asynchronously with JSON objects because of this MongoDB integrates really well with NodeJS. MongoDB can be integrated into a project by using Object Document Mapper(ODM) Mongoose, which helps to create schemas and models for application data. So we can easily organize our application code following model-view-controller pattern.

7.3.1.3 R language

Our repository will be using R script for data validation and generating the metric from submitted data files. All script is written in R which can be downloaded here:https://cran.r-project.org. As for other languages, R has a large collection of packages with specific functionality. These packages often rely on previously existing packages (dependencies).

The functions that will be most useful to us, are found in the packages ’pavo’ and ’lightR’. pavo does have dependencies so these need to be loaded as well. If pavo is installed in R, the dependencies are automatically installed as well. Another package lightR is also very useful. It’s main function is to parse proprietary spectral file formats. These functions would be used to extract values from the raw files to then calculate metrics. This package is not yet available on CRAN so can follow the instruction provided here: https://github.com/Bisaloo/lightr to install this package.

In R, first install 'pavo' and its dependencies: install.packages('pavo')

once installed packages need to be called before being used in the console.

library(pavo)

7.3.2 Project Installation steps

For any type of modification or update, nature's palette project can be downloaded from it's git repository. To run the project locally we need to install our application prerequisites first then we can follow these steps to execute the application in local machine:

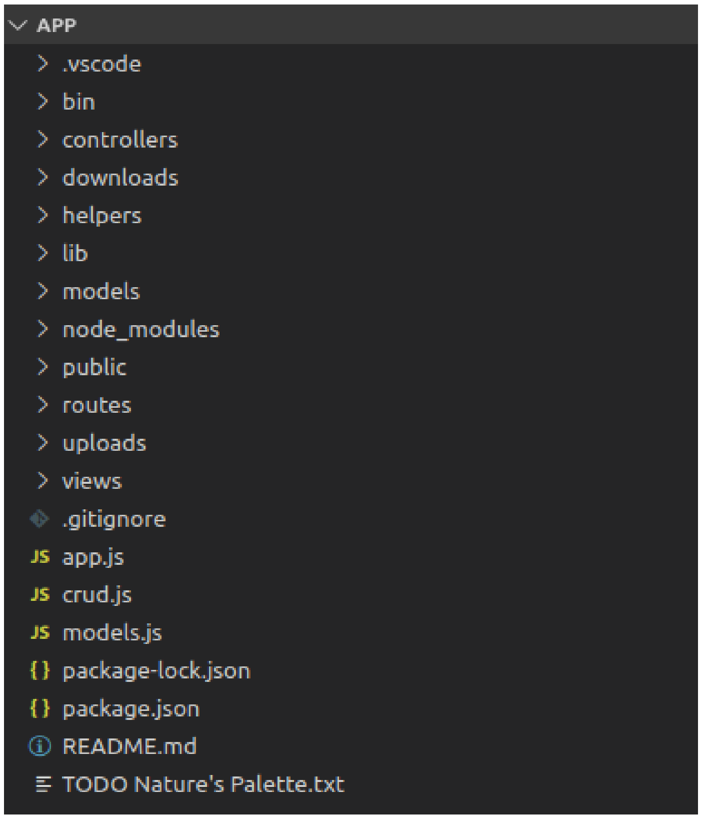
- Clone it
- Open models.js file and replace the mongoDB mongoose connection string
- Open command line terminal and change the path to project folder cd Nature's_Palette
- Use npm to install dependencies npm install
- The main file for the project is app.js. To run this file, type node app.js

The project runs at http://localhost:3000/

7.4 Project Structure

Our structure is based on the Model-View-Controller (MVC) design pattern. This pattern helps in rapid and parallel development. and also is great for separating the responsibility of the different parts of app and makes your code easier to maintain. MVC design pattern can be effectively implemented with an Express web application, which is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is the most popular Node web framework, and the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:

- Write handlers for requests with different HTTP verbs at different URL paths (routes).
- Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
- Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- Add additional request processing "middle-ware" at any point within the request handling pipeline.



```
APP
  .vscode
  bin
  controllers
  downloads
  helpers
  lib
  models
  node_modules
  public
  routes
  uploads
  views
  .gitignore
  app.js
  crud.js
  models.js
  package-lock.json
  package.json
  README.md
  TODO Nature's Palette.txt
```

Figure 7.1: Project Structure

Let's look at our application where a user can login, register. upload, search and download spectral data. Below is the files and folders structure.

controllers: Define our app route handlers and business logic.

helper: Helper lets the Code and functionality to be shared by different parts of the project. Writes utility/helper functions here which can be used by any controllers.

middlewares: Before handling the incoming request to the routes express middlewares process them first. By writing middleware, we can interpret all incoming requests to the route handler.

models: Models represent data, implements business logic and handles storage. Between controller and database models act as middleware. We can define some schema and do validation before writing anything to the database. We are using Object Data Modeling (ODM) like Mongoose that comes with great methods and features to use in the schema itself.

routes: It define our app routes, with HTTP methods. For example, we can define everything related to the upload. router.post('/upload/start', controller.uploadStart) router.put('/upload/:submit', controller.uploadSubmit) router.get('/upload', controller.authenticate)

public: Public folder consist of all static files like styling, JavaScrip, images.

views: View folder contains templates for our application front-end. These templates are rendered and served by routes.

app.js: App.js is the entry point of our application. It initializes the application and bind everything together.

package.json: This file takes care of the application dependencies, and the version of your projectthe scripts to run with the npm command. It remembers all nodejs packages that our app depends on also maintain their versions.

7.5 Work Environment and Demo

Nature's Palette system is developed with node.js and express framework using the Visual Studio integrated development environment (IDE). We are using GIT as our version control system and our repository will be hosted on Gitlab for the duration of the development.

The application can be accessed through this link:

```
https://gitlab.com/nature-s-palette/
```

The demo of the project will be hosted on Cloud Compute Canada. The project can be accessed through this link:

```
http://34.70.63.215/
```

Chapter 8

Hosting and Deploying

The application deployment and hosting process offers numerous complex challenges. Doing local development with Node Application is straightforward. Testing the application in a local machine is easier at scale without hosting it on the public staging environment. But things get complicated is when we want to put our app in production, on a web server.

8.1 Application Deployment

Application deployments define the package of software components that make up an application in a particular environment, e.g. development or production [26]. We can deploy our application in physical or virtual servers in the cloud. These cloud infrastructures allow developers to build and deploy applications by providing Virtual Machines (VMs) which are composed of hardware elements simulated with software. Simulated hardware runs on the different real underlying hardware.

8.1.1 Cloud resources

We are using Compute Canada Arbutus cloud resources as our online platform. This can be thought of as Infrastructure as a Service, which is a form of cloud computing that provides a virtual instance of a computer system in a layer abstracted from the actual hardware. We have set up our cloud resource for production purposes. The server computer provides the production environment where we can run our application website for external consumption. The environment includes:

- Computer hardware on which the website runs.
- Operating system (Linux).
- Programming language run time and framework libraries on top of which our website is written.
- Web server infrastructure that includes a web server, reverse proxy, load balancer, etc.
- Databases on which our website is dependent.

8.1.2 Setting up the Server

Once we know required software for our production environment, we need to setup the server for production ready. Firstly we created and set up our virtual machines in Arbutus cloud then we installed Linux (Ubuntu 18.04) as our operating system for our server. After that we installed all necessary packages, to host and deploy a production ready NodeJs application. Details follow:

SSH key: We used SSH Keys to Authenticate and connect to our server.

Node & NPM: To install NodeJS.

```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
```

```
sudo apt install nodejs
node --version
```

This will install the latest versions of Node & NPM. The tools in build-essential are required by some npm modules when installing.

Git: To clone our project from Git repository in our server, we installed Git.

```
sudo add-apt-repository ppa:git-core/ppa
sudo apt-get update
sudo apt-get install git
```

We have used textbf/app location in the server for our project folder, and followed following commands to clone our git repository to the project folder.

```
sudo mkdir app
cd app
sudo git clone https://gitlab.com/nature-s-palette/nature-
palette---prototype-version.git
npm install
```

Nginx: We used the Nginx webserver to handle all requests from the web. Nginx will directly handle the request for static content. SSL certificates can also be served through Nginx. All other requests, Nginx will forward to our application server.

```
sudo -s
nginx=stable
add-apt-repository ppa:nginx/$nginx
apt-get update
```

```
    apt-get install nginx
    exit
```

Our application local server listening on port 3000 which allows us to host our website online on our server local IP but by default, browsers are looking at port 80 where the server sends and receives HTML pages or data from a web client. So we need to forward all requests from the web clients to our localhost and vice versa. To do this we need to configure our Ngnix server. Here is our Nginx configuration.

```
  upstream node_server {
    server 127.0.0.1:3000 fail_timeout=0;
    }
  server {
    listen 80 default_server;
    listen [::]:80 default_server;
    index index.html index.htm app.js;
    server_name _;
    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_redirect off;
        proxy_buffering off;
        proxy_pass http://node_server;
    }
    location /public/ {
        root /app;
    }
```

```
}
```

This configuration will make available all static files from app/public/ at the /public/ path. It will forward all other requests to the instance of our app listening at the port 3000. To use the above configuration, we saved it in /etc/nginx/sites-available/app and then did the following:

```
sudo rm /etc/nginx/sites-enabled/default
sudo ln -s /etc/nginx/sites-available/app/etc/nginx/
sites-enabled/node-app
sudo /etc/init.d/nginx restart
```

The above commands remove the default configuration, then it make active our configuration and finally it restarts Nginx so the latest configuration will be loaded.

PM2: To ensure that our node application is always on, even when the application crashes or the server is restarted, we used PM2. Note that we did not use the npm start command to run our application. Instead, we have used PM2 that allows our application to run in the background. Nginx will forward the appropriate requests to our server. To install PM2 globally and to run the application with PM2, type the following command while in our project directory,

```
npm install pm2 -g
pm2 start app.js
```

To make sure that PM2 restarts when our server reboots, run the following command in server's terminal,

```
pm2 startup ubuntu
```

Firewall Setup: We have executed the following in server's terminal, to enable the firewall and to configure the firewall to allow HTTP, HTTPS and SSH access.

```
sudo ufw enable
sudo ufw allow http
sudo ufw allow https
sudo ufw allow ssh
```

8.2 Hosting our application

In order to host our application on server we need to adjust server's file permission.

8.2.1 File permission:

To set up the right folder permissions for a website on a linux server which is run by a web server like Nginx, we have followed several steps. Our website is made of static content like HTML pages, CSS, images and some dynamic content which will be generated by our webserver on the fly, for example, a JavaScript that manages file upload. So in order to display the static content to the public Nginx needs the read permission as well as the write permission to write data into the site folder as instructed by the script files. In our scenario we have a user, called rabeya, the website folder is located in /app/nautrepalette/ and the web server belongs to the www-data user group.

So the user rabeya will be the owner of our website directory and also have the full permissions like read, write, execute. Group owner will the webserver and initially will have read and execute permissions but for some folder it will have the write permission too. By doing this it will restrict other user and group to access so that no one can alter the website directory.

Set user as the owner: To get started, first we need to login into the server and run the following command,

```
chown -R rabeya /app/
```

Set the web server as the group owner: To set the www-data as the group owner of website directory which includes every file and folder inside the directory, we need to give the appropriate permission to the www-data group.

```
chgrp -R www-data /app/
```

750 permissions for everything: The third command sets the 750 permissions, where 7 is read, write and execute for the owner (i.e. rabeya), 5 is read and execute for the group owner (i.e. the web server), zero permissions for others.

```
chmod -R 750 /app/
```

Once again this is done recursively and applied on all files and folder in this directory. From the parent folder, new files and folders will inherit the group ownership.

Inherit group ownership: The last command makes new files and directories created by the web server will have the same group ownership of app/ folder, which we set to www-data with the second command. The s flags will set mode with setuid/setgid.

```
chmod g+s /var/www/html/app/
```

Give server write access: We have folders that need to be writable by the web server, we modified the permission values for the group owner so that www-data has write access. So we run this command on each writable folder. We have to be careful to apply this only where necessary and for security reasons, we can not apply this on the whole website directory.

```
chmod g+w /var/www/html/app/<writable-folder>
```

Once the file permissions is setup, our website can be accessed and browsed from internet. If all works as expected, we can visit our application by typing our public IP on the browser.

Chapter 9

Application Overview

In this chapter, we will provide an overview of our pilot web application. The application is called Nature's Palette- an open-access digital repository for spectral data.

Nature's Palette provides a prototype machine-readable and publicly accessible spectral data repository. One of the biggest hurdles preventing the submission of spectral data to archives, is the stressfull process of curating the data using standardized protocols. Therefore the main part of this project is curatorial pipeline that enables researchers to provide their data in a flexible way (e.g., single file with all metadata and thousands of raw files associated with it), which will then be standardized for consistent searches.

9.1 Using Nature's Palette

Register: Only a registered researcher with valid ORCID can submit their spectral data to the repository.

Login: For a user (Researcher) to upload their data into the portal, they must login

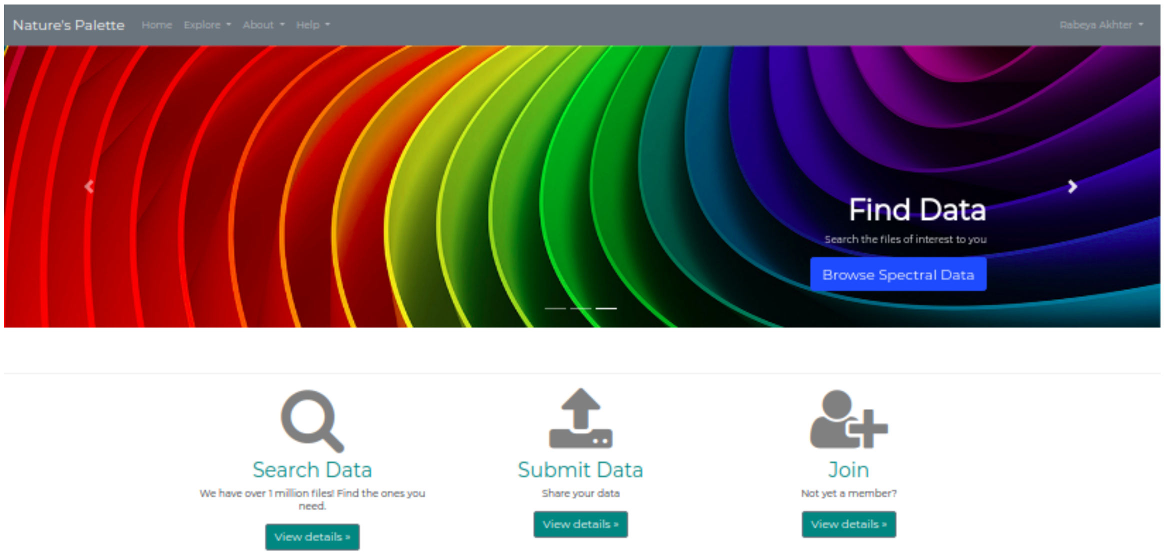


Figure 9.1: Home page

into the system using ORCID.

Upload: An authenticated researcher can upload their raw data and meta-data into the repositories, which can be downloaded by a guest user.

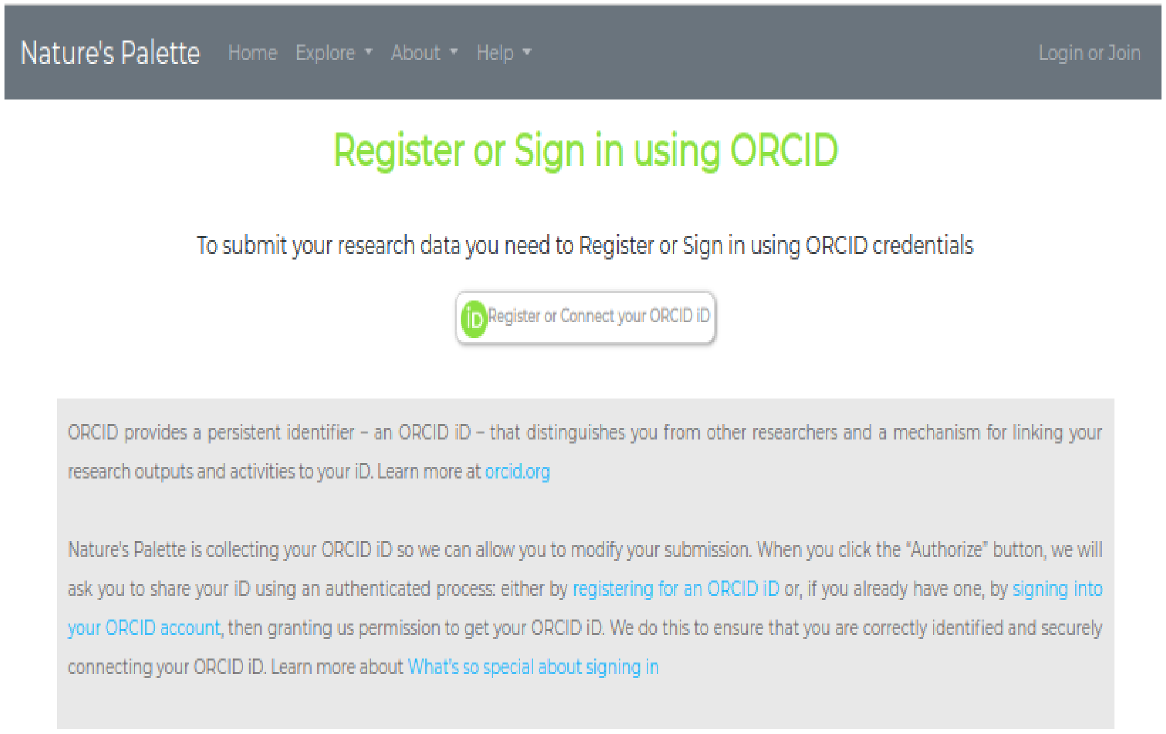
Search: The search feature enables the researcher to search for data files using search terms (Darwin core terms).

Download: Researcher will be able to download the data files based on the search result.

Add new search term: Admin have the privilege to add new search term which will assist the researcher to search for their desired data.

9.1.1 Authentication

Enabling users to register or sign into our system using their ORCID credentials can save them time and effort; they don't have to keep track of multiple usernames and passwords, and we will immediately obtain an authenticated ORCID ID.



Register or Sign in using ORCID

To submit your research data you need to Register or Sign in using ORCID credentials

ORCID provides a persistent identifier – an ORCID iD – that distinguishes you from other researchers and a mechanism for linking your research outputs and activities to your iD. Learn more at orcid.org

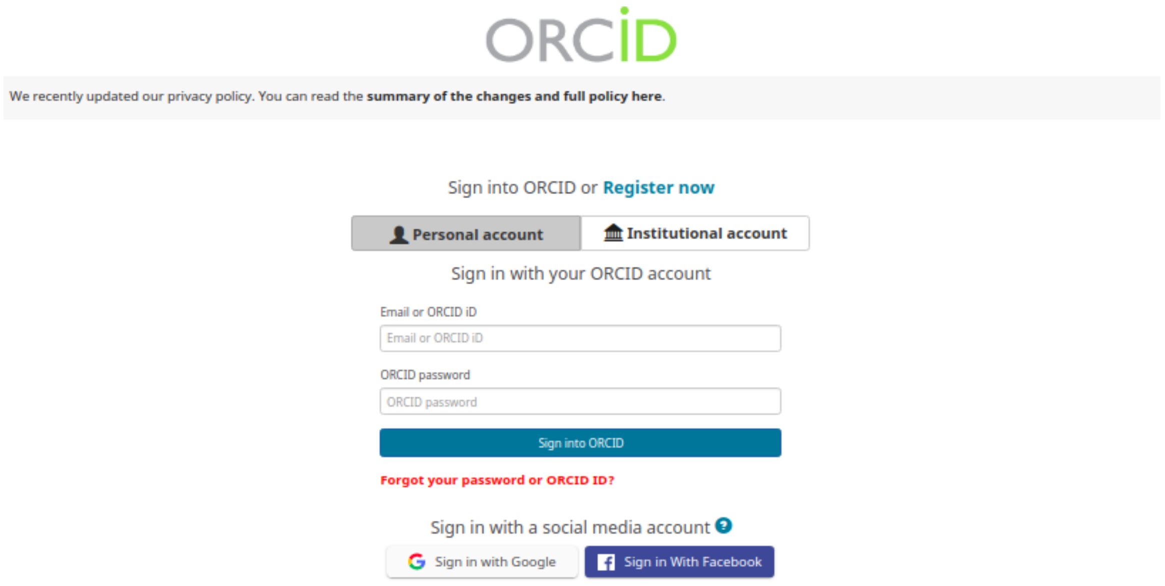
Nature's Palette is collecting your ORCID iD so we can allow you to modify your submission. When you click the "Authorize" button, we will ask you to share your iD using an authenticated process: either by registering for an ORCID iD or, if you already have one, by signing into your ORCID account, then granting us permission to get your ORCID iD. We do this to ensure that you are correctly identified and securely connecting your ORCID iD. Learn more about What's so special about signing in

Figure 9.2: Login/Registration page

Like social sign-in, for instance, sign in using Facebook or Google, as offered on many websites, ORCID sign-in is like that. The first thing that users will see is a screen inviting them to sign in to our system using ORCID. Researchers will use their ORCID username and password or linked alternate sign-in credentials to log into the ORCID website or they can sign up if they don't have an account with ORCID. The sign-in options are displayed as illustrated below. When the user successfully sign in to the system using ORCID sign-in and linked their accounts, their ID is going to be displayed within our system with a hyperlinked HTTPS URI.

Overall ORCID Login is a simple process, with the following steps.

- The user enters your application and selects the desired social network provider.
- A login request is send to the ORCID API.


We recently updated our privacy policy. You can read the **summary of the changes and full policy here**.

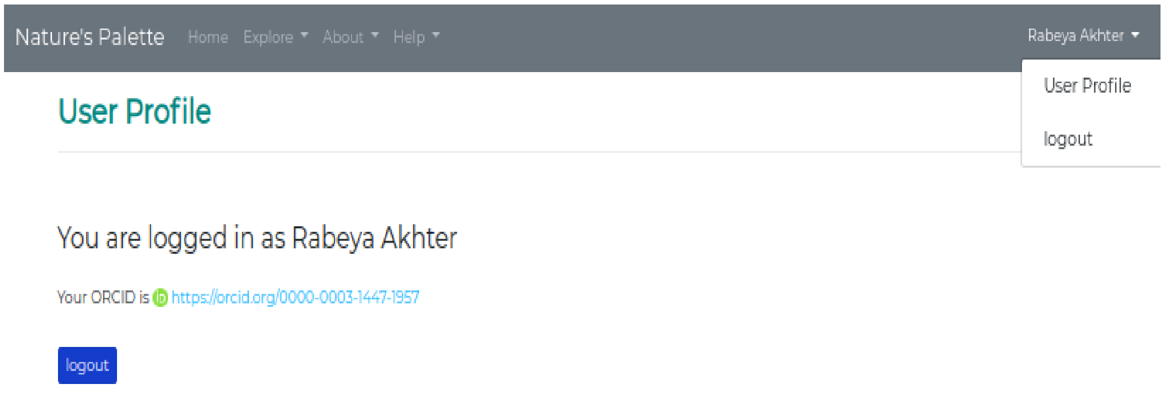
Sign into ORCID or **Register now**

Sign in with your ORCID account

Sign in with a social media account

Figure 9.3: ORCID Login

- Once ORCID API confirms the user's identity, a current user will get access to our application.
- A new user will be registered as a new user and then logged into the application.



User Profile

You are logged in as Rabeya Akhter

Your ORCID is https://orcid.org/0000-0003-1447-1957

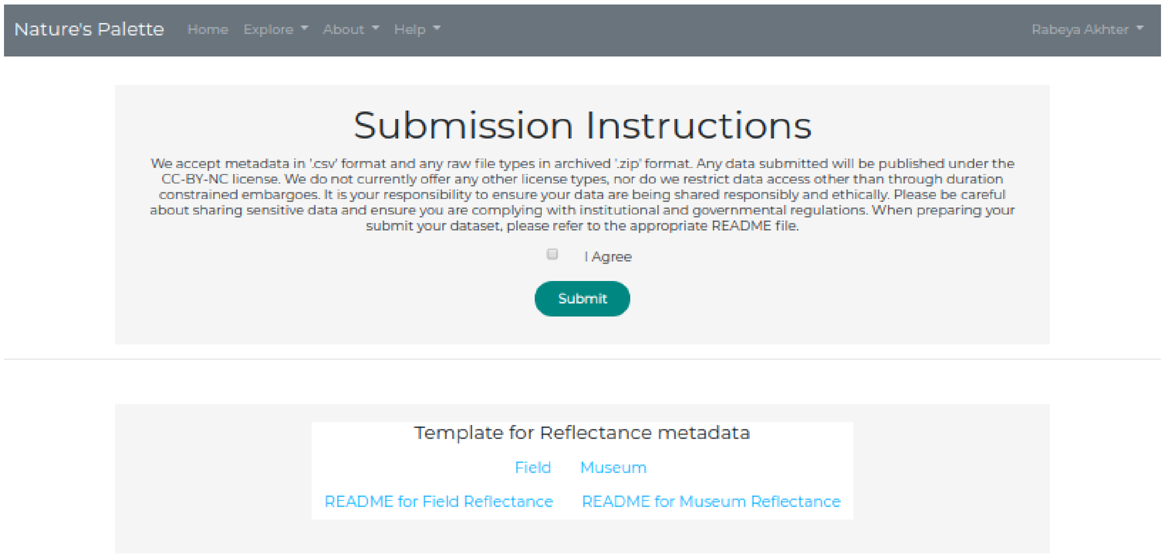
Figure 9.4: User Profile

9.1.2 Submit Data

Data submission is only available to the registered researcher. Only a registered researcher can upload spectral data into our system.

Submission of data is a simple process with the following steps: Researcher selects submit data option.that will take the them to the submission instructions page.

- Before starting the submission process, the researcher will be advised with instructions.
- Also template for metadata file will be provided to the researcher.
- Template represents prefered Darwin cores and metadata fields.
- If the researcher agrees and selects start, submission process will be begun.



Submission Instructions

We accept metadata in '.csv' format and any raw file types in archived '.zip' format. Any data submitted will be published under the CC-BY-NC license. We do not currently offer any other license types, nor do we restrict data access other than through duration constrained embargoes. It is your responsibility to ensure your data are being shared responsibly and ethically. Please be careful about sharing sensitive data and ensure you are complying with institutional and governmental regulations. When preparing your submit your dataset, please refer to the appropriate README file.

Template for Reflectance metadata

Figure 9.5: Submission page

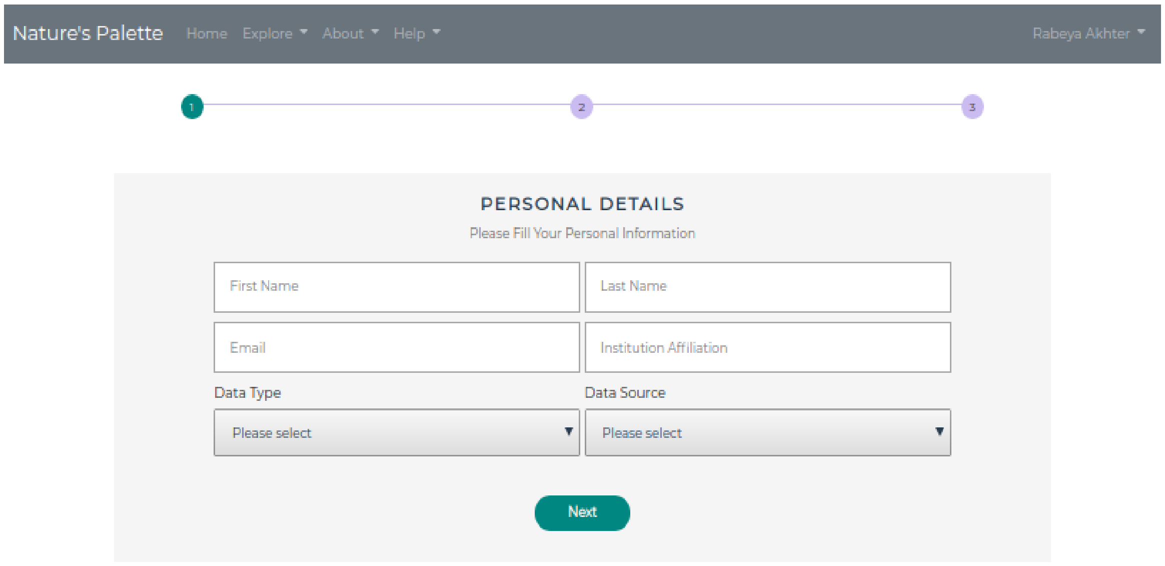
For now reflectance metadata templates for 'Field' and 'Museum' are available to the researchers.

9.1.2.1 Submission Form

Basic Information:

Researchers will provide basic information related to the data files. Basic information fields are Dublin core fields. Field details:

- Name- First and last name of the researcher. *Required by default.
- Email- The email address of the researcher. *Required by default
- Institution Affiliation- The researcher will list their affiliation, usually with a university or research institution. Not required by default.



PERSONAL DETAILS

Please Fill Your Personal Information

First Name	Last Name
Email	Institution Affiliation
Data Type	Data Source
Please select	Please select

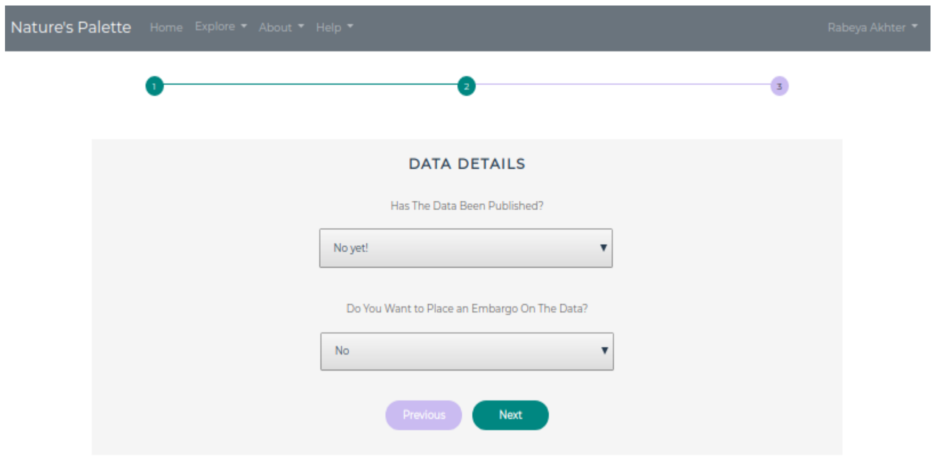
Figure 9.6: Submission Step-1

- Type of Data- Drop down (Reflectance, Transmittance, Irradiance). Only Reflectance type is accepted now. In fututre Transmittance, Irradiance will be added.
- Data from- Drop down (Field, Museum). Where the data was collected from.
- Published- Yes/ No. If yes then enter,
  - Reference- Citation of the publication

  - DOI- Digital object identifier

  Otherwise hide reference field.

- Embargo- Yes/No. The embargo is a period of time set by the researcher where access to the archived data is restricted in a digital repository until the embargo period expires.
  - If yes then
    - Date- date for publishing data (Max one year from submission).
  - Hide Date field otherwise
- Submission Date- Collect the date from system.
- Next- After filling the required fields hit 'Next' button to go to next page.



DATA DETAILS

Has The Data Been Published?

No yet!

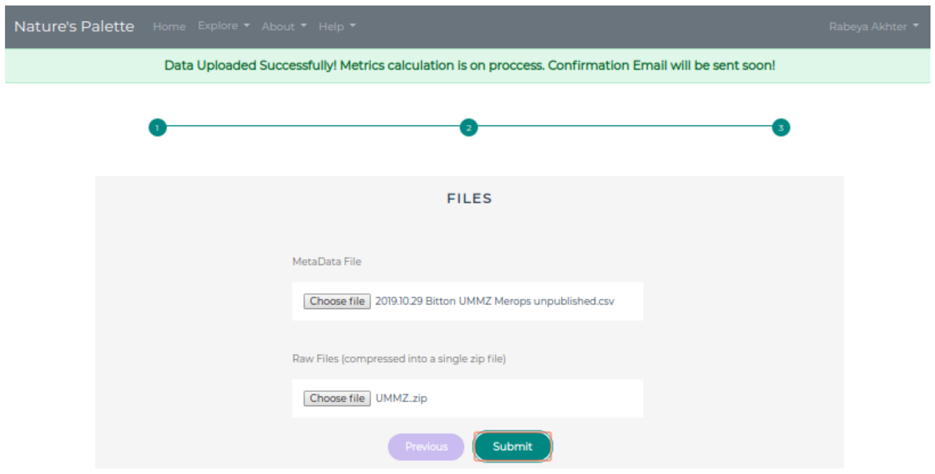
Do You Want to Place an Embargo On The Data?

No

Figure 9.7: Submission Step-2

Upload Files

- First upload the metadata file and then attach compressed raw files.
- Complete submission by clicking submit button.
- Validate form's data-
  - Metadata name error
  - File name error
  - Match mandatory metadata field



Data Uploaded Successfully! Metrics calculation is on proccess. Confirmation Email will be sent soon!

2019.10.29 Bitton UMMZ Merops unpublished.csv

UMMZ.zip

Figure 9.8: Submission Step-3

The validation process passes the 1st check, a confirmation message will be displayed. After data submission following task will be performed by server:

- Calculates metrics using R script from submitted raw files.
- Notifies researcher via email if there any error during calculation.
- Stores metrics data into database.

9.1.3 Search

Researchers can search the repository using the following search terms (Darwin core): **institutionCode, collectionCode, catalogueNumber, class, order, family, genus, specificEpithet, infraspecificEpithet, sex, lifeStage, country**, (Database specific term: **Patch**). Following a query which returns > 0 files, the system will display metadata for each matched unique measurement (should display only one row per groups of repeated measurements). In display table only below metadata fields will be shown. **genus, specificEpithet, infraspecificEpithet, sex, lifeStage, Patch**

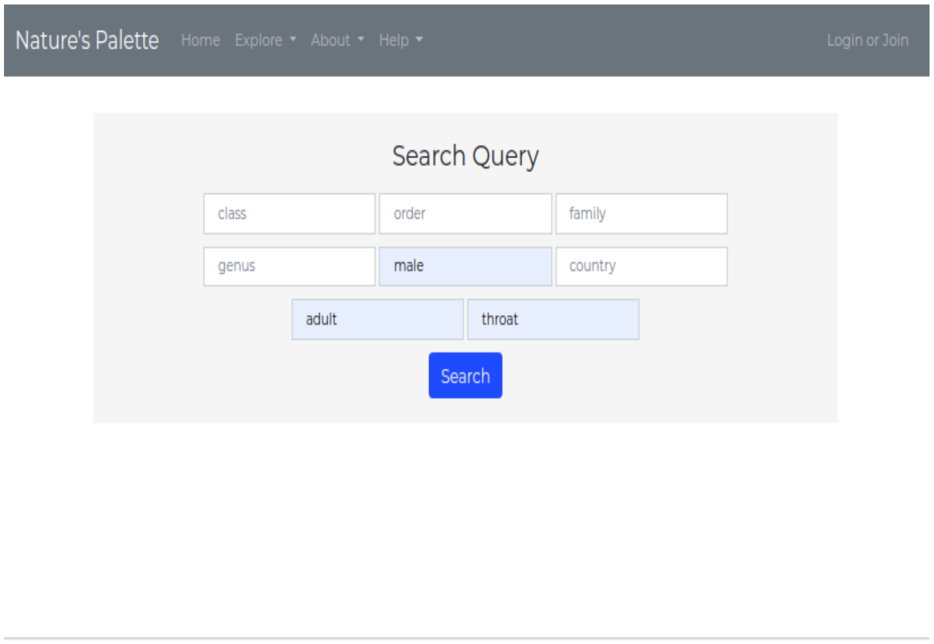
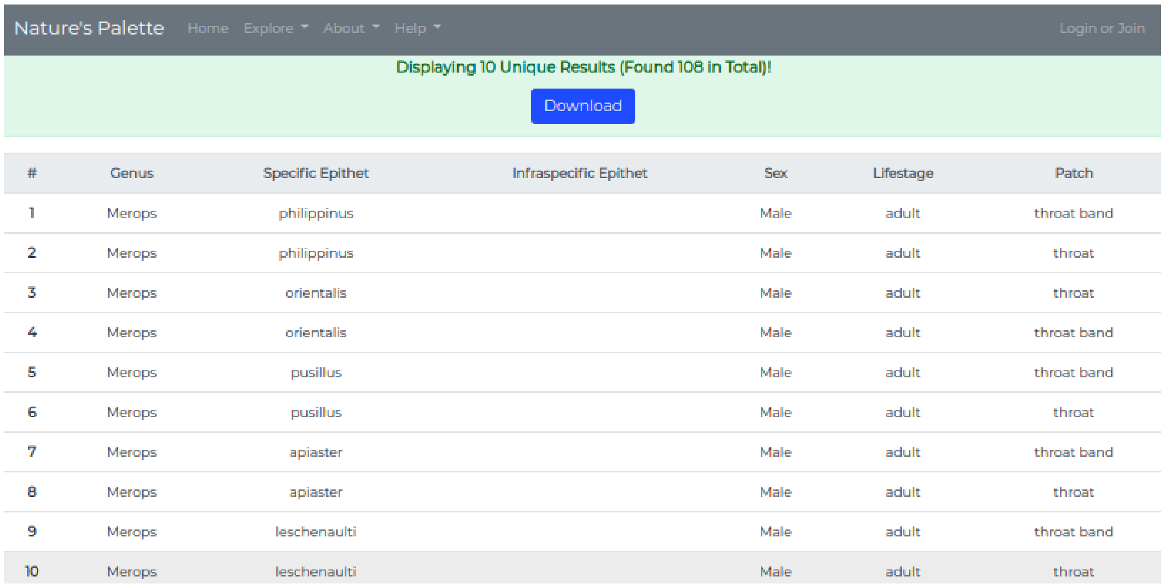


Figure 9.9: Search filters



Displaying 10 Unique Results (Found 108 in Total)!

#	Genus	Specific Epithet	Infraspecific Epithet	Sex	Lifestage	Patch
1	Merops	philippinus		Male	adult	throat band
2	Merops	philippinus		Male	adult	throat
3	Merops	orientalis		Male	adult	throat
4	Merops	orientalis		Male	adult	throat band
5	Merops	pusillus		Male	adult	throat band
6	Merops	pusillus		Male	adult	throat
7	Merops	apiaster		Male	adult	throat band
8	Merops	apiaster		Male	adult	throat
9	Merops	leschenaulti		Male	adult	throat band
10	Merops	leschenaulti		Male	adult	throat

Figure 9.10: Search result

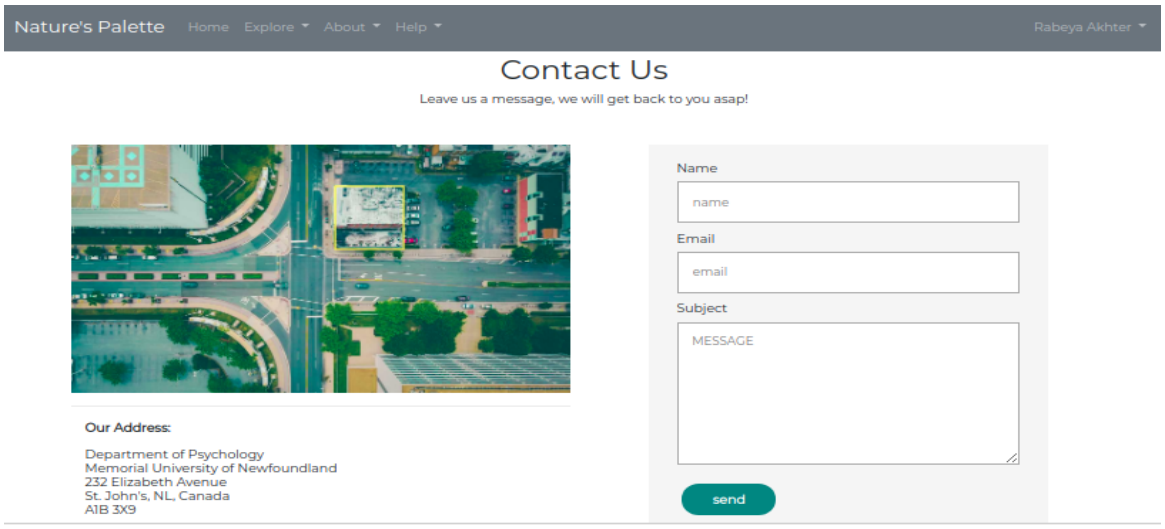
9.1.4 Download

The filtered result can be downloaded, if the researcher selects 'Download All' button. Retrieval of data has to supply metadata file along with raw files for query results only. So download will generate a package file consisting of:

- One file in tabular format containing all the metadata of the raw files relevant to query result. Also includes an additional column with 'SubmissionID'
- All raw data files identified by the query.
- A file with all the submission information from the identified 'SubmissionID'.

9.1.5 Contact Form

Contact form helps engage research community, grow mailing list, and receive feedback straight form users to improve user experience.



Contact Us

Leave us a message, we will get back to you asap!

Our Address:

Department of Psychology
Memorial University of Newfoundland
232 Elizabeth Avenue
St. John's, NL, Canada
A1B 3X9

Figure 9.11: Contact Us Page

9.1.6 About and Help

About: An About page is a special web page on a site where researchers/visitors will learn more about the repository. This section helps to give researchers more insight into who is involved with this repository and exactly what it does. The history of the repository is provided, and the histories of the people in charge are ex-

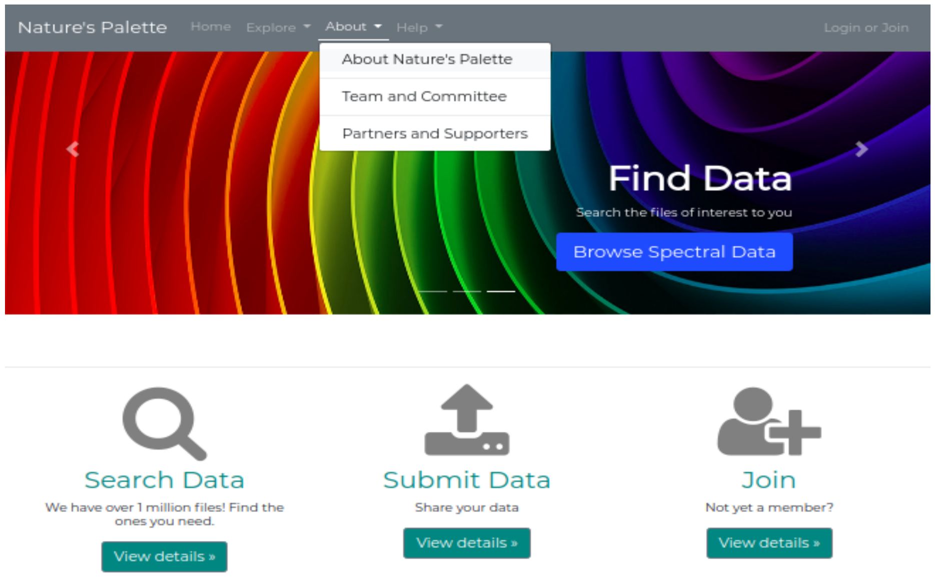


Figure 9.12: About Page

pressed through short articles, usually accompanied by photographs.

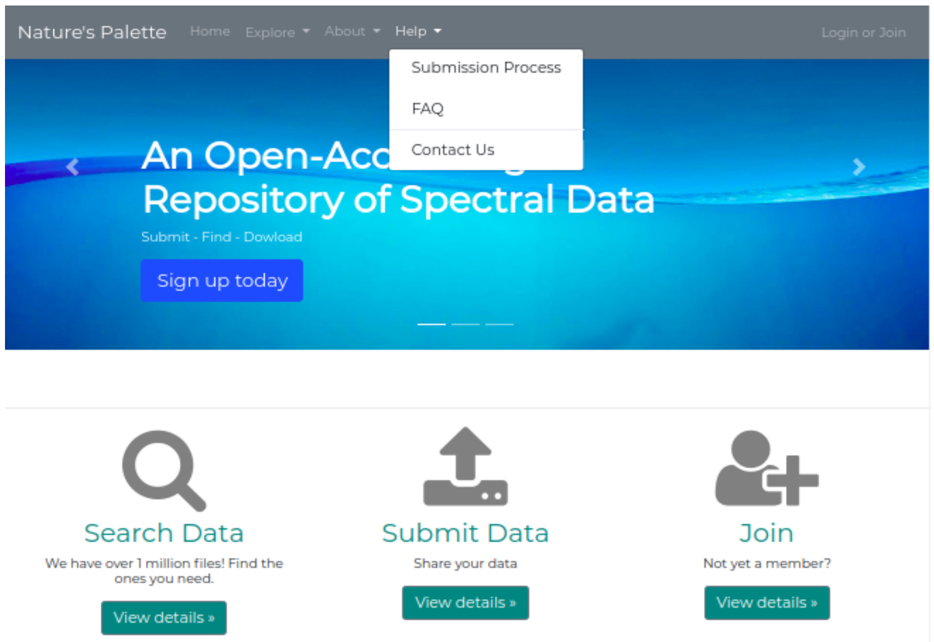


Figure 9.13: Help Section

Help: To provide assistance to users we developed 'FAQ', 'Contact Us'. Also to assist with data submission there's a submission process option under "Help". Help systems should be conveniently accessible in locations where users can possibly need answers to their questions, e.g., once they start employing a website, and once they may gain advantage from useful information. The effectiveness of a help system includes a direct relationship to the standard of the site's design. A badly designed help system could be a good its content but makes for poor quality user experience.

Nature's Palette is an open-access digital repository for spectral data with advanced search functions. It allows researchers to simply supply their spectral data without rummaging onerous processing which could be a major barrier to data sharing. it's geared towards researchers from everywhere the globe who are capturing, publishing and sharing animal ecology data. Nature's Palette's task is to amass an open access repository and make it available to the general public. In this way, it makes open access to research data for everyone.

Chapter 10

Feedback and Implications

User feedback and implications are information collected from users/customers about their reactions to a product, service, or website experience. We used feedback and insight from the researcher and website visitors to improve the user experience of our application. Below are some examples of how Nature's Palette meets the needs of researchers across the country.

10.1 Researcher/public use of Nature's Palette

10.1.1 Case 1

Jeremy is a student from Brazil who is conducting research for his MSc thesis on the morphological variation in a species of bird called the Rufous Trogon. He knows that there is much variation in plumage colouration and he has measured several specimens at his nearby museum. However, this species of trogon is found across all of Latin America and Jeremy does not have the funds to travel to other museums, or capture birds in the wild. Without Nature's Palette, Jeremy would have to contact individual researchers across the Americas and ask them individually to share their data. Because of general poor data standardization, it would take much time for Jeremy not only to obtain the data, but also to sort through it and make sure he understands what he received from individual contributors. These tasks would take a few months to complete. In contrast, by using Nature's Palette, Jeremy would simply conduct a search for his species and download all information available, which has already been curated in a standardized way. This would take a few minutes at most.

10.1.2 Case 2

Juan Pablo is an artist from Nicaragua. He has teamed up with a local frog enthusiast who wants to produce the first field guide to the frogs of Nicaragua. This work would be of great interest to conservationists, local educators, tour guides, and tourists. The problem with painting frogs, however, is that they are difficult to find and those that have been captured and kept in museum collections loose their colours. Painting the frogs using realistic colours then, is very challenging. Juan Pablo could get some of the information he needs to paint accurate portraits by searching Nature's Palette, and comparing the spectral data obtained from the frogs of interest to those for his paints. This would guarantee him an accurate representation of his subjects, even if he has never seen one before.

10.1.3 Case 3

Kevyn is a PhD student who will be conducting research on birds in Costa Rica. She wants to present hand-built models with feathers dyed to make sure they look as much as possible to her species of interest. The problem is that birds can see ultraviolet light, and their feathers reflect ultraviolet light, but Kevyn like all humans cannot see these colours. To make her models as accurately as possible, she could 1) travel to Costa Rica, capture a few birds and measure their colours, 2) find a museum collection that has these birds and measure them, or 3) find a researcher who has these measurements. Alternatively, first looking in Nature's Palette using a simple search would possibly allow her to find the data she needs, and avoid wasting precious time and/or money obtaining what she needs to start her work.

10.1.4 Case 4

John is a leading researcher on the evolution of plumage coloration in Australia; he has measured hundreds of species on the continent. He has now formed an international collaboration that will ask planet-wide questions, a game changer in his field. The main difficulty is to find a way to coordinate other researchers, potentially hundreds, to share these data. How and where these data will be stored, and who will have access and protect the valuable resource is already causing some frictions among the initial potential contributors. The obvious solution now is to encourage all those interested in the project to submit their data individually to Nature's Palette. The submissions will be secured, searchable, findable, and referenceable. Such a project, BirdColourBase, has been in the works for 3 years with little progress because of the scope, change in leadership, and group dynamics. Several of its members are eagerly awaiting the release of Nature's Palette.

10.1.5 Case 5

Luis just published a paper documenting the colour differences between male and female butterflies of several species. The journal where he submitted his findings requires that he makes all his data available before they publish his article. Traditionally, Luis would have 2 main options for these data: 1) include them in supplemental material associated with the article, or 2) submit them as part of a package to an online repository such as FigShare or Dryad. Luis is concerned that if he includes his data as supplemental material, individuals who do not have a subscription to the journal would not have access to his data, and even those who do may not find his article. He also does not have the 120 $ USD needed to submit to Dryad. With Nature's Palette, Luis can now upload all the spectral data used for his research for free, and other users will also have access to his data without cost. Luis will receive a DOI for his submission so that others can reference his dataset. This will fulfill the journal's requirement for data availability, and make the data easily accessible.

10.1.6 Case 6

Greg is an expert in red-bellied newts. He has observed them in the wild and completed several projects on their behaviour. Recently, Greg was asked to write a species account for this newt (a summary of all known information). Much of the information needed for writing this paper is found in articles, but the figure presenting the reflectance data from the newt is in a journal that does not allow reproductions. If Greg can find the raw data on Nature's Palette, he could produce his own figures without violating any international law on copyrights, and include features that he thinks will be of interest to his audience.

Chapter 11

Discussion and Conclusion

11.1 Future work

The long-term vision of the project includes providing advanced searches based on traditional functions (e.g., using georeferenced data, position in color space), and specialised modular analytical tools that can be custom combined, as well as updating the package "pavo" to search and request data from the repository, providing a seamless and repeatable workflow.

Finally, we aim to develop samples of services that can be provided by the repository specific for the analysis and extraction of spectral data. For example, the database could be queried to provide all spectral cures that could be perceived as "pink" by an animal with a specific visual system.

11.2 Conclusions

As spectral data, especially light measurements taken in different environment and at different depths in lakes and ocean can provide information about the ecology of species and ecosystems, the repository will have appeal for use by scientists in the general fields of Biology and Ecology.

Our system is designed to provide researchers with an avenue to access repository where they can upload their research data and find similar data to help with their own research. Information provided by the researchers are curated and processed so that search results are returned in a timely manner.

Easy discovery, open access and usability of this repository will create new computational research techniques and will be used in more advanced studies.

We predict that this repository will have tremendous success, could one day hold data representing the colours of all animals in the world, and provide a tremendous resource of natural light environment data.

Bibliography

[1] R. Maia, C. M. Eliason, P.-P. Bitton, S. M. Doucet, and M. D. Shawkey. pavo: an R package for the analysis, visualization and organization of spectral data. *Methods in Ecology and Evolution*, 4(10):906–913, 2013.

[2] M. D. Eaton. Human vision fails to distinguish widespread sexual dichromatism among sexually "monochromatic" birds. *Proceedings of the National Academy of Sciences*, 102(31):10942–10946, 2005.

[3] M. D. Eaton and S. M. Lanyon. The ubiquity of avian ultraviolet plumage reflectance. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1525):1721–1726, 2003.

[4] K. J. Burns and A. J. Shultz. Widespread cryptic dichromatism and ultraviolet reflectance in the largest radiation of Neotropical songbirds: Implications of accounting for avian vision in the study of plumage evolution. *The Auk*, 129(2):211–221, 2012.

[5] P. O. Dunn, J. K. Armenta, and L. A. Whittingham. Natural and sexual selection act on different axes of variation in avian plumage color. *Science advances*, 1(2):e1400155, 2015.

[6] J. K. Armenta, P. O. Dunn, and L. A. Whittingham. Quantifying avian sexual

dichromatism: a comparison of methods. *Journal of Experimental Biology*, 211(15):2423–2430, 2008.

[7] Wikipedia. Open-access repository. `https://en.wikipedia.org/wiki/Open-access_repository`, Last accessed on 2020-03-10.

[8] U. Rachel Heery and S. Anderson. Digital Repositories Review. 2005.

[9] https://duraspace.org/dspace/. Dspace. `https://duraspace.org/dspace/about/`, Last accessed on 2020-08-20.

[10] https://duraspace.org/fedora/. Fedora. `https://duraspace.org/fedora/about/`, Last accessed on 2020-08-20.

[11] Wikipedia. Data Preservation. `https://en.wikipedia.org/wiki/Data_preservation`, Last accessed on 2020-04-10.

[12] https://www.dit.ie/. FAIR Data Principles. `https://www.dit.ie/dsrh/data/fairdata/`, Last accessed on 2020-03-11.

[13] https://www.copyright.com/. What Are FAIR Data Principles? `https://www.copyright.com/blog/what-are-fair-data-principles/`, Last accessed on 2020-03-11.

[14] https://www.uml.org/. Unified Modeling Languag. `https://www.uml.org/`, Last accessed on 2020-08-22.

[15] https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/. Metadata and Its Importance in a Data Driven World. , Last accessed on 2020-08-22.

[16] https://whatis.techtarget.com/definition/Dublin-Core. Dublin Core. , Last accessed on 2020-08-22.

[17] J. Wieczorek, D. Bloom, R. Guralnick, S. Blum, M. Döring, R. Giovanni, T. Robertson, and D. Vieglais. Darwin Core: An Evolving Community-Developed Biodiversity Data Standard. *PLOS ONE*, 7, doi:10.1371/journal.pone.0029715.

[18] H. M. Schaefer. Visual communication: evolution, ecology, and functional mechanisms. In P. Kappeler, editor, *Animal Behaviour: Evolution and Mechanisms*, pages 3–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[19] D. Osorio and M. Vorobyev. A review of the evolution of animal colour vision and visual communication signals. *Vision research*, 48(20):2042–2051, 2008.

[20] S. Johnsen. How to measure color using spectrometers and calibrated photographs. *Journal of Experimental Biology*, 219(6):772–778, doi:10.1242/jeb.124008.

[21] IBM. Software development. `https://www.ibm.com/topics/software-development`, Last accessed on 2020-03-10.

[22] Technopedia. Software development. `https://www.techopedia.com/definition/16431/software-development`, Last accessed on 2020-03-10.

[23] https://www.guru99.com/software-development-life-cycle-tutorial.html. Systems development life cycle. , Last accessed on 2020-08-21.

[24] https://www.guru99.com/agile-scrum-extreme-testing.html. Agile Methodology & Model: Guide for Software Development & Testing. , Last accessed on 2020-08-21.

[25] Node.js. Node.js. `https://nodejs.org/en/`, Last accessed on 2020-04-4.

[26] Essential Project Documentation & Tutorials. Essential Project Documentation & Tutorials. `https://enterprise-architecture.org/docs/application_architecture/application_deployments/`, Last accessed on 2020-04-4.

Appendix A

Detail description of sequence diagram

A.1 Boundary, entity, and control classes

Every use case has corresponding boundary, entity, and control classes.

A.1.1 Boundary Objects:

UploadButton: Button used by a Registered User to initiate the UploadFiles use case.

UploadTermsTemplate: Template used for the input of the Upload Terms including User Details and uploaded files details. This template is presented to the User when the “UploadFiles” function is selected. The “UploadTermsTemplate” contains fields for specifying all needed attributes of the user and the uploaded files and a button (or other control) for submitting the completed template.

UploadFilesTemplate: Template used for the selection of the meta file and raw files. This template is presented to the User after the user submits the "Upload-TermsTemplate" and the system verifies the validity of the entered values. "Upload-FilesTemplate" contains fields for specifying the meta file and all associated raw files, a button (or other control) for submitting the completed template, and a field for displaying notices to the user (such as successful upload).

SearchButton: Button used by a User to initiate the SearchData use case.

SearchByTermsTemplate: Template used for the input of the SearchTerms. This template is presented to the User when the "SearchData" function is selected. The "SearchByTermsTemplate" contains fields for specifying all needed attributes of the search query and a button (or other control) for submitting the completed template.

SearchResult: Screen used for displaying the search results of the user. It is presented to the user after the "SearchByTerms" function is selected. The "SearchResult" contains fields to display Meta Data found by the search query, fields for selecting files to be downloaded before starting the "DownloadData" use case, and a field for displaying notices to the user (such as successful download).

DownloadButton: Button used by a User to initiate the DownloadData use case.

RegisterUserButton: Button used by a User to initiate the RegisterUser use case.

UserDataTemplate: Template used for the input of the UserData. This template is presented to the User when the "RegisterUser" function is selected. The "User-DataTemplate" contains fields for specifying the attributes of user, a button (or other control) for submitting the completed template, and a field for displaying notices to the user (such as successful user creation).

ManageSearchTermsButton: Button used by an Admin to initiate the Manage-SearchTerms use case.

ManageSearchTermsTemplate: Template used for the Addition, Removal or Modification of the SearchTerms. This template is presented to the Admin when the “ ManageSearchTerms ” function is selected. The “ManageSearchTermsTemplate” contains fields for displaying, modifying and removal of all current SearchTerms, control for adding new SearchTerms, a button (or other control) for submitting the completed template, anda field for displaying notices to the user (such as successful SearchTerms Modification).

A.1.2 Entity Objects

User: Any person using the user system without registering, they will have limited access to some system functions (such as not being able to upload files).

RegisteredUser: A user that has registered to the system, and have access to system functionality except Admin functions, All registered users will be identified by their unique credentials(such as email and password) created at registration.

Admin: A special Type of RegisteredUser with Admin privileges enabling access to full system functionality including administrative functions such as “Manage-SearchTerms”.

MetaTermsDictionary: Object that contains rules on required and accepted MetaData Terms and their data types to be used for verification of SearchTerms, Upload Terms and User Data.

MetaData: Holds the metadata for a single raw file in the system, it is extracted from MetaFiles uploaded to the system, where each row in the MetaFile generates a single MetaData instance.

RawFile: Holds measurements information from a single measurement process, can be attached to a single MetaData object and a single VisualModelMetrics object. It is uploaded to the system by a single user and is never modified after that. Can be downloaded by any user.

VisualModelMetrics Holds Visual Models data calculated from a single raw file, it is generated as soon as a raw file is added to the system.

SearchQuery: Holds the query for a single search process by a single user. Composed of one to multiple SearchTerms, at most a single Geographical Region and at most a single Color Region. It is responsible for aggregating the sub queries generated by each of it's components and executing the search operation, holding the result, and refining it based on changes to its components. It can be attached to one or more SearchResult Object and notifies all of them on each change to the search result.

SearchTerm: Holds key, value parameters for a single Darwin or Dublin Core related to a single search query.

A.1.3 Control Objects:

UploadDataControl: Manages the "UploadData" function. This object is created when the "RegisteredUser" selects the "UploadButton" button. It then creates an "UploadTermsTemplate" object and presents it to user. After submitting the template, this object then collects information from "UploadTermsTemplate" template, validates submitted terms using "MetaTermsDictionary", and creates an "Upload-FilesTemplate" object and presents it to user. After submitting the template, this object then collects information from “UploadFilesTemplate” template, requests upload of submitted “MetaFile” from the “FileDownloaderAndUploader“ then waits for an acknowledgment of successful upload. When the acknowledgment is received, it requests verification of the selected “MetaFile” and “RawFiles” from the “DataParserAndVerifier” and if the verification is successful, it requests upload of the submitted “RawFiles” from the “FileDownloaderAndUploader“ then waits for an acknowledgment of successful upload. When the acknowledgment is received, it requests the parsing of all data contained in the files from the “DataParserAndVerifier” then waits for an acknowledgment of success. When the acknowledgment is received, it notifies user of a successful upload operation.

SearchDataControl: Manages the “SearchData” function. This object is created when the “User” selects the “SearchButton” button. It then creates an “SearchByTermsTemplate” object and presents it to the user. After submitting the template, this object then collects the information from the “SearchByTermsTemplate” template, validates the submitted terms using the “MetaTermsDictionary”, and creates an ”SearchQuery” object. It then creates a “SearchResult” object and passes it a reference of the “SearchQuery” to facilitate its subscription to the “SearchQuery” updates. It then passes the collected information from the “SearchByTermsTemplate” template to the “SearchQuery”, and requests the execution of search operation.

DownloadDataControl: Manages the “DownloadData” function. This object is created when the “User” selects the “DownloadButton” button. It then requests the generation of “MetaFile” of the selected “MetaData” from the “DataParserAndVerifier“ then waits for an acknowledgment of success and reception of the “MetaFile” object. When the acknowledgment is received, it requests the packaging of “MetaFile” and the selected selected “RawFiles” into a single packaged file from the “FileDownloaderAndUploader“ then waits for an acknowledgment of success and reception of the packaged file.When the acknowledgment is received, it requests the start of the download process from the “FileDownloaderAndUploader“.

RegisterUserControl: Manages the “RegisterUser” function. This object is created when the “User” selects the “RegisterUserButton” button. It then creates a “UserDataTemplate” object and presents it to the user. After submitting the template, this object then collects the information from the “UserDataTemplate” template, validates the submitted terms using the “MetaTermsDictionary”. It then creates a ”RegisteredUser” object, sets its values from the information collected from the “UserDataTemplate” template, and requests to save the newly created “RegisteredUser”. When the operation is successful, it requests from the “UserDataTemplate” to notify the user of successful user registration operation.

ManageSearchTermsControl: Manages the “ManageSearchTerms” function. This object is created when the “Admin” selects the “ManageSearchTermsButton” button. It then creates a “ManageSearchTermsTemplate ” object and presents it to the user. After submitting the template, this object then collects the information from the “ManageSearchTermsTemplate” template, validates the submitted terms using the “MetaTermsDictionary”. It then requests to save the “SearchTerms” from the ”MetaTermsDictionary” object. When the operation is successful, it requests from the “ManageSearchTermsTemplate” to notify the user of successful user terms modification operation.