

# Nature's Palette Validator

Shawon Ibn Kamal

Department of Computer Science

Memorial University of Newfoundland

Supervised by Dr. Adrian Fiech and Dr. Pierre-Paul Bitton

A dissertation submitted to the Department of Computer Science  
in partial fulfillment of the requirements for the degree of  
Bachelor of Science (Honours) in Computer Science

April 2020

# Abstract

Nature's Palette is an open-access repository that allows researchers to share, search and download spectral data. This prevents substantial resources from being wasted on reproducing the same data for different research projects in animal ecology and evolution. However, researchers that collected data in the past followed different standards that best fit their needs and sharing them in the repository becomes a tedious process due to the strict validation process. Hence, I developed an offline cross-platform tool that will help researcher validate their dataset before their final submission to the repository.

# Acknowledgment

Foremost, I would like to thank my supervisors Dr. Pierre-Paul Bitton and Dr. Adrian Fiech for their consistent support and guidance. Their expertise and enthusiasm to assist in any way have helped me throughout the Honours Project. Special thanks to my colleagues at Blue Communications, Jon Drover, and Mike Rose for supporting and being easy on me in such a difficult period. Finally, I'm deeply indebted to my friends and family for always being there for me.

# Table of Contents

**Abstract**

**Acknowledgments**

**Table of Contents**

**List of Tables**

**List of Figures**

**Chapter 1: Introduction**

1.1 Background

1.2 Problem Description

1.3 Goals

**Chapter 2: Application Development**

2.1 Software Engineering Life Cycle

2.2 Requirement Analysis

2.3.1 Functional Requirements

2.3.2 Non-Functional Requirements

2.3.3 Use Case Descriptions

**Chapter 3: Implementation and Coding**

3.1 Framework Description: Electron

3.2 Development Platform

3.3 Project Structure

3.4 Processes in Electron

3.5 Time Complexity and Algorithms

## **Chapter 4: Results and Discussion**

### 4.1 Application Overview

### 4.2 Test Cases

## **Chapter 5: Conclusion**

## **Bibliography**

# List of Figures

Figure 1 Project Structure	20
Figure 2 GUI: Nature's Palette Validator	25
Figure 3 GUI: Feedback	26
Figure 4 GUI: Fix Issues within the Application	27

# Chapter 1

## Introduction

### 1.1 Background

The interactions between light environment, coloration, and color vision have always been popular research subjects in animal ecology and evolution. With portable spectrometers becoming more accessible in the last couple of decades, the objective quantification of the spectral properties of light and the study of animal coloration has highly progressed. Study on such a topic often requires thousands of spectral measurements from multiple light sources, the transmission properties of the medium, and animal integuments. However, the publicly available data on these measurements for researchers are very limited. Most of the available data are either not properly curated or do not include metadata, which prevents them from being searched and used in further research. Hence, additional time and funding have been wasted by researchers in recreating these data for their projects.

This lack of accessible data inspired the development of Nature's Palette which is an open access repository for spectral data. The purpose of this project is to allow researchers to easily query curated spectral data that is free to download and reuse. It also supports researchers to submit their data.

## 1.2 Problem Description

For the data in the repository to be found, used, preserved, and reused in perpetuity, we need to ensure they all have standardized metadata. Nature's Palette uses Dublin Core and Darwin Core standards as they best fit the type of content the repository holds. These terms describe the nature of the data (e.g., what animal, what body part was measured, how it was measured), and where the data come from (i.e., who, when where, etc....) However, researchers that have collected data in the past followed different standards that best fit their needs. So, there have been issues in validating the submission of spectral data to the repository due to many submissions not meeting the standards. Although, the online repository provides feedback on what changes are required to ensure it passes the validation step, uploading large datasets with limited internet capability and waiting for feedback could become a tedious process. As a result, I will overcome this problem by developing an application that facilitates data curation.

## 1.3 Goals

The goal of this project is to build a separate standalone offline application to pre-validate researchers' datasets and metadata files before the researcher uploads them to the online repository. It will use the existing validation rules and provide necessary user-friendly guidance on how to resolve any issues regarding the validation. The objectives of this project are:



1. Exploring different causes of failed validation such as typo, file-metadata mismatches, unstandardized format, etc.; find solutions for them and implement them.
2. Developing the application based on a proper framework that users can easily run locally without too many dependencies or downloads. The application must be cross-compatible across Windows, macOS, Linux, and ChromeOS.
3. Developing using JavaScript to keep it consistent with the existing web application.
4. Improving matching algorithms to show more possible matches in the feedback and ensure optimal time and space complexity.
5. Improving the output structure of feedback by listing all the potential solutions.
6. Implementing methods to allow one-click automatic changes to data or metadata files from the potential solutions.

# Chapter 2

## Application Development

### 2.1 Software Engineering Life Cycle

Software Development Life Cycle (SDLC) is a process used by the software development team to design, develop and test software. SDLC aims to produce high-quality software that meets client expectations and reaches completion within the expected time and cost. I have used the Agile model for our SDLC. The Agile methodology is a way to manage a project by breaking it up into several iterations. Each phase iteration goes through a cycle of requirement analysis, designing, implementation, testing, and maintenance. It involves constant collaboration with clients and continuous improvement at every stage.

### 2.2 Requirement Analysis

We must determine user expectations for the product before jumping into implementation with code. This involves frequent communication with the team and the users to determine specific feature expectations, resolve conflict and ambiguity in requirements, and document all aspects of the project development process from start to finish. This helps developers deliver a system that meets users' quality expectations.

### 2.3.1 Functional Requirements

Functional requirements are application functionalities that the developers must implement to enable users to accomplish tasks. The functional requirements for this project are:

- Allow users to validate metadata files and raw data files.
- Show feedback on the validation process and allow users to fix issues within the application.
- Export archived validated dataset that will pass the validation step in the online repository.

### 2.3.2 Non-Functional Requirements

Non-functional requirements define how the system should perform. Unlike functional requirements, these requirements are optional and are not related to the system requirements. The non-functional requirements for this project are:

- **Compatibility:** The application must be cross-compatible across Windows, macOS, Linux, and ChromeOS.
- **Usability:** The interface must be intuitive.
- **Readability:** The feedback from the validation steps must be easily understandable to researchers.
- **Data Integrity:** Completeness, accuracy, and consistency of the data.

- Performance: The application must be able to complete validation steps at optimal speed.
- Internet Access: The application must be usable with no internet connection.

## 2.3.3 Use Case Descriptions

### 2.3.3.1 Validate Metadata file and Dataset

Name:	Validate Metadata file and Dataset
Participating actor:	Researcher
Entry condition:	The application must be running.
Flow of events:	<ol style="list-style-type: none"> <li>1. The researcher selects to choose a metadata file.</li> <li>2. The system opens a file dialog prompting the researcher to select a single CSV file.</li> <li>3. The researcher selects a single metadata CSV file.</li> <li>4. The system gains access to the file for validation.</li> <li>5. The researcher selects to choose the raw data files.</li> <li>6. The system opens a file dialog allowing the researcher to select a folder containing the raw data files.</li> <li>7. The researcher selects a folder containing the raw data files.</li> <li>8. The system gains access to the folder containing the raw data files.</li> <li>9. The researcher selects to validate.</li> </ol>

	<p>10. The system does all the validation checks.</p> <p>(See validation rule UVR-1, UVR-2, and UVR-3.)</p> <p>11. The system finds that all the validation checks have passed and displays the confirmation message.</p>
Exit condition:	<ul style="list-style-type: none"> <li>• Feedback is displayed to the researcher.</li> </ul>
Alternative flows:	<ul style="list-style-type: none"> <li>• 11a. System finds errors during validation. <ul style="list-style-type: none"> <li>◦ The system lists down the checks that have failed and allows researchers to fix the issues.</li> </ul> </li> </ul> <p>(See Use Case 2.)</p>

### 2.3.3.2 View Feedback and Modify Data

Name:	View Feedback and Modify Data
Participating actor:	Researcher
Entry condition:	The validation has failed.
Flow of events:	<ol style="list-style-type: none"> <li>1. The system prompts whether to modify existing files or make a backup of files and modify them.</li> <li>2. The researcher selects to modify existing files.</li> <li>3. The system identifies and reports the list of required metadata headers and fields in the CSV that are missing.</li> <li>4. The researcher manually adds the missing metadata headers and fields in the CSV using a spreadsheet editor.</li> </ol> <p>The researcher returns to the application and reinitiates validation.</p>

	<ol style="list-style-type: none"><li>5. The system identifies and reports the list of duplicate filenames in the metadata file.</li><li>6. The system offers to fix issues within the application.</li><li>7. The researcher chooses to fix the issues.</li><li>8. The system prompts the researcher to make changes to the "FileName" column of the metadata file by suggesting possible correct filenames.</li><li>9. The researcher accepts or rejects the suggestions.</li><li>10. The system makes changes to the metadata file and reinitiates validation.</li><li>11. The system identifies and reports the filenames in the metadata file not found in the raw data file folder.</li><li>12. The system offers to fix the issues.</li><li>13. The researcher chooses to fix the issues.</li><li>14. The system prompts the researcher to make changes to the "FileName" field in metadata by suggesting possible correct filenames.</li><li>15. The researcher accepts or rejects the suggestions.</li><li>16. The system makes changes to the metadata file and reinitiates validation.</li><li>17. The system identifies and reports the raw data files without metadata.</li><li>18. The system offers to fix the issues.</li></ol>
--	---

	<p>19. The researcher chooses to fix the issues.</p> <p>20. The system prompts the researcher to rename raw data files by suggesting possible correct filenames.</p> <p>21. The researcher accepts or rejects the suggestions.</p> <p>22. The system makes changes to the raw data files and reinitiates validation.</p>
Exit condition:	<ul style="list-style-type: none"> <li>• The researcher went through all the validation steps.</li> </ul>
Alternative flows:	<ul style="list-style-type: none"> <li>• 2a. The researcher selects to make a backup of files and modify them. <ul style="list-style-type: none"> <li>○ The system prompts the researcher to select a directory.</li> <li>○ The researcher selects a directory.</li> <li>○ The system copies files over to the directory to make all future changes to the newly created backup files.</li> </ul> </li> <li>• 3a. All required metadata headers and fields are filled. <ul style="list-style-type: none"> <li>○ The system informs the researcher that the validation check passed for headers.</li> </ul> </li> <li>• 5a. There are no duplicate filenames in the metadata file. <ul style="list-style-type: none"> <li>○ The system informs the researcher that the validation check passed for this step.</li> </ul> </li> <li>• 7a. The researcher chooses to export the issues in CSV</li> </ul>

	<p>and fix issues outside of the application.</p> <ul style="list-style-type: none"> <li>○ The system prompts the researcher to select a directory using a directory dialog.</li> <li>○ The researcher selects a directory.</li> <li>○ The issues are exported to the directory in CSV format.</li> </ul> <ul style="list-style-type: none"> <li>● 11a. All metadata rows contain valid filenames. <ul style="list-style-type: none"> <li>○ The system informs the researcher that the validation check passed for this step.</li> </ul> </li> <li>● 13a. The researcher chooses to export the issues in CSV and fix issues outside of the application. <ul style="list-style-type: none"> <li>○ The system prompts the researcher to select a directory using a directory dialog.</li> <li>○ The researcher selects a directory.</li> <li>○ The issues are exported to the directory in CSV format.</li> </ul> </li> <li>● 17a. All raw data files contain metadata. <ul style="list-style-type: none"> <li>○ The system informs the researcher that the validation check passed for this step.</li> </ul> </li> <li>● 19a. The researcher chooses to export the issues in CSV and fix issues outside of the application. <ul style="list-style-type: none"> <li>○ The system prompts the researcher to select a</li> </ul> </li> </ul>
--	--



	<p>directory using a directory dialog.</p> <ul style="list-style-type: none"> <li>○ The researcher selects a directory.</li> <li>○ The issues are exported to the directory in CSV format.</li> </ul>
--	---

### 2.3.3.3 Save Validated Metadata File and Raw Data Files

Name:	Save Validated Metadata File and Raw Data Files
Participating actor:	Researcher
Entry condition:	The researcher went through all the validation steps.
Flow of events:	<ol style="list-style-type: none"> <li>1. The research selects to export validated metadata file and raw data files.</li> <li>2. The system opens a file dialog allowing the researcher to choose a folder for output.</li> <li>3. The research responds to the prompt.</li> <li>4. The system truncates data to only include the validated metadata and raw data files, archives the files, and exports them to the selected folder.</li> </ol>
Exit condition:	<ul style="list-style-type: none"> <li>● Files are saved to the researcher's computer.</li> </ul>

# Chapter 3

## Implementation and Coding

### 3.1 Framework Description: Electron

After thoroughly reviewing the software requirements and given that the existing repository is built using Node.js, we decided to build the application using Electron. Electron is a free and open-source software development framework maintained by GitHub for building native applications that run on Windows, macOS, and Linux. The framework comes with a package of Chromium and NodeJS, which allows development using familiar web technologies (JavaScript, HTML, and CSS) and requires no additional installation requirements from the user. It is one of the most popular frameworks for building cross-platform native applications and is used on applications such as Discord, Visual Studio Code, and Slack. Hence, we decided to use Electron to build this application.

### 3.2 Development platform

#### 3.2.1 Prerequisites

To set up the development environment of the application, there are certain prerequisites to be fulfilled on the system. The development must be carried out in a compatible operating system – Windows, macOS, or Linux. The system must have the Node.js package manager also known as NPM. It is the largest package

manager register and offers numerous tools and libraries available readily for use. To distribute a production build of the application, the system must also have Yarn which is also a package manager that behaves like NPM.

### 3.2.2 Running in a development environment

Once all the prerequisites of the development platform are met, it is quite straightforward to get the application running. The project folder can be opened on any IDE or text editor of the developer's choice. In the terminal execute "npm install" to let NPM fetch and install all the dependencies of the application. Once the installation is finished, the development environment can be started by executing "npm start" on the terminal.

### 3.2.3 Distribution

One of the great things about Electron is how easy it is to distribute the application for production. The distribution is handled by electron-builder on Windows, macOS, and Linux easily along with auto-update support. The application can be packaged in a distributable format such as dmg, windows installer, or deb package by executing "yarn app:dist" on the project folder. The package directory can be generated without creating an installer by executing "yarn app:dir" which builds faster and is helpful for testing purposes.

## 3.3 Project Structure

The project structure is kept consistent with the original online repository by following an MVC (Model – View – Controller) design pattern. This is a widely used design pattern that ensures modularity of different kinds of components and allows parallel development efficiently. The file and folder structure are described below:

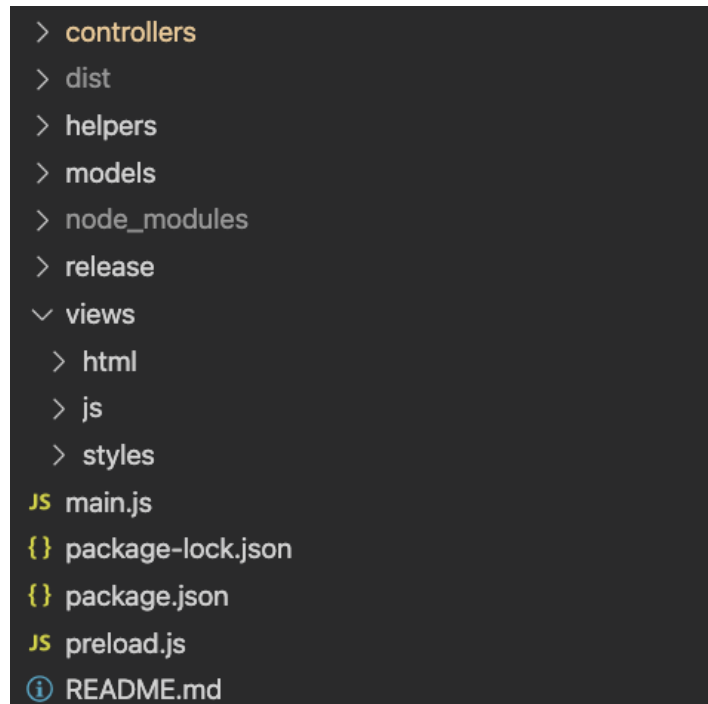


Figure 1 Project Structure

**controllers:** Controllers contain the flow control logic of the application. It determines what response to send back to a user when a user makes a request. This keeps functionalities separate from the main.js file and makes the code more manageable.

**dist:** Dist contains the distribution files of the target system after executing “yarn app:dist” or “yarn app:dir.”

**helpers:** Helpers are used to providing some functionality which isn't the main goal of the application. These can be used by any controllers and help reduce duplicate code.

**models:** Models stores data and its related logic. It represents data that is being transferred between controllers.

**node\_modules:** This folder contains all the dependencies of the application that are downloaded after executing "npm install."

**views:** Views folder contains all the necessary static files that are rendered on the front page of the application. This folder is responsible for the user interface and experience.

**main.js:** main.js is the entry point of the application. It is responsible for connecting all the components of controllers and views.

**package.json:** This is the configuration file of the application which stores the name and description of the application along with the list of dependencies and list of commands that are executable by the terminal.

**preload.js:** This file is responsible for bridging the code from main.js to expose data in the renderer thread (views).

## 3.4 Processes in Electron

### 3.4.1 Process Model

Electron uses a multi-process model that behaves like the Chromium architecture. Each process is responsible for different tasks to prevent overhead and crashes.

#### The Main Process

This is the entry point of the application which is written in `main.js`. Each application has only one main process which acts similar to the server-side of a web application. `main.js` runs on a Node.js environment and can require modules and use Node.js APIs. This handles window management, and application lifecycle and has access to native APIs that control native desktop functionalities.

### **The Renderer Process**

The renderer process starts when a new window is created. It is responsible for rendering web content in the window. The `views` directory and `preload.js` are the main files that handle the renderer process. The renderer process has no access to Node.js APIs and essentially acts similar to the client-side of a web application.

#### **3.4.2 Context Isolation**

In the renderer process, it is important that the website loaded in “webContents” has no access to Electron's internal API or Node.js environment due to security requirements. This is done using context isolation where the website cannot access any variable from the preload script. The preload script defines the APIs of the renderer process and using “contextBridge,” it safely exposes the APIs to the website.

#### **3.4.3 Inter-Process Communication**

Inter-Process Communication (IPC) allows the main and renderer process to communicate and perform most tasks. The IPC channels are “ipcMain” and “ipcRenderer” and they can communicate from both directions.

An example of inter-process communication is as follows:

#### indexRenderer.js

```
// Handler for selecting metadata
document.getElementById('selectMetaData').addEventListener('click', async (event) => {
  event.preventDefault();
  const meta = await window.api.selectMeta();
  replaceText('metadataOutput', "Meta selected: "+meta);
});|
```

This is a static JavaScript in the renderer process that handles the button click for selecting a metadata file. It triggers “window.api.selectMeta()” which is a API exposed from preload.js using “contextBridge.”

#### preload.js

```
const { ipcRenderer, contextBridge } = require('electron')

const API = {
  | selectMeta: () => ipcRenderer.invoke("selectMeta"),
  | }

contextBridge.exposeInMainWorld("api", API);
```

The “selectMeta()” API triggered from indexRenderer.js then triggers “ipcRenderer.invoke(‘selectMeta’)”. The ipcRenderer sends the message to the main process and waits for a response.

#### main.js

```
ipcMain.handle("selectMeta", validationController.selectMeta);|
```

The ipcMain in main.js handles the message, lets the controller open a file dialog, and returns the source as a response. The response is received by ipcRenderer in the renderer process and can be displayed as web content.



# Chapter 4

## Results and Discussion

### 4.1 Application Overview

To use the Nature's Palette Validator tool, a researcher requires three things:

1. A folder consisting of raw data files of different measurements.
2. A metadata CSV that corresponds to the raw data and is built using the template provided on the Nature's Palette website.
3. A compatible system – Windows, macOS, or Linux.

To run the application, a researcher can simply open the executable and the application will start running.

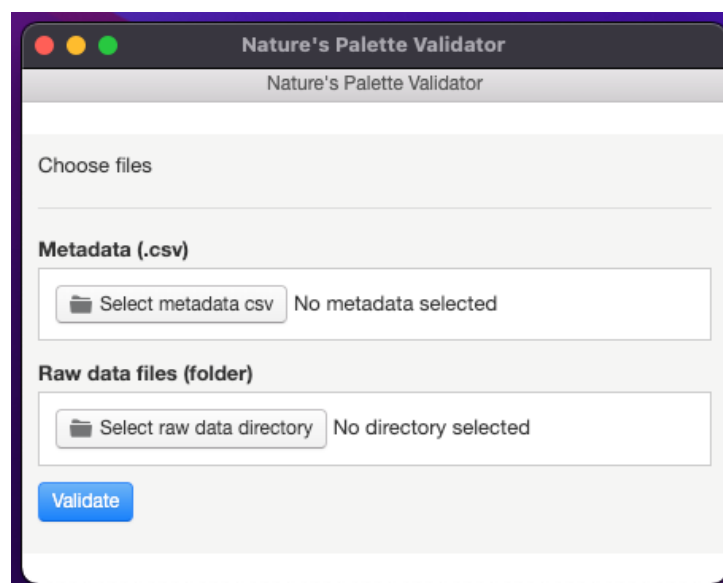


Figure 2 GUI: Nature's Palette Validator

Once the researcher selects the metadata file and the directory of raw data files, the feedback screen will be displayed to the researcher.

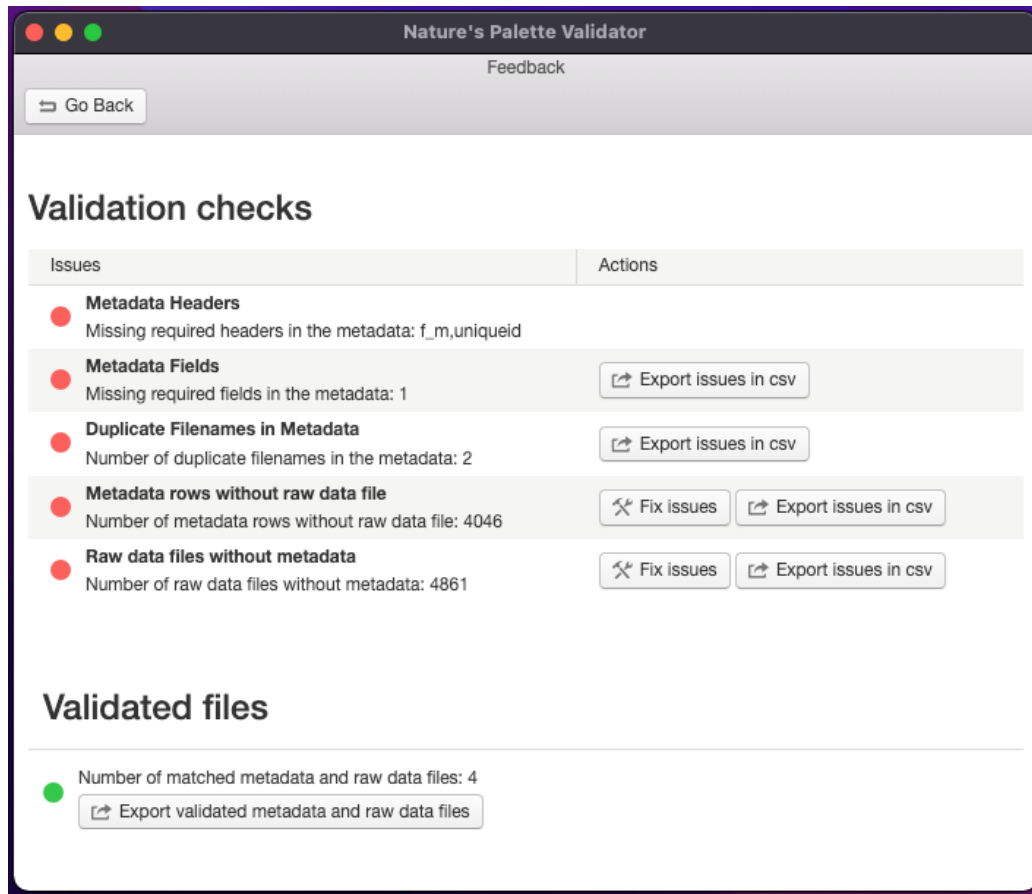


Figure 3 GUI: Feedback

The researcher can go through each validation step as discussed in the use case descriptions and decide to fix issues from within or outside of the application.

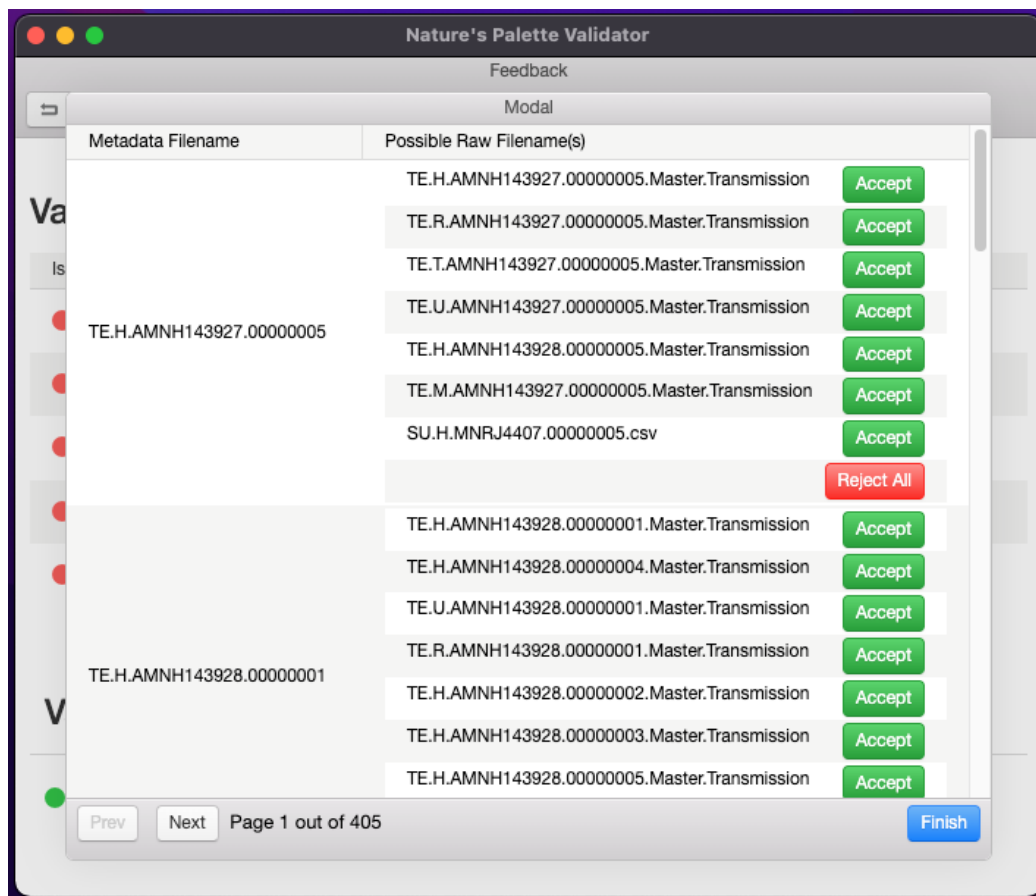


Figure 4 GUI: Fix Issues within the Application

## 4.2 Test Cases

In this section, I will cover a few cases to test the validation checks of the application

### Test Case 1



*Purpose:*

Verify that the application detects a missing header in the metadata file.

*Setup:*

Use a metadata file with missing required headers.

*Test Data:*

Action	Output
Validate the metadata file and the raw dataset through the application.	 <b>Metadata Headers</b> Missing required headers in the metadata: f_m,uniqueid
Add the missing required headers outside of the application.	
Validate the metadata file and raw dataset again.	 <b>Metadata Headers</b> All the required headers have been filled.

## Test Case 2


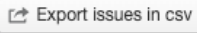
*Purpose:*


Verify that the application detects a missing required field in the metadata file.

*Setup:*

Use a metadata file with missing required fields.

*Test Data:*

Action	Output
Validate the metadata file and the raw dataset through the application.	 <b>Metadata Fields</b> Missing required fields in the metadata: 2 
Export the issues into a CSV file.	

Open the exported CSV file in a spreadsheet application.	<table border="1"> <thead> <tr> <th>filename</th><th>missingFields</th></tr> </thead> <tbody> <tr> <td>TE.H.AMNH143927.00000001.Master.Transmission</td><td>cataloguenumber</td></tr> <tr> <td>TE.H.CM63927.00000003.Master.Transmission</td><td>genus</td></tr> </tbody> </table>	filename	missingFields	TE.H.AMNH143927.00000001.Master.Transmission	cataloguenumber	TE.H.CM63927.00000003.Master.Transmission	genus
filename	missingFields						
TE.H.AMNH143927.00000001.Master.Transmission	cataloguenumber						
TE.H.CM63927.00000003.Master.Transmission	genus						
Add the missing fields in the metadata CSV file.							
Validate the metadata file and raw dataset again.	<div>  <b>Metadata Fields</b>  All required fields have been filled. </div>						

### Test Case 3



#### *Purpose:*

Verify that the application detects duplicate filename field values in the metadata file.

#### *Setup:*

Use a metadata file with a duplicate filename

#### *Test Data:*

Action	Output
Validate the metadata file and the raw dataset through the application.	<div>  <b>Duplicate Filenames in Metadata</b>  Number of duplicate filenames in the metadata: 1 <div>  Export issues in csv </div> </div>

Export the issues into a CSV file.				
Open the exported CSV file in a spreadsheet file.	<table><tr><th>filename</th></tr><tr><td>TE.H.AMNH143927.00000005.Master.Transmission</td></tr><tr><td>TE.H.AMNH143928.00000001.Master.Transmission</td></tr></table>	filename	TE.H.AMNH143927.00000005.Master.Transmission	TE.H.AMNH143928.00000001.Master.Transmission
filename				
TE.H.AMNH143927.00000005.Master.Transmission				
TE.H.AMNH143928.00000001.Master.Transmission				
Fix duplicate filenames using a spreadsheet editor.				
Validate the metadata file and raw dataset again.	<div><div></div><div><b>Duplicate Filenames in Metadata</b> There are no duplicate filenames in the metadata.</div></div>			




#### Test Case 4

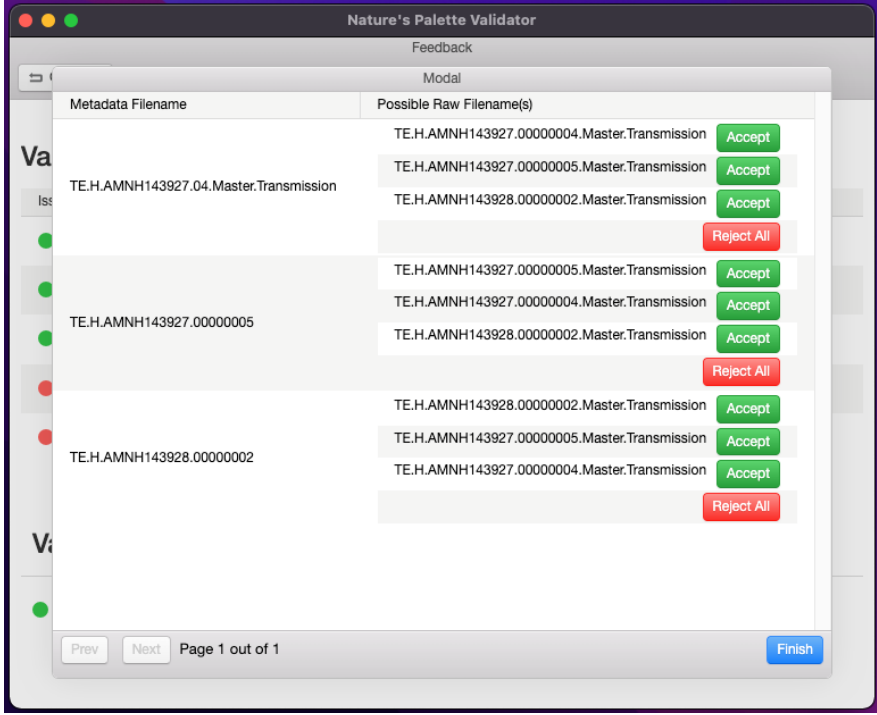
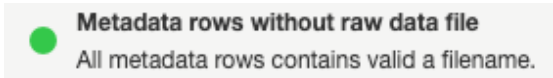
*Purpose:*

Verify that the application suggests an appropriate raw data file to metadata that does not correspond to any raw data file.

*Setup:*

Use a metadata file with incorrect filename spellings.

Action	Output
Validate the metadata file and the raw	<div>  <b>Metadata rows without raw data file</b>        Number of metadata rows without raw data file: 3       <div>  Fix issues          Export issues in csv       </div> </div>

dataset through the application.	
Choose to fix issues within the application.	
Accept the appropriate changes and click Finish.	

## Test Case 5


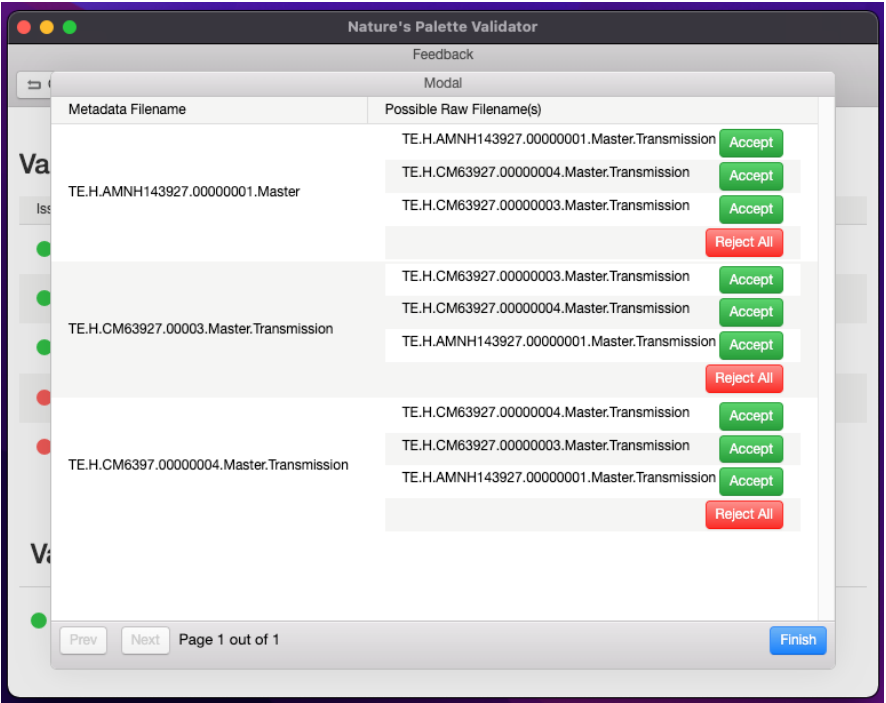

### Purpose:

Verify that the application suggests appropriate metadata to a raw data file that does not have any metadata.

### Setup:

Use raw data with incorrect filename spellings.

*Test Data:*

Action	Output
Validate the metadata file and the raw dataset through the application.	 <b>Raw data files without metadata</b> Number of raw data files without metadata: 3 <span>Fix issues</span> <span>Export issues in csv</span>
Choose to fix issues within the application.	
Accept the appropriate changes and click Finish.	 <b>Raw data files without metadata</b> All files has a valid a metadata.



# Chapter 4

## Conclusion

To summarize, the goal of the project is to build an offline cross-platform tool that will help researchers validate their dataset before submission to Nature's Palette. We followed the Agile SDLC model and were in constant contact with the clients throughout the project. We discussed the functionality of the application and described use case descriptions to understand user and system behavior. I studied several appropriate software frameworks to use for this project and decided to choose Electron to develop the tool according to the use case descriptions.

The next steps in this project would be to widely test the application by sending it to researchers to get their dataset validated. Based on their feedback, the tool can be easily improved by modifying current functionality or by adding new ones if required.

We predict this validator tool will be an important addition to the Nature's Palette project and will help researchers conveniently prepare their dataset to be shared in the online repository.

# Bibliography

- [1] R. Akhter. Nature's Palette – An Open Access Repository of Spectral Data.  
*School of Graduate Studies, Department of Computer Science, Memorial  
University of Newfoundland*
- [2] <https://electronjs.org/>. Electron. <https://electronjs.org/docs/latest/>, Last  
accessed on 2022-03-14.
- [3] E. S. Ristad and P. N. Yianilos. Learning String-Edit Distance. *IEEE  
Transactions on Pattern Analysis and Machine Intelligence*, 20(5), 1998.
- [4] A. Rasheed, B. Zafar, T. Shehryar, N. A. Aslam, M. Sajid, N. Ali, S. H. Dar,  
and S. Khalid. Requirement Engineering Challenges in Agile Software  
Development. *Hindawi Mathematical Problems in Engineering*, 2021.
- [5] M. Aoyama. Web-Based Agile Software Development. *IEEE Software*, 1998.
- [6] D. Sheiko. Cross-Platform Desktop Application Development: Electron,  
Node, NW.js, and React. *Packt*, 2017.
- [7] <https://itnext.io/>. String similarity – the basic know your algorithms guide.  
[https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-  
3de3d7346227/](https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227/), Last accessed on 2022-04-01.