xrf-explorerV2

FILE: XRF_ExplorerV2_softwaretransferdocument.pdf

General information on the software documentation
version: 2024 October 02

---

xrf-explorerV2 is a research software toolkit developed by students at Eindhoven University of Technology (TU/e), for the Van Gogh Museum Amsterdam in a partnership with ASML. The toolkit is intended to be used for the integrated exploration of multimodal images, spectral data, and chemical mappings of a painting. The toolkit is developed to assist in the conservation science practice.

Project
The original prototype (V1) was created by a TU/e master's student from January–September 2023. The current version (V2) was developed by a team of TU/e bachelor students from April–July 2024.

Documentation
Five documentation files were delivered by the development team.
Together they form a comprehensive documentation package:

- XRF_ExplorerV2_userrequirementsdocument.pdf
- XRF_ExplorerV2_softwaredesigndocument.pdf
- XRF_ExplorerV2_acceptancetestplan.pdf
- XRF_ExplorerV2_softwaretransferdocument.pdf
- XRF_ExplorerV2_softwareusermanual.pdf

Acknowledgements
For the development, testing, and documenting of xrf-explorer, test data was provided by the Museum of Modern Art (New York) to the development team. It consisted of images and xrf scanning data of the "Portrait of Joseph Roulin" painting by Vincent van Gogh. Images of this painting are included in some of the documentation files for the purpose of explaining the software usage.

For more detailed information on this painting, including image licenses, we refer to:
https://www.moma.org/collection/works/79105

Software
The xrfexplorerV2 software itself (codebase and code documentation) can be found at the github repository "Olive-Groves/xrf-explorer" (see: https://github.com/olive-groves).

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Software Transfer Document

2IPE0 SOFTWARE ENGINEERING PROJECT

July, 2024

GROUP 6

PROJECT TEAM

Adrien Verriele	1710303
Diego Rivera Garrido	1674196
Dirk Burgers	1653873
Iliyan Teofilov	1671952
Ivan Ivanov	1661469
Jan Bulthuis	1696866
Lotte Lakeman	1668137
Massimo Leal Martel	1662147
Pablo Benayas Penas	1667939
Ruben Savelkouls	1695347
Sonia Maxim	1675656

PROJECT MANAGERS

Antreas Efstathiou
Konstantinos Chanioglou

SUPERVISOR

Prof dr. R. Bloo

CLIENTS

Ana Martins
Lars Maxfield
Marco Roling

ABSTRACT

This document, the Software Transfer Document (STD), serves to guide the user through the process of building, installing and developing the XRF Explorer 2.0. The XRF Explorer 2.0 is a web application aimed at conservation scientists, that facilitates the analysis of paintings composition. This application builds upon the foundations laid by Dominique van Berkum's master thesis project introducing the proof of concept for XRF-XPLORER [1], and it is tailored to fit the requirements of the clients.

The STD also contains a report on the acceptance test, any modifications made thereafter, and a unit test report. The document is intended for future developers and testers, but also for users of the XRF Explorer 2.0.

Contents

DOCUMENT STATUS SHEET

General

Document title	Software Transfer Document
Document identifier	STD/1.0
Document authors	Adrien Verriele Lotte Lakeman
Document status	Final

Document history

Version	Date	Authors	Reason
0.1	24-04-2024	Lotte Lakeman	Created document from document template.
0.2	16-05-2024	Lotte Lakeman	Created the first draft of the abstract and introduction.
0.3	06-06-2024	Adrien Verriele	Wrote sections 2-4.
0.4	24-06-2024	Adrien Verriele	Wrote sections 5-7.
0.5	25-06-2024	Adrien Verriele	Wrote sections 8.
0.6	27-06-2024	Adrien Verriele	Wrote sections 9.1.
0.6.1	29-06-2024	Adrien Verriele	Started section 9.2
0.6.2	30-06-2024	Adrien Verriele	Continued section 9.2
0.6.3	01-07-2024	Adrien Verriele	Continued section 9.2
0.6.4	02-07-2024	Adrien Verriele	Continued section 9.2
1.0	04-07-2024	Adrien Verriele Lotte Lakeman	Finalized the test report.

DOCUMENT CHANGE RECORDS

Version	Date	Section	Reason
0.1	24-04-2024	Entire document	Created the document
0.3	06-06-2024	Sections 2-4	Initial version of these sections
0.4	24-06-2024	Sections 5-7	Initial version of these sections
0.5	25-06-2024	Section 8	Initial version of this section
0.6	02-07-2024	Section 9	Initial version of this section
1.0	04-07-2024	Section 9	Finalized test report, thus finalizing the entire document

1 INTRODUCTION

1.1 Purpose

This document, the Software Transfer Document (STD), serves as a document to transfer the XRF Explorer 2.0 to the clients. It contains all the prerequisites and instructions to build and install the XRF Explorer 2.0. Additionally, a report on the Acceptance Test Plan (ATP) can be found and any changes to the software thereafter in agreement with the clients are also documented in the STD. Upon reading and following the steps of this document, the reader should be able to continue development on the XRF Explorer 2.0.

The document is intended for future developers of the XRF Explorer 2.0. Prior knowledge of Git, and the command-line is expected from the reader. Additionally, experience in Python [2], Flask [3], Vue [4], TypeScript [5], and D3 [6] (or the ability to easily pick these up) is expected from the reader.

1.2 Scope

The XRF Explorer 2.0 is an open-source web application intended for conservation scientists developed by a group of students for their Software Engineering Project (SEP). The intended goal of the software is to help conservation scientists better visualize and understand Micro-XRF data alongside RGB, UV and X-Ray images of paintings.

The XRF Explorer 2.0 streamlines the visualization process of paintings and aids conservation scientists in their analysing process. The XRF Explorer 2.0 supports multiple charts (elemental charts (line and bar) and spectral charts) to visualize painting data. Additionally it uses a layer system to facilitate switching between different image types of the painting the user wants to analyse. The application also has a viewer in which the painting's images are visualized. Special attention was paid to ensuring that the web application shows the high resolution RGB image in the viewer without noticeable latency.

A proof of concept created by Dominique van Berkum for their master's thesis in Data Science and Artificial Intelligence existed prior to the development of the XRF Explorer 2.0 [1]. This proof of concept was requested by the same clients as the XRF Explorer 2.0. The XRF Explorer 2.0 turns this proof of concept into a real, web-based application with additional "quality of life" improvements.

1.3 List of definitions

Term	Description
Base image	High resolution RGB image of the painting.
Bitmasks	A two-dimensional boolean matrix utilized to specify selected pixels within an image. Each element of the matrix corresponds to a pixel in the associated image and is set to true if the pixel is selected and false otherwise. color clusters computed using the color segmentation algorithm.
Clients	Ana Martins, Lars Maxfield, and Marco Roling.
Color clusters	The colors resulting from applying the color segmentation algorithm to the whole RGB image/per element.
Color segmentation	Clustering of the pixels in the image according to their color, possible limited to the pixels containing above a threshold of a certain element.
Color segmentation window	Visualization view that displays the distribution of color segments throughout the painting.

Continued on next page

Term	Description
Control point	A point in pixel space used to define the coordinates of a distinguished feature in an image.
Data cube	raw or processed data cube.
Data registration	The process of aligning images using (given) control points such that all the images have the same dimension and orientation.
Deep clone	Copy of an object whose properties do not share the properties of the source object.
Dialog	A small, temporary window that appears on the screen to facilitate user interaction, often requiring user input or action before it can be dismissed.
Element chart	Chart that displays the elemental composition across the painting as well as the selected area in the form of line/bar charts.
Dimensionality reduction	Transformation of data from high-dimensional space into low-dimensional space.
Elemental composition	Abundance and distribution of the elements present in the painting.
Elemental map	Map/image derived from the processed data, visualizing the distribution of one element across the painting.
Elements	Chemical elements present in the painting.
Embedding	The output resulting from the application of a dimensionality reduction method to a dataset is referred to as an embedding.
Layer	Discrete compound that contains an individual contextual image or elemental distribution map.
Layer system	Structured arrangement of different layers, which allows for the layers to be moved, removed and added.
Lens	Mouse position driven ocular that looks through the top layer to a selected layer below.
Lens viewing mode	Mode in which the user controls the lens.
(Main) Viewer	Main large interactive visualization where spatial data is presented.
Mipmap	Pre-calculated, lower resolution version of an image of a certain level. The higher the level, the lower the resolution.
Polygon selection tool	Selection tool which allows for creation of polygonal shapes (both for the dimensionality reduction view and the main viewer).
Processed data	Processed datacube (3-dimensional) of elemental distribution data obtained from processing the raw data that, for each pixel, gives the abundance of the different elements present.
Data source	Collection of raw, processed and contextual data that correspond to the same painting and are used to analyze said painting.
Raw data	Raw datacube (3-dimensional) of spectral data obtained from the XRF scanner that, for each pixel, gives the intensity of the X-ray fluorescence emitted at different energies the elements in the painting.
Recipe	A set of control points over a set of images of the painting, linked across multiple images for perspective correction and alignment to register the data.

Continued on next page

Term	Description
Rectangle selection tool	Selection tool which allows for creation of rectangular shapes (both for the dimensionality reduction view.
Rem	CSS unit of measurement that represents the font size of the HTML root element
Rpl file	File containing information about the spectral data, such as the dimensions.
Selection tool	Tool that allows the user to select a subset of the painting.
Selected areas	Areas selected by the user using the selection tool in the main viewer.
Spectra	Representation of the intensity (counts) of the fluorescence emitted by the materials in the painting as a function of energy (KeV).
Spectral chart	Chart of the average fluorescence spectrum acquired over the whole painting or selected area.
Toolbar	Tool bar located at the middle bottom of the application window, provides quick access to the selection tools, lens mode, and some main viewer settings.
Visualization views	Any of the visualizations that are not the main viewer, where different plots and graphs can be displayed.
Workspace	Collection of files that make up a data source together with the "`workspace.json`" file which describes the files the project contains and some extra parameters.

1.3.1 Acronyms and abbreviations

Term	Description
ATP	Acceptance Test Plan
URD	User Requirements Document
XRF scanner	X-Ray fluorescence scanner
DRS	Dimensionality reduction system
URC	User Requirements Constraint
URF	User Requirements Functional
STD	Software Transferal Document
SDD	Software Design Document

1.4 List of references

[1] D. V. B. van Berkum, "Xrf-xplorer: An interactive visual exploration tool for micro-x-ray fluorescence scanning data on paintings," Master's thesis, Eindhoven Univerisity of Technology, September 2023.

[2] "Python." `https://www.python.org/`.

[3] "Flask." `https://flask.palletsprojects.com/en/3.0.x/`.

[4] "The progressive javascript framework." `https://vuejs.org/`.

[5] “Typescript is javascript with syntax for types..” https://www.typescriptlang.org/.

[6] “The javascript library for bespoke data visualization.” https://d3js.org/.

[7] “Pytest.” https://docs.pytest.org/en/8.2.x/.

[8] “Pytest action.” https://github.com/pavelzw/pytest-action.

[9] “Wemake python styleguide.” https://github.com/wemake-services/wemake-python-styleguide.

[10] “Eslint.” https://eslint.org/docs/latest/use/getting-started.

[11] “Xrf explorer 2.0 gihub repository.” https://github.com/olive-groves/xrf-explorer.

[12] “Xrf explorer 2.0: User requirements document,” tech. rep., Eindhoven University of Technology, 2024.

[13] “Xrf explorer 2.0: Acceptance test plan,” tech. rep., Eindhoven University of Technology, 2024.

[14] “Xrf explorer 2.0: Software design document,” tech. rep., Eindhoven University of Technology, 2024.

[15] “Xrf explorer 2.0: Software user manual,” tech. rep., Eindhoven University of Technology, 2024.

[16] “Sphinx documentation.” https://www.sphinx-doc.org/en/master/.

2 BUILD PROCEDURE

This section provides a detailed guide on the necessary steps and prerequisites for building the XRF Explorer 2.0 software on a local machine for development with Windows (versions 10 or 11) or Ubuntu as an operating system. While the software is expected to be compatible with most recent Ubuntu versions, it has been tested only on version 24.04.

2.1 Prerequisites

Before proceeding with the build process, ensure that the following software tools are installed on your local machine. These tools are necessary for building, managing dependencies, and running the XRF Explorer 2.0 software.

- **Chocolatey (only for Windows)**
  - *Description:* Package manager for Windows, used to install make, which is needed to build the documentation.
  - *Installation Windows:* Follow the instructions outlined in `https://chocolatey.org/install`
- **Git** (Optional)
  - *Description:* Used for source code management and version control. Recommended for development.
  - *Installation on Windows:* Download and install Git from `https://git-scm.com/download/win`.
  - *Installation on Ubuntu:* Run the command `sudo apt install git` in the terminal.
- **Python 3.12**
  - *Description:* Programming language used to run the back-end of the application. It also requires `pip`, its package manager, which comes installed with it on Windows.
  - *Installation on Windows:* Download the appropriate installer from `https://www.python.org/downloads/` and follow the on-screen instructions.
  - *Installation on Ubuntu:* Run the command `sudo apt install python 3.12` and `sudo apt install python3-pip` in the terminal.
- **npm**
  - *Description:* Package manager for node, used to manage project dependencies on the frontend.
  - *Installation for both Windows and Ubuntu:* Follow the instructions outlined in `https://nodejs.org/en/download/package-manager`

2.2 Steps for building the application

1. **Obtain the project repository**
   - *Option 1: Clone the repository* - You can clone the project's Git repository to your local machine by executing the following command in your desired directory:

     ```
     git clone --branch main https://github.com/olive-groves/xrf-explorer.git
     ```
   - *Option 2: Manually download the repository archive* - You can download a .zip file of the project repository by visiting `https://github.com/olive-groves/xrf-explorer`. Make sure to select the *main* branch, then click the green *Code* button, and choose "Download ZIP".
2. **Open the project directory** - Navigate to the directory where you cloned or extracted the project.

   ```
   cd xrf-explorer
   ```
3. **Create and Activate the Python virtual Environment**

- Install `virtualenv`:

  ```
  pip install virtualenv
  ```

- Create a virtual environment named `env`:
  On Windows:

  ```
  py -m venv env
  ```

  On Linux:

  ```
  python3.12 -m venv env
  ```

- Activate the virtual environment:
  On Windows:

  ```
  env\Scripts\activate
  ```

  On Linux:

  ```
  source env/bin/activate
  ```

4. **Install Backend Dependencies** - Install all required Python packages listed in `requirements.txt`:

   ```
   pip install -r requirements.txt
   ```

5. **Install Frontend Dependencies**

   - Navigate to the frontend directory:

     ```
     cd xrf_explorer/client
     ```

   - Install all required Node.js packages:

     ```
     npm install
     ```

6. **Build the Frontend** - Build the Vue.js frontend for production:

   ```
   npm run build
   ```

   When developing on the frontend, using

   ```
   npm run dev
   ```

   will start a development server with hot-reloading allowing you to see the changes in real-time as you code.

7. **Build the documentation**

   - Install make:
     On Windows, use Chocolately:

     ```
     choco install make
     ```

     On Linux, make is already installed by default.

   - Navigate to the root of the project:

     ```
     cd xrf-explorer
     ```

   - Build the ReST files:

     ```
     sphinx-apidoc(.exe) -f -o ./documentation/sphinx/source/ .
     ```

     The .exe extension may be omitted in some environments.

   - Navigate to the sphinx folder:

     ```
     cd documentation/sphinx
     ```

   - Generate HTML documentation:

     ```
     make html
     ```

2.3 Continuous Integration

This section will describe how the Continuous Integration and Continuous Development (CI/CD) pipeline is set up for the Github repository. This is a common procedure in software development that ensures that new changes to the code base do not break the code or violate coding standards, so that it can be safely integrated into the existing code. In this section the CI/CD system implemented with Github Actions is described.

2.3.1 Branching system

The *main* branch is the branch containing the deployed code. It has been tested, approved and considered stable and ready for use. An update of the software is realised when code is merged from the *development* branch, requiring one approval.

The *development* branch is the branch into which finished new *feature* branches are merged, before joining the *main* branch. It is the only branch that can be merged into *main*. Pushing directly to it is not permitted.

The other branches are so called *feature* branches, each used for the implementation and testing of a single feature. When the implementation and testing are finished and the feature is considered done, a Pull Request has to be made to have the ability to merge it into *development*.

A Pull Request requires two approvals to be merged into *development*. Moreover, the Pull Request of a branch will be only accepted if the last commit to the branch passes all the CI/CD pipeline, and if all its comments have been resolved.

2.3.2 Pipeline

When a new commit is pushed to a branch, the test pipeline is triggered. It performs a series of tests on the code to check the functionalities and correctness of the program, as well as the coding style. All tests are automated so that the user does not need to perform any other step. The following tests are performed:

1. **Python unit tests**
   The back-end dependencies are installed, and all the methods of the test classes inside `xrf-explorer/tests/` are executed and checked for their assertions to be true, with `pytest` [7] and `pytest-actions` [8].
2. **Python code style**
   The Python code inside `xrf_explorer/server/` is inspected and its style is checked with `wemake-python-styleguide` [9] to satisfy Python's conventional coding style. This check is not required to be fully successful for the test to pass.
3. **Typescript code style**
   The TypeScript code of the Vue.js components inside `xrf-explorer/xrf_explorer/client/` is inspected and its style is checked with `ESLint` [10] to satisfy TypeScript's conventional coding style.
4. **Client build**
   The front-end dependencies are installed, then `npm run build` is executed to ensure that the front-end build without errors.
5. **Lines of comment to code ratio**
   The percentage of commented lines per file per language is computed using `cloc-action`. The result is stored in a build `artifact` called `commentsReport.md`. This acts as a first indication of the degree of documentation of the code. This tests does not require a minimum comment percentage and is purely indicative.

2.4 Possible errors

In this section, the errors that may occur while executing the build procedure are discussed and possible fixes are proposed. The previously given code is assumed to be syntactically correct.

- In case the prerequisites listed in chapter 2.1 are not installed, an error can occur where a certain package is not recognized as an internal or external command when executing an instruction from chapter 2.2. To solve this, the package needs to be installed by following the corresponding instructions before the build can be executed again.

3 INSTALLATION PROCEDURE

In this section, the environment of the software and the different steps to install it are described. The possible problems during the installation and the needed time are detailed.

3.1 Environment and installation steps

First, the environment in which the application should be installed and the steps required to successfully install and run it are explained.

3.1.1 Environment

- **Server side:** The back-end of the application is meant to be installed on a server from which it can serve requests remotely. It requires a x86-64 based architecture with at least 16GB of RAM. The supported operating systems are Windows 10 and 11, and Ubuntu Server LTS version 24.04 or higher. The application is run inside a Python virtual environment (see Installation steps 3.1.3). It is also possible to run the application locally on the same machine that accesses the client, provided it satisfies the above requirements.
- **Client side:** The front-end of the application is accessed via a web browser. The supported browsers are Google Chrome version 124 or higher, Mozilla Firefox version 125 and higher and Safari version 17 or higher.

3.1.2 Software requirements

For the application to be able to run, the following software has to be installed on the machine:

- **Git** (Optional)
  - *Description:* Used for cloning the source code onto the server.
  - *Installation on Windows:* Download and install Git from `https://git-scm.com/download/win`.
  - *Installation on Ubuntu:* Run the command `sudo apt install git` in the terminal.
- **Python 3.12**
  - *Description:* Programming language used to run the back-end of the application. It also requires `pip`, its package manager, which comes installed with it on Windows.
  - *Installation on Windows:* Download the appropriate installer from `https://www.python.org/downloads/` and follow the on-screen instructions.
  - *Installation on Ubuntu:* Run the command `sudo apt install python 3.12` and `sudo apt install python3-pip` in the terminal.
- **npm**
  - *Description:* Package manager for JavaScript, used to manage project dependencies on the front-end.
  - *Installation for both Windows and Ubuntu:* Follow the instructions outlined in `https://nodejs.org/en/download/package-manager`

3.1.3 Installation steps

1. **Obtain the source code**
   - *Option 1: Clone the repository* - You can clone the project's Git repository to the server machine by executing the following command in your desired directory:

     ```
     git clone --branch main https://github.com/olive-groves/xrf-explorer.git
     ```

- *Option 2: Manually download the repository archive* - You can download a .zip file of the project repository by visiting `https://github.com/olive-groves/xrf-explorer`. Make sure to select the *main* branch, then click the green *Code* button, and choose ”Download ZIP”.

2. **Open the project directory** - Navigate to the directory where you cloned or extracted the project.

   ```
   cd xrf-explorer
   ```

3. **Create and Activate the Python virtual Environment**

   - Install `virtualenv`:

     ```
     pip install virtualenv
     ```

   - Create a virtual environment named `env`:
     On Windows:

     ```
     py -m venv env
     ```

     On Linux:

     ```
     python3.12 -m venv env
     ```

   - Activate the virtual environment:
     On Windows:

     ```
     env\Scripts\activate
     ```

     On Linux:

     ```
     source env/bin/activate
     ```

4. **Install Backend Dependencies** - Install all required Python packages listed in `requirements.txt`:

   ```
   pip install -r requirements.txt
   ```

5. **Install Frontend Dependencies**

   - Navigate to the frontend directory:

     ```
     cd xrf_explorer/client
     ```

   - Install all required Node.js packages:

     ```
     npm install
     ```

6. **Build the Frontend** - Build the Vue.js frontend for production:

   ```
   npm run build
   ```

7. **Set IP address and port** (Only for remote server setup)

   - Open the *backend.yml* file in `xrf-explorer/config`.
   - Change the **”bind-address”** and **”port”** attributes to `0.0.0.0` and the desired port.

8. **Run the application code**

   - Navigate back to the root of the project directory:

     ```
     cd ../../
     ```

   - Run the python in the terminal for local use, or as a SystemD service for server use:
     Locally on Linux:

     ```
     python run.py
     ```

     For server use on Ubuntu:

     - Create a user to run the application.

       ```
       sudo useradd -m xrf
       ```

     - Ensure all xrf-explorer files are owned by the `xrf` user.

       ```
       sudo chown xrf:xrf /{path to xrf-explorer}/ -R
       sudo chown xrf:xrf /{path to uploads directory}/ -R
       ```

     - Create the SystemD service by storing the following in `/etc/systemd/system/xrf-explorer.service`, making sure to change `/opt/xrf-explorer` to the installation directory of the application, and to change `/etc/xrf-explorer/config.yml` to the location where the configuration file is stored.

```
[Service]
User=xrf
Group=xrf
WorkingDirectory=/opt/xrf-explorer
ExecStart=/opt/xrf-explorer/env/bin/python ./run.py -c /etc/xrf-explorer/config.yml
Restart=always

[Install]
WantedBy=multi-user.target
```

- Start and enable the service

```
sudo systemctl enable --now xrf-explorer
```

On Windows:

```
start python run.py
```

To access the client of the application, open a supported web browser on the client machine and navigate to the IP address or URL of the server. If the application is run locally, navigate to `localhost:<port>`.

3.2 Possible problems

In this section, the problems that may arise when executing the installation procedure are described, alongside with fixes.

- In case the prerequisites listed in chapter 3.1.2 are not installed, an error can occur that a certain package is not recognized as an internal or external command when executing a step of chapter 3.1.3. To solve this, the package needs to be installed by following the corresponding instructions before the build can be executed again.

4 CONFIGURATION ITEM LIST

This section describes the deliverables that will be sent to the customer. All documents will be transferred in PDF format, while the source code, which includes the program code and its own documentation, will be accessible from the Github repository [11].

4.1 Documentation

The following documents will be handed to the customer:

- User Requirements Document [12]
- Acceptance Test Plan [13]
- Software Design Document [14]
- Software User Manual [15]
- Software Transfer Document (this document)

4.2 Source code structure

The source code of the project can be accessed on the Github page [11]. The final version of the project is contained in the *main* branch.

The source code is structured as follows:

- The `config` directory contains the configuration files for the back-end, which sets all necessary folder paths and constants used across the back-end code.
- The `documentation` directory contains the files needed to build the code documentation with Sphinx [16]. It consists of both the Python documentation, which is automatically generated, and the Vue.js documentation, which is made manually for each Vue.js file, and explains how to navigate the corresponding part of the user interface.
- The `logs` directory contains the server logs that are printed by the Python code.
- The `tests` directory contains all necessary files for Python unit testing. Each Python module has its own test file, in the same file structure the the original code. The `resources` folder contains a special configuration file for the tests and resource files accessed by the test functions to assess their correctness (images, text files, etc).
- The `xrf_explorer` directory contains the actual code of the application. It consists of two sub directories: `client` for the front-end code and `server` for the back-end code:
  - Inside `client`, the different configuration files for the front-end code can be found, as well as the actual code in the `src` folder. The unit tests for the front-end are located in `src/components/\_\_tests\_\_`
  - The `server` folder contains the Python modules for the different features, as well as the `routes.py` file which contains the API routes called by the client. The `data` folder contains the data sources that have been uploaded by the users from the front-end. The `temp` directory stores the temporary files accessed by the clients, generated from the primary data source files.

5 ACCEPTANCE TEST REPORT SUMMARY

The acceptance test for XRF Explorer 2.0 was performed on June 19th, 2024 and lasted from 11:00 until 13:00. The test procedures described in version 1.4 of the Acceptance Test Plan [13] were followed. This document was signed by the customer before the acceptance test was performed, thereby agreeing that the test procedures presented in it correctly assess the capabilities of the software. The three clients, the two project managers, the supervisor and the eleven team members were present during the test. Two of the team members executed all 13 test procedures, which contained 65 tests overall. Following the definition of passing from the Acceptance Test Plan [13], all acceptance tests were completed successfully. Thus, the clients agreed that the acceptance test had passed, while requesting minor changes regarding the interface and user experience mainly. These requests are discussed in section 7.

6 SOFTWARE PROBLEM REPORTS

Here the problems encountered during the acceptance test and the software transfer process are listed and described.

- **Problem while uploading spectral data file:** It occurred some times that the upload of a 10GB file could not completely succeed. However it appeared that this was due to a network configuration error on the clients' computers and was acknowledged not to be an issue from the application.

7 SOFTWARE CHANGE REQUESTS

In this section the changes requested by the clients following the acceptance test are described and explained.

7.1 High priority changes

The following change requests have been selected by the clients as having a higher priority. These are the changes that have actually been implemented in the software transfer phase, as explained in section 8.

- **Workspace creation without spectral or elemental cube:** In some scenarios the user does not have the spectral or elemental cube files, therefore it should be possible to be able to create a workspace with none, one, or both of them.
- **Conversion of spectral chart x-axis from channel to energy:** The spectral chart should be plotted against energy, a concrete unit, rather than channels to be more understandable by users.
- **Ability to delete workspaces from the client:** Users should have the ability to delete workspaces by themselves through the client directly, rather than by deleting files on the server.
- **Long filenames not hiding delete button in workspace creation dialog:** When the users imports a file with a long name, the name should not hide the delete button for this file in the workspace creation dialog.
- **Better visibility of the toolbar in light mode:** While in light mode, there should be enough contrast in the toolbar for the user to see which tool they are hovering their cursor over.
- **User guidance for the scale of the dimensionality reduction threshold:** The dimensionality reduction threshold should either be changed to a percentage or a message should be displayed to inform the user of its possible values.
- **Loading icon on spectral chart:** Users should be informed that the spectral chart data is loading with an icon.
- **Moving reset painting position button to the toolbar:** The ”reset painting position” button should be moved to the toolbar to prevent users from clicking it by mistake.
- **Better readability of elemental chart:** The elemental chart should display the selection elemental average in the form of a line chart, and only the global elemental average in the form of a bar chart.
- **Change in the lasso/freehand selection name:** The lasso/freehand selection should use the same name, and instead be called polygon selection, since it is closer to what it actually performs.
- **Easier option to clear selection:** The user should have the option to clear the selection easily, either via a keybind or a button.
- **Readability of elemental map mixing:** The pixels where two or more elemental maps intersect should be better visible to the user, either thanks to a proper color mixing calculation or the possibility to reorder the stack of elemental layers.
- **Option to lock the lens:** The user should have the option to lock the lens in place within the main viewer and move their mouse elsewhere.
- **Option to change the number of colors in the color segmentation:** The user should have the possibility to change the number of colors partitioned by the color segmentation, or to change the elemental threshold for the segmentation.
- **Notify users when the workspace being created already contains files:** The user should be notified when a folder with the same name as the workspace they are creating already contains some files, so they can choose to reuse these files or not.

7.2 Low priority changes

The following change requests have been assigned a lower priority by the clients, and were not implemented due to time constraints.

- **User guidance to reset sliders:** There should be a specific button or message to inform the user how to reset the slider values, double clicking without explicitly being informed of the possibility is not intuitive.
- **Ability to zoom in and out of the spectral chart:** Users should be able to zoom in and out inside of the spectral chart, or dynamically change the x-axis range, to better analyze the graphs.
- **Language of some labels and texts:** The labels and texts native to the browser should also be displayed in English, rather than the browser's configured language.
- **Display software version:** The software version should be visible within the client.
- **Explanations for each button:** Each button should display explanation about its functionality when hovered over.

8 SOFTWARE MODIFICATION REPORTS

In this section, the modifications implemented in response to the high priority change requests listed in section 7.1 during the software transfer phase are described.

8.1 Dimensionality reduction threshold

For a more intuitive user experience, the threshold determining the pixels selected for the dimensionality reduction has been changed from a grayscale integer (0-255) to a percentage.

8.2 Color segmentation parameters

To improve the adaptability of the software, the possibility to change the color segmentation parameters has been added. Users can select the number of color clusters to generate and a percentage elemental threshold to determine the selected pixels. This modification is visible in Figure 1.

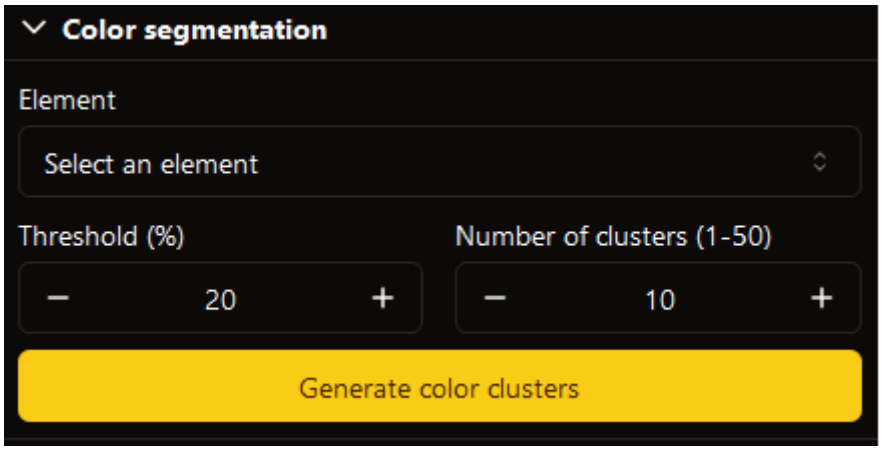

Figure 1: Color segmentation parameters

8.3 Elemental line and bar chart

To improve the readability of the elemental chart, specifically in comparing a selection of the painting to the global average over the painting, the global elemental average is represented as a bar chart, while the selection average is represented as a line on top of it. This modification is visible in Figure 2.

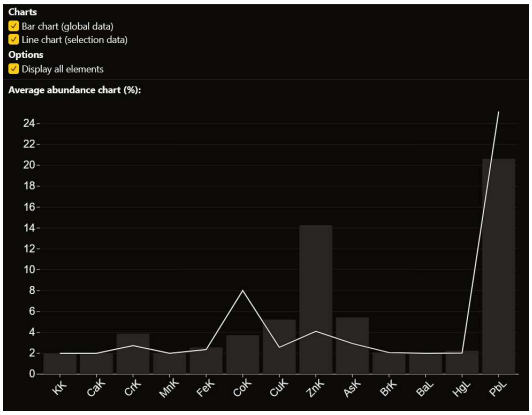

Figure 2: Elemental chart

8.4 Spectral chart rescaling

For a more intuitive and concrete reading of the spectral chart, the x-axis has been changed from channels (0-4096) to energy, in keV (0-40). This also includes the selection of the excitation energy for the theoretical spectrum, and the selection of binning parameters in workspace creation.

8.5 Reset client and painting position functionality moved

To prevent the risk of users accidentally resetting the client by right-clicking, the ”reset client” prompt has been moved to the main drop-down menu in the header bar. It also displays a dialog asking for the user to confirm their choice. The ”reset painting position” functionality has also been moved to the toolbar for better visibility. These changes can be seen in Figures 3 and 4 respectively.

Figure 3: Reset client dialog

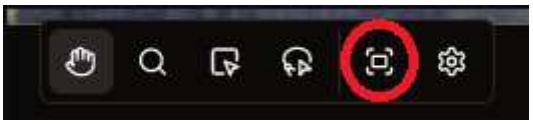

Figure 4: Reset painting position button

8.6 Order independent elemental map transparency

To improve the visibility of the intersection of two elemental maps, a possible option was to allow users to reorder the stack of elemental maps, so that maps on lower layers will be seen through the ones on higher levels depending on their opacity. This option had the disadvantage of being dependent of the order of the maps, as seeing a map through another one will not give the same result visually as the other way around. However, the chosen approach, was to compute a new color for the intersection depending on the original map color. This way, users can clearly identify where two or more maps intersect.

8.7 Workspace management

To allow more file management from the client side, without having to perform this from the server, a dialog to delete workspaces has been added. Users can access it from the ”delete project” button in the workspace window, which can be seen in Figure 5. Moreover, when creating a workspace with the name of a folder that already contains some files, a dialog appears to list these files, and give the user the choice to reuse them. This dialog can be seen in figure 6.

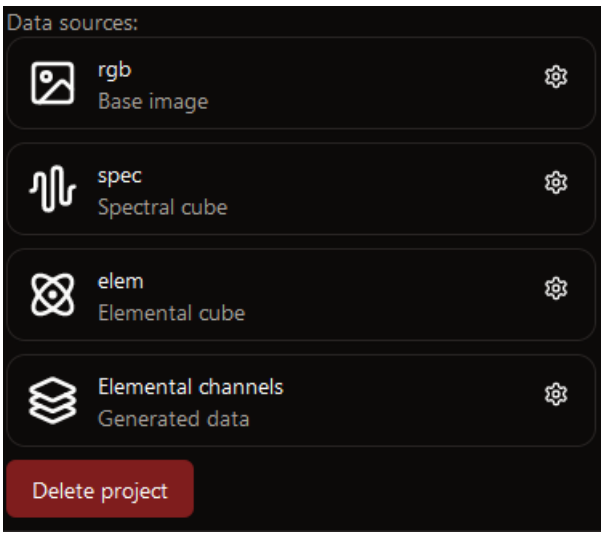

Figure 5: Delete project button

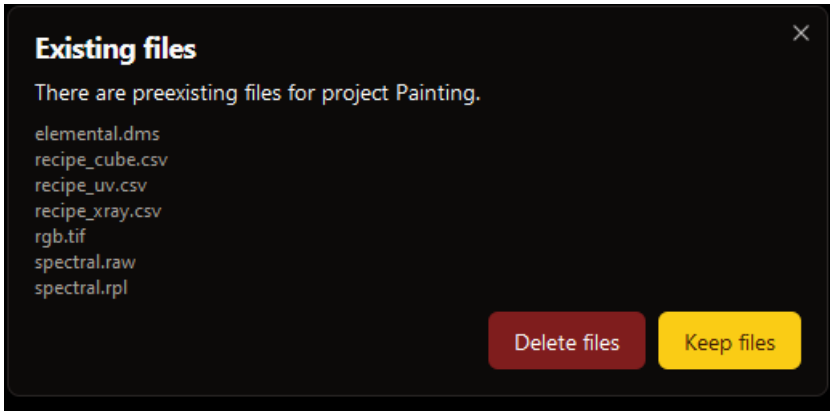

Figure 6: Existing files dialog

8.8 Workspace creation without elemental or spectral cube files

For a better adaptability of the software, the obligation of uploading an elemental or spectral cube file to create a workspace has been removed. Users can create a workspace with or without these files. All actions requiring a missing files will be disabled.

8.9 User interface improvements

Finally, various small changes have been made to the front-end of the application to improve the user experience.

- **Long filenames handling:** Filenames that are too long are cropped in the workspace creation interface in order not to hide their respective delete button.
- **Color scheme contrast:** The contrast of both the light and dark color schemes has been increased to better distinguish selected components.
- **Spectral chart loading icon:** A loading wheel has been added to indicate users that the data for the spectral chart is being fetched.
- **Lasso and freehand selection renaming:** All occurrences of ”lasso” or ”freehand” selection in the back-end and the front-end have been replaced with ”polygon” for a more accurate description of its behavior.

- **Clear selection button:** A button has been added for an easier reset of the selection.
- **Lens locking option:** The possibility for users to right click to lock the lens in place and freely move their mouse has been added.

9 UNIT TEST REPORT

9.1 Unit test location and execution

This section describes the location of the tests for the XRF Explorer application can be found, as well as instructions to execute them.

9.1.1 Tests location

Back-end tests location

The unit tests for the back-end of the application are located in the `/tests` subdirectory of the root folder of the project. This folder contains at least one Python test file for each python mode of the source code. The Python test file consists of a class, containing tests and helper functions. The `resources` folder contains the necessary resources for running the tests, such as dummy files and folders to simulate a workspace environment. These files are organised in subfolders, each containing the resource files to test a specific module.

Front-end tests location

The unit tests for the front-end of the application are located in `xrf_explorer/client/src/components/__tests__` from the root folder of the project. This folder contains `utils.test.ts` which tests the `utils` code file.

9.1.2 Tests execution

The test execution procedure is described below. The instructions assume that the build procedure has already been followed successfully.

Back-end tests execution

1. Open a terminal and navigate to the root folder of the project.

   ```
   cd xrf-explorer
   ```

2. Execute all tests.

   ```
   pytest /tests
   ```

3. (Optional) For a more extensive and explanatory output the following command can be run instead.

   ```
   pytest /tests -vv
   ```

front-end tests execution

1. Open a terminal and navigate to the front-end folder of the project.

   ```
   cd xrf-explorer/xrf_explorer/client
   ```

2. Execute all tests.

   ```
   npm run test
   ```

Following these steps will show the test results in the terminal, indicating for each test if it succeeded or failed, with the error code or result in the latter case.

9.2 Test report

This section lists the tests of the application and their outcome. It also details the test coverage with an explanation for low or no coverage of certain units.

9.2.1 Unit test results

The results of the unit tests are listed below, grouped by module and function, together with whether they passed or failed. It should be noted that the function listed is the name of the actual function and not of the test function.

API routes

Route	Test description	Result
`/api`	returns the correct number of routes	passed
`/api/data_sources`	returns the correct number of workspaces	passed
`/api/<data_source>/workspace`	returns the correct number of files in the workspace	passed
`/api/<data_source>/workspace`	returns status code `404` when there does not exist a workspace with the provided name	passed
`/api/<data_source>/workspace`	returns status code `200` when there exists a workspace with the provided name	passed
`/api/<data_source>/create`	returns status code `200` with the provided name and create a workspace folder with this name	passed
`/api/<data_source>/create`	returns status code `400` with an error message when there already exists a workspace with the provided name	passed
`/api/<data_source>/create`	returns status code `500`with an error message when the configuration does not exist	passed
`/api/<data_source>/remove`	returns status code `200` with the provided name and deletes the `workspace.json` file inside the folder with this name	passed
`/api/<data_source>/remove`	returns status code `500`with an error message when the configuration does not exist	passed
`/api/<data_source>/delete`	returns status code `200` with the provided name and deletes the folder with this name	passed
`/api/<data_source>/delete`	returns status code `400` with an error message when there does not exist a workspace with the provided name	passed
`/api/<data_source>/upload/<file>/<byte>`	returns status code `200` with a validation message and creates a file	passed

Continued on next page

Route	Test description	Result
/api/<data_source>/upload/<file>/<byte>	returns status code 500 with an error message when the configuration does not exist	passed
/api/<data_source>/data/convert	returns status code 200 with a validation message	passed
/api/<data_source>/data/convert	returns status code 500 when there does not exist a workspace with the provided name	passed
/api/<data_source>/data/bin_raw	returns status code 200 with a validation message	passed
/api/<data_source>/bin_raw	returns status code 200 with a validation message when the raw data is already binned	passed
/api/<data_source>/bin_raw	returns status code 500 with an error message when there does not exist a workspace with the provided name	passed
/api/<data_source>/get_offset	returns the correct offset from the rpl file	passed
/api/<data_source>/get_offset	returns 0 when there does not exist a workspaces with the provided name	passed
/api/<data_source>/element_averages	returns 0 when there does not exist a workspace with the provided name	passed
/api/<data_source>/element_averages	returns an object of the correct length	passed
/api/<data_source>/element_averages_selection	returns status code 400 with an error message when the provided selection is invalid	passed
/api/<data_source>/element_averages_selection	returns status code 400 with an error message when the type of the provided selection is invalid	passed
/api/<data_source>/element_averages_selection	returns status code 400 with an error message when the points of the provided selection are not in a list	passed
/api/<data_source>/element_averages_selection	returns an object of the correct length when the provided selection is valid	passed
/api/<data_source>/data/element/names	returns the correct element names	passed
/api/<data_source>/image/<image_name>	returns status code 200 and non-None data	passed
/api/<data_source>/image/<image_name>	returns status code 404 with an error message when the provided workspace does not contain a file with the provided image name	passed
/api/<data_source>/image/<image_name>/size	returns status code 200 with the correct image size	passed

Continued on next page

Route	Test description	Result
/api/<data_source>/image/<image_name>/size	returns status code 404 with an error message when the provided workspace does not contain a file with the provided image name	passed
/api/<data_source>/image/<image_name>/recipe	returns status code 200 with the correct Recipe	passed
/api/<data_source>/image/<image_name>/recipe	returns status code 404 with an error message when the provided workspace does not contain a file with the provided image name	passed
/api/<data_source>/data/size	returns status code 200 with the correct cube size	passed
/api/<data_source>/data/recipe	returns a non-empty recipe	passed
/api/<data_source>/data/size	returns status code 200 with the correct cube size	passed
/api/<data_source>/data/recipe	returns status code 404 with an error message when there does not exist a workspace with the provided name	passed
/api/<data_source>/data/elements/map/<element_number>	returns status code 200 and non-None data	passed
/api/<data_source>/data/elements/map/<element_number>	returns status code 404 with an error message when there does not exist a workspace with the provided name	passed
/api/<data_source>/get_average_data	returns status code 200 and the correct number of points	passed
/api/<data_source>/get_average_data	returns status code 404 with an error message when there does not exist a workspace with the provided name	passed
/api/<data_source>/get_element_spectrum/<element>/<excitation>	returns status code 200 and the correct number of points and peaks	passed
/api/<data_source>/get_element_spectrum/<element>/<excitation>	returns status code 404 with an error message when there does not exist a workspace with the provided name	passed
/api/<data_source>/get_selection_spectrum	returns status code 400 with an error message when the type of the provided selection is not valid	passed
/api/<data_source>/get_selection_spectrum	returns status code 400 with an error message when the provided selection has no type	passed
/api/<data_source>/get_selection_spectrum	returns status code 400 with an error message when the points of the provided selection are not valid	passed
/api/<data_source>/get_selection_spectrum	returns status code 200 with the correct number of points when provided valid arguments	passed

Continued on next page

Route	Test description	Result
/api/<data_source>/cs/clusters/<element_number>/<k>/<threshold>	returns status code 200 and a non None object when provided a a threshold of 100	passed
/api/<data_source>/cs/clusters/<element_number>/<k>/<threshold>	returns status code 200 and a non None object when provided a a threshold of 0	passed
/api/<data_source>/cs/clusters/<element_number>/<k>/<threshold>	returns status code 200 and the same clusters as its previous calls	passed
/api/<data_source>/cs/bitmask/<element_number>/<k>/<threshold>	returns status code 200 and a non None object when provided a a threshold of 100	passed
/api/<data_source>/cs/bitmask/<element_number>/<k>/<threshold>	returns status code 200 and a non None object when provided a a threshold of 0	passed
/api/<data_source>/cs/bitmask/<element_number>/<k>/<threshold>	returns status code 500 with and error message when the configuration does not exist	passed
/api/<data_source>/dr/embedding/<element_number/<threshold>	returns status code 500 with and error message when there does not exist a workspace with the provided name	passed
/api/<data_source>/dr/overlay/<element_number>/<threshold>	returns status code 400 with and error message when there does not exist a workspace with the provided name	passed
/api/<data_source>/dr/embedding/mapping	returns status code 400 with and error message when there does not exist a workspace with the provided name	passed

Color segmentation

Function	Test description	Result
get_cluster_using_k_means	extracts the inital color clusters	passed
get_cluster_using_k_means	logs an error and returns an empty array when the provided workspace does not exist	passed
merge_similar_colors	merges similar colors	passed
merge_similar_colors	logs an error and returns an empty array when the provided colors and bitmasks are empty	passed
combine_bitmasks	merges several bitmasks	passed
combine_bitmasks	returns empty when the provided bitmasks are empty	passed
get_elemental_clusters_using_k_means	extracts color clusters of the RGB image per element	passed

Continued on next page

Function	Test description	Result
`get_elemental_cluster_using_k_means`	logs an error and returns empty arrays when the provided workspace does not exist	passed
`get_elemental_cluster_using_k_means`	logs an error and returns empty arrays when the provided image does not exist	passed
`merge_similar_colors`	logs an error and returns an empty array when the provided bitmasks are empty	passed
`image_to_lab` and `image_to_rgb`	convert a LAB image to the correct RGB color and vice versa	passed
`rgb_to_lab` and `lab_to_rgb`	convert an RGB color to its correct LAB equivalent and vice versa	passed
`convert_to_hex`	converts RGB colors to their correct hexadecimal equivalent	passed
`save_bitmask_as_png`	returns `true` when provided a correct bitmask and path	passed
`save_bitmask_as_png`	returns `false` when the provided path does not exist	passed
`save_bitmask_as_png`	returns `false` when the provided bitmask is empty	passed
`get_path_to_cs_folder`	returns the correct path to the generated color segmentation folder	passed
`get_path_to_cs_folder`	logs an error and returns `None` when the configuration contains an invalid workspace	passed

Contextual images

Function	Test description	Result
`get_contextual_image_path`	returns the path to the base image	passed
`get_contextual_image_path`	returns the path to a contextual image	passed
`get_contextual_image_path`	logs an error and returns `None` when the specified path does not exist	passed
`get_contextual_image_path`	logs an error and returns `None` when the specified config does not exist	passed
`get_contextual_image_path`	logs an error and returns `None` when the specified painting does not exist	passed
`get_contextual_recipe_path`	returns the path to the base image Recipe	passed
`get_contextual_recipe_path`	returns the path to the a contextual image Recipe	passed

Continued on next page

Function	Test description	Result
get_contextual_recipe_path	logs an error and returns None when the specified path does not exist	passed
get_contextual_image	returns the image at the specified path	passed
get_contextual_image	logs an error and returns None when the specified path leads to an invalid image	passed
get_contextual_image	logs an error and returns None when the specified path does not exist	passed
get_contextual_image_size	returns the size of the image at the given path	passed
get_contextual_image_size	returns None when the specified path leads to an invalid image	passed
get_contextual_image_size	returns None when the specified path does not exist	passed
get_path_to_base_image	returns the correct path to the base image	passed

Dimensionality reduction

Function	Test description	Result
generate_embedding and create_embedding_image	log an error and return error when the configuration does not exist	passed
generate_embedding	logs an error and returns error when the provided element is invalid	passed
generate_embedding	logs an error and returns error when the provided elemental cube is invalid	passed
generate_embedding	logs an error when the umap arguments are invalid	passed
generate_embedding	generates the embedding files when given the default threshold	passed
generate_embedding	generates the embedding files when given a threshold of 0	passed
generate_embedding	logs an error when the provided threshold is too high	passed
create_embedding_image	creates an embedding file when given a valid elemental overlay	passed
create_embedding_image	creates an embedding file when given a valid RGB image overlay	passed
create_embedding_image	logs an error, returns None and does not create a file when no embedding is specified in the configuration	passed

Continued on next page

Function	Test description	Result
create_embedding_image	logs an error, returns None and does not create a file when an invalid element is provided	passed
create_embedding_image	logs an error, returns None and does not create a file when the provided overlay type is invalid	passed
create_embedding_image	logs an error, returns None and does not create a file when the provided contextual image is invalid	passed
get_image_of_indices_to_embedding	returns None when the provided workspace does not exist	passed
get_image_of_indices_to_embedding	logs an error and returns None when the provided workspace does not contain an image	passed
create_image_of_indices_to_embedding	returns None when the provided workspace does not exist	passed
create_image_of_indices_to_embedding	logs an error and returns None when the provided workspace does not contain an elemental cube	passed
create_image_of_indices_to_embedding	returns None when the provided workspace does not contain the generated files	passed
create_image_of_indices_to_embedding	creates an image with the selected points in the embedding when the provided arguments are valid	passed
create_image_of_indices_to_embedding	logs an error and returns None when the provided embedding contains nan values	passed

Elemental data processing

Function	Test description	Result
get_element_names	returns the name of the elements in the elemental cube	passed
get_element_names	logs an error and returns None when the provided workspaces does not exist	passed
get_short_element_names	logs an error and returns None when the provided data workspaces does not exist	passed
get_elemental_map	returns the elemental map of an element from the elemental cube	passed
get_elemental_map	logs and error and returns an empty array when the provided element type is invalid	passed

Continued on next page

Function	Test description	Result
`get_element_averages`	returns the correct average par element of the elemental cube	passed
`get_element_averages`	logs an error and returns `None` provided workspaces does not exist	passed
`convert_elemental_cube_to_dms`	correctly converts a .cvs elemental cube to a .dms one	passed
`convert_elemental_cube_to_dms`	logs an error and returns `None` for an invalid folder path	passed
`convert_elemental_cube_to_dms`	logs an error and returns `None` for an cube name including a file extension	passed
`get_element_averages_selection`	returns an average for each element	passed
`get_elemental_data_cube`	returns the elemental cube as an array	passed

File upload

Function	Test description	Result
`get_data_sources_names`	returns the list of uploaded workspaces	passed
`get_data_sources_names`	returns an empty list when no workspaces have been uploaded	passed
`get_data_sources_names`	returns an empty list when an invalid config is provided	passed
`get_data_sources_files`	returns the correct files in a workspaces	passed
`get_data_sources_files`	returns an empty list when a non-existent workspace name is provided	passed
`get_data_sources_files`	returns an empty list when an invalid config is provided	passed

Image registration

Function	Test description	Result
`register_image_to_image`	logs an error and returns `false` when the provided path to the reference image does not exist	passed
`register_image_to_image`	logs an error and returns `false` when the provided path to the registered image does not exist	passed
`register_image_to_image`	logs an error and returns `false` when the provided path to the recipe does not exist	passed
`register_image_to_image`	logs an error and returns `false` when the provided path to the destination does not exist	passed
`register_image_to_image`	successfully registers the image when the provided parameters are valid	passed
`get_image_registered__to_data_cube`	logs an error and returns `None` when the specified workspace is invalid	passed
`get_image_registered__to_data_cube`	logs an error and returns `None` when the specified image name is invalid	passed
`get_image_registered__to_data_cube`	successfully returns the registered image when the provided parameters are valid	passed

Selection

Function	Test description	Result
get_selection	logs an error and returns None when the specified workspace is invalid	passed
get_selection	returns an array None when the specified workspace is valid	passed
get_scaled_cube_coordinates	returns the correct scaled coordinates in the data cube	passed
get_selection	returns the correct number of selected pixels in the data cube	passed
get_selection	returns the same data when the same area is selected through rectangle or lasso selection	passed
get_selection	returns the correct data when some selection coordinates are negative	passed
get_selection	returns the correct data when the polygon selection crosses itself	passed
get_selection	logs an error and returns None when the provided rectangle selection does not have 2 points	passed
get_selection	logs an error and returns None when the provided polygon selection has only 2 points	passed
get_selection	logs an error and returns None when the provided cube type is invalid	passed
get_selection	returns an array when the provided arguments are invalid	passed
deregister_coord	returns the correct data cube coordinates corresponding to base image coordinates	passed

Spectral data handling

Function	Test description	Result
parse_rpl	returns the content of the rpl file	passed
parse_rpl	logs an error and returns an empty dictionary when the rpl file is empty	passed
parse_rpl	logs an error and returns an empty dictionary when the rpl file does not exist	passed
get_spectra_params	returns the correct low, high and bin_size of a workspace	passed
get_spectra_params	raises an error when the workspace does not exist	passed

Continued on next page

Function	Test description	Result
update_bin_params	correctly converts and updates the binning parameters in a workspace for default parameters	passed
update_bin_params	correctly converts and updates the binning parameters in a workspace for custom parameters	passed
update_bin_params	correctly converts and updates the binning parameters in a workspace when there is no offset in the rpl file	passed
bin_data	does not modify the spectral data when provided low=0, high=channel_number and bin_size=1	passed
bin_data	correctly computes the bins and modifies the spectral data when given custom parameters	passed
bin_data	does not modify the spectral data when given default parameters	passed
bin_data	logs an error when the rpl file of the workspace is empty	passed
bin_data	logs an error when the spectral data size does not correspond to the parameters	passed
bin_data	raises an error when there is no raw file in the workspace	passed

Spectral data processing

Function	Test description	Result
get_average_global	returns the correct average of the entire spectral data	passed
get_data_selection	returns the correct average of the selected spectral data	passed
get_data_selection	logs an error and returns an empty array when the configuration does not exist	passed
get_raw_data	returns the correct spectral data	passed
mipmap_exists	returns false when the mipmap does not exist	passed
mipmap_exists	returns false when the workspace does not exist	passed
mipmap_exists	returns true when the provided level is 0	passed
mipmap_exists	returns true when a mipmap has been created	passed

Continued on next page

Function	Test description	Result
get_raw_data	creates a mipmap of the correct size when it is called with a non-zero mipmap level	passed
get_theoretical_data	returns a list of the correct length	passed
get_theoretical_data	logs an error and returns None when the provided element is not valid	passed

Workspace handling

Function	Test description	Result
get_path_to_workspace	logs an error and returns None when the configuration does not exist	passed
update_workspace	logs an error and returns None when the configuration does not exist	passed
get_path_to_workspace	logs an error and returns None when the provided workspace does not exist	passed
update_workspace	logs an error and returns None when the provided workspace does not exist	passed
get_path_to_workspace	returns the correct path to the workspace.json file when the provided workspace exists	passed
update_workspace	returns True and correctly updates the workspace.json file when the provided workspace exists	passed
get_workspace_dict	logs an error and returns None when the configuration does not exists	passed
get_spectral_cube_recipe_path	logs an error and returns None when the configuration does not exists	passed
get_spectral_cube_recipe_path	returns None when the workspace does not contains a workspace.json file	passed
get_base_image_name	logs an error and returns None when the configuration does not exists	passed
get_base_image_name	returns None when the workspace does not contain a workspace.json file	passed
get_base_image_name	returns the correct name of the base image of the provided workspace	passed
get_base_image_path	logs an error and returns None when the configuration does not exist	passed
get_base_image_path	returns None when the workspace does not contain a workspace.json file	passed
get_elemental_cube_path_from_name	logs an error and returns None when the configuration does not exist	passed

Continued on next page

Function	Test description	Result
get_elemental_cube_path_from_name	logs an error and returns None when the workspace does not contain an elemental cube	passed

Front-end library utils

Function	Test description	Result
hexToRgb	converts random hexadecimal colors to their correct RGB equivalent	passed
rgbToHex	converts random RGB colors to their correct hexadecimal equivalent	passed
hexToRgb	converts black and white hexadecimal colors to their correct RGB equivalent	passed
rgbToHex	converts black and white RGB colors to their correct hexadecimal equivalent	passed
hexToRgb	converts red, green and blue hexadecimal colors to their correct RGB equivalent	passed
rgbToHex	converts red, green and blue RGB color to their correct hexadecimal equivalent	passed
sortRectanglePoints	sorts the two points defining a rectangle in the correct order	passed
remToPx	converts random numbers in rem to their correct equivalent in pixels	passed
pxToRem	converts random numbers in pixels to their correct equivalent in rem	passed
remToPx	converts 0 rem to the correct equivalent in pixels	passed
pxToRem	converts 0 pixels to the correct equivalent in rem	passed
remToPx	converts a negative number in rem to its correct equivalent in pixels	passed
pxToRem	converts a negative number in pixels to its correct equivalent in rem	passed
remToPx	converts a large number in rem to its correct equivalent in pixels	passed
pxToRem	converts a large number in pixels to its correct equivalent in rem	passed
cn	combines a text color with a background color into a single class	passed
cn	combines a font type with a text size into a single class	passed

Continued on next page

Function	Test description	Result
`cn`	combines a background color with a text size into a single class	passed
`deepClone`	creates a deep clone of an object	passed
`deepClone`	creates a deep clone of a nested object	passed
`deepClone`	creates a deep clone of an array	passed
`deepClone`	creates a deep clone of an empty object	passed
`deepClone`	creates an object such that modifying it does not modify the original object	passed
`flipSelectionAreaSelection`	correctly flips a polygon selection vertically	passed
`flipSelectionAreaSelection`	correctly flips a rectangle selection vertically	passed
`flipSelectionAreaSelection`	correctly flips an empty selection vertically	passed

Front-end workspace utils

Function	Test description	Result
validate_workspace	returns true with no message when provided a correct workspace	passed
validate_workspace	returns false with the correct message when provided an empty workspace	passed
validate_workspace	returns false with the correct message when provided a workspace with just a base image name	passed
validate_workspace	returns false with the correct message when provided a workspace with just a base image name and location	passed
validate_workspace	returns false with the correct message when provided a workspace with a base image name, location and contextual image name	passed
validate_workspace	returns false with the correct message when provided a workspace with just a base image name, location and contextual image name and location	passed
validate_workspace	returns false with the correct message when provided a workspace with a base image name and location, a contextual image name and location, and a spectral cube name	passed
validate_workspace	returns false with the correct message when provided a workspace with a base image name and location, a contextual image name and location, and a spectral cube name and location	passed
validate_workspace	returns false with the correct message when provided a workspace with a base image name and location, a contextual image name and location, a spectral cube name, location and rpl file	passed
validate_workspace	returns false with the correct message when provided a workspace with a base image name and location, a contextual image name and location, a spectral cube name, location, rpl file and Recipe	passed
validate_workspace	returns false with the correct message when provided a workspace with a base image name and location, a contextual image name and location, a spectral cube name, location and rpl file, and an elemental cube name	passed
validate_workspace	returns false with the correct message when provided a workspace with a base image name and location, a contextual image name and location, a spectral cube name, location, rpl file and Recipe, and an elemental cube name and location	passed
validate_workspace	returns true with no message when provided a workspace with a base image name and location, a contextual image name and location, a spectral cube name, location and rpl file, and an elemental cube name, location and Recipe	passed
validate_workspace	returns false with the correct error message when provided a workspace with duplicate file names	passed

9.2.2 Coverage report

In this section the test coverage for all files is listed. Explanations have been included for those files with less than 100% coverage.

Coverage report

Module	Coverage	Uncovered lines
Back-end	**96%**	
xrf_explorer/__init__.py	88%	15
xrf_explorer/server/__init__.py	100%	
xrf_explorer/server/color_segmentation/__init__.py	100%	
xrf_explorer/server/color_segmentation/color_seg.py	99%	257-258
xrf_explorer/server/color_segmentation/helper.py	95%	28
xrf_explorer/server/dim_reduction/__init__.py	100%	
xrf_explorer/server/dim_reduction/embedding.py	100%	
xrf_explorer/server/dim_reduction/general.py	100%	
xrf_explorer/server/dim_reduction/overlay.py	100%	
xrf_explorer/server/file_system/__init__.py	100%	
xrf_explorer/server/file_system/cubes/__init__.py	100%	
xrf_explorer/server/file_system/cubes/convert_csv.py	100%	
xrf_explorer/server/file_system/cubes/convert_dms.py	92%	86-87, 156-158
xrf_explorer/server/file_system/cubes/elemental.py	83%	94, 96-98, 127-129, 158-159, 161-163, 190, 192, 275, 291-292, 296, 302, 307, 322-324
xrf_explorer/server/file_system/cubes/spectral.py	90%	94, 97, 106, 113, 155, 161, 168-169, 171, 181-183, 260-261
xrfexplorer/server/file_system/helper.py	94%	52-53
xrf_explorer/server/file_system/sources/__init__.py	100%	
xrf_explorer/server/file_system/sources/data_listing.py	100%	
xrf_explorer/server/file_system/workspace/__init__.py	100%	
xrf_explorer/server/file_system/workspace/contextual_images.py	87%	74-75, 104-105, 156-157, 162-163, 165, 179, 198
xrf_explorer/server/file_system/workspace/file_access.py	92%	88, 102-103, 130-131, 139, 153-154, 176, 180

Continued on next page

Module	Coverage	Uncovered lines
xrf_explorer/server/file_system/workspace/workspace_handler.py	91%	68-70
xrf_explorer/server/image_register/__init__.py	100%	
xrf_explorer/server/image_register/register_image.py	85%	149-150, 159, 222, 374-375, 380, 385-387, 390, 393-395, 398-400, 402, 405, 407, 414
xrf_explorer/server/image_to_cube_selection/__init__.py	100%	
xrf_explorer/server/image_to_cube_selection/image_to_cube_selection.py	98%	242-243
xrf_explorer/server/routes./__init__.py	100%	
xrf_explorer/server/routes./color_segmentation.py	93%	43, 47, 94, 124
xrf_explorer/server/routes./dim_reduction.py	90%	33, 57, 77
xrf_explorer/server/routes./elemental_cube.py	93%	73, 140, 182-184
xrf_explorer/server/routes./general.py	100%	
xrf_explorer/server/routes./helper.py	94%	49-50
xrf_explorer/server/routes./images.py	93%	39, 71, 96
xrf_explorer/server/routes./project.py	93%	
xrf_explorer/server/routes./spectral_cube.py	91%	54-55, 112, 138-140
xrf_explorer/server/spectra/__init__.py	100%	
xrf_explorer/server/spectra/spectra.py	99%	227
Front-end	**81%**	
xrf_explorer/client/src/lib/utils.ts	96%	34-36
xrf_explorer/client/src/components/workspace/utils.ts	65%	31-46

The files that do not achieve 100% test coverage are listed below, together with explanations.

- **Main __init__:** The only line not covered was un-testable.
- **Color segmentation and color segmentation helper:** The few lines of code that are not covered by tests are error handling lines for errors coming from external libraries (such as `numpy` or OpenCV), which we cannot force and therefore cannot test.
- **Cubes files:** The lines which are not covered either test for general exception throws or are safety checks which cannot theoretically be reached as they are already covered in other functions called within the functions themselves.
- **Contextual images:** The lines not covered by unit tests are double checks for the absence of config files. It is not possible to reach these checks, as they are done by a prior, parent, function. Theoretically speaking these checks are redundant for any child functions, but they are present as an extra safety net. All cases of the parent function are fully checked.
- **Workspace files:** The parts which are not tested are edge cases for when the input is `None`. Reaching the returns of these checks was very difficult in a test environment and thus priority went to assuring that all check cases were being covered.

- **Register:** The lines not covered by the unit test are part of a very special edge case. For this edge case, a base image should not be present while still having a valid workspace. This scenario is already tested extensively in the front-end under `workspace/utils.ts`. To reach this edge case in the back-end, the front-end checks would first have to be bypassed by the user. Simulating this edge case in the back-end proved very difficult and near impossible.
- **Image to cube selection:** The two lines which are not tested are a safety check for a hard-to-reach edge case.
- **Routes tests:** As the routes are the core of the communication between the back-end and front-end, extra attention was paid to ensuring these were tested properly. There, however, remained some edge cases which could not be tested properly. These are extra checks for potential server errors which cannot be simulated easily.
- **Spectra:** The only line not covered by the test cases is an edge case of a library which was unreachable through unit tests.
- **General front-end:** In general, the front-end has limited unit tests. There are two main reasons for this lack of unit tests. The first reason is that most functions are not exported, making them unreachable for unit testing. These functions could have been exported, but this would have been bad practice. All these functions are not required to be exported, as nearly all are contained within their respective window classes. Making those functions publicly accessible would have worsened the overall code quality. We thus prioritized clarity of the code base by ensuring a consistent function visibility style. The second reason is the heavy test coverage of the back-end. The front-end makes a lot of API requests but does very little computations of its own, for this, it relies on the back-end. We thus argue that by ensuring the back-end has proper test coverage, we can also be sure that the data and results in the front-end are correct. Furthermore, testing of the actual UI and whether it makes the correct calls is monitored through the Acceptance Test Plan [13].
  The file lib/utils does have extensive test coverage as this file holds all central helper functions for the front-end.
- **workspace/utils:** The lines not covered through unit tests make use of API calls and are tested in the ATP.
- **lib/utils:** The only lines not covered for this module are for one function to reload the page (which is tested through the ATP instead).