university of groningen

PASTAQ

Architecture Document

Software Engineering Project Feb 2022
Students: Tudor Dragan, Teresa Ferreira, Cristian Iacob, Mohammed Nacer Lazrak, Björn Schönrock, Dominic Therattil & Kaitlin Vos
First Supervisor: Lars Andringa
Second Supervisor: Dr A. Capiluppi

Contents

1 Abstract

The PASTAQ-GUI already provides a decent means of using the PASTAQ code, however, certain elements and functionalities of the GUI could be further improved in order to allow for better and more accessible usage of PASTAQ. The goal of our project is then to improve the GUI in order to cover the aforementioned issues such that they are fixed. This document serves to provide insight into the technologies and architecture of our project, along with the additions made by our team to the project and the means through which we maintained a high quality product. As far as our additions are concerned we further explain them by exploring their respective requirements. The primary targets of this document are researchers, data scientists, chemists, biologists and our client; however, readers with all levels of experience that have an interest in the PASTAQ-GUI may find the document useful in assessing our overall project.

2 Introduction

The system as a whole covers the functionalities needed to convert .raw data to .csv data with accompanying quality control plots as PNGs, the process can be observed in the figure below, although we are mostly focused on making the use of the PASTAQ pipeline more accesible.

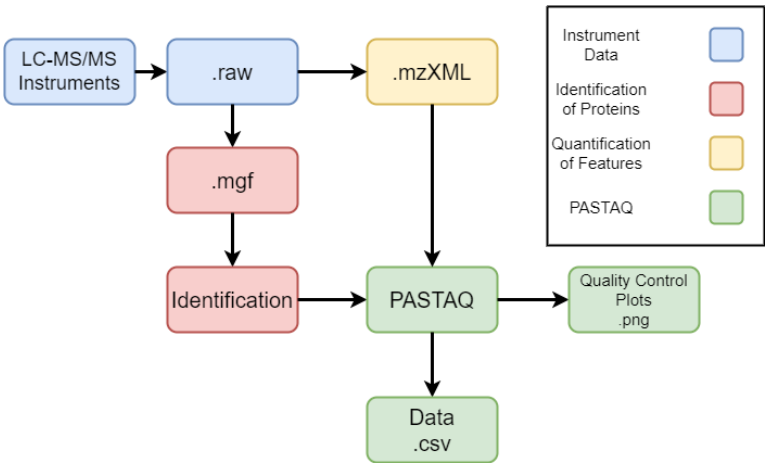

Figure 1: Overview of the ecosystem where PASTAQ is used

Our objective is the rework of the GUI into a stable GUI with a new look and added tooltips, which is to be used on popular operating systems such as Windows, Linux and macOS. Additionally, the PASTAQ code will be refactored to get rid of code smells, the PASTAQ-GUI will be made such that .mgf files are automatically converted to .mzID and the PASTAQ-GUI will perform a MSFragger database search using the given parameters without having to open any alternative program.

Throughout this document we will be discussing the measures taken to ensure a high quality for the final product, along with the way our changes were implemented and the way they affected the final product.

3 Technology Stack

3.0.1 Languages

- Python: the language of our PASTAQ-GUI

3.0.2 Technologies

Tools: Currently, there will only be one option to process files from .mgf to .mzID file, which includes MSFragger and idconvert. For different identification processes, different tools can be used.

- MSFragger: an ultrafast database search tool for peptide identification in mass spectrometry-based proteomics
- idconvert: a command line tool for converting between various file formats (pepXML, protXML, mzIdentML)

Libraries:

- doctest: a module that allows the easy generation of tests based on output from the standard Python interpreter
- eigen: a C++ template library for linear algebra
- pybind11: a lightweight header-only library that exposes C++ types in Python and vice versa
- zlibstatic: a software library used for data compression
- PyQt5: a comprehensive set of Python bindings for Qt5

4 Architectural Overview

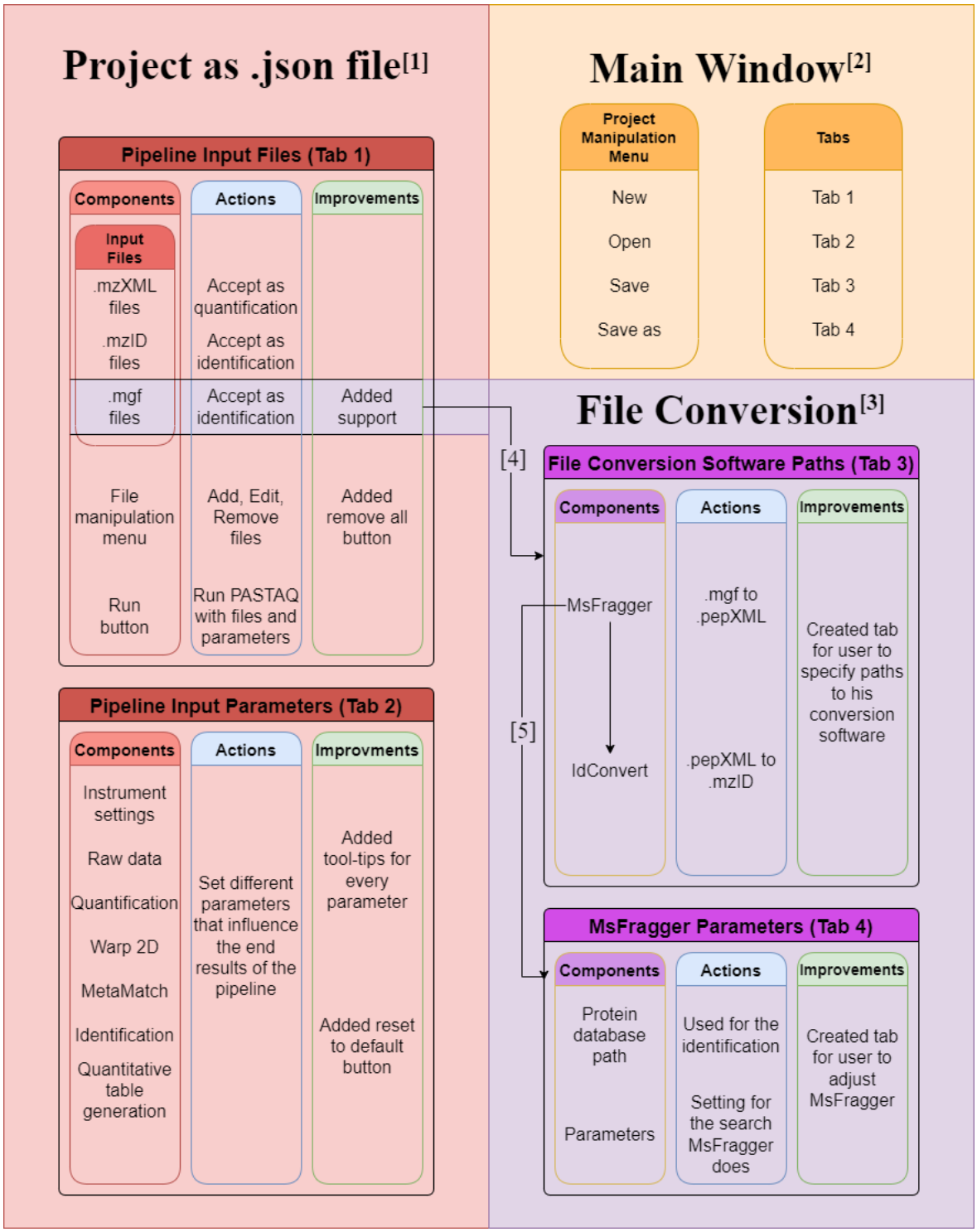

Figure 2: Overview of the Architecture

1. The user's input (files and parameters) is saved as a .json file. This represents the current project and is used to run the PASTAQ pipeline. This is dealt with by the first and second tabs.

2. The main window contains the menu that deals with opening and saving projects and all the tabs that are used for input.

3. The file conversion is dealt with by the third and fourth tab.

4. Only .mgf files need to be processed.

5. MSFragger requires its own parameters that need to be input by the user, otherwise the default values will be used.

5 File Procedure

5.1 File Acceptance

Since we want to allow the users to be able to upload processed identification files as well as files that still need processing, we need to allow the user to upload .mzID files and .mgf files. After the user makes/opens a project and selects the .mzML/.mzXML files, they will select the identification files. To further explain the upload options for the identification files the user has, we assume that the user has a directory with .mgf and .mzID files, as an example:

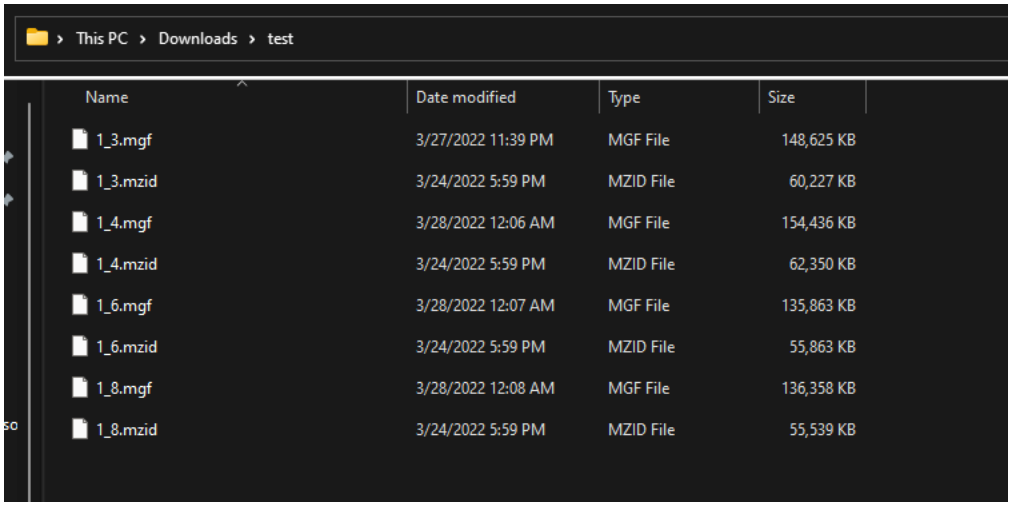

Figure 3: Directory with .mgf and .mzID files

When the user has made/opened a project in the GUI and added the .mzML/.mzXML files in the raw file column, they can select one or multiple rows to edit. Then the following window will open, already indicating to the user that they can upload either .mgf files or .mzID files:

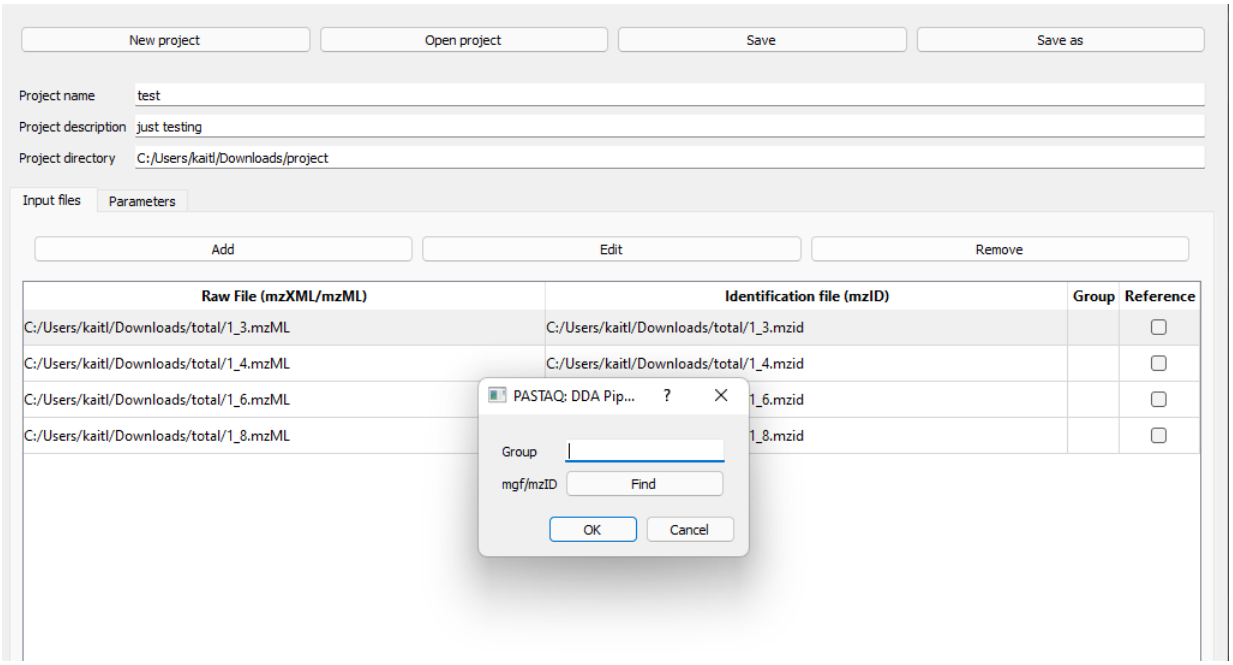

Figure 4: Edit window

The .mgf extension is included in the accepted file types for the identification files. This way, the user can upload either file type and it will look like the following:

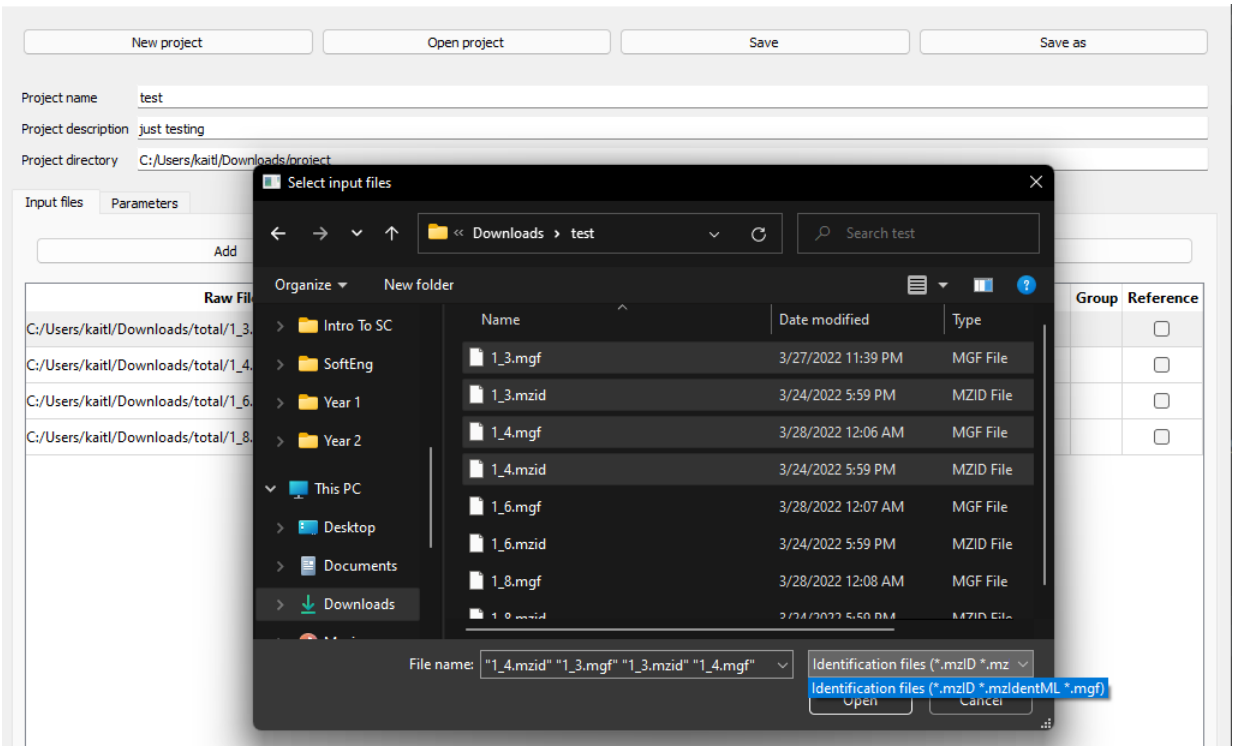

Figure 5: Updated file types

Alternatively, separate file extensions are also possible, to allow the user to choose which file type

they want to add as identification files and it would look like the following, for .mzID files, and .mgf files respectively,:

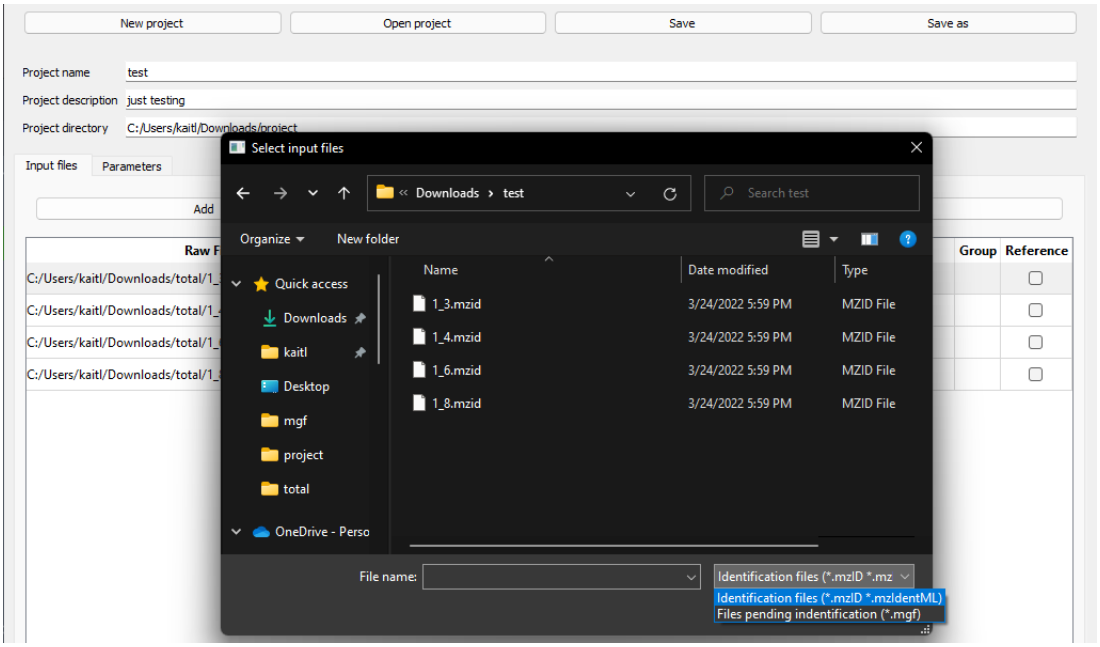

Figure 6: .mzID option

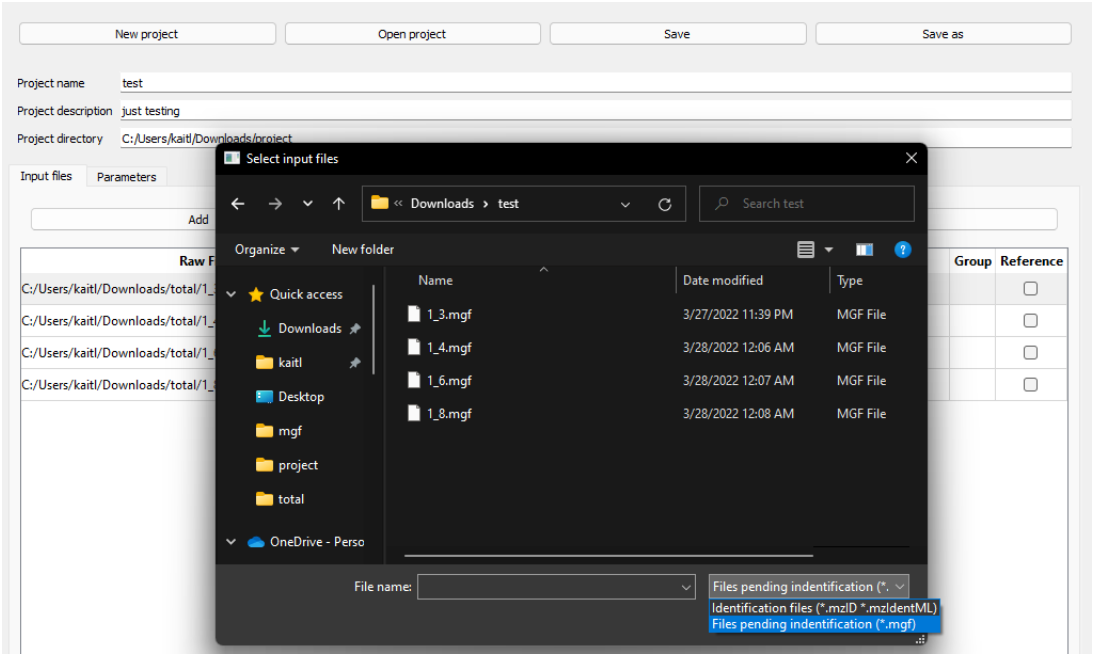

Figure 7: .mgf option

However, we prefer the first option, an inclusive file type, so that the user will not have to differentiate between the two options. They will be able to upload .mzID/.mgf files only or a mix of the two. This will ultimately increase user-friendliness since the user does not have to bother choosing between the

two file types or only being able to upload multiple files of the same format only.

To provide the user with the option to choose any of the three file formats, .mzML, .mzXLM and .mgf, and as a combination, we fulfill the requirements R1.1, R1.2 and NFR4.3.

5.2 Automated Identification Process

Now that the user is able to upload both .mgf files and .mzID files, we can start the automatic identification procedure.

After the user has selected their identification files, we differentiate between the two file formats. When the file is an .mzID, it gets added to the GUI in the usual manner, and the path to the file is saved in the .json file of the project. When the user has selected an .mgf file, before it gets added to the GUI, we process the file to an .mzID. For this procedure, we will use the technologies MSFragger and idconvert.

For the manual identification process, the user would have to run MSFragger [1] with their .mgf file. There are 4 ways of running MSFragger. To run it through the command line, the user will also need a .params file containing the parameters for the search. MSFragger requires the Java 8 Runtime Environment and a protein database in FASTA format. The path to the database is to be specified in the .params file. After the identification, MSFragger stores the result in a .pepXML file. The user will have to convert this .pepXML file into an .mzID file with the help of idconvert. This tool is only accessible through ProteoWizard [2].

The automatic processing will be done through commands as well. To allow this, the Java requirement has to be verified in order to be able to proceed with the actual identification process. If it is not fulfilled, the user will be notified with a pop-up window. The user is also asked to provide the GUI with the paths to the MSFragger .jar file and ProteoWizard "idconvert.exe" file and the FASTA format database. Entering the paths will be possible through an additional tab. The user will be met with a file dialog in the scenario where they want to browse. It is also possible for the user to paste the path, in case they already have the path ready to input. Once the user has chosen their path, the confirm button must be pressed. This button will evaluate whether the paths exists, has the required format and whether the path is accessible. Each path is evaluated individually. When the confirmation has been completed, the paths will be saved in the project, so the user does not need to enter them again. This tab looks like the following:

[1] https://github.com/Nesvilab/MSFragger/wiki/Preparing-MSFraggerDownloading-MSFragger
[2] https://proteowizard.sourceforge.io/

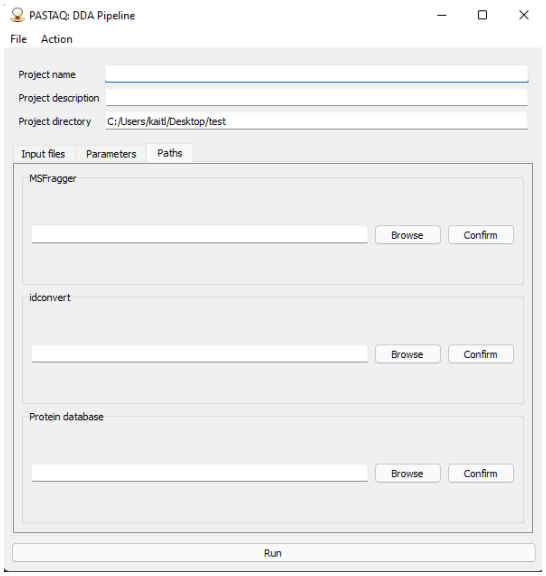

Figure 8: Paths tab

If the user enters an invalid path, they will be met with a message dialog which clarifies the error. The message dialog will appear and can only be closed manually. This will make it clear to the user that something went wrong. It will be impossible to oversee to avoid any confusions. For example, when the user enters a path to a .pdf in the MSFragger container which is clearly not a valid path for the MSFragger .jar file, it will look like the following:

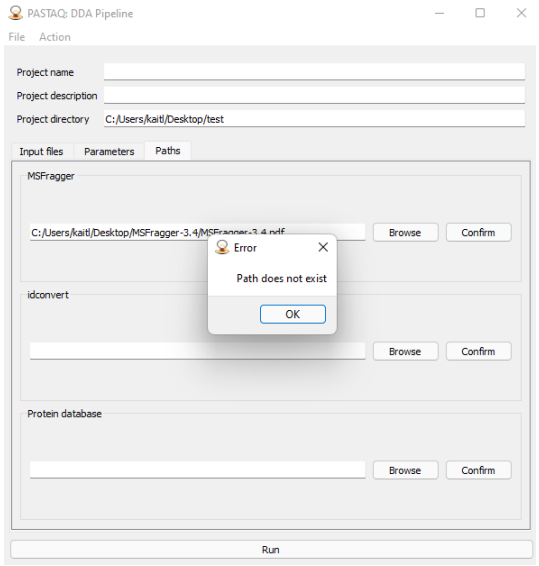

Figure 9: Path error

To increase the user friendliness, a tab with changeable parameters for MSFragger is provided, so the user does not need to construct the .params file themselves. After providing the GUI with all the necessary information, the only thing the user has to do is wait. The identification process, depending on the size of the file and the hardware components and specifications, might take a while.

The automatic processing will be executed in the same order as if it were done manually. Since there is no use for the .pepXML, we discard it once it becomes obsolete. The .mzID file will be placed in the same directory as the .mgf file.

Once the identification is done, the .mzID file will automatically be loaded into the GUI and the PASTAQ pipeline can be executed.
To file procedure of the two different file formats can be visually summarized, displaying the following:

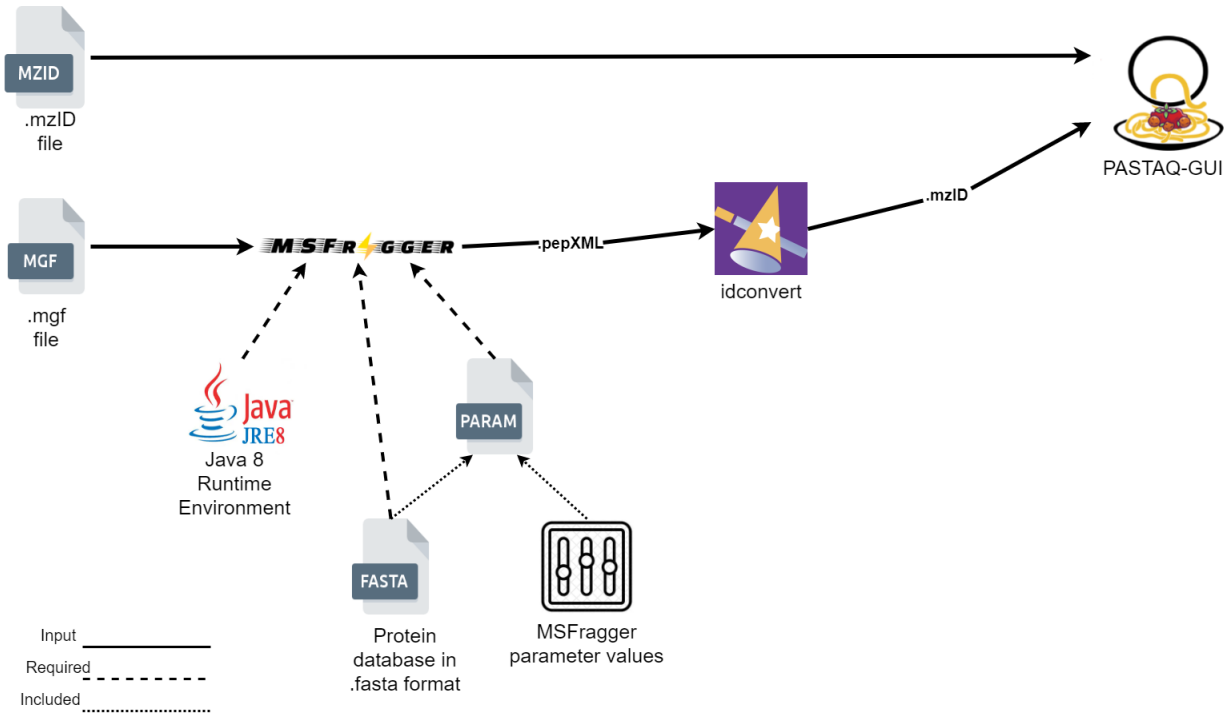

Figure 10: File procedure

Our automatic processing therefore fulfills R1.3 to R1.11, as well as NFR4.3 to a certain degree. The requirements for the identification are hard and requires knowledge to some extent, however, the process does not need to be done manually which takes some weight off from the user. Providing the user with an additional tab to set the identification parameters also contributes to this, since they will not have to create and fill out a .params file themselves.

Alternatively, we could ask the user to provide the desired values for the MSFragger parameter only. Then it would be our responsibility to verify whether the requirements have been fulfilled and locate the aforementioned directories and FASTA format database. The search for the directories and/or files usually take a while since they could be anywhere, in any sub-directory. This would require us to check all paths until we have found targets. If the technologies are not even present, this would all have been wasted time. Since the identification process already takes up a lot of time, we favored our original plan, to ask the user to provide the paths. Additionally, asking for the user to specify the paths also informs the user of the requirements for the identification process. After having identified these complications, we are confindentthat our choice serves multiple purposes.

Another option would be to not ask for the desired parameters. Under this link [3], multiple example .params files have been uploaded by the same lab that also released MSFragger. These could be

[3] https://github.com/Nesvilab/MSFragger/tree/master/parameter$_files$

considered "default" .params files. We could ask the user to choose the search they want executed and use the correct default file accordingly. The .params file contains the path to the FASTA format database, which will then be changed accordingly. However, since our main audience are researchers, we think it is not likely for them to be satisfied by default values because they might not align with their goals. We want the user to be able to have an identification process that would satisfy all their needs in their research, which our original plan allows.

6 Visual Design

6.1 Tooltips

The parameters tab in the GUI includes a lot of parameters that can be set to the user's desired values. These values can be set through drop-down lists, checkboxes, spin boxes and many more. Since not everyone might be familiar with the parameters, the terminology used or may know the parameter with a different name, we included tooltips. By implementing this, we have fulfilled R1.8. They contain the description of the parameters.
When the user navigates to the parameters tab, they will see the following:

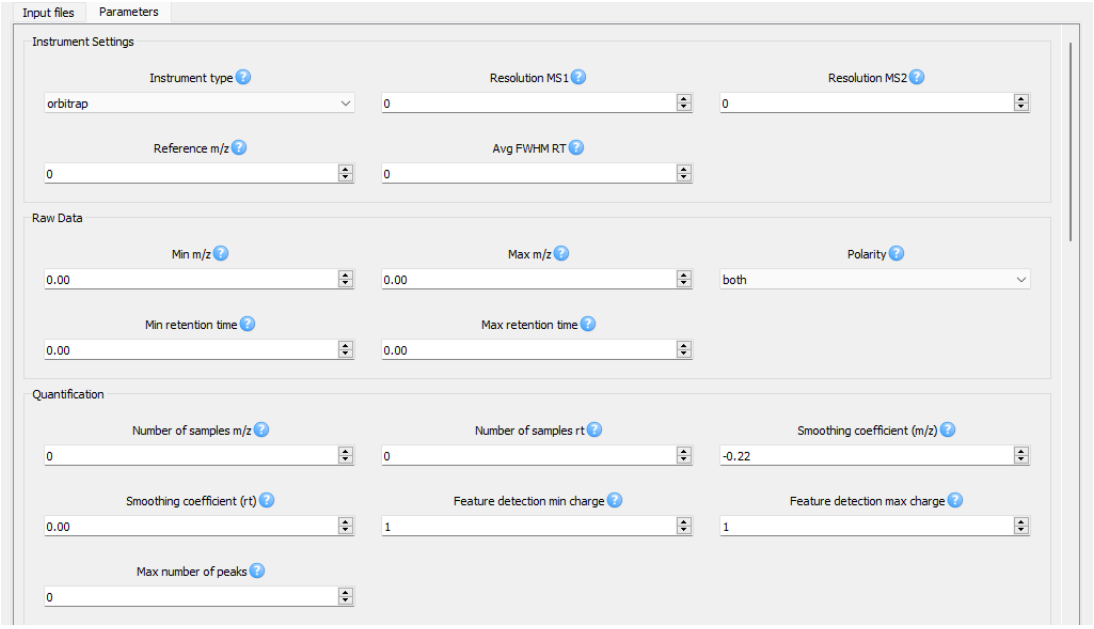

Figure 11: Parameter Tab

We included the infamous question mark icon to make it abundantly clear that all the parameters contain tooltips.
The parameter label includes the parameter name and the question mark icon. When the user hovers over the parameter label with their mouse, the tooltip will appear which will look like this:

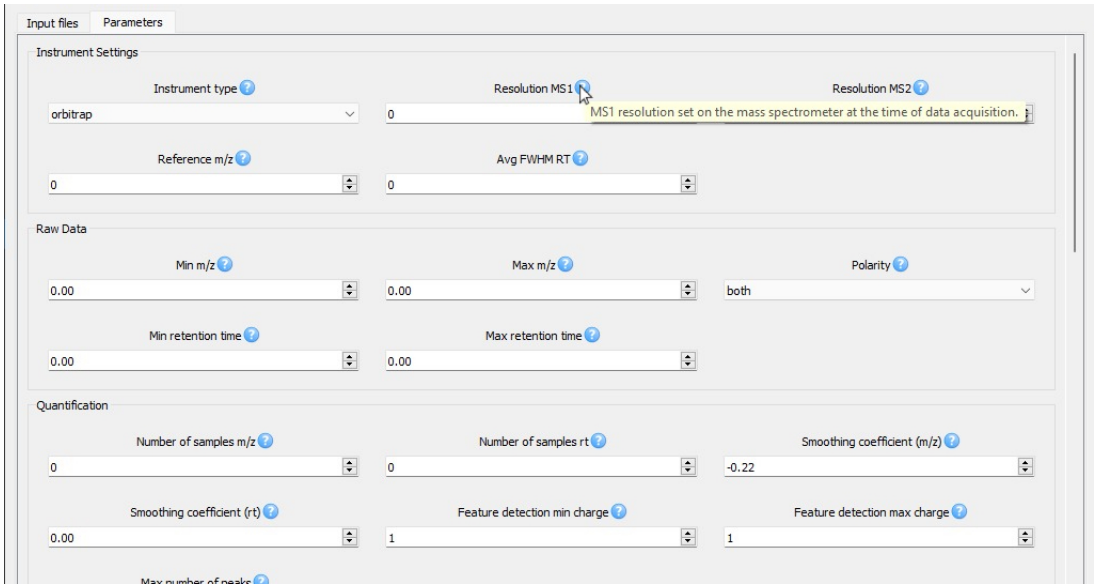

Figure 12: Tooltip

This tab allows users to scroll down. Since there is a tooltip for each parameter, each label adheres to this design shown in the previous figures. This allows for consistency within the same tab.
Including tooltips that are identifiable by the most common tooltip question mark icon makes the GUI more user-friendly, since it explains what each parameter means. Based on the tooltips, the user can also look up more elaborated information if needed. This way, our added feature contributes to another requirement, namely NFR4.1.

6.2 GUI Rework

The PASTAQ-GUI has been upgraded significantly. All the changes contribute to the user experience. Some features allow the user to complete certain tasks in a more time efficient manner and have been implemented for the user's convenience. Some features can be considered corrections to the original design and some features have been implemented to give the user more control. We have divided the features into three categories: the purely visible changes, the visual errors we have corrected and the functional features that have been implemented for the user's convenience.
The current GUI looks like the following:

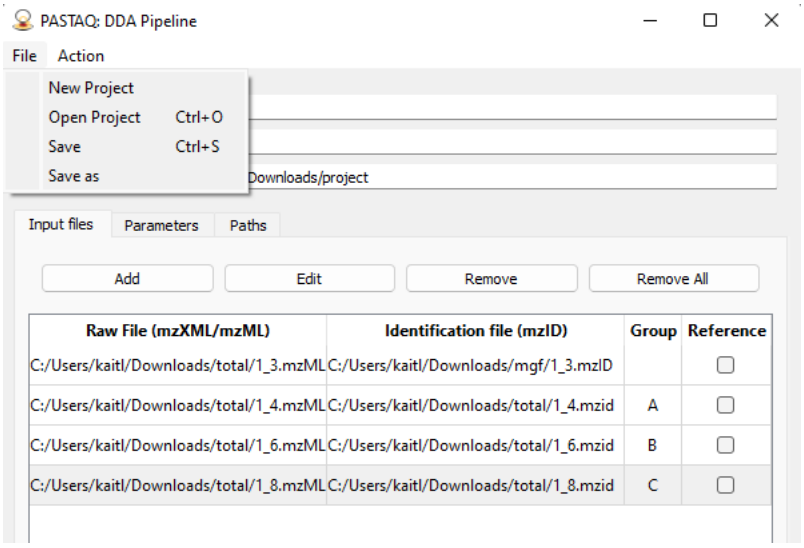

Figure 13: PASTAQ-GUI

Visual changes:

Icon
An icon has been added to the GUI to represent the PASTAQ. The icon has been added to make the GUI more recognizable between all the other applications. The icon is visible in the top left corner of Figure 13. This feature satisfies NFR4.5.

Color palette
A more diverse color palette is integrated in the GUI to accentuate the buttons and the separate panels. This has been implemented to soothe the eyes and make it more pleasing to look at. Additionally, a dark mode option is offered to the user. This feature has been implemented by many software applications today because it causes less strain on the eyes when the user has low-light conditions. Thus, NFR4.6 and NFR4.7 are satisfied.

Progress bar
The speed of the PASTAQ pipeline depends solely on the amount of files, their sizes and the computational power. On a normal computer, running PASTAQ can take a significant amount of time. To inform the user on the progress being made and indicate how much more is still left, a progress bar is integrated. This allows the user to optimize their time management. This feature completes NFR4.8.

Dynamic size
To allow the user to organize their own workspace and give them as much control as possible, the GUI does not only allow the fullscreen option but the window itself is also be resizable. Additionally,

the parameters tab is resizable as well. These features cover NFR4.9, NFR4.10 and NFR4.11.

Visual errors:

Cancel button
While running the PASTAQ pipeline a new window is opened to display the PASTAQ log. This window also contains a cancel button, in case the user chooses to abort the process. However, after completion, the cancel button was still visible. This issue has now been resolved, therefore R1.13 has been completed.

Functional features:

Pop-up
When the user attempts to close the GUI, they will be met with a dialog window. The window offers three options to proceed with. The user can either save the project and close the GUI, discard any changes they might have made and close the GUI, or cancel the action and not leave the GUI. The GUI does not automatically save any changes. To provide the user with as much control over their project as possible, they are made aware of the current state and are asked to choose any of the three options. This also clarifies the consequence of the project when closing. This feature satisfies R 1.14 to R1.16.
The following figure shows what the pop-up looks like:

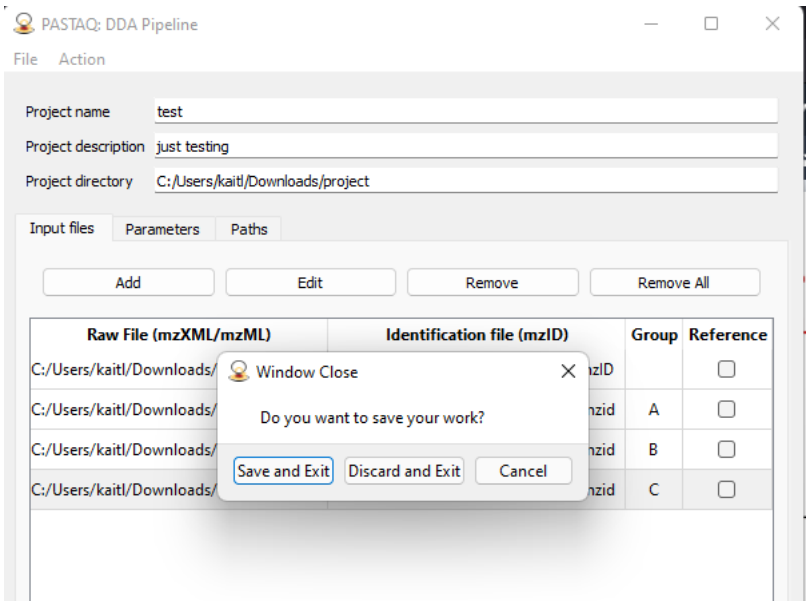

Figure 14: Pop-up

Buttons
The following buttons have been added to the GUI to allow the user to save time and/or effort:

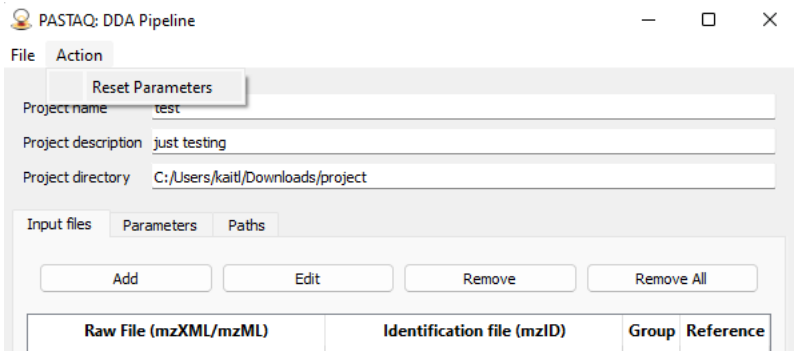

Figure 15: Reset parameters

- A button to reset parameters, in case the user desires to recover the default values, per R1.16 The button is listed in a drop-down navigation menu to allow more actions to be added in the future and not clutter the GUI.
- A button to select all files, which is of importance since a lot of files could be involved in running the pipeline that would take too much time to select manually, per R1.17
- A button to delete all the files, which is covered by the same reasoning and is covered by R1.19, The button has been placed in the row of buttons at the top of the "Input files" tab to stay consistent.

Drop-down navigation menu
The GUI now provides a drop-down navigation menu which allows the user to either create a new project, open an existing project, save a project or save the project under a specific name. These options are listed in the drop-down menu in the left corner. This feature has been implemented in many software applications and is, therefore, one of the most common ones. This satisfies requirement 1.17 and is shown in the top left corner of Figure 13.

Shortcuts
The standard shortcuts are integrated in the GUI. When the user wishes to save a project or open a project, the usual shortcuts are applicable. The shortcuts depend on the operating system but can still be considered universal shortcuts. The shortcuts are mentioned in R1.18, R1.20, R2.3 and R2.4. The shortcuts applicable for the Windows operating system are indicated by the keys to be pressed mentioned in the drop-down navigation menu, visible in the top-left corner of Figure 13.

7 Quality Assurance

As we work on our objectives, we are going to follow the developmental structure outlined below in order to ensure high quality for our final program, this structure relies on manual testing done by developers and automatic testing through continuous integration using GitHub Actions.

Figure 16: QA Diagram

7.1 Supported Operating Systems

We decided to make the software usable and stable on all major operating systems, namely Windows, Linux and macOS. This is reflected in UC6 which we aptly named Operating system, or more specifically, in its requirement NFR4.3.

The software was verified to be usable on all major operating systems as our team uses the three aforementioned operating systems and successfully managed to run the software. On the topic of long-term stability, the software is kept stable on all major operating systems by testing the product both manually and automatically on machines running the operating systems both natively or in a virtual machine.

7.2 Major Errors

We have decided that our final product should be delivered without any major errors still present. Currently, we have identified the following types of major errors within the software:

Process interrupting errors:
Reflected in UC4 and tackled in R2.2, is an error tied to the interruption of file processing. In the current version of the GUI, cancelling the processing of data within a run of the software leads to the program crashing. We plan to tackle this problem by modifying the code to correct the way the thread is being handled. The new implementation will remove the created files and also display a confirmation message when cancelling in the middle of file processing.

8 Continuous Integration

Through Github Continuous Integration, automatic processes can be run whenever changes to the code are being made. When a commit is made, Github CI will automatically invoke several actions that can be used for i.e. testing or compiling the code. Github CI is used for the automation of several processes.

8.1 Executables

Github CI is used to generate executables. When a commit is tagged with a version number, Github will automatically create a release. It will then run jobs on Windows, MacOS and Linux to create executable files for all of these operating systems. These files will then be uploaded as release assets. This way, the user does not have to perform complex installation steps and compile the code, but can instead simply launch a single file to use the application.

8.1.1 Windows

On Windows, PyInstaller is used to generate the executable. The installation of PASTAQ on Windows is more complex than on other platforms. Firstly, zlibstatic has to be installed by downloading and building it from the source using cmake. Then, PASTAQ is downloaded and built from the source and additional dependencies are installed. After that, PyInstaller can be used to package the application and its dependencies into an .exe file, that is then uploaded to Github. This Windows support satisfies R2.5.

8.1.2 MacOS

On MacOS, PyInstaller is used to generate the executable. First of all, the dependencies like PASTAQ and PyQt5 are installed through pip. Then, PyInstaller is used to package the application and all its dependencies into an .app file. This file is then uploaded to Github in a .zip file. Providing the same support for MacOS satisfies R2.6.

8.1.3 Linux

On Linux, PyInstaller is used to generate the executable. The dependencies like PASTAQ and PyQt5 are installed through pip. Then, an executable is generated using PyInstaller. This file is finally uploaded to Github as a release asset. The executable for Linux is mentioned in R2.7.

The aforementioned executables are listed like the following:

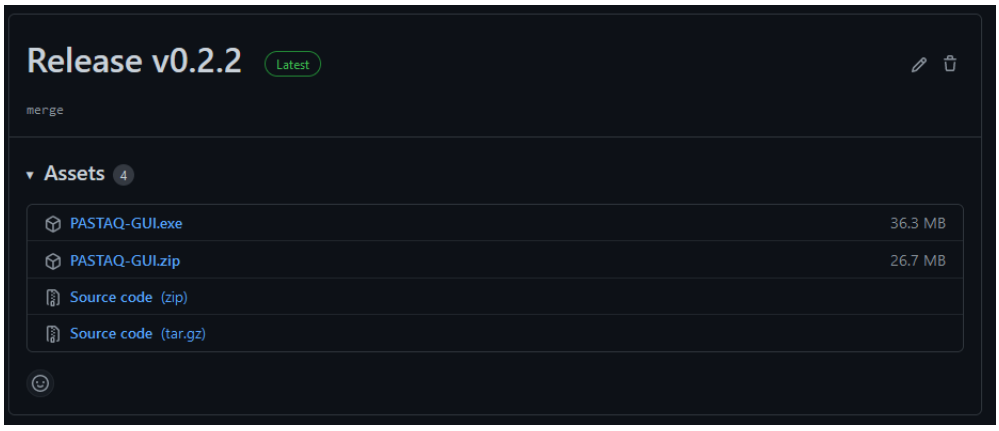

Figure 17: Executables

They can be found under the following link [4], which is the repository for our project.

[4] https://github.com/tudor-dragan/PASTAQ-GUI/releases

9 Team Organization

Table 1: Team Organization

Team Member	Main Focus
Björn Schönrock	CI and GUI
Kaitlin Vos	Tooltips, File processing and GUI
Tudor Dragan	GUI
Mohammed Nacer Lazrak	GUI
Dominic Therattil	GUI
Teresa Ferreira	GUI
Cristian Iacob	GUI

10 Change log

Table 2: Change Log

Date	Time	Team Member	Changes	Version
12.2.2022	13.58	Kaitlin Vos	Inserted change log table	1.0
14.03.2022	13:33	Kaitlin Vos	Structure and team organization	1.1
15.03.2022	23:50	Cristian Iacob	Added title page, QA diagram & descriptions for technology stack	1.2
16.03.2022	00:47	Kaitlin Vos	Technology stack & detailed structure	1.3
21.03.2022	17:22	Cristian Iacob	wrote introduction, moved QA diagram to introduction	1.4
21.03.2022	22:57	Kaitlin Vos	Tooltip section & organized document	1.5
26.03.2022	15:00	Björn Schönrock	started CI section	1.6
28.03.2022	02:06	Cristian Iacob	fixed introduction and quality assurance sections & small latex fixes & moved sections so they follow more logically	1.7
29.03.2022	19:01	Kaitlin Vos	Tooltip and file procedure section completed	1.8
30.03.2022	15:00	Björn Schönrock	CI section & fixed document structure	1.9
31.03.2022	12:39	Kaitlin Vos	File procedure aligns with presentation	1.10
01.04.2022	12:05	Critian Iacob	Fixed document formatting, added stability section and rewrote supported OS subsection	1.11
01.04.2022	13:15	Critian Iacob	Moved supported OS subsection to QA, changed Stability section to Major errors	1.12
01.04.2022	13:30	Critian Iacob	Reformat sections based on priority	1.13
01.04.2022	16:10	Critian Iacob	Wrote GUI Rework beginnings	1.14
01.04.2022	16:30	Critian Iacob	Wrote Abstract Section	1.15
02.04.2022	14:27	Teresa Ferreira	Added Architecture Overview Diagram	1.16

Continued on next page

Table 2: Change Log (Continued)

Date	Time	Team Member	Changes	Version
02.04.2022	22:27	Tudor Dragan	Added New Architecture Overview Diagram	1.17
03.04.2022	23:46	Kaitlin Vos	GUI rework and diagram for file procedure	1.18
04.04.2022	11:13	Kaitlin Vos	Proof-read entire document, fixed typos, syntax, logic	1.4.1