Mass Spectrometer Interface

A Desktop Application for Reading Instrument Data

Cousins Photosynthesis Lab in the School of Biological Sciences at WSU



**Team Linnaea Borealis**

****

Kyler Kupp, Erik Holtrop

**Assignment Notes**

Length = minimum of 5 pages text + appendices as needed - though, this should be \*MUCH\* longer than 5 pages if you leverage all of your prior documents  
  
Sections that do not count to content for page limit:

* Cover page
* table of contents
* pictures
* tables
* images
* diagrams

Posted as a single self‐contained file (no links to outside resources.)

Posted as a PDF file.

Typed single‐spaced.

Typed with black text.

Typed with #11 font size.

Typed using Arial font or similar font.

Typed with one inch margins on sides, top and bottom.

**Please erase this page in your final document.**

**TABLE OF CONTENTS**

[I. Introduction 4](#_Toc183427541)

[II. Team Members - Bios and Project Roles 4](#_Toc183427542)

[II.1. Erik Holtrop 4](#_Toc183427543)

[II.2. Kyler Kupp 4](#_Toc183427544)

[III. Project Requirements 4](#_Toc183427545)

[IV. Solution Approach 5](#_Toc183427546)

[IV.1. System Overview 5](#_Toc183427547)

[IV.2. Architecture Design 5](#_Toc183427548)

[IV.3. Data design 11](#_Toc183427549)

[IV.4. User Interface Design 12](#_Toc183427550)

[V. Test Plan 15](#_Toc183427551)

[VI. Alpha Prototype Description 15](#_Toc183427552)

[VI.1. [Subsystem Name] 15](#_Toc183427553)

[VII. Alpha Prototype Demonstration 16](#_Toc183427554)

[VIII. Future Work 16](#_Toc183427555)

[IX. Glossary 16](#_Toc183427556)

[X. References 16](#_Toc183427557)

[XI. Appendices 16](#_Toc183427558)

# Introduction

Your introduction should describe your project in sufficient detail such that the alpha-prototype description will be easy to follow. This should include material similar (lifted from and edited) from your Project Description document. The sections to focus on will be:

* Introduction - likely needs major editing since the project has evolved
* Background and Related Work
* Project Overview
* Client and Stakeholder Identification and Preferences

Add a brief paragraph to the introduction and explain the purpose of this document as a summary of the project’s progress and technical details about your engineering efforts.

# Team Members - Bios and Project Roles

Our team brings together a diverse set of skills and experiences, ranging from software engineering to data visualization and algorithm development. Each member contributes unique expertise and a shared commitment to delivering a robust and efficient data collection and visualization system. Below, you’ll find an overview of our team members, highlighting their academic backgrounds, technical interests, and specific roles within the project.

## ****Erik Holtrop****

Erik Holtrop is a dedicated computer science and mathematics student at Washington State University, where he is set to graduate in 2025. His academic journey has honed his skills in software engineering, algorithms, and mathematical computing, particularly through coursework like Cpt\_S322 Software Engineering (Python) and Cpt\_S350 Design and Analysis of Algorithms. Erik’s technical expertise includes Python, C/C++, C#, SQL, and Haskell, along with experience in Python libraries such as MatPlotLib, PyEDA, and PyQT. His responsibilities include managing data structures, debugging, project management, and integrating mathematical algorithms into the system.

## ****Kyler Kupp****

Kyler Kupp is a driven computer science student at Washington State University, aiming to complete his degree in 2025. He combines academic excellence with hands-on industry experience, having worked as a Software Engineer Intern at Monson Fruit Company. In this role, Kyler developed monitoring programs, automated reporting systems, and contributed to software managing critical production processes. His technical toolkit includes proficiency in Python, C#, SQL, and JavaScript, with familiarity in libraries like PyEDA and SFML. Kyler is responsible for UI/UX, stakeholder relations, and documentation oversight.

# Project Requirements

Use this section to define the requirements for your project. Lift materials from the Project Requirements document as needed, notably sections II and III. This will orient your reader to the quantified requirements expected from your final design and ensure they are ready for the Solution Approach.

While you can copy directly from your current Requirements document, make sure to have an introduction paragraph stating what this section includes.

# Solution Approach

This section aims to outline our project’s approach. This includes the different parts of the project modules (Architecture Design), data storage and manipulation methods (Data Design), and how the project modules will outwardly look (User Interface Design). These details are intended to elucidate objectives and methods for current and future developers, but to do so using language that can mostly still be understood by stakeholders for overview.

## System Overview

The project is designed to provide a comprehensive platform for managing and interpreting Mass Spectrometer data through various modules. It aims to present data insights through graphical and numerical perspectives while offering flexible data formatting and integration capabilities. The system consists of five primary modules: the first three focus on different ways to visualize and log Mass Spectrometer data, the fourth module reformats data from a secondary Mass Spectrometer, and the fifth combines data streams from three distinct instruments. The general design for each of the modules is to accept input from a file or folder, which is selected by the user, process it in some way (reformat, normalize, derive calculations from, etc.), and then output that processed information, either to graphs on the screen, or to new files.

## Architecture Design

### Overview

Team Linnaea Borealis' architecture design focuses on Module 5. Each module in the system operates as a standalone program, but Module 5 is the primary module requiring architectural design innovation, as most other modules remain stable and do not necessitate changes. The team has adopted a Pipe and Filter architecture for this design, which is particularly well-suited for this system's data processing requirements. The system requires data to be read from a file, normalized, plotted, and then transformed. This process fits Pipe and Filter architecture well. This approach allows for clear separation of concerns, where data flows through a series of independent filters (or stages), each performing specific tasks such as reading, processing, and visualizing data. This modular design enhances flexibility, scalability, and maintainability, making it easier to integrate new features or modify existing ones without affecting the entire system. By leveraging this architecture, Team Linnaeus ensures a clean and efficient flow of data from input to output, supporting both current and future needs. The system begins with the User Interface (UI) component, which allows users to specify a file path and view plotted data and calculation results. The UI interacts with the File Reader, which reads raw data from the CSV file provided by the user and passes it to the Data Processor. The Data Processor then normalizes the raw data by applying a natural logarithmic transformation, preparing it for further steps. Once processed, the data is sent to the Graphing Engine, which updates the current graph by integrating the new data. The updated graph is then displayed in the UI for the user and sent to the Calculation Engine. The Calculation Engine performs further computations on the processed data, producing analytical results that are also displayed in the UI such as the mean of a selection of data. This flow of data, facilitated by the modular design, ensures efficiency, flexibility, and ease of maintenance.

A diagram of a system

Description automatically generatedFigure 1: System Block Diagram

### Subsystem Decomposition

This section outlines how the system has been decomposed into its major subsystems, each corresponding to the core components identified earlier: the User Interface (UI), File Reader, Data Processor, Graphing Engine, and Calculation Engine. The decomposition was designed to ensure that each subsystem represents a manageable unit of work for a single developer, with clearly defined responsibilities and minimal overlap. Each subsystem has been assigned a specific functionality: the UI handles user interactions, the File Reader manages data input, the Data Processor handles transformations, the Graphing Engine visualizes the data, and the Calculation Engine performs analytical operations. The rationale behind this decomposition emphasizes cohesion and coupling. High cohesion is maintained within each subsystem by ensuring that each is focused on a single task or closely related set of tasks. Meanwhile, coupling between subsystems is minimized by defining clear and straightforward interfaces for data exchange, allowing for seamless interaction without unnecessary dependencies. This modular design not only improves the maintainability and scalability of the system but also supports flexible integration of future enhancements.

#### User Interface

##### Description

The User Interface (UI) subsystem is responsible for managing the interaction between the user and the system. Its primary function is to display graph data generated by the graphing engine as the results of calculations performed by the Calculation Engine. Additionally, the UI allows the user to input the file path of the CSV data, which is then passed to the File Reader subsystem for processing. The UI acts as a central hub, presenting outputs from different subsystems in a coherent and user-friendly manner.

##### Concepts and Algorithms Generated

The User Interface subsystem was designed with simplicity and ease of use in mind, utilizing a basic file input mechanism and graphical display capabilities. Concepts such as event-driven programming were considered, allowing the UI to react to user inputs (e.g., file path selection) and update dynamically when data from other subsystems (such as the graph or calculations) is received. The selected solution leverages standard UI frameworks to achieve these goals with minimal latency and high responsiveness. The decision to keep the UI simple was influenced by the need to maintain a clear separation of concerns, ensuring that complex data processing is handled by other subsystems. Trade-offs included balancing performance with usability, ensuring the UI remains responsive even when handling large datasets.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| GetFilePath | File Reader | The GetFilePath service will pass over the user-selected file paths for data acquisition. Three file paths should be returned, one for each instrument utilized by the system. |
| GetDataSelection | Calculation Engine | This service will return the bounds of the data selected by the user using two movable vertical bars. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| CalculateMean | Calculation Engine |
| GetLatestGraph | GraphingEngine |

#### File Reader

##### Description

The File Reader subsystem is responsible for reading data from three different files, each representing data from a distinct instrument used by the system. After reading the raw data from these files, the File Reader passes the data along to the Data Processor for further processing. This subsystem acts as the initial stage in the data pipeline, ensuring that the raw information is properly retrieved and made available for downstream components.

##### Concepts and Algorithms Generated

The File Reader subsystem was designed with the need to efficiently handle multiple file inputs. The concept of parallel file reading was considered to improve performance when handling large datasets, but ultimately, a sequential approach was selected due to the relatively manageable file sizes. The File Reader ensures that each file is opened, its contents are read, and the data is collected into a format that can be easily passed to the Data Processor. Special considerations were made to handle potential file reading errors, such as missing or corrupted files. In such cases, error handling mechanisms trigger notifications to the user. The trade-off involved in this design was balancing robustness with simplicity, ensuring that file reading remains a lightweight process while effectively managing edge cases.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| ReadFileData | Data Processor | This service reads the contents of the three files, each corresponding to an instrument's data. The service accepts the file paths as input (provided by the User Interface) and returns the raw data from all three files as output. The data is passed on to the Data Processor for formatting and normalization. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetFilePath | User Interface |

#### Data Processor

##### Description

The Data Processor subsystem is responsible for transforming raw data received from the File Reader into a format that is more suitable for further analysis. Its primary task is to normalize the data using the natural logarithmic transformation, making the data easier to process and visualize. Once the data is formatted and normalized, the Data Processor sends the processed data segment to the Graphing Engine, where the new data is added to the current graph. This subsystem ensures the integrity and usability of the data as it flows through the system.

##### Concepts and Algorithms Generated

The main algorithm used by the Data Processor is the natural logarithmic transformation, which helps normalize the data. This technique was selected because it reduces variance in the data and handles large discrepancies in value magnitudes, making it more suitable for visual representation and computation. Alternative normalization methods, such as z-score normalization, were considered but ultimately not chosen, as the natural log method better aligned with the requirements of the data's structure and scale. Special considerations included handling cases where the raw data contains zero or negative values, as these would cause issues during the logarithmic transformation. Error handling mechanisms and data validation were incorporated to account for such anomalies, ensuring robust processing. The trade-off in this approach involved balancing data transformation speed and the complexity of handling edge cases.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| ProcessData | Graphing Engine | This service receives the raw data from the File Reader, processes it by formatting and applying a natural logarithmic normalization, and then sends the processed data to the Graphing Engine for visualization. The input consists of the raw data from three different files, and the output is the normalized and formatted data ready for graphing. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| ReadFileData | File Reader |

#### Graphing Engine

##### Description

The Graphing Engine subsystem is responsible for visualizing the processed data received from the Data Processor. Its main role is to update the current graph by incorporating the newly processed data segment and rendering it in the user interface. This allows the user to observe real-time visual changes in the data as it is processed. The Graphing Engine ensures that the graphical representation is clear, accurate, and continuously updated as new data is received.

##### Concepts and Algorithms Generated

The Graphing Engine employs algorithms to dynamically update the graph with new data segments. Various graphing techniques were considered, including line charts and bar charts, but a continuous line chart was selected as the best fit for real-time data visualization. The primary concept is to ensure that the graph scales efficiently as more data is added, while also maintaining clarity and responsiveness.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| UpdateGraph | User Interface | This service receives the processed data from the Data Processor and updates the current graph displayed in the User Interface. The input is the normalized data, and the output is an updated visual graph reflecting the latest data. The service ensures that the graph is refreshed in real-time as new data is processed. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetProcessedData | Data Processor |

#### Calculation Engine

##### Description

The Calculation Engine subsystem is responsible for performing various calculations on the data, including allowing the user to select a segment of data and calculating the mean of that segment. This subsystem plays a crucial role in providing numerical insights into the processed data, complementing the visual representation provided by the Graphing Engine.

##### Concepts and Algorithms Generated

The primary algorithm used by the Calculation Engine is the calculation of statistical measures such as the mean. The mean is calculated by summing the selected data points and dividing by the number of points. Several approaches were considered, including pre-calculating statistics for all data and storing them for quick access. However, the selected solution was to compute the required statistics on-demand based on the user’s selection. This minimizes resource consumption and ensures that calculations are based on the latest data segment selected by the user.

##### Interface Description

Services Provided:

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Provided To** | **Description** |
| CalculateMean | User Interface | This service calculates the mean of the selected data segment. The input is the processed data for the user-selected segment, and the output is the mean value. The mean is displayed in the User Interface as part of the calculation results. |

Services Required:

|  |  |
| --- | --- |
| **Service Name** | **Service Provided From** |
| GetDataSelection | User Interface |
| GetLatestGraph | GraphingEngine |

## Data design

This section covers the different ways our project stores data, both while it’s running, and when the data is stored for later. This overview will be a fairly technical overview, meant mostly for developers. Each subsection covers a different data structure used in the project.

### Shared Singleton

A shared singleton is a design pattern that forces only one instance of an object to exist throughout the whole program. Although this design pattern does not usually describe a data structure, in modules 1-3, most important data structures are members of a shared singleton, named sharedData. These members include fileList (subsection IV.2) and dataPoints (subsection IV.3)

### Basic List

In modules 1-3, we use a basic list to store the names of files that are read from. This list is named fileList and is stored in the sharedData shared singleton (subsection IV.1).

### Dictionary

In modules 1-3, we use a Python dictionary to store the mass spectrometer data, named dataPoints. The keys are time points, and the values are tuples of isotope masses. This dictionary is stored in the sharedData shared singleton (subsection IV.1).

### Log Table

In modules 1-3, there are certain data points that can be pulled out from the graph, or otherwise calculated from the data. These can be saved into a table built into the UI. These logs are then stored directly in the PyQT UI component QTableWidget.

### CSV Export

Modules 1-3 can export logged data from the log table (subsection IV.4) into a CSV file, which matches the format of the table.

### Pandas DataFrame

Module 4 uses a DataFrame from the Pandas Python library to store data from Mass Spectrometer 2. This is very similar to a dictionary approach (subsection IV.3), but effectively replaces the need for unique keys with ordering and more importantly allows for the use of Pandas methods like to\_csv at a small performance cost.

### CSV Series

Module 4 exports the DataFrames (subsection IV.6) as CSV’s with ~8 rows each, for use in modules 1-3. Each row contains a time signature and isotope masses. These CSV’s are named with numbers in order of their time signatures.

## User Interface Design

The user interface will first allow the user to select acquisition folders for data from the Mass Spectromer, for modules 1-4, or for data from each of the three instruments, in the case of module 5. It will share the same design as shown in Figure 2. However, it will have three separate options for each type of data that the system utilizes. This portion of the interface corresponds with use case UC-4: Select Input Files.

A screenshot of a computer

Description automatically generated

Figure 2: File Selection

The largest section of the user interface will display a black graph such as the one shown in Figure 3. To the left of the graph, color coded labels for each data stream are present to provide an intuitive viewing experience. The graph is populated by line-graphs, color coded for each data stream. This corresponds to the use case UC-5: View Graph. The graph will be easily scalable using the scroll wheel to zoom in or out. Below the graph and to the right of Figure 3 a slider is shown. This slider allows the user to adjust the speed at which data is plotted from 0.5x to 32x. This corresponds to use case UC-7: Change Plotting Speed. On the left of the slider is also a pause/resume button as well as a start button that allows the user to toggle the plotting of data. In addition to the primary graph, modules 1-3 have additional graphs for calculations derived from the primary data.

A screen shot of a graph

Description automatically generatedFigure 3: Graphing and Calculations

Using the leftmost button in Figure 3 that features two vertical lines, the user can toggle the visibility of “mean bars”. These vertical bars pictured in Figure 4 can be shifted left to right using the cursor to select a specific portion of data. This corresponds to the use case UC-8: Select Data Points.

A blue grid with white lines

Description automatically generatedFigure 4: Data Segment Selection

Available calculations will be featured below the graph and its controls. Here the user will be able to use the “Get Mean” button to calculate the mean inside of the selected region. This corresponds to use case UC-9: Calculate Mean.



Figure 5: Mean Calculation

In addition to allowing users to scale the graph using their scroll wheel, users will be able to achieve more precise scaling by left clicking the graph to open further adjustment options. As illustrated in Figure 6, users will be able to input the exact bounds of each axis, and the graph will adjust according to their selection.

A screenshot of a computer program

Description automatically generated

Figure 6: Detailed Graph Scaling

Overall, the user interface is designed for an intuitive data acquisition and visualization experience. Users can select acquisition folders for three instruments, with options tailored for each data type, as shown in Figure 2 (UC-4: Select Input Files). The main area features a black graph displaying color-coded line graphs, accompanied by labeled indicators for clarity (UC-5: View Graph). Users can zoom using the scroll wheel and adjust the plotting speed with a slider (UC-7: Change Plotting Speed). Controls for toggling mean bars and selecting data points further enhance functionality (UC-8: Select Data Points), while a “Get Mean” button allows for quick calculations of averages within selected regions (UC-9: Calculate Mean). Additionally, precise graph scaling options enable users to customize their view, ensuring a comprehensive and user-friendly interface.

# Test Plan

This section provides an overview of the steps we take to test different elements of our project. This overview includes the overall flow, the unit tests for individual parts, and integration/system testing for combinations of different parts. The processes outlined are very particular to the context of our project; we’re very aware of our stakeholders and the integration plan that they prefer. Ultimately this section is intended to outline what we consider the ideal methods of testing, in order to keep development in line with those practices.

## Testing Strategy

The following is our loose approach to testing a particular module or feature:

1. Identify the requirement(s) involved in this module/feature. This should either come from the Requirements and Specifications Document or be added to the Requirements and Specifications Document before continuing.
2. Establish the test(s) that will be used. In other words, identify the process of using the module or feature. Document these tests in the Testing Plan Document.
3. Identify any necessary dependencies. This includes other components and input data. Include assumptions about these dependencies in the Testing Plan Document.
4. Build a representation of what acceptable results look like. This must consider our assumptions made in the previous step. For example, an Excel graph of a data acquisition: the particular data acquisition should be clarified in the previous step, with the Excel graph built off it in this step. This mockup(s) should either be included in the Testing Plan Document, or in the relevant module’s “Testing” folder with reference to it in the document.
5. Perform the test(s).
6. If the test(s) is unsuccessful, fix it if possible. If the test(s) is not successful by next standup meeting, prepare a short explanation or document explaining the issue.
7. If the test is successful, move the relevant GitHub issue to Review/QA, or from Review/QA to Done.

Ultimately, our strong connection to our primary stakeholder, Dr. Cousins, allows us to adapt our development process to a more flexible approach that handles opportunities and issues as they come up. Sometimes the requirements are vague, and the following approach may be more effective than generating more specific requirements and a mockup:

1. Implement the most obvious executions of a requirement.
2. Present those executions to the client/stakeholder(s). Receive feedback.
3. If one of the implementations is acceptable, move the relevant GitHub issue to Review/QA, or from Review/QA to Done.
4. If none of the implementations are acceptable, either return with novel implementations or revert to the primary approach, depending on team consensus.

Our delivery process is basically Continuous Delivery. Our client prefers executable files over python scripts, so a new deployment must be manually created by a team member each time. The modular nature of the project lends itself to creating a new iteration of each improved module every sprint. In this regard, our development is continuously integrated with monthly releases.

## Test Plans

This section outlines the comprehensive plan for testing the mass spectrometer interface system, detailing the strategies for unit, integration, system, and user acceptance testing. Each testing phase is designed to identify and address potential faults at different stages of development, from individual software units to the entire integrated system. The testing plan follows a systematic approach, beginning with isolated unit testing to validate the smallest components of the application, followed by integration testing to ensure smooth communication between modules. System testing will then evaluate the application’s compliance with overall requirements, focusing on functional, performance, and stress tests to confirm reliability under varying conditions. Finally, user acceptance testing will involve end-users to validate that the system meets their needs and is ready for operational deployment.

### Unit Testing

The primary objective of unit testing for this system is to validate the functionality and reliability of individual components, or “units,” by isolating them from the rest of the code and checking for bugs or unexpected behavior. Specifically, the unit tests will cover core functionalities, including data parsing from CSV files, data transformations, calculations, and graphical display setup. Unit tests will be designed for each function and method within the modules, verifying both expected outputs and error-handling mechanisms when presented with invalid data inputs. Using Python’s pytest framework, tests will be semi-automated to streamline the process and improve reliability. Mocking will be employed to simulate data inputs and dependencies where necessary, especially for modules that rely on external data sources or interactions. This approach ensures that each unit functions independently and accurately, laying a stable foundation for subsequent integration and system testing phases.

### Integration Testing

The purpose of integration testing in this system is to identify faults that may arise when individual components interact, focusing on groups of components rather than isolated units. This phase will ensure that data flows smoothly between components. For instance, components responsible for parsing CSV files and transforming data will be integrated and tested as a cohesive unit to confirm that each stage performs as expected in the broader workflow. To manage dependencies, a test data set simulating real-world CSV inputs will be used to validate functionality and data consistency across components. Python’s pytest framework will be used for semi-automated integration tests, while pytest-mock will aid in simulating dependencies, ensuring that testing conditions are controlled and predictable.

### System Testing

System testing will be conducted to ensure that the mass spectrometer interface system operates as a cohesive unit, meeting all specified requirements. This phase will involve executing a series of planned tests to validate both functional and non-functional aspects of the system as a whole.

#### Functional testing:

In functional testing, we will develop a comprehensive set of test cases based on the functional requirements outlined in the project documentation. Each functional requirement will correspond to at least one test case. Each standalone system will be tested in realistic scenarios to ensure that they meet user expectations. Test cases will be prioritized to focus on critical user paths and high-risk areas, ensuring that the most relevant features are validated first. Any failures or discrepancies found during testing will be documented and addressed promptly to enhance system reliability.

#### Performance testing:

To assess the system's performance, we will conduct performance testing that focuses on response times, resource utilization, and overall system stability. This will include stress testing the systems by simulating high-load conditions using large datasets to determine how the system performs under pressure. For example, we will measure the speed at which the system can plot all data points from a large sample at once. Key metrics such as processing speed, memory usage, and data handling capacity will be monitored. If any performance issues arise, they will be investigated and resolved to ensure the system meets the expected performance benchmarks.

#### User Acceptance Testing:

User Acceptance Testing will involve lab researchers in evaluating the system based on their operational needs. We will organize testing sessions where lab researchers will perform key tasks, such as loading CSV files, converting data, plotting data, and processing calculations in the same way they would during their academic research. Feedback will be gathered during these sessions to identify any areas requiring adjustment or enhancement. This testing phase is crucial for ensuring that the system is user-friendly and meets the designated requirements. Any issues identified will be prioritized for resolution to ensure the system is fully prepared for operational use.

## Environment Requirements

To ensure comprehensive testing of the mass spectrometer interface system, this section outlines the necessary and desired properties of the testing environment. The setup will enable thorough verification of each module’s functionality, accuracy, and performance in processing and visualizing mass spectrometry data.

The testing environment should be equipped with a Windows 10 or Windows 11 operating system. The environment will rely on Python, version 3.8 or higher, to match the development specifications of the system. Key Python libraries, such as PyQt5 for graphical user interface elements and pandas for handling data and CSV file operations, should be pre-installed. Each module has a requirements.txt file that can be used with pip to install the necessary libraries. Additionally, a sample CSV file will be prepared, including real mass spectrometer data, to evaluate the system’s handling of data directly from a file.

Special tools will aid in testing, such as the pytest framework for semi-automated unit and integration testing. Mocking tools, like unittest.mock or pytest-mock, will be used to simulate various data inputs and scenarios, particularly in modules focused on data handling and conversion. Performance monitoring, using tools like memory\_profiler or timeit, will help assess memory usage and processing time for modules dealing with larger datasets.

Module 4 has extra special requirements for its extra specific use case. The module takes in data from a Mass Spectrometer through an EZ-Tap serial-to-usb listener device. Both the Mass Spectrometer and the EZ-Tap are assumed for the testing and usage of module 4. In order to ensure these conditions, we use remote-access to test on the computer that has the EZ-Tap plugged into it.

These environment specifications aim to provide a controlled, reliable setting that ensures system stability, accuracy, and performance across diverse data scenarios and software setups.

# Alpha Prototype Description

\*\*\*This is the first section that is truly new for this document\*\*\*

Describe your alpha prototype implementation. Please format this section according to what you think is the best way to describe your prototype. The following is just a suggestion.

Explain what parts/subsystems of your proposed architecture your team started to implement. Mention the current progress at each part, i.e., how much of the proposed functionality you have completed.

For the parts that you already started to work on, explain whether you have tried to integrate them with the rest of system. In other words, explain which of the interfaces in your architecture have you started to implement.

If you have performed any tests on your alpha prototype (or the subsystems of your prototype), briefly explain your findings.

I recommend to include plenty of images and pictures of the following where appropriate:

- any diagrams/figures that visualize various features of your prototype;

- the screenshots of your user interfaces;

- the screenshots of your test programs;

- pictures of your team testing and debugging the devices, programs, etc.

A well-thought and clear diagram is better than long and descriptive text.

If your document starts to be very long due to screenshots and diagrams, please put at least some of them into an appendix to this document.

For each subsystem that you have implemented in your alpha prototype, you may include the following sub-sections.

## [Subsystem Name]

### Functions and Interfaces Implemented

List and describe the implemented functionality. Explain the remaining work.

### Preliminary Tests

Report any test results for the unit and integration tests that you performed on your prototype. This subsection is a good place to include screenshot images from your tests (if applicable). A notable component here would be to include the results of your CI/CD status. Hopefully master still builds, right?

# Alpha Prototype Demonstration

Summarize the highlight of your prototype demonstration to your mentor. The items to discuss in this section may include the following. (Please include all other necessary details in addition to the following).

1. Summary of what you showed to your mentor.
2. Your mentor’s comments/suggestions on your prototype.
3. Your mentor’s questions to your team and your responses to those questions.

After testing your prototype and demonstrating it to your mentor, you will have a better idea whether the initial design you proposed earlier will work. Additionally your mentor might suggest modifications to your current design. In this section list and explain all design modifications that you plan to make based on your preliminary test results and mentor comments (if applicable).

# Future Work

List the major tasks for the second semester and briefly explain your plan to complete them.

# Glossary

Define technical terms used in the document.

# References

Cite your references here. -- Ensure you’re pulling them from your earlier works!

For the papers you cite give the authors, the title of the article, the journal name, journal volume number, date of publication and inclusive page numbers. Giving only the URL for the journal is not appropriate.

For the websites, give the title, author (if applicable) and the website URL.

Please use either Chicago or IEEE format for your citations

# Appendices

As needed, copy over your appendices for the various sections. You can have as many appendices as required. Normally, they’re numbered with letters:  
Appendix A  
Appendix B  
…  
Appendix *n*