CA400 Project

Functional Specification

Project Title	comet - Code Metrics & Analysis Tool
Student 1 - Name	Kevin McGonigle
Student 1 - Student Number	16318486
Student 2 - Name	James Miles
Student 2 - Student Number	16349533
Date	22/11/2019

0. Table of Contents

1. Introduction

1.1 Overview

For our final year project, we will be developing a powerful, comprehensive and user-friendly code metrics and analysis tool, to which we have given the name “comet” (code-metrics). The ideal end-goal of this tool would be for it to take the form of an IDE plugin that will display useful data relating to the performance, structure and size of a codebase in a clear, concise and intuitive manner. The data in question will encompass a wide variety of informative metrics and analytics including cyclomatic complexity, maintainability index and coverage. These measures will be computed on a back-end server when a codebase is submitted for analysis and will be displayed to the user in a clean, interactive and navigable front-end user interface in the form of graphs, tables and diagrams.

As humanity has trended towards placing an increasingly greater emphasis on technological advancement, software development has become one of the fastest growing occupations on the planet. The Bureau of Labor Statistics (BLS) of the U.S. Department of Labor projects that the employment of software developers is projected to increase by 21% from 2018 to 2028. With such a demand for new software development professionals and growth in the scale and scope of the industry comes a demand for tools that assist in the production of reliable, efficient and manageable code, as well as providing an educational resource for new and experienced developers to better understand their code, how it works and how it could be improved. Comet does exactly this, making it a universally ideal solution for developers of varying skill levels and experience to increase familiarity, clarity and understanding of even the most complex and convoluted codebases.

When the user launches comet from their IDE, the codebase that they are working with will be automatically uploaded to a back-end server for analysis. In a real-world, business-scale setting, this server would ideally be a powerful physical server or virtual machine that is maintained and owned by a customer of comet, on which the comet API will be hosted. Our project however will likely utilise a web hosting service such as an Amazon Web Services EC2 instance to host comet’s back-end API-based infrastructure as a proof of concept. Once the codebase has been received, the python-based back-end will begin parsing the code and computing each of the metrics and analytics before returning the computed values to the front-end. From there, the front-end user interface - implemented using React and Redux - will interpret these values and subsequently create and display to the user the various graphical representations of the relevant measures as they are requested.

This specification is not intended to be rigid and final. The layout of screens and certain behaviours described in this document may be subject to change and are included here to illustrate the underlying functionality of comet and as a general blueprint for how we plan to implement the system at this early stage. Iterative feedback from users over time will help develop the look and feel of the application.

1.2 Business Context

The business context of comet is not strictly applicable in our use case due to the limitations surrounding the use of comet's API within a company. It is very unlikely that a business would be willing or even legally permitted to send their data to any external source. Companies wishing to protect their software source code often ensure that employee contracts include clauses that expressly prohibits the employee from disclosing, storing or copying its confidential information. An employee sending any part of the company's code to an off-site comet API would be in breach of these terms. Considering this restraint, the priority of comet's functionality within a business environment was deemed to be secondary to that of the more critical features. This resulted in a focus towards a general user application with the capacity for further expansion in the event of on overestimation of our estimated development time.

However, there does exist a solution to the NDA barrier. Expansion into the business market would require that the server-side comet API be distributed to industrial clients for hosting on their own server(s). This would then provide them with the ability to send requests and receive comet's replies within the scope of their own network - thus, avoiding the transferal of data outside their server scope. Considering the power and size of some companies' distributed systems, more components may be required to account for the complexities these systems bring - such as load balancers or the means to configure the API to suit the business requirements.

1.3 Glossary

comet - The name given to the system being described in this document.

2. General Description

2.1 Product / System Functions

comet will ideally take the form of a plugin for a popular IDE such as Microsoft's Visual Studio. Taking Visual Studio as an example, with comet installed, an option under the "Analyze" menu labelled "Analyze with comet" will be available. Upon selecting this option, the codebase of the currently open Visual Studio solution or

project will be uploaded to the comet API and a web form dialog will be displayed. Once the codebase has been received by the server-side comet API, the individual files will be parsed and interpreted, and the various metrics for the codebase as a whole will be computed.

Meanwhile, the client-side user interface will begin displaying the code metrics hub page. This page will consist of a file tree on the left-hand side of the display, in which the

2.2 User Characteristics & Objectives

2.2.1 User Scenario 1: Jane is a secondary level student beginning to explore the world of programming. However, Jane has found many of the online tutorials to be overwhelming, confusing and as a visual learner Jane is struggling to come to grips with some of the more complicated beginner topics. Jane's current tutorial is explaining the use of functions and imports. Placing the tutorial's code into comet would yield a tree graph structure like so:

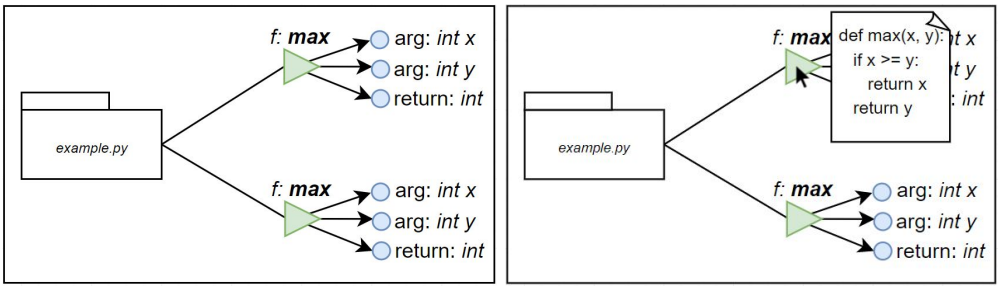


Fig 2.2.1.1 A file's functions & those function's corresponding attributes and return value

Fig 2.2.1.2 A code snippet is displayed showing the function's code contents

A concise visual breakdown of the code is generated - displaying the file via a file icon, functions as green triangles and variables as blue circles (2.2.1.1). From these trees graphs it is evident that there exists a file called example.py that contains two functions - max and min. Both functions take two arguments - integer *x* and integer *y* and both return an integer. Extending this further, hovering the mouse over a function will present Jane with a view of the highlighted function's corresponding code (2.2.1.2).

Later within the tutorial Jane becomes confused with the import functionality - so she again inputs the tutorial's code to comet. (2.2.1.3) displays both files, showing the connections between the functions.py file and the main.py file. The enablement of Jane to inspect snippets of code allows her to view how exactly these files are imported and used. She now knows that the line "import * from functions" (2.2.1.3)

is responsible for the importation of the max and min functions to the main file. Expanding upon this further, Jane wishes to try and implement her own way of using imports. She creates a new file with the previous import statement and a *call_functions* method which calls both the max and min functions. She then uploads this to comet to be displayed. The relationships between files are shown via dotted lines which show the connection between the max and min functions and the call_functions function (2.2.1.3). Upon the hovering of the call_function, Jane is displayed her function's contents (2.2.1.4).

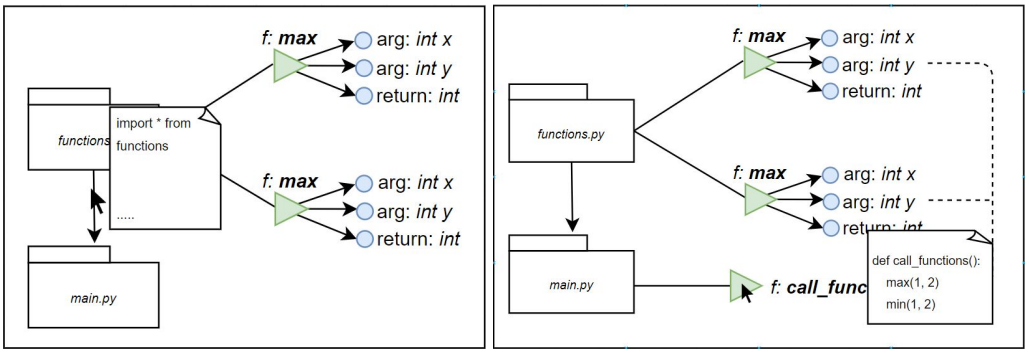


Fig 2.2.1.3 Code snippet displaying where functions.py's functions are imported into main.py

Fig 2.2.1.4 The relationship between the functions.py & main.py is displayed via the code snippet and dotted lines connecting the nodes.

comet provides Jane with a powerful, accessible and educational tool she can utilise to further her own knowledge. comet's simplistic nature creates an environment of accessible learning through a friendly and concise user interface - making the application available to those of all ages.

2.2.2 User Scenario 1: John is third level student seeking to expand his knowledge into larger code bases. As a student, John has had little to no experience with more convoluted code bases - utilising only a few files at a single time. John knows he will need to become familiar and comfortable before his placement on Intra. John has tried dissecting and understanding various open source applications, but his attempts so far have been futile. John is looking for a concise representation of a file directory, showing the relations between files and their most important information. He uploads a sample directory with the following file structure to comet to which produces:

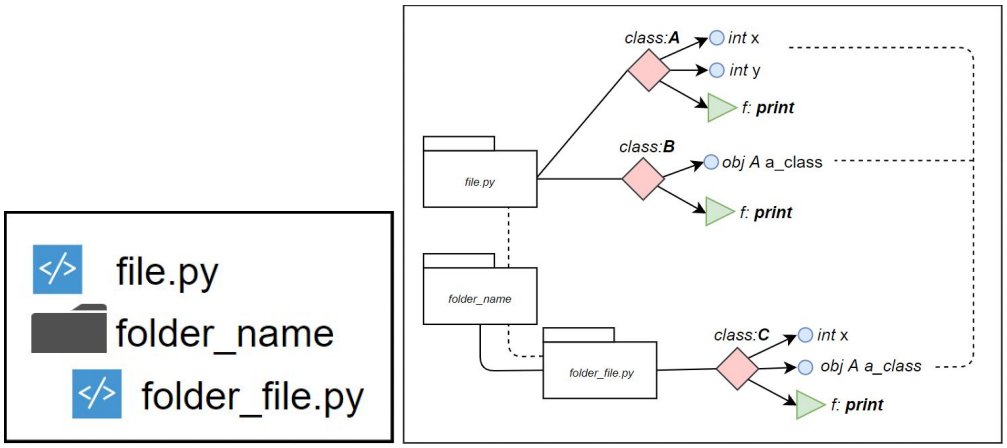


Fig 2.2.2.1 John's uploaded file structure

Fig 2.2.2.2 The produced tree graph from 2.2.2.1 directory

The generated tree graph succinctly displays the directory (2.2.2.2) - showing each folder or file's relationships and contents. *file.py* contains two classes, *class A* and *class B. Class A* takes the parameters x and y, however *class B* takes a *class A* object. John can see the relationship between this argument and the *A class* via a dotted line leading back to its parent class (A). *file.py* and *folder_file.py* are connected to by a dotted line. The dotted line, going between *file.py and folder_name.py*, shows the relationship between the two files and if John were to place his mouse pointer over this line, it would show the A class being imported to *folder_file.py*.

John then uploads a project he has been working on, in which he has created a simple server - client connection through the python socket modules. The treegraph displays a directory with three files - *client.py, server.py, main.py* and a singular external module *socket (2.2.2.3)*. Both *client.py* and *server.py* have imported from socket. Furthering this, John is able to see that the socket module is used in both the *Class* and *Server* class. *main.py* imports both files and includes two functions with dotted lines leading towards their respective parents (2.2.2.4).

Upon the placement of the mouse over the Server's start node the code snippet displays the start function's content (2.2.2.4). This functionality enables John to dive deeper into the mechanics of the code while still being able to reference the base code seamlessly. This gives John an efficient form of understanding the code's fundamentals through a visual map of the underlying code's structure.

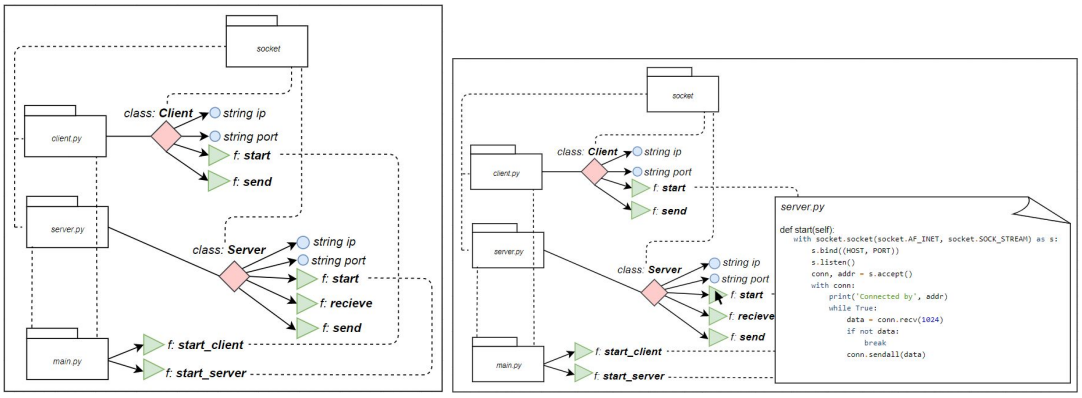


Fig 2.2.2.3 View of the Server - Client system

Fig 2.2.2.4 Code snippet from the Server's start function

2.3 Operational Scenarios

2.3.1 Open Create Modal

Use Case	Open Create Modal
Goal in Context	Launch the "Create" modal.
Preconditions	The user is on the comet home page, which has successfully loaded.
Success End Condition	The Create modal opens successfully.
Failure End Condition	The Create modal does not open and an error message is displayed to the user.
Actors	User & System
Trigger	User clicks the "Create" button.
Description	**Action**
1.	Set the Create modal display state to visible.

2.3.2 Open Upload Modal

Use Case	Open Upload Modal
Goal in Context	Launch the "Upload" modal.
Preconditions	The user is on the comet home page, which has successfully loaded.
Success End Condition	The Upload modal opens successfully.

Failure End Condition	The Upload modal does not open and an error message is displayed to the user.
Actors	User & System
Trigger	User clicks the “Upload” button.
Description	**Action**
1.	Set the Upload modal display state to visible.

2.3.3 Uploading through the Create Modal

Use Case	Uploading through the Create Modal
Goal in Context	Upload the text in the text box to be parsed and analysed in the Create modal.
Preconditions	The user has opened the Create modal. The user has entered some text in the text box.
Success End Condition	The text in the textbox is uploaded and parsed successfully by the system.
Failure End Condition	The text in the textbox is not uploaded successfully or an error occurs when parsing. An error is displayed to the user.
Actors	User & System
Trigger	User clicks the “Upload” button.
Description	**Action**
1.	The text in the text-box is parsed and formatted into JSON.
2.	The JSON data is sent to the server-side API.
3.	The server parses and validates the text, then calculates the relevant metrics.

4.	The server returns the calculated data to the client.
5.	The user is notified of the successful upload and parsing of the text.
6.	The user is redirected to the Metrics Hub.

2.3.4 Uploading through the Upload Modal

Use Case	Uploading through the Upload Modal
Goal in Context	Upload the selected file to be parsed and analysed in the Upload modal.
Preconditions	The user has opened the Upload modal. The file(s) to be uploaded have been selected.
Success End Condition	The file(s) are uploaded and parsed successfully by the system.
Failure End Condition	The file(s) are not uploaded successfully or an error occurs when parsing. An error message is displayed to the user.
Actors	User & System
Trigger	User clicks the “Upload” button.
Description	**Action**
1.	The file(s) are queued for upload.
2.	The file(s) are sent to the server-side API.
3.	The server parses and validates the files, then calculates the relevant metrics.
4.	The server returns the calculated data to the client.
5.	The user is notified of the successful upload and parsing of the file(s).
6.	The user is redirected to the Metrics Hub.

7.	The client-side application creates the various graphs and arranges them on the Metrics Hub.

2.3.5 Open the Metrics Hub

Use Case	Open the Metrics Hub
Goal in Context	Display an initial summary of the code that has been uploaded.
Preconditions	The user has uploaded some code for analysis. The server has computed the relevant metrics. The user has been redirected to the Metrics Hub.
Success End Condition	The metrics summary for all code uploaded is displayed in the Metrics Hub.
Failure End Condition	An error occurs. Metrics are not displayed and an error message is displayed to the user.
Actors	User & System
Triggers	The user uploads code for analysis and the relevant metrics are computed and returned to the client.
Description	
Step	**Action**
1.	The client-side application creates the various graphs and arranges them on the Metrics Hub.

2.3.6 Selecting a File to Analyse

Use Case	Selecting a File to Analyse
Goal in Context	Change the file for which the metrics and analytics are being displayed on the Metrics Hub by selecting a new file from the file tree.

Preconditions	The user has uploaded some files. The Metrics Hub has loaded.
Success End Condition	The metrics for the selected file are calculated and displayed.
Failure End Condition	An error occurs. The displayed metrics remain unchanged and an error message is displayed to the user.
Actors	User & System
Trigger	The user clicks on a different file in the file tree.
Description	
Step	**Action**
1.	The newly selected file is communicated to the server.
2.	The server calculates the metrics for the selected file and returns them.
3.	The client-side application creates the various graphs and arranges them on the Metrics Hub.

2.3.7 Opening the Tree Graph

Use Case	Opening the Tree Graph
Goal in Context	Change the main display of the Metrics Hub from the Heatmap and Metric Cards to the Tree Graph.
Preconditions	The user has uploaded some files. the Metrics Hub has loaded. The user is on the Heatmap page.
Success End Condition	The Heatmap and metric cards are hidden. Tree Graph is displayed.
Failure End Condition	An error occurs. The Tree Graph is not displayed and an error message is displayed to the user.

Actors	System, User
Trigger	User clicks Tree Graph button in Metrics Hub toolbar.
Description	
Step	**Action**
1.	Obtain tree graph information from server-side API.
2.	Hide the Heatmap and Metric cards.
3.	Change the “Tree Graph” button to the “Heatmap” button.
4.	Generate and display the Tree Graph.

2.3.8 Opening the Heatmap

Use Case	Opening the Heatmap
Goal in Context	Change the main display of the Metrics Hub from the Tree Graph to the Metric Cards and Heatmap.
Preconditions	The user has uploaded some files. the Metrics Hub has loaded. The user is on the Tree Graph page.
Success End Condition	Tree Graph is hidden. The Heatmap and metric cards are displayed.
Failure End Condition	An error occurs. The Heatmap is not displayed and an error message is displayed to the user.
Actors	System, User
Trigger	User clicks Heatmap button in Metrics Hub toolbar.
Description	
Step	**Action**
1.	Obtain Heatmap information from

	server-side API.
2.	Hide theTree Graph.
3.	Change the “Heatmap” button to the “Tree Graph” button.
4.	Generate and display the Heatmap and Metric Cards.

2.3.9 Displaying a Tree Graph Node’s Summary Card

Use Case	Displaying a Tree Graph Node’s Summary Card
Goal in Context	Open a node’s metric summary card.
Preconditions	The user has uploaded some files. the Metrics Hub has loaded. The user is on the Tree Graph page.
Success End Condition	The chosen node’s summary card is displayed.
Failure End Condition	An error occurs. The chosen node’s summary card is not displayed. An error message is displayed to the user.
Actors	System, User
Trigger	User mouses-over a tree graph node.
Description	
Step	**Action**
1.	Retrieve node metrics and information from the server-side API.
2.	Set specified node’s summary card display state to visible.
3.	Update the information displayed on the card.

2.4 Constraints

2.4.1 Accessibility

Users who have disabilities should not be alienated from comet's usage by a failure of comet to supply those who need extra utilities to explore the webpage. comet is required to comply with the Web Content Accessibility Guidelines (W3, 2018, https://www.w3.org/TR/2018/REC-WCAG21-20180605/) applying the concept of accessible design and practice of accessible development.

2.4.2 Usability

Since our goal was to create a system that would provide users with a seamless and educational resource, it is imperative that the system is easy to navigate, use, understand and should require minimal effort and previous knowledge from the user.

2.4.3 Respect for Standards

The Python API must be implemented to adhere to PEP8 (Python, 2013, https://www.python.org/dev/peps/pep-0008/) rules and the Javascript React Web Application must follow the style guides set out by Airbnb (Airbnb React/JSX Style Guide, 2019, https://github.com/airbnb/javascript/tree/master/react). Documentation of style guides in relation to components, libraries and other external resources should be followed.

2.4.4 Performance, Throughput & Response Times

While the expected wait time for many functions cannot be accurately estimated presently a user should not be expected to wait an inordinate amount of time for the execution of a simplistic task, such as the opening of a modal. Response times are of vital importance in the retention of a user's attention and the archival of their goal. Analysis wait times will depend on the size of the code base, however wait times must have a balanced correspondence between code base and execution time, ie : therefore a small code base should not take the same amount of time as a large code base to compute and vice versa.

Operations on large code bases have the potential to become exponentially more expensive to compute, therefore it is essential to produce an algorithm that can efficiently parse and execute metric analysis. The current estimation for acceptable sending, parsing and receiving of a 1000 line file should be less than 3 seconds. This estimation assumes the average Irish internet speed of 15.6mb (akamai, [state of the internet],

https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf, 2017) and also the location of the AWS server is within proximity of Ireland (EU). Assuming a request takes between 10ms and 100ms to reach both a user or the server, then comet will have an execution range of 2.8s - 2.9s to analysis a 1000 line file. User's with larger latency may have slightly longer wait times.

3. Functional Requirements

3.1 Client-Side Requirements

3.1.1 Website Landing Page

3.1.1.1 User Interface

- Description: The website landing page will contain two buttons, "Create" and "Upload".
- Criticality: High. A website landing page will be essential, especially in the early versions prior to the IDE plugin implementation of comet.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:

3.1.1.2 "Create" Button Functionality

- Description: Upon clicking the "Create" button, the Create modal dialog will be displayed.
- Criticality: Medium. It is not considered absolutely necessary to comet's functionality to provide a Create modal.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
    - 3.1.1.1 User Interface - The "Create" button must exist for its functionality to be implemented.

3.1.1.3 "Upload" Button Functionality

- Description: Upon clicking the "Upload" button, the Upload modal dialog will be displayed.
- Criticality: High. comet must provide a way for users to upload their code for analysis.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:

    - 3.1.1.1 User Interface - The “Upload” button must exist for its functionality to be implemented.

3.1.2 Create Modal

3.1.2.1 User Interface

- Description: When the Create modal dialog is launched, the user will first be presented with a dialog where they will pick the file type of the initial file from a drop down menu. The Create modal dialog will then be displayed in the centre of the window, over the website landing page. The modal will contain a textbox for code entry with tabs at the top for each file being created. There will be a button for creating new tabs, a button for closing the Create modal and a button for submitting the created files. Each tab will also have an individual “Close tab” button that will be hidden unless the tab is being moused-over.
- Criticality: High. It is not considered absolutely necessary to comet’s functionality to provide a Create modal; but, if implemented, the Create modal must have a user interface as specified.
- Technical Issues: The implementation of tabs may prove challenging.
- Dependencies:
    - 3.1.1.2 “Create” Button Functionality - The “Create” button must successfully launch the Create modal.

3.1.2.2 “Close” Button Functionality

- Description: Upon clicking the “Close” button, a confirmation dialog will be displayed informing the user that closing the dialog will discard any progress and asking if they wish to continue closing the modal. If yes, the Create modal dialog will be closed, discarding all progress and returning the user to the website landing page. If no, then no change will occur.
- Criticality: High. It is not considered absolutely necessary to comet’s functionality to provide a Create modal; but, if implemented, the Create modal must provide a way to close the dialog.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
    - 3.1.2.1 User Interface - The “Close” button must exist for its functionality to be implemented.

3.1.2.3 “Submit” Button Functionality

- Description: Upon clicking the “Submit” button, a confirmation dialog will be displayed asking the user to confirm their desire to submit the created code.

If yes, the files will be created and sent to the server-side API for analysis. If no errors occur, then a notification message will be displayed informing the user that the upload and parse were successful. The user will then be redirected to the Metrics Hub. If an error occurs, an error message will be displayed, informing the user of the issue. In this case, the user will not be redirected and will remain on the Create modal.

- Criticality: High. It is not considered absolutely necessary to comet's functionality to provide a Create modal; but, if implemented, the Create modal must provide a way to submit any created code.
- Technical Issues: Potential technical issues could arise with sending the files to the server-side API or with error handling.
- Dependencies:
    - 3.1.2.1 User Interface - The "Submit" button must exist for its functionality to be implemented.

3.1.2.4 Textbox Functionality

- Description: Each tab shall have its own corresponding textbox that will allow multi-line text entry.
- Criticality: High. It is not considered absolutely necessary to comet's functionality to provide a Create modal; but, if implemented, the Create modal must provide a method of text-entry for writing code
- Technical Issues: It may prove challenging to correctly implement a separate text box for each individual tab.
- Dependencies:
    - 3.1.2.1 User Interface - The textbox must exist for its functionality to be implemented.

3.1.2.5 "New Tab" Button Functionality

- Description: Upon clicking the "New Tab" button, the user will be prompted with a dialog asking them to specify the file type of the new file. Once specified, a new tab will be added to the tab list and the active tab will be changed to this new tab, with a blank text box being displayed.
- Criticality: Medium. It is not considered absolutely necessary to comet's functionality to provide a Create modal; but, if implemented, it is desirable that the user should be able to utilise tabs to create multiple files.
- Technical Issues: Difficulties may be experienced with the implementation of tab creation.
- Dependencies:
    - 3.1.2.1 User Interface - The "New Tab" button must exist for its functionality to be implemented.

3.1.2.6 “Close Tab” Button Functionality

- Description: When a given tab is being moused-over, the “Close Tab” button for said tab will change from being hidden to being displayed. Clicking this button will cause a prompt to be displayed, asking the user to confirm their desire to close the tab and discard its progress. If yes, the tab will be removed from the tab-list and its textbox discarded. Otherwise, no change will take place.
- Criticality: Medium. It is not considered absolutely necessary to comet’s functionality to provide a Create modal; but, if implemented, it is desirable that the user should be able to delete tabs.
- Technical Issues: Difficulties may be experienced with the implementation of tab deletion.
- Dependencies:
    - 3.1.2.1 User Interface - The “Close Tab” button must exist for its functionality to be implemented.

3.1.2.7 Change/Rename Tab Functionality

- Description: When the user clicks on the tab that is currently selected, the filename as displayed in the tab will become editable. When the user clicks on a tab that is not selected, the active tab will change to the tab that was clicked on.
- Criticality: Medium. It is not considered absolutely necessary to comet’s functionality to provide a Create modal; but, if implemented, it is desirable that the user should be able to change and rename tabs.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies: 3.1.2.1 User Interface - Tabs must exist for this functionality to be implemented.

3.1.3 Upload Modal

3.1.3.1 User Interface

- Description: The Upload modal dialog will be displayed in the centre of the window, over the website landing page. The modal will contain a “Staging Area” for adding, removing and displaying files to be submitted. There will be a button for browsing for files, a button for closing the Upload modal and a button for submitting the uploaded files.
- Criticality: High. comet must provide a way for users to upload their code for analysis.
- Technical Issues: This requirement is not expected to pose any technical issues.

- Dependencies:
    - 3.1.1.2 “Upload” Button Functionality - The “Upload” button must successfully launch the Create modal.

3.1.3.2 “Close” Button Functionality

- Description: Upon clicking the “Close” button, the Upload modal dialog will be closed, returning the user to the website landing page.
- Criticality: High. It is essential that the Upload modal is able to be closed.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
    - 3.1.3.1 User Interface - The “Close” button must exist for its functionality to be implemented.

3.1.3.3 “Submit” Button Functionality

- Description: Upon clicking the “Submit” button, a confirmation dialog will be displayed asking the user to confirm their desire to submit the uploaded code. If yes, the files will be sent to the server-side API for analysis. If no errors occur, then a notification message will be displayed informing the user that the upload and parse were successful. The user will then be redirected to the Metrics Hub. If an error occurs, an error message will be displayed, informing the user of the issue. In this case, the user will not be redirected and will remain on the Upload modal.
- Criticality: High. The Upload modal must provide a way to submit any created code.
- Technical Issues: Potential technical issues could arise with sending the files to the server-side API or with error handling.
- Dependencies:
    - 3.1.3.1 User Interface - The “Submit” button must exist for its functionality to be implemented.

3.1.3.4 “Browse” Button Functionality

- Description: When the “Browse” button is clicked, a file browser dialog will be launched that will allow the user to select one or more files to be added to the “Staging Area”. When the files have been selected, they will be displayed in a grid on the “Staging Area”.
- Criticality: High. The Upload modal must provide a way to select the files to be uploaded.
- Technical Issues: Arranging the selected files on the “Staging Area” may pose some difficulties.
- Dependencies:

- 3.1.3.1 User Interface - The “Browse” button and “Staging Area” must exist for this requirement to be met.

3.1.3.5 “Staging Area” Click-and-Drag Functionality

- Description: The “Staging Area” shall allow the user to click-and-drag one or more files (from a File Explorer, for example) into it. When the mouse click is released, each of the files that were being dragged will be added to the “Staging Area” and displayed in a grid.
- Criticality: Low. The same objective for this requirement is achieved by the “Browse” button. This requirement simply serves as a factor for improving convenience.
- Technical Issues: Working out the mechanics behind implementing click-and-drag may be challenging.
- Dependencies:
  - 3.1.3.1 User Interface - The “Staging Area” must exist for this requirement to be met.

3.1.3.6 Staged Document Removal Functionality

- Description: Each document being displayed in the “Staging Area” will be able to be removed by right-clicking them, bringing up a context menu with one option labelled “Remove”. Clicking this option will remove the file from the “Staging Area” grid.
- Criticality: Medium. This behaviour is desirable, but not strictly necessary.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
  - 3.1.3.1 User Interface - The “Staging Area” must exist for this requirement to be met.

3.1.4 Metrics Hub

3.1.4.1 User Interface

3.1.4.1.1 File Tree

- Description: On the left-hand side of the Metrics Hub, there will be a “File Tree” that displays each of the files that were uploaded for analysis.
- Criticality: High. The File Tree is essential to navigating the available metrics.
- Technical Issues: Maintaining the correct file-folder structure.
- Dependencies:
  - None.

3.1.4.1.2 Toolbar

- Description: At the top of the Metics Hub, there will be a toolbar with two buttons, “Settings”, and “Tree Graph” when the Heatmap is open and “Heatmap” when the Tree Graph is open.
- Criticality: High. The Toolbar is necessary to ensure navigability.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
    - None.

3.1.4.1.3 Metric Cards

- Description: When the Heatmap page is open, there will be four “Metric Cards”, stacked vertically to the left of the Heatmap. These cards will display a loading animation while awaiting metric calculation results, the arrival of which will cause the card to change to a graphical display of the relevant metric.
- Criticality: High. This is an integral feature for comet to deliver.
- Technical Issues: Displaying metric data in some cases may be a challenge.
- Dependencies:
    - None.

3.1.4.1.4 Heatmap

- Description: The Heatmap will be displayed graphically on the right-hand side of the Metrics Hub when the Heatmap page is open. It should consist of a number of segments, each segment initially representing one file being analysed and with a size proportional to the size of said file. The colour of each segment will be dependent on the value of the metric being analysed - initially, cyclomatic complexity.
- Criticality: High. This is an integral feature for comet to deliver.
- Technical Issues: This will be a challenging feature to implement correctly across the board.
- Dependencies:
    - None.

3.1.4.1.5 Tree Graph

- Description: On the Tree Graph page, a Tree Graph will fill the area of the Metrics Hub that is not occupied by the File Tree or the toolbar. The Tree Graph will consist of a number of nodes that are connected by lines, in accordance with the file/class structure being analysed.
- Criticality: High. This is an integral feature for comet to deliver.

- Technical Issues: This will be a challenging feature to implement correctly across the board.
- Dependencies:
    - None.

3.1.4.2 Selecting the Active File

- Description: The user should be able to change the active file for which the metrics being displayed represent by left-clicking on a file in the file tree. This will highlight the file in the file tree, indicating that it is active and included in the metrics. If the file that was clicked on was already active, it will be set to inactive and will not be highlighted or included in the metric calculations.
- Criticality: High. It is essential that the user is able to specify which files that they wish to analyse at a given time.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
    - 3.1.4.1.1 File Tree - The file tree must exist for this requirement to be met.

3.1.4.3 “Settings” Button Functionality

- Description: Clicking the “Settings” button in the Toolbar of the Metrics Hub will display a context menu containing a number of settings available for customisation (to be determined at a later date).
- Criticality: Medium. Desirable as it increases the customizability of the application, but not strictly necessary for the primary functionality of comet.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
    - 3.1.4.1.2 Toolbar - The toolbar and “Settings” button must exist for this requirement to be met.

3.1.4.4 “Tree Graph” Button Functionality

- Description: Clicking the “Tree Graph” button will change the active page to the Tree Graph page, hiding the Heatmap and Metrics Cards and displaying the Tree Graph.
- Criticality: High. It is essential that the user has the ability to view the Tree Graph.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:

    - 3.1.4.1.2 Toolbar - The toolbar and “Tree Graph” button must exist for this requirement to be met.
    - 3.1.4.1.5 Tree Graph - The Tree Graph must exist for this requirement to be met.

3.1.4.5 “Heatmap” Button Functionality

- Description: Clicking the “Heatmap” button will change the active page to the Heatmap page, hiding the Tree Graph and displaying the Heat Map and Metrics Cards.
- Criticality: High. It is essential that the user has the ability to view the Heatmap.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:
    - 3.1.4.1.2 Toolbar - The toolbar and “Heatmap” button must exist for this requirement to be met.
    - 3.1.4.1.4 Heatmap - The Heatmap must exist for this requirement to be met.

3.1.4.6 Displaying Tree Graph Node Summary

- Description: Mousing over a node in the tree graph will display a summary card adjacent to the node with a basic summary of a variety of metrics relating to that node.
- Criticality: Medium. A desirable feature but not strictly necessary for the delivery of the key functionality of comet.
- Technical Issues: The placement and sizing of the summary card may be a challenge.
- Dependencies:
    - 3.1.4.1.5 Tree Graph - The Tree Graph must exist for this requirement to be met.

3.1.4.7 Collapsing/Expanding a Tree Graph Node’s Child Nodes

- Description: Clicking on a node in the tree graph will hide its child nodes if such child notes exist and are visible. If such child nodes exist but are already hidden/collapse, then clicking on the parent node will show/expand these child nodes.
- Criticality: Low. This is mostly an aesthetic requirement and should be considered as additional functionality to be added if possible.
- Technical Issues: This requirement is not expected to pose any technical issues.
- Dependencies:

        - 3.1.4.1.5 Tree Graph - The Tree Graph must exist for this requirement to be met.

3.1.4.8 Navigating the Tree Graph Area

- Description: By clicking and dragging anywhere in the Tree Graph, the user will be able to move the Tree Graph's area both horizontally and vertically. The user may also use the scroll wheel to zoom in and out of the Tree Graph area.
- Criticality: Medium. While not strictly necessary for the functionality of the Tree Graph, this would greatly improve the navigability of the resource and is therefore desirable behaviour.
- Technical Issues: Accurately replicating the desired movement using the input received may be difficult.
- Dependencies:
    - 3.1.4.1.5 Tree Graph - The Tree Graph must exist for this requirement to be met.

3.1.4.9 Navigating the Heatmap

- Description: By clicking on a segment of the heatmap, the scope of the map will be changed to the same level of the segment that was selected and the metrics displayed will represent the selected segment.
- Criticality: Medium. While not strictly necessary for the functionality of the Heatmap, this would greatly improve the navigability and utility of the heatmap and is therefore considered to be highly desired behaviour.
- Technical Issues: Redrawing the heatmap as the scope changes may pose some technical issues.
- Dependencies:
    - 3.1.4.1.4 Heatmap - The Heatmap must exist for this requirement to be met.

3.2 Server-Side Requirements

3.2.1 Receiving Initial Information

- Description: The system must be able to accept JSON requests.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.2 Validation of Information

- Description: The system must utilise Swagger to validate the received requests (3.2.1) meet comet's API standards and as a means of error catching.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.3 Get A File/s Information

- Description: The system must provide the functionality parse the received valid request (3.1.1, 3.1.2) and return the relevant file details, such as: file name, file type, size etc.
- Criticality: High.
- Technical Issues:
- Dependencies

3.2.4 File Hashing

- Description: The system must read a file and generate a hash from the contents.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.5 Cache Request

- Description: The system must request the existence of this hash within the database.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.6 Cache File Validation

- Description: The system must check for the file hash within the hash database.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.7 Fetch Pre-Existing Calculations

- Description: Upon the success of (3.2.6) the system must fetch the pre-existing metrics and treegraph from the correct directory. It must then return this data.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.8 Fetch File Grammar

- Description: Upon the success of (3.2.5) and subsequent failure of (3.2.6) the system must then retrieve the specified file type's grammar from a grammar directory.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.9 Generate Parse Tree

- Description: The system must provide a generic parser functionality that can be passed text and a grammar set (3.2.8) and generate a parse tree from said arguments.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.10 Calculate Metrics

- Description: On the success of (3.2.9), the system must be able to calculate each of the relevant metrics.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.11 Shape Data

- Description: The system must reformat the data into a JSON object with all relevant metric and tree graph information.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.12 Update Database with New Data

- Description: Upon the failure of 3.2.4 and the success of all subsequent requirements the system must update the file database with the pre-computed hash.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.13 Update Metrics and Tree Graph Directory

- Description: The shaped data (3.2.9) should be stored in a JSON object within the file directory.
- Criticality: High.
- Technical Issues:
- Dependencies:

3.2.14 Return Data

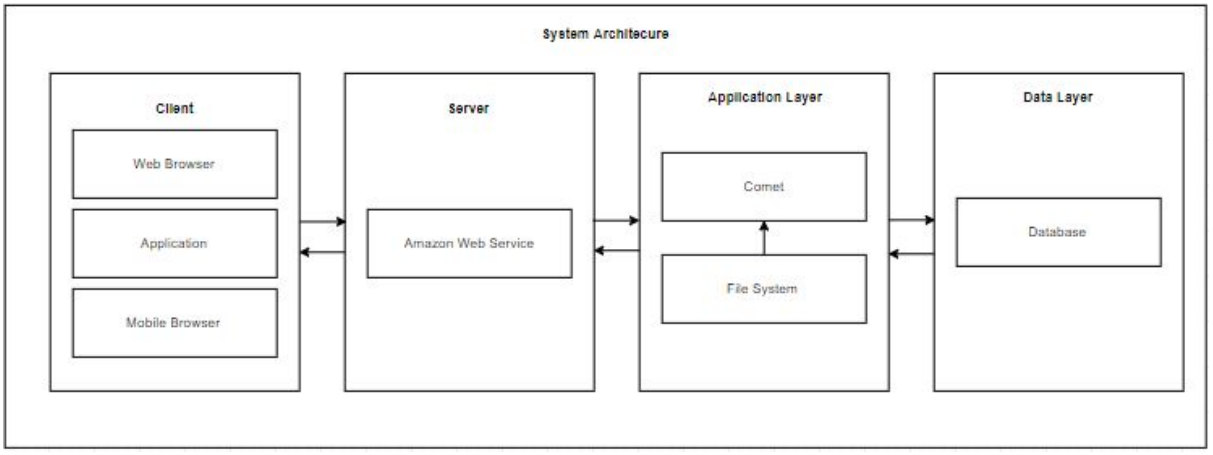
- Description: The system must reformat the data into a JSON object with all relevant metric and tree graph information.
- Criticality: High.
- Technical Issues:
- Dependencies:

4. Non-Functional Requirements

4.1 Client-Side Requirements

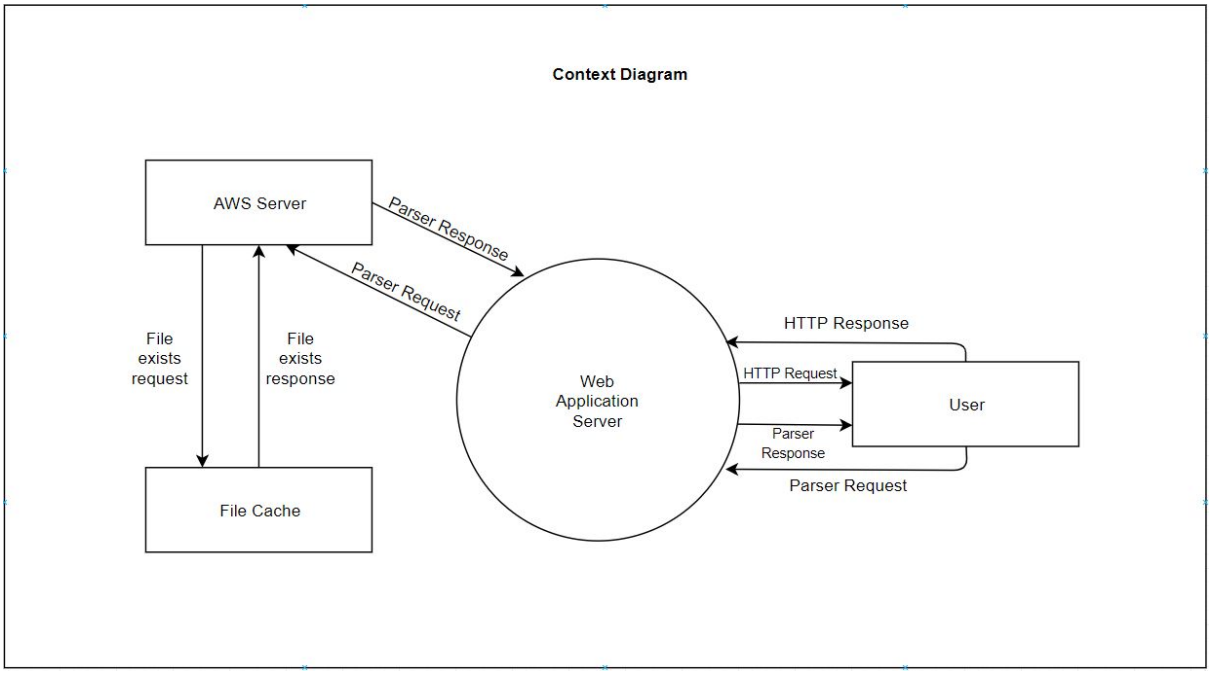
4.2 Server-Side Requirements

5. System Architecture

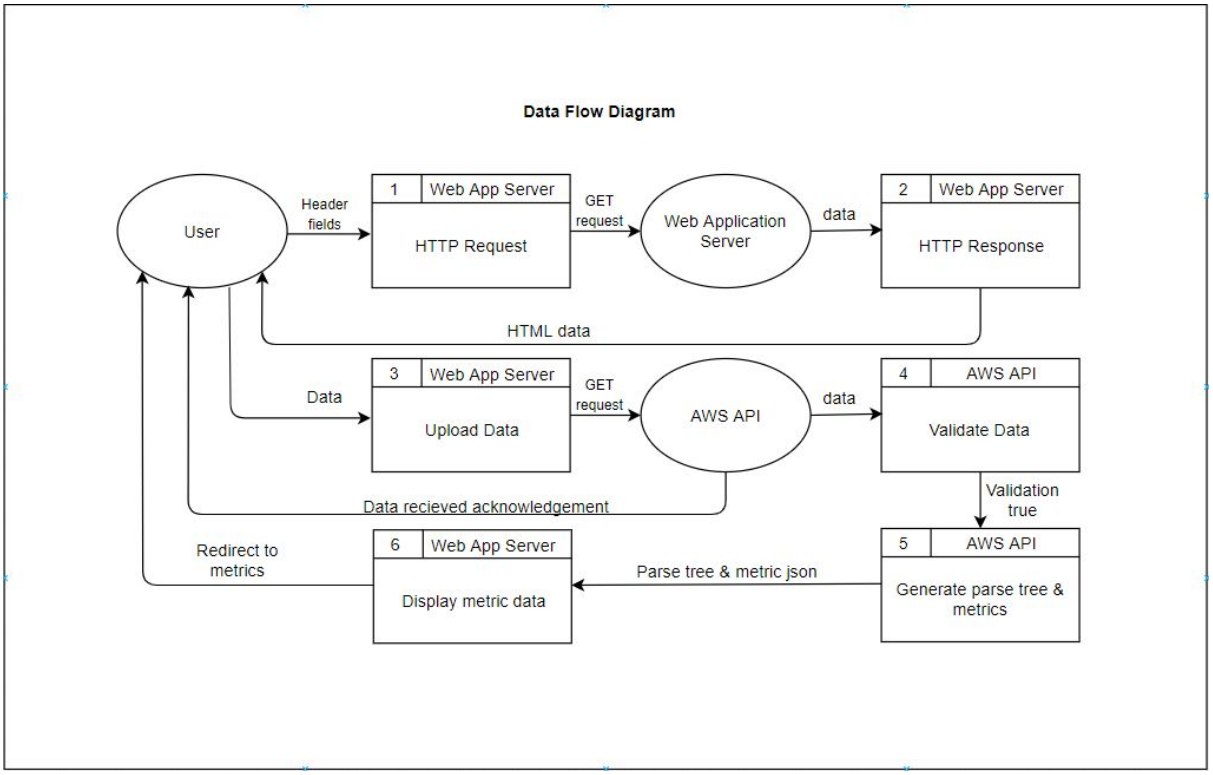


6. High-Level Design

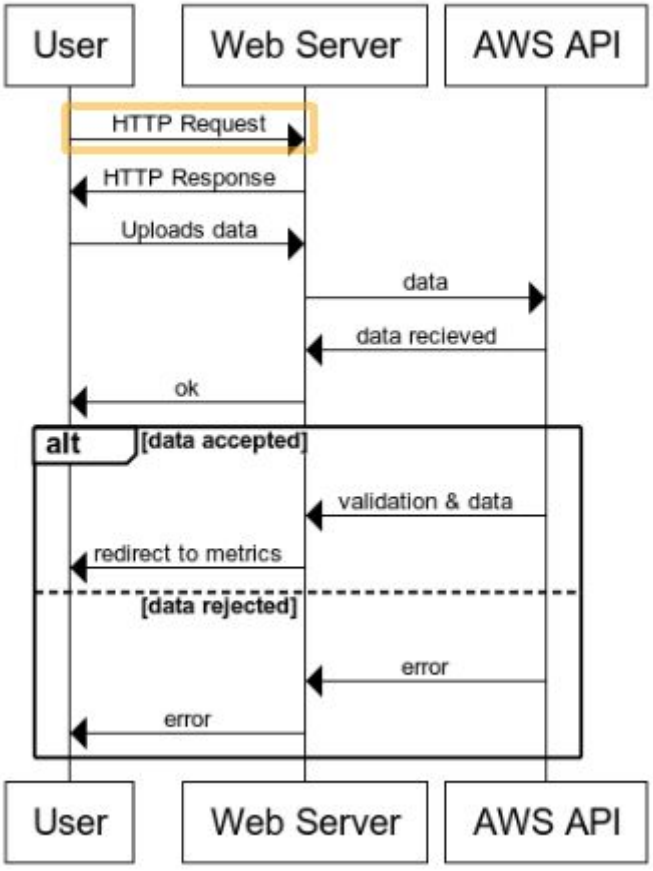
6.1 Context Diagram



6.2 Data Flow Diagram

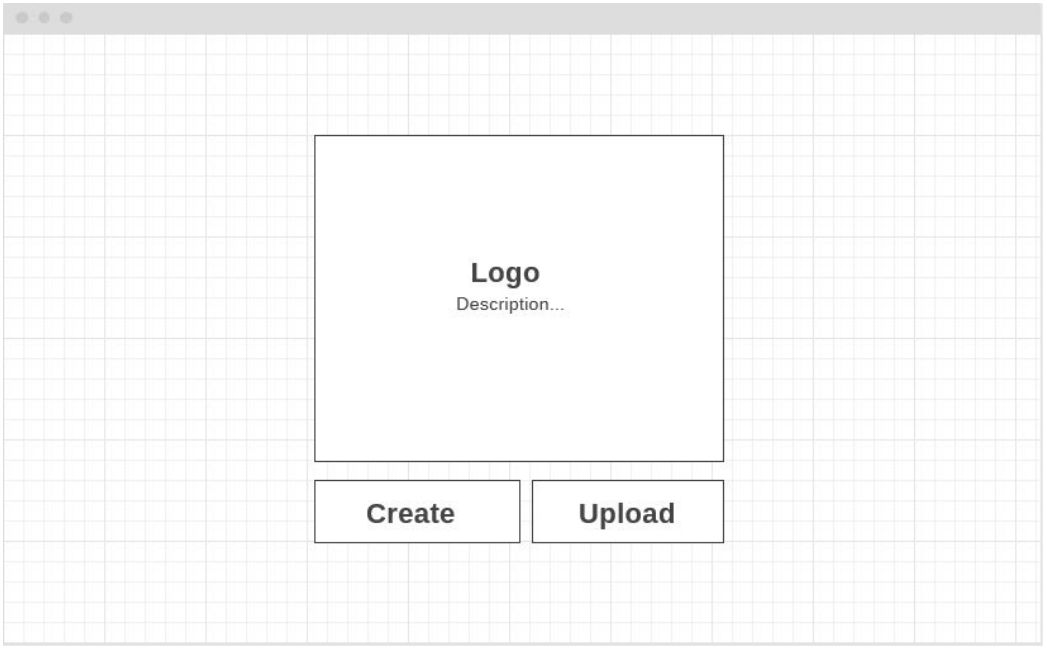


6.3 Sequence Diagram

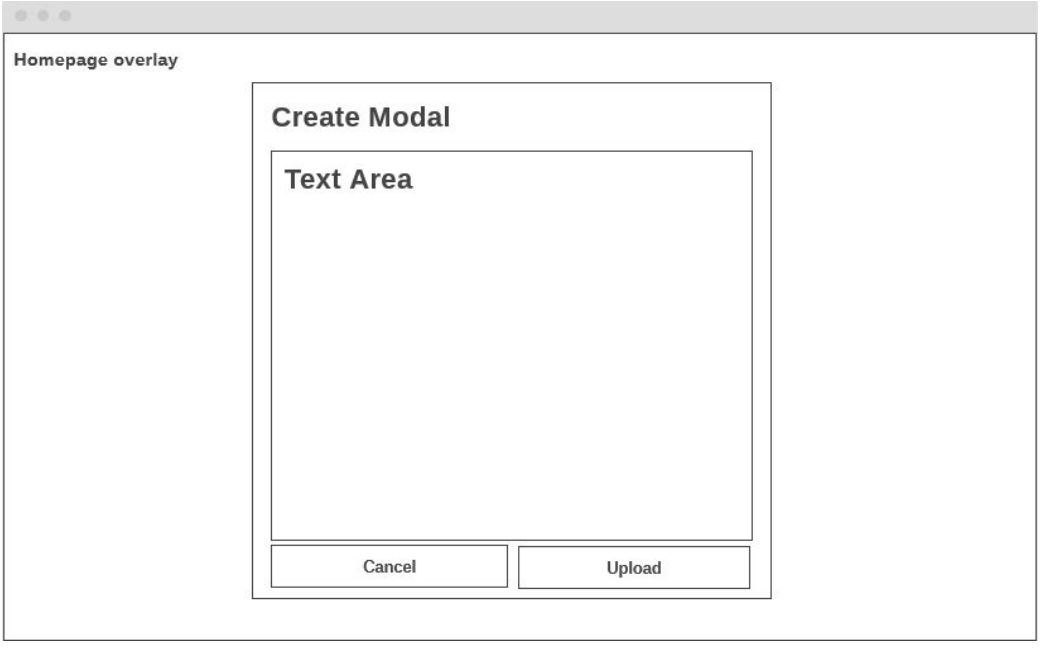


6.4 User Interface Mock-Ups

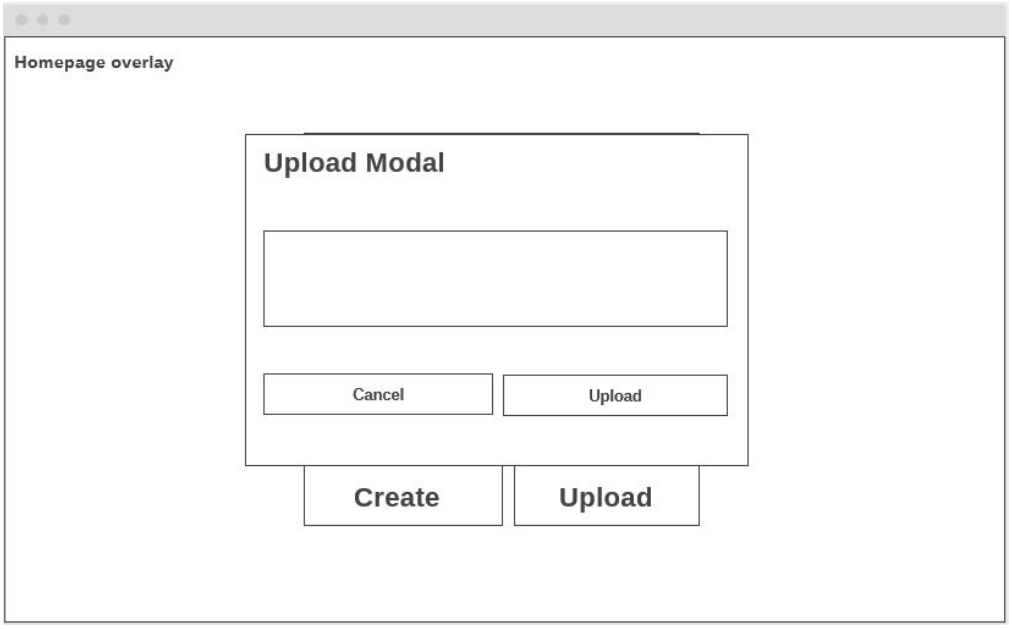
6.4.1 Website Landing Page



6.4.2 Create Modal

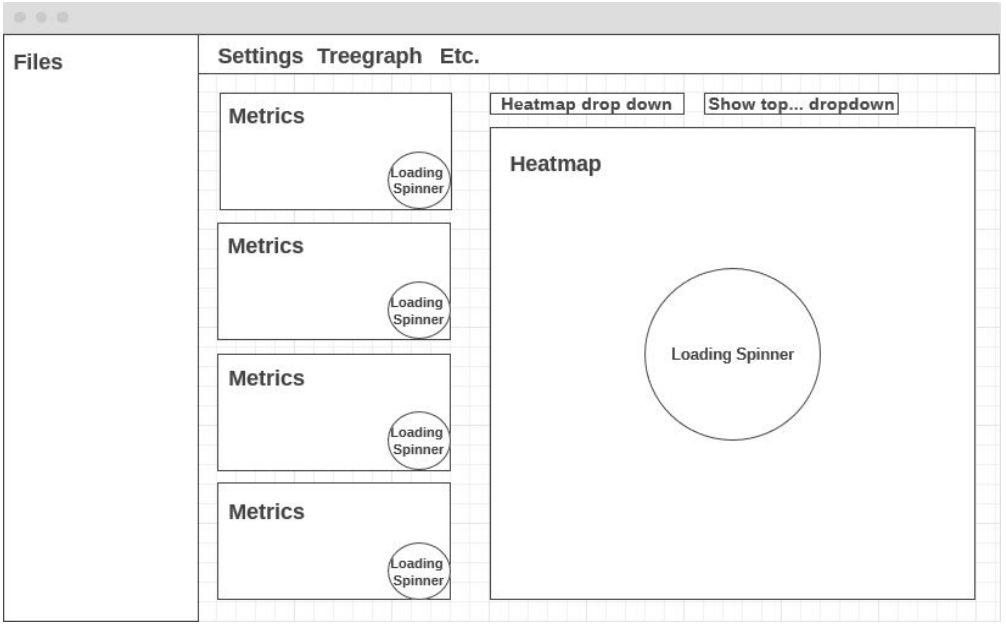


6.4.3 Upload Modal

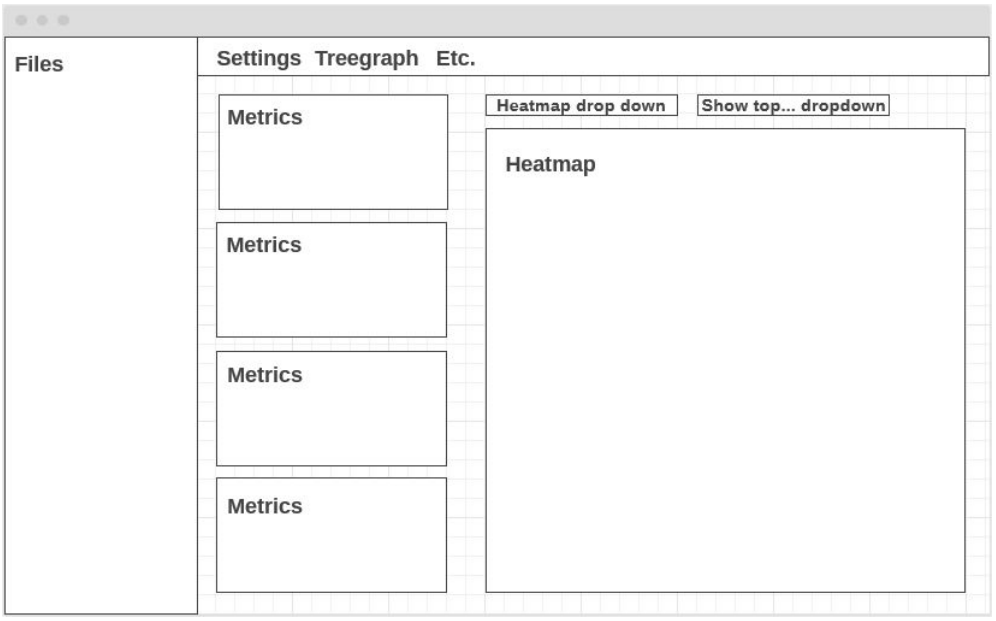


6.4.4 Metrics Hub

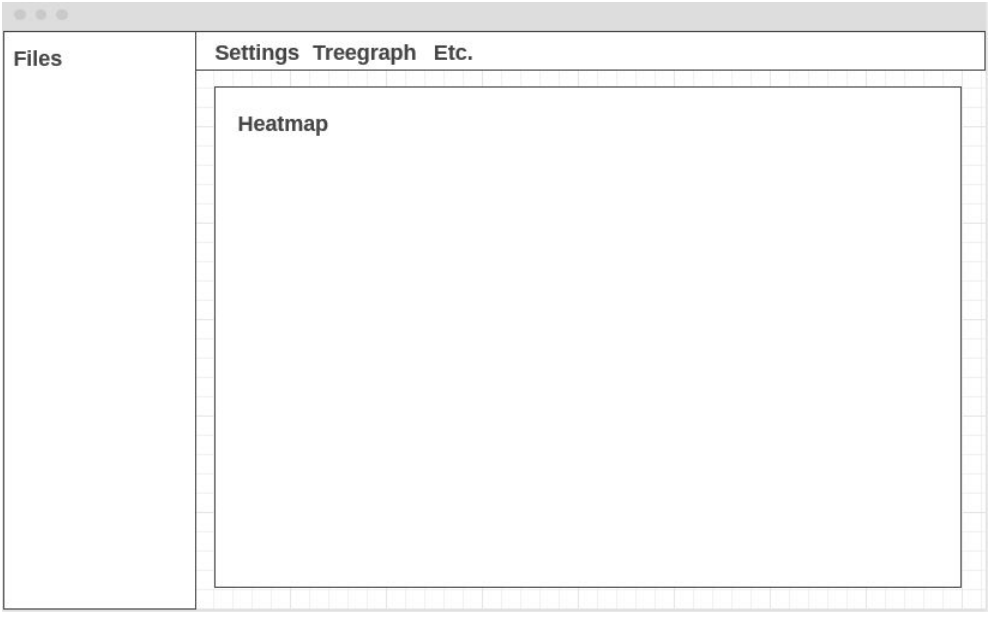
6.4.4.1 Heatmap (Loading)



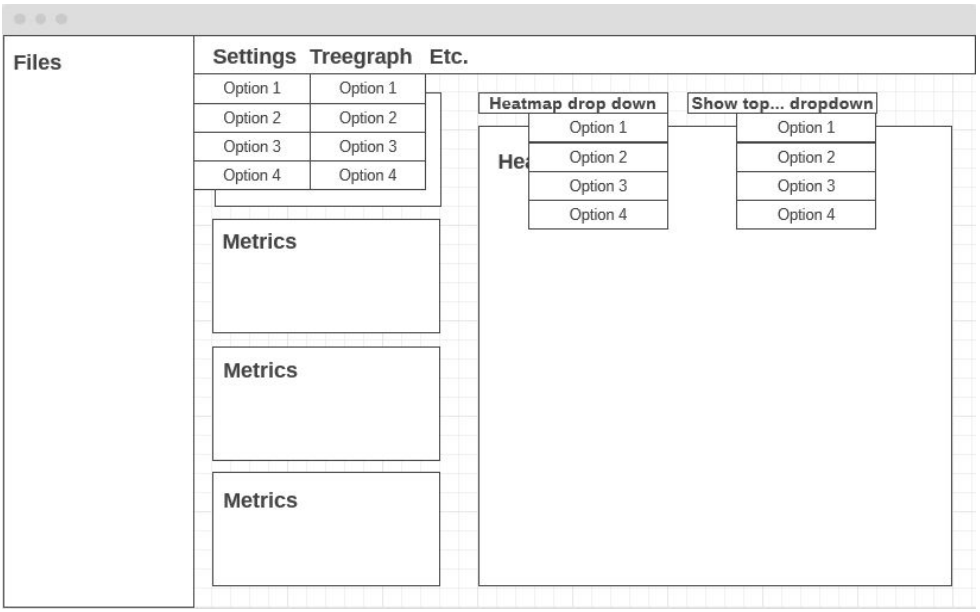
6.4.4.2 Heatmap (Loaded)



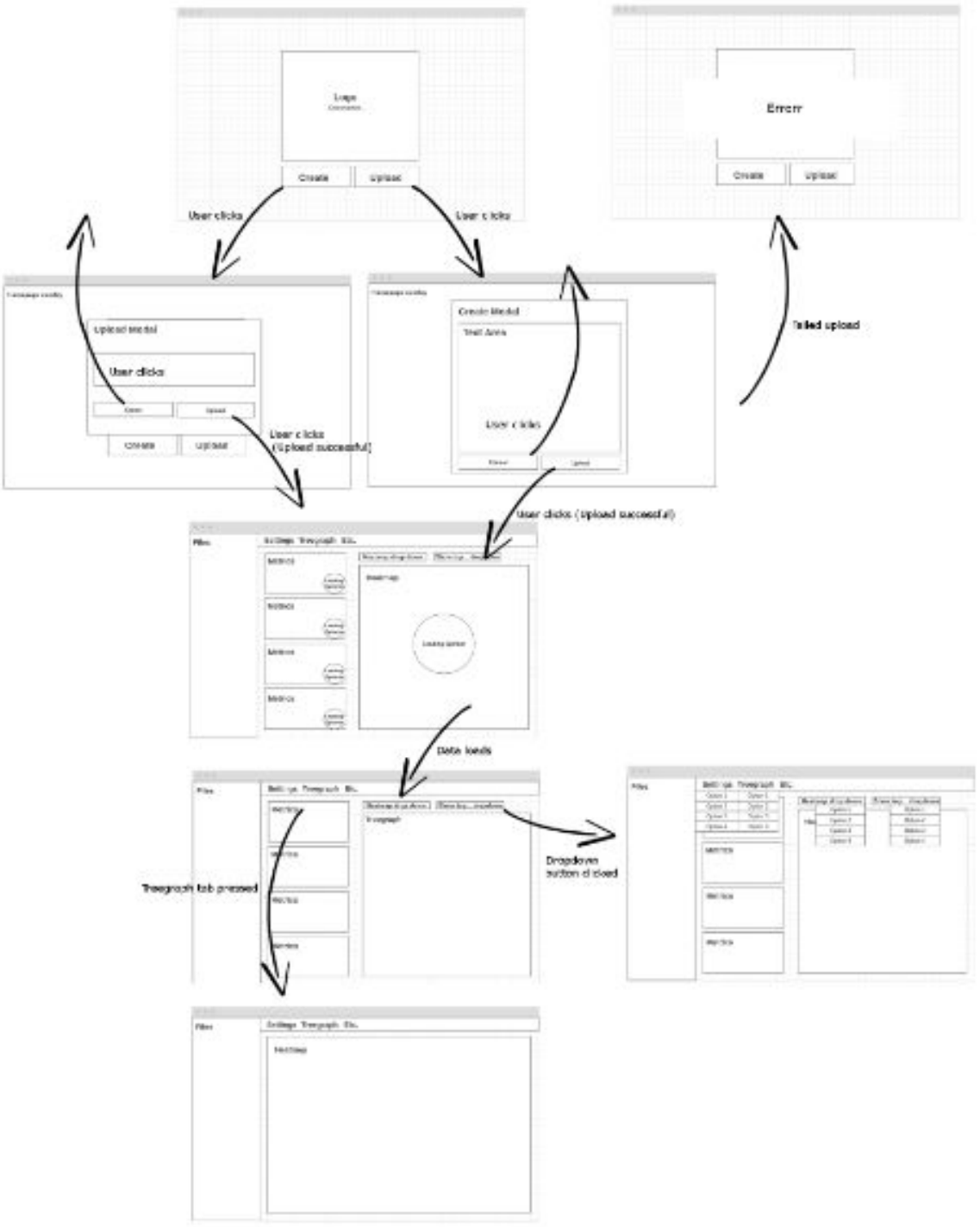
6.4.4.3 Tree Graph



6.4.4.4 Context Menus



6.5 User Flow Diagram



7. Development Plan

7.1 Epics

7.1.1 Initial Setup

- React boilerplate setup
    - Reducers, state, tests & basic setups
- Flask boilerplate setup
    - Simple API to return hardcoded data
- Amazon Web Services setup
    - Must be able to run Python 3.7 files

7.1.2 Homepage

- Basic API
    - Django/Flask
    - Swagger used to validate requests
    - I should be able to send a request to the AWS and receive a response.
    - Temporary hard coded information should be returned from the AWS.
- Homepage
    - Should adhere to the wireframe UX designs. A title, subtitle, logo and two buttons should be displayed.
- Simple error/success alerts
    - On the success or failure of a user action - I should have the ability to display this information to them in an alert form.
- Upload Modal
    - I should be able to open a modal that allows a user to upload files. The modal should be opened by the click of the Upload button on the Homepage.
    - I should be able to receive alerts whether they be standard, success or error within the modal.
    - When the “Upload” button is pressed, it should read and format the file data before being passed to an API to hit the AWS.
    - If I fail to upload I should be displayed an error message.
    - If I succeed to upload I should be redirected to a loading webpage.
- Create Modal
    - I should be able to open a modal that provides a user with a text area to place their data within.
    - I should be able to receive alerts whether they be standard, success or error within the modal.

    - When the “Upload” button is pressed, it should read and format the file data before being passed to an API to hit the AWS.
    - If I fail to upload I should be displayed an error message.
    - If I succeed to upload I should be redirected to a loading webpage.
- Loading Webpage
    - Upon the successful uploading of a file, I should be displayed a success alert within the create/upload modal before being redirected to a page that shows a spinning icon while the data loads.

7.1.3 Parser

- Receive information
- Determine file type, language type and any other constants required for the parsing of the data
- Fetch file-types grammar from grammar store
- Pass grammar and data to parser to try and create AST/something
- If it fails, return fail to Web Server
- If success, calculate metrics and treegraph from the data
- Return data

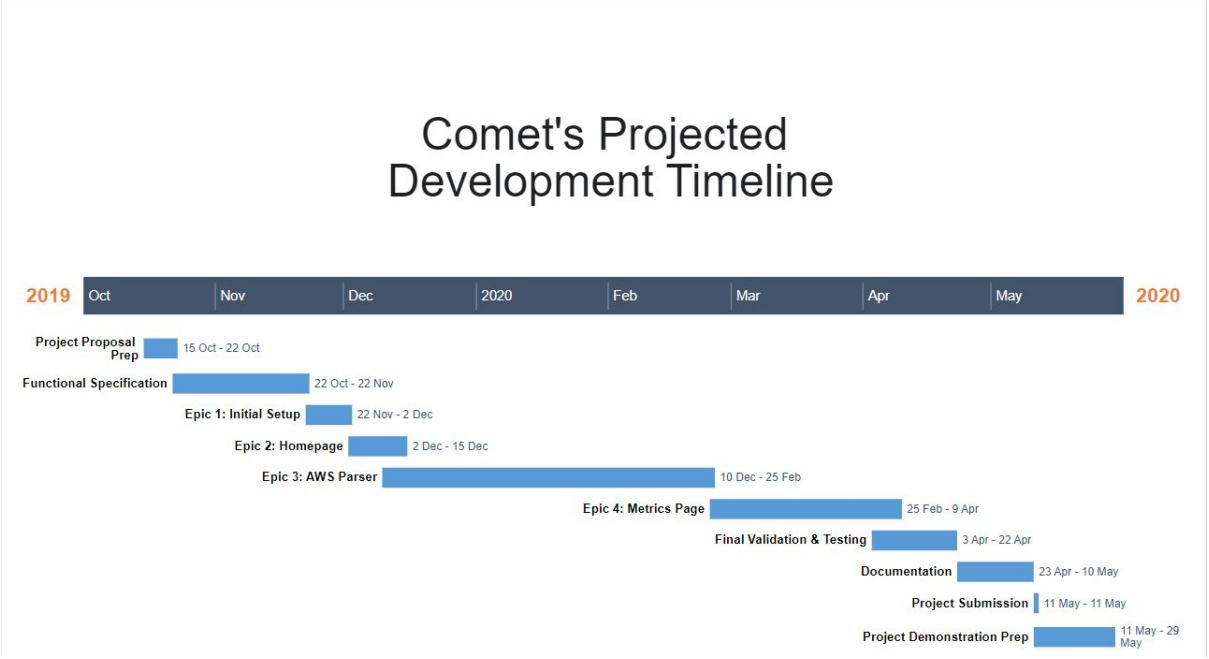
7.1.4 Metrics Page

- TreeGraph
    - Utilising the React Tree Graph library, be able to display the previously received data.
    - Nodes should be displayed in the following form and will have individual colours at a later date:
        - Files: displayed as a file icon
        - Classes: displayed as a diamond
        - Functions: displayed as a square
        - Variables: displayed as a circle
    - Nodes should be connected by a plain visible line
    - On the clicking of a node, it should hide all child nodes if they are not already hidden, otherwise it should reopen the node’s children.
    - Upon **an undecided action** it should open the clicked node’s modal, displaying all relevant information to this node including:
        - Files: Display all file information
        - Classes: Display all class arguments, class methods, class vars
        - Functions: Display all function arguments, function vars and return type
        - Variables: var type, name and information
- File/s Section

    - Adhere to wireframe designs.
    - I should be able to view a list of the file/s that have been uploaded by me. Clicking a file should redirect to a new metrics page with information relating to that file.
- Toolbar
    - I should be able to click the settings icon on the toolbar which should reveal a dropdown. **(need to talk about what goes within this)**
    - The toolbar should contain two tabs - a Metrics tab and a TreeGraph tab. Upon the clicking of the Metrics tab I should be displayed all information relating to the selected file (LOC, Types, Heatmap and other components defined in wireframe designs). Upon the clicking of the Treegraph tab I should be displayed the file's generated TreeGraph component.
- Lines of Code Card
    - I should be displayed a card which provides LOC information. The amount of lines within the selected file should be displayed and a LOCC as a percentage should also be displayed.
    - A small "Display on Heatmap" button should be displayed within this card. Upon the clicking of this, the heatmap should display the LOC heatmap for this file/s.
- Types Card
    - I should be displayed each type used within the file with the count of said type displayed beside it.
    - A small "Display on Heatmap" button should be displayed within this card. Upon the clicking of this, the heatmap should display the types heatmap for this file/s.
- Heatmap
    - This should display a divided heatmap determined by the Heatmap's Level dropdown selection (Files, Classes, Functions, Types) and by the Selected Metric's dropdown (Cyclomatic Complexity, LOC and so forth).
    - When a section of the Heatmap is clicked, this region should become selected. If a selected tile is clicked, it should become unselected.
    - Upon right clicking of a section a context menu should be displayed giving the options: Properties, Select, Unselect, Jump to Code)
    - Properties upon click should display a modal with all relevant property information to the selected section.
    - Select should select a section.
    - Unselect should deselect a section if applicable.
    - Jump to Code should display a modal displaying the code of the selected region.

- Heatbar
    - A key displayed beside the Heatmap to convey to users the reasons behind each section's colour.
- Heatmap Dropdown
    - Upon clicking of this component, a dropdown should be supplied containing: Cyclomatic Complexity, LOC and ……..
    - The clicking of a dropdown item should update the Heatmap to display the selected heatmap information.

7.2 Preliminary Schedule



8. Appendices