# Functional Specification for START Web Application


## Adam Gray (20364103) & Niall Kelly (20461772)


8/11/2023

## 0. Table of contents


1. Introduction
    - Overview
    - Business Context
    - Glossary
2. General Description
    - Product / System Functions
    - User Characteristics and Objectives
    - Operational Scenarios
    - Constraints
3. Functional Requirements
    - Integrated Development Environment (IDE)
    - Containerized Code Execution
    - Access to Documentation and Videos
    - Debugger
4. System Architecture
    - System Overview
    - Component Interaction
    - System Architecture Diagram

5. High-Level Design
    - Context Diagram
    - Sequence Diagram
6. Preliminary Schedule
    - Gantt Chart
7. Concept Design
    - IDE/Debugger
    - Resource Page
8. Appendices

# 1. Introduction

- **Overview**

We will be developing a web application that will allow new programmers to learn our previously developed programming language, START, on the web in an easily accessible way. Our aim with this project is to make our language, and programming in general, more accessible to an average computer user who may have no previous experience with programming at all. This web application will be developed using a tech stack consisting of Django, Python, HTML, CSS, JavaScript, Docker and some small elements of START as well. The back end will be primarily Python with the front end consisting of HTML, CSS and JavaScript, and this will then use a contained environment to run the user's written code using Docker.

The user's will firstly be able to read about START, as well as having access to a set of informative videos that will explain not only the language itself, but also the concepts of programming in general. The user will then be able to write their own code to solve some simple programming exercises, and test the concepts they have learned. The user will be given an IDE built into the web application, and will be able to write their code in this IDE, and then run it to see the output.

One of the main reasons for developing this web application, after making a more accessible language, to make programming itself for accessible, and as well as this, make an environment that is easy to use and understand, providing a simple to use debugger and will use animations to show the user where they may be making mistakes in their code.

- **Business Context**

Currently there are no plans to monetize our system, as we plan for it to be free to use for all, to make it as accessible as possible.

Potential does exist, like our previous project, for this to become a way of teaching younger audiences how to program, and even have this as a potential seller to institutions and schools, but no such plan is in place as of now, as we believe free software is best for everyone.

- **Glossary**

    - **Django** - Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design

    - **Docker** - Docker is an open platform for developing, shipping, and running applications

    - **Integrated Development Environment (IDE)** - A software application that helps programmers develop software code efficiently. It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application

# 2. General Description

### 2.1 Product / System Functions

Our web application will be a web based version of our programming language START, which is a novice friendly programming language. It will still be targeted to novice programmers, and will be a way for them to learn the language, and programming in general, in a more accessible way, as many people do not know about GitHub, where our language is currently available.

- Our app will be seen on the web, and will be styled using HTML, CSS and JavaScript.
- The user's of the site will be able to read about the languages features, using the documentation we have already written, and will be able to watch informative videos about the language, and programming concepts. This will be a good way to "dip their toes" into programming, and see if it is something that takes their interest before writing more complex code. We will implement these pages using HTML, CSS and JavaScript, while storing the files and videos needed in the back end database.
- The user will then be able to go the the IDE, and write their own code, and run it to see the output. This will be a good way for them to test the concepts learned previously from the videos and documentation. This IDE will be a mostly JavaScript based IDE, with the code being run in a contained environment using Docker. We will take the user's code and pass it to the container where it will be run, the output will be collected, and then sent back to the user for them to see.
- The user will also have access to a debugger that they can use to find errors in their code. This simple debugger will be animated to show the user where they have made an error. This will be a good way for the user to learn how to debug their code, and will be a good way for them to learn how to find errors themselves in the future.

### 2.2 User Characteristics and Objectives

Our project is still targeted at beginner programmers, and as such will be designed to be as accessible as possible. We will be using a simple design for all elements of the site to make programming "less intimidating" for new programmers. After our research into beginner programmers and programming languages earlier this year, as well as the data we collected from testing our language with new programmers, we have came up with a key list of characteristics that our project must follow, these are:

- To make an environment that is not only easy to navigate but easy to use, to make the user feel more comfortable with programming. This will be done to allow for the assumption that some users may have very little experience with computers, and as such, may not be able to navigate a more complex environment.
- To give the user one space to learn about, write and run their code, to cut down any frustration that may come from learning programming on their own.
- To create an easy to use debugger to help the user understand how to look for errors in their code, and to help them learn how to debug their code themselves.
- To provide useful resources for the user to learn about programming concepts in START, and programming in general, to help them learn how to program, as well as cutting down time spent needing to do outside research about topics.

### 2.3 Operational Scenarios

**Scenario 1: User watches some videos**
The user has just entered the site and has decided to watch some videos to introduce themselves to the world of programming. They navigate to the page that contains the videos, and from there they selected a video to watch, from here the video opens and the video is played for the user.

**Scenario 2: User reads some documentation**
The user has entered the site and has decided they want to learn about a certain feature of the language, so they navigate to the page that has the language documents on it. From here they can select the feature they want to learn about, from here the document will open for the user to read.

**Scenario 3: User writes some code**

The user is now ready to write some code, they open the built in web IDE and write some code. As they write code it is correctly formatted and highlighted for them to see. The code is then ready to be ran when the user is ready.

**Scenario 4: User runs their code**

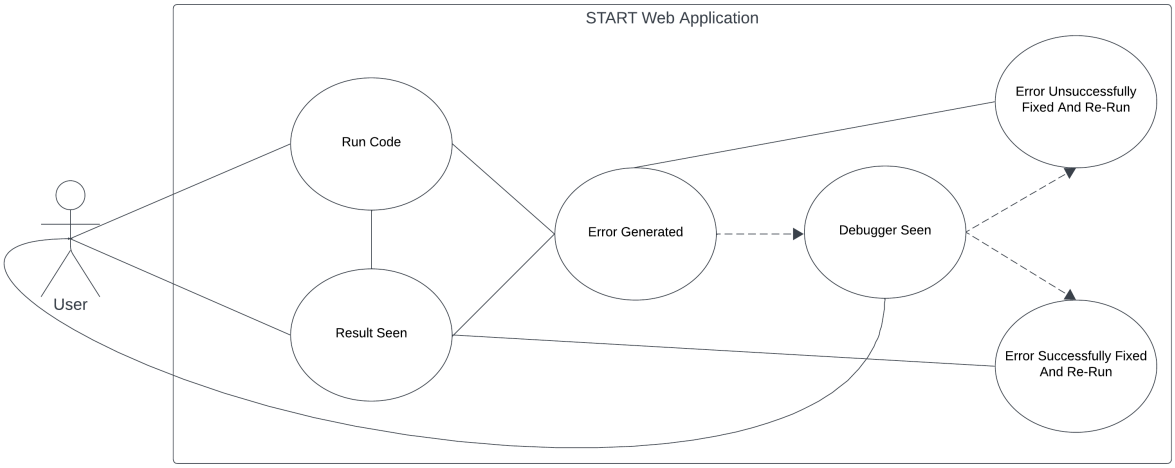
Once the user has watched some videos and read some documentation, they decide they want to try writing some code themselves. They navigate to the IDE page and select a problem they want to try to solve, or even just open the IDE to write some of their own code with no specified problem. From here they will then see the IDE and will be able to write their code seamlessly to solve the problem they have selected. Once the code is written they will be able to run the code directly from the web application, this will then pack up the user code and send it to a docker container to be executed, and the output will then also be packed and returned to the web application for the user to see.

**Scenario 5: User's code error's**

If the user writes some code that contains an error, when they run the code, the Docker Container will execute the code and return to the user an error, as opposed to a result. Once this error is returned, it will be displayed to the user in the output field. This will let the user know what they have done wrong, and suggest how to fix it too.

**Scenario 6: User debugs their code**

If the user has written some code that has some errors in it, they will realize that their code is not working as expected. They will then be able to use the built in START debugger to find the errors. This will be a simple debugger that will be animated to show the user where they have made an error in their code. This will be implemented using primarily JavaScript, to allow us to quickly and easily animate the debugger.




The above Use Case shows how a user would interact with the IDE to run their own written code, and the flow of the system they would see given what happened with their code.

1. If the user writes successful code first try, they will immediately see their desired result, and such will be the end of their flow for that current program.
2. If the user wrote unsuccessful code, they would see an error instead of their result, from there they would then use the debugger to fix their code.
    a. From here if the user successfully fixes their code and re-run's it, they will then see their desired result, as before.
    b. However, if the user fails to fix their code correctly and re-run's the code, they will start the loop again at the error generated stage. From here they would return to the debugger and start to fix their code again. When the user finally fixes their code, they will finally see their desired result.

## 2.4 Constraints

### Time Constraints

Given the size and scope of the project it will be extremely important to keep track of our time at all stages of development to prevent us running out of time towards the end of the project. We have decided to use the agile methodology to develop our project. This will allow us to develop the project in a set of sprints, and will allow us to develop

the project in a more efficient way. This will also allow us to develop the project in a more flexible way, making us less limited to a set of requirements, and will allow us to change the project as we go along, to make it more user oriented.

**Platform Constraints**

Much like our third year project, this project will be developed solely on the Windows operating system. This is due to the fact that we would prefer to develop on the same operating system as the majority of our user's will be using when they use our web application. This will allow us to test the application on the same operating system as the majority of our user's, and will allow us to fix any issues that may arise on this operating system. However, this means we will have limited testing on UNIX based operating system's such as Linux distributions and MacOS. This may cause some issues with the application on these operating systems, but we will try to fix any issues that may arise on these operating systems.

# 3. Functional Requirements

## Integrated Development Environment (IDE)

**Description:**

Users must be able to access an integrated development environment (IDE) within the web application where they can write and run START code. Without this there is no reason for the application to exist, and such this will be the most important feature to tackle first.

**Criticality:**

Critical

**Technical issues:**

- Syntax highlighting
- Ability to run user code in a contained environment (Docker)
- Implementing code execution and return of results to the user interface

**Dependencies with other requirements:**

The IDE is central to the application's purpose, but will require the container to work flawlessly for it to work fully.

## Containerized Code Execution

**Description:**

The web application must provide a secure and isolated environment (Docker container) for executing user-written START code within the integrated development environment (IDE). This is essential to ensure consistent execution environments regardless of the user's machine.

**Criticality:**

Critical

**Technical issues:**

- Containerization using Docker
- Securely passing user code to the container
- Capturing and returning the output to the user

**Dependencies with other requirements:**

Containerized code execution is a core component of the IDE functionality and is essential for a reliable user experience.

## Access to Documentation and Videos

**Description:**

Users must have seamless access to documentation about the START programming language and a comprehensive library of instructional videos explaining programming concepts and language features. This is critical for providing educational resources to users.

**Criticality:**

High

**Technical issues:**

- Developing a user-friendly interface for browsing and searching documentation and videos

- Storing documents and videos efficiently in the backend database
- Ensuring proper categorization of content

**Dependencies with other requirements:**

While not directly relying on any other component of the application, without the IDE being implemented the user will have no place to write code after using these resources.

### Debugger

**Description:**

Users should have access to a simple debugger tool integrated into the IDE, allowing them to identify and fix errors in their code. This feature aids users in learning how to debug their code effectively.

**Criticality:**

High

**Technical issues:**

- Developing a user-friendly debugger interface with animation capabilities
- Integrating the debugger seamlessly with the code editor
- Ensuring the debugger accurately highlights code errors

**Dependencies with other requirements:**

The debugger is essential for helping users learn to debug their code, and it is closely linked to the IDE functionality, and such, cannot be implemented without the IDE working fully first.

## 4. System Architecture

**System Overview:**

Our system architecture for our web application consists of several components that all work together to deliver the final product to the user to use. These components include:

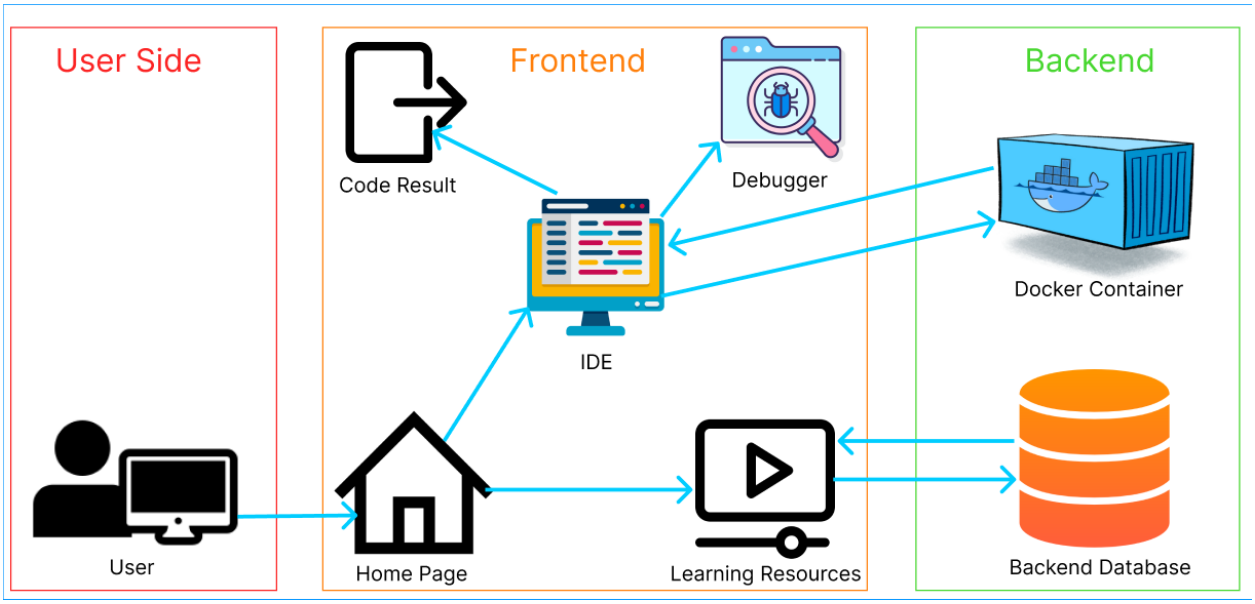
- **Application Frontend:** This is the visual interface that the user's will see and interact with when using the application. Via a web browser such as Chrome or Bing they will see a series of pages built using HTML, CSS and JavaScript, which together form the usable site. The pages include resources to learn and watch, an IDE and a debugger.
- **Application Backend:** The backend handles the user requests as well as delivering content to the frontend of the application. It will be built using Django, a popular Python web framework. The backend will also store all of the content the user will use to learn START such as videos and documentation. These will be stored in Django's database.
- **Containerized Code Execution Environment:** This will be a Docker container that will be used to execute the user's written code. We are choosing this method to provide a consistent environment for all user code to be executed in, regardless of the machine the user is currently using.

**Component Interaction:**

- User interacts with the front end of the web application, from here they can access all pages such as the resources needed to learn, IDE and debugger.
- The documentation and videos are retrieved from the backend database and displayed to the user.
- The IDE allows the user to run their code, which is then passed to the Docker container.
- The Docker container runs the user code within itself and then returns the output to the IDE for the user to see.
- The Debugger is part of the IDE, and helps users to identify and fix errors present within their code.

**System Architecture Diagram:**

The below diagram shows how the components of the system interact with each other when the user is using the application, as discussed in the above sections.




# 5. High-Level Design

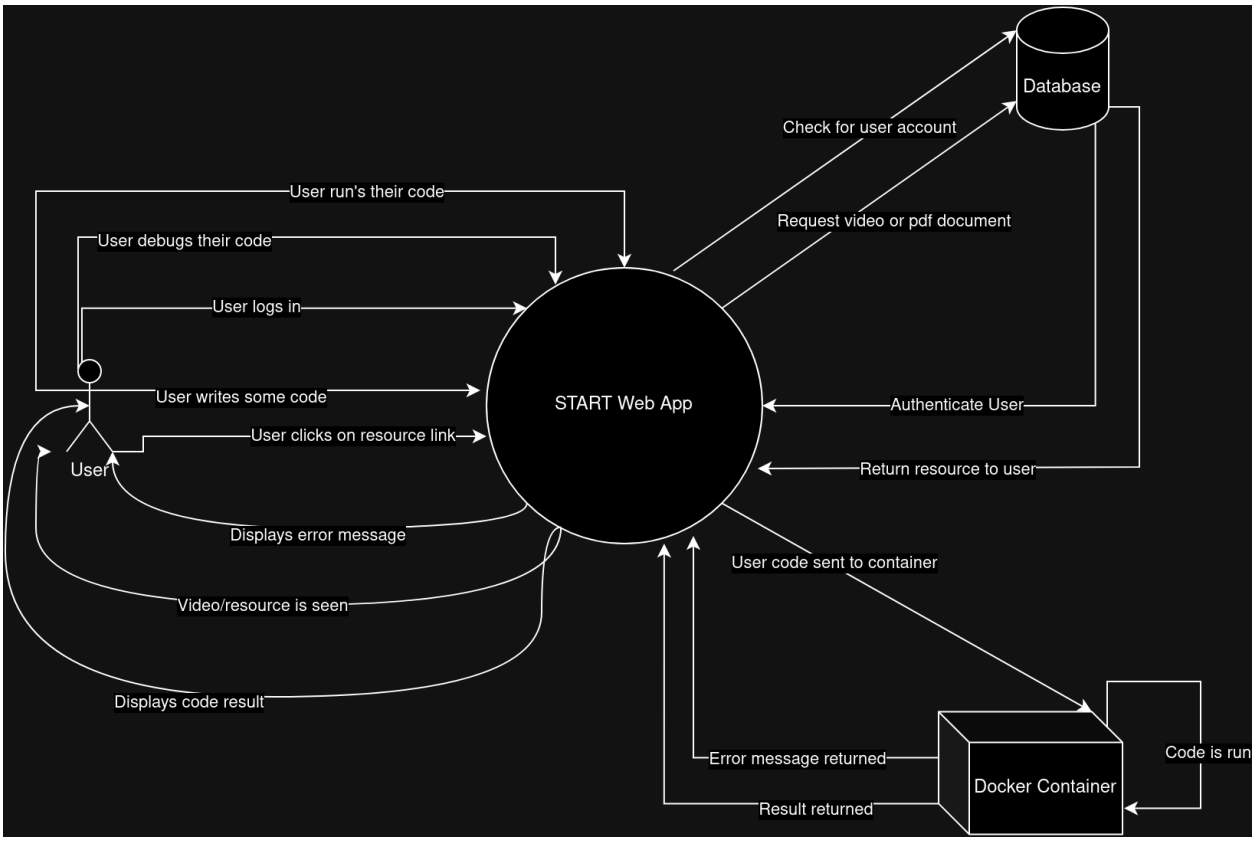
## Context Diagram

The below context diagram shows how the major components of the system as well as the user interact with each other. Given the user preforms an action, there is an appropriate response, for example:

**User logs into the site** - Once the user attempts to log in with their credentials, their information is then sent to the data base to check if the user exists, and if they do they will be authenticated and will continue to have access, otherwise, they will not be allowed access.

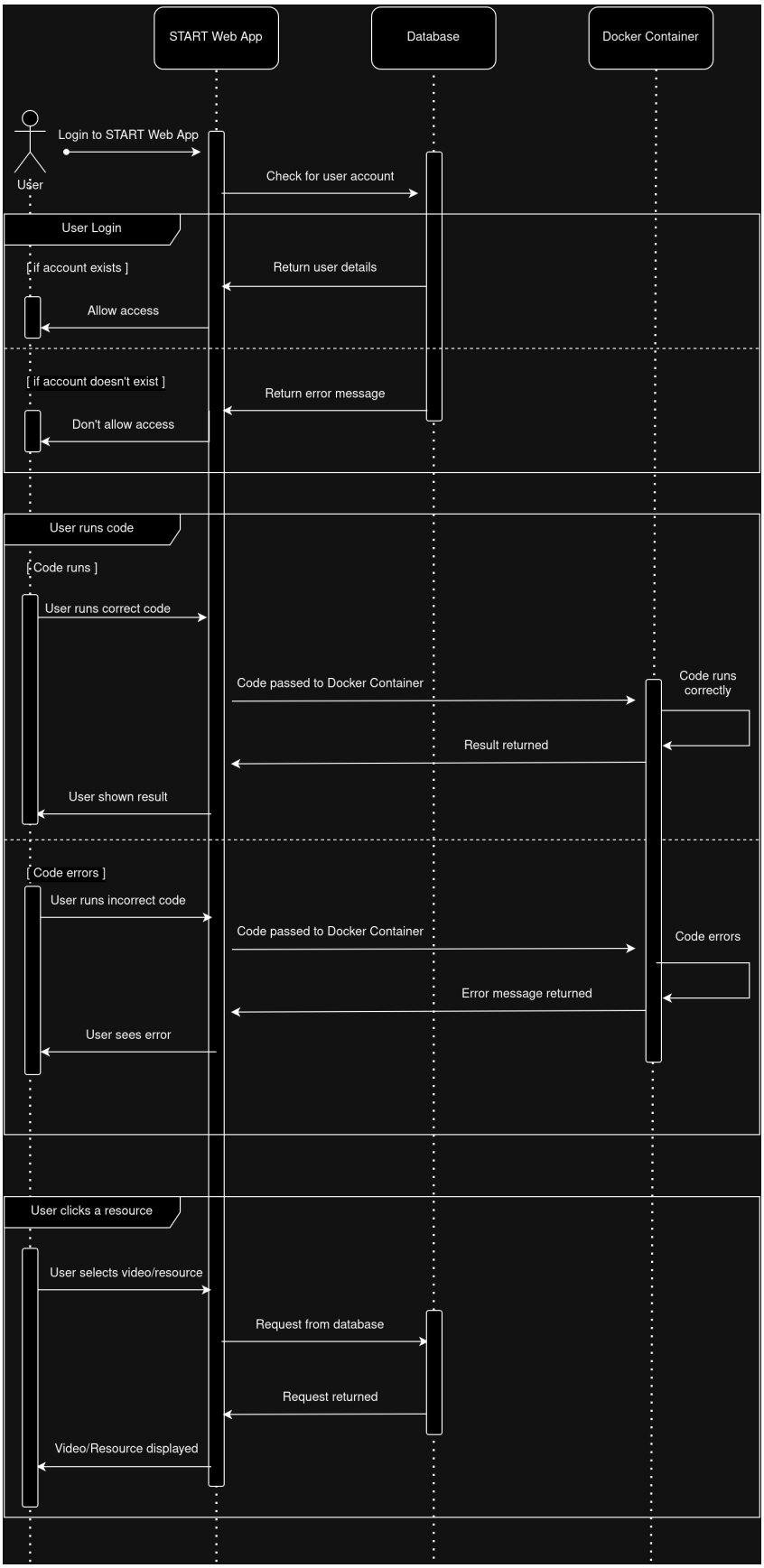
**User runs their code -** When the user runs the code they have written, the code is sent to the docker container to be executed. The code is then run, and if successful, a result is returned to be displayed to the user, however, if the code is incorrect, an error message will be returned to the user instead.

**User clicks on a resource** - If the user clicks on one of the learning resources available on the Web App, their request will be sent to the database, and from there, the resource will be returned and displayed to the user.



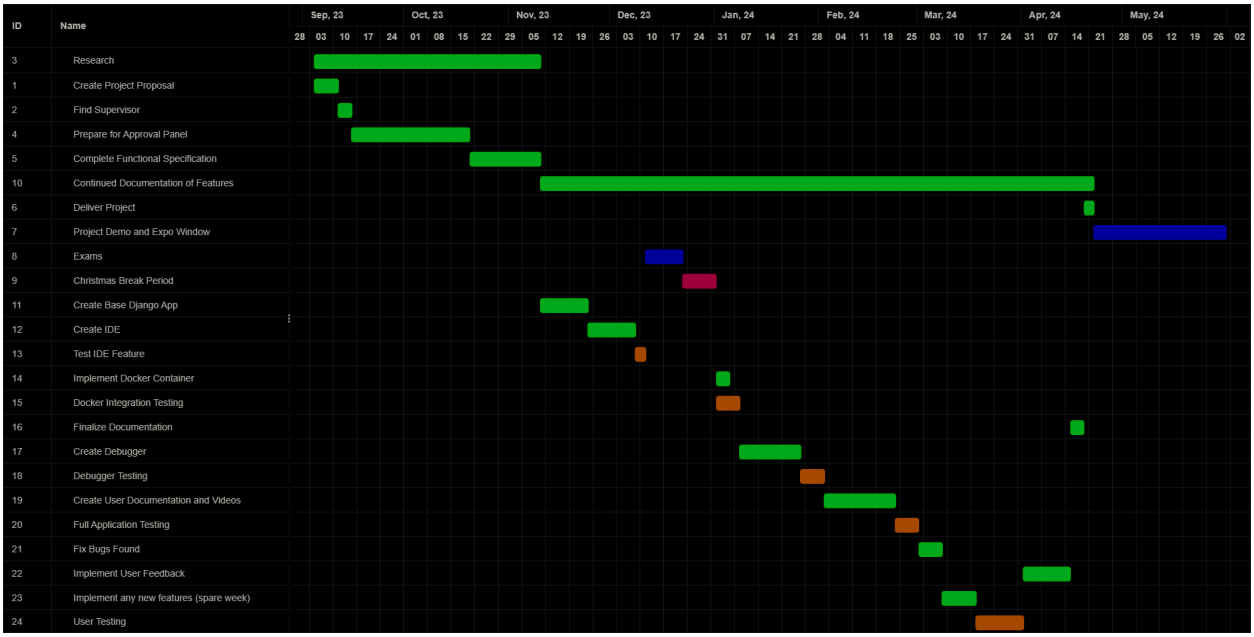

## Sequence Diagram

The below sequence diagram shows the flow of interactions between different components of the system and the user, shown in some use cases for the system.


START Web App
Database
Docker Container
User
Login to START Web App
Check for user account
User Login
[ if account exists ]
Return user details
Allow access
[ if account doesn't exist ]
Return error message
Don't allow access
User runs code
[ Code runs ]
User runs correct code
Code passed to Docker Container
Code runs correctly
Result returned
User shown result
[ Code errors ]
User runs incorrect code
Code passed to Docker Container
Code errors
Error message returned
User sees error
User clicks a resource
User selects video/resource
Request from database
Request returned
Video/Resource displayed

# 6. Preliminary Schedule

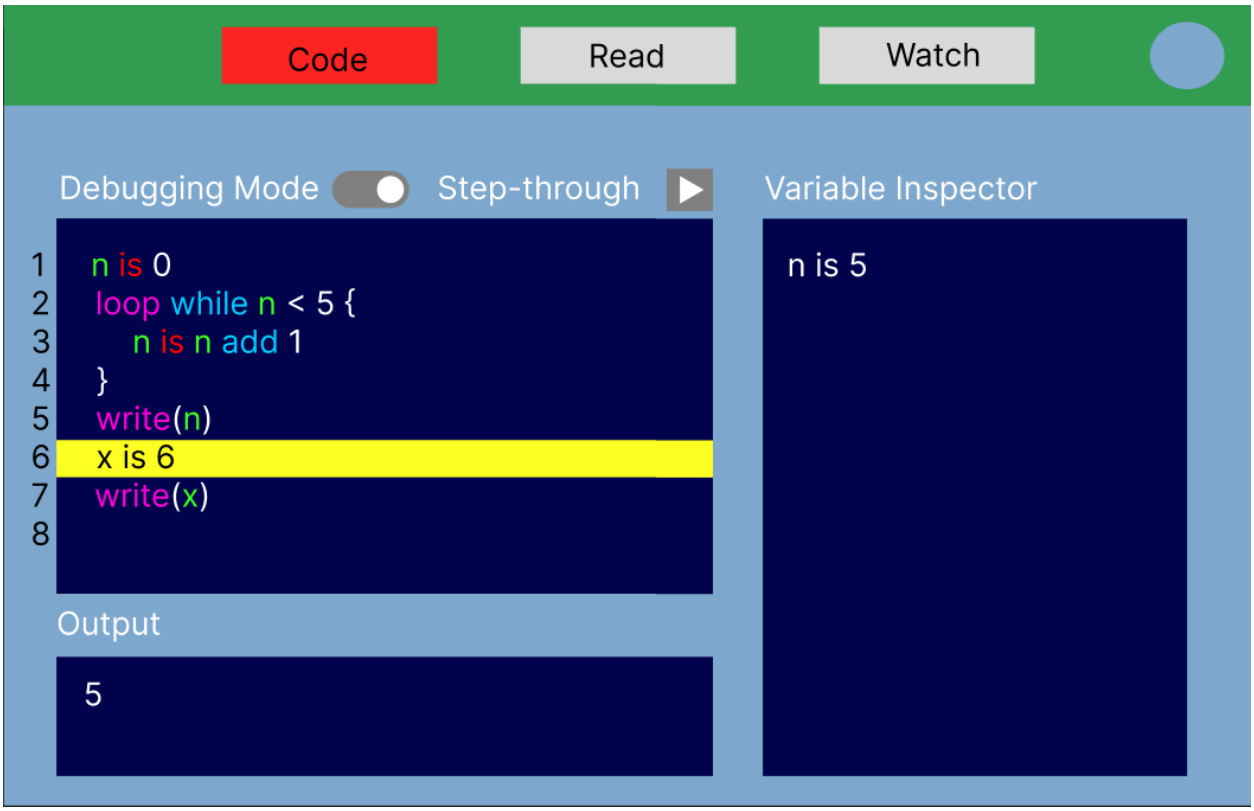
Predicted timeline of our Project progress using a Gantt Chart.




# 7. Concept Design

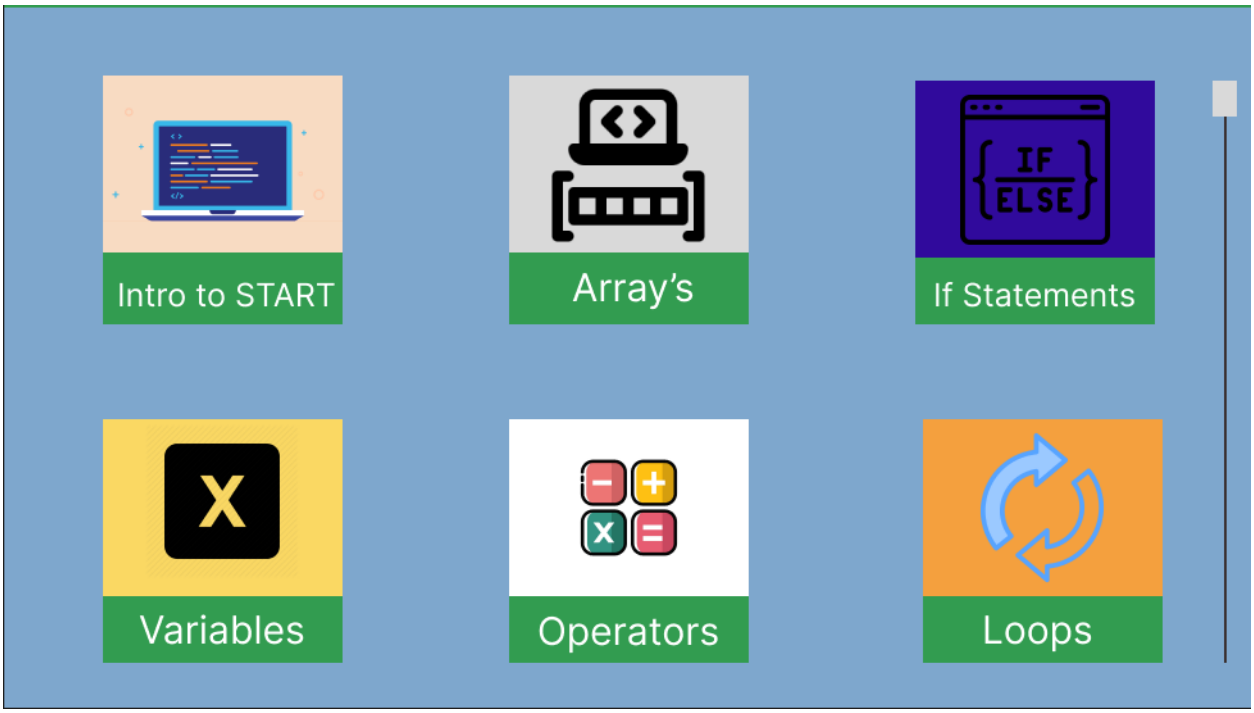
## IDE/Debugger:

This page of the application would contain the space needed for the user to write, run and debug their own written code. As seen the IDE would offer syntax highlighting, the output would be seen at the bottom, and the debugger would be seen on the side.




**Resource Page:**

The resource page for videos and documentation could potentially use a card concept for the displaying of different types of information the user might need to know about.




# 8. Appendices

- **Django**
- **Docker**