Functional Specification - Snap-Py

Student 1: Luigi Di Paolo - 21725939
Student 2: Andy Vodopi - 21477816
Supervisor: Stephen Blott
Date Finished: 22/11/2024

0. Table of contents

1. Introduction

1.1 Overview

This project is a novel visual programming tool designed to make learning Python more accessible and engaging for beginners.

Learning to code can be daunting for many people, especially for beginners who are taking their first steps into programming. Despite the growing emphasis on computer literacy and coding in education, a significant number of students struggle and often fail introductory programming courses. Research shows that several factors contribute to these challenges, including the cognitive load associated with learning complex syntax, difficulty in grasping abstract concepts, and the lack of immediate feedback on errors or logic gaps in the code (Weeda et al., 2023). These issues create barriers to entry for beginners, who often become discouraged early on and give up before they develop a solid foundation.

Block-based coding, however, has emerged as an effective tool to introduce programming in an accessible, intuitive way. With block-based coding, learners can assemble code by manipulating visual blocks rather than writing complex syntax. Studies show that this approach can significantly improve algorithmic thinking, as it allows users to focus on core concepts like loops, conditionals, and data flow without being overwhelmed by details of syntax (Weintrop & Wilensky, 2015). Additionally, block-based programming engages users through a playful interface, making the learning process more enjoyable and less intimidating.

Python is one of the most widely recommended languages for beginners due to its readability, concise syntax, and high level of abstraction. Many educators and researchers have highlighted Python's suitability for introductory programming because it minimises distractions caused by complex syntax and lets students focus on learning core programming concepts more quickly (Cutting & Stephen, 2021). Python's straightforward structure and versatility make it an ideal choice for an educational programming tool that could help a wide range of learners, from middle school students to adult novices.

Our goal is to create a novel educational programming tool that combines the best aspects of block-based programming with Python's accessible syntax and abstraction levels. The tool we are developing will allow users to create Python code by manipulating blocks, offering them the cognitive benefits of a visual programming environment while gradually familiarising them with the structure and syntax of Python code. In this way, beginners can focus on developing logical and computational thinking skills without being burdened by the frustration of syntax errors or unfamiliar constructs.

An additional key feature of our tool is a built-in interpreter with step-through capabilities. This interpreter will allow students to execute their code one step at a time, visually following the logic flow and observing variable values as they change. By having this interactive, step-by-step interpreter, beginners will be able to understand the inner workings of their code, make connections between block structures and Python code, and easily debug errors. This feature addresses the common problem in programming education where students

struggle to visualise how the code executes, often leading to frustration and misunderstandings about control flow and data manipulation.

1.2 Business Context

N/A

1.3 Glossary

- **OAuth 2.0**: OAuth 2.0, which stands for “Open Authorization”, is a standard designed to allow a website or application to access resources hosted by other web apps on behalf of a user. It replaced OAuth 1.0 in 2012 and is now the de facto industry standard for online authorization (Okta, n.d.).
- **Block-based coding:** Block-based coding means that instead of typing a coding language, you drag and drop jigsaw-like pieces together to build things using tools like Scratch or Thunkable (Dunlop, n.d.).

2. General Description

2.1 Product / System Functions

2.1.1 Core Functionality: User Authorisation (OAuth 2.0)

To ensure secure access, users will be authorised via OAuth 2.0, a robust authorization protocol that enables secure and token-based access without requiring users to share sensitive credentials directly. OAuth 2.0 will allow users to be authorised through trusted third-party providers (like Google or Microsoft). Once authorised, each user will gain unique access permissions to their workspaces and code, enhancing security while simplifying the sign-in process.

2.1.2 Core Functionality: Workspace Creation

Each user can create personalised workspaces where they can organise, build, and test their Python programs. Workspaces are intended as separate, structured environments where users can experiment with code blocks, write Python code, and work on various programming exercises or projects.

2.1.3 Core Functionality: Workspace Deletion

Users will have the ability to delete workspaces they no longer need. Deletion will permanently remove the workspace along with all associated files and code, helping users manage their workspace lists efficiently. This functionality also helps users avoid clutter and maintain focus on active projects. Deletion processes will incorporate a confirmation step to prevent accidental workspace removal.

2.1.4 Core Functionality: Python Program Creation with Drag-and-Drop Blocks

Users can create Python programs by dragging and dropping code blocks into a workspace. This block-based approach abstracts away complex syntax, enabling beginners to focus on core programming concepts and logical flow without worrying about syntax errors. Each block will represent Python constructs like loops, conditions, variables, and functions, giving users a visual and interactive way to compose programs that generate equivalent Python code in real time.

2.1.5 Core Functionality: Block-Based Editing of Python Programs

When users edit a Python program, any changes made in the block-based workspace will immediately update the underlying Python code. This synchronised editing capability should allow users to see how modifications in blocks directly reflect in Python syntax, reinforcing the connection between block-based structures and text-based coding.

2.1.6 Core Functionality: Program Execution

Users can execute the Python programs they build in their workspaces. This execution functionality interprets the block-generated Python code and runs it, displaying output and providing immediate feedback on program behaviour. The execution environment is designed to handle Python's interactive features, allowing users to see results or errors and understand how their code performs step-by-step in real time.

2.1.7 Extra Functionality: Debugging with Step-Through Execution

For enhanced learning, the tool will offer a debugging feature that enables users to step through code execution block by block. As users progress through each line, the corresponding block in the workspace will be highlighted alongside its Python code equivalent, providing a clear visual map of the program's execution flow.

2.2 User Characteristics and Objectives

2.2.1 User Characteristics

This tool targets beginner programmers, especially those with minimal coding experience. Key user groups include:

- **Primary, Secondary, and Introductory College Students:** Ideal for younger students and those in introductory programming courses, offering a hands-on way to learn coding basics.
- **Educators and Instructors:** Teachers seeking an accessible, visual tool to introduce programming fundamentals will find this tool supports interactive learning.
- **Self-Learners:** Adults or independent learners interested in programming but unsure about traditional code syntax can benefit from this tool's simplified approach.
- **Organisations and Coding Bootcamps:** This tool is suitable as an entry-level resource, enabling quick skill-building before progressing to text-based coding.

2.2.2 Objectives

The main objective of this tool is to provide an intuitive, engaging learning experience that facilitates a smooth transition from block-based programming to standard text-based coding, building foundational skills for deeper exploration.

2.3 Operational Scenarios

2.3.1 User Authorisation with OAuth 2.0

USE CASE 1	User Authorisation with OAuth 2.0	
Description	Allows the user to securely log in using OAuth 2.0	
Actors	User	
Preconditions	User is on the login page.	
Triggers	User selects a third-party provider for authorisation.	
Basic Flow	**Step**	**Action**
	1	User selects OAuth provider (e.g., Google, Microsoft).
	2	System redirects to provider's authorisation page.
	3	User authorises app access.
	4	System authorises user and redirects them to dashboard.
Postconditions	User is logged in and can create or modify workspaces.	
Exceptions	1	Authorization denied by user.
	2	Authorisation failure due to network issues.

2.3.2 User Creates Workspace

USE CASE 2	User Creates Workspace	
Description	User creates a new workspace to organise their Python projects.	
Actors	User	
Preconditions	User selects “Create Workspace” option from dashboard page.	
Triggers	User selects “Create Workspace” option.	
Basic Flow	**Step**	**Action**

	1	User clicks “Create Workspace.”
	2	System prompts user to name workspace.
	3	User provides a name.
	4	System creates the workspace.
Postconditions	New workspace is available to the user for editing.	
Exceptions	1	1. Invalid workspace name.
	2	2. Network or server error.

2.3.3 User Deletes Workspace

USE CASE 3	User Deletes Workspace	
Description	User deletes an existing workspace.	
Actors	User	
Preconditions	User is logged in and has an existing workspace.	
Triggers	User selects “Delete Workspace” option.	
Basic Flow	**Step**	**Action**
	1	User clicks on workspace options and selects “Delete.”
	2	System prompts for confirmation.
	3	User confirms deletion.
	4	System deletes the workspace and all associated data.
Postconditions	Workspace is deleted, and user’s workspace list is updated.	
Exceptions	1	User cancels deletion.
	2	Network or server error.

2.3.4 User Edits Block

USE CASE 4	User Edits Canvas
Description	User modifies or deletes a code block on the workspace canvas.
Actors	User

Preconditions	User is logged in and has an open workspace.	
Triggers	User selects a block to edit.	
Basic Flow	**Step**	**Action**
	1	User deletes or modifies block.
	2	System updates the workspace canvas.
Postconditions	Canvas is updated, and the generated Python code reflects changes.	
Exceptions	1	Unsupported block modification.
	2	2. System or network error.

2.3.5 User Adds Block to Canvas

USE CASE 5	User Adds Block to Canvas	
Description	User adds a new block on the workspace canvas.	
Actors	User	
Preconditions	User is logged in and has an open workspace.	
Triggers	User selects a new block to add to canvas from workbench.	
Basic Flow	**Step**	**Action**
	1	User fills in details for the new block.
	2	User drops the new block onto the canvas.
	3	System updates the workspace canvas.
Postconditions	Canvas is updated, and the generated Python code reflects changes.	
Exceptions	1	Unsupported block details entered.
	2	User attempts to connect the new block with an incompatible block.
	3	2. System or network error.

2.3.6 User Edits Generated Python Code

USE CASE 6	User Edits Generated Python Code
Description	User edits the Python code generated from blocks directly.

Actors	User	
Preconditions	User is logged in and has added/edited blocks.	
Triggers	User clicks in the Python code editor.	
Basic Flow	**Step**	**Action**
	1	User edits the Python code.
	2	System synchronises changes to reflect on blocks (if applicable).
Postconditions	Python code is saved, and any linked block modifications are updated.	
Exceptions	1	Syntax errors in edited code.
	2	System or network error.

2.3.7 User Runs interpreter in Normal Mode

USE CASE 7	User Runs interpreter in Normal Mode	
Description	User runs the code to see the final output without stepping through individual commands.	
Actors	User	
Preconditions	User has a complete Python program in the workspace.	
Triggers	User clicks the run in normal mode button.	
Basic Flow	**Step**	**Action**
	1	User clicks the run button.
	2	System interprets and executes the code.
	3	System displays output.
Postconditions	User views the output in the output console.	
Exceptions	1	Code contains errors, resulting in execution failure.
	2	System error.

2.3.8 User Runs Interpreter in Debugging Mode

USE CASE 8	User Runs Interpreter in Debugging Mode
Description	User runs the interpreter in debugging mode to step through each code block and line of Python code.

Actors	User	
Preconditions	User has a complete Python program in the workspace.	
Triggers	User clicks the run in debugging mode button.	
Basic Flow	**Step**	**Action**
	1	User clicks the run in debugging mode button.
	2	System steps through each line of code, highlighting the corresponding block.
	3	User proceeds through steps manually.
	4	System displays output.
Postconditions	User views the output in the output console.	
Exceptions	1	Code contains errors, resulting in execution failure.
	2	System error.

2.4 Constraints

Below is a list of possible constraints that the team will face during the development of this project:

- Time Constraints: The project timeline may limit the scope and depth of features implemented.
- Server Limitations: Using free-tier servers could result in restricted performance, storage, or bandwidth.
- Python Version: The interpreter and code generated will only support one python version (3.x).
- Python Feature Scope: Incorporating all Python capabilities is impractical; only essential features will be implemented to align with the project's learning objectives.

3. Functional Requirements

New user registration:

Description

- The system shall allow new users to register using OAuth 2.0, enabling them to sign up with third-party providers such as Google or GitHub.

Criticality

- This is an extremely important requirement for obvious reasons. A user must be registered to use the web app.

Technical issues

- The main challenge with implementing user registration using OAuth 2.0 will be the integration of third-party authentication providers into the web app. Ensuring secure handling of tokens and managing user data retrieved from these providers are also key considerations.

Dependencies with other requirements

- Without registering, users cannot use the web app. Therefore, all other requirements depend on this one.

User Login/Logout:

Description

- The system shall allow registered users to log in and log out of their accounts.

Criticality

- This requirement is essential. Users must be logged in to access and use the application, and they must be able to log out to ensure security.

Technical issues

- Similar to user registration, the primary challenge will be integrating an authentication service like OAuth for managing user sessions and securely handling login/logout processes.

Dependencies with other requirements

- All subsequent requirements presume that the user is logged in; thus, these functions are foundational.

User creates workspace:

Description

- The system shall allow users to create new workspaces for organising their projects.

Criticality

- This is a highly important requirement, as the ability to create workspaces is central to the organisation and management of user projects within the application.

Technical issues

- Challenges include ensuring that workspace creation is seamlessly integrated into the user's account, with appropriate data structures to store workspace information and manage concurrent creation requests.

Dependencies with other requirements

- Requires that the user is logged in.

User deletes workspace:

Description

- The system shall allow users to delete existing workspaces.

Criticality

- This requirement is significant, as users need control over their projects and data, including the ability to remove unwanted or outdated workspaces.

Technical issues

- Key challenges include implementing data validation to prevent accidental deletions and ensuring the process is secure and irreversible only after confirmation.

Dependencies with other requirements

- Requires that the user is logged in and has existing workspaces.

User drags blocks into canvas:

Description

- The system shall allow users to drag coding or function blocks into the workspace canvas for developing code visually.

Criticality

- This feature is crucial for interactive code building and enhances the user experience by making coding more intuitive.

Technical issues

- The main challenge is implementing a robust drag-and-drop interface with responsive behaviour that updates in real time. Ensuring cross-browser compatibility may also be an issue.

Dependencies with other requirements

- Requires the user to be logged in and have an active workspace.

User edits generated Python code:

Description

- The system shall allow users to manually edit the Python code generated from the dragged blocks.

Criticality

- This requirement is very important for users who want to fine-tune or customise their code beyond what is possible with pre-built blocks.

Technical issues

- Challenges include maintaining code integrity when switching between visual blocks and manual code editing. Ensuring edits are tracked and changes do not break the visual representation is also important.

Dependencies with other requirements

- Requires that the user is logged in, has an active workspace, and has already dragged blocks into the canvas.

User runs interpreter in normal mode to see output of code:

Description

- The system shall allow users to run the Python code in normal mode to view its output.

Criticality

- This requirement is important for testing and verifying that the code behaves as expected.

Technical issues

- Challenges involve implementing a safe and isolated runtime environment for executing code and displaying output.

Dependencies with other requirements

- Requires that the user is logged in, has an active workspace, and has code available to run.

User runs interpreter in debugging mode:

Description

- The system shall allow users to run the Python code in debugging mode, enabling them to step through the code and observe variable states.

Criticality

- This is an important requirement for users who need to diagnose and resolve issues in their code.

Technical issues

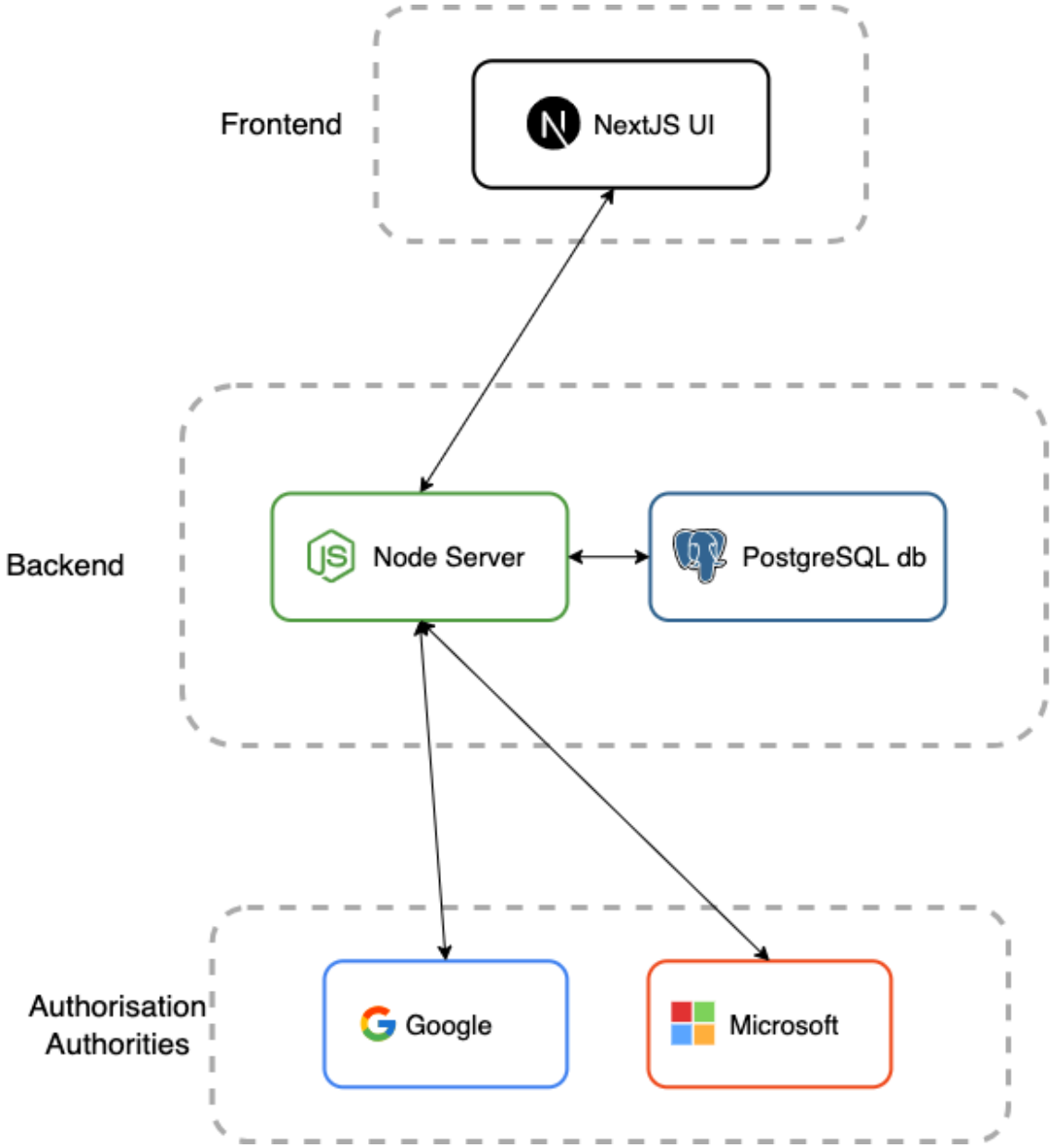
- Implementing debugging functionality requires an advanced runtime capable of supporting step-by-step execution, breakpoints, and variable inspection.

Dependencies with other requirements

- Requires that the user is logged in, has an active workspace, and has code available to debug.

4. System Architecture

4.1 Architecture Diagram



4.2 Architecture Description

The system architecture is divided into three main components: the frontend, the backend, and the OAuth authorisation providers.

4.2.1 Frontend

The frontend will be built using Next.js, a React-based framework. Next.js is chosen for its server-side rendering capabilities, static site generation, and excellent performance

optimizations. It will provide a modern and responsive user interface while enabling seamless interactions with the backend.

4.2.2 Backend

The backend will use a Node.js server with a PostgreSQL database. Node.js will handle server-side logic and API endpoints, while PostgreSQL will store and manage the application's data.

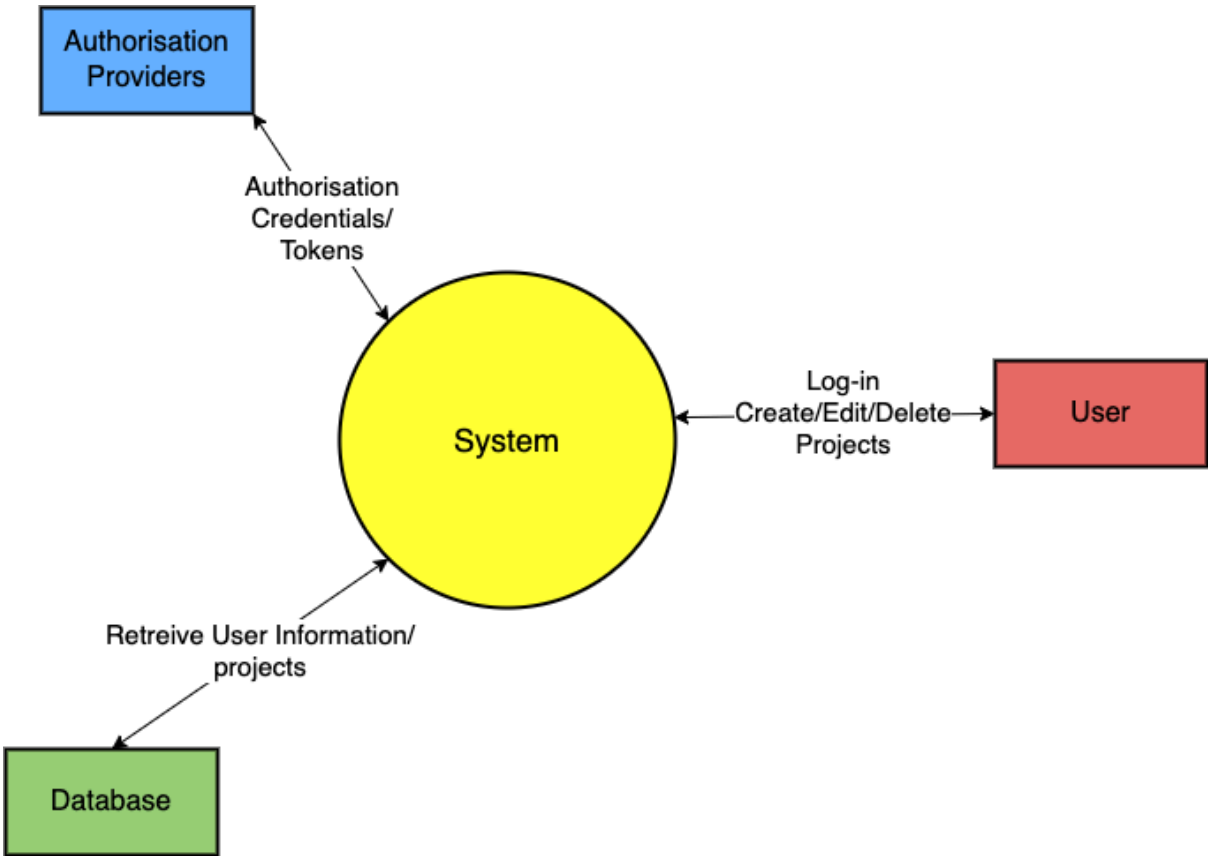
4.2.3 Authorisation Providers

Authorisation will be handled by OAuth 2.0 providers such as Google and Microsoft. These providers will manage user authorisation, ensuring secure and reliable identity verification.

5. High-Level Design

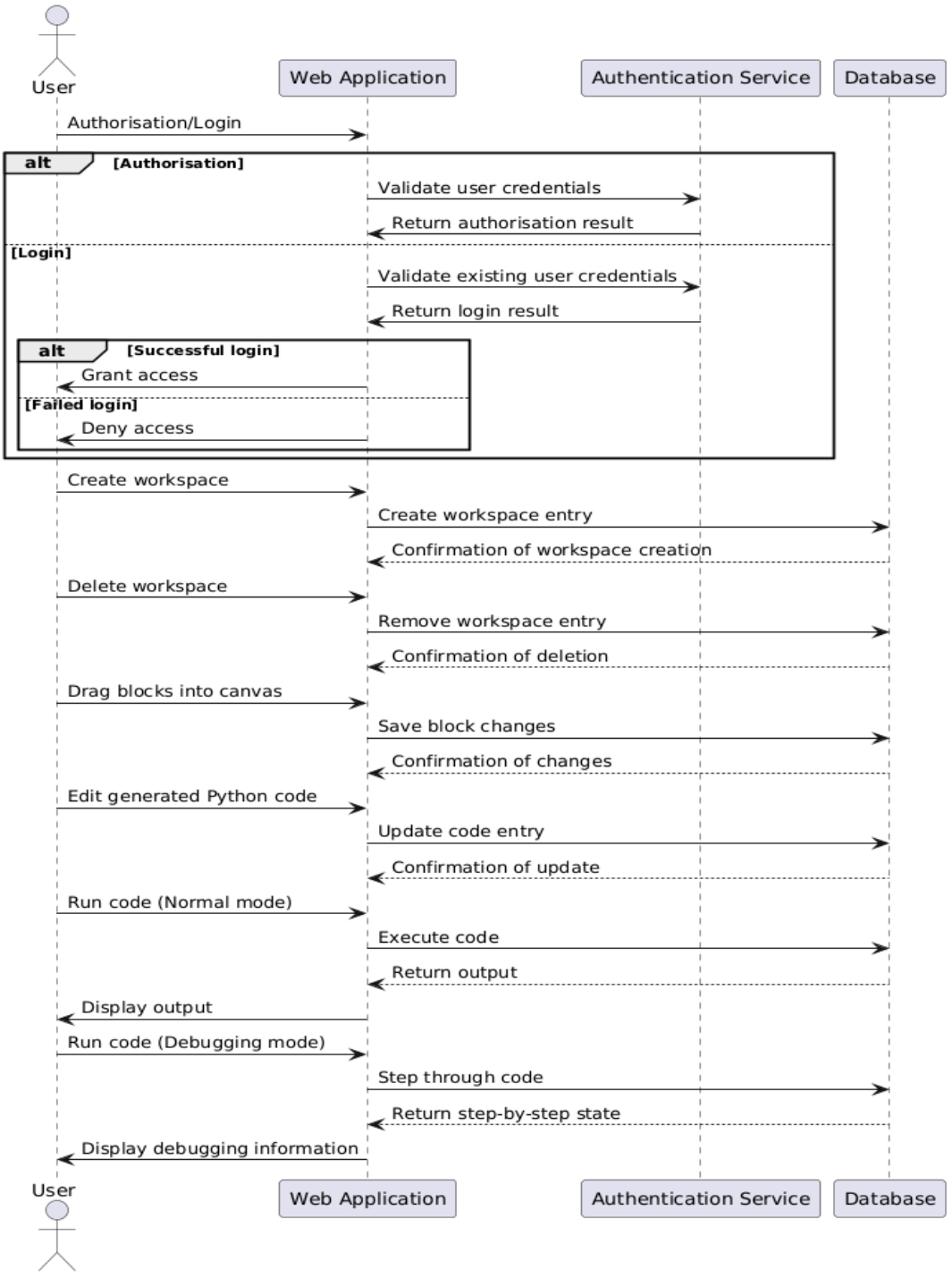
5.1 System Context Diagram

This system context diagram describes the boundaries of the system and how it interacts with its entities and systems. It depicts the managers with elevated access permissions and the employees with more restricted access, as well as showing information about how it interacts with the authentication and database systems.



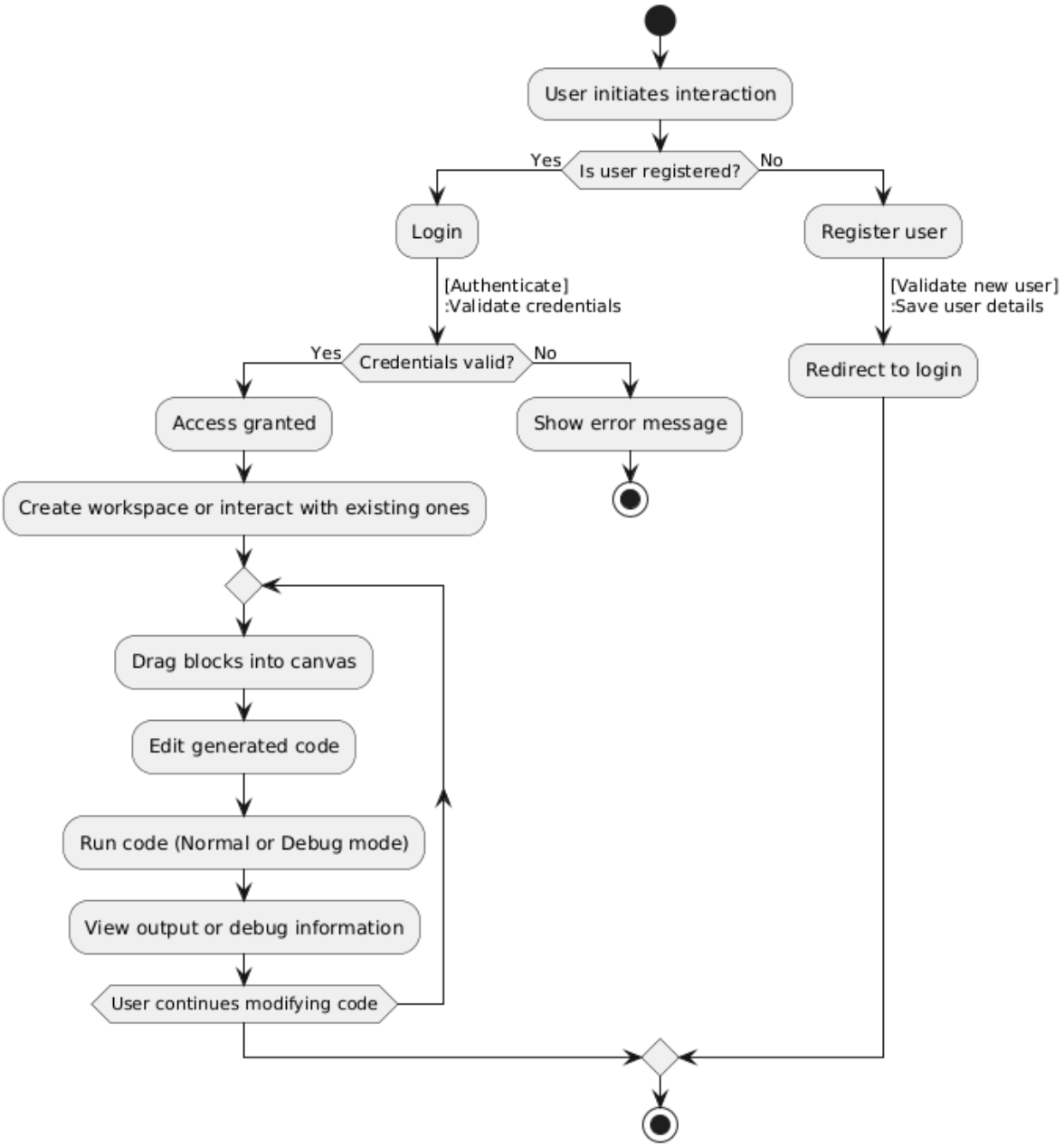
5.2 Sequence Diagram

This sequence diagram gives an overview of the sequence of interactions a user has with our system from a temporal perspective. It describes it from the initial user action and how that action then propagates throughout the entire system.



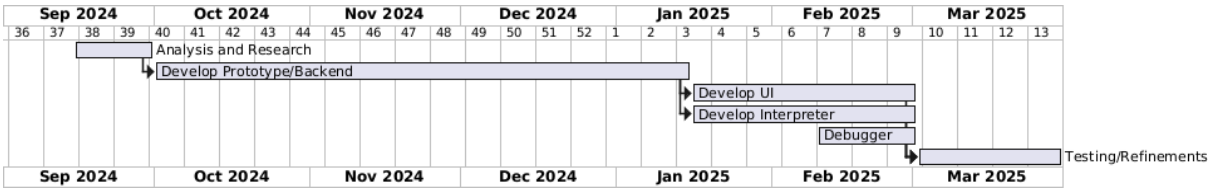
5.3 Activity Diagram

This activity diagram demonstrates the behavioural flow of the user when they interact with the system. It models the workflow of the system by breaking down the logical interactions a user can have with it.



6. Preliminary Schedule

6.1 Gantt Chart



7. Appendices

7.1 References

- Weeda, R., Smetsers, S., & Barendsen, E. (2023). Unraveling novices' code composition difficulties. Computer Science Education, 1–28. https://doi.org/10.1080/08993408.2023.2169067
- Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question. Proceedings of the 14th International Conference on Interaction Design and Children. https://doi.org/10.1145/2771839.2771860
- Cutting, V., & Stephen, N. (2021, August). A Review on using Python as a Preferred Programming Language for Beginners. Researchgate.net; International Research Journal of Engineering and Technology (IRJET). https://www.researchgate.net/profile/Vineesh-Cutting/publication/359379004_A_Review_on_using_Python_as_a_Preferred_Programming_Language_for_Beginners/links/6238883f14c740613d45172b/A-Review-on-using-Python-as-a-Preferred-Programming-Language-for-Beginners.pdf
- Okta. (n.d.). What is OAuth 2.0 and what does it do for you? Auth0. https://auth0.com/intro-to-iam/what-is-oauth-2
- Dunlop, S. (n.d.). Subject Guides: Coding: a Practical Guide: Block-based coding. Subjectguides.york.ac.uk. https://subjectguides.york.ac.uk/coding/scratch