Friday 11am - Team 2 - IT Project

IT Project - Friday 11am - Team 2

👋 **Welcome to the IT Project Confluence Page**

- 🔭 Project Inception Documents
- 🗓️ Meetings History
- 🟦 Decision Log/Document
- ⏳ Jira Overview
- 🎯 Sprint Documentation

About

This page contains all the key documentation pertaining to the development process of the 'AuthorGuard' web application. This includes project inception documents, such as requirements, design & UX/UI design documents, as well as records of meeting minutes, notes, key decisions, Sprint reviews & retrospectives, as well as other important events and ceremonies throughout the Agile development process.

Mission and vision

We aim to develop a web-based interface to allow non-technical users to use stylographic technologies to detect plagiarism in academic writing. This usable interface, branded with the name "AuthorGuard" is a full-stack web application that leverages powerful modern day machine learning techniques to detect with a high level of accuracy the likelihood that a given piece of writing was written by a particular individual.

The development team aims to deliver a functional, efficient and easy to use web-application to allow for the usage of these machine learning algorithms by a non-technical academic audience to help protect authorial integrity in an era of Large Language Models & Contract Cheating.

Meet the Team

- **Product Owner:** Ayush Tyagi - ayusht@student.unimelb.edu.au
- **Scrum Master:** Ke Liao - keliao@student.unimelb.edu.au
- **Developer:** Jack Perry - perryja@student.unimelb.edu.au
- **Developer:** Josh Costa - jncosta@student.unimelb.edu.au
- **Developer:** Bryce Copeland- bacopeland@student.unimelb.edu.au

Contact us

- Jira Board

Important Pages

- 🔍 Requirements Document
- 🗓️ Meetings History

- 📘 Decision Log/Document
- ⏳ Jira Overview
- 🎯 Sprint Documentation

🔭 Project Inception Documents

This is a landing page for the variety of documents pertaining to the inception and scope of the project.

Document Name	Key Purposes	Link
Requirements Document	User Stories, Functional & Non-Functional Requirements	🔍 Requirements Document
Design Document	Conceptual Design, System Component & Dynamics	🖌️ Design Document
UX/UI Design Document	User Interaction, Web Design	✏️ UX/UI Design Document

🔍 Requirements Document

IT Project - Requirements Document

Team Members: Ayush Tyagi, Ke Liao, Jack Perry, Josh Costa, Bryce Copeland

---

Table of Contents:

Introduction

The aim of this document is to serve as a comprehensive repository of all important artifacts from the project inception phase pertaining to the requirements identification, modelling & discussion. This includes motivational modelling, functional & non-functional requirements tables, user stories and target audience personas. These artifacts will serve as a continuous point of discussion in the decision-making process for the development of this project.

Motivational Modelling Table - Do/Be/Feel List

To begin to understand the requirements of the project, a do/be/feel list was created as part of the motivational modelling process.

Who	Do	Be	Feel
• Student • Staff	• Upload documents to a chosen profile • Compare a new document with an existing profile • Store/Manage user profiles	• Usable (friendly interface) • Reliable • Scalable • Honest • Professional • Secure	• Proud • Authoritative • Empowered • Satisfied • Confident • Eager

Motivational Models:

Figures 1 & 2 each depict a motivational model summarising the requirements identified in the motivational modelling table above. Figure 1 outlines the minimum viable product, the bare essential requirements for a functional application. Figure 2 expands on Figure 1 to include all user stories and stretch goals.

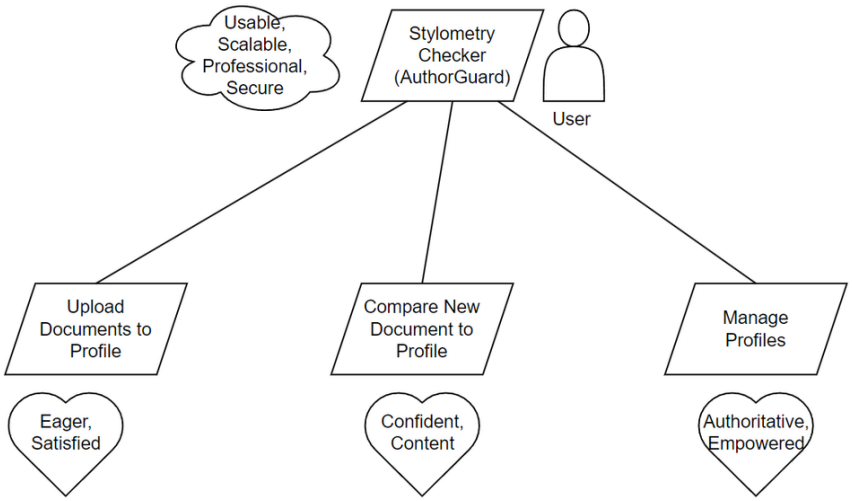


Figure 1 - Motivational Model of Minimum Viable Product

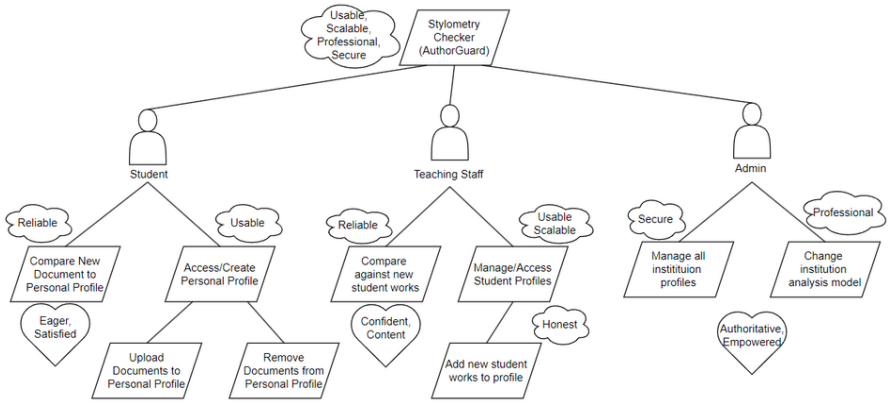


Figure 2 - Motivational Model ft. Stretch Goals & User Stories

Functional Requirements

The following table outlines the functional requirements of the application. Note: Medium and Low Priority are Stretch Goal.

Requirement	Description	Difficulty	Priority
Uploading Document	• Front-end until submitted to profile	• Low	High: • Core Functionality
Communication with the algorithm	• Back-end • Need for the results that the program outputs to user	• High:	High: • Need it for analysis • Core Functionality
Display the score	• Output the Score of Analysis on the UI layer	• Low	High: • Core Functionality
Profile Storage	• Storing the information of User profiles across sessions	• Medium: ◦ Requires managing and querying databases	Medium: • Need to store across sessions • However, not as important as having the core functionalities of stylistic analysis working.
Edit/Delete Document in Profile	• Edit and/or Delete documents already in profile	• Medium: ◦ Same as Above	Medium: • One of the first expansion after making sure the core functionality works.
Authentication and Access Control	• Make sure Users only have access to information they are supposed to • Username & Passwords	• High	Medium: • Making sure unauthorized users don't access data.
Score Breakdown	• Gives the breakdown of similarity and difference of writing area. • Evidence based approach	• High ◦ Need significant understanding of underlying Algorithm ◦ Significantly increased back-end workload	Low: • A potential Expansion
Social Media integration	• Login in with social media	• Medium	Low: • A potential Expansion

Non-Functional Requirements

The following table defines the non-functional requirements of the application.

Requirement	Description	Difficulty	Priority
Usability of the Web App	• Clean UI ◦ Someone Bad with Technology can use	Medium	High(Want clean UI to allow use by all users)
Reliability	• Uptime	Medium	Medium:

	• Accuracy of results displayed		• Want the app work most of the time and be accurate
Performance	• How fast app loads • How fast does the app get results • How fast are the uploads	Unknown	Low: • Ideally the app should respond quickly. • However, the development process is better spent on making sure all necessary features are added
Scalability	• Making sure the app is usable for individuals or large organization/universities	Low	Medium: • Want to be able to work for students and university • However, creating a functional usable app is more important than this
Sup-portability of Documents	• Support uploading of documents in different format such as .doc, .docx, .pdf, .txt, etc.	Variable(depends on which document types we want to include and how many): • Need to ensure app extracts text from the file properly	Low

User Stories

To connect the functional requirements to the target audience. It is important to create a series of user-stories to intertwine the functional and non-functional requirements to the user experience of the application. These will influence the decision-making process during development.

User Story Template: *As a [role], I want to [action] so that I can [benefit].*

Acceptance Criteria: *Conditions to be fulfilled before user story is satisfied.*

Priority: *How essential is this functionality (High/Medium/Low)? Is this a short, mid or long term goal?*

User Story	Acceptance Criteria	Priority
As a user, I want to compare a singular document to a group of documents so that I can see the writing style similarity percentage.	• Able to upload group of documents • Able to upload an individual document • Able to initiate a comparison between the group of documents and the individual document • The resulting writing style similarity percentage is displayed to user	High
As a user, I want to add/remove documents to/from the group, so that I can easily change which documents are being compared against.	• Able to remove previously uploaded documents from group • Able to upload and append additional documents to group	High
As a user, I want to create a personal account so that my uploaded documents are saved between browsing sessions.	• Able to create a (private) account which has a group of documents uniquely associated with it	High
As a teaching staff member, I want to access my students' existing works so that I can compare it against their new work.	• Able to access an individual student's profile from a shared database of student profiles • Able to compare a new document with a student's existing documents	Medium
As a teaching staff member, I want to add a student's new work to their existing profile, so that their profile remains up to date.	• Able to request an edit to a shared student profile	Medium
As an administrator, I want profile editing privileges so that I can create/delete/maintain profiles as my institution requires.	• Able to append/remove student profiles to/from the shared database • Able to append/remove documents from student profiles • Able to accept/decline profile edit requests from staff	Medium
As an administrator, I want to be able to easily select/update the analysis model for my institution.	• The option to select/update the model applied for accounts institution-wide. • Possibility to extend to class-wide, or even individual, if there is sufficient reason to need multiple models.	Low
As a user, I want to be able to upload my documents in any format easily.	• Able to upload/store documents in common formats. PDF, docx, etc. • Parse usable text from documents, while keeping them stored in their original form.	Medium

As a user, I want to connect my account to social media so that I can login easier.	• Allow login using other platforms (e.g. Google, Facebook)	Low
As a user, I want the comparison results to highlight the major similarities/differences in the text, so that I can easily identify potential problem areas.	• Highlight specific outliers in the text which the algorithm identifies (e.g. variation in word length, unique word usage)	Low

Personas

To further understand the target audience, persona artifacts were created to summarise the different types of users of the application, their motivations for using it and benefits they gain from their usage of it. These are based on the user-stories defined previously.

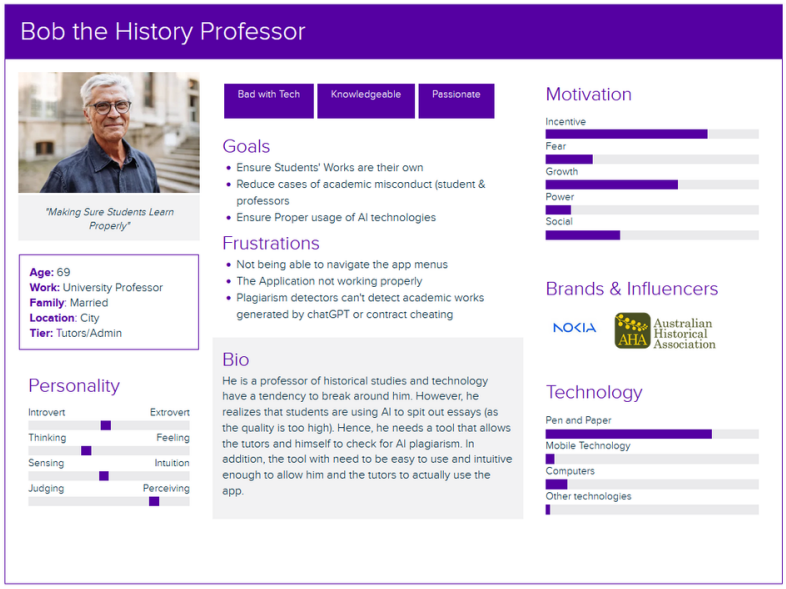


Figure 3 - Bob the History Professor Persona

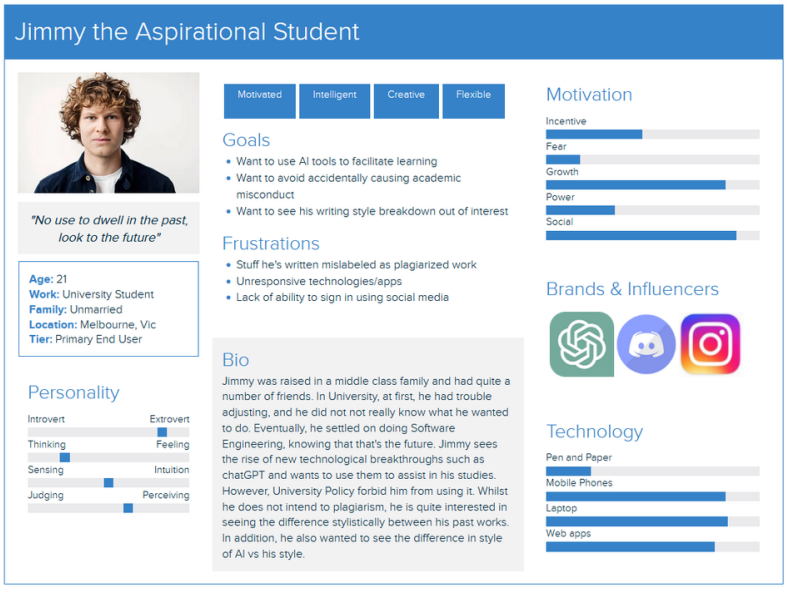
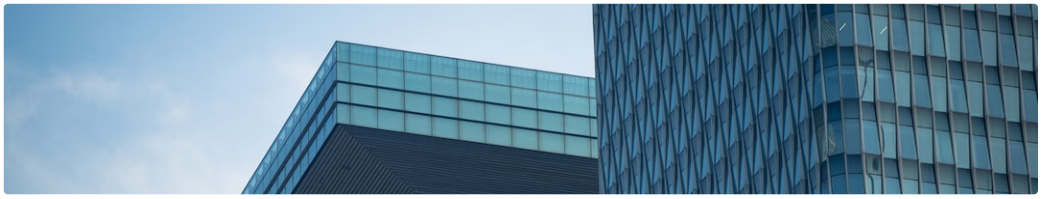


Figure 4 - Jimmy the Aspirational Student Persona



🖋 Design Document

IT Project - Design Document

Team Members: Ayush Tyagi, Ke Liao, Jack Perry, Josh Costa, Bryce Copeland

---

Table of Contents:

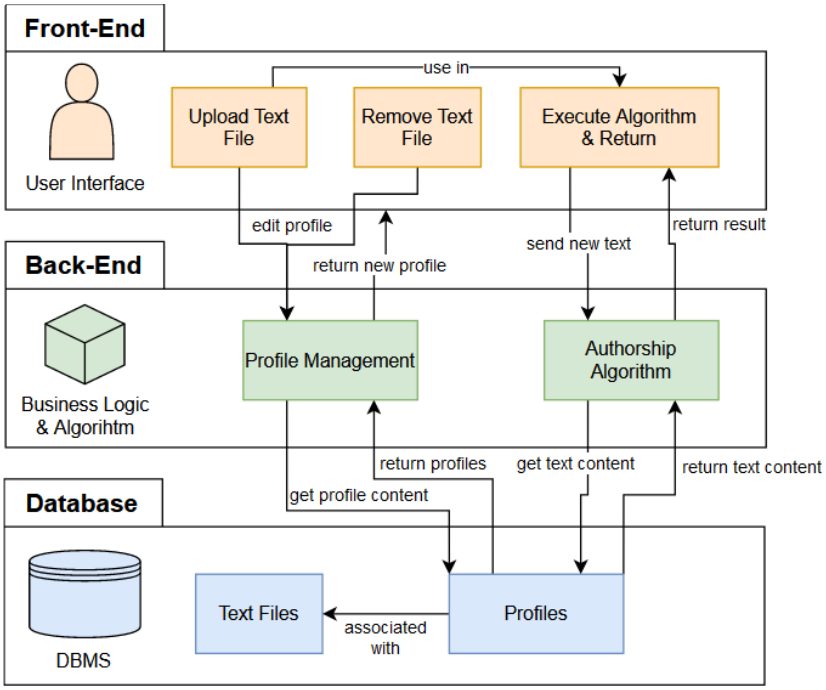


Figure 1 - Ideal Architectural Components & Basic Functions

The Front-End is the User Interface of the system, which in this case is the webpage displayed to the user. The primary aims of this component are to take inputs from the users (such as creating profiles, uploading text files, clicking the execute button for the algorithm, etc.) and be able to interpret and send those inputs to the server side Back-End. It will also need to receive back results and change the display according to the new information (alterations to profile, algorithm result, etc.).

The Back-End component is the interface for the Front-End to perform useful operations. It handles the business logic, such as profile management, and queries the Database API to execute changes as required by the user inputs from the Front-End. It is also in charge of running the authorship algorithm and returning useful results, which will involve getting all the text files for the profile from the Database and running on the algorithm on the new text file and returning the result to the front-end to display.

The Database component is where the profiles, text files and user information will need to be stored. A relational database will be used to easily capture and manage the relationships between profiles, text files and users. The Database will be accessible via an API which will need to be setup. After which, the Back-End can use that API to safely interact with the Database.

Architectural/Deployment Strategies

Following on from the previous section, the specific deployment setup for the prescribed architecture will need to be carefully considered. The following Figure outlines the contenders for the development & deployment of each of the critical layers of the system.

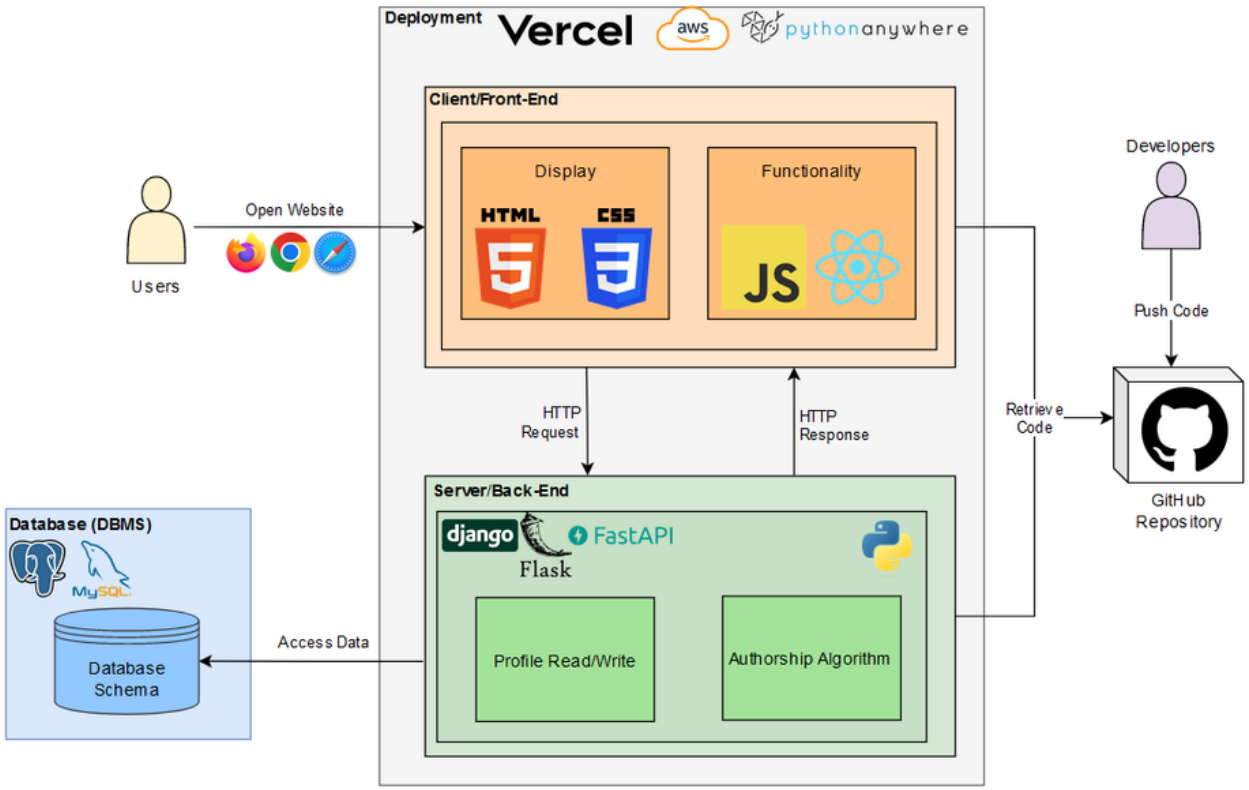


Figure 2 - System Architecture Options

As shown in the figure, the client-side that users interact with, or the front-end, will use HTML & CSS for the display and the functionality will be enabled either by standard JavaScript or by React, depending on developer experience and feasibility of implementation,

This client-side will then send HTTP requests to the server-side, which will be a Python-based server as the algorithm for authorship verification is provided via a Python API hence it would be the most efficient way to create the application. One of Django, Flask or FastAPI would need to be used here to implement the HTTP request and responses as well as the interaction with the DBMS.

These two components/layers will be deployed via one of Vercel, AWS or PythonAnywhere. There are other contenders but these have been identified as the best candidiates due to pricing, developer experience & capacity. The small-scale nature makes these services suitable for the deployment of both the front-end and back-end of the application. This will also make it easier to connect the client and server sides as these deployment options provide services to simplify this process.

In terms of the database, the server-side code will need to query to the database and retrieve the data and return it to the client via HTTP responses. The team has decided that a relational database, such as PostgreSQL or MySQL are good options for this service given the team's experience with relational databases.

System Design

The dynamic behavior of the system can be showcased in the follow Figures 3 and 4, which showcase the basic version of the system and a desired extended version of the system as domain models.

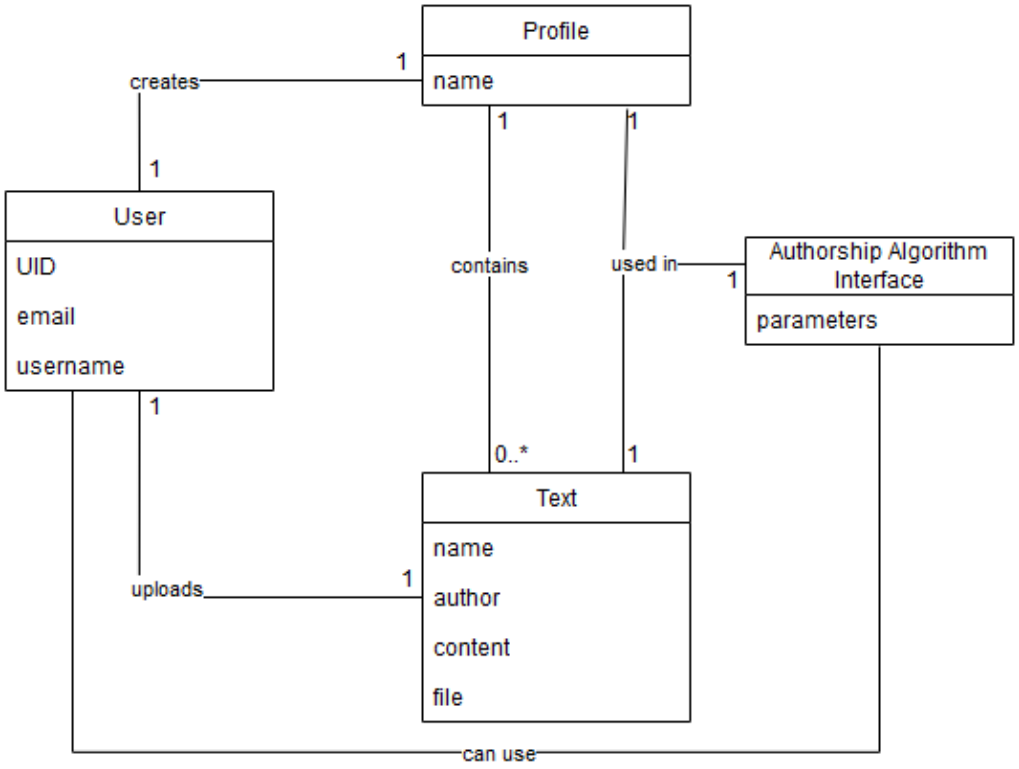


Figure 3 - Basic System Dynamics (Minimum Viable Product Domain Model)

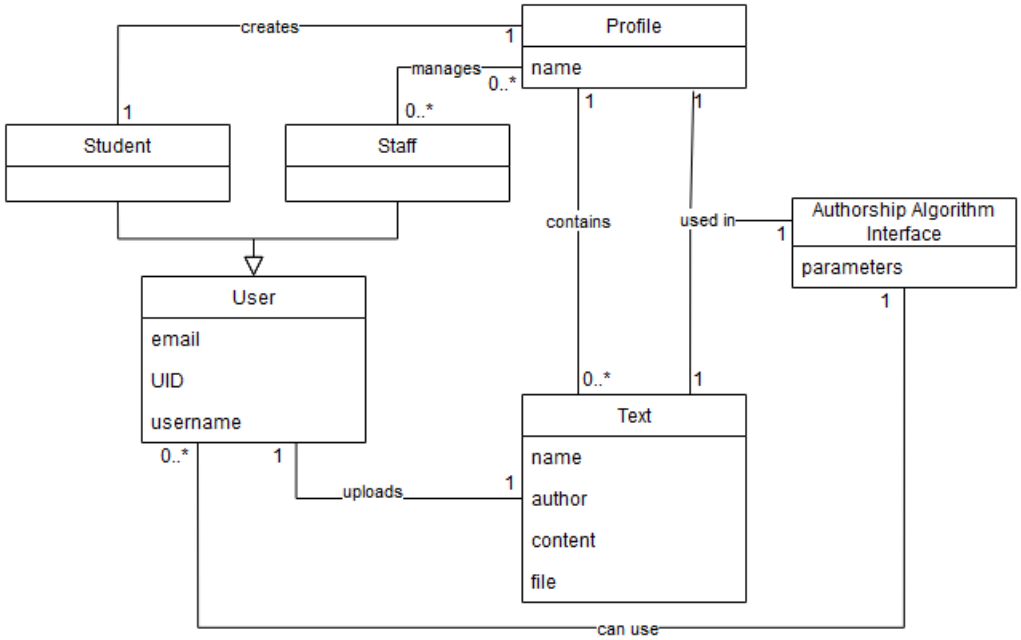


Figure 4 - Extended System Dynamics (Stretch Goals Domain Model)

These two diagrams summarises the basic system dynamics (the minimum requirements) and the extended system dynamics (a more ideal version with more user stories included). The team intends to deliver a functional version of system that strongly aligns with the dynamics showcased in Figure 2 and then build upon it, improving all components until a version that captures the extended system dynamics can be developed and deployed.

Another key component of system design is the UI & Web Page design, which is outlined separately in the UX/UI Design Document, including the diagrams showcasing the desired look, style and navigation options of the webpage.

Database Design

Given the full-stack nature of the web-application, an external database will be required to store the users, their profiles and the document information for each file that they have uploaded to that profile. Fortunately, this means that the database schema will be relatively simple, which is suitable for the relatively small-scale of this project.

The following figure provides an example of a database diagram for this project.

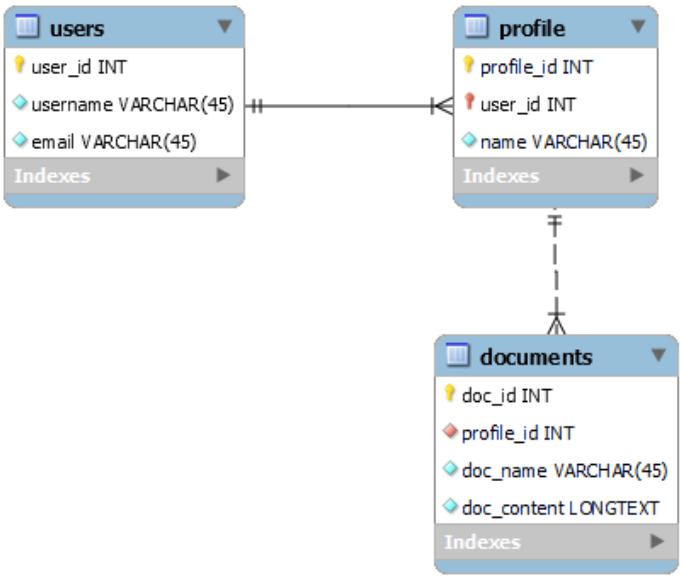


Figure 5 - Simple Design Database Diagram

This will be the database schema that team will initial deploy with as it captures all the key relationships between entities in this system for the purposes of data storage. An API will need to be constructed for this database.

In terms of future improvements, user types will ideally be added (students, educators, admins), and their respective profile associations may also need to be changed. Authentication will add additional columns to user tables as login information, such as passwords, will need to be stored in a secure way (such as hashing with a salt). The team may also user established authentication services to avoid this potential security risk.

✏️ UX/UI Design Document

IT Project - UX/UI Design Document

Team Members: Ayush Tyagi, Ke Liao, Jack Perry, Josh Costa, Bryce Copeland

---

Table of Contents:

Introduction

This document serves as a separate extension to the design documentation for the project to explicitly and solely contain information & artifacts pertaining to the UX/UI design process. Given the agile nature of requirements and the limited deadline for this project, this document will be segmented into sections/designs based on the different stages of required functionality. Each design contained in this document will contain a conceptual wireframe of the structure & navigation flow of user interface. The designs will also have a list of key functional components for the front-end and their purposes. In the case of designs that are building upon a previous, those designs will clearly outline the new components and the rationale for their inclusion in the user interface.

Design #1 - Essential Requirements

Wireframe

The wireframe for the design that only encapsulates the essentially requirements acts as the baseline for all future designs in terms of functionality flows and capabilities (the minimum viable product, or the MVP). A partial and full annotated wireframe for this design are depicted in Figures 1 and 2 respectively below.

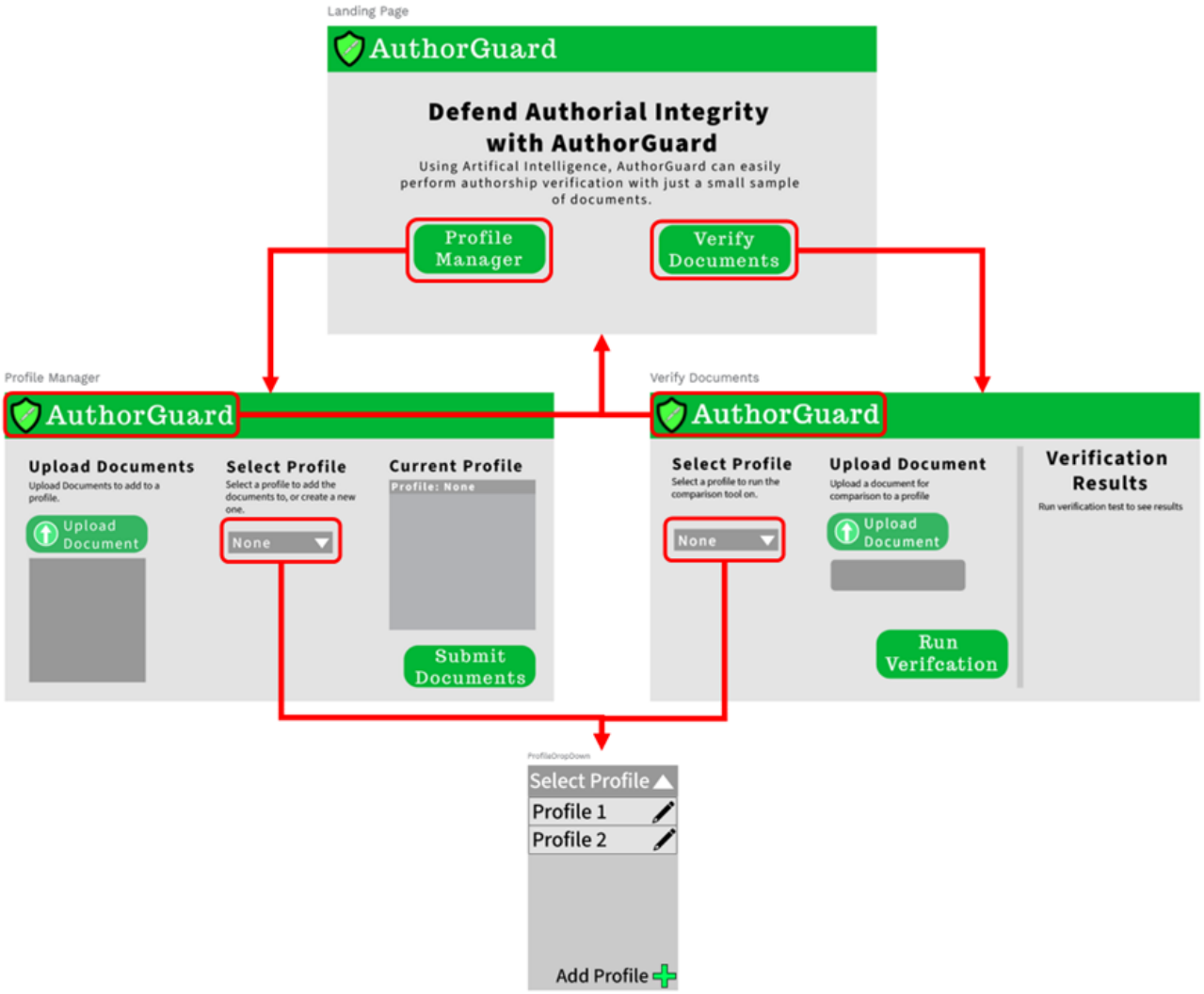


Figure 1 - Essential Requirements UX/UI Partial-Annotation Wireframe

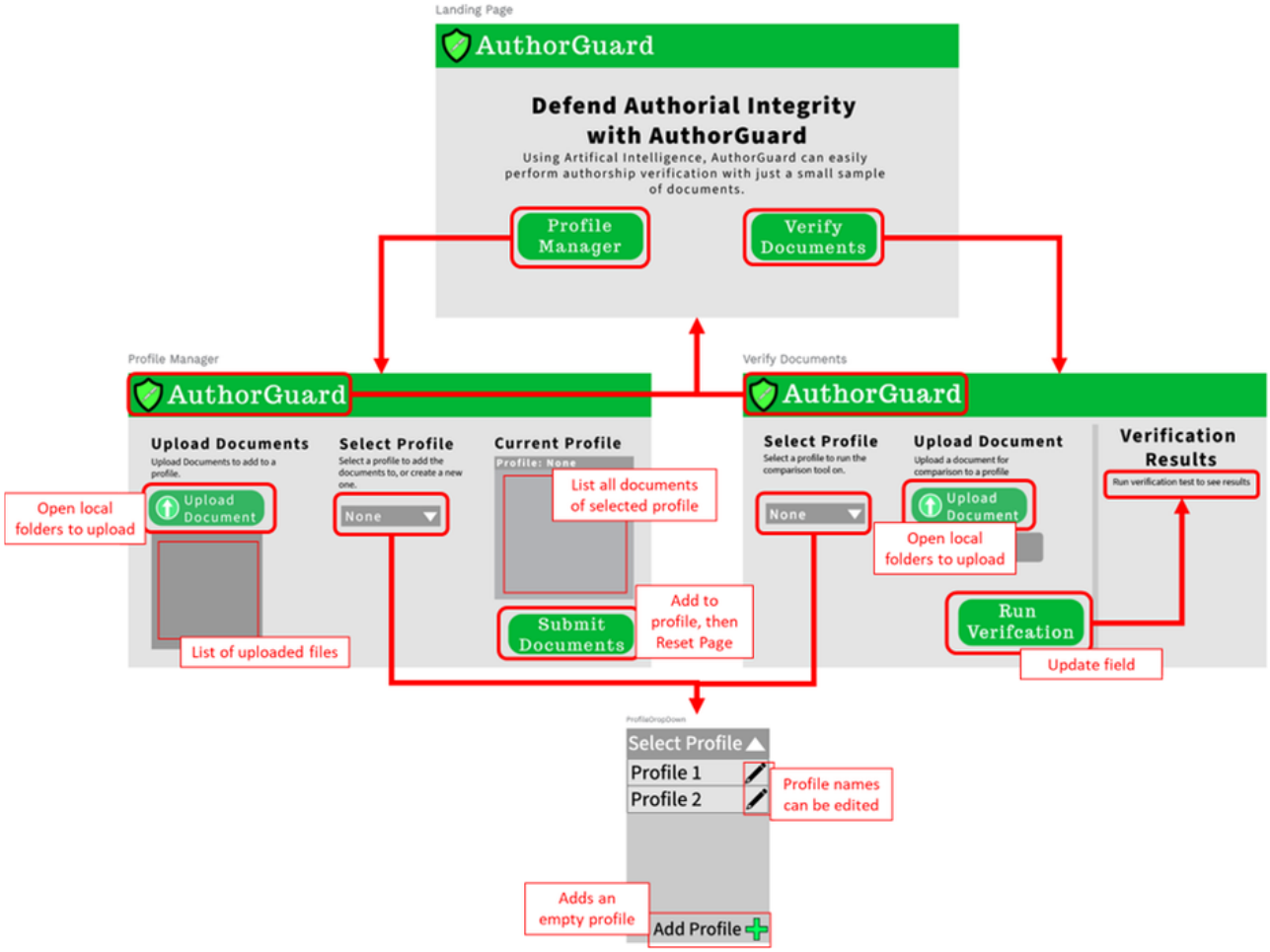


Figure 2 - Essential Requirements UX/UI Full-Annotation Wireframe

Key Components

The key components of this design are the following:

- Buttons: Enable navigation between pages, trigger an instruction (upload document from user machine, update profile, trigger a verification test.
- Drop-Down List: List all profiles are user has, dynamic profile addition, dynamic profile renaming, profile selection
- Document Lists: List the documents that are uploaded, list documents currently in a profile.

📘 Decision Log/Document

Table of Contents

Introduction

This document aims to record all the pivotal decisions made throughout the project's development through a series of logs of key decisions, their rationale, impact and progress.

Decision Log

The following table is a record of all the decisions made by the team in the context of their date, what the decision was, the context of the decision, the rationale behind the decision, the immediate impact that this decision will have, the implementation methods the team will use to enforce this decision and finally the status of the decision if it requires ongoing action.

Inception Stage Decisions: 28/07/2023 → 07/08/2023						
Date	**Decision**	**Context**	**Rationale**	**Impact**	**Implementation**	**Status**
28/07/2023	Productivity & Communication Tools Decision	What communication tools should be used for the project	To ensure simple workflow	Additional overhead during inception	Create and share various tools (Confluence, Jira, Slack, Github, Penpot, Xtensio, Lucidchart, etc.)	Completed
28/07/2023	Development Methodology Decided Upon	The team decided to implement the Agile methodology	To align with modern industry development processes	Agile ceremonies/processes will be observed & practiced actively	Ensure agile methods are respected & implemented, recorded documentation	Completed
04/08/2023	Assigned Team Roles Decision	Assign roles & responsibilities to team members	To ease division of labor and allow for responsibility management	Different team members have differing responsibilities	Record on Confluence, act upon for project.	Completed
04/08/2023	Decided upon conceptual project aims	After client meeting, high-level project goals decided	To guide the direction all future work	Will form the foundation of future decisions	Record decisions in project aims	Completed

Sprint 1 Decisions: 08/08/2023 → 25/08/2023						
Date	**Decision**	**Context**	**Rationale**	**Impact**	**Implementation**	**Status**
08/08/2023	First Sprint Commenced (09/08/2023 → 25/08/2023)	The First Sprint, Project Inception, was decided to begin	To begin project inception formally	Jira Backlog populated, responsibilities assigned	Backlogs guides tasks to be completed	Completed
08/08/2023	Functional & Non-Functional Requirements Decided Upon	The functional & non-functional requirements were agreed upon	To allow for the creation of comprehensive requirements document.	Requirements documentation work added to backlog.	Create artifacts such as motivational model, requirement tables, user stories, personas	Completed
08/08/2023	Conceptual Architectural Design Decided Upon	The conceptual architecture/structure of the project was agreed upon	To allow for architectural analysis & design modelling	Design documentation work added to backlog	Create artifacts such as domain models, architecture models.	Completed
11/08/2023	UI Sketches Decided Upon	From a series of candidates, a basic sketch of UX/UI was decided to be prototyped	To allow for the creation of the prototype and UX/UI design document	Common Vision for UX/UI design & front-end design	Create prototype artifact and document in UX/UI Design Document	Completed
15/08/2023	User Stories & Requirements Document Finalization	Key requirements have been finalized within current context	To allow productivity to continue to other areas	User Stories & Scope are more clearly defined	Ensure Requirements Documentation is comprehensive & complete	Completed
15/08/2023	Product Name & Basic UX/UI Documentation Finalized	Basic UX/UI design document created & approved by team, inc. Web app name (AuthorGuard)	To allow for development on front-end and common vision	Use new project name when required, implement as per UX/UI specification	Follow UX/UI document for front-end development	Completed
18/08/2023	Model Weight Decision	Team performed testing on provided codebase with model weights to ‘save’ a trained model	To allow the machine learning to be used without needing to retrain	The team will create the API with a saved model weights file	Obtain a trained model and create API that access aforementioned model	Completed
22/08/2023	Question List Decision	Team prepared for client meeting by formalizing list	In order to ensure productivity	Question list will need to be followed during the client meeting	Follow questions, but still remain agile in allowing some free-flow	Completed

		of talking points & questions to ask.	during the client meeting.	for maximum productivity/efficiency.	discussion, but stick to the schedule.	
25/08/2023	Client Approval Decision	The Team showcased all requirement & design artifacts to the client	To seek the stamp of approval from the client in regards to the scope of the project.	The client gave approval for the requirements & design scope hence development can begin.	Sprint 2 will be focused on the development of the web application.	Completed
25/08/2023	Sprint 1 Review & Retrospective Future Goals Decision	The team performed a Sprint Review & Retrospective for Sprint 1	To conclude the first sprint of development and reflect upon progress and adapt accordingly.	The team set conceptual goals and strategies for the next sprint.	The goals and strategies will be broken down and discussed during Sprint 2 planning.	Completed
Sprint 2 Decisions: 29/08/2023 → 22/09/2023						
Date	**Decision**	**Context**	**Rationale**	**Impact**	**Implementation**	**Status**
29/08/2023	Second Sprint Commenced (29/08 → 22/09)	The Second Sprint, development was decided to begin.	To initiate the formal development stage of the project.	Jira Backlog updated with new user stories and tasks to be performed.	Backlog dictates what tasks need to be completed for development	In Progress
29/08/2023	Decision to use Django	The team decided to use Django Framework	To allow for the development of architectural components with full-stack awareness.	The project will be a Django based application, changes development methods & deployment options	Django project will need to be setup in GitHub repository, team needs familiarity with Django	Completed
29/08/2023	Peer Programming, Code Style & Code Review Decision	The team agreed upon principles for these agile concepts.	In order to improve development with agile methodology.	Team will need to abide by the agreements outlined in the documentation.	Records of Coding Documents will be new artefacts added to the Confluence.	In Progress

💻 Coding Documents

This is the landing page for the various coding relating documents for the development of the AuthorGuard web application.

Document Name	Key Purposes	Link
Code Style Document	Code Style Guides, Coding Practices	⌨️ Code Style Document
Peer Programming Document	Peer Programming Rationale, Guide & Process	🤝 Peer Programming Document
Code Review Document	Code Review Rationale, Guide & Process	👓 Code Review Document

⌨️ Code Style Document

IT Project - Code Style Document

Team Members: Ayush Tyagi, Ke Liao, Jack Perry, Josh Costa, Bryce Copeland

---

Table of Contents

Overview

It is important to maintain readable code for the purposes of this project. This is to primarily ensure that team members all agree upon a consistent style and can easily read each other's code, but also to ensure that any future work on the project can understand the code base, which is the aim of this document.

This document provides a comprehensive guide to the code styles used for the purposes of the development of this project. These code styles align with the popular conventions used in industry. The team will use **Prettier** to enforce the code style for all code relating to the project.

Code Style Guide

Python

For Python, the PEP8 code style conventions should be followed as much as possible. This is useful as it is universally recognised and all team members have a strong level of experience and expertise with the formatting of PEP8. The following list outlines the key components of the PEP8 style guide

- Variable/Function Names: Underscores
- Class Names: Pascal Case
- Constants: Uppercase with underscores

```
my_variable
def my_function():

class MyClass():

MY_CONSTANT
```

- Line Limit: 79 characters
- Indentation: 4 character indentation per level

```
if (statement):
    indented code
```

- Comments
  - Use docstrings for classes, functions, methods

    - Use docstrings for classes, functions, methods
    - Use comments for complex code, clear and concise
- Import: Standard Modules, then external modules, newline per import
- Organise code into clear functions and classes when required
- Formatting
    - Spaces around operators
    - Space after comma
    - Align related lines vertically
- Git/GitHub (Version Control)
    - Use branches for new features and pull requests
    - Write clear and descriptive commit messages

JavaScript:

For JavaScript, the team will keep it as similar as possible to python conventions for convenience of understanding, with exceptions for some fundamental standard conventions of JavaScript.

- Variable/Function Names: camel case
- Class Names: Camel Case
- Constants: Uppercase with underscores

```
myVariable
function myFunction() {}

class MyClass {}

MY_CONSTANT
```

- Line Limit: 79 characters
- Indentation: 4 character indentation per level

```
if (statement) {
    indented code
}
```

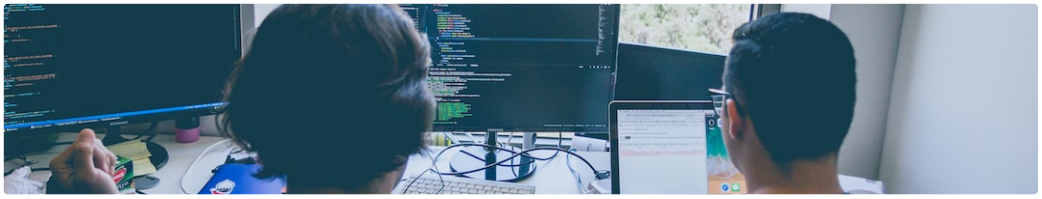
- Comments
    - Use /**..*/ for classes, functions, methods
    - Use comments for complex code, clear and concise
- Imports
    - Only import what is required
    - Use import over require generally
- Organize code into clear functions and classes when required
- Formatting
    - Spaces around operators
    - Space after comma
    - Align related lines vertically
- Git/GitHub (Version Control)
    - Use branches for new features and pull requests
    - Write clear and descriptive commit messages

HTML/CSS:

- Variable/Attribute Names: lower case separated by hyphen(-)
- Version Control:
  - Same as other
- Line Length: 120
- Blank line at the end of file.
- Indentation: 4 char per indentation level

```
<body>
    <p>
        penguin
    </p>
</body>
```

- Use <!--*content* → for commenting purposes
  - Use comment to help clarify HTML blocks
  - Use comments for complex code, clear and concise
- Space around equal signs



🤝 Peer Programming Document

IT Project - Peer Programming Document

Team Members: Ayush Tyagi, Ke Liao, Jack Perry, Josh Costa, Bryce Copeland

---

Table of Contents

Overview

In alignment with the Agile methodology, it is important for the team to engage in Peer Programming throughout the code development stage of the project's lifespan. Peer programming involves two programmers working together on the same code to write it together. It is important to remember that despite the fundamental guidelines outlined in this document, it is important that the team remains flexible to change and is willing to alter the guidelines when it appears appropriate to do so.

One of the main reasons the team is engaging in peer programming is to maximize knowledge sharing within the team. Given the range of expertise within the team and the member's willingness and motivation to learn, it is important to cultivate a culture of learning throughout the project's development and hence peer programming is a highly effective way to allow the distribution of skills within the team.

Furthermore, one of the other main motivating factors for peer programming is to increase the quality of the code and prevent bugs. Although ideally these aspects will be covered again during code reviews, coding in a cooperative way will help reduce the burden of the code reviews, which will lead to an increase in productivity of the development, which is of utmost importance for a project on such a limited time frame.

Beyond this, peer programming also enables better team communication and ensures that team members are well informed as to the whole scope of the project as they will understand more elements of the code base.

Roles and Responsibilities

For the purposes of this project, to align with proper agile methodology, the team will generally use the following roles throughout peer programming. However, it is worth reiterating that these roles are flexible, and during a session, the roles can swap to allow for more flexible working and ensure both team members are focused throughout the programming session.

The Driver

The driver is the person who is writing the code during the peer programming session. They can be thought of as the 'scribe', who converts the ideas of discussion into machine-operable code.

The main responsibilities of the driver include:

- Writing the code based on the discourse between the team members
- Adhering to the code style guide
- Navigating the technical implementation details & intricacies

- Incorporate new ideas into the code base

The Navigator

The navigator is the person who is reviewing the code during the peer programming session. They mainly serve to offer ideas to the driver and double-check the working of the driver throughout the session.

The main responsibilities of the navigator include:

- Active focus and review of the code in the session
- Envision the future code to write with respect to the goals
- Offer ideas and feedback to the driver
- Locate bugs and discuss solutions

Procedure

The procedure of a peer programming session will generally follow the structure and steps below, but it is important that the team is flexible in regards to this and is willing to deviate when the productivity of the sessions requires them to do so.

Preparation

- Setup development environment, branch.
- Discuss goals for the session and estimate time that the goals will take to implement
- Assign roles (driver & navigator)

Implementation

- Discuss the first goal/user story to implement and the requirements of the user story
- Brainstorm ideas for an approach and decide on the best option
- Driver writes code whilst navigator reviews code
- Active communication during session
- Swap roles after a time period or after a goal is achieved

Review

- Review the progress during the peer programming session
- Make any finals changes and commit to branch
- Prepare for code review

Record Policy

During the session, notes should be taken for the session in terms of key discussion, decisions and changes made to the code base. It is also important to document the key knowledge discovered and provide a rationale for the solutions implemented.

These notes should be recorded as per this format for the team to see and any critical insights made can be discussed in the upcoming team meetings as a notable discussion point if it has implications for the scope of the project.

In terms of code change records, the team should clearly use comments throughout the code when appropriate to improve readability and allow for other team members to further develop the code base. It is also important that the team member’s use commit messages that are descriptive enough to capture the key changes made to the code base so that team members can understand these changes at a conceptual level easily.

👓 Code Review Document

IT Project - Code Review Document

Team Members: Ayush Tyagi, Ke Liao, Jack Perry, Josh Costa, Bryce Copeland

---

Table of Contents

Overview:

The purpose of Code Reviews is to ensure the quality, correctness and readability of the code by having fellow peers review it. Having group members evaluate each other's code helps reduce errors and promotes sharing of knowledge, which benefits all members of the team.

Roles & Responsibility:

Author: Writes code to the best of their ability, following code style requirements and providing helpful comments. Inform the group when code is ready to be reviewed and be open to constructive criticism and feedback.

Reviewer: Thoroughly examines the code, ensuring code quality, correctness and readability are maintained. Provide concise, constructive feedback to the author to help them improve.

Timing & Deadlines:

Pending branch merges should be reviewed in a timely manner to ensure the main branch is kept up to date with successfully reviewed code. It is the code author’s responsibility to inform group members that their code is ready to be reviewed, likewise it is the reviewer’s responsibility to review the code as soon as possible and resolve any issues in order to avoid unnecessary delays.

Workflow:

Authors should commit their code on a separate branch, and inform the group when their code is ready for review. Once a peer member has reviewed the code, the author and reviewer should discuss any areas of concern or potential ways to improve before making the required improvements and finally, merging onto main branch.

Reviewer Considerations:

When reviewing code, the reviewer should keep the following points in mind:

- Code Style, does the code follow the requirements outlined in the Code Style document
- Functionality, does the code function as intended?
- Readability, is the code easy to follow and understand?

Code Review Tools:

- GitHub: assists with version control and enables separate branches and merging after code is reviewed
- Slack: enables communication between group members to indicate when code is ready to be reviewed
- Zoom: members can voice call in order to quickly resolve any issues within the code before merging

Communication:

Group members are expected to communicate in a productive way, providing constructive feedback with the aim of resolving issues, sharing knowledge and benefitting the team. Conflict resolution will be handled on a case by case basis, where minor conflicts are discussed and dealt with between author and reviewer. In the unlikely event of a major conflict or issue, the case can be discussed at the next team meeting to ensure a fair decision is reached.