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:

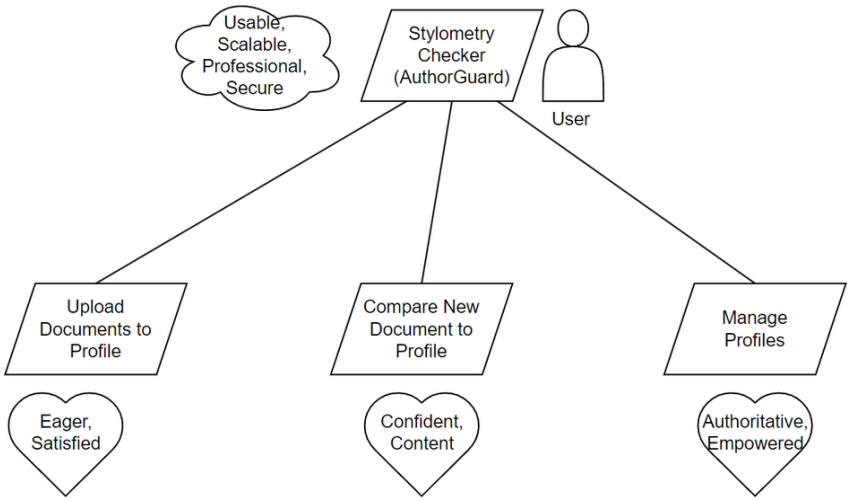


Figure 1 - Motivational Model of Minimum Viable Product

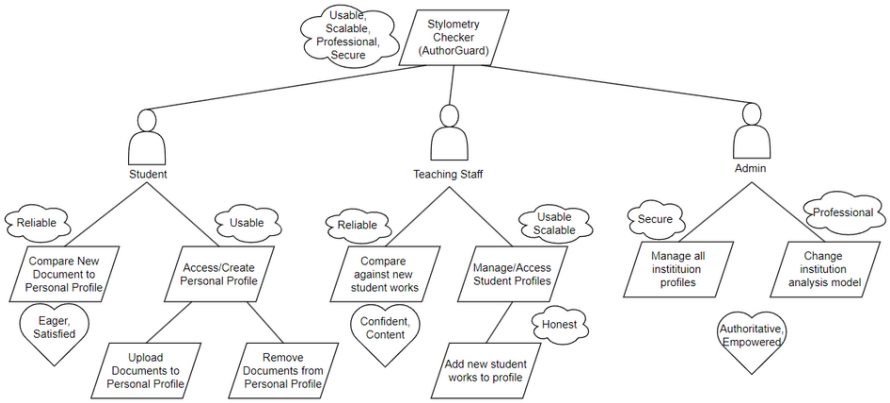


Figure 2 - Motivational Model ft. Stretch Goals & User Stories

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 • Accuracy of results displayed	Medium	Medium: • 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 student/teacher, 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	High

	• The resulting writing style similarity percentage is displayed to user	
As a student/teacher, 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 student/teacher, 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 student/teacher, 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 student, 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 student/teacher, 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.

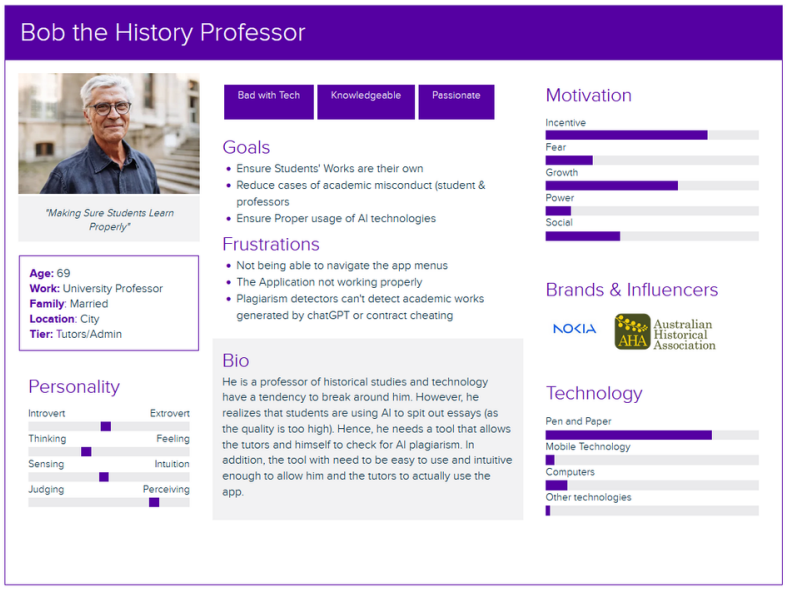


Figure 3 - Bob the History Professor Persona

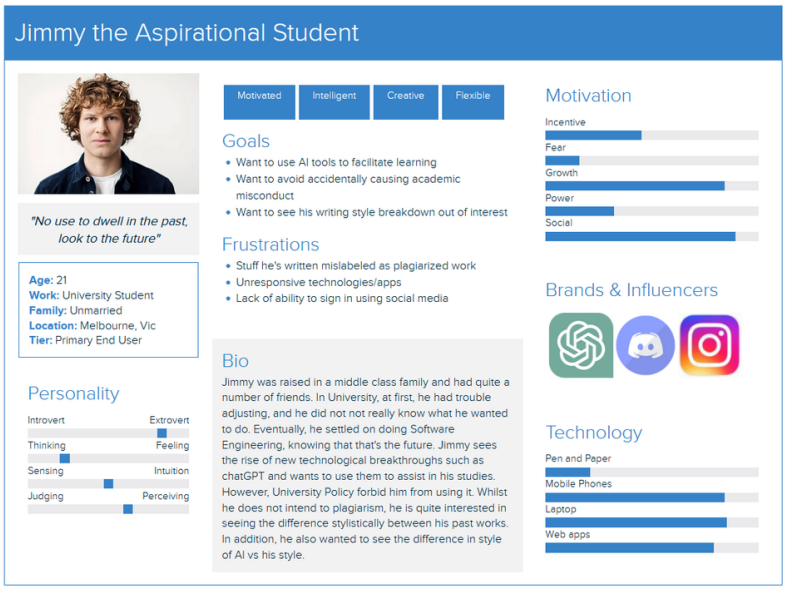
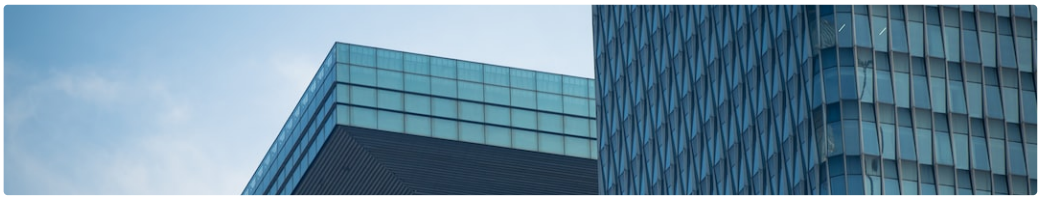


Figure 4 - Jimmy the Aspirational Student Persona



🖋 Design Document

IT Project - Design Document

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

---

Table of Contents:

Introduction

This document seeks to outline the specific design considerations and decisions made in regard to the delivery of the project at a high level. This document aims to provide clarity on the overall system design and serves to guide the development & deployment of the application during later stages of the project. This document should be used in tandem with the Requirements Document to gain a full understanding of the scope, goals and motivations for the project and the consequential decisions during the delivery of the project.

System Overview

The core purpose of the system is to provide a usable interface for non-technical people to have access to a novel authorship verification algorithm. The idea is to provide a way for an academic audience, namely students and educators, from non-technical disciplines to be able to upload their own documents to the system and create a profile.

Users will then be able to use a profile's documents to compare with a newly uploaded document and receive meaningful and interpretable results from the algorithm informing them as to the likelihood the new document was written by the same person that wrote all documents on the profile they are comparing against. This will be achieved via web-based application that users will be able to open and use in an intuitive way.

A more comprehensive list and scope of the requirements & user-stories can be found on the Requirements Document. This section mainly only serves to provide a more concise version of the general purpose of the system at a conceptual level.

System Architecture

At a conceptual level, the overall design & functionality of the system is mapped into components adjacent to those in Figure 1 below. This ideal model follows the standard methodology/principles of full-stack software design and development.

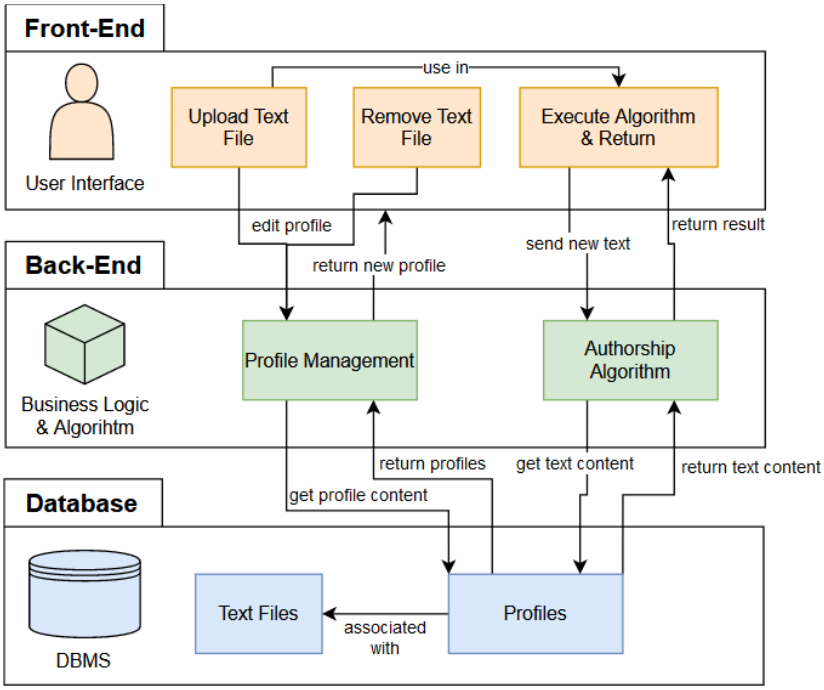


Figure 1 - Ideal Architectural Components & Basic Functions

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.

Conceptual 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.

Note - The specific deployment strategy agreed upon by the team can be found in the Deployment Document, which outlines the architecture and services that will be used by the web application including an updated diagram of the system architecture

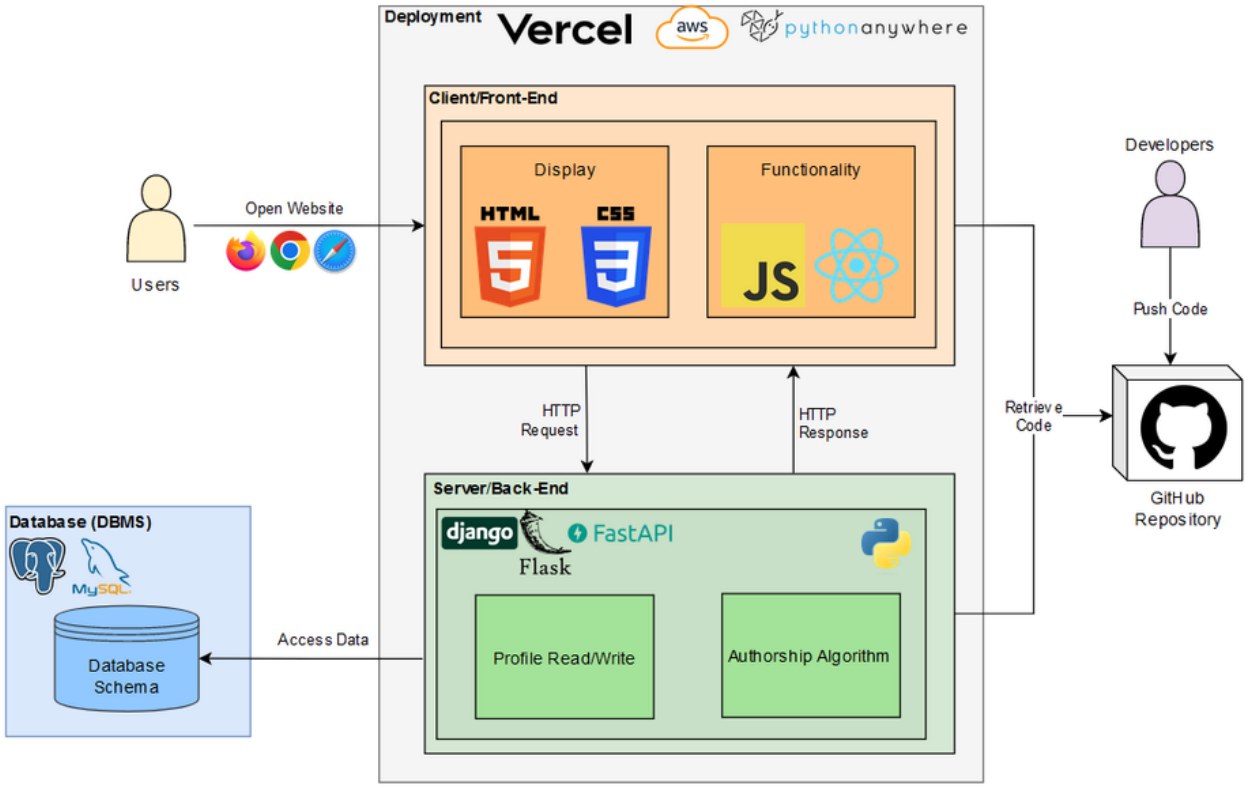


Figure 2 - System Architecture Options

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.

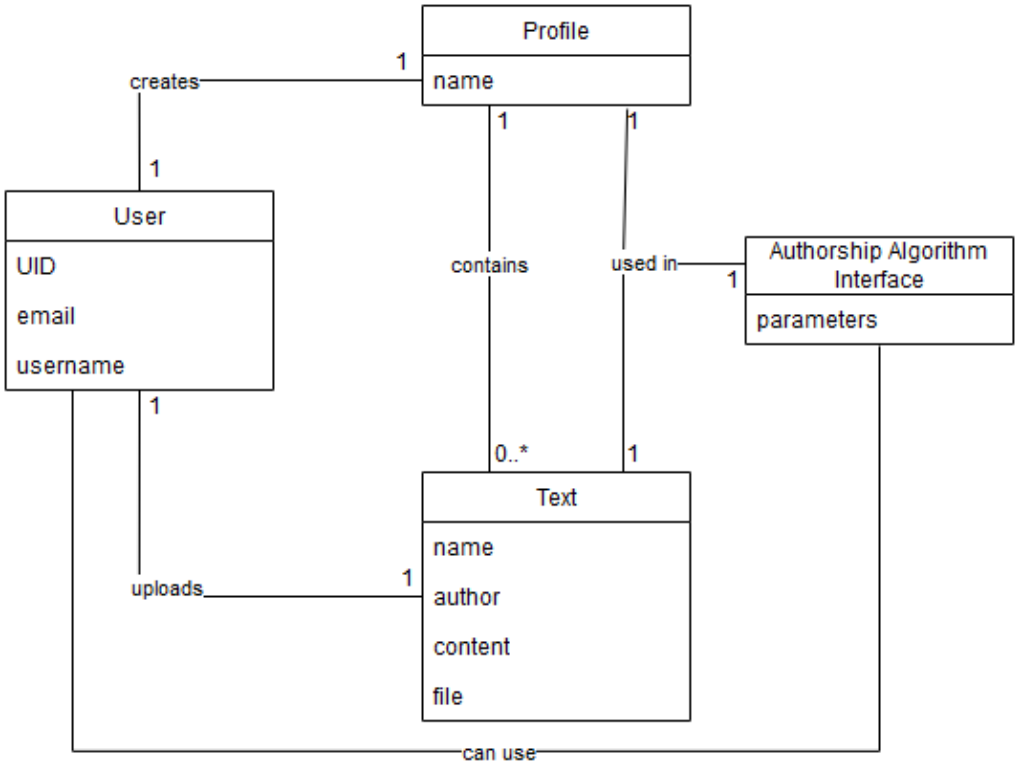


Figure 3 - Basic System Dynamics (Minimum Viable Product Domain Model)

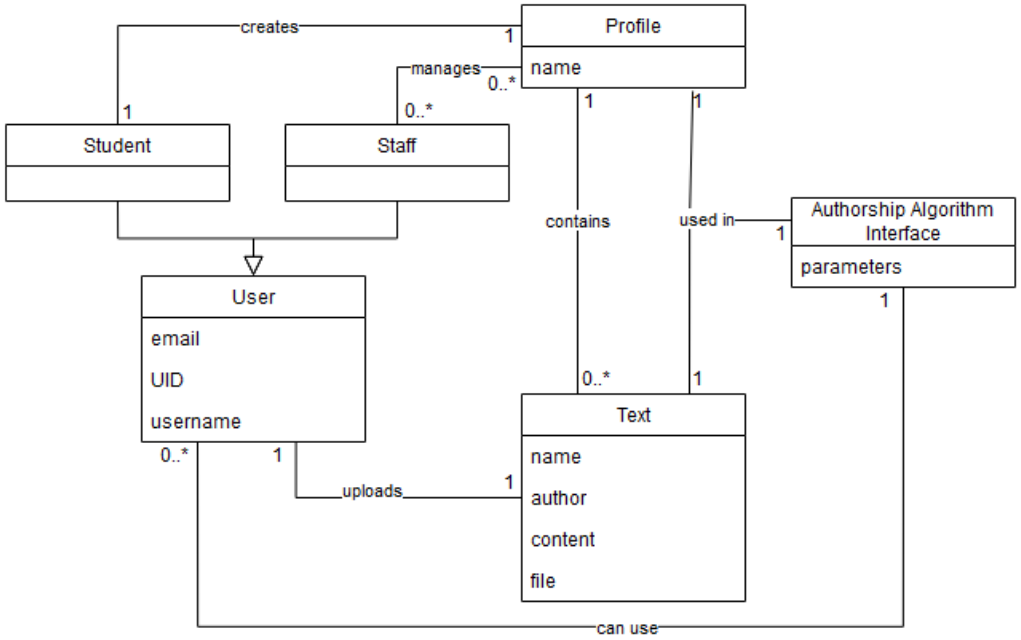


Figure 4 - Extended System Dynamics (Stretch Goals Domain Model)

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.

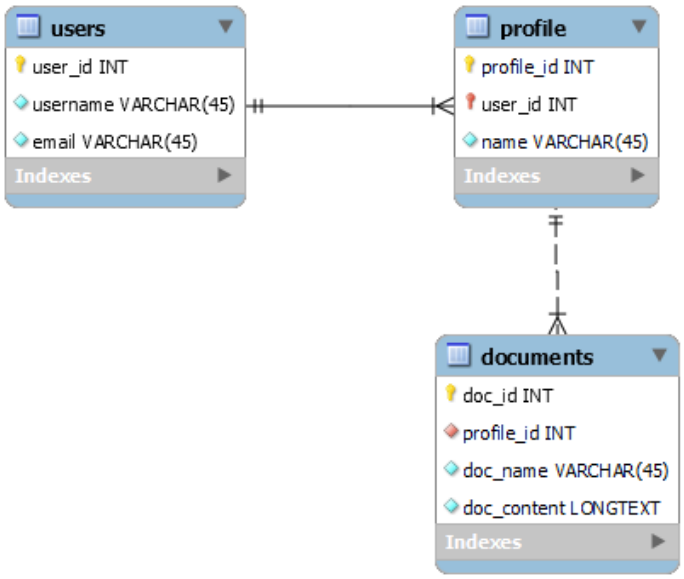


Figure 5 - Simple Design Database Diagram

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.

Essential Requirements Prototype

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.

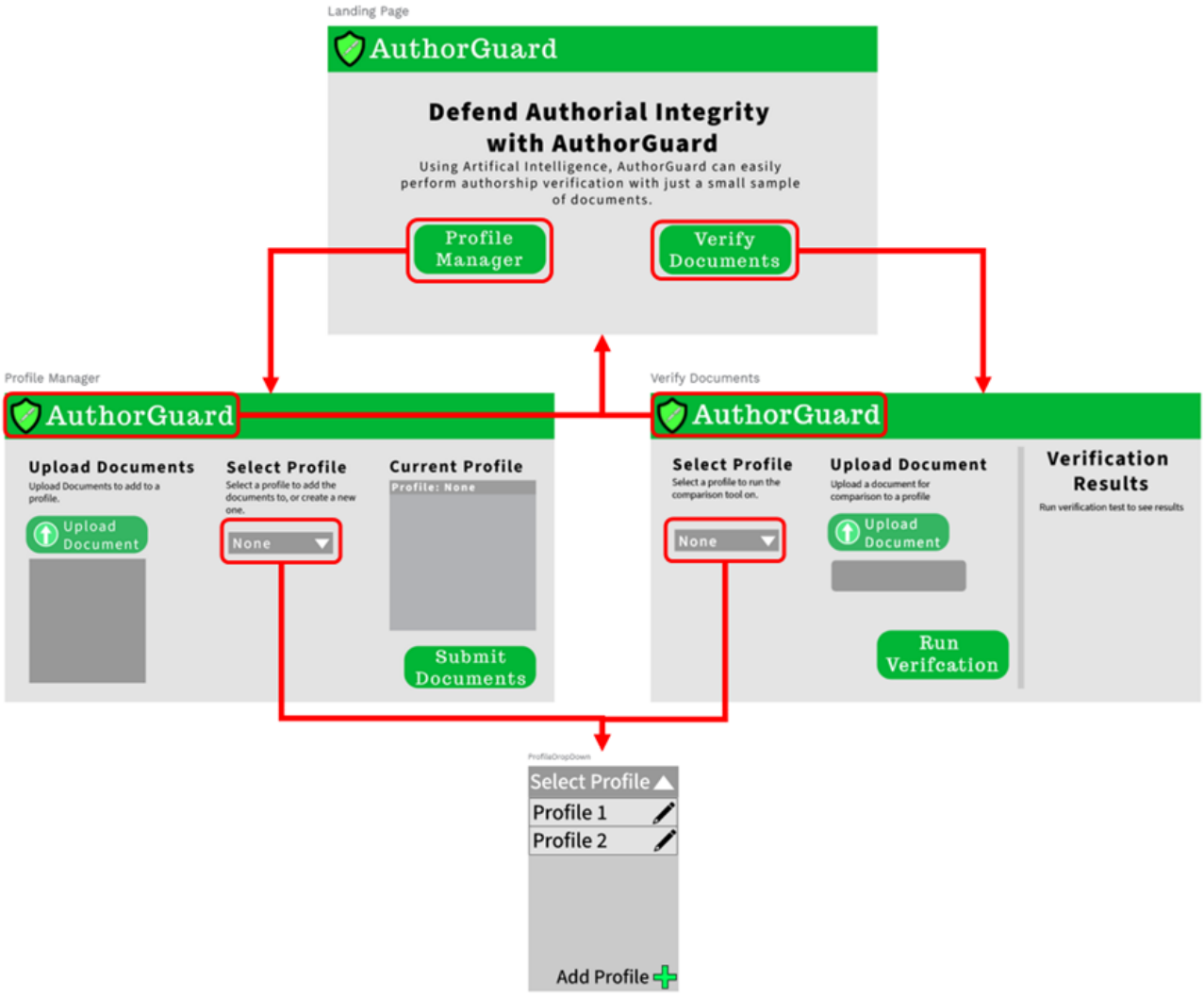


Figure 1 - Essential Requirements UX/UI Partial-Annotation Wireframe

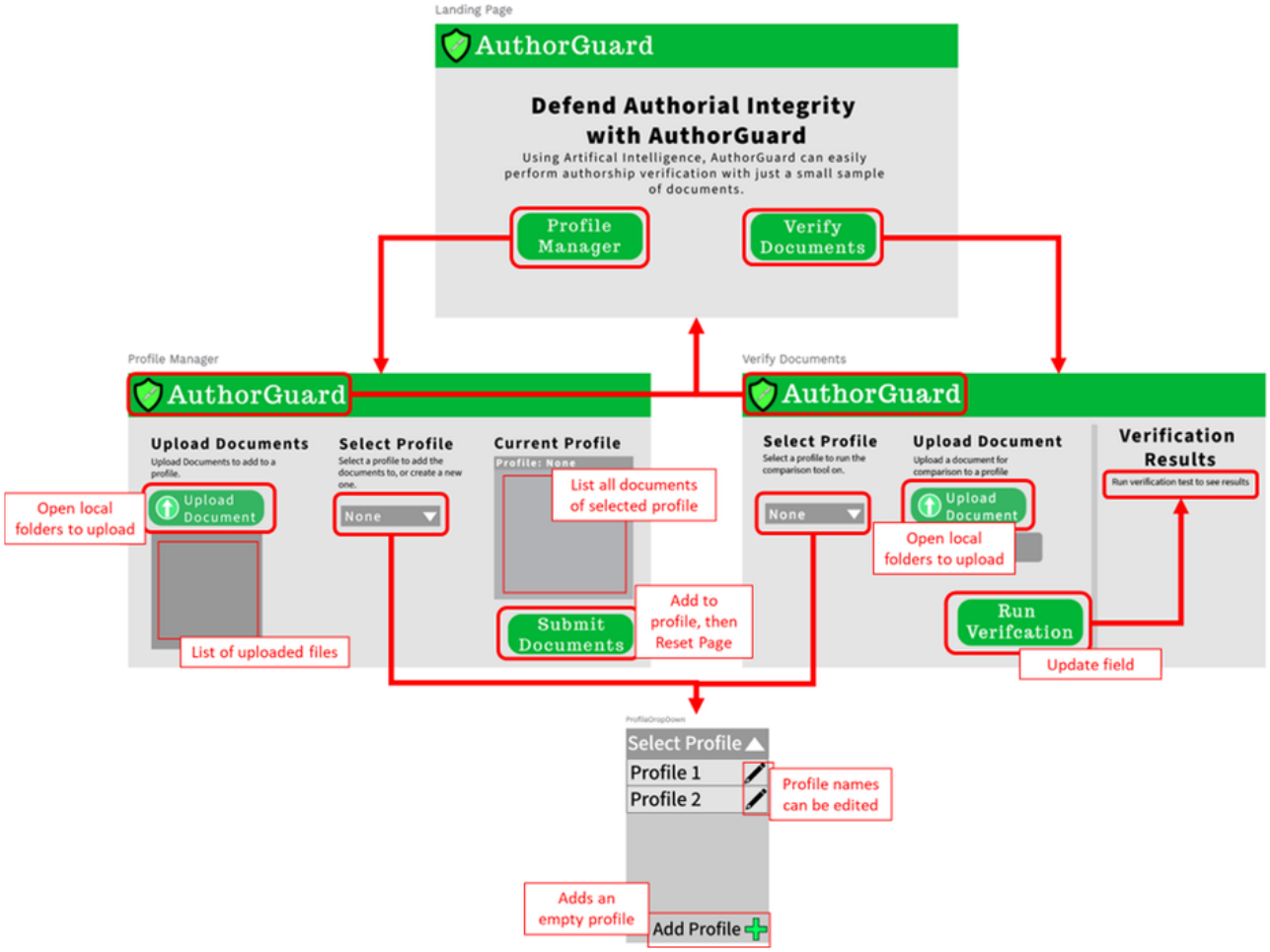


Figure 2 - Essential Requirements UX/UI Full-Annotation Wireframe

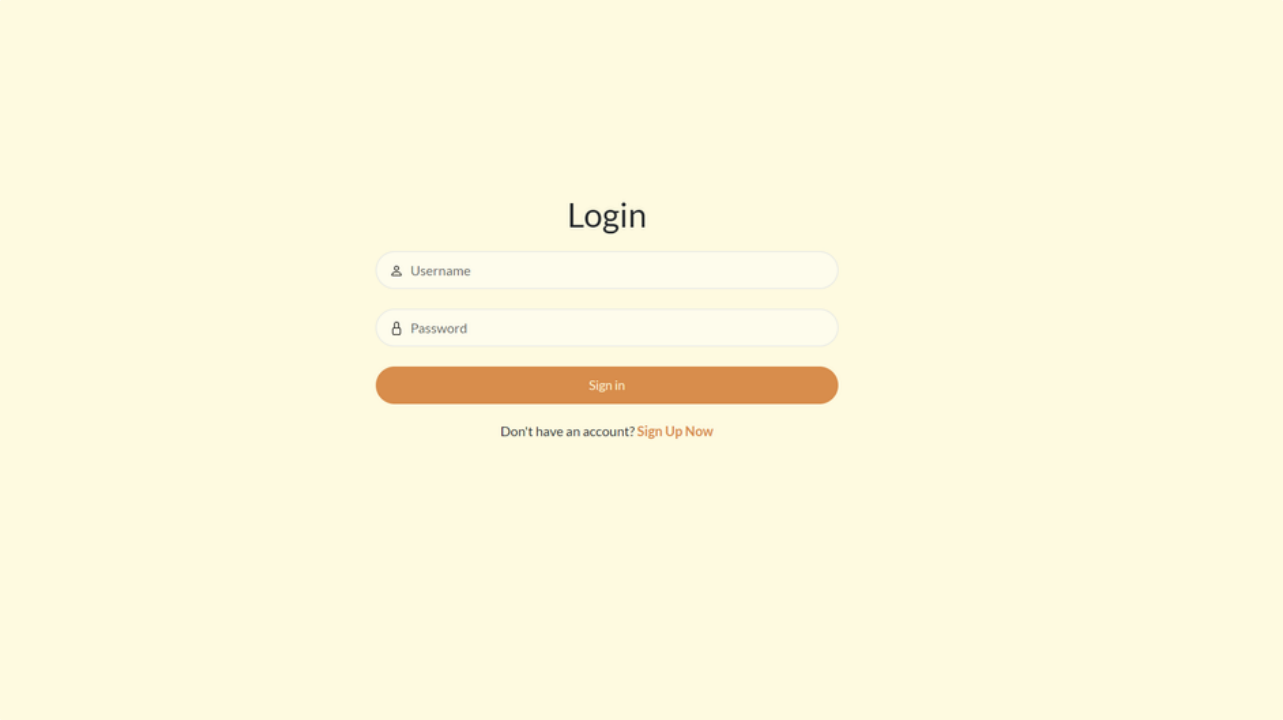
Key Components

The key components of this design are the following:

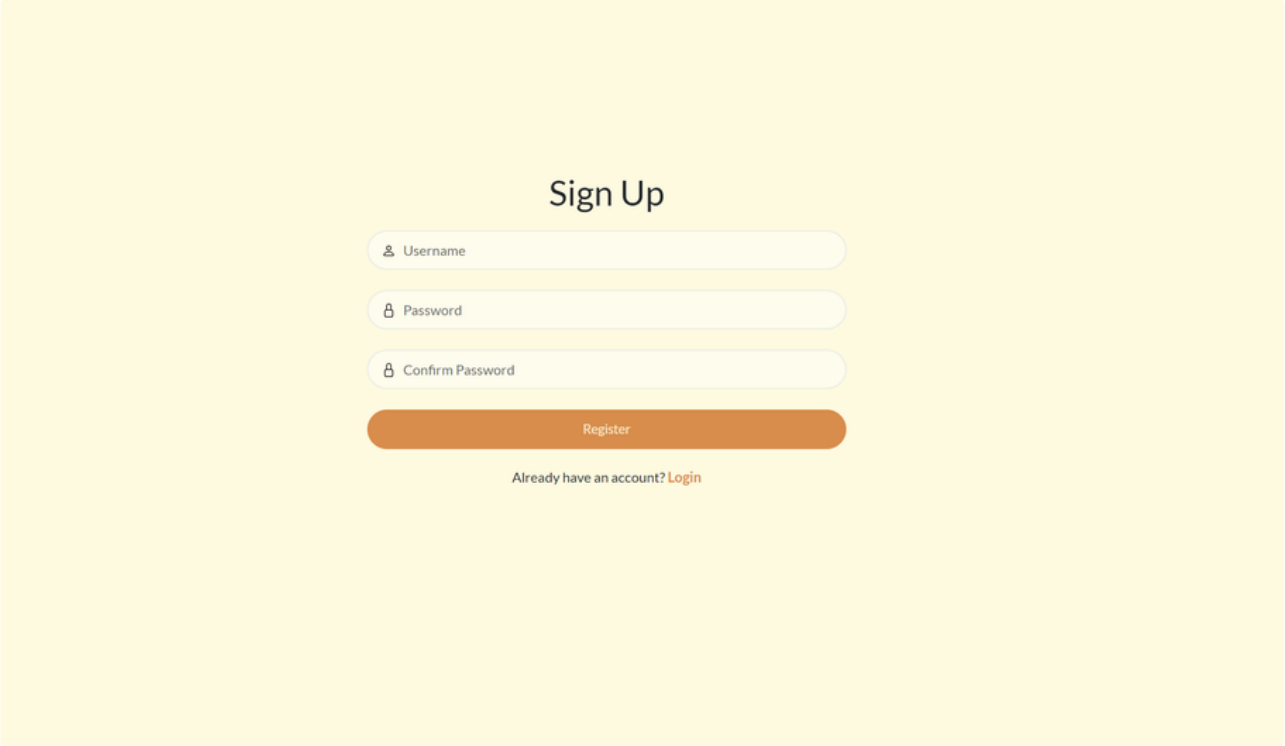
- Buttons: Enable navigation between pages, trigger an instruction (upload document from user machine, update profile, trigger a verification test.
- Drop-Down List: List all profiles are user has, dynamic profile addition, dynamic profile renaming, profile selection
- Document Lists: List the documents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

Implemented Design (Sprint 2)

The following images represent the current implementation of the frontend design as of 24/09/23. This design mimics the prototype in terms of functionality and features but is a more attractive design visually for an enhanced user experience.

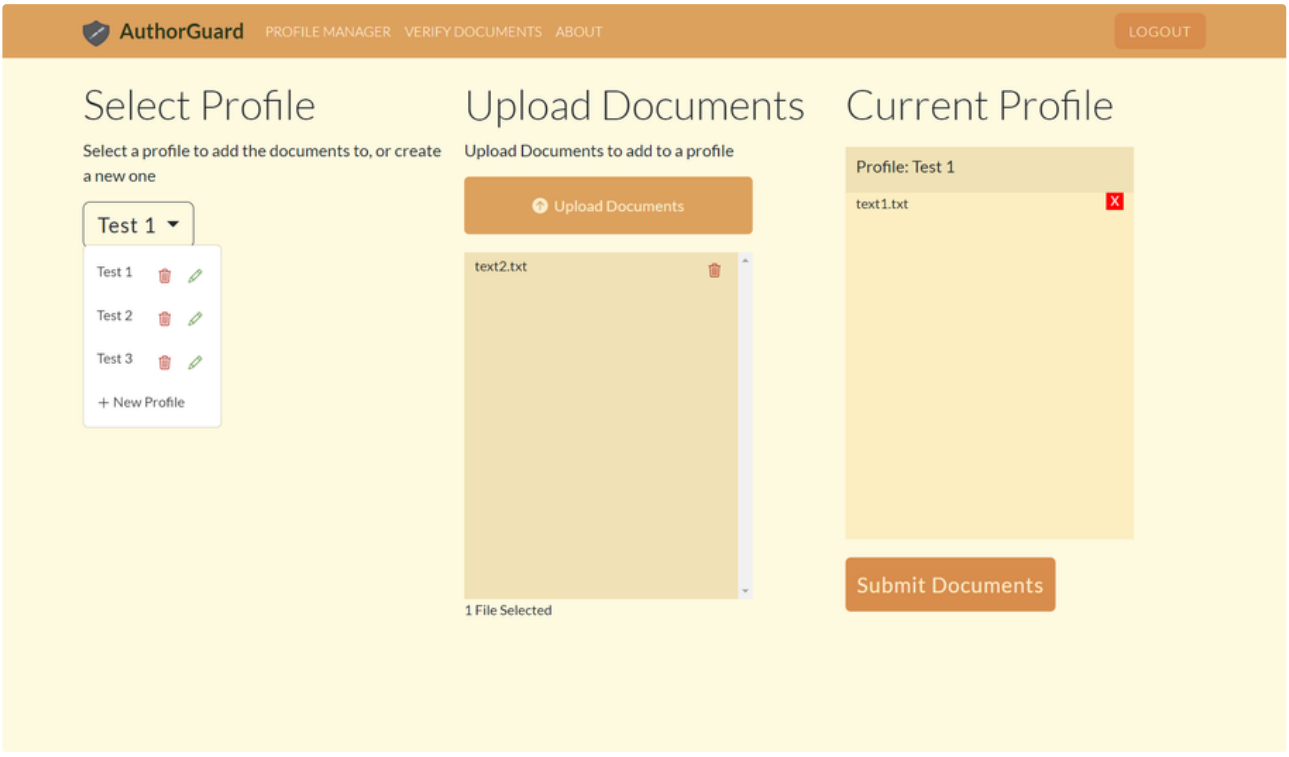


Login Page

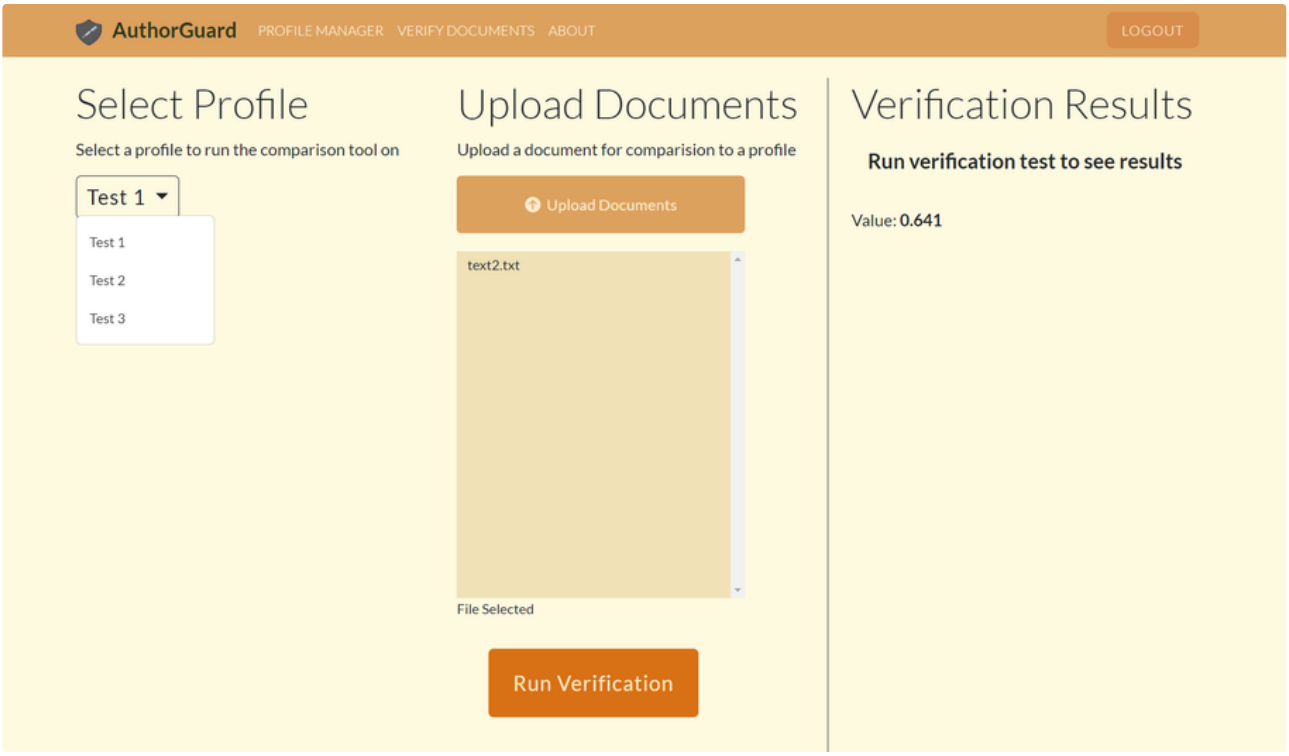


Registration Page

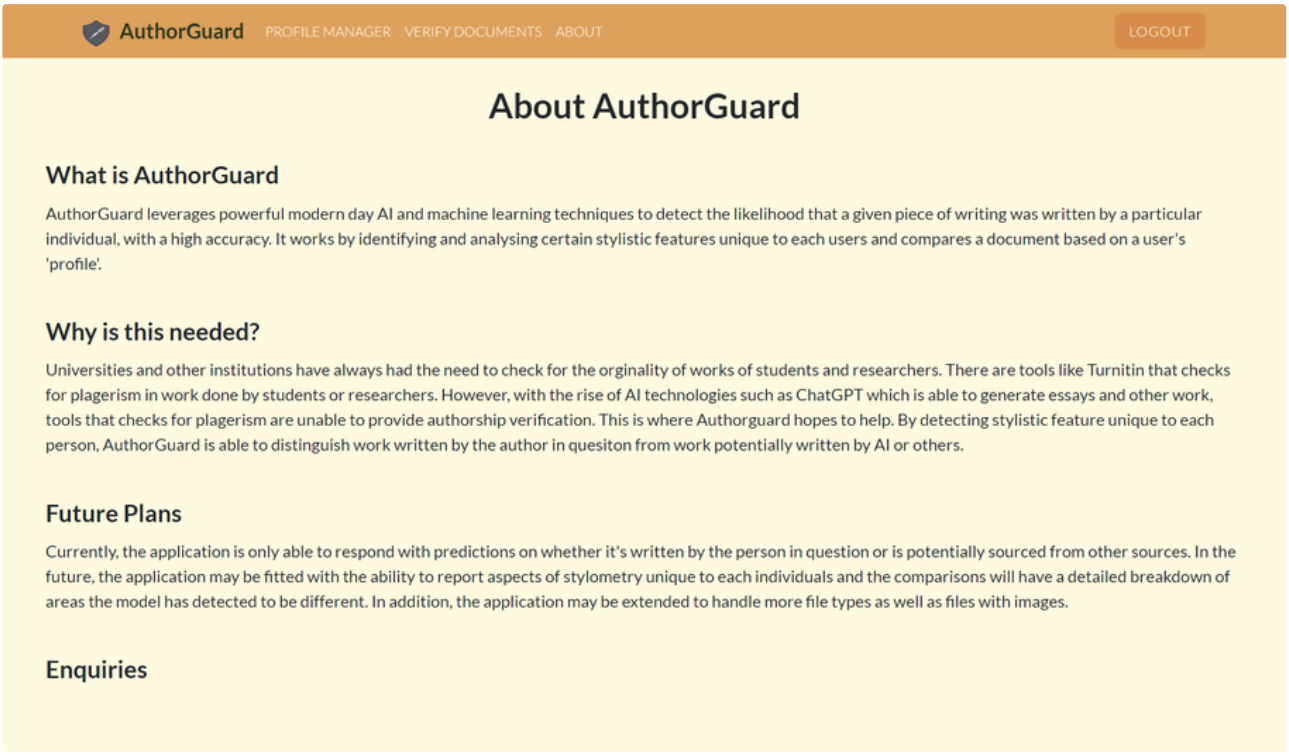
Landing Page



Profile Manager Page



Verify Document Page



About Page

Design Resources & Research:

- Colour Palettes: Popular Color Palettes - Coolors
- 9 Of The Best UI Design Examples [2023 Inspiration]

📘 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-	To guide the direction all future work	Will form the foundation of future decisions	Record decisions in project aims	Completed

		level project goals decided				
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 of talking points & questions to ask.	In order to ensure productivity during the client meeting.	Question list will need to be followed during the client meeting for maximum productivity/efficiency.	Follow questions, but still remain agile in allowing some free-flow discussion, but stick to the schedule.	Completed
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.	Completed
01/09/2023	Submission of progress assessments	The progress assessment	To fulfill the project	Added documents to checklist.	Continue to update documents	Completed

		checklist were submitted	assessment requirements.		references in checklist	
05/09/2023	Decision to use standard HTML/CSS/JS front-end with JQuery/AJAX requests	The team decided to use standard web development as opposed to a framework like React	The team is more familiar with these languages, given short development time, lower learning curve and faster deliverables	The team should further learn more about these languages int he context of full-stack development	Team will act upon this decision through learning and using these languages for development.	Completed
08/09/2023	Basic Django App Functionality Decided	The team agreed upon the basic starting point for the Django app	To satisfy the user stories required for the MVP	Team will further familiarize themselves with Django and code base.	The team will extend upon this app to add additional desired user stories	Completed
08/09/2023	Code Review Record format decided	The code review record format was agreed upon	To ensure that reviews and responsibility is well-recorded	The team will use this record from now on	When a reviewer completes a review, they should record it	Completed
08/09/2023	Test Case Record format decided	The test case record format was agreed upon	To ensure consistently when it comes to testing	The team will use this record system from now on	When tests cases are created & ran, they should be documented	Completed
12/09/2023	Database Decision Finalized	The decision to use PostgreSQL was agreed upon by the team	To allow the team to start considering deployment options and DBMS specifics	Research into database hosting + integration with Django	Host database on external server, connect with Django application.	Completed
19/09/2023	Amazon RDS decided upon	The team decided to use Amazon RDS deploy the database	To allow the team to migrate out of the SQL-lite to a production ready database.	Instance will need to be created, connected	Ensure secure implementation and connectivity to database	Completed
19/09/2023	Docker Agreed Upon	The team decided to use Docker to enable deployment of the app	To allow for easy hand-off and build of the application, Docker will be used	The app will be built into a docker container and deployed for production	The team will need to use GitHub actions + Docker in order to automate building	Completed
22/09/2023	Agreed to rework front-end UI	The team decided agreed	To enhance the user experience	The team will make a number	More backlog tasks relating to	Completed

		to work on improvements to the UI based on supervisor feedback	in the final product.	of changes to the front-end to improve user attractiveness.	the front-end will need to be created and performed.	
23/09/2023	Sprint 2 Review & Retrospective Future Goals Decision	The team performed a Sprint Review & Retrospective for Sprint 2	To conclude the second 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 3 planning.	Completed

💻 Development & Deployment 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
Back-End Documentation	Documentation on information related to the back-end	📗 Back-End Documentation
Deployment Document	Details on Deployment of the Program	🏁 Deployment 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
}
```

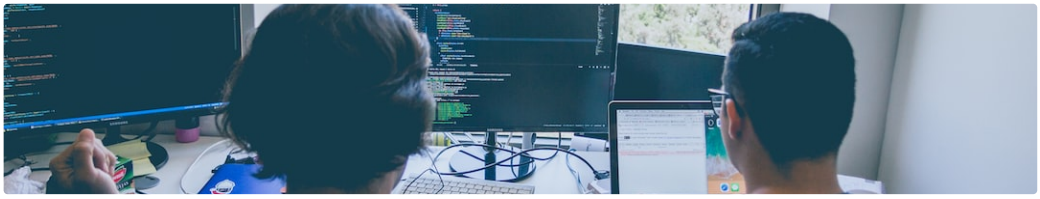
- Comments
    - Use /**..*/ for classes, functions, methods
    - Use comments for complex code, clear and concise
- Imports
    - Only import what is required
    - Use import over require generally
- Organize code into clear functions and classes when required
- Formatting
    - Spaces around operators
    - 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 feature branch merges should be reviewed in a timely manner to ensure the development 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:

When adding a new feature, authors should create a separate feature branch from the development branch, commit their changes, 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 improvements, resolve any issues, then merge back onto the development branch (squashing merge conflicts in the process).

When the team is satisfied with the current development branch and sufficient testing has been conducted, the development branch can be merged onto the main branch, where the latest full product release is deployed.

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.

Back-End Documentation

IT Project - Back-End Documentation

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

---

Table of Contents

Overview

This document contains the back-end documentation for the updated author verification machine learning algorithm code as well as the Django back-end for the web application. The author verification algorithm section contains information regarding the general approach for the notebooks that run the algorithm, and the methodology for adapting this into a usable module for Django. Likewise, the Django section contains information about the Django code base.

Note any text annotated with the `'code'` style in this document either relates to a filename, directory name or function/class name in the code base. This can all be found in the code repository found here.

Authorship Verification Machine Learning Information

Updated PAN_14 Notebook

The `'PAN_14_demo_extended.ipynb'` (located in `'/PAN14_Model_Code'` on the repository) is an extended version of the original Jupyter notebook provided to the team during the project inception phase.

The original contained all the code to compile and test the model against the provided data set.

We have since added a number of functions that allow the model weights to be saved and reloaded from profiles without retraining, as well running atomically on single profiles, and returning a single score or prediction as a result.

Under the heading 'Production Use' at the tail of the file there are many helper functions, definitions for adapted single-case functions, as well as a class that that wrap the entire model, allowing it to be exported into a library and loaded by other programs.

The new extended notebook focuses on adapting this original code base into a usable interface for the Django back-end to easily call the algorithm for a given set of known documents and an unknown document and return interpretable results, returning numeric score that is converted to a pass or fail in the back-end based on a pre-defined threshold.

As each model is composed of 3 separate networks, as well as metadata, we have added code to the notebook to support saving of profiles into a single folder that can be easily loaded. These folders contain all the model weights and metadata (such as validity threshold, embedding dimension and word vector size, which are generated during compilation) removing the dependence of the library on hard-coded values and allow the code to be significantly more adaptable to newer profiles with minimal changes under the hood.

All tests cases to ensure the validity of the model and the new functions we've created are contained within the notebook, and will run automatically as part of the compilation and testing process. Each of the testing blocks will return statistics on the level of accuracy our

functions have in relation to the model, with comments and annotations inside give more details on the structure and the purpose of each section.

Stylometry Library (stylometry.py)

Encapsulates the class defined in the notebook, along with the required dependencies and helper functions.

Use of the module only requires the import of the `StyloNet` class, allowing the model to be loaded with a given profile. Profiles for the project are saved under the `'/stylometry_models'` folder within the Django project.

Google Co-Lab

As part of the testing process we run the notebook in both a local and Google Co-Lab environment, to ensure it portable and can run in both cloud and local environments.

An additional cell is included near the beginning of the notebook that prints out of the Co-Lab environment information if running in Co-Lab. In addition, the main import block at the beginning of the code lists the version of each important library, such as nltk, gensim and tensorflow, to improve the reporting and allow for troubleshooting of bugs that can occur when switching tensorflow versions. (This particular feature was added as part of some troubleshooting involving changes to the Adam optimizer between tensorflow versions which causes some issues with compilation at the time)

Django Back-End Code Information

This section contains information on each of the files in the Django framework in the context of the AuthorGuard web application. This document will not explain concepts that are a core part of the Django framework and hence were not changed for the purposes of this project. See the Django Documentation if one wishes to learn more about the default files that come with any new Django project.

Project Files Overview

The following list explains the core functionality of each of the files as part of the Django framework in the context of the web application, some experience with Django is recommended to fully understand the operations of each file. The list is sorted by directory, and these directories can be found in the `'/stylometryproject'` directory (in the root of the GitHub)

`'/stylometryapp'` files

- `admin.py`
  - This file contains the admin view permissions for the database, which in particular is the Profile and Document viewing in the admin panel when the web application is running.
- `apps.py`
  - This file contains the apps of the project, which in this case, is only the Stylometry App (AuthorGuard), this is where the `StyloNet` class from the custom-made stylometry library is loaded.
- `forms.py`
  - This file contains any forms that are required in the web application. The form for initializing a document instance to the back-end is located here.
- `models.py`
  - This file contains the classes that form the tables and their relations to one another in the database. It can be thought of it as an object-oriented relational database schema, providing a clear link between the raw tabular data and the usable Python classes. In the case of this app, the two tables of note are the `Profile` and `Document` tables and the attributes that are stored with instances of each of these.
- `utils.py`
  - This file contains supplementary (utility) functions that are considered too complex or verbose to be placed in `views.py`. The file currently includes utility functions for both the authorship verification algorithm and file type treatment.

- `views.py`
  - This file contains the key functionalities of the web app. It forms the basic operations that the front-end can query the back-end to perform. The views are called upon by the JavaScript front-end through HTTP requests and return data to the front-end through HTTP responses. Some notable view functions include webpage navigation views, profile creation, deletion, editing, document submission and deletion and running the verification algorithm.

`'/stylometryproject'` files

- `asgi.py`
  - This file is a Django file and only serves to expose the ASGI callable for the project, which in this case is the `stylometryproject`.
- `settings.py`
  - This file contains a number of important settings that alter the operation of the Django application. In the AuthorGuard web application, it outlines the settings relating to the database that the application attempts to connect to as well as a number of settings relating to Django's user authentication system, which is used to manage the users of the web app.
- `urls.py`
  - This file contains the key URLS for the operation of the web app. This includes both the navigation (user-known) urls, such as `/home`, `/profile` or `/login`. However, it also includes the URLs that the user's browser and front-end will use to call upon views in the back-end through HTTP requests, thus connecting the client and server side code in an intuitive manner.
- `wsgi.py`
  - This file is similar to `asgi.py` except it exposes the WSGI callable for the project, which once again in this case is the `stylometryproject`.

Deployment Document

IT Project - Deployment Document

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

---

Table of Contents

Overview

The purpose of this document is contain the documentation of the deployment strategy for the AuthorGuard web application. The deployment of the application to production is a complex process that involves many interconnected technologies, services which each have their own unique challenges, risks and benefits. The aim of this document is to summarize the components of the production architecture and provide a rationale behind the team's decision making for these different production subsystems.

Deployment Strategy

Overview

The deployment strategy mainly focuses around using Docker as a highly flexible way to deploy the Django web application, with the hosting of the database and docker container on Amazon Web Services. The team wanted a simple to deploy, yet highly extendible and portable deployment strategy and Docker is a great solution for the team's desires in this regard. Likewise, AWS' platform as a service (PaaS) deployment was also suitable for the team due to both its relative simplicity and also vast availability of tools and tutorials relating to deployment via AWS.

Docker

The team prioritized flexibility through Docker due to the hand-off at the end of the project, which will mean it is much simpler for any future team to deploy as docker is a containerized deployment method. This means that any future teams working on the project will be able to deploy the docker to whatever service they wish (that supports Dockers, which is most modern web application deployment services), making it easier to both integrate with existing architecture, as well as add additional functionality to the application.

Amazon Web Services

The team will use Amazon Web Services (AWS) to deploy the majority of the code base to production for this web application. The external database that is connected to the back-end will be a PostgreSQL database hosted by Amazon's Relational Database Service (RDS). The reason behind this is that Amazon RDS is a highly efficient way to setup a database and it is simple to connect it to a Django application in a secure manner through GitHub secrets to store the sensitive login information.

Beyond this, the docker container itself will be deployed on Amazon's Elastic Container Service (ECS). Due to the aforementioned database also being deployed via AWS, this means that it is much easier to ensure better security for the web application. This is because both the database and server hosting are controlled on Amazon infrastructure, which means that the AWS' Virtual Private Cloud (VPC) and secret's manager are able to be integrated together, which allows for secure communication between the server and database, a critical part of the functionality of the application.

GitHub

During the early stages of development, the GitHub repository was merely used to allow for parallel work on the project from team members through branches and to create a common shared code base. However, for the purposes of deployment, the team will use GitHub's powerful automation tools offered by GitHub actions to simplify the testing, building and deployment process.

In terms of testing, the team will use Django's testing framework as well as GitHub actions to automate testing of any code changes whenever a pull request is made to either the 'main' or 'development' branches. This will ensure that code is thoroughly tested and integrates with the current code review system, allowing for a simpler work flow that abides by the agile methodology.

In terms of building and deployment, the team will use GitHub actions to automatically build the Docker for the Django application on commits to 'main'. This docker will then be sent to AWS for automatic deployment through GitHub actions. As a result of this, it is important that most code commits are made to individual branches, then merged on to 'development' and only when a new release is to be pushed to production should the 'development' branch be merged onto main.

This workflow ensures that team members can still individually work on the project through feature branches and create pull requests that will be thoroughly tested and reviewed, thus ensuring that code is properly tested. Then when a new release is ready, the team will merge the deployment branch to main and then the automated build and deployment of the Docker will simplify the process of pushing new features to production.

Deployment Architecture Diagram

The following diagram is a visual summary of the above mentioned deployment strategy and indicates some of the key relationships between components of the production architecture.

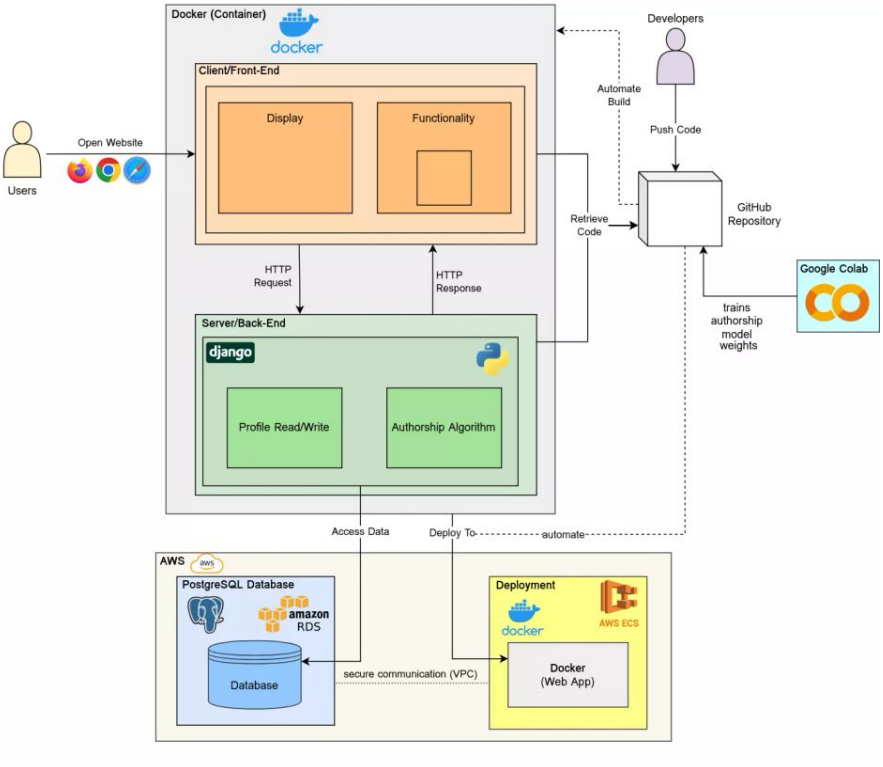


Figure 1 - Deployment Architecture Diagram

🕶 Code Review Record

IT Project - Code Review Record

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

---

Table of Contents

Overview

This document pertains to the record keeping for code reviews of the development of the AuthorGuard web application. It is important that code is reviewed before it is pushed to production, hence code reviews are a regular part of the development process for the team.

The Code Review Document specifies the complete guidelines and conventions when it comes to code review for this project. This document mainly only contains the Review Table, which outlines all the code reviews that have occurred and their details

Code Review Table

The following is the code review table for the project, which outlines each review's number (ID), reviewers, author and notes on code style, functionality and other feedback.

Review ID	Reviewer Names	Author Name	Code Style and Readability	Functionality	Other Feedbacks
1	Ayush Tyagi, Ke Liao, Josh Costa, Bryce Copeland	Jack Perry	• Follow Requirements for most part • One line was too long • Follows conventional Django template and style	• Added Navigation • Added Models to Django • Added Profile Management • Added Profile Editing & Deletion • Added Document uploading to Profiles • Added Uploading Documents for Verification • Added Placeholder Verification Display	• Integration worked mostly ◦ Only one minor display bug regarding trying to verify a file against empty profile
2	Josh Costa	Ayush Tyagi	• Satisfies style requirements	• Works as intended, profile select is now	

				more user-friendly	
3	Jack Perry	Ayush Tyagi	• Cleaned up HTML requirements • New changes meet style guide	• Kept functionality consistent with previous (mainly style changes)	

4	Ke Liao	Jack Perry	• Code looks clean
			• Only minor stylistic problem
			  ◦ in that one line too long
				• The authentication system works well and as expected
				  ◦ Other users cannot view the profile created by current user
				  ◦ Registration and login system working
				  ▪ Registration system checks for weak password and do not allow registration of username that already exists.
					• Maybe we could create a system to allow guest to use it without registration(but the files is only temporary)
					  ◦ Extension?

5	Jack Perry	Bryce Copeland	• Code Style is correct and consistent with rest of project • Good library based design, easy to use and implement	• Algorithm API is modular and easily implementable into Django • Able to run the algorithm through the web application and return the value • Cleaned up files (cache, gitignore, etc.)	• Stretch Goal: Consider possibility of using different model weights (by user selection potentially)
6	Jack Perry	Ke Liao	• Code Style is consistent with current code base	• Added CSS to login screen • No major functional difference - as intended	
7	Ke Liao	Bryce Copland	• Code Style is correct and consistent with rest of project	• Now prints to console the names and versions of modules loaded	• The addition of the requirements.txt with thepython

				◦ helps find problems in the event that python modules fail to load properly.	modules needed helps save time.
8	Ke Liao, Josh Costa	Jack Perry	• Some lines are too long ◦ PEP8 indicates lines 79 Char max length	• No real functionality changes ◦ Only testing added	• Test cases will help in potentially identifying issues from future changes.
9	Jack Perry	Josh Costa	• Satisfies code style requirements	• Restricted uploaded to only .txt, works as intended	• Stretch Goals: Add in .doc, .docx, and .pdf
10	Jack Perry	Ke Liao	• Satisfies code style requirements	• Files can now be uploaded additive for easier usability	• Stretch Goal: Add buttons to remove previously uploaded files
11	Jack Perry	Ayush Tyagi	• Satisfies code style requirements	• Can no longer click verify multiple times • Works as intended • Logout Button looks better	
12	Jack Perry	Bryce Copeland	• Satisfies code style requirements	• No Major Functional Changes, mainly cleaning up Stylometry Algorithm code base	
13	Jack Perry	Ayush Tyagi	• Satisfies code style requirements	• Added CSS to the registration page ◦ No functional change (intended)	
14	Josh Costa	Jack Perry	• Satisfies code style requirements	• Successfully added GitHub actions for testing	
15	Josh Costa	Jack Perry	• Satisfies code style requirements	• Updated README with clearer project details & cleaned up repository	

16	Jack Perry	Bryce Copeland	• Satisfies code style requirements • Removed Hard-Coded Variables	• Used Django sessions to reload previously selected profile information ◦ Mainly for a better user experience	
17	Ke Liao	Ayush Tyagi	• Satisfies code style requirements	• Massive bug in which incorrect file is deleted for the file upload function.	• Bugs needs to be resolved ◦ Additional comments relating to bug on github • Also merge conflict need to be sorted
18	Ke Liao, Jack Perry	Ayush Tyagi	Satisfied code style requirements • Additional changes to make code more readable	• Bug for file deletion fixed ◦ Deletion of files to be uploaded works properly • Now spamming verify button no longer works	
19	Jack Perry	Ke Liao	• Satisfies code style requirements	• Added limit to file name changes • Can view documents on verify page	
20	Jack Perry	Ayush Tyagi	• Satisfies code style requirements	• Documents appear correctly on verify Page • Reworked UX/UI for better user interaction and usability	
21	Jack Perry	Bryce Copeland	• Satisfies code style requirements ◦ Good Practice in avoiding hard coded variables	• The model weights can now be loaded in a more efficient and better organized manner • Better handoff	

🔬 Test Case Document

IT Project - Test Case Document

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

---

Table of Contents

Overview

This document pertains to the record keeping for the testing of the AuthorGuard web application. As per the agile methodology, code is not done until it is tested and hence it is crucial that the testing for this project is performed in systematic and well documented manner. This is particularly important for potential future hand off of the project and also will help ensure that the code is performed the expected tasks correctly.

This document contains the Acceptance Criteria - Given, When Then, table, which outlines the acceptance criteria for test cases in terms of the user stories as specified in the requirements document. However unlike in the requirements document, only user stories that are fully implemented & tested are included.

This document also contains the Test Case Table, which is a record of the test cases ran on the code base for this project and records the description, conditions, expected vs actual results and any notes/actionable to take away from the testing. These tests have all been automated via GitHub actions and are ran whenever code is pushed or a pull request is made to either the ‘main’ or ‘development’ branches.

Finally, this document also contains the Test Case Log, which records which test cases were ran for commits and record the result.

Acceptance Criteria - Given When Then

The following is the acceptance criteria - given, when then table for the project. Compared to the requirements document which outlines all requirements, including stretch goals, this table only contains functional/implemented requirements and the acceptance requirements for each user story.

ID	User Story	Given	When	Then
1	As a user, I want to create profiles to store groups of documents in an intuitive manner.	• I have a group of documents	• I upload the documents, name the profile, and click submit	• I can see that the documents are uploaded to the profile
2	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.	• I have singular document • I have a profile of documents	• I click the “Run Verification” button on the verify tab.	• I can see whether verification algorithm’s result on whether the singular document was

				written by the same person as the group.
3	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.	• I have a profile of documents • I want to change to a different profile	• I click the Select Profile drop down • And select a new profile	• I can see which profile I am currently comparing the singular document against.
4	As a user, I want to create a personal account so that my uploaded documents are saved between browsing sessions.	• I want to use the application again (retain information)	• I click the “Register” button after filling in my details	• A new user is created, an profiles can be created and documents can be uploaded to that profile
5	As a user, I want to be able to upload my documents in any format easily.	• I have non .txt files (.docx, .doc, .pdf) • I want to compare them	• I upload the document to either the profile storage or singular document comparison	• The document content is extracted from the file type for use in the algorithm.

Test Case Table

The following is the test case table for the project. These test cases are all automated and simulated through the ‘tests' directory in the code base.

These tests are ran whenever a pull request is made to either the ‘main’ or ‘development’ branches, which ensures that code is rigorously tested before being pushed to production. Team members should run ‘python manage.py test’ before creating a pull request as well.

ID	Test Case Title	Description/Objective	Steps	Code for testing (or N/A)	Expected Result	Actual Result	Notes
0	*Example: Upload Button*	*User is able to upload file to the server*	1. *Click upload button* 2. *Choose file* 3. *Upload*	*N/A*	*File is uploaded successfully to server*	*Same as expected*	*This is just an example, not formalised*
1	Profile Page Navigation	User can navigate to the profile page	1. Click profile page button	• views.py: 'profile_page_view' • profile.html	HTML/CSS of Profile Page with JS Functionality is sent to Client	Same as expected	Test Case Code: tests/test_profile.py - ‘test_navigation’
2	Create Profile	User can create a profile	1. Click profile Drop down 2. Click '+' at bottom 3. Add name to profile 4. Submit and view empty	• views.py: ‘create_profile’ • dropdown.js • profile.html	Profile is created on server,	Same as expected	Test Case Code: tests/test_profile.py - ‘test_create_profile’

			profile				
3	Delete Profile	User can delete a previously created profile	1. Click Profile Drop down 2. Select Delete Button on Profile of choice	• views.py: ‘delete_profile’ • dropdown.js • profileManage.js • profile.html	Profile is deleted from view and from server (and documents)	Same as expected	Test Case Code: tests/test_profile.py - ‘test_profile_delete’
4	Edit Profile Name	User can edit the name of a previously created Profile	1. Click Profile Drop down 2. Click edit button 3. Form to put new name appears 4. Submit new name	• views.py: ‘edit_profile’ • dropdown.js • profileManage.js • profile.html	Profile name is changed on both display and internally in database	Same as expected	Test Case Code: tests/test_profile.py - ‘test_profile_edit’
5	Get Profile Name	Retrieve the name of a profile based on an internal ID	1. When profile is selected, profile name is known to front-end	• views.py: ‘get_profile_name’ • docDisplay.js	Profile name is retrieved and changed on page correctly.	Same as expected	Test Case Code: tests/test_profile.py - ‘test_get_profile_name’
6	Add Document to Profile	Add uploaded document to profile	1. Click (or drag) document into field of upload 2. Select a profile 3. Press Submit	• views.py: ‘add_profile_docs’ • submit.js • uploadButton.js • profile.html	Document name and content is added to database, with FK to profile	Same as expected	Test Case Code: tests/test_documents.py - ‘test_add_document'
7	Delete Document from Profile	On a profile with documents already uploaded, delete the document from the profile	1. Select a Profile 2. In document display, click 'X' next to a document	• views.py: ‘delete_document' • docDisplay.js • profile.html	Document is removed from display and removed from database	Same as expected	Test Case Code: tests/test_documents.py - ‘test_delete_document'
8	Get all Documents of Profile	Retrieve the names of all documents associated with an internal profile ID	1. When a profile is selected, all document names associated are known to front-end	• views.py: ‘get_documents' • docDisplay.js	All documents associated with profile are retrieved and changes made	Same as expected	Test Case Code: tests/test_documents.py - ‘test_get_documents'

9	Login Navigation	Test that app prompts user to login when navigating to functional pages	1. New User clicks either profile manager or verify page 2. User is prompted to login	• views.py: ‘register' • base.html • login.html	login.html is returned to the user when it is required to	Same as expected	Test Case Code: tests/test_login.py - ‘test_login_nav'
10	Login	User can login to a previously registered account	1. User is on login page 2. User enters username and password 3. User presses submit	• urls.py: ‘/login/' • base.html • login.html • success. html	User is successful in logging in and now can access restricted pages freely.	Same as expected	Test Case Code: tests/test_login.py - ‘test_login'
11	Registration	User can register a new account	1. User on login page 2. User clicks ‘Register’ 3. User enters information for new account 4. User submit information	• views.py: ‘register' • base.html • register.h tml • success. html • login.html	New user is created on database (that can be logged into)	Same as expected	Test Case Code: tests/test_login.py - ‘test_registration'
12	Logout	User can logout of the page	1. User clicks logout button on toolbar	• urls.py: ‘/logout/’ • base.html • login.html	User is logged out (returned to login page)	Same as expected	Test Case Code: tests/test_login.py - ‘test_logout'