Friday 11am - Team 2 - IT Project . . . 2
Project Inception Documents . . . 4
Requirements Document . . . 5
Design Document . . . 10
UX/UI Design Document . . . 15
Decision Log/Document . . . 25
Development & Deployment Documents . . . 31
Code Style Document . . . 32
Peer Programming Document . . . 35
Code Review Document . . . 37
Front-End Documentation . . . 39
Back-End Documentation . . . 41
Deployment Document . . . 46
Code Review Record . . . 48
Test Case Document . . . 55
Progress Assessment Change Logs . . . 59
Progress Assessment 2 - Change Log . . . 60
Progress Assessment 3 - Change Log . . . 62
Handover Documents . . . 64
Requirements Summary . . . 65
Design & Architecture Summary . . . 67
UX/UI Design Summary . . . 68
Code Base Documentation Guide . . . 69
Deployment Guide . . . 70
Security & Ethics Report . . . 73

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.

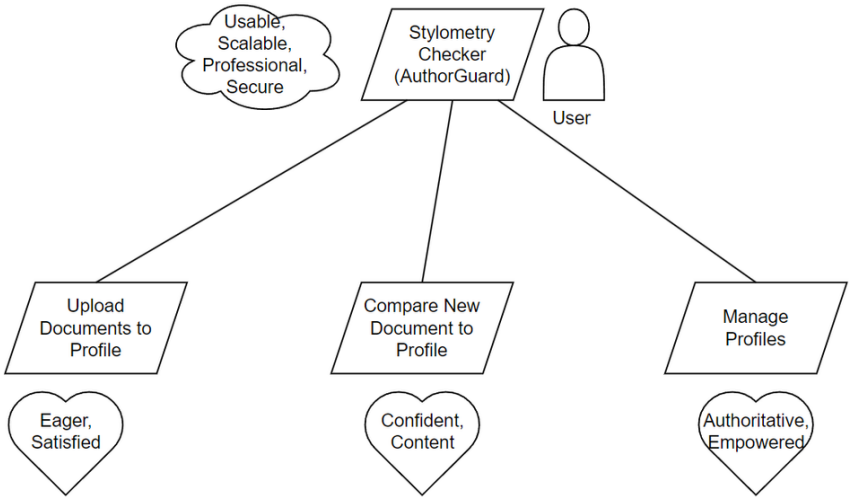


Figure 1 - Motivational Model of Minimum Viable Product

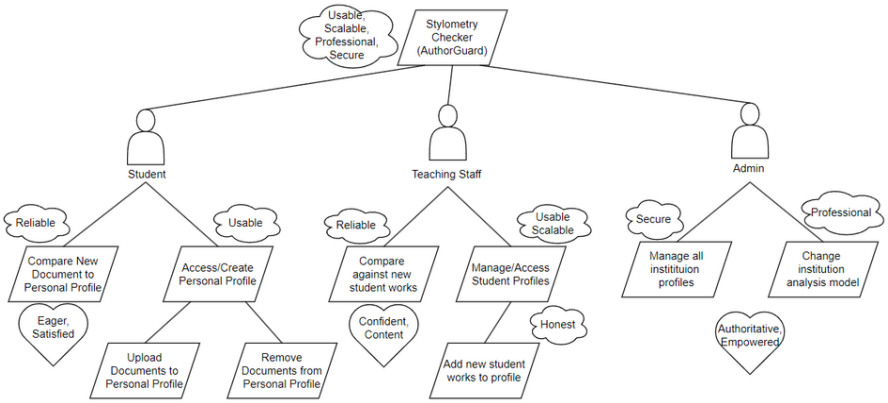


Figure 2 - Motivational Model ft. Stretch Goals & User Stories

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 • 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 • The resulting writing style similarity percentage is displayed to user	High
As a student/teacher, I want to add/remove documents to/from the group, so that I can easily	• Able to remove previously uploaded documents from group	High

change which documents are being compared against.	• Able to upload and append additional documents to group	
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.

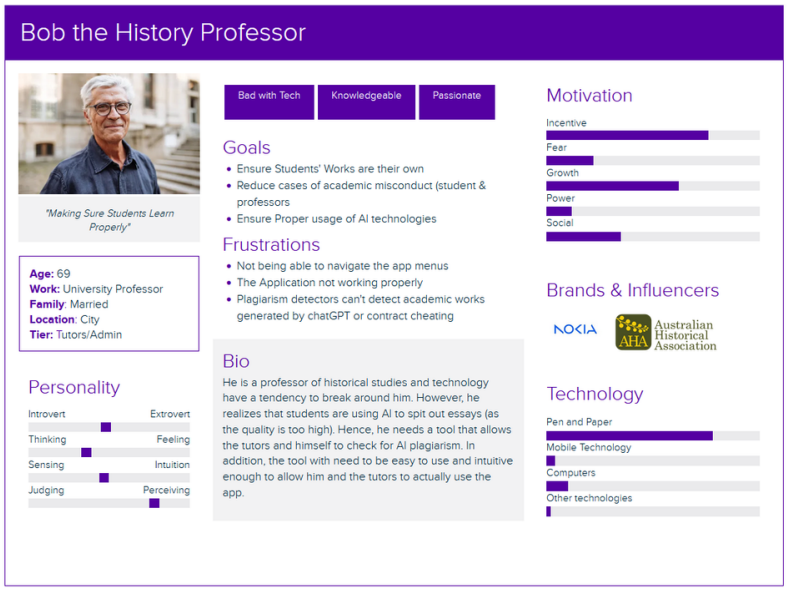


Figure 3 - Bob the History Professor Persona

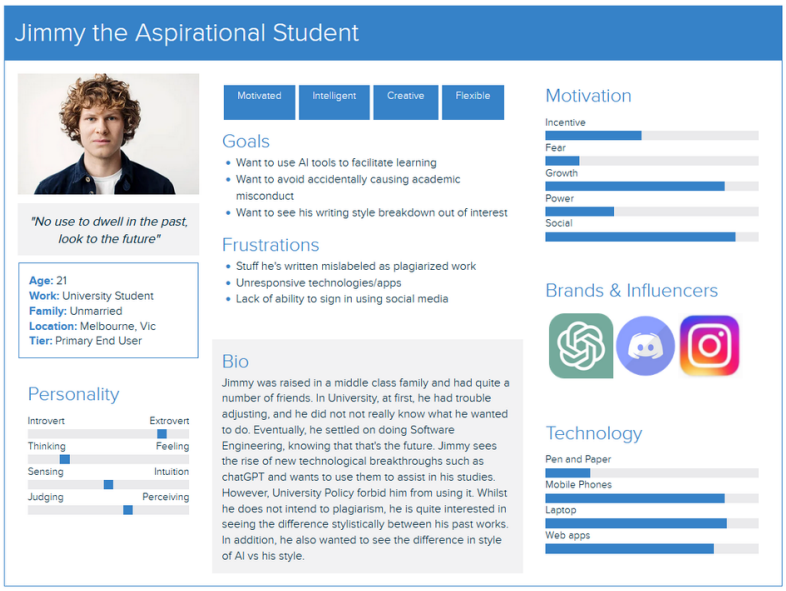
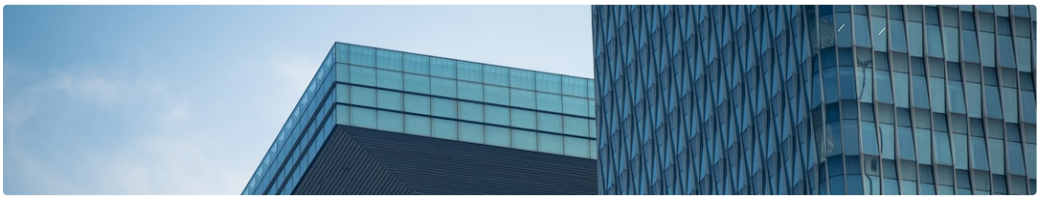


Figure 4 - Jimmy the Aspirational Student Persona



🖋 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.

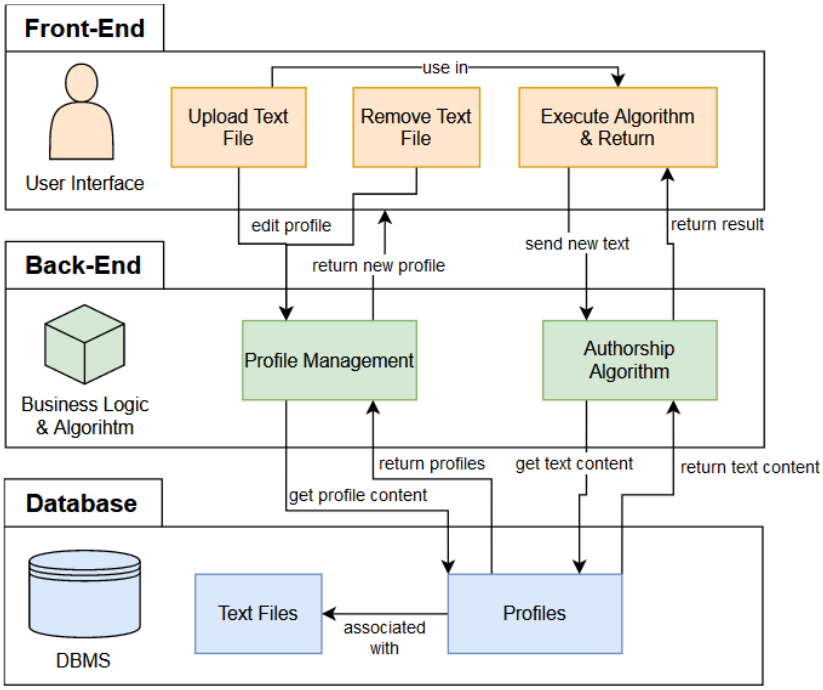


Figure 1 - Ideal Architectural Components & Basic Functions

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

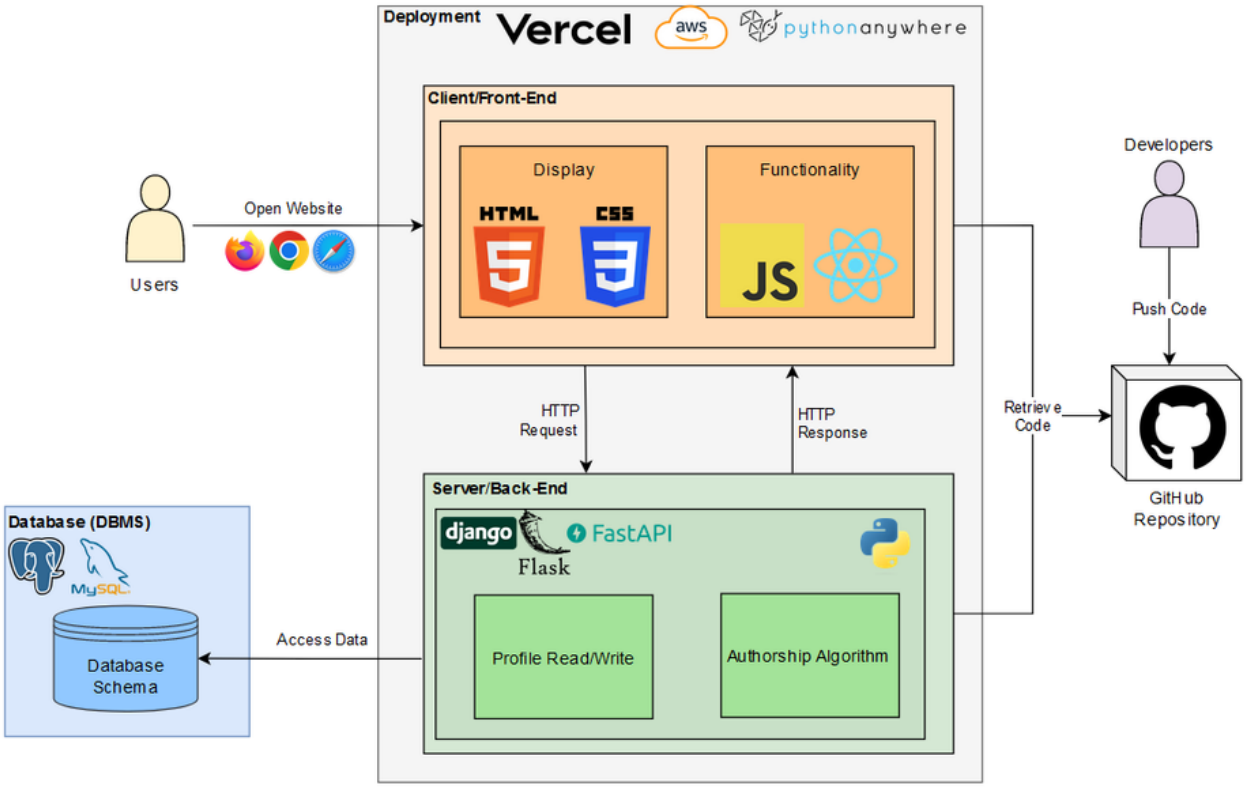


Figure 2 - System Architecture Options

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.
The final version of the system architecture can be found in the Deployment Document.

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.

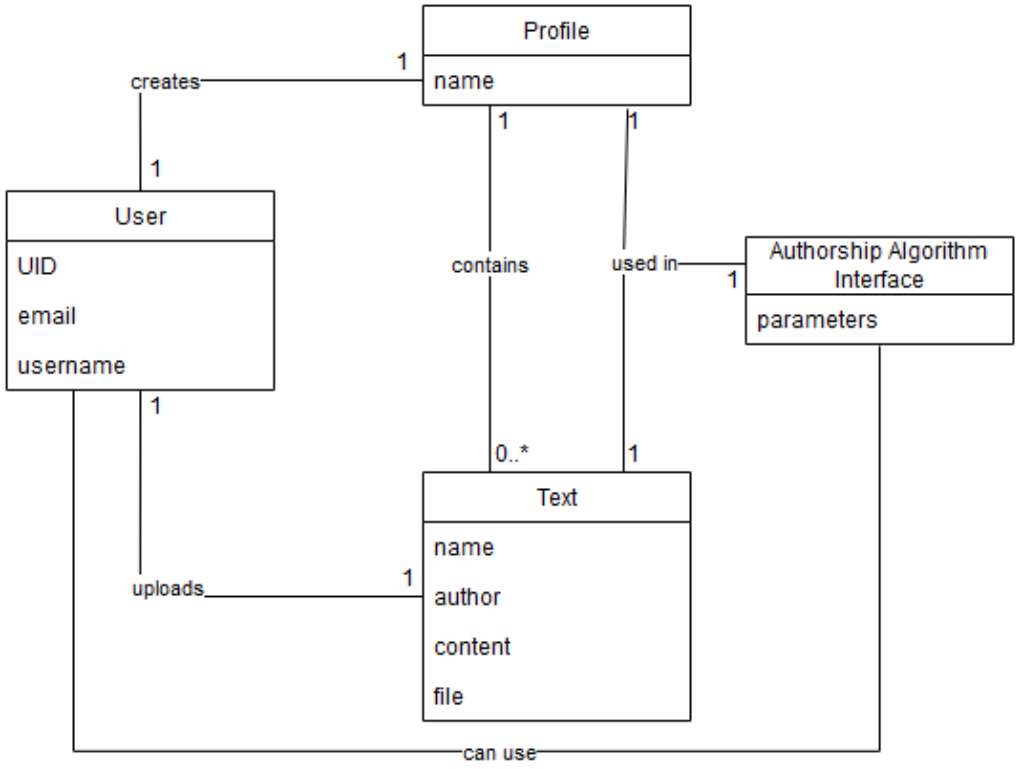


Figure 3 - Basic System Dynamics (Minimum Viable Product Domain Model)

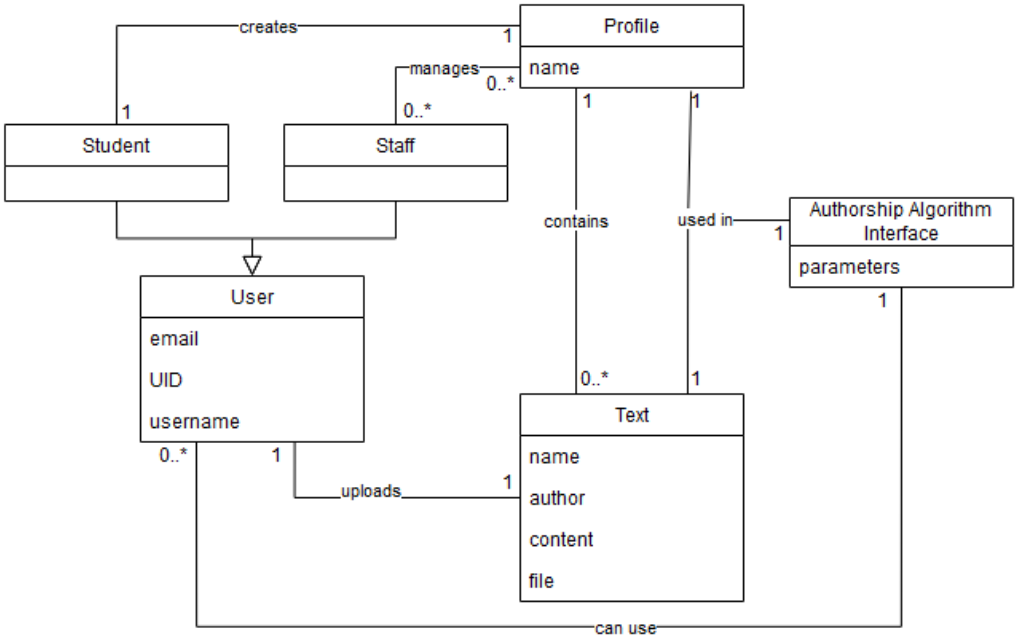


Figure 4 - Extended System Dynamics (Stretch Goals Domain Model)

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.

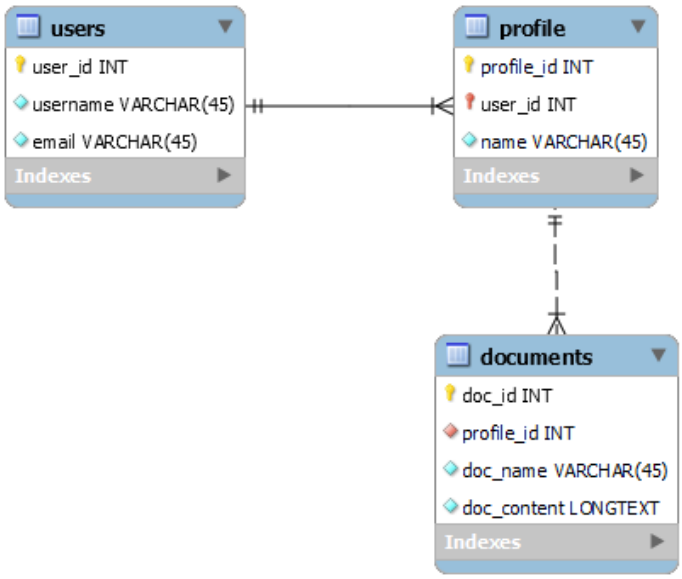


Figure 5 - Simple Design Database Diagram

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.

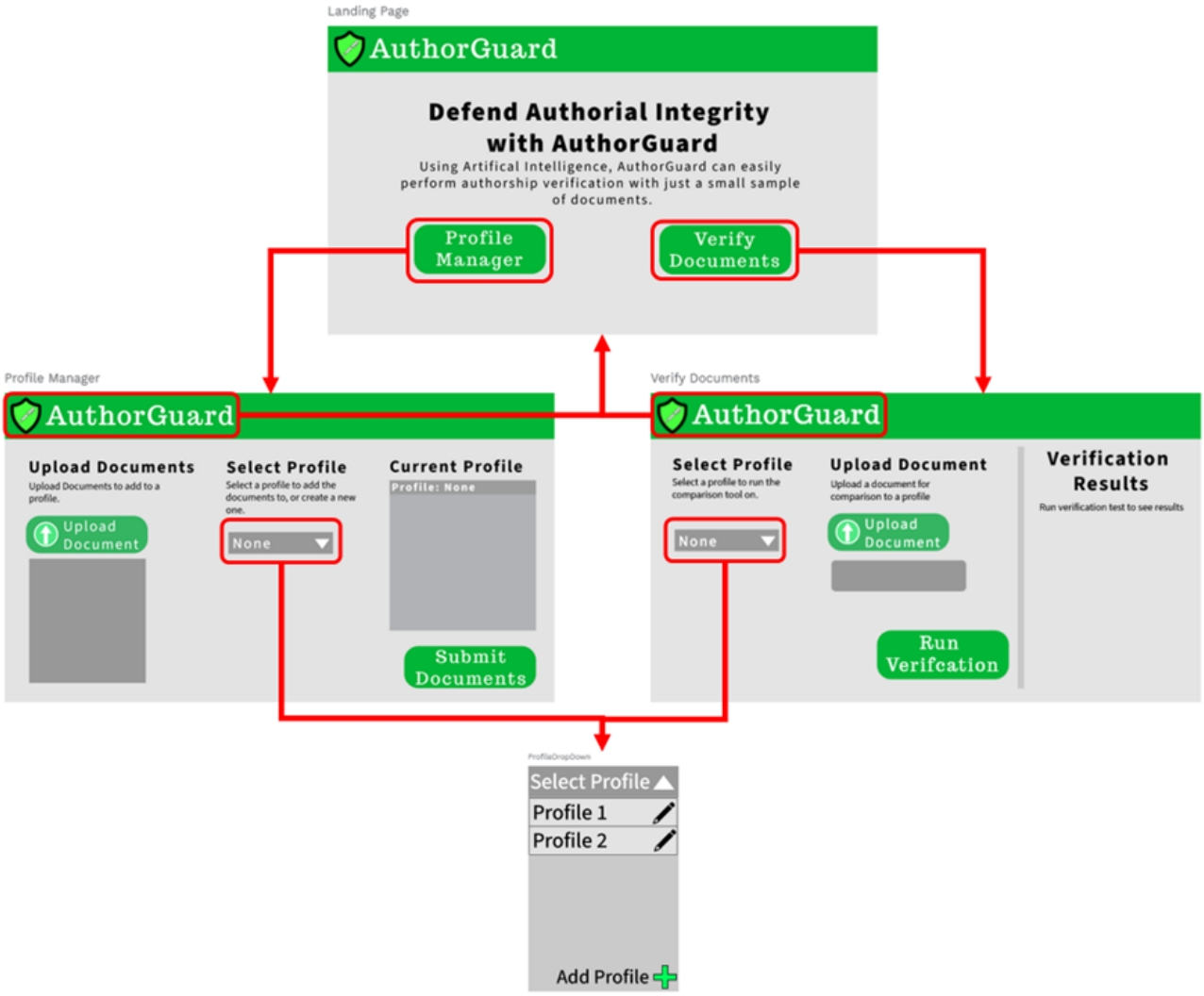


Figure 1 - Essential Requirements UX/UI Partial-Annotation Wireframe

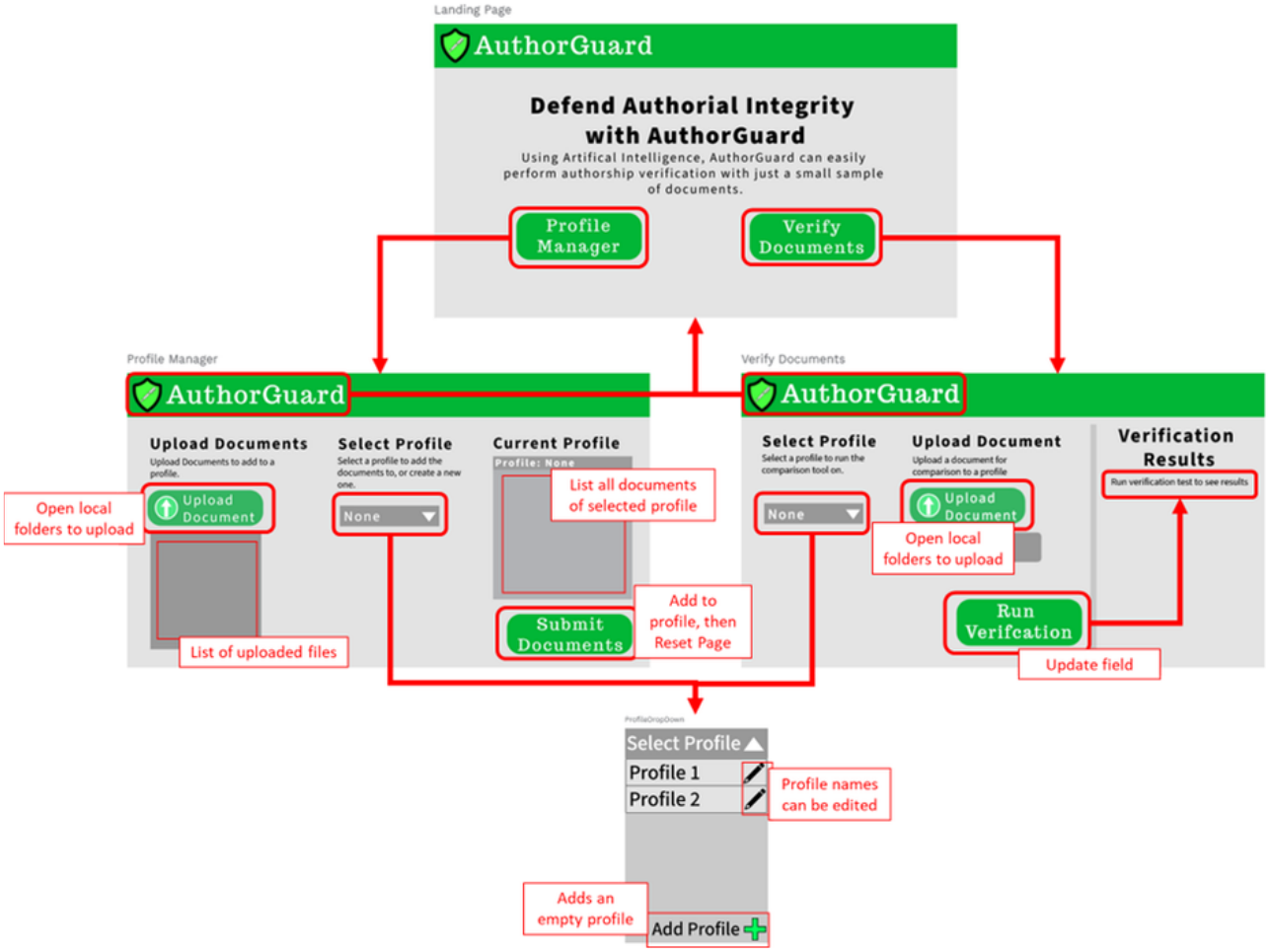


Figure 2 - Essential Requirements UX/UI Full-Annotation Wireframe

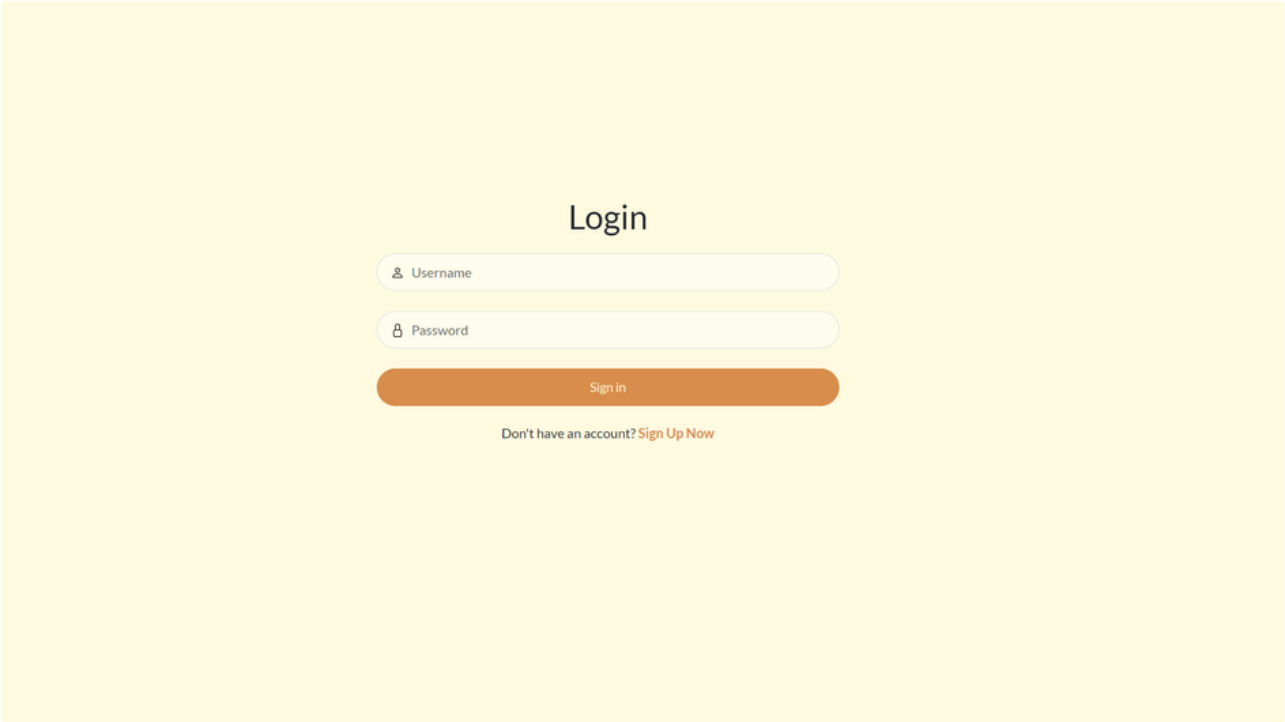
Key Components

The key components of this design are the following:

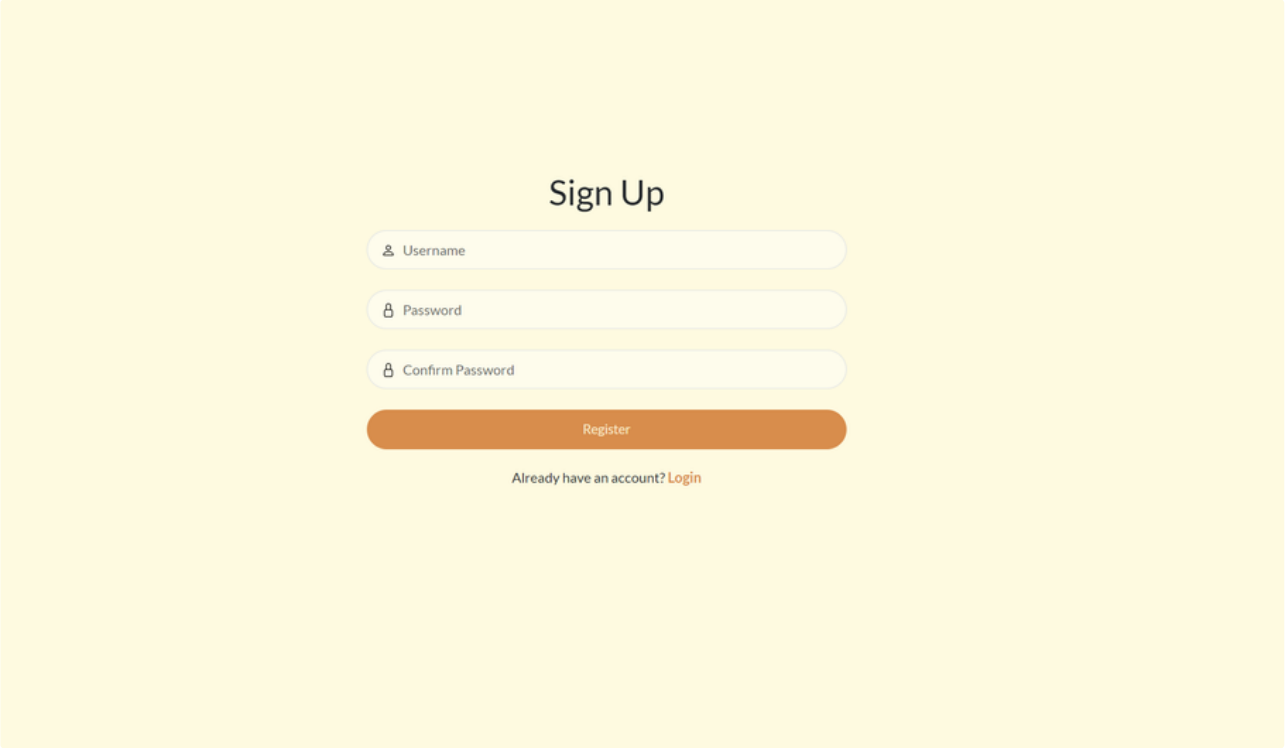
- Buttons: Enable navigation between pages, trigger an instruction (upload document from user machine, update profile, trigger a verification test.
- Drop-Down List: List all profiles are user has, dynamic profile addition, dynamic profile renaming, profile selection
- Document Lists: List the documents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

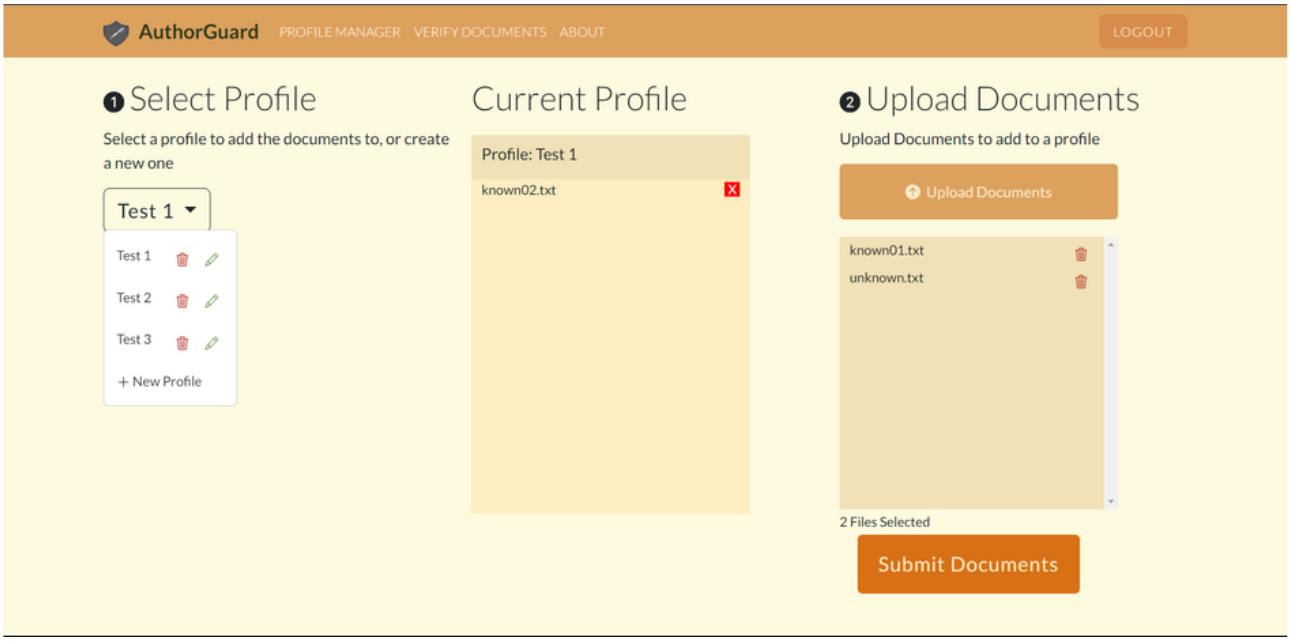


Login Page

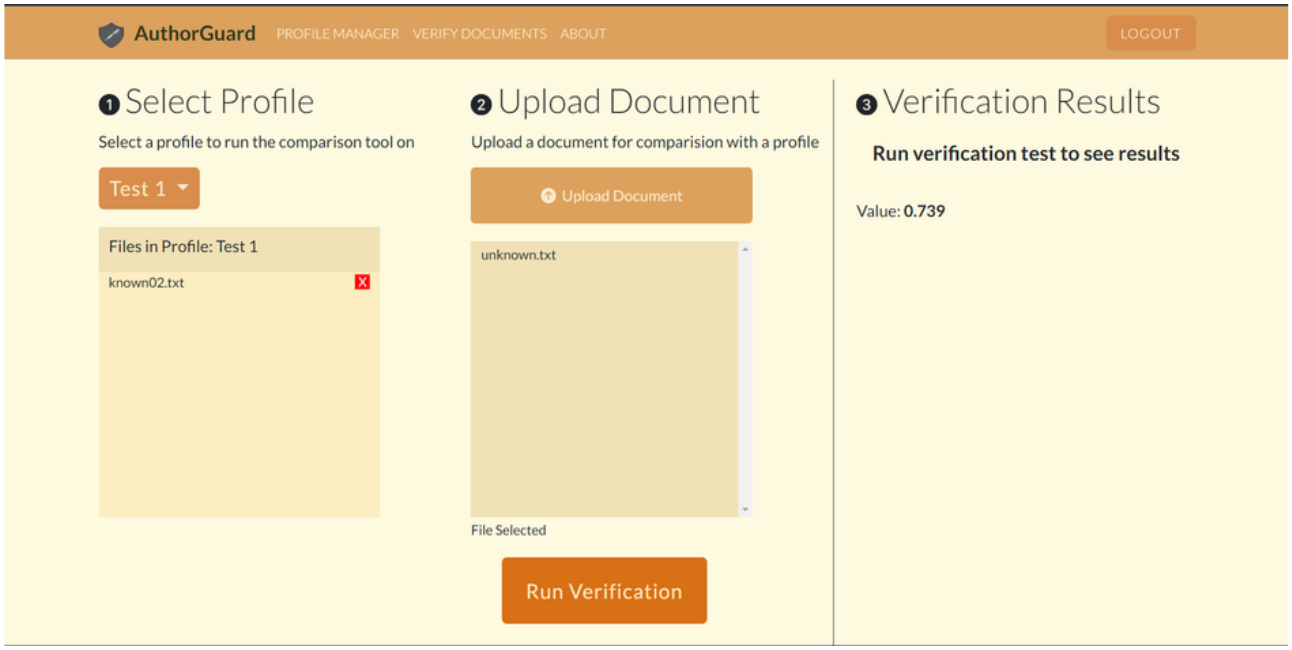


Registration Page

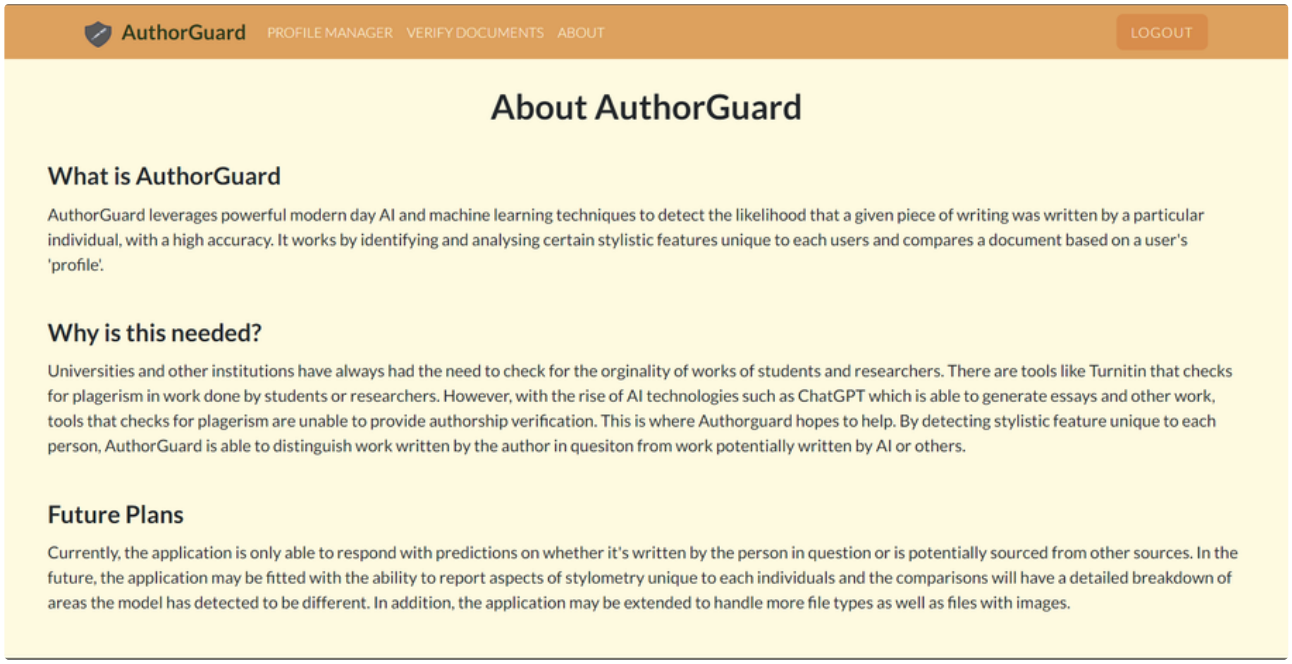
Landing Page



Profile Manager Page



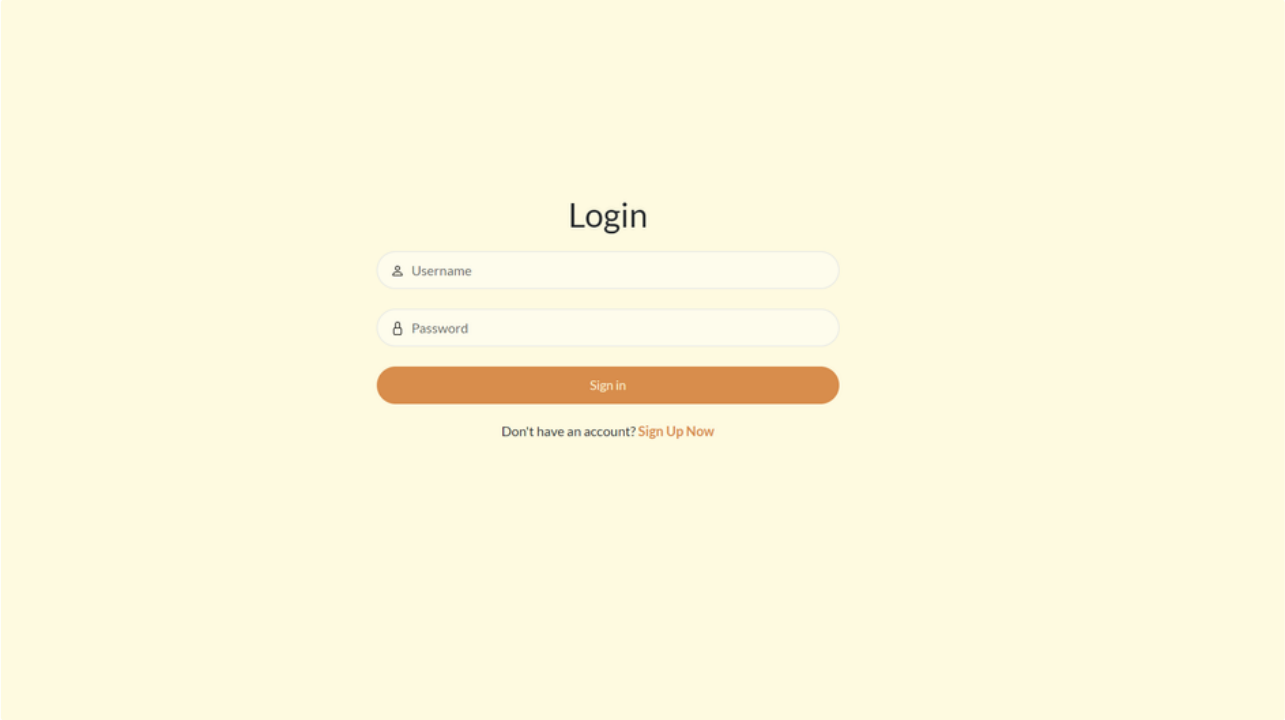
Verify Document Page



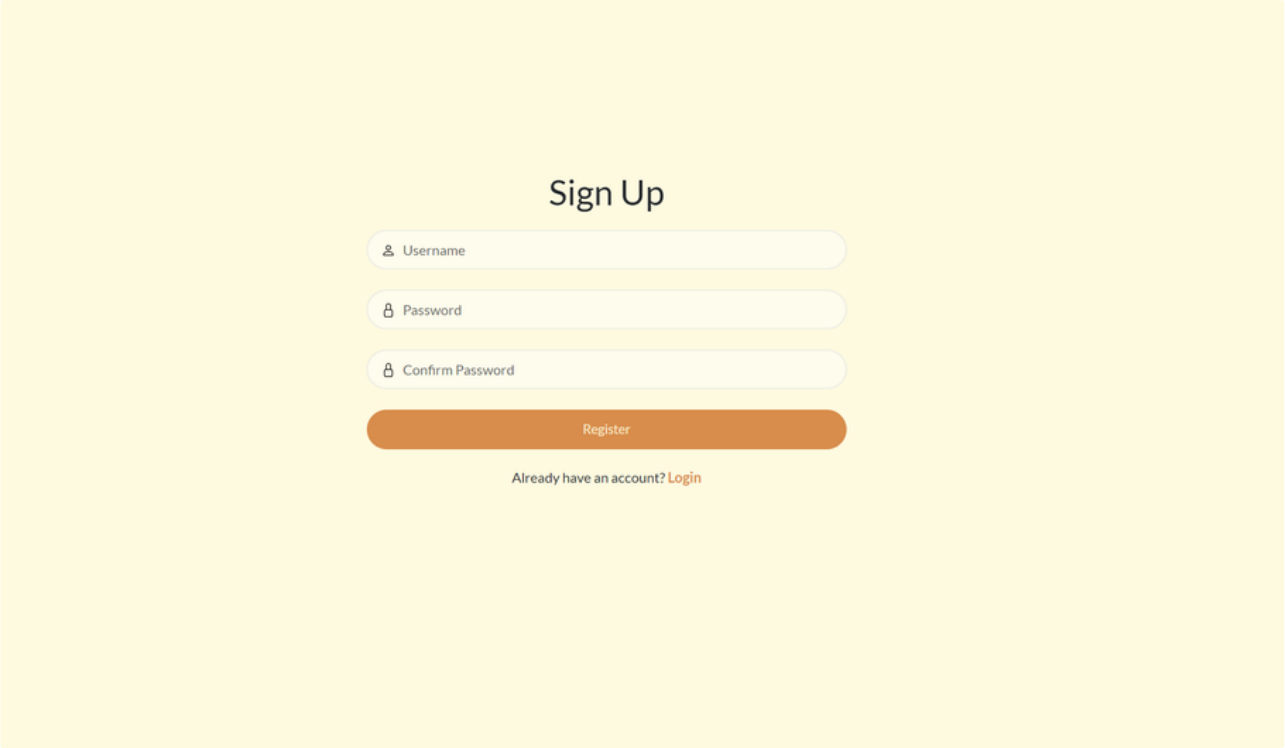
About Page

Finalised Design (Sprint 3)

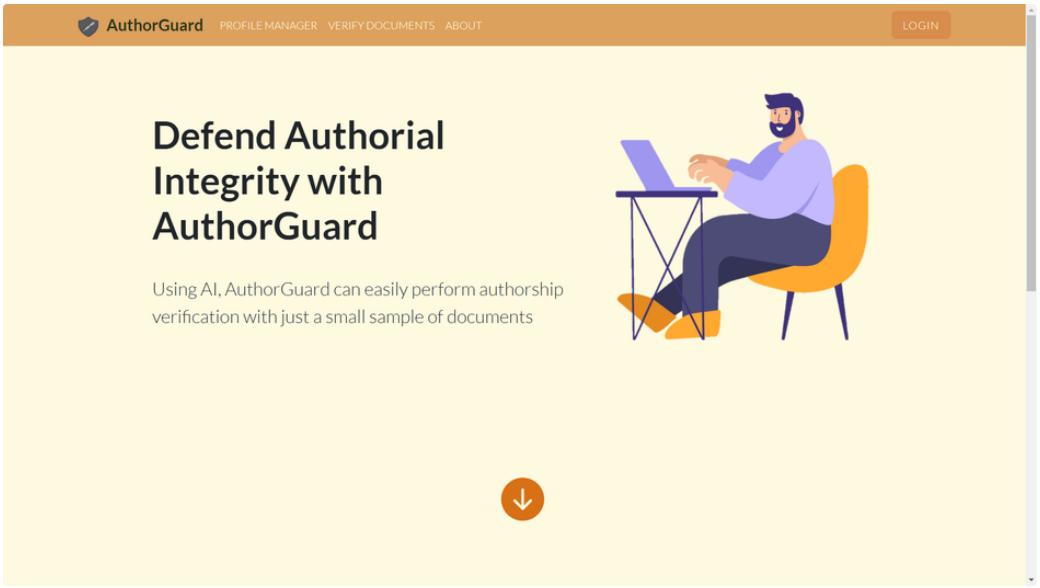
The following images show the finalised implementation of the frontend design as of 20/10/23. This includes further UX and usability improvements as suggested by the client, providing additional prompts and suggestions for new users, as well as displaying more in-depth document feature comparisons.



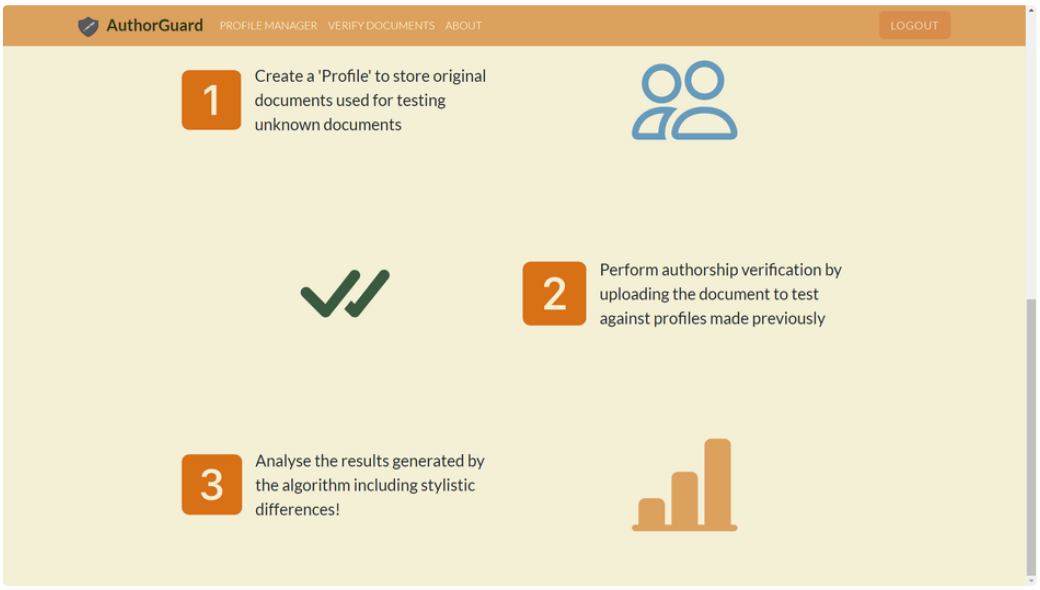
Login Page



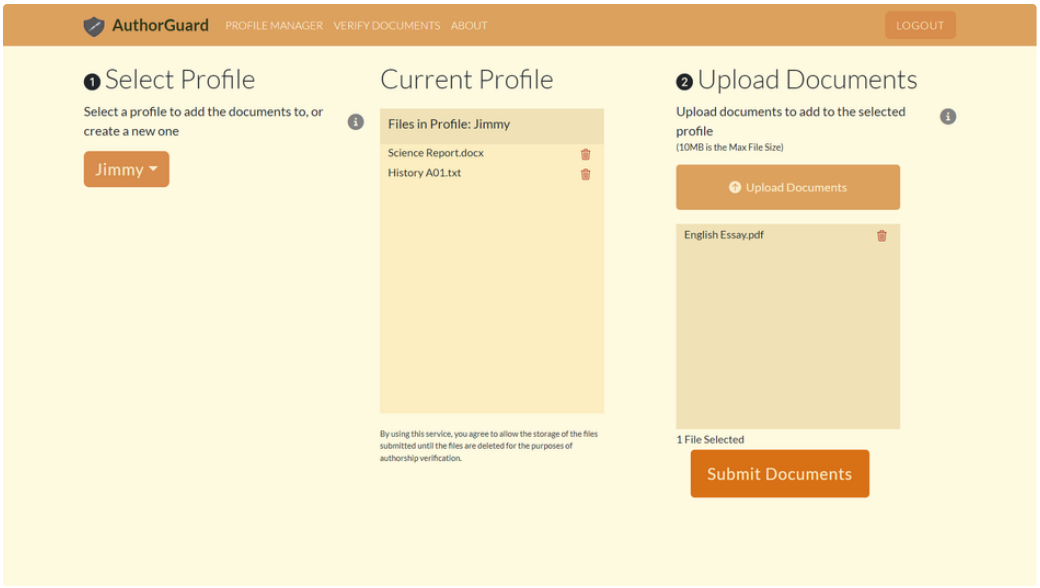
Registration Page



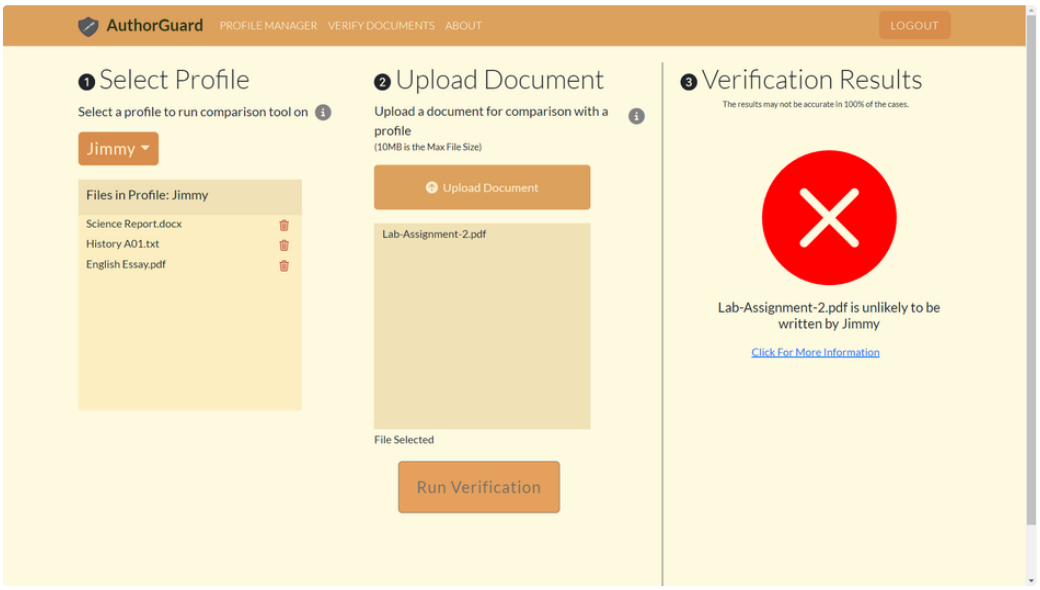
Landing Page



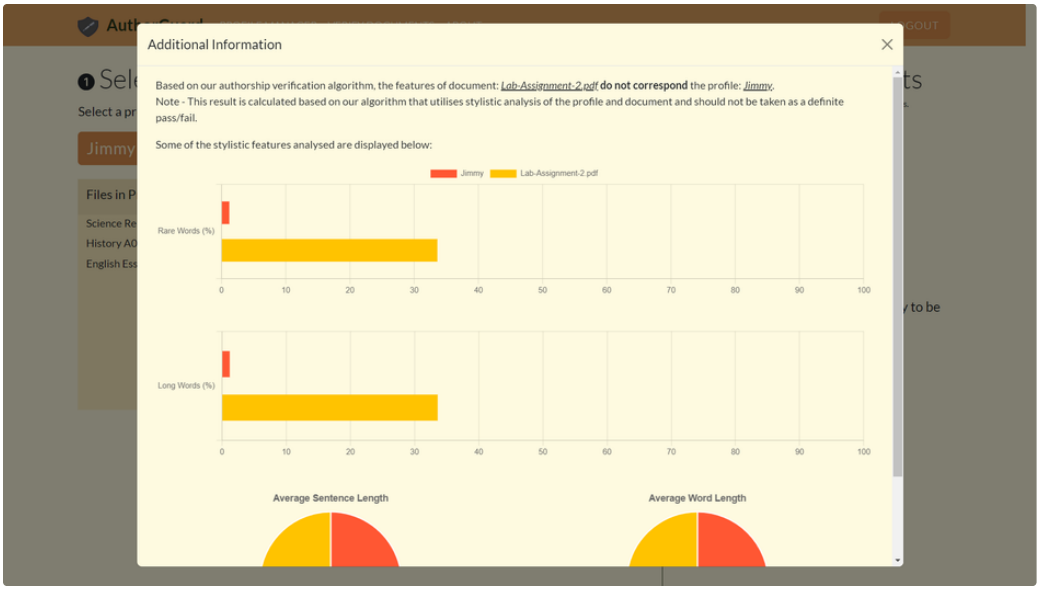
Landing Page cont.



Profile Manager Page



Verify Documents Page



Results Additional Information 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	Completed
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
Sprint 3 Decisions: 26/09/2023 → 20/10/2023						
Date	**Decision**	**Context**	**Rationale**	**Impact**	**Implementation**	**Status**
26/09/2023	Second Sprint Commenced (26/09 → 20/10)	The Third Sprint, deployment, testing and handover was decided to begin.	To finish off development, deploy and test the app to production and handover.	Jira Backlog updated with new user stories and tasks to be performed.	Backlog dictates what tasks need to be completed for development	In Progress
26/09/2023	Decision to create Handover Documents	The team decided to create a set of summary documents for handover	To allow for simpler handover and a clear summary of all important information	Jira Backlog updated with new tasks related to document summary creation.	Backlog dictates what tasks need to be completed for the creation of these documents	Completed
06/10/2023	Decision to use private Docker Hub	The team will store the image of the app on a private Docker Hub repository	So that a .env file can exist in the image without causing security issues	A repository will be setup and used, authentication tokens	Create the repository and push images to it	Completed
06/10/2023	Decision to use AWS Elastic Beanstalk	The team will use an elastic beanstalk application to host the image	In order to create a publicly accessible deployed application	AWS Eleastic Beanstalk instance created and application deloyed	AWS EB will be created, connected to Docker Hub private repo via secure S3 bucket to store authentication token	Completed
06/10/2023	Decision to support additional file types	The team decided to make the web app support more file types	In order to increase the supportability and usability of the app	Tasks added to backlog	Developers will implement this feature to front-end and back-end code base	Completed

06/10/2023	Decision to present additional details to user after verification	The team decided that the web app should show more results information from the model	To increase the usefulness of the application and assist users in understanding the output	Tasks added to the backlog, source graphical library for various graphs to display	Change verification view response to include additional numerical information, display on front-end	Completed
13/10/2023	Decision to fix file size limits	The team decided to implemented and fix the file size limiter	In order to improve security and usability of the web application	Tasks added to Jira	Team members will need to implemented these features	Completed
13/10/2023	Decision to add information icons + make website front-end better	The team decided to redo the home page to make it have instructions + info icons	In order to assist users of the website and make it more ‘professional’	Tasks added to Jira	Tickets will need to be resolved, documentation updated accordingly	Completed
17/10/2023	Decision to add additional information to about page	The team decided to bolster the content on the about page	In order to give me context to the project to users who look at the page	About page will be more significant and tasks will be added to Jira.	Write up will be needed and changes made to HTML	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
Front-End Documentation	Documentation on information related to the front-end	🟧 Front-End Documentation
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 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
```

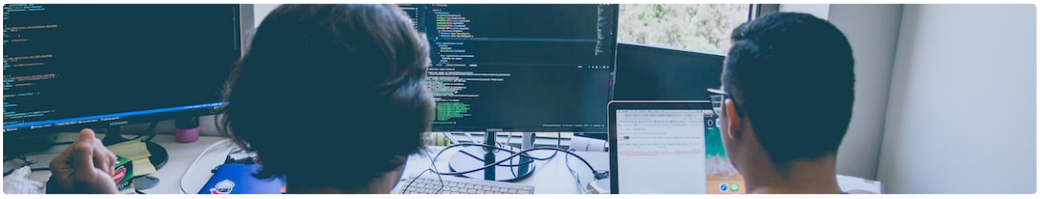
- Line Limit: 79 characters
- Indentation: 4 character indentation per level

```
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.

Front-End Documentation

IT Project - Front-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 front-end code used to create the 'AuthorGuard' web application. The team used standard JavaScript, HTML, & CSS to create the web application, mainly using JQuery to connect with the Django back-end to send HTTP requests to the server. This document explains the purpose of each `.js`, `.html` and `.css` file in the directory in terms of their scope and functionality.

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.

Code Base

This section outlines each of the main directories containing front-end files and explains their purpose and scope and how they connect to each other. Note that directories will be explained from the context of `stylometryproject/stylometryapp` as the root directory rather than the repository root.

Templates Directory (HTML)

The templates directory contains all the `.html` files that the Django-app uses to display the web page on the client-side. It is worth noting that these HTML files, although mostly in traditional HTML, do contain some elements that are in the Django Template Language, which has the ability to create Django-integrated elements using the `{% ... %}` notation. The team did not use this excessively, but it was used in some scenarios where it was much simpler to implement than using a JavaScript approach.

The following is a list of files and their purpose in the `templates/` directory.

- `index.html` - contains the HTML elements for the home page of the website
- `about.html` - contains the HTML elements for the about page of the website
- `profile.html` - contains the HTML elements for the profile manager page of the website
- `verify.html` - contains the HTML elements for the verify documents page of the website
- `registration/` - this directory contains more HTML files that are used for the login portion of the webpage
  - `base.html` - the base HTML for the login and registration pages
  - `login.html` - extending base.html, contains the HTML elements for logging into the website
  - `register.html` - extending base.html contains the HTML elements for registering an account
  - `success.html` - extending base.html contains the HTML page for a successful login, often not displayed to user

Static Directory

The static directory contains the images, JavaScript and Cascading Style Sheet files that define the behavior and look of the ‘AuthorGuard’ web application. The directory is sorted into three folders based on the three file types mentioned above.

- `images/` - this directory contains the images used on the website
  - `Favicon.png` - this the image for the icon that appears on the toolbar of the browser
  - `Logo.png` - this is the image for the 'AuthorGuard' shield icon that appears on the website itself
- `scripts/` - this directory contains the `.js` JavaScript files that add functionality to the website, these scripts are included in the correct HTML when they are required. Most functionality is achieved by sending a HTTP request to the server-side, which the JavaScript is in charge of ensuring that these are sent at the right time and with the right information. The connectivity to the server side is achieved through the JQuery library.
  - `docDisplay.js` - contains the functionality for the document display window, including document deletion
  - `infoIcon.js` - contains the code to flip the info icon text box to the central side
  - `profileManage.js` - contains the functionality for profile creation, editing and deletion
  - `removePlaceholder.js` - contains the placeholder for when no profile is selected
  - `uploadDocs.js` - contains the functionality for the uploading documents on the profile manager
  - `verify.js` - contains the functionality to run the verification algorithm when button is pressed
  - `verifyUtils.js` - contains utility functions for the verify page (unknown document upload, info, etc.)
- `styles/` - this directory contains the `.css` cascading-style sheet files that the website elements use to have the color and style of the HTML elements.
  - `about.css` - contains the style sheet for the about page
  - `app.css` - contains globally constant styles for all pages on the website
  - `dropdown.css` - contains the style sheet for the profile drop down menu
  - `infoIcon.css` - contains the style sheet for the info icons that appear on the page
  - `loadingSpinner.css` - contains the style sheet for the spinning loading when running the verification
  - `registrationPg.cs`s - contains the style sheet for the registration & login pages
  - `resultsBox.css` - contains the style sheet for the results display
  - `uploadbutton.css` - contains the style sheet for the upload buttons on both profile and verify

Static Files Directory

This directory, named `staticfiles/` and located at `root/stylometryproject/staticfiles`, is where when the CLI command `python manage.py collectstatic` outputs the copy of all the files in the app. This is required when creating a Docker container of the application and hence should be run after changes are made to the front-end. However, the files themselves in here should never be changed by developers,instead only be automatically edited by Django itself after running the aforementioned command.

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'` in 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 end of the notebook there are a number of proof-of-concept test cases, that lead up the definition of a self-contained class that is able to load and run the model from a saved profile. The use of which is explained below under Stylometry Library.

After training and testing the model snapshot can be saved as a profile, using functions under 'Profile Management and Cleanup'. Models are saved into subdirectory under `/PAN14_Model_Code/models`, and include all the weights and metadata required to exactly replicate the results and accuracy in the notebook in the production environment.

Each named profile directory contains:

- `model_weights`
  - Tensorflow-generated model weights from training.
  - Note: when reloading the model the optimizer variables are not included. This doesn't make a different to the running results, but training a loaded model will yield different results.
- `word2vec_model`

- `word2vec.model`
  - Saved Gensim Word2Vec model generated during training.
- `nltk_data`
  - NLTK data required for language pre-processing.
  - This directory is included to ensure the profile has everything required to replicate results exactly.
  - This folder can be safely removed from the profile, and the library will attempt to redownload it to the same location within the profile directory on load.
- `manifest.json`
  - Manifest for the profile, contains the profile name, subdirectory locations and model-specific variables such as the Embedding Dim and Validity Threshold. These values are consistent when retraining using the same source code, but are included to minimise hard-coded values in the library.
  - If any keys are missing from the manifest, the library will attempt to use default values when possible.

Stylometry Library (stylometry.py)

Provides an interface for loading and running a model profile through abstract classes, with a simple interface that requires minimal change to implement into a program.

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, by default, within the Django project.

The `StyloNet` class is initialized with the profile name and (optionally) the base directory, and provides the functions:

- `score`
  - Takes a dictionary in the form `{'known': texts, 'unknown': text}` and generates a percentage score, as a float.
- `predict`
  - Takes the same input form as `score`, but outputs the prediction as a Bool based on the model-specific validity threshold, as well as the same percentage score as `score`, in the form of a tuple `(prediction, score)`. Score is included as `score` is still run internally, just to make the information available for debugging.
- `score_batch` and `predict_batch`
  - Both take an input in the form of either a list of `{'known': texts, 'unknown': texts}` or dictionary with these as values. These function utilize tensorflow's more efficient batch function to process a large number of inputs at once, which is significantly more efficient than a single operation.
  - Any list or dictionary that is fed into the function will be mirror in the output. That is, if a list is used a list of floats will be returned in the same ordering, and if a dictionary is used an equivalent dictionary with float values for identical keys will be used.

In addition to the model the library contains the class `TextAnalytics` to return style data from a text. The class is initialized with a text or list of texts, with any shape or depth, provided all end elements are strings. The class is designed to be easily expandable to improve analytics, and currently contains the functions:

- `rare_words_freq`
  - Percentage value of rare word (<= 2 occurances) frequency
  - Optional parameter changes this threshold.
- `long_words_freq`
  - Percentage value of long word (> 6 letters) frequency
  - Optional parameter changes threshold length
- `sentence_length_avg`
  - Returns an average of the sentence lengths throughout the text.
- `sentence_length_distrib`
  - Returns a distribution of sentence length around the average as a tuple of floats `(above, equal, below)`.

    - In this function the average is rounded to the nearest whole number, as you can't have a real sentence with a non-whole-number length.
- `sentence_count`
    - Return the total sentence count for the text
- `most_common_words`
    - Returns an ordered list of the most frequently used words in the text.
    - By default returns 5 values, but this can be changed with an optional parameter.
- `word_count`
    - Integer word count for the text
- `word_length_avg`
    - Average of word lengths
- Note: most of the functions provided by this class are centered around provided percentage values before counts, as it makes it easier to compare style with different text lengths, or by combining multiple texts. This class also has flexible input allowing any arrangement of lists or strings to be inputted, and it will automatically strip, lemmatize, tokenize and normalise the text.
- All functions that work on words will omit punctuation, common English joining words (or stop words in NLTK) and Lemmatize the words before doing comparisons. This is to be more in line with the values used to generate the style vectors for the model, and give more accurate analytics.

In addition to these classes, the library contains a number of internal helper functions and classes for defining the model and processing text, most of which are ported over from `PAN14_demo_extended.ipynb`

The helper functions are intended to be used within the module, with the only required external imports being `StyloNet` and `TextAnalytics` classes, but the follow are used in the backend for general text and array processing:

- `unwrap`
    - Recursive function that will remove redundant, single-element lists. Will return either a single value or the first multi-element list it runs into.
- `flatten`
    - Recursively flattens lists of strings into a single strings, separating each join with a new-line.
    - Can handle lists of all dimensions, as well as inconsistent dimensions, as long as they all end in strings.
- `strip_text`
    - Flattens, strips white space and normalizes text. Will return with a single string containing the entire text, or separate by newlines when `split` parameter is set to true.

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 is portable and can run in both cloud and local environments.

However, due to hardware limitations of our local machines, the initial training of the stylometry model weights was conducted externally on Google Co-Lab, as they provided stronger cloud computing resources.

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 libraries, such as nltk, gensim and tensorflow. (This feature was added after issues with inconsistency when using slightly different versions of tensorflow)

Stylometry Model Diagram

The following diagram describes the operation of the `StyloNet` class in the context of the AuthorGuard web application.

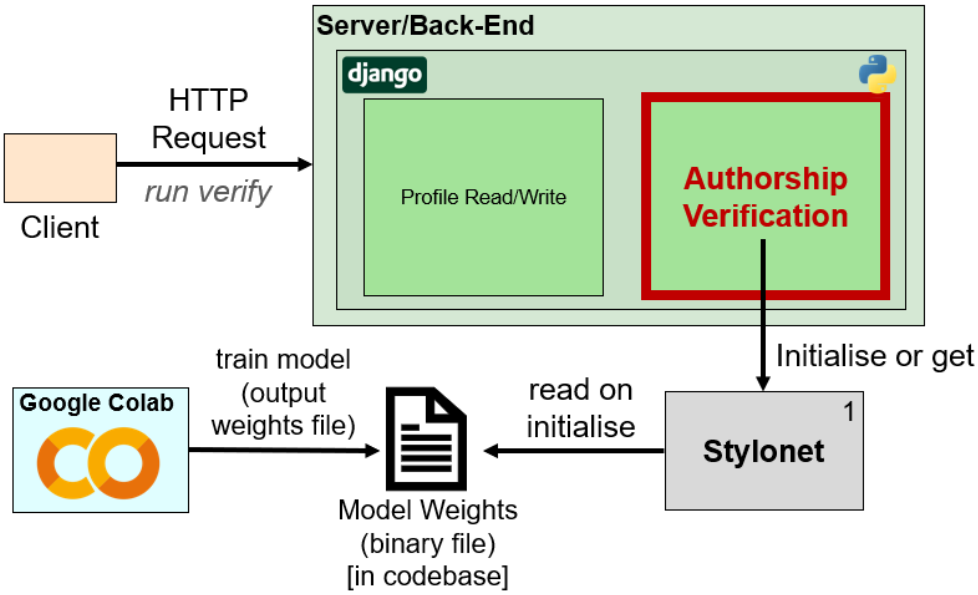


Figure 1 - Stylometry Model Diagram

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`.

`'stylometry_models'` is the default base directory for models loaded by `stylometry.py`. Place all model profile folders in here, and use the key `STYLOMETRY_PROFILE` under `stylometryproject/settings.py` to select the profile name the server loads.

- The path where profiles are stored can be changed from the default with the `STYLOMETRY_PROFILE_DIR` key.

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 Docker Hub. This image will then be pulled by an AWS Elastic Beanstalk (EB) instance. However, this Docker Hub will be a private repository for security reasons, meaning that an authorization token to login will be required. This token will be stored in an S3 bucket that the EB application will have access to in order to pull the image successfully. After this stage, the application will then be deployed according to the `'dockerrun.aws.json'` file, which indicates where AWS should pull the image from and how to get the authorization token for this repository. AWS EB will then automatically deploy the Docker by building it and exposing the ports configured by the `'dockerrun.aws.json'`. Thus, the team will use AWS EB to efficiently deploy the Docker image.

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.

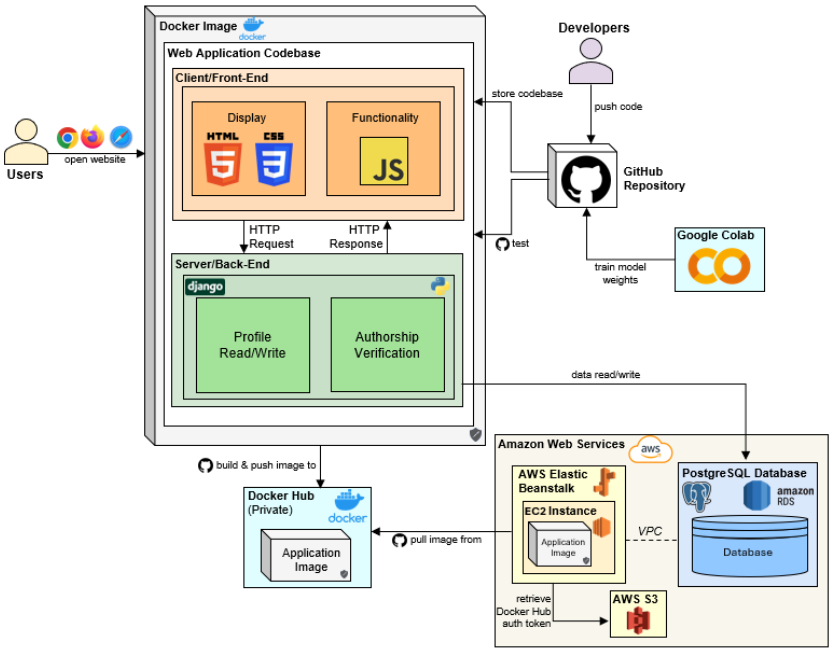


Figure 1 - Deployment Architecture Diagram

Note that any transition/arrow text that has the GitHub logo on it has been automated via GitHub actions.

CI/CD Pipeline

The team created a CI/CD pipeline to automate the aforementioned deployment architecture. The following diagram provides a visual explanation of the CI/CD pipeline.

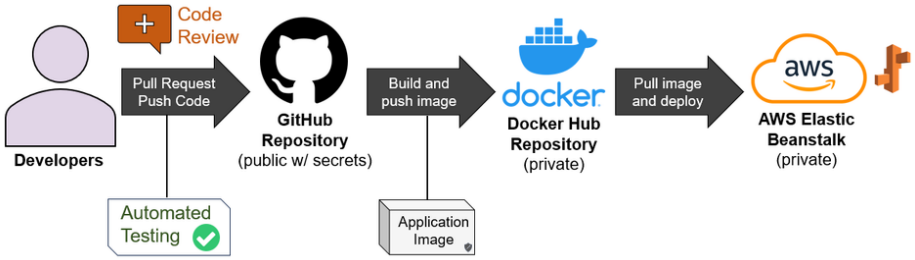


Figure 2 - CI/CD Pipeline Diagram

The code for the GitHub actions that create this CI/CD pipeline can be found in the workflows defined in the `.github/` directory.

- `.django.yml` is used for the automated testing using Django's testing framework
- `.docker.yml` is used for both building and pushing the image to Docker Hub, as well as logging to the AWS Elastic beanstalk instance and sending the deployment package.

The Deployment Guide outlines in more depth the exact configuration of the team's CI/CD pipeline.

🕶 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 Copeland	• 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	

22	Ke Liao, Jack Perry	Bryce Copeland	• Satisfies code style requirements
			  ◦ Removal of global variables
				• No Major Functional Difference	• Removal of global variables
					  ◦ Also fixes bug where when a

			▪ Good practice		profile deleted, if it was selected, the uniqueCurrentProfileId is still that of deleted profile
23	Bryce Copeland	Jack Perry	• Satisfies code style requirements	• Wrote Dockerfile for application • Able to create container for the web application	
24	Bryce Copeland	Jack Perry	• Satisfies code style requirements	• Removed CSRF Exempt Decorators and added CSRF Protect Decorators ◦ Better Security	
25	Jack Perry	Bryce Copeland	• Code is better configurable with env file + secrets integration	• Added compose file for Docker • Edited env variable integration ◦ Better for GitHub	
26	Jack Perry	Bryce Copeland	• Satisfies code style requirements	• Optimized build size for the Docker Container • No Major Functional Changes ◦ SQL Container can be created	
27	Ke Liao	Bryce Copeland	• Satisfies code style requirements	• Bug for which a profile id is selected despite everything profile are deleted is properly fixed ◦ Implemented on the server side	• Other minor fixes such as fixing spelling mistakes also included
28	Ayush Tyagi	Ke Liao	• Satisfied code style requirements • Updated some wording and grammar issues	• Disclaimers added for ethical practice • Fixed bug with verification button not re-enabling	

			for better readability	when document is deleted from profile	
29	Ke Liao	Ayush Tyagi	• Satisfied Code Style Requirements • Removed redundant parts of code • Some minor changes to code for better style and wording.	• Added loading spinner animation when verify button clicked • Made add current profile box more responsive with different states	
30	Ke Liao	Ayush Tyagi	• Satisfies code style requirements	• Added 'deleted profile' alert ◦ prevents accidental deletions	• All delete buttons now look same
31	Ayush Tyagi	Ke Liao	• Satisfies code style requirements • Commented out old implementation	• Added visual representation of ‘result’ obtained from verification • Previously used to just give the number obtained from algorithm	• Changed favicon source
32	Jack Perry	Bryce Copeland	• Satisfies code style requirements	• ?next query after login is now handled correctly by front-end	
33	Jack Perry	Josh Costa	• Satisfies code style requirements	• Upload now accepts docx and pdf file types	• Stretch goals: add additional test cases for docx and pdf
34	Ayush Tyagi	Ke Liao	• Satisfies code style requirements	• Added client-side file size handling	
35	Jack Perry	Ayush Tyagi	• Satisfies code style requirements • Cleaned up front-end modular-nature fof files	• Allows the enter key to be pressed for the submission of the “create profile” form	
36	Josh Costa	Jack Perry	• Satisfies code style requirements	• Fixed File Size Checking ◦ Both Profile & Verify Pages • Can upload in batches without	

				breaking system either (each file less than 10 MB)	
37	Jack Perry	Ayush Tyagi	• Satisfies code style requirements	• Displays additional information on clicking ‘more details’ ◦ Graphs to display features	• Stretch Goal: More features + more graphs (if needed) - not hard to add
38	Ayush Tyagi	Jack Perry	• Satisfied code style requirements	• Added ‘info icon’ for more user information on how to use the app	• Perhaps wording could be a little changed but functionality is good
39	Jack Perry	Ke Liao	• Satisfied code style requirements	• No Major Change, only lock upload button after submitting documents to prevent spamming	
40	Ke Liao	Ayush Tyagi	• Satisfied Code Style Requirements • Some minor changes to make the code look cleaner	• Redid the landing page completely ◦ To improve its visual design and convey info about how to use the product	
41	Josh Costa	Jack Perry	• Satisfied code style requirements	• Added more detail to about page • No functional change besides visuals on about page	

🔬 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
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 profile	• 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’
3	Delete Profile	User can delete a previously created profile	1. Click Profile Drop down	• views.py: ‘delete_profile’	Profile is deleted from view and from server	Same as expected	Test Case Code: tests/test_profile.py - ‘test_profile_delete’

			2. Select Delete Button on Profile of choice	• dropdown.js • profileManage.js • profile.html	(and documents)		
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	• 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'

			2. User is prompted to login				
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'

Progress Assessment Change Logs

This page is the landing page to the changes between progress assessments.

Progress Assessment	Date Range	Link
Progress Assessment 1	24/07/2023 → 03/09/2023	N/A
Progress Assessment 2	04/09/2023 → 24/09/2023	Progress Assessment 2 - Change Log

Progress Assessment 2 - Change Log

This page is a log of key changes between Progress Assessment 1 and Progress Assessment 2. This page is purely for assessment purposes to simplify the process of recording changes on one page.

Date Range: 04/09/2023 → 24/09/2023

Table of Contents

Key Changes

- Consolidated link between motivational model and user stories
- Continued frontend design improvements
- Documented Google Colab use in backend
- Restructured team's use of main, development and feature branches
- Testing implementation: Test Case Document
- Deployment considerations: Deployment Document
- Back-End Code Documents: Back-End Documentation

Feedback Table

This table outlines the key feedback from Progress Assessment 1, and the changes made to implement this feedback.

Feedback	Improvement	Rationale
Seek further communication with client outside of meetings.	The team's product owner sought further clarification and recommendations from the client when necessary.	Continued communication with the client ensures the team's product vision is aligned with the client's needs.
Ensure motivational model 'User' is aligned with user stories.	Clarified which users from the motivational model (student, teacher, admin) are aligned with each user story. Requirements Document | User Stories	This resolves any inconsistencies between the motivational model and related user stories.
Frontend could be improved aesthetically, research existing designs.	Ongoing design tweaks and changes throughout the development process, as demonstrated here: UX/UI Design Document | Implemented Design (Sprint 2)	As the team's frontend skills increase and further design research is conducted, improvements can be iteratively applied to the web app's design.
Document how Google Colab was used for generating model weights.	Backend functionality, along with Google Colab's role in generating model weights, has been outlined here: Back-End Documentation	Documentation of the project's Backend functionality promotes better transparency and allows potential future developers to better understand how the system works.
Employ use of branches and pull requests on GitHub.	The team now follows a feature branch, pull request, code review, merge process, clarified here: Code Review Document | Workflow:	This approach follows better coding practices and is more aligned with industry standards.

	GitHub - Friday-11am-Team-2/IT-Project: IT Project 2023	

Progress Assessment 3 - Change Log

This page is a log of key changes between Progress Assessment 2 and Progress Assessment 3. This page is purely for assessment purposes to simplify the process of recording changes on one page.

Date Range: 25/09/2023 -> 20/10/2023

Table of Contents

Key Changes

- More Comprehensive Codebase Documentation
  - 🟧 Front-End Documentation
  - 🟩 Back-End Documentation
  - 🏁 Deployment Document
- Handover Documents summarising each project area and where to find more information
  - 🗂️ Handover Documents
- Front-End Usability Improvements
  - ✏️ UX/UI Design Document | Finalised Design (Sprint 3)

Feedback Table

This table outlines the key feedback from Progress Assessment 2, and the changes made to implement this feedback.

Feedback	Improvement	Rationale
Use more specific language regarding Jira tickets, to allow for better subdivision of labour.	Team members attempted to create more specific ticket names, clearly outlining the specific tasks they were working on.	Clearer language in Jira ticket naming promotes better communication and understanding of which tasks are completed, delegated or unresolved.
Include how Google Colab is used for model weight generation.	Explicit explanation of Google Colab's usage is now documented here: 🟩 Back-End Documentation | Google Co Lab	Clarification of external resource usage promotes better transparency and understanding of our project's development process.
In deployment diagram, Docker should not be wrapped by AWS.	Updated deployment diagram to accurately reflect relationship between Docker and AWS: 🏁 Deployment Document | Deployment Ar chitecture Diagram	Correct representation of system architecture dissuades misunderstanding of how the application functions.
You can further use GitHub actions for automated testing.	Further integrated automated testing and deployment, outlined here: 🏁 Deployment Document | CI/CD Pipelin e Workflow runs · Friday-11am-Team-2/IT -Project	Both the back-end code and deployment process have automated testing via GitHub actions, promoting better overall product quality and reducing manual testing.

Handover Documents

This is the landing page for the documents that summarise the final state of the ‘AuthorGuard’ web application at the end of the project development timeframe. These documents are meant to be used in tandem with the more comprehensive documentation and more serve as a way to quickly find relevant information pertaining to an aspect of the project’s development and implementation.

ID	Name	Link
1	Requirements Summary	Requirements Summary
2	Design & Architecture Summary	Design & Architecture Summary
3	UX/UI Design Summary	UX/UI Design Summary
4	Code Base Documentation Guide	Code Base Documentation Guide
5	Deployment Guide	Deployment Guide
6	Security & Ethics Report	Security & Ethics Report

Requirements Summary

IT Project - Requirements Summary

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

---

Table of Contents

Overview

The purpose of this project was to create a web application that provided non-technical users, particularly those in an education setting, with an interface in which they can compare and analyse the writing styles of different texts through use of a stylometry machine learning algorithm. This document intends to provide a high level summary of this stylometry project's key requirements from a technical viewpoint, as identified by the team during the inception phase.
A more detailed explanation can be found here: 🔍 Requirements Document

Motivational Models

The team initially used a do/be/feel list in tandem with motivational models in order to identify the intended users of the application, the ways in which they can interact with the app, and how the experience should be from both a technical and emotional perspective.

- Intended users: staff and students in either a school or university environment
- Interactions: create a profile, upload documents, compare writing styles of documents, manage profiles and documents
- Technical impressions: useable, reliable, scalable, honest, professional, secure
- Emotional experience: proud, authoritative, empowered, satisfied, confident, eager

These features were then compiled into minimum viable product and stretch goal product diagrams, which can be viewed here: 🔍 Requirements Document | Motivational Models:

Functional & Non-functional Requirements

Extending on these intended interactions and technical impressions of the users, the team compiled a list of functional and non-functional requirements respectively. Functional requirements outlined the key features of the application, and provided the rough structure as to how a user would navigate the site, for example:

- sign-in/create profile (authentication & access control)
- upload/delete documents from profile (integration with database)
- run stylometry algorithm (communication with back-end)
- interpret output and display results (communication with front-end)

On the other hand, the non-functional requirements related more to the application's infrastructure, and directly impacts to how useable the product would be when deployed at a larger scale. The main considerations identified were:

- Usability: a clean user interface, able to be comfortably navigated by non-technical users
- Reliability: accuracy of algorithm results
- Performance: application response time, how long the algorithm takes to run
- Scalability: can the application handle traffic at a school/university level

More detailed explanations of the identified requirements can be found here:
🔍 Requirements Document | Functional Requirements
🔍 Requirements Document | Non Functional Requirements

User Stories

User stories build on the rough guidelines provided by the functional requirements, and help to outline which types of users are using which specific features of the app for what purposes. The team considered user stories relating to both the minimum viable product and stretch goal product, and categorised them according to how import they were to the core needs of the users.

- High Priority
  - Students/Teachers can create a profile which saves their documents
  - Students/Teachers can add/remove documents from their profile
  - Students/Teachers can compare writing styles of their uploaded documents
- Medium Priority
  - Students/Teachers can upload documents of any common text file type
  - Teachers can access a database of student profiles containing students' submitted work
  - Teachers can compare new student work with their existing work from the database
  - Administrators can create/delete/edit profiles within their institution
- Low Priority
  - Administrators can change the analysis model for their institution
  - Students can sign-up/login through common social media platforms
  - Students/Teachers are shown the key areas of similarity/difference between compared documents

The full list of user stories is available here: 🔍 Requirements Document | User Stories

Personas

The team created two example personas to better understand the target audience of the product, one university student and one university teacher. These personas provide further insight into the various backgrounds, intentions and technological capabilities of the target user demographic, enabling the team to better align our product goals with user needs.

The two most notable takeaways from the personas were:

- Ease of use: application should be very easy to navigate, understand and use as needed
- Intentions: comparing documents to check for potential indicators of plagiarism and academic misconduct

The team reflected on these key areas of interest, which reinforced the importance of a clear useable interface, as well as the need to clarify to users how the stylometry algorithm should be used and how results should be interpreted, outlining the fact that any and all results should not be taken as evidence of academic misconduct.

Both personas can be seen here: 🔍 Requirements Document | Personas

Design & Architecture Summary

IT Project - Design & Architecture Summary

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

---

Table of Contents

Overview

This document aims to provide a general idea of the project's architectural design, including what the team's main focusses were and why certain frameworks were used over others. The project's full design document and deployment diagram can be found here:

🖌️ Design Document

🔳 Deployment Document | Deployment Architecture Diagram

System Architecture

During the inception phase the team opted to follow industry standard full-stack development, separating the different project functionality into 3 distinct sub-sections, those being the front-end, back-end and database.

🖌️ Design Document | System Architecture

Front-End

The main required functionality of the front-end was to act as the user interface, providing the user with ways to upload texts, remove texts from their profile, request the algorithm to run on a set of texts and displaying the results. Given that everyone on the team had little to no experience with front-end languages and frameworks, the team opted to use simple HTML, CSS and JavaScript, as those alone were sufficient to create the desired product.

Back-End

The back-end handles all of the requests relating to profiles and algorithm calls, passing back desired information and serving as the intermediary between the front-end and the database. The team opted to use the python web framework Django to handle back-end functionality for two main reasons: everyone on the team had experience in python and it meshed well with the project's source code which consisted of python functions in Jupyter Notebooks.

Database

The database is responsible for storing the user credentials, individual profiles and the text documents associated with those profiles. Ultimately the team settled on PostGreSQL as the database of choice, since it has comprehensive integration with Django projects and acts as a relational database, which suited our needs.

🖌️ Design Document | Database Design

UX/UI Design Summary

IT Project - UX/UI Design Summary

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

---

Table of Contents

Overview

This document aims to assist any future development team of the ‘AuthorGuard’ web application to provide all resources pertaining to the UX/UI design of the system. This document does not contain the actual UX/UI design justifications & wireframes as much as it contains the links to the existing documentation in one centralized location to assist any future developers working on the project.

UX/UI Design Links

This sections contains links to where information can be located related to the front-end design.

- ✏️ UX/UI Design Document - contains the history of UX/UI design iterations and explanations
- 🟧 Front-End Documentation - contains the code base information for the front-end code, which creates the UX/UI design of the web application.

Code Base Documentation Guide

IT Project - Code Base Documentation Guide

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

---

Table of Contents

Overview

This document aims to assist any future development team of the ‘AuthorGuard’ web application to locate the documentation for the existing code base. ‘AuthorGuard’ is a Django-Python based web app and as a result it abides by the general standard principles for any Django application. As a result of this, the Django documentation is quite useful for developers who have not worked with Django previously.

This document does not contain the actual documentation as much as it contains the links to the existing documentation in one centralized location to assist any future developers working on the project.

Code Base Links

The following is a list of links to useful documentation to understand the code base.

- GitHub - Friday-11am-Team-2/IT-Project: IT Project 2023 - Repository Link, including a README
- Repository Directory Information - Outlines each directory in the repository at a high level
- Front-End Documentation - Outlines the JavaScript, HTML & CSS Front-End code base documentation for the ‘AuthorGuard’ web application
- Back-End Documentation | Django Back End Code Information - Outlines the Django code base Information in the context of the ‘AuthorGuard’ application
- Back-End Documentation | Authorship Verification Machine Learning Information - Outlines information about the Stylometry model operations
- Technical Information - Outlines information on running Django application locally as well as Docker container based manual deployment.
- Test Case Document - Outlines a list of all implemented test cases in Django’s testing system.
- Deployment Document - Outlines the deployment architecture at a high level. explaining the basic operations of running the Docker image through AWS Elastic Beanstalk
- Deployment Guide - Outlines a more in-depth step-by-step system for deploying the web application to Elastic Beanstalk through Docker Hub, including repository, CI/CD, database and deployment information.

These links contain all of the documentation the team has written to assist other developers to understand the existing code base.

IT Project - Deployment Guide

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

---

Table of Contents

Overview

In modern day web applications, an important step to move towards a production-ready software system is the creation of a CI/CD pipeline to abide by the agile methodology and the corresponding deployment architecture used by the system to host and run the web server application. For this project, the team created a simple CI/CD pipeline that pushes from the GitHub repository to a AWS Elastic Beanstalk (EB) instance via first building a Docker image of the application and pushing that to the repository first.

The decision by the team to use a Docker container to store the image of the web application was made with practical deployment and handover in mind as Docker is a platform agnostic service. meaning that any team that wishes to deploy the 'AuthorGuard' web application can integrate it more easily with any existing infrastructure they already possess.

This document outlines the steps the team took to create the CI/CD pipeline in the context of the architecture of the system. This document will also discuss the notable issues that are a product of the team's lack of experience in production-ready systems and hence any team that actually aims to create a production-ready version of the 'AuthorGuard' web application with users should ensure the team has sufficient expertise in Docker and the team's cloud computing service of choice (AWS, Azure, etc.).

It is important to remember throughout this document that the team's original aim was not to create a truly production-ready application, but to create a test deployment to demonstrate the 'AuthorGuard web application could in theory work in a production-ready context if additional cloud engineering work was done on the application.

CI/CD Pipeline

As mentioned before, the team created a complete CI/CD pipeline using GitHub actions. The following diagram provides a high level overview of the pipeline the team used for this project.

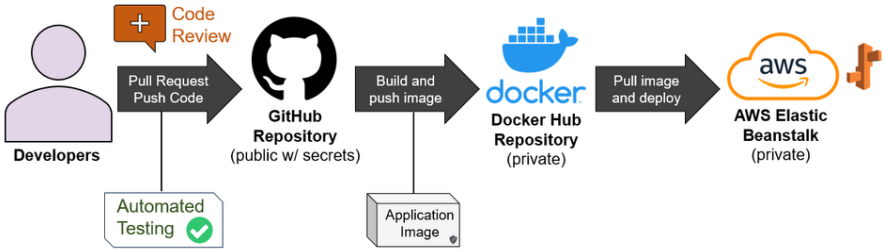


Figure 1 - CI/CD Pipeline Diagram

In terms of continuous integration, the team used pull requests, code reviews, and most importantly, automated testing via GitHub actions & Django's testing framework to create meaningful and useful test cases that ensure that all key operations of the web application are intact. These tests are outlined in the Test Case Document and cover all the main functional requirements of the 'AuthorGuard' web application. These tests are ran on every pull request as well as every push to either the 'main' or 'development' branches of the GitHub repository.

Furthermore, in terms of continuous deployment, the team used an additional GitHub action workflow that is able to deploy to AWS Elastic Beanstalk through Docker. The workflow first builds the Docker image based on the Dockerfile in the repository, encoding the required GitHub secrets into the image. As a result of this system, the repository which stores the Docker image must be private as this image could be pulled and the secret environment variables could be leaked, which would be a significant security risk.

The workflow then pushes this image to the repository, using GitHub secrets to login to Docker Hub securely. After this, the workflow then logs into AWS using an IAM user which has the permissions required to deploy the web application. The action zips and sends the '`Dockerrun.aws.json`' file to the Elastic Beanstalk instance. This file commands the EB to deploy the web application by pulling from the Docker Hub repository named in the file. However, an authorisation token is required for this which is stored in a secure S3 bucket that is connected to the EB instance so that the token can be used to login to Docker Hub and pull the image from the private repository. Thus, this securely deploys the web application to an EB environment by way of GitHub actions and Docker images.

Deployment Steps

Before outlining the steps to replicate the current deployment system and architecture the team has, it is imperative to note the team's lack of experience in the area of production-ready web services using cloud computing. As a result of this, these steps are almost certainly not the best nor the correct way to properly deploy this web application in a production-ready manner.

Hence, any team further developing this project should likely redo the infrastructure to a service they are more familiar with as it will enable the team to ensure the application is secure and integrated with any existing architecture they already control. Fortunately, the decision by the 'AuthorGuard' team to create the application via a Docker image means that the image can easily be built, pushed, pulled and finally deployed to any modern cloud hosting service, which will make this process relatively simple with the existing code base.

The following outlines the main steps the team used to create the current CI/CD pipeline, with links to any documentation, tutorials or question & answer threads that are of relevance to each step.

Setup GitHub Repository

1. Download the zip of our team's repository GitHub - Friday-11am-Team-2/IT-Project: IT Project 2023
2. Create new repository for development team
3. Extract and commit all files in the zip to the new repository
4. Clone the local repository
5. Setup the following environment variables through GitHub secrets
   a. Django Secret Key
      i. Obtain this via running `cd stylometryproject` then `python manage.py shell -c "from django.core.management.utils import get_random_secret_key; print(get_random_secret_key())"` in your CLI
      ii. Save this with the name `SECRET_KEY` in the secrets tab
   b. PostgreSQL Database Information
      i. A new team may want to use a service other than RDS, and as a result, the name of variables can be changed to whatever they wish, but they will also need to be changed under `settings.py`
      ii. Create your own PostgreSQL external database (for RDS, see Create and Connect to a PostgreSQL Database with Amazon RDS)
      iii. Save the following environment variables to secrets
         - `RDS_ENGINE`
         - `RDS_DB_NAME`
         - `RDS_USERNAME`
         - `RDS_PASSWORD`
         - `RDS_HOSTNAME`
         - `RDS_PORT`
      iv. After doing this, the Django app (and Docker image) will now be connected to the web application
   c. Docker Login Information
      i. If using the Docker build and push stages of the CI/CD pipeline, a new team will need to create a Docker Hub account (Docker Hub Container Image Library | App Containerization)
      ii. Save the login information to following environment variables
         - `DOCKERHUB_USERNAME`
         - `DOCKERHUB_PASSWORD`

Docker Hub Repository Setup

If the new team wishes to use Docker Hub to store the image of the web application, follow these steps.

1. Create a **private** Docker Hub repository
2. Change the name of the repository in `.github/docker.yml` on line 50 to the following
   a. `tags:<your docker account name>/<your application name>:latest`
      i. Replacing the <> fields with your information
3. If intending to use AWS Elastic Beanstalk, change line 8 of `Dockerrun.aws.json` to the following
   a. `"Name":"<your docker account name>/<your application name>:latest",`

The docker stage of the CI/CD pipeline is secure enough for production, so these steps can be replicated without as much concern as the following AWS elastic beanstalk steps. Just ensure that the Docker login information is secure and the image will therefore be protected.

AWS Elastic Beanstalk Setup

If the new team is intended to replicate the Elastic Beanstalk deployment system previously used follow the steps in this section. However, it is important to once again note that this is by far the area the team has the least experience in with cloud computing, so it is highly inadvisable to do this if the new team is legitimately deploying to production-ready with actual users due to the likelihood of security risks. That said, the IT Project team has done a reasonable level of diligence in preventing any obvious security errors, but using infrastructure and tools the new team is familiar with will likely be much less risky and may even be simpler.

The following are the steps & links to other information for further assistance used to setup the team's current CI/CD pipeline.

1. Create an AWS Account (Cloud Services - Amazon Web Services (AWS))
2. Create Elastic Beanstalk application & environment to run Docker images (for help, see Website & Web App Deployment - AWS Elastic Beanstalk - AWS & Deploying Elastic Beanstalk applications from Docker containers - AWS Elastic Beanstalk)
3. Modify the lines 73, 74 & 77 in `.github/docker.yml` with the following information, changing the <> fields with the new team's instance information.
   a. `application_name: <your application name>`
   b. `environment_name: <your environment name>`
   c. `region: <your EB region>` (currently `ap-southeast-2`)
4. Create a new AWS IAM User Group and corresponding User that has the following permissions (for help, see Access Management- AWS Identity and Access Management (IAM) - AWS)

a. `AdministratorAccess-AWSElasticBeanstalk`
b. `AmazonS3ReadOnlyAccess`

5. Generate an access key for the newly created user and store the information in the following GitHub secrets (for help, see Managing access keys for IAM users - AWS Identity and Access Management)
   a. `AWS_ACCESS_KEY_ID`
   b. `AWS_SECRET_ACCESS_KEY`
6. Follow this Tutorial's Section from 'Making Docker Repo Private' up to and including 'Uploading to AWS' (Deploying Docker Image to AWS Elastic Beanstalk using GitHub Actions)
   a. This will setup the S3 bucket that stores the secure authorization key to access the private Docker Hub repository and tells you how to create the `dockercfg` file required to be uploaded to the bucket.
   b. Do not commit the `dockercfg` file to the GitHub repository as it contains sensitive information, ensure it is not leaked and is only pushed to the S3 bucket.
7. In `Dockerrun.aws.json`, make the following change to line 4
   a. `"Bucket": "<your bucket name>",`
   b. If you didn't name the file `dockercfg`, you will also need to update the `Key` value on line 5.

If all steps above were followed, the CI/CD pipeline of the web application should now be complete and now on push to main, GitHub actions should be able to build & push the image to Docker Hub, and then after this, tell AWS EB to pull the image and run on port 8000 by sending `Dockerrun.aws.json` in the deploy.zip package to the EB environment.

Further Deployment Tips

If the team is intending to use another service other than AWS to host the web application or attempting to build the current architecture into production-ready, here is some general tips of advice based on the current state of the 'AuthorGuard' application.

- If the new team still wishes to use Docker Hub images but use a new hosting service, then only the '`Generate Deployment Package`' and '`Deploy to EB`' stages of the GitHub action in `docker.yml` need to be replaced with the specific action of the new chosen hosting service. Many services already have existing GitHub actions which may be useful here.
- The team may also want to integrate the database in a better manner as the current system essentially assumes the database and hosting platform are completely independent of one another. As a result, it is likely better for both security & convenience to redo the database connectivity in the application (ensure it still works with Docker if you are using it, see this for assistance Adding a database to your Elastic Beanstalk environment - AWS Elastic Beanstalk) The original team did not do this due to our inexperience with cloud computing technology and also to allow the system design to work with a purely external database service, which the team originally considered as a potential option.
- The team may also want to use `docker-compose` files to mount the environment variables in a more secure way so that the Docker image is not itself required to be secret. Our team lacked the experience with AWS and Docker to correctly setup this configuration in the CI/CD but a more experienced team may have more success (see Using the Docker platfor m branch - AWS Elastic Beanstalk).
- The .env file system is likely a security risk because although all the actual values are stored in GitHub secrets, if at any point the image is compromised, the whole database and secret key would need to be redone and changed in GitHub secrets. This could likely be avoided with the aforementioned docker-compose files, but in the context of AWS, the team was unable to find a secure & well-designed way to store the environment variables besides writing them into the Docker image and using a private repository.
- Currently, as AWS EB does not automatically support HTTPS, this will need to be done if pushing the application to production for use by legitimate users. The team did not wish to spend the financial resources that would be needed to do this given the small-scale of the project but it is recommended for any future teams pushing to an actual user base (Confi guring HTTPS for your Elastic Beanstalk environment - AWS Elastic Beanstalk).

IT Project - Security & Ethics Report

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

---

Table of Contents

Overview

This document contains two reports the team wrote. The first report is the ‘Ethics Report’ and it is about the ethics of the ‘AuthorGuard’ web application, exploring issues such as data privacy, team member code of conduct as well as machine learning ethical considerations. Beyond this, the second report is the ‘Security Report’ which outlines some of the security considerations and provides a security evaluation of the project as of the final release, drawing attention its strengths and weaknesses.

Ethics Report

The issue of ethics in the modern software and information technology industry is a critical and evolving concern that impacts all facets of the software development process. The ethical considerations of a project are complex and influence the requirements and design decisions made in regard to the software (Thomson and Schmoldt 2001). This ethics report aims to discuss the potential ethical issues arising from the creation of our team’s ‘AuthorGuard’ web application, the capstone project for COMP30022. Although smaller in scale compared to other full-stack software projects in the industry, there are still a plethora of considerations in regard to the ethics of planning, developing, deploying and handover of the software.

A lack of understanding of the importance of ethics in software development is a common issue among university-level computing and software development students (Calluzo and Cante 2004). Hence for this project, the team made sure to discuss and brainstorm possible ethical issues that could arise during the project inception phase and identify strategies and design decisions that will help rectify these concerns.

The most profound and evident area of concern regarding the development and deployment of the web application in this project was the nature of the documents required to be stored for the algorithm to execute its main functional goal. Given that the primary goal of the web application is to analyse user uploaded documents and perform authorship verification and display these results to the user, there are clear ethical concerns about storing and analysing the text content of personal user documents.

The first issue emerging from this is the fact that team members have the administrative privileges to read document contents that have been uploaded to user created profiles. This evidently creates the capacity for misuse as user information can be accessed without the user’s knowing consent, which most reasonable people would regard as a breach of ethical conduct in regard to user data privacy (Hand 2018). Likewise, the algorithm and server-side program itself also analyses the text content of the documents, which raises further ethical questions and concerns.

The team implemented two major techniques to help minimise this issue. The first technique was to ensure that the website included a disclaimer about the nature of the document storage and the potential risks of uploading personal documents to the database, encouraging users to depersonalise documents if they wish to avoid these concerns. This technique serves to amend this ethical issue as it promotes transparency, making users aware of the risks of using the application and the ways their uploaded documents can be accessed and analysed.

The second technique was to create a clear and concise code of conduct and ethics policy in the handover documents. This outlines in detail the ways in which the program interacts with user’s personal information and a recommended code of conduct to be enforced if the program was to be scaled up, in which only certain users can access the database information.

Beyond this issue, another key ethical concern for the program is the nature of how the application's machine learning model is trained, as well as how it is used. As a machine learning model, the stylometry algorithm is subject to bias based on the training data, and this can have problematic consequences on users (Yapo and Weiss 2018). The training dataset for the model for this project was the 'PAN14-Author-Profiling' dataset, which contains documents that are both publicly available and depersonalised.

Although this helps prevent unauthorised use of others' written works, there is still potential for biases to occur as a result of the dataset's lack of writing style diversity. A model trained on one style of writing, such as creative texts, would provide less accurate results when analysing a different writing style, such as a scientific report. The team partially catered for this by allowing administrator users to train and import their own model weights, to better suit their specific needs. Furthermore, disclaimers were implemented to ensure that users are familiar with the potential inaccuracies and biases that the model could produce, and understand that results are not guaranteed to be correct.

Another area in which ethical issues could arise is code transparency. The team's code repository is publicly available for this project. This is beneficial as it means that users can use the app with the confidence that there is no malicious code in the application as the codebase is publicly known, giving a relatively sound ethical code guarantee to the users.

Thus, there are a number of ethical issues regarding the development and deployment of the 'AuthorGuard' web application. However, the team has taken reasonable precautions to avoid the negative implications and consequences of failing to properly account for these ethical problems.

Security Report

The issue of security in the modern software and information technology industry is an essential consideration throughout all stages of the development process for a software project. Given the prevalence of IT technologies & software systems in our everyday life, it is critical that software designers consider security throughout their DevOps work (Kumar, Khan and Khan 2015). In the context of our 'AuthorGuard' web application in COMP30022, the team took a number of measures to ensure the system was secure by considering a variety of common security risks and concerns, focussing particularly on sensitive data access, database security, file-checking and Docker image privacy. Likewise, the team also considered some common web application attacks, such as Cross-Site Request Forgery and SQL injection attacks.

As the 'AuthorGuard' web application is Django-Python based, it inherently requires use of sensitive information to function as intended. This includes the 'secret key', which is used for cryptographic signing to ensure secure communication between the client and server sides, and must be kept private to avoid enabling bad actors to tamper with sent packets. To maintain information security, the team generated a new secret key for the web application and secured it in GitHub secrets, which is a secure way to store sensitive information, and allows the application to access the secret key as an environment variable.

Similarly, our application uses Amazon Web Service's Relational Database Service (AWS RDS) to host an external database for the web server. In order to enable the Django application to access the database API, the sensitive database login information (including username, password, and hostname) needed to be securely stored, otherwise the database could become compromised which increases vulnerabilities to many different attacks (Murray 2010), such as unauthorized viewing, modification, or deletion of database entries. Again, the team used GitHub secrets to store database information so that the web application can access this information as environment variables when running.

Beyond this, since 'AuthorGuard' encourages users to upload their own documents to run the analysis model on, another handful of security vulnerabilities emerge from this. To mitigate these risks, the team implemented both client and server side checking of uploaded documents to prevent any illegitimate documents from entering the system. The client side filters out disallowed file types, and sends the file content of the documents as a byte string to the server. The server then double checks the file type is valid, before decoding the data into meaningful text information and storing it in the database. The result of this is that actual documents are not stored on the database, instead only storing textual information that those documents contained, which prevents malicious files from being uploaded to the database, thus improving security.

Furthermore, there were security considerations involving the team's use of Docker to host and deploy the web application to an AWS Elastic Beanstalk (EB) instance. In particular, the complexity around mounting environment variables using AWS EB, combined with the team's inexperience using AWS, resulted in the docker image being encoded with sensitive environment variables. This could present a major risk if the image was to be pushed to a public container repository.

However, the team opted to use Docker Hub's private repository system which allows for secure storage of sensitive containers. As a result, the team had to store the private repository access token on an AWS S3 bucket, which is a secure way to store sensitive data for AWS services to use when running and deploying web applications. One advantage of this method is that access to the bucket can be setup through AWS Identity Access Management (IAM), ensuring only authorised users can access the repository access token. The AWS and Docker Hub login information are also stored on GitHub secrets to allow the CI/CD pipeline actions to build, push and deploy the image in a secure way.

Beyond these security considerations, the team also ensured that the Django web application was protected against certain common web attacks, such as a Cross-Site Request Forgery (CSRF) attack; in which an attacker forces users to execute undesired actions on a web application (Kombade and Meshram 2012). Django supports in-built CSRF protection decorators that can be placed on the server-side to ensure that HTTP requests which would cause database changes (POST requests) are verified against the current user that is logged in. This security mechanism works by using a cookie that stores a CSRF token on the client-side and the same token on the server-side, rejecting any POST HTTP request that contains a mismatched token. The team ensured that the web application had these decorators before being deployed to an Elastic Beanstalk instance, thus preventing CSRF attacks.

Likewise, the team also considered the security risks of a SQL injection attack, a common web attack in which SQL keywords are placed into user inputs and uploaded to the server-side, which can cause issues such as unintended database access and modification (Shar and Tan 2012). In order to mitigate this risk, the team intentionally constructed the codebase in a way that aligns with Django's in-built SQL injection prevention mechanisms, that is, to ensure that database access is only done through Django's Object-Relational Mapper (ORM).

Overall, the security measures in place are suitable given the scope and requirements of the project, and the team have made efforts to bolster the web application's security throughout the development process. There are, of course, ways to further improve security, such as obtaining a TLS certificate and configuring the web server to use HTTPS, which encrypts data during transmission to prevent unauthorised reading and tampering of files. Additionally, configuring the docker to mount private environment variables at runtime would reduce the risk of exposing these variables if the image is compromised.

Thus, the team performed and implemented a number of measures to help mitigate common security risks, including sensitive data, file management and CSRF and SQL injection attacks. By way of these protection measures, the team was able to bolster the security of the 'AuthorGuard' web application.

References

- Calluzzo, V.J. and Cante, C.J. (2004). Ethics in Information Technology and Software Use. *Journal of Business Ethics*, 51(3), pp.301–312. Accessible at https://link.springer.com/article/10.1023/B:BUSI.0000032658.12032.4e
- Hand, D.J. (2018). Aspects of Data Ethics in a Changing World: Where Are We Now? *Big Data*, [online] 6(3), pp.176–190. https://www.liebertpub.com/doi/abs/10.1089/big.2018.0083
- Kombade, R.D. and Meshram, B.B., 2012. CSRF vulnerabilities and defensive techniques. *International Journal of Computer Network and Information Security*, *4*(1), p.31. Accessible at https://j.mecs-press.net/ijcnis/ijcnis-v4-n1/IJCNIS-V4-N1-4.pdf
- Kumar, R., Khan, S.A. and Khan, R.A., 2015. Durable security in software development: Needs and importance. *CSI Communication*, *39*(7), pp.34-36. Accessible at https://www.academia.edu/download/39362220/Durable_Security_in_Software_Development_need_and_importance.pdf
- Murray, M.C., 2010. Database security: What students need to know. *Journal of information technology education: Innovations in practice*, *9*, pp.IIP-61. Accessible at Database Security: What Students Need to Know
- Shar, L.K. and Tan, H.B.K., 2012. Defeating SQL injection. *Computer*, *46*(3), pp.69-77. Accessible at https://ieeexplore.ieee.org/abstract/document/6265060/
- Thomson, A.J. and Schmoldt, D.L. (2001). Ethics in computer software design and development. *Computers and Electronics in Agriculture*, [online] 30(1-3), pp.85–102. Accessible at Ethics in computer software design and development
- Yapo, A. and Weiss, J. (2018). *Ethical Implications of Bias in Machine Learning*. [online]. Accessible at ScholarSpace :: Home . Ethical Implications of Bias in Machine Learning