School of Electronics, Electrical Engineering and Computer Science

CSC3069 Software Engineering Enterprise Project Team Report

PATHFINDER:

Navigating Academic Success - A Comprehensive Report on the Development, Implementation, and Evaluation of an Intelligent Module Selection and Academic Progress Tracking System

Authors:
Dean Logan
Ross McAllister
Kyle McComb
Conor Nugent

Friday 12th January 2024

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

CSC3069 – SOFTWARE ENGINEERING ENTERPRISE PROJECT

Contents

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the CSC3069 Module Handbook and in the provided MS Word Template.
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. The page number is within the specified page range (min 50 max 60), excluding bibliography and Appendixes
4. Is clearly presented and proof-read,
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

We declare that we have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism – https://www.qub.ac.uk/directorates/sgc/learning/LearningResources/Plagiarism/ - and that the attached submission is our own original work. No part of it has been submitted for any other assignment and we have acknowledged in our notes and bibliography all written and electronic sources used.

Each Team Members Signature *Date of Submission* 12/01/2024

Student Name	Student Number	Signature
Dean Logan	40294254	***Dean Logan***
Ross McAllister	40291577	***Ross McAllister***
Kyle J. McComb	40295231	***Kyle J. McComb***
Conor Nugent	40296257	***Conor Nugent***

A signed and completed declaration sheet must be included on the second page of the submission of the Software Engineering dissertation submitted for assessment.
Work submitted without a completed declaration sheet will **NOT** be marked.

Abstract

This paper presents the development, implementation, and evaluation of PATHFINDER, an innovative system designed to assist students in their academic journey. The system facilitates informed module selection by aligning students' interests and career goals with available courses. Additionally, PATHFINDER aids in tracking academic progress, utilising a robust calculation mechanism based on module grades and overall degree requirements. The report provides a detailed account of the project's lifecycle, encompassing requirements gathering, design documentation, implementation details, software testing methodologies, user evaluation processes, and effective project management strategies. Through the exploration of each phase, this paper sheds light on the challenges, successes, and lessons learned during the development of PATHFINDER. The results of user evaluations attest to the system's effectiveness in empowering students to make informed academic decisions and monitor their progress, contributing to a more fulfilling and successful educational experience.

The PATHFINDER project can be found within this GitHub repository https://github.com/KyleMcComb/Pathfinder.

Acknowledgements

Thanks to John Williamson, Computing Science, University of Glasgow for giving permission for EEECS to use and adapt their project report template.

Thanks to Dr Charles J. Gillan, Senior Lecturer, Queen's University Belfast, for his guidance as our Project Champion and his support throughout the duration of the project.

Thanks to Adam Logan, fellow Technology Degree Apprentice for providing useful insight while completing a user evaluation.

Thanks to Scott McDonald, Computer Science Student at Queen's University Belfast for providing useful insight while completing a user evaluation.

List of Figures

List of Tables

1 Introduction

1.1 Project Vision

The discipline of Electronics, Electrical Engineering and Computer Science (EEECS) is vast, diverse, and always evolving. As a result, it encompasses many sub-disciplines such as; Artificial Intelligence, Computer Networks, Cyber Security, Embedded Systems and so forth. A key challenge faced by students is making informed decisions about module selection and what career path they want to go down due to the vast avenues that The School of Electronics, Electrical Engineering and Computer Science at Queen's University Belfast offers.

Pathfinder, a novel system developed by the team was designed to address this problem in CSC3068 by providing detailed module recommendations and career guidance based off the interests of EEECS students through a central chatbot. This not only equips students with the necessary knowledge but also carves out a clear path to follow in both academia and industry.

Distinctive features of the Pathfinder system include the Central NPL/ML Chatbot, a web-scraping spider, a module information interface, a grade dashboard, a Django Admin interface and a login and signup page.

Features of Pathfinder System	
Distinctive Features of Pathfinder	**Brief Description of each feature**
Central NLP/ML (Natural Language Processing and Machine Learning) Chatbot	Serves as the main interface for EEECS students, offering tailored module recommendations based on student interests, detailed module overviews, and career opportunities linked to module choices.
Web-Scraping Spider	Periodically scrapes (monthly) Indeed for up-to-date career information and job listings, integrating this data with the chatbot to provide relevant career opportunities aligned with student's module interests. This is performed by the admin
Module Information Interface	Offers detailed information about each EEECS module, including name, code, lecturer, level, semester, weight, and assessment details. Features a search bar and filtering options for easy navigation.
Grade Dashboard	Allows students to track their academic progress, displaying overall degree progress, grades for each module, module and assessment averages, and remaining requirements for degree completion.
Django Admin interface	Facilitates efficient database management for administrators, enabling CRUD operations, simplifying complex database tasks, and ensuring data integrity through ORM. Includes backup management capabilities with local and cloud (Microsoft Azure) storage options.
Login and Signup Page	Enables account creation for students, incorporating 2FA/TOTP for enhanced security. The signup process includes essential student information, and the login page is equipped with CAPTCHA to prevent automated attacks.

The central chatbot is a core feature of Pathfinder, it lays the foundation for the system and will be the main point of access for EEECS students. This chatbot can provide a list of all modules in EEECS for all years (1st to 4th Masters). It can also provide a list of all modules that are relevant to your interest. For example, if a student typed: "I like Artificial Intelligence", it would recommend a list of modules that are related to or include Artificial Intelligence, whether that be in the name or description. An example of a module recommended would be the "CSC2062 – Introduction to AI and Machine Learning" module. The chatbot would provide the student with a detailed overview including; the module code and name, the level, the weight, what lecturer teaches the module, each assessment, what type of assessment it is and the percentage each is worth. The semester the module is in and the pathways the module is available to. The student will then be prompted to enter in a number corresponding to the module in the list and whether they are interested in the module, if so, the chatbot will register their interest for this module as well as provide the relevant jobs web scraped from Indeed that match the module. However, if they aren't interested in any of the modules, they can simply enter "0" and the chatbot will not register their interest. This benefits prospective and enrolled students in several ways. One major benefit is that student interest is a key driver for effective learning and engagement, as emphasised by (Harackiewicz, Smith, and Priniski, 2016).

Their research highlights the importance of promoting interest in educational settings, arguing that when students find the content personally relevant, their engagement and learning outcomes improve significantly. By aligning modules with student interests, it not only improves their ability to learn but it also shapes a clear path that the student can follow, both in academia and industry. This benefits prospective students as they can get an overview of modules they're interested in, that different pathways offer, and this can guide them when applying for a pathway.

In CSC3068, Pathfinder did not have the capability to identify languages other than English and this had a negative impact for international students. However, since then, the team has integrated a language adapter which can detect the following languages: Spanish, French, German, Italian, Simplified Chinese, Traditional Chinese, Malay, Urdu, Hindi, Yoruba, and Bengali. The chatbot can identify each language and will then reply in the specified language with the following: "Sorry, this chatbot only supports English. Please use https://translate.google.com/ Google Translate to communicate in English.". This will then take the student to Google translate with the specified detected language URL. This benefits international students as it will still allow them to interact with the chatbot and will help in not isolating them as this is an important resource useful to all EEECS students.

Complementing the chatbot's tailored guidance is the web scraping spider which is used for gathering detailed and relevant career information and jobs from Indeed. This spider webscrapes Indeed every month and is performed by the admin, this ensures that the data is kept up-to-date and relevant to the actual job market at the time. This is integrated with the chatbot after recommending modules based off the student's interest. The chatbot will take this information from the spider and provide relevant jobs based on their relevancy to modules the student is interested in. This benefits prospective, enrolled and also alumni students in distinct but interconnected ways. For example, prospective students benefit as it provides context into future career possibilities associated with different modules – showcasing real-time job market data linked to specific EEECS modules. This helps prospective students make informed decisions about their potential academic journey. For those already enrolled in their EEECS degrees, the integration of up-to-date job market information helps students align their current and future academic choices with relevant career opportunities. For alumni, the Pathfinder system remains a valuable resource for ongoing career development. As the job market evolves, alumni can use Pathfinder to stay informed about new roles or emerging fields that align with their academic background. This ensures that even after graduation, EEECS alumni have a tool that supports their learning and professional growth. As such while the benefits differ for each type of student, they all aim to provide the student or alumni with a clear path to aim for, no matter the stage they're currently at.

Another key exciting feature of this novel system is a detailed Module Information interface. This page includes the following: The module name and description, the module code, the lecturer who teaches it, what stage (level) the module is, what semester, the weight of the module, the assessment details within the module – number of assessments, the type of assessment and the percentage each is worth. In addition, this also includes a search bar and filter – this can be used to search for a particular module or filter modules based off semester, stage and pathway. This is not only beneficial to currently enrolled students but also prospective students. This feature provides a detailed overview of every module in EEECS from 1st year to 4th year (Masters), which can help them make a decision on the modules and pathway they want to choose to aim in their academic and professional goals.

An important and exciting feature for EEECS students is the grade dashboard within Pathfinder. This provides a place for EEECS students to view their overall degree progress, their grades for each module they took in stage 1, 2, 3 (and 4 if they took a masters), it also shows them their module average across all stages, their assessment average across all stages and how much percent left they need to earn their degree. Students can enter their grades for each assessment they took for each module across all stages. This dashboard provides a clear and easy way to view their modules and provides them with a tailored dynamic graph based on the grades they entered for each of their modules/assessments for each stage. This benefits currently enrolled EEECS students as it allows them to monitor their academic progress and provides a clear visualisation of their performance in various modules and what they need to get to achieve their desired degree classification. In addition, it also benefits alumni as it provides a central location and acts as a historical record to view all of their previous grades for each stage instead of having to track down multiple transcripts.

The Django Admin interface in the Pathfinder system is an exciting feature that enhances the database management for admins (designated lecturers), suited for even users with limited technical expertise. It's user interface simplifies complex database operations, enabling admins to efficiently handle critical tasks. This includes CRUD (Create, Read, Update and Delete) functionality, this is important for maintaining the dynamic content of the system. This interface benefits the admins as it allows them to easily create new student profiles, update module details or remove outdated information, ensuring the database remains current and relevant without the need to write extensive SQL queries. It does this through ORM (Object-Relational Mapping) – the interface abstracts complex database interactions. The benefit of this is that it saves time, minimises potential errors with manual database operations but it also adds a layer of security and ensures data integrity. Changes via the Django admin interface remain consistent and reliable. Another significant feature of this

interface is its capability to manage backups. Admins can create, restore, and manage backups both locally and via the cloud – Microsoft Azure. By utilising a dual backup system, this also enhances data security and ensures that Pathfinder can quickly recover and maintain uninterrupted service in the event of unforeseen data loss. In addition, the date is updated in real-time - this means that any changes made in the database are instantly reflected across Pathfinder. This is important regardless of if the admin is adding new modules, updating existing content or restoring data from backups as it assures that students always have the most up-to-date information.

The final notable key feature that will be discussed for Pathfinder is the Login and Signup page which also incorporate 2FA/TOTP. The signup page requires the student to enter their student number, name, student email, pathway, stage and semester. This will then send the request to the admin and the admin will then add the student to the system. Next is the login page, this requires the student to enter their student number, password and then complete a CAPTCHA to login. This is to prevent automated attacks and reduces spam registrations. Once logged in, in the settings page, 2FA (Two-Factor Authentication) can be enabled and a QR code is available to scan with the student's designated authenticator app (Google, Microsoft etc). Upon logging in again, they will be prompted for an TOTP (Time-based one-time password). Both 2FA and TOTP adds an extra layer of security, this is especially important for EEECS students as their account contains sensitive academic information and personal details. The benefit of the login/signup page is that it allows students to create accounts so that they can get tailored module/career recommendations and utilise the grade dashboard to their specific modules (if applicable).

As previously discussed, the main beneficiaries of Pathfinder are prospective EEECS students, currently enrolled EEECS students (1st to 3rd or 4th if studying a masters) and EEECS alumni. In essence, the goal of Pathfinder is aid EEECS students in their academic and professional development.

1.2 Domain Analysis

The current existing systems I will talk about in this domain are relevant to Queen's University Belfast but aren't specifically intended for EEECS students. They are tailored to all students who attend the university.

The first existing system is QSIS – Queen's Student Information System. This provides all QUB students with class enrolments, grades, transcripts and more. This system is tailored to help students enroll and register, however it struggles when students want more detailed information about their choice of modules and what impact it has on their subsequent years/stages at the university. If they do a certain module, will they be limited from doing other modules in the next stage? This system doesn't provide what Pathfinder sets out to do. In addition, the limitations of QSIS is that it has an outdated Oracle database, this makes it incredibly hard to maintain but also hard to update the system with new changes as it would require an entire overhaul which would leave the system inoperable for students and thus not a possibility. In addition, it has poor mobile support, this can make it hard for students when they need to enroll for modules and aren't able to use a laptop or desktop – especially when these modules are popular, and availability is limited. The user interface of QSIS is also out-dated and hard to navigate. Students frequently check "Academic Record" when they need to view their results, however these are actually located in "Exams/Graduation". This can make viewing what you need to difficult. In addition, the data provided to students is limited. They can view their overall grade, but they can't get a detailed breakdown into how much each stage is worth, their grade of each assessment for each module, their overall degree progress and how much they have left to earn their degree. These are all limitations that the system – Pathfinder aims to solve.

The second existing system I will talk about in this domain is – Queen's University Belfast "AI Chatbot" which use's Microsoft Copilot studio. The team found this system to be tailored to prospective and currently enrolled students for all subjects at QUB. This system aims to use AI to cover broad and general questions such as; "When was Queen's founded?" or "When are the staff holidays". Thus it's limitations are that it struggles with specific subject academic queries, such as providing all or specific EEECS modules whether that be for a certain stage or subject. Or providing detailed module recommendations.

While both systems aim to aid students throughout their academic journey at Queen's University Belfast, they don't adequately support EEECS students in making informed decisions about module selections and career paths which Pathfinder aims to solve.

1.2.1- Comparative analysis of Pathfinder

The unique approach the team took for Pathfinder is that the team tailored it specifically to EEECS students, providing relevant and specific guidance on module selection and their career path for the EEECS industry. This contrasts with QSIS or QUbot which provides broader and more general functionality but lacks depth in any specific field/school.

Pathfinder also uses sophisticated algorithms to interact with students in a conversational manner. This is a significant leap from the basic query-response functionality of the AI Chatbot used by Queen's University. Pathfinder can understand and respond to complex queries about EEECS modules, suggest modules based on student interests, and even link these

modules to potential career paths. The system's integration with a web scraping spider to gather relevant and up-to-date job market data is a novel feature. This allows EEECS students to not only explore their academic modules but also to understand how these modules align with current job market trends and opportunities. This integration is rare in academic advising tools and offers students practical insights into industry and their career planning.

This system is also developed with a modern user interface that is easy to use and fully responsive across devices. This ensures ease of access and usability, addressing the limitations of a less mobile-friendly system like QSIS. The design consideration the team took was to have a seamless user experience regardless of whether the student was on desktop or mobile.

Pathfinder benefits prospective, currently enrolled, and even alumni students which might not be the case for QSIS and QUbot. QSIS is primarily for students already enrolled or enrolling in the university while QUbot focusses on prospective and newly enrolled students. Our system is open to everyone but most importantly is beneficial to each type of student. Prospective students can see which modules they're interested in for different pathways, current students can get module recommendations for next semester/year and alumni can benefit from viewing a breakdown of their degree classification, module results for each year and assessment results for each module – this breakdown doesn't exist in QSIS.

1.2.2- Discussion of alternative solutions

Alternative implementations for this system were considered. One of these alternatives was an interactive system without chatbot functionality. Initially, a concept for an interactive software tool was considered. This tool would allow EEECS students to input their interests and receive recommendations for modules and potential career paths based on these inputs. It would function as a straightforward query-response system without conversational capabilities. While functional, the team believed this approach lacked the interactive and engaging elements that a chatbot offers. It was more static and less capable of providing a personalised and adaptive experience. Additionally, the absence of conversational interaction meant a less engaging user experience, which is crucial for student involvement and interest. Interactive tools are common, and the team believed this approach lacked an innovative edge. A simple query-response system, though useful, did not stand out as particularly novel or exciting.

Another considered alternative was a mobile application tailored to provide module and career guidance specifically for EEECS students. This app would have been designed for ease of use on mobile devices. However, there were a couple limitations. A primary limitation was accessibility, prospective students and those not fully integrated into the university may be less inclined to download an application. In addition, developing for multiple platforms (Android and IOS) presents significant resource and maintenance challenges, for example, the system would require regular updates to stay compatible with the latest version of Android and IOS. A mobile application, while convenient for some, could inadvertently exclude those without access to compatible devices or those who prefer a desktop experience. A web-based solution like Pathfinder ensures that no student is left out due to device limitations. Another limitation would be the functionality the team would be able to offer to EEECS students. Developing the application for mobile could make certain complex features such as the web scraping spider or chatbot not feasible as mobile platforms have stricter limitations especially when dealing with background processes or data handling. The Django Admin interface would also not translate well to a mobile interface. This is a vital feature for Pathfinder as it allows admins to easily make changes to the database through the interface. By creating a mobile application, this interface may not be possible or would have to be hosted separately and would add another layer for the admin to access.

Pathfinder was the optimal implementation choice compared to these two alternatives. There are many reasons for this. Unlike the interactive system without a chatbot, Pathfinder's chatbot offers a more dynamic, engaging user experience. Its conversational interface allows for more natural interactions and makes it easier for EEECS students to explore their interests in modules and career in a user-friendly way. The chatbot can adapt responses based on user inputs, providing a tailored experience that a simple query-response system cannot match. Pathfinder is a web-based platform and is accessible to all students regardless of the devices they own. This makes it much more accessible as students won't have to download an application. They can simply connect via a browser even with a mobile phone as the system is designed to be responsive to any device whether that be mobile phones, tablets, laptops, or desktops. This system has the convenience of mobile access without the limitations and resource demands of a standalone mobile application. Pathfinder can also have more complex functionality than a mobile application such as a web scraping spider or an admin interface. This improves the overall system for not only EEECS students but also administrators. Pathfinder is set apart from traditional systems as it makes use of novel and exciting technology like web scraping and tailored chatbot functionality. This provides EEECS students with specific and tailored module recommendations based off student interests and career guidance with the latest job data so it's relevant to the current job market.

2 Updated Requirements

The requirements section outlines the process to develop changes made by the team to Pathfinder's requirements since its previous submission. These updates range from rewording existing requirements for further clarification to implementing various new requirements generally to improve the system's security.

2.1 Requirements Elicitation & Analysis Process

The team followed the requirements process activities defined by (Hickey, 2003, p. 3) to aid in the development of a complete requirements specification. The activities defined are elicitation, modelling, triage, specification, and verification, where it was noted that the activities of this process are to be completed in parallel and not in series as many assume.

- Elicitation: determining the needs of stakeholders. (Admin, Current Enrolled Students, Prospective Students and Alumni).
- Modeling: Analysing models of requirements for incompleteness and inconsistency.
- Triage: Determine the subset of requirements appropriate for specific releases.
- Specification: Documentation of desired behaviour.
- Verification: Determine the consistency, completeness, and suitability of requirements.

As recommended by (I. Sommerville, 2016), the team undertook a user-centred approach to elicitation and analysis by engaging with the system's intended users, namely the students in EEECS. This approach aimed to understand the users' pain points with the current system (QSIS and module description document) and to identify their requirements for a more effective and efficient system.

From the techniques described by (Tiwari et al, 2012, p.6) the first step chosen was an internal brainstorming session to determine the needs of our stakeholders. As the team comprises students (intended users), this technique allowed for faster idea generation, which the team could be confident was an excellent foundation upon which to build. From this, the team conducted informal interviews with the students outside the team to encourage honest feedback on their expectations of the system. Research was then conducted to explore the current state of the problem, as presented in the initial report.

Following these interviews and brainstorming sessions, the team modelled the students' feedback into critical features that were frequently mentioned. These features were further analysed to derive a complete set of functional and non-functional requirements to implement. From this, the team performed the triage activity to create two sets of requirements, one for the first prototype and another for the viable product submission. To ensure the verifiability of requirements, the team strongly emphasised that all requirements were testable and that their completion could be confidently asserted after testing.

Our team adopted a more agile development approach for our prototype submission for CSC3068. The first set of requirements could be iterated throughout the development process based on the performance of implemented features and possible additions. The team's membership as intended users of the system had significant advantages of quick iteration and feedback collection without formal meetings, which could have been time-consuming to schedule. This enabled the team to better tailor the system to the preferences and needs of potential users.

The team held a meeting at the beginning of CSC3069 to update the requirements section by reviewing each requirement and determining whether it had been completed or not. Insights gained from the previous module were also used to make any necessary updates to the requirements list; some of these insights were obtained by feedback from the project champion, and others came after our Alpha and Beta testing. This allowed the team to ensure that all requirements were implemented and aligned with the project's goals and original problem statement. During this meeting, the team also planned the final development process, which helped ensure all requirements were met and the project was completed successfully. The updated list is documented below; all the remaining requirements are shown in Appendix 2.1.1.

2.2 Non-Functional Requirements

Requirement Number: Chatbot-002	**Requirement Type:** Non-Functional
Description: The ability of the chatbot to know that the language being entered is not English, possibly identify what language is being entered, then reply with a standard error message in that language saying, “I’m sorry, but I can only take responses that are in English, here is the student help information link that may be able to help you. “. Also, be able to separate random characters from an actual language and respond with an appropriate message.	
Rationale: Students who speak another language may try to use that language for the chatbot, so there needs to be an error message explaining that the chatbot only accepts English within the language they entered to ensure the student understands how the chatbot works.	
Fit Criterion: When the word “Hola” is given to the chatbot, it can identify this as Spanish and respond with the error message above in Spanish. The chatbot must accept “hdfjkahfda” as input and reply with an error message.	
Update: The team changed the response message from providing an email to the student support office to say: "Here is the student help information link that may be able to help you. "	
Reasoning: This method allows students to seek help through formal communication methods with the University’s student support. An advantage to this is that it enables Pathfinder to provide the most up-to-date information to students without having to make changes to Pathfinder. Another benefit is that students may find their required support faster through other options such as FAQ’s instead of only through email.	

Requirement Number: GUI-004	**Requirement Type:** Non-Functional
Description: The website must be responsive; it must be able to adapt to various standard screen resolutions. (E.g., usable up to 24-inch 1440p, commonly on 24-inch 1080p (16:9 ratio by 1920 x 1080 resolution), down to mobile devices, and with everything in between).	
Rationale: The website will be used on various devices with different screen sizes. Having the website be responsive and change with the size of the browser will increase accessibility as it gives users the ability to use the website both on larger devices like a monitor or on smaller devices like a mobile phone.	
Fit Criterion: Check if all UI elements are visible on a 24-inch screen on both 1080p and 1440p, then resize the window to varying sizes, ensuring that the website adapts to these changes and resizes the UI elements accordingly. Then, check that the website is suitable for mobile use by navigating to the website on a Galaxy Fold (280px X 653px) mobile and confirm that all UI elements are displayed correctly.	
Update: Added specific standard screen resolutions within the description to which the application must scale correctly. Within the fit criteria, it is now stated that 1440p is the maximum resolution supported down to the Galaxy Fold (280 X 653), a small and thin mobile device.	
Reasoning: Adding these specific resolutions allows for more straightforward requirement testing. These screen sizes have been selected as they are common standards across the display industry, with the Galaxy Fold being an extreme edge case on the small end.	

2.3 Functional Requirements

Requirement Number: GUI-Chatbot-002	**Requirement Type:** Functional
Description: There must be a place for the user to see their past inputs and the responses from the chatbot within the current session and display them in a readable manner. The website shows both the messages the chatbot has received and the responses that the chatbot has returned.	
Rationale: The user needs a place to see the conversation they have been having with the chatbot.	
Fit Criterion: Confirm that text is being displayed on the website and that it is clear which text is from the chatbot or the user.	
Update: Added extra clarification within the description: “See their past inputs and the responses from the chatbot within the current session”.	
Reasoning: Beforehand, this requirement could have been confused with stateful saving, allowing the student to see their past conversations. With the update, the requirement is more apparent that it is only within the current session that the student should be able to see their previous messages and the Chatbot’s response.	

Requirement Number: GUI-Login-004	**Requirement Type:** Functional
Description: The front end must have input for a Captcha challenge-response test.	
Rationale: This will be used to determine humans from computers and deter bot attacks.	
Fit Criterion: Users can only log into the system after completing the test.	
Update: This is an entirely new requirement.	

Requirement Number: GUI-Login-005	**Requirement Type:** Functional
Description: There should be a place to allow users to input a multi-factor authentication code.	
Rationale: This will be used to make user's accounts more robust to hackers.	
Fit Criterion: If users have multi-factor enabled, they must enter their code before logging on. If the user does not have it enabled, they are not required to input anything.	
Update: This is an entirely new requirement.	

Requirement Number: Login-006	**Requirement Type:** Functional
Description: Allow users to register a TOTP device to their account.	
Rationale: To allow for multifactor authentication, users must be able to register devices to send their one-time passcodes.	
Fit Criterion: After registering a device, a code is successfully sent to the device when the user attempts to log in.	
Update: This is an entirely new requirement.	

Requirement Number: Login-007	**Requirement Type:** Functional
Description: Allow the system to check if a user has multifactor authentication enabled by having a TOTP device registered to their account.	
Rationale: The system needs to know when to ask for an OTP when a user is logging on.	
Fit Criterion: A one-time password is requested by the system when a TOTP device is detected as registered to a user. If one is undetected, the user can log in without an OTP.	
Update: This is an entirely new requirement.	

Requirement Number: Login-008	**Requirement Type:** Functional
Description: If the user has Multifactor authentication activated, they must enter their one-time passcode before being allowed to log in.	
Rationale: This will be used to make user's accounts more robust to hackers.	
Fit Criterion: When required, the one-time passcode must be entered correctly to allow a user to log in. If not, the user is shown an error.	
Update: This is an entirely new requirement.	

Requirement Number: Login-009	**Requirement Type:** Functional
Description: The user must complete a Captcha form before being allowed to log in.	
Rationale: This will be used to make user's accounts more robust to hackers.	
Fit Criterion: All users must complete the Captcha test successfully before being able to log in.	
Update: This is an entirely new requirement.	

Requir ement Number: Backup-001	**Requirement Type:** Functional
Description: The database backup must be stored locally on the admin’s computer.	
Rationale: This allows for an additional location for the backup to be stored instead of only on the cloud. It can potentially be used as a faster recovery method than cloud backup.	
Fit Criterion: One or more versions of the database must be backed up locally and then restored precisely as saved.	
Update: This is an entirely new requirement.	

Requirement Number: Backup-002	**Requirement Type:** Functional
Description: The database backup must be able to be stored in the cloud.	
Rationale: This allows for an additional location for the backup to be stored instead of only on local hardware. It can potentially be a more redundant recovery method than storing on local machines which are more prone to breaking.	
Fit Criterion: One or more versions of the database must be backed up on the cloud and restored precisely as saved.	
Update: This is an entirely new requirement.	

Requirement Number: Backup-003	**Requirement Type:** Functional
Description: The admin must be able to create a database backup whenever they want.	
Rationale: Any restriction to the admin creating a backup could cause data loss in the event of unfortunate timing.	
Fit Criterion: The admin can log in and create a new database backup at any time.	
Update: This is an entirely new requirement.	

Requirement Number: Backup-004	**Requirement Type:** Functional
Description: The admin must be able to fully restore the system from a local or cloud backup.	
Rationale: In the event of a disaster and the database is wiped, it can be recreated with a backup as a recovery option.	
Fit Criterion: After backups have been created, the database must be able to be recreated with a backup.	
Update: This is an entirely new requirement.	

Requirement Number: Backup-005	**Requirement Type:** Functional
Description: The admin must be able to roll back the system from a local or cloud backup.	
Rationale: In the event of data being lost or overridden, a rollback of the database would serve as the fastest recovery option.	
Fit Criterion: After backups have been created, the database must be able to be reverted to a previous state.	
Update: This is an entirely new requirement.	

Requirement Number: Backup-006	**Requirement Type:** Functional
Description: The admin must have the ability to delete a backup from a local or cloud backup.	
Rationale: Old or unnecessary backups should be able to be deleted to recoup storage.	
Fit Criterion: After a backup has been created, the admin is able to delete it.	
Update: This is an entirely new requirement.	

Requirement Number: Backup-007	**Requirement Type:** Functional
Description: The admin must be alerted (emailed) whenever there is a status update around a backup (when creating a backup, if it has been completed or failed).	
Rationale: Depending on the size of the database, creating a new backup can take a long time. If the admin stops monitoring the progress, they may miss any failures. Therefore, the admin should receive a notification ensuring they have a status record and do not miss errors.	
Fit Criterion: When creating a backup, a notification is sent to the admin if it fails or succeeds.	
Update: This is an entirely new requirement.	

Requirement Number: Database-002	**Requirement Type:** Functional
Description: The database must store information about all the different pathways offered within EEECS (excluding BIT).	
Rationale: The database must store information about the pathways so that the chatbot can correctly identify which pathway the student is on and what modules are provided for that pathway, required and optional.	
Fit Criterion: Confirmation that a table with the required columns has been created.	
Update: The description has been updated with "(excluding Business Information Technology)".	
Reasoning: This pathway was removed from the requirement as it requires information about some modules outside the EEECS school.	

Requirement Number: Database-004	**Requirement Type:** Functional
Description: The database must contain information on lecturers. This should include their name, email, and module(s) they teach.	
Rationale: The database must store information about lecturers so that the chatbot can identify lecturers that relate to modules. This allows the chatbot to provide additional support to students by referring them to the lecturer.	
Fit Criterion: Confirmation that a table with the required columns has been created.	
Update: The description updated to remove 'storing passwords. Updated to remove the mention of allowing the Lecturer to log in.	
Reasoning: Pathfinder currently does not have any features for Lecturers, therefore the need for storing a password and allowing them to log into the system is not required.	

3 Design

The general structure of the Pathfinder system has mainly remained the same, with critical improvements being made to the database, use of the system for the users, and security.

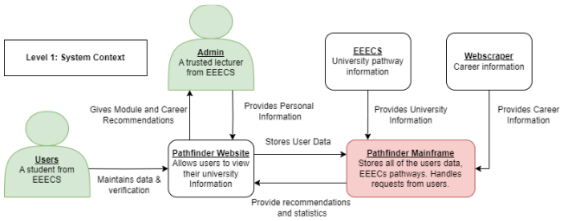


Figure 3.0.1 – Level One System Context C4

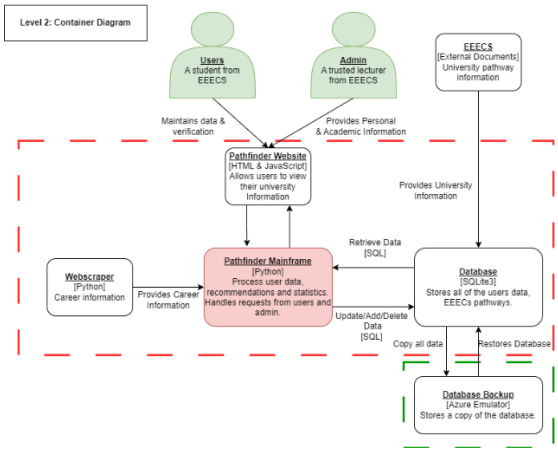


Figure 3.0.2 - Level Two Container

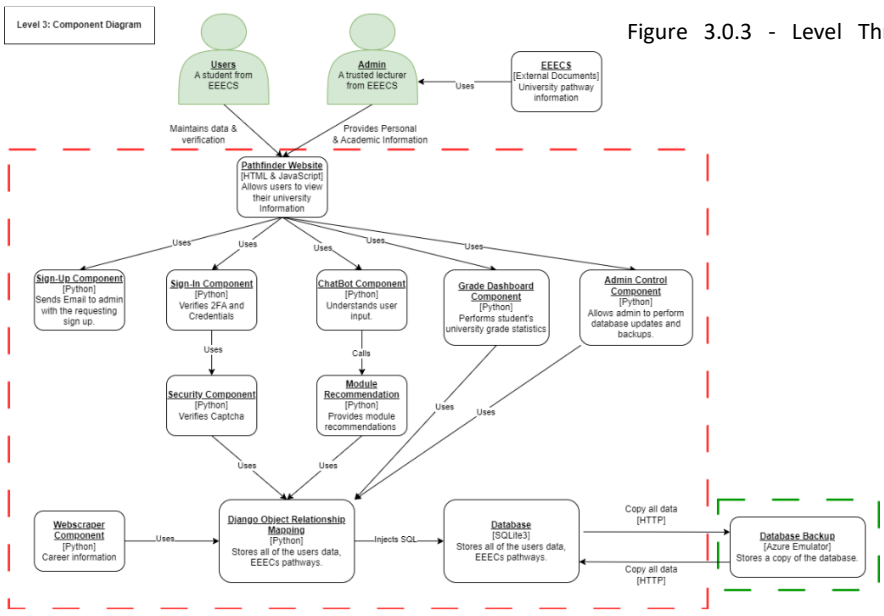
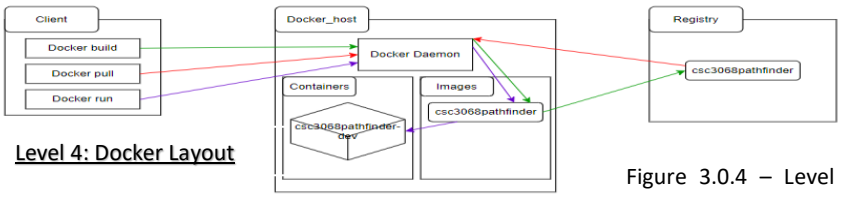


Figure 3.0.3 - Level Three Component



Level 4: Docker Layout

Figure 3.0.4 – Level Four Docker

The team selected the C4 model, which is described by (Miro, 2023), for designing the architecture diagram. The C4 model is based on abstractions that reflect how software architects and developers think about and build software. The team designed three levels of the C4 model: Context diagram (Fig 3.0.1), Containers diagram (Fig 3.0.2) and Components diagram (Fig 3.0.3). The team created a docker layout model (Fig 3.0.4) as the level 4 code diagram was too similar to the class diagram (Fig 3.1.3). Compared to other architecture design models, simple box diagrams lack the detailed semantics that

The team wanted to display in detail. Unified Modelling Language is more suitable for capturing complex system architectures, and Architecture Description Languages are typically used in more complex and critical systems.

Figure 3.0.3 was updated to incorporate the database backup feature within our system. A separate container has a Microsoft Azure emulator to show an external backup provider. The two databases communicate over HTTP protocol to copy and restore data. The design to use an emulator allowed the team to implement and test functionality without paying for Microsoft's service while also a simple change of one URL path to an Azure account if Pathfinder would go into production.

3.1 Entity Relationship Diagram, ORM and Class Diagram

The ER diagram (Fig 3.1.1) had been designed for CSC3068 with naming conventions from the document that contained all pathways from 2022 to 2023 in mind. In September, an updated version for 2023 to 2024 was released that the team used to update the stored data while keeping the same naming convention to align with the official document from the EEECS school. Each assignment record has a unique ID corresponding to the following formula: 'XXXXY' where 'XXXX' is the module code, e.g. 3058, and 'Y' is the assignment number, e.g. 2 for the second assignment. This was done to help make assignments recognisable immediately for admins when working with the database. Similarly, the ID for each record in the Module table is seven characters long to contain the Module code, e.g., CSC2058, and the IDs for the Pathway table are four characters long to match the UCAS code. An enlarged ER diagram can be seen in Appendix 3.1.1.

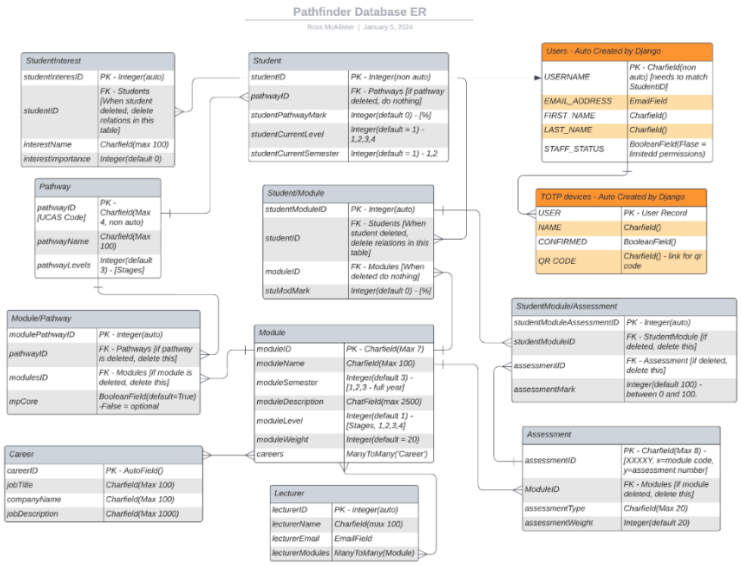


Figure 3.1.1 – Database Entity Relationship Diagram

Django automatically created the Users table highlighted in orange; this was a very useful feature as it allowed for automatic encryption, helping to secure sensitive data within our system. This table also allowed accessibility levels to be added to the website, only allowing admins to view the database and make updates. This functionality is controlled through the 'Staff_Status' Boolean field. Each record within the 'Student' table will have a corresponding record in the Users table, joined by having the Username field = the student's ID.

A new table, 'TOTP devices', was created to include storing TOTP devices. A new record can be created with a relationship with a specific User. The new record contains a URL to a QR code and a confirmed status. The user then uses this code to scan into the Google Authenticator app, where they can receive their 2FA code. After the 2FA is used successfully once, then the confirmed Boolean will update to true to signify it is working.

For the CSC3069, there were further updates to the ER diagram that the team carried out, and they are outlined below.

Module Table:

1. The default value of 3 was added to the 'moduleSemester' field to create a full-year module record easier.
2. Increased the 'moduleDescription' character limit from 250 to 2500 due to exceeding the original limit when adding actual descriptions.
3. A new field, 'careers', was added to provide many-to-many relationships with the 'Career' table. The 'Many To Many' type is provided by Django, which makes creating links, deleting links and queries through links much more accessible to develop and maintain, as described by the official documentation (Django Project, n.d).

Lecturer:

1. The team added a new field, ‘lecturerModules’, with the ‘ManyToMany’ type as a replacement for the table with the same name that could be deleted.

Student:

1. The default value of the field ‘pathwayMark’ was changed from 100 to 0. This is because when a new record is created, the student should be given a score of 0%, from which their score can be increased later.
2. A new field, ‘studentCurrentSemester’, was added to track which semester the student is currently completing. This adds an additional layer of detail that the chatbot can use.

StudentModule:

1. For the same reason as the first point in ‘Student’, the default for ‘studentModuleMark’ was changed from 100 to 0.

Career:

1. This table was added to the database to facilitate the chatbot feature, providing career recommendations to users.

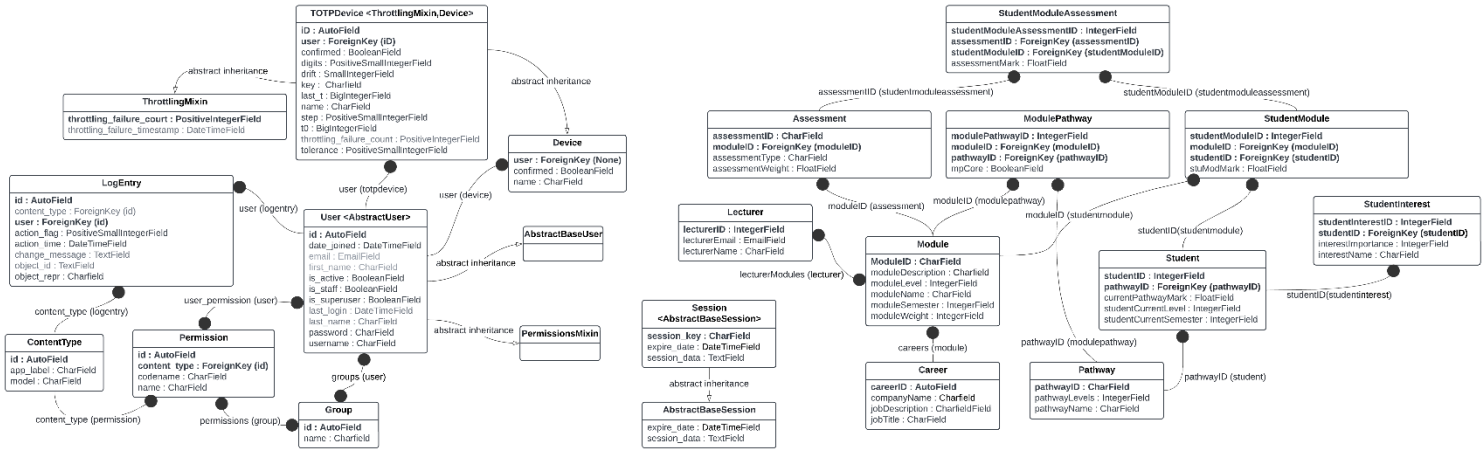


Figure 3.1.2 – Object Role Model for Database

After implementing the database, Django created the Object Role Model (Fig 3.1.2) to show the object relationships. On the left, the diagram shows the Django-generated tables (Users and TOTP Devices) and other objects that Django created necessary to allow for different access levels. The team followed the ER diagram (Fig 3.1.1) and created database tables that can be viewed on the right. An enlarged diagram can be found in Appendix 3.1.2.

The team has updated the class diagram (Fig 3.1.3) to reflect the new implementations of the web scraper and chatbot language capabilities. A new flow at the bottom of the diagram shows the web scraper created from the Scrappy library. The ‘Indeed Job Spider’ gathers job information and updates the Career table. Additionally, the Chatbot uses a new class, ‘Language Adapter,’ when a language other than English is detected. This will then reply to the user in the detected language, informing them that the chatbot only understands English. An enlarged diagram can be found in Appendix 3.1.3.

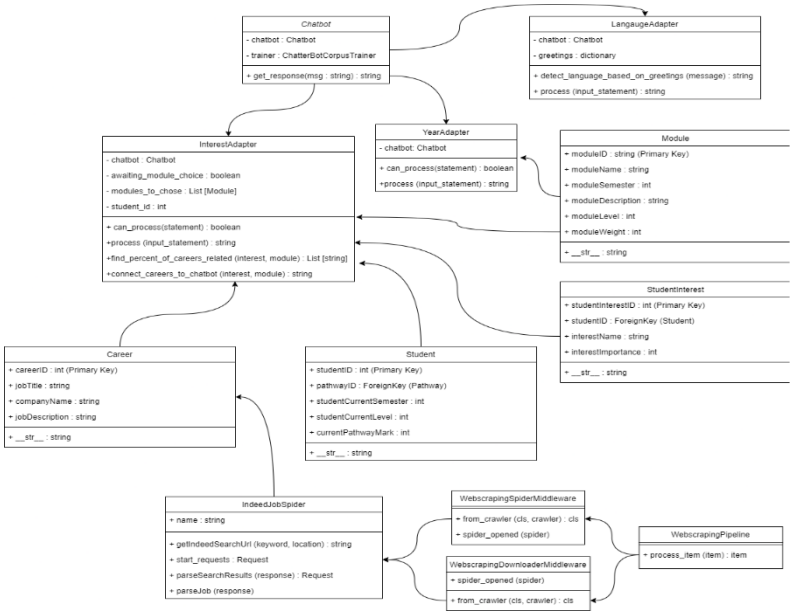


Figure 3.1.3 – Class Diagram for Chatbot and Web scraper

3.2 Use Case and Sequence Diagrams

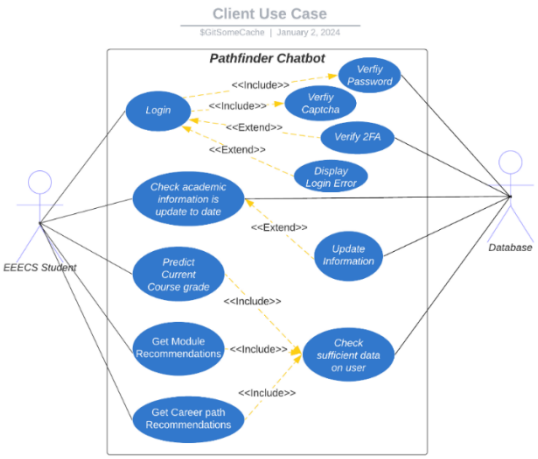


Figure 3.2.1.1 – Client Use Case Diagram

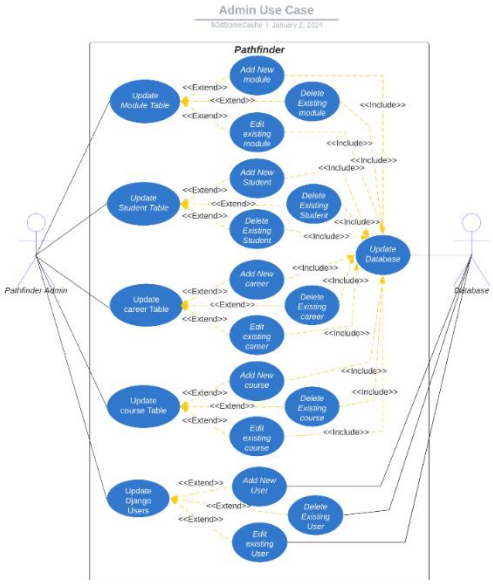


Figure 3.2.1.2 – Admin Use Case Diagram

The team created two use case diagrams for Pathfinder to outline the user's possible interactions with the system for both the clients (students) and the admin. Enlarged diagrams are in Appendix 3.2.1 with the use case descriptions. Improvements for CSC3069 include updating the Client's diagram (Fig 3.2.1.1) to show the possible interaction with Two-factor/Multifactor Authentication that the students can implement as additional security. The login functionality now also requires the students to pass the Captcha form.

Additionally, the functionality of updating the Django User's table, which was previously missing, was implemented for the admin (Fig 3.2.1.2). This shows the process of allowing the admin to edit existing users, through which they can reset their TOTP devices and passwords.

The team created a sequence diagram for each use case, showing the user interactions over time. Below are the updates made to the Sequence diagrams. All diagrams that were kept the same can be found in Appendix 3.2.2.

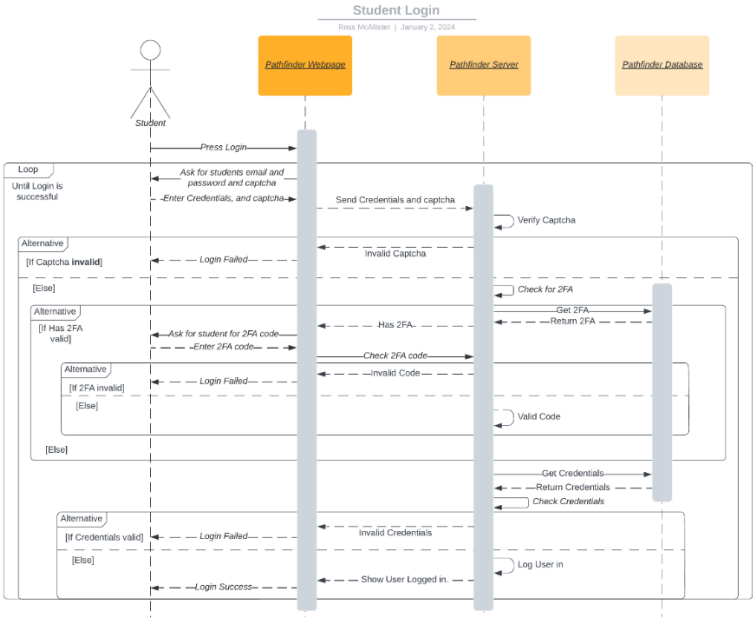


Figure 3.2.2.1 - Student Login Sequence Diagram

The login diagram (Fig 3.2.2.1) became significantly more complex to add to the various Captcha and two-factor authentication checks. The system now checks for valid a captcha form, if the user has 2FA enabled, and if their input code is correct.

The team made a design choice pertaining to the admin, which involved giving them the option to use either the Command Line or the Website UI to modify the database. This decision was made because experienced administrators could take advantage of the fast-paced nature of the Command Line, while the UI provides a clear visual representation of the database and is more user-friendly for the inexperienced. Below is an example of a use case with two sequence diagrams: the left diagram is UI (Fig 3.2.2.2), and the right is the Command Line (Fig 3.2.2.3).

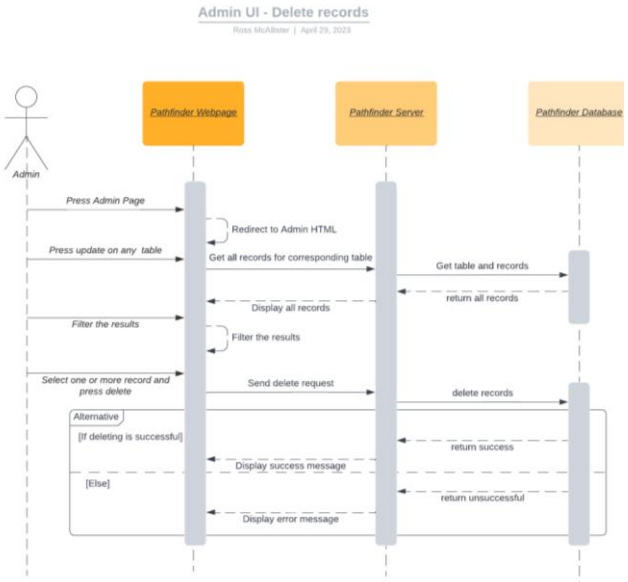


Figure 3.2.2.2 – Admin delete records sequence diagram

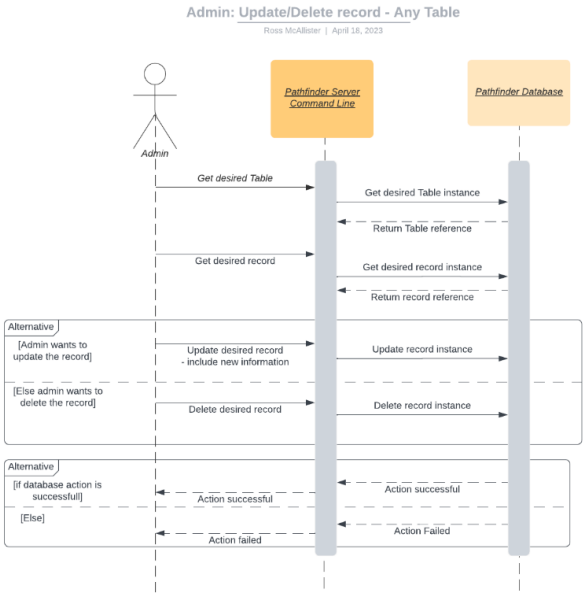


Figure 3.2.2.3 – Admin update/delete records sequence diagram (Command Line)

3.3 UI Design

Wireframes were chosen to show the UI design as they give the web design a clear vision of how the product is meant to look, making development a smoother experience as the front-end developer has a clear target to hit.

Upon visiting the website, users are greeted by the chatbot on the homepage, which also provides instructions on how to use it. Users can initiate a conversation with the chatbot by providing information to select a module or access the sidebar. Within the sidebar, the user has the following menu options:

- To find more information about the modules available in EEECS, users can visit the module information page and use the search/filter options provided.
- There is an external link that will take the user to the official Queen's University Belfast Site.
- If they wish to sign up to the system, they can navigate to 'Sign Up' and fill in the required options to make a request. The request will then be sent to the admin for approval.
- Alternatively, users with an existing account can log into the system.

After logging in, users will see either the admin panel (if they are an admin) or the grade dashboard (for students). The admin panel will allow them to view and modify the database. The grade dashboard is helpful for students to track their progress in a single place, including current progress, module grades, individual assessments, and stats. This information is valuable to identify areas of focus and can motivate students to work harder. The settings page will display the users' personal information stored in the system. Additionally, accessibility options, such as changing the website's theme and font size, will be available to all users, even those who are not logged in.

From the submission of CSC3068, the user interface could remain largely the same in terms of design. One area that had to be adapted was when the team was ensuring compatibility with smaller mobile devices. Comparing the diagrams (previous diagram Figure 3.3.1.1, updated diagram Figure 3.3.1.2), it can be seen that the grade bar chart had to be removed from the mobile view as there wasn't enough room to display this element accurately. The statistic boxes have also changed from horizontally to vertically aligned to ensure an appealing fit.

Figure 3.3.1.1 – Old grade dashboard wireframe

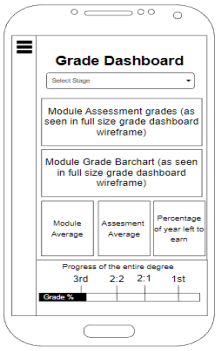
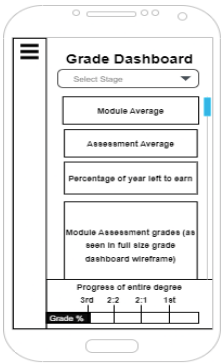
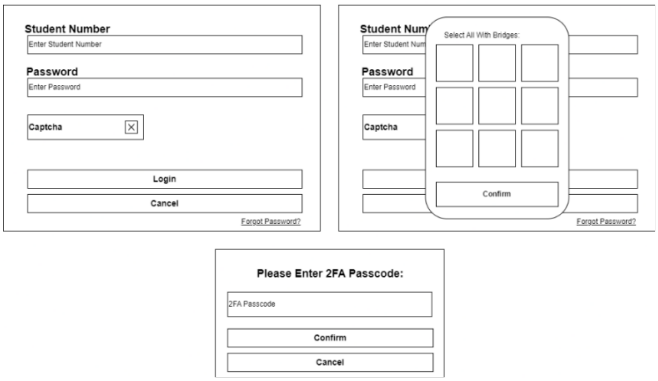


Figure 3.3.1.2 – New grade dashboard wireframe



Due to the enhanced security of adding Captcha and 2FA, the login and settings diagrams (Fig 3.3.1.3) also needed to be updated to allow users to interact with these elements. The new login pop-up required a new captcha field that will bring up the additional test. Once this test is passed, the Captcha box will be filled with a tick where the user can progress to login. If the user has 2FA attached to their account, they will get the third pop-up where they must enter their code; from here, they can log in.

Figure 3.3.1.4 – New login wireframe



The new settings page design (Fig 3.2.2.5) now has the required options to allow users to update the TOTP devices to which they would receive their 2FA passcodes. This is done by scanning the QR code.

All remaining UI designs can be found in Appendix 3.3.1.

Student Settings Page

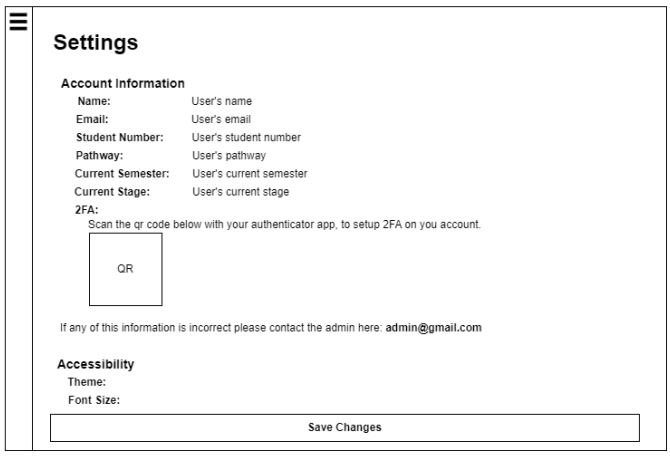


Figure 3.3.1.5 – New settings page wireframe

4 Implementation

The implementation section details system improvements since the CSC3068 submission, including enhancements to security and usability through additional features to the Login and Chatbot. For some context the last submission had a login system that allowed for a user to enter a username and password, then if these matched what was contained in the database then the user would be logged in. The chatbot had basic sentiment analysis by checking for the words “like” and “dislike” then recommending a module based on that.

Key modifications include the addition of Captcha and 2FA to ensure the safety of users data, along with quality of life improvements to the chatbot including (but not limited to) adapted language handling, allowing for responses in the native language of international students that was lacking in the last system causing international students to feel as if they were being left out, going against the University diversity values. This section also discusses the implementation of web scraping and its influence on project requirements. It also covers the introduction of backups as a new requirement, imposing time constraints on implementation.

The iterative changes demanded flexibility, requiring continuous reassessment and adjustments to align with evolving specifications. The development team adopted a strategic, collaborative approach, employing agile methodologies for incremental adjustments. Regular meetings and communication channels facilitated real-time information exchange, while proactive system assessments identified potential impacts from changing specifications. Detailed documentation ensured transparency and traceability, highlighting the team's adaptability in responding to challenges and maintaining system integrity.

4.1 Language and Library Choices

The language selected was Python (3.7.9), this was selected as the primary programming language due to its readability, versatility, and extensive ecosystem, aligning with the requirements for web application development. Another reason why this was chosen was due to the team’s experience in Python allowing the team to make changes quickly.

Feature	Framework/Library	Reasoning
Development Environment	Django Framework (3.2)	Django, a high-level web framework, was chosen to structure and streamline web development. Its features, including built-in security measures and an ORM system, influenced the overall project architecture.
Chatbot Integration	ChatterBot (1.0.8) and ChatterBot Corpus (1.2.0)	ChatterBot and its Corpus extension were instrumental in seamlessly integrating a conversational agent into the web application. The language learning capabilities provided by ChatterBot Corpus contributed to enhanced chatbot interactions.
Natural Language Processing	spaCy (3.5.0)	spaCy was employed for advanced natural language processing tasks within the chatbot. Its capabilities improved language understanding and user interactions.
Database Backup	Django Database Backup (4.0.2)	Django Database Backup was chosen to ensure data integrity and recovery capabilities. It played a crucial role in addressing requirements related to database management and backup strategies.
Cron Job Scheduling	Django Crontab (0.7.1)	Django Crontab facilitated the automation of periodic tasks, such as database backups, aligning with the need for a scheduled task execution system.
Azure Integration	Azure Core (1.29.1) and Azure Storage Blob (12.17.0)	Azure Core and Azure Storage Blob were selected for their compatibility with Microsoft Azure services. These libraries supported storage operations and other interactions with Azure components.
Azure Integration	Azure Core (1.29.1) and Azure Storage Blob (12.17.0)	Azure Core and Azure Storage Blob were selected for their compatibility with Microsoft Azure services. These libraries supported storage operations and other interactions with Azure components.
Language Detection	Langdetect (1.0.9)	langdetect contributed to language detection functionality, addressing requirements related to multilingual support and text processing.
CAPTCHA Integration	Django reCAPTCHA (3.0.0)	Django reCAPTCHA integrated Google's reCAPTCHA service into web forms, enhancing security against automated abuse and potential threats.
Two-Factor Authentication (2FA)	Django OTP (1.2.3) and qrcode (7.4.2)	Django OTP and qrcode were pivotal in implementing Two-Factor Authentication. They provided tools for OTP creation, validation, and QR code generation.
English Language Model for spaCy	en-core-web-sm (3.5.0)	en-core-web-sm, a spaCy model, improved English language processing capabilities within the chatbot, aligning with the application's language-centric features.
Web Scraping	Scrapy (2.9.0), urllib3 (2.0.4), scrapeops-scrapy-proxy-sdk (1.0)	Scrapy, urllib3, and scrapeops-scrapy-proxy-sdk were employed for web scraping functionalities, supporting data extraction from websites for components like the chatbots knowledge about careers.

4.2 Login

The login system developed during CSC3068 has been significantly improved within this submission. The requirements Login-001 (authentication), and Login-002 (authorisation) have been met since the first submission but the remainder of the login requirements (Login-003 to Login-006) have now been implemented along with the new login requirements that were added since the last submission (Login-007 to Login-009). A high-level overview of the login process can be seen in figure 4.2.1, this overview does not contain POST requests or CSRF tokens.

The first requirement that will be discussed is Login-003, which is centered around encryption. The default algorithm for encryption within Django is pbkdf2_sha256 (Password-Based Key Derivation Function 2 with SHA-256) (Django Project, n.d.a), recommended by NSIT for its computational intensity and resistance to brute-force attacks (Meltem et al., 2010). To enhance security, the encryption process involves a substantial number of iterations, set at 260,000. Additionally, a unique and randomly generated salt value is employed for each user's password. This salt is combined with the user's password and passed through the PBKDF2-HMAC-SHA-256 algorithm. The resulting hash represents the final encrypted form of the password. The combination of a strong algorithm, high iteration count, and unique salt significantly bolsters the security of stored passwords, mitigating the risk of common cryptographic attacks. This multi-layered approach ensures that even if the hash were to be exposed, it would be computationally intensive to reverse-engineer the original password (Meltem et al., 2010).

Login-004 (remember me functionality) is quite simple so this will be brief, after a user's login credentials are verified, the system checks whether the "remember me" checkbox was selected, if so the session cookie for this session is set to expire in two weeks.

This stores the session within the database in the session table on the server allowing the user to return to the session within the next two weeks. This two-week value is an arbitrary number, the reason is stored for a limited amount of time is to mitigate security risks, as it reduces the window of opportunity for unauthorised access in the event of session hijacking or compromised user credentials.

To fulfill the requirement Login-005 and the new requirements Login-007 and Login-008, a method enabling users to add a TOTP (Time-based One-Time Password) Device was implemented. This involved the creation of a TOTP Device table within the database, dedicated to storing registered TOTP devices associated with a user's account. After a user log in, they can navigate to the settings page, where they encounter a QR code. By scanning this QR code with either the Microsoft or Google authentication app, a corresponding record is generated in the TOTP device table, indicating successful registration.

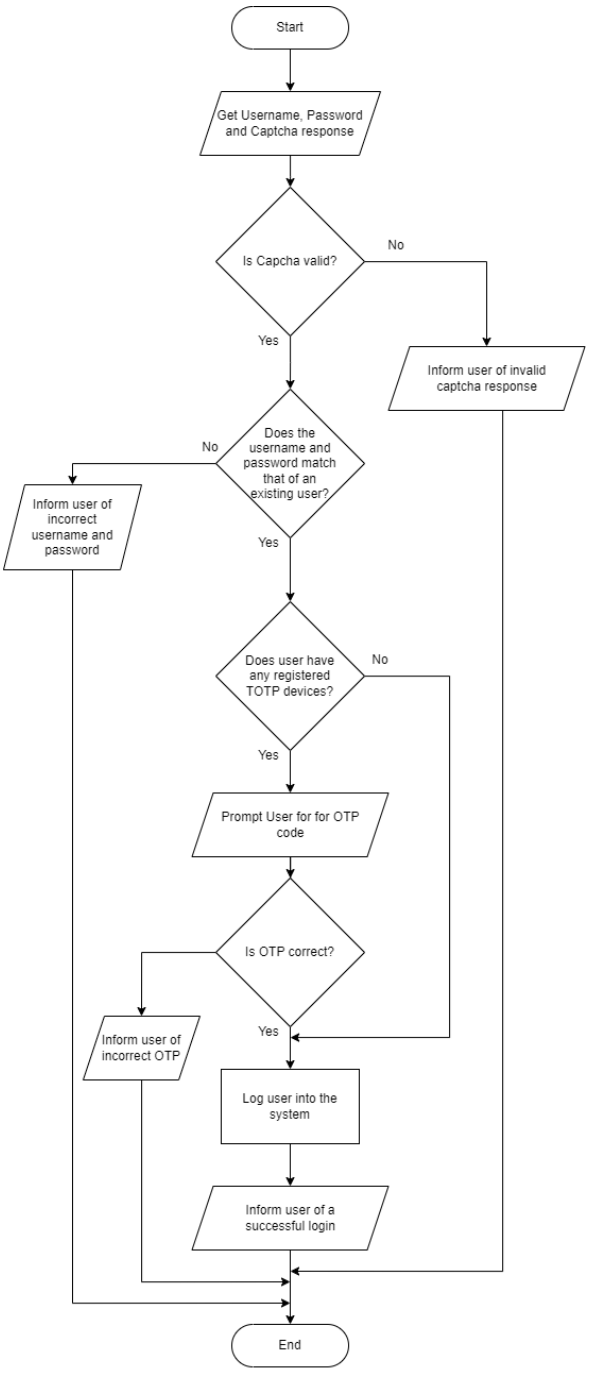


Figure 4.2.1 – High-level overview of login (excluding POST and CSRF).

The process begins by generating a secret key utilising the django_otp.plugins.otp_totp.models library, as illustrated below:

```
def createSecretKey(request):
    return TOTPDevice.objects.create(user=request.user, confirmed=True).config_url
```

Subsequently, a QR code is created for this secret key using the qrcode library (see code snippet in Appendix 4.2.1) and sent via HTTPResponse to the front end, where it is displayed on the settings page (refer to code snippet in Appendix 4.2.2).

Upon successful username and password validation, the code checks if the user has 2FA enabled by querying the hasTwoFactorEnabled function (see code snippet in Appendix 4.2.3). If 2FA is active, the system proceeds to verify the TOTP code provided by the user using the verifyTotpCode function (see code snippet in Appendix 4.2.4).

To determine if a user has 2FA enabled, the hasTwoFactorEnabled function authenticates the user and checks for associated TOTP devices. If at least one TOTP device is confirmed, indicating 2FA activation, a JSON response signals that 2FA is enabled; otherwise, the response indicates its absence. The TOTP code verification is carried out by verifyTotpCode, which iterates through the user's TOTP devices, validating the provided code. If a match is found, the user is successfully authenticated; otherwise, an error response is returned.

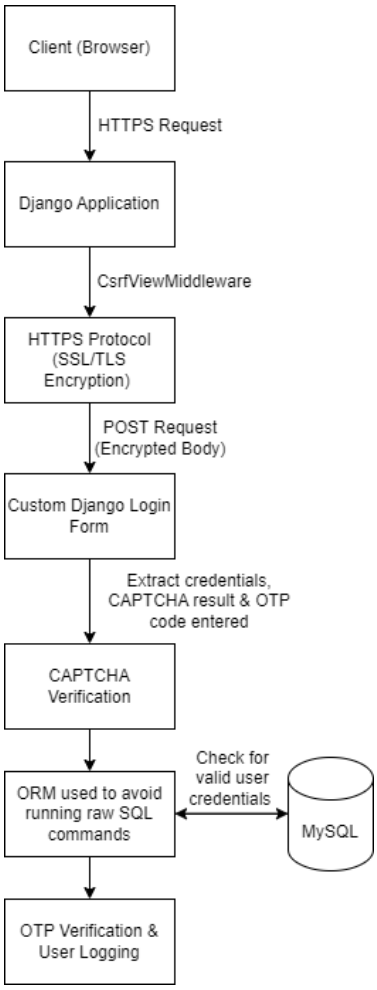


Figure 4.2.2 – High-level sequence diagram for flow of login.

Login-006 achieves protection against XSS and SQLi attacks by utilising two of Django's systems (Duisebekova et al., 2021). The first is CSRF (Cross-Site Request Forgery) tokens (Django Project, n.d.b), and the second involves the use of Django's ORM (Object-Relational Mapping) (Django Project, n.d.c). The decision to incorporate these features was informed by the research paper "Django as Secure Web-Framework in Practice" (Duisebekova et al., 2021), which outlines how the use of CSRF tokens and Django's ORM can effectively prevent XSS and SQLi attacks.

The implementation of CSRF begins with the generation of a CSRF cookie by the CsrfViewMiddleware. This cookie, containing a random secret value, is sent with each response through the use of the django.middleware.csrf.get_token() function, enhanced with a random mask to counteract BREACH attacks. Simultaneously, all outgoing POST forms include a hidden 'csrfmiddlewaretoken' field, its value comprising the secret and a mask. This mask is regenerated with every call to get_token(), ensuring the dynamic alteration of the form field value in each response, a process facilitated by a template tag. A very high overview of the security features within the login process can be found in figure 4.2.2

During request validation, a strict check is applied to ensure the presence and accuracy of the CSRF cookie and 'csrfmiddlewaretoken' field. Any discrepancies lead to a 403 error. The validation involves comparing only the secret, not the entire token, allowing for dynamic tokens while maintaining a consistent secret. For HTTPS requests, an additional security layer is implemented using strict referrer checking by CsrfViewMiddleware. This ensures requests originate from the exact domain. This helps guard against potential man-in-the-middle attacks, particularly in HTTPS scenarios where HTTP Set-Cookie headers may be accepted.

Django's ORM is instrumental in preventing SQL Injection attacks by parameterising queries and escaping user inputs. All database interactions, including queries and updates, are performed through the ORM, ensuring that user inputs are sanitised and properly handled. This abstraction layer adds an additional security barrier, significantly reducing the risk of malicious SQL Injection attempts (Duisebekova et al., 2021).

A Captcha mechanism, implemented in compliance with requirement Login-009, has been added to the system to prevent automated bots from exploiting vulnerabilities. Using Django's custom forms, Captchas are displayed on the login page to ensure user requests originate from humans, not automated scripts. The system utilises the django-recaptcha library and Google reCAPTCHA API, allowing administrators to configure Captcha domains and difficulty levels. Verification through the custom form enhances security against automated attacks and confirms the accuracy of entered usernames and passwords, as detailed in the provided code snippet in Appendix 4.2.5.

The POST method is also being used to help protect users while logging in. This practice ensures that critical data, such as login credentials, is not exposed in URLs and is sent in the request body instead. The use of POST methods aligns with security best practices, mitigating the risk of information exposure through browser history or server logs (Duisebekova et al., 2021).

In considering future enhancements, the reliance on default settings across various built-in Django features, including encryption, and CSRF raises potential concerns. While the documentation assures that the defaults are suitable for "most uses" (Django Project, n.d.a) and research papers support Django's default security features (Duisebekova et al., 2021), there is room for further exploration to enhance the system's security. Exploring alternatives beyond the default configurations may provide an opportunity to fortify the system against potential vulnerabilities.

One potential improvement would be changing the default hashing to Argon2[6]. Argon2's flexibility allows adjustments to time and memory costs, accommodating evolving hardware capabilities and maintaining a high level of security. By implementing Argon2, the system can further fortify its defenses against unauthorised access attempts. To integrate Argon2 into the system, the argon2-cffi library can be used (Django Project, n.d.a), which provides a straightforward implementation of the Argon2 algorithm.

4.3 Backup

All aspects of the backup are new from the CSC3068 submission as the requirements for the backup were added after that submission. The first requirement added was Backup-001, accomplished by using the Django-backup library (Django Project, n.d.d) that generated a .dump file from the MySQL database stored on the server. This is achieved by running "python manage.py dbbackup" into the terminal. Additionally, the default location for storing backups within settings.py was modified to ensure that the local backup is stored within the correct directory (Django Project, n.d.d).

As having someone run a terminal command is not ideal to meet requirement Backup-003, along with the fact that it would not cause any email alerts therefore not implementing Backup-007, it was seen as a better decision to create a function run this process instead which can be seen below:

```
def createBackupFile():
    try:
        call_command('dbbackup', exclude_tables='django_session') # Triggers the database
backup using Django's 'dbbackup' management command
        backupStatusEmail("Backup file created successfully")
        return True
    except Exception as e:
        backupStatusEmail("Backup file creation failed", True, e)
        return False
```

You will notice that to run the dbbackup command the function “call_command” has been used, this is part of the django.core.management library and allows for django specific terminal commands to be executed within code.

You’ll also notice one of the arguments in call_command is `exclude_tables='django_session'` this is due to a bug that was found within development, after testing the code by creating and creating multiple backups repeatedly, it was observed that the time in creating these backups started to take longer and longer. After some investigation it was revealed that the length of the .dump files were slowly increasing, once analysing the .dump files it was noticed that the session table was duplicating the contents of the backups as, apparently, the Django session table will store the SQL commands that are being carried out through the ORM. As the session table does not store any permanent data that would need to be recovered in the event of recovery from a backup it was best to exclude it from the backup files.

Another notable aspect of the above code is the use of the try and except block to identify if a backup was successfully created which allows an email to be sent using the backupStatusEmail function, that allows the system to meet the requirement Backup-007. A high-level email alert architecture diagram can be seen in figure 4.3.1.

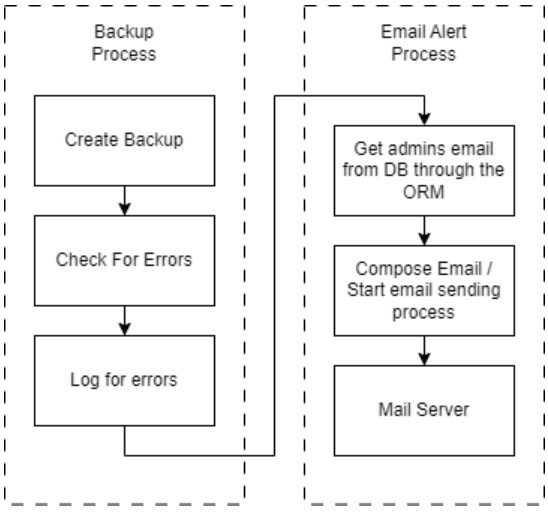


Figure 4.3.1 – High-level email alert architecture

To fulfill the requirement of backing up the database to the cloud (Backup-002), the implementation leverages the Azurite emulator—a powerful tool that emulates Azure Storage services. Given the financial constraints of a university project, Azurite provides a cost-effective solution for testing and developing cloud-based applications without the need for a paid Azure subscription.

Azurite is an open-source Azure Storage Emulator that mimics the behavior of Azure Storage services, including Blob, Queue, and Table storage (pauljewellmsft et al., 2023). Developed by the Azure Storage team, Azurite is written in Node.js and can be run locally, allowing developers to replicate the Azure Storage environment for testing purposes (Dustdar and Furutanpey, 2022). This allows the team to run this within a docker container (pauljewellmsft et al., 2023), this container will serve as our “cloud storage” which ensures compatibility with Azure services while remaining within the budget constraints of a university project. This approach not only facilitates the development process but also prepares the system for seamless integration with Azure Storage in a production environment (Dustdar and Furutanpey, 2022).

Blob storage is the chosen storage solution for backing up the database due to its scalability, flexibility, and efficiency in handling large binary data (Waly, 2017) (Mazumdar et al., 2016) (akashdubey-ms et al., 2023), such as database dumps.

Blob storage in Azure is ideal for storing unstructured data (Mazumdar et al., 2016) (akashdubey-ms et al., 2023), making it suitable for saving backup files in the form of .dump files.

To simulate an actual Azure Storage instance, Azurite is run in a separate Docker container. This containerization approach allows for easy management of dependencies and ensures that the Azurite environment is isolated from the rest of the application components. The following Docker command is used to run Azurite:

```
docker run -d -p 10000:10000 -p 10001:10001 mcr.microsoft.com/azure-storage/azurite
```

This runs the container on the respective ports which allows us in the code to connect to this Azurite instance using a connection string. The connection string can be changed to one for an actual Azure storage account, this is the only change needed in the code to ensure that an actual Azure storage account will work with the project.

To upload the file to the storage account a connection to the storage account blob client is made using the connection string (see Appendix 4.3.1). Then using this client, the local version of this backup file can be read and uploaded to the Blob, as seen below (note: destinationBlobName is a parameter for the function that is responsible for uploading the blob, it simple represents the name of the blob that is being created, which is the same as the filename being uploaded):

```
with open(filePath, "rb") as file:
        blobClient =
containerClient.get_blob_client(destinationBlobName)
blobClient.upload_blob(file)  # Upload the file to the
blob storage
```

For requirements Backup-004, Backup-005 and Backup-006 (full restore, rollback and deletion from both cloud and local backups), these requirements were implemented by first getting the local backup version to work. To "restore" a database, it means taking the data from a backup and adding it into the current database, meaning that data added since that backup was created will remain in the database after the restore is complete, luckily Django has a command for this.

Similarly to creating a backup, the goal is to run this command, however unlike dbbackup, dbrestore does not allow you to specify a file using call_command(). To get around this a function was created that will run the "python manage.py dbrestore" command in the terminal and specify the file path of the backup that is to be used (see Appendix 4.3.2). Making the file path a parameter allows the code to be reused for both local and cloud backups, this reusability is a common theme throughout the backup process. Another note within this function is a VACUUM command is executed on the database, this reclaims space from the inserts, updates and deleted objects for future use in the database, in other words "cleaning" the database file.

Implementing rollback functionality (bringing the database back to the exact same state it was in when the backup was created), required a bit of work around. As Django does not offer any libraries that allow for rollback functionality, it was decided the best course of action was to first delete everything that is on the database, then use the restore function described above to add in the data from the backup back into the database. This causes the same effect as if there was a rollback command built in, to delete all the data from the database the "flush" command was run using call_command().

As seen in the flowchart to the right (figure 4.3.2), the only difference between when a user selects a cloud or local backup is that for cloud, a temp file is created locally that allows the same functions to be used for restoring and rollback, then the temp file is deleted at the end.

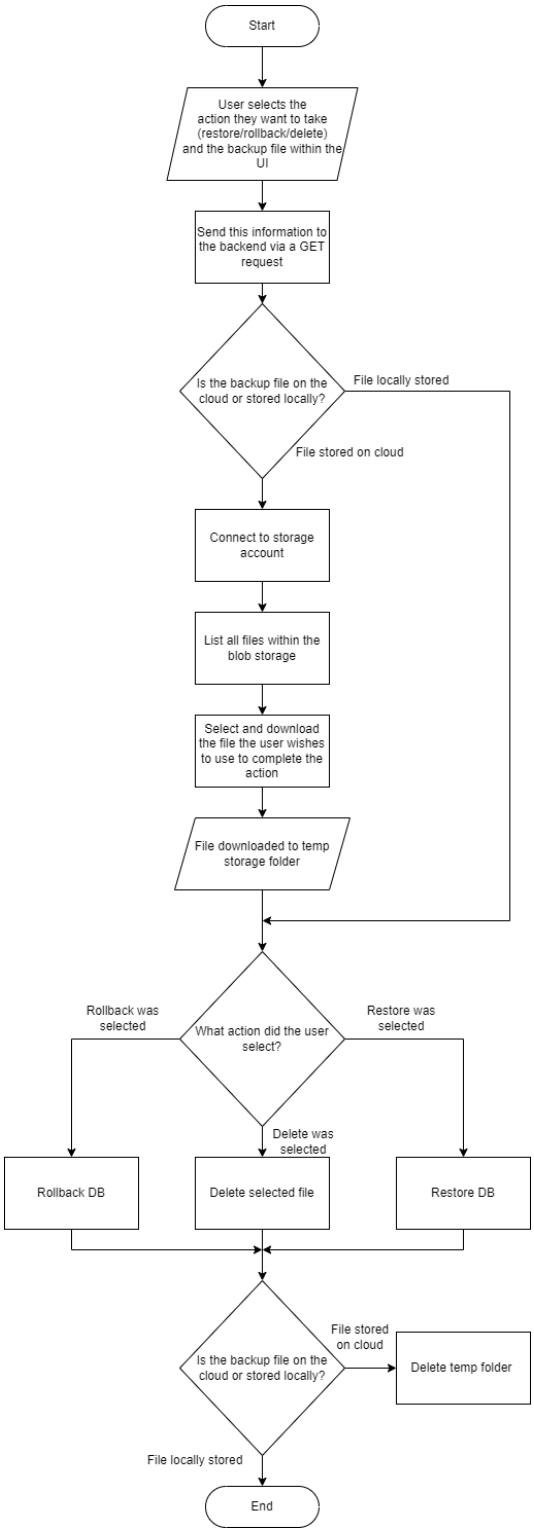


Figure 4.3.2 – High level overview of backup

Another thing to note is that the “delete selected file” is different between cloud and local, this is simply that when it is a cloud file selected, it gets deleted from the blob container.

The last feature surrounding the creation of a backup is the use of django_crontab. This library allows the system to utilise Cron jobs within Unix (note: the system is using a docker image that is based on Debain) which allows for the creation of a backup to be scheduled for every two weeks on Sunday at 2am GMT, which was deemed to be a suitable downtime for the system as it is designed for student. The scheduled job involves creation of a local backup, then the uploading of this file to the cloud to ensure there are two copies of the backup file.

The first and most obvious improvement to the system is the replacement of the aforementioned Azurite emulator, and replacing this with an Azure storage account that is hosted offsite of the primary server. This also outlines another improvement that should be made, this being the lack of a formalised 3-2-1 backup strategy. The 3-2-1 strategy, consisting of three total copies of data, two of which are local but on different devices, and one copy stored offsite, is a widely recognised best practice for data backup and recovery (Ruggiero and Heckathorn, 2013). This can be done by introducing an additional offsite backup location, such as a separate cloud provider or dedicated backup server. The inclusion of offsite backups provides an extra layer of protection against catastrophic events that may impact the primary server.

A summary table of the benefits of a 3-2-1 backup can be seen below:

Backup Strategy Comparison Table		
Backup Element	**Current Strategy**	**3-2-1 Strategy**
Number of Backup Copies	2	3
Local Storage Backup Copies	1	2
Offsite Storage Backup Copies	1	1
Backup Frequency	Every two weeks (Scheduled on Sunday at 2am GMT)	Continual (Regularly scheduled)
Backup Verification Mechanism	Email alerts for status	Checksum or hash functions
Error Handling Mechanism	Generic pass/fail error messaged and logging	Detailed error messages and logging
Storage Impact	Deletes oldest backup when new one is created (keeps 10 local backups and 20 on the cloud).	Rolling backups, minimising storage allowing for more to be help.

To guarantee the reliability of backups, a verification mechanism can be integrated. This involves regularly checking the integrity of backup files, ensuring they are not corrupted and can be successfully restored. Implementing checksums or hash functions can aid in verifying the integrity of backup files, enhancing the overall reliability of the backup process.

While the current implementation incorporates email alerts for backup status, future improvements could focus on refining error handling mechanisms. This includes providing more detailed error messages and implementing a comprehensive logging system to track backup-related events. Enhanced error handling ensures timely identification and resolution of issues, contributing to the overall robustness of the backup system.

Rolling backups, also referred to as incremental or differential backups, selectively capture data changes since the last backup, effectively reducing storage requirements and backup operation duration. The process begins with an initial full backup, followed by incremental backups that document changes from the prior backup. In the event of data loss, the restoration involves applying the full backup and sequentially adding incremental backups to recover the dataset (Jan Bergstra and Mark Burgess, 2008). This methodology offers advantages such as diminished storage overhead, expedited backup procedures, and optimised resource usage. Particularly beneficial for sizeable datasets and environments necessitating frequent backups, rolling backups provide a nuanced recovery approach, enabling users to restore data to specific timestamps, thus enhancing adaptability and minimising potential data loss (Jan Bergstra and Mark Burgess, 2008).

This stands in contrast to the current implementation, which creates a full backup of the system, significantly impacting storage. To address these storage constraints, the current strategy involves deleting the oldest backup when a new one is created, both for cloud (see code snippet in Appendix 4.3.3) and local (see code snippet in Appendix 4.3.4) backups, maintaining at least 20 and 10 backups, respectively. As mentioned above the rolling backup would be an improvement as it would allow for more backups to be stored while minimising impact on storage space.

4.4 Web Scraping

Web scraping was implemented to meet the requirements Web-Scraping-001 and Web-Scraping-002, enabling the extraction of job-related information from the Indeed website. The Scrapy framework was chosen through research into various academic articles (Thomas and Mathur, 2019) (Mitchell, 2018) it was identified as a powerful and flexible tool for web scraping (Thomas and Mathur, 2019) that is often used as the standard for data analysis in python (Mitchell, 2018). Through implementation of the spider there were some limitations that go against the original goal of the requirements.

However, before the limitations of this web scraping process is discussed let's explain how it works. Indeed, is a popular website for job postings, therefore, to gather career information it was decided that indeed would be a suitable website to scrap this information from. Therefore, a spider class needs to be created with suitable methods.

The first thing that needs to be done when creating a spider is to implement a function that will create the request URL with the needed parameters. Indeed's URL is as follows 'https://www.indeed.com/jobs?q=software engineer&l=Belfast&start=0&filter=0', as you may be able to see; **q** stands for the search query, in this case, q=software engineer. **l** stands for the location that is to be searched for jobs, in this case, l=Belfast. **start** stands for the starting point for the pagination, this will be set to 0 to ensure that all pages will be searched. Using this knowledge of the URL, the following method was created:

```
    def getIndeedSearchUrl(self, keyword, location, offset=0):
        parameters = {"q": keyword, "l": location, "filter": 0, "start": offset}
        return "https://www.indeed.com/jobs?" + urlencode(parameters)
```

The next step in the process is to extract the job listings from the information returned from the Indeed endpoint. This was done by running the spider to retrieve all the information, then analysing this manually to identify where the job information is stored. It turns out job information is stored within a hidden JSON file, under <script id="mosaic-data" type="text/javascript"> tag, under window.mosaic.providerData["mosaic-provider-jobcards"]. This means that the information within this tag needs to be extracted which can be done using the following regex:

```
scriptTag = re.findall(r'window.mosaic.providerData\["mosaic-provider-
jobcards"\]=(\{.+?\});', response.text)
```

If the offset is 0 (indicating the initial page), the function calculates the total number of job results, the function then generates requests for pagination, iterating through subsequent pages in increments of 10. Then requests will be generated for each one of these job listings so that each job page can be scraped for the specific job information (see Appendix 4.4.1).

Now that the response to the requests for each of the job listings is here this data can be extracted. This follows a similar process as before, the response is analysed to find the job information, said information is stored within a JSON file in a script tag, the information is extracted from the script tag using regex then converted to a JSON blob to better access the information within the tag:

```
# Extract JSON data using regex from the page source

scriptTag = re.findall(r"_initialData=(\{.+?\});", response.text)

if scriptTag is not None:
    # Parse JSON data
    jsonBlob = json.loads(scriptTag[0])
    job = jsonBlob["jobInfoWrapperModel"]["jobInfoModel"]
```

The key difference between scraping the search page and scraping the job posting page is what is being done with the data afterwards. In the case of the search page, it was to conduct further scraping, in the case of the job posting the information is being added to the database).

As there are many responses to add to the database, it was decided the best way to do this was to utilise pythons threading library to run the function response for adding the data to the careers table concurrently. This is done by creating a thread and assigning the addCareers function as the target to be executed by the thread. The function takes job_title, company_name, and job_description as arguments:

```
t = threading.Thread(target=addCareers, args=(job_title, company_name, job_description))
```

By offloading time-consuming tasks like addCareers to a separate thread, the main thread can remain responsive and not be blocked during the execution of potentially time-consuming operations. This also allows for multiple jobs to be processed and analysed at the same time improving speed.

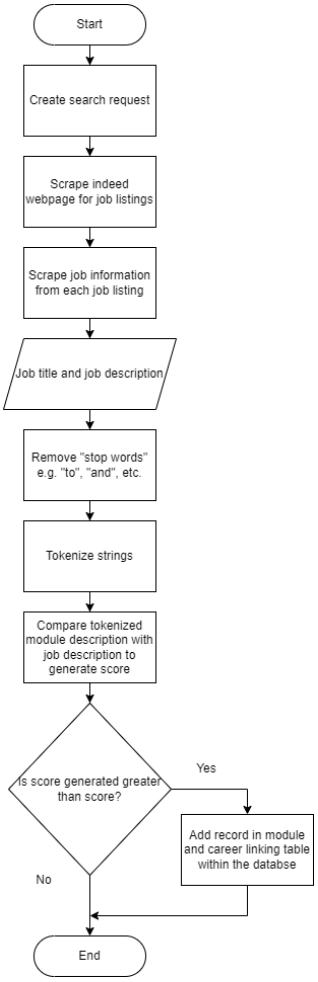


Figure 4.4.1 – High level overview of web scraping.

Once the career information is added to the database it is linked to a module that is like it. This is done to help the chatbot identify what careers relate to what modules so when a student asks about information for a career or a module the chatbot has careers and modules to associate with each other. This is done through the calculating the similarities between the job posting see figure 4.4.1 for a high-level overview of the web scraping processes including calculating the similarities.

The function defines a set of stop words (stop_words) that are common words often excluded from analysis in natural language processing tasks. These words are typically non-informative and are removed to focus on more meaningful content.

The textual information from both the module and the career is tokenised and cleaned. Tokenisation involves breaking text into individual words or tokens. The tokenize_string function is assumed to perform this task. After tokenisation, the stop words are removed from the token sets.

The tokens from the module name and description are combined into a single set (module_tokens). This set represents all the unique words present in the module's name and description, excluding stop words.

The similarity score is calculated based on the number of common tokens (words) between the module and the job description. The more common tokens, the higher the similarity score. The score is the count of tokens that are both in the module and the job description.

If this score is above the threshold of 5, then a module and a career are said to be linked, therefore, a link between the module and the career is made in the form of a record in the career and module linking table.

As mentioned in the beginning of this section there are limitations that affected successfully completing WebScraping-002 requirement which revolved around scheduled scraping. This limitation was caused by Indeed's robust bot protections. To circumvent these protections, the project relies on a third-party ScrapeOps API. While effective, this solution presents a financial challenge as the ScrapeOps API offers only a limited number of free API calls before initiating charges. Given the financial constraints of a university project, this limitation affects the scalability and sustainability of the web scraping process.

In response to these limitations, an investigation was conducted into creating our own antibot protection measures to reduce (possibly eliminate) the reliance on the ScrapeOps API. This involved following the strategies laid out in “Web Scraping with Python“(Mitchell, 2018) which involved creating fake request headers, and fake cookies. While doing further research into bypassing these bot protections reverse proxy lists were identified as a possible solution through the academic book “Practical Web Scraping for Data Science” (Broucke and Baesens, 2018).

Modifying request headers to mimic legitimate browser requests. This involves setting user-agent strings, accepting various content types, and incorporating other header parameters to resemble genuine user interactions. Implementing mechanisms to handle and manage cookies appropriately. Browsers often use cookies to maintain session information, and proper cookie management can contribute to a more authentic user-like behavior.

Introducing randomness in the timing of requests to simulate human-like browsing patterns. This helps avoid predictable scraping behaviors, making it harder for detection mechanisms to identify automated activities. Utilising a pool of reverse proxies to obfuscate the origin of requests. This approach involves routing requests through different IP addresses, making it challenging for Indeed to trace and block specific sources.

While investigating the implementation of these anti-bot features, it was proven difficult to find a suitable and reliable proxy list. A reliable proxy list is crucial for maintaining anonymity and avoiding IP bans. Unfortunately, despite efforts to identify a suitable list, none met the project's requirements. After a discussion with the project champion Charles Gillian, he suggested keeping the implementation with the ScrapeOps API and instead of having the spider run on a schedule only use it to populate the database at the start with career data.

Therefore, one future improvement to the implementation would be identifying a suitable proxy list which could be implemented alongside the other measures mentioned above. Another improvement would be choosing a different website to scrap like NIJobs as it has less robust antibot measures, however this comes with the disadvantage of having a lower amount of job posting leading to possibly worse recommendations due to the use of this less rich data source.

4.5 Chatbot

Similarly, to the login system, the foundational elements of the chatbot were established during the initial prototype in CSC3068. Once again, substantial enhancements have been implemented to elevate the chatbot's functionality. These improvements extend beyond the Interest Adapter, as discussed in the previous CSC3068 report. The InterestAdapter is specifically crafted to fulfill requirements related to sentiment analysis and the processing of user input to provide optimal recommendations (as outlined in requirements: Chatbot-001, Chatbot-003, and Chatbot-004).

Additionally, a LanguageAdapter class has been integrated to address Chatbot-002. This integration is aimed at fostering inclusivity and diversity within the chatbot, aligning more closely with the University's overarching goals.

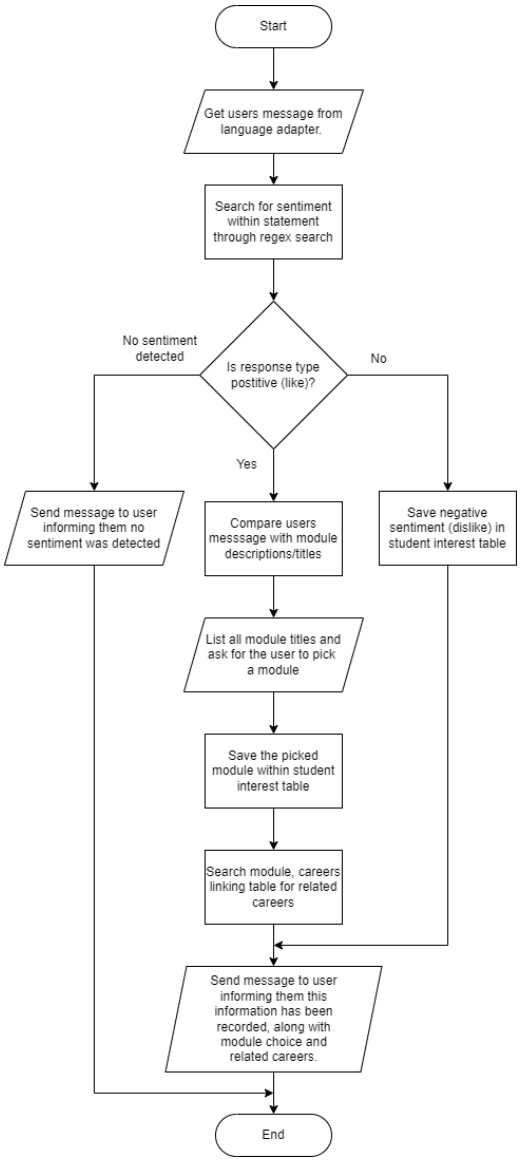


Figure 4.5.1 – High level overview of the interest adapter.

The InterestAdapter plays a crucial role in our chatbot, offering an intuitive interface for students to express and update their academic interests. It seamlessly integrates into the conversation, allowing users to convey preferences naturally through phrases like "I like," "I dislike," or "I hate.", this was discussed in the last submission so will only be briefly mentioned. To identify a module to recommend, first the sentiment of the statement is analysed by performing a regex search on the user's response to find a selection of negative or positive statements. This regex search can be seen below:

```
dislike_match = re.search(r'\b(dislike|do not
like|don\'t like|dont like)\b',
statement_text)
hate_match = re.search(r'\bhate\b', statement_text)
like_match = re.search(r'\blike\b', statement_text)
```

Once this is determined the phrase found by the regex function is removed and a comparison is done on the remaining characters (only when the sentiment is positive) in the user's response, this is compared against the module information within the database to identify any modules that are similar with the positive statement. Then a list of these modules that are determined to have similar descriptions are presented to the user for them to choose from.

This real-time interaction dynamically updates user interests, contributing to a continuously evolving user profile. As users express their likes or dislikes, the chatbot provides personalised module recommendations, enhancing their awareness of academic offerings aligned with their preferences.

The InterestAdapter integrates with the database, establishing connections between user interests and potential career paths using the links made in the web scraping feature (see section 4.4). This is done by checking which module has been recommended for the student based on their interests, then seeing the careers that are associated with that module, allowing the chatbot to provide the user with the job titles. Users gain insights into how their academic preferences align with relevant IT job roles, adding a practical dimension to their academic journey.

To ensure accurate and relevant responses, the adapter intelligently manages context, distinguishing between general likes/dislikes and specific module preferences. This dynamic context handling enhances the conversational flow and user experience. Please see figure 4.5.1 above for a reference to a high-level overview of the algorithm.

Under the hood, each interaction is associated with a unique user identifier (student ID), enabling personalised updates and accurate attribution of user interests. Regular expressions are employed to identify key phrases indicative of user preferences, providing a flexible framework for recognising diverse user inputs. These likes/dislikes are stored within the database to ensure that when a student returns their interests are remembered to inform the chatbot when making future recommendations to this student.

The language adaptation feature within the chatbot seamlessly integrates into the system, enhancing user communication across diverse linguistic preferences. This advanced functionality is achieved through the implementation of a custom LanguageAdapter class, which employs a sophisticated approach to language recognition and response generation. Refer to figure 4.5.2 on the right side of this page for a high-level overview of the algorithm utilised.

The process initiates with the detect_language_based_on_greetings method (see Appendix 4.5.1), an intelligent mechanism that identifies the language based on common greetings present in user input. This initial step ensures a personalised and context-aware interaction right from the outset.

In instances where language detection through greetings proves inconclusive, the system seamlessly integrates the robust langdetect library. This library, employed in the detect method, provides a reliable means of identifying the language, ensuring comprehensive coverage for user input.

The response generation mechanism is dynamic and context-aware, driven by a meticulously curated dictionary, language_responses (see Appendix 4.5.2). This dictionary contains tailored responses for a spectrum of languages, not only acknowledging the detected language but also providing user-friendly guidance. Users are encouraged to use Google Translate for effective communication in English, emphasising a commitment to facilitating clear and meaningful conversations.

To maintain a seamless user experience, the system utilises a deterministic seed for language detection, promoting consistency in language identification across various interactions.

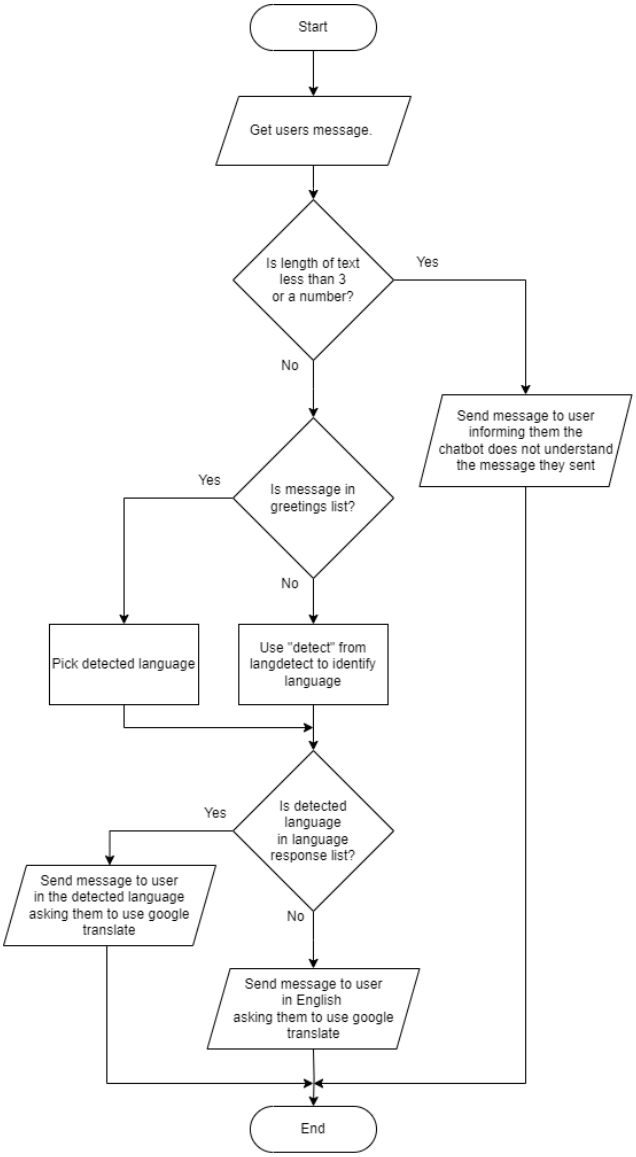


Figure 4.5.2 – High level overview of the language adapter.

This implementation of the chatbot does have its limitations. The first and perhaps most obvious one being the finite number of supported languages for the error response. An improvement can be made here to include additional languages to ensure the most possible coverage to ensure that there is no confusion for our users.

Another limitation surrounding the language adaptation is the fact that even though the chatbot can respond to different languages, this response is limited to asking the user to use google translate in the language of the initial message. This is less than ideal for the user as this means they will have to constantly switch between the chatbot page and google translate to interact with the chatbot, this has the potential for the user to become confused and possible mistype translations from google translate. To improve this google translate could be implemented directly into the chatbot translating the messages sent and received to avoid the user from switching between pages.

An improvement that could be made specifically to the sentiment analysis would be the use of a Natural Language Processing (NLP) model that could be used to identify positive and negative sentiment from any statement regardless of specific phrases that may be used. Thus, improving the range of responses that the chatbot could handle without throwing a generic response to the user.

Another way in which this could be improved is by leveraging other large language models (LLMs) like the popular generative AI model, ChatGPT. ChatGPT could be incorporated into the chatbot by sending the users message to OpenAI's API if the logic within the current implementation couldn't handle the request, this would replace a generic response from the chatbot with a response from ChatGPT, which would seek to further help the user and give them more detailed information.

5 Software Testing

Software testing is a fundamental aspect of software development that ensures the quality, security and reliability of a software application, as IEEE describes Software Testing as, *"To plan and execute tests, software testers must consider the software and the function it computes, the inputs and how they can be combined, and the environment in which the software will eventually operate"* (Whittaker, 2000). The team's approach to testing reflects these principles and is specifically tailored to the features and functionalities detailed in the implementation section of the report.

In the realm of software testing, the team have implemented various strategies to evaluate and enhance the security and usability of their project. This includes the addition of critical security features such as Captcha and Two-Factor Authentication (2FA) to fortify the login process. Such measures are essential in safeguarding user data and preventing unauthorised access.

Moreover, the chatbot functionality underwent significant enhancements. The implementation of advanced language handling and sentiment analysis represents a leap forward in making the chatbot more inclusive and responsive to user needs. This aspect of testing highlights the team's commitment to improving user experience.

The implementation of web scraping also demonstrates the team's effort to ensure that the application can efficiently gather and process external data. This feature required rigorous testing to ensure accuracy and reliability.

5.1 Frontend Functional Testing – Test Plan

The team's approach to frontend functional testing for the project will follow a blend of Manual Scripted Testing and Exploratory Testing, as successfully implemented in CSC3068. This dual-method strategy ensures a comprehensive evaluation of the system's frontend across different user journeys.

Manual Scripted Testing:

The team will create test cases for each frontend component, aligning with the user's expected journey or the ''happy'' path. This method will ensure that each page functions as intended under normal use conditions. Simultaneously, the team will develop test scripts for ''unhappy'' path scenarios which simulate unexpected user behaviors. These tests are important for revealing hidden errors in the frontend, thereby enhancing the depth and reliability of the testing process.

The frontend pages targeted for Manual Scripted Testing include:

- Home Page (Chatbot Page): Testing for functionality, responsiveness and chatbot interaction accuracy.
- Login Page: Ensuring secure authentication, error handling and user feedback mechanisms.
- Sign-Up Page: Verifying user registration process, form validations and data handling.
- Settings Page: Checking for user preference management and effective application of changes.
- Module Information Page: Assessing the accuracy and presentation of module details.
- Grade Dashboard Page: Validating the display and accuracy of user-specific grade information.
- Admin Page: Testing for management of the database, devices, users and backups.

Exploratory Testing:

IEEE describes Exploratory testing as, *"testing without detailed pre-specified test cases, i.e., unscripted testing. ET is not a single testing technique or strategy; it is rather an approach to testing where test design is performed as part of test execution instead of having a test design phase before execution"* (Itkonen and Rautiainen, 2005). The Exploratory Testing will be an unscripted where test design and execution occur simultaneously. This approach encourages us to think outside the box, crafting unique and innovative test scenarios on the fly. Exploratory Testing will mainly focus on the ''unhappy'' paths, targeting unexpected user behaviors and system responses. This method is particularly effective in uncovering issues that might be overlooked in scripted testing, thus broadening the testing scope.

For each of the frontend pages, Exploratory Testing will involve:

- Probing beyond standard use cases to uncover hidden issues.
- Testing for edge cases and unusual user input scenarios.
- Evaluating system resilience and error handling under atypical conditions.

By combining Manual Scripted and Exploratory Testing, the team aim to achieve a robust and thorough assessment of the frontend. A sample of the test carried out can be seen in the Appendix 5.1.1.

5.2 Frontend Functional Testing – Retrospective

Following the outlined test plan, the frontend functional testing combined Manual Scripted Testing and Exploratory Testing, as successfully demonstrated in CSC3068.

Manual Scripted Testing:

In this phase, the team executed test cases for each frontend component. This approach ensured the system functioned as intended under normal use conditions and helped identify issues in 'unhappy' path scenarios.

Test Case Results:

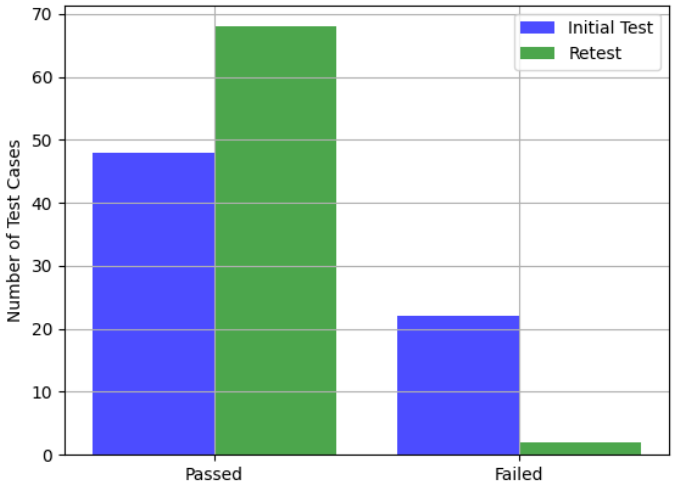
The team derived a total of 70 test cases, focusing on both 'happy' and 'unhappy' paths across key frontend pages including the Home Page, Login Page, Sign-Up Page, Settings Page, Module Information Page, Grade Dashboard Page, and Admin Page. The initial testing phase from CSC3068 yielded the following results:

- Passed: 48
- Failed: 22

After addressing issues identified in the initial test, the team conducted a retest. The outcomes were significantly improved, demonstrating the effectiveness of the correction efforts (results can be seen in figure 5.2.1):

- Initially Passed Cases: All 48 previously passed test cases maintained their status.
- Improvement in Failed Cases: 20 out of the 22 initially failed cases passed in the retest.

Figure 5.2.1 – Frontend Test Cases Results Comparison.



Exploratory Testing:

In this unscripted phase, the team engaged in creative and on-the-fly testing, focusing primarily on 'unhappy' paths. This approach was instrumental in uncovering issues that might have been missed in scripted testing. These issues included mostly problems found on different screen sizes with various widgets and animations. Finding these issues early enough enabled them to be resolved.

5.3 Backend Functional Testing – Test Plan

The backend test plan will evolve to incorporate mostly automated testing with some manual scripted testing, focusing on the chatbot, web-scraping functionalities, grade dashboard and database operations. Just like in CSC3068, the team will utilise the Python unittest framework for creating and managing test cases. A sample of the results for these can be seen in Appendix 5.3.1 and some screenshots for the automated tests can be seen in Appendix 5.3.2.

Chatbot testing:

- Focusing on automated tests for the Chatbot's custom logic adapters, including InterestAdapter, YearAdapter and LanguageAdapter.
- Test cases will provide predefined messages to the get_response() function and verify the chatbot's responses.
- Include both positive and negative test scenarios to ensure robustness and error handling.

Web-scraping functionality testing:

- Manual scripted tests to verify the accuracy and efficiency of the web-scraping process.
- Ensure the chatbot accurately displays career information retrieved through web-scraping.
- Test for error handling in scenarios where web-scraping encounters unexpected website structures or data formats.

Grade dashboard testing:

- Implement tests to validate the functionality of the grade dashboard.
- Test cases will cover data retrieval, display accuracy and user-specific information rendering.
- Include tests for various user roles and permissions, ensuring appropriate access control.

Database functionality testing:

- Automated testing to confirm the integrity of database operations.
- Focus on the creation of tables, field validation and relationships between tables.
- Use a methodical approach to compare actual database schemas with expected structures.

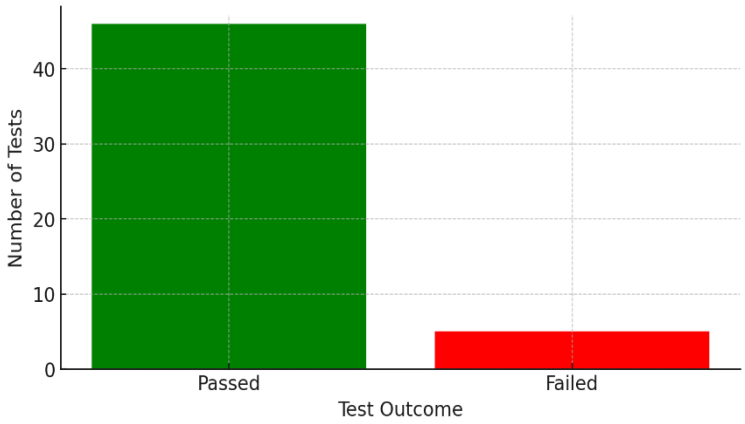
5.4 Backend Functional Testing – Retrospective

The variance in automated test quantity between CSC3068 was not significant; instead, considerable changes were observed in the actual test cases. The subsequent sections delve into various facets of backend testing, with the corresponding pass/fail outcomes illustrated in Figure 5.4.1.

Chatbot testing: The automated tests for the chatbots custom logic adapters (InterestAdapter, YearAdapter, and LanguageAdapter) using the Python unittest framework proved effective. 19 tests passed with 4 fails. Some of the tests for the InterestAdapter were written before the development (test-driven development). This encouraged the team to think about how the code was going to be used before it was written which led to cleaner and more maintainable code. Additionally, it enabled the team to catch bugs early and led to a higher code coverage which resulted in higher quality code.

Web-scraping functionality testing: While manual tests for web-scraping accuracy and efficiency were planned, they could not be executed due to time constraints. Similarly, tests for error handling if the website went down were not completed. Additionally, due to time constraints, the plan to use the ChatGPT API to verify web-scraped career information and to compare module and career descriptions was not worked on. This could have added a significant value in terms of content validation and enhancing the chatbot's utility.

Figure 5.4.1 – Backend Functional Testing Results.



Grade dashboard testing: The tests implemented for the grade dashboard focused on data retrieval, display accuracy and user-specific information rendering, including user roles and permissions. The success in these tests suggested that the dashboard is functioning well in presenting accurate and role-appropriate information. 19 tests passed with 0 fails.

Database functionality testing: Automated tests confirmed the integrity of database operations, including table creation, field validation and table relationships. This ensures that the backend database supports the application reliably. All 9 tests passed.

5.5 Security Testing – Test Plan

There are different approaches to security testing as stated by IEEE, *"Functional security testing: To determine whether security mechanisms, such as access control and cryptography settings are implemented and configured according to the requirements*" and *"Adversarial security testing: To determine whether the software contains vulnerabilities by simulating an attacker's approach - based on risk-based security testing"* (Tøndel, Jaatun and Jensen, 2008). Based on these approaches, here are the different security tests the team decided to use:

Login system security testing:

- Test for authentication mechanisms, ensuring secure and accurate user validation.
- Verify the effectiveness of encryption algorithms and password handling.
- Include tests for Two-Factor Authentication (2FA), CAPTCHA and other security features.

Backend security testing:

- Focus on testing against common security threats like SQL Injection, XSS and CSRF attacks.
- Verify the secure handling of sensitive data, including user information and system configurations.

5.6 Security Testing – Retrospective

The purpose of the security testing was to identify vulnerabilities. The team decided to use manual scripted testing to test each security element of the system.

When testing the authentication and authorisation, the system successfully prevented bypassing the login without valid credentials. Additionally, in the database, it was verified that the passwords were encrypted correctly and cannot be easily decrypted.

When trying to inject scripts like <script>alert('XSS')</script> into the chatbot input field, an alert with the text 'XSS' appeared. With more time the application should validate the user inputs further, so these scripts are not executed but displayed as plain text. The possible implications of a cross-site scripting attack (XSS) could be compromising user data through stealing cookies, phishing or accessing sensitive information.

CSRF attacks were not able to be tested due to time constraints. However, some SQL injection vulnerabilities were tested by entering malicious SQL commands into multiple input boxes. All tests passed showing robustness against these types of attacks. The tests for the Two-Factor Authentication passed all tests for both correct and incorrect codes.

A small sample of the tests discussed can be seen in Appendix 5.6.1.

5.7 System Integration Testing – Test Plan

The System Integration Testing (SIT) plan will focus on ensuring that the various components of the software system work seamlessly together. This phase is crucial for verifying the integrated functionality of the entire system, particularly how different software modules communicate and operate within the unified system. The goal is to detect and address any anomalies in system behavior resulting from integration.

The team will continue utilising Manual Scripted Testing during the SIT phase. This approach has proven effective in previous phases and allows for comprehensive testing of the system.

Key Integration Areas:

- Chatbot and database interaction: Testing how the chatbot interacts with the database, ensuring accurate retrieval and updating of information.
- Chatbot and web-scraping integration: Verifying that the chatbot correctly displays career information obtained through web-scraping.
- Integration of grade dashboard: Ensuring that the grade dashboard accurately reflects user-specific grade information and interacts seamlessly with other system components.
- Web interface integration: Testing the integration of the chatbot, web-scraping functionalities and grade dashboard within the Django web application, ensuring a cohesive user experience.

5.8 System Integration Testing – Retrospective

The System Integration Testing (SIT) phase was conducted at the end of Sprint 3 spanning over a week long period testing the entire system end to end to identify potential bugs within Pathfinder.

The results concluded all key integration areas worked together correctly, with no major issues found. Although, during SIT the team identified and addressed a number of minor bugs. These included issues related to UI elements such as page routing animations. The resolution of these bugs enhanced the overall user experience.

In addition, The database proved to be robust and did not require any modifications, which is a testament to the thorough preliminary design and development work. The benefits of using this approach was made clear from the results. The SIT phase was the final testing stage before choosing to sign off on the functions and proceeding to obtain user feedback.

6 User Evaluation

The user evaluation section covers the methodology used for gathering user feedback on the system and how this feedback was taken into consideration and the necessary modifications that were made based on this feedback. This feedback would be gathered mainly through UAT (User Acceptance Testing).

The users of the Pathfinder system are EEECs students at Queen's University Belfast. Team members engaged with EEECs students, both informally and during organised feedback sessions. This dynamic interaction facilitated instant feedback on evolving features, usability, and overall system performance. The proximity of the development team to the users fostered a real-time exchange of thoughts and ideas, allowing the team to address issues promptly and integrate user suggestions effectively.

6.1 Methodology

The formal methods of user evaluation, namely alpha tests and beta tests in UAT (User Acceptance Testing), played a crucial role in gaining structured insights into the system's performance and user satisfaction. These formal evaluations provided a systematic and controlled environment for assessing the system's readiness and functionality.

In the alpha tests, the team distributed the system to fellow students, instructing them to interact with it as if it were already rolled out by Queens. This simulated real-world usage scenarios and allowed for the observation of how users approached and utilised the system. Following their interaction, users were provided with a standardised feedback form. This form prompted them to share their thoughts on various aspects, including the system's usability, its effectiveness in facilitating their tasks, and any suggestions for improvement.

Similarly, during the beta tests in UAT, the team continued to refine the system by incorporating feedback from a broader user base. The structured nature of these tests ensured that the evaluation process was consistent across different users and usage scenarios. The feedback forms collected during this phase provided quantitative and qualitative data that served as a valuable resource for identifying strengths, weaknesses, and potential areas for enhancement.

After the completion of each formal evaluation session, the team conducted thorough discussions to analyse the findings. This collaborative review process helped in uncovering patterns, common themes, and recurring issues identified by users. Subsequent follow-up sessions with users were conducted to address any lingering questions or to seek clarification on specific feedback points.

The structured feedback collected through these formal methods not only aided in identifying immediate areas for improvement but also contributed to a comprehensive understanding of user behavior and expectations. This deeper insight allowed the team to prioritise enhancements based on user needs and align the system more closely with the intended user experience.

As the team is comprised of EEECs students, this gave the opportunity for the team to gather informal feedback in a unique way. As the users are EEECs students as well this meant that often while the team was working in the CSB or the Library, the development team was surrounded by their users. This allowed the team to sit within groups with other students completing their own final year work where quick on-the-fly feedback was able to be given while features were being developed.

As the team decided to take an Agile Development approach to the project this form of informal feedback came into play seamlessly with the iterative and collaborative nature of Agile methodologies. The Agile Development approach emphasises continuous user involvement throughout the development process, encouraging frequent interactions and feedback loops (Harleen K. Flora, and Swati V. Chandem, 2014).

By intertwining informal feedback within the Agile Development framework, the team maximised the benefits of user involvement in shaping the system. This approach not only accelerated the identification and resolution of issues but also ensured that the system aligns closely with the expectations and requirements of the EEECs student user base. The synergy between Agile principles and informal feedback collection proved instrumental in creating a responsive and user-centric development process.

6.2 Informal Feedback

As mentioned within 6.1 above the team often developed the system in rooms with other EEECs students which allowed the team to gather quick feedback during development. This not only resulted in features being created from scratch beside users allowing them to give their feedback on what they would want before it was even developed, but also allowed the team to quickly show users features halfway through development and gather their thoughts.

An example of this can be seen within the login page. Originally the login page was a pop-up like sign-up. However while developing 2FA this was implemented by adding an additional pop-up for the 2FA code, while creating this a Computer Science student who was working on their own project peered over and noticed how this looked on the screen (figure 6.2.1) and noted that it seemed quite “messy” having so many layers being displayed on the screen, after asking some of the other EEECs students within the room they agreed with this statement.

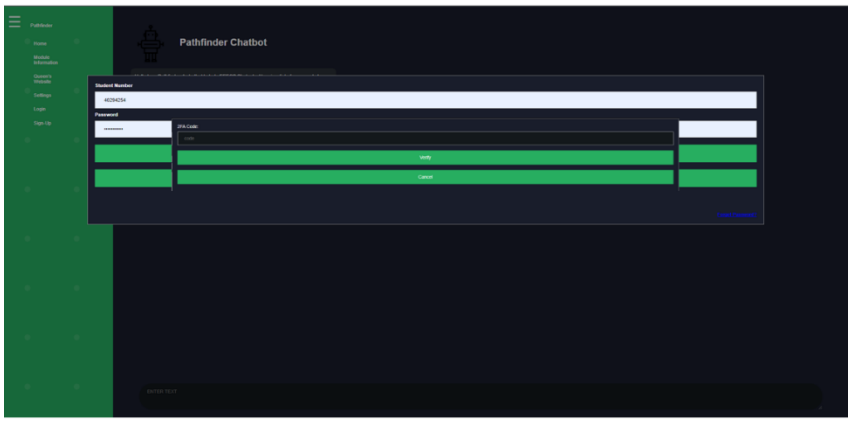


Figure 6.2.1 – Login popup with 2FA popup open, before user feedback.

This allowed the team member who was developing this feature to stop working before any more progress was made and change how the login screen was displayed to better reflect the users feedback, this involved creating a separate login page instead of a popup that allows for the 2FA to remain a popup as the users suggested that keeping the 2FA and having a separate login page feel more in line with other websites. This led to the new login page seen in figure 6.2.2 with the 2FA popup open.

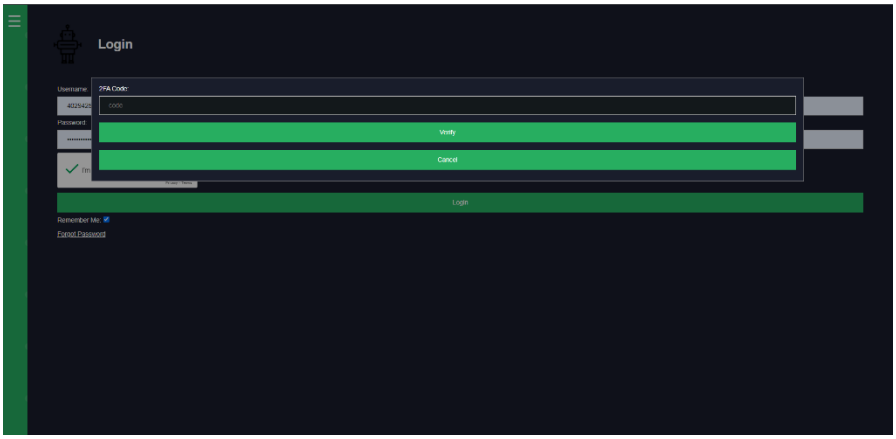


Figure 6.2.2 – Login page with 2FA popup open, after user feedback.

6.3 Project Champion Informal Feedback

During the development of the Pathfinder system, the Project Champion provided invaluable informal feedback, emphasising the importance of security features. This guidance played a crucial role in shaping key aspects of the system.

The Project Champion stressed the importance of incorporating 2FA into the system. This recommendation was rooted in the increasing need for robust security measures in digital platforms, especially in an academic environment where sensitive student data is handled. The Champion suggested using methods like OTP (One-Time Password) generation and QR code scans, which align with modern security practices and provide an additional layer of protection against unauthorised access.

To further improve the system’s security, the Project Champion recommended integrating CAPTCHA into the login process. This feature serves as a defense mechanism against automated attacks and ensures that login attempts are made by actual humans. The CAPTCHA implementation was advised to be user-friendly while effectively filtering out bots and automated scripts.

These enhancements, guided by the Project Champion’s feedback, significantly improved the overall functionality and security of the Pathfinder system.

6.4 Alpha Tests

During the alpha testing phase of the Pathfinder system, various aspects of the system were thoroughly evaluated by a group of users. The feedback gathered from these users was instrumental in guiding the system's refinement. Below is a summary of the key points and concerns raised by the users during this phase.

Users generally found the system intuitive and user-friendly. The well-organised interface facilitated easy navigation, allowing users to locate and utilise different features efficiently. A notable improvement, as per user feedback, was the transformation of the login page from a pop-up style to a dedicated page format. This change was well-received as it aligned more closely with conventional web practices and improved the overall user experience. Additionally, the users stated the system showed good performance and quick response times.

Users also provided specific feedback regarding the colour scheme used throughout the website, which originally featured a predominantly green theme, including the chatbot messages. However, participants suggested a shift towards red for the overall website theme, aligning it with Queen's University's primary colour, which is red. This change was recommended to enhance the system's alignment with the university's branding and to create a more cohesive visual experience for the users. The suggestion to retain green for the chatbot messages was appreciated, as green is commonly associated with messaging and communication apps, thereby maintaining a sense of familiarity and intuitive interaction for the users. The team took this feedback into consideration, understanding the subtle yet impactful role of colour in user experience. As a result of this, the team implemented this colour scheme change as it aligned more with Queen's University Belfast. This gave a sense of familiarity to EEECS Students.

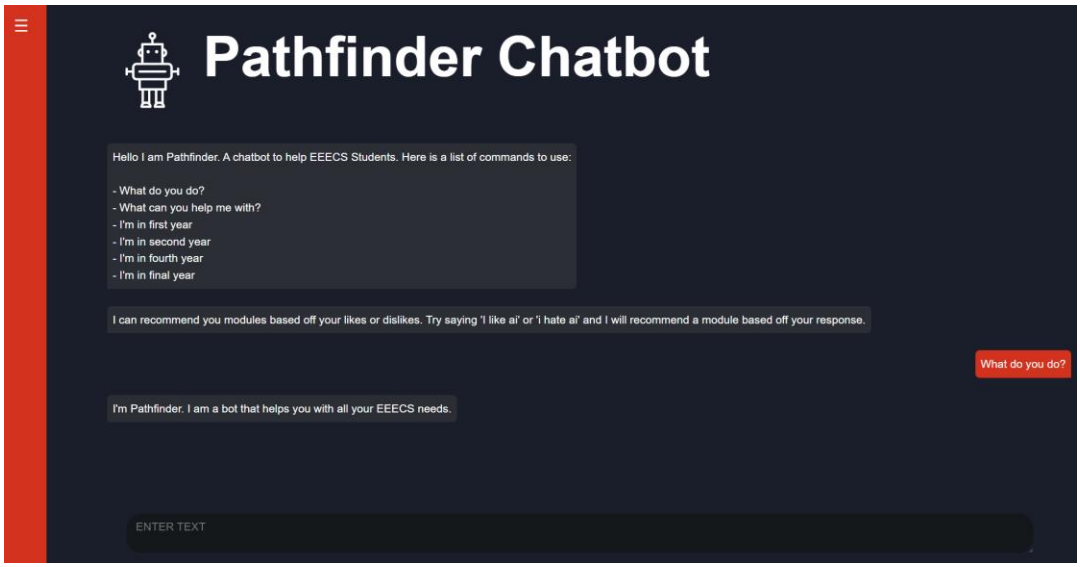


Figure 6.4.1 – Screenshot of updated colour scheme for Pathfinder based on specific user Feedback.

A significant concern raised by users was the absence of a confirmation prompt in the backup and restore functionality. Users pointed out that critical actions such as restoring, rolling back and deleting backups lacked an ''are you sure'' prompt, which could potentially lead to accidental data manipulation or loss. Due to time restraints, this functionality was not implemented post alpha testing.

The chatbot's sentiment analysis capabilities, particularly its response to expressions of likes and dislikes, was highlighted as a positive feature. However, users noted limitations in the chatbot's language adaptation feature. While the suggestion to use Google Translate was appreciated, users recommended the integration of direct translation services within the chatbot for a more seamless interaction in various languages.

Additionally, the feedback from users indicated a need for more detailed information on how the chatbot discussed specific careers related to the modules offered. This aspect is particularly crucial for students as they make informed decisions about their academic paths and future careers. It was highlighted that an improvement to the chatbot could be displaying visual representations of how different modules can lead to various career options. This would help students to map out potential career paths based on their module choices. This additional functionality was not implemented due to time restraints.

The implementation of CAPTCHA and Two-Factor Authentication (2FA) in the login process was well-received. Users expressed increased confidence in the system's security measures and appreciated the modern approach to authentication, especially the use of QR codes for setting up 2FA.

Overall, users were impressed with the Pathfinder system, particularly with its agile and responsive approach to development. The system's ability to rapidly integrate user feedback was highly appreciated. For future enhancements, users recommended further development of the chatbot's module and careers recommendation capabilities, more confirmation prompts and exploring the possibility of integrating direct translation services.

The alpha testing phase proved to be a crucial step in the development of the Pathfinder system, providing valuable insights into user behavior, preferences, and expectations. The feedback collected during this phase was pivotal in shaping the system into a more user-centric and robust tool, tailored to the specific needs and requirements of its user base.

6.5 Beta Tests

In the beta testing phase of the Pathfinder system, the team engaged an external tester (Adam Logan - Fellow Technology Degree Apprentice - EEECS Student at QUB) to provide unbiased feedback on the system's usability, interface, security features, and overall functionality. This phase was pivotal in obtaining a fresh perspective on the system's performance, highlighting areas that required attention and confirming aspects that were functioning well.

Responses From Tester #1		
Question #	**Question**	**Response – External Tester #1 Adam Logan (Fellow Technology Degree Apprentice - EEECS Student at QUB)**
1-general-usability	Was it easy to navigate through the system? Could you find all the features you were looking for?	Yes, it was easy to navigate although on every single page the text was massive, this needs to be changed. The small size should be the medium size.
2-general-usability	How clear and intuitive did you find the user interface? Were there any elements that were confusing or unclear?	Very clear, with the exception of that there is a large blank space below the graph on the grade dashboard and on a large screen (32") the text was extremely large.
3-general-usability	How was the response time of the system? Did you experience any delays or lags?	No, it was very responsive
1-login-security	How smooth was the login process? Did you encounter any difficulties while logging in?	No difficulties.
2-login-security	Were the CAPTCHA and Two-Factor Authentication (2FA) features easy to use? Did they add to or detract from your overall experience?	Yes they were easy to use and did not detract from my experience
3-login-security	Did you feel that your data was secure while using the system?	Yes
1-chatbot	How effective was the chatbot in understanding your queries and providing relevant responses?	Good at understanding basic queries, but when I tried to combine them for example as "I'm in second year and I would like to get into security" it only seemed to understand the first part. Sometimes it would recommend modules that are in "Level 1" when I stated I was in second. Not particularly great when asked clarification questions. I liked it directing to the student help page when it was confused.

2-chatbot	Did the chatbot accurately understand and respond to your sentiments (likes, dislikes)?	If given basic statements, such as “I like …” it is very good as it not only gives obvious modules with just the name in the title but also modules that cover the topic and related topics.
3-chatbot	If you tested the chatbot in a different language, how well did it handle the language adaptation?	Handled it well for some languages not all, a list of supported languages would be good.
4-chatbot	How useful did you find the careers information obtained through web scraping? Was it relevant and up to date?	I don’t think it was up to date as it only ever stated the same job titles.
1-general	Did you encounter any errors? If so, were they handled gracefully?	Did not encounter any errors outside those already mentioned with the chatbot
2-general	Did you feel that any essential features were missing or could be improved?	Everything seems to be here
3-general	Are there any features or aspects of the system you think could be improved?	Other than what has been mentioned with the chatbot and the font sizes, no.
4-general	How satisfied are you with the experience of using the system?	8/10

Key takeaways from the beta testing include:

- General Usability: The tester found the system easy to navigate, indicating a user-friendly design. However, issues such as oversized text and unused space on large screens were identified, pointing towards the need for a more responsive and refined interface design.
- Login and Security: The smooth login process and effective implementation of security features like CAPTCHA and 2FA were well-received, reflecting the system’s robust security protocols.
- Chatbot Functionality: The chatbot showed competence in handling basic queries and sentiments but exhibited limitations in processing complex inquiries and in multilingual support. This feedback underscored the necessity for further development in the chatbot’s natural language processing capabilities.
- Overall Experience: The beta tester’s experience was largely positive, with a high satisfaction rating. Minor improvements were suggested, particularly in the chatbot functionality and interface design.

This beta testing phase was essential for validating the system in a real-world scenario, ensuring that the Pathfinder system meets the high standards of usability and functionality expected by its user base. The insights gained were invaluable in guiding the final stages of system refinement.

While the feedback was mostly positive, the tester did have some critiques / issues with the system. As a result, the team took this feedback and fixed some key issues the user was having. Specifically, in regards to font size and grade dashboard issue. In addition, the beta tester also commented on that a “list of supported languages would be good”. The issue with implementing this is that the chatbot cannot fully converse in the support languages. The chatbot can only identify, detect, and respond in that designated language to alert the user that this chatbot can only communicate in English. The chatbot will then provide a link to google translate for that specific language. Therefore, a list of supported languages would confuse EEECS international students. Hence, the team decided not to implement this change.

The first issue the team addressed was the font size being too big, especially for a 32 inch monitor. Based off this, the team reduced the size of size, medium and large. This ensured that the system was still accessible for users with vision issues however it also made the system accessible for users who have bigger monitors and thus their text will be naturally bigger.

Figure 6.5.1 – Screenshot of large text before the change.

Figure 6.5.2 – Screenshot showing the implementation change.

The beta tester also noticed that the grade dashboard resulted in a large gap underneath the graph and the text was “extremely large” on their 32 inch display. This issue is now fixed, below is the screenshot of the implementation change.

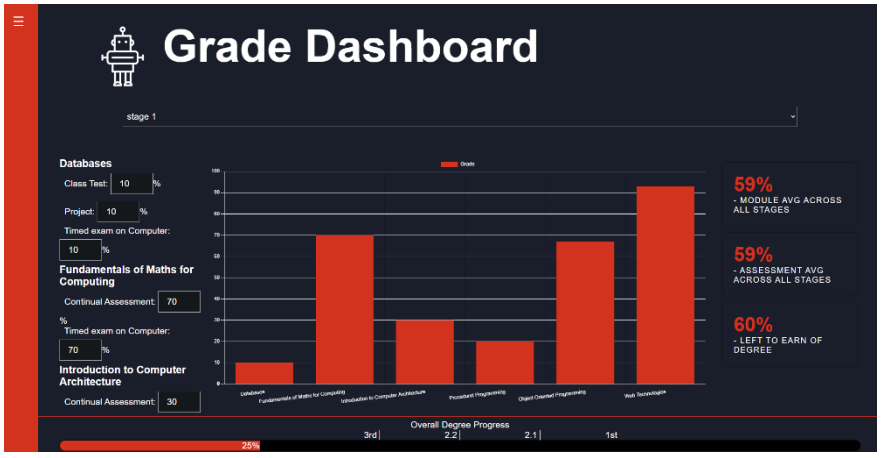


Figure 6.5.3 – Showcases the implementation change to fix grade dashboard user feedback issue.

For the next beta tester, the team decided to change the questions and focus more on the security aspects of the project. The reason for this was because the team wanted to ensure that all security features work as Pathfinder handles sensitive academic student data therefore, it's crucial that these security features work as intended.

Responses From Tester #2		
Question #	**Question**	**Response – External Tester #2 Scott McDonald (EEECS Student at QUB)**
1-settings	How was your experience in setting up 2FA with the QR code?	Setting up 2FA was seamless. Scanning the QR code and linking it with Microsoft Authenticator was straightforward and quick.
2-settings	Did the authenticator app (Microsoft Authenticator) work well with Pathfinder's 2FA system?	Yes, Microsoft Authenticator synced perfectly with Pathfinder's 2FA. It was reliable and I received the TOTP promptly every time.
3-settings	Were there any difficulties or confusion in the settings page, particularly regarding 2FA setup?	The setup process was clear. However, I noticed there's no option to disable 2FA once it's enabled. A toggle switch for this would be useful for user control.
4-settings	How do you feel about the system's security with the current 2FA implementation?	The 2FA adds a good layer of security. It's reassuring to have such a robust system, especially when dealing with sensitive academic data.
5-feedback	Is there any specific feature you think could improve the 2FA experience in Pathfinder?	A toggle button to enable or disable 2FA would be a great addition for flexibility. Currently, it's a bit rigid since you can't turn it off once enabled.
6-login	What was your experience with the CAPTCHA feature during login?	I appreciated the inclusion of a CAPTCHA during the login process. It added an extra layer of security and worked effectively without being overly intrusive.
7-overall	How satisfied are you with the security features, especially the 2FA, TOTP, and CAPTCHA aspects of the system?	Overall, I'm very impressed with the security features. The 2FA, TOTP, and CAPTCHA all contribute to a secure and trustworthy system. My only suggestion would be the addition of a 2FA toggle.

The feedback that the team received from External Tester #2 (Scott McDonald) was generally positive in regard to the security features the team set up. The feedback illustrated that setting up 2FA was "seamless", through scanning the QR code and linking it with their authenticator app – Microsoft Authenticator. Upon setting up 2FA, the tester explained that this was reliable, and they always received the TOTP prompt. In addition, the tester appreciated the inclusion of a CAPTCHA during login, this added an extra layer of security and it worked.

However, the tester did mention that once 2FA is enabled, they are unable to turn it off. The team realised this was an oversight and decided to implement this change in order to allow EEECS students to turn on or off 2FA. This will now include a toggle button which will show red for disabled and green for enabled. In addition, when a student turns on 2FA they will be alerted with a message, so the student is aware. In addition, when they disable 2FA, they will get a "WARNING: 2FA is now disabled" alert message – this is crucial to make sure the student is aware.

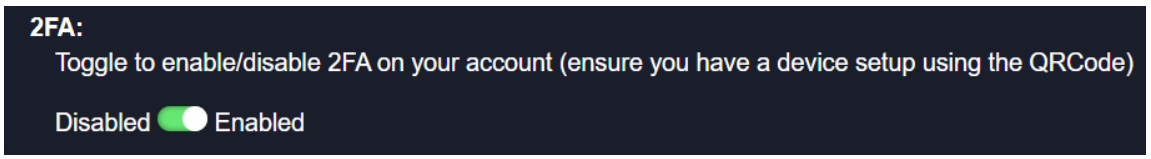


Figure 6.5.4 – Showing toggle button for 2FA enabled.

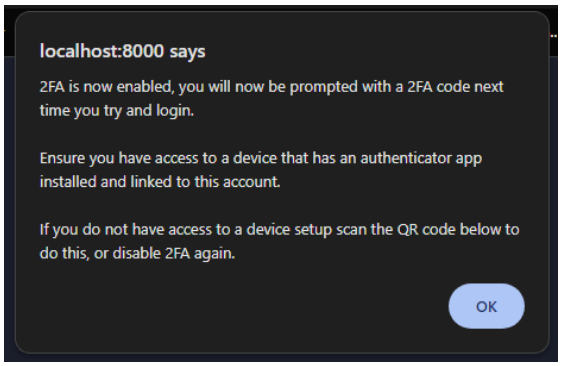


Figure 6.5.5 – Showing alert message when enabling 2FA.

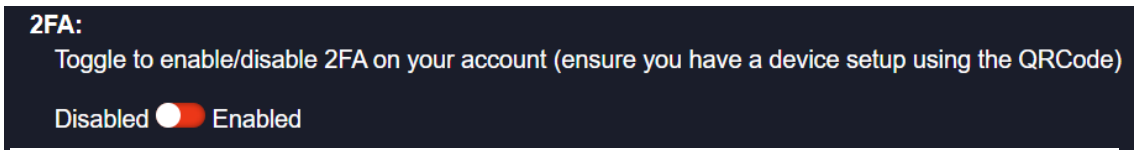


Figure 6.5.6 – Showing toggle button for 2FA disabled.

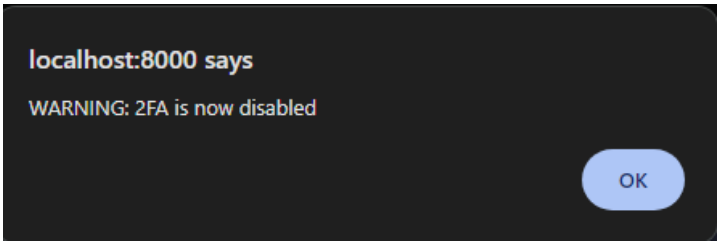


Figure 6.5.7 – Showing alert message when disabling 2FA.

6.6 User Acceptance Testing – Role-Play Based Tester Retrospective

The team employed a unique approach to testing due to the limited availability of certain user groups. This involved role-play-based testing, where team members assumed the roles of different users. This approach was crucial for assessing the system's usability and functionality from diverse user perspectives.

The testing was divided into two main streams: one focusing on the student user experience and the other on the administrative functionalities. This division was strategically chosen to ensure a comprehensive evaluation of the system from both ends. The student user experience stream tested the interface, the ease of navigation, the effectiveness of the chatbot, and the overall user journey. The administrative functionalities stream, on the other hand, focused on the backend operations, including data management, security features, and system settings.

One of the key findings from the student user experience stream was the system's intuitive design and user-friendly interface. The testers, role-playing as students, appreciated the seamless integration of the chatbot and its helpful guidance in navigating through the system. However, they noted multilingual support was limited to recognising only certain phrases which could be expanded. Due to time restraints, the multilingual support for the chatbot was not further worked on.

The administrative functionalities stream uncovered several insights. Testers role-playing as administrators pointed out the lack of detailed instructions on the admin page, which made it challenging to understand and manage certain features. They suggested the inclusion of a comprehensive guide within the admin interface to facilitate easier navigation and management.

A significant observation made during the testing was the system's robust security measures, particularly the implementation of CAPTCHA and Two-Factor Authentication (2FA). However, the testers also recommended the addition of more user-friendly security features that do not compromise on the ease of access for legitimate users as the 2FA required users to have the Microsoft Authenticator app.

7 Project Management

This section of the report will detail the various ways the team worked together to complete the requirements of the Pathfinder project. This covers the planning for the project, the ways in which the team communicated and worked together, and the management of tasks and version control of the code. This also covers the various risks that were identified and how the team navigated these.

7.1 Roadmap

Figure 7.1.1 shows a revised version of the roadmap, compared to the roadmap submitted at the end of CSC3068 there has been three extra additions, these are timelines for the Backup Management System, Security/Login Improvements, and the Server Deployment, this is a high-level overview of our roadmap showing some key milestones, a more detailed breakdown can be found at Appendix 7.1.1.

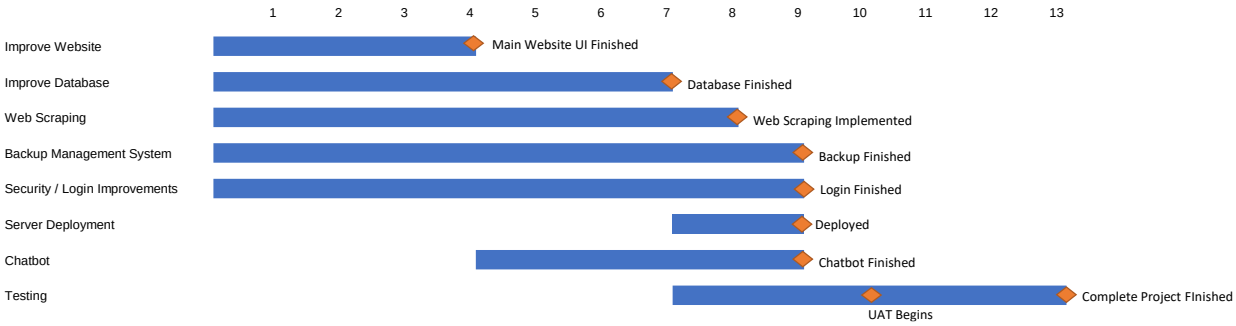


Figure 7.1.1 – High level overview of the roadmap.

Some smaller changes to the roadmap can be seen in this more detailed roadmap at Appendix 7.1.1, which includes a timeline for multi-language support within the Chatbot section, more detail on what is necessary for improving data integrity, additional tables that will need to be added within the database section for the web scraped data and the improvements to the login.

7.2 Collaborative Environments

The team leveraged a diverse set of tools to foster effective collaboration and address various requirements within a hybrid working environment.

For project management, the team opted to maintain Jira as the preferred platform. Jira played a pivotal role in enhancing collaboration by facilitating seamless task assignment among team members. This functionality empowered each team member to easily discern their responsibilities, fostering clarity and accountability. Moreover, it offered an intuitive interface that enabled other team members to gain insight into who was actively working on different aspects of the project. The teams Jira can be found at this link https://csc3068-project.atlassian.net/jira/software/projects/PF/issues/.

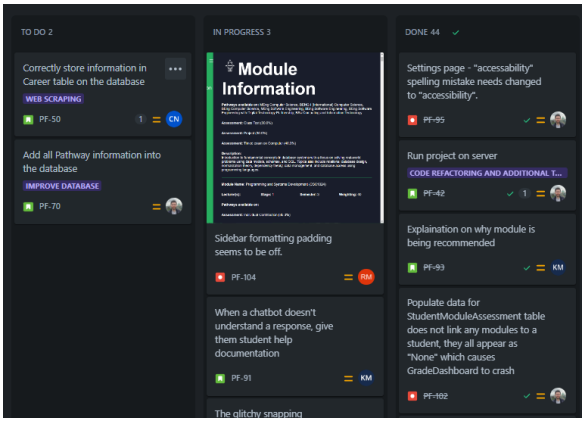


Figure 7.2.1 – Jira Board Example.

A standout feature of Jira was its Board functionality, which served as a dynamic tool for visualising the real-time status of issues assigned to each team member. By leveraging the Board, the team achieved a high level of transparency, enabling efficient tracking of project progress as issues transitioned from the initial "To Do" stage to completion, an example of the Board can be seen in figure 7.2.1. This visual representation provided a comprehensive and easily accessible overview of the project's advancement. Another advantage of the Board is that it can be divided into Sprints allowing for only the current Sprints issues to be shown, allowing for the team to better focus on the current tasks at hand.

In contrast to the team's earlier use of Jira in CSC3068, a refined approach to issue types was implemented. Notably, the team designated overarching challenges, exemplified in Figure 7.1.1, under the "Epic" issue type. This strategic categorisation allowed for a holistic view of major project components and facilitated a more organised and strategic project breakdown. Associated tasks were structured as "Story" issues, promoting a seamless distribution of work among team members. This hierarchical structure not only enhanced project organisation but also contributed to better team

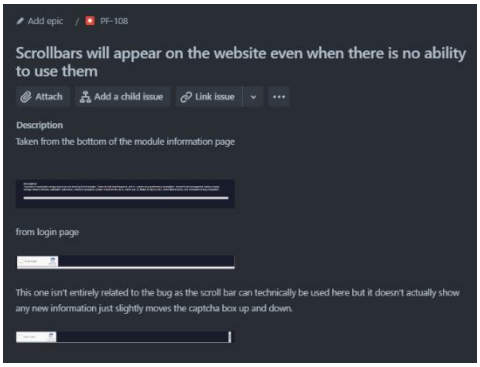


Figure 7.2.2 – Issue Description Example.

coordination and productivity. Jira's issues also allow for greater documentation of a problem when it is needed which can be seen in figure 7.2.2.

The incorporation of the "Bug" issue type brought about notable advantages. It provided a systematic means to flag and document any bugs that might have been overlooked during the completion of corresponding Stories. This proactive bug tracking mechanism proved invaluable in maintaining the overall quality of the project deliverables. The team could easily keep track of identified bugs, prioritise them based on severity, and efficiently organise fixes. Consequently, the structured use of the "Bug" issue type in Jira significantly contributed to a more robust quality assurance process within the project lifecycle.

Another tool the team used for project management, more specifically source control management is GitHub (link to repo https://github.com/KyleMcComb/Pathfinder). Once again, this tool was used throughout CSC3068 for this purpose however, for CSC3069 there have been significant changes to how the team interacted with the tool.

Before the team primarily used GitHub solely to share code where each team member pushed and pulled to a single branch. The singular branch model led to conflicts and difficulties in managing concurrent workstreams, especially when dealing with complex features and bug fixes.

Recognising the need for a more scalable and efficient version control strategy, the team adopted a branching model in GitHub. By creating separate branches for specific tasks, such as feature development or bug fixing, team members could work independently without risking interference with the main codebase. This branching approach allowed for focused development efforts, fostering a more organised and systematic workflow, an example of this can be seen in Appendix 7.2.1. Branching also allowed the team to have separate versions of the code for server deployment as port numbers needed to be changed depending if the code was being hosted locally or on the server, an example of this can be seen in figure 7.2.3.

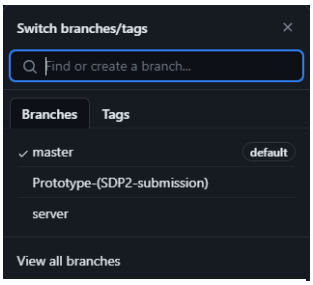


Figure 7.2.3 – Branches

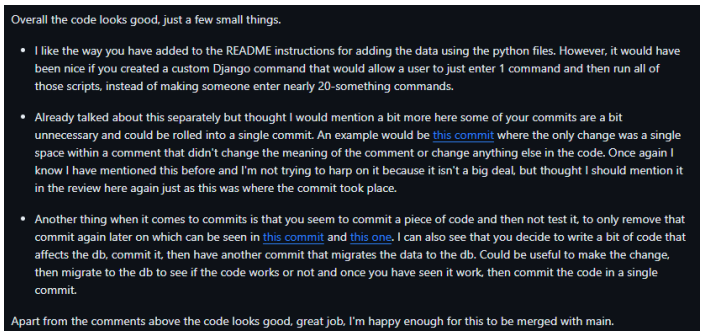


Figure 7.2.4 – Comment by Dean Logan to Ross McAllister within a pull request, link to pull request here https://github.com/KyleMcComb/Pathfinder/pull/10

The implementation of separate branches in GitHub was further strengthened by the team's adoption of a pull request-based workflow. Each branch underwent a thorough code review process before being merged into the main branch, this facilitated a collaborative and systematic evaluation of code quality. This approach promoted knowledge sharing among team members, allowing for the identification of potential issues, improvements, and best practices, an example can be seen in figure 7.2.4. Additionally, this process provided a valuable learning opportunity for team members, fostering a culture of continuous improvement and knowledge exchange within the development team. All pull requests can be found at this link https://github.com/KyleMcComb/Pathfinder/pulls?q=is%3Apr along with screenshots in Appendix 7.2.2.

A development tool which is worth quickly noting is the use the VSCode extension Live Share, Live Share enables developers to collaborate remotely in real-time by allowing multiple members to edit a single file on a team members computer in real-time (Microsoft, n.d.a), similarly to multiple people working on a live Google Docs document. This tool allowed the team to participate in remote paired programming sessions once again improving the team's ability to knowledge share and collaborate in a remote environment. Some commits for these sessions can be seen in Appendix 7.2.3.

Google Calendar was used for scheduling meetings and time management, the team decided it was best to have regular meetings twice a week (Appendix 7.2.4), one being online and the other being in person to give a better balance of remote and in person workflows. Another use of this calendar was to share each other's University timetable allowing for easier scheduling of ad-hoc meetings. Google Calendar was specifically chosen to fulfill this role due to the team's familiarity with it as it's the primary tool used at PwC for time management.

The final tool that was used for collaboration was Discord, this served as our primary tool for communication throughout the project. Discord was chosen due to the team's familiarity with it from previous University projects and the ability to divide conversations into separate channels which allowed for increased productivity and better

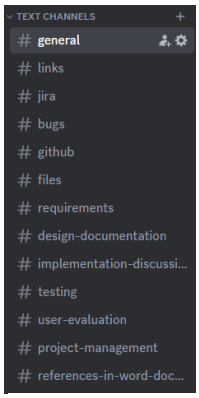


Figure 7.2.5 – Discord channels

documentation as it was clear where a particular piece of information can be found, this can be seen in figure 7.2.5. Discord also allowed integration with two of our other tools Jira (Firnity, 2022) and GitHub (jagrosh, n.d.a) which enabled team members to get notified when any updates took place such as commits, or progress being made on certain tickets.

7.3 Risk Management

Risk	Mitigation	Explanation
Data integrity issues when updating assignment grades.	Implemented signals in the database to automate updating of related records.	To minimise the risk of data integrity issues during the update of assignment grades, the project team implemented signals within the database. These signals automatically trigger necessary updates and ensure that all related records are synchronised accurately. This approach reduces the likelihood of manual errors and inconsistencies in assignment grade data, contributing to the overall reliability and integrity of the system.
Exclusion of students due to non-inclusive design, leading to reputational damage.	Incorporated accessibility options within settings, focusing on colour-blind friendly design.	To address concerns about inclusivity, the project prioritised accessibility in design. The system now includes options within settings to accommodate various accessibility needs, with a particular emphasis on creating a black and white / high contrast interface. This proactive approach ensures that the system is accessible to a diverse student population, mitigating the risk of excluding individuals based on design choices and preventing potential reputational damage. Another accessibility option includes the ability to change the font size to include those with slight eye impairment.
Inadequate handling of multilingual content.	Integrated chatbot language support for more inclusive interaction with international students.	Recognising the importance of catering to international students, the system implemented language support through a chatbot. This feature enables the chatbot to give some form of error response in the student's language allowing for the student to better understand what is happening with the chatbot and why it cannot understand the student, the chatbot also directs the student to a google translate page with the detected language allowing the student to potentially still use the chatbot. By embracing a multilingual approach, the project enhances the inclusivity of the system and fosters a more effective communication channel for students from diverse linguistic backgrounds.
XSS and SQLi vulnerabilities	Employed security features like CSRF tokens and ORM for sanitising inputs.	To bolster the security of the system against cross-site scripting (XSS) and cross-site request forgery (CSRF) vulnerabilities, the project leveraged Django's built-in security features. CSRF tokens were implemented to validate and authenticate requests, while the Object-Relational Mapping (ORM) functionality was utilised for input sanitisation. These measures fortify the system against common web security threats, ensuring a robust defense mechanism and safeguarding sensitive data from potential attacks.
Password and data leaks.	Enhanced security through hidden passwords on the page, encryption of sensitive data, and additional login security steps like 2FA and Captcha.	To mitigate the risk of password and data leaks, the project implemented multiple security measures. Passwords are now hidden on the page to prevent unauthorised access visually. Sensitive data is encrypted to protect it from unauthorised access during transmission and storage. Additionally, enhanced login security measures such as two-factor authentication (2FA) and Captcha were introduced to add an extra layer of protection against unauthorised access and potential data breaches.
Ineffective backup confirmation prompts, leading to accidental data manipulation.	Introduced user-friendly confirmation dialogs for critical backup and restore actions.	To prevent accidental data manipulation during backup and restore actions, user-friendly confirmation dialogs were introduced. These dialogs provide clear and explicit prompts, ensuring that users are fully aware of the potential consequences before proceeding with critical actions. This mitigation measure reduces the likelihood of unintentional errors and enhances the overall reliability of the backup and restore processes.
Oversized interface elements and unused space on large screens.	Applied responsive design principles to adjust content dynamically to different screen sizes, as per user feedback.	Addressing concerns about interface elements and space utilisation on large screens, the project adopted responsive design principles. The interface now dynamically adjusts its content based on different screen sizes, optimising the user experience for various devices. This responsive design approach ensures that the system remains visually appealing and functional across a range of devices, mitigating the risk of oversized elements and inefficient use of screen space as highlighted by user feedback.
Team member absences impacting project progress.	Implemented a shared workstream model where each member's responsibilities are known and can be taken over by others.	To address the risk of team member absences affecting project progress, a shared workstream model was implemented. Each team member's responsibilities are well-documented and known to others, allowing for a smooth transition of tasks in case of absences. This ensures that critical project activities can continue without disruption, minimising the impact of individual absences on the overall progress of the project.

Some risks that the team didn't get to mitigate:

- Dependency on external libraries leading to potential functionality issues. This would be mitigated through reducing reliance on external libraries, focusing on in-house development of components.
- Chatbot's limited up-to-date career information from web scraping. This would be mitigated through regularly updating the database with current career information and improve web scraping mechanisms.
- 2FA requiring specific apps (like Microsoft Authenticator), limiting accessibility. This would be mitigated through exploring and implementing more universal 2FA methods to accommodate a broader range of users.
- Administrative interface lacks detailed instructions, making management challenging. This would be mitigated through developing and including a comprehensive guide within the admin interface for better usability.

7.4 Challenges

A primary challenge the team faced was with parallel development using GitHub. The team would frequently encounter push conflicts when attempting to merge completed branches together. The cause of this was most commonly the binary database files, which meant they couldn't be resolved within the GitHub website or via the individual's IDE. Instead, the command line had to be used every time, specifying which version of the file to keep (theirs or ours). On top of this, it was often difficult to determine which file version was being kept, leading to the database files being corrupted and significant time being lost for the individual performing the merger. Git provides a series of commands that allows for reverting commits and merges and even selecting files to restore from previous commits, which the team utilised to mitigate code loss. However, it was not a perfect solution.

Although the team had occasional pair programming, there were rare occurrences of entire team development within the same environment. This presented the challenge of team members experiencing communication difficulties at times due to the lack of spontaneous in-person conversations, which can lead to misunderstandings or missed details. While some team members shared a similar understanding, this was not always the case with all members. During solo feature development, the lack of possibilities for pair programming and a team environment led to isolation, affecting morale, and belonging. One mitigation of these issues came through the extensive use of the README.md file, where detailed instructions on using certain aspects of code were documented to help provide a clear understanding. An area for improvement would be to incorporate regular team-building exercises that could be as simple as grabbing lunch together after university classes; a small change like this could enhance team morale substantially.

Due to the lack of automated UI testing, developers were responsible for manually testing the front end with every feature. This repetitive and laborious task was time-consuming and prone to errors, resulting in a less efficient testing process. Due to the lack of automation, the front-end bugs were often detected late, causing delays in the development cycle. With the time constraints of CSC3069, the team could not learn automated UI testing in time. Therefore, the alternative solution became prioritising test cases for the UI to maximise test coverage and focusing on areas prone to frequent changes or critical to Pathfinder's functionality. Another mitigation technique required regular maintenance of the existing manual test cases, ensuring that they were kept up to date with the changing code, thus stopping false failures.

8 Discussion and Conclusion

8.1 Summary

In summary, the team implemented a web-based system, “Pathfinder” which aims to help EEECS students at Queen’s University Belfast make informed decisions about module selection and career paths by providing students with detailed information they might need and specific to them and their interests.

The NLP (Natural language processing) and ML (Machine Learning) Chatbot was the main feature of this system and was designed and developed using custom algorithms and logic for processing user inputs, determining module recommendations based on interests and guiding EEECS students through their potential career paths. Following from CSC3068, this chatbot was improved through more detailed responses and career guidance which utilised real and relevant job market data.

In CSC3068, the Chatbot lacked multi-language detection. The Project Champion talked about the importance of International Students, and this was an important feature. To address this, the team implemented a language adapter into the Chatbot. This feature would be able to detect the following languages; Spanish, French, German, Chinese (Simplified), Chinese (Traditional), Malay, Urdu, Hindi, Yoruba, Bengali and Italian. These languages are the most common for international students. The chatbot will recognise the language inputted and reply in that language with the following: “Sorry, this chatbot only supports English. Please use *Link to Google Translate in their own language to English*. Google Translate to communicate in English.”. International students are an important aspect to Queen’s University Belfast and this Chatbot aims to include all EEECS students regardless of language ability.

While in CSC3068, the main focus was on the Chatbot. In CSC3069, the team focussed on incorporating and enhancing features that complimented the Chatbot. This included features such as; the Web-scraping spider which would gather detailed and relevant career information and jobs from Indeed every month, performed by the admin. This would aid the Chatbot in providing detailed career guidance that would be relevant to the modules they were interested in e.g. “Artificial Intelligence”. The team also implemented a detailed module information interface – this provided a detailed overview of EEECS modules and allowed students to filter it based off exactly what they wanted. Either through the search bar or filtering for certain stages, semesters, or pathways. The team also prioritised implementing best modern practice security features such as 2FA (Two-Factor Authentication, TOTP, CAPTCHA and CSRF Tokens for XSS encryption) as this is very important when dealing with sensitive academic student data. The team also focussed on in the early stages of CSC3069, implementing more EEECS data. This included adding all EEECS modules, descriptions, assessments, module codes, lecturers, pathways etc and linking these all together.

The previous system submission in CSC3068, lacked responsive design best practice, this meant that it was hard for EEECS students to navigate or use the website on their mobile phone. To tackle this, in CSC3069, the team carried out an entire overhaul for each component of the system (Chatbot, Module Information Interface, Settings, Grade Dashboard, Login, Signup and Admin Interface) to consider different devices and how the website would be displayed on each device. The website is now accessible to all devices – mobile phones, tablets, laptops, or desktops. This was rigorously tested by the team through testing on multiple browsers, multiple devices and through the Responsiveness Chrome Developer Tool.

An aspect the team failed to implement in CSC3068 was the integration of a backup system. The team thought this was an important issue, especially when dealing with academic data. As a result, the team implemented a combination of local and cloud backups utilising a dual-backup system to address this gap. The local backup utilises the Django-backup library to create .dump files for local backups and the cloud backup includes integration of Azurite (cloud backup simulation). The backups are scheduled every two weeks, and the system will delete the oldest backup when a new one is created. Keeping a limited number of backups (10 for local and 20 in cloud). In addition, the backup system includes email alerts for backup status to the admin – this provides the admin with an extra layer of monitoring and security. This implementation has set up the foundations for a 3-2-1 backup strategy.

8.2 Impact

A social impact that the team identified in the early stages of the project was accessibility – the system’s ability to cater to a diverse student body. For example, the system has features to benefit visually impaired and colour-blind students. The first feature is a font size adjustment, this makes the system more readable for visually impaired students. The second feature is the implementation of a light, dark and high contrast mode. The high contrast mode is crucial for users with colour vision deficiencies – it provides a clear distinction between different elements on the screen. The light and dark mode can help reduce eye strain which is beneficial to students who spend extended periods of time using a computer and the system.

Another important social impact the team identified at the start of CSC3069 was the absence of accessibility and inclusion for international students. These students typically can be alienated, and the team didn't want to alienate them further. As a result, the team implemented a language adapter that would identify the language and reply in that specific language (e.g. Chinese Simplified), telling them that unfortunately the chatbot only supports English however, the student would be provided with a Google Translate link to their designated language and they could then use this to communicate with the chatbot. This feature enhances the engagement of international students and helps them integrate into the EEECS community. Ideally this system would be further enhanced to support full conversation in these languages however this wasn't possible for this sprint due to time and resource constraints.

An economic impact that the team identified at the start of CSC3069 when integrating the dual backup system was the potential cost when integrating cloud backups. As a result, the team decided to implement Microsoft Azurite emulator instead of a full cloud account. The reason for this is because it was a cost-effective approach whilst also still delivering the same functionality. This allowed the team to demonstrate the system's cloud functionality/backup system without the costs. In traditional development and testing phases, cloud emulators are regularly used by developers as they allow them to mimic cloud environments, test applications and troubleshoot issues without incurring the costs associated with real cloud services.

A major privacy and data security concern the team identified early into the project was the handling of sensitive academic student data (especially when dealing with GDPR – General Data Protection Regulation). In CSC3068, the system implemented security features such as the PBKDF2 Algorithm, with an SHA256 hash – this ensured all sensitive EEECS student data was encrypted. Passwords were also hidden behind asterisks when students tried to sign up or login (Password Masking). However, in CSC3069, we aimed to improve on our current security features and implement the best practices according to guidance from the Project Champion. These features were the following: 2FA (Two-factor authentication), TOTP (Time-based one-time password) and CAPTCHA – these all added an extra layer of verification and protected against bots trying to enter the system. In addition, the team also implemented CSRF tokens to prevent XSS encryption which was an important security feature as it protects the system against malicious scripts or data theft. All the mentioned above are crucial to this system as the team dealt with sensitive academic student data.

8.3 Maintenance

The Django Admin Interface is a vital part of Pathfinder for system maintenance. It provides the admin a user-friendly interface for administrative tasks, crucial for keeping the system updated and functioning efficiently. Admins (Designated lecturers) can easily perform CRUD (Create, Read, Update, and Delete) operations – this is very important when managing dynamic content such as student profiles, module details and more. The interface is designed to be simple and easy to use regardless of technical expertise. Therefore, this shouldn't pose a problem for any EEECS lecturers. In addition, the use of ORM (Object-Relational Mapping) in Django simplifies database interactions, thus reducing the need for complex SQL queries, this minimises errors and ensures data integrity. The database, containing modules, lecturer information and more, requires regular updates each semester or year to stay accurate and relevant to EEECS courses. The Django interface facilitates these updates ensuring this information is up-to-date. In terms of scalability, the Django framework supports scalable web application architecture, therefore as the number of users or the volume of data grows, the system can be scaled up to meet increased demands without major overhauls. In addition, thanks to the use of ORM, this allows the system to be largely independent of the database backend. This means that if there's a need to switch to a more scalable database solution (Moving from a smaller SQL database to a larger cloud-based solution e.g. Google Firestore), the change can be made with minimal if any alterations to the application code. By utilising Django and ORM, the team have future-proofed the system. Preparing it to handle increased loads and can be adapted to more powerful databases as needed. In addition, this approach is cost-effective as the upgrading databases or scaling up can be done as required, avoiding unnecessary expenses on overly powerful solutions before they are needed.

In addition, the admin interface acts as a central hub for the system. All the data feeds directly into the Chatbot, Module Information Interface and Grade Dashboard. By having a central hub for data management, the maintenance of the system becomes more streamlined and efficient. Admins can make updates or changes in one place, and these changes are automatically propagated through the system. Thus, reducing workload and potential for error compared to updating multiple systems individually. In addition, this centralised approach ensures uniformity in the information pressed across the different features of the system. Any changes made, will be directly reflected uniformly across chatbot responses, module information display and grade dashboard. Regarding scalability, as the needs of the EEECS department evolve, the system can adapt more easily. Whether it's updating the curriculum, changing module information, or incorporating new courses, these changes can be made centrally.

The web-scraping spider is also an important aspect of Pathfinder and will also require maintenance. The admin is required to run this spider periodically (every month) as this is essential to ensure that the career information and job listings provided to students are current and reflect the latest market trends. Ideally this would be scheduled to run at a certain

time each month. In regards to scalability, the spider would need to be programmatically changed in the event that the job listing website (Indeed) changed significantly as this would impact how the spider would retrieve the data. Ideally, this spider would have the ability to modify the scraping logic or parameters in a web interface however this is unfortunately not the case. It would need to be changed by a developer or someone with technical expertise. The spider has plenty of room to grow, this spider could be scaled to multiple job websites, it could enhance the type and detail of the information it acquires and make this more tailored to the EEECS student.

8.4 Reflection

The team successfully implemented all features the team set out to. At the end of the team's final sprint, all tasks in Jira had been completed and marked off as done. In addition, the project has met its vision as discussed in the introduction of this report. The project vision for Pathfinder was to develop a comprehensive and evolving system that aids EEECS students at Queen's University Belfast in making well-informed decisions about their module selection and career paths, leveraging the vast and diverse sub-disciplines within the EEECS discipline.

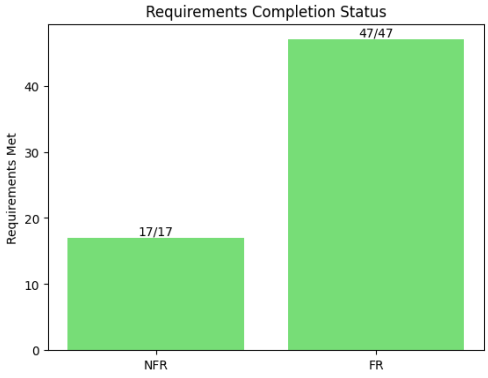


Figure 8.4.1 – Completion of Non-Functional Requirements (NFR) (17/17) and Non-Functional Requirements (FR)

The team has satisfied 17 out of 17 non-functional requirements (NFR). Following on from this, the team has also satisfied 47 out of 47 functional requirements (FR). This shows that the team and the project has accomplished everything the team set out to do at the start of CSC3068 and CSC3069.

The team has learned valuable lessons on what worked well but also the areas for improvement. One lesson the team learned was the importance of adopting a mobile-first approach from the outset of CSC3068. When the team designed the system in CSC3068, it did not have mobile in mind, as a result, this required the team to completely overhaul the UI for each component of the system. If the team had designed the system with mobile compatibility and responsiveness from the start, this would have saved considerable time and resources which could have been spent on other areas of the project.

Another lesson learned was the importance of alpha and beta testing. The team took feedback from these tests from real EEECS students, and this allowed the team to acknowledge what the project was doing well but also areas the team could improve on. The team then took this feedback and integrated changes in order to improve the overall experience for EEECS students. An example of an oversight the team learned from this test was that the font size was too big on a 32-inch monitor. None of the development team had monitors bigger than 27 inches and thus this was an oversight the team missed. The team then corrected this and fixed this issue.

Another lesson learned was adopting an agile and user-centric development proved invaluable. This allowed the team to be more responsive to changing requirements and user feedback, leading to a more refined and useful system. This approach ensures that the system remains relevant and beneficial to the intended audience – EEECS students.

One of the most important lessons the team learned, was implementing security features, especially when handling sensitive academic student data. The team acknowledged this from the start of CSC3068 through implementing the PBKDF2 Algorithm, with an SHA256 hash. Aswell as password masking. However, the team further developed additional security features in CSC3069 according to guidance from the project champion on the modern best practices. This included 2FA, TOTP, CAPTCHA and CSRF Tokens for XSS Encryptions.

One area of improvement for the project would be that while the Chatbot was effective and aligned with the project vision, the technology used for it was a limitation. This was discovered through alpha and beta tests. Students commented on how certain elements could be improved. In hindsight, the team would have used a different technology or library rather than Chatterbot due to the technical limitations the team faced and the workarounds the team had to deploy in order to get around this.

On reflection, the team worked together in collaboration using a mixture of software. This included Jira – for organising and assigning tasks alongside providing a sprint plan to Microsoft Teams and Discord which was the main form of communication the team used. This would be through scheduled meetings. The team also used a mixture of hybrid working. Some meetings were online, while others were in-person. The team found this to be an effective setup and it allowed the team to achieve what the team set out to.

Pathfinder, as a comprehensive and evolving system, has met and exceeded its initial vision. Designed to assist EEECS students at Queen's University Belfast, the project has provided students with the guidance needed when it comes to module selection, alongside academic and career guidance. Pathfinder provides detailed information but also tailored based on individual student interests. The system encompasses a wide array of features to benefit the student such as the

Chatbot, web-scraping spider, module information interface and the grade dashboard. In addition, the Pathfinder project has taught the team valuable lessons in software development. Not only to pursue a career path in industry but also academia. The team followed the SDLC (Software Development Life Cycle) model, and this included analysing requirements for stakeholders, planning, Designing, Implementing, Testing, and deployment.

8.5 Future work

8.5.1 Host a cloud-based database solution in a separate microservice for enhanced security and scalability

Looking forward to the continued development of the Pathfinder system, a key area of future development involves enhancing the system's architecture for improved security and scalability. One significant step in this direction is the plan to host the database in a separate microservice. Currently Pathfinder uses SQLite and Django. SQLite is an excellent solution for lightweight and simple applications, however as Pathfinder grows and deals with more students and data, it will need a more scalable and secure database solution. A cloud-based solution such as Microsoft Azure or Google Firestore would be ideal due to their significant improvements in scalability, reliability, and security over SQLite. This would be relatively simple for the team to integrate due to Django's ORM being agnostic.

8.5.2 Expand system out to all university subjects.

Currently Pathfinder is specifically for only EEECS students. Ideally in the future, this would later be rolled out to all university subjects. Allowing to benefit all students, regardless of their subject or department. This however would involve adding all of this data to the database, and then configuring each Pathfinder component to work for all subjects/departments. For example, the web-scraping spider would need to be configured to scrape for all relevant jobs for each department/subject. In addition, the Chatbot would then need to be altered to work with all subjects/departments. All of this would increase the scale of the application. This would not be feasible without addressing the first point – using a cloud-based scalable database.

8.5.3 Amalgamate this system into Queen's University Belfast website via a designated portal. Link with QSIS

Currently Pathfinder is its own standalone application. It's separate to QSIS and the Queen's University Belfast application. To make use of the system and encourage students to use it, it would be a wise decision to amalgamate this system into the Queen's University website through a designated portal. This would allow students to easily access it and they could even use the same login as they usually do. Pathfinder could then be linked with QSIS to better integrate with the university's current system. This would be beneficial as it would provide one place for students to access everything they need to. In addition, it would allow the data to be up-to-date and could be directly updated/accessed without any manual intervention. Another small feature which could be implemented in the future would be that the lecturer's enter the grades instead of the students in order to stop data redundancy.

8.5.4 – Fully implement common international languages to aid international students.

International students are a key part of Queen's University Belfast. As a result, it's important that they are catered for and not excluded. Therefore, in the future, Pathfinder should implement full support for the most common/popular native languages for International students such as Chinese, Malay etc. Currently, Pathfinder can identify the following languages; Spanish, French, German, Italian, Simplified Chinese, Traditional Chinese, Malay, Urdu, Hindi, Yoruba, and Bengali. The chatbot can detect the language and will then respond in that language telling the user that they only support English and to use Google translate to communicate with the chatbot. The chatbot will then provide a google translate link to that specific language. While this is a good implementation and benefits international students, if the team had more time, ideally the team would like to implement full functionality for a number of these languages to aid International students and to make them feel welcome/included.

8.5.5 – Automatic signup without admin intervention

Whenever a student sign up using their name, student number, student email, stage, pathway and semester, an email will be sent to the administrator. The admin will then have to manually add this student into the database. While this implementation does provide better security and verification, it does add maintenance and continuous monitoring to the admin. It can also cause delays for students and thus leave them waiting and unable to use/benefit from the system. In the future, the team would like to implement an automatic signup feature. This would ideally, take the student's information, verify they are a real EEECS student, and it would then automatically add this student to the database and activate their account. This would benefit not only the admin but also the student. It would alleviate the tasks/responsibilities of the admin and ensure that the student can instantly sign up and start using the system.

Responsibilities

Student Name	Primary Responsibilities
Dean Logan	Report: • All the Implementation section. • Sections 6.1 and 6.2 of User Evaluation. • Sections 7.1 and 7.2 of Project Management. Code: • Backup (local and cloud) • Setup of Azurite • Server deployment • Login security updates • Refactor of backend
Ross McAllister	Report: • All the Requirement's section. • All the Design section. • Section 7.4 of Project Management. Code: • Mobile UI • Populate Database
Kyle McComb	Report: • All the Introduction section. • All the Conclusion section. • Section 6.5 of User Evaluation. Code: • Chatbot • Populate Database
Conor Nugent	Report: • All the Testing section. • Sections 6.3, 6.4 and 6.6 of User Evaluation. • Section 7.3 of Project Management. Code: • Testing • Web Scraping

Agreed Contribution

Student Name	Primary Responsibly
Dean Logan	25%
Ross McAllister	25%
Kyle McComb	25%
Conor Nugent	25%

References

akashdubey-ms, pauljewellmsft, normesta, stevenmatthew, dheerajy1, tamram, jimmart-dev, v-dalc, jangelfdez, DCtheGeek, DennisLee-DennisLee, CarlRabeler, bandersmsft, SyntaxC4, alkohli, mhopkins-msft, v-kents, damabe, ramankumarlive and jeffreyjirwin (2023). Introduction to Blob (object) Storage - Azure Storage. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-GB/azure/storage/blobs/storage-blobs-introduction [Accessed 23 Nov. 2023].

Bergstra, Jan and Burgess, Mark (2008). 2.6 - System Backup: Methodologies, Algorithms and Efficiency Models. [online] ScienceDirect. Available at: https://www.sciencedirect.com/science/article/abs/pii/B9780444521989500124 [Accessed 23 Nov. 2023].

Biryukov, A., Dinu, D. and Khovratovich, D. (2016). Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. 2016 IEEE European Symposium on Security and Privacy (EuroS&P). [online] Available at: https://ieeexplore.ieee.org/document/7467361 [Accessed 20 Nov. 2023].

Broucke, S. vanden and Baesens, B. (2018). Practical Web Scraping for Data Science: Best Practices and Examples with Python.[online] Google Books. Apress. Available at: https://www.google.co.uk/books/edition/Practical_Web_Scraping_for_Data_Science/rkBWDwAAQBAJ?hl=en&gbpv=1&dq=proxy+lists+for+web+scraping&pg=PA189&printsec=frontcover [Accessed 23 Nov. 2023].

Django Project (n.d.a). Password management in Django | Django documentation | Django. [online] docs.djangoproject.com. Available at: https://docs.djangoproject.com/en/3.2/topics/auth/passwords/ [Accessed 20 Nov. 2023].

Django Project (n.d.b). Cross Site Request Forgery protection | Django documentation | Django. [online] docs.djangoproject.com. Available at: https://docs.djangoproject.com/en/3.2/ref/csrf/ [Accessed 20 Nov. 2023].

Django Project (n.d.c). Django. [online] Django Project. Available at: https://docs.djangoproject.com/en/4.2/topics/db/models/ [Accessed 20 Nov. 2023].

Django Project (n.d.d). Django Database Backup — django-dbbackup 4.0.2 documentation. [online] django-dbbackup.readthedocs.io. Available at: https://django-dbbackup.readthedocs.io/en/master/ [Accessed 23 Nov. 2023].

Django Project (n.d). Many-to-many relationships | Django documentation | Django. [Online] Available at: https://docs.djangoproject.com/en/5.0/topics/db/examples/many_to_many/ [Accessed 04 Dec. 2023].

Duisebekova, K., Khabirov, R. and Zholzhan, A., 2021. Django as Secure Web-Framework in Practice. Вестник Казахской академии транспорта и коммуникаций им. М. Тынышпаева, (1), pp.275-281. [online] Available at: https://elibrary.ru/item.asp?id=46118703 [Accessed 20 Nov. 2023] or https://scholar.googleusercontent.com/scholar?q=cache:xe-i-UG0t-UJ:scholar.google.com/&hl=en&as_sdt=0,5 [Accessed 20 Nov. 2023].

Dustdar, S. and Furutanpey, A. (2022). Persistable, Distributable and Versionable Application State for Cloud Service Emulators DIPLOMARBEIT zur Erlangung des akademischen Grades Software Engineering & Internet Computing eingereicht von. [online] Available at: https://web.archive.org/web/20220313140738id_/https:/repositum.tuwien.at/bitstream/20.500.12708/19724/1/Furutanpey%20Alireza%20-%202022%20-%20Persistable%20Distributable%20and%20Versionable...pdf [Accessed 23 Nov. 2023].

Firnity (2022) Discord for jira, Atlassian Marketplace. [online]. Available at: https://marketplace.atlassian.com/apps/1229335/discord-for-jira?tab=overview&hosting=cloud [Accessed: 22 Nov. 2023].

Flora, Harleen K. and Chande, Swati V. (2014). A Systematic Study on Agile Software Development Methodologies and Practices. [online] Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5d2ad96c2dce23e6875dfeeae70a4cf122372457 [Accessed 26 Nov. 2023].

Harackiewicz, J.M., Smith, J.L., & Priniski, S.J. (2016). Interest Matters: The Importance of Promoting Interest in Education. Policy Insights from the Behavioral and Brain Sciences, 3(2), 220-227. https://doi.org/10.1177/2372732216655542 [Accessed 04 Dec. 2023].

Hickey, A. (2003). Requirements Elicitation and Elicitation Technique Selection: A Model for Two Knowledge-Intensive Software Development Processes. Place of publication: ResearchGate. [Online] Available at:

https://www.researchgate.net/publication/221179085_Requirements_Elicitation_and_Elicitation_Technique_Selection_A_Model_for_Two_Knowledge-Intensive_Software_Development_Processes [Accessed 04 Dec. 2023].

Itkonen, J. & Rautiainen, K. (2005). Exploratory testing: A multiple case study. 2005 International Symposium on Empirical Software Engineering. Available at https://ieeexplore.ieee.org/abstract/document/1541817.

jagrosh (n.d.a). Simple github -> discord webhook, Gist. [online]. Available at: https://gist.github.com/jagrosh/5b1761213e33fc5b54ec7f6379034a22 [Accessed: 22 Nov. 2023].

Jan, Mazumdar, P., Agarwal, S. and Banerjee, A. (2016). Microsoft Azure Storage. Apress eBooks. [online] Available at: pp.35–52. doi:https://doi.org/10.1007/978-1-4842-2083-2_3 [Accessed 23 Nov. 2023].

Meltem, S., Turan, E., Barker, W., Burr, L., Chen, Locke, G. and Gallagher, P. (2010). NIST Special Publication 800-132 Recommendation for Password-Based Key Derivation Part 1: Storage Applications. [online] Available at: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf [Accessed 20 Nov. 2023].

Microsoft (n.d.a). Live Share Marketplace Page. [online]. Available at: https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare [Accessed 20 Nov. 2023].

Mitchell, R. (2018). Web Scraping with Python: Collecting More Data from the Modern Web. [online] Google Books. 'O'Reilly Media, Inc.' Available at: https://books.google.co.uk/books?hl=en&lr=&id=TYtSDwAAQBAJ&oi=fnd&pg=PT30&dq=web+scraping&ots=y1CZrJpnah&sig=IfUe7VsVAyU2-O5NnXoaEtH2Yig&redir_esc=y#v=onepage&q&f=false [Accessed 23 Nov. 2023].

Miro. C4 Model for software architecture. [Online] Available at: https://miro.com/diagramming/c4-model-for-software-architecture/ [Accessed 04 Dec. 2023].

pauljewellmsft, tesar-tech, tamram, ddobric, normesta, hyoshioka0128, SaiKishor-MSFT, changeworld, DerekHerman-MSFT, SumanthMarigowda, adamralph, mhopkins-msft, huypub, wadepickett, twooley and jongio (2023). Use Azurite emulator for local Azure Storage development. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/azure/storage/common/storage-use-azurite?tabs=visual-studio [Accessed 23 Nov. 2023].

Ruggiero, P. and Heckathorn, M.A. (2013). Data Backup Options US-CERT United States Computer Emergency Readiness Team. [online] Miralishahidi.ir. Available at: https://www.miralishahidi.ir/resources/data_backup_options.pdf [Accessed 23 Nov. 2023].

Sommerville, I. (2016). Software Engineering 10th Edition. [Online] Available at: https://ieeexplore.ieee.org/abstract/document/7420501 [Accessed 04 Dec. 2023].

Thomas, D.M. and Mathur, S. (2019). Data Analysis by Web Scraping using Python. 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA). [online] Available at: doi:https://doi.org/10.1109/iceca.2019.8822022 [Accessed 23 Nov. 2023].

Tiwari, Saurabh & Rathore, Santosh & Gupta, Atul. (2012). Selecting Requirement Elicitation Techniques for Software Projects. Place of publication: 2012 CSI 6th International Conference on Software Engineering, CONSEG 2012. 10.1109/CONSEG.2012.6349486. [Online] Available at: https://www.researchgate.net/publication/255720277_Selecting_requirement_elicitation_techniques_for_software_projects [Accessed 04 Dec. 2023].

Tøndel, I.A., Jaatun, M.G. & Jensen, J. (2008). Learning from Software Security Testing. 2008 IEEE International Conference on Software Testing Verification and Validation Workshop. Available at https://ieeexplore.ieee.org/abstract/document/4567022.

Waly, M. (2017). Learning Microsoft Azure Storage: Build large-scale, real-world apps by effectively planning, deploying, and implementing Azure storage solutions. [online] Google Books. Packt Publishing Ltd. Available at: https://books.google.co.uk/books?hl=en&lr=&id=EEBPDwAAQBAJ&oi=fnd&pg=PP1&dq=microsoft+azure+blob+storage+.dump+files&ots=pmvaFD4OKI&sig=shc1IUfpIhKgTE2lak-wkRcenCg&redir_esc=y#v=onepage&q=microsoft%20azure%20blob%20storage%20.dump%20files&f=false [Accessed 23 Nov. 2023].

Whittaker, J.A. (2000). What is software testing? And why is it so hard? IEEE Software, 17(1). Available at https://ieeexplore.ieee.org/abstract/document/819971.

Appendix

2.1.1 All Remaining Requirements:

Requirement Number: GUI-Database-001	**Requirement Type:** Non-Functional
Description: The tables within the database should be displayed in a readable manner, possibly in table format. Users should only be able to see data they have access to.	
Rationale: Any user who needs to view the table should be able to read the data correctly.	
Fit Criterion: Confirm that the tables are easily read and that users can only see data they have access to.	

Requirement Number: GUI-Database-002	**Requirement Type:** Non-Functional
Description: Users must get a suitable error message when invalid data is trying to be added to the database. This must happen if the user is trying to add or update a record in the database.	
Rationale: Helps users identify when they are entering incorrect information and how to correct it.	
Fit Criterion: Attempt to add invalid data when trying to add a record to the database, do the same when trying to update a record	

Requirement Number: GUI-001	**Requirement Type:** Non-Functional
Description: The website must include a high contrast/colour blind option, which will change the website's colour scheme to white and black.	
Rationale: The website needs to be inclusive for all students including those that may have sensitive eyesight or are colour blind.	
Fit Criterion: Whenever the high contrast button is pressed then confirm that the website changes all colours to black and white.	

Requirement Number: GUI-002	**Requirement Type:** Non-Functional
Description: The website must include a toggle for a light mode and dark mode.	
Rationale: This will allow users to change the theme of the website allowing it to best fit their preference making the website more usable. This also helps users suffer from eye strain (if they are using the website in a dark environment, they may wish to view darker colours).	
Fit Criterion: Whenever the website loads, press the toggle to confirm it switches to the other theme, then press the toggle again to confirm that the theme can be switched back.	

Requirement Number: GUI-003	**Requirement Type:** Non-Functional
Description: The website must include a method that allows the users to change the font size for the website (radio button that has 3 options small, medium, and large).	
Rationale: Some users may wish to change the text size on the website to better fit their preferences, maybe a user prefers to have larger text as it's harder for them to see.	
Fit Criterion: The size of the text on the website must correspond with the option selected by the user. First, select small then verify that the text size has decreased to the correct size, then select medium and verify that the text size has increased, then select large to verify that the size increases to the correct size. Does this in the opposite direction to ensure that the text size decreases.	

Requirement Number: GUI-005	**Requirement Type:** Non-Functional
Description: A list of clear instructions on how to use the system. This will be displayed to the user once they navigate to the home page.	
Rationale: The system needs a way to clearly tell users what the system does and how it works so that they understand what to do.	
Fit Criterion: Navigate to the chatbot page and verify that a list of instructions appears onload.	

Requirement Number: ModuleInformation-001	**Requirement Type:** Functional
Description: User must be able to search for a module on the module information page.	
Rationale: It makes it easier for a user to find information on a given module, making the system easier to use.	
Fit Criterion: Go to the module information page and search for "data" then verify that all the modules that contain "data" in the title appear.	

Requirement Number: ModuleInformation-002	**Requirement Type:** Functional
Description: Users must be able to filter modules based on stage, semester, and pathway. These filters must be able to stack with each other (more than 1 filter can be applied at a time), and the filters must be applied when a user uses the search as well.	
Rationale: It makes it easier for a user to find module information.	
Fit Criterion: First apply a filter, verify the filter has worked successfully, then remove the filter and repeat for every filter. Then add a random selection of 2-3 filters and verify the correct modules are shown. Once this is done, then try searching for "data" and verify.	

Requirement Number: SignUp-001	**Requirement Type:** Functional
Description: User must be able to enter the information needed for the admin to verify that the student goes to Queens (student number, name, email, pathway, current stage, and semester). Then an email should be sent to the admin with this information in it.	
Rationale: The admin needs this information to ensure that only students of Queens have access and that the correct information will be added to the database. The system will send the email to ensure the correct format in the email is used making it easier for the admin to add the user to the database.	
Fit Criterion: Enter information for each of the fields in the sign-up page and once the form is submitted check the admins email account and verify that an email with the same information entered into the sign-up form is within an email.	

Requirement Number: SignUp-002	**Requirement Type:** Functional
Description: An email to the admin will not be sent unless all information for the sign-up has been entered.	
Rationale: Saves the admin time following up on sign-up requests with missing information.	
Fit Criterion: Attempt to send a sign-up request lacking information for each field and verify that an error message telling the user all fields are required. Then once all fields are filled try submitting the form again and verifying that the form is submitted.	

Requirement Number: GradeDashboard-001	**Requirement Type:** Functional
Description: Whenever marks are added for an assessment, the system will automatically update the students' grade for the module.	
Rationale: This needs to be done to ensure that the students' overall grade for a module aligns with the grades they have received (data integrity).	
Fit Criterion: Confirm whenever a student's mark is updated that the overall grade for that module is changed to reflect that.	

Requirement Number: GradeDashboard-002	**Requirement Type:** Functional
Description: Whenever a module grade is updated, the overall percentage the student has for their degree also updates.	
Rationale: This needs to be done to ensure that the students' overall grade for their degree aligns with the grades they have received in a module (data integrity).	
Fit Criterion: Confirm whenever a student's mark is updated that the overall grade for that module is changed to reflect that.	

Requirement Number: GradeDashboard-003	**Requirement Type:** Functional
Description: The system can calculate statistics like module averages and assessment averages.	
Rationale: This will give students deeper insight into how they are performing in their degree.	
Fit Criterion: Manually calculate a student's module and assessment average then compare this to what the system has produced.	

Requirement Number: GUI-Chatbot-001	**Requirement Type:** Functional
Description: There must be a place for the user to enter text to the chatbot.	
Rationale: The user needs a place to enter text to talk to the chatbot.	
Fit Criterion: Confirm that there is a place where the user can enter text.	

Requirement Number: Chatbot-001	**Requirement Type:** Functional
Description: Must be able to perform sentiment analysis on a user's input.	
Rationale: The chatbot needs to be able to tell if a sentence is positive or negative to give a recommendation.	
Fit Criterion: Chatbot must be able to tell "I like Python" has a positive sentiment towards Python. Chatbot must be able to tell "I don't like Python" has a negative sentiment towards Python. Chatbot must identify "Hi my name is John" as having no sentiment.	

Requirement Number: Chatbot-003	**Requirement Type:** Functional
Description: The chatbot will process user input (user's interests and academic records) and provide suitable recommendations for modules.	
Rationale: Helps students pick modules as students usually aren't provided with enough information to make an informed decision.	
Fit Criterion: Testing the chatbot with different user input to ensure the different outputted courses and modules are suitable.	

Requirement Number: Chatbot-004	**Requirement Type:** Non-Functional
Description: The ability of the chatbot to write a response based on a module recommendation. The response should include a reason why this module (or modules) is being recommended based on the information the user has provided and give some other relevant details about the module(s).	
Rationale: Helps students understand why the chatbot selected this module (giving them more information about the module in the process) and allows them to receive this information in a more digestible manner.	
Fit Criterion: The module malware analyses have been picked by the chatbot so the response crafted will include the fact that it includes 2 exams, involves the use of assembly language, and that it's related to careers within cyber security.	

Requirement Number: Chatbot-006	**Requirement Type:** Non-Functional
Description: When a user enters a statement that can be interpreted as asking for instructions (e.g. "help", "what do you do?", "list instructions", etc) the chatbot will respond with instructions on how to use the chatbot.	
Rationale: This improves usability for the chatbot as it means students can have instructions on demand in the event they forget how to use the chatbot or are new to using the system.	
Fit Criterion: Enter "help" and "what do you do", then verify that the chatbot responds with instructions on how to use the chatbot.	

Requirement Number: Chatbot-005	**Requirement Type:** Functional
Description: When the user provides the chatbot with the stage the student is in, the chatbot will then display all modules for that stage. Similarly, when the student enters what pathway they're in.	
Rationale: This provides an easy and quick way for a student to discover what modules are available for a given stage or pathway directly through the chatbot, meaning if they see a module that is on a different pathway that sparks their interest, they can contact the university directly.	
Fit Criterion: Enter to the chatbot "list modules for stage X" (X being stage number) for every stage and confirm that the modules listed are available for the given stage. Do the same for all the pathway names.	

Requirement Number: GUI-GradeDashboard-001	**Requirement Type:** Non-Functional
Description: Students will be able to see their current overall grade in a readable format. Done by using a progress bar that will fill with the progress of the degree.	
Rationale: It's best for a student to be able to see their overall grade as it helps them keep track of how far along they are in their degree.	
Fit Criterion: Confirm that the correct grade is showing for the student and that the progress bar is "filled" by the correct amount (i.e., if the student has 50% of their degree completed the progress bar will be filled 50% of the way.	

Requirement Number: GUI-GradeDashboard-002	**Requirement Type:** Non-Functional
Description: Students will be able to see their current grades for each module within a bar chart.	
Rationale: This will help students keep track of their progress for the modules, it will also help students compare scores between modules, so they can see what topics require more focus.	
Fit Criterion: Confirm that the correct grade is showing for each module and this is represented within the graph correctly.	

Requirement Number: GUI-Login-001	**Requirement Type:** Functional
Description: A page for the user to enter their username and password to login along with a suitable error message if they enter the wrong credentials. This should contain 2 text fields for email/username and password.	
Rationale: The user must have somewhere within the website to enter in their credentials to login. They also need a way to tell if they have entered the correct credentials or not.	
Fit Criterion: Confirm there is a text field for email/username and another one for a password, enter incorrect credentials to confirm that an error message explaining the credentials are wrong is displayed, then enter correct credentials to ensure the error message does not display.	

Requirement Number: GUI-Login-002	**Requirement Type:** Non-Functional
Description: On the login page, whenever the user is typing their password, it must be hidden by *	
Rationale: Whenever a user is entering their password on the website it must be hidden to avoid bad actors from "shoulder surfing" and being able to record their password and gaining access into their account.	
Fit Criterion: Confirm when a user is entering any text into the password field that instead of the characters that are being entered * is being displayed instead.	

Requirement Number: GUI-Login-003	**Requirement Type:** Non-Functional
Description: A visual indicator for the user that they are currently logged in.	
Rationale: This is so that a user knows if they are logged into their account or if they still need to sign in.	
Fit Criterion: Log into the system with valid credentials, then confirm that there is an indicator to which account is logged into the current session. Ensure that before logging in there is an indicator that no user is currently signed in.	

Requirement Number: GUI-Login-003	**Requirement Type:** Non-Functional
Description: A visual indicator for the user that they are currently logged in.	
Rationale: This is so that a user knows if they are logged into their account or if they still need to sign in.	
Fit Criterion: Log into the system with valid credentials, then confirm that there is an indicator to which account is logged into the current session. Ensure that before logging in there is an indicator that no user is currently signed in.	

Requirement Number: Database-001	**Requirement Type:** Functional
Description: The database must store information about the different modules that are available across EEECS, this information includes the module code, the name of the module, a description, whether it is optional, the weighting of the module, what stage and semester the module takes place and what requirements (if any) needed.	
Rationale: The database must store information about the modules so that the chatbot will be able to access this information to decide what module choices best suit the students' requirements.	
Fit Criterion: Confirmation that a table with the required columns has been created.	

Requirement Number: Database-003	**Requirement Type:** Functional
Description: The database must store relevant information about students. This should include their name, email, password, their current pathway, their interests, what stage they're in and their current percentage of their degree.	
Rationale: The database must store information about students so that the chatbot can find out the necessary information about a student to give a informed recommendations on which modules to pick. Also, so that login information for each of the users can be stored.	
Fit Criterion: Confirmation that a table with the required columns has been created.	

Requirement Number: Database-005	**Requirement Type:** Functional
Description: The database must contain information on the different careers available, this should include the name of the career, the company who is offering the role, a description of the role and the salary (if a salary is listed).	
Rationale: The database must store information about different careers so that the chatbot will be able to access this information and give the student information about careers that may relate to certain modules.	
Fit Criterion: Confirmation that a table with the required columns has been created.	

Requirement Number: Database-006	**Requirement Type:** Functional
Description: The database must contain information on each of the assessments that a module contains.	
Rationale: The database must store information about each of the assessments that a module has to help the chatbot make recommendations to students.	
Fit Criterion: Confirmation that a table with the required columns has been created.	

Requirement Number: Database-007	**Requirement Type:** Functional
Description: The database must contain all the modules linked to their respective pathways.	
Rationale: The database must link the module table and the pathway table together to track the relationships between modules and the pathways so that the chatbot will be able to know which modules are connected to which pathway.	
Fit Criterion: Confirmation that a table with the required columns has been created.	

Requirement Number: Database-008	**Requirement Type:** Functional
Description: The database must contain information about the different careers that relate to modules.	
Rationale: The database must link the module table and the career table together to track the relationships between modules and the careers so that the chatbot will be able to know which modules are related to which careers.	
Fit Criterion: Confirmation that a table with the required columns have been created.	

Requirement Number: Database-009	**Requirement Type:** Functional
Description: The database must contain the grades that the student has for each of their modules, this should also act to record what modules the student is on.	
Rationale: The database must link the module table and the student table together to so the chatbot knows which modules the student has already taken and the grade they achieved, helping it give recommendations for other modules.	
Fit Criterion: Confirmation that a table with the required columns have been created.	

Requirement Number: Database-010	**Requirement Type:** Functional
Description: The database must contain information on the students' grade for each of the assessments within the modules.	
Rationale: The database must link the assessment table and the student table together to store what marks a student scored on each assessment.	
Fit Criterion: Confirmation that a table with the required columns have been created.	

Requirement Number: Database-011	**Requirement Type:** Functional
Description: Information about the database should be backed up once a week at off-peak times.	
Rationale: This is to ensure that if any data loss occurs then the system will have a backup to perform recovery from.	
Fit Criterion: Confirmation there is an existence of another database separated from the original.	

Requirement Number: Database-AccessRights-001	**Requirement Type:** Functional
Description: A student should only be able to view their own information like modules they are enrolled in along with assessments and grades. They should be to view personal information like name, email, student number and what pathway they are on.	
Rationale: A student should be allowed to view their own data within the settings page and the grade dashboard page as they have a right to see any personal data which is stored about them. They should also be able to view the modules they're enrolled on and grades to track progress of their degree. This is also essential so students can contact an admin if their information is incorrect.	
Fit Criterion: Log into the system as a student and confirm that you cannot create, update, or delete any data and view only view personal information in the settings page relating to the logged in student, and confirm that the information that can be viewed on the grade dashboard page is only related to the student logged into the system.	

Requirement Number: Database-AccessRights-002	**Requirement Type:** Functional
Description: An admin should have complete access to the system (create, read, update, and delete all records and tables).	
Rationale: The admin needs to have complete access to the system in case something goes wrong and to edit any necessary details.	
Fit Criterion: Whenever the admin logs in they will be able to do anything to any table.	

Requirement Number: Website-001	**Requirement Type:** Functional
Description: Must support the following browsers and versions: Chrome v113.0, Firefox v113, Edge v113and Edge for Android v111, Chrome for Android v112 and Chrome for IOS v112.	
Rationale: The website must support the latest versions of some of the most common browsers (as of time of writing) to ensure compatibility with user devices.	
Fit Criterion: Navigate to the address of the website on all browsers and ensure that all functionality works for all browsers listed in the requirement.	

Requirement Number: Website-002	**Requirement Type:** Functional
Description: Website must be compatible with touch screen devices such as android phones, IOS phones and window devices.	
Rationale: This is to ensure the inclusion for all users no matter what input method they decide to use.	
Fit Criterion: Navigate to the website using one of the supported browsers using one of the listed devices and confirm that all functionality of the website while using a touch screen (e.g., all buttons still work and when a user selects a text field the touch keyboard appears and the text field still accepts this input).	

Requirement Number: Website-003	**Requirement Type:** Functional
Description: The project must be able to run using docker on a Linux server	
Rationale: Ensure that the project can be deployed so that users can access the website without having to have the files saved locally on their device.	
Fit Criterion: Navigate to the corresponding address for the server within one of the supported web browsers on a separate device and confirm that the website appears and all functionality is working.	

Requirement Number: WebScrapping-001	**Requirement Type:** Functional
Description: The data scraped from the web needs to be formatted to then be stored within the corresponding table in the database.	
Rationale: Information on careers must be kept up to date so that students can see information that is relevant. This data needs to be stored within a database to avoid the system needing to scrap the web every time a user asks about information relating to careers.	
Fit Criterion: Confirm that the information from the web scraping process is present within the database.	

Requirement Number: WebScrapping-002	**Requirement Type:** Functional
Description: Must scrape the web regularly to get the most up to date information on various careers (E.g., salaries and companies).	
Rationale: Information on careers must be kept up to date so that students can see information that is relevant.	
Fit Criterion: Check the information received from the web scrapping process against the information on the websites themselves and confirm they're the same.	

Requirement Number: Login-001	**Requirement Type:** Functional
Description: The correct user is logged into the system when valid credentials are given. (Authentication).	
Rationale: This is to ensure that the correct user is logged into their account.	
Fit Criterion: Enter the credentials for a given account and ensure the correct account has been logged into the system.	

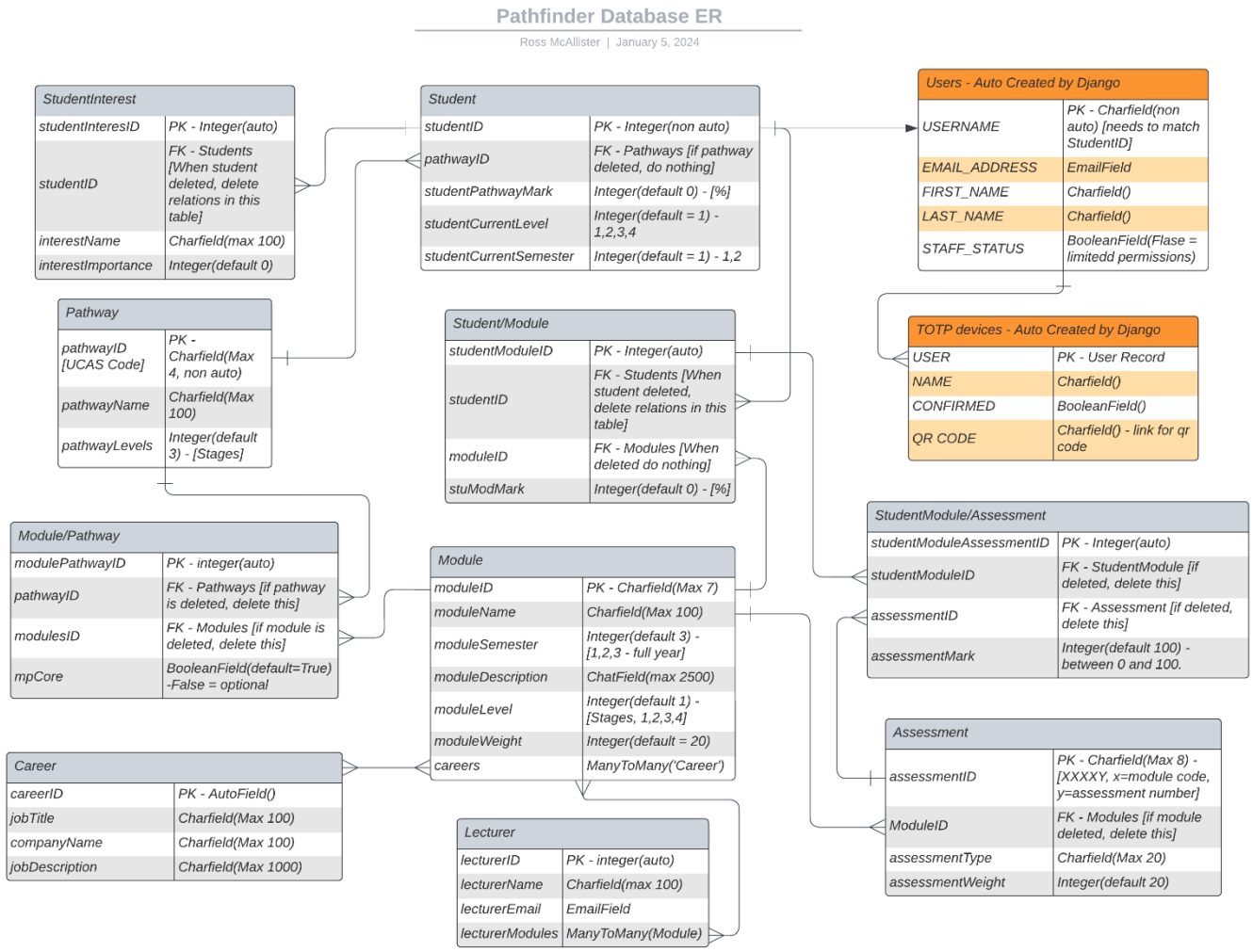
Requirement Number: Login-002	**Requirement Type:** Functional
Description: Ensure the current user who has logged in has the correct access rights assigned to them. (Authorization).	
Rationale: To make sure that users don't get access to data that they should not be seeing.	
Fit Criterion: Enter the credentials for a given account and confirm that the access rights match for that account, check with both a student and admin account. (Verify the correct access level has been granted by confirming that the data displayed on screen is the same as the access rights laid out in the database-AccessRights section).	

Requirement Number: Login-003	**Requirement Type:** Non-Functional
Description: Passwords must be encrypted within the database.	
Rationale: Passwords should not be stored in plaintext in case there is a data breach.	
Fit Criterion: Create a password for any account "password" then verify within the database if the password has been encrypted or if it is still in plaintext.	

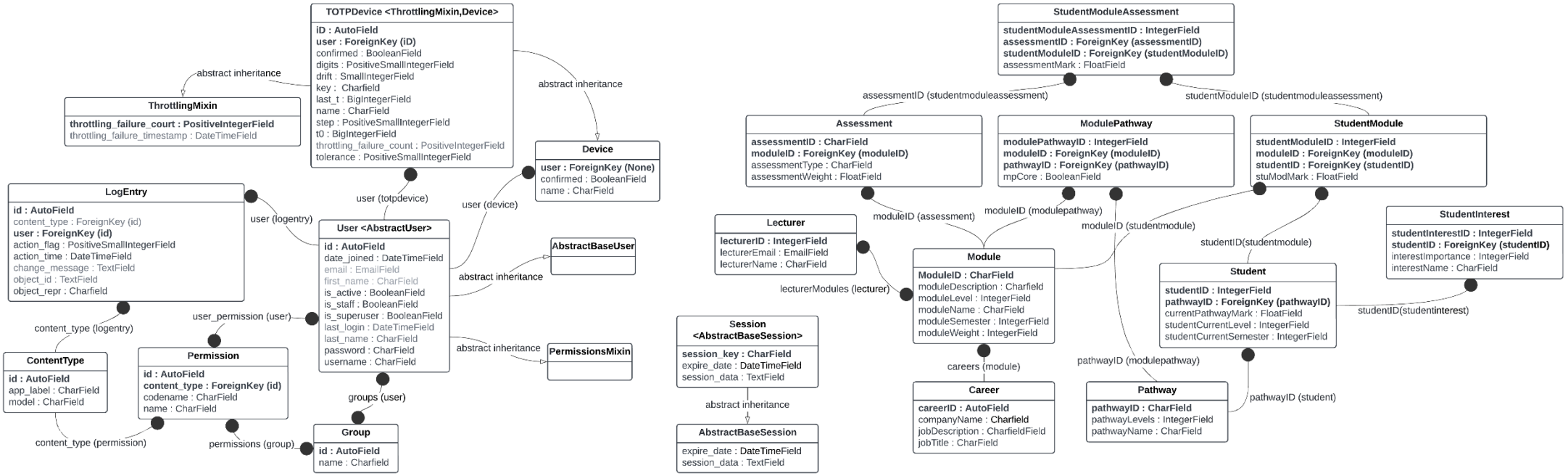
Requirement Number: Login-004	**Requirement Type:** Functional
Description: The ability for the user to be “remembered”.	
Rationale: This is so the user does not have to login every time they visit the page.	
Fit Criterion: Log into the system then close the browser, reopen the browser, and navigate to the website and see if the login was remembered.	

Requirement Number: Login-005	**Requirement Type:** Non-Functional
Description: Login system must not be susceptible to SQLi attacks or XXS attacks.	
Rationale: The system needs to be protected against 2 of the most common attacks on websites that could affect the confidentiality, integrity and accessibility of the system.	
Fit Criterion: Confirm that whenever a user uses the Google and Microsoft SSO that the correct user is logged into the system.	

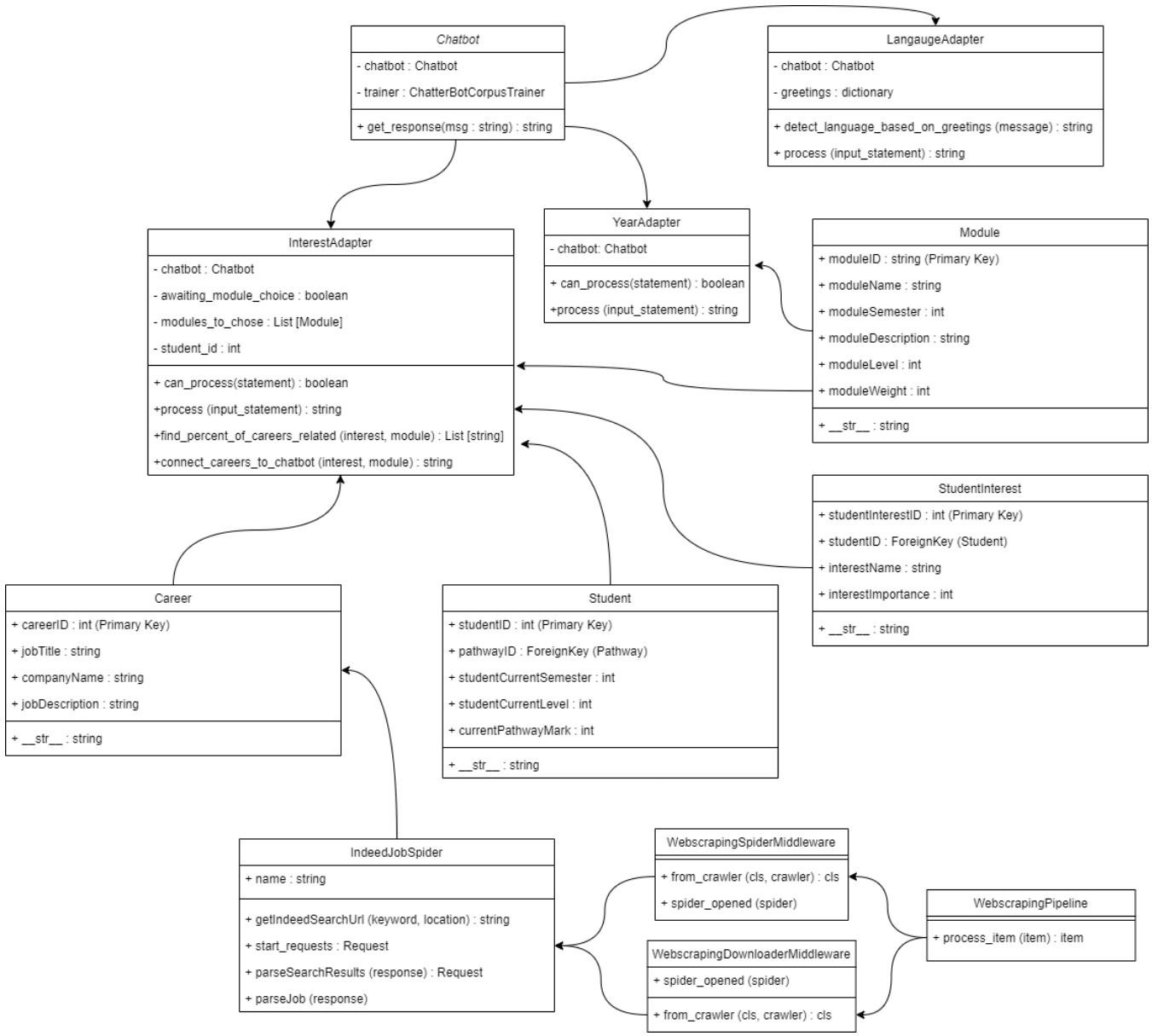
3.1.1 Entity Relationship Diagram



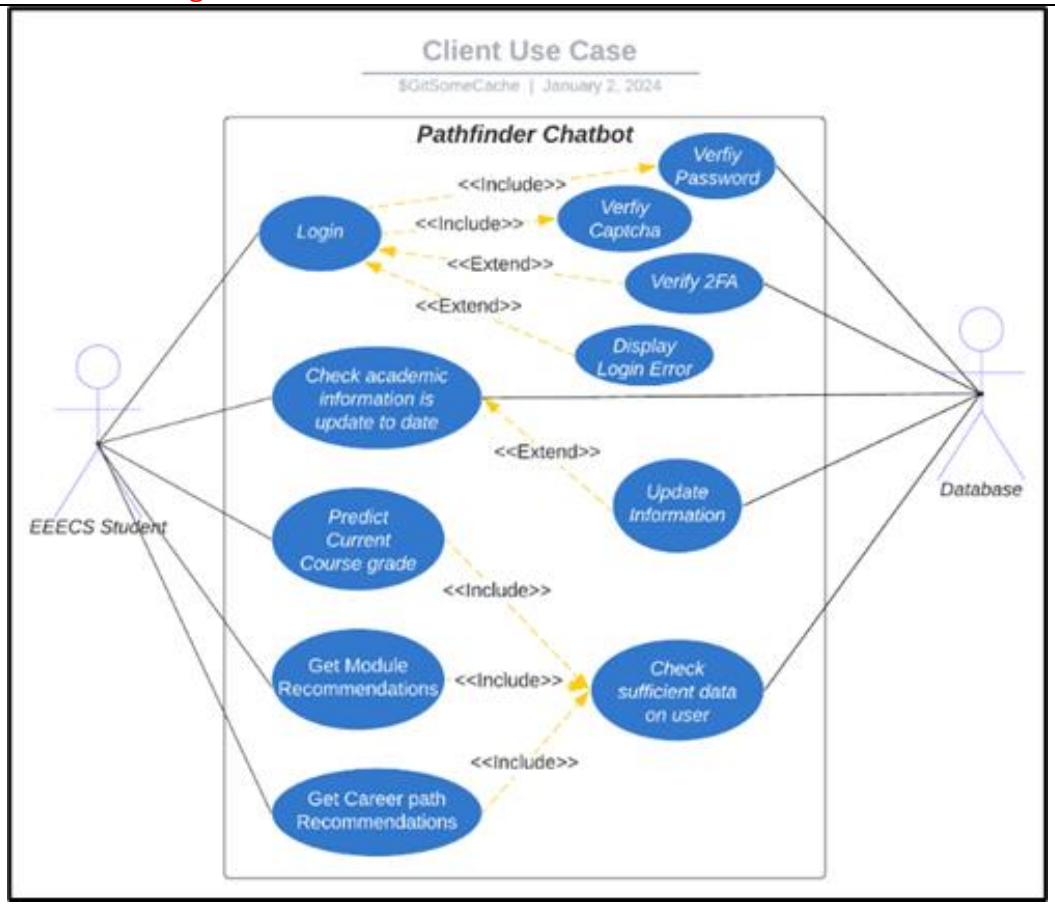
3.1.2 Object Relationship Model



3.1.3 Class Diagram



3.2.1 Use Case Diagrams



Flow of Events for the "Login" use-case

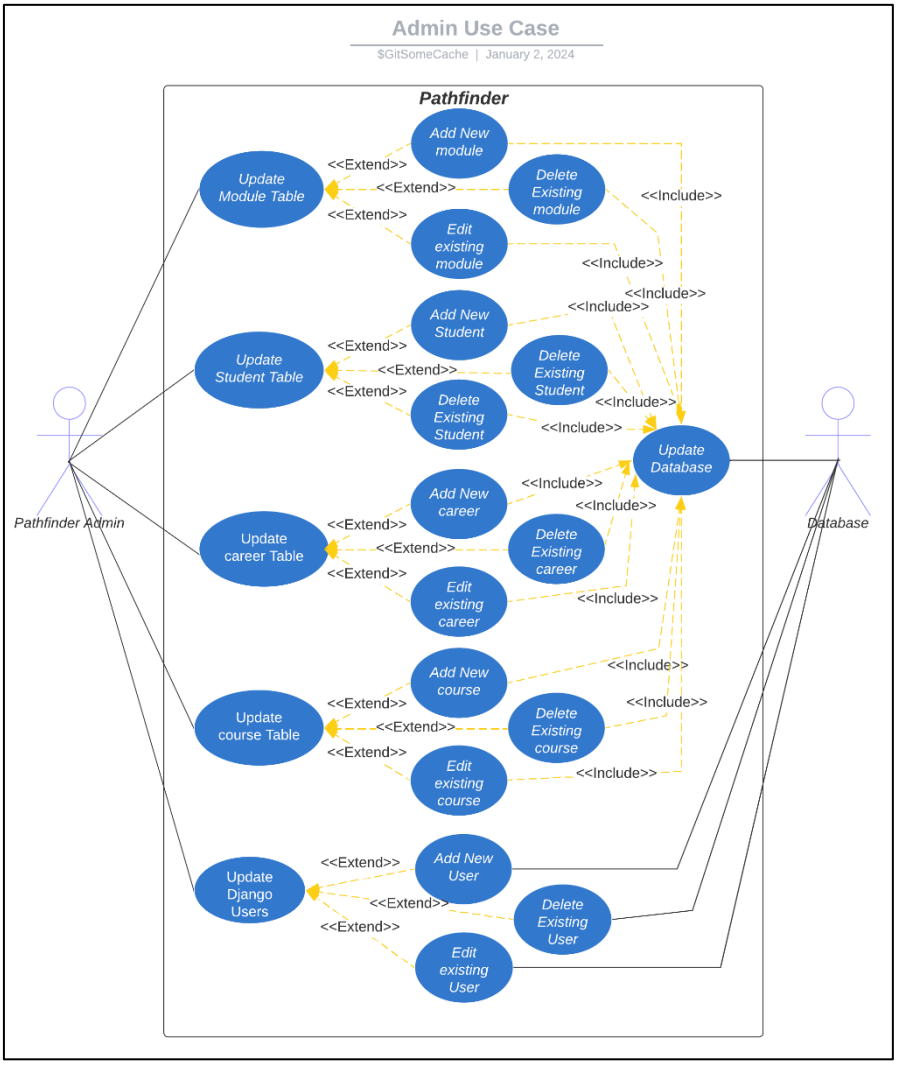
Objective	**To allow the system to access the user's saved data.**
Precondition	The user has just clicked the link for the chatbot website. In the navigation bar there user then clicks the Login button and is greeted with a popup.
Main Flow	1. The user is asked to enter their username. 2. The user is asked to enter their password. 3. The user is asked to complete a captcha form. 4. The user is asked for their 2FA code. 5. The user's credentials are checked against the ones stored in the database and are verified to be equal. 6. The system allows the user to progress within the system.
Alternate Flows	1. Invalid username 2. Invalid password 3. Invalid Captcha 4. Invalid 2FA code -> Display Login Error. The user is not allowed to access the system further. 5. 2FA Is not enabled, therefore it is not asked for by the system.
Post-condition	User has been logged in and the system now has previous information on the users' past modules and grades.

Flow of Events for the "Check academic information is up to date" use-case	
Objective	**To ensure that the system holds valid information about the user to ensure accuracy.**
Precondition	The user has logged in. The user has set up their details in the system before.
Main Flow	1. The user is prompted with the information that is currently stored. 2. The user accepts that the information is correct. -> Post-Condition 3. The user selects that they need to change some information. 4. The user inputs the data that needs to be changed and it is stored into the system. 5. The system loops back to step 1 and repeats.
Alternate Flows	1. First time login, the system needs to prompt the user and ask for the required information. 2. The system shows the user their input and asks them to verify it is correct. 3. The user information is stored into the database. -> The user progresses in the system to post-condition.
Post-condition	Accurate information is stored within the system about the user, the user can ask the system about the other use cases.

Flow of Events for the "Predict current course grade" use-case	
Objective	**By using the user's current grades, the system can calculate what grade they are currently sitting on, regarding their whole course.**
Precondition	The user has logged in. The user has relevant data stored.
Main Flow	1. The user asks the system to calculate their current course grade. 2. The system verifies that the user has enough data stored by accessing the database. 3. The system uses the user's module grades and module weights to perform the calculation. 4. The system prompts the user with their estimated course grade.
Alternate Flows	1. The user doesn't have enough data, e.g., their course isn't stored, or none of their previous module grades are stored. 2. The system prompts the user with an error, explaining there isn't enough information to calculate.
Post-condition	The system prompts the user with either their calculated grade or an error message. The user can then ask for another use case.

Flow of Events for the "Get module recommendations" use case	
Objective	**By using the user's previously selected modules, likes and dislikes, the system can provide module recommendations to the user for the next academic year.**
Precondition	The user has logged in. The user has relevant data stored.
Main Flow	The user asks the system to provide module recommendations. The system asks for their likes and dislikes. The system verifies that the user has enough data stored by accessing the database. The system uses the user's previous module choices, likes + dislikes to provide a recommendation. The system prompts the user with the modules with relevant information about the modules and where they can find further information.
Alternate Flows	1. The user doesn't have enough data, e.g., they haven't begun 1st year yet (no previous module choices). 2. The system prompts the user with an error, explaining there isn't enough information to predict.
Post-condition	The system prompts the user with either a list of modules or an error message. The user can then ask for another use case.

Flow of Events for the "Get career path recommendations" use-case	
Objective	**By using the user's previously selected modules, likes and dislikes, the system can provide career path recommendations to the user.**
Precondition	The user has logged in. The user has relevant data stored.
Main Flow	The user asks the system to provide career path recommendations. The system asks for their likes and dislikes. The system verifies that the user has enough data stored by accessing the database. The system uses the user's previous module choices, likes + dislikes to provide a recommendation. The system prompts the user with career path recommendations, with relevant career information and what modules would help with this career choice.
Alternate Flows	1. The user doesn't have enough data e.g., their course isn't stored or not enough known about user's likes and dislikes to provide an accurate recommendation. 2. The system prompts the user with an error explaining there isn't enough information to recommend a career.
Post-condition	The system prompts the user with either a career recommendation or an error message. The user can then ask for another use case.



Flow of Events for the "Update Module" use case.

Objective	**Allow the admin to update the information stored on modules within EEECs**
Precondition	Admin user has access to the database
Main Flow	1. Admin selects the option to update the modules. 2. Admin decides to add a new module. 3. Admin enters the relevant information for the new module and links the module to the 1 or many courses that it belongs to. 4. Admin links the module to relevant career job entries
Alternate Flows	3. The admin decides to update an existing module. The admin selects the module to update, and they are shown all the module's fields. The admin can then change the fields or links to other tables. 4. The admin decides to delete a module that no longer exists in EEECs. The admin selects the row that the module is in and deletes it. The admin then saves the table.
Post-condition	The new updated information is saved to the database.

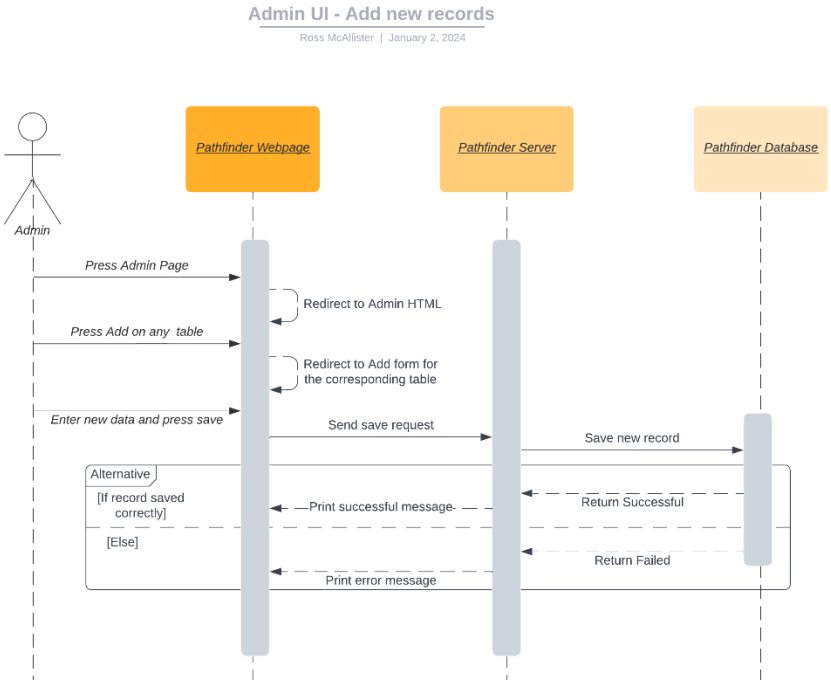
Flow of Events for the "Update User" use case.	
Objective	**Allow the admin to update the user table with students inside EEECs**
Precondition	Admin user has access to the database
Main Flow	1. The admin wants to add a new student as a user to the system. 2. The admin takes the student's basic information (name, student number, current course) and uses it to make a new entry into the user table. 3. A new password is generated for the user, and the admin will send this to them. The user will be able to change this later.
Alternate Flows	5. The admin decides to update an existing user. The admin selects the user to update, and they are shown all the user's fields. The admin can then change the fields or links to other tables. 6. The admin decides to delete a user that no longer exists in EEECs. The admin selects the row that the user is in and deletes it. The admin then saves the table.
Post-condition	The new updated information is saved to the database.

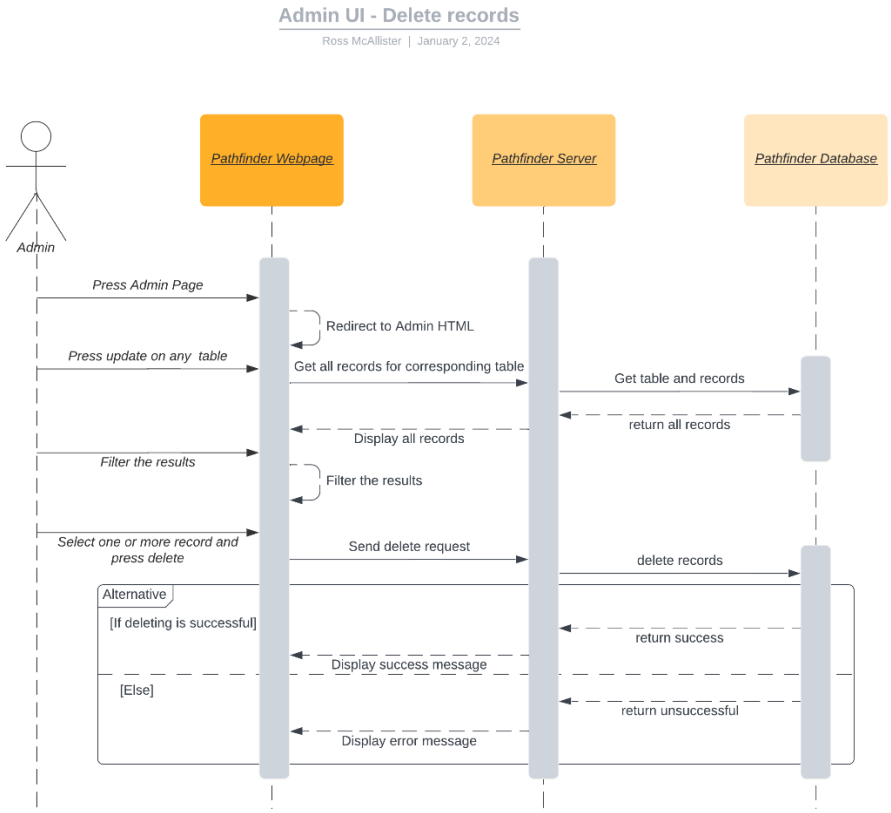
Flow of Events for the "Update Career" use-case	
Objective	**Allow the admin to update the career table with industry jobs and careers**
Precondition	Admin user has access to the database
Main Flow	1. The admin wants to add a new career to the system. 2. The admin takes the information (useful modules, average salary etc) of the new career and uses it to make a new entry. 3. The table is then saved
Alternate Flows	4. The admin decides to update an existing career. The admin selects the career to update, and they are shown all the career fields. The admin can then change the fields or links to other tables. 5. The admin decides to delete a career that is no longer relevant. The admin selects the row that the career is in and deletes it. The admin then saves the table.
Post-condition	The new updated information is saved to the database.

Flow of Events for the "Update Course" use-case	
Objective	**Allow the admin to update the course table with courses within EEECs**
Precondition	Admin user has access to the database
Main Flow	1. The admin wants to add a new course to the system. 2. The admin takes the information (course code, name etc) of the new course and uses it to make a new entry. 3. The admin creates the necessary links from the new course to all its modules. 4. The table is then saved.
Alternate Flows	5. The admin decides to update an existing course. The admin selects the course to update, and they are shown all the course's fields. The admin can then change the fields or links to other tables. 6. The admin decides to delete a no longer relevant course. The admin selects the row that the course is in and deletes it. The admin then saves the table.
Post-condition	The new updated information is saved to the database.

Flow of Events for the "Update Django Users" use case.	
Objective	**Allow the admin to update the user table that was created by Django with User records, that link to Student records.**
Precondition	Admin user has access to the database.
Main Flow	1. The admin wants to add a new User to the system. 2. The admin takes the information (course code, name etc) of the new User and uses it to make a new entry. 3. The admin creates the necessary link from the new user to a student record. 4. The table is then saved.
Alternate Flows	5. The admin decides to update an existing user. The admin selects the user to update, and they are shown all the user's fields. The admin can then change the fields or links to other tables such as passwords and TOTP fields. 6. The admin decides to delete a user that is no longer relevant. The admin selects the record that the user has and deletes it. The admin then saves the table.
Post-condition	The new updated information is saved to the database.

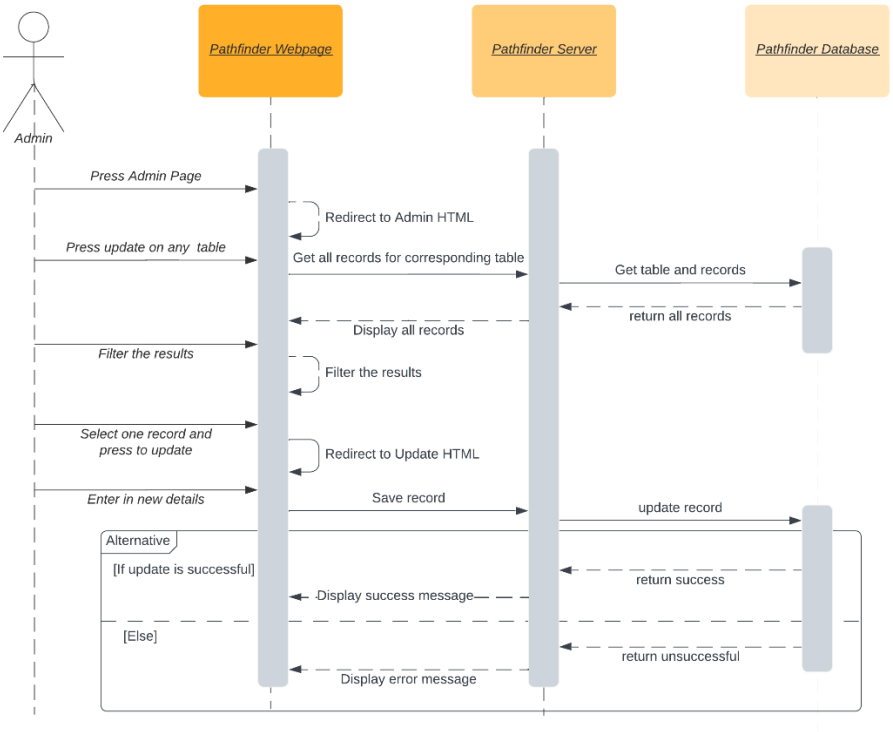
3.2.2 Sequence Diagrams:





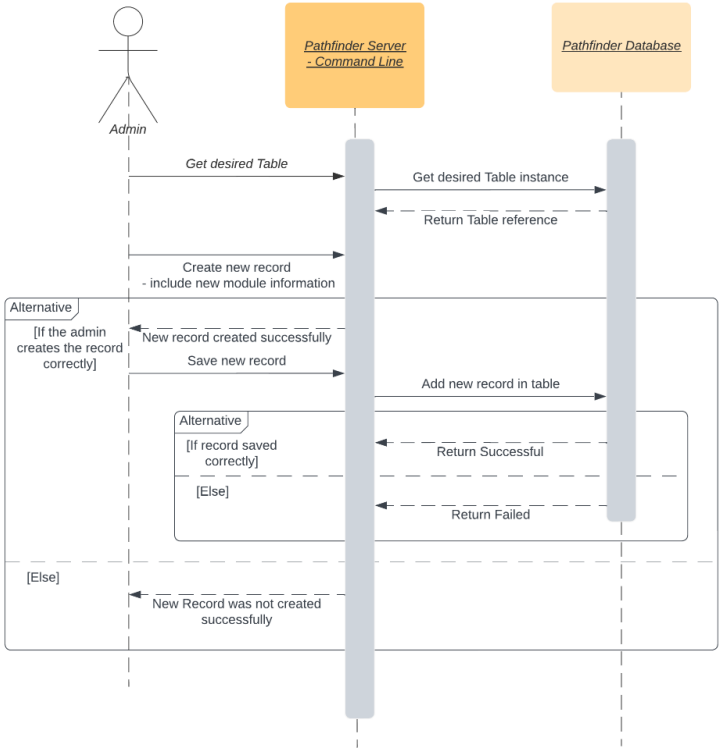
Admin UI - Update record

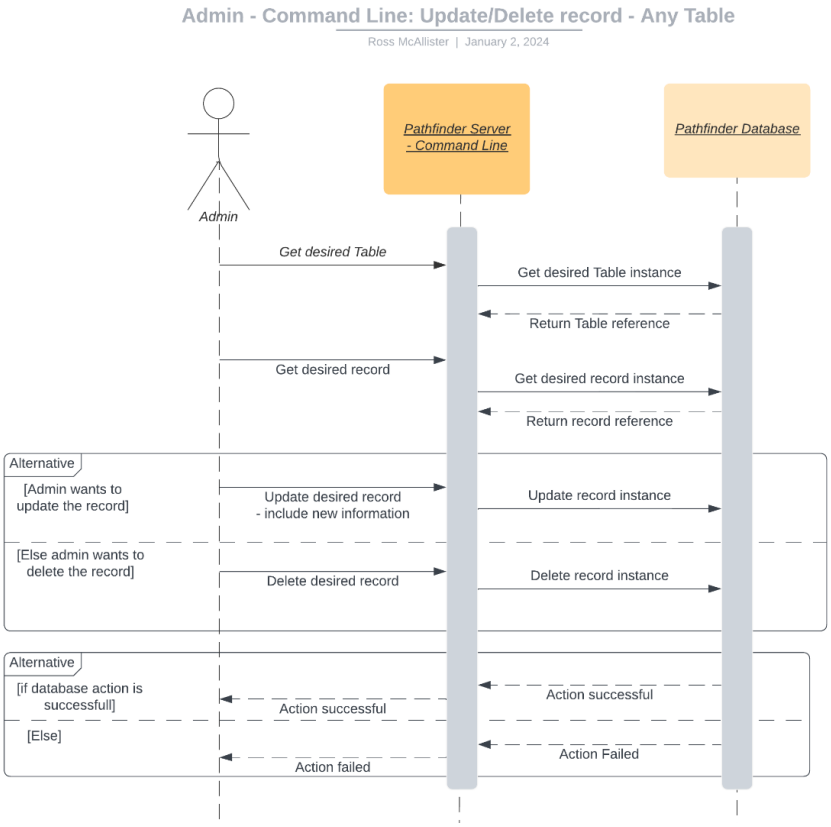
Ross McAllister | January 2, 2024

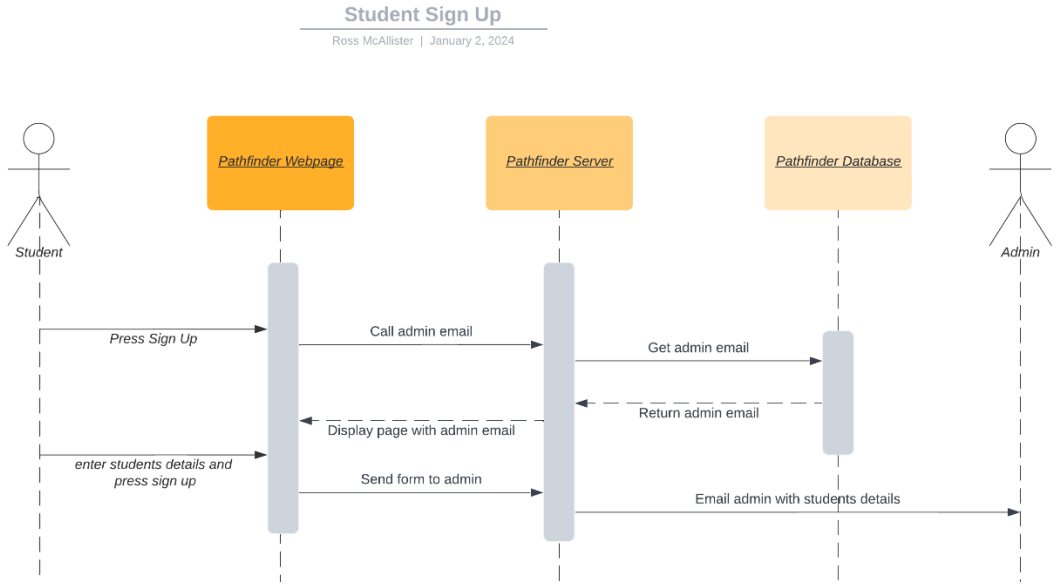


Admin - Command Line: Add new record - Any Table

Ross McAllister | January 2, 2024

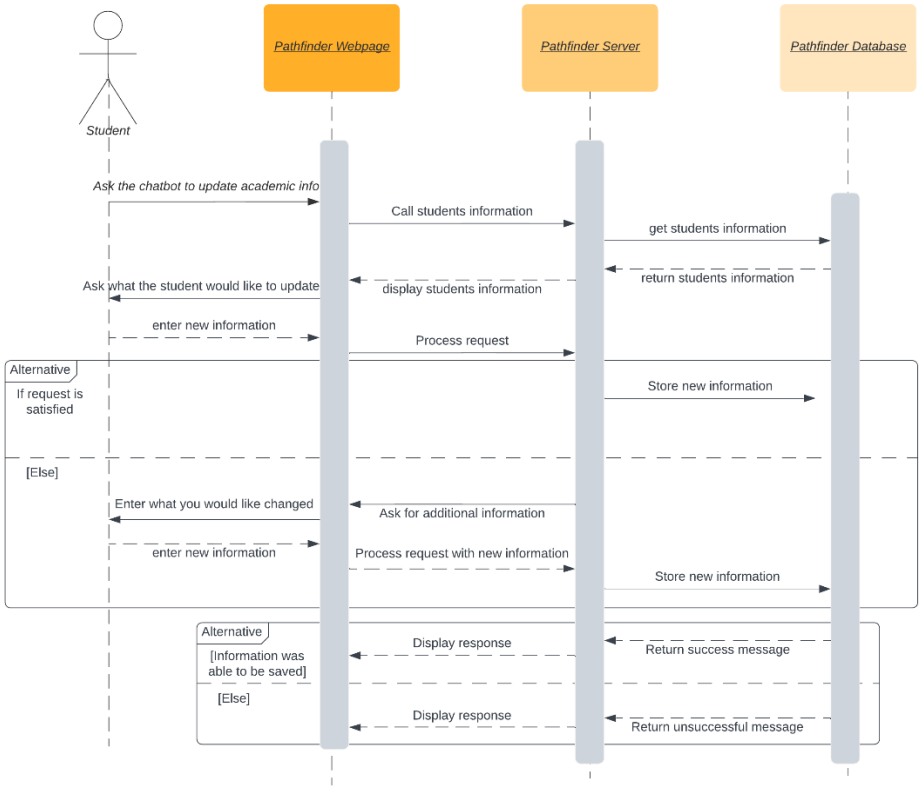



Admin - Command Line: Update/Delete record - Any Table
Ross McAllister | January 2, 2024
Admin
Pathfinder Server - Command Line
Pathfinder Database
Get desired Table
Get desired Table instance
Return Table reference
Get desired record
Get desired record instance
Return record reference
Alternative
[Admin wants to update the record]
Update desired record - include new information
Update record instance
[Else admin wants to delete the record]
Delete desired record
Delete record instance
Alternative
[if database action is successfull]
Action successful
Action successful
[Else]
Action Failed
Action failed


Student Sign Up
Ross McAllister | January 2, 2024
Student
Pathfinder Webpage
Pathfinder Server
Pathfinder Database
Admin
Press Sign Up
Call admin email
Get admin email
Return admin email
Display page with admin email
enter students details and press sign up
Send form to admin
Email admin with students details

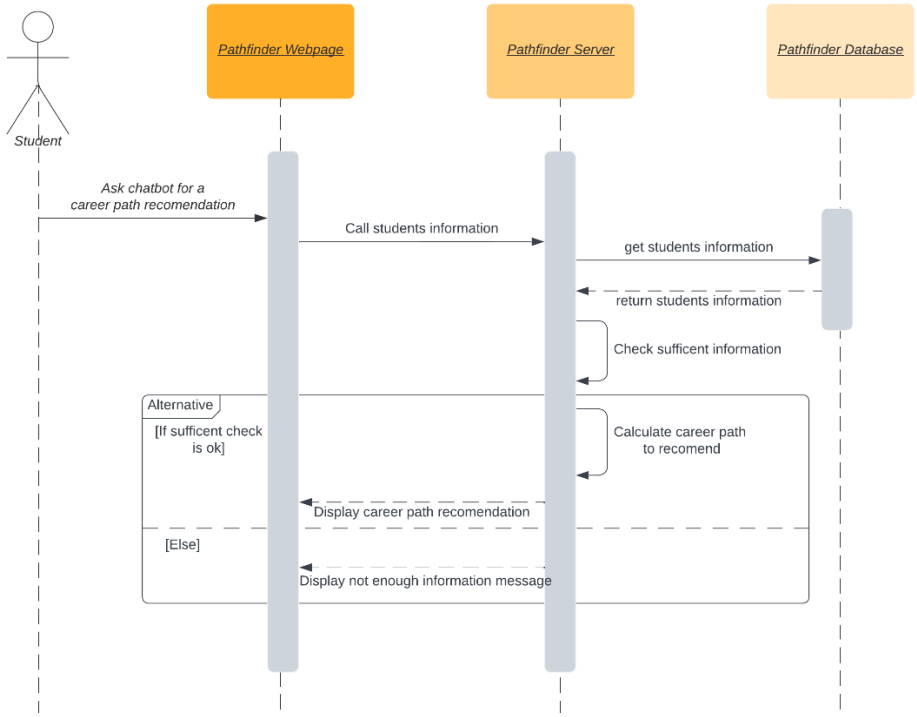
Check academic information

Ross McAllister | January 2, 2024



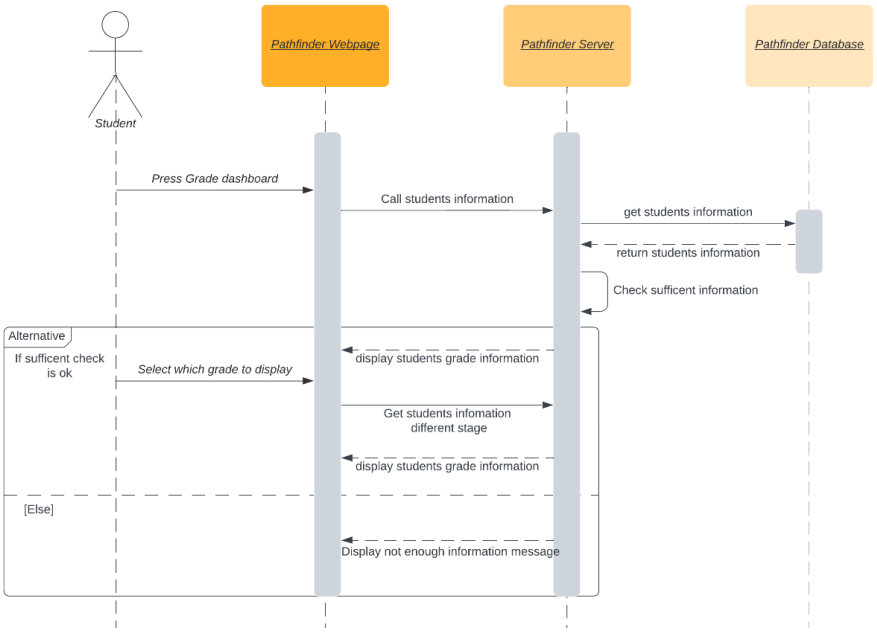
Career Path Recomendation

Ross McAllister | January 2, 2024



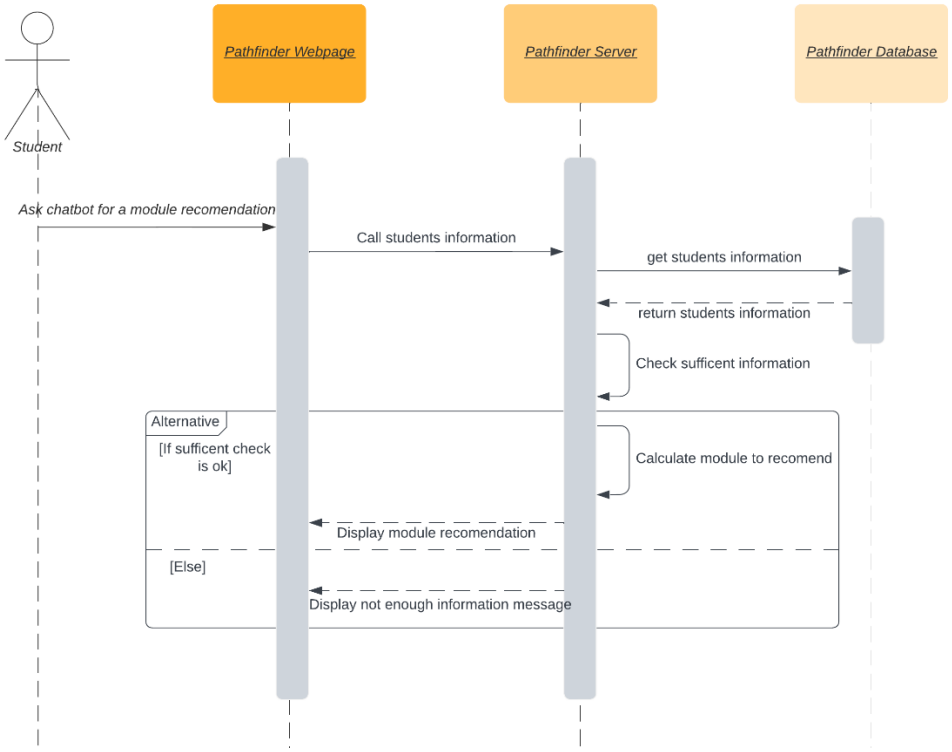
Get Course Stats

Ross McAllister | January 2, 2024



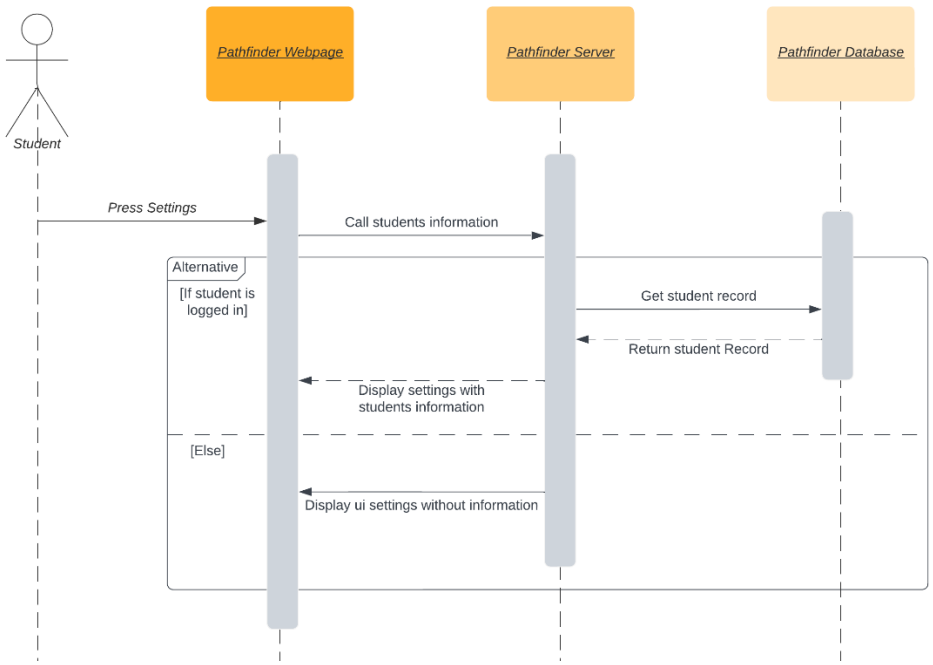
Module Recomendation

Ross McAllister | January 2, 2024



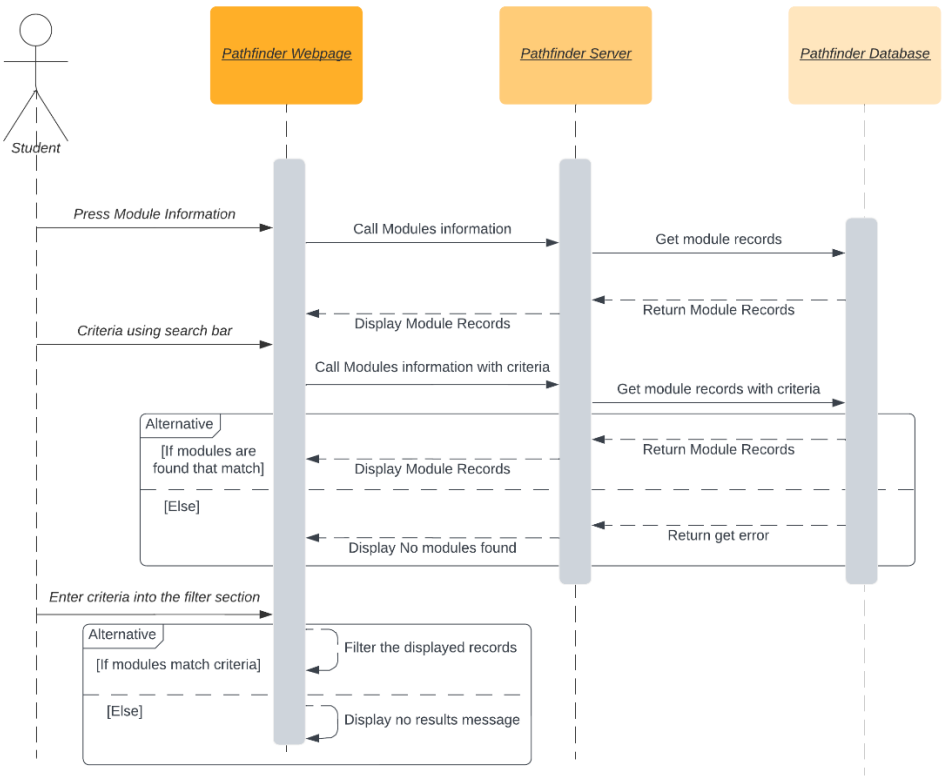
View Settings

Ross McAllister | January 2, 2024



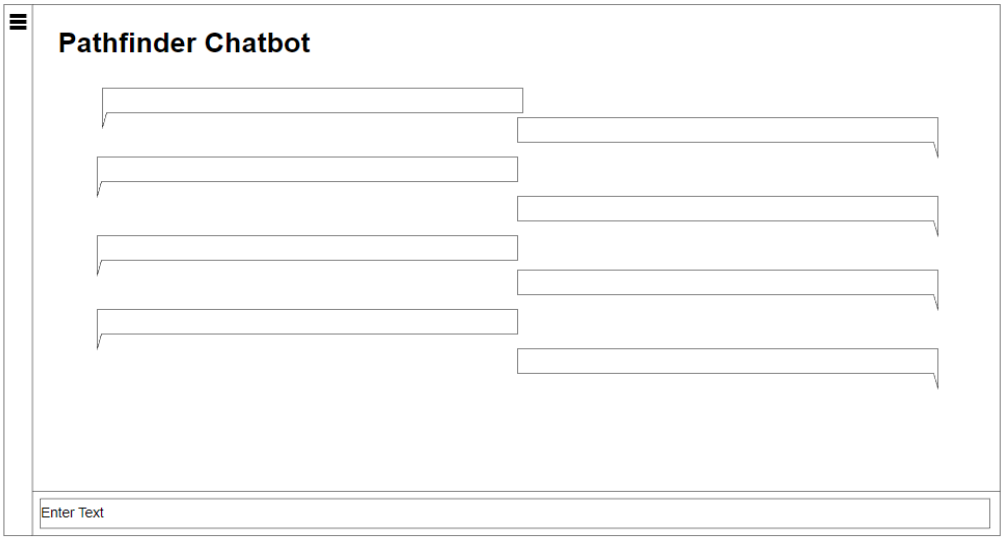
View Module Information

Ross McAllister | January 2, 2024

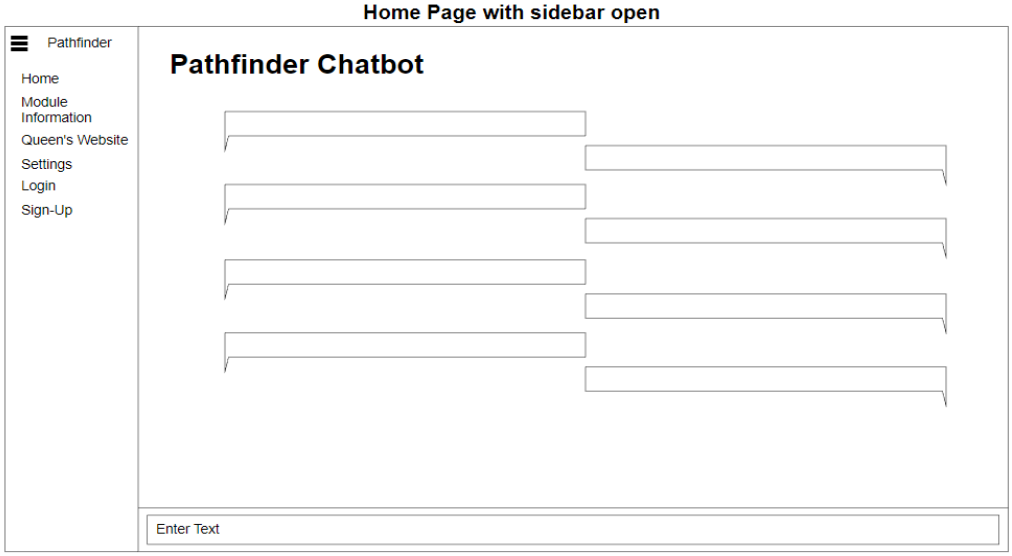


3.3.1 UI Wireframes

Home Page with sidebar closed



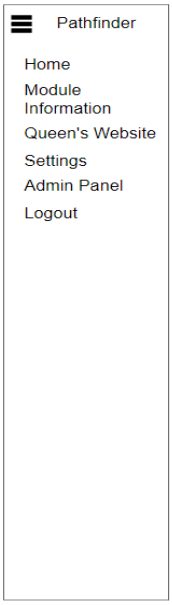
Home Page with sidebar open



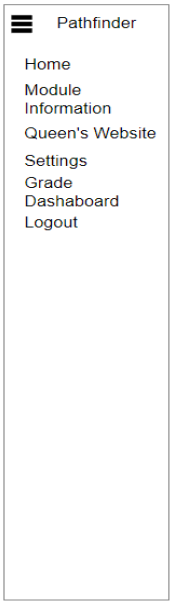
Examples of what each of the side bars (navigation bar) look like depending

When user is not logged in

Admins side bar

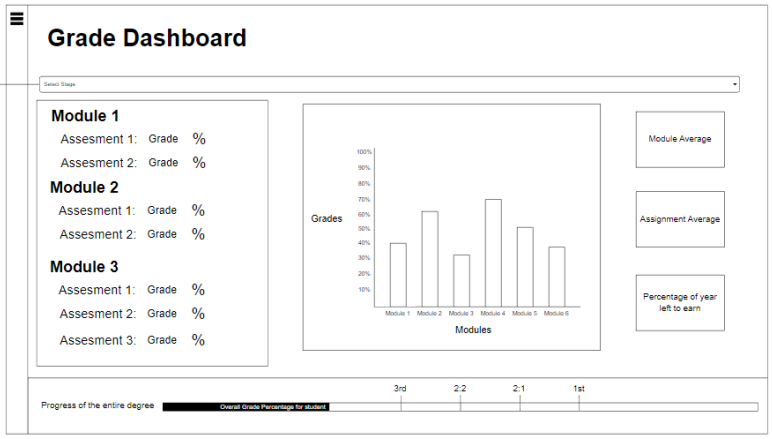


Student side bar

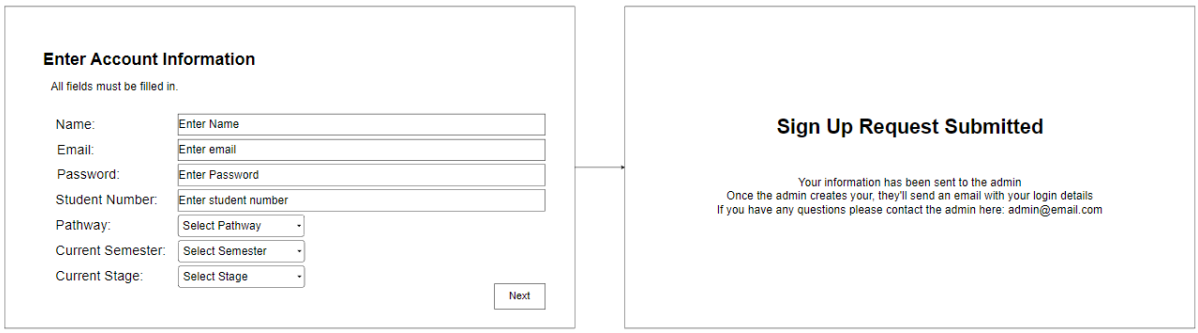


Example of what the stage selector dropdown will look like. This will change the number of stages displayed based on the students current stage

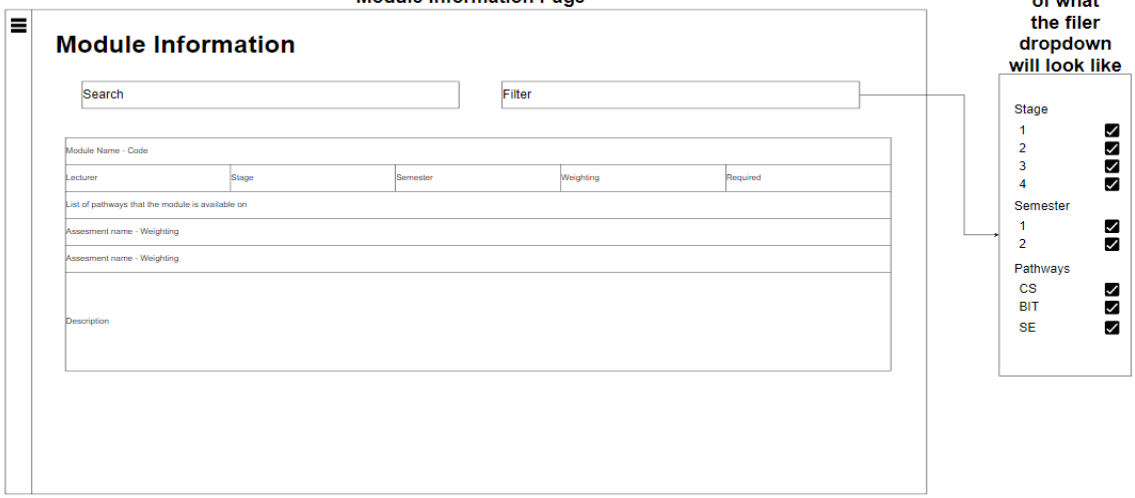
Grade Dashboard



Sign Up Pop Up



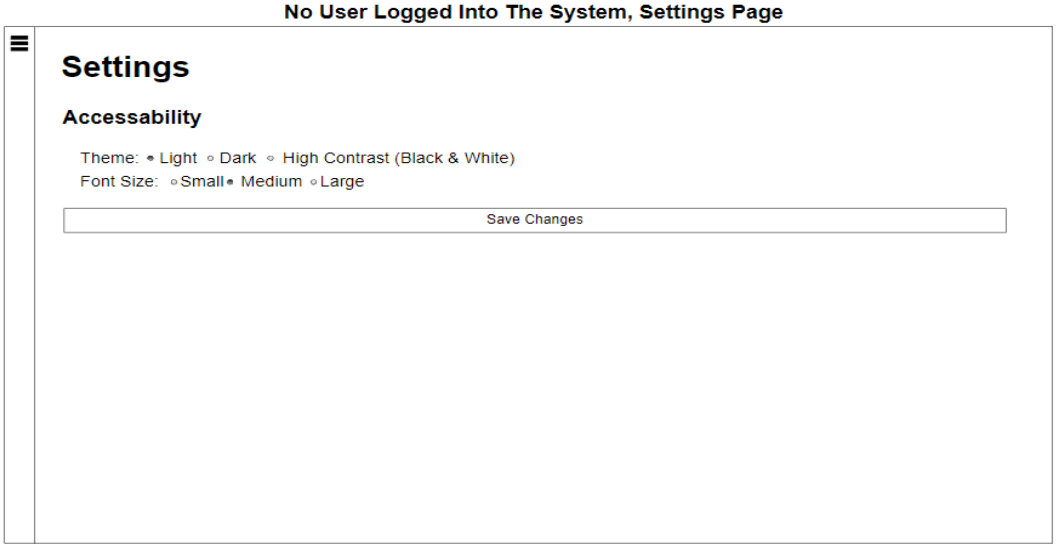
Module Information Page



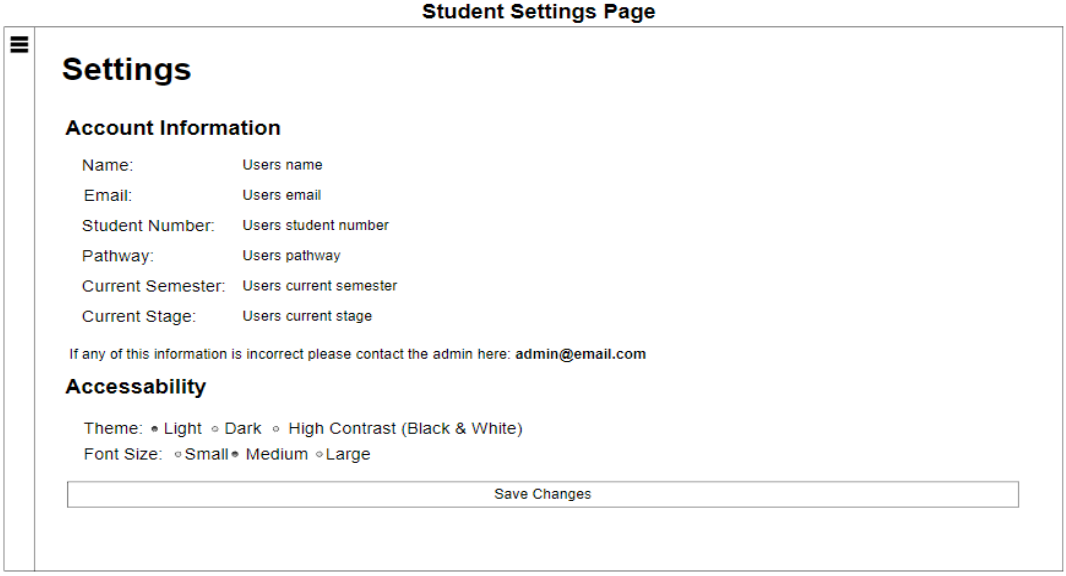
Example of what the filer dropdown will look like

(Note: The table shown for the module will be repeated based on the modules shown)

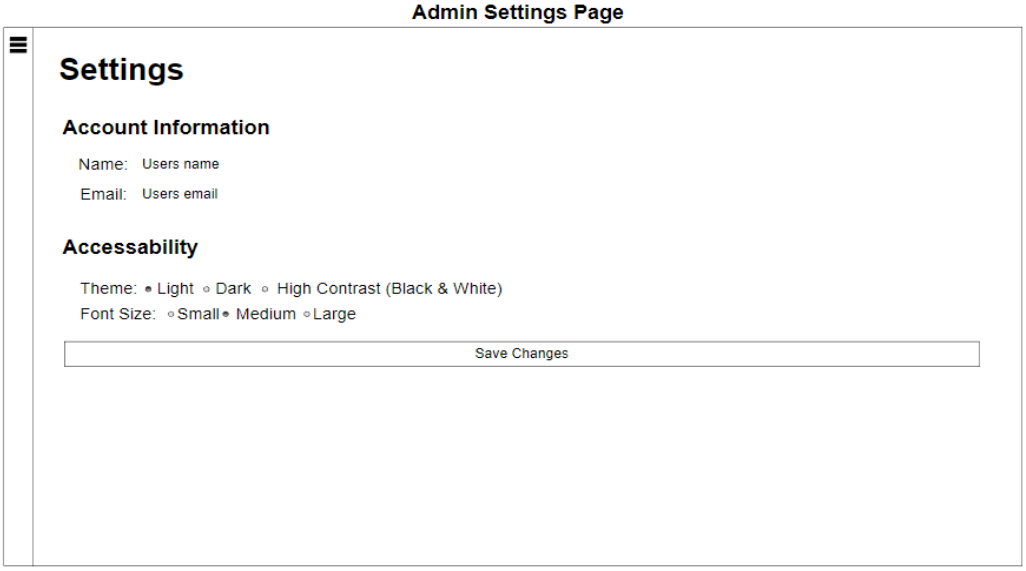
No User Logged Into The System, Settings Page



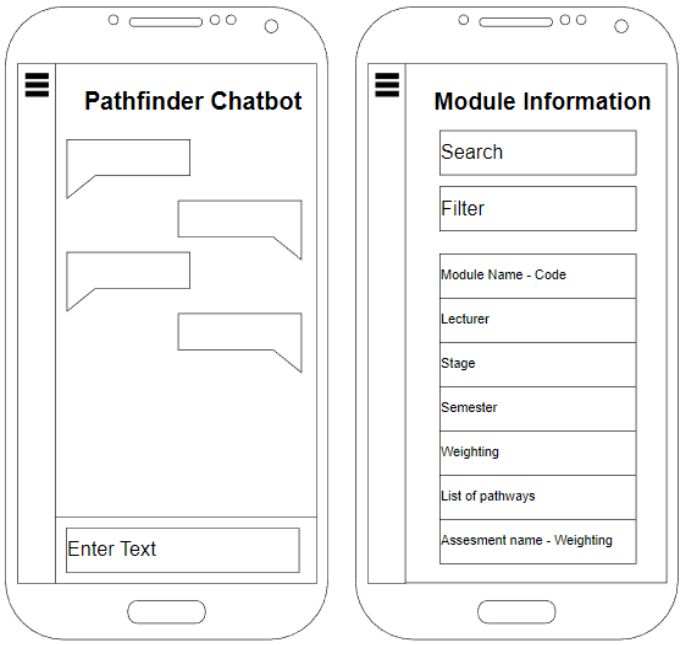
Student Settings Page

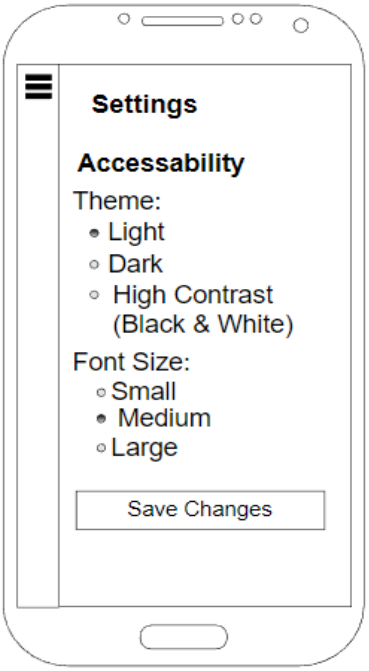


Admin Settings Page

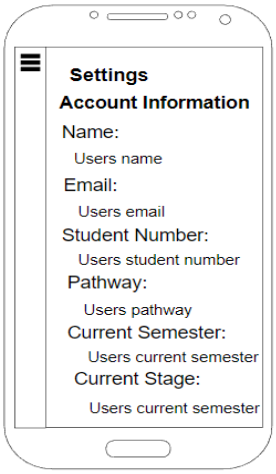


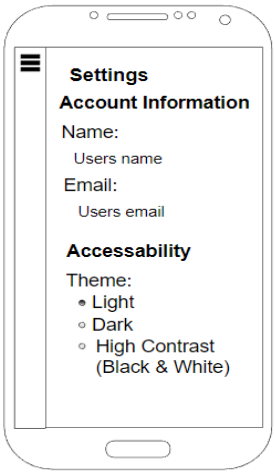
Mobile Wireframes
(Note: pop ups, dropdowns and side bar remain the same)





(Note: Accessability options will still be on these pages for admin and student, they just have to scroll further down for them,wireframe for thse can be seen for the none logged in user view of the settings page)





4.2.1 Generate QR Code Snippet

```
"""
    @Author: @DeanLogan
    @Description: Generates a QR code image from a given URL.
    @param: url - The URL to encode in the QR code.
    @return: Base64-encoded image data of the QR code.
"""
def generateQRCode(url):
    # Create a QR code instance with specified parameters
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )

    # Add the URL data to the QR code
    qr.add_data(url)

    # Generate the QR code image and fit it
    qr.make(fit=True)

    # Create an image with specified fill and background colors
    img = qr.make_image(fill_color="black", back_color="white")

    # Create an in-memory buffer for the image
    buffer = io.BytesIO()
    img.save(buffer)

    # Seek to the beginning of the buffer and encode the image in Base64
    buffer.seek(0)
    base64_image = base64.b64encode(buffer.read()).decode("utf-8")

    return base64_image  # Return the Base64-encoded image data
```

4.2.2 Display QR Code Snippet

```
"""
    @Author: @DeanLogan
    @Description: Displays a QR code image containing a secret key.
    @param: request - The HttpRequest object containing metadata about the request.
    @return: HttpResponse containing the QR code image in PNG format.
"""
def displayQRCode(request):
    try:
                        return     HttpResponse(generateQRCode(createSecretKey(request)),
content_type="image/png")
    except:
        return HttpResponse("Error generating QR code")
```

4.2.3 Check if 2FA is Enabled for a Given Account Code Snippet

```
"""
    @Author: @DeanLogan
    @Description: Checks if a user has two-factor authentication (TOTP) enabled.
     @param: request - The HttpRequest object containing the user's authentication information.
    @return: JsonResponse indicating whether two-factor authentication is enabled ('true' or 'false').
"""
def hasTwoFactorEnabled(request):
    if request.method == 'POST':
        try:
            # Attempt to authenticate the user with provided username and password
            form = CustomLoginForm(request, data=request.POST)
             authenticatedUser = authenticate(request, username=form['username'].value(), password=form['password'].value())

            # Check if the authenticated user has any TOTP devices and if at least one is confirmed
            totpDevices = TOTPDevice.objects.filter(user=authenticatedUser)
            for device in totpDevices:
                if device.confirmed:
                      return JsonResponse({'hasTwoFactorEnabled': 'true'})  # Two-factor authentication is enabled
        except TOTPDevice.DoesNotExist:
            pass

       return JsonResponse({'hasTwoFactorEnabled': 'false'})  # Two-factor authentication is not enabled

    else:
       return JsonResponse({'errors': 'Invalid request method'})  # Invalid request method (not POST)
```

4.2.4 Verify 2FA Code Supplied by the User Code Snippet

```
"""
    @Author: @DeanLogan
     @Description: Verifies a TOTP code against the TOTP devices associated with an authenticated user.
    @param: authenticatedUser - The authenticated user whose TOTP devices will be checked.
    @param: request - The HttpRequest object containing the TOTP code to verify.
    @return: True if the TOTP code is valid for any of the user's TOTP devices, False otherwise.
"""
def verifyTotpCode(authenticatedUser, request):
    try:
         totpDevice = TOTPDevice.objects.filter(user=authenticatedUser) # Get all TOTP devices associated with the authenticated user
        code = request.POST['code'].replace(" ", "") # Remove spaces from the TOTP code (if any
        for device in totpDevice:
            if device.verify_token(code):
                return True # TOTP code is valid for this device
    except TOTPDevice.DoesNotExist:
        # Handle the case where the user doesn't have a TOTP device
        pass
    return False # TOTP code is not valid for any of the user's TOTP devices
```

4.2.5 Verify Form Submission Code Snippet

```
"""
    @Author: @DeanLogan
    @Description: Verifies user authentication, including two-factor authentication (2FA) if enabled.
     @param: request - The HttpRequest object containing user authentication and 2FA information.
    @return: JsonResponse indicating whether the user is logged in ('true' or 'false') and any errors.
"""
def verify(request):
    if request.method == 'POST':
        form = CustomLoginForm(request, data=request.POST)

        # Check if the form is valid; if not, the user may have failed the captcha
        if form.is_valid():
            # Attempt to authenticate the user using provided username and password
                                    authenticatedUser = authenticate(request, username=form.cleaned_data['username'], password=form.cleaned_data['password'])

            # Check if 2FA is enabled for the user and verify the TOTP code if necessary
                        if json.loads(hasTwoFactorEnabled(request).content.decode('utf-8'))['hasTwoFactorEnabled'] == 'true' and verifyTotpCode(authenticatedUser, request) is False:
                return JsonResponse({'loggedIn': 'false', 'errors': 'Failed 2FA, wrong code'})

            # Log in the user if authentication is successful
            login(request, authenticatedUser)

            # Set session timeout based on 'remember_me' value
            if form.cleaned_data.get('remember_me', False):
                request.session.set_expiry(1209600)  # Two weeks (in seconds)

            return JsonResponse({'loggedIn': 'true'})  # User is successfully logged in

        else:
           return JsonResponse({'loggedIn': 'false', 'errors': 'Failed Captcha, or invalid credentials'})  # Captcha or credentials validation failed

    else:
       return JsonResponse({'errors': 'Invalid request method'})  # Invalid request method (not POST)
```

4.3.1 Restore Backup From .dump File Code Snippet

```
"""
@Author: @DeanLogan
@Description: Restores a database backup from the specified file using Django's 'dbrestore'
management command.
@param: filePath - The path to the backup file to restore from.
"""
def restoreFromBackup(filePath):
    # Determine the platform (OS) and adjust the command accordingly
    if platform.system() == "Windows":
        print("windows")
        subprocess.run(" ".join([
            "python", "manage.py", "dbrestore",
            "-I",
            f'"{filePath}"',
            "--noinput",
            "--skip-checks"
        ]), shell=True)  # Run the command as a single string using shell (Windows)
    else:
        print("unix")
        subprocess.run([
            "python", "manage.py", "dbrestore",
            "-I",
            f"{filePath}",
            "--noinput",
            "--skip-checks"
        ])  # Run the command as a list (Linux, macOS)
    with connection.cursor() as cursor:
        cursor.execute("VACUUM;")
```

4.3.2 Connect To Container Client Code Snippet

```
"""
    @Author: @DeanLogan
    @Description: Retrieves or creates a Blob Storage container client using a connection string and container name.
    @return: A container client object for interacting with the specified container.
"""
def getContainerClient():
    try:
        blobServiceClient = BlobServiceClient.from_connection_string(CONNECTION_STRING) # Create a BlobServiceClient using the provided connection string
        containerClient = blobServiceClient.get_container_client(CONTAINER_NAME) # Get or create a container client using the specified container name

        # Check if the container exists; if not, create it
        if not containerClient.exists():
            containerClient.create_container()

       return containerClient # Return the container client object for further interaction
    except:
        return None
```

4.3.3 Delete Oldest Cloud Backup Code Snippet

```
"""
    @Author: @DeanLogan
    @Description: Deletes the oldest backup blob from the Azure Blob Storage container if the number of blobs exceeds a limit.
    @return: True if a blob was deleted successfully, False otherwise.
"""
def deleteOldestBackupBlob():
     containerClient = getContainerClientWithTimeout(0.4) # Get a container client, if there is no response in 0.4 secs containerClient is None
    try:
        blobList = containerClient.list_blobs()

       backupFiles = [(blob.name, blob.last_modified) for blob in blobList] # Convert the blob list to a list of (blob_name, last_modified) tuples
        print(f"num of files: {len(backupFiles)}")
        if len(backupFiles) >= 20:
           backupFiles.sort(key=lambda x: x[1]) # Sort backup files by last_modified time (oldest first)

            # Delete the oldest backup blob
            oldestBackupName = backupFiles[0][0]
            return deleteBlob(oldestBackupName)
        return True
    except Exception as e:
        backupStatusEmail("Oldest backup blob deletion of cloud backup failed", True, e)
        return False
```

4.3.4 Delete Oldest Local Backup Code Snippet

```
"""
    @Author: @DeanLogan
    @Description: Deletes the oldest local backup file if the number of existing backup
files exceeds a limit.
    @return: True if a backup file was deleted successfully, False otherwise.
"""
def deleteOldestBackupFile():
    try:
            backupFiles  =  glob.glob(os.path.join(DBBACKUP_STORAGE_OPTIONS['location'],
'*.dump')) # Check the number of existing backup files
        if len(backupFiles) > 10:
             backupFiles.sort(key=os.path.getmtime) # Sort backup files by modification
time (oldest first)

            # Delete the oldest backup file
            oldestBackup = backupFiles[0]
            os.remove(oldestBackup)
            deleteOldestBackupFile()
        backupStatusEmail("Oldest backup file deleted successfully")
        return True
    except Exception as e:
        backupStatusEmail("Oldest backup file deletion failed", True, e)
        return False
```

4.4.1 Extract Job Listings from Indeed Search Page Code Snippet

```
    '''
        @Author: @DeanLogan
          @Description: Parses the search result pages for job listings and handles
pagination.
        @param: response - The response object containing the search result page data.
        @return: Yields requests for individual job pages and further search result pages.
    '''
    def parseSearchResults(self, response):
        # Callback for handling the search result pages
        location = response.meta['location']
        keyword = response.meta['keyword']
        offset = response.meta['offset']

        # Extract JSON data using regex from the page source
              scriptTag   =   re.findall(r'window.mosaic.providerData\["mosaic-provider-
jobcards"\]=(\{.+?\});', response.text)

        if scriptTag is not None:
            jsonBlob = json.loads(scriptTag[0])
            # Paginate Through Jobs Pages
            if offset == 0:
                # Get total number of job results
               metaData                                                              =
jsonBlob["metaData"]["mosaicProviderJobCardsModel"]["tierSummaries"]
                numResults = sum(category["jobCount"] for category in metaData)
                if numResults > 1000:
                    numResults = 50
                # Generate requests for pagination
                for offset in range(10, numResults + 10, 10):
                    url = self.getIndeedSearchUrl(keyword, location, offset)
                        yield  scrapy.Request(url=url,  callback=self.parseSearchResults,
meta={'keyword': keyword, 'location': location, 'offset': offset})
            # Extract Jobs From Search Page
            jobsList = jsonBlob['metaData']['mosaicProviderJobCardsModel']['results']
            for index, job in enumerate(jobsList):
                if job.get('jobkey') is not None:
                      jobUrl                                                          =
'https://www.indeed.com/m/basecamp/viewjob?viewtype=embedded&jk=' + job.get('jobkey')
                    yield scrapy.Request(url=jobUrl,
                            callback=self.parseJob,
                            meta={
                                'keyword': keyword,
                                'location': location,
                                'page': round(offset / 10) + 1 if offset > 0 else 1,
                                'position': index,
                                'jobKey': job.get('jobkey'),
                            })
```

4.4.2 Calculate Similarity Code Snippet

```
# Function to calculate similarity between a module and a career. - Conor Nugent
def calculate_similarity(module, career):
    # Define a set of stop words to be excluded from analysis.
      stop_words  =  set(["to",  "and",  "of",  "in",  "for",  "the","this","an",  "on",
"with","is","a","into","will","as","their","use","or","at","all","how"])

    # Tokenize and clean module and career information.
    module_name_tokens = tokenize_string(module.moduleName)
    module_desc_tokens = tokenize_string(module.moduleDescription)
    job_desc_tokens = tokenize_string(career.jobDescription)

    # Remove stop words from the tokens.
    module_name_tokens -= stop_words
    module_desc_tokens -= stop_words
    job_desc_tokens -= stop_words

    # Combine tokens from module name and description.
     module_tokens  =  module_name_tokens.union(module_desc_tokens)  #  The  union  method
returns a set that contains all the items from both sets, without duplicates.

    # Calculate the similarity score based on common tokens.
     score  =  len(module_tokens.intersection(job_desc_tokens)) # The intersection method
returns a set containing all the items that are present in both sets.

    return score
```

4.5.1 Detecting Language on Greetings Code Snippet

```
    # created by Kyle McComb
    def detect_language_based_on_greetings(self, message):
        words = message.lower().split()
        for lang, greet_list in self.greetings.items():
            for greet in greet_list:
                if greet in words:
                    return lang
        return None
```

4.5.2 Language Responses Dictionary Code Snippet

```
language_responses = {
    'en': None,
    'es': f"Lo siento, este chatbot solo admite inglés. Por favor, use <a href='{base_url.format('es')}' target='_blank'>Google Translate</a> para comunicarse en inglés.",
    'fr': f"Désolé, ce chatbot ne prend en charge que l'anglais. Veuillez utiliser <a href='{base_url.format('fr')}' target='_blank'>Google Translate</a> pour communiquer
    en anglais.",
    'de': f"Entschuldigung, dieser Chatbot unterstützt nur Englisch. Bitte verwenden Sie <a href='{base_url.format('de')}' target='_blank'>Google Translate</a> um auf
    Englisch zu kommunizieren.",
    'zh-cn': f"对不起，此聊天机器人仅支持英语。 请使用<a href='{base_url.format('zh-CN')}' target='_blank'>Google翻译</a>与英文交流。",
    'zh-tw': f"對不起，此聊天機器人僅支持英語。 請使用<a href='{base_url.format('zh-TW')}' target='_blank'>Google翻譯</a>與英文交流。",
    'ms': f"Maaf, chatbot ini hanya menyokong Bahasa Inggeris. Sila gunakan <a href='{base_url.format('ms')}' target='_blank'>Google Translate</a> untuk berkomunikasi dalam
    Bahasa Inggeris.",
    'ur': f"معاف کیجئے، یہ چیٹ بوٹ صرف انگریزی زبان کو سمجھتا ہے۔ براہ کرم <a href='{base_url.format('ur')}' target='_blank'>گوگل ترجمہ</a> استعمال کریں۔",
    'hi': f"मुझे खेद है, यह चैटबॉट केवल अंग्रेजी समर्थन करता है। कृपया <a href='{base_url.format('hi')}' target='_blank'>Google अनुवाद</a> का उपयोग करें।",
    'yo': f"Mo dupe, chatbot yii ṣe aṣaǫkan gege bi English nikan. Jowo lo <a href='{base_url.format('yo')}' target='_blank'>Google Translate</a> lati sọrọ ni ede Geesi.",
    'bn': f"দুঃখিত, এই চ্যাটবটটি কেবল ইংরেজি সমর্থন করে। দয়া করে <a href='{base_url.format('bn')}' target='_blank'>Google অনুবাদ</a> ব্যবহার করতে ইংরেজিতে যোগাযোগ করুন।",
    'it': f"Mi dispiace, questo chatbot supporta solo l'inglese. Si prega di utilizzare <a href='{base_url.format('it')}' target='_blank'>Google Translate</a> per
    comunicare in inglese."
}
```

5.1.1 Frontend Functional Testing Results

Test results of the tests conducted during the Manual Scripted Testing Phase					
Function Under Test	**Number of Passed Test Cases (excluding retests)**	**Number of Failed Test Cases**	**Failed Test Case #**	**Number of Successful retests**	**Total number of test cases per function**
Chatbot user text	2	0	N/A	2	2
Admin database access	8	0	N/A	8	8
Grade dashboard display data	3	2	GradeDashboard-001-01 GradeDashboard-001-02	5	5
Accessibility settings	8	0	N/A	8	8
Chatbot GUI elements	2	1	GUI-005-03	3	3
Login GUI elements	4	3	GUI-Login-003-01 GUI-Login-004 GUI-Login-005	7	7
Displaying correct database tables	7	2	database-005-01 database-008-01	9	9
Database backups functionality	8	8	database-011-01 Backup-001 Backup-002 Backup-003 Backup-004 Backup-005 Backup-006 Backup-007	7	8
Student data access	3	0	N/A	3	3
Cross-Browser and Cross-Platform tests	3	8	website-001-04 website-001-05 website-001-06 website-002-01 website-002-02 website-002-03 website-003-01 website-004-01	11	11
Module information page	6	2	ModuleInformation-001-01 ModuleInformation-001-03	6	8
Signup page	1	1	SignUp-002-01	2	2

Test Case ID: Backup-001-01
Test Case Description: (Testing requirement Backup-001) Verify the database backup is stored locally on the admin's computer and can be restored as saved.
Test Steps: 1. Initiate a database backup process. 2. Verify the backup file is created on the admin's local computer. 3. Restore the database from the backup file. 4. Check if the restored database matches the original database exactly.
Expected Outcome: The database backup should be successfully created and stored locally on the admin's computer. Upon restoration, the database should match the original database in all aspects.
Actual Outcome: The database backup was successfully created and stored locally on the admin's computer. The restored database matched the original database precisely.
Pass/Fail?: Pass

Test Case ID: Backup-002-01
Test Case Description: (Testing requirement Backup-002) Verify the database backup can be stored in the cloud and restored as saved.
Test Steps: 1. Initiate a database backup process to the cloud. 2. Verify the backup file is created in the cloud storage. 3. Restore the database from the cloud backup file. 4. Check if the restored database matches the original database exactly.
Expected Outcome: The database backup should be successfully created and stored in the cloud. Upon restoration, the database should match the original database in all aspects.
Actual Outcome: The database backup was successfully created and stored in the cloud. The restored database matched the original database precisely.
Pass/Fail?: Pass

Test Case ID: Backup-003-01
Test Case Description: (Testing requirement Backup-003) Verify the admin can create a database backup at any time.
Test Steps: 1. Log in as the admin. 2. Attempt to initiate a database backup at different times, including outside of usual business hours. 3. Confirm that the backup process starts each time.
Expected Outcome: The admin should be able to initiate the database backup process at any time without restrictions.
Actual Outcome: The admin was able to successfully initiate the database backup process at various times, including outside of normal business hours.
Pass/Fail?: Pass

Test Case ID: Backup-004-01
Test Case Description: (Testing requirement Backup-004) Verify the admin can fully restore the system from a local or cloud backup.
Test Steps: 1. Create a backup of the current database and store it both locally and in the cloud. 2. Wipe the current database or simulate a database failure. 3. Attempt to restore the database from both the local and cloud backups. 4. Verify the integrity and completeness of the restored database against the original.
Expected Outcome: The system should be fully restored from both the local and cloud backups, with the restored database matching the original.
Actual Outcome: The system was successfully restored from both the local and cloud backups, and the restored databases matched the original in all aspects.
Pass/Fail?: Pass

Test Case ID: Backup-005-01
Test Case Description: (Testing requirement Backup-005) Verify the admin can roll back the system from a local or cloud backup.
Test Steps: 1. Create a backup of the current database and store it both locally and in the cloud. 2. Make significant changes to the current database. 3. Roll back the changes by restoring the database from the backups. 4. Compare the rolled-back database to the state before the changes were made.
Expected Outcome: The database should be successfully rolled back to its previous state using both local and cloud backups.
Actual Outcome: The database was successfully rolled back to its previous state using both local and cloud backups. The rolled-back database matched the state before changes were made.
Pass/Fail?: Pass

Test Case ID: Backup-006-01
Test Case Description: (Testing requirement Backup-006) Verify the admin can delete a backup from a local or cloud storage.
Test Steps: 1. Create backups of the database and store them locally and in the cloud. 2. Attempt to delete these backups from both the local and cloud storage. 3. Confirm that the backups are no longer available in the storage locations.
Expected Outcome: The admin should be able to successfully delete the backups from both local and cloud storage.
Actual Outcome: The admin successfully deleted the backups from both local and cloud storage, and they were no longer available.
Pass/Fail?: Pass

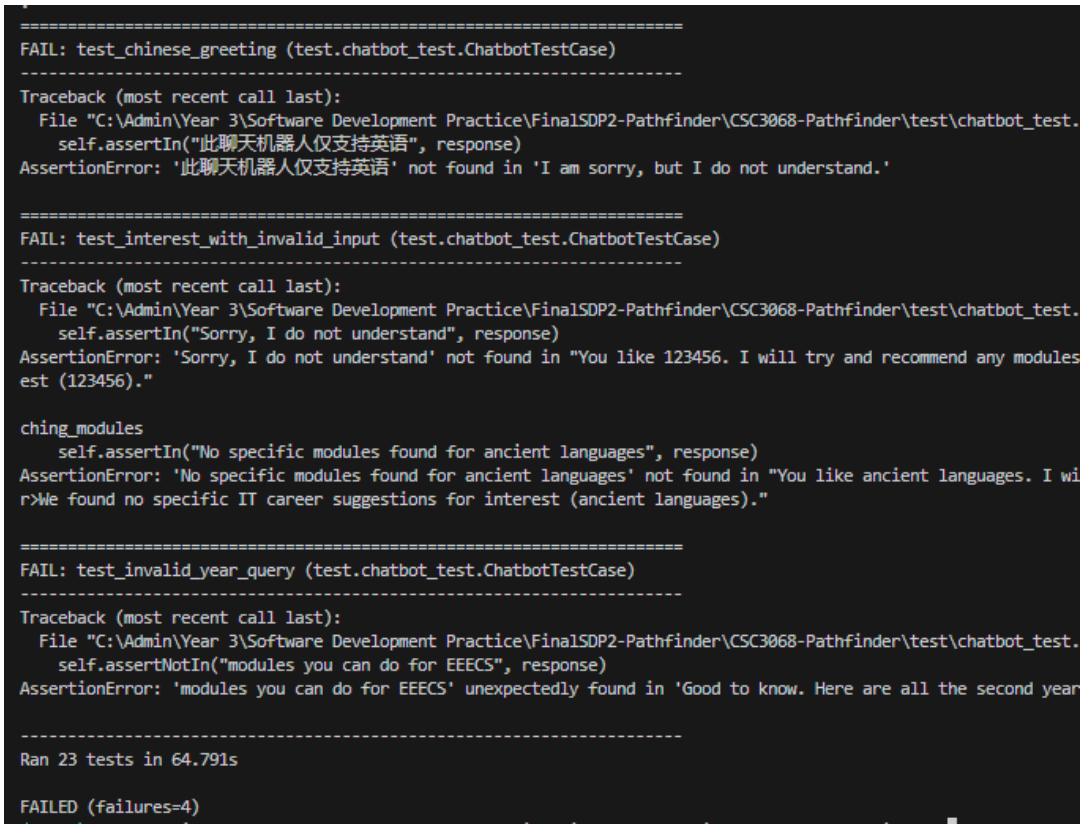
Test Case ID: Backup-007-01
Test Case Description: (Testing requirement Backup-007) Verify the admin is alerted via email for backup status updates.
Test Steps: 1. Initiate a backup process. 2. Monitor the admin's email for notifications regarding the backup process. 3. Confirm receipt of emails indicating either success or failure of the backup process.
Expected Outcome: The admin should receive email notifications regarding the status of the backup process (success or failure).
Actual Outcome: The admin received email notifications indicating the status of the backup process as it progressed, including success and failure notices.
Pass/Fail?: Pass

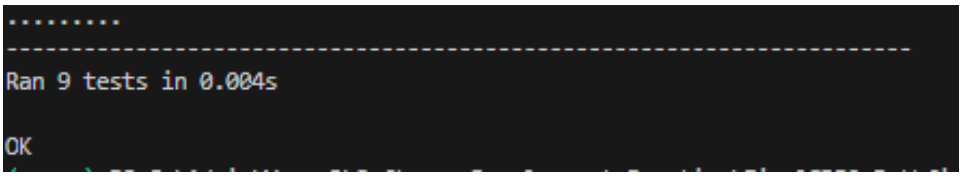
5.3.1 Backend Functional Testing Results

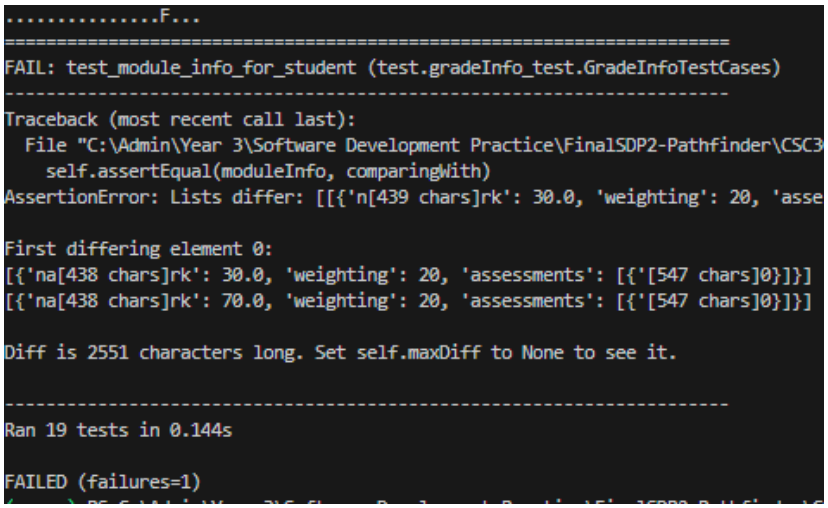
Function Under Test	Number of Passed Test Cases	Number of Failed Test Cases	Failed Test Case #	Total number of test cases per function
Chatbot Interest tests	6	2	test_interest_with_invalid_input test_interest_with_no_matching_modules	8
Chatbot Language tests	7	1	test_chinese_greeting	8
Chatbot Year tests	7	1	test_invalid_year_query	8
Database tables tests	9	0	N/A	9
Grade info - average mark tests	5	0	N/A	5
Grade info - module tests	2	1	test_module_info_for_student	3
Grade info – assessment tests	4	0	N/A	4
Grade info – more validation tests	7	0	N/A	7

- Chatbot Unit Tests link - https://github.com/KyleMcComb/Pathfinder/blob/master/test/chatbot_test.py
- Database Unit Tests link - https://github.com/KyleMcComb/Pathfinder/blob/master/test/database_test.py
- Grade Info Unit Tests link - https://github.com/KyleMcComb/Pathfinder/blob/master/test/gradeInfo_test.py

5.3.2 Backend Functional Testing Screenshots







5.6.1 Security Testing Results

Function Under Test	Number of Passed Test Cases	Number of Failed Test Cases	Failed Test Case #	Total number of test cases per function
Login credentials tests	2	0	N/A	2
User access levels tests	2	0	N/A	2
Database encryption tests	1	0	N/A	1
Remember me test	1	0	N/A	1
CAPTCHA test	1	0	N/A	1
Two-Factor Authentication tests	3	0	N/A	3

Test Case ID: Security-001-01
Test Case Description: (Testing requirement GUI-Login-004) Verify if Captcha is displayed on the login page
Test Steps: 1. Navigate to the login page of the system. 2. Observe if a Captcha challenge is present on the page.
Expected Outcome: A Captcha challenge-response test should be visibly present on the login page.
Actual Outcome: A Captcha challenge-response test was visibly present on the login page.
Pass/Fail?: Pass

Test Case ID: Security-001-02
Test Case Description: (Testing requirement Login-009) Test the functionality of the Captcha challenge
Test Steps: 1. Navigate to the login page of the system. 2. Complete the Captcha challenge. 3. Attempt to log in to the system.
Expected Outcome: The user should only be able to log in after successfully completing the Captcha challenge.
Actual Outcome: The user was able to log in only after successfully completing the Captcha challenge.
Pass/Fail?: Pass

Test Case ID: Security-002-01
Test Case Description: (Testing requirement GUI-Login-005) Testing the input field for multi-factor authentication code for users with multi-factor authentication enabled
Test Steps: 1. Navigate to the login page. 2. Enter valid credentials for a user account that has multi-factor authentication enabled. 3. Observe if a prompt appears requesting a multi-factor authentication code. 4. Enter the correct multi-factor authentication code provided to the user. 5. Attempt to complete the login process.
Expected Outcome: A prompt for a multi-factor authentication code should appear after entering valid credentials. The user should be able to input the multi-factor authentication code. After entering the correct code, the login process should complete successfully, granting access to the user account.
Actual Outcome: Upon entering valid credentials for an account with multi-factor authentication enabled, a prompt for the multi-factor authentication code appeared as expected. The correct multi-factor authentication code was entered, and the login process completed successfully, granting access to the user account.
Pass/Fail?: Pass

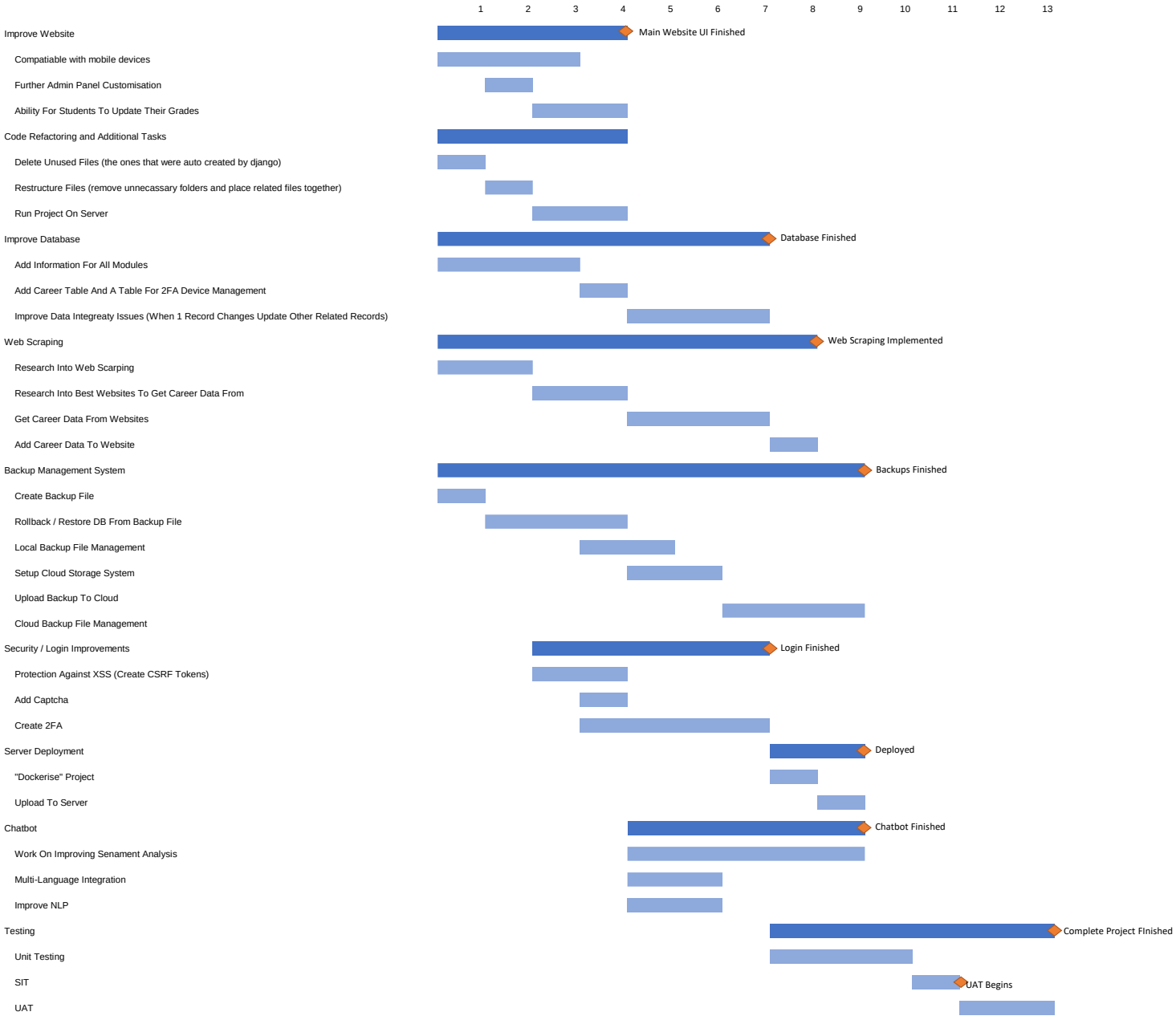
Test Case ID: Security-002-02
Test Case Description: (Testing requirement GUI-Login-005) Testing the absence of multi-factor authentication code prompt for users without multi-factor authentication enabled
Test Steps: 1. Navigate to the login page. 2. Enter valid credentials for a user account that does not have multi-factor authentication enabled. 3. Observe if the login process is completed without requesting a multi-factor authentication code.
Expected Outcome: No prompt for a multi-factor authentication code should appear after entering valid credentials. The login process should complete successfully without requiring a multi-factor authentication code, granting access to the user account.
Actual Outcome: After entering valid credentials for an account without multi-factor authentication enabled, the login process completed seamlessly without prompting for a multi-factor authentication code. The user was granted access to their account without any unnecessary steps.
Pass/Fail?: Pass

Test Case ID: Security-003-01
Test Case Description: (Testing requirement Login-006) Verify the ability to register a TOTP device to a user's account
Test Steps: 1. Log in to the admin account. 2. Navigate to the admin page for the TOTP device registration. 3. Follow the process to add a new TOTP device, including entering device details and confirming registration.
Expected Outcome: The user should be able to register a TOTP device to their account without any errors, and the device should be listed in their account settings.
Actual Outcome: The user successfully registered a TOTP device, and the device was listed in their account settings.
Pass/Fail?: Pass

Test Case ID: Security-003-02
Test Case Description: (Testing requirement Login-008) Test the functionality of the TOTP system during login
Test Steps: 1. Log out of the user account. 2. Attempt to log back in. 3. Upon prompt, enter the one-time passcode sent to the registered TOTP device.
Expected Outcome: After entering the correct one-time passcode from the registered device, the user should gain access to their account.
Actual Outcome: The user received a one-time passcode on their registered device and successfully accessed their account after entering the correct passcode.
Pass/Fail?: Pass

Test Case ID: Security-004-01
Test Case Description: (Testing requirement Login-007) Test login process for accounts without a registered TOTP device
Test Steps: 1. Log in to a user account that does not have a TOTP device registered. 2. Observe the login process, particularly whether an OTP is requested or not.
Expected Outcome: For accounts without a registered TOTP device, the system should allow the user to log in without requesting an OTP.
Actual Outcome: The user was able to log in without an OTP request, as no TOTP device was registered to the account.
Pass/Fail?: Pass

7.1.1 Detailed Roadmap



7.2.1 Additional Branches

Branches

New branch

Overview | Yours | Active | Stale | All

Search branches...

Default

Branch	Updated	Check status	Behind | Ahead	Pull request
master	Deleted 20 minutes ago		Default	

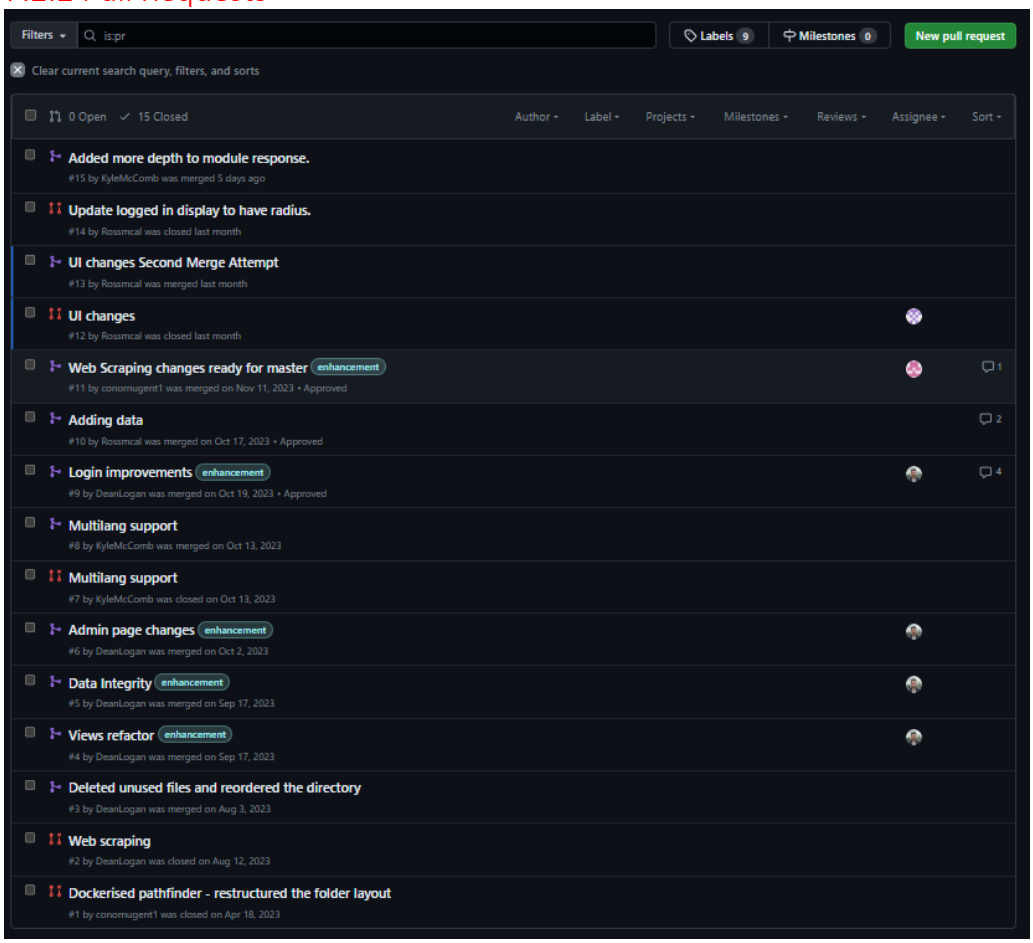
Your branches

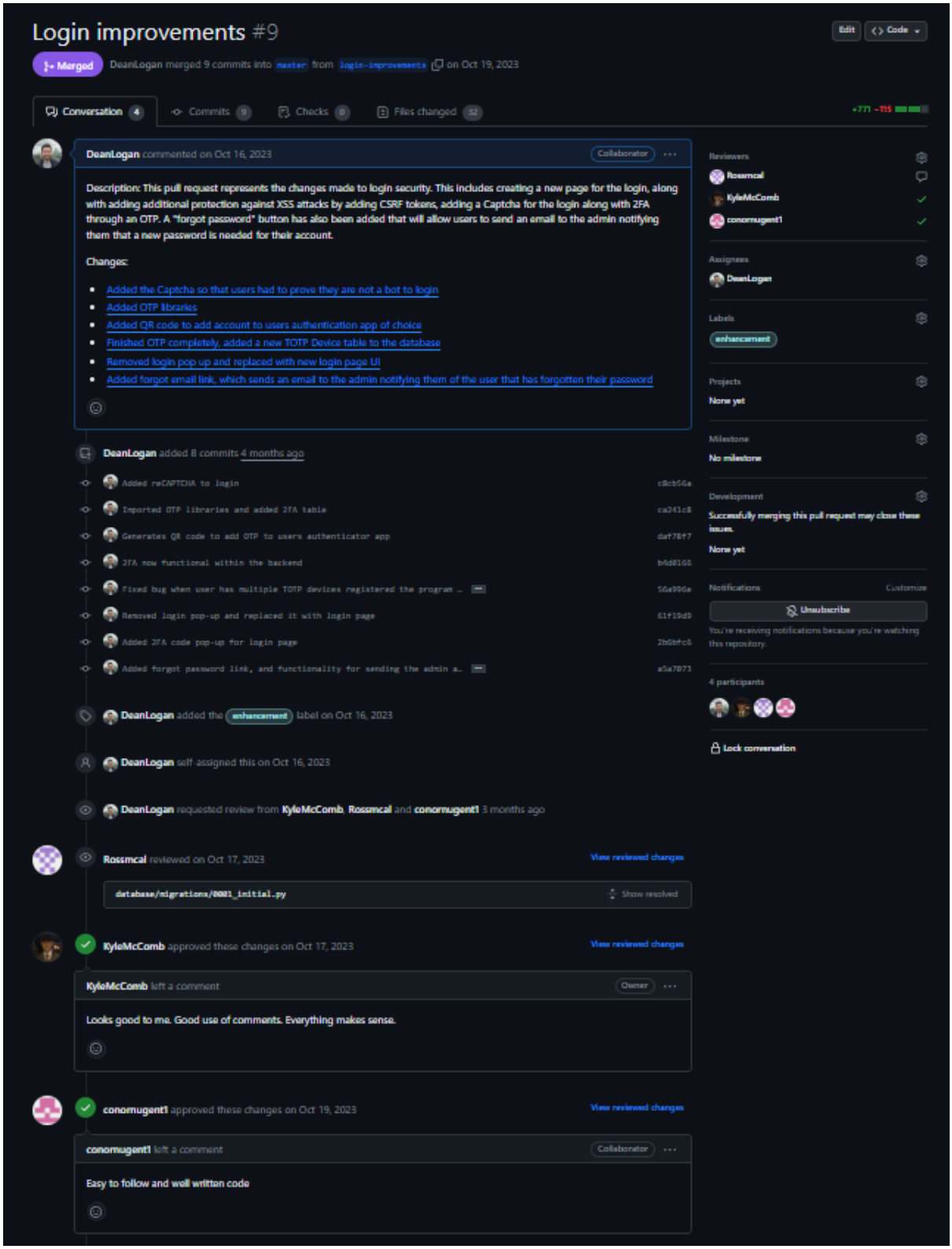
Branch	Updated	Check status	Behind | Ahead	Pull request
web-scraping	Deleted 2 months ago			

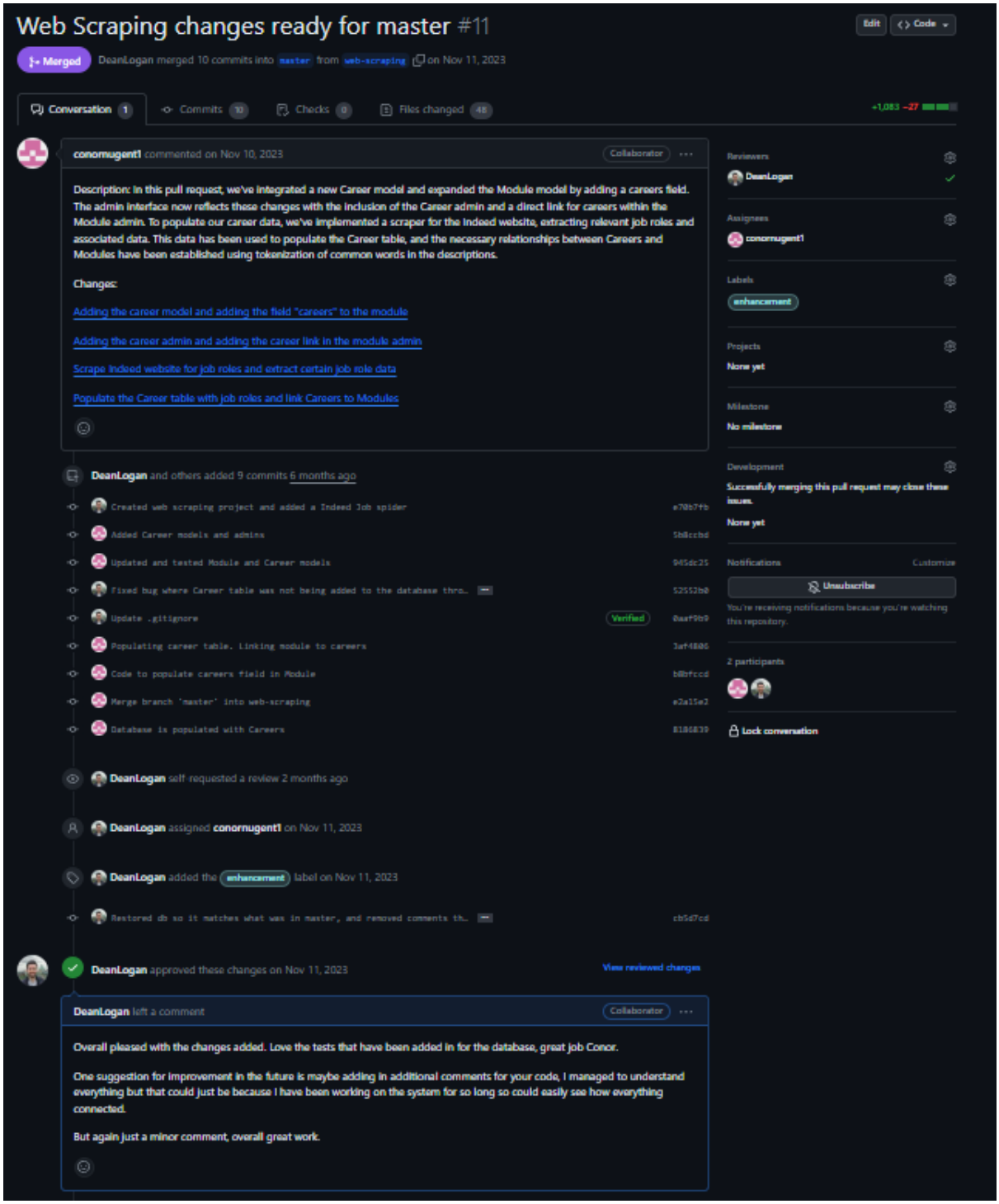
Active branches

Branch	Updated	Check status	Behind | Ahead	Pull request
server	Deleted 2 hours ago			
test-changes	Deleted 3 days ago			
ui-changes	Deleted 3 weeks ago			
web-scraping	Deleted 2 months ago			

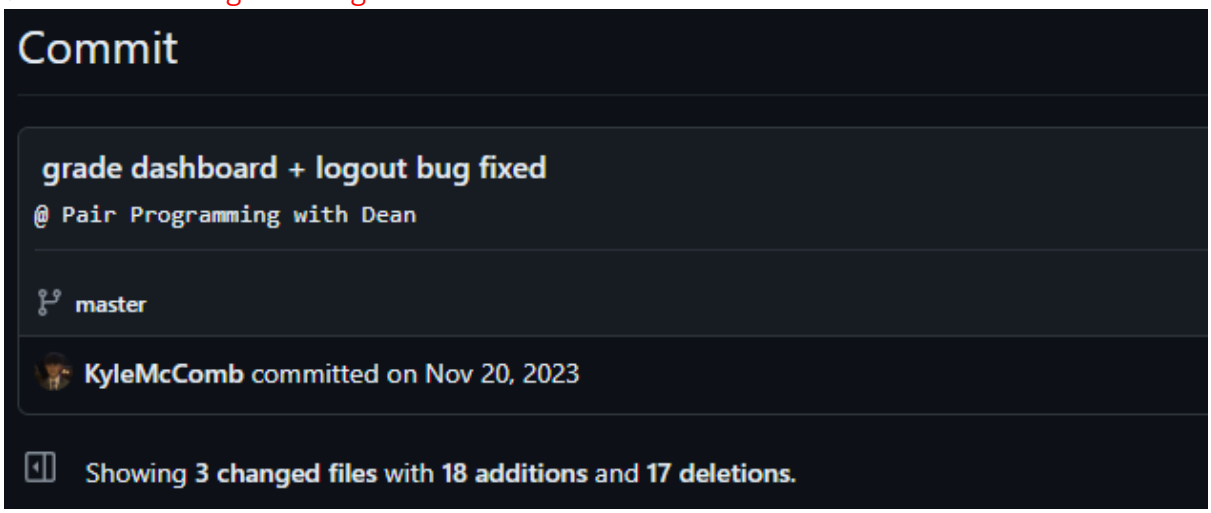
7.2.2 Pull Requests



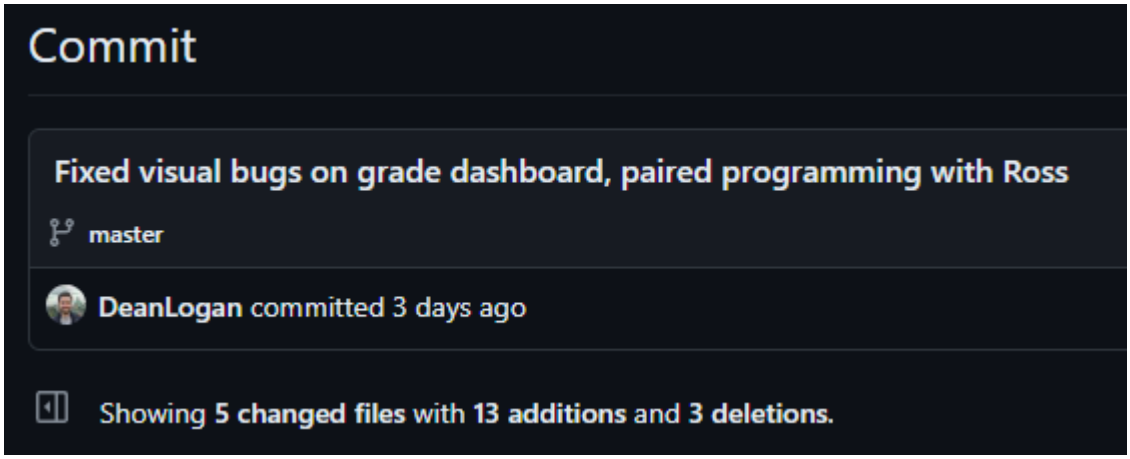

Login improvements #9
Edit
Code
Merged
DeanLogan merged 9 commits into master from login-improvements on Oct 19, 2023
Conversation 4
Commits 9
Checks 0
Files changed 32
+771 −115
DeanLogan commented on Oct 16, 2023
Collaborator
Description: This pull request represents the changes made to login security. This includes creating a new page for the login, along with adding additional protection against XSS attacks by adding CSRF tokens, adding a Captcha for the login along with 2FA through an OTP. A "forgot password" button has also been added that will allow users to send an email to the admin notifying them that a new password is needed for their account.
Changes:
Added the Captcha so that users had to prove they are not a bot to login
Added OTP libraries
Added QR code to add account to users authentication app of choice
Finished OTP completely, added a new TOTP Device table to the database
Removed login pop up and replaced with new login page UI
Added forgot email link, which sends an email to the admin notifying them of the user that has forgotten their password
DeanLogan added 8 commits 4 months ago
Added reCAPTCHA to login
Imported OTP libraries and added 2FA table
Generates QR code to add OTP to users authenticator app
2FA now functional within the backend
Fixed bug when user has multiple TOTP devices registered the program ...
Removed login pop-up and replaced it with login page
Added 2FA code pop-up for login page
Added forgot password link, and functionality for sending the admin a...
DeanLogan added the enhancement label on Oct 16, 2023
DeanLogan self assigned this on Oct 16, 2023
DeanLogan requested review from KyleMcComb, Rossmcal and conormugent1 3 months ago
Rossmcal reviewed on Oct 17, 2023
View reviewed changes
database/migrations/0001_initial.py
Show resolved
KyleMcComb approved these changes on Oct 17, 2023
View reviewed changes
KyleMcComb left a comment
Owner
Looks good to me. Good use of comments. Everything makes sense.
conormugent1 approved these changes on Oct 19, 2023
View reviewed changes
conormugent1 left a comment
Collaborator
Easy to follow and well written code
Reviewers
Rossmcal
KyleMcComb
conormugent1
Assignees
DeanLogan
Labels
enhancement
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
4 participants
Lock conversation


Web Scraping changes ready for master #11
Edit
Code
Merged
DeanLogan merged 10 commits into master from web-scraping on Nov 11, 2023
Conversation 1
Commits 10
Checks 0
Files changed 48
+1,083 −27
conornugent1 commented on Nov 10, 2023
Collaborator
Description: In this pull request, we've integrated a new Career model and expanded the Module model by adding a careers field. The admin interface now reflects these changes with the inclusion of the Career admin and a direct link for careers within the Module admin. To populate our career data, we've implemented a scraper for the Indeed website, extracting relevant job roles and associated data. This data has been used to populate the Career table, and the necessary relationships between Careers and Modules have been established using tokenization of common words in the descriptions.
Changes:
Adding the career model and adding the field "careers" to the module
Adding the career admin and adding the career link in the module admin
Scrape Indeed website for job roles and extract certain job role data
Populate the Career table with job roles and link Careers to Modules
DeanLogan and others added 9 commits 6 months ago
Created web scraping project and added a Indeed Job spider
Added Career models and admins
Updated and tested Module and Career models
Fixed bug where Career table was not being added to the database thro…
Update .gitignore
Verified
Populating career table. Linking module to careers
Code to populate careers field in Module
Merge branch 'master' into web-scraping
Database is populated with Careers
DeanLogan self requested a review 2 months ago
DeanLogan assigned conornugent1 on Nov 11, 2023
DeanLogan added the enhancement label on Nov 11, 2023
Restored db so it matches what was in master, and removed comments th…
DeanLogan approved these changes on Nov 11, 2023
View reviewed changes
DeanLogan left a comment
Collaborator
Overall pleased with the changes added. Love the tests that have been added in for the database, great job Conor.
One suggestion for improvement in the future is maybe adding in additional comments for your code, I managed to understand everything but that could just be because I have been working on the system for so long so could easily see how everything connected.
But again just a minor comment, overall great work.
Reviewers
DeanLogan
Assignees
conornugent1
Labels
enhancement
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
2 participants
Lock conversation

7.2.3 Paired Programming

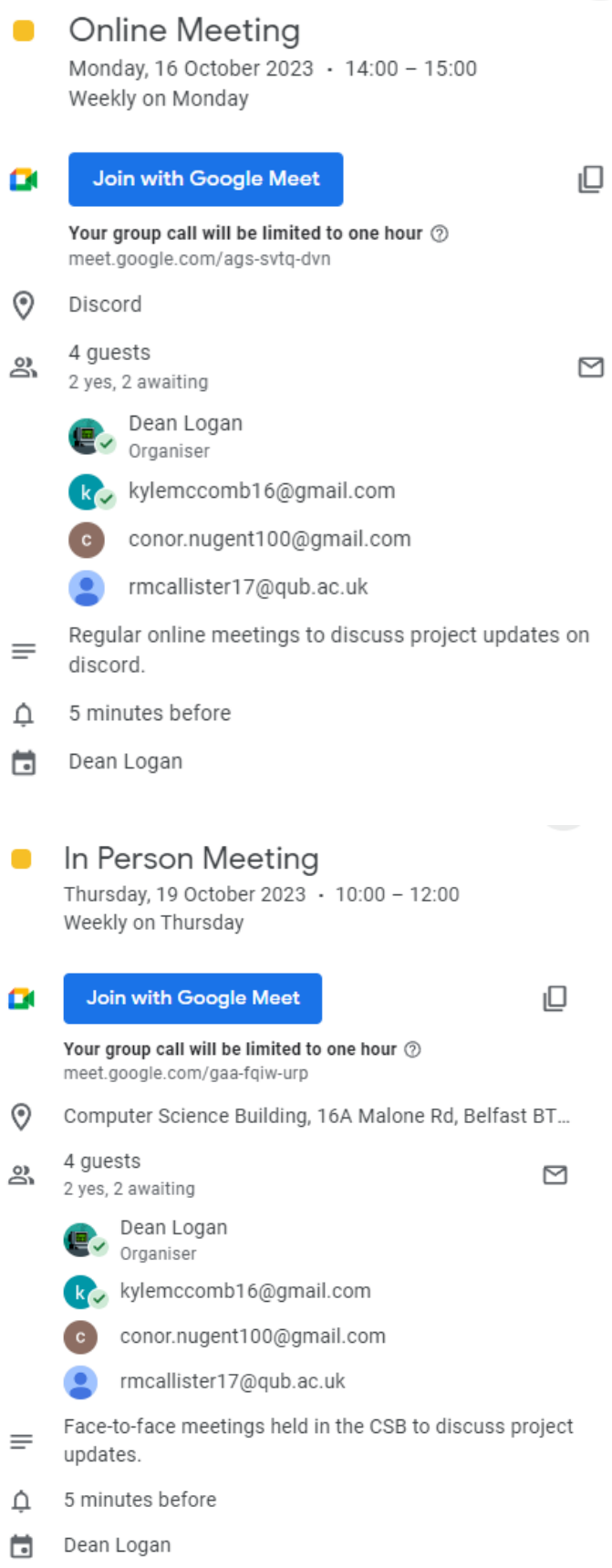


Full commit for the above can be found at this link
https://github.com/KyleMcComb/Pathfinder/commit/edc4f4d3f76c1b7386cbce163a5c8e30deb4eb38



Full commit for the above can be found at this link
https://github.com/KyleMcComb/Pathfinder/commit/ed80a5b0fd422af0f680a441b7cbccfb7b8201f7

7.2.4 Google Calendar Use



7.2.5 Meeting Minutes

September 15th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Kick-off meeting; How to improve website.
- Minutes:
    - Introduction and alignment of project objectives.
    - Distribution of initial tasks with a focus on the improve website phase.
    - Agreement on weekly online meetings via Discord and physical meetings when possible.

September 19th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister
- Agenda: Quick check-in on website improvements.
- Minutes:
    - Status update on individual tasks for improve website phase.
    - Identification of potential blockers and solutions discussion.
    - Scheduling of next in-person meeting.

September 22nd: In-Person Meeting at Computer Science Building

- Attendees: Dean Logan, Ross McAllister, Conor Nugent
- Agenda: Review of initial website improvements.
- Minutes:
    - Presentation of website design improvements.
    - Discussion on database architecture for the improve database phase.
    - Assignment of roles for database improvement tasks.

October 6th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Status update on improve database phase.
- Minutes:
    - Review of database improvements.
    - Planning for web scraping implementation.

October 16th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Progress check on web scraping development.
- Minutes:
    - Demonstrations of web scraping modules in progress.
    - Coordination of testing procedures for new code.

October 19th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Completion of web scraping and commencement of backup management system.
- Minutes:
    - Confirmation of web scraping implementation.
    - Discussion of backup strategies and commencement of backup system development.

October 27th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Interim report on backup management system.
- Minutes:
    - Update on backup system implementation.
    - Discussion on encryption and security measures for the backup.
    - Agreement on deadlines for the security/login improvements phase.

November 3rd: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Conor Nugent

- Agenda: Progress check on backup management system.
- Minutes:
    - Evaluation of the backup system's progress.
    - Initiation of security/login improvements.
    - Preliminary discussion on server deployment strategies.

November 10th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Security feature implementation and server preparation.
- Minutes:
    - Review of implemented security features.
    - Planning of server deployment steps.
    - Identification of tasks for the chatbot development phase.

November 17th: In-Person Meeting at Computer Science Building

- Attendees: Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Finalisation of security/login improvements and server deployment.
- Minutes:
    - Completion of login security enhancements.
    - Successful deployment of the project server.
    - Begin phase for chatbot development.

December 1st: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Review of chatbot progress and testing strategies.
- Minutes:
    - Presentation of chatbot functionalities.
    - Planning of comprehensive testing phase.
    - Organisation of the project's final review.

December 11th: Online Meeting via Discord

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Final preparations for project review and User Acceptance Testing (UAT).
- Minutes:
    - Final check on all project components against the roadmap.
    - Pre-UAT bug fixing and feature polishing session.
    - Review of documentation and presentation materials for final submission.

December 14th: In-Person Meeting at Computer Science Building

- Attendees: Dean Logan, Kyle McComb, Ross McAllister, Conor Nugent
- Agenda: Project completion and preparation for User Acceptance Testing (UAT).
- Minutes:
    - Final review of the project against the roadmap.
    - Discussion and planning for UAT.
    - Reflection on the project process and learning outcomes.