



**QUEEN'S
UNIVERSITY
BELFAST**

Pathfinder

Team Name: \$GetSomeCache

Date: 11-05-2023

Name: Dean Logan (40294254), Section(s): Requirements, Sprint Contribution 33%

Name: Conor Nugent (40296257), Section(s): Testing, Sprint Contribution 20%

Name: Ross McAllister (40291577), Section(s): Design, Sprint Contribution 23%

Name: Kyle McComb (40295231), Section(s): Implementation, Sprint Contribution 24%

All members contributed equally to Project Management. All members also contributed to design however Ross McAllister completed the most work within the design section.

Team Report: Interim Report (SDP2a)

CSC3068, Software Development Practice

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

1 Requirements

Problem Statement

Students studying Queen's EECS courses have a hard time picking modules that are in line with their career aspirations and their likes based on the current format of how the information about modules is delivered to the students. Students also struggle to keep track of how they are progressing through their degree (e.g., how far away are they from achieving their desired grade).

Product Vision

To create a system (Pathfinder) that will assist students in delivering information about modules that are available to them along with relevant career information for these modules. This will be done using a chatbot that will aid students in picking modules based on information provided by the student (likes/dislikes, career choices, etc), improving student satisfaction.

Students will also be able to search and filter all modules within EECS on a separate page where they will be able to see information on all modules. The motivation for adding this feature is to aid students in the event they wish to switch pathways so they can investigate the modules for other pathways in an easier manner than looking through a lengthy PDF.

Pathfinder will also provide a dashboard for students that will display information relating to their grades throughout their time at Queen's (overall degree percentage, module grades, assessment grades and useful statistics). The motivation for adding this feature is to help students track their progress throughout their degree, motivating students to work harder to obtain better grades. It also gives students an easier way to view how well they are progressing throughout their degree.

Requirements Elicitation & Analysis Process

As recommended by Ian Sommerville in "Software Engineering" [1], the team undertook a user-centred approach to requirements elicitation and analysis by engaging with the intended users of the system, namely the students in EECS. This approach aimed to understand the users' pain points with the current system (QIS and module description document) and to identify their requirements for a more effective and efficient system. Informal meetings were conducted with the students to encourage honest feedback, and research was conducted to explore the current state of the problem, as presented in the initial report.

Subsequently, the team summarised the students' feedback into key features that were frequently mentioned by multiple students. These features were further analysed to derive the requirements needed to implement them. To ensure the verifiability of requirements, the team placed a strong emphasis on ensuring that all requirements were testable and that their completion could be confidently asserted after testing.

As the team comprises students and a more agile development approach is being adopted, the requirements may be iterated upon throughout the development process based on the performance of implemented features and possible additions. The team's membership as students and intended users of the system confers advantages in terms of quick iteration and feedback collection without the need for formal meetings, which may be time-consuming to schedule. This enables the team to better tailor the system to the preferences and needs of potential users.

Requirements

GUI-Database

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	

Chatbot

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 be able 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-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 contact information for the student support office that may be able to help you: <email>". 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 in that language explaining that the chatbot only accepts English within the language that they entered to ensure the student understands how the chatbot works.	
Fit Criterion: When the word "Hola" is given to the chatbot it will be able to identify this as Spanish and respond with the error message above in Spanish. Must accept "hdfjkahfda" as input and respond with an error message.	

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-005	Requirement Type: Functional
Description: When the user provides the chatbot with what 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: Chatbot-006	Requirement Type: 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.	

GUI

Requirement Number: GUI-001	Requirement Type: Non-Functional
Description: The website must include a high contrast/colour blind option which will change the colour scheme of the website 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-004	Requirement Type: Non-Functional
Description: The website must be responsive; it must be able to adapt to various screen resolutions. (E.g., usable on 24-inch monitors, mobile devices and 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 bigger 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, 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 mobile and confirm that all UI elements are being displayed correctly.	

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 on load.	

ModuleInformation

Requirement Number: ModuleInformation-001	Requirement Type: Functional
Description: User must be able to search for a module on the module information page.	
Rationale: Makes it easier for a user to find out 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: Makes it easier for a user to find out 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	

SignUp

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.	

GradeDashboard

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: Ability for the system to 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.	

GUI-Chatbot

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.	

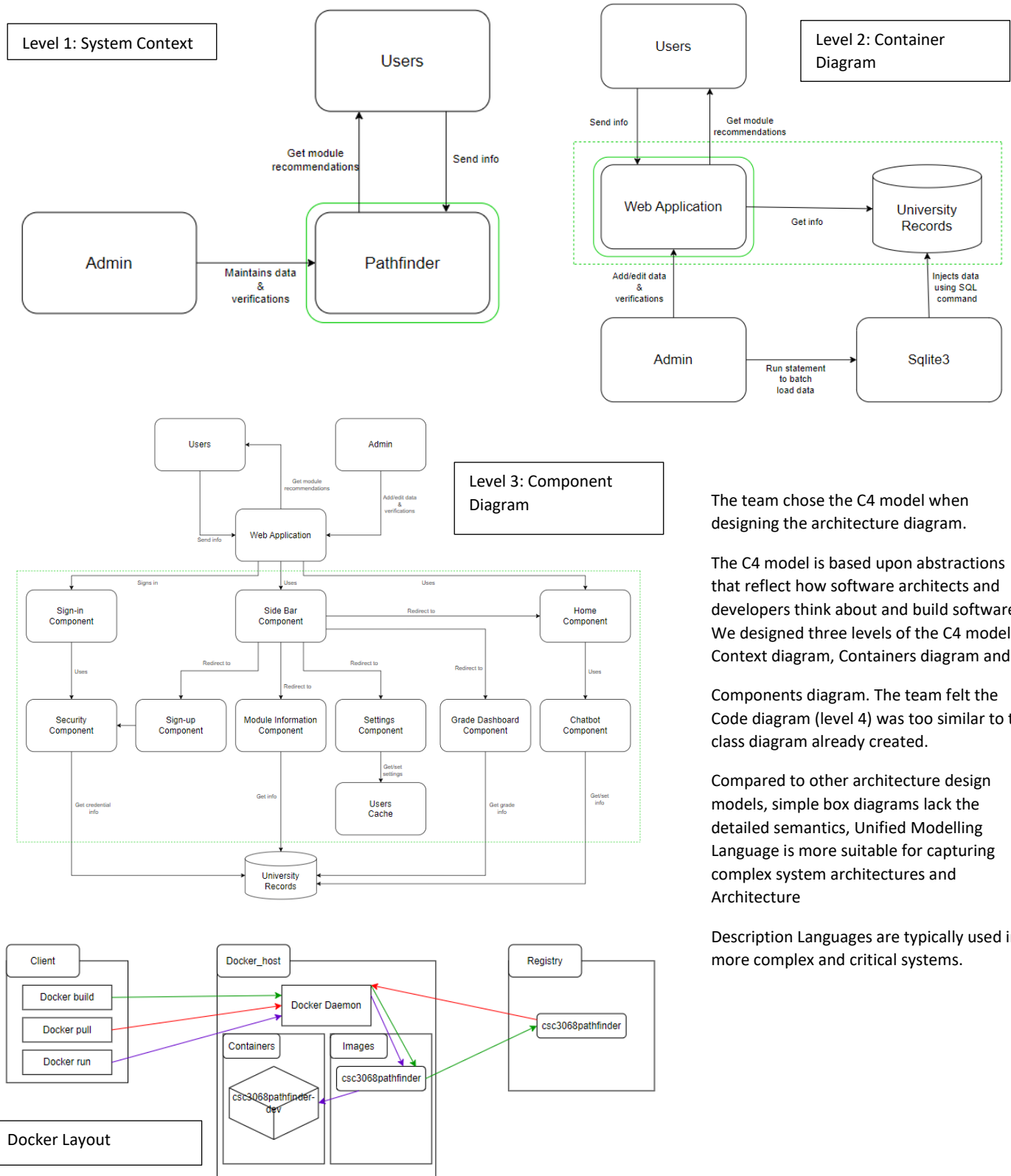
Requirement Number: GUI-Chatbot-002	Requirement Type: Functional
Description: Must be a place for the user to see their past inputs and the responses from the chatbot in a readable manner. Somewhere that displays both the messages the chatbot has received and the responses that the chatbot has sent back to the website.	
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 from the user.	

Requirements for GUI-GradeDashboard, GUI-Login, Database, Database-AccessRights, Website, WebScraping and Login can all be found within Appendix under A) Appendix: Additional Requirements.

2 Design

Software Architecture

Larger versions of the diagrams below can be found in Appendix B): Software Architecture.



The team chose the C4 model when designing the architecture diagram.

The C4 model is based upon abstractions that reflect how software architects and developers think about and build software. We designed three levels of the C4 model: Context diagram, Containers diagram and

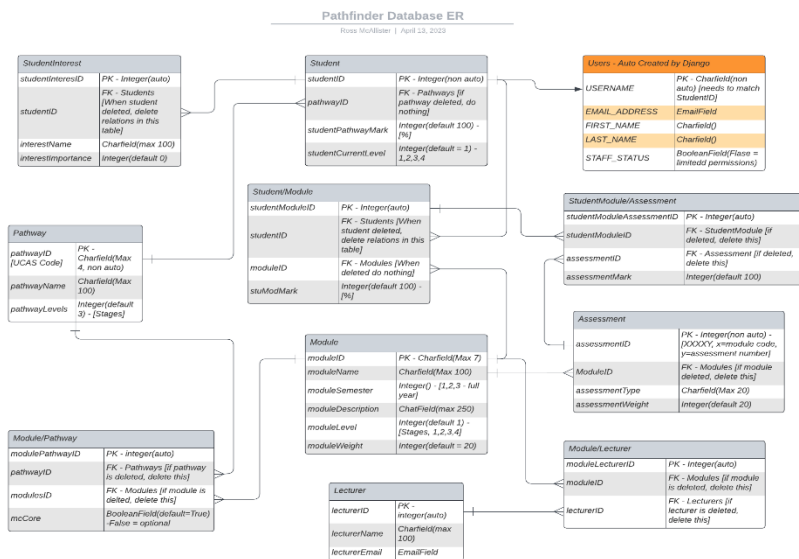
Components diagram. The team felt the Code diagram (level 4) was too similar to the class diagram already created.

Compared to other architecture design models, simple box diagrams lack the detailed semantics, Unified Modelling Language is more suitable for capturing complex system architectures and Architecture

Description Languages are typically used in more complex and critical systems.

System Design (Entity Relationship Diagram and Class Diagram)

Enlarged diagrams on this page can be seen in appendix C.1 & C.2



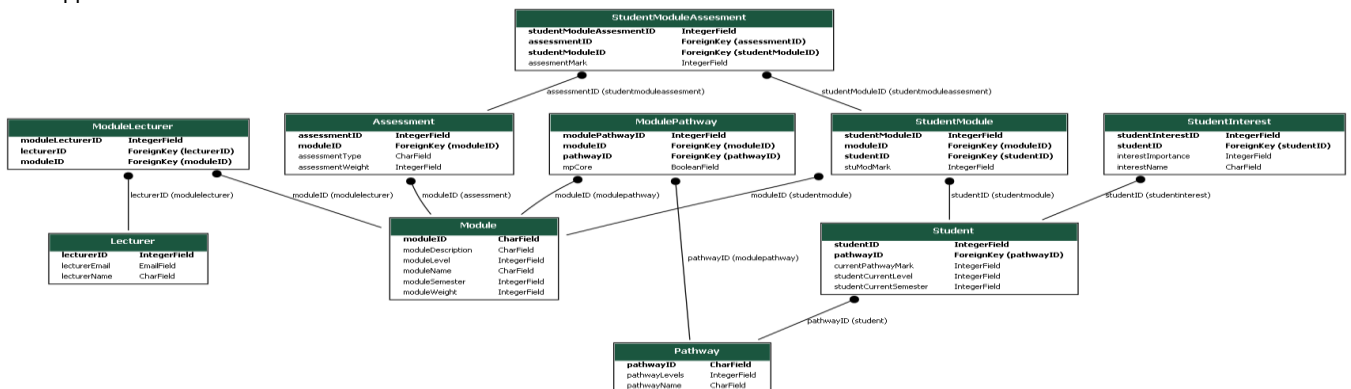
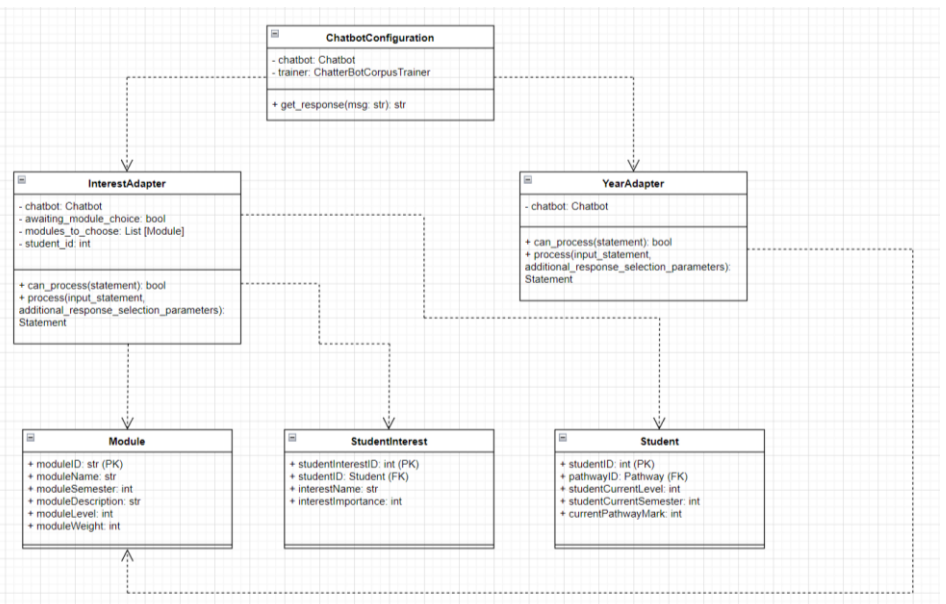
This diagram has been designed with naming conventions from the document that contains all pathways from 2022 to 2023.

To begin with the Users table highlighted in orange is auto created by django, this was a very useful feature as it allowed for automatic encryption helping to keep the sensitive data secured within our system. This table also allowed for 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. For each record within the Student table will have a corresponding record in the Users table, joined by having the Username field = the students ID.

Each assignment has a unique ID that corresponds 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 at a glance for admins when working with the database.

Similarly the ID for each record in the Module table is 7 characters long to contain the Module code E.g CSC2058, as well as the ID's for the Pathway table is 4 characters long to match the UCAS code.

The logic system has been shown in a class diagram to the right and the ORM for the database can be seen below. Once again a larger version of these diagram can be seen in the appendix within section C.



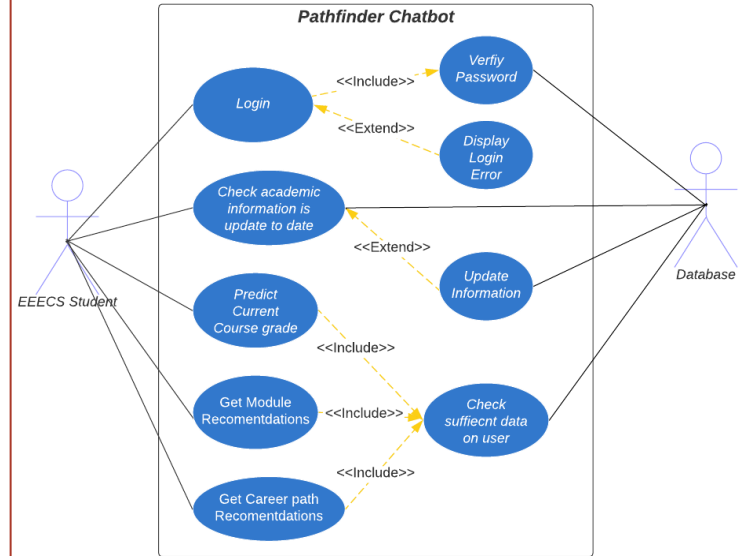
System Design

Below are two use case diagrams for our system. enlarged diagrams and their descriptions can be found in Appendix C.4.

Client Use Case

\$GitSomeCache | February 25, 2023

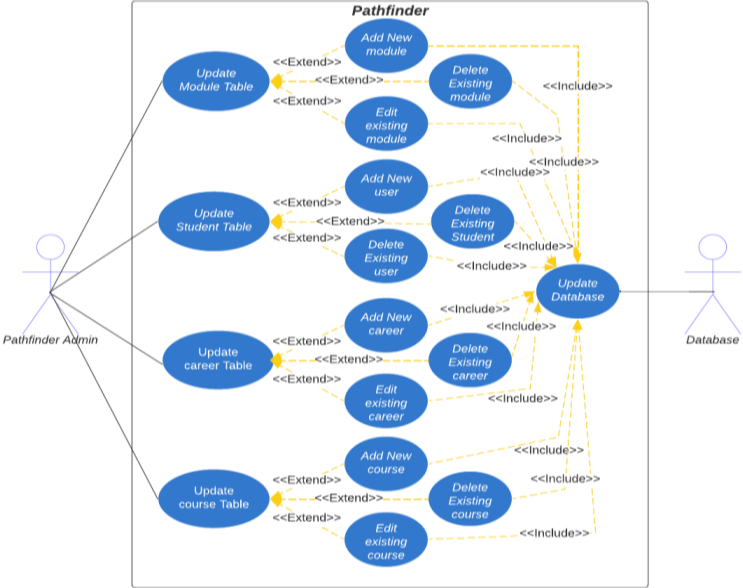
Pathfinder Chatbot



Admin Use Case

\$GitSomeCache | April 29, 2023

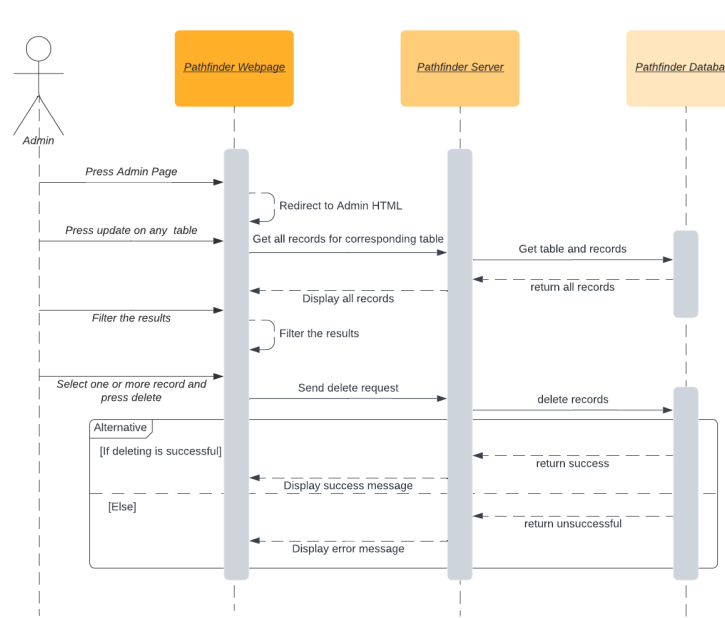
Pathfinder



We have created sequence diagrams for each use case please find two examples below. All sequence diagrams can be found within the appendix in section C.5.

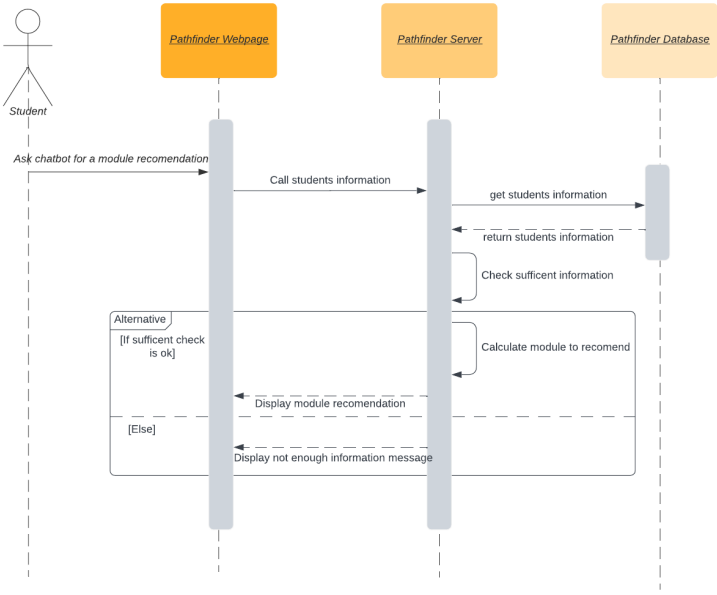
Admin UI - Delete records

Ross McAllister | April 29, 2023



Module Recommendation

Ross McAllister | April 29, 2023

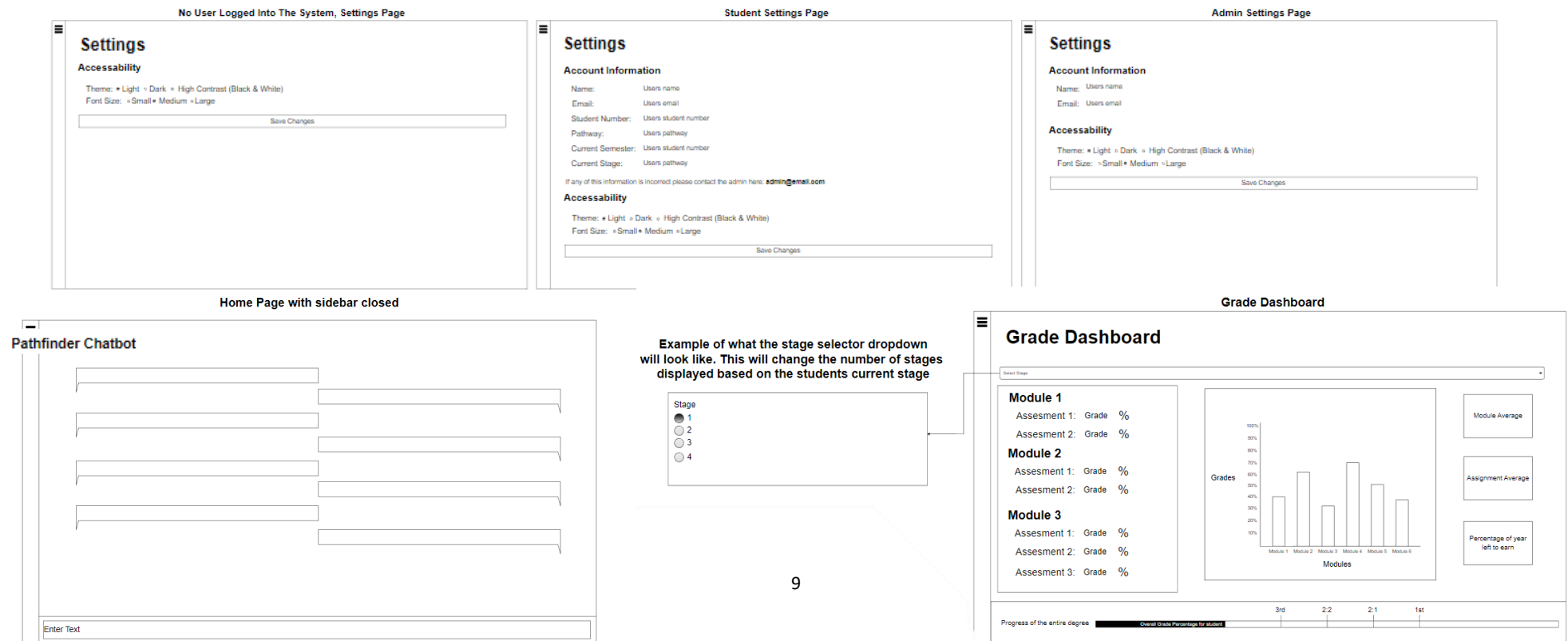


One design choice involving the Admin that we chose to go with was allowing them to either use Command Line or the Website UI to make changes to the database. This is because experienced admins would take advantage of the speed from command-line whereas the UI provides a clear visual of the database and can be easier to use for inexperienced

UI Design (Wireframes)

Below are wireframe diagrams for some of the pages for the website (additional wireframes and larger versions of the diagrams below can be found within the Appendix under D) Additional Wireframes) and a brief description of how the user may interact with the system. 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 like, making development a smoother experience as the front-end developer has a clear target to hit.

Whenever the user navigates to the website, they are presented with the home page which includes the chatbot and instructions on how to use the chatbot, they can start having a conversation with the chatbot by providing it with information to try and pick a module or they can open the side bar where they are presented with a couple options. They can go to the module information page to search/filter for modules and find more information about the available modules in EEECS, they can navigate to the main Queen's site, they can request to sign up to the system by filling in some options which will then be sent to the admin, or if they already have an account they can log into the system. Once a user has logged into the system they will have a new option in the navigation menu, admin panel if the user is an admin or grade dashboard if the user is a student. The admin panel will allow the admin to create, update, delete and view the entire database along with the ability to create user accounts and reset passwords. The grade dashboard page will allow students to see their current grades for their modules and assessments for those modules along with some useful stats like average module grade and assessment grade, they will also be able to see their overall degree percentage to help students track their progress. The settings page will allow users to see their personal information that's stored on the system and allow all users (even ones not logged in) to access the accessibility options, like changing the theme of the website and font size.



3 Implementation

3.1 Present state of the system:

The current implementation of the Pathfinder chatbot system is hosted on GitHub (<https://github.com/KyleMcComb/CSC3068-Pathfinder>). The key technology components of the implementation so far include:

Python: The team chose Python as the primary programming language for the backend because of its versatility, readability, and rich ecosystem of libraries. Python's extensive support for natural language processing and machine learning libraries, such as ChatterBot, spaCy, and TensorFlow, makes it an ideal choice for building a chatbot system[3].

Django: Django is a user-friendly web framework that makes it easy to create web applications quickly using Python. It was chosen because it comes with many built-in tools that simplify common web development tasks, like managing requests, displaying web pages, and handling forms. Django also includes a powerful Object-Relational Mapping (ORM) system, which makes working with databases much more straightforward and allows for smooth integration with various databases, including SQLite, which was decided to be used.

SQLite: For the database, the team opted for SQLite, a lightweight, serverless, and self-contained SQL database engine. SQLite is an excellent choice for the project due to its ease of setup, low maintenance overhead, and sufficient performance for a relatively small-scale application like theirs. It allows them to store student data, module information, and chat logs efficiently and access them through Django's ORM.

ChatterBot: ChatterBot is a Python library designed for creating chatbots with minimal configuration. It uses natural language processing and machine learning techniques to understand user input and generate appropriate responses. ChatterBot simplifies the development of the chatbot system by providing a robust foundation for processing and responding to user queries, allowing the team to focus on refining the user experience and integrating domain-specific knowledge related to module choices and career paths.

HTML, CSS, and JavaScript: The frontend of the chatbot system was developed using HTML, CSS, and JavaScript. HTML provides the basic structure and content for the web pages, while CSS is used for styling and layout. To enhance CSS capabilities, Sass, a CSS preprocessor that allows for variables and nesting, is employed, which makes the stylesheets more organised, maintainable, and scalable. JavaScript enables interactive features, such as updating the chat window without needing to refresh the page, providing a more seamless user experience.

The codebase for the project is entirely written by the group, except for the ChatterBot library, which is a third-party component that they utilise in their implementation. The team developed custom algorithms and logic for processing user inputs, determining module recommendations based on interests and grades, and guiding users through their potential career paths.

3.2 Reflection on past technology choices:

The technology choices in the previous project brief were the following:

TensorFlow: Initially, the team considered using TensorFlow for their chatbot project. However, they chose Chatterbot for the prototype due to its simplicity, ease of setup, and specifically tailored design for creating chatbots. This decision allowed them to quickly set up a functioning chatbot and focus more on enhancing the user experience and tailoring the chatbot to address the needs of EECS students. Although Chatterbot was more lightweight and required fewer resources compared to TensorFlow, which is more suitable for large-scale or demanding projects, the team has not ruled out the possibility of integrating TensorFlow in the future as the chatbot's logic becomes more complex. As mentioned later in Section 3.3, the team plans to explore various technology choices to improve the natural language processing of their chatbot as it evolves. This may include incorporating advanced natural language processing techniques or machine learning models such as TensorFlow to enhance Pathfinder's conversational capabilities.

Python: The choice of Python for the chatbot project was maintained due to its versatility and suitability for their requirements. Python is a highly flexible language with a vast array of libraries available, such as ChatterBot and spaCy, which provide essential functionalities for the chatbot. This extensive library support enables the team to efficiently build and refine their chatbot system, making Python an ideal choice for their project.[4]

Scrapy (Web Scraping): The team decided that they wouldn't have enough time to include web scraping in the prototype. However, they plan on keeping it as a technology choice later on in the project and in future sprints.

3.3 Future Technology Choices:

Looking forward, the team has several technology choices to consider for the upcoming sprints as they plan to expand the scope and functionality of their chatbot system:

Firstly, they plan to add web scraping as a possible feature. The reason for this is to keep the chatbot's information up to date and relevant. This will involve choosing a suitable library, such as BeautifulSoup or Scrapy (previous technology choice), in order to efficiently extract and process data from relevant websites, such as university course pages or job listings. This feature will ensure that their chatbot remains a valuable and reliable source of information for EEECS students.

Secondly, they want to improve the natural language processing of their chatbot. As the chatbot system evolves, they may consider incorporating more advanced natural language processing techniques in order to better understand user queries and generate more accurate responses. This could be through exploring libraries like spaCy or integrating machine learning models such as TensorFlow or PyTorch to enhance Pathfinder's conversational capabilities.

By considering these future technology choices, the team aims to create a more comprehensive, user-friendly, and secure chatbot system that can effectively guide EEECS students in making informed decisions about their module choices and career paths.

3.4 Code snippets

Relevant code snippets and pseudocode for the core algorithms have been included in the Appendix, with clear references to the corresponding code files in the GitHub repository.

3.5 Algorithm

The interesting/complex algorithm for their project involves Python, specifically using the Chatterbot Library: The two custom logic adapters, Interest Adapter and Year Adapter.

InterestAdapter Algorithm Description: Please see Appendix E (Implementation) For Pseudo Code

1. Initialize InterestAdapter: Set up the InterestAdapter with initial values for awaiting_module_choice, modules_to_choose, and student_id.
2. Check if input is relevant: Determine if the user's input contains keywords (like, dislike, or hate) to decide whether InterestAdapter should process the input.
3. Generate response based on interest: The process method handles the user's input and generates an appropriate response:
 - a. If awaiting_module_choice is True, process the module choice and update the student's interest accordingly.
 - b. If awaiting_module_choice is False, identify the user's interest (like, dislike, or hate) and the subject of interest, and display relevant modules based on the user's preference.
4. Return the response: The adapter returns the generated response along with its confidence value.

YearAdapter Algorithm Description: Please see Appendix E (Implementation) For Pseudo Code

1. Initialize YearAdapter: Set up the YearAdapter with the chatbot instance and any other required arguments.
2. Check if input is relevant: Determine if the user's input contains keywords related to year or stage (e.g., "first year", "2nd year", "stage 3") to decide whether YearAdapter should process the input.
3. Generate response based on year or stage: The process method handles the user's input and generates an appropriate response by identifying the mentioned year or stage and retrieving the list of modules for the corresponding year or stage.
4. Return the response: The adapter returns the generated response along with its confidence value.

Chatbot Configuration Description: Please see Appendix E (Implementation) For Pseudo Code

1. Configure ChatBot instance: Set up the ChatBot instance with the name 'Pathfinder', the storage adapter, logic adapters (including the custom InterestAdapter and YearAdapter), and the database URI.
2. Initialize ChatterBotCorpusTrainer: Create a ChatterBotCorpusTrainer instance, passing the configured ChatBot instance as an argument.
3. Set dataset path: Define the root directory and dataset path for the custom dataset (my_corpus.yml) to be used for training the chatbot.
4. Train ChatBot: Train the ChatBot instance using the custom dataset. Implement error handling to capture and report any exceptions that occur during training.
5. Define get_response function: Create a function called get_response that takes the user's message as an argument, converts it to lowercase, and returns the chatbot's response after processing the input.

This chatbot file serves as the main driver for the chatbot. It sets up the chatbot configuration, trains the chatbot with the custom dataset, and provides functions to handle user input and generate responses using the trained chatbot instance. The custom InterestAdapter and YearAdapter, which are integrated into the chatbot through the logic_adapters configuration, contribute to the chatbot's ability to generate contextually appropriate responses based on user input.

How it will be tested:

Testing Sentiment Analysis (Chatbot-001):

To ensure the chatbot accurately identifies sentiments, we will provide inputs with positive, negative, and neutral sentiments. The chatbot should correctly recognize the sentiment, such as understanding "I like Python" as positive, "I don't like Python" as negative, and "Hi my name is John" as neutral.

Testing Language Detection and Error Messages (Chatbot-002):

We will test the chatbot's ability to recognize non-English languages and respond accordingly by giving it inputs in various languages, like "Hola" (Spanish). The chatbot should detect the language and reply with an error message in the identified language. We will also check if the chatbot can handle random character inputs, such as "hdfjkahfda," by responding with an appropriate error message.

Testing Module Recommendations (Chatbot-003):

To verify that the chatbot offers relevant module recommendations, we will provide it with different user inputs representing a range of interests and academic records. We will then assess the relevance and suitability of the suggested modules to confirm that the chatbot is making accurate recommendations.

Testing Recommendation Responses (Chatbot-004):

We will evaluate the chatbot's ability to create well-crafted responses based on module recommendations by examining the quality and relevance of the information provided. For instance, if the chatbot recommends a malware analysis module, the response should include pertinent details such as the number of exams, the use of assembly language, and the module's relevance to careers in cybersecurity.

Testing Display of Modules (Chatbot-005):

We will test the chatbot's capability to display modules for a specific stage or pathway by querying it with statements like "list modules for stage X" and "list modules for pathway Y". The chatbot should return accurate lists of modules corresponding to the given stage or pathway.

Testing Instruction Provision (Chatbot-006):

To ensure the chatbot effectively provides instructions, we will prompt it with phrases like "help" and "what do you do?". The chatbot should respond with clear and concise instructions on its usage, which we will evaluate for its clarity and comprehensiveness.

4 Testing

Understanding the Role of Testing in Software Engineering (SE) and its Application to Pathfinder:

Testing plays a critical role in the software engineering process as it ensures that the software meets its intended requirements, functions correctly and is free from defects. The testing process helps identify and correct errors or bugs that may exist in the software before it is released to end-users, which improves its quality and reliability.

For Pathfinder, testing is crucial to ensure that the chatbot, module search page and dashboard features work as intended and provide accurate and useful information to students. The testing process must be comprehensive to ensure that it is free from errors or bugs that could negatively impact students' experience and academic progress. Therefore, the testing process must be incorporated throughout the software development lifecycle, including unit testing, integration testing, system testing, and acceptance testing.

Testing Plan for Pathfinder:

To test the Pathfinder system, a combination of automated and manual testing will be utilized. The different types of testing to be conducted include unit testing, integration testing, system testing, acceptance testing, and exploratory testing.

a. Unit Testing:

Unit testing involves testing individual components of the software to ensure they function correctly. For Pathfinder, unit testing will be used to test the chatbot's custom logic adapters (InterestAdapter and YearAdapter) and the database tables' fields. The Python unittest framework will be used to contain all the related test cases within one class. The resource for using unittest is found here <https://code.visualstudio.com/docs/python/testing>. The team opted for unittest instead of pytest since the former allows for better readability of test cases.

b. Integration Testing:

Integration testing involves testing how different components work together. For Pathfinder, integration testing will be used to test how the chatbot interacts with the database and the web interface. This will be achieved by testing the chatbot and its logic adapters within the Django web application. The Python unittest framework will be used for integration testing.

c. System Testing:

System testing involves testing the entire system as a whole to ensure that it meets the requirements and expectations of end-users. For Pathfinder, system testing will be used to ensure that the chatbot, module search page and dashboard features work together seamlessly. System testing will be carried out using both automated and manual testing methods.

d. Acceptance Testing:

Acceptance testing involves ensuring that the system meets the requirements and expectations of end-users. In the case of Pathfinder, acceptance testing will be conducted by involving end-users (team members that didn't develop the certain functionality) in the testing process through user acceptance testing. This will ensure that the system meets the needs and expectations of students.

e. Exploratory Testing:

Exploratory testing involves identifying issues that may not be detected through scripted testing. In the case of Pathfinder, exploratory testing will be conducted by interacting with the chatbot through the web application to identify any issues that may not have been identified during the scripted testing.

4.1 Code Testing

The testing approach for Pathfinder includes both automated and manual testing. While the prototype focused more on the user interface, manual testing was prioritized to thoroughly test its functionality. Automation testing will become more prominent as further implementation of Pathfinder's features is developed.

During the testing process, the team encountered some challenges with test organization and module importation. The issue of having separate test files in different modules was resolved by creating a single test module containing all the test cases, improving ease of access. This solution was discovered through the Stack Overflow resource <https://stackoverflow.com/questions/56780892/python-unit-test-module-throws-modulenotfounderror-no-module-named-tests-test>, and it streamlined the testing process by consolidating all the test cases.

The current test cases primarily fall under the category of black-box testing, where they are unaware of the internal code structure and implementation details. For example, when testing the chatbot, automation tests were conducted by providing predefined messages to the `get_response()` function and verifying if the chatbot responds correctly using its custom logic adapters (`InterestAdapter` and `YearAdapter`). These tests rely solely on the input and output of the system, making them black-box tests. Currently, there are ten automation test cases for testing prompts to the chatbot, with seven passing. The remaining three tests are intentionally planned to fail as the correct answers have not yet been implemented.

Several bugs were identified during the testing process and subsequently fixed. One bug was discovered while implementing the `test_stage_1()` test case. By using `assertEqual()`, the team noticed a discrepancy between the actual chatbot response and the expected response. The bug originated from the `YearAdapter` class, where the for loop that adds each module to the response was incorrectly replacing the response with the last module. This issue was resolved by adding a "+" to the statement assigning the modules to the response, ensuring all the modules are included.

Another bug was found when implementing the `test_unknown_interest_response()` test case. Regardless of the number of modules that matched the user's interest (even zero matches), the response indicated that an interest was found in one or more modules, leaving the module name and code empty. To fix this bug, a check was added to count the modules and display the response only if the module count was one or more, preventing empty module information from being displayed.

To test the database and verify the creation of tables with the correct fields, automation tests were employed. The fields of each model being tested were retrieved using a method found in the Stack Overflow resource <https://stackoverflow.com/questions/3647805/get-models-fields-in-django>, and they were compared to a list of hard-coded fields from the models. The tests consisted of eight test cases, each testing a different database table and all of them passed. Initially, all the database automation tests failed when the `get_fields()` method retrieved all the fields including the ones directly linked (foreign keys) to that table, leading to a mismatch with the hard-coded fields from the models that didn't have some of the foreign keys. However, the issue was resolved and the tests now accurately verify the database table structures.

Overall, automation tests have been conducted for the chatbot's logic adapters and the database tables' fields, using the Python unittest framework. Bugs have been identified and fixed, and the testing process has helped improve the accuracy and reliability of the system.

Code reviews involve a thorough examination of the codebase by one or more team members to identify potential issues, improve code readability and ensure adherence to best practices. For Pathfinder, the team used various methods to conduct code reviews using the version control system GitHub. These methods included in-person or pair programming reviews, branch comparisons and inline comments.

In-person or pair programming reviews allowed for direct interaction, discussion and immediate feedback on the code changes. The GitHub branch comparisons were very useful for seeing differences between two branches, reviewing the changes and leaving feedback on specific lines of code. Finally, by adding inline comments, the team could focus discussions on specific code snippets.

To conclude, by code reviewing within a certain time after the code is pushed to GitHub, collaboration was promoted among team members, which ultimately helped improve the quality and maintainability of the codebase.

4.2 User Evaluation

The team conducted alpha testing to evaluate the Pathfinder system's functionality and gather feedback from team members that didn't code that specific functionality. The alpha testing aimed to assess the system's usability, identify any issues or bugs and gather insights for further improvements.

The team prepared a large number of manual test cases to be performed during the alpha testing. Each test case defined a unique test ID number that corresponded to a requirement, the description of the test, the test steps, the expected outcome, the actual outcome and whether the test passed or failed.

The team were provided with different test data to use and tasks to perform, such as using the chatbot to explore module options, searching for specific modules and accessing the dashboard to view grades and statistics.

Based on the findings from the manual test cases, the team made iterative improvements to the Pathfinder system. This involved addressing identified usability issues, fixing bugs, refining the chatbot's responses and enhancing the dashboard's features and presentation.

Key findings from the alpha testing included:

Usability feedback: The team provided feedback on the intuitiveness and ease of use of the chatbot interface, module search page and dashboard. This feedback helped identify areas that required improvements to enhance the user experience.

Bug identification: The team encountered a few bugs during the testing process, such as incorrect responses from the chatbot or unexpected behaviour on the module search page. These bugs were documented and will be addressed in further development.

Some security test cases were included in the manual tests, for example, trying to access admin functionality with student accounts, testing for SQLi attacks (Login-005-01) and testing for XSS attacks (Login-005-02). All the test cases can be found in Appendix F) Additional Test Cases.

Test Case ID: Login-005-01
Test Case Description: Test for SQLi attacks.
Test Steps: <ol style="list-style-type: none">1. Enter the following text into the username field: " or ""=".2. Enter the following text into the password field: " or ""=".3. Click the login button.
Expected Outcome: The login should not be successful and an error message should be displayed stating that the login failed.
Actual Outcome: The login was not successful and an error message was displayed stating that the login failed.
Pass/Fail?: Pass

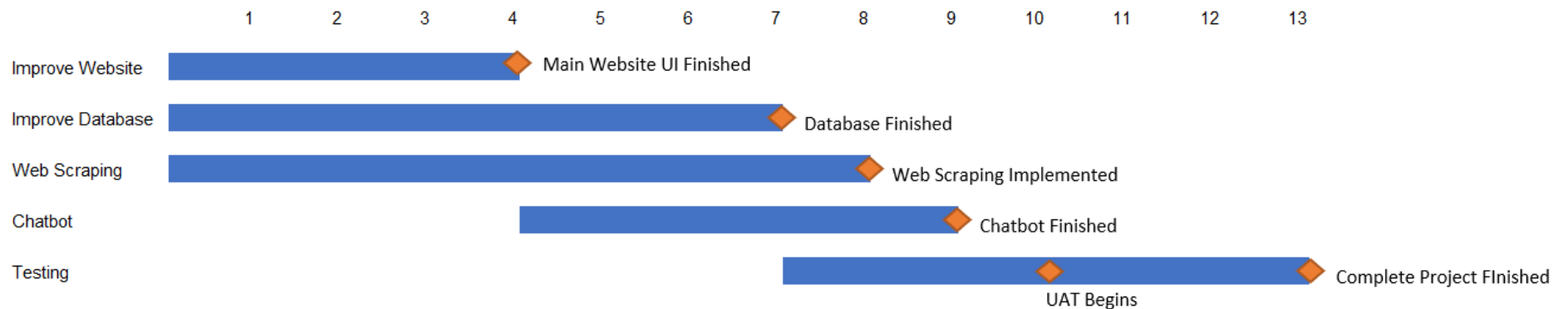
Test Case ID: Login-005-02
Test Case Description: Test for XSS attacks.
Test Steps: <ol style="list-style-type: none"> 1. Enter the following text into the username field: "<script>alert('Hello World!');</script>". 2. Enter any password. 3. Click the login button.
Expected Outcome: The login should not be successful and an error message should be displayed stating that the login failed.
Actual Outcome: The login was not successful and an error message was displayed stating that the login failed.
Pass/Fail?: Pass

Overall, the alpha testing provided valuable insights into the strengths and weaknesses of the Pathfinder system, allowing the team to make informed decisions for further development. It helped identify areas for improvement and refine the system's functionality.

5. Project Management: Roadmap and Sprint Plan Update

Roadmap

The following's a revised roadmap which includes our key success criteria/goals for finishing the system within semester 2 and the milestones throughout semester 1. A more detailed view can be found at Appendix G) Roadmap. (Note: the numbers along the top represent the weeks within the semester)



Success Criteria for System:

- All Functional and Non-Functional requirements completed.
- Sentiment analysis improved from prototype (accuracy of 98%).
- All tasks completed on time in guidance with the gantt chart above.
- Database to contain all modules for courses within EECS.
- Data integrity must be improved upon, e.g., whenever an assessment grade changes the grade for that module must also change.
- Improvements made to the UI for the website:
 - All animations must be smooth with no jitter.
 - Website must be compatible with mobile devices.

Sprint Plan (September – October)

The overall goal for our sprint returning after the summer is to ensure all team members are on the same page before returning to development of main features. This process will involve reflection of previous sprints as well as tidying up code. It is also important for team members to take some time to familiarise themselves with the code again which we hope will be a quick process.

Below is a table containing the main features for our next sprint plan.

Feature	Motivation for prioritisation	Testing to be performed	Risks	Mitigation plan	Assigned To
Ability for students to input and change their module assignment grades.	Critical for the system to provide grade estimations. This is a high priority as it will complete the Grade estimation feature.	Check that the chatbot understands the user wants to input/update a grade. Ensure that inputs are correctly stored for the logged in user within the database.	Incorrect grades are saved, or grades are saved to the incorrect user.	Develop validation checks on the user that is logged in before saving any data to the table. Have validation that the user must press to have the changes be saved.	Kyle McComb
Create a database table for industry career paths.	This will lay the foundation for the career path recommendation feature to be developed later down the line.	Ensure that data can be entered and retrieved from the database. Check that each column has the appropriate data type.	Updating the database models corrupts other existing tables.	Have a backup for the database saved on GitHub first and then begin to make changes to the database.	Ross McAllister
Ui bug fixes.	Ensure the UI is flexible on different screen sizes for accessibility. If we can get the UI bug fixed, then more focus will be available for critical features in the next sprint.	Can utilise Model based testing to automatically run through paths on the UI, while pressing buttons. Attempt to load the website on different sized screens to explore any additional bugs that appear.	Attempts to fix bugs cascades and creates new bugs, this will tell us that our UI code is fragile.	Create a bug tacker list to be able to trace back all changes made and ensure it is reversible.	Dean Logan
General Refactoring of code.	This is an important task to ensure our code is maintainable. It will also lend to easier future development as the code is easy to read.	For each function that is altered it needs to be retested using the existing test cases.	Any changes can affect the possibility of having compile errors.	Like the UI bug fixing, create a record of renaming/deleting/updating on lines of code.	Ross McAllister & Kyle McComb.
Containerise the source code using docker and upload to Charles' server.	This is a fundamental part of the design architecture for our project. We do not want to leave this until later as any major issues will need to be found and fixed in sufficient time.	Attempt to access the website through different machines, which are disconnected from the server. Check that inputs into the website are saved in the database as well as calculations are performed correctly (E.g., get estimated grade)	The biggest risk will be causing any damage to the server, this can include overriding any existing files.	Have a meeting with Charles before attempting to work on the server first to ensure we know the correct steps to take to run the docker image.	Conor Nugent

Project Setup and Management Tools:

During the project the team communicated primarily through discord, this is because discord allowed us to communicate with each other through various text channels allowing for conversations to be easily organised. This also provided the team with voice channels and the ability to share screens making it easier to show each other progress and help fix any problems.

The collaborative environment for code sharing (and code management)was GitHub. This is because with GitHub, the team easily created and shared repository with everyone, allowing the team to work on the project simultaneously. Moreover, GitHub provides a wide range of collaboration tools, such as pull requests, issues, and comments, which facilitate seamless communication and feedback among project contributors. Additionally, GitHub allows the team to track changes to the code and easily revert to previous versions if necessary, ensuring the integrity and reliability of the project.

GitHub also allows the team to track who made changes helping to identify where problems arisen, create branches which allows the team to work on different features of the project simultaneously without interfering with each other's work, then allowing the team to merge the code once the team is satisfied that the feature has been implemented correctly.

Here is a link to the GitHub repository <https://github.com/KyleMcComb/CSC3068-Pathfinder>

Jira was used as the tool for managing issues that were assigned to members of the team and track progress of the prototype. As Jira is customizable, the team changes the standard format of the issues to better suit the development style of the team, removing unnecessary information from the issues tabs. The ability to assign individuals to

The team also made use of the ability to break a problem up into smaller tasks (epic, story, sub-task, etc) to assign these to individuals to improve productivity within the team and to avoid one member "blocking" another. This also helped team members see who was responsible for certain features and contact the correct team member if they have any questions.

Jira also provided the ability for the team to track the progress of issues using the project board, allowing team members to see what features have been finished or are yet to be finished helping the team to see the progress being made by other team members.

Here is a link to the Jira project <https://csc3068-project.atlassian.net/jira/software/projects/PF/boards/4>

Updates on issues and risks, including action taken and to be taken to mitigate them:

Colour scheme options: To ensure that the user interface is accessible to all users, the team has implemented options for switching colour schemes, including a light/dark mode and a high contrast mode. This feature is working as expected.

Language Identification: At present, our prototype currently doesn't have the capability to identify languages other than English. In order to mitigate this, the team will implement language detection functionality in the next sprint for the chatbot. If a non-English language is detected, the bot will be programmed to respond with an appropriate error message in the detected language. This message will explain that the bot only supports English. The team doesn't have plans to support multiple languages at the moment.

Data encryption and access rights: The team has implemented PBKDF2 algorithm[5] with a SHA256 hash using Django to ensure all sensitive data is encrypted and not stored in plaintext to protect users in the event of a data breach. We also don't store the passwords in settings.py[6]. Access rights have also been established to ensure that only authorised users can view specific data.

Password hiding: Passwords are now hidden behind asterisks when users enter their credentials. This provides an additional layer of security and privacy. This feature has been implemented successfully.

A) Appendix: Additional Requirements

GUI-GradeDashboard

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.	

GUI-Login

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.	

Database

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-002	Requirement Type: Functional
Description: The database must store information about all the different pathways that are offered within EEECS.	
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 is provided for that pathway, required and optional ones.	
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-004	Requirement Type: Functional
Description: The database must contain information on lecturers, this should include their name, email, password, and module(s) they teach.	
Rationale: The database must store information about lecturers so that login information for each lecturer can be stored and that the chatbot will be able to identify Lecturers that relate to modules or be able to provide additional support.	
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: Non-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.	

Database-AccessRights

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.	

Website

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.	

WebScraping

Requirement Number: WebScraping-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: WebScraping-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.	

Login

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).	

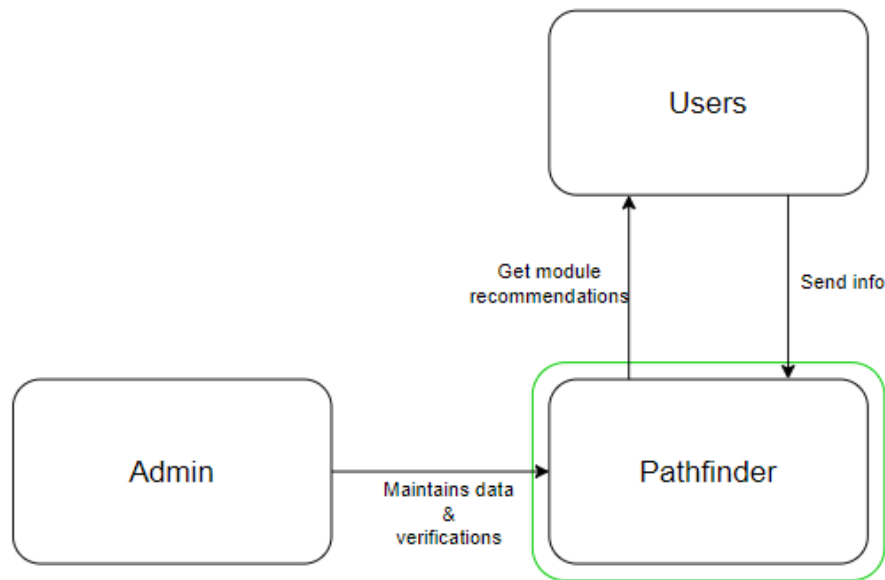
Requirement Number: Login-003	Requirement Type: Non-functional
Description: Passwords must be encrypted within the database	
Rationale: Passwords should not be stored in plaintext in case there is a data breach.	
Fit Criterion: Create a password for any account "password" then verify within the database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

Requirement Number: Login-004	Requirement Type: Non-functional
Description: The ability for the user to be "remembered".	
Rationale: This is so that 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.	

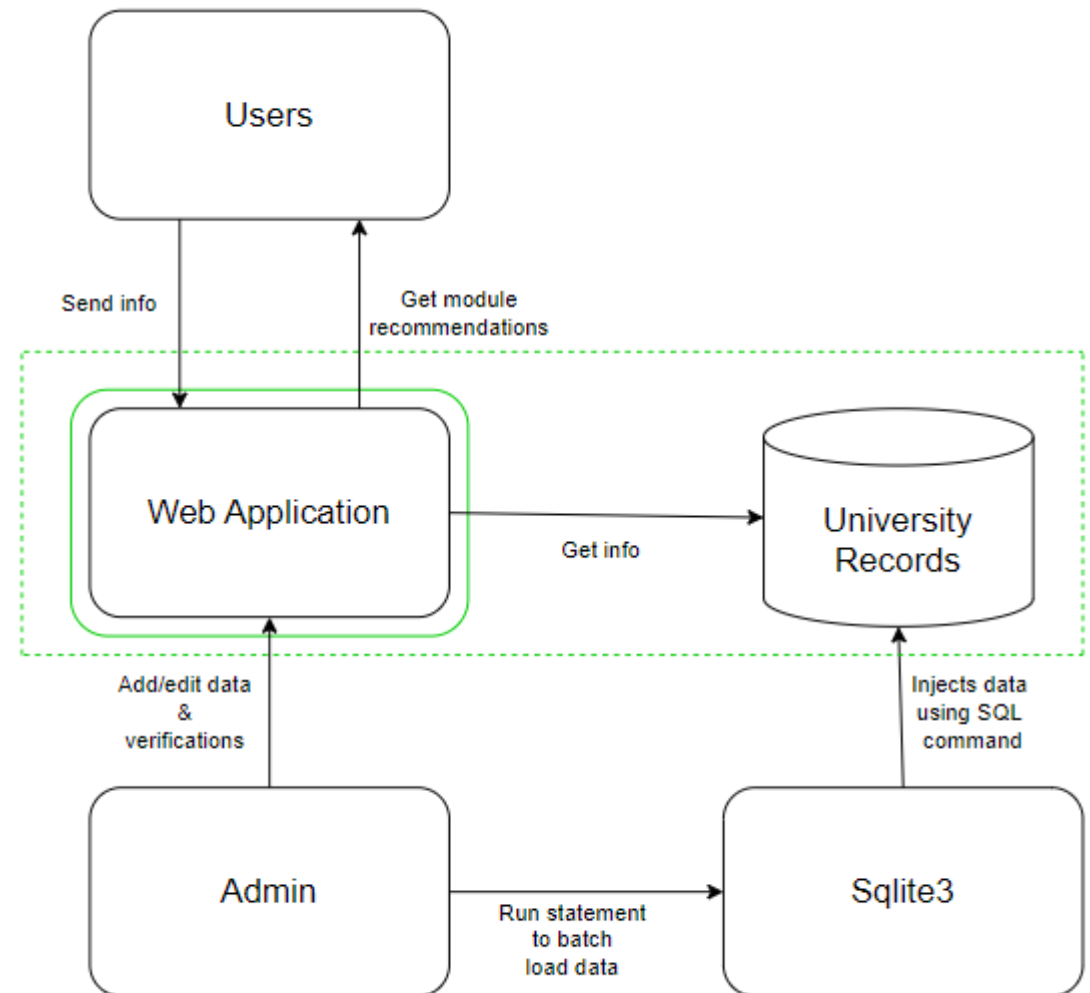
Requirement Number: Login-005	Requirement Type: Non-functional
Description: Login system must not be susceptible to SQLi attacks or XSS attacks.	
Rationale: The system needs to be protected against 2 of the most common attacks on websites that could affect the confidentiality, integrity and accessibility 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.	

B) Appendix: Software Architecture

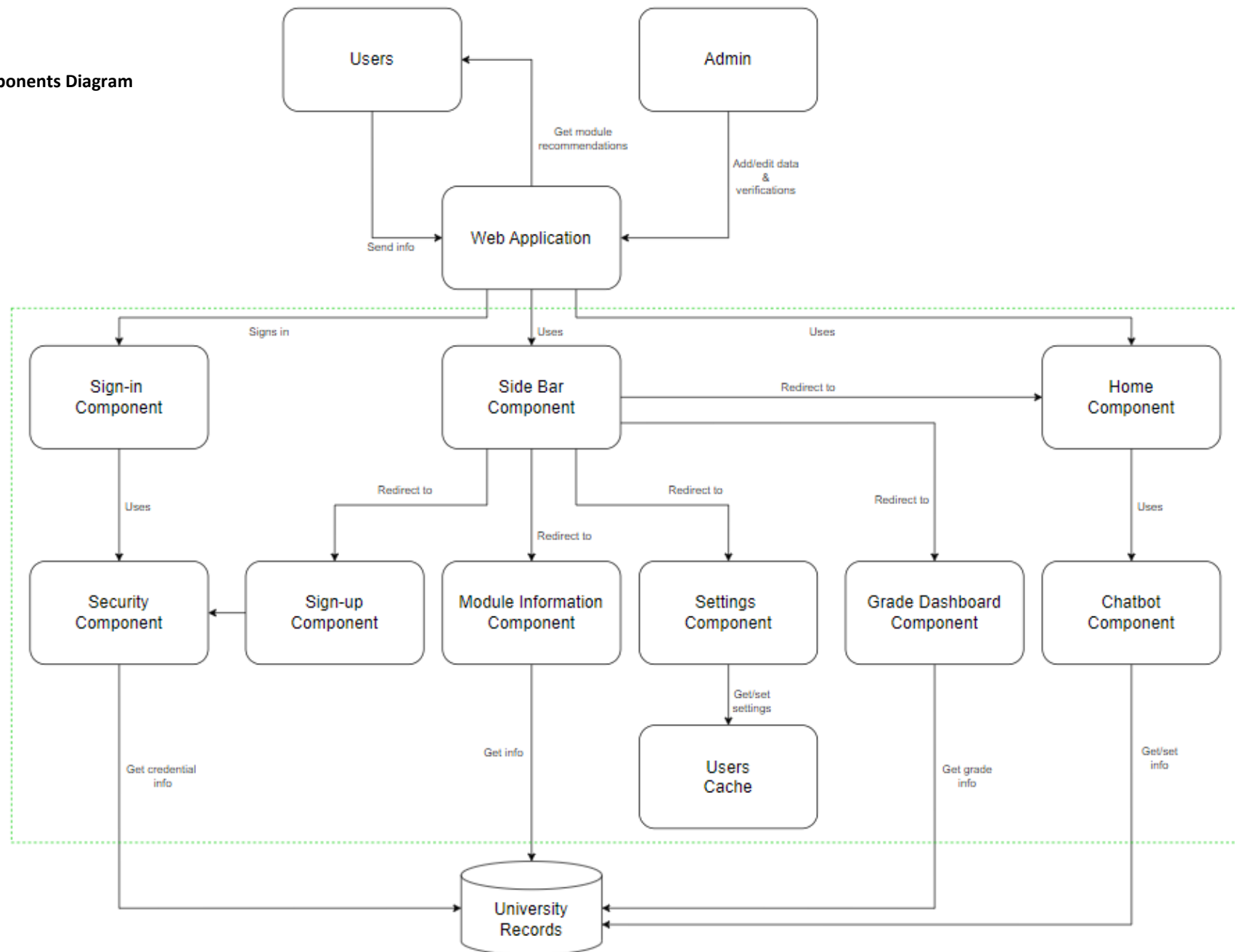
Level 1: System Context Diagram



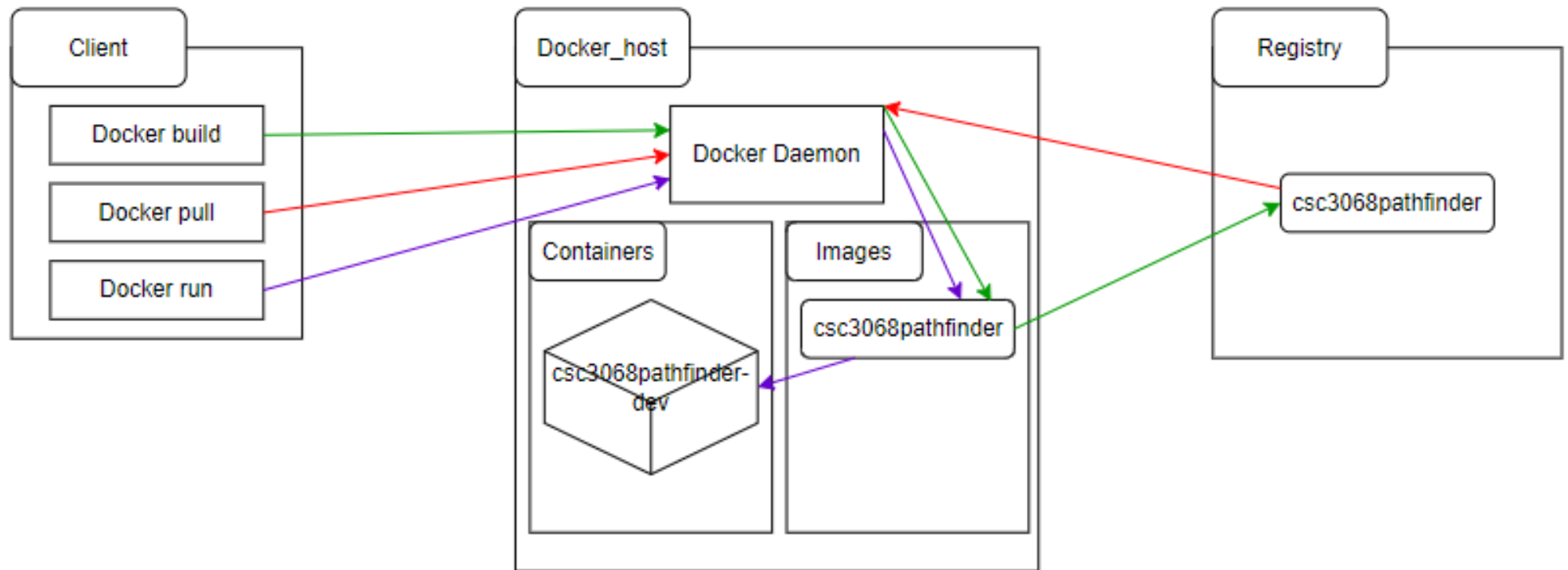
Level 2: Container Diagram



Level 3: Components Diagram

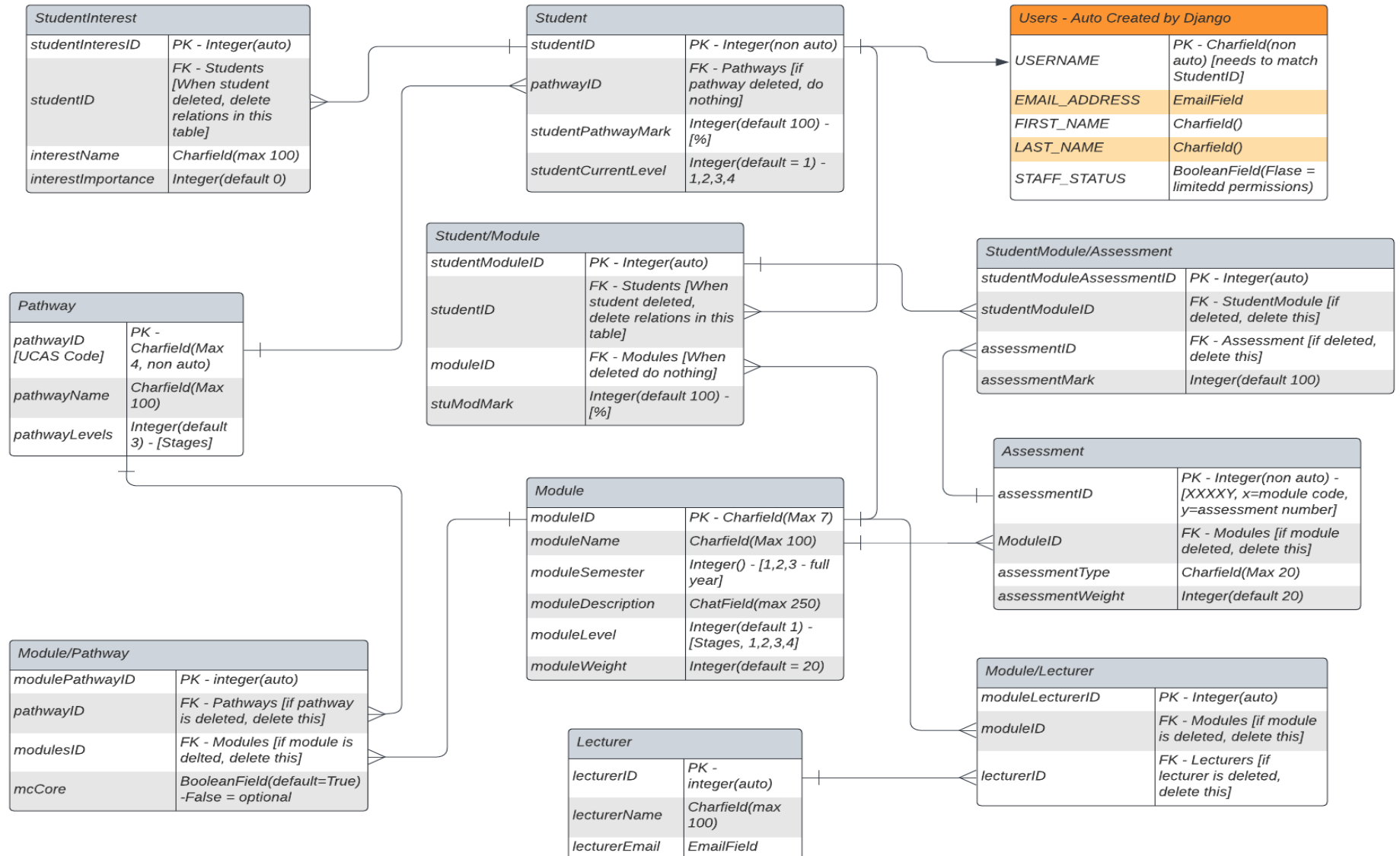


Docker Layout

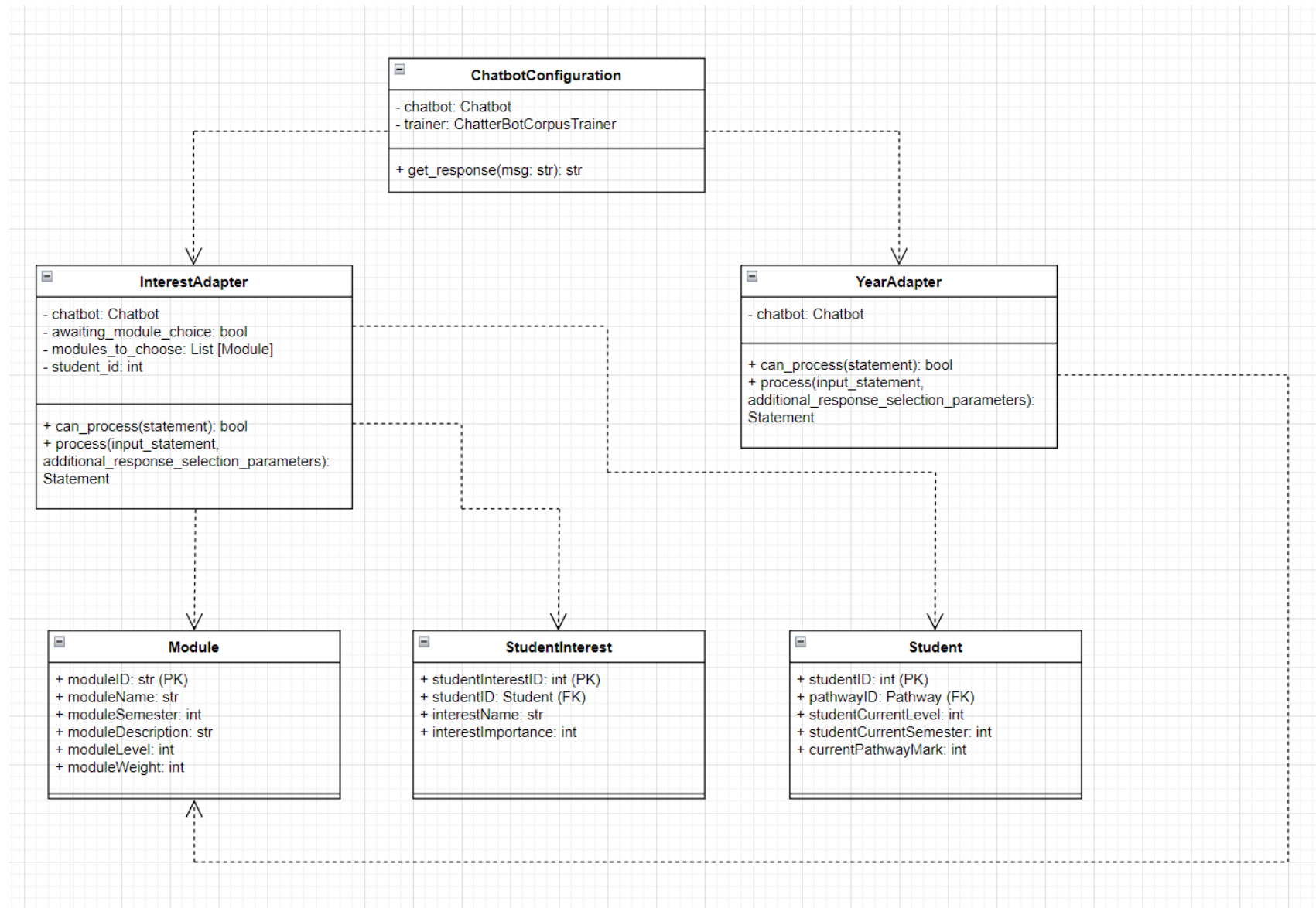


C) Appendix: System Design

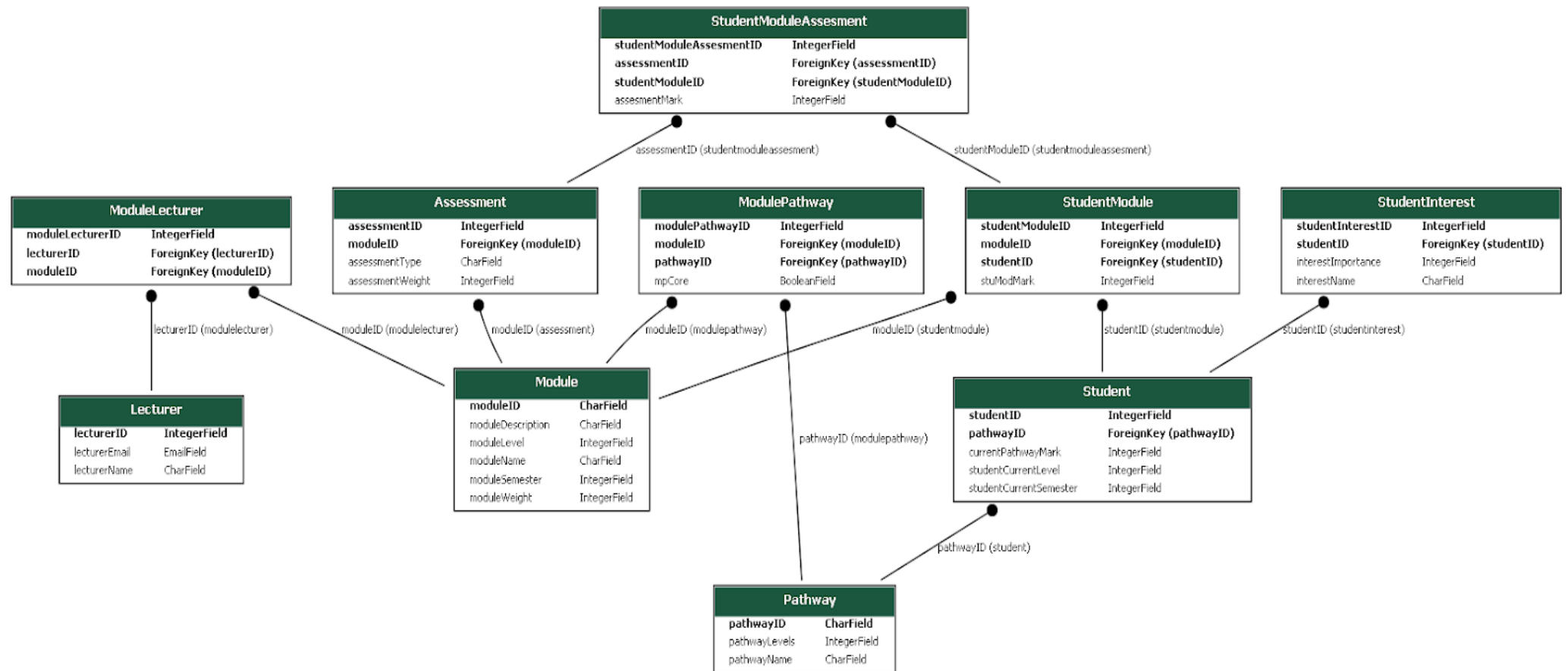
C.1: Entity Relationship Diagram



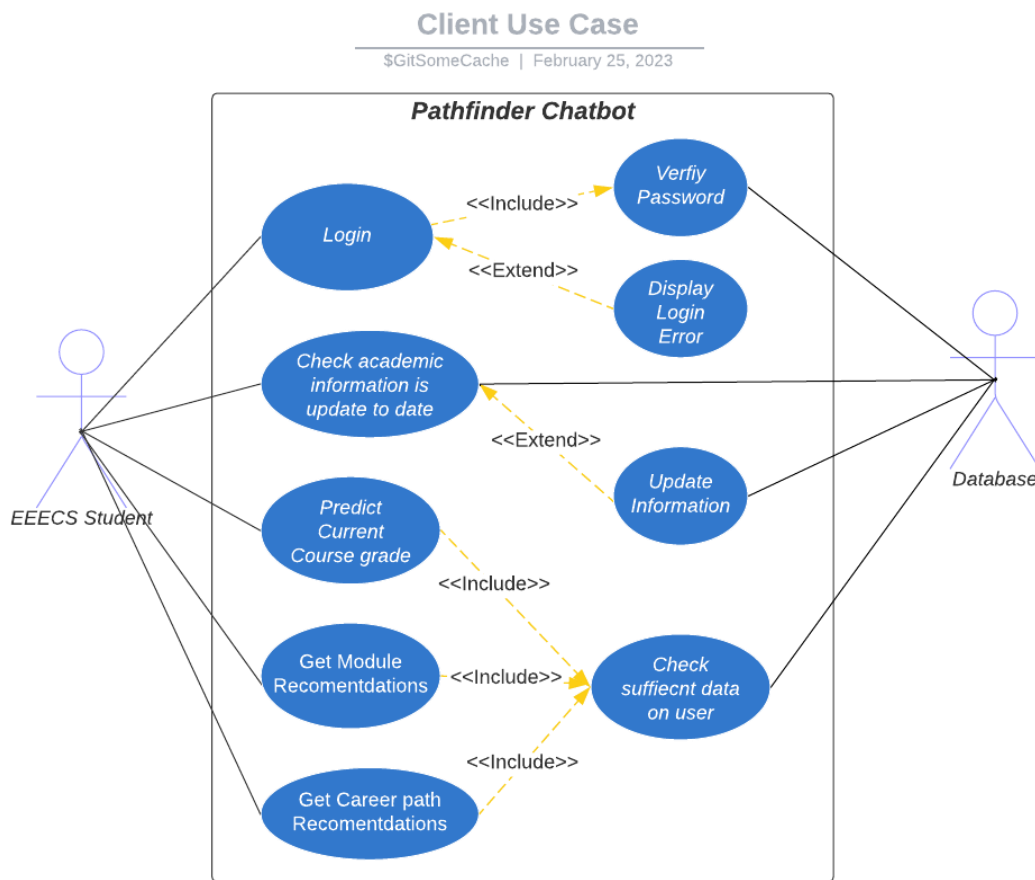
C.2 Logic System Class Diagram



C.2: Database Class Diagram (ORM)



C.4: 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, the system first prompts for Login
Main Flow	<ol style="list-style-type: none"> 1. The user is asked to enter their username. 2. The user is asked to enter their password. 3. The user's credentials are checked against the ones stored in the database and are verified to be equal. 4. The system allows the user to progress within the system.
Alternate Flows	<ol style="list-style-type: none"> 1. Invalid username 2. Invalid password -> Display Login Error. The user is not allowed to access the system further.
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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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 users current grades, the system can calculate what grade they are currently sitting on in regards to their whole course
Precondition	The user has logged in. The user has relevant data stored.
Main Flow	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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.

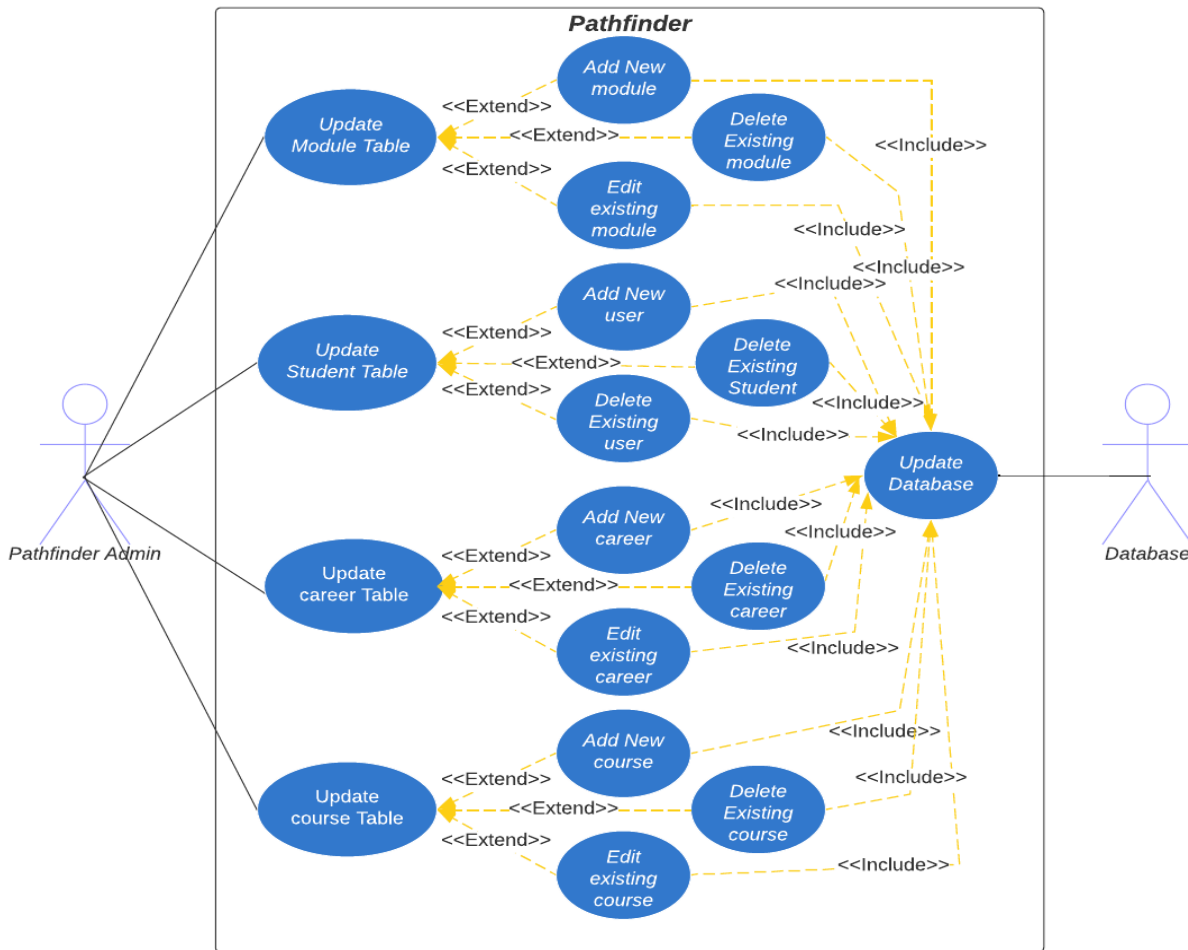
Flow of Events for the “Get module recommendations” use-case	
Objective	By using the user’s previously selected modules, likes and dislikes, the system can provide module recommendations to the user for the next academic year.
Precondition	The user has logged in. The user has relevant data stored.
Main Flow	<ol style="list-style-type: none"> 1. The user asks the system to provide module recommendations. 2. The system asks for their likes and dislikes. 3. The system verifies that the user has enough data stored by accessing the database. 4. The system uses the user’s previous module choices, likes + dislikes to provide a recommendation. 5. The system prompts the user with the modules with relevant information about the modules and where they can find further information.
Alternate Flows	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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. 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	<ol style="list-style-type: none"> Admin selects the option to update the modules. Admin decides to add a new module. Admin enters the relevant information for the new module and links the module to the 1 or many courses that it belongs to. Admin links the module to relevant career job entries
Alternate Flows	<ol style="list-style-type: none"> The admin decides to update an existing module. The admin selects the module to update, and they are shown all of the module’s fields. The admin can then change the fields or links to other tables. 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.

Admin Use Case

\$GitSomeCache | April 29, 2023



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	<ol style="list-style-type: none"> 1. The admin wants to add a new student as a user to the system. 2. The admin takes the students 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	<ol style="list-style-type: none"> 7. The admin decides to update an existing user. The admin selects the user to update, and they are shown all of the user's fields. The admin can then change the fields or links to other tables. 8. 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.

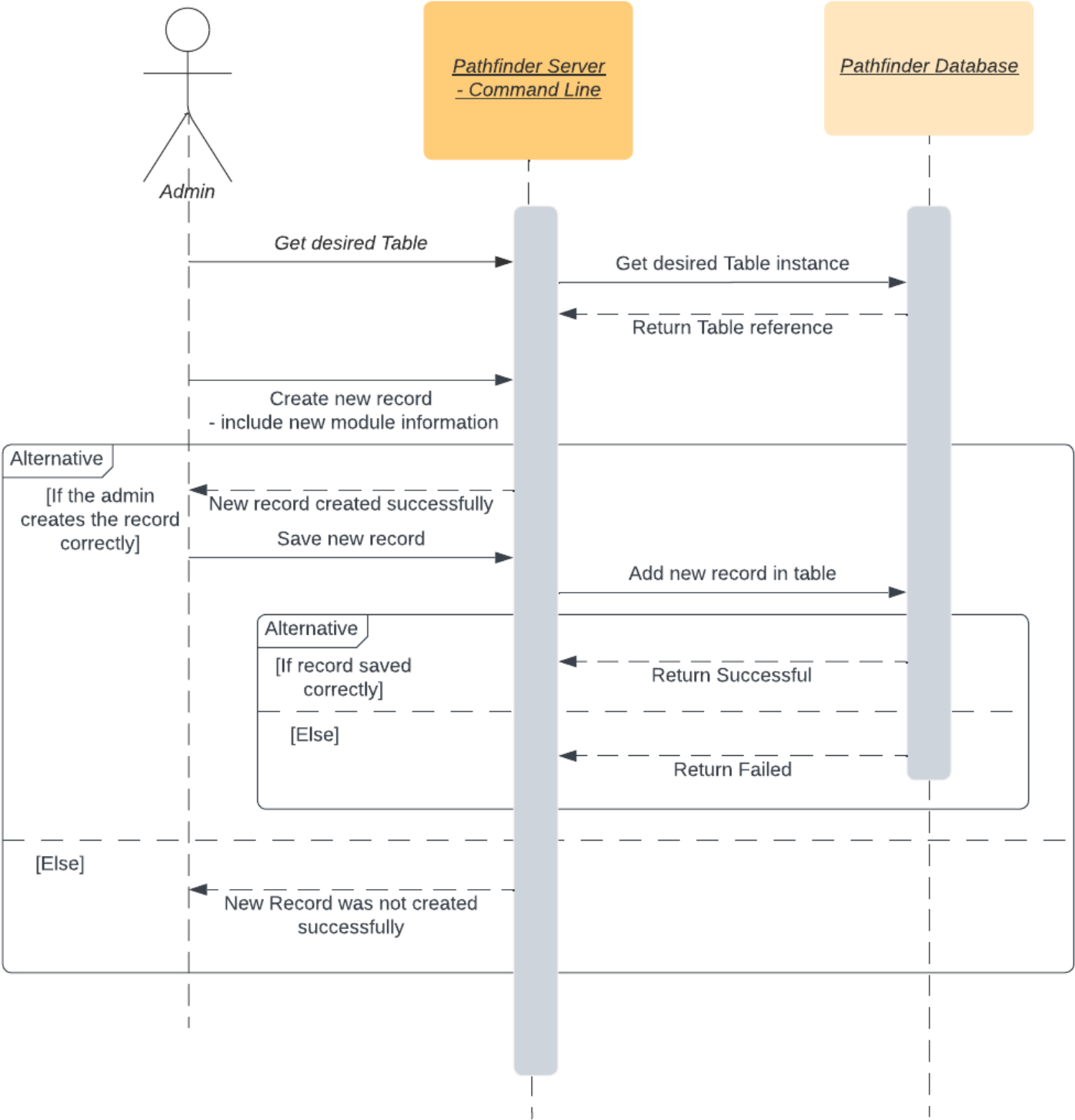
Flow of Events for the “Update Career” use-case	
Objective	Allow the admin to update the career table with industry jobs and careers
Precondition	Admin user has access to the database
Main Flow	<ol style="list-style-type: none"> 1. The admin wants to add a new career to the system. 2. The admin takes 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	<ol style="list-style-type: none"> 4. The admin decides to update an existing career. The admin selects the career to update, and they are shown all the career’s 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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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 course that is no longer relevant. 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.

C.5: Sequence Diagrams

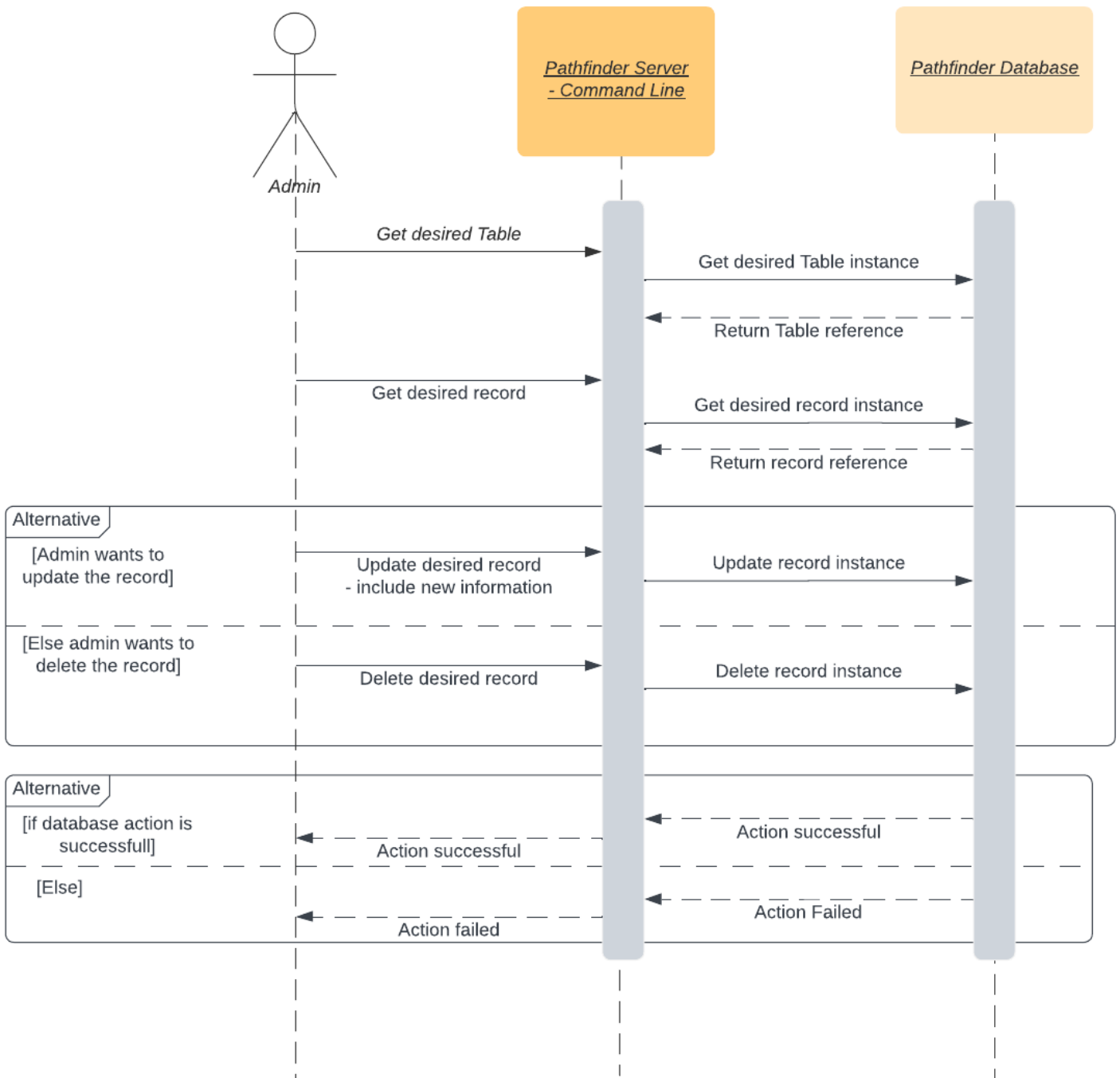
Admin - Command Line: Add new record - Any Table

Ross McAllister | April 29, 2023



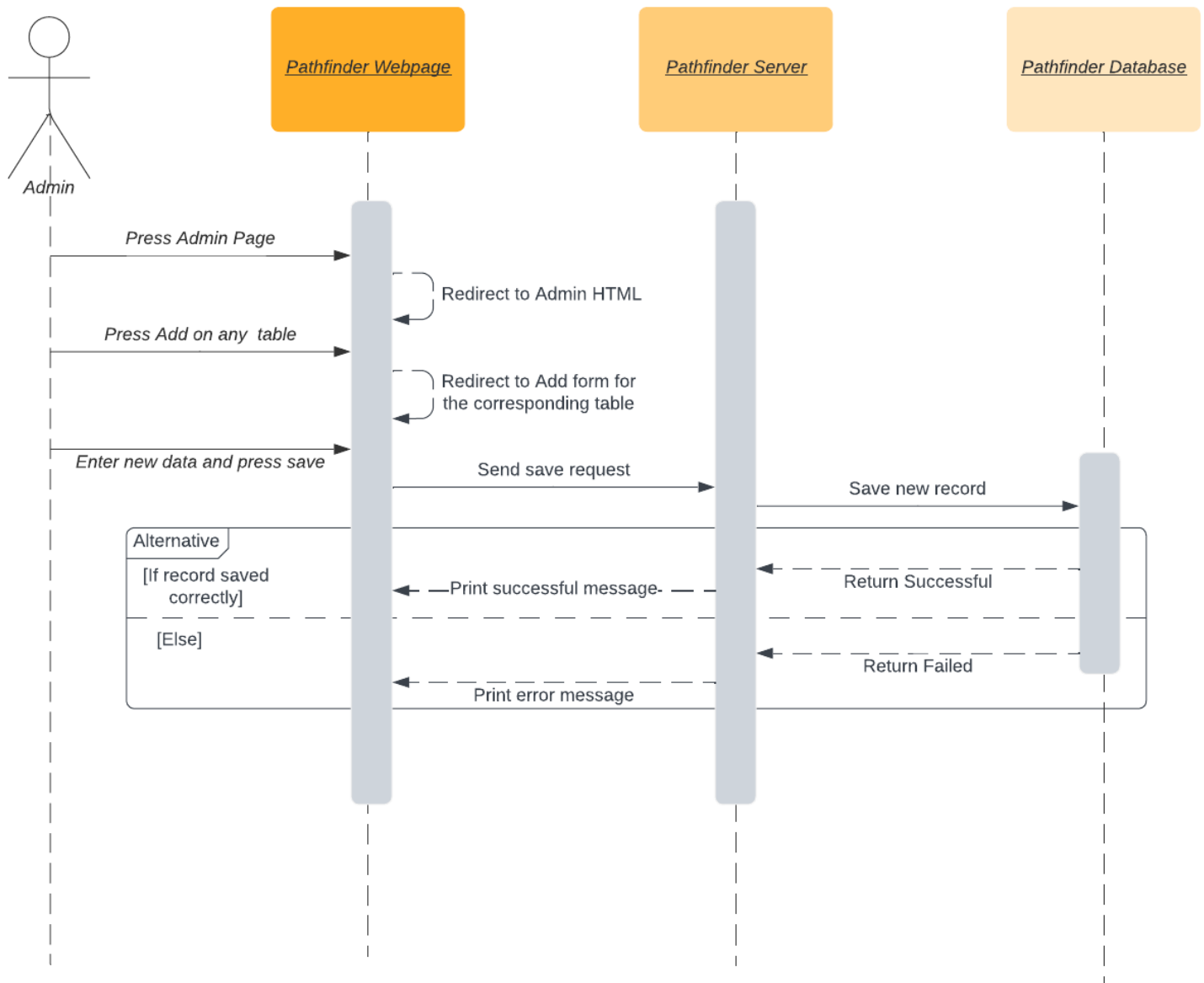
Admin - Command Line: Update/Delete record - Any Table

Ross McAllister | April 29, 2023



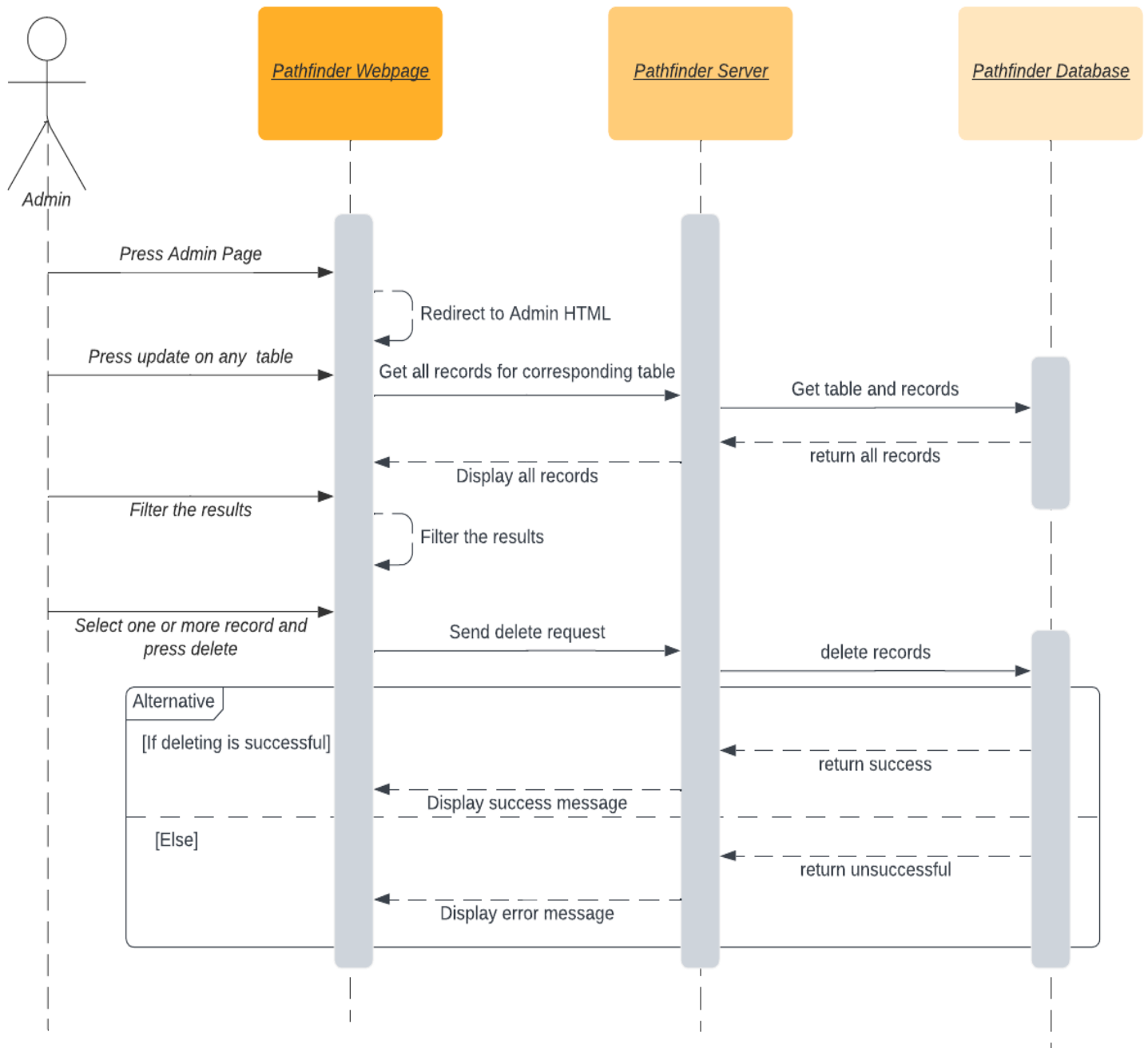
Admin UI - Add new records

Ross McAllister | April 29, 2023



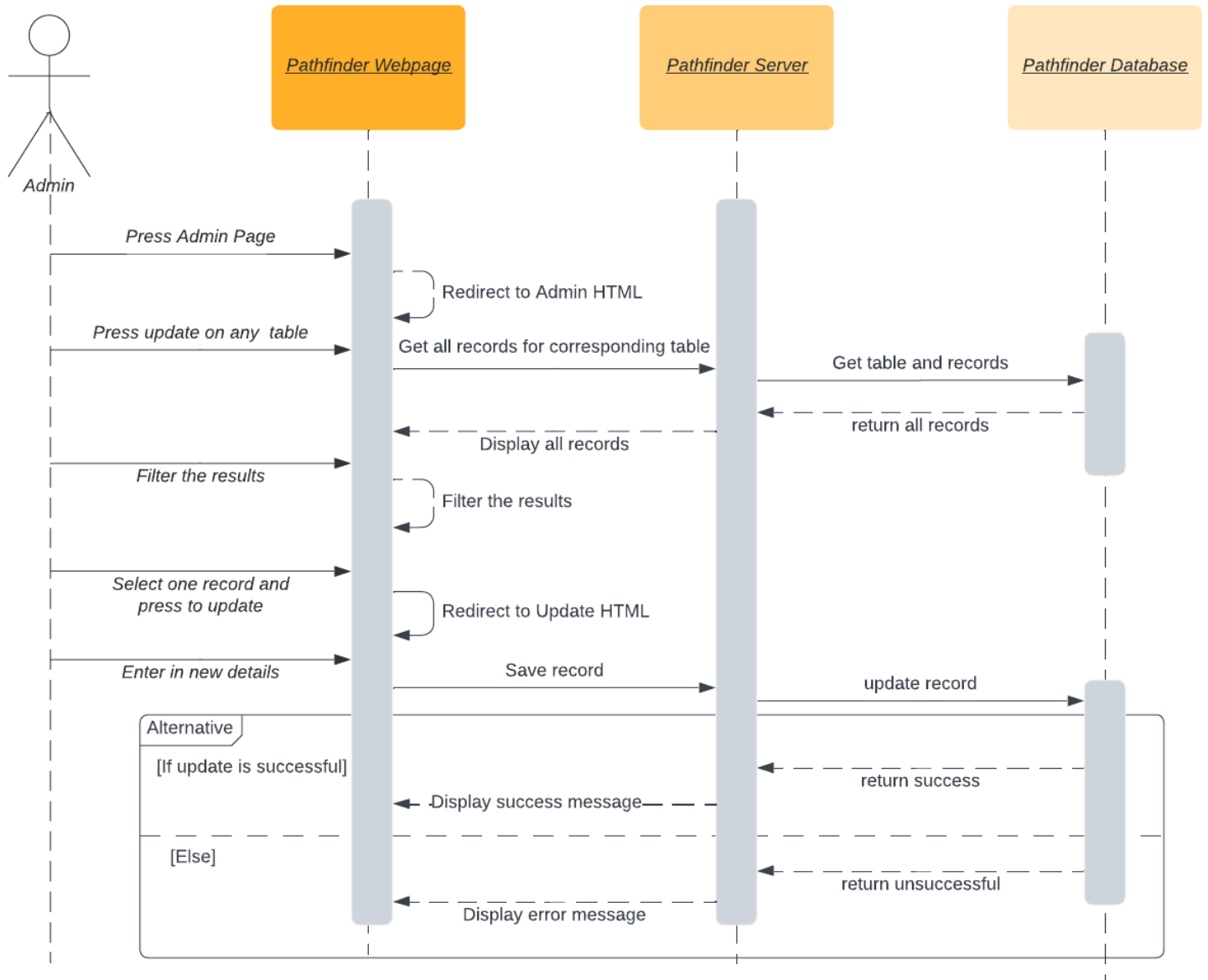
Admin UI - Delete records

Ross McAllister | April 29, 2023



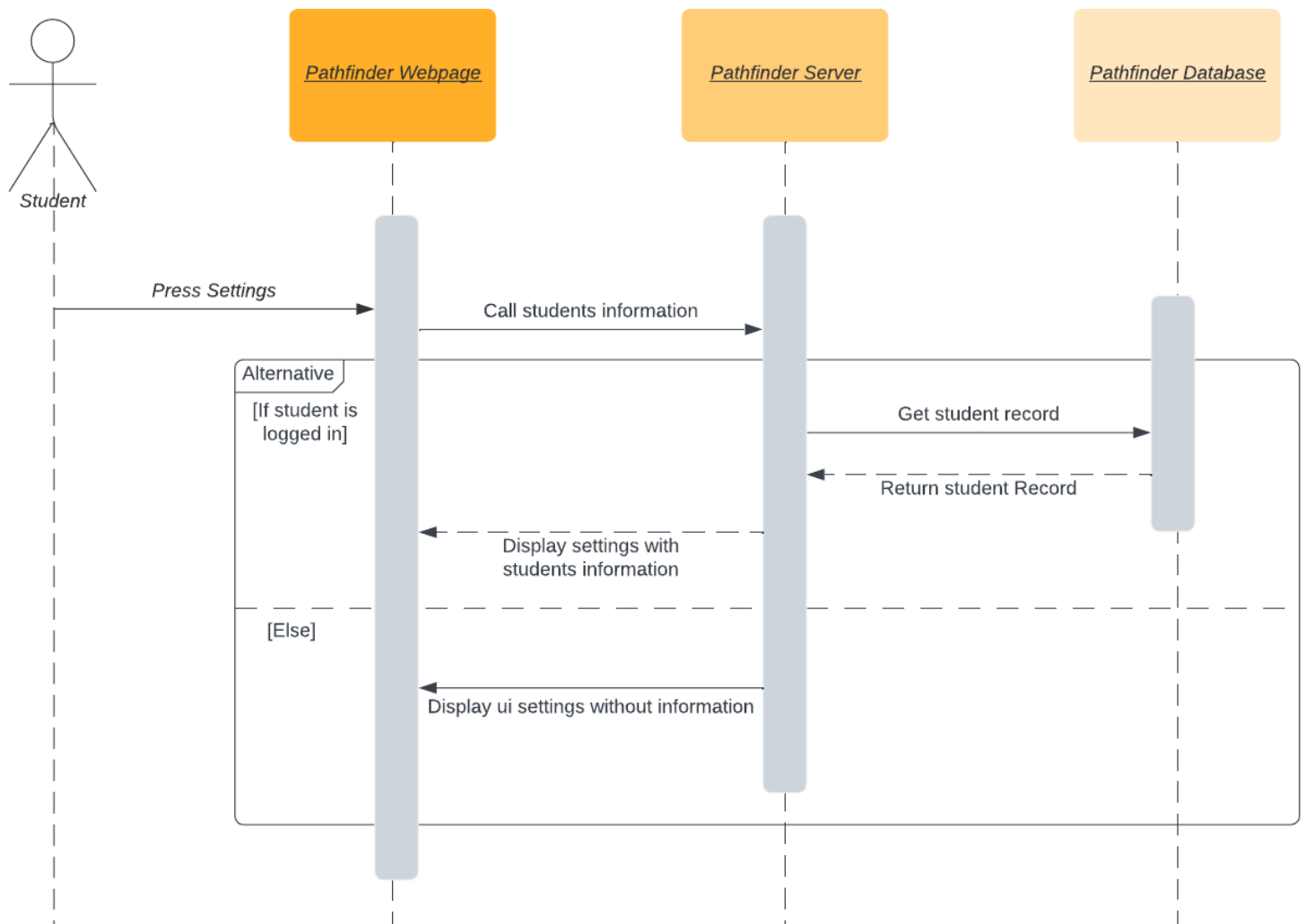
Admin UI - Update record

Ross McAllister | April 29, 2023



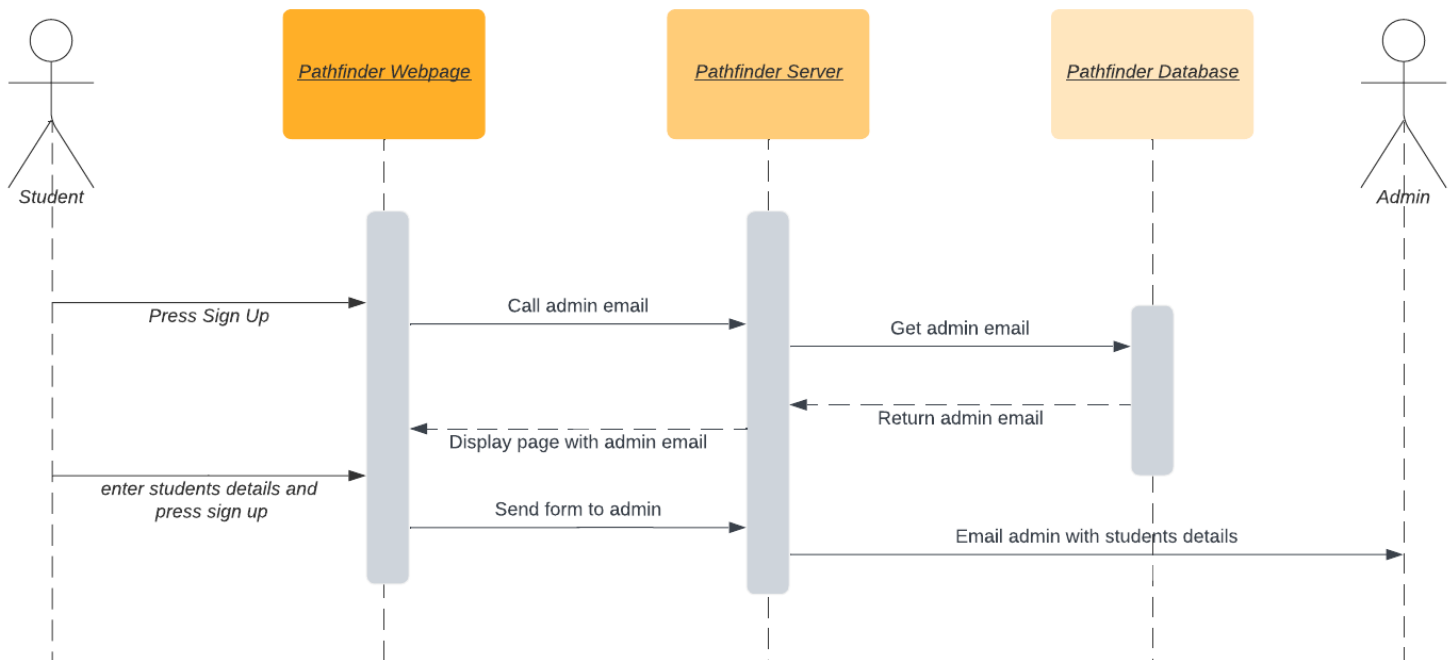
View Settings

Ross McAllister | April 29, 2023



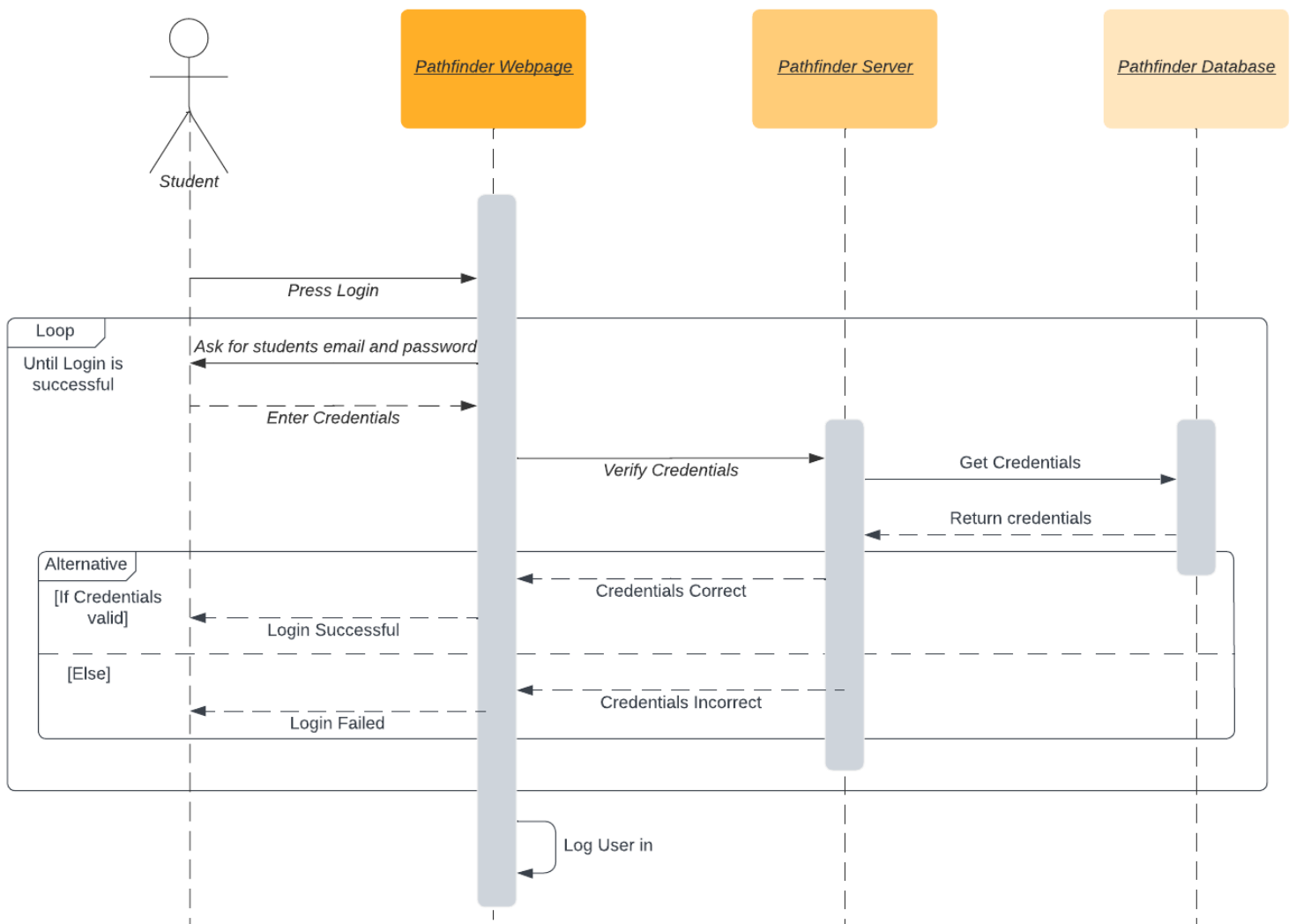
Student Sign Up

Ross McAllister | April 29, 2023



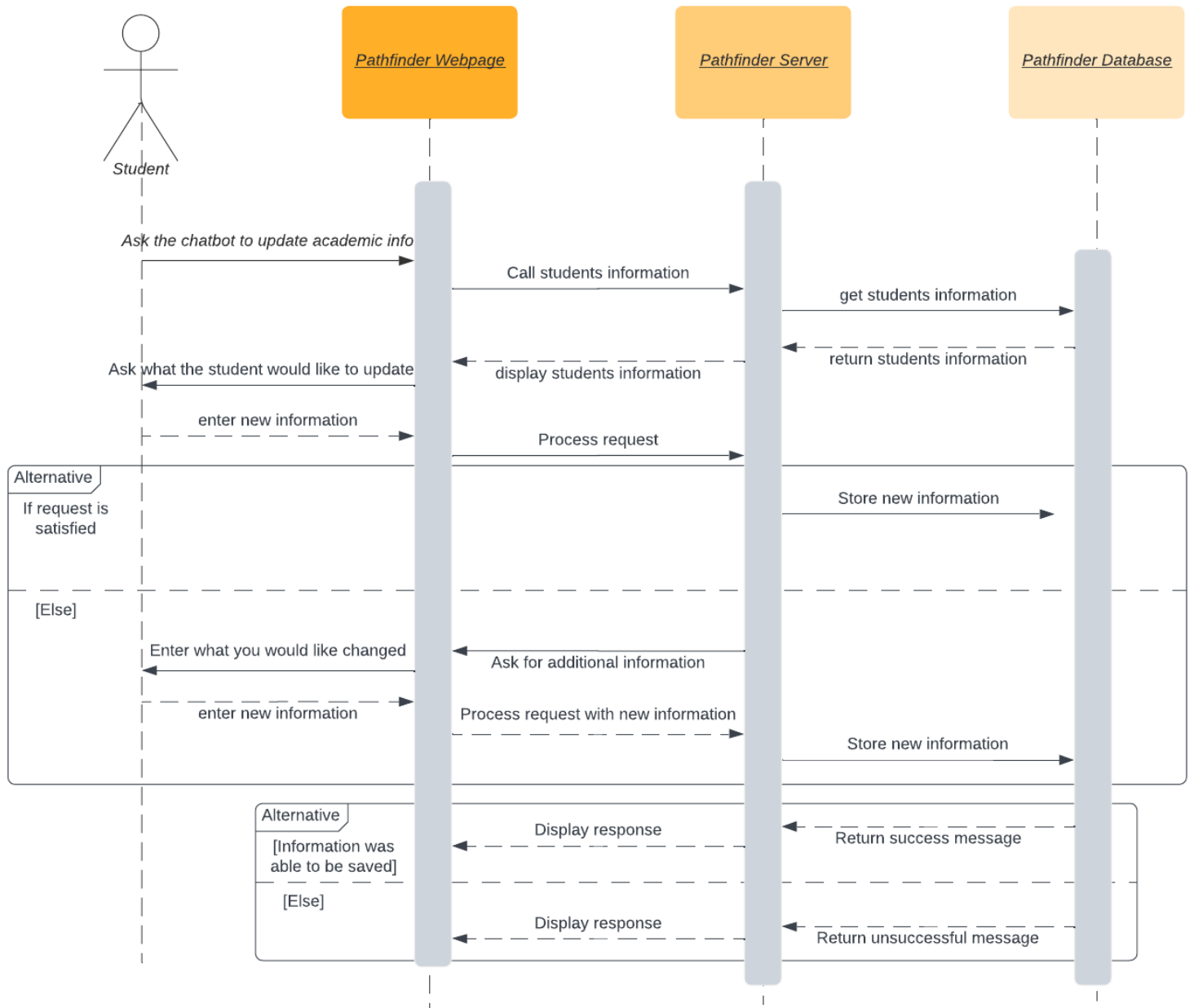
Student Login

Ross McAllister | April 29, 2023



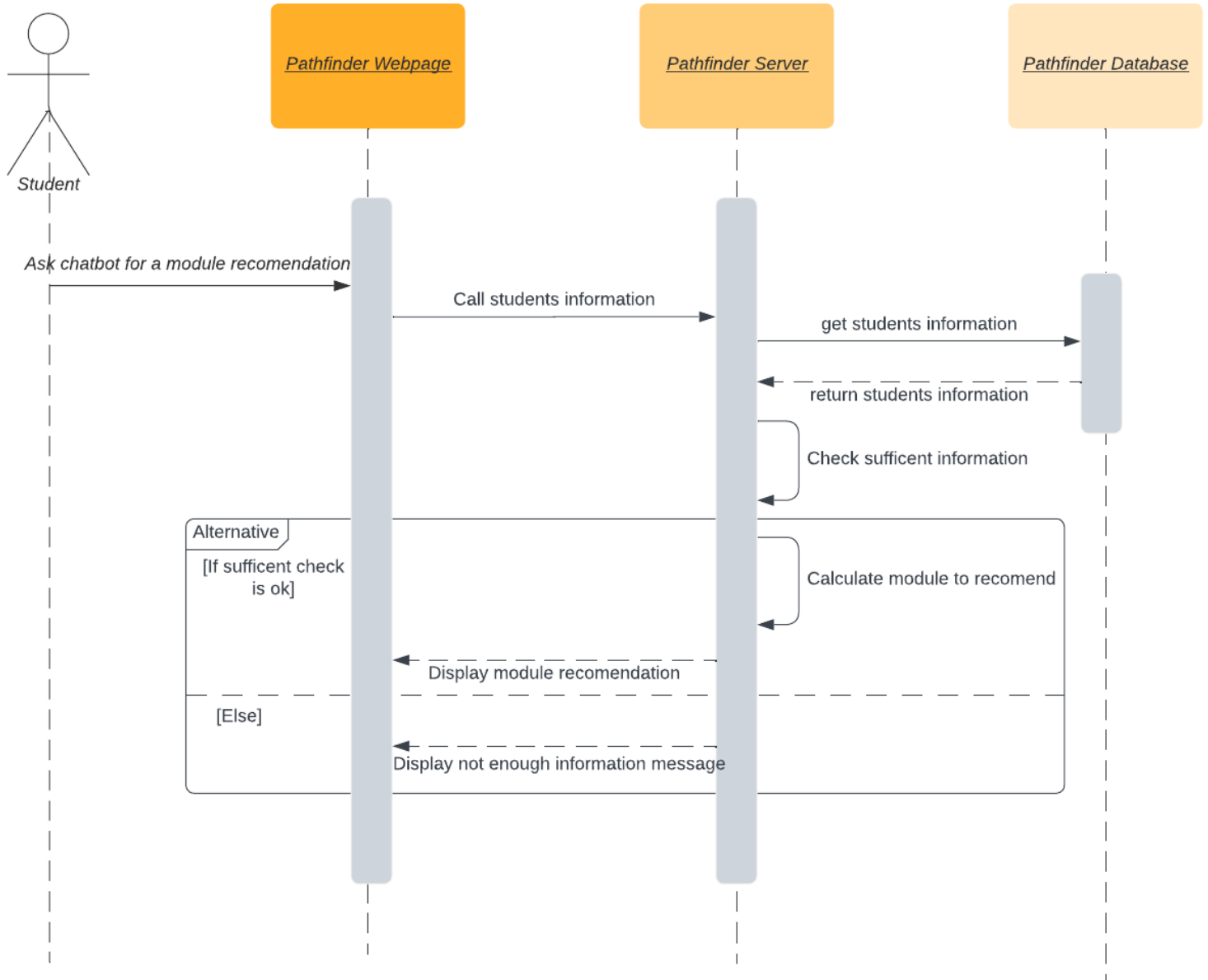
Check academic information

Ross McAllister | April 29, 2023



Module Recommendation

Ross McAllister | April 29, 2023

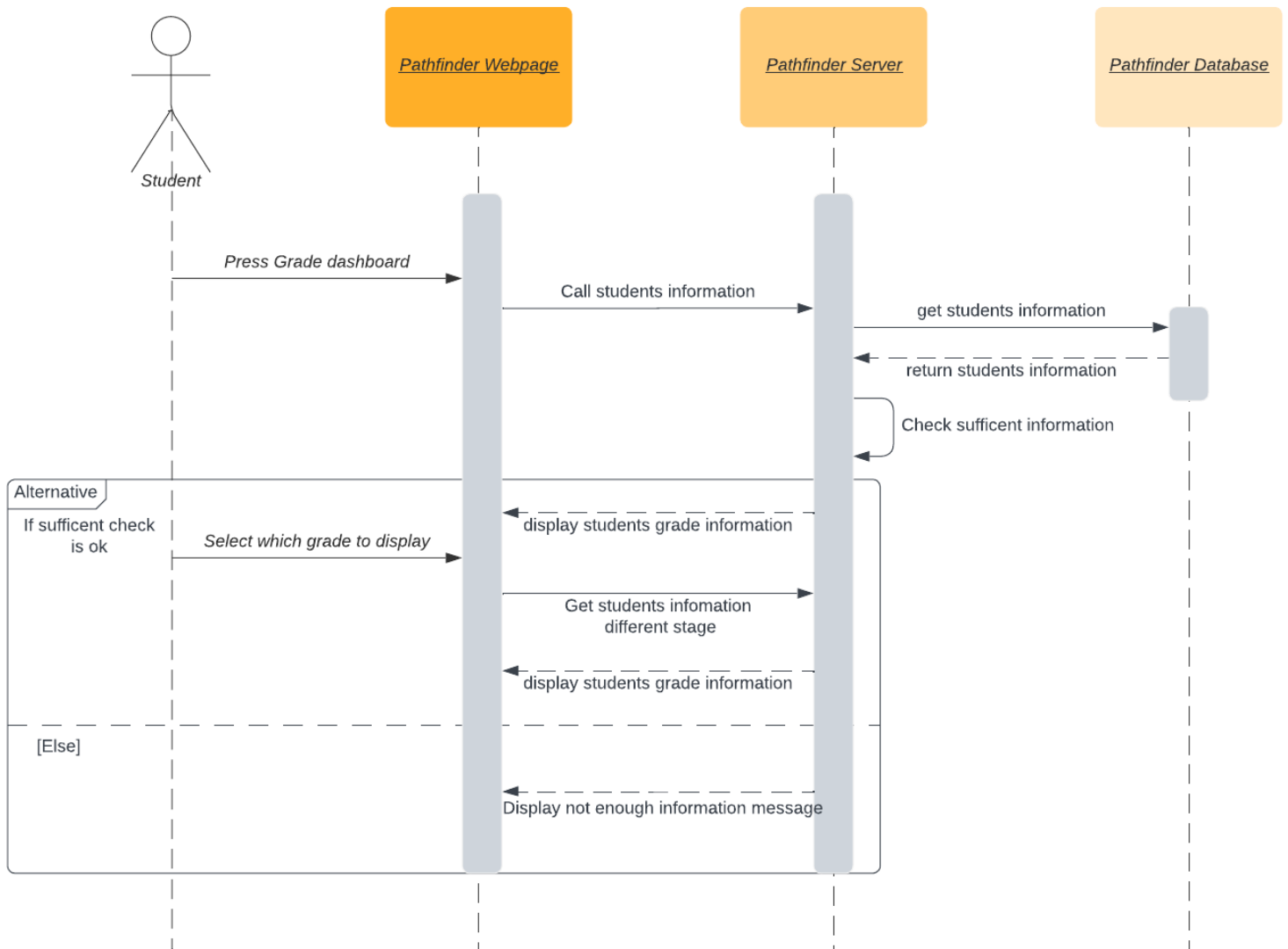


View Module Information

Ross McAllister | April 29, 2023

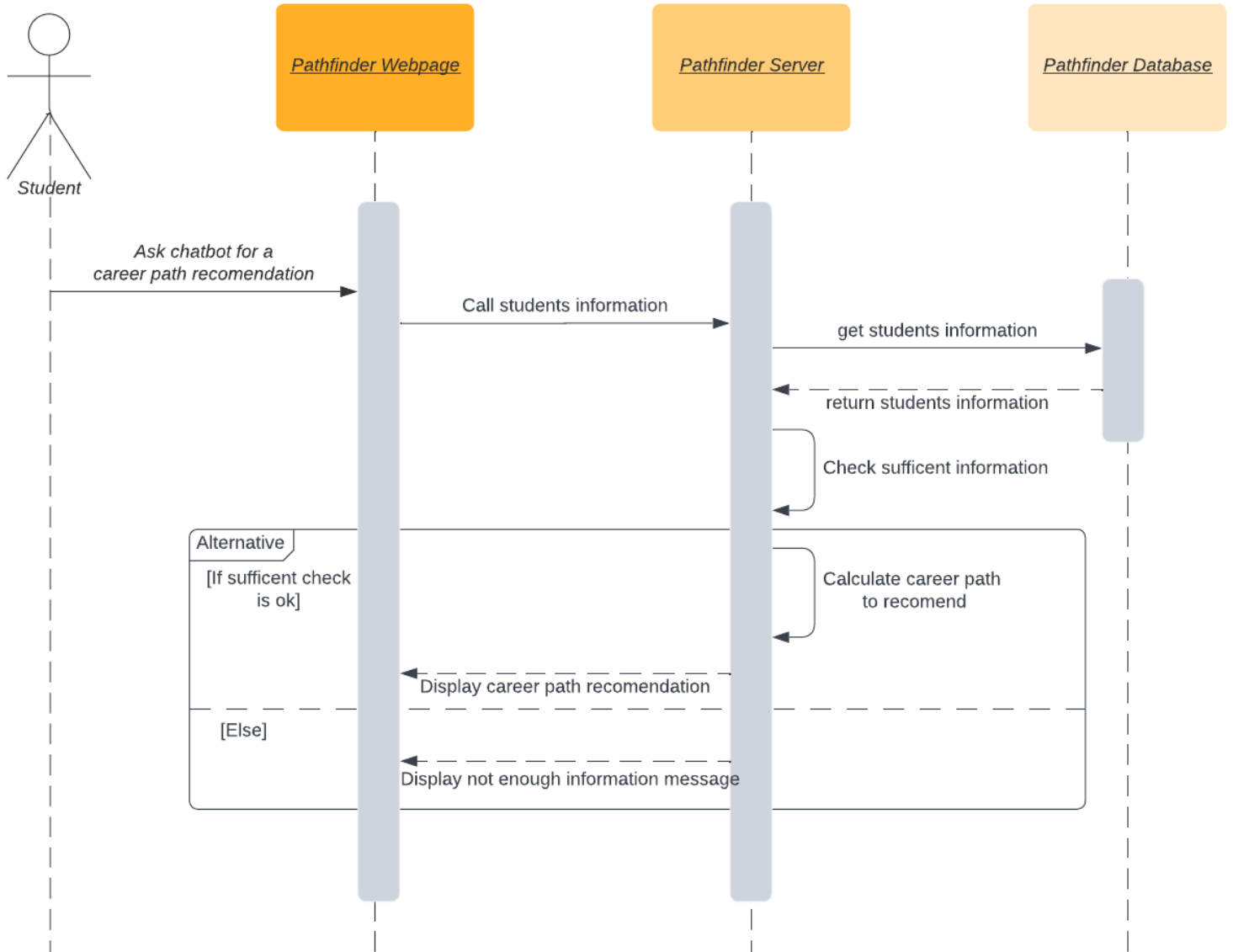
Get Course Stats

Ross McAllister | April 29, 2023



Career Path Recommendation

Ross McAllister | April 29, 2023



D) Appendix: Wireframes

Home Page with sidebar closed

Pathfinder Chatbot

Enter Text

Home Page with sidebar open

Pathfinder

Home

Module Information

Queen's Website

Settings

Login

Sign-Up

Pathfinder Chatbot

Enter Text

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

When user is not logged in

 Pathfinder

Home

Module Information

Queen's Website

Settings

Login

Sign-Up

Admins side bar

 Pathfinder

Home

Module Information


Queen's Website

Settings

Admin Panel

Logout

Student side bar

 Pathfinder

Home

Module Information

Queen's Website

Settings

Grade Dashboard

Logout

Login pop up

Student Number

Enter Student Number

Password

Enter Password

Login

Cancel

[Forgot Password?](#)

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

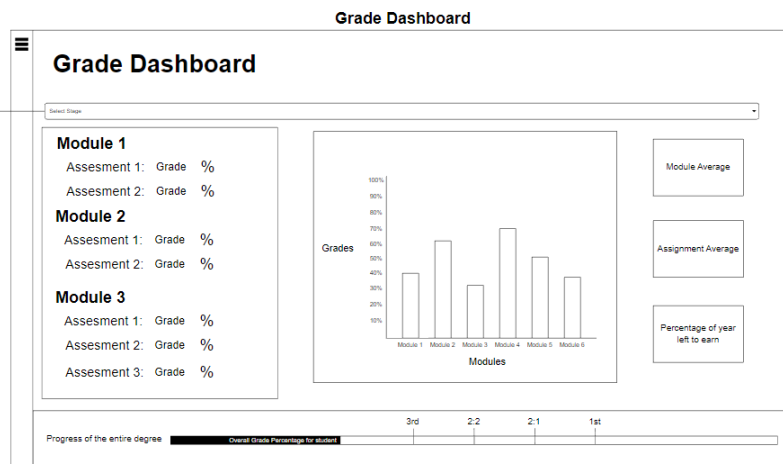
Stage

☒ 1

☐ 2

☐ 3

☐ 4



Sign Up Pop Up

Enter Account Information

All fields must be filled in.

Name:

Email:

Password:

Student Number:

Pathway:

Current Semester:

Current Stage:

Next

Sign Up Request Submitted

Your information has been sent to the admin
Once the admin creates your, they'll send an email with your login details
If you have any questions please contact the admin here: admin@email.com

Module Information Page

Module Information

Search

Filter

Module Name - Code				
Lecturer	Stage	Semester	Weighting	Required
List of pathways that the module is available on				
Assessment name - Weighting				
Assessment name - Weighting				
Description				

Example of what the filter dropdown will look like

Stage

1 ☒

2 ☒

3 ☒

4 ☒

Semester

1 ☒

2 ☒

Pathways

CS ☒

BIT ☒

SE ☒

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

No User Logged Into The System, Settings Page



Settings


Accessibility

Theme: ☐ Light ☐ Dark ☐ High Contrast (Black & White)

Font Size: ☐ Small ☒ Medium ☐ Large

Save Changes

Student Settings Page



Settings

Account Information

Name: Users name

Email: Users email

Student Number: Users student number

Pathway: Users pathway

Current Semester: Users current semester

Current Stage: Users current stage

If any of this information is incorrect please contact the admin here: admin@email.com


Accessibility

Theme: ☐ Light ☐ Dark ☐ High Contrast (Black & White)

Font Size: ☐ Small ☒ Medium ☐ Large

Save Changes

Admin Settings Page



Settings

Account Information

Name:

Email:

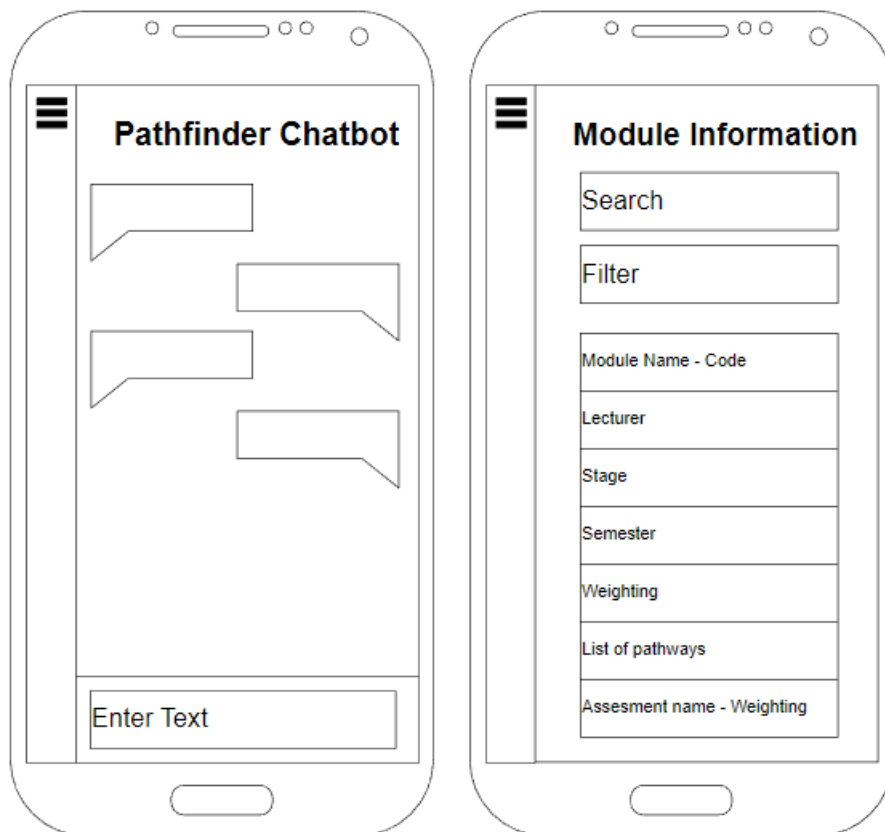
Accessibility

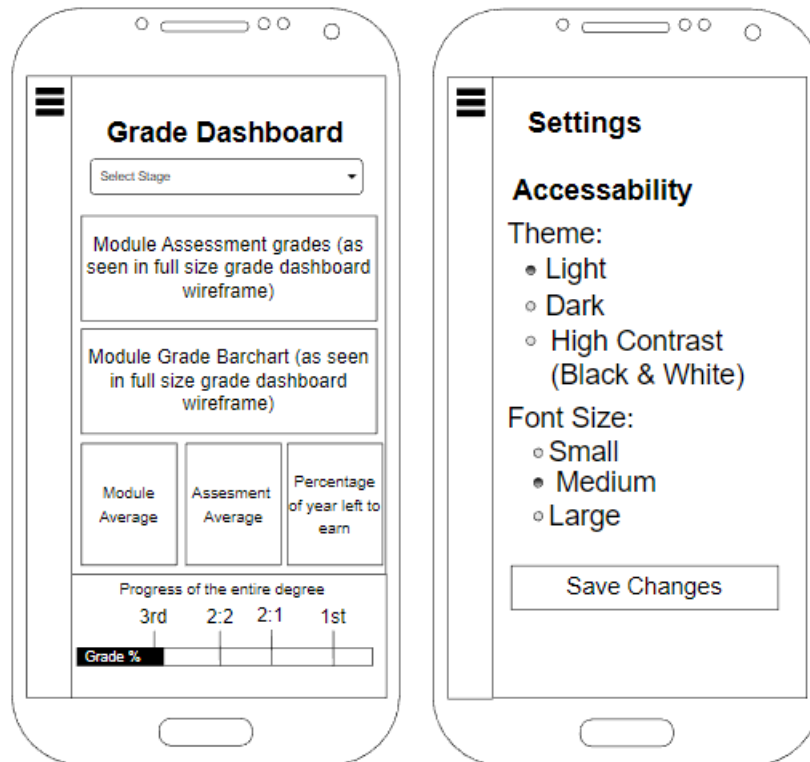
Theme: ☒ Light ☐ Dark ☐ High Contrast (Black & White)

Font Size: ☐ Small ☒ Medium ☐ Large

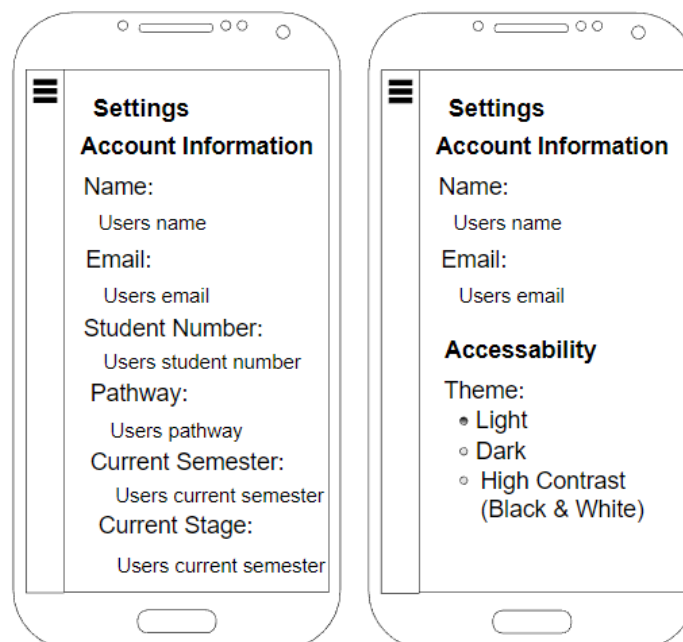
Save Changes

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





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



E) Appendix: Implementation

Code Snippets / Github: <https://github.com/KyleMcComb/CSC3068-Pathfinder>

Interest Adapter Code Snippet part 1:

```
CSC3068-Pathfinder / Pathfinder / main / interest_adapter.py
KyleMcComb Added option to choose which module user is
Code Blame 84 lines (74 loc) · 4.37 KB
1 import re
2 from chatterbot.logic import LogicAdapter
3 from chatterbot.conversation import Statement
4 from database.models import Module, StudentInterest, Student
5 from django.core.exceptions import MultipleObjectsReturned
6
7 class InterestAdapter(LogicAdapter):
8     def __init__(self, chatbot, **kwargs):
9         super().__init__(chatbot, **kwargs)
10        self.awaiting_module_choice = False
11        self.modules_to_choose = []
12        self.student_id = 40191566 # using this id as a placeholder atm.
13
14    def can_process(self, statement):
15        statement_text = statement.text.lower()
16        return self.awaiting_module_choice or re.search(r'\b(like|dislike|hate)\b', statement_text) is not None
17
18    def process(self, input_statement, additional_response_selection_parameters=None):
19        statement_text = input_statement.text.lower()
20        response = Statement('')
21
22        if self.awaiting_module_choice:
23            try:
24                choice = int(statement_text)
25                if 0 <= choice <= len(self.modules_to_choose):
26                    if choice != 0:
27                        module = self.modules_to_choose[choice - 1]
28                        student = Student.objects.get(studentID=self.student_id)
29                        try:
30                            interest_obj = StudentInterest.objects.filter(studentID=student, interestName=module.moduleName).first()
31                            if interest_obj:
32                                interest_obj.interestImportance = 1
33                                interest_obj.save()
34                            else:
35                                interest_obj = StudentInterest.objects.create(studentID=student, interestName=module.moduleName, interestImportance=1)
36                                interest_obj.save()
37                        except MultipleObjectsReturned:
38                            interest_obj = StudentInterest.objects.filter(studentID=student, interestName=module.moduleName).first()
39                            interest_obj.interestImportance = 1
40                            interest_obj.save()
41                        response.text = f"Thanks for your input! I've updated your interest for {module.moduleName}."
42                    else:
43                        response.text = "Thanks for your input! I will not update your interest for the given modules."
44                else:
45                    response.text = f"Sorry, you need to enter a number between 0 and {len(self.modules_to_choose)}."
46            except ValueError:
47                response.text = f"Sorry, you need to enter a number between 0 and {len(self.modules_to_choose)}."
48
```

Interest Adapter Code Snippet part 2:

```
48
49     self.awaiting_module_choice = False
50     self.modules_to_choose = []
51
52     else:
53         dislike_match = re.search(r'\bdislike\b', statement_text)
54         hate_match = re.search(r'\bhate\b', statement_text)
55         like_match = re.search(r'\blike\b', statement_text)
56
57         if dislike_match:
58             interest = statement_text[dislike_match.end():].strip()
59             response_type = 'dislike'
60         elif hate_match:
61             interest = statement_text[hate_match.end():].strip()
62             response_type = 'hate'
63         elif like_match:
64             interest = statement_text[like_match.end():].strip()
65             response_type = 'like'
66
67         student = Student.objects.get(studentID=self.student_id)
68         modules = Module.objects.filter(moduleName__icontains=interest)
69
70         if response_type == 'like':
71             if modules.count() >= 1:
72                 response.text = f"That's great! Here are some modules related to {interest}:<br>"
73                 for i, module in enumerate(modules, start=1):
74                     response.text += f"{i}. {module.moduleID}: {module.moduleName}<br>"
75                 response.text += f"Which one are you most interested in? Please enter the corresponding number or enter 0 if you are not interested in any."
76
77                 self.awaiting_module_choice = True
78                 self.modules_to_choose = list(modules)
79             else:
80                 response.text = f"You like {interest}. I will try and recommend any modules that contain {interest}."
81         else:
82             response.text = f"Thanks for providing me with this information. I won't recommend any modules that contain {interest}."
83
84         response.confidence = 1
85     return response
```

Year Adapter Code Snippet:

CSC3068-Pathfinder / Pathfinder / main / year_adapter.py

KyleMcComb Added code to output all modules based off the

Code Blame 44 lines (38 loc) · 2.52 KB

```
1  import re
2  from chatterbot.logic import LogicAdapter
3  from chatterbot.conversation import Statement
4  from database.models import Module
5
6  class YearAdapter(LogicAdapter):
7      def __init__(self, chatbot, **kwargs):
8          super().__init__(chatbot, **kwargs)
9
10     def can_process(self, statement):
11         statement_text = statement.text.lower()
12         return re.search(r'^(first year|second year|final year|fourth year|1st year|2nd year|3rd year|4th year|year 1|year 2|year 3|year 4|stage 1|stage 2|stage 3)\b', statement_text) is not None
13
14     def process(self, input_statement, additional_response_selection_parameters=None):
15         statement_text = input_statement.text.lower()
16         response = Statement("")
17
18         year_match = re.search(r'^(first year|second year|final year|fourth year|1st year|2nd year|3rd year|4th year|year 1|year 2|year 3|year 4|stage 1|stage 2|stage 3)\b', statement_text)
19         if year_match:
20             year = year_match.group()
21
22             if year in ["first year", "1st year", "year 1", "stage 1"]:
23                 first_year_modules = Module.objects.filter(moduleID__startswith="CSC1")
24                 response.text = "Good to know. Here are all the first-year modules you can do for EECS:<br>"
25                 for module in first_year_modules:
26                     response.text += f"- {module.moduleID}: {module.moduleName}<br>"
27             elif year in ["second year", "2nd year", "year 2", "stage 2"]:
28                 second_year_modules = Module.objects.filter(moduleID__startswith="CSC2")
29                 response.text = "Good to know. Here are all the second-year modules you can do for EECS:<br>"
30                 for module in second_year_modules:
31                     response.text += f"- {module.moduleID}: {module.moduleName}<br>"
32             elif year in ["final year", "3rd year", "year 3", "stage 3"]:
33                 final_year_modules = Module.objects.filter(moduleID__startswith="CSC3")
34                 response.text = "Good to know. Here are all the final year modules you can do for EECS:<br>"
35                 for module in final_year_modules:
36                     response.text += f"- {module.moduleID}: {module.moduleName}<br>"
37             elif year in ["fourth year", "4th year", "year 4", "stage 4"]:
38                 fourth_year_modules = Module.objects.filter(moduleID__startswith="CSC4")
39                 response.text = "Good to know. Here are all the fourth-year modules you can do for EECS:<br>"
40                 for module in fourth_year_modules:
41                     response.text += f"- {module.moduleID}: {module.moduleName}<br>"
42
43         response.confidence = 1
44         return response
```

Chatbot Configuration Code Snippet:

```
CSC3068-Pathfinder / Pathfinder / main / chatbot_settings.py
DeanLogan123 Added docker files to the master branch, along with updating the README

Code Blame 43 lines (37 loc) · 1.32 KB

1 from chatterbot import ChatBot
2 from chatterbot.trainers import ChatterBotCorpusTrainer
3 from .interest_adapter import InterestAdapter
4 from .year_adapter import YearAdapter
5 import os
6
7 chatbot = ChatBot(
8     'Pathfinder',
9     storage_adapter='chatterbot.storage.SQLStorageAdapter',
10    logic_adapters=[
11        {
12            'import_path': 'main.interest_adapter.InterestAdapter' # Use the custom InterestAdapter
13        },
14        {
15            'import_path': 'main.year_adapter.YearAdapter' # Use the custom YearAdapter
16        },
17        {
18            'import_path': 'chatterbot.logic.BestMatch',
19            'default_response': 'I am sorry, but I do not understand.',
20            'maximum_similarity_threshold': 0.9
21        }
22    ],
23    database_uri='sqlite:///database.sqlite3'
24 )
25
26 trainer = ChatterBotCorpusTrainer(chatbot)
27
28 ROOT_DIR = os.path.dirname(os.path.abspath(__file__))
29 DATASET_PATH = os.path.join(ROOT_DIR, 'data/my_corpus.yml')
30
31 # Train the chatbot with the English corpus
32 #trainer.train("chatterbot.corpus.english")
33 #trainer.train("chatterbot.corpus.english.greetings")
34
35 # Error handling
36 try:
37     trainer.train(DATASET_PATH)
38 except Exception as e:
39     print(f"Error during training: {e}")
40
41 def get_response(msg):
42     msg_lowercase = msg.lower() # Code to convert user response to lowercase
43     return str(chatbot.get_response(msg_lowercase))
```

Pseudo Code for Chatbot & Logic Adapters:

Interest Adapter Pseudo Code:

```
FUNCTION InterestAdapter:
```

```
    Initialize InterestAdapter
```

```
    FUNCTION can_process(statement):
```

```
        IF awaiting_module_choice OR 'like', 'dislike', 'hate' FOUND in  
statement:
```

```
            RETURN True
```

```
        ELSE:
```

```
            RETURN False
```

```
    FUNCTION process(input_statement):
```

```
        IF awaiting_module_choice:
```

```
            Process user's module choice
```

```
        ELSE:
```

```
            Detect 'like', 'dislike', or 'hate' keyword
```

```
            Process user's interest based on keyword
```

```
            IF 'like':
```

```
                Suggest relevant modules and prompt for user's preferred  
module
```

```
            ELSE:
```

```
                Store user's preference to avoid recommending disliked modules
```

```
        RETURN response
```

Year Adapter Pseudo Code:

```
FUNCTION YearAdapter:
```

```
    Initialize YearAdapter
```

```
    FUNCTION can_process(statement):
```

```
        IF 'first year', 'second year', 'final year', etc. FOUND in  
        statement:
```

```
            RETURN True
```

```
        ELSE:
```

```
            RETURN False
```

```
    FUNCTION process(input_statement):
```

```
        Detect year-related keyword in statement
```

```
        IF 'first year', '1st year', etc.:
```

```
            Retrieve and display first-year modules
```

```
        ELSE IF 'second year', '2nd year', etc.:
```

```
            Retrieve and display second-year modules
```

```
        ELSE IF 'final year', '3rd year', etc.:
```

```
            Retrieve and display final-year modules
```

```
        ELSE IF 'fourth year', '4th year', etc.:
```

```
            Retrieve and display fourth-year modules
```

```
    RETURN response
```

Pseudo Code for Chatbot Configuration file:

FUNCTION Chatbot:

Initialize Chatbot with:

- SQLStorageAdapter
- InterestAdapter
- YearAdapter
- BestMatch logic adapter

Setup database for chatbot

Create ChatterBotCorpusTrainer instance with chatbot

Train chatbot with custom dataset

FUNCTION get_response(msg):

Convert msg to lowercase

Get chatbot response for msg

RETURN response

F) Appendix: Additional Test Cases

Green: Passed

Orange: Partially Passed

Red: Failed

GUI-Chatbot

Test Case ID: GUI-Chatbot-001-01
Test Case Description: Ensure that the text input field is displayed on the website.
Test Steps: <ol style="list-style-type: none">1. Open the website.2. Check that a text input field is displayed on the page.3. Click on the text input field to ensure that it is selectable.4. Attempt to enter text into the text input field.5. Verify that the text appears in the field.
Expected Outcome: Text input field is displayed on the website.
Actual Outcome: Text input field is displayed on the website.
Pass/Fail?: Pass

Test Case ID: GUI-Chatbot-002-01
Test Case Description: Verify that the chat history is displayed on the website in a readable manner.
Test Steps: <ol style="list-style-type: none"> 1. Open the website and interact with the chatbot. 2. Send several messages to the chatbot and receive responses. 3. Look for a chat history section on the page and verify that it is displaying the messages and responses in a clear and readable manner. 4. Verify that it is easy to distinguish between the user's messages and the chatbot's responses. 5. Verify that the chat history is scrollable and that older messages are not cut off or truncated.
Expected Outcome: Chat history is displayed on the website in a readable manner.
Actual Outcome: Chat history is displayed on the website in a readable manner.
Pass/Fail?: Pass

GUI-Database

Test Case ID: GUI-Database-001-01
Test Case Description: Verify that the tables in the database are displayed in a readable manner.
Test Steps: <ol style="list-style-type: none"> 1. Access the database interface. 2. Navigate to the section where the tables are displayed. 3. Verify that the tables are displayed in a clear and easy-to-read format including column headers. 4. Verify that the data in each table cell is readable and not truncated or cut off.
Expected Outcome: The tables in the database are displayed in a readable manner.
Actual Outcome: The tables in the database are displayed in a readable manner.
Pass/Fail?: Pass

Test Case ID: GUI-Database-001-02
Test Case Description: Verify that users can't see data they do not have access to.
Test Steps: <ol style="list-style-type: none"> 1. Log in to the database interface as a user with limited access. 2. Attempt to view data from tables that the user does not have access to. 3. Verify that the user is not able to view any data from tables they do not have access to. 4. Verify that any attempts to bypass access controls, such as by manipulating the URL or using browser developer tools, are unsuccessful. 5. Attempt to log in as a user with higher privileges and verify that they are able to view data from all tables they have access to.
Expected Outcome: Users can't see data they do not have access to.
Actual Outcome: Users can't see data they do not have access to.
Pass/Fail?: Pass

Test Case ID: GUI-Database-002-01
Test Case Description: Verify that an error message is displayed when attempting to add a new record with invalid data.
Test Steps: <ol style="list-style-type: none"> 1. Access the database interface. 2. Navigate to the "Add New Record" section. 3. Enter invalid data into one or more of the fields. 4. Attempt to submit the record.
Expected Outcome: An error message should be displayed indicating which field(s) contain invalid data and how to correct it.
Actual Outcome: An error message is displayed indicating which field(s) contain invalid data and how to correct it.
Pass/Fail?: Pass

Test Case ID: GUI-Database-002-02
Test Case Description: Verify that an error message is displayed when attempting to update a new record with invalid data.
Test Steps: <ol style="list-style-type: none"> 1. Access the database interface. 2. Navigate to a record that needs to be updated. 3. Modify one or more fields with invalid data. 4. Attempt to save the updated record.
Expected Outcome: An error message should be displayed indicating which field(s) contain invalid data and how to correct it.
Actual Outcome: An error message is displayed indicating which field(s) contain invalid data and how to correct it.
Pass/Fail?: Pass

GUI-Grade Dashboard

Test Case ID: GUI-GradeDashboard-001-01
Test Case Description: Verify that the graph displays the correct overall grade for a student.
Test Steps: <ol style="list-style-type: none"> 1. Login as a student. 2. Access the Grade Dashboard section. 3. Check if the graph displays the correct overall grade for the student.
Expected Outcome: The graph displays the correct overall grade for the student.
Actual Outcome: The graph displays the correct overall grade for the student.
Pass/Fail?: Pass

Test Case ID: GUI-GradeDashboard-001-02
Test Case Description: Verify that the graph displays the current semester grades for a student.
Test Steps: <ol style="list-style-type: none"> 1. Login as a student. 2. Access the Grade Dashboard section. 3. Check if the graph displays the current semester grades for the student.
Expected Outcome: The graph displays the current semester grades for the student.
Actual Outcome: The graph displays the current semester grades for the student.
Pass/Fail?: Pass

Test Case ID: GUI-GradeDashboard-001-03
Test Case Description: Verify that the graph displays what percentage of the degree the student has completed compared to how much they have left to complete.
Test Steps: <ol style="list-style-type: none"> 1. Login as a student. 2. Access the Grade Dashboard section. 3. Check if the graph displays the percentage of the degree the student has completed compared to how much they have left to complete.
Expected Outcome: The graph displays what percentage of the degree the student has completed compared to how much they have left to complete.
Actual Outcome: The graph displays what percentage of the degree the student has completed compared to how much they have left to complete.
Pass/Fail?: Pass

GUI

Test Case ID: GUI-001-01
Test Case Description: Verify that the website includes a high contrast/colour blind option.
Test Steps: <ol style="list-style-type: none">1. Open the website.2. Check that a high contrast/colour blind option is displayed on the page.3. Click on the high contrast/colour blind option to ensure that it is selectable.
Expected Outcome: High contrast/colour blind option is displayed on the screen and is selectable.
Actual Outcome: High contrast/colour blind option is displayed on the screen and is selectable.
Pass/Fail?: Pass

Test Case ID: GUI-001-02
Test Case Description: Verify that the website's colour scheme can be changed to white and black.
Test Steps: <ol style="list-style-type: none">1. Open the website.2. Navigate to the high contrast/colour blind option.3. Select the option to change the website's colour scheme to white and black.
Expected Outcome: The website's colour scheme is changed to white and black.
Actual Outcome: The website's colour scheme is changed to white and black.
Pass/Fail?: Pass

Test Case ID: GUI-002-01
Test Case Description: Verify that the website includes a toggle for light and dark mode.
Test Steps: <ol style="list-style-type: none"> 1. Open the website. 2. Check that a toggle for light and dark mode is displayed on the page. 3. Click on the toggle for light and dark mode to ensure that it is selectable.
Expected Outcome: A toggle for light and dark mode is displayed on the screen and is selectable.
Actual Outcome: A toggle for light and dark mode is displayed on the screen and is selectable.
Pass/Fail?: Pass

Test Case ID: GUI-002-02
Test Case Description: Verify that the website's colour theme can be changed to light or dark mode.
Test Steps: <ol style="list-style-type: none"> 1. Open the website. 2. Navigate to the toggle for light and dark mode. 3. Select the toggle to change the website's colour theme to dark mode. 4. Ensure the website's theme is changed to dark mode. 5. Select the toggle to change the website's colour theme to light mode. 6. Ensure the website's theme is changed to light mode.
Expected Outcome: The website's colour theme can be changed to both light and dark mode.
Actual Outcome: : The website's colour theme can be changed to both light and dark mode.
Pass/Fail?: Pass

Test Case ID: GUI-003-01
Test Case Description: Verify that the "small" font size option changes the text size on the website to a smaller size.
Test Steps: <ol style="list-style-type: none"> 1. Open the website. 2. Select the "small" font size option from the radio button on the website. 3. Verify that the text size on the website has decreased to the correct size.
Expected Outcome: The text size on the website should be smaller and match the small font size option.
Actual Outcome: The text size on the website is smaller and matches the small font size option.
Pass/Fail?: Pass

Test Case ID: GUI-003-02
Test Case Description: Verify that the "medium" font size option changes the text size on the website to a medium size.
Test Steps: <ol style="list-style-type: none"> 1. Open the website. 2. Select the "medium" font size option from the radio button on the website. 3. Verify that the text size on the website has increased to the correct size.
Expected Outcome: The text size on the website should be medium and match the medium font size option.
Actual Outcome: The text size on the website is medium and matches the medium font size option.
Pass/Fail?: Pass

Test Case ID: GUI-003-03
Test Case Description: Verify that the "large" font size option changes the text size on the website to a larger size.
Test Steps: <ol style="list-style-type: none"> 1. Open the website. 2. Select the "large" font size option from the radio button on the website. 3. Verify that the text size on the website has increased to the correct size.
Expected Outcome: The text size on the website should be larger and match the large font size option.
Actual Outcome: The text size on the website is larger and matches the large font size option.
Pass/Fail?: Pass

Test Case ID: GUI-004-01
Test Case Description: Verify that the website is responsive by testing it on different screen resolutions.
Test Steps: <ol style="list-style-type: none"> 1. Open the website on a desktop computer with a large screen. 2. Resize the browser window to a smaller size and check if the website adjusts its layout accordingly. 3. Repeat step 2 for different screen resolutions, such as tablet or mobile devices.
Expected Outcome: The website layout should adjust to fit different screen resolutions, ensuring that all content remains accessible and usable.
Actual Outcome: The website layout adjusts to fit different screen resolutions, ensuring that all content remains accessible and usable.
Pass/Fail?: Pass

Test Case ID: GUI-005-01
Test Case Description: Verify that instructions are displayed to the user when they navigate to the chatbot page.
Test Steps: <ol style="list-style-type: none"> 1. Navigate to the chatbot page. 2. Verify that a list of clear instructions is displayed on the page.
Expected Outcome: A list of clear instructions on how to use the chatbot system should be displayed to the user when they navigate to the chatbot page.
Actual Outcome: A list of clear instructions on how to use the chatbot system is displayed to the user when they navigate to the chatbot page.
Pass/Fail?: Pass

Test Case ID: GUI-005-02
Test Case Description: Verify that the instructions are easy to understand.
Test Steps: <ol style="list-style-type: none"> 1. Navigate to the chatbot page. 2. Read through the instructions provided. 3. Verify that the instructions are easy to understand and follow.
Expected Outcome: The instructions provided on the chatbot page should be clear and easy to understand so that users can easily understand how to use the system.
Actual Outcome: The instructions provided on the chatbot page are clear and easy to understand so that users can easily understand how to use the system.
Pass/Fail?: Pass

Test Case ID: GUI-005-03
Test Case Description: Verify that the instructions are comprehensive.
Test Steps: <ol style="list-style-type: none"> 1. Navigate to the chatbot page. 2. Read through the instructions provided. 3. Verify that the instructions are comprehensive and cover all aspects of using the chatbot system.
Expected Outcome: The instructions provided on the chatbot page should be comprehensive and cover all aspects of using the chatbot system so that users have a complete understanding of how to use the system.
Actual Outcome: The instructions provided on the chatbot page cover all aspects of using the prototype chatbot system. Although, with more functionality added, the instructions will be more comprehensive.
Pass/Fail?: Pass

GUI-Login

Test Case ID: GUI-Login-001-01
Test Case Description: Test if the login page contains the required elements.
Test Steps: <ol style="list-style-type: none"> 1. Open the login page on both the website. 2. Check if there are two text fields for email/username and password. 3. Check if there is a login button. 4. Check if there is a link to reset the password.
Expected Outcome: The login page should contain two text fields for email/username and password, a login button and a link to reset the password.
Actual Outcome: The login page contains two text fields for email/username and password, a login button and a link to reset the password.
Pass/Fail?: Pass

Test Case ID: GUI-Login-001-02
Test Case Description: Test if an error message appears when incorrect credentials are entered.
Test Steps: <ol style="list-style-type: none"> 1. Enter incorrect email/username and password on the login page. 2. Click the login button. 3. Check if an error message appears.
Expected Outcome: An error message should appear when incorrect credentials are entered, informing the user that the credentials are incorrect.
Actual Outcome: An error message appears when incorrect credentials are entered, informing the user that the credentials are incorrect.
Pass/Fail?: Pass

Test Case ID: GUI-Login-002-01
Test Case Description: Test if the password field on the login page displays * instead of each character of the actual password.
Test Steps: <ol style="list-style-type: none"> 1. Open the login page on both the website. 2. Enter a password in the password field. 3. Check if the characters are being hidden by *.
Expected Outcome: The password field should display * instead of each character of the actual password as the user is typing.
Actual Outcome: The password field actually displays a dot instead of each character of the actual password as the user is typing which is the same concept.
Pass/Fail?: Pass

Test Case ID: GUI-Login-003-01
Test Case Description: Test if there is a visual indicator on the website showing that the user is logged in.
Test Steps: <ol style="list-style-type: none"> 1. Open the website. 2. Log into the system with valid credentials. 3. Look for a visual indicator that shows that the user is logged in.
Expected Outcome: There should be a visual indicator on the website showing that the user is logged in.
Actual Outcome: The visual indicators on the website showing that the user is logged in are the “Settings page” options on the side bar which only show when a user is logged in. An more visible indicator would be to display the user’s name on the home page.
Pass/Fail?: Pass

Test Case ID: GUI-Login-003-02
Test Case Description: Test if there is no visual indicator of anyone logging in before any user has logged in.
Test Steps: <ol style="list-style-type: none"> 1. Open the website. 2. Look for a visual indicator that shows if the user is logged in before logging in.
Expected Outcome: There should be no visual indicator before logging in.
Actual Outcome: There is no visual indicator of anyone logging in before any user has logged in.
Pass/Fail?: Pass

Database-Tables

Test Case ID: database-001-01
Test Case Description: Verify that the database table for storing module information has been created with the required columns.
Test Steps: 1. Check the database schema to verify that a table named Module exists. 2. Verify that the table has columns for module code, module name, module description, weighting, semester and level.
Expected Outcome: The database table Module should exist and have the required columns for storing module information.
Actual Outcome: The database table Module exists and has the required columns for storing module information.
Pass/Fail?: Pass

Test Case ID: database-002-01
Test Case Description: Verify that a table is created to store information about the different pathways.
Test Steps: 1. Check the database schema to verify that a table named Pathway exists. 2. Verify that the table contains columns for pathway code, pathway name and pathway description.
Expected Outcome: A table named Pathway should exist in the database with the required columns to store information about the different pathways offered within EEECS.
Actual Outcome: A table named Pathway exists in the database with the required columns to store information about the different pathways offered within EEECS.
Pass/Fail?: Pass

Test Case ID: database-003-01
Test Case Description: Verify that a Student table with the required columns is created in the database.
Test Steps: <ol style="list-style-type: none"> 1. Check the database to confirm that a table named Student exists. 2. Verify that the Student table contains the following columns: pathway, currentPathwayGrade, level and semester.
Expected Outcome: A table named Student should be created in the database with the required columns to store relevant information about students.
Actual Outcome: A table named Student is created in the database with the required columns to store relevant information about students.
Pass/Fail?: Pass

Test Case ID: database-004-01
Test Case Description: Verify that a Lecturer table with the required columns has been created.
Test Steps: <ol style="list-style-type: none"> 1. Check the database schema for the existence of a table named Lecturer. 2. Verify that the Lecturer table has columns for name, email and password. 3. Verify that the Lecturer table is linked to the ModuleLecturer table to access the module(s) they teach.
Expected Outcome: A table named Lecturer should exist in the database with columns for name, email, password and access to the module(s) taught.
Actual Outcome: A table named Lecturer exists in the database with columns for name, email, password and access to the module(s) taught.
Pass/Fail?: Pass

Test Case ID: database-005-01
Test Case Description: Verify that a Career table with the required columns has been created.
Test Steps: <ol style="list-style-type: none"> 1. Check if the Career table exists in the database. 2. Check if the table has the required columns for career name, company name, role description and salary.
Expected Outcome: The table for careers information should exist and have the required columns.
Actual Outcome: The table has not been created yet so the test has failed
Pass/Fail?: Fail

Test Case ID: database-006-01
Test Case Description: Verify that a Assessment table with the required columns has been created.
Test Steps: <ol style="list-style-type: none"> 1. Check if the Assessment table exists in the database. 2. Verify that the table has columns for the assessment name, type, weighting and moduleID.
Expected Outcome: The Assessment table should exist and have the required columns.
Actual Outcome: The Assessment table exists and has the required columns.
Pass/Fail?: Pass

Test Case ID: database-007-01
Test Case Description: Verify that a ModulePathway table with the required columns has been created.
Test Steps: <ol style="list-style-type: none"> 1. Check if the ModulePathway table exists in the database. 2. Verify that the table has columns for the ModuleID, PathwayID and whether it's a core module or not on that pathway.
Expected Outcome: The ModulePathway table should exist and have the required columns.
Actual Outcome: The ModulePathway table exists and has the required columns.
Pass/Fail?: Pass

Test Case ID: database-008-01
Test Case Description: Verify that a ModuleCareer table with the required columns has been created.
Test Steps: <ol style="list-style-type: none"> 1. Check if the ModuleCareer table exists in the database.
Expected Outcome: The ModuleCareer table should exist and have the required columns.
Actual Outcome:
Pass/Fail?:

Test Case ID: database-009-01
Test Case Description: Verify that a StudentModule table with the required columns has been created.
Test Steps: <ol style="list-style-type: none"> 1. Check if the StudentModule table exists in the database. 2. Verify that the table has columns for the ModuleID, StudentID and the student's grade for that module.
Expected Outcome: The StudentModule table should exist and have the required columns.
Actual Outcome: The StudentModule table exists and has the required columns.
Pass/Fail?: Pass

Test Case ID: database-010-01
Test Case Description: Verify that a StudentModuleAssessment table with the required columns has been created.
Test Steps: <ol style="list-style-type: none"> 1. Check if the StudentModuleAssessment table exists in the database. 2. Verify that the table has columns for the StudentModuleID, AssessmentID and the student's grade for that assessment.
Expected Outcome: The StudentModuleAssessment table should exist and have the required columns.
Actual Outcome: The StudentModuleAssessment table exists and has the required columns.
Pass/Fail?: Pass

Test Case ID: database-011-01
Test Case Description: Verify that the system creates a backup of the database once a week at off-peak times.
Test Steps: <ol style="list-style-type: none"> 1. Check the system logs to verify that the backup process was initiated at the designated off-peak time. 2. Verify that a new backup file has been created and stored in a secure location. 3. Attempt to restore the database from the backup file.
Expected Outcome: The system should create a backup of the database once a week at off-peak times. The backup file should be successfully created and stored in a secure location. The restore process should also be successful, indicating that the backup file contains the necessary data.
Actual Outcome:
Pass/Fail?:

Database-Access Rights

Test Case ID: database-AccessRights-001-01
Test Case Description: Test if a student can only view their own information within the settings page
Test Steps: <ol style="list-style-type: none"> 1. Login as a student. 2. Navigate to the 3. Verify that only the student's information has been displayed
Expected Outcome: The students information will be displayed.
Actual Outcome: The students information will be displayed
Pass/Fail?: Pass

Test Case ID: database-AccessRights-001-02
Test Case Description: Test if a student can only view modules they are currently enrolled in.
Test Steps: <ol style="list-style-type: none"> 1. Login as a student. 2. Ensure the student is enrolled in at least one module. 3. Access the module table in the database. 4. Verify that only modules that the student is enrolled in are displayed.
Expected Outcome: The student can only view modules that they are currently enrolled in.
Actual Outcome: The student cannot access the database, but can see all of their modules in the Grade Dashboard.
Pass/Fail?: Pass

Test Case ID: database-AccessRights-001-03
Test Case Description: Test if a student can only view assessments for modules they are currently enrolled in.
Test Steps: <ol style="list-style-type: none"> 1. Login as a student. 2. Ensure the student is enrolled in at least one module with assessments. 3. Access the assessment table in the database. 4. Verify that only assessments for modules that the student is currently enrolled in are displayed.
Expected Outcome: The student can only view assessments for modules that they are currently enrolled in.
Actual Outcome: The student cannot access the database, but can see all of their assessments in the Grade Dashboard.
Pass/Fail?: Pass

Test Case ID: database-AccessRights-002-01
Test Case Description: Verify that the admin can view all records in all tables.
Test Steps: <ol style="list-style-type: none"> 1. Login as an admin user. 2. Select any table. 3. Verify that all records in the table are visible. Repeat for all tables in the system.
Expected Outcome: The admin should be able to see all records in all tables.
Actual Outcome: The admin is able to see all records in all tables.
Pass/Fail?: Pass

Test Case ID: database-AccessRights-002-02
Test Case Description: Verify that the admin can create a new record in any table.
Test Steps: <ol style="list-style-type: none"> 1. Login as an admin user. 2. Select any table. 3. Click on the "Add new record" button. 4. Fill in the required fields and click the "Save" button.
Expected Outcome: The new record should be added to the selected table.
Actual Outcome: The new record is added to the selected table.
Pass/Fail?: Pass

Test Case ID: database-AccessRights-002-03
Test Case Description: Verify that the admin can update an existing record in any table.
Test Steps: <ol style="list-style-type: none"> 1. Login as an admin user. 2. Select any table. 3. Click on the record you want to edit. 4. Make the required changes to the record and click the "Save" button.
Expected Outcome: The record should be updated with the changes made by the admin.
Actual Outcome: The record is updated with the changes made by the admin.
Pass/Fail?: Pass

Test Case ID: database-AccessRights-002-04
Test Case Description: Verify that the admin can delete any record from any table.
Test Steps: <ol style="list-style-type: none"> 1. Login as an admin user. 2. Select any table. 3. Click on the record you want to delete. 4. Click the "Delete" button.
Expected Outcome: The record should be deleted from the selected table.
Actual Outcome: The record is deleted from the selected table.
Pass/Fail?: Pass

Website

Test Case ID: website-001-01
Test Case Description: Test website on Chrome v113.0.
Test Steps: <ol style="list-style-type: none">1. Open the website on Chrome v113.0.2. Verify that all website features are functional.
Expected Outcome: All website features are functional on Chrome v113.0.
Actual Outcome: All website features are functional on Chrome v113.0.
Pass/Fail?: Pass

Test Case ID: website-001-02
Test Case Description: Test website on Firefox v113.
Test Steps: <ol style="list-style-type: none">1. Open the website on Firefox v113.2. Verify that all website features are functional.
Expected Outcome: All website features are functional on Firefox v113.
Actual Outcome: All website features are functional on Firefox v113.
Pass/Fail?: Pass

Test Case ID: website-001-03
Test Case Description: Test website on Edge v113.
Test Steps: <ol style="list-style-type: none"> 1. Open the website on Edge v113. 2. Verify that all website features are functional.
Expected Outcome: All website features are functional on Edge v113.
Actual Outcome: All website features are functional on Edge v113.
Pass/Fail?: Pass

Test Case ID: website-001-04
Test Case Description: Test website on Android v111.
Test Steps: <ol style="list-style-type: none"> 1. Open the website on Android v111. 2. Verify that all website features are functional.
Expected Outcome: All website features are functional on Android v111.
Actual Outcome:
Pass/Fail?:

Test Case ID: website-001-05

Test Case Description: Test website on Chrome for Android v112.

Test Steps:

1. Open the website on Chrome for Android v112.
2. Verify that all website features are functional.

Expected Outcome: All website features are functional on Chrome for Android v112.

Actual Outcome:

Pass/Fail?:

Test Case ID: website-001-06

Test Case Description: Test website on Chrome for iOS v112.

Test Steps:

1. Open the website on Chrome for iOS v112.
2. Verify that all website features are functional.

Expected Outcome: All website features are functional on Chrome for iOS v112.

Actual Outcome:

Pass/Fail?:

Test Case ID: website-002-01
Test Case Description: Verify the website's homepage can be accessed on an Android phone using the Chrome browser and all touch screen functions work properly.
Test Steps: <ol style="list-style-type: none"> 1. Open the Chrome browser on an Android phone. 2. Navigate to the website's homepage. 3. Verify that all touch screen functions on the homepage (e.g., buttons, text fields) work properly.
Expected Outcome: All touch screen functions on the homepage work properly on the Android phone using the Chrome browser.
Actual Outcome:
Pass/Fail?:

Test Case ID: website-002-02
Test Case Description: Verify the website's homepage can be accessed on an IOS phone using the Safari browser and all touch screen functions work properly.
Test Steps: <ol style="list-style-type: none"> 1. Open the Safari browser on an IOS phone. 2. Navigate to the website's homepage. 3. Verify that all touch screen functions on the homepage (e.g., buttons, text fields) work properly.
Expected Outcome: All touch screen functions on the homepage work properly on the IOS phone using the Safari browser.
Actual Outcome:
Pass/Fail?:

Test Case ID: website-002-03
Test Case Description: Verify the website's homepage can be accessed on a Windows tablet using the Edge browser and all touch screen functions work properly.
Test Steps: <ol style="list-style-type: none"> 1. Edge browser on a Windows tablet. 2. Navigate to the website's homepage. 3. Verify that all touch screen functions on the homepage (e.g., buttons, text fields) work properly.
Expected Outcome: All touch screen functions on the homepage work properly on the Windows tablet using the Edge browser.
Actual Outcome:
Pass/Fail?:

Test Case ID: website-003-01
Test Case Description: Verify that the website can be run on a Linux server using Docker.
Test Steps: <ol style="list-style-type: none"> 1. Install Docker on a Linux server. 2. Build the Docker image of the project on the server. 3. Start the Docker container and expose the necessary ports. 4. Navigate to the server's IP address on a web browser on a separate device. 5. Verify that the website appears and all functionality is working.
Expected Outcome: The website should appear and all functionality should be working as expected when accessed from the server's IP address on a web browser on a separate device.
Actual Outcome:
Pass/Fail?:

Webscraping

Test Case ID: WebScraping-001-01
Test Case Description: Verify that the web scraping process is executed at the scheduled time.
Test Steps: <ol style="list-style-type: none">1. Set up the web scraping process to run at a specific time.2. Wait for the scheduled time to arrive.3. Check the database to see if the information has been updated with the latest data.
Expected Outcome: The database should contain the most up-to-date information on the careers after the scheduled web scraping process has run.
Actual Outcome:
Pass/Fail?:

Test Case ID: WebScraping-001-02
Test Case Description: Verify that the web scraping process retrieves the correct information.
Test Steps: <ol style="list-style-type: none">1. Identify a specific piece of information on a career-related website (e.g., salary information).2. Manually retrieve the information from the website and record it.3. Set up the web scraping process to retrieve the same piece of information from the same website.4. Check the database to see if the retrieved information matches the manually recorded information.
Expected Outcome: The retrieved information from the web scraping process should match the manually recorded information.
Actual Outcome:
Pass/Fail?:

Test Case ID: WebScraping-001-03
Test Case Description: Verify that the web scraping process handles errors smoothly.
Test Steps: <ol style="list-style-type: none"> 1. Introduce an error into the web scraping process (e.g., attempt to scrape information from a website that is currently down). 2. Check the database to see if the web scraping process handled the error smoothly (e.g., by logging the error and moving on to the next website).
Expected Outcome: The web scraping process should handle errors smoothly and continue to update the database with information from other websites.
Actual Outcome:
Pass/Fail?:

Test Case ID: WebScraping-002-01
Test Case Description: Verify that the scrapped data is being stored correctly in the corresponding database table.
Test Steps: <ol style="list-style-type: none"> 1. Start the web scraping process. 2. Check the corresponding database table for the new data.
Expected Outcome: The scrapped data is correctly stored in the corresponding database table.
Actual Outcome:
Pass/Fail?:

GradeDashboard

Test Case ID: GradeDashboard-001-01

Test Case Description: Test if the system updates the student's grade automatically when marks are added for an assessment.

Test Steps:

1. Login as an admin.
2. Add a student's mark for an assessment.
3. Check the overall grade of the student for that module.
4. Verify that the grade has been automatically updated to reflect the new mark.

Expected Outcome: The overall grade of the student for that module should be updated to reflect the new mark automatically.

Actual Outcome: The overall grade of the student for that module is not updated automatically to reflect the new mark.

Pass/Fail?: Pass

Test Case ID: GradeDashboard-001-02

Test Case Description: Test if the system does not update the student's assessment grade if the added mark is invalid.

Test Steps:

1. Log in as an admin.
2. Add an invalid mark for an assessment in a module (e.g., negative mark, mark greater than the maximum).
3. An error message should appear.
4. Verify that the assessment mark has not been updated.

Expected Outcome: The overall grade of the student for that assessment should not be updated if the added mark is invalid.

Actual Outcome: The invalid assessment mark is allowed to be added.

Pass/Fail?: Fail

Test Case ID: GradeDashboard-002-01
Test Case Description: Verify that when a module grade is updated for a student, the overall percentage for their degree is updated correctly.
Test Steps: <ol style="list-style-type: none"> 1. Set up a test scenario where a student has completed some modules and has an overall percentage for their degree. 2. Update the grade for one of the modules and confirm the change is saved in the system. 3. Calculate the expected overall percentage for the student's degree based on the new grade for the module and the existing grades for the other modules. 4. Check that the overall percentage for the student's degree matches the expected value.
Expected Outcome: The overall percentage for the student's degree should be updated to reflect the new grade for the module.
Actual Outcome: The overall percentage for the student's degree is not updated to reflect the new grade for the module.
Pass/Fail?: Pass

Test Case ID: GradeDashboard-003-01
Test Case Description: Manually calculate the average mark for a module using the marks received for each assessment within that module, then compare this to the average mark calculated by the system.
Test Steps: <ol style="list-style-type: none"> 1. Collect marks for each assessment within a module. 2. Calculate the average mark for the module manually. 3. Check the average mark calculated by the system for that module.
Expected Outcome: The average mark calculated by the system matches the manually calculated average mark.
Actual Outcome: The average mark calculated by the system matches the manually calculated average mark.
Pass/Fail?: Pass

Test Case ID: GradeDashboard-003-02
Test Case Description: Manually calculate the average mark for assessments using the marks received for each assessment, then compare this to the average mark calculated by the system.
Test Steps: <ol style="list-style-type: none"> 1. Collect marks for each assessment completed by the student. 2. Calculate the average mark for the assessments manually. 3. Check the average mark calculated by the system for the assessments.
Expected Outcome: The average mark calculated by the system matches the manually calculated average mark.
Actual Outcome: The average mark calculated by the system matches the manually calculated average mark.
Pass/Fail?: Pass

ModuleInformation

Test Case ID: ModuleInformation-001-01
Test Case Description: Verify that the user can search for a module by typing the module code in the search bar.
Test Steps: <ol style="list-style-type: none"> 1. Go to the module information page. 2. Type the moduleID in the search bar and press enter. 3. Verify that the module information for the corresponding moduleID is displayed.
Expected Outcome: The module information for the corresponding moduleID is displayed.
Actual Outcome: The module information for the corresponding moduleID is not displayed.
Pass/Fail?: Fail

Test Case ID: ModuleInformation-001-02
Test Case Description: Verify that the user can search for a module by typing the module name in the search bar.
Test Steps: <ol style="list-style-type: none"> 1. Go to the module information page. 2. Type the module name in the search bar and press enter. 3. Verify that all the modules that contain the search keyword in the title appear.
Expected Outcome: All the modules that contain the search keyword in the title appear.
Actual Outcome: All the modules that contain the search keyword in the title appear.
Pass/Fail?: Pass

Test Case ID: ModuleInformation-001-03
Test Case Description: Verify that the user can clear the search bar.
Test Steps: <ol style="list-style-type: none"> 1. Go to the module information page. 2. Type the module name in the search bar and press enter. 3. Verify that all the modules that contain the search keyword in the title appear. 4. Click on the "x" button on the right-hand side of the search bar. 5. Verify that the search bar is empty and all modules are displayed.
Expected Outcome: The search bar is empty and all modules are displayed.
Actual Outcome: There is no "x" button to clear the search bar. The user has to manually remove the data in the search bar.
Pass/Fail?: Pass

Test Case ID: ModuleInformation-002-01
Test Case Description: Apply stage filter, verify correct modules are shown.
Test Steps: <ol style="list-style-type: none"> 1. Go to module information page. 2. Apply filter for stage 2. 3. Verify that only modules for stage 2 are shown.
Expected Outcome: Only modules for stage 2 are shown.
Actual Outcome: Only modules for stage 2 are shown.
Pass/Fail?: Pass

Test Case ID: ModuleInformation-002-02
Test Case Description: Apply semester filter, verify correct modules are shown.
Test Steps: <ol style="list-style-type: none"> 1. Go to module information page. 2. Apply filter for semester 1. 3. Verify that only modules for semester 1 are shown.
Expected Outcome: Only modules for semester 1 are shown.
Actual Outcome: Only modules for semester 1 are shown.
Pass/Fail?: Pass

Test Case ID: ModuleInformation-002-03
Test Case Description: Apply pathway filter, verify correct modules are shown.
Test Steps: <ol style="list-style-type: none"> 1. Go to module information page. 2. Apply filter for pathway "MEng Computer Science". 3. Verify that only modules for the " MEng Computer Science" pathway are shown.
Expected Outcome: Only modules for the " MEng Computer Science" pathway are shown.
Actual Outcome: Only modules for the " MEng Computer Science" pathway are shown.
Pass/Fail?: Pass

Test Case ID: ModuleInformation-002-04
Test Case Description: Apply multiple filters, verify correct modules are shown.
Test Steps: <ol style="list-style-type: none"> 1. Go to module information page. 2. Apply filter for stage 2, semester 1, and pathway " MEng Computer Science". 3. Verify that only modules for stage 2, semester 1 and the " MEng Computer Science" pathway are shown.
Expected Outcome: Only modules for stage 2, semester 1 and the " MEng Computer Science" pathway are shown.
Actual Outcome: Only modules for stage 2, semester 1 and the " MEng Computer Science" pathway are shown.
Pass/Fail?: Pass

Test Case ID: ModuleInformation-002-05
Test Case Description: Apply filters and search for module containing "learning", verify correct modules are shown.
Test Steps: <ol style="list-style-type: none"> 1. Go to module information page. 2. Apply filter for stage 3 and pathway " MEng Computer Science". 3. Search for "learning". 4. Verify that only modules for stage 3, the "Computer Science" pathway and containing "learning" in the title are shown.
Expected Outcome: Only modules for stage 3, the "Computer Science" pathway and containing "learning" in the title are shown.
Actual Outcome: Only modules for stage 3, the "Computer Science" pathway and containing "learning" in the title are shown.
Pass/Fail?: Pass

Signup

Test Case ID: SignUp-001-01
Test Case Description: Verify that a user can enter valid information for each field on the sign-up page.
Test Steps: <ol style="list-style-type: none"> 1. Access the sign-up page. 2. Enter valid information for each field. 3. Submit the form.
Expected Outcome: The system should send an email to the admin with the information entered on the sign-up form.
Actual Outcome: The system sends an email to the admin with the information entered on the sign-up form.
Pass/Fail?: Pass

Test Case ID: SignUp-002-01
Test Case Description: Verify that all fields on the sign-up page are required and cannot be left empty.
Test Steps: <ol style="list-style-type: none"> 1. Access the sign-up page. 2. Leave one or more required fields empty. 3. Submit the form.
Expected Outcome: An error message should be displayed next to each empty required field indicating that the field is required.
Actual Outcome: The system sends an email to the admin with the information entered on the sign-up form.
Pass/Fail?: Fail

Chatbot

Test Case ID: Chatbot-001-01
Test Case Description: Test sentiment analysis for positive sentence.
Test Steps: <ol style="list-style-type: none"> 1. Input "I like coding" into the chatbot. 2. Chatbot performs sentiment analysis on the input. 3. Verify that the chatbot correctly identifies the input as having a positive sentiment towards coding.
Expected Outcome: Chatbot correctly identifies "I like coding" as having a positive sentiment towards coding.
Actual Outcome: Chatbot correctly identifies "I like coding" as having a positive sentiment towards coding.
Pass/Fail?: Pass

Test Case ID: Chatbot-001-02
Test Case Description: Test sentiment analysis for negative sentence.
Test Steps: <ol style="list-style-type: none"> 1. Input "I hate maths" into the chatbot. 2. Chatbot performs sentiment analysis on the input. 3. Verify that the chatbot correctly identifies the input as having a negative sentiment towards maths.
Expected Outcome: Chatbot correctly identifies "I hate maths" as having a negative sentiment towards maths.
Actual Outcome: Chatbot correctly identifies "I hate maths" as having a negative sentiment towards maths.
Pass/Fail?: Pass

Test Case ID: Chatbot-001-03
Test Case Description: Test sentiment analysis for neutral sentence.
Test Steps: <ol style="list-style-type: none"> 1. Input "What time is it?" into the chatbot. 2. Chatbot performs sentiment analysis on the input. 3. Verify that the chatbot correctly identifies the input as having no sentiment.
Expected Outcome: Chatbot correctly identifies "What time is it?" as having no sentiment.
Actual Outcome: Chatbot correctly identifies "What time is it?" as having no sentiment.
Pass/Fail?: Pass

Test Case ID: Chatbot-002-01
Test Case Description: Test chatbot's ability to identify non-English language input and respond with appropriate error message in that language.
Test Steps: <ol style="list-style-type: none"> 1. Enter the word "Bonjour" (French for "Hello") as input to the chatbot. 2. Verify that the chatbot identifies the language as French and responds with the error message "I'm sorry but I can only take responses that are in English, here is the contact information for the student support office that may be able to help you: <email>" in French.
Expected Outcome: Chatbot should correctly identify the language and respond with the appropriate error message in that language.
Actual Outcome: Chatbot doesn't correctly identify the language and respond with the appropriate error message in that language.
Pass/Fail?: Fail

Test Case ID: Chatbot-002-02
Test Case Description: Test chatbot's ability to identify random characters as input and respond with appropriate error message.
Test Steps: <ol style="list-style-type: none"> 1. Enter the string "hdfjkahfda" as input to the chatbot. 2. Verify that the chatbot responds with the error message "I'm sorry but I can only take valid input as a response."
Expected Outcome: Chatbot should identify the input as random characters and respond with the appropriate error message.
Actual Outcome: Chatbot doesn't identify the input as random characters and respond with the appropriate error message.
Pass/Fail?: Fail

Test Case ID: Chatbot-003-01
Test Case Description: Test chatbot's ability to recommend modules based on user's interests.
Test Steps: <ol style="list-style-type: none"> 1. Enter "I like maths". 2. Wait for chatbot's response.
Expected Outcome: Chatbot recommends modules related to programming and web development, such as "Fundamentals of Maths for Computing"
Actual Outcome: Chatbot recommends modules related to maths, such as "Fundamentals of Maths for Computing"
Pass/Fail?: Pass

Test Case ID: Chatbot-003-02
Test Case Description: Test chatbot's ability to recommend modules based on user's academic record.
Test Steps: <ol style="list-style-type: none"> 1. Enter "I have taken a lot of maths courses". 2. Wait for chatbot's response.
Expected Outcome: Chatbot recommends modules related to maths, such as "Fundamentals of Maths for Computing"
Actual Outcome: Chatbot doesn't recommend any modules.
Pass/Fail?: Fail

Test Case ID: Chatbot-004-01
Test Case Description: Test chatbot's ability to provide information about the module recommended.
Test Steps: <ol style="list-style-type: none"> 1. Provide input to the chatbot about your interests and academic records. 2. Verify that the chatbot recommends a module. 3. Check the response crafted by the chatbot for details about the recommended module(s).
Expected Outcome: The response should provide some other relevant details about the module(s), such as exams, required skills, career prospects, etc.
Actual Outcome: The response doesn't provide any other relevant details about the module.
Pass/Fail?: Fail

Test Case ID: Chatbot-005-01
Test Case Description: Test if the chatbot displays all modules available for Stage 1.
Test Steps: <ol style="list-style-type: none"> 1. Enter "list modules for stage 1" into the chatbot. 2. Verify that the chatbot displays all modules available for Stage 1.
Expected Outcome: The chatbot displays all modules available for Stage 1.
Actual Outcome: The chatbot displays all modules available for Stage 1.
Pass/Fail?: Pass

Test Case ID: Chatbot-005-02
Test Case Description: Test if the chatbot displays all modules available for Stage 2.
Test Steps: <ol style="list-style-type: none"> 1. Enter "list modules for stage 2" into the chatbot. 2. Verify that the chatbot displays all modules available for Stage 2.
Expected Outcome: The chatbot displays all modules available for Stage 2.
Actual Outcome: The chatbot displays all modules available for Stage 2.
Pass/Fail?: Pass

Test Case ID: Chatbot-005-03
Test Case Description: Test if the chatbot displays all modules available for Stage 3.
Test Steps: <ol style="list-style-type: none"> 1. Enter "list modules for stage 3" into the chatbot. 2. Verify that the chatbot displays all modules available for Stage 3.
Expected Outcome: The chatbot displays all modules available for Stage 3.
Actual Outcome: The chatbot displays all modules available for Stage 3.
Pass/Fail?: Pass

Test Case ID: Chatbot-005-04
Test Case Description: Test if the chatbot displays all modules available for the " MEng Computer Science" pathway.
Test Steps: <ol style="list-style-type: none"> 1. Enter "list modules for MEng Computer Science pathway" into the chatbot. 2. Verify that the chatbot displays all modules available for the " MEng Computer Science" pathway.
Expected Outcome: The chatbot displays all modules available for the "MEng Computer Science" pathway.
Actual Outcome: The chatbot doesn't display any modules available for the " MEng Computer Science" pathway.
Pass/Fail?: Fail

Test Case ID: Chatbot-006-01
Test Case Description: Testing the chatbot's response to the user entering "help" or a statement that can be interpreted as asking for instructions.
Test Steps: <ol style="list-style-type: none"> 1. Enter "help" to the chatbot. 2. Observe the chatbot's response.
Expected Outcome: The chatbot responds with instructions on how to use the chatbot.
Actual Outcome: The chatbot doesn't respond with instructions on how to use the chatbot.
Pass/Fail?: Fail

Login

Test Case ID: Login-001-01
Test Case Description: Valid login credentials are entered, and the user is logged in.
Test Steps: <ol style="list-style-type: none">1. Enter valid login credentials (username and password).2. Click on the login button.
Expected Outcome: The system should log in the user and redirect them to the home page.
Actual Outcome: Valid login credentials are entered, and the user is logged in.
Pass/Fail?: Pass

Test Case ID: Login-001-01
Test Case Description: Invalid login credentials are entered and the user is not logged in.
Test Steps: <ol style="list-style-type: none">1. Enter invalid login credentials (incorrect username or password).2. Click on the login button.
Expected Outcome: The system should not log in the user and display an error message indicating that the credentials entered are invalid.
Actual Outcome: The system doesn't log in the user and displays an error message indicating that the credentials entered are invalid.
Pass/Fail?: Pass

Test Case ID: Login-002-01
Test Case Description: Verify that a student account can only access student-related information.
Test Steps: <ol style="list-style-type: none"> 1. Log in using valid student credentials. 2. Attempt to access admin-only pages or data.
Expected Outcome: The system should not allow the student to access admin-only pages or data and should instead redirect them to an error page or notify them that they do not have the necessary access rights.
Actual Outcome: The system doesn't show any admin-only pages or data when logged in as a student.
Pass/Fail?: Pass

Test Case ID: Login-002-02
Test Case Description: Verify that an admin account can access both student and admin-related information.
Test Steps: <ol style="list-style-type: none"> 1. Log in using valid admin credentials. 2. Attempt to access both student and admin-only pages or data.
Expected Outcome: The system should allow the admin to access both student and admin-only pages or data without any issues.
Actual Outcome: The system allows the admin to access both the module information page which students can also access and the admin page which only the admin can access.
Pass/Fail?: Pass

Test Case ID: Login-003-01

Test Case Description: Verify that when a user creates a new account with password "password", the password is encrypted in the database.

Test Steps:

1. Create a new user account with the password "password".
2. Access the database and check the password field for the newly created account.

Expected Outcome: The password field should contain an encrypted string instead of the plaintext "password".

Actual Outcome: The Django administration page shows only an encrypted password. Therefore, raw passwords are not stored.

Pass/Fail?: Pass

Test Case ID: Login-003-02

Test Case Description: Verify that when an existing user changes their password to "password", the password is encrypted in the database.

Test Steps:

1. Create a new user account with a different password.
2. Change the password of the newly created account to "password".
3. Access the database and check the password field for the account.

Expected Outcome: The password field should contain an encrypted string instead of the plaintext "password".

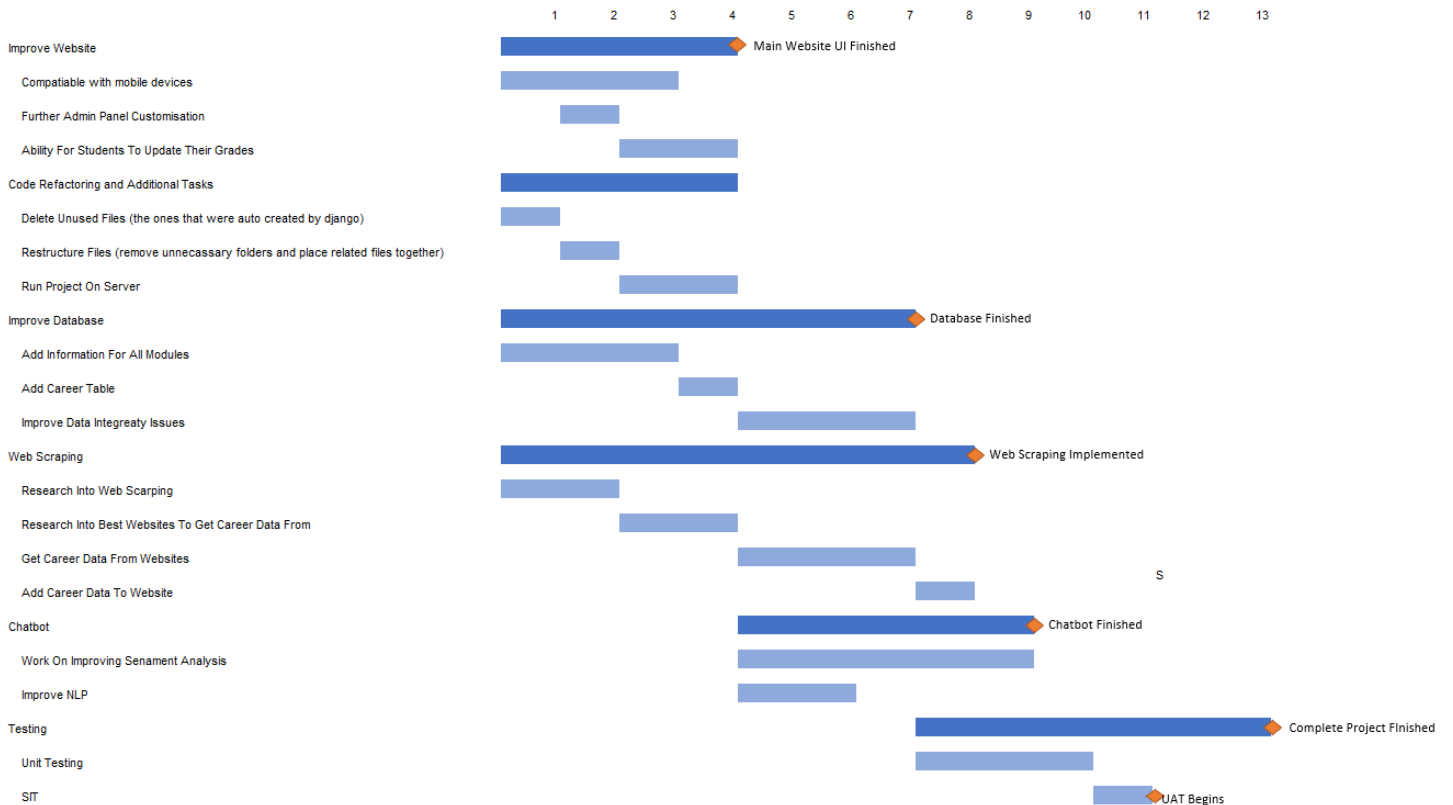
Actual Outcome: The Django administration page shows only an encrypted password. Therefore, raw passwords are not stored.

Pass/Fail?: Pass

Test Case ID: Login-004-01
Test Case Description: Verify that the “remember me” functionality is available on the login page and works as expected.
Test Steps: <ol style="list-style-type: none"> 1. Log in to the system with the “remember me” option selected. 2. Close the browser. 3. Reopen the browser and navigate to the website.
Expected Outcome: The user should be automatically logged in without having to enter their credentials again.
Actual Outcome: The user is automatically logged in without having to enter their credentials again. Although, the “remember me” option isn’t showing on the login popup.
Pass/Fail?:

G) Appendix: Expanded Roadmap

Complete roadmap view (the numbers along the top represent the weeks within the semester):

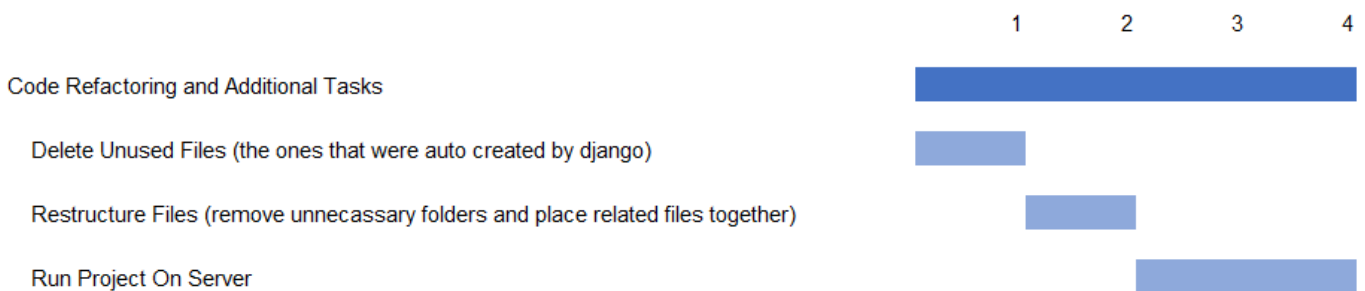


Further zoomed in for each epic

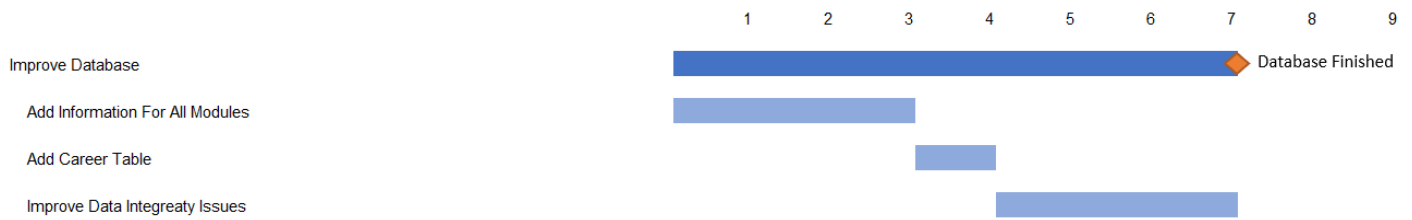
Improve Website:



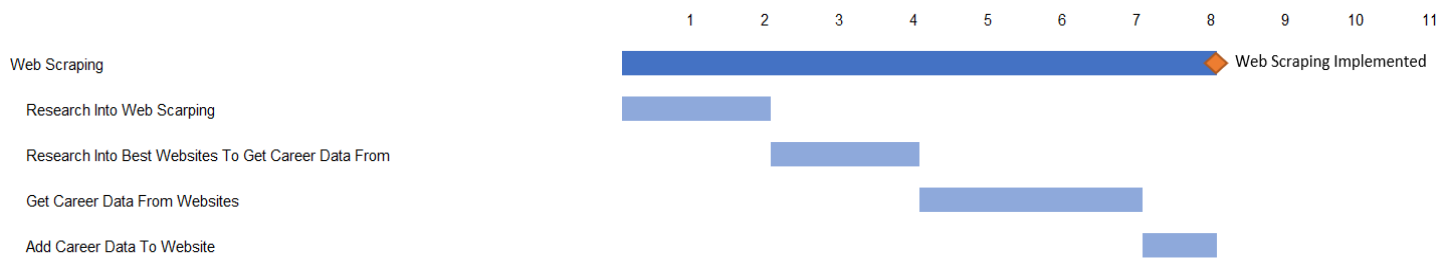
Code Refactoring:



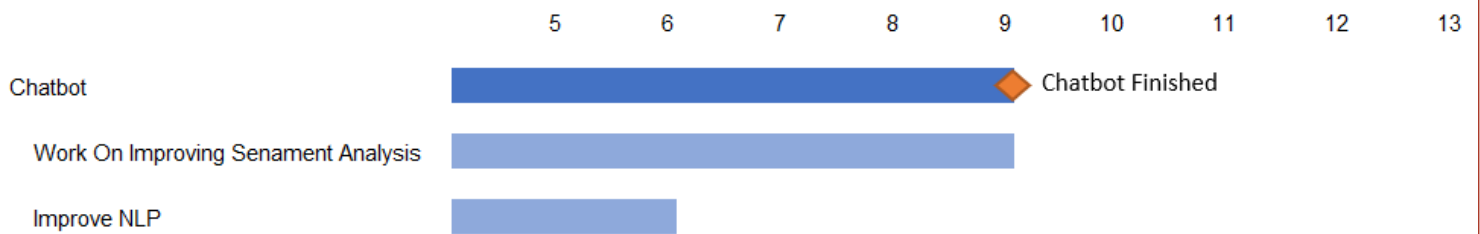
Improve Database:



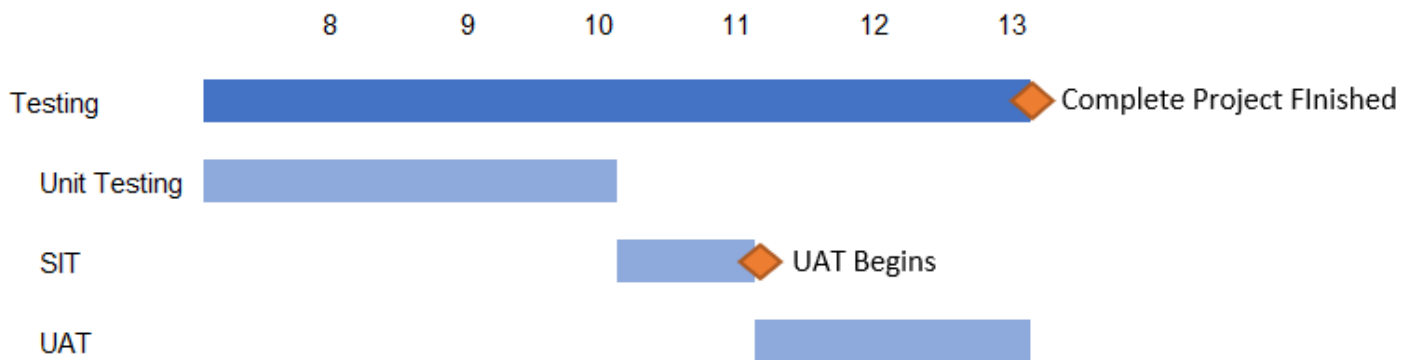
Web Scraping:



Chatbot (note: starts from week 5, removed the previous blank links from diagram to further improve readability):



- Testing (note: starts from week 8, removed the previous blank links from diagram to further improve readability):



References

- [1] I. Sommerville, Software engineering 10th Edition. Pearson Education, 2016.
- [2] [B. R. Ranoliya, N. Raghuwanshi and S. Singh, "Chatbot for university related FAQs," 2017 International Conference on Advances in Computing, Communications and Informatics \(ICACCI\), Udupi, India, 2017, pp. 1525-1530, doi: 10.1109/ICACCI.2017.8126057.](#)
- [3] Gupta, V., Mathur, S., & Singh, B. (2018). Chatbot using Python. International Journal for Research in Applied Science & Engineering Technology (IJRASET), 6(II). Retrieved from <https://www.ijraset.com/research-paper/chatbot-using-python>
- [4] Teoh, T.T., Rong, Z. (2022). Python for Artificial Intelligence. In: Artificial Intelligence with Python. Machine Learning: Foundations, Methodologies, and Applications. Springer, Singapore. https://doi.org/10.1007/978-981-16-8615-3_1
- [5] <https://docs.djangoproject.com/en/4.2/topics/auth/passwords/> - Django Docs
- [6] ¹ Rubio, D. (2017). Django Application Management. Springer. <https://doi.org/10.1007/978-1-4842-2787-9>