POLITECNICO
MILANO 1863

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Dorsa Motiallah, Sujata Chaudhury, Darkhan Islam
Version:	1.0
Date:	07-January-2025
Download page:	https://github.com/dorsamotiallah/MotiallahChaudhuryIslam
Copyright:	

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of the Students & Companies (S&C) project is to create a platform that efficiently matches university students with suitable internship opportunities based on their skills, experiences, and preferences, while also meeting the needs of companies seeking to recruit talent. The platform streamlines the entire internship process, from profile creation and internship listings to matching, interview management, and feedback collection. It aims to improve the overall internship experience, enhance the selection process, and provide universities with tools to monitor and ensure the quality of internships, ultimately benefiting students, companies, and academic institutions alike. The S&C platform creates a dynamic ecosystem where students, companies, and universities collaborate seamlessly to improve the internship matching process. The system's features not only save time for both students and companies but also enhance the quality and success of internships, ultimately leading to more fulfilling and productive experiences for all parties involved.

1.1.1 Goals

G1 Allows registered students to create their profile and CV and search for internships and apply to them.

G2 Allows registered companies to create and post advertisement for available internship roles and select suitable candidate.

G3 Provide an efficient and intelligent matchmaking process between students and companies based on the students' skills and preferences, the companies' internship descriptions.

G4 Support both students and companies during the selection process, including interview scheduling, questionnaire, feedback collection, and facilitating communication.

G5 Allows registered universities to monitor the internship statuses of their students and take action on any issues or complaints.

1.1.2 Scope

The scope of this Design Document (DD) is to evaluate the technical, operational, and organizational aspects of the software engineering project in order to assess its alignment with business goals. This document discusses the architecture used in this project, highlights all the main components to be created and how they interact. This document also introduces the implementation and testing principles to be used to ensure robustness of the software.

1.2 Definitions, Acronyms, Abbreviations

In this section some information about terminology is provided, in order to clarify terms, acronyms, and abbreviations used in the document, ensuring easy understanding and reference for readers.

1.2.1 Definitions

- **Student:** Student is a type of user who is looking for an internship.
- **Company:** Company is a type of user which has internship positions available and searching for students to fill those positions.
- **University:** University is a type of user who monitors the ongoing selection process between a student and a company.

1.2.2 Acronyms

- **C&S:** Company & Students, that is the name of the platform.
- **UI:** User Interface
- **DB:** Database

1.2.3 Abbreviations

- **WPn:** n-th World Phenomena
- **SPn:** n-th Shared Phenomena
- **Gn:** n-th Goal
- **Dn:** n-th Domain Assumption
- **Rn:** n-th Requirement

1.3 Revision History

Revised On	Version	Description
07-Jan-2025	1.0	Initial Release of Document

Table 1: Revision History

1.4 Reference Documents

- Assignment document A.Y. 2024/2025
  (Requirement Engineering and Design Project: goal, schedule and rules)
- Software Engineering 2 A.Y. 2024/2025 Slides
  (Lecture slides provided during the course)

1.5 Document Structure

- **Introduction:** Introduction chapter gives a brief description of the project. This chapter discusses the different goals and scope of the S&C system along with definitions.
- **Architectural Design:** This section discusses the design and interaction of the different components of the S&C system at different levels and the architecture used for the system.
- **User Interface Design:** A set of user interfaces are mentioned is presented in this section. These interfaces are extended from those of the RSAD document.
- **Requirements Traceability:** Each requirement from the RASD document is mapped with the design components introduced in this document.
- **Implementation, Integration and Test Plan:** This section provides a description of the implementation integration and testing details.
- **Effort Spent:** This section shows the effort spent to write this document by each team member.
- **Bibliography:** This contains the references to any documents and software used to write this document.

2 Architectural Design

This section discusses all the architectural decisions taken for building the Students & Companies (S&C). In the overview sub-section, the overall architecture of the system is presented along with a diagram which differentiates the architectural layers within the system. Then component view of the software is presented with all the important components and the relationship between all the components are clearly defined. Then in the deployment view decisions related to the deployment of the software is articulated. The flow of interactions between actors and components are shown using sequence diagrams in the Runtime View section.

2.1 Overview: High-level components and their interaction

The figure below is gives a high-level description of the S&C system. The different users of system interacts with the client side of our web application which is called the front-end of our application and is in the Application Layer. The back-end comprises of the business logic layers which is used for carrying out all the server applications. Lastly, the data layer which is used for storage and retrieval of the data.

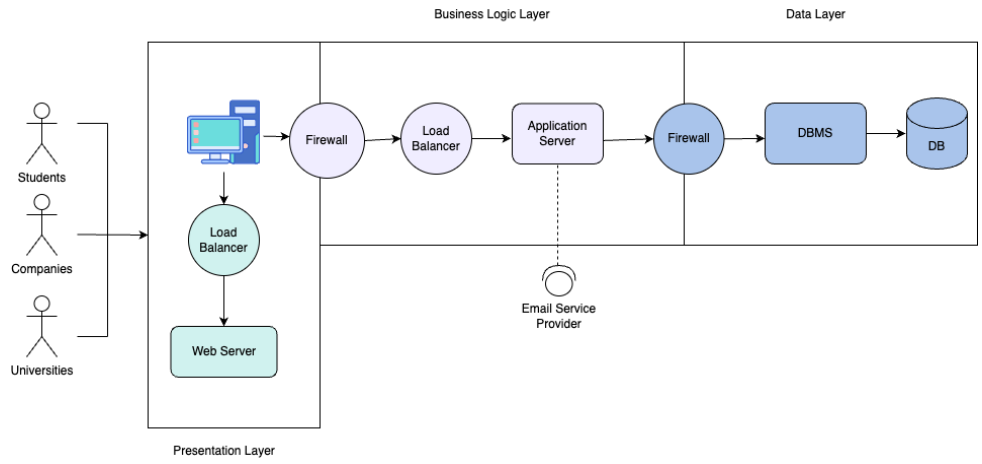

Figure 1: Presentation Layer, Business Logic Layer and Data Layer

2.2 Component View

In this section we show the components of the S&C and their relationships.

Subsystems:

1. **User Interfaces:**
   - **Student Portal, Company Portal, University Admin Portal:** Interfaces for respective users to interact with the system.
2. **Core Systems:**
   - **Recommendation Engine:** Matches students and internships based on data.
   - **Feedback and Analytics Module:** Improves recommendations using feedback.
   - **Notification Service:** Sends messages/alerts to users.
   - **Selection Management System:** Manages interviews and selection processes.
   - **Monitoring and Complaints Module:** Tracks internships and manages complaints.
   - **Security and Access Control:** Ensures authentication and secure access.

3. **Data Storage:**
    - **Database:** Stores CVs, internship details, feedback, and complaints.

Connection Types:

- `-( Required Interface:` Indicates a component requests services (e.g., portals requesting recommendations).
- `-(0- Message Interface:` Represents communication or event-driven messaging (e.g., notifications).
- `-0)- Service Interface:` Represents a request-response model (e.g., selection processes).
- `..> Dependency:` Highlights one component's dependency on another (e.g., feedback improving recommendations).

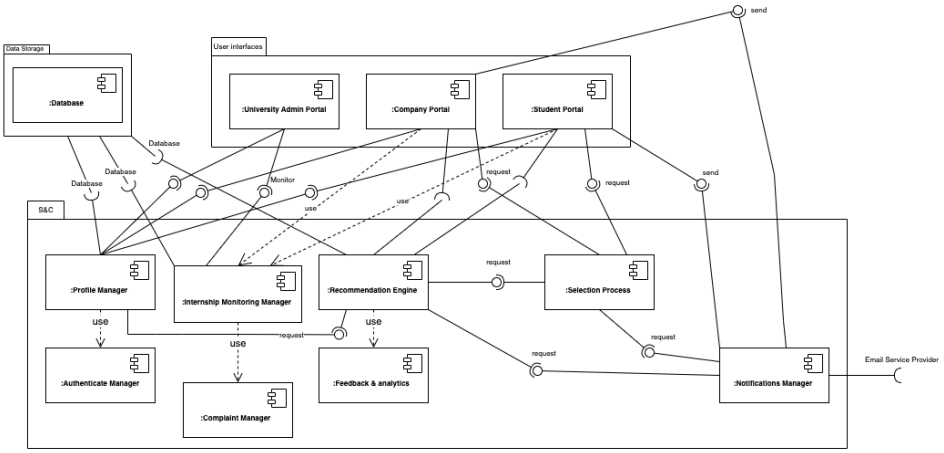

Figure 2: UML Diagram for Students & Companies System

2.3 Deployment View

The S&C web application is deployed in cloud and uses the benefits that are offered by cloud deployment compared to traditional deployments which will be discussed later.

Client device and access the S&C web page using any browser of choice. The web pages are loaded from the web server in cloud using HTTP calls. Firewall is used to provide security and allow trusted requests. Elastic load balancer is used to evenly distribute incoming traffic in web servers and application servers. Lastly, the required data are stored in the DB. A standby DB is also added to provide with data connection in case of failure of the main DB.

Advantages of Cloud -

- **Scalability and Flexibility:** Cloud platforms provide the ability to scale resources up or down quickly based on demand. This is especially useful for businesses with fluctuating needs or growth. Cloud systems automatically scale resources to meet workload changes without manual intervention, optimizing resource utilization.
- **Cost Efficiency:** Cloud services offers a pay-per-usage, thus paying only for the resources actually used. Cloud services offer reduced maintenance cost and low infrastructure costs.

- **High Availability and Reliability:** Cloud providers usually have data centers in multiple locations, offering built-in redundancy. This ensures high availability and reduces the risk of system outages.

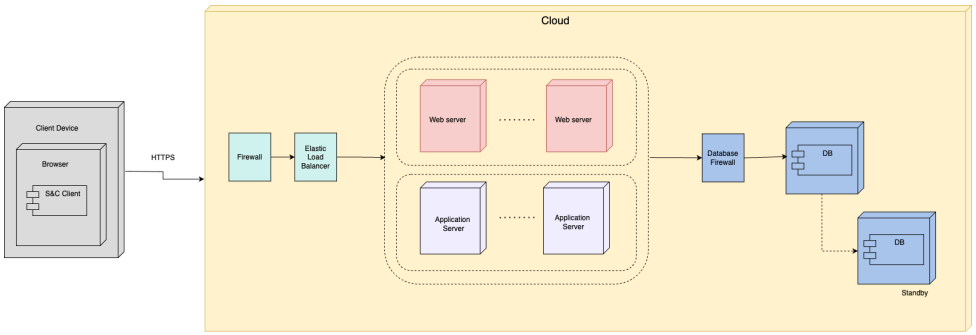

Figure 3: Deployment Diagram

2.4 Runtime View

- **User Sign Up**

The following diagram is the sequence diagram for the User Sign Up use case depicting all the components engaged for the stated use case. First the User goes to website of S&C in the browser and tries to Sign Up by giving all the required details. Using API calls the details of the user is sent to the Profile Manager component in the webs server. Profile Manager then sends the data to Entity Manager which connects the database to our server. Then finally data is added to the database. If saving of data is unsuccessful error message is shown to the user else success message is shown.

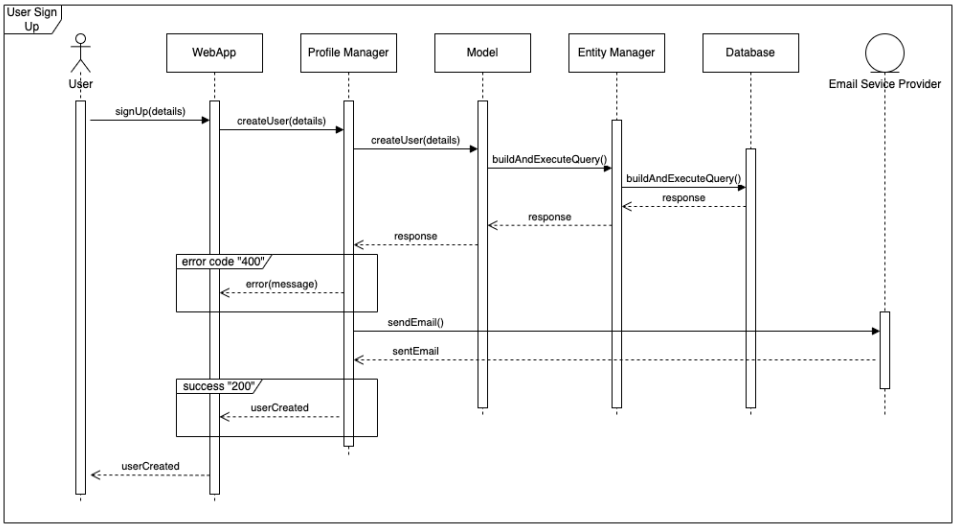

Figure 4: [UC1] Sign Up Sequence Diagram.

- **User Log In**

The following diagram is the runtime sequence diagram for the User Log In use case. When a registered user tries a go on the S&C WebApp and tries to log in an API request is sent from the client to the Profile Manager component in the web server along with the user entered credentials. These credentials are matched with the stored information in the database. If matched a token is generated by the Authenticate Manager and sent back to the client and the user's profile is visible.

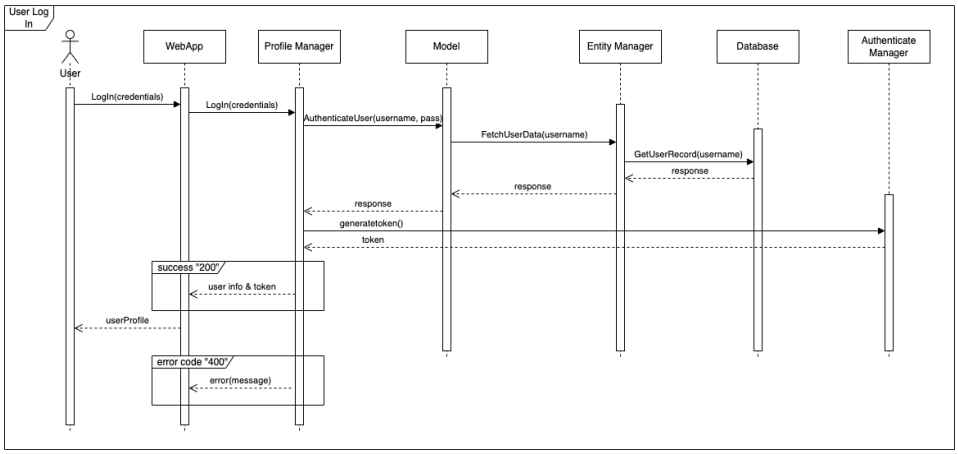

Figure 5: [UC2] Log In Sequence Diagram.

- **Creating CV**

The following diagram is the runtime sequence diagram for the Creating CV use case. A student user can create CV in the S&C application which is used for applying for internship. For creation of CV the student fills out all the form present in the S&C browser giving all details like education, skills, experiences etc. Once the user is satisfied the user can click on submit which calls the CreateCV API. The student's CV is ultimately saved in the database.

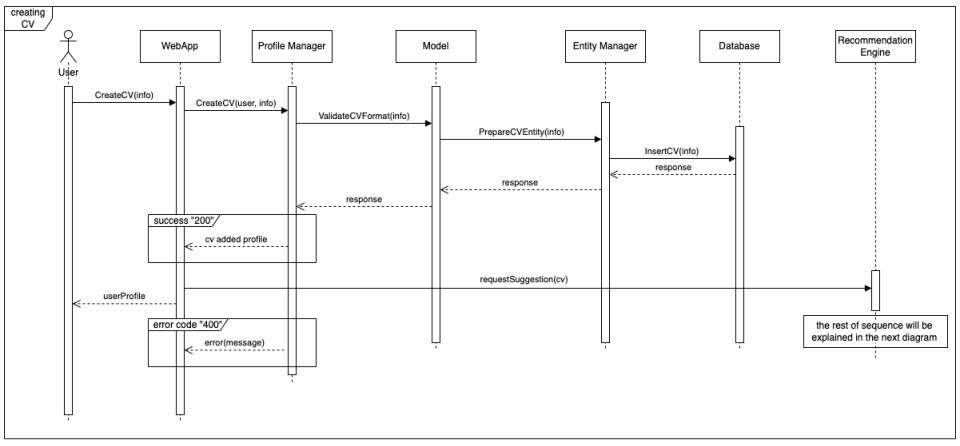

Figure 6: [UC3] Creating CV Sequence Diagram.

- **Get Suggestions on CV**

  While writing the CV, student can send requests to the Recommendation Engine in the application to get smart suggestions on the CV content. The Recommendation Engine analyzes the CV and provides tailored suggestions. These suggestions are displayed to the user, who can edit their CV based on the recommendations. Once the user finalizes and saves the edited CV, it is updated and stored in the system's database. The updated CV is reflected on the client-side. The following diagram illustrates the runtime sequence diagram for the Get Suggestions on CV use case.

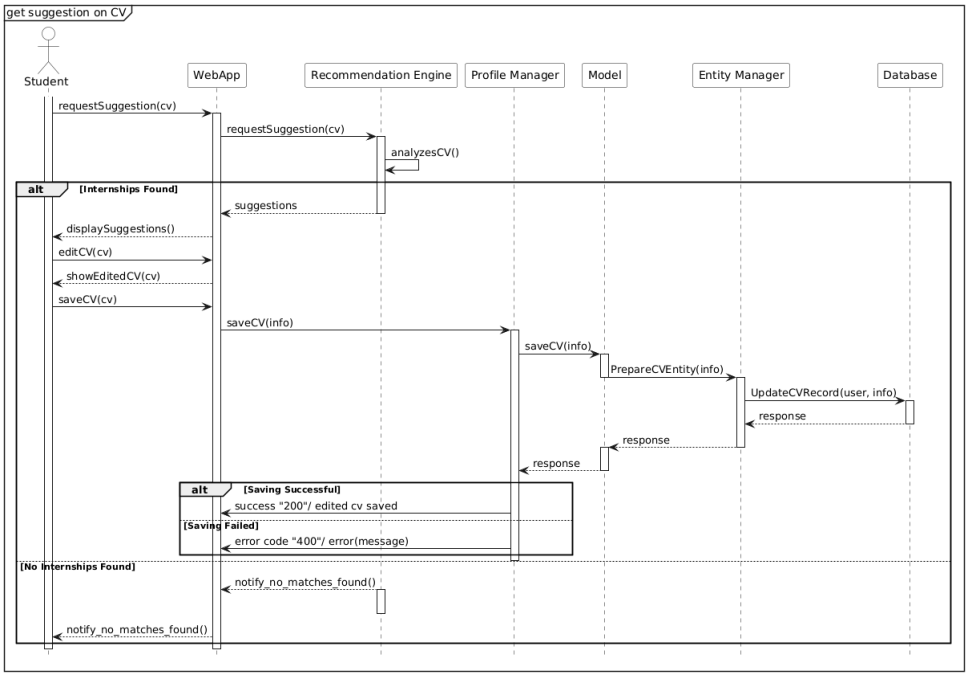

Figure 7: [UC4] Get suggestion on CV Sequence Diagram.

- **Post Internship Advertisement**

  After logging-in in the S&C website a company recruiter can post internship advertisement. To post the user fills out all the necessary details related to the internship position like job description, skills required, compensation in the website and then clicks on "Post" button and all the details related to the job is then saved in the Database. The user is then returned to their profile where they can see the posted internship. The following diagram is the runtime sequence diagram for the Post Internship Advertisement use case.

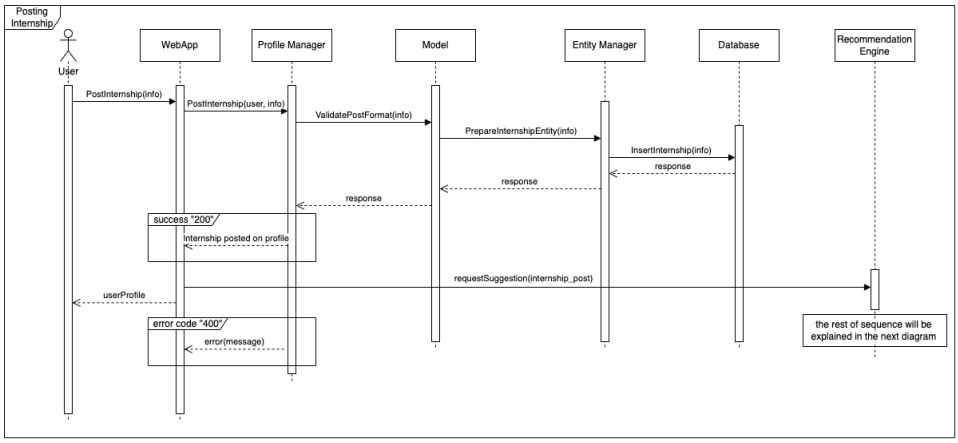

Figure 8: [UC5] Post Internship Sequence Diagram.

- **Get Suggestions on Ad**

  During the writing of an internship advertisement, companies can request suggestions from the S&C system to enhance the post. The content of the job post is sent to the Recommendation Engine, which analyzes the content and provides suggestions to improve the advertisement. These suggestions are displayed to the company user, who can either edit and finalize the ad or reject the suggestions. Once the edited advertisement is saved, it is updated in the system and reflected in the database. The following diagram illustrates the runtime sequence diagram for the Get Suggestions on Ad use case.

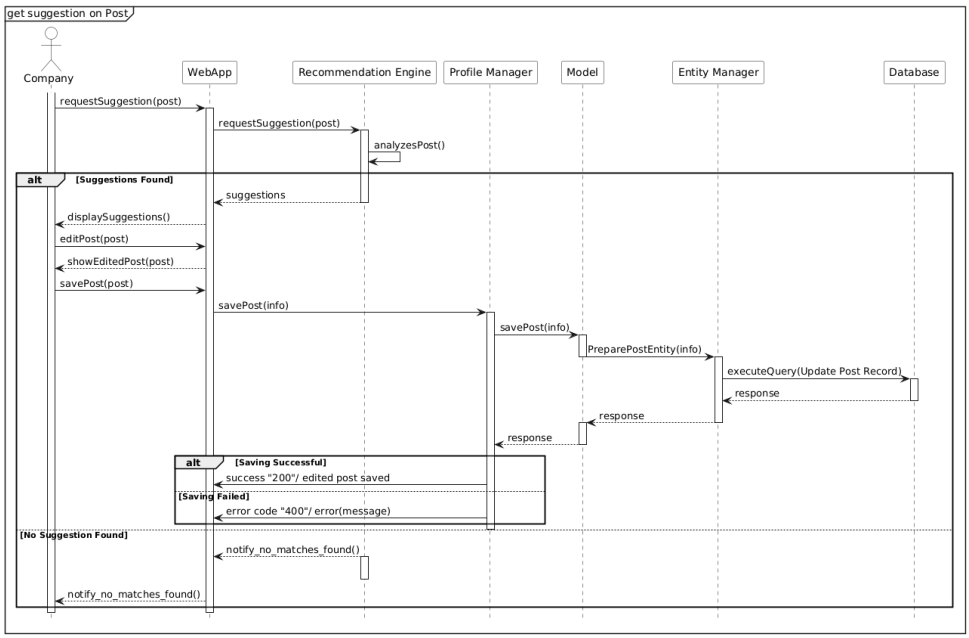

Figure 9: [UC6] Get suggestion on Internship Post Advertisement Sequence Diagram.

- **Define a Questionnaire**

  After posting an internship post the company personnel can create a Questionnaire to be sent to the students who are shortlisted for the next step. The company personnel can define the questionnaire for a particular internship offer by fetching details of the internship from the server. Once satisfied the user saves then questionnaire in the database for later use. The following diagram is the runtime sequence diagram for the Define a Questionnaire use case.

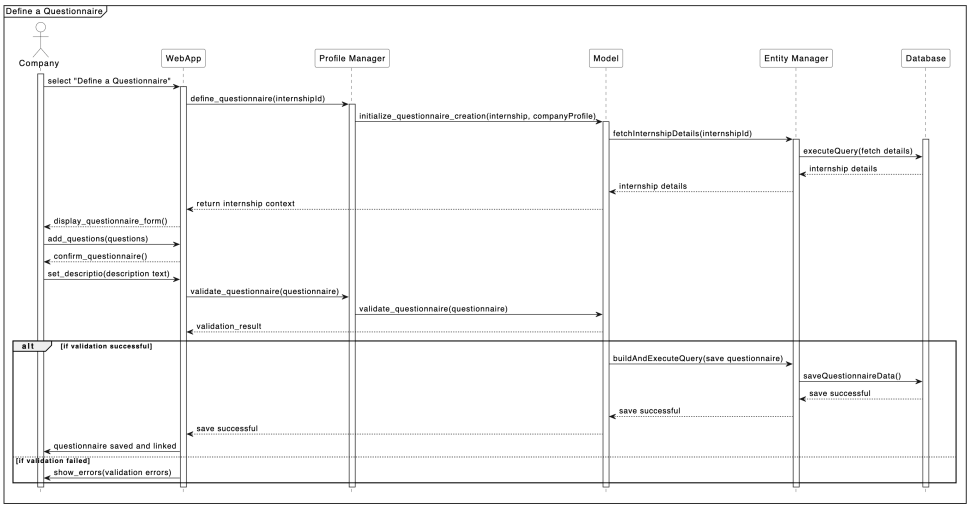

Figure 10: [UC7] Define a Questionnaire for the Internship Post Sequence Diagram.

- **Search Internship**

  Students can search for internship by typing required keywords like job roles, position, location in the search internship tab of the S&C website. ElasticSearch will be used for filtering and available internships are sent to the student. The student can then open each internship and view it's details. The following diagram is the runtime sequence diagram for the Search Internship use case.

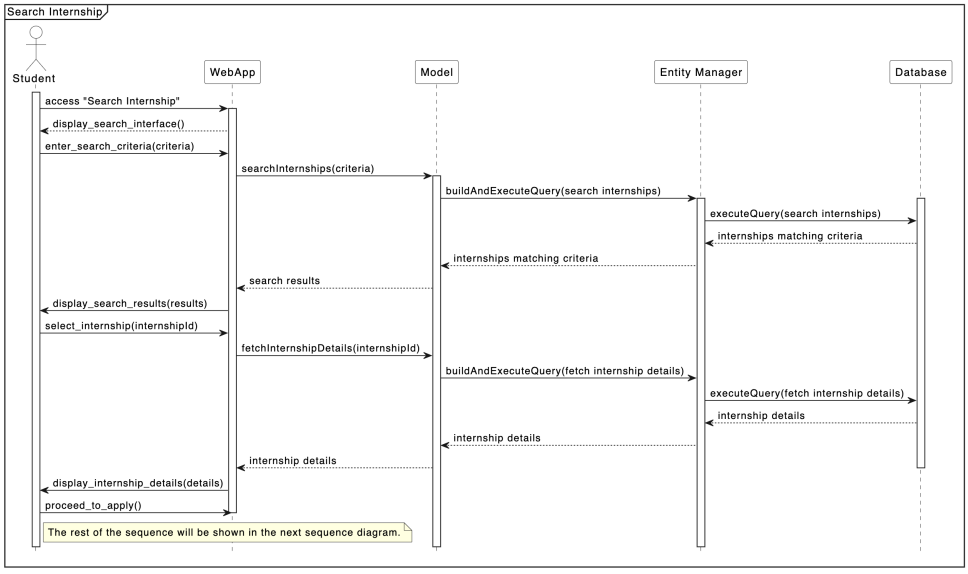

Figure 11: [UC8] Search Internship Sequence Diagram.

- **Apply Internship**

  A student can apply to any desired internship post by clicking on the "Apply" button. Student will be prompted to select application materials like CV, cover letter. Once selected the students application for that internship will be saved in the database. Once applied is submitted Notification Manager will send notification to the company. The following diagram is the runtime sequence diagram for the Apply Internship use case.

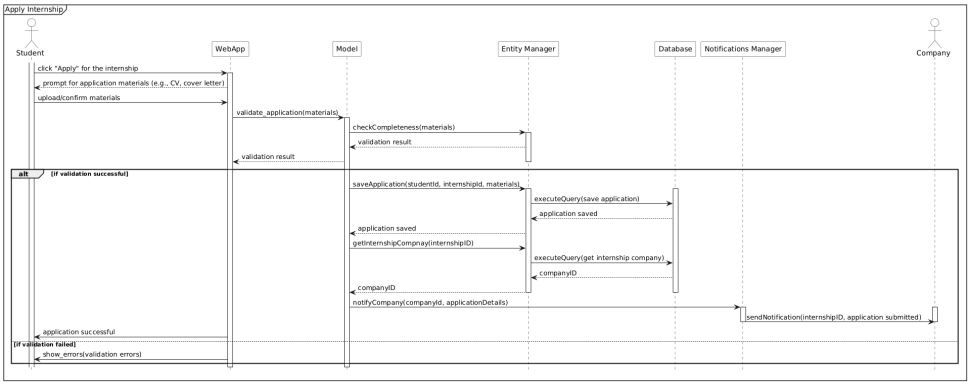

Figure 12: [UC9] Apply Internship Sequence Diagram.

- **Accept/Reject Application**

  A company recruiter can view the applications of students who have applied for their internship post. Company recruiter can fetch the the list of all applicants. Then user then clicks on "Accept" or "Reject" for all applications for the next step. The students are notified of the same. The application status of each application is then updated accordingly in the database. The following diagram is the runtime sequence diagram for the Accept/Reject Application use case.

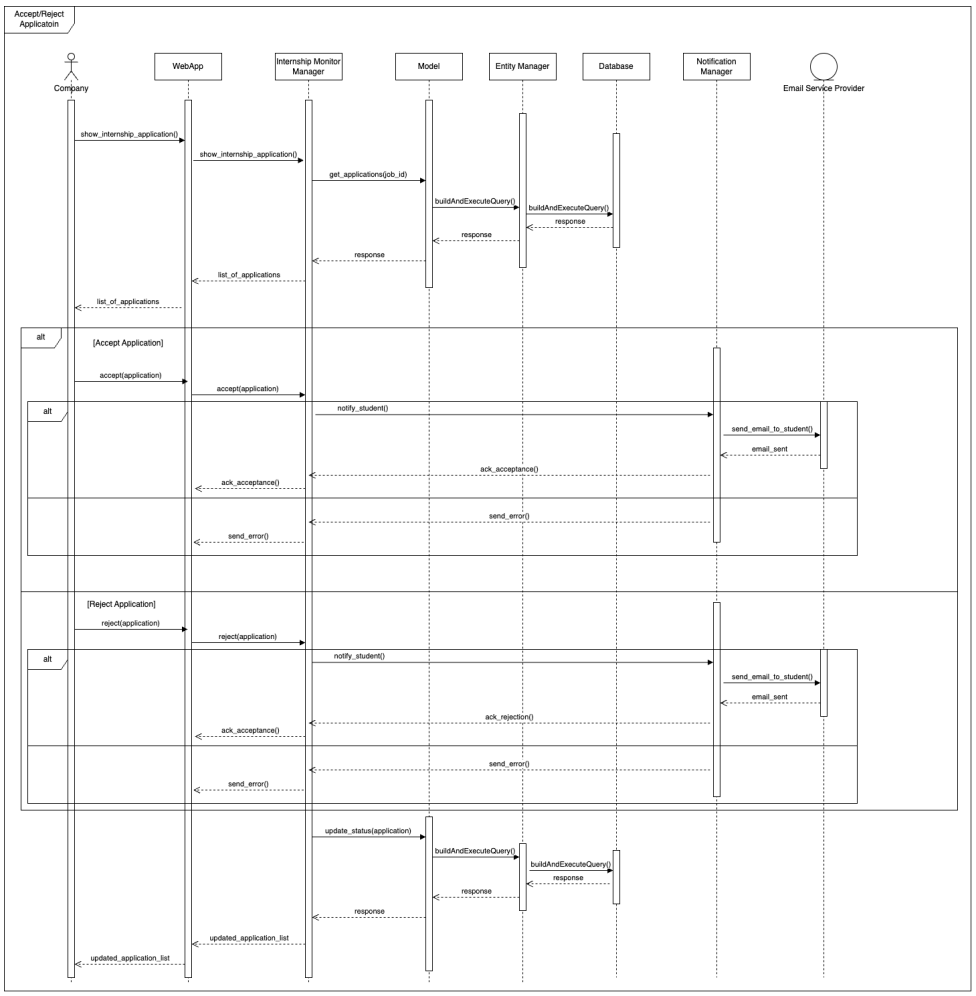

Figure 13: [UC10] Accept/Reject Application Sequence Diagram

- **Match Students and Internships**

  In S&C platform matchmaking system is there to notify students of available internship and notify companies and eligible students with required characteristics for an open internship are available. Recommendation Engine analyses CVs of students and descriptions of internships and for each student creates list of suitable companies and for each company creates list of suitable candidates and notifies each parties using email. This recommendation process is carried out at a periodic interval (for e.g. every 24 hours). The following diagram is the runtime sequence diagram for the Match Student and Internship Sequence Diagram use case.

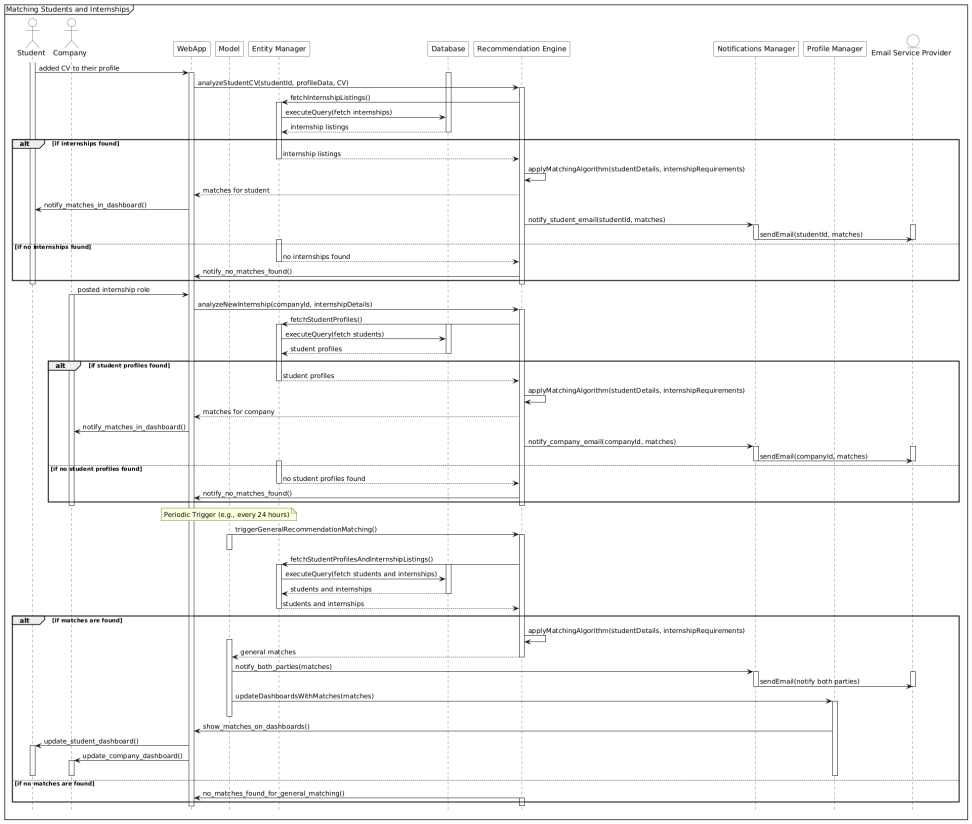

Figure 14: [UC11] Match Student and Internship Sequence Diagram

- **Selection Process**

  Once a company accepts an application or a match-making recommendation is accepted by both the company and student the selection process starts. A communication channel is opened between the company and the student both the parties are notified. The previously defined questionnaire is forwarded to the student to be answered. The Student then fills up the questionnaire and submits it which is then can be read by the company. The company can then either select the student for the interview or reject them. In the both the cases the students will be notified and the selected student will be added to the interview shortlist. The following diagram is the runtime sequence diagram for the Selection Process Sequence Diagram use case.

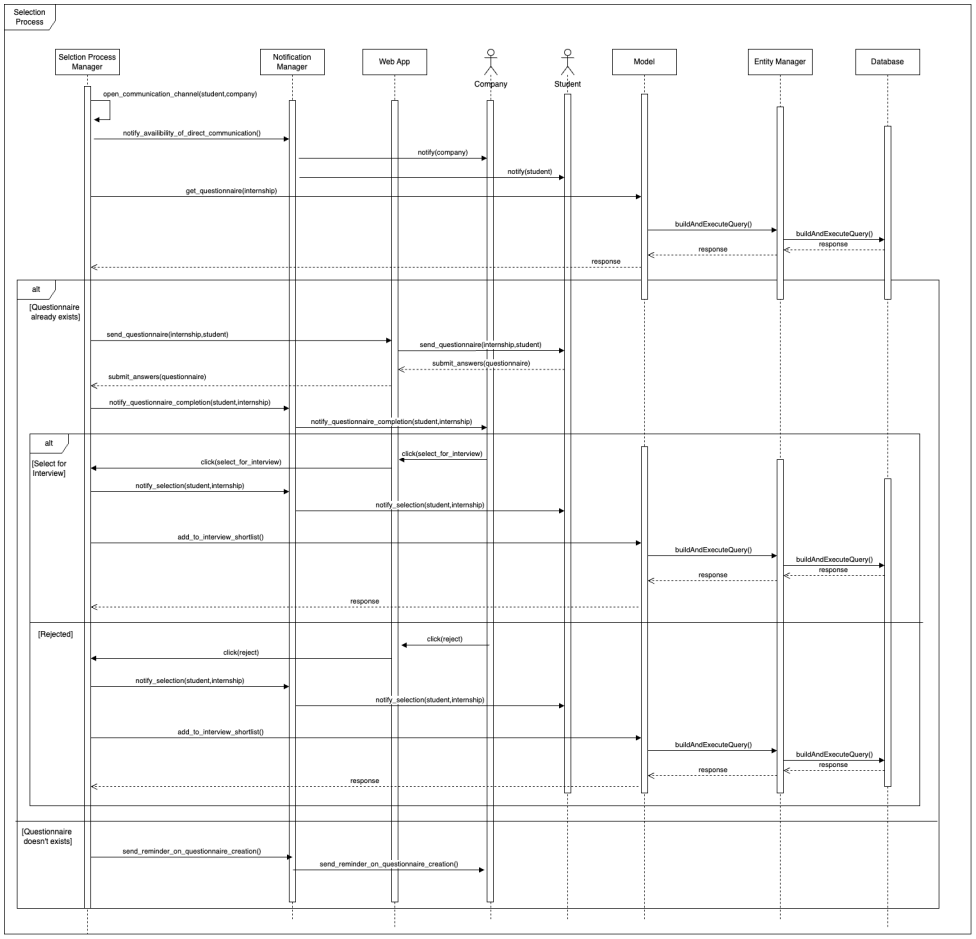

Figure 15: [UC12] Selection Process Sequence Diagram

- **Set Up Interview (S&C Platform Manage Interview Process)**

  Once company selects a student for interview the Selection Process Manager will request both student and company to select range of available dates in the calender. Once dates are selected an interview date will be selected and interview details along with dates and location or interview link will be sent to both the student and company using Notification Manager and the data will be saved in the database. The following diagram is the runtime sequence diagram for the Set Up Interview (S&C Platform Manage Interview Process) Sequence Diagram use case.

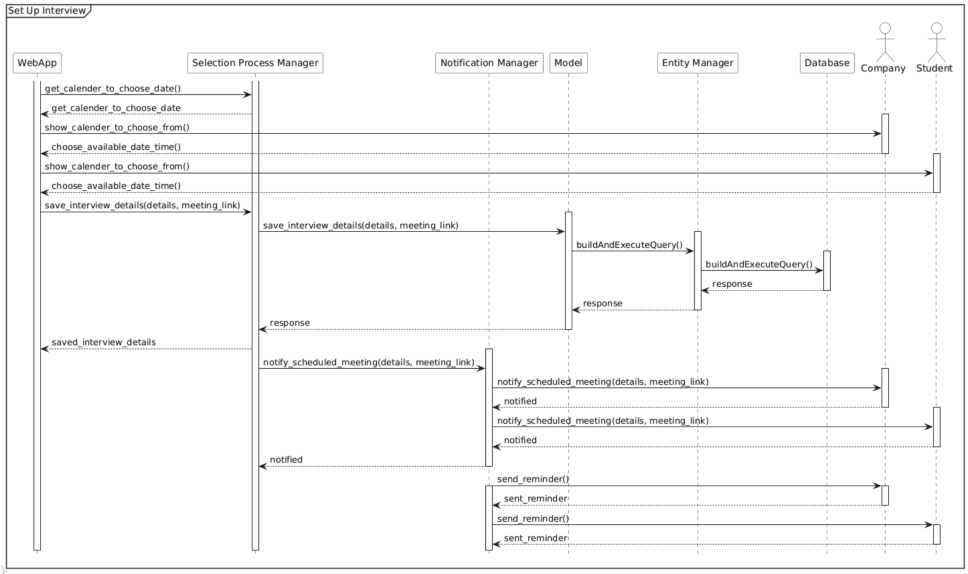

Figure 16: [UC13] Set Up Interview Sequence Diagram

- **Finalize Decision (company finalize the decision for selection process)**

  Company can see the lift of the interview students and from there hire or reject a student for that post. If the company decides to hire a student notification is sent to that student. If the company rejects then notification is sent to that student. The decisions are then updated in the databases. If no students have been interviewed yet it is reflected in the website. The following diagram is the runtime sequence diagram for the Finalize Decision Sequence Diagram use case.

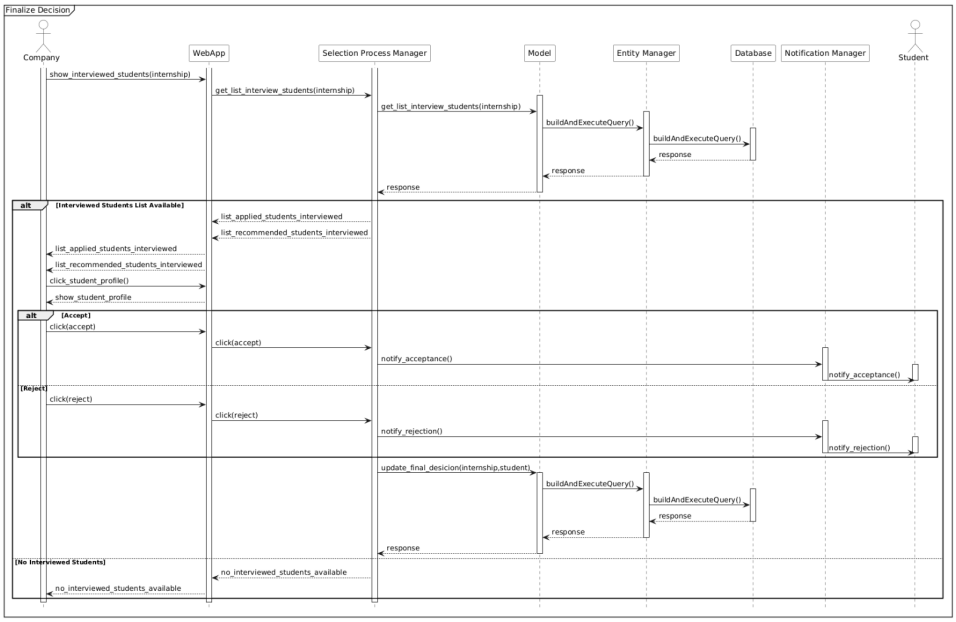

Figure 17: [UC14] Set Up Interview Sequence Diagram

- **Collect Feedback**

  After completion of the interview or while using of the application the students and companies are requested by the Feedback and Analytics manager to give feedback on the website and it's applications. The feedback given by the users are send saved in the database. The feedback is then used by the developers of the system to improve the recommendation engine. After improvement the recommendation engine can suggest better and more accurate matches to students and companies. The following diagram is the runtime sequence diagram for the Collect Feedback Sequence Diagram use case.

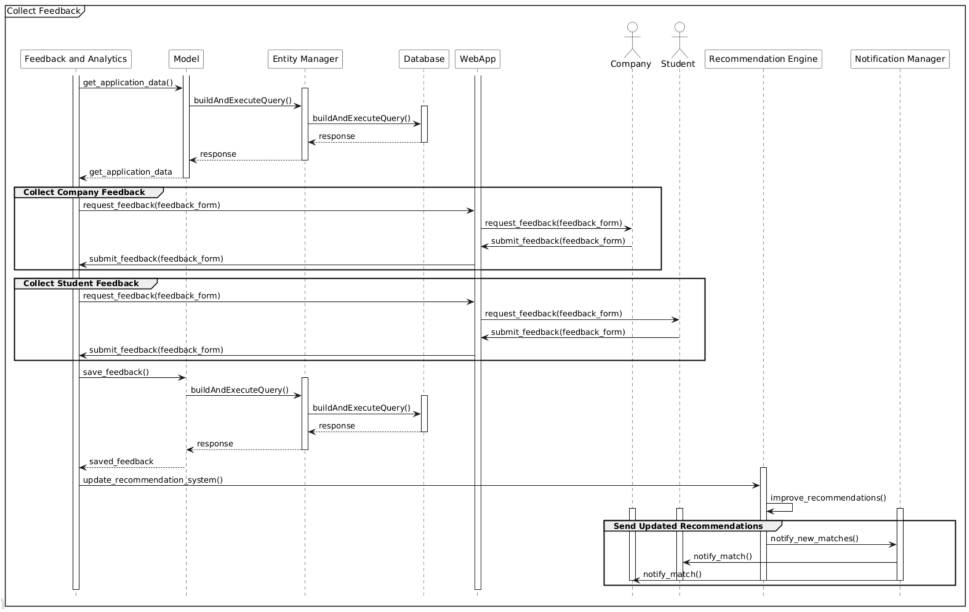

Figure 18: [UC15] Collect Feedback Sequence Diagram

- **Monitor Internship Status**

  Users can monitor the internship status. University can monitor internships of it's students. After login university administrator can go to university's profile page and from there can see the list of their students. For each students university administrator can get the internship details and progress from database which is handled by Internship Monitor Manager. The following diagram is the runtime sequence diagram for the Internship Monitor Manage Sequence Diagram use case.

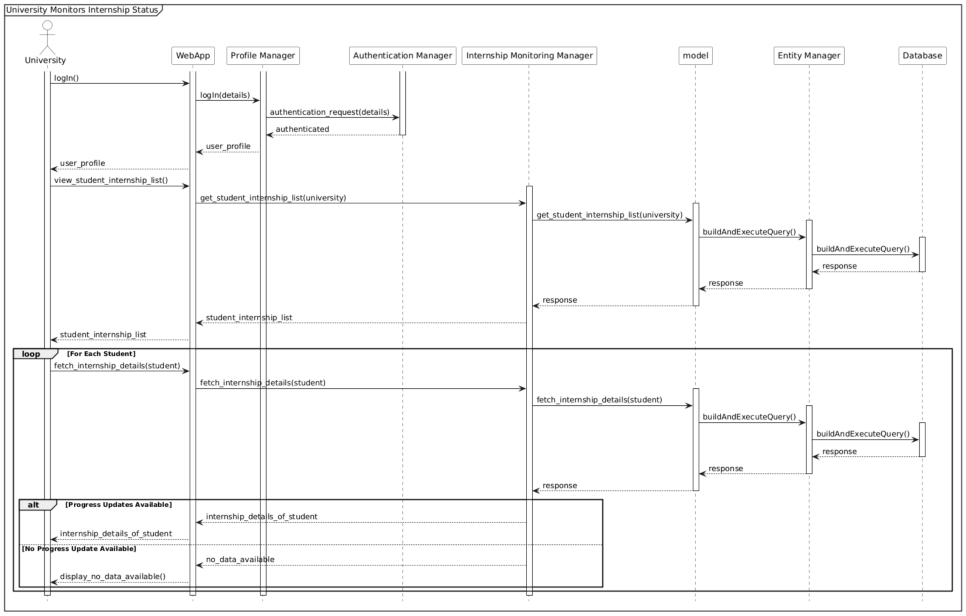

Figure 19: [UC16] Monitor Internship Status Sequence Diagram

- **Handle Complaints and Issues**

  University can login in to the S&C web application then click on show complaints. It will fetch complaints through the Complaints Manager. If any complaints exist, university can take any decision which will be notified to the students and then mark the complaint as "Resolved" for flag for further actions required. In both the cases the complaint status and details will be updated in the database. The following diagram is the runtime sequence diagram for the Handle Complaints and Issues Sequence Diagram use case.

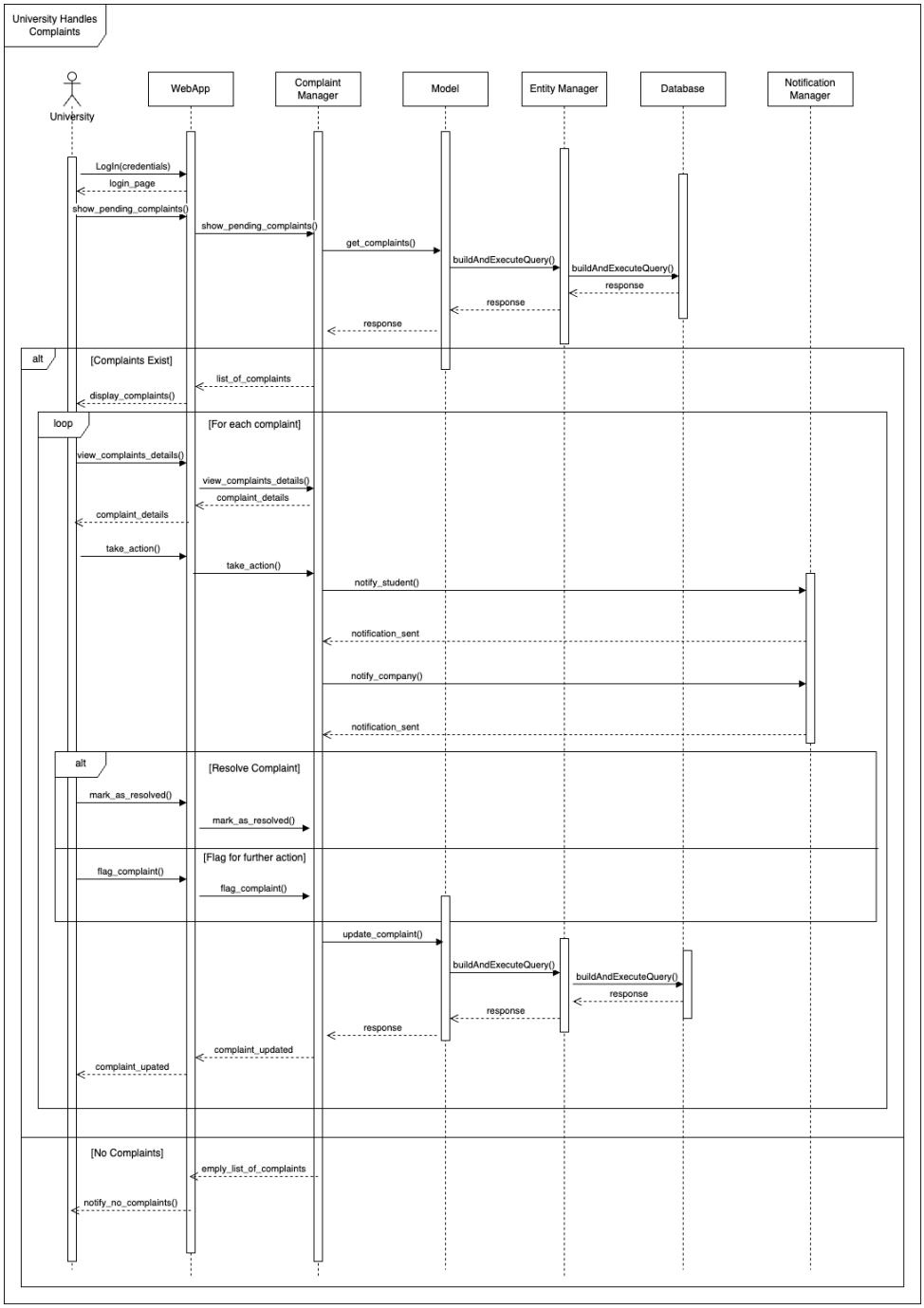

Figure 20: [UC17] University Handles Complaints Sequence Diagram

2.5 Component interfaces

In this section as to represent Component interfaces we are representing API endpoints. The main focus is on the methods used, the parameters required and on the responses.

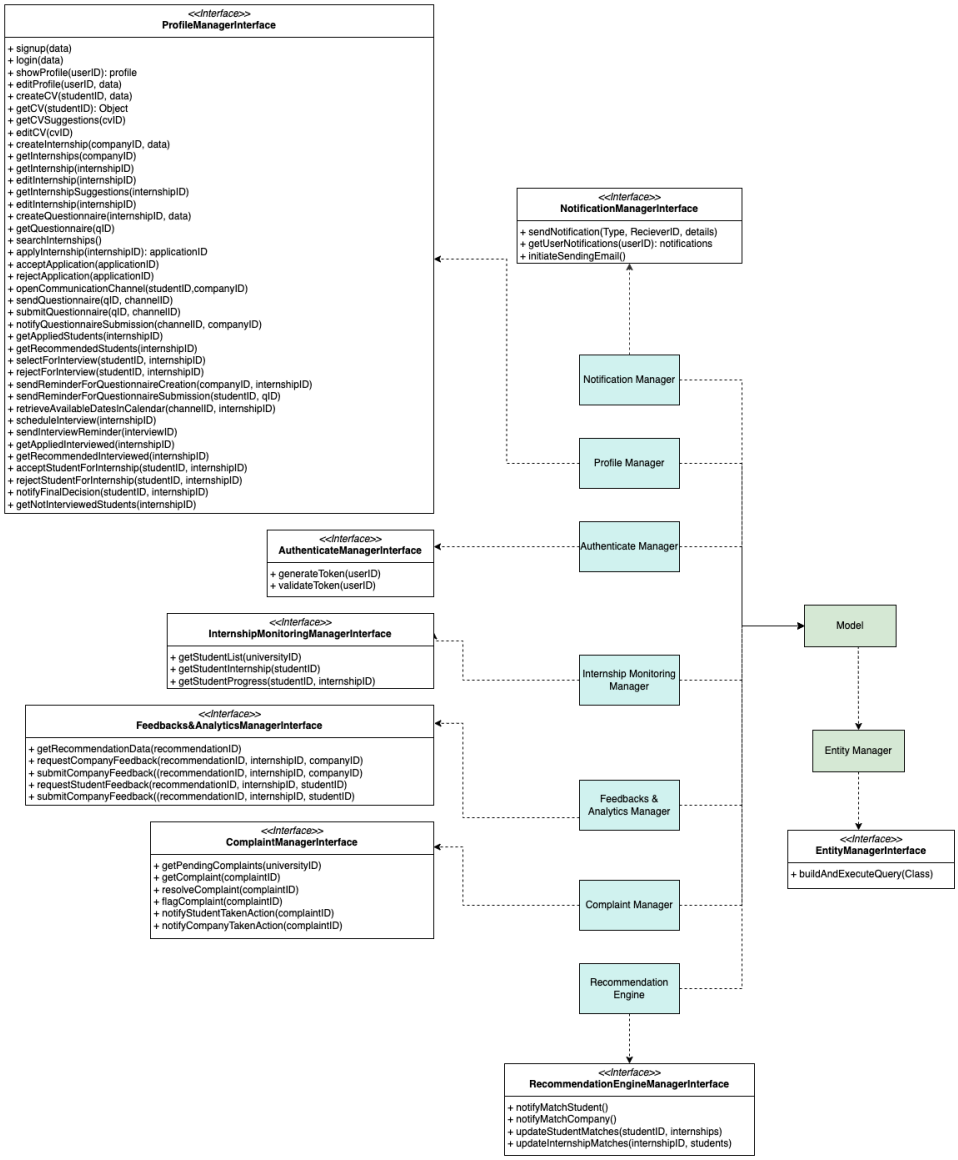

Figure 21: Component Interfaces Class Diagram

2.5.1 API Endpoints

- **POST /api/users/signup**
  - **Description:** Registers a new user.
  - **Request Body:** `{ name, email, password, role, ...}`
  - **Response:**
    - 201 Created: `{ userId }`
    - 400 Bad Request: Validation errors.

- **POST /api/users/login**
  - **Description:** Logs in an existing user.
  - **Request Body:** `{ email, password }`
  - **Response:**
    - 200 OK: `{ token, userDetails }`
    - 401 Unauthorized: Invalid credentials.
- **GET /api/users/profile/{userId}**
  - **Description:** Fetches the profile of a user.
  - **Path Parameters:** `userId`
  - **Response:**
    - 200 OK: `{ userProfile }`
    - 404 Not Found: User does not exist.
- **PUT /api/users/editprofile/{userId}**
  - **Description:** Updates the profile of a user.
  - **Request Body:** `{ profileUpdates }`
  - **Response:**
    - 200 OK: `{ success: true }`
    - 400 Bad Request: Validation errors.
- **POST /api/cv/create**
  - **Description:** Creates a CV for a student.
  - **Request Body:** `{ studentId, cvDetails }`
  - **Response:**
    - 201 Created: `{ cvId }`
    - 400 Bad Request: Validation errors.
- **GET /api/cv/getcv/{studentId}**
  - **Description:** Retrieves the CV of a specific student.
  - **Path Parameters:** `studentId`
  - **Response:**
    - 200 OK: `{ cvDetails }`
    - 404 Not Found: No CV found for the specified student.
- **POST /api/cv/suggestions/{studentId}**
  - **Description:** Requests suggestions for improving a specific student's CV.
  - **Path Parameters:** `studentId`
  - **Request Body:** `{ details: [...] }`
  - **Response:**
    - 200 OK: `{ suggestions: [...] }`
    - 404 Not Found: CV not found for the specified student.

        * 400 Bad Request: Validation errors.

- **POST /api/cv/edit/{cvId}**
  - **Description:** Allows a student to edit an existing CV.
  - **Path Parameters:** `cvId`
  - **Request Body:** `{ updatedCvDetails }`
  - **Response:**
    * 200 OK: `{ success: true, message: "CV updated successfully." }`
    * 404 Not Found: No CV found with the specified `cvId`.
    * 400 Bad Request: Validation errors in the provided CV details.

- **POST /api/companies/internships/create/{companyId}**
  - **Description:** Creates a new internship posting for a specific company.
  - **Path Parameters:** `companyId`
  - **Request Body:** `{ title, description, requirements }`
  - **Response:**
    * 201 Created: `{ internshipId }`
    * 400 Bad Request: Validation errors.

- **GET /api/companies/internships/{companyId}**
  - **Description:** Retrieves a list of internships for a specific company.
  - **Path Parameters:** `companyId`
  - **Query Parameters:** `{ filterOptions }`
  - **Response:**
    * 200 OK: `{ internships: [...] }`
    * 401 Unauthorized: Access denied.

- **GET /api/internships/details/{internshipId}**
  - **Description:** Retrieves detailed information about a specific internship.
  - **Path Parameters:** `internshipId`
  - **Response:**
    * 200 OK: `{ internshipDetails: ... }`
    * 404 Not Found: No internship found with the specified `internshipId`.
    * 400 Bad Request: Invalid internship ID.

- **POST /api/companies/internships/suggestions/{companyId}/{internshipId}**
  - **Description:** Requests suggestions for improving a specific internship post created by a specific company.
  - **Path Parameters:** `companyId`, `internshipId`
  - **Request Body:** `{ details: [...] }`
  - **Response:**
    * 200 OK: `{ suggestions: [...] }`
    * 404 Not Found: Internship not found for the specified company.

    - 400 Bad Request: Validation errors.

- **POST /api/internships/edit/{internshipId}**
  - **Description:** Allows a company to edit an existing internship posting.
  - **Path Parameters:** `internshipId`
  - **Request Body:** `{ updatedInternshipDetails }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Internship updated successfully." }`
    - 404 Not Found: No internship found with the specified `internshipId`.
    - 400 Bad Request: Validation errors in the provided internship details.

- **POST /api/internships/questionnaire/{internshipId}**
  - **Description:** Creates a questionnaire for an internship.
  - **Request Body:** `{ questions: [...], internshipId }`
  - **Response:**
    - 201 Created: `{ questionnaireId }`
    - 400 Bad Request: Validation errors.

- **GET /api/internships/questionnaire/{internshipId}**
  - **Description:** Retrieves the questionnaire for an internship.
  - **Path Parameters:** `internshipId`
  - **Response:**
    - 200 OK: `{ questionnaireDetails }`
    - 404 Not Found: Questionnaire not found.

- **POST /api/notifications/student/matches/{studentId}**
  - **Description:** Sends a notification to a student about matching internships found.
  - **Path Parameters:** studentId
  - **Request Body:** `{ "matches": [...] }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Notification sent to student." }`
    - 404 Not Found: Student not found.
    - 500 Internal Server Error: Notification sending failed.

- **POST /api/notifications/company/matches/{companyId}**
  - **Description:** Sends a notification to a company about matching students found for an internship.
  - **Path Parameters:** companyId
  - **Request Body:** `{ "matches": [...] }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Notification sent to company." }`

    - 404 Not Found: Company not found.
    - 500 Internal Server Error: Notification sending failed.

- **POST /api/dashboards/student/update-matches{studentId}**
  - **Description:** Updates a student's dashboard with the latest matches found by the recommendation engine.
  - **Path Parameters:** studentId
  - **Request Body:** `{ "matches": [...] }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Dashboard updated successfully." }`
    - 404 Not Found: Student not found.
    - 500 Internal Server Error: Dashboard update failed.

- **POST /api/dashboards/company/update-matches/{companyId}**
  - **Description:** Updates a company's dashboard with the latest matches found by the recommendation engine.
  - **Path Parameters:** companyId
  - **Request Body:** `{ "matches": [...] }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Dashboard updated successfully." }`
    - 404 Not Found: Company not found.
    - 500 Internal Server Error: Dashboard update failed.

- **GET /api/internships/search**
  - **Description:** Allows a student to search for internships based on specified criterias.
  - **Request Body:** A JSON object containing the search criteria.
  - **Response:**
    - 200 OK: `{ internships: [...] }`
      - Returns a list of internships that match the given criteria.
    - 400 Bad Request: Validation errors in the provided criteria.
    - 401 Unauthorized: User is not authenticated.

- **POST /api/internships/apply/{internshipId}**
  - **Description:** Allows a student to apply to a specific internship.
  - **Request Body:** A JSON object containing the application details.
  - **Path Parameters:**
    - `internshipId`: The ID of the internship the student is applying to.
  - **Response:**
    - 200 OK: `{ success: true, message: "Application submitted successfully." }`
    - 400 Bad Request: Validation errors in the application details.
    - 401 Unauthorized: User is not authenticated.

    - 404 Not Found: Internship not found.

- **POST /api/internships/applications/accept/{internshipId}/{applicationId}**
  - **Description:** Allows a company to accept a student's application for a specific internship.
  - **Path Parameters:**
    - `internshipId`: The ID of the internship the application belongs to.
    - `applicationId`: The ID of the student's application.
  - **Response:**
    - 200 OK: `{ success: true, message: "Application accepted successfully." }`
    - 400 Bad Request: Validation errors or missing data.
    - 401 Unauthorized: User is not authenticated or authorized.
    - 404 Not Found: Internship or application not found.

- **POST /api/internships/applications/reject/{internshipId}/{applicationId}**
  - **Description:** Allows a company to reject a student's application for a specific internship.
  - **Path Parameters:** internshipId, applicationId
  - **Response:**
    - 200 OK: `{ success: true, message: "Application rejected successfully." }`
    - 400 Bad Request: Validation errors or missing data.
    - 401 Unauthorized: User is not authenticated or authorized.
    - 404 Not Found: Internship or application not found.

- **POST /api/communication/open/{studentId}/{companyId}**
  - **Description:** Opens a communication channel between a student and a company during the selection process.
  - **Path Parameters:** studentId, companyId
  - **Response:**
    - 200 OK: `{ success: true, message: "Communication channel opened successfully." }`
    - 400 Bad Request: Validation errors in the request.
    - 401 Unauthorized: User is not authenticated.

- **POST /api/questionnaire/send/{qId}/{studentId}/{internshipId}/{channelId}**
  - **Description:** Sends a questionnaire from the company to a student as part of the selection process.
  - **Path Parameters:** studentId, internshipId, channelId, qId
  - **Request Body:** `{ "questions": ["string", "string", ...] }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Questionnaire sent successfully." }`
    - 404 Not Found: Internship or student not found.
    - 400 Bad Request: Validation errors in the request body.

- **POST /api/questionnaire/submit//{qId}{studentId}/{internshipId}/{channelId}**
  - **Description:** Allows a student to submit answers to a received questionnaire.
  - **Path Parameters:** studentId, internshipId, channelId, qId
  - **Request Body:** `{ "answers": ["string", "string", ...] }`
  - **Response:**
    * 200 OK: `{ success: true, message: "Answers submitted successfully." }`
    * 404 Not Found: Questionnaire not found.
    * 400 Bad Request: Validation errors in the answers.
- **POST /api/questionnaire/notify/{channelId}/{companyId}/{internshipId}**
  - **Description:** Notifies the company when a questionnaire has been submitted by the student.
  - **Path Parameters:** channelId, companyId, internshipId
  - **Response:**
    * 200 OK: `{ success: true, message: "Notification sent successfully." }`
    * 500 Internal Server Error: Failed to send the notification.
- **GET /api/internships/applied-students/{internshipId}**
  - **Description:** Retrieves a list of students who applied for a specific internship.
  - **Path Parameters:** `internshipId`
  - **Response:**
    * 200 OK: `{ students: [...] }`
    * 404 Not Found: No internship found with the specified `internshipId`, or no students applied.
    * 400 Bad Request: Invalid internship ID.
- **GET /api/internships/recommended-students/{internshipId}**
  - **Description:** Retrieves a list of students recommended by the system for a specific internship.
  - **Path Parameters:** `internshipId`
  - **Response:**
    * 200 OK: `{ recommendedStudents: [...] }`
    * 404 Not Found: No internship found with the specified `internshipId`, or no recommended students available.
    * 400 Bad Request: Invalid internship ID.
- **POST /api/interviews/select/{studentId}/{internshipId}**
  - **Description:** Adds a student to the interview shortlist for a specific internship.
  - **Path Parameters:** studentId, internshipId
  - **Response:**
    * 200 OK: `{ success: true, message: "Student added to interview shortlist." }`

    - 404 Not Found: Internship or student not found.
    - 400 Bad Request: Validation errors in the request.

- **POST /api/interviews/reject/{studentId}/{internshipId}**
  - **Description:** Rejects a student for a specific internship and removes them from the selection process.
  - **Path Parameters:** studentId, internshipId
  - **Response:**
    - 200 OK: `{ success: true, message: "Student rejected successfully." }`
    - 404 Not Found: Internship or student not found.
    - 400 Bad Request: Validation errors in the request.

- **POST /api/questionnaire/reminder/{studentId}/{internshipId}**
  - **Description:** Sends a reminder to the student about an incomplete questionnaire.
  - **Path Parameters:** studentId, internshipId
  - **Response:**
    - 200 OK: `{ success: true, message: "Reminder sent successfully." }`
    - 404 Not Found: Questionnaire not found.
    - 500 Internal Server Error: Failed to send the reminder.

- **POST /api/questionnaire/reminder/{studentId}/{qId}**
  - **Description:** Sends a reminder to a student to complete and submit a specific questionnaire.
  - **Path Parameters:** `studentId`, `qId`
  - **Response:**
    - 200 OK: `{ success: true, message: "Reminder sent successfully." }`
    - 404 Not Found: Student or questionnaire not found with the specified IDs.
    - 400 Bad Request: Invalid student ID or questionnaire ID.
    - 500 Internal Server Error: Failed to send the reminder.

- **GET /api/interviews/calendar/{internshipId}/{channelId}**
  - **Description:** Retrieves available dates and times for scheduling an interview.
  - **Path Parameters:** internshipId, channelId
  - **Response:**
    - 200 OK: `{ availableDates: [ "YYYY-MM-DD HH:mm", ... ] }`
    - 400 Bad Request: Invalid internship ID or unavailable calendar data.
    - 401 Unauthorized: User is not authenticated.

- **POST /api/interviews/schedule/{internshipId}**
  - **Description:** Schedules an interview for a specific internship by providing date, time, and meeting link details in the student's and company's dashboard.
  - **Path Parameters:**
    - `internshipId`: The ID of the internship for which the interview is being scheduled.

  - **Request Body:** `{ "date": "YYYY-MM-DD", "time": "HH:mm", "meetingLink": "string" }`
  - **Response:**
    * 200 OK: `{ success: true, message: "Interview scheduled successfully." }`
    * 400 Bad Request: Invalid date, time, or missing meeting link.
    * 404 Not Found: Internship not found.
    * 401 Unauthorized: User is not authenticated.

- **POST /api/interviews/notify/{interviewId}**
  - **Description:** Sends notifications to the student and the company about the scheduled interview.
  - **Path Parameters:** interviewId
  - **Request Body:** `{ "message": "string" }`
  - **Response:**
    * 200 OK: `{ success: true, message: "Notifications sent successfully." }`
    * 404 Not Found: Interview ID not found.
    * 500 Internal Server Error: Failed to send notifications.

- **POST /api/interviews/reminder/{interviewId}**
  - **Description:** Sends reminders to both the student and company before the interview.
  - **Path Parameters:**
    * `interviewId`: The ID of the interview for which reminders will be sent.
  - **Response:**
    * 200 OK: `{ success: true, message: "Reminders sent successfully." }`
    * 404 Not Found: Interview ID not found.
    * 500 Internal Server Error: Failed to send reminders.

- **GET /api/internships/interviewed-applied-students/{internshipId}**
  - **Description:** Retrieves a list of students who were interviewed and had applied for a specific internship.
  - **Path Parameters:** internshipId
  - **Response:**
    * 200 OK: `{ students: [...] }`
    * 404 Not Found: Internship not found.
    * 400 Bad Request: Invalid internship ID.

- **GET /api/internships/interviewed-recommended-students/{internshipId}**
  - **Description:** Retrieves a list of students who were interviewed based on the system's recommendations for a specific internship.
  - **Path Parameters:** internshipId
  - **Response:**
    * 200 OK: `{ students: [...] }`

    - 404 Not Found: Internship not found.
    - 400 Bad Request: Invalid internship ID.

- **POST /api/internships/accept/{internshipId}/{studentId}**
  - **Description:** Accepts a student for a specific internship and notifies them.
  - **Path Parameters:** internshipId, studentId
  - **Response:**
    - 200 OK: `{ success: true, message: "Student accepted and notified." }`
    - 404 Not Found: Internship or student not found.
    - 400 Bad Request: Invalid parameters.

- **POST /api/internships/reject/{internshipId}/{studentId}**
  - **Description:** Rejects a student for a specific internship and notifies them.
  - **Path Parameters:** internshipId, studentId
  - **Response:**
    - 200 OK: `{ success: true, message: "Student rejected and notified." }`
    - 404 Not Found: Internship or student not found.
    - 400 Bad Request: Invalid parameters.

- **POST /api/internships/notify/{internshipId}/{studentId}**
  - **Description:** notifies the student of the final decision for a specific student in the context of an internship (either accept or reject).
  - **Path Parameters:** internshipId, studentId
  - **Request Body:** `{ "message": "string" }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Final decision notified." }`
    - 404 Not Found: Internship or student not found.
    - 400 Bad Request: Invalid decision or parameters.

- **GET /api/internships/no-interviewed-students/{internshipId}**
  - **Description:** Checks if there are no interviewed students left for finalization in a specific internship.
  - **Path Parameters:** internshipId
  - **Response:**
    - 200 OK: `{ noStudents: true | false }`
    - 404 Not Found: Internship not found.
    - 400 Bad Request: Invalid internship ID.

- **GET /api/recommendations/data/{recommendationId}**
  - **Description:** Retrieves detailed data about a specific recommendation.
  - **Path Parameters:** recommendationId
  - **Response:**

    - 200 OK: `{ recommendationData: ... }`
    - 404 Not Found: Recommendation not found.
    - 400 Bad Request: Invalid recommendation ID.

- **POST /api/feedback/request/company/{internshipId}/{companyId}/{recommendationId}**
  - **Description:** Sends a request to the company for feedback on a specific recommendation.
  - **Path Parameters:** internshipId, companyId, recommendationId
  - **Response:**
    - 200 OK: `{ success: true, message: "Feedback request sent to the company." }`
    - 404 Not Found: Internship, company, or recommendation not found.
    - 400 Bad Request: Invalid recommendation ID.

- **POST /api/feedback/submit/company/{internshipId}/{companyId}/{recommendationId}**
  - **Description:** Allows a company to submit feedback on a specific recommendation.
  - **Path Parameters:** internshipId, companyId, recommendationId
  - **Request Body:** `{ "feedback": Object }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Feedback submitted successfully." }`
    - 404 Not Found: Internship, company, or recommendation not found.
    - 400 Bad Request: Validation errors in the feedback.

- **POST /api/feedback/request/student/{internshipId}/{studentId}/{recommendationId}**
  - **Description:** Sends a request to a student for feedback on a specific recommendation.
  - **Path Parameters:** internshipId, studentId, recommendationId
  - **Response:**
    - 200 OK: `{ success: true, message: "Feedback request sent to the student." }`
    - 404 Not Found: Internship, student, or recommendation not found.
    - 400 Bad Request: Invalid recommendation ID.

- **POST /api/feedback/submit/student/{internshipId}/{studentId}/{recommendationId}**
  - **Description:** Allows a student to submit feedback on a specific recommendation.
  - **Path Parameters:** internshipId, studentId, recommendationId
  - **Request Body:** `{ "feedback": Object }`
  - **Response:**
    - 200 OK: `{ success: true, message: "Feedback submitted successfully." }`
    - 404 Not Found: Internship, student, or recommendation not found.
    - 400 Bad Request: Validation errors in the feedback.

- **POST /api/feedback/save/{recommendationId}**

  - **Description:** Saves feedback provided by a student or company for a specific recommendation to the database.
  - **Path Parameters:** recommendationId
  - **Request Body:** `{ "feedback": Object }`
  - **Response:**
    * 200 OK: `{ success: true, message: "Feedback saved successfully." }`
    * 404 Not Found: Recommendation not found.
    * 400 Bad Request: Validation errors in the feedback.

- **GET /api/internships/student-list/university/{universityId}**
  - **Description:** Retrieves a list of students and their internships associated with a specific university.
  - **Path Parameters:** universityId
  - **Response:**
    * 200 OK: `{ students: [...] }`
    * 404 Not Found: University not found or no students associated.
    * 400 Bad Request: Invalid university ID.

- **GET /api/internships/details/student/{studentId}**
  - **Description:** Retrieves detailed internship information for a specific student.
  - **Path Parameters:** studentId
  - **Response:**
    * 200 OK: `{ internshipDetails: ... }`
    * 404 Not Found: Student or internship details not found.
    * 400 Bad Request: Invalid student ID.

- **GET /api/internships/progress/student/{studentId}/{internshipId}**
  - **Description:** Retrieves progress updates for a student's internship.
  - **Path Parameters:** studentId, internshipId
  - **Response:**
    * 200 OK: `{ progressUpdates: [...] }`
    * 404 Not Found: No progress updates available for the student.
    * 400 Bad Request: Invalid student ID.

- **GET /api/complaints/pending/university/{universityId}**
  - **Description:** Retrieves a list of pending complaints for the university.
  - **Path Parameters:** universityId
  - **Response:**
    * 200 OK: `{ complaints: [...] }`
    * 404 Not Found: University or complaints not found.
    * 400 Bad Request: Invalid university ID.

- **GET /api/complaints/details/{complaintId}**

  - **Description:** Retrieves the details of a specific complaint.
  - **Path Parameters:** complaintId
  - **Response:**
    * 200 OK: `{ complaintDetails: ... }`
    * 404 Not Found: Complaint not found.
    * 400 Bad Request: Invalid complaint ID.

- **POST /api/notifications/student/complaint/{studentId}/{complaintId}**
  - **Description:** Sends a notification to a student regarding the action taken on their complaint.
  - **Path Parameters:** studentId, complaintId
  - **Response:**
    * 200 OK: `{ success: true, message: "Notification sent to student." }`
    * 404 Not Found: Student or complaint not found.
    * 500 Internal Server Error: Failed to send notification.

- **POST /api/notifications/company/complaint/{companyId}/{complaintId}**
  - **Description:** Sends a notification to a company regarding the action taken on a complaint involving them.
  - **Path Parameters:** companyId, complaintId
  - **Response:**
    * 200 OK: `{ success: true, message: "Notification sent to company." }`
    * 404 Not Found: Company or complaint not found.
    * 500 Internal Server Error: Failed to send notification.

- **POST /api/complaints/resolve/{complaintId}**
  - **Description:** Marks a specific complaint as resolved.
  - **Path Parameters:** complaintId
  - **Response:**
    * 200 OK: `{ success: true, message: "Complaint marked as resolved." }`
    * 404 Not Found: Complaint not found.
    * 400 Bad Request: Invalid complaint ID.

- **POST /api/complaints/flag/{complaintId}**
  - **Description:** Flags a specific complaint for further action or investigation.
  - **Path Parameters:** complaintId
  - **Response:**
    * 200 OK: `{ success: true, message: "Complaint flagged for further action." }`
    * 404 Not Found: Complaint not found.
    * 400 Bad Request: Invalid complaint ID.

- **GET /api/notifications/{userId}**
  - **Description:** Retrieves notifications for a user.
  - **Path Parameters:** `userId`
  - **Response:**
    * 200 OK: `{ notifications: [...] }`
    * 401 Unauthorized: Access denied.

2.6 Selected Architectural Styles and Patterns

2.6.1 Three-Tier Architecture

The Three-Tier architecture divides the system into three logical layers:

- **Presentation Tier (Web Server):** This layer provides the user interface, typically through web pages or web applications. It is implemented using technologies like HTML, CSS, and JavaScript. This tier allows interaction between the user and the system.
- **Business Logic Tier (Application Server):** This layer acts as the intermediary, processing and implementing the business logic. It ensures the rules and workflows of the system are correctly executed. It is often responsible for making decisions and handling API requests.
- **Data Storage Tier (Database Server):** This is the backend layer responsible for managing and storing data. It runs on a Database Management System (DBMS), ensuring data persistence and security.

The Three-Tier architecture offers modularization, scalability, and separation of concerns, which simplifies maintenance and enhances control and security in the system.

2.6.2 Model-View-Controller (MVC) Pattern

The Model-View-Controller (MVC) pattern is adopted for its modular and maintainable software architecture. It is particularly well-suited for web applications:

- **Model:** Manages the data, application logic, and rules of the system. It encapsulates the core functionality and ensures consistency in data handling.
- **View:** Represents the data visually to the users, such as charts, tables, or forms. Multiple views can exist for the same data, catering to different user needs.
- **Controller:** Acts as the intermediary between the Model and the View. It processes user inputs, communicates with the Model to fetch or manipulate data, and updates the View accordingly.

This pattern provides strong decoupling between the components, enhancing reliability, reusability, and clarity of the codebase. The distributed nature of this pattern supports scalable applications by efficiently handling user interfaces and data processing.

2.6.3 RESTful APIs

RESTful APIs (Representational State Transfer) are implemented to ensure seamless communication between the client and server. These APIs:

- Use standard HTTP methods: GET, POST, PUT, DELETE, etc., for data exchange.
- Are stateless, ensuring each request is self-contained and independent of previous ones.

RESTful APIs are simple to implement, lightweight, and reduce server load, making them an ideal choice for scalable systems requiring smooth communication and data exchange.

2.6.4 Cloud Hosted

The S&W is hosted on cloud and used third party cloud service providers for to manage and deliver services and applications. Some advantages of cloud deployment other than the already mentioned are as follows:

- **High Availability:** Cloud providers offer uptime guarantees and data is often replicated across multiple data centers to ensure availability.
- **Security:** Cloud providers often implement strict security policies and invest heavily in infrastructure to protect data.

2.7 Other Design Decisions

2.7.1 Token-Based Authentication and Authorization

The system employs a token-based authentication and authorization mechanism to secure access and maintain user sessions. Key features include:

- **Token Generation:** Upon login, the server generates a unique token, such as a JWT (JSON Web Token), which is shared with the client.
- **Token Usage:** The token must be included in the header of every subsequent request requiring authentication or authorization.
- **Security Features:** Tokens are used for secure operations, such as identifying users, validating their roles, and safely sending sensitive data like email invitations.

This approach enhances security by ensuring only authenticated users can access restricted resources, while the stateless nature of tokens reduces the load on the server.

2.7.2 Relational Database

The system uses relational database for storing and querying of data. In relational database data is stored in a structured format using tables (called relations). This structure allows for easy querying, retrieval, and manipulation of data using a specialized query language, most commonly SQL. There are several upsides of using relational database, some of which are as follows:

- **Structured Data Storage:** Data is stored in a highly organized and logical manner, which makes it easy to access, manage, and update.
- **Scalability:** Relational databases can handle large volumes of data and can scale horizontally or vertically to accommodate more records or users.
- **Flexible Queries:** It provides powerful querying capabilities, allowing retrieval and manipulation of data in complex ways.

3 User Interface Design

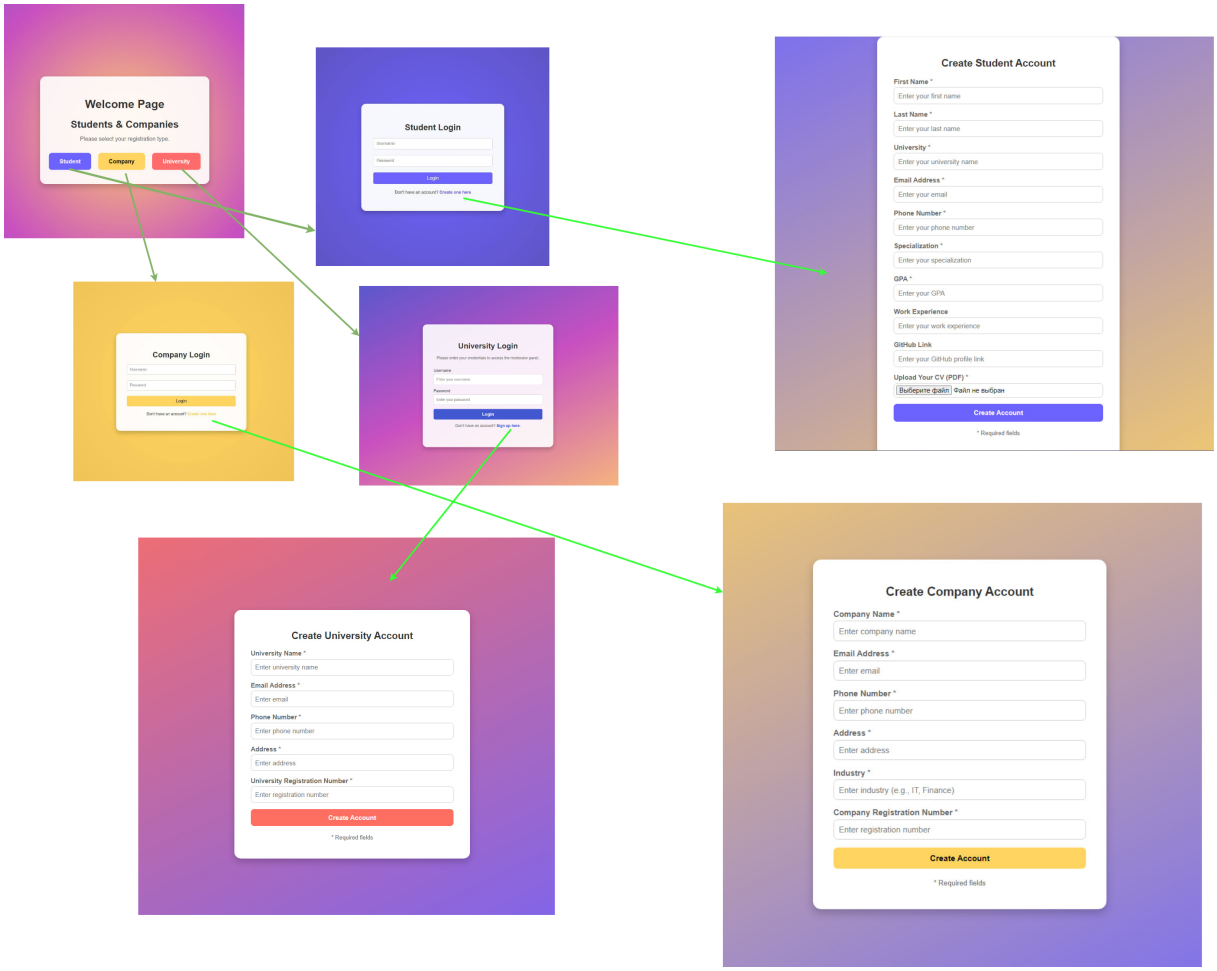

Figure 22: User Journey from starting page and Navigation Flow for Students, Companies, and Universities on the Platform.

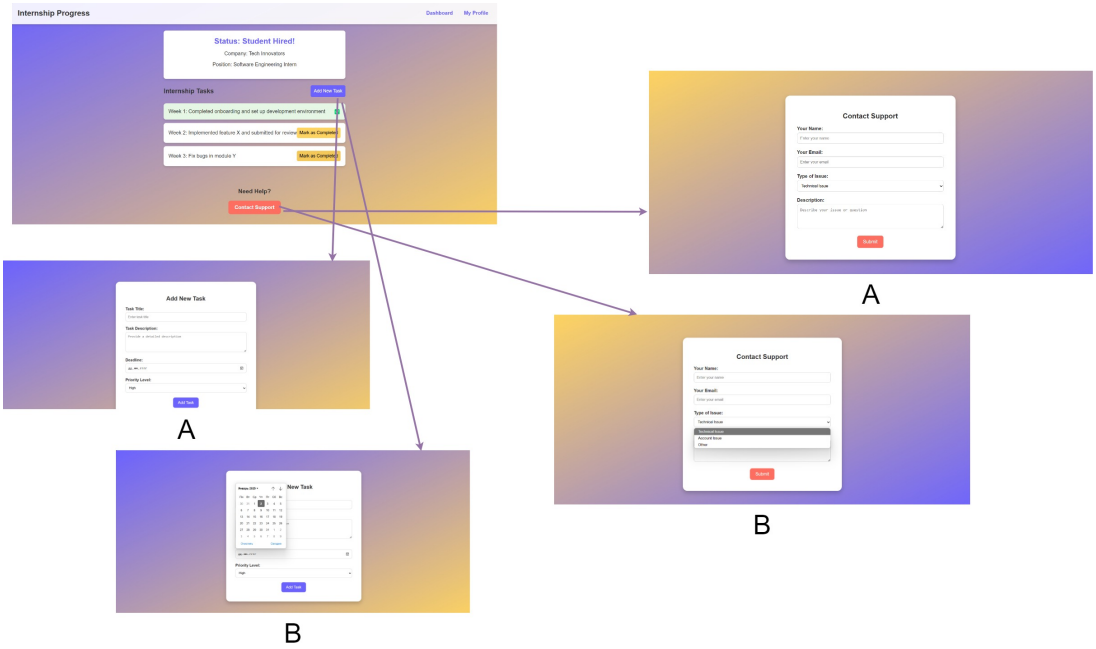

Figure 23: Internship Progress and Task Management Flow: Users can track their internship progress, add new tasks (A), and contact support (B) for assistance with various issues.

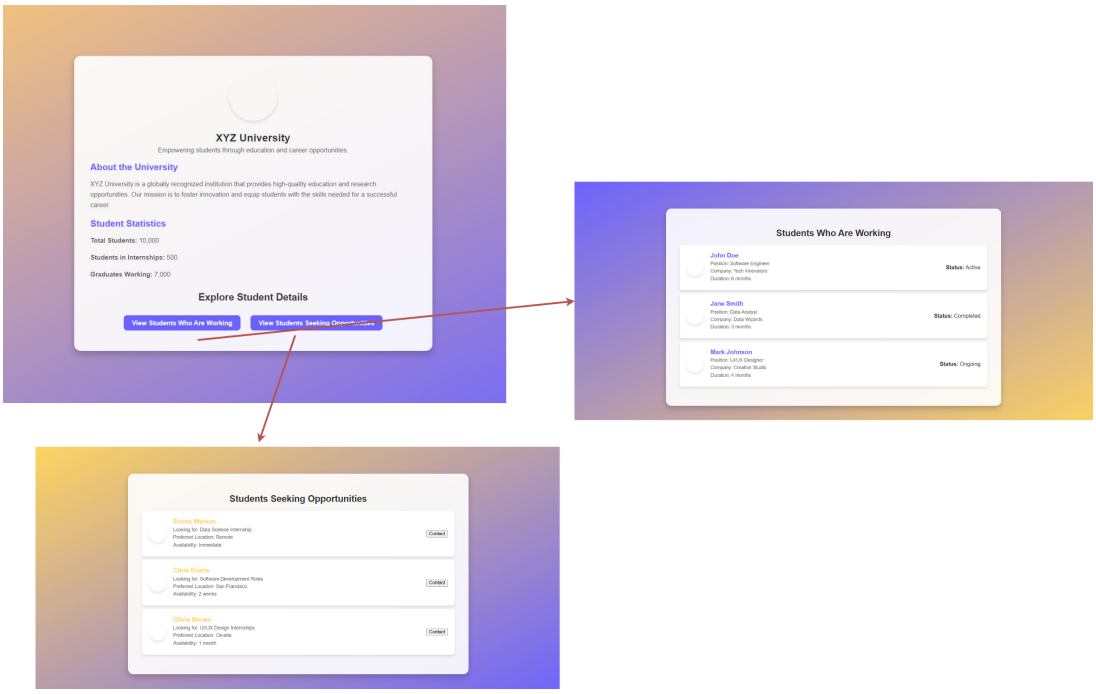

Figure 24: University Dashboard Navigation: The interface allows universities to view statistics and access detailed lists of students who are currently working or seeking opportunities, helping administrators manage student engagement efficiently.

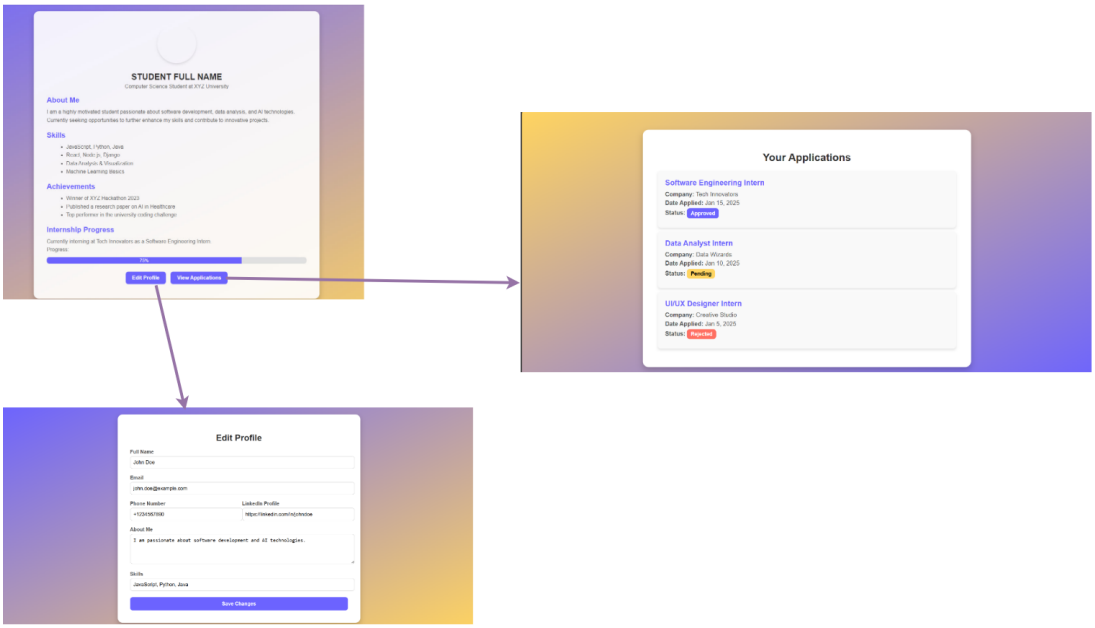

Figure 25: Student Profile and Actions: A comprehensive student profile dashboard displaying skills, achievements, and internship progress. The navigation includes options to edit personal information and view detailed application statuses.

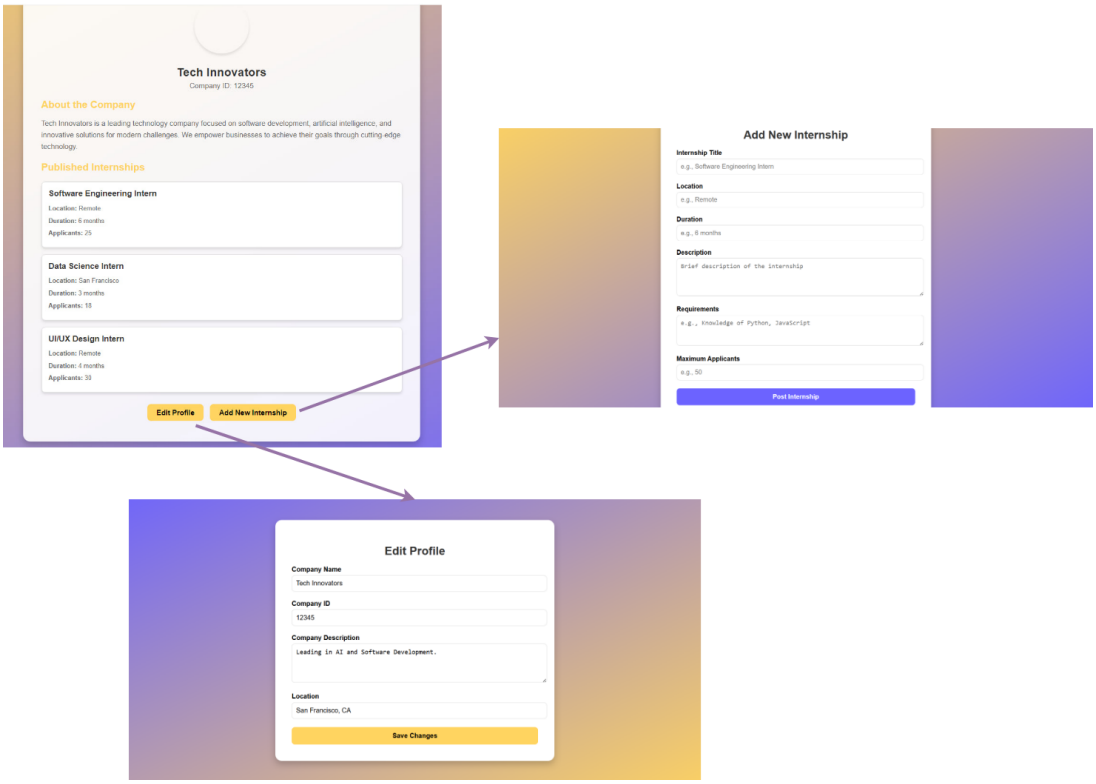

Figure 26: Company Profile and Management: An interface for company representatives to manage their profiles and internship offerings. Includes options to edit company details and add new internships with customizable parameters.

4 Requirement traceability

Requirements	[R1] The S&C system allows new users (students, universities, and companies) to register by giving their credentials (e.g., name, email address, password).
Components	• Database (Stores user information securely) • Profile Manager (Handles user profile creation and updates) • Authenticate Manager (Manages user authentication and credentials) • Notification Manager (Sends confirmation email after registration)

Requirements	[R2] The system allows users (students, universities, and companies) to login.
Components	• Database (Verifies credentials against stored data) • Profile Manager (Manages user session after login) • Authenticate Manager (Validates user credentials and manages sessions) • Notification Manager (Notifies users of successful login or errors)

Requirements	[R3] The system allows students to create and update their profile (experience, skills, CV).
Components	• Database (Stores student profile data such as skills, experience, and CV) • Profile Manager (Handles student profile creation and updates) • Recommendation Engine (Provides feedback to students about profile completeness)

Requirements	[R4] The system allows companies to create and update company's profile (about, field of work, achievements).
Components	• Database (Stores company details and achievements) • Profile Manager (Manages company profile data) • Company Portal (Provides interface for companies to manage their profiles)

Requirements	[R5] The system allows universities to create and update their profile.
Components	• Database (Stores university profile information) • Profile Manager (Manages university data) • University Admin Portal (Provides interface for universities to manage their profiles)

Requirements	[R6] The system helps students in creating their CVs by giving intelligent suggestions for making a stronger CV and pointing out mistakes.
Components	• Database (Stores CV data) • Profile Manager (Integrates feedback with user profiles) • Recommendation Engine (Provides intelligent suggestions based on user input) • Recommendation Engine (Analyzes user data for profile improvement)

Requirements	[R7] Students can search for internship opportunities by clicking on the "Search" button.
Components	• Database (Stores internship data) • Search Engine (Handles internship searches) • Recommendation Engine (Suggests internships based on student profiles) • Notification Manager (Notifies students of available internships)

Requirements	[R8] Students can apply to the desired internship by clicking on the "Apply" button for an internship post.
Components	• Database (Stores submitted applications) • Profile Manager (Tracks application status) • Notification Manager (Notifies companies about new applications)

Requirements	[R9] The system provides personalized recommendations to the students for internships based on students' profiles and skills which match with available internships.
Components	• Database (Stores internship listings and student profiles) • Recommendation Engine (Handles recommendation logic) • Notification Manager (Alerts students about recommended internships)

Requirements	[R10] Students can automatically apply to any recommended internship by clicking on the "Accept" button.
Components	• Database (Stores auto-applied internship data) • Profile Manager (Updates user application status) • Recommendation Engine (For getting the matched internships) • Notification Manager (Alerts users of successful application)

Requirements	[R11] Companies can post internship opportunities and descriptions which include application domain, tasks to be performed, skills required.
Components	• Database (Stores internship listings) • Company Portal (Provides interface for companies to post internships) • Profile Manager (For posting internship description) • Notification Manager (Notifies users of new internship postings)

Requirements	[R12] The system can provide recommendations to improve the internship posts by companies.
Components	• Database (Stores internship descriptions and related data) • Recommendation Engine (Analyzes internship posts and suggests improvements) • **R**ecommendation Engine (Provides suggestions for improvement)

Requirements	[R13] The companies can prepare a questionnaire in the system to be forwarded to the students to get additional information from them when the selection process starts.
Components	• Company Portal (Provides an interface for companies to create questionnaires) • Profile Manager (Handles student and company profile management) • Database (Stores questionnaire data and responses) • Notification Manager (Notifies students about questionnaire assignments)

Requirements	[R14] The system can recommend companies about available students who match the job description and skills by applying statistical analysis based on the characteristics of students and internships.
Components	• Recommendation Engine (Handles matching students with internships based on their profiles) • Profile Manager (Manages student and company profiles) • Database (Stores student, company, and internship data for analysis) • Notification Manager (Notifies companies about matching students)

Requirements	[R15] The system notifies students and companies once every day to recommend new matches.
Components	• Notification Manager (Manages daily notifications about new matches) • Recommendation Engine (Suggests new matches) • Database (Stores match history and user preferences)

Requirements	[R16] Companies can track applications and review candidate CVs.
Components	• Company Portal (Displays applications and CVs for review) • Database (Stores applications and CVs) • Profile Manager (Handles the management of student profiles)

Requirements	[R17] Companies can accept or reject an application or a suggestion by the system by clicking on "Accept" or "Reject" buttons for each application.
Components	• Company Portal (Provides accept/reject functionality for applications) • Selection Process (Manages the acceptance/rejection process) • Database (Stores the updated status of applications) • Notification Manager (Notifies students of acceptance/rejection)

Requirements	[R18] A messaging channel is opened when a recommendation is accepted by both student and company or an application by student is accepted by company.
Components	• Student Portal (Provides interface for messaging students) • Company Portal (Provides interface for messaging companies) • Selection Process (Manages messaging after acceptance) • Database (Stores message history) • Notification Manager (Notifies both students and companies of new messages)

Requirements	[R19] The prepared questionnaire is forwarded to the student.
Components	• Student Portal (Displays questionnaire to students) • Database (Stores student responses to questionnaires) • Notification Manager (Notifies students about new questionnaires)

Requirements	[R20] The system helps in the management of the selection process by scheduling interviews.
Components	• Selection Process (Manages interview scheduling) • University Admin Portal (Manages scheduling of university-administered interviews) • Notification Manager (Notifies both student and company about scheduled interviews) • Database (Stores interview scheduling data)

Requirements	[R21] The system sends interview link or interview location to both student and company.
Components	• Selection Process (Provides interview link and location data) • Database (Stores interview details and links) • Notification Manager (Notifies both student and company) • Email Service (Sends interview details via email)

Requirements	[R22] The system allows the company to update the interview results.
Components	• Company Portal (Provides an interface to update interview results) • Selection Process (Stores and processes interview results) • Database (Stores interview results) • Notification Manager (Notifies students about interview results)

Requirements	[R23] Interview results are updated in the platform and users are notified.
Components	• Selection Process (Handles interview result updating) • Notification Manager (Sends interview result notifications) • Database (Stores updated results)

Requirements	[R24] The system collects feedback from students and companies to then use the gathered data to better its analysis and recommendation algorithm.
Components	• Feedback & Analytics (Processes collected feedback) • Recommendation Engine (Uses feedback for improvements) • Database (Stores feedback data) • Notification Manager (Notifies users about feedback requests) • Feedback & Analytics (For improving the recommendation model)

Requirements	[R25] The system has a dedicated page to keep track and monitor all the ongoing search and selection processes for all three types of users (students, university, companies).
Components	• User Interfaces • Internship Monitor Engine (Monitors the overall status of selection) • Database (Stores real-time data of ongoing processes)

Requirements	[R26] During the selection process and during the interview the involved parties can raise concerns or complaints which will be monitored by the university.
Components	• Complaint Manager (Manages complaint submission and tracking) • University Admin Portal (Monitors and resolves complaints) • Notification Manager (Notifies concerned parties about complaints) • Database (Stores complaint data)

Requirements	[R27] The system allows the university to decide on required actions to perform like warning to respective parties or interruption of internship and updates the same on the platform.
Components	• University Admin Portal (Manages actions and decisions) • Notification Manager (Notifies concerned parties about actions) • Database (Stores decision data and actions)

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

The implementation plan defines the sequential steps for developing the system's components, ensuring that each feature and module is designed, developed, and tested effectively. The Bottom-Up methodology will be used to break down the entire system into simplest possible subsystems that recursively build up to the required solution. Each component will follow the Single Responsibility Principle (SRP) and will be implemented using Test-Driven Development (TDD).

5.1.1 Development of Core Components

User Authentication and Profile Management: The objective is to implement secure registration and login functionalities for students, companies, and universities, with each user type having their own profile and corresponding data. To achieve this, we will develop the user registration functionality, allowing for input of name, email, password, and role selection. We will also implement login and session management to ensure secure access. Each user type will be able to create and edit their profile, with students entering their academic and experience details, while companies provide internship descriptions. For this, we will use OAuth 2.0 for secure authentication and PostgreSQL/MySQL for storing user data.

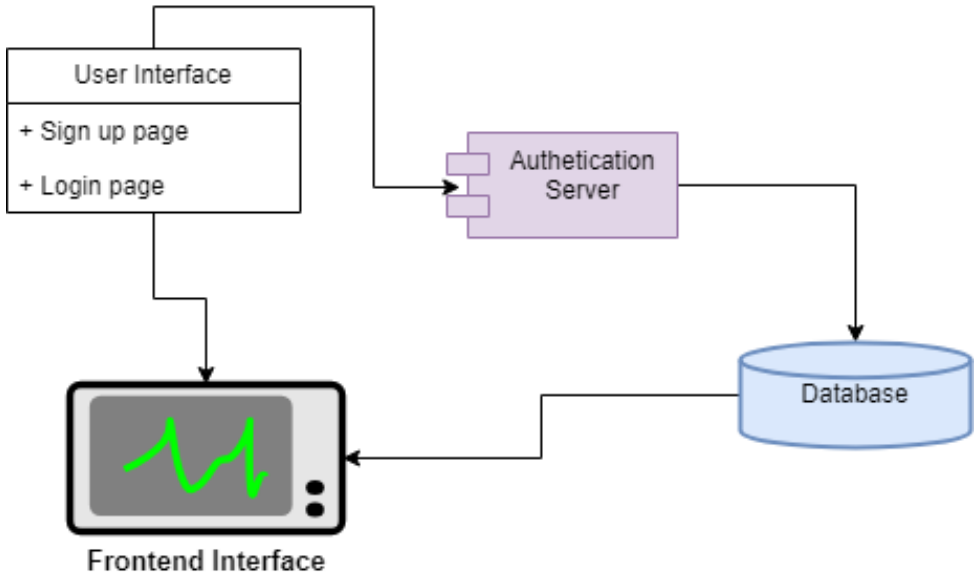

Figure 27: User Authentication and Profile Management.

Internship Search and Application System: The objective of this component is to provide students with a way to search for, view, and apply for internships. This will involve developing a search functionality that uses filters like location, skills, and internship type. The system will display relevant internship details such as job descriptions, company names, and locations. Students will also have the option to apply directly to internships by clicking the "Apply" button. Elasticsearch will be used to index and search internship listings, and we will integrate this functionality with the backend API to display search results in real-time.

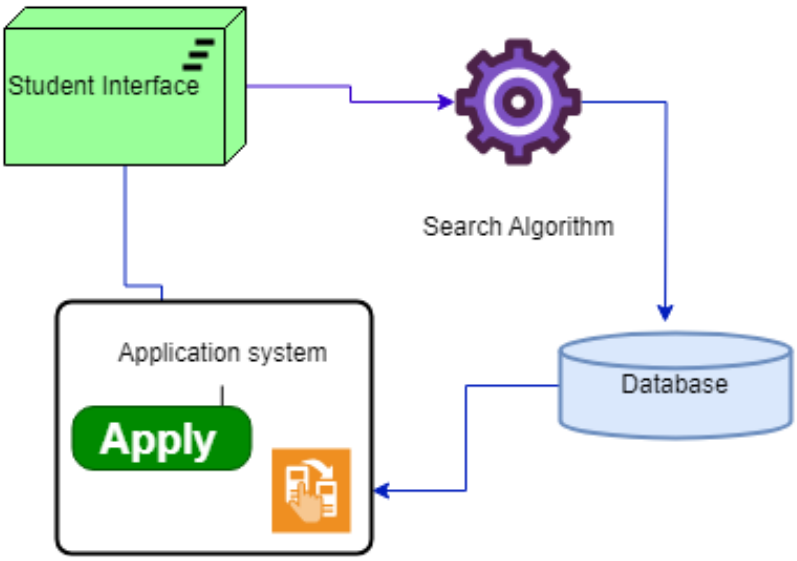

Figure 28: Internship Search and Application System.

Internship Posting: Companies will be able to post internships, and the system will automatically match students with relevant positions based on their profiles. To implement this, we will create the "Post Internship" functionality, which will allow companies to add internship roles. The frontend for posting the internship details will show writing improvement suggestions to the author to increase the likelihood of finding suitable matches with Student profiles. The backend service will be developed using Node.js. The system will use the Recommendation Engine described below to recommend internships to students, based on their profile data, such as skills, experience, and location preferences.

Recommendation/Matching Engine: The Recommendation/Matching Engine will run as a Cron Job at regular intervals of 24 hours for each Student to identify Internships relevant to their skills and background. In order to achieve performance optimization, an Embedding will be calculated ahead-of-time for each Student profile using a Neural Network trained on anonymized data of past Student profile and Internship matches. The Neural Network will use Matrix Factorization to learn the representation of Latent Factors from the available profile and internship attributes and create the Embedding. This Embedding will be re-calculated whenever a Student updates any attributes of their profile. The Recommendation/Matching Engine will use a Model-Based Collaborative Filtering (MCBF) to match new Student profiles to Internships based on the pre-calculated Embeddings to optimize compute resource utilization. The system will use the Notification Manager to alert both students and companies about new matches. The backend logic will be developed using Python, and machine learning algorithms will be applied for the matching process.

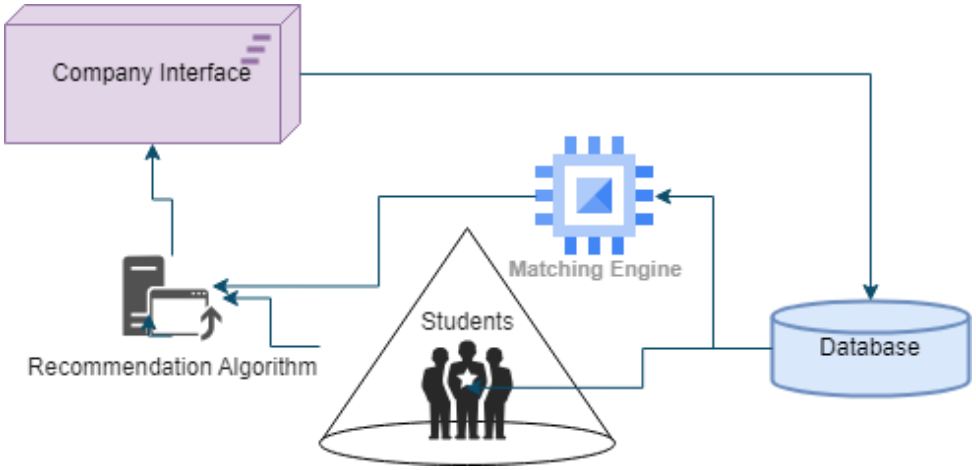

Figure 29: Internship Posting and Matching Engine.

Interview Scheduling and Feedback Collection: The system will support companies in scheduling interviews with selected candidates and collecting feedback after the interview. The interview scheduling tool will allow students and companies to select available time slots. Integration with a video conferencing system like Zoom will allow for virtual interviews, and the system will also collect feedback after each interview. For scheduling, we will use the Google Calendar API, and Zoom will be integrated for video calls. The feedback system will be custom-built to collect and store feedback from both students and companies.

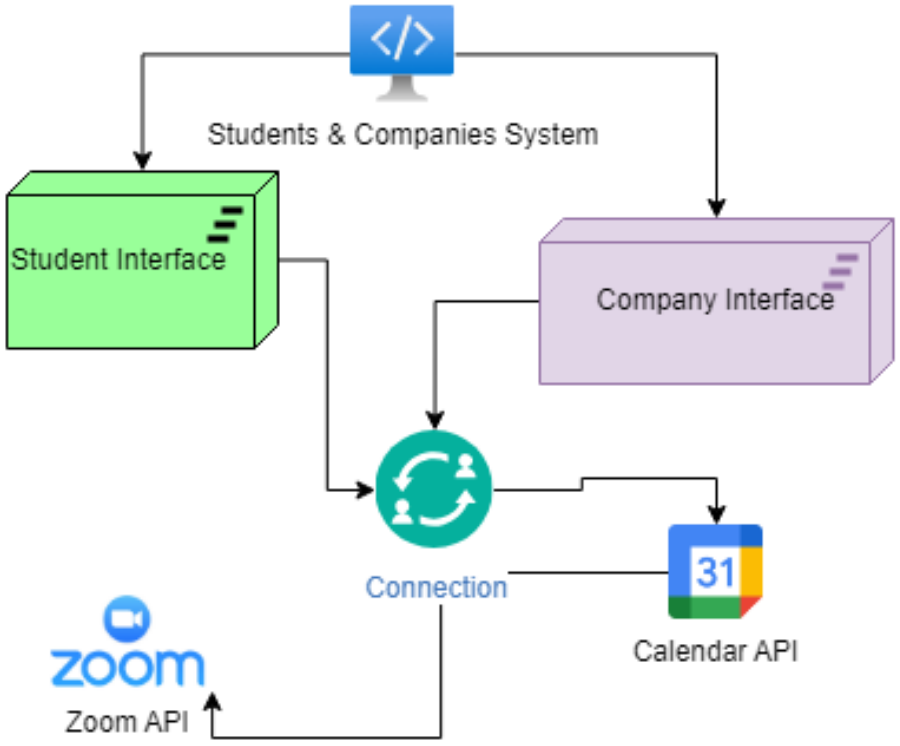

Figure 30: Interview Scheduling.

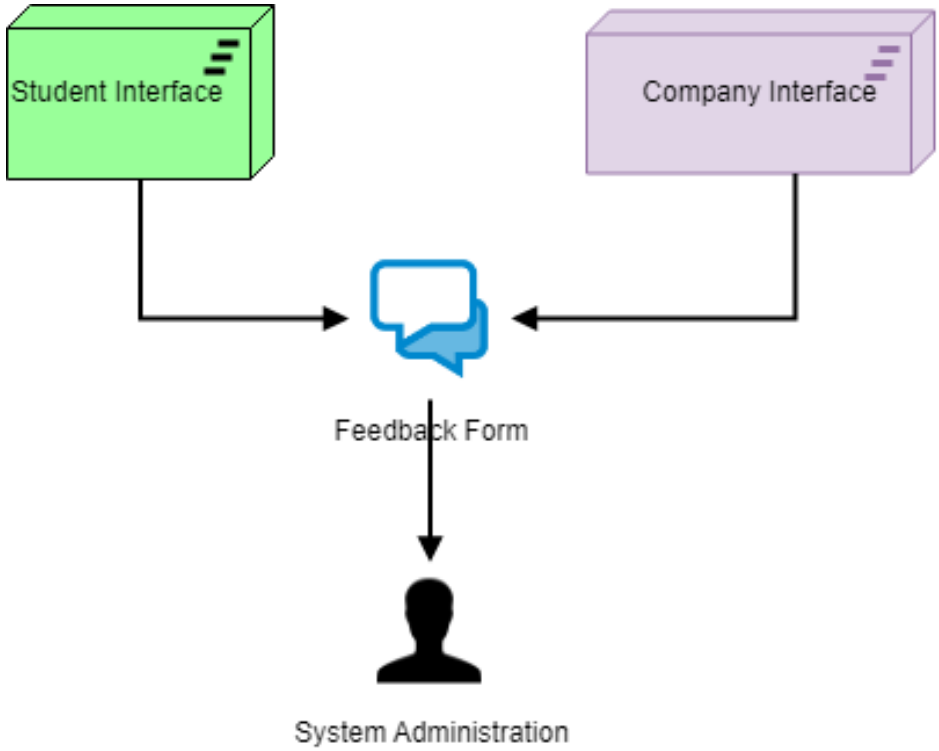

Figure 31: Feedback Collection.

Complaint Handling and University Monitoring System: This component will enable universities to monitor student internship progress and handle complaints from both students and companies. Universities will be able to track the status of all students' internship applications using a dedicated dashboard. A complaint reporting and resolution workflow will be developed to allow students and companies to raise and resolve issues. Notifications will be sent to the involved parties (students, companies, universities), and the university will oversee the resolution process. The notification system will be built using email or push notifications, and the interface for universities will be user-friendly and easily accessible.

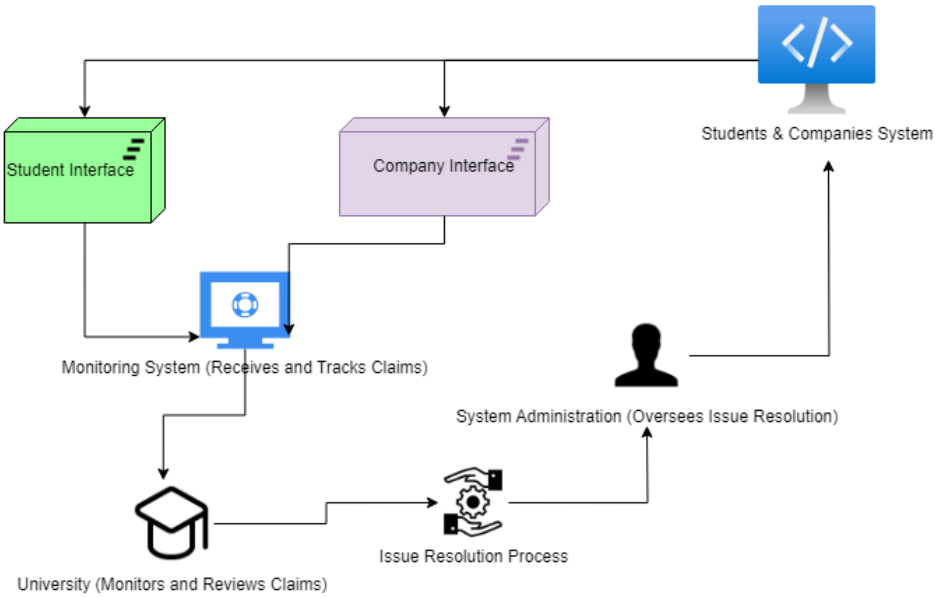

Figure 32: Complaint Handling and University Monitoring System.

5.1.2 Technology Stack

- **Frontend:** ReactJS or Angular will be used to build responsive and interactive user interfaces.
- **Backend:** Node.js with Express.js or Django will handle business logic and REST API services.
- **Database:** PostgreSQL or MySQL will store structured data such as user profiles, internship listings, and applications.
- **Search Engine:** Elasticsearch will be used for indexing and filtering internship listings efficiently.
- **Recommendation System:** Python or JavaScript-based machine learning models will be used for matching students with internships.
- **Video Conferencing:** Zoom API will be integrated for virtual interviews.
- **Notification System:** Email services like NodeMailer or SendGrid will be used for sending user notifications.

5.2 Integration Plan

Once the core components are developed, we will proceed with the integration phase. This phase involves combining the various subsystems, ensuring smooth data flow, and confirming that all functionalities work together.

5.2.1 Integration of User Authentication and Profile Management

The user authentication system will be integrated with the frontend, allowing users to sign up, log in, and access their profile pages. Role-based views (Student, Company, University) will be displayed based on the user's role.

5.2.2 Integration of Internship Search and Application System

The search functionality will be connected to the backend to fetch real-time data, and the "Apply" functionality will be integrated to submit student applications to companies.

5.2.3 Integration of Internship Posting and Matching Engine

Companies will be able to post internships, and the system will match these postings with student profiles. The matching engine will use the student's profile data to recommend internships, and notifications will inform both students and companies about new matches.

5.2.4 Integration of Interview Scheduling and Feedback Collection

The interview scheduling feature will be integrated, allowing both students and companies to select available time slots. Zoom integration will ensure video meetings, and feedback will be collected and stored after interviews.

5.2.5 Integration of Complaint Handling and University Monitoring

Universities will be able to monitor the internship status of students and resolve complaints through the system. The complaint submission and resolution process will be integrated, and notifications will inform all relevant parties.

5.2.6 End-to-End Testing

We will ensure that data flows seamlessly between the frontend, backend, and database. Additionally, we will test for any UI/UX issues and resolve any discrepancies. Unit tests for individual components and integration tests for interactions between modules will be conducted.

5.3 Testing Plan

Testing will be done iteratively during development, followed by extensive system testing to ensure robustness and correctness.

5.3.1 Unit Testing

The goal is to verify that individual components, such as user authentication, profile creation, internship search, matching algorithm, and interview scheduling, work correctly.

5.3.2 Integration Testing

We will verify that the components work together seamlessly. For example, we will check if user registration leads to successful login and profile creation, if students can apply for internships, and if the matching system works as intended.

5.3.3 Functional Testing

This type of testing will ensure that the system meets all functional requirements. Students should be able to search for internships, companies should be able to post internships, and universities should be able to monitor student progress and handle complaints.

5.3.4 Performance Testing

We will conduct load testing to simulate multiple users accessing the system simultaneously. We will also test the system's response time and performance under peak usage conditions.

5.3.5 Security Testing

The security of the platform will be tested to ensure that user credentials are encrypted, and the system is protected against common security threats such as SQL injection and XSS attacks.

5.3.6 Technology Stack

- **Testing Tools:** Jest or Mocha for unit testing the individual components of the system.
- **Integration Testing Tools:** Postman or Supertest for testing REST API integrations between frontend and backend.
- **Performance Testing:** Apache JMeter or LoadRunner for conducting load and performance tests.
- **Security Testing:** OWASP ZAP or Burp Suite for vulnerability scanning and security testing.

5.3.7 User Acceptance Testing (UAT)

Actual users (students, companies, universities) will test the system to ensure it meets their needs. Feedback will be collected, and any necessary adjustments will be made before deployment.

6 Effort Spent

This section shows the amount of time that each member has spent to produce the document. Please notice that each unit is the result of coordinated work among all the members.

UNIT	MEMBERS	HOURS
Section 1	S. Chaudhury	3h
Section 2.1	S. Chaudhury	2h
Section 2.2	D. Motiallah	2h
Section 2.3	S. Chaudhury	3h
Section 2.4 (Sequence diagrams)	D. Motiallah & S. Chaudhury	16h
Section 2.4 (Sequence diagrams descriptions)	S. Chaudhury	5 h
Section 2.5	D. Motiallah	6h
Section 2.6 & 2.7	D. Motiallah & S. Chaudhury	4h
Section 3	D.Islam	12h
Section 4	D.Islam	7h
Section 5	D.Islam, S. Chaudhury	13h
Redaction	S. Chaudhury	5h

7 References and Tools

1. GitHub: https://www.github.com
2. GitHub Actions: https://github.com/features/actions
3. The UI have been made with:
    - JavaScript documentation:https://developer.mozilla.org/en-US/docs/Web/JavaScript
    - Node.js documentation: https://nodejs.org/api/all.html
    - HTML documentation: https://html.spec.whatwg.org/
    - Bootstrap CSS framework: https://getbootstrap.com/docs/5.3/getting-started/introduction/
4. Zoom API: https://developers.zoom.us/docs/api/
5. Calendar API: https://developers.google.com/calendar/api/guides/overview?hl=ru
6. Diagrams have been made with draw.io: https://app.diagrams.net/