POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Students & Companies

DD Document
Software Engineering 2 Project

Authors: **Paone A. Pasqual M. Restelli M.**

Advisor: Prof. Camilli Matteo
Academic Year: 2024-25

Contents

1 Introduction

1.1. Purpose

This document represents the Design Document (DD). The purpose of the DD is to provide overall guidance to the architecture of the software product discussed in the RASD.

It contains the description of the architecture chosen to design the Students&Companies web application. Furthermore, it describes how the UI will look like, and it maps the requirements defined in the RASD to the architectural components defined in this document. It also includes a set of design characteristics required for the implementation by introducing constraints and quality attributes, and it gives a presentation of the implementation, integration, and testing plan.

The DD is addressed to the software development team who will have to implement the described system, and it has the purpose of guiding them through the development process.

1.2. Scope

Students & Companies (S&C) is a platform linking students with internship opportunities and corporations. Companies can advertise roles, get candidate recommendations, and send invitations. Students can search for roles actively or receive notifications for matches. Mutual interest brings to the selection process, composed of interviews, and possibly to an internship being carried out. During internships, S&C gathers feedback, manages complaints, and involves universities when needed to resolve issues.

The platform is implemented using the 3-tier design pattern, a client-server software pattern that separates applications into three logical layers, each with distinct responsibilities. More implementation choices are covered in later sections of this document.

1.3. Definitions, Acronymous, Abbreviations

Abbreviation	Description
RASD	Requirements Analysis & Specification Document
DD	Design Document
G*	Goal
D*	Domain assumption
R*	Functional requirement
S&C	Students & Companies
STs	Students
COMs	Companies
UNs	Universities
UML	Unified Modelling Language
UI	User Interface

Table 1.1: List of Definitions, Acronymous, and Abbreviations

1.4. Revision History

- Version 1.0 (07/01/2025)

1.5. Reference Documents

The document is based on the following materials:

- IEEE Standard Documentation For DD
- The specification of the RASD and DD assignment of the Software Engineering II course a.a. 2024/25
- Slides of the course on WeBeep

1.6. Document structure

This document is composed of six chapters:

1. **Introduction:** This chapter describes the scope and purpose of this DD. It also includes the structure of the document and a set of definitions, acronyms, and abbreviations used.
2. **Architectural Design:** This chapter presents the architectural design choices. It includes an overview of the designed architecture, all the components, the interfaces, and the technologies used for designing the application. It also details the main functions of the interfaces and the processes in which they are utilized (Runtime view and component interfaces). Finally, it explains the design patterns chosen and recommended for system development.
3. **User Interface Design:** This chapter illustrates how the UI should look. It complements and expands upon the User Interfaces subsection in the RASD.
4. **Requirements Traceability.** This chapter explains how the requirements defined in the RASD are mapped to the design elements defined in this document.
5. **Implementation, Integration, and Test Plan.** This chapter outlines the planned order for implementing the system's subcomponents, integrating them, and testing the integration.
6. **Effort Spent.** This chapter provides information on the number of hours each group member has worked on this document.

2 | Architectural Design

2.1. Overview

We chose to design the Students&Companies (S&C) web application using a three-tier architecture with remote presentation. These three tiers correspond to a client tier, application tier, and data tier. The client (presentation) tier serves the needs of students, companies, and universities using the S&C platform. The application layer is the one that processes user requests, performs statistical analysis, and interacts with the internal data tier. The data tier is responsible for data management and storage. It handles all database operations, including data retrieval, updates, and management. To better improve and ensure load distribution and scalability, it was chosen to implement a load balancer between the client and application layer, in particular inside the web server.

A 3-tier architecture is suitable for a career service application as it makes the system easier to manage and scale. It ensures reliability through load balancing, which helps handle high traffic during peak usage. This architecture supports advanced features like job matching and analytics, integrates well with cloud services, and provides flexibility for future updates, such as adding mobile apps or AI tools. Its structure also improves security by isolating sensitive data in the database layer. The S&C architecture specializes in three different groups of components each with its own characteristics and responsibilities:

1. **Presentation layer:** The Presentation (client) Tier is the application's front end, managing the user interface and interactions for data input and display. Its goal is to display data, collect inputs,and provide a seamless user experience.
2. **Application layer:** It handles tasks like user authentication, recommendation matching algorithms, managing user accounts, handling notifications, and communication between the user interface and the database.
3. **Data layer**: Stores and manages critical data such as user profiles, student CVs, project descriptions, and application history. It ensures data security and efficient retrieval for real-time use in the application.

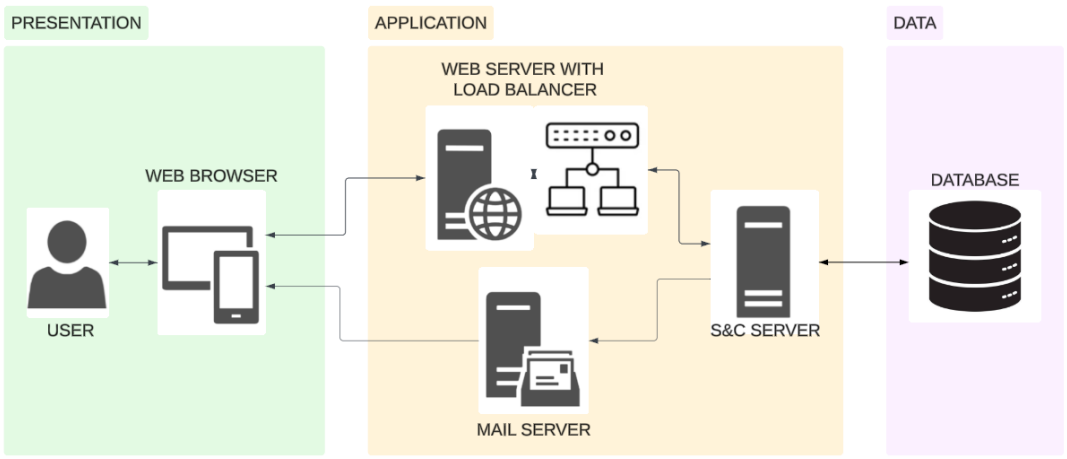

Figure 2.1: High level architecture

2.2. Component view: Component diagram

Figure 2.2 presents a detailed diagram illustrating the components that constitute our application. Below is an explanation of each component and its purpose.

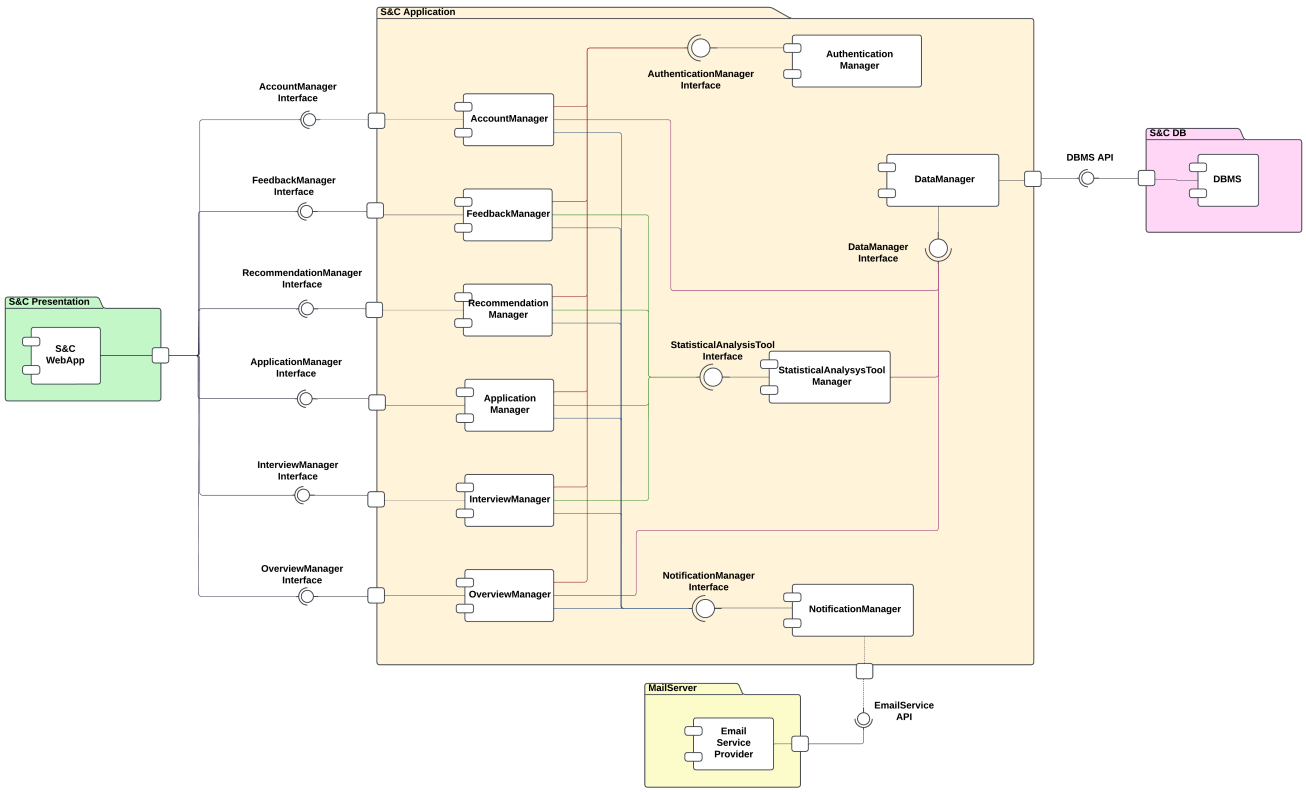

Figure 2.2: Component Diagram

- **WebApp:** The WebApp serves as the primary interface for all users: students, companies, and universities, allowing them to interact with the application via their web browser. User requests are processed and sent to the appropriate backend components through the defined interfaces.
- **Account Manager:** The Account Manager is a backend module responsible for managing all aspects of account registration, login, and user data. It handles the creation of different user profiles and ensures the validity of the submitted data. The Account Manager communicates with the Notification Manager to send notifications to users as needed. To save the new user's data, the Account Manager interacts with the Data Manager to store the information in the database. However, before performing any database operations, the Account Manager first requests permission from the Authentication Manager to ensure that the user is authenticated and authorized to perform the action.
- **Feedback Manager:** The Feedback Manager is responsible for gathering user feedback from both students and companies. It proactively requests feedback when needed through the Notification Manager, ensuring user engagement. This module collaborates with the Statistical Analysis Tool by providing updated feedback data, enabling continuous improvement of the recommendation system and analysis capabilities.
- **Recommendation Manager:** The Recommendation Manager is the module that is responsible for requesting and receiving internship matches between students and projects from the Statistical Analysis Tool. After obtaining the matches, it communicates the results to the Notification Manager, which then notifies the involved users about the recommended opportunities.
- **Application Manager:** The Application Manager handles student applications in three scenarios: when a student proactively applies for an internship they are interested in when a student accepts a recommendation, and when he responds to an invitation from a company that has previously accepted the system's recommendation. After processing the applications, it communicates with the Notification Manager to notify both students and companies about the submitted applications. Additionally, it interacts with the Statistical Analysis Tool to provide data for further analysis and improvement of the recommendation system.

- **Interview Manager:** The Interview Manager oversees the entire interview and final selection process. It communicates with the Notification Manager to keep both students and companies informed about new developments throughout the process. Additionally, it interacts with the Statistical Analysis Tool to provide data, such as information on who was selected for the job, helping to refine the system and adjust parameters for future job recommendations.
- **Overview Manager:** The Overview Manager is responsible for monitoring ongoing internships. It communicates with the Notification Manager to keep universities, students, and companies involved in the project updated on progress. Additionally, it interacts with the Authentication Manager to obtain permission for accessing data and then works with the Data Manager to insert and register the relevant updates.
- **StatisticalAnalysisTool Manager:** The Statistical Analysis Tool Manager is responsible for generating matches between students and internship projects based on student CVs and project descriptions. It also provides suggestions to students for improving their CVs and to companies for enhancing their project descriptions. These tasks are performed using a recommendation system powered by AI and machine learning. The tool is continuously refined through feedback from both students and companies about its previous recommendations. It communicates with the Data Manager to access the necessary data, with prior authentication provided by the Authentication Server.
- **Notification Manager:** The Notification Manager is responsible for sending notifications to all users across various channels. It interacts with the email service API to send email notifications and also delivers real-time updates directly within the web app. When delivering notifications through the app, it follows the chain of requests, processing them in reverse order to ensure that all relevant parties receive the appropriate updates.
- **Authentication Manager:** The Authentication Manager is responsible for managing access rights to data, ensuring that users can perform read or write operations based on their role and permissions.
- **Data Manager:** The Data Manager is responsible for managing and storing all data within the system. It handles data retrieval, updates, and ensures that information is accurately recorded and stored. The module interacts with other components to provide the necessary data for operations while maintaining data integrity and security.

- **Email Service Provider:** The Email Service Provider is an external module that interacts with the Notification Manager to send email notifications to users. It handles the delivery of emails for various events, such as application updates, interview schedules, and system alerts.
- **DBMS:** The DBMS (Database Management System) is responsible for managing and organizing the system's database. It handles data storage, retrieval, and manipulation, ensuring data integrity, security, and efficient access. The DBMS interacts with other modules to support operations such as user authentication, data analysis, and application management, storing and managing all relevant information within the system.

2.3. Deployment view

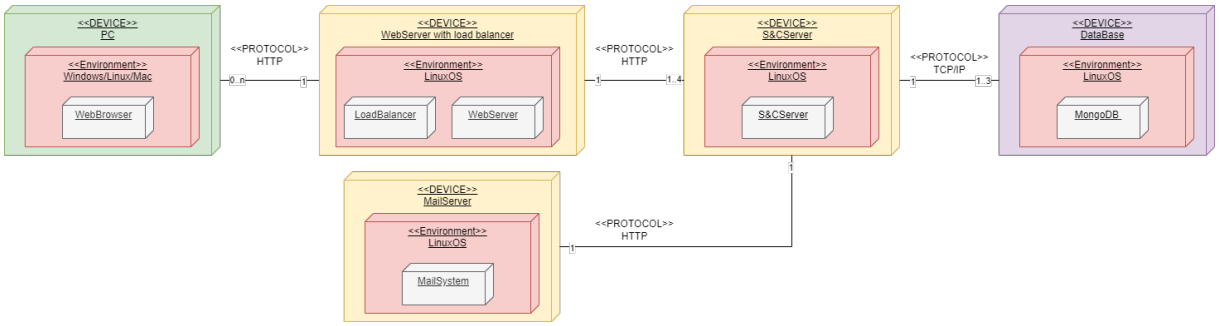

Figure 2.3: Deployment view

The deployment view illustrates how the Students & Companies system components are distributed across various nodes. Users interact with the platform through a Web Browser running on a client device (Windows, Linux, or MacOS), which communicates with the Web Server over HTTPS. The Web Server includes a Load Balancer to efficiently distribute requests among multiple backend services, ensuring scalability and reliable performance under varying traffic loads.

The Application Server, deployed on a Linux environment, handles core application logic, hosting the managers responsible for user accounts, recommendations, feedback, and other functionalities. The Mail Server, running on a separate Linux node, manages email notifications, ensuring communication between the platform and its users. Data storage and retrieval are managed by the Database Server, powered by MongoDB on a Linux environment, which interacts with the application layer over TCP/IP. This setup ensures a clear separation of responsibilities, allowing for modular development and easy scalability.

2.4. Runtime view

All the sequence diagrams described in the Requirements Analysis and Specification Document (RASD) will be elaborated in greater detail. In this expanded version, we will identify and specify the individual components of the application that are utilized to fulfill each specific functionality.

Here below we will include a visual example (sequence diagrams) of how each application component contributes to the implementation of the corresponding functionality. By doing so, we hope to provide a clearer understanding of the interaction between various components and their roles in achieving the described features and requirements.

Note on the sequence diagram: we did not model calls to the DBMS, since every time the DataManager is called it handles them all in the same way with the methods we will show in the next chapter.

UC1: Student creates an account

This sequence diagram outlines the student registration process in the web app. It covers data submission, duplicate account checks, email verification, and optional CV upload. Notifications are handled via email, and validated accounts are saved in the database. Once the CV has been uploaded some calls are made to the statisticalAnalysisTool, but their return, the improvement suggestions and the matches, is not modeled since it will be handled in other use cases.

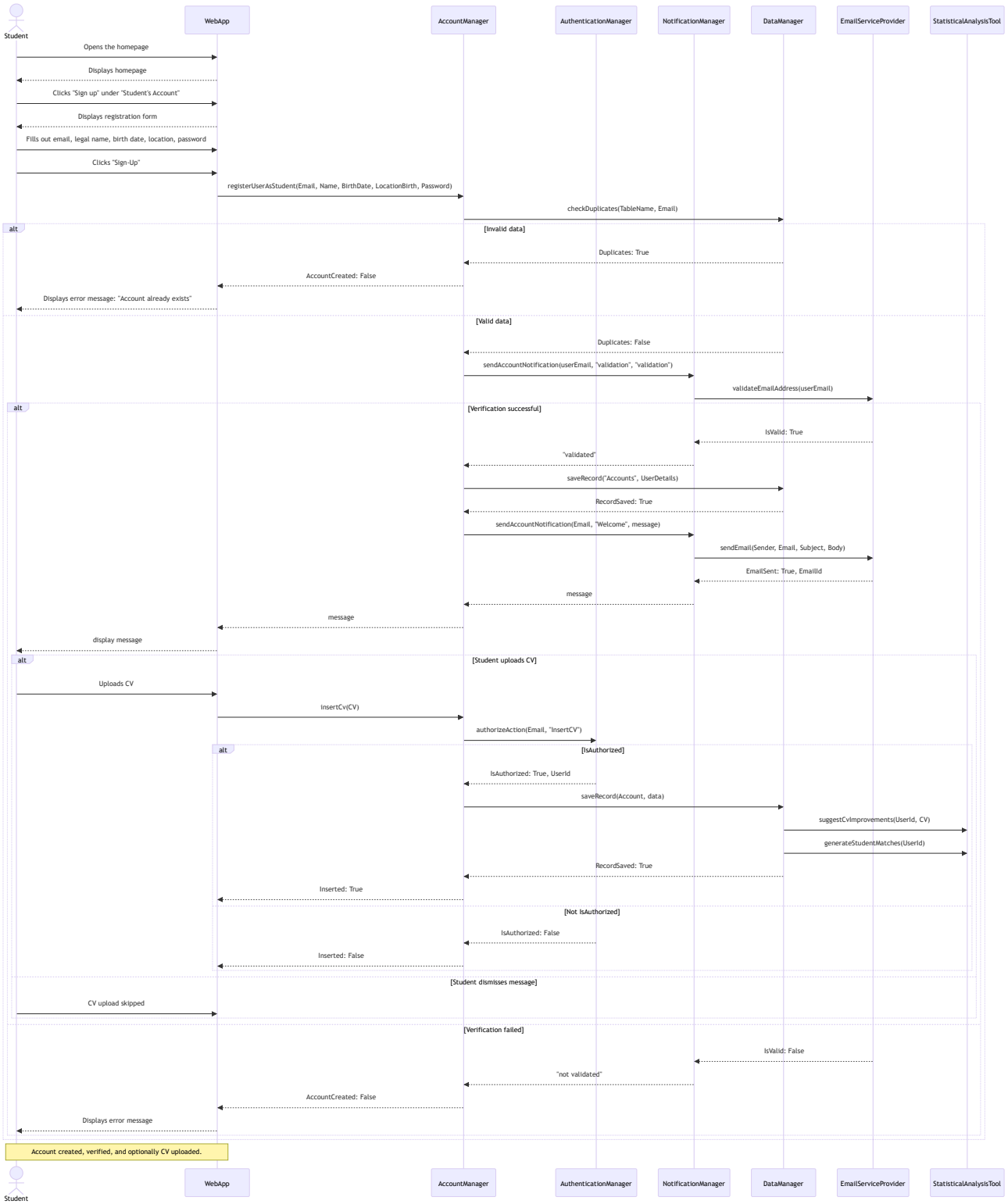

Figure 2.4: UC1

UC2: Company creates an account

This sequence diagram outlines the company registration process in the web app. It includes data submission, duplicate account checks, email verification, and project description upload. Notifications are managed via email, and validated accounts are stored in the database. Once the project description has been uploaded some calls are made to the statisticalAnalysisTool, but their return, the improvement suggestions and the matches, is not modeled since it will be handled in other use cases.

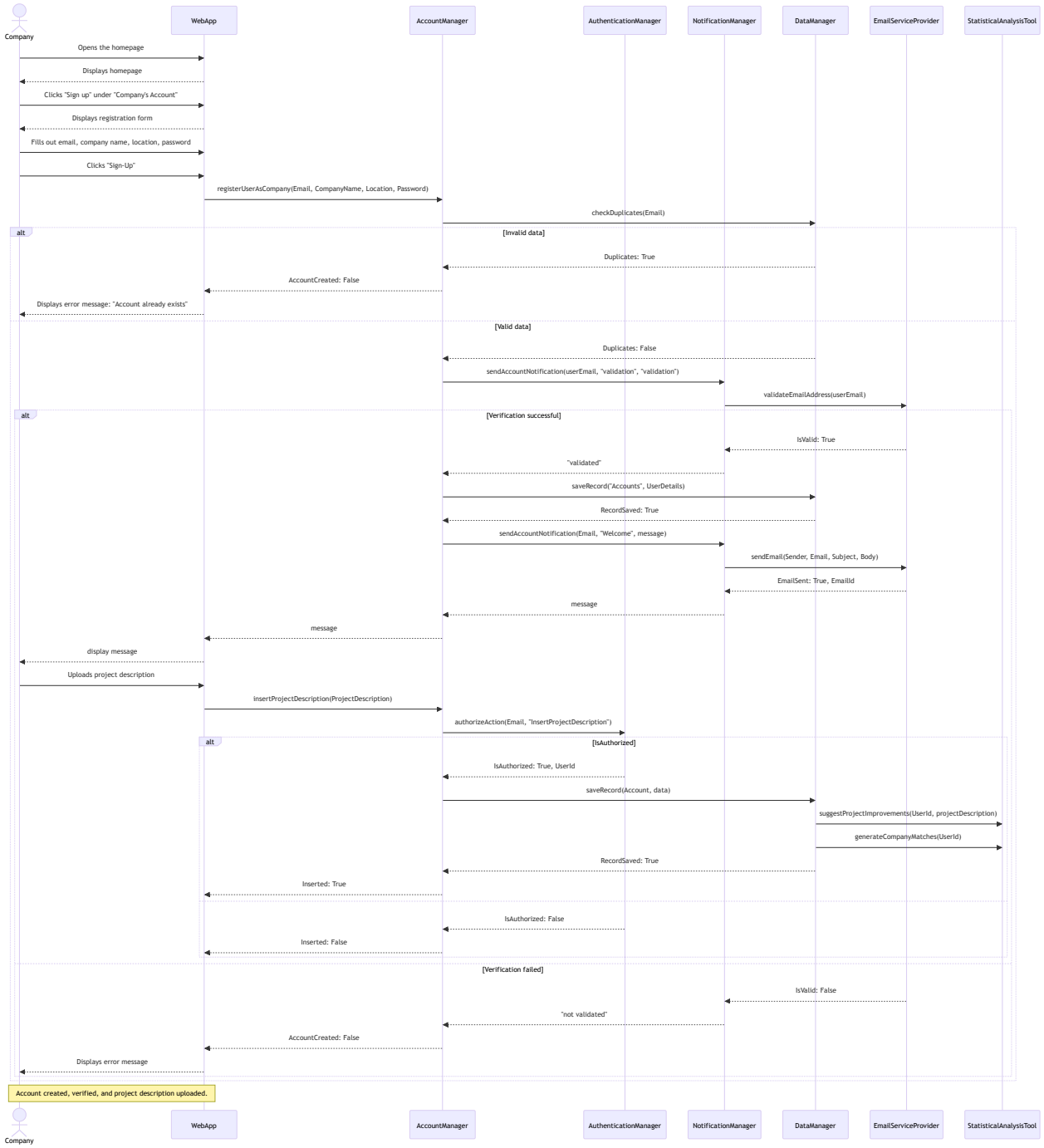

Figure 2.5: UC2

UC3: Student searches and applies for an internship

This sequence diagram describes the internship application process for students in the web app. It includes navigation to the global page, viewing internship details, and applying if interested. Successful applications trigger data updates in the system and notifications to companies via email. If the student is not interested, they return to the global page.

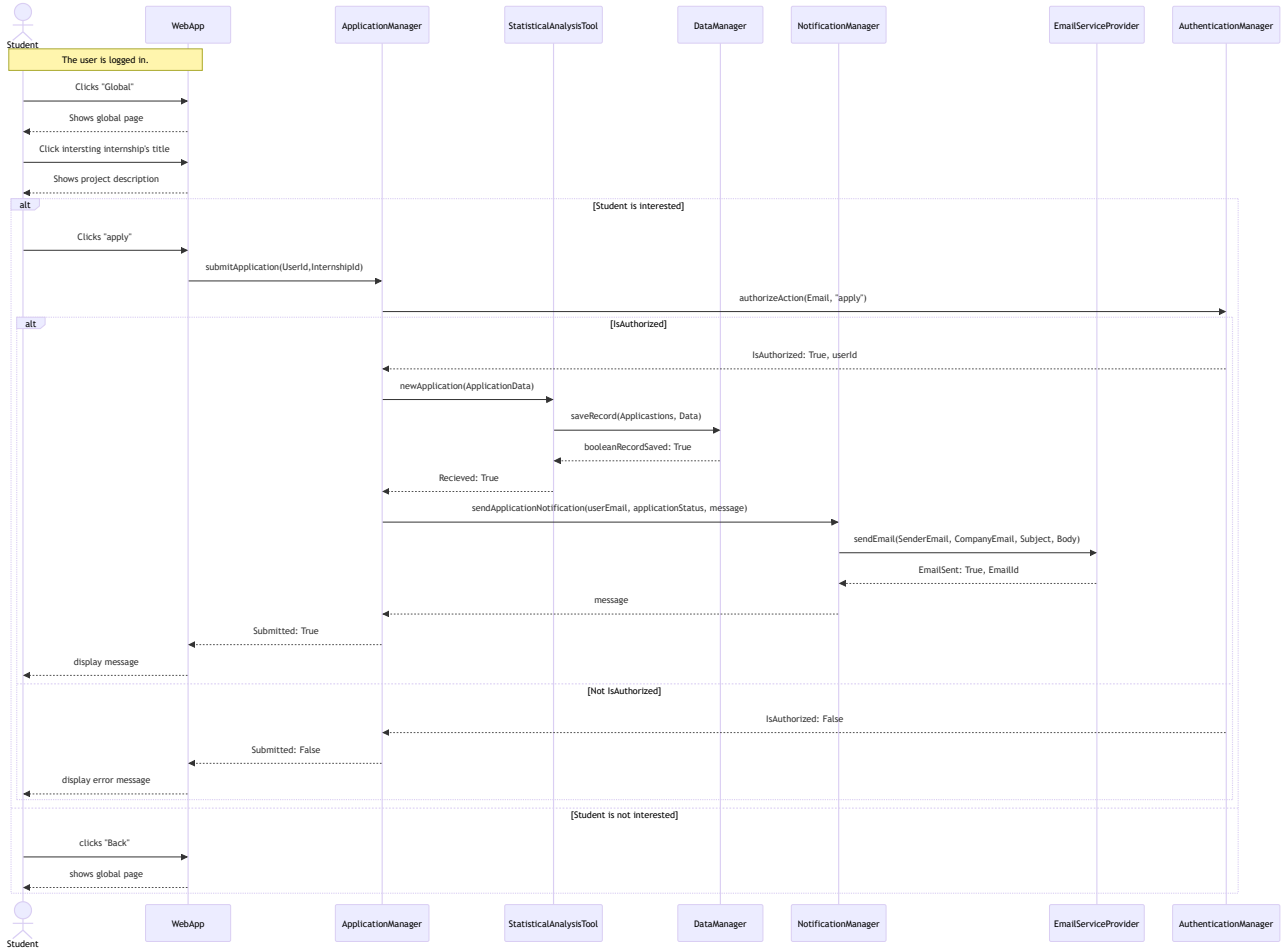

Figure 2.6: UC3

UC4: Student recieves an internship recommendation

This sequence diagram illustrates the recommendation handling process for students in a web app. Recommendations are generated by the StatisticalAnalysisTool, notified to the student via email, and displayed in the app. The display of this notification is modeled as a message call to the WebApp, this is due to the fact that we chose to model all calls to the WebApp that show a notification using the word: message (a convention also applied in future diagrams); the arrow is not dotted since here this message call is not directly a return of another one, but it starts from the RecommendationManager. Students can either accept the recommendation by applying for the internship, which updates the system and notifies the company, or reject it, which removes the match and informs the company of the rejection.

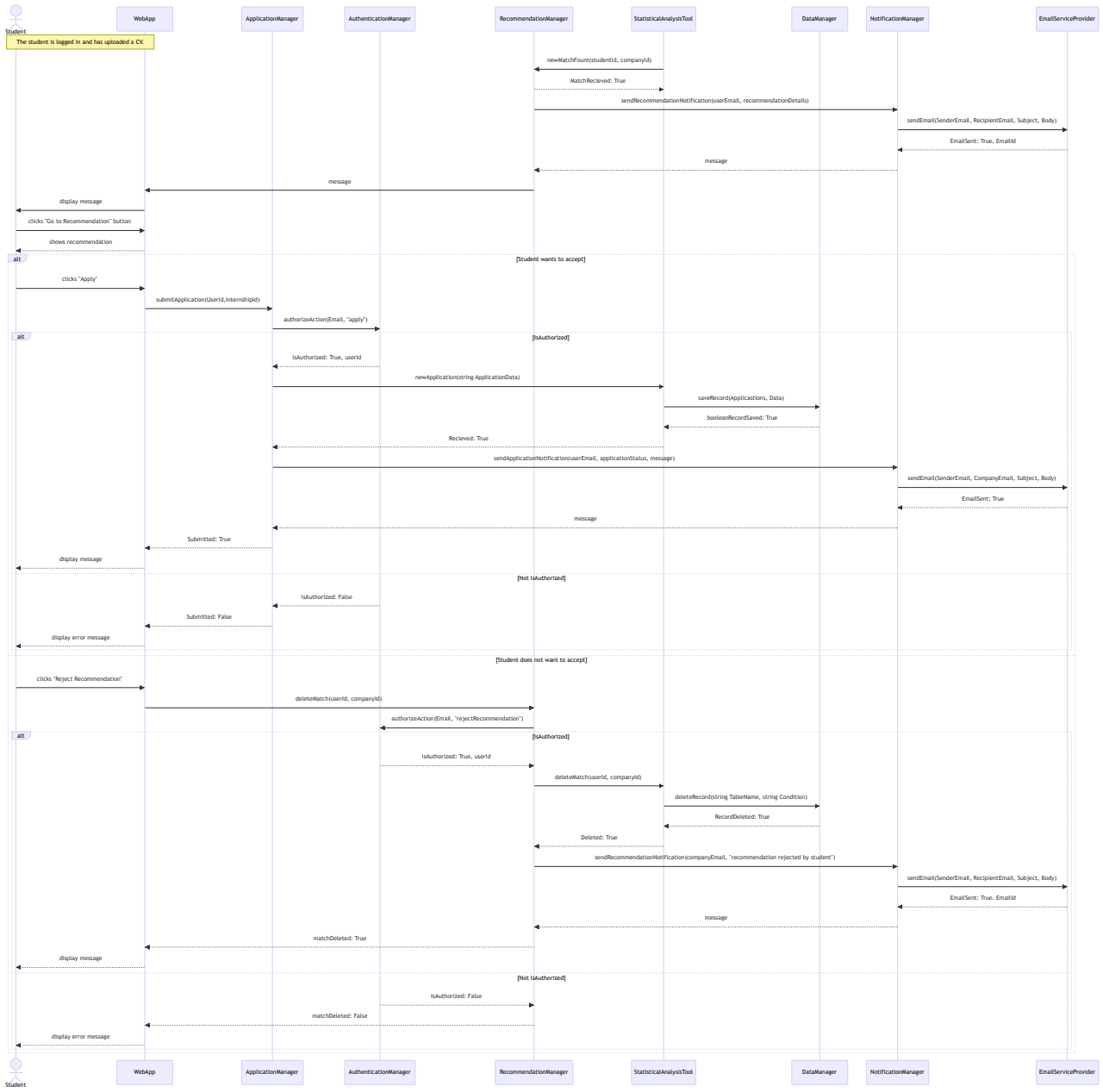

Figure 2.7: UC4

UC5: Company recieves a student's recommendation

This sequence diagram illustrates the recommendation handling process for companies in a web app. After logging in and uploading a project description, companies are notified of student matches generated by the StatisticalAnalysisTool via email and app notifications (same procedure described in the previous diagram's description). Companies can choose to accept or reject the recommendations. If they are interested in a student, and the student has already applied, the company accepts the application and notifies the student. If the student hasn't applied, the company invites the student to apply. If the company isn't interested, and the student has applied, the application is rejected, and the student is notified. If the student hasn't applied, the company rejects the recommendation, removes the match, and notifies the student.

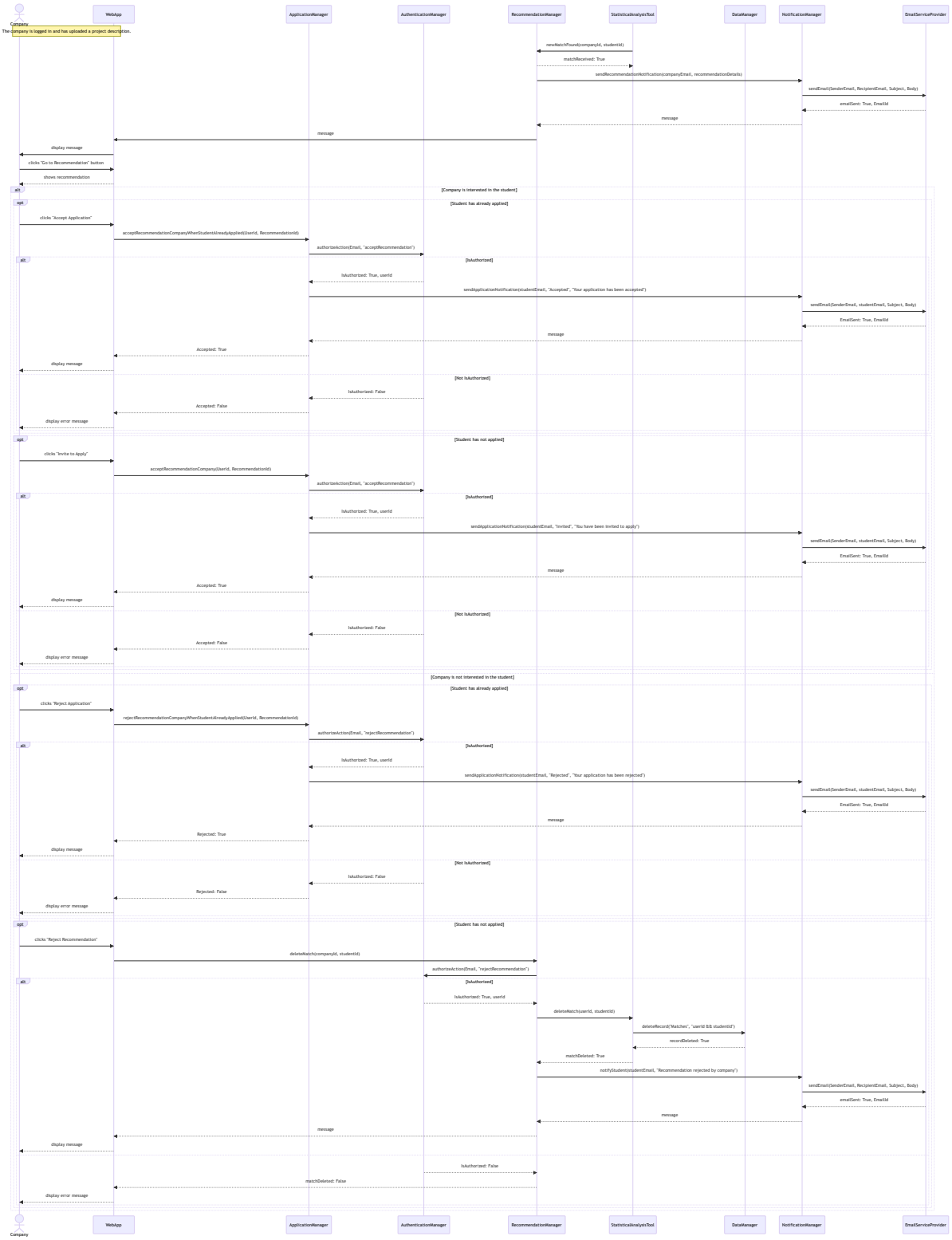

Figure 2.8: UC5

UC6: Interview

This sequence diagram illustrates a simplified version of the interview use case. We chose to do a simplified sequence diagram denoting only the basic flow of events, since otherwise it would have been very big and unreadable. The flow of events described is the case in which the university initiates contact, and the system notifies the student of it; then the university creates an assessment, and evaluates it after the student has responded (we did not model the student's action of filling the assessment). The student is then selected and the company schedules the interview using the system.

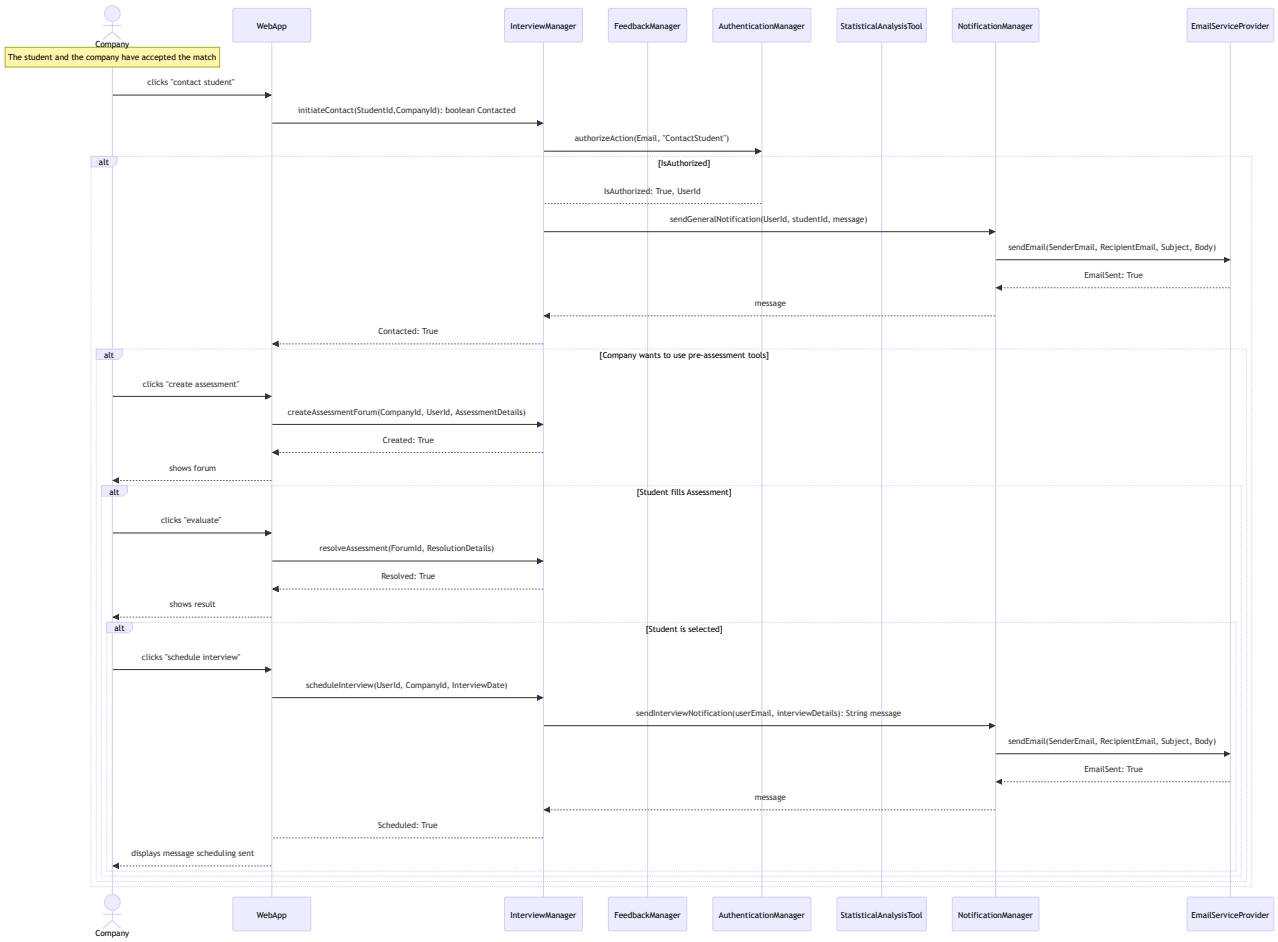

Figure 2.9: UC6

UC7: User leaves feedback

This sequence diagram illustrates the process of leaving internship feedback in the web app. After the user participates in an internship, they receive a feedback notification via email. If the user decides to leave feedback, they fill out a form, which triggers an authorization check. If authorized, the feedback is submitted, saved in the database, and confirmed in the web app. If not authorized, the submission is rejected. If the user chooses not to leave feedback, the message is simply closed without any action.

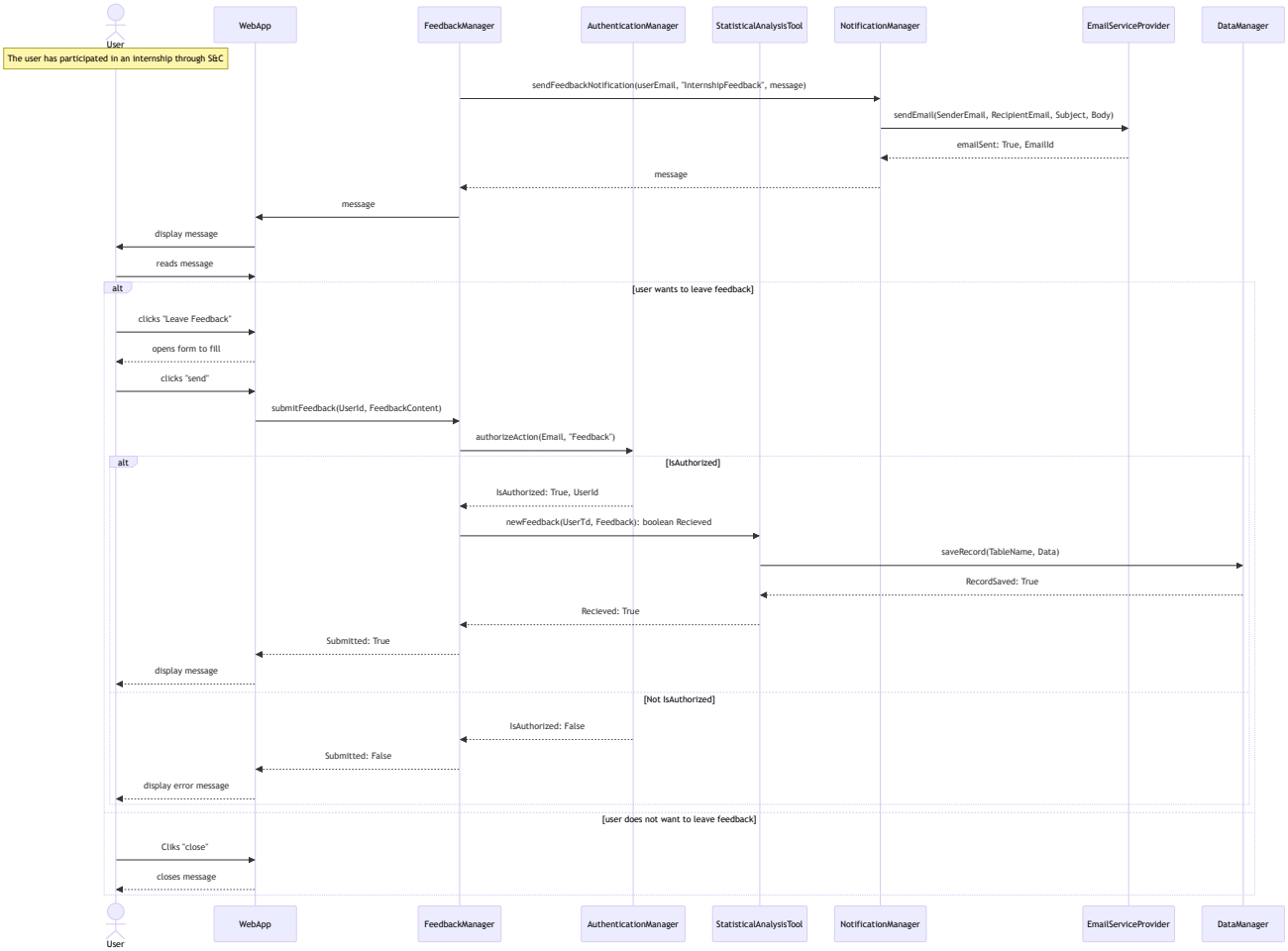

Figure 2.10: UC7

UC8: Student recieves suggestion on CV

This sequence diagram describes the process of CV improvement suggestions for a student. After the student uploads their CV, the StatisticalAnalysisTool generates improvement suggestions and sends a notification via email through the NotificationManager and EmailServiceProvider. The suggestion is then displayed on the web app (as previously described). If the student finds the suggestion useful, they can upload a new version of the CV, which triggers an authorization check through the AuthenticationManager. If authorized, the CV is updated, and the suggestion process continues by calling the statisticalAnalysisTool (the return is not shown since it would be a loop of this use case). If the student doesn't find the suggestion useful, they can close it, and no changes are made to the system.

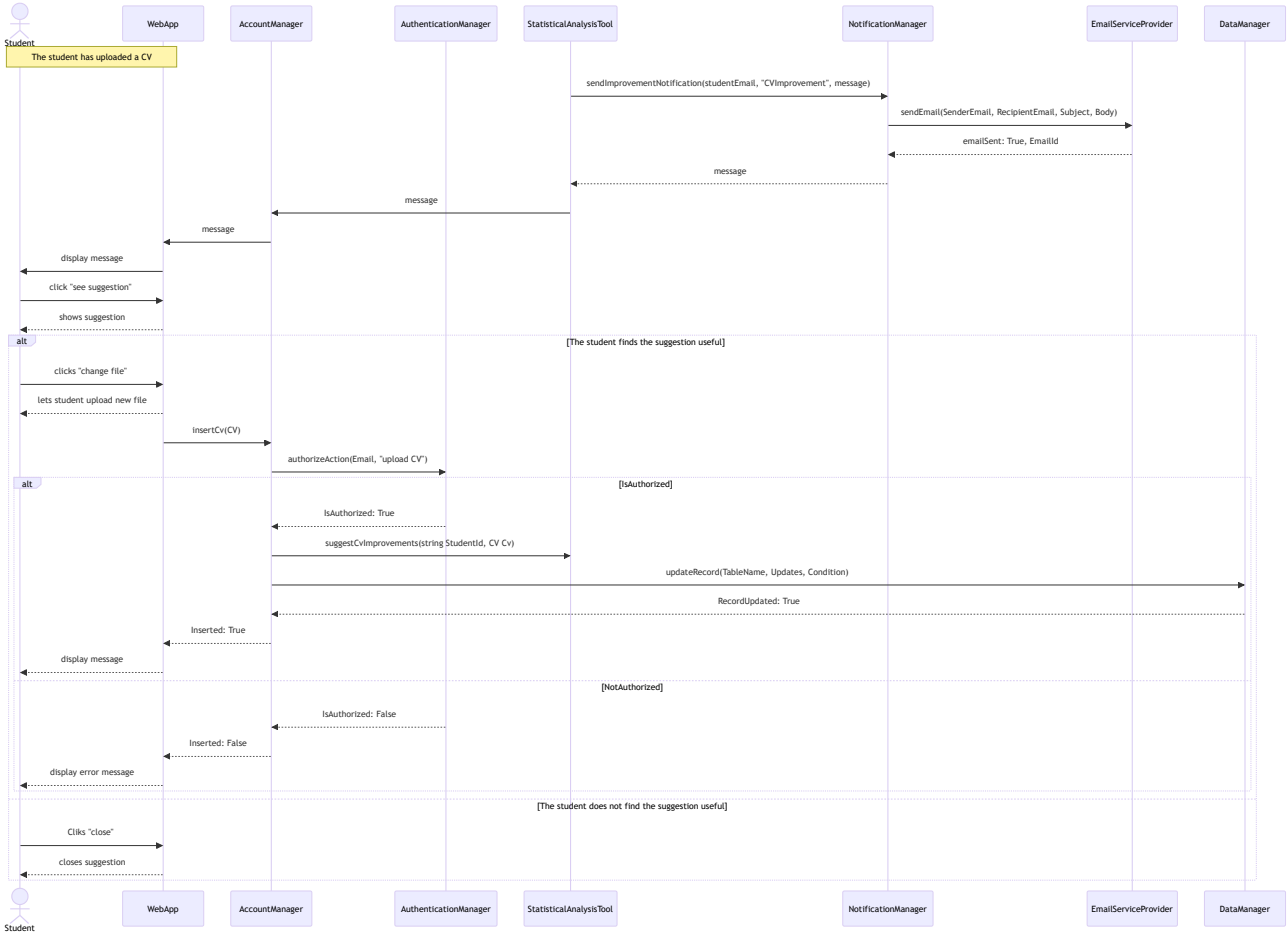

Figure 2.11: UC8

UC9: Company recieves suggestion on project description

This sequence diagram describes the process for a company to handle project description improvement suggestions the web app. After uploading a project description, the company receives a suggestion via email. If the company finds the suggestion useful, they can modify the project description, triggering an authorization check. If authorized, the description is updated, the changes are saved in the database and the statisticalAnalysisTool is notified. If not authorized, the modification is rejected. If the company does not find the suggestion useful, they can simply close the suggestion without making any changes.

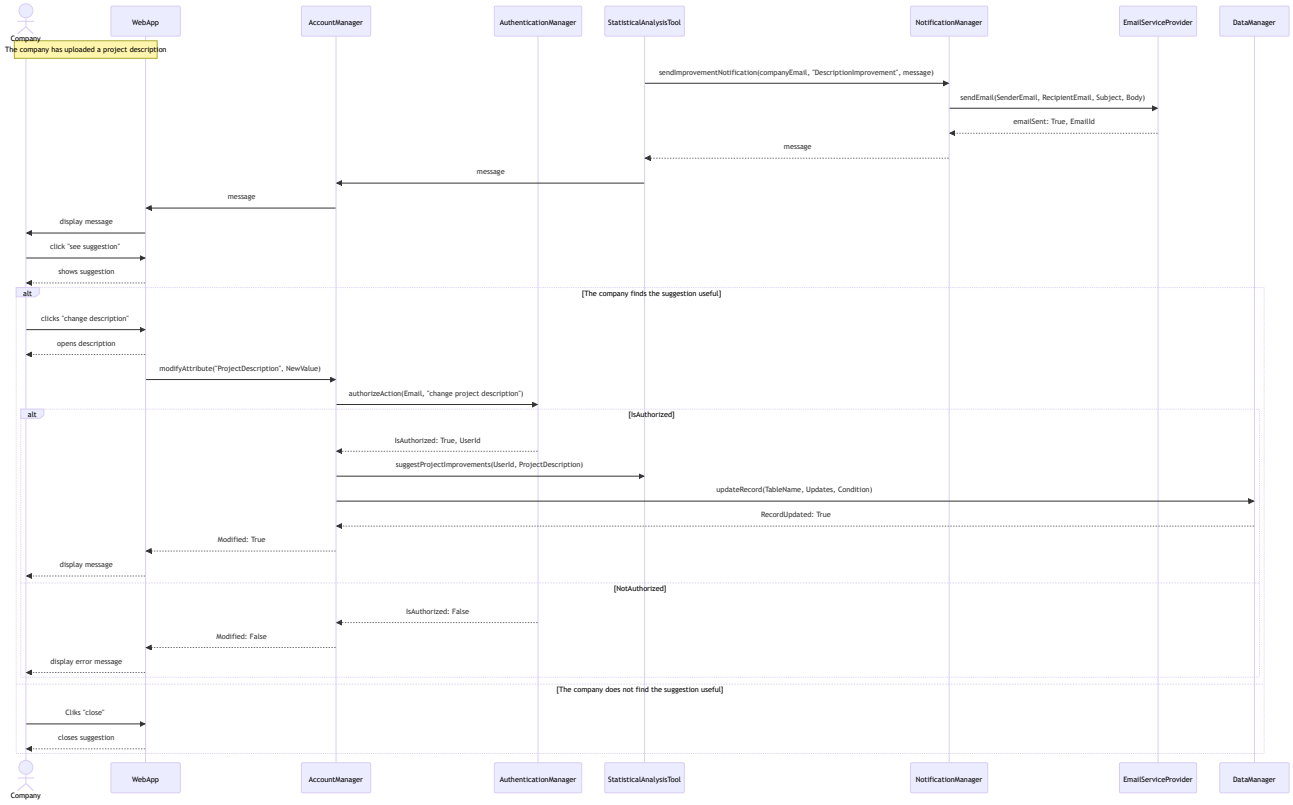

Figure 2.12: UC9

UC10: User makes a complaint

This sequence diagram describes the process in which a user interacts with the web app to monitor and leave complaints about their ongoing internship. The user navigates to the "Internships to Monitor" section, views internship details, and submits a comment or complaint. The complaint is saved and a notification is sent to the university.

Once the university logs in, they review the internship and complaints. If the complaint is deemed important, the university contacts the participant to manage the issue, resolves the complaint, and updates its status. If the complaint is critical, the university may terminate the internship, sending notifications to all relevant parties.

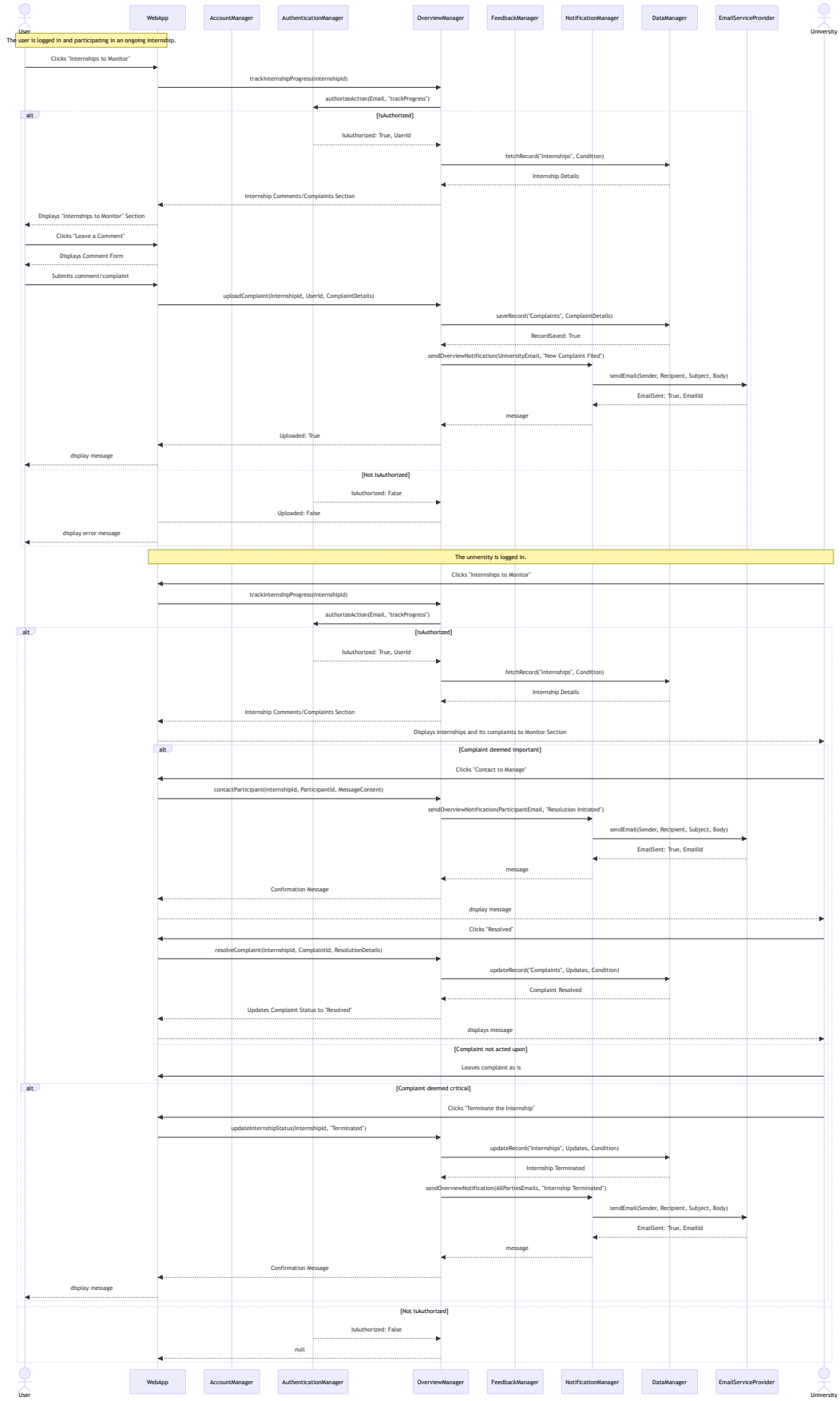

Figure 2.13: UC10

2.5. Component interfaces

The Component Interfaces here described are the ones exposed in the components. Only the most relevant method parameters and methods are shown.

- **Account Manager Interface**
  - **registerUserAsStudent(string Email, string Name, data BirthDate, string LocationBirth, string Password) : boolean AccountCreated**

    This method takes all the data in input and contacts the DataManager to register a new StudentUser in the system.
  - **registerUserAsCompany(string Email, string CompanyName, string LocationAddress, string Password) : boolean AccountCreated**

    This method takes all the data in input and contacts the DataManager to register a new CompanyUser in the system.
  - **registerUserAsUniversity(string Email, string UniversityName, string LocationAddress, string Password) : boolean AccountCreated**

    This method takes all the data in input and contacts the DataManager to register a new UniversityUser in the system.
  - **loginUser(string Email, string Password) : boolean LoginSuccessful**

    This method takes the user's email and password, checks weather they are registered and if they are it logs them into their account
  - **insertCv(CV Cv): boolean Inserted**

    This method allows a StudentUser to upload their Cv on their profile.
  - **deleteCv(): boolean Deleated**

    This method allows a StudentUser to delete their Cv from their profile.
  - **insertProjectDescription(string ProjectDescription): booleand Inserted**

    This method allows a CompanyUser to upload a new project description on their profile.
  - **deleteProjectDescription(string ProjectName): booleand Deleted**

This method allows a CompanyUser to delete a project description from their profile.

- **modifyAttribute(string AttributeName, string NewValue): boolean Modified**

  This method allows users to modify any data present on their profile.

- **Authentication Manager Interface:**
  - **authorizeAction(string Email, string Action) : boolean IsAuthorized, string UserId**

    This method determines whether the specified user has the necessary permissions to perform the given action. It returns 'true' if the action is allowed, otherwise 'false'.

- **Feedback Manager Interface**
  - **submitFeedback(string UserId, string FeedbackContent): boolean Submitted**

    This method allows users to submit their feedback. The content is processed and forwarded to the Statistical Analysis Tool for further use.

  - **getFeedbackHistory(string UserId): list FeedbackList**

    This method retrieves the history of feedback provided by a specific user for review or analysis purposes.

- **Application Manager Interface:**
  - **submitApplication(string UserId, string InternshipId): boolean Submitted**

    This method allows a student to submit an application for a specified internship. It processes the application and records it in the system.

  - **acceptRecommendationStudent(string UserId, string RecommendationId): boolean Accepted**

    This method processes a student's acceptance of a recommendation provided by the system and creates an application for the corresponding internship.

  - **acceptRecommendationCompany(string UserId, string RecommendationId): boolean Accepted**

This method processes a company's acceptance of a recommendation provided by the system and creates an invitation for the Student for the corresponding internship.

- **acceptRecommendationCompanyWhenStudentAlreadyApplied(string UserId, string RecommendationId): boolean Accepted**

  This method processes a company's acceptance of a recommendation provided by the system when the student has already applied to that position.

- **rejectRecommendationCompanyWhenStudentAlreadyApplied(string UserId, string RecommendationId): boolean Rejected**

  This method processes a company's rejection of a recommendation provided by the system when the student has already applied to that position.

- **respondToInvitation(string UserId, string InvitationId): boolean Responded**

  This method allows a student to respond to an invitation from a company and generates an application if the invitation is accepted.

- **Interview Manager Interface:**
  - **createAssessmentForum(string CompanyId, string UserId, string AssessmentDetails): boolean Created**

    This method allows a company to create a forum for conducting a prior assessment with a student. The forum includes assessment details and discussion tools for evaluations.

  - **resolveAssessment(string ForumId, string ResolutionDetails): boolean Resolved**

    This method finalizes the outcome of a prior assessment conducted in the forum and records the resolution in the system.

  - **evaluateAssessment(string ForumId, string EvaluationFeedback): boolean Evaluated**

    This method records feedback and evaluations on the assessment process, providing insights for both the student and the company.

  - **scheduleInterview(string UserId, string CompanyId, datetime InterviewDate): boolean Scheduled**

This method schedules an interview between a student and a company and records the details in the system.

- **updateInterviewStatus(string InterviewId, string Status): boolean Updated**

  This method allows updating the status of an interview (e.g., scheduled, completed, canceled) and ensures the system reflects the current progress.

- **finalizeSelection(string CompanyId, string UserId): boolean Finalized**

  This method records the final selection decision made by the company and updates the relevant data.

- **initiateContact(string StudentEmail, string CompanyEmail): boolean Contacted**

  This method is used by the community to initiate the first contact with the student; it calls the NotificationManager to do so.

- **Recommendation Manager Interface:**
  - **newMatchFount(studentId, companyId): Boolean matchRecieved**

    This method is called by the StatisticalAnalysisTool when a new match is found and is used to notify both parties.

  - **deleteMatch(studentId, companyId): Boolean matchDeleted**

    This method is called by the WebApp when a student or a company rejects a match, it then calls operations from other managers to delete the match from the system and notify the parties involved.

- **Overview Manager Interface:**
  - **trackInternshipProgress(string InternshipId): map ProgressData**

    This method retrieves the current progress data for a specific internship, including details about the student, company, and related activities.

  - **updateInternshipStatus(string InternshipId, string StatusUpdate): boolean Updated**

    This method updates the status of an ongoing internship (e.g., in progress, paused, completed) and records the changes in the system.

- **logInternshipUpdate(string InternshipId, string UpdateDetails): boolean Logged**

  This method registers detailed updates regarding the internship in the database. It communicates with the Authentication Manager for permission and the Data Manager to store the data.

- **uploadComplaint(string InternshipId, string UserId, string ComplaintDetails): boolean Uploaded**

  This method allows students or companies to file a complaint about the internship. The complaint is logged in the system for review and resolution by the university.

- **resolveComplaint(string InternshipId, string ComplaintId, string ResolutionDetails): boolean Resolved**

  This method allows universities to resolve complaints raised during the internship. Resolutions are documented and communicated back through the app.

- **reportComplaintOutcome(string InternshipId, string ComplaintId): boolean Reported**

  This method updates the system with the outcome of externally resolved complaints, ensuring all data remains centralized and accessible.

- **contactParticipant(string InternshipId, string ParticipantId, string MessageContent): boolean Contacted**

  This method enables the university to directly contact a student or company involved in the internship by sending a message through the app.

- **StatisticalAnalysisTool Manager Interface:**
  - **generateStudentMatches(string StudentId): list Matches**

    This method generates a list of potential matches between students and internship projects based on student CVs and project descriptions using the recommendation system.

  - **generateCompanyMatches(string ProjectId): list Matches**

    This method generates a list of potential matches between students and internship projects based on student CVs and project descriptions using the recommendation system.

- **suggestCvImprovements(string StudentId, CV Cv): list Suggestions**

  This method analyzes a student's CV and provides personalized suggestions for improvements to enhance their profile and compatibility with internship opportunities.

- **suggestProjectImprovements(string CompanyId, string ProjectDescription): list Suggestions**

  This method evaluates a company's project description and provides recommendations for making it more appealing and clear to potential candidates.

- **refineRecommendations(string FeedbackData): boolean Refined**

  This method processes feedback received from students and companies about previous recommendations to update and improve the recommendation algorithm.

- **fetchData(string DataRequest): map Data**

  This method interacts with the Data Manager to retrieve necessary data for analysis, ensuring access permissions are obtained through the Authentication Server.

- **newApplication(string ApplicationData): boolean Recieved**

  This method is called when a new application is done by a student and the StatisticalAnalysisTool needs to know it. It then calls the data manager interface to update the database.

- **deleteMatch(string StudentId, string CompanyId): boolean Deleted**

  This method deletes a match that was found by this system if it was rejected by one or more parties.

- **newFeedback(string UserTd, string Feedback): boolean Recieved**

  This method is called by the FeedbackManager when there is new feedback given by a user to be analyzed.

• Notification Manager Interface:

- **sendAccountNotification(userEmail: String, notificationType: String, message: String): String message**

  This method is used to send notifications related to account activities, such

as registration confirmations, password updates, or account deactivations. It's typically called by the Account Manager. The method takes the userEmail (the ID of the user to notify), a notificationType (like "Registration"), and a message containing the notification content. The message is both sent by email through the notification manager and the Account Manager receives the formatted message through the backward chain and passes it along to the WebApp, where it's displayed to the user in the notification page.

– **sendFeedbackNotification(userEmail: String, feedbackType: String, message: String): String message**

  This method is used to send notifications requesting or acknowledging user feedback, such as a prompt to review an internship. It's typically called by the Feedback Manager. The method takes the userEmail (the email of the user to notify), a feedbackType (like "InternshipFeedback"), and a message containing the notification content. The message is sent by email through the Notification Manager, and the Feedback Manager receives the formatted message through the backward chain and passes it along to the WebApp, where it is displayed to the user to encourage engagement.

– **sendImprovementNotification(userEmail: String, improvementType: String, message: String): String message**

  This method is used to send notifications on improvements on a student's CV or a company's project description that the StatisticalAnalysisTool found. The method takes the userEmail (the email of the user to notify), a improvement-Type (like "CVFeedback"), and a message containing the notification content. The message is sent by email through the Notification Manager.

– **sendRecommendationNotification(userEmail: String, recommendationDetails: String): String message**

  This method is used to send notifications about internship recommendations to students or companies, informing them about relevant matches. It's typically called by the Recommendation Manager. The method takes the userEmail (the email of the user to notify) and recommendationDetails (like "You have been recommended for the Software Engineer position at TechCorp"). The message is sent by email through the Notification Manager, and the Recommendation Manager receives the formatted message through the backward chain and passes it along to the WebApp, where the user is notified about the

recommendation on their dashboard.

- **sendApplicationNotification(userEmail: String, applicationStatus: String, message: String): String message**

  This method is used to notify users about updates related to their internship applications, such as submission confirmations, acceptance, or rejections. It's typically called by the Application Manager. The method takes the userEmail (the email of the user to notify), an applicationStatus (like "Submitted" or "Accepted"), and a message detailing the update. The message is sent by email through the Notification Manager, and the Application Manager receives the formatted message through the backward chain and passes it along to the WebApp, where the user sees the application update on the notification page.

- **sendInterviewNotification(userEmail: String, interviewDetails: String): String message**

  This method is used to notify users about interview-related updates, such as scheduling, rescheduling, or cancellations. It's typically called by the Interview Manager. The method takes the userEmail (the email of the user to notify) and interviewDetails (like "Your interview is scheduled for July 5th at 3:00 PM"). The message is sent by email through the Notification Manager, and the Interview Manager receives the formatted message through the backward chain and passes it along to the WebApp, where the user sees the interview details on their notification page.

- **sendOverviewNotification(userEmail: String, progressUpdate: String): String message**

  This method is used to notify users about updates on the progress of ongoing internships, ensuring students, universities, and companies are informed about important milestones. It's typically called by the Overview Manager. The method takes the userEmail (the email of the user to notify) and a progressUpdate (like "Your student has reached 80% of the internship goals"). The message is sent by email through the Notification Manager, and the Overview Manager receives the formatted message through the backward chain and passes it along to the WebApp, where it's displayed on the notification page for relevant users.

- **sendGeneralNotification(userEmail: string, toUserIds: List[String], message: String): String message**

This method is used for sending generic notifications that don't fit into other specific categories. It supports notifications across various channels and can target multiple users simultaneously. It's called by any module requiring an ad hoc notification. The message is sent by the Notification Manager, and the calling module receives the formatted message through the backward chain and passes it along to the WebApp, where it's displayed in the user's notification area.

- **Data Manager:**
  - **checkDuplicates(string TableName, string Email) : boolean Duplicates**

    This method when called checks if the email inserted already exists in the database.
  - **saveRecord(string TableName, map<string, string> Data) : boolean RecordSaved**

    This method saves a new record in the specified table. It returns 'true' if the record is successfully stored.
  - **fetchRecord(string TableName, string Condition) : map<string, string> FetchedRecord**

    This method retrieves a single record from the specified table that matches the provided condition. Returns the record as a key-value map.
  - **fetchAllRecords(string TableName, string Condition) : list<map<string, string» RecordsList**

    This method retrieves all records from the specified table that match the provided condition. Returns the data as a list of key-value maps.
  - **updateRecord(string TableName, map<string, string> Updates, string Condition) : boolean RecordUpdated**

    This method updates records in the specified table based on the condition provided. Returns 'true' if the update is successful.
  - **deleteRecord(string TableName, string Condition) : boolean RecordDeleted**

    This method deletes records from the specified table that match the provided condition. Returns 'true' if the deletion is successful.

  - **provideDataAccess(string ModuleName, string DataRequest) : map<string, string> AccessedData** This method grants secure data access to a specific module, returning the requested data while enforcing access control policies.

- **Email Service Provider:**
  - **sendEmail(string SenderEmail, string RecipientEmail, string Subject, string Body) : boolean EmailSent, string EmailId**

    This method sends an email to the specified recipient with the provided subject and body content. Returns ‘true‘ if the email is successfully sent.

  - **sendTemplatedEmail(string SenderEmail, string RecipientEmail, string TemplateName, map<string, string> TemplateData) : boolean EmailSent, string EmailId**

    This method sends an email using a predefined template. It takes the template name and a mapping of dynamic data to populate the template fields.

  - **trackEmailDelivery(string EmailId) : string DeliveryStatus**

    This method checks the delivery status of a previously sent email using its unique email ID. It returns statuses such as "Delivered," "Failed," or "Pending."

  - **configureEmailSettings(string SMTPServer, int Port, string SenderEmail, string SenderPassword) : boolean SettingsConfigured**

    This method configures the SMTP settings for sending emails, including the server address, port, sender email, and authentication details.

  - **validateEmailAddress(string EmailAddress) : boolean IsValid**

    This method checks whether the given email address is in a valid format and can be used for sending emails.

  - **resendFailedEmails(string[] FailedEmailIds) : boolean Resent**

    This method retries sending emails that previously failed to be delivered. Returns ‘true‘ if all retries are successful.

- **DBMS:**
  - **postData(string TableName, map<string, string> Data) : boolean IsStored**

This method inserts the provided data into the specified table within the database. Returns ‘true‘ if the operation is successful.

- **getData(string Query) : list<map<string, string > > RetrievedData**

  This method executes the provided query to fetch data from the database. It returns the results as a list of records.

- **updateData(string TableName, map<string, string> Updates, string Condition) : boolean IsUpdated**

  This method updates records in the specified table based on the condition provided. Returns ‘true‘ if the update is successful.

- **deleteData(string TableName, string Condition) : boolean IsDeleted**

  This method removes records from the specified table that match the provided condition. Returns ‘true‘ if the deletion is successful.

- **manageAccessControl(string UserId, string[] Permissions) : boolean AccessManaged**

  This method updates access control policies for a specific user by assigning or revoking permissions.

2.6. Selected architectural styles and patterns

2.6.1. Three Tier Architecture

In the Students & Companies project, we chose a three-tier architecture as explained in Section 2.1. This architecture separates the system into three layers (or tiers) : the Presentation layer, the Application layer, and the Data layer. For this implementation, we opted for a remote presentation layer, where the graphical user interface (GUI) resides entirely on the client side, such as in a web browser or a mobile app. This decision brings several advantages: it reduces the server's workload by allowing it to focus on processing business logic and managing data, improves the responsiveness of the user interface since it is processed locally on the user's device, and reduces bandwidth usage while improving network efficiency.

The three-tier architecture distributes responsibilities effectively within the Students & Companies application. The Presentation tier handles all user interactions, providing a seamless interface for students, companies, and universities to input and access data. The Application tier serves as the processing hub, managing tasks like user authentication, recommendation algorithms, and communication between the user interface and the database. Finally, the Data tier is responsible for securely storing and managing essential data, such as user profiles, CVs, project descriptions, and application histories, while performing all database operations to ensure reliability and data integrity.

The primary benefit of using a three-tier architecture, and the reason it was chosen for the Students & Companies system, is its modularity. Each tier can be developed, maintained, and operated independently on separate infrastructures. This separation simplifies updates and extensions by allowing modifications to be made to one tier without affecting the others. It enhances scalability and performance, making it easier to handle increased loads, and provides flexibility for evolving and adapting the system to future needs while maintaining its reliability and stability.

2.6.2. Component Based Development

Our approach is component-based, meaning the system is built using distinct, independent components that each handle specific functionalities. This approach offers significant advantages both in the short and long term. For example, by defining and developing components like the Application Manager or Feedback Manager, each responsible for a specific task, we can work on them independently without affecting the rest of the system. This modularity makes it easier to maintain, update, or even replace individual components without disrupting the entire system.

2.6.3. REST Architecture

Our architecture is designed around REST principles, ensuring a stateless system where the server processes each request independently, while the client manages its own state. This approach improves scalability and efficiency by decoupling state management from the server, enabling each request to include all the necessary information for processing. A significant advantage of this design is cacheability: frequently accessed data, such as internship listings or static resources, can be cached, reducing server load and improving response times for users.

The REST architecture also ensures a uniform interface, with APIs designed around resources rather than actions. Endpoints like /emails, /data, or /records allow operations through standard HTTP methods (GET, POST, PUT, DELETE), providing a consistent and predictable structure across the system. The methods we declared in the APIs, such as sendEmail and postData, are our custom implementations and are mapped to RESTful operations (e.g., POST for creating resources or GET for retrieving them), ensuring that the system aligns with REST principles while retaining flexibility.

Additionally, REST aligns perfectly with our modular and scalable deployment architecture. The stateless nature of REST makes it easy to distribute requests across multiple servers using a load balancer, ensuring efficient handling of high traffic. By adopting a hybrid approach, combining preconfigured services (e.g., for email handling) with custom RESTful endpoints, the system achieves flexibility while adhering to REST principles. Overall, REST provides the scalability, adaptability, and consistency required to meet the needs of the Students & Companies platform and its users.

2.6.4. Model View Controller

The Model-View-Controller (MVC) pattern is a great fit for our system and works well with the architecture choices we've made. It divides the application into three distinct layers: the Model manages the data (consistent with the Data Layer in our three-tier architecture), the View handles user interaction and display (aligned with the Presentation Layer), and the Controller connects both the data and the user interface (matching the Application Layer's role of processing business logic).

By using MVC, we maintain the modularity and separation of concerns we discussed earlier with the three-tier architecture. The MVC pattern enhances our ability to scale and maintain the system since each layer can evolve independently. For instance, we can modify the user interface (View) or improve the application logic (Controller) without impacting how the data is managed (Model). Additionally, MVC supports the RESTful principles we implemented, ensuring that each request is stateless and can be handled independently. This consistency between architectural choices ensures that the application remains flexible, maintainable, and easily adaptable to future changes.

2.7. Other design decisions

2.7.1. MongoDB

We chose to implement our databases using a NoSQL model, specifically MongoDB, as it is well-suited for modern web applications that require scalability, flexibility, and fast development cycles. MongoDB's schema-less design allows us to handle evolving data requirements effectively and simplifies integration with our architecture.

we divided the data into three distinct collections within separate MongoDB databases:

- **User Database:** This database is responsible for managing user-related data, including profiles for both students and companies, login credentials, and session details. It is accessed during user authentication (login/registration) and session management. MongoDB's document model allows us to store and query diverse user types efficiently while adapting to changing data requirements.
- **Opportunities Database:** This database handles all data related to internship opportunities, including job postings, candidate recommendations, invitations, and match notifications. Each request involving the creation, update, or retrieval of internship-related data interacts with this database. MongoDB's flexible schema is ideal for storing hierarchical data, such as job descriptions, candidate lists, and interview schedules, in a single, cohesive structure.
- **Internship Feedback Database:** This database manages feedback and issue tracking during internships. It stores data related to feedback from students, companies, and universities, as well as complaint logs and resolution updates. The use of embedded documents ensures that related feedback and complaint data can be stored together for efficient access and analysis

MongoDB was chosen for its scalability, built-in sharding for distributing data across servers, and support for advanced aggregation pipelines, which are ideal for handling complex queries efficiently.

3 | User Interface Design

This section presents screenshots of the Students and Companies app, highlighting its interface and key features, giving a clear view of how users interact with the platform.

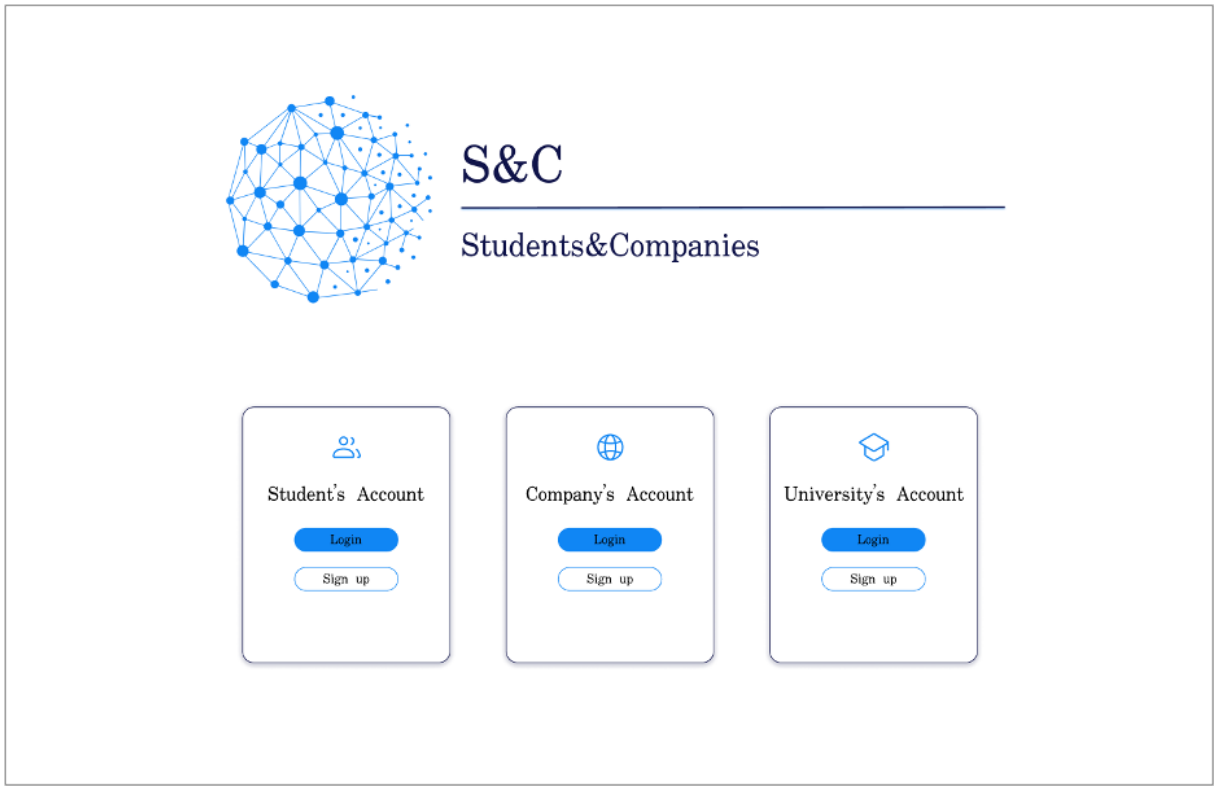

Figure 3.1: Login Page

The Login Page allows users to choose whether to create a new account or log into an existing one. Users can select their role (student, company, or university) to access the appropriate features of the app.

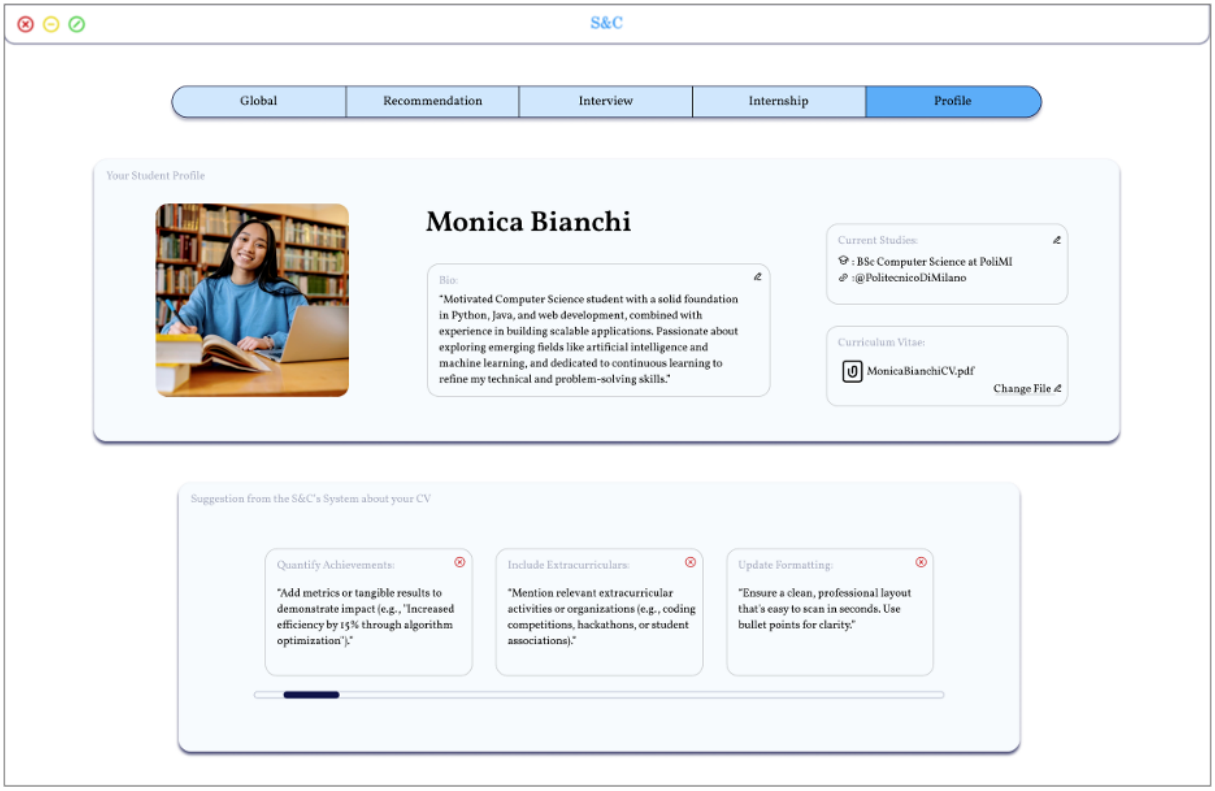

Figure 3.2: Student's Profile Page

The Student's Profile Page displays the student's photo, bio, and education information, with options to view and update this information. It also allows the student to upload, delete, or re-upload their CV. Below the profile details, the page shows personalized suggestions from the Students and Companies system, offering hints to improve their CV.

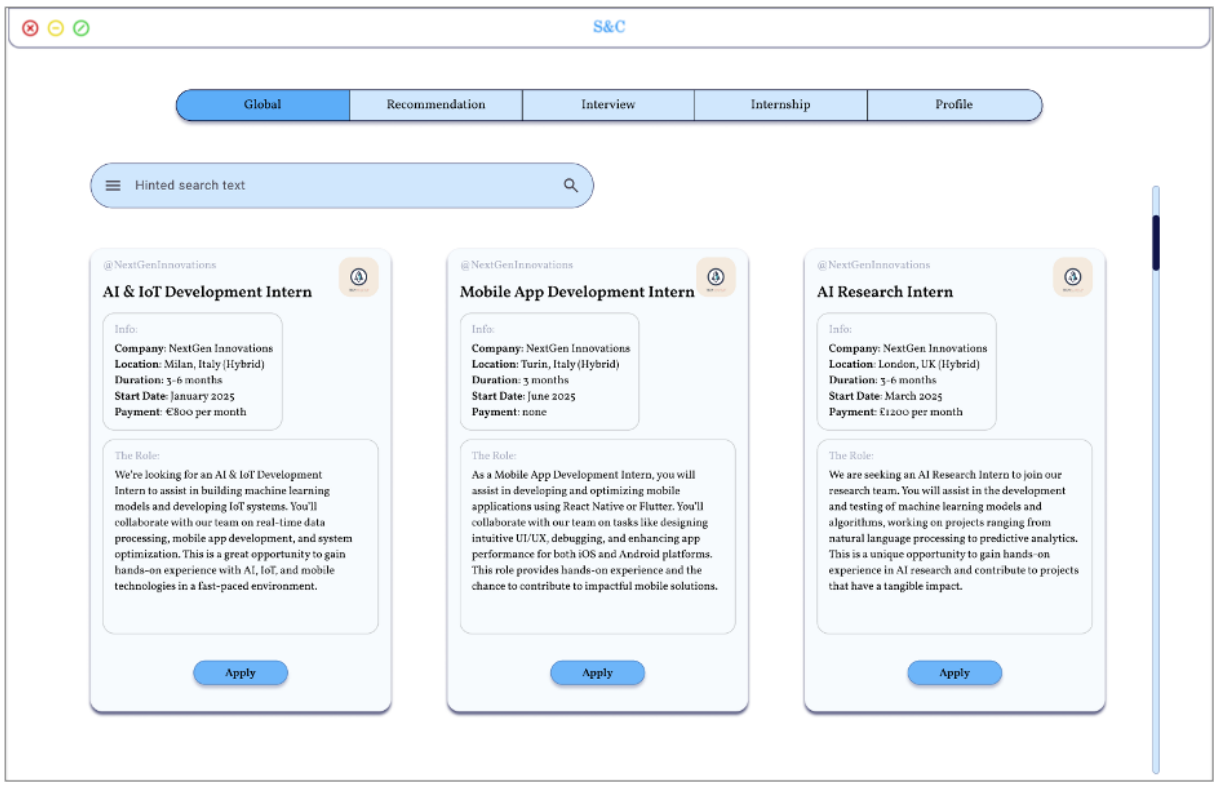

Figure 3.3: Student's Global Searching Page

The Student's Global Searching Page allows students to browse all available internship project descriptions. They can proactively search for internships that match their interests and skills, using keywords or specific parameters. Students also have the option to apply directly to internships that suit them best.

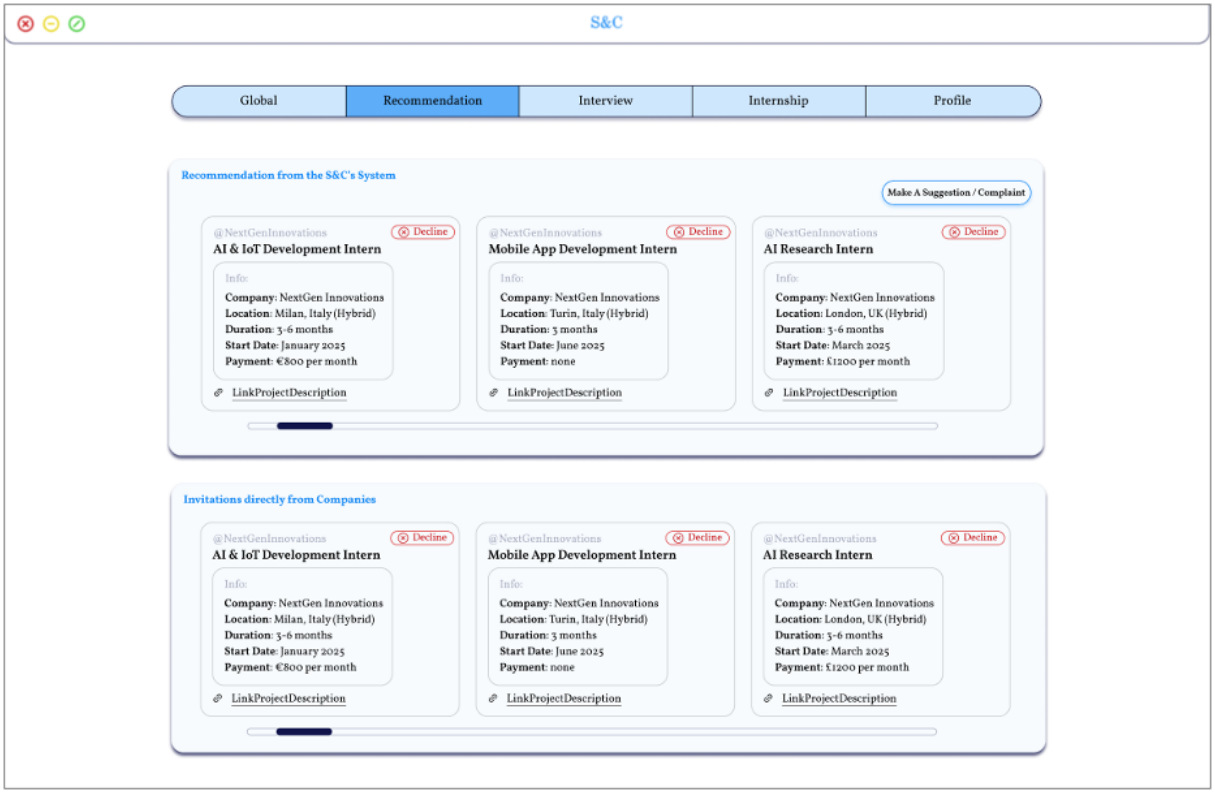

Figure 3.4: Student's Recommendations Page

The Student's Recommendations Page displays internship opportunities suggested by the Students and Companies system, tailored to the student's profile. The student can accepts the recommendation applying for the position. Below these recommendations, the student can also view invitations sent directly from companies. In both cases, the student can apply directly to the internships.

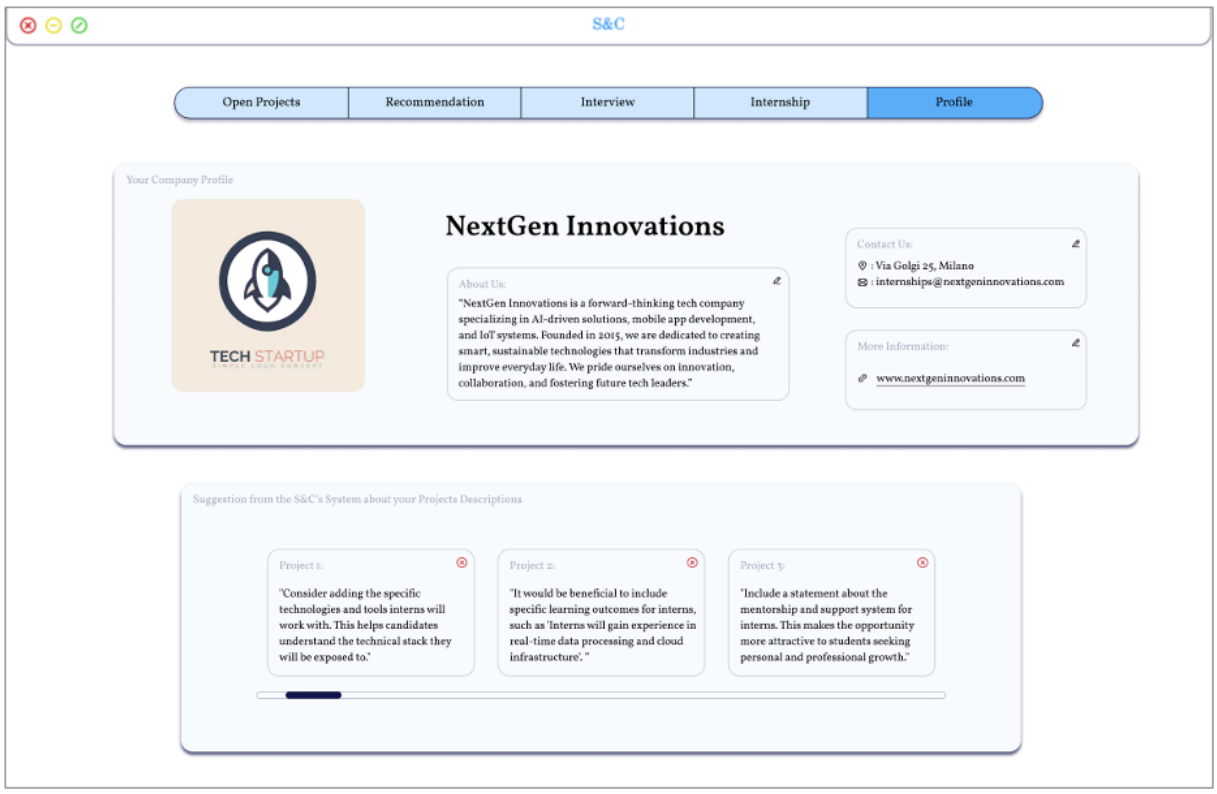

Figure 3.5: Company's Profile Page

The Company's Profile Page displays the company's bio and all the information they've provided about themselves, with the option to edit and update this content. Below, the page shows suggestions from the Students and Companies system on how to improve the descriptions of each posted internship project, making them more appealing and informative.

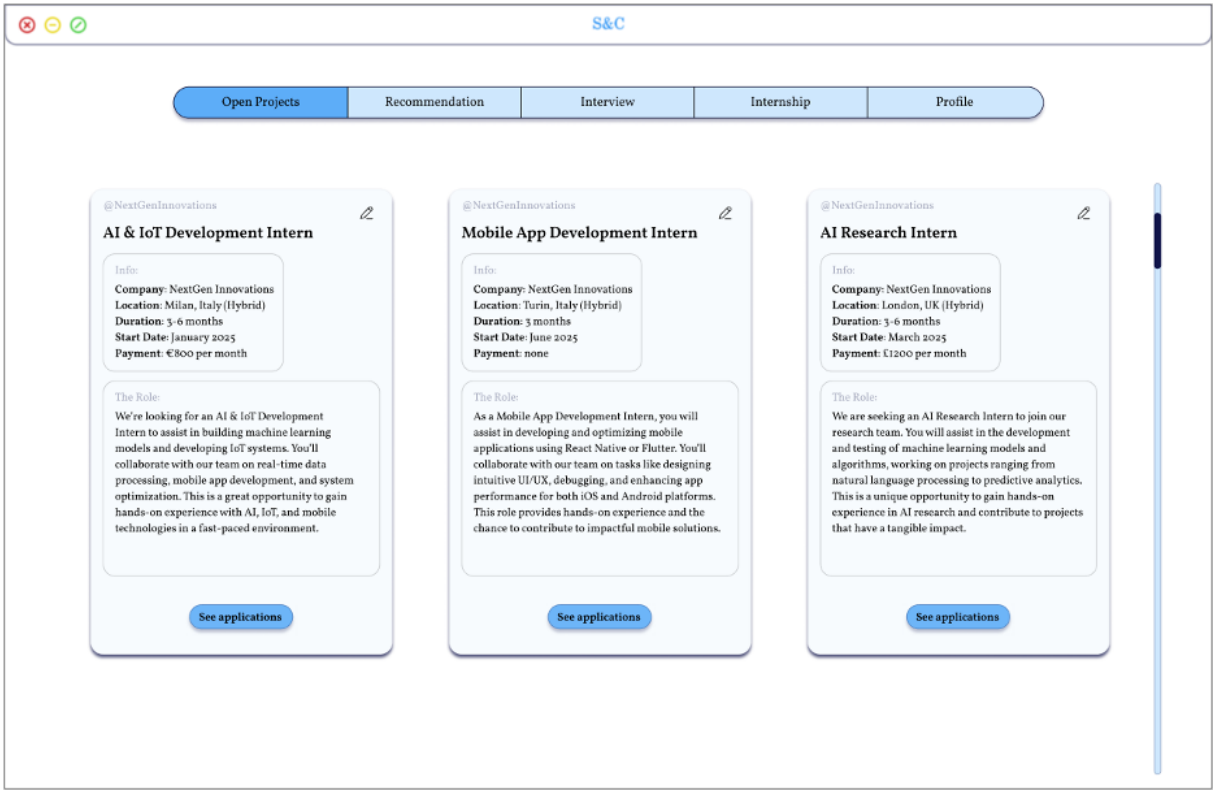

Figure 3.6: Company's open projects Page

The Company's Open Projects Page displays all the internship project descriptions that the company has published. It offers the option to modify these descriptions and provides direct links to view the applications received for each internship.

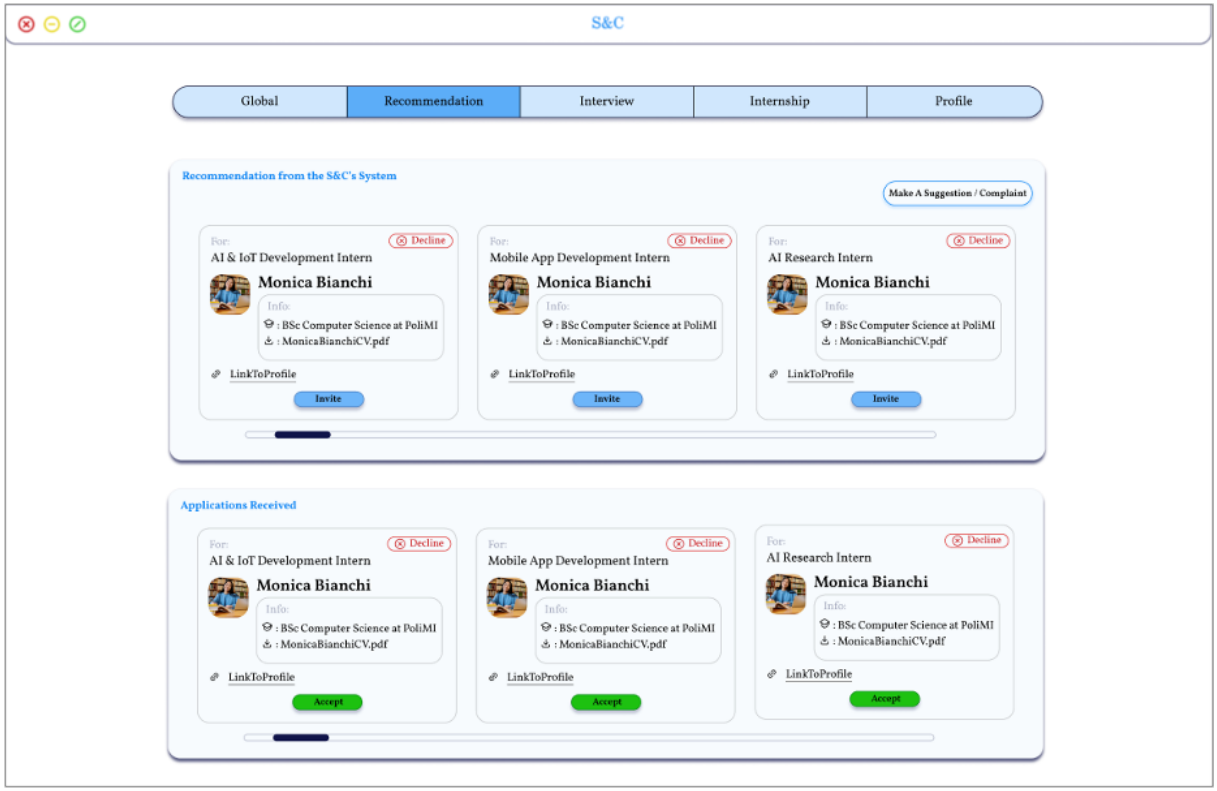

Figure 3.7: Company's Recommendations Page

The Company's Recommendations Page displays a list of recommended candidates from the system, providing a preview of their profiles with a direct link to view more details. The company can accept recommendations and invite students to apply. Below the recommendations, the page also shows all the applications the company has received from students.

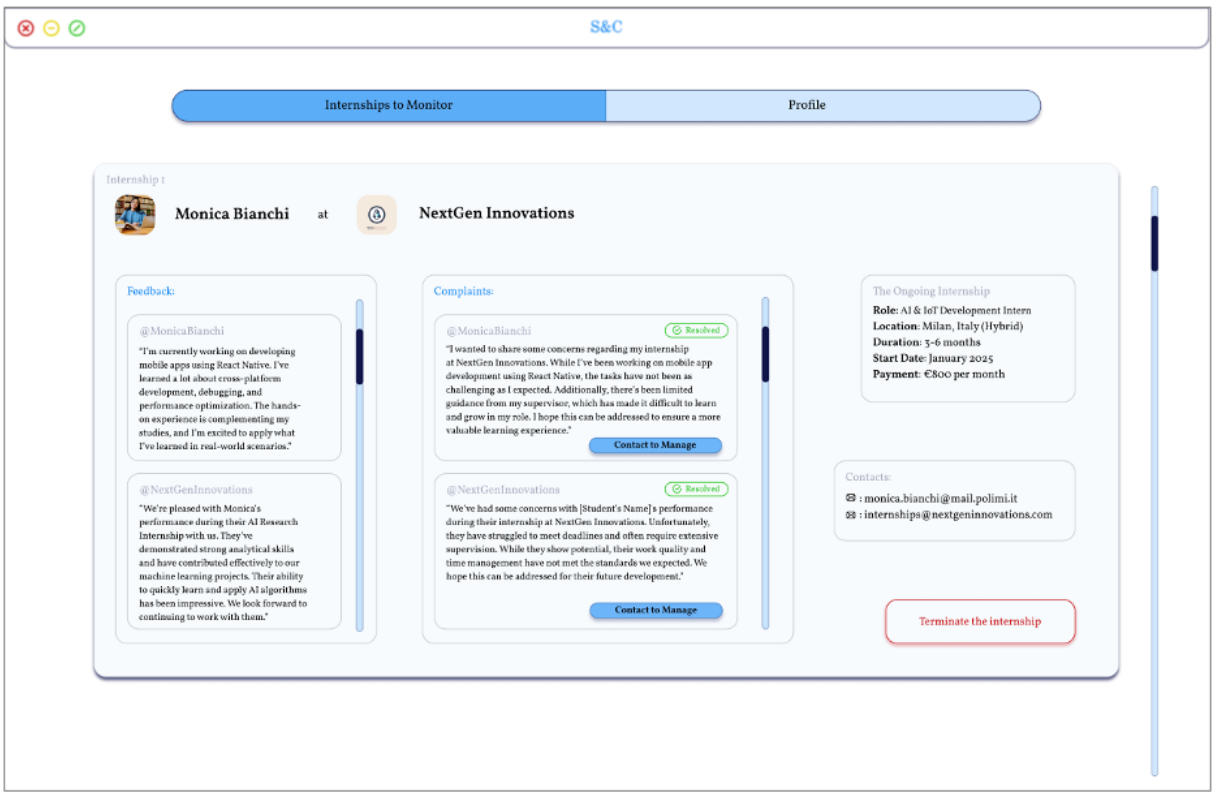

Figure 3.8: University's ongoing internship Monitoring Page

The University's Ongoing Internship Monitoring Page provides a comprehensive overview of all ongoing internships, including details about the student, company, role, and relevant contact information. The page features sections for both feedback and complaints, which can be submitted by either the student or the company. Complaints can be managed within the app or resolved externally, with the option to report the resolution back on the platform.

4 | Requirements Traceability

This chapter outlines the traceability framework implemented to maintain alignment between the project objectives, stakeholder needs, and the final deliverables. Establishing clear links between requirements, design, implementation, and testing minimizes the risk of unfulfilled specifications and promotes a transparent development process.

[R1] The system allows unregistered users to create an account.

- **WebApp:** Provides the user interface for new users to enter their registration details.
- **AccountManager:** Manages the account creation process, validates user details and ensures proper storage of data.
- **AuthenticationManager:** Verifies that the registration process complies with security protocols.
- **DataManager:** Handles the storage of new user data in the database.
- **NotificationManager:** Sends notifications to users about the successful creation of their accounts.
- **EmailServiceProvider:** Sends email confirmation or verification messages to new users.
- **DBMS:** Stores all account information securely in the database.

[R2] The system allows students to upload their CV.

- **WebApp:** Provides the interface for students to upload their CVs.
- **AccountManager:** Ensures that only authenticated students can upload their CVs.
- **AuthenticationManager:** Validates the session and permissions of the student uploading the CV.
- **DataManager:** Stores the uploaded CV securely in the database.

- **StatisticalAnalysisToolManager:** Analyzes the CV to improve recommendations and provide insights for better matching.
- **DBMS:** Stores CV data for future reference and analysis.

[R3] The system allows companies to publish new internships.

- **WebApp:** Provides the interface for companies to add new internship postings.
- **AccountManager:** Ensures only authenticated company representatives can post internships.
- **AuthenticationManager:** Verifies that the company representative has the necessary permissions.
- **DataManager:** Stores the internship postings in the database.
- **StatisticalAnalysisToolManager:** Analyzes the internship postings to match them with student profiles.
- **DBMS:** Safely stores internship data for future searches and recommendations.

[R4] The system allows companies to add a description to their internships.

- **WebApp:** Provides the interface for adding detailed descriptions to internships.
- **AccountManager:** Verifies that authenticated companies can update internship details.
- **AuthenticationManager:** Checks permissions for modifying internship descriptions.
- **DataManager:** Updates and stores the descriptions in the database.
- **StatisticalAnalysisToolManager:** Analyzes the internship descriptions to match them with student profiles.
- **DBMS:** Safely records the updated internship details.

[R5] The system allows students to browse available internships.

- **WebApp:** Displays a list of internships for students to browse through.
- **ApplicationManager:** Manages the search and retrieval of available internships.
- **StatisticalAnalysisToolManager:** Analyzes browsing trends and preferences to refine recommendations.
- **DataManager:** Retrieves and organizes internship data for display.

- **DBMS:** Stores internship details to facilitate searches and retrievals.

[R6] The system allows users to filter internships when performing a search.

- **WebApp:** Provides filtering options for criteria such as location, domain, and duration.
- **ApplicationManager:** Processes search queries and applies filters to display relevant results.
- **StatisticalAnalysisToolManager:** Enhances filtering by analyzing patterns in user preferences.
- **DataManager:** Applies filter criteria to the database and retrieves relevant results.
- **DBMS:** Stores filterable data for internships, enabling detailed searches.

[R7] When finding an internship that suits their interests, the system allows students to apply for it.

- **WebApp:** Enables students to submit their applications through the platform.
- **ApplicationManager:** Manages the submission of applications and notifies the relevant company.
- **StatisticalAnalysisToolManager:** Tracks application data to refine matching and recommendations.
- **AuthenticationManager:** Verifies the permissions and authenticity of the student applying.
- **NotificationManager:** Notifies both the student and the company about the application status.
- **EmailServiceProvider:** Sends confirmation emails to both parties regarding the application.
- **DataManager:** Processes and records application data for audit and tracking purposes.
- **DBMS:** Stores applications securely for reference and analysis.

[R8] When finding a CV that suits their interests, the system allows companies to request that student's application.

- **WebApp:** Allows companies to request applications from students through the interface.

- **ApplicationManager:** Processes company requests and sends notifications to the respective students.
- **StatisticalAnalysisToolManager:** Updates its data based on the company's preferences and feedback.
- **AuthenticationManager:** Verifies the company's permissions before granting access to student data.
- **NotificationManager:** Notifies students about application requests from companies.
- **EmailServiceProvider:** Sends email notifications to students regarding company interest.
- **DataManager:** Tracks requests for applications and ensures data integrity.
- **DBMS:** Stores records of application requests and student profiles for future reference.

[R9] When a new internship that might interest some students becomes available, the system notifies them.

- **WebApp:** Displays notification messages to students about new internships.
- **RecommendationManager:** Handles the new match found.
- **StatisticalAnalysisToolManager:** Analyzes new internships and matches them with student profiles.
- **NotificationManager:** Sends real-time notifications about new opportunities.
- **EmailServiceProvider:** Sends email alerts to students about new internships.
- **DataManager:** Updates the database with new internship information and matches.
- **DBMS:** Stores and indexes notifications and recommendation data.

[R10] When a student's CV that corresponds to a company's needs becomes available, the system informs them.

- **WebApp:** Provides a dashboard where companies can view matched CVs.
- **EmailServiceProvider:** Sends notifications to companies regarding matching CVs.
- **StatisticalAnalysisToolManager:** Finds matches between a student's CV and internships descriptions posted by the company.

- **DataManager:** Accesses both the student's CV and the company's internship descriptions.
- **RecommendationManager:** Handles the new match found.
- **NotificationManager:** Sends notifications to the company once a matching CV is found.
- **AuthenticationManager:** Ensures the company has appropriate access permissions.
- **DBMS:** Stores matching data securely and makes it accessible for visualization.

[R11] The system allows students to accept a recommendation, applying for that particular internship.

- **WebApp:** Provides an interface for students to view and accept recommendations.
- **EmailServiceProvider:** Sends confirmation emails upon recommendation acceptance.
- **Recommendation Manager:** Handles the new match found.
- **Data Manager:** Retrieves recommendations linked to the student's profile.
- **Application Manager:** Sends and manages the student's application after accepting the recommendation.
- **Authentication Manager:** Ensures the user is logged in with a student account and has privileges to perform this action.
- **Statistical Analysis Tool Manager:** Records acceptance feedback to improve the matching algorithm.
- **DBMS:** Stores all associated recommendation and application data securely.

[R12] The system allows companies to accept a recommendation, inviting the candidate that was proposed.

- **WebApp:** Provides an interface for companies to view and accept recommendations.
- **EmailServiceProvider:** Sends invitations to students after company acceptance.
- **Recommendation Manager:** Handles the new match found.
- **Data Manager:** Retrieves recommendations linked to the company profile.

- **Notification Manager:** Notifies the proposed candidate of the company's acceptance.
- **Authentication Manager:** Ensures the company is logged in with the appropriate account and privileges.
- **Statistical Analysis Tool Manager:** Records acceptance feedback to refine recommendations.
- **DBMS:** Stores data related to company recommendations securely.

[R13] The system allows students to accept an invitation of a company for a particular internship, applying for it.

- **WebApp:** Provides an interface for students to view and accept invitations.
- **EmailServiceProvider:** Sends confirmation emails upon invitation acceptance.
- **Application Manager:** Manages the submission of the internship application after invitation acceptance.
- **Notification Manager:** Sends notifications confirming the invitation acceptance.
- **Data Manager:** Retrieves invitations linked to the student's profile.
- **Authentication Manager:** Verifies the user's privileges and account type.
- **Statistical Analysis Tool Manager:** Records feedback to enhance the recommendation system.
- **DBMS:** Stores all related invitation and application data securely.

[R14] When the two parties have accepted a recommendation, or when an application or invitation is accepted, the system allows them to establish contact.

- **WebApp:** Provides an interface for users to exchange contact details or initiate communication.
- **EmailServiceProvider:** Sends confirmation and contact-sharing emails.
- **Statistical Analysis Tool Manager:** Records interaction data for analytics.
- **Notification Manager:** Notifies users when contact is established.
- **Interview Manager:** Assists in scheduling meetings or interviews as needed.
- **Authentication Manager:** Ensures that users have appropriate privileges.
- **Data Manager:** Retrieves and stores contact interaction data.

- **DBMS:** Maintains records of all communication actions securely.

[R15] When conducting an interview, the system supports the companies with the interview process.

- **WebApp:** Provides an interface for companies to schedule, manage, and track interviews.
- **EmailServiceProvider:** Sends interview invitations, reminders, and updates to both students and companies.
- **Interview Manager:** Handles interview scheduling, tracking progress, and managing interview-related interactions.
- **Notification Manager:** Sends reminders about upcoming interviews and updates on interview outcomes.
- **Statistical Analysis Tool Manager:** Analyzes feedback and data from the interview process to refine the system.
- **Authentication Manager:** Ensures only authorized users can schedule and manage interviews.
- **Data Manager:** Stores and retrieves interview data such as schedules, feedback, and outcomes.
- **DBMS:** Maintains secure records of all interview-related data.

[R16] When conducting an interview, the system supports the companies with the finalization of the selection.

- **WebApp:** Allows companies to update the interview outcomes and finalize selections.
- **EmailServiceProvider:** Notifies students about their selection status.
- **Interview Manager:** Tracks interview progress and helps finalize decisions.
- **Notification Manager:** Informs students of the final decision, whether selected or not.
- **Data Manager:** Stores final selection details and related data.
- **DBMS:** Securely manages selection outcomes and records for future reference.

[R17] The system allows students and companies to provide feedback and suggestions to perform statistical analysis.

- **WebApp:** Provides a feedback submission form for both students and companies.
- **EmailServiceProvider:** Sends reminders to submit feedback.
- **Feedback Manager:** Collects and organizes feedback data for analysis.
- **Statistical Analysis Tool Manager:** Processes feedback to generate actionable insights.
- **Notification Manager:** Requests feedback and confirms submission to users.
- **Authentication Manager:** Ensures only authorized users can provide feedback.
- **Data Manager:** Stores feedback securely for statistical analysis.
- **DBMS:** Maintains all feedback records securely.

[R18] The system provides suggestions to students regarding how to make their CVs more appealing.

- **WebApp:** Displays personalized CV improvement suggestions.
- **EmailServiceProvider:** Sends suggestions via email.
- **Statistical Analysis Tool Manager:** Analyzes CV data and trends to generate suggestions.
- **Account Manager:** Updates and manages the student's CV in the system.
- **Data Manager:** Accesses CV and job requirement data for analysis.
- **Notification Manager:** Notifies students about CV improvement recommendations.
- **DBMS:** Stores CV data securely for further analysis.

[R19] The system provides suggestions to companies regarding how to make their project descriptions more appealing.

- **WebApp:** Displays personalized project description suggestions.
- **EmailServiceProvider:** Sends suggestions via email.
- **Statistical Analysis Tool Manager:** Analyzes project description trends to generate improvement suggestions.
- **Account Manager:** Updates and manages project descriptions in the system.
- **Data Manager:** Accesses project and CV data for analysis.

- **Notification Manager:** Notifies companies about improvement recommendations.
- **DBMS:** Stores project description data securely for further analysis.

[R20] During the matchmaking process, the system allows all users to keep track of its execution and outcome.

- **WebApp:** Provides a dashboard to track matchmaking progress and results.
- **EmailServiceProvider:** Sends updates on matchmaking status.
- **Recommendation Manager:** Monitors and displays matchmaking outcomes.
- **Statistical Analysis Tool Manager:** Offers detailed insights into matchmaking efficiency.
- **Notification Manager:** Updates users in real-time on matchmaking progress.
- **Authentication Manager:** Ensures secure access to matchmaking data.
- **DBMS:** Stores matchmaking data securely.

[R21] During the internship, the system allows all interested parties to monitor it.

- **WebApp:** Provides an interface for monitoring internship progress.
- **EmailServiceProvider:** Sends periodic updates on the internship.
- **Overview Manager:** Tracks and displays ongoing internship updates.
- **Notification Manager:** Alerts stakeholders about internship milestones or issues.
- **Data Manager:** Records and retrieves internship monitoring data.
- **Authentication Manager:** Ensures secure access to monitoring features.
- **DBMS:** Maintains records of internship progress securely.

[R22] During an ongoing internship, the system allows all users to complain.

- **WebApp:** Provides a complaint submission interface.
- **EmailServiceProvider:** Sends complaint submission confirmations and updates.
- **Overview Manager:** Logs and tracks complaints.
- **Notification Manager:** Notifies relevant parties about complaints.
- **Authentication Manager:** Validates the identity and permissions of users submitting complaints.

- **Data Manager:** Stores complaints for resolution.
- **DBMS:** Maintains a secure record of complaints.

[R23] During an ongoing internship, the system allows all users to communicate problems.

- **WebApp:** Enables problem reporting through a dedicated interface.
- **EmailServiceProvider:** Sends updates on reported problems.
- **Overview Manager:** Facilitates problem resolution and tracking.
- **Notification Manager:** Updates stakeholders on problem status and resolution.
- **Authentication Manager:** Ensures only authorized users can report issues.
- **Data Manager:** Logs reported problems securely.
- **DBMS:** Stores problem reports and updates securely.

[R24] During an ongoing internship, the system allows all users to provide information on its status.

- **WebApp:** Offers forms for submitting internship status updates.
- **EmailServiceProvider:** Notifies stakeholders about new updates.
- **Overview Manager:** Tracks status updates and displays them to relevant parties.
- **Notification Manager:** Alerts stakeholders about status changes.
- **Data Manager:** Records and retrieves status updates.
- **Authentication Manager:** Verifies user permissions for updating internship status.
- **DBMS:** Secures all status-related records.

[R25] When reports or complaints about the status of an ongoing internship are made, the system allows Universities to see them.

- **WebApp:** Provides a dashboard for viewing reports and complaints.
- **EmailServiceProvider:** Notifies universities about new reports or complaints.
- **Overview Manager:** Displays reports and complaints for university review.
- **Notification Manager:** Alerts universities about updates on complaints.
- **Data Manager:** Retrieves stored complaints for review.

- **Authentication Manager:** Restricts access to authorized university representatives.
- **DBMS:** Maintains secure complaint records.

[R26] When complaints about the status of an ongoing internship are made, the system allows Universities to handle them.

- **WebApp:** Enables universities to address complaints via a dedicated interface.
- **EmailServiceProvider:** Notifies users about complaint resolutions.
- **Overview Manager:** Tracks resolution progress and logs actions.
- **Notification Manager:** Updates stakeholders on complaint resolution progress.
- **Data Manager:** Stores resolution details and actions.
- **Authentication Manager:** Ensures only authorized university staff handle complaints.
- **DBMS:** Secures records of complaint resolutions.

5 | Implementation, Integration, and Testing

This chapter explains how the platform described will be implemented and tested. The primary aim of testing is to identify and address the majority of the bugs in the code produced by the development team. Additionally, a detailed description of the integration of components within the code is provided in Section 5.2. Section 5.1 presents an overview of the most important implementation strategies used in the project.

5.1. Implementation Plan

This section describes the strategies for implementing, integrating, and testing the various components of the system. The implementation approach combines the advantages of both bottom-up and thread-based strategies.

A thread-based strategy is functional because it makes progress visible to users and stakeholders. It allows for fewer drivers than expected but introduces greater complexity during integration.

The top-down methodology will be employed by first designing a basic sketch of the system. Then, more complex functionalities will be added incrementally as validated thread units. This strategy enables different teams to work in parallel, accomplishing tasks independently. Once validated, these units will be integrated into the overall software architecture.

5.1.1. Features Identification

The implementation process begins with identifying the key features required for the system. These features align with the goals specified in the RASD document and correspond to the primary system components outlined in Figure 2.2. The key features include:

1. **Account Creation and User Login:** Supports secure account creation with role-based access for students and companies, enabling personalized profiles and safeguarding information.
2. **Internship Posting:** Companies can post structured, detailed internship opportunities categorized by domain, location, and duration for better visibility and applicant relevance.
3. **CV and Project Analysis:** Automated analysis of student CVs and company project descriptions ensures alignment by evaluating skills, achievements, and project scope.
4. **Recommendation System:** Data-driven algorithms connect students with suitable internships and help companies identify potential candidates.
5. **Interview Management:** Companies can schedule interviews, send invitations, and evaluate candidates using pre-defined tools to streamline the selection process.
6. **Feedback Collection:** Enables feedback from students and companies, with aggregated insights used to improve internships and the platform.
7. **Complaint Management:** Mechanisms for submitting, addressing, and resolving complaints ensure fairness and accountability among stakeholders.
8. **Internship Monitoring:** Tools for tracking progress and performance promote accountability and timely issue resolution.

These components collectively form the core features and capabilities of the platform.

5.2. Component Integration and Testing

Integrating the components of the system is a critical phase that ensures all modules work together seamlessly. The integration process follows these steps:

1. **Unit Validation:** Each component is tested individually to ensure it performs as intended. Stubs and drivers are employed to simulate missing parts during testing.
2. **Thread Integration:** Components validated in isolation are integrated incrementally. Threads representing functional workflows (e.g., user registration, internship posting) are built and tested sequentially.
3. **System Assembly:** Once all threads are validated, they are integrated into the overall system. Compatibility and interactions between threads are thoroughly tested to ensure no conflicts arise.
4. **Final Testing:** After the entire system is assembled, it undergoes a full round of testing, including functional, performance, usability, load, and stress testing, to verify its readiness for deployment.

This systematic approach to integration and testing minimizes errors and ensures that the platform meets its specified requirements.

5.3. System Testing

The S&C platform must be thoroughly tested to ensure that the implemented functionalities align with expectations. During development, each component will be tested individually. However, not all components can be tested in isolation from the entire architecture; therefore, stubs and drivers will be used to replace missing components as needed. Once a thread unit is tested and validated, it will be integrated into the software architecture. When the entire architecture is complete, comprehensive testing will be conducted.

The primary objective of system testing is to verify that the platform meets the functional and non-functional requirements specified in the RASD document. In this process, participation from all stakeholders, including developers and end-users, is essential. The testing steps include:

- **Functional Testing:** This test verifies whether the functional requirements are fulfilled. The most effective way to conduct functional testing is to execute the software as described in the RASD use cases and ensure they are satisfied.
- **Performance Testing:** The goal of this test is to identify bottlenecks that may affect response time, utilization, and throughput. It can also detect inefficient algorithms, hardware/network issues, and optimization opportunities. This test involves loading the system with the expected workload, measuring performance, and comparing results to identify areas for improvement.
- **Usability Testing:** This method evaluates the usability of a website, application, or other digital product by observing real users as they attempt to complete tasks.
- **Load Testing:** Load testing is conducted to identify potential issues such as memory leaks, buffer overflows, or memory mismanagement. This test determines the upper limits of the system's components by increasing the workload until the system can no longer sustain it for a pre-established duration.
- **Stress Testing:** This test ensures that the system can recover gracefully after failure. For stress testing, system resources may be increased or decreased to evaluate the system's resilience.

5.3.1. Extra:

Continuous feedback from users and stakeholders is crucial throughout the development process. Feedback should be solicited each time a new feature is implemented. During alpha testing, it is important to assess user satisfaction. Psychologists or other experts may be consulted to design appropriate questionnaires for the testers involved. Alpha testing helps identify malfunctions before beta testing.

Beta testing, conducted in a production environment, allows real users to uncover any bugs or issues before the general release. Even after release, it is vital to collect logs from the application to assist developers in debugging and improving the system.

6 | Effort Spent

Member of group	Effort spent	
Arianna Paone	Introduction	1h
	Architectural Design	18h
	Requirements Traceability	4h
	User Interface Design	1h
	Implementation, Integration, and Testing	1h
	Homework	3h
	Total	**28h**
Matteo Pasqual	Introduction	2h
	Architectural Design	13h
	Requirements Traceability	4h
	User Interface Design	1h
	Implementation, Integration, and Testing	5h
	Homework	3h
	Total	**27h**
Matilde Restelli	Introduction	1h
	Architectural Design	13h
	Requirements Traceability	4h
	User Interface Design	5h
	Implementation, Integration, and Testing	1h
	Homework	3h
	Total	**27h**

Table 6.1: Effort spent by each member of the group.