POLITECNICO DI MILANO

Computer Science and Engineering

Design Document

Students&Companies (S&C)

Software Engineering 2 - Project

A.Y. 2024 - 2025

07 January 2025

Version 2.0

Author:
Biagio Fabio Schilirò

Professor:
Matteo Camilli

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This document outlines the **D**esign **D**ocument (***D.D.***) for the Students&Companies System.
Its primary objective is to provide a detailed and precise explanation of the underlying infrastructure, including an high-level overview of the technologies and the components used for the implementation of the system.

This document is designed for development teams responsible for implementing the system's features, providing a thorough guide to ensure consistency and clarity during the development process.

1.1 Purpose

In today's competitive job market, internships have become essential for university students in order to gain practical experience, apply academic knowledge, explore career paths and build professional networks. At the same time, companies continuously seek fresh talent with innovative ideas and up-to-date knowledge.

However, matching students with the right internships remains a challenge: students often struggle to find opportunities aligned with their skills and goals, while companies have to cope with constant difficulties in finding suitable candidates.
Indeed, traditional methods like job boards and career centers frequently are very inefficient, leaving both the sides with unmet expectations.

The **Students&Companies System** (S&C) addresses these challenges by streamlining the internship matchmaking process: it will ensure that students are paired with opportunities that facilitate their growth and will help companies to quickly identify candidates who can add value to their teams.

1.2 Scope

The **Students&Companies System** is designed as a user-friendly web application aimed at streamlining the matching process between students and companies for internship opportunities.
In order to achieve the platform's objectives (*for a detailed overview about them refer to the R.A.S.D. Document - Section 1.1.1*), the system must include the following key functionalities:

→ **Student's Perspective**

1. **Profile Management**: the platform must provide an intuitive interface for students to manage their profiles, with the ability to upload their CV
2. **Internship Search and Filtering**: students should have access to a search feature that will allow them to browse available internship proposals and filter results based on their preferences
3. **Application Management**: sudents must be able to submit applications for internship proposals and manage them, including viewing details, tracking the status (e.g. Sent, Under Review, Selection Process, etc...) and withdrawing if necessary.
4. **Selection Process**: The system should allow students to take part in the selection process by filling out received questionnaires and scheduling interviews in collaboration with the company
5. **FeedBack**: After starting the internship, students must be able to provide feedback on their experience through comments or complaints. Complaints should have a resolution mechanism, enabling students to mark them as resolved once addressed by the company

→ **Company's Perspective**

1. **Internship Proposal Management**: the platform must allow companies to publish internship proposals and manage active proposals, including editing or closing them when positions are no longer available
2. **Application Evaluation**: companies must have the ability to review applications submitted by students and consult the attached CVs to assess suitability for the role. They should also be able to reject applications that do not meet the required criteria or initiate a selection process for suitable candidates
3. **Selection Process**: companies can send questionnaires to candidates as part of the selection process; these questionnaires can be generated with the help of A.I. APIs and, if necessary, modified by the company to better suit their needs. Companies can also propose interview schedules for candidates, coordinating with them through the platform. Before accepting or rejecting a candidate, at least one questionnaire or an interview must have been scheduled and completed as part of the selection process
4. **FeedBack**: After starting the internship, companies can provide comments and complaints on the student's performance. Complaints should have a resolution mechanism, enabling companies to mark them as resolved once addressed by the student.

The system will follow a Three-Tier Architecture, with the presentation tier implemented using the M-V-C (Model-View-Controller) Pattern to enhance modularity and user experience.

Additionally, from the hardware perspective, load balancers and firewalls have been incorporated to optimize system performance and strengthen security.

In the following section it is provided a detailed explanation of these architectural choices and the implementation of each component.

1.3 Definition, Acronyms and Abbreviations

In this section there are all the definitions, acronyms, and abbreviations that will be used in the subsequent discussions and that are essential to be clarified.

1.3.1 Definitions

In this section some key definitions, which may be useful to know before proceeding, are listed.

Term	Definition
Students	One of the two main users of the platform, the ones who are actively looking for internship opportunities.
Companies	The other relevant users of the platform, the ones that offer internship opportunities
Internship	A temporary position that allows students to gain practical experience in a professional setting
Selection Process	The set of activities conducted during the evaluation of students, after their application approval; it includes interview schedulings, assessments and skill tests to determine whether they are qualified to participate in the internship
System	The collection of hardware and software tools that deliver the desired service, referred to here as S&C in its entirety.

Table 1.1: Definitions

1.3.2 Acronyms

In order to avoid any misunderstanding, a list of acronyms used in the following sections is provided in the table below:

Acronyms	Meaning
S&C	Students&Companies
CV	Curriculum Vitae
HTTPS	Hyper Text Transfer Protocol Secure
HTTP	Hyper Text Transfer Protocol
SMTP	Simple Mail Transfer Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
SHA-256	Secure Hash Algorithm
DB	Database
DBMS	Database Management System

Table 1.2: Acronyms

1.3.3 Abbreviations

In this section it is reported the table of the abbreviations used in the document:

Abbreviations	Meaning
R	Requirement
w.r.t.	with reference to
e.g.	exempli gratia
i.e.	id est
etc.	etcetera

Table 1.3: Abbreviations

1.4 Revision History

This section highlights the updates made to the document throughout its compilation process.

Date	Revision	Notes
07/01/2025	v.2.0	Final release

1.5 Reference Documents

The following documents have been indispensable in the creation of this document:

- *S&C RASD Document*
- *Course slides on WeeBeep*
- *RASD assignament document*
- *DD review by Prof. M. Camilli*

1.6 Document Structure

This DD Document is structured as follows:

1. **Introduction**: this section provides a comprehensive overview of the document, outlining the key features and functionalities of the S&C System
2. **Architectural Design**: this section delivers a high-level analysis of the system's functionalities, core responsibilities and principal components.
   It also explains the strategies employed, their impact on the system and highlights the architectural styles and design patterns utilized.
3. **User Interface Design**: this section includes a visual representation of the user interfaces, including the mockups previously introduced in the RASD document and the new implemented ones.
   It also details the functionalities available to students and companies to achieve the S&C Goals
4. **Requirements Traceability**: this section presents a tabular mapping of functional requirements outlined in the RASD document to those considered and analyzed in the Design Document
5. **Implementation, Integration and Test Plan**: in this section the process of implementing the system is described, including how its components are integrated.
   Additionally, it provides a detailed explanation of the testing methods used to validate the system
6. **References**: this section lists the various documents consulted and analyzed during the writing of this document

Chapter 2

Architectural Design

2.1 Overview: High-level components and their integration

The S&C System employs a 3-Tier Architecture, as explained in detail in Section 2.6.
This architectural approach structures the system into three distinct logical layers, each one with a specialized role:

- **Presentation Tier** (User Interface): this layer serves as the primary interaction point for users, delivering an intuitive and accessible interface to facilitate interaction with the system
- **Application Tier** (Processing Layer): acting as the system's core, this layer handles data processing and business logic, ensuring the execution of operations and interactions between the user and the platform
- **Data Tier** (Storage and Management): this layer is responsible for securely storing and managing the data required by the application, enabling efficient retrieval and updates as necessary

An high-level overview of the S&C Architecture is presented in the following image.

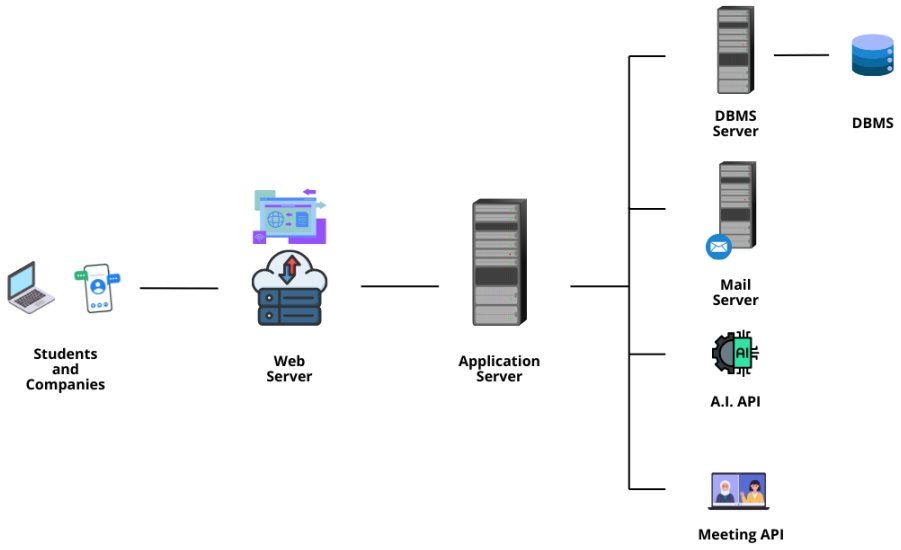

Figure 2.1: S&C Architecture Overview

As illustrated in the image above, users, represented by both students and companies, interact with the system via the **Web Server**, which functions as a middleware layer, bridging the front-end interface with the back-end services.

The Web Server receives user inputs and forwards them to the **Application Server** for processing; the Application Server handles core business logic, allowing interactions with the **Mail Server** and **DBMS Server** and managing communication with external services such as the A.I. API and Meeting APIs.

2.2 Component View

This section represents the system's structure by describing its components and their interactions; it also explains how each component functions, the interfaces they provide and the ways these interfaces enable communication within the system.

The following Component Diagram illustrates the system's structure and organization:

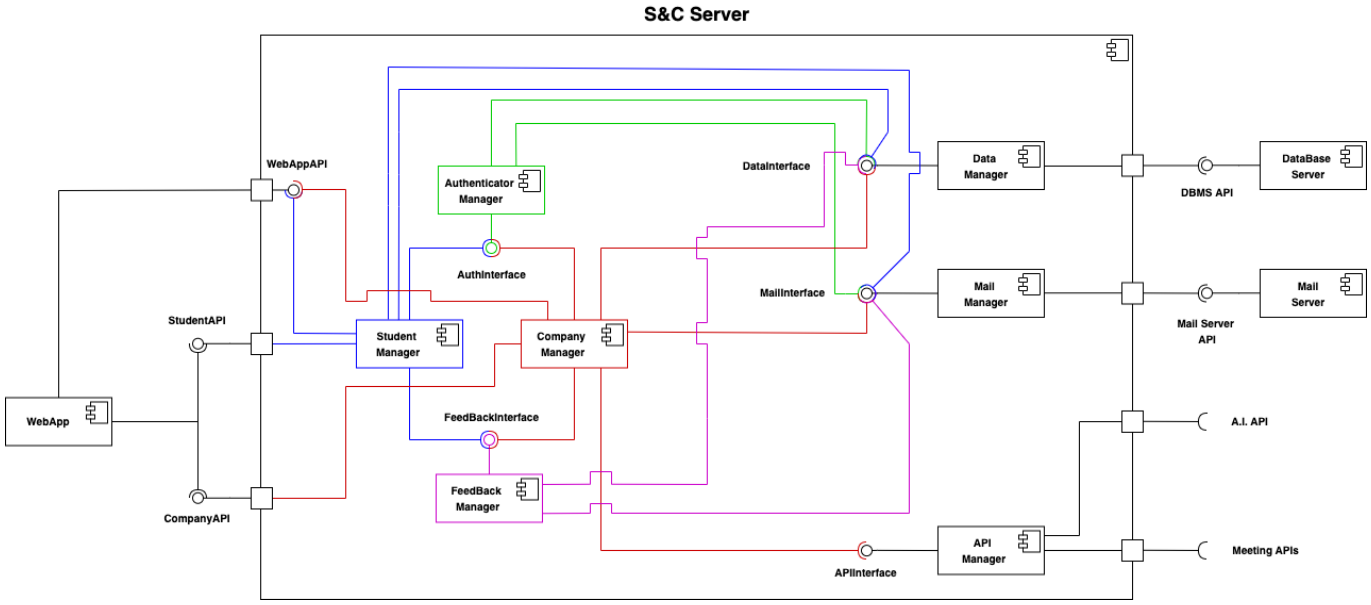

Figure 2.2: S&C Component Diagram

Specifically, the following description outlines the key components of the S&C system and the way in which they interact:

- **WebApp**: this component represents the web application used by both students and companies and serves as the interface for accessing system functionalities via the **StudentAPI** and **CompanyAPI**
- **Authenticator Manager**: it is responsible for managing the login and the registration processes for both students and companies through the **AuthInterface**.
  This component interacts with the **DataInterface** to access stored user information and uses the **MailInterface** to send verification codes during the registration phase

- **Student Manager**: it provides functionalities specifically designed for students. It interacts with the DataManager and MailManager respectively for data handling and notifications and, additionally, it also collaborates with the Feedback Manager via the **FeedbackInterface** to manage feedback-related operations
- **Company Manager**: this component handles the operations available to companies, including accessing data and sending notifications.
  Moreover, it interacts with the API Manager via the **APIInterface** to leverage the A.I. API (for generating interview questions) and the Meeting API (for creating interview links)
- **FeedBack Manager**: it is dedicated to managing the feedback creation, either it is a comment or a complaint. This component also allow companies and students to mark complaints as resolved once addressed.
  Notice that, in order to reduce redundancy, feedback-related operations are centralized here instead of being implemented within the Student Manager and Company Manager
- **Data Manager**: it manages the access to persistent data stored in the database.
  Given its role, this component is a critical dependency for most of the other components in the depicted architecture
- **Mail Manager**: it facilitates the access to the mail server via the **MailInterface**.
  This component enables operations like sending notifications and delivering confirmation codes whenever it is required
- **API Manager**: it handles the interactions with external APIs.
  This dedicated component simplifies the process of integrating future API-related functionalities, ensuring extensibility.
  It currently allow the access to the A.I. API and Meeting API via the **APIInterface**.

2.3 Deployment View

This section presents the deployment view of the system, offering a detailed overview of the physical architecture used to implement the system itself. In particular, it outlines the distribution of components across the nodes and describes the communication protocols that facilitate interaction between these nodes.

As illustrated in the Deployment Diagram (see Figure 2.3), the system is based on a 3-Tier Architecture composed by the WebApp, the S&C Server and the Databases.
Additionally, it also incorporates some supporting components to ensure security, scalability, and robustness.
The tiers and the elements of the system work as follows:

- **WebApp**: the WebApp serves as the front-end layer of the system, handling user interactions by both capturing input and displaying output.
  Functioning as the presentation tier, it runs on the user's web browser and communicates with the application tier via the HTTPS protocol, ensuring secure data transmission
- **Firewalls**: firewalls are employed as network security devices to monitor and regulate data packets entering and leaving the system. Indeed, based on a predefined set of

security rules, they can reject suspicious packets (e.g. potentially malicious ones according to the given rules). The system employs a Multi-Zone Architecture with two types of firewalls:

  - DMZ Firewall: it monitors the requests coming from the WebApp, allowing only those that pass initial security checks to access the system
  - Private Network Firewall: it protects the sensitive data within the private network of the system, ensuring it is only accessible by internal system processes

- **Load Balancer**: the load balancer ensures efficient traffic distribution across the several S&C servers, improving the system's availability, performance and scalability
- **S&C Server**: it represents the application tier, hosting the system's core logic and running on multiple Node JS instances for redundancy and performance. It processes and manages data received from the WebApp (presentation tier), interacts with the data tier through the TCP/IP protocol and communicates with the mail server via the SMTP protocol
- **MySQL Server**: it constitutes the data tier, responsible for securely storing and managing data processed by the application tier

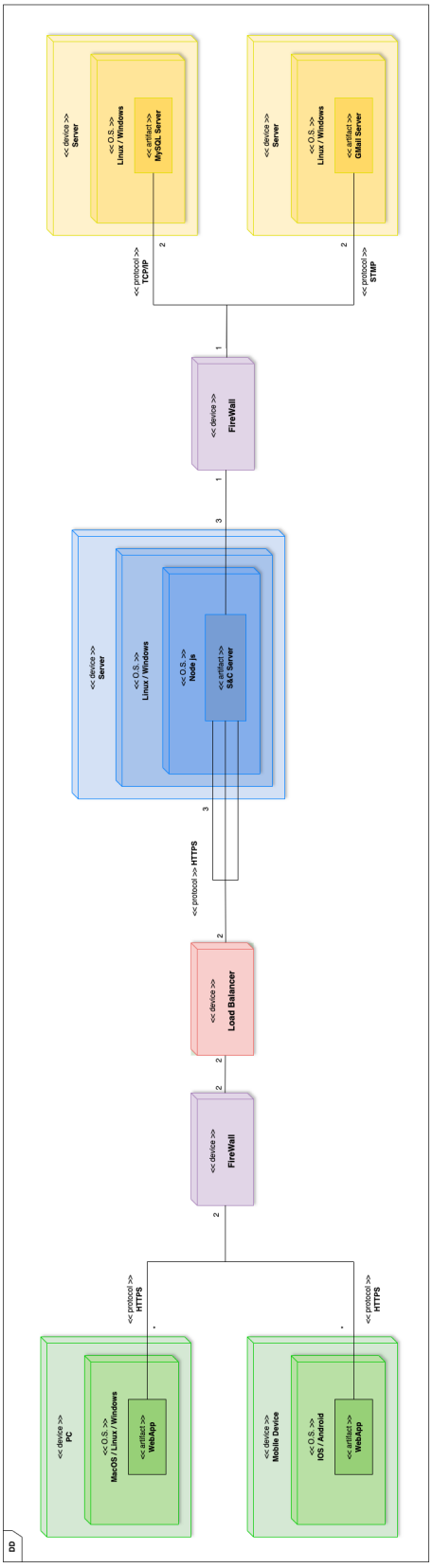

Figure 2.3: S&C Deployment Diagram

2.4 Runtime View

This section describes how components interact dynamically in real time to achieve the desired functionalities

2.4.1 Sign-Up and Log-In

These diagrams represent the interactions required to allow the user registration and the login processes, both for students and companies. The key component in these interactions is the **Authenticator Manager**, which handles all the validation checks and registration phases after receiving the command from either the Student Manager or the Company Manager.

Student Operations

As shown in Figure 2.4, the sequence diagram illustrates the **student registration process**. In this flow, the user submits a form containing all the required information to the WebApp, which will forward this request to the Student Manager. If the student is already registered, he will be notified of the error; otherwise, if the student is not yet registered, the Student Manager interacts with the Mail Manager to send a confirmation code to the student's email. The student is then required to input the received code into the WebApp: if the code is incorrect, an error is displayed, while, if the code is correct, the registration is finalized and the user is redirected to the login page.

The sequence diagram shown in Figure 2.5 illustrates the **student login process**.
The student submits the login form containing the username/email and password to the WebApp, which forwards the request to the Student Manager.
The Student Manager first verifies whether the student is registered or not and, in the case in which the student exists, it also checks the correctness of the provided password.
If all the validation checks are successful, the user is redirected to the corresponding homepage.

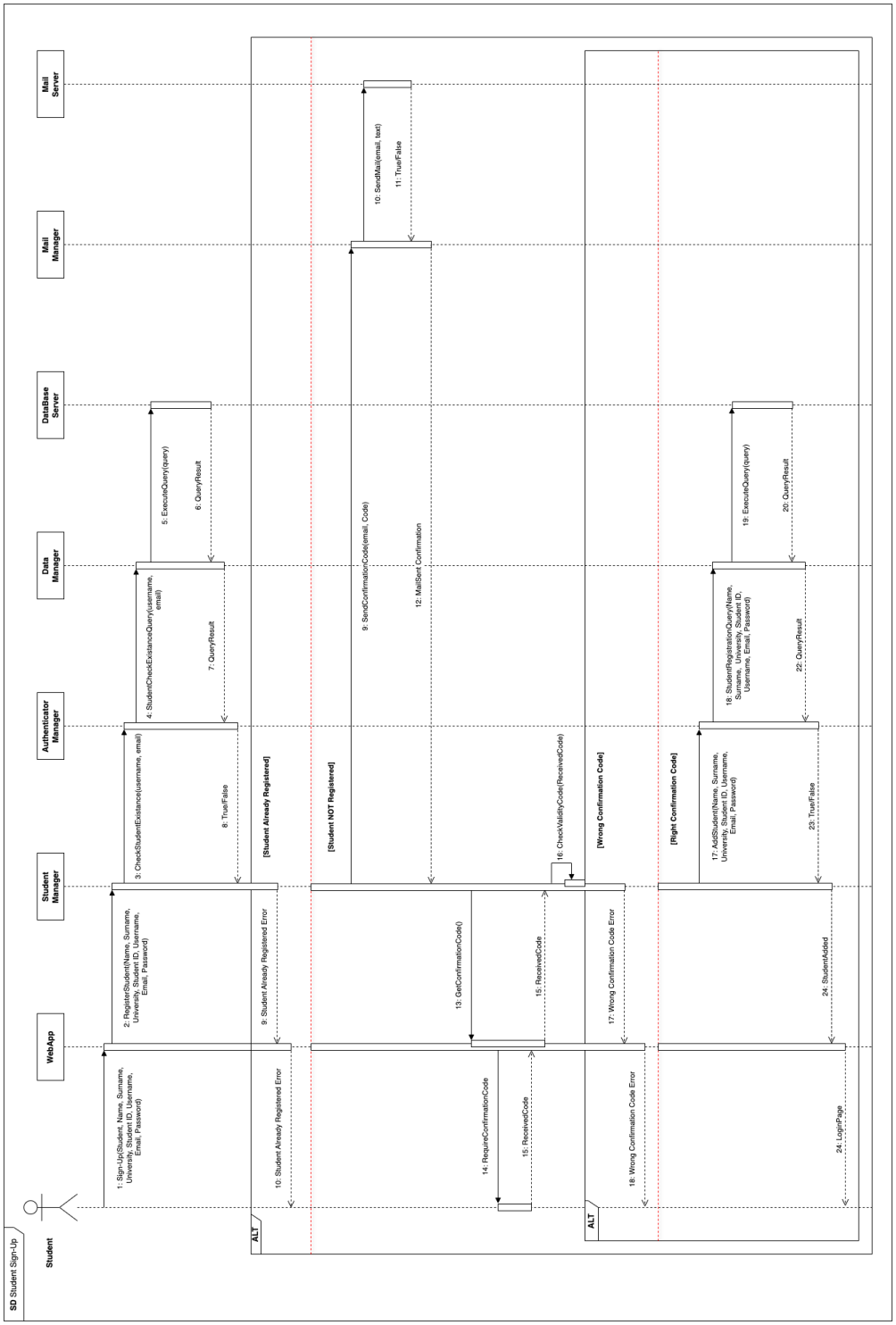

Figure 2.4: Student Registration Sequence Diagram

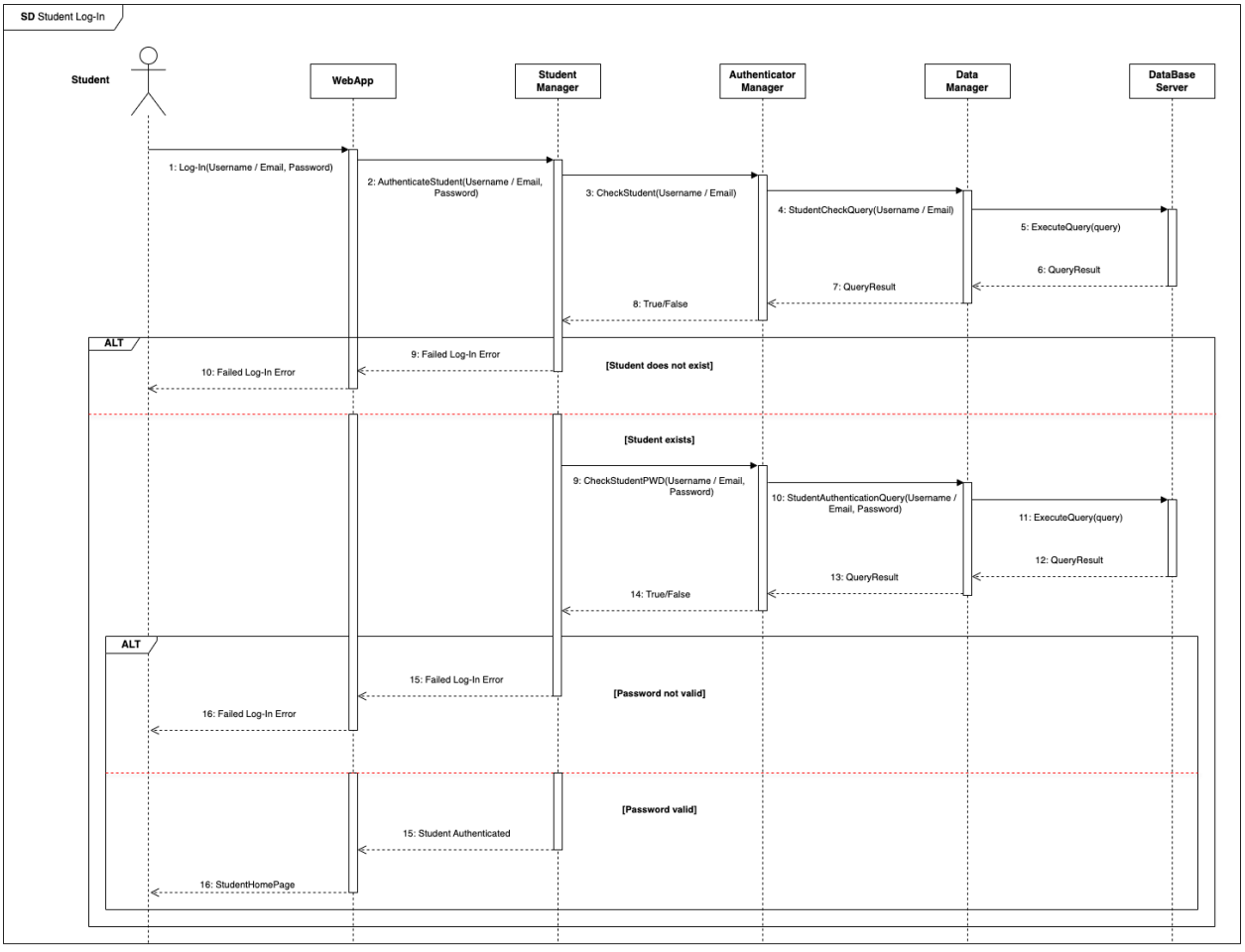

Figure 2.5: Student Login Sequence Diagram

Company Operations

Figures 2.6 and 2.7 depict the sequence diagrams for the registration and login processes of a company.
The main difference compared to the student case lies in the data submitted and the specific component involved, which in this case is the Company Manager.

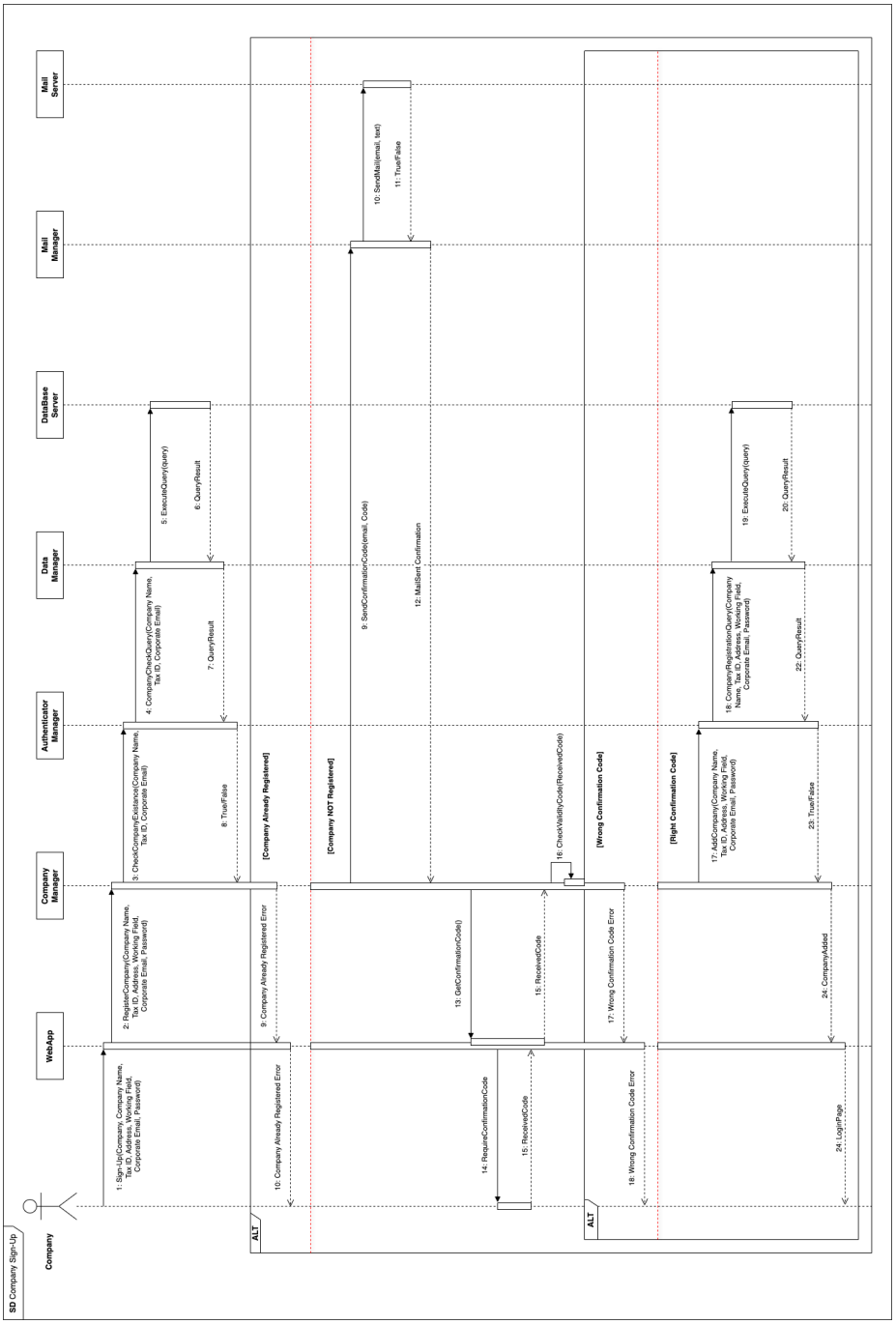

Figure 2.6: Company Login Sequence Diagram

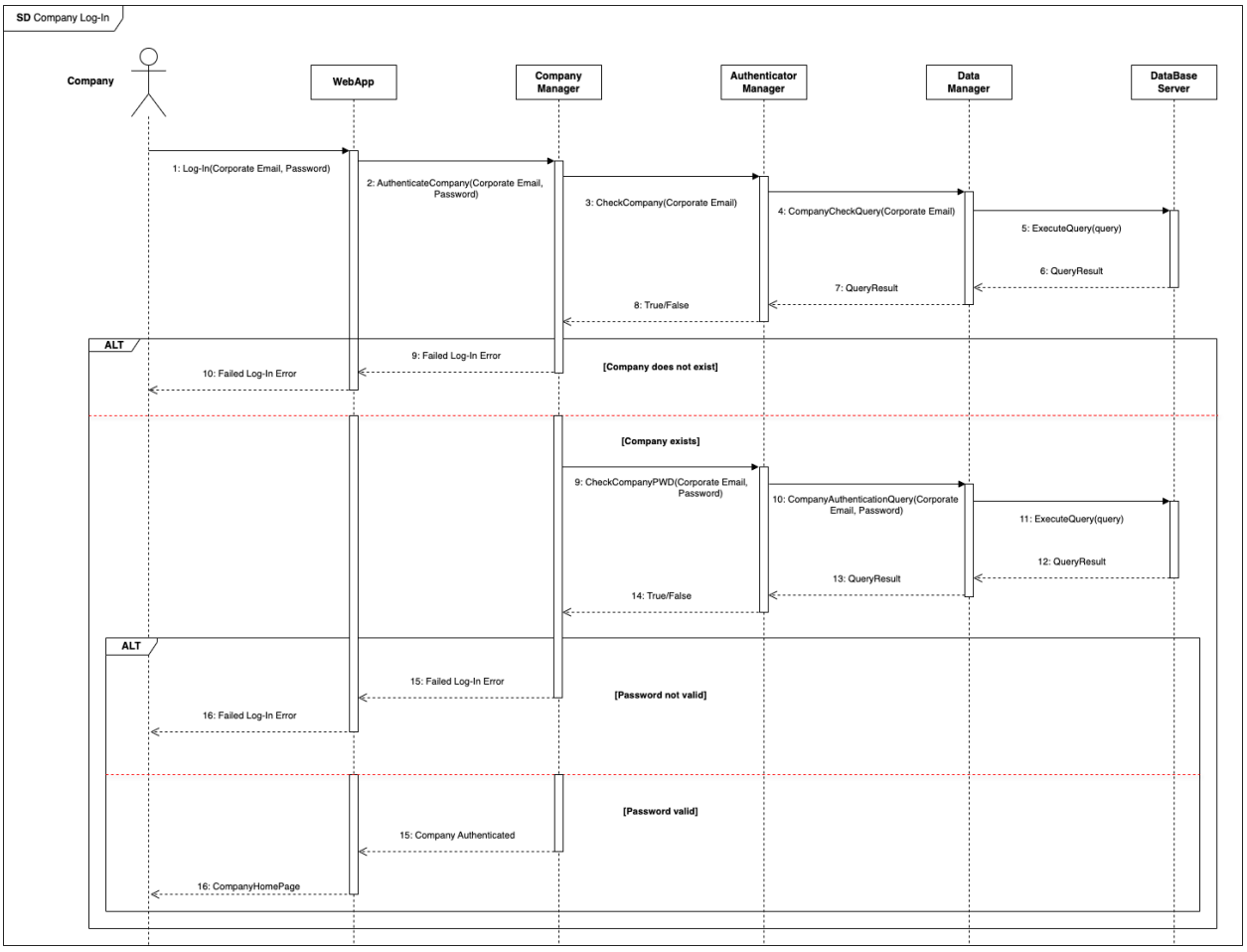

Figure 2.7: Company Login Sequence Diagram

2.4.2 Send an Application

The sequence diagram in Figure 2.8 illustrates the interactions involved in the send application process. First, the student performs a search using specific filters, whose field can also be left empty. If any result is retrieved, the student selects an internship proposal to view its details; once the internship information is displayed, the student proceeds to submit the application. At this point, the Student Manager retrieves the student's CV from the database and attaches it to the application.
After these steps are completed, the application is stored in the system and a confirmation message is displayed to the student.

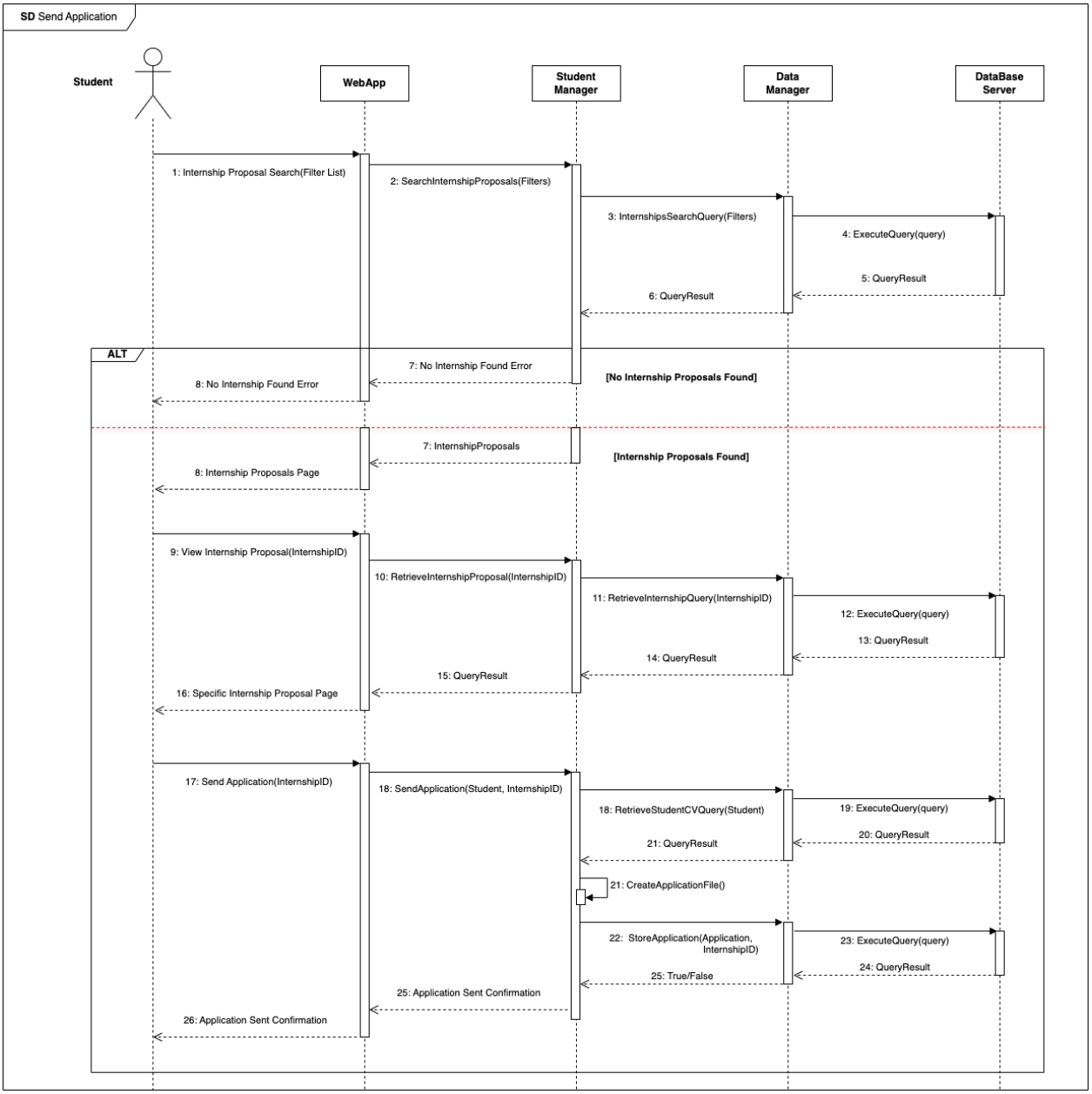

Figure 2.8: Send Application Process Sequence Diagram

2.4.3 Manage Applications

The sequence diagram in Figure 2.9 illustrates the process of managing student-submitted applications. Specifically, the student can access to the list of all the submitted applications and, if desired, view detailed information about a specific application (e.g. its status, a description of the associated internship, etc...). If the student chooses to view a specific application, he also have the option to delete it, provided that the application has not been marked as "Under Review" by the company.

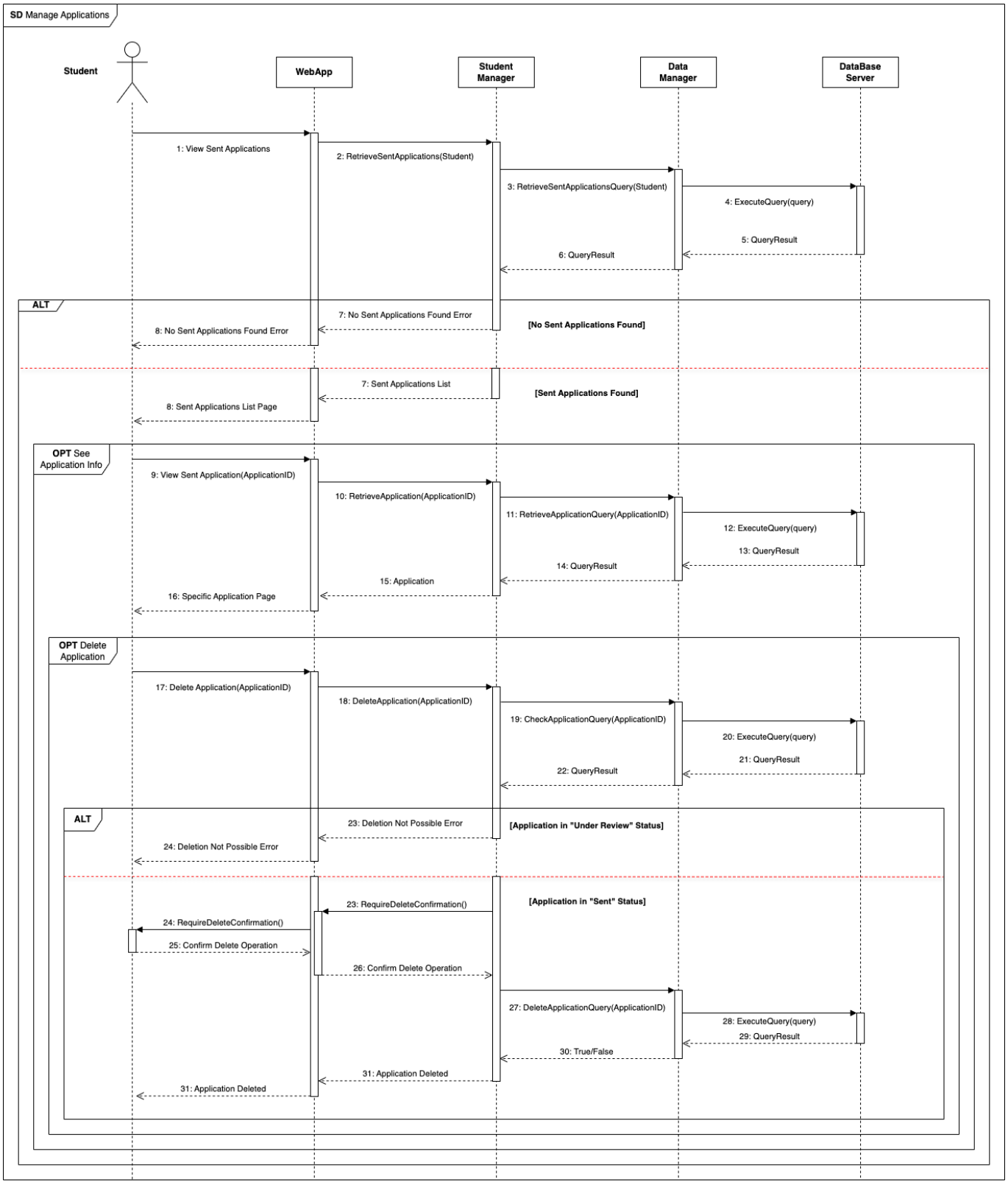

Figure 2.9: Manage Applications Process Sequence Diagram

2.4.4 Sustain the Selection Process

Figure 2.10 depicts the sequence diagram related to the student selection process.
The process involves the student retrieving a specific application and choosing between two optional actions:

- Fill Out a Questionnaire: the system retrieves the questionnaire associated with the application and, if the questionnaire has expired, it notifies via mail the company that the student has accessed an expired questionnaire. If the questionnaire is valid, the student accesses it, writes the answers and submits the responses; the system then notifies the company that the questionnaire has been successfully filled out.
- Schedule an Interview: the student views the details of a proposed interview, including the scheduled date. If the student is unavailable on the proposed date, he can request a rescheduling, notifying the company of the change request. If the student agrees to the proposed date, he send an acceptance and the company is notified of the confirmation.

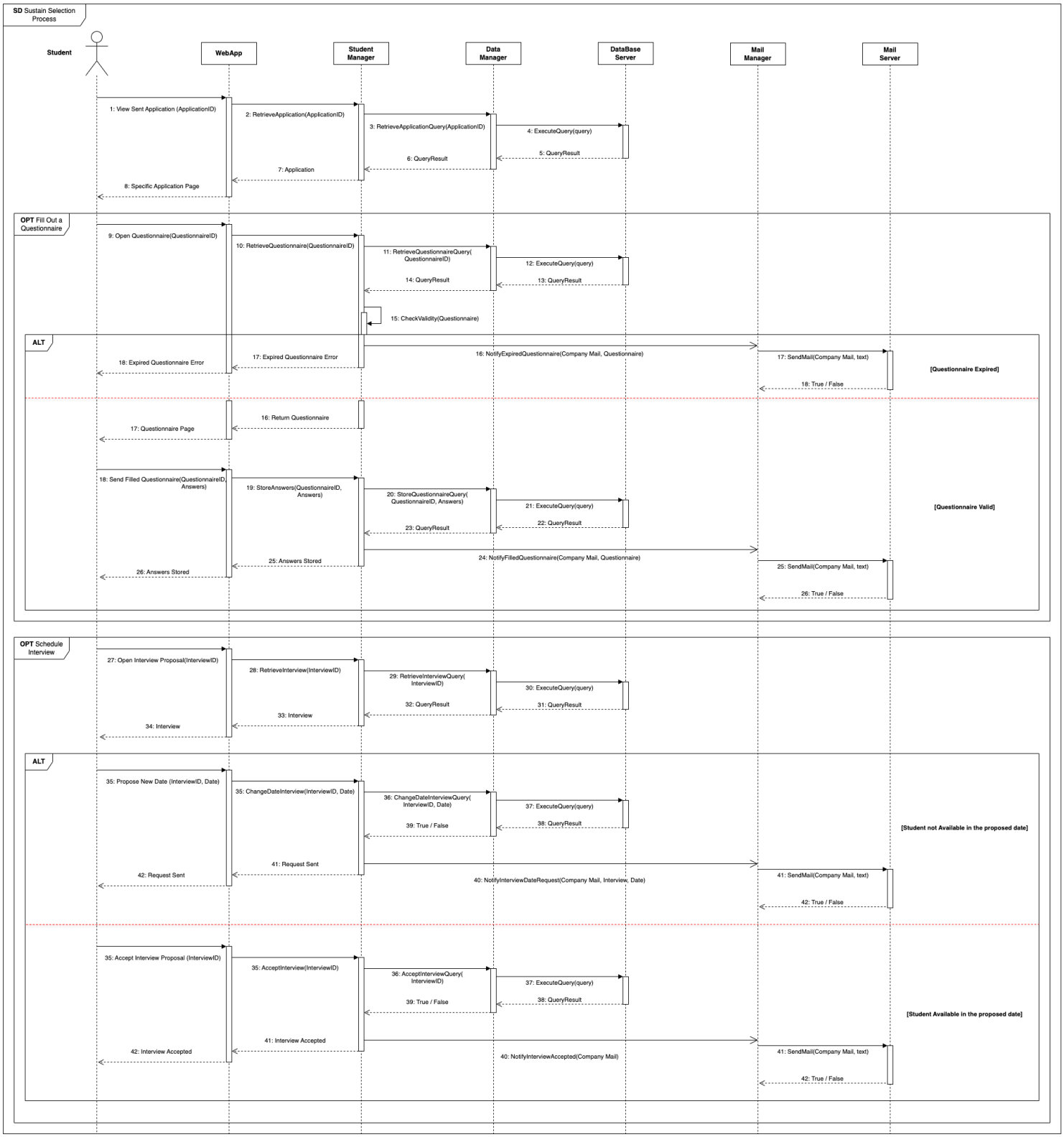

Figure 2.10: Sustain Selection Process Sequence Diagram

2.4.5 Write an Internship FeedBack

The sequence diagram in Figure 2.11 illustrates the process of submitting feedback for an ongoing internship experience.
Specifically, the student retrieves all feedback related to the internship, considering the contributions from both the company and themselves, and then can perform two actions:

- Write a Comment: the system validates the comment to ensure it does not contain not allowed words and, if the comment is valid, it is stored in the system; moreover, the company is notified via email
- Submit a Complaint: similar to the comment process, the system validates the complaint before storing it. Once validated, the complaint is saved and the company is notified. Additionally, the student can mark the complaint as solved if the company has addressed the issue.

It is important to note that the process is essentially identical in the case of companies, with the necessary adjustments to reflect the context (e.g. the use of the Company Manager Component instead of the Student Manager one).

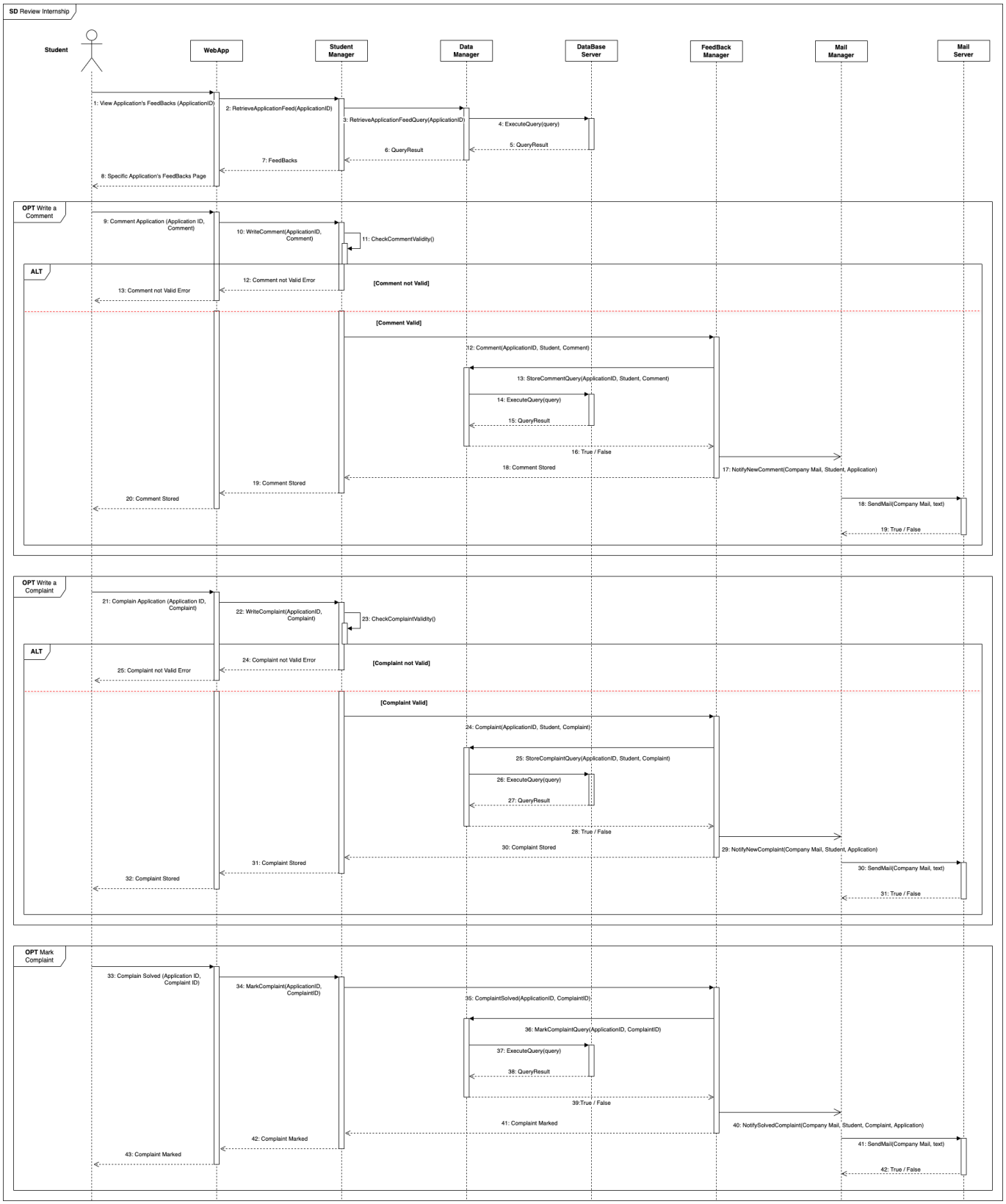

Figure 2.11: Review Internship Process Sequence Diagram

2.4.6 Create a Questionnaire

Figure 2.12 illustrates the sequence diagram for the questionnaire creation process.
The company starts an iterative interaction with the A.I. API, providing a prompt that specifies the command to generate specific questions to test the student's skills; in response, the API returns a list of questions.
Once the company considers the list satisfactory, it has the option to review and modify the questions as needed.
After finalizing the content, the questionnaire is submitted to the student.

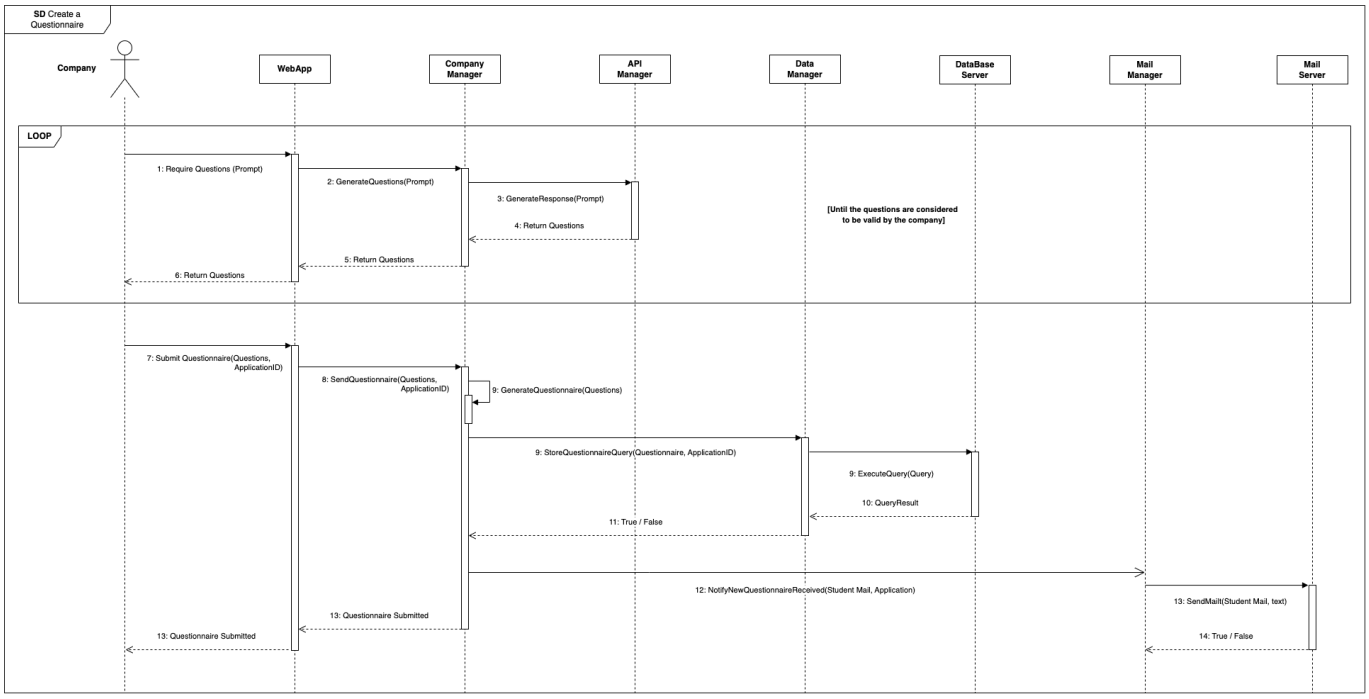

Figure 2.12: Create Questionnaire Process Sequence Diagram

2.4.7 Generate Interview Proposal and Interview Link

The sequence diagram in Figure 2.13 illustrates the process of generating an interview link or proposal.
Initially, the company reviews the previously sent interview proposal: if the student has sent a new date request, the company responds by sending an updated proposal (either choosing the suggested date or proposing another one).
Alternatively, if the student has accepted the proposal, the company requires the system to generate a meeting link using the selected Meeting Service; the system then notifies the student that the link has been generated and provides the link to the company.

It is worth noting that the initial process of sending the interview proposal is not explicitly addressed in this diagram, as it is functionally equivalent to the scenario where the student has requested a new interview date.

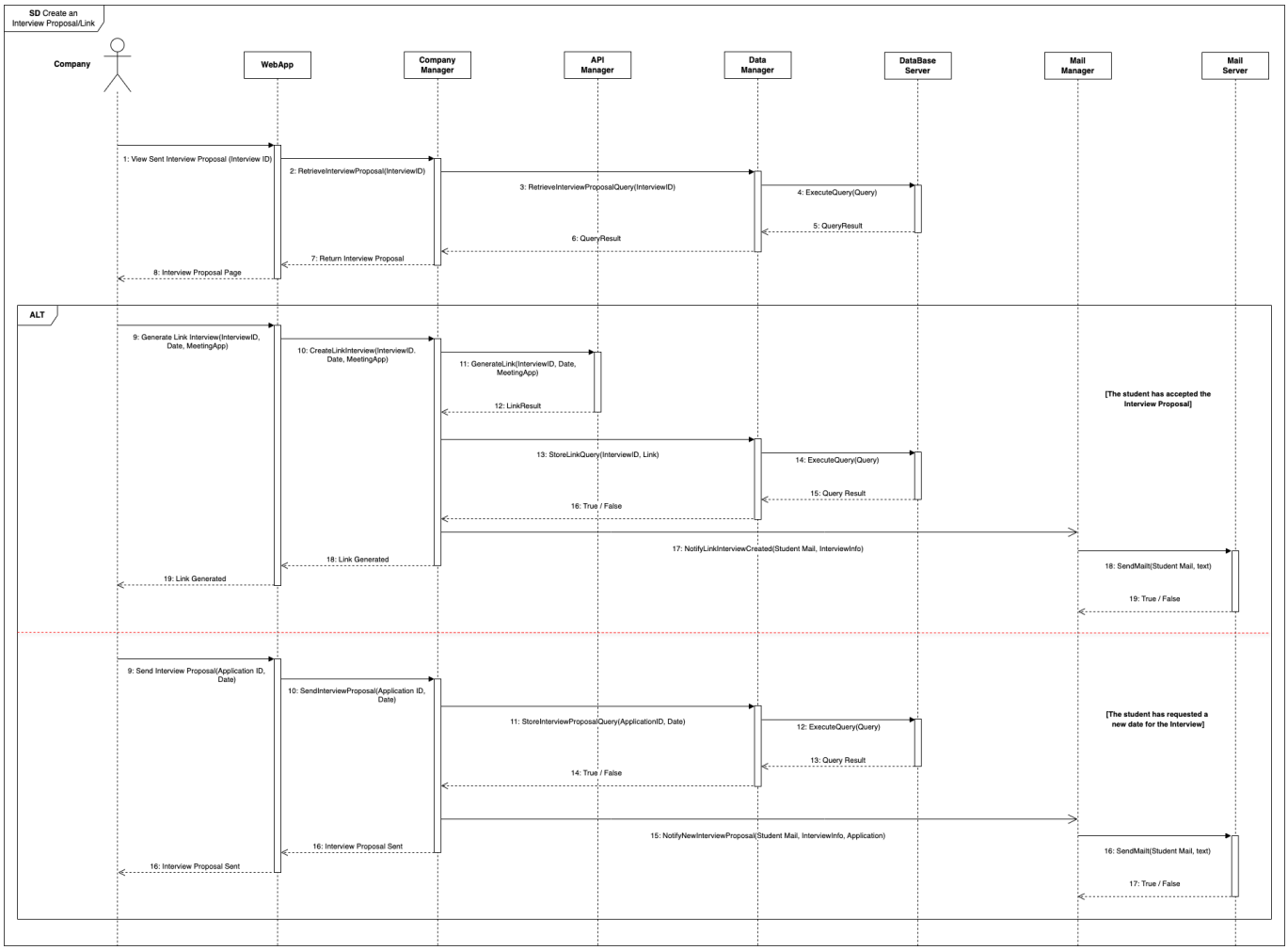

Figure 2.13: Generate Interview Proposal and Link Sequence Diagram

2.4.8 Manage an Application Request

Figure 2.14 illustrates the sequence diagram for the application evaluation process conducted by a company.
The process begins with the company retrieving the list of applications submitted for a specific internship proposal: if at least one application exists, the company can set its status to "Under Review", provided that the student has not withdrawn the application, as this operation becomes unavailable in such cases. The company can then proceed to evaluate the application, either immediately or at a later time: during the evaluation, the company retrieves the student's CV and updates the application status accordingly.
Once the status is changed, the system notifies the student of the update.

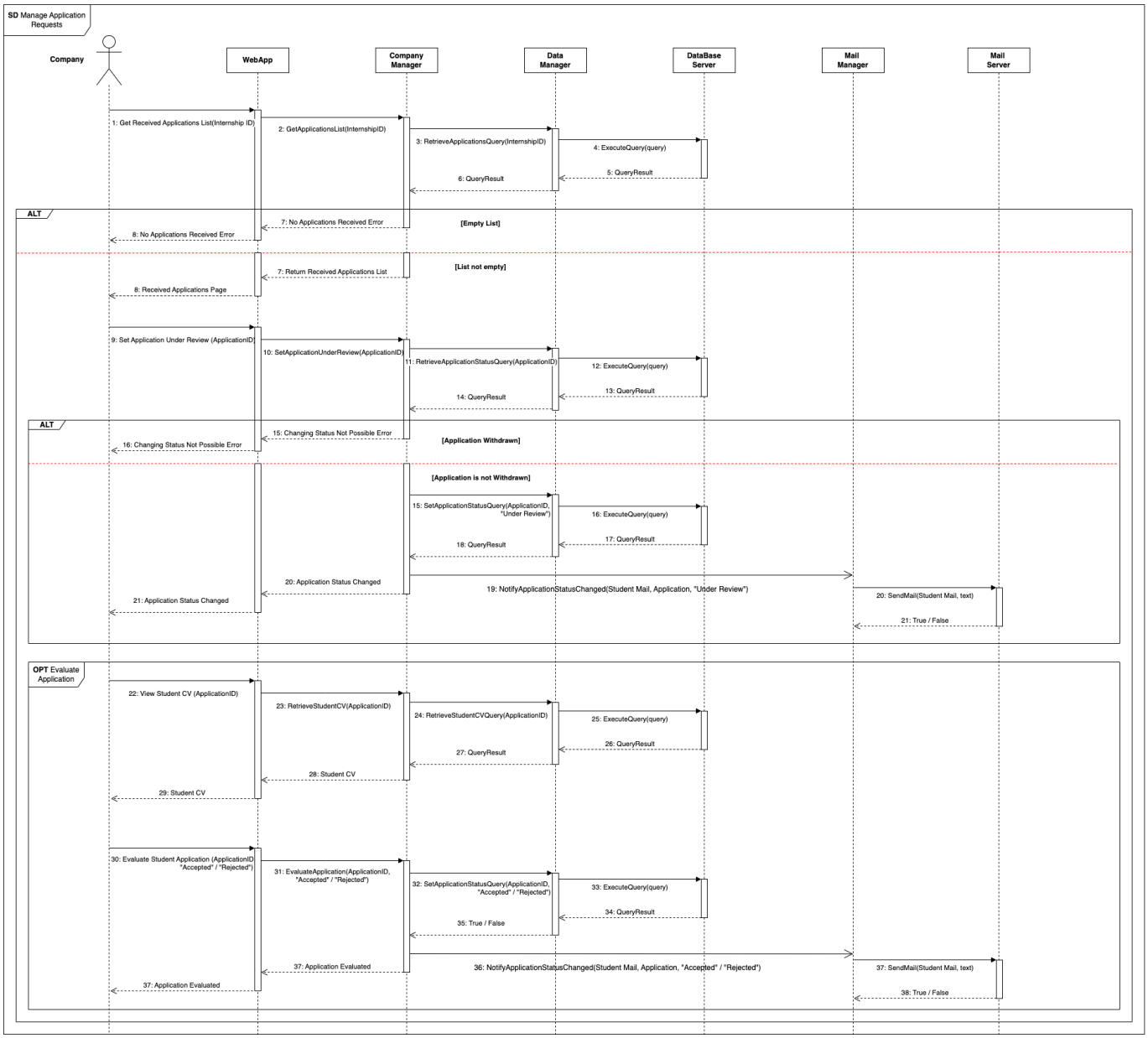

Figure 2.14: Manage Application Request Sequence Diagram

2.5 Component Interfaces

In this section, the methods offered by each Component Interface are detailed: the methods define the functionality exposed by the respective components, enabling interaction and integration within the system.

2.5.1 Authenticator Manager Component

The methods provided by the Authenticator Manager Component are the followings:

- **CheckStudentExistance(Username, Email)**
  This method is used during the student registration process to verify whether a student with the given username and email already exists in the system.
  It returns a boolean value:

  - True: Indicates that the student is already registered
  - False: Indicates that no such student exists, allowing the registration to proceed

- **CheckStudent(Username** / **Email)**
  This method is used during the student login process to verify whether the provided username or email exists in the system.
  It returns a boolean value:

  - True: Indicates that the username or email is present in the system, allowing the login process to proceed
  - False: Indicates that no such username/mail exists

- **CheckStudentPWD(Username** / **Email, Password)**
  This method is used during the student login process to verify whether the provided password matches the username or email provided.
  It returns a boolean value:

  - True: Indicates that the password is valid and the login process can proceed
  - False: Indicates that the password does not match the provided username or email

- **CheckCompanyExistance(CompanyName, TaxID, CorporateEmail)**
  This method is used during the company registration process to verify whether a company with the given company name, tax ID and corporate email already exists in the system.
  It returns a boolean value:

  - True: Indicates that the company is already registered
  - False: Indicates that no such company exists, allowing the registration to proceed

- **CheckCompany(CorporateEmail)**
  This method is used during the company login process to verify whether the provided corporate email exists in the system.
  It returns a boolean value:

  - True: Indicates that the corporate email is present in the system, allowing the login process to proceed
  - False: Indicates that no such corporate mail exists

- **CheckCompanyPWD(CorporateEmail, Password)**
  This method is used during the company login process to verify whether the provided password matches the corporate email provided.
  It returns a boolean value:

  - True: Indicates that the password is valid and the login process can proceed
  - False: Indicates that the password does not match the provided username or email

- **AddStudent(Name, Surname, University, StudentID, Username, Email, Password)**
  This method is used during the student registration process to create and store the student's profile in the system.
  It returns a boolean value:

  - True: Indicates that the operation was successfully executed and the student profile has been added to the system
  - False: Indicates that the operation failed and the student profile has not been added

- **AddCompany(CompanyName, taxID, Address, WorkingField, CorporateEmail, Password)**
  This method is used during the company registration process to create and store the company's profile in the system.
  It returns a boolean value:

  - True: Indicates that the operation was successfully executed and the company profile has been added to the system
  - False: Indicates that the operation failed and the company profile has not been added

2.5.2 Data Manager Component

The methods provided by the Data Manager Component are the followings:

- **StudentCheckExistanceQuery(Username, Mail)**
  This method is used during the student registration process to construct and execute a query that checks whether a student with the specified username and email already exists in the system.
  It returns the result of the query

- **StudentCheckQuery(Username / Mail)**
  This method is used during the student login process to construct and execute a query that checks whether the provided username or email exists in the system.
  It returns the result of the query

- **StudentAuthenticationQuery(Username / Mail, Password)**
  This method is used during the student login process to construct and execute a query that checks whether the provided password matches the provided username or email.
  It returns the result of the query

- **StudentRegistrationQuery(Name, Surname, University, StudentID, Username, Email, Password)**
  This method is used during the student registration process to construct and execute a query that inserts the student's profile into the system.
  It returns the result of the query

- **CompanyCheckExistanceQuery(CompanyName, TaxID, CorporateMail)**
  This method is used during the company registration process to construct and execute a query that checks whether a company with the specified company name, tax ID and corporate email already exists in the system.
  It returns the result of the query

- **StudentCheckQuery(Corporate Mail)**
  This method is used during the company login process to construct and execute a query that checks whether the provided corporate email exists in the system.
  It returns the result of the query

- **CompanyAuthenticationQuery(Corporate Mail, Password)**
  This method is used during the company login process to construct and execute a query that checks whether the provided password matches the provided corporate email.
  It returns the result of the query

- **CompanyRegistrationQuery(CompanyName, TaxID, Address, WorkingField, CorporateEmail, Password)**
  This method is used during the company registration process to construct and execute a query that inserts the company's profile into the system.
  It returns the result of the query

- **InternshipSearchQuery(Filters = none)**
  This method is used to construct and execute a query for retrieving internship proposals based on the student's search criteria.
  The student may optionally specify a list of filters to tailor the search; the method constructs the query according to the provided filters and executes it. It returns the result of the query:

  - If filters are provided: Returns a list of internship proposals that match the specified criteria
  - If no filters are specified: Returns all available internship proposals

- **RetrieveInternshipQuery(InternshipID)**
  This method is used to construct and execute a query for retrieving all the information related to an Internship ID.
  It returns the result of the query

- **RetrieveStudentCVQuery(Student)**
  This method is used to construct and execute a query for retrieving the CV associated to a specific student.
  It returns the result of the query

- **StoreApplicationQuery(Application, InternshipID)**
  This method is used to construct and execute a query for storing the application sent by a student for a specific internship proposals.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the application has been stored into the system
  - False: Indicates that the operation failed and the application has not been stored

- **RetrieveSentApplicationsQuery(Student)**
  This method is used to construct and execute a query for retrieving the list of applications sent by a specific student.
  It returns the list of sent applications

- **RetrieveApplicationQuery(ApplicationID)**
  This method is used to construct and execute a query for retrieving all the information related to an Application ID.
  It returns the result of the query

- **CheckApplicationQuery(ApplicationID)**
  This method is used to construct and execute a query to check whether an application has been set to the "Under Review" status by the company.
  It returns a boolean value:
  - True: Indicates that the status is "Under Review" and the deletion is not possible anymore
  - False: Indicates that the status is still "Sent" and the delete operation is possible

- **DeleteApplicationQuery(ApplicationID)**
  This method is used to construct and execute a query for deleting an Application.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the application has been deleted
  - False: Indicates that the operation failed and the application has not been deleted

- **RetrieveQuestionnaireQuery(QuestionnaireID)**
  This method is used to construct and execute a query for retrieving a specific questionnaire received by the company.
  It returns the result of the query

- **StoreQuestionnaireQuery(QuestionnaireID, Answers)**
  This method is used to construct and execute a query for storing the application sent by a student for a specific received questionnaire.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the answers have been stored into the system
  - False: Indicates that the operation failed and the answers have not been stored

- **RetrieveInterviewQuery(InterviewID)**
  This method is used to construct and execute a query for retrieving the information related to a received Interview Proposal sent by the company
  It returns the result of the query

- **ChangeDateInterviewQuery(InterviewID)**
  This method is used to construct and execute a query for storing a change date request associated to a specific interview proposal.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the request has been stored into the system
  - False: Indicates that the operation failed and the request has not been stored

- **AcceptInterviewQuery(InterviewID)**
  This method is used to construct and execute a query for storing the student's acceptance to a specific interview proposal.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the answer has been stored into the system
  - False: Indicates that the operation failed and the answer has not been stored

- **RetrieveApplicationFeedQuery(ApplicationID)**
  This method is used to construct and execute a query for retrieving all the feedback written by both the company and the student about an on-going internship experience.
  It returns the list of the feedback (i.e. comments and complaints)

- **StoreCommentQuery(ApplicationID, Student / Company, Comment)**
  This method is used to construct and execute a query for storing a comment made by a student or a company about the on-going internship experience
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the comment has been stored into the system
  - False: Indicates that the operation failed and the comment has not been stored

- **StoreComplaintQuery(ApplicationID, Student / Company, Complaint)**
  This method is used to construct and execute a query for storing a complaint made by a student or a company about the on-going internship experience
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the complaint has been stored into the system
  - False: Indicates that the operation failed and the complaint has not been stored

- **MarkComplaintQuery(ApplicationID, ComplaintID)**
  This method is used to construct and execute a query for marking an already existing complaint as solved.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the complaint has been marked as solved
  - False: Indicates that the operation failed and the complaint has not been marked as solved

- **StoreQuestionnaireQuery(Questionnaire, ApplicationID)**
  This method is used to construct and execute a query for storing a questionnaire made by a company suring the selection process.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the questionnaire has been stored into the system
  - False: Indicates that the operation failed and the questionnaire has not been stored

- **RetrieveInterviewProposalQuery(InterviewID)**
  This method is used to construct and execute a query for retrieving the information related to a sent Interview Proposal.
  It returns the result of the query

- **StoreLinkQuery(InterviewID, Link)**
  This method is used to construct and execute a query for storing the link created for a specific interview proposal.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the link has been stored into the system
  - False: Indicates that the operation failed and the link has not been stored

- **StoreInterviewProposalQuery(ApplicationbID, Date)**
  This method is used to construct and execute a query for storing an interview proposal (with the relared proposed date) associated to an existing Application.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the interview proposal has been stored into the system
  - False: Indicates that the operation failed and the interview proposal has not been stored

- **RetrieveApplicationsQuery(InternshipID)**
  This method is used to construct and execute a query for retrieving the list of applications received for a specific internship proposal.
  It returns the list of received applications

- **RetrieveApplicationStatusQuery(ApplicationID)**
  This method is used to construct and execute a query for retrieving the status of a given application.
  It returns the application's status

- **RetrieveStudentCVQuery(ApplicationID)**
  This method is used to construct and execute a query for retrieving the student's CV appended to a specific received application.
  It returns the student's CV

- **SetApplicationStatusQuery(ApplicationbID, Status)**
  This method is used to construct and execute a query for changing the status of an existing application.
  The option for the Status parameter are "Under Review", Selection Process", "Accepted", "Rejected", "Internship".
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the status has been changed
  - False: Indicates that the operation failed and the status has not been changed

2.5.3 DataBase Server Component

The methods provided by the DataBase Server Component are the followings:

- **ExecuteQuery(Query)**
  This method is used to execute a provided query on the database.
  It returns the result of the query

2.5.4 Mail Manager Component

The methods provided by the Mail Manager Component are the followings:

- **SendConfirmationCode(Email, Code)**
  This method is used to send a mail to a provided mail address: it will create the body of the mail, including the code itself, and forward the command to the mail server.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the mail has been sent
  - False: Indicates that the operation failed and the mail has not been sent

- **NotifyExpiredQuestionnaire(CorporateEmail, Questionnaire)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that a specific questionnaire was not completed by the student within the required timeframe and the student attempted to access it; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyInterviewDateRequest(CorporateEmail, Interview, Date)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the student has required a reschedule of an interview proposal and sent a suggested date; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyInterviewAccepted(CorporateEmail, Interview)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the student has accepted the sent interview proposal; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyNewComment(Email, User, Application)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the User has added a new comment to the provided application; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyNewComplaint(Email, User, Application)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the User has added a new complaint to the provided application; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifySolvedComplaint(Email, User, Application, Complaint)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the User has marked the complaint submitted to the provided application as solved; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyNewQuestionnaire(StudentEmail, Application)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the student has received a new questionnaire to fill out for a specific application; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyLinkInterviewCreated(StudentEmail, InterviewInfo)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the company has created the link for the interview and providing all the interview information as remainder; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyNewInterviewProposal(StudentEmail, InterviewInfo, Application)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the company has sent a new interview proposal for a specific application and providing the interview information; the method then forwards the email to the mail server for delivery.
  It does not return any value.

- **NotifyApplicationStatusChanged(StudentEmail, Application, Status)**
  This method is used to send a mail to a company mail address: it will create the body of the email, highlighting that the company has changed the status of a given application to the parameter 'Status'; the method then forwards the email to the mail server for delivery.
  It does not return any value.

2.5.5 Mail Server Component

The methods provided by the DataBase Server Component are the followings:

- **SendMail(Email, Text)**
  This method is used to send a mail to a provided mail address using as body the provided text.
  It returns a boolean value:
  - True: Indicates that the operation was successfully executed and the mail has been sent
  - False: Indicates that the operation failed and the mail has not been sent

2.5.6 API Manager Component

The methods provided by the API Manager Component are the followings:

- **GenerateLink(Date, MeetingApp)**
  This method is used to generate an interview link using a specific MeetingAPI chosen by the company based on its preference. The link is created for the specified date.
  It returns the created interview link

- **GenerateResponse(Prompt)**
  This method is used to generate a set of questions by interacting with the AI API; the questions are created based on the prompt provided by the company.
  It returns the set of generated questions

2.5.7 Student Manager Component

The methods provided by the Student Manager Component are the followings:

- **RegisterStudent(Name, Surname, University, StudentID, Username, Email, Password)**
  This method is used to perform the student registration process, including all the necessary validation checks and storing the student's information in the system.
  It returns either a confirmation of success (i.e. if the registration process is completed without any issues) or an error message (i.e. if any of the checks fails)
- **AuthenticateStudent(Username / Email, Password)**
  This method is used to perform the student login process, including all the necessary validation checks.
  It returns either a confirmation of success (i.e. if the registration process is completed without any issues) or an error message (i.e. if any of the checks fails)
- **SearchInternshipProposals(Filters = none)**
  This method is used to retrieve a list of internship proposals available in the system; the student can optionally provide filters to narrow down the search based on specific criteria.
  It returns either the list of internship proposals (i.e. if at least one proposal matches the search criteria) or an error message (i.e. if no internship proposals are found)
- **RetrieveInternshipProposal(InternshipID)**
  This method is used to retrieve all the information related to a specific internship proposal.
  It returns either the retrieved information or an error message (i.e. if the information has not been successfully retrieved)
- **SendApplication(Student, InternshipID)**
  This method is used to submit an application for a specific internship proposal; the process also includes attaching the student's CV to the application.
  It returns either a confirmation of successful submission (i.e. if the application is successfully processed) or an error message (i.e. if the application process fails)
- **RetrieveSentApplications(Student)**
  This method is used to retrieve all the applications sent by a given student.
  It returns either the list of sent applications (i.e. if at least one application has been sent) or an error message (i.e. if no application has been sent by the given student)
- **RetrieveApplication(ApplicationID)**
  This method is used to retrieve all the information related to a specific sent application.
  It returns either the retrieved information or an error message (i.e. if the information has not been successfully retrieved)

- **DeleteApplication(ApplicationID)**
  This method is used to delete a sent application, including all the required checks to verify whether the operation is allowed or not.
  It returns either confirmation of successful deletion or an error message (i.e. if the application has been set to the "Under Review" status by the company)

- **RetrieveQuestionnaire(QuestionnaireID)**
  This method is used to retrieve a specific questionnaire sent to the student by the company.
  It returns either the questionnaire or an error message (i.e. if the questionnaire is expired - In this case a notification mail is sent to the related company)

- **StoreAnswers(QuestionnaireID, Answers)**
  This method is used to submit the student's answers to a specific questionnaire sent to the student by the company; additionally, a notification mail is sent to the company.
  It returns either the confirmation of successful submission or an error message (i.e. if the answers have not been successfully stored)

- **RetrieveInterview(InterviewID)**
  This method is used to retrieve a specific Interview Proposal sent to the student by the company.
  It returns either the Interview Proposal or an error message (i.e. if the Interview Proposal has not been retrieved)

- **ChangeDateInterview(InterviewID, Date)**
  This method is used to require a change in the date of a specific Interview Proposal, sending also a suggested possible date. A notification mail will also be sent to the company.
  It returns either the confirmation of successful submission or an error message (i.e. if the Change Date Request has not been successfully sent)

- **AcceptInterview(InterviewID)**
  This method is used to accept an Interview Proposal sent to the student by the company. A notification mail will also be sent to the company.
  It returns either the confirmation of the operation or an error message (i.e. if the acceptance of the request has not been successfully sent)

- **RetrieveApplicationFeed(ApplicationID)**
  This method is used to retrieve all the feedback submitted for a specific application (i.e. the one in the "Internship" status) both by the student and the company.
  It returns either the list of submitted feedback or an error message (i.e. if feedback has been retrieved)

- **WriteComment(ApplicationID, Comment)**
  This method is used to submit a comment for a specific application in the "Internship" status; the process includes validating the comment and storing it within the system. Additionally, a notification mail will also be sent to the interested counterpart.
  It returns either the confirmation of submission or an error message (i.e. invalid input or system errors)

- **WriteComplaint(ApplicationID, Complaint)**
  This method is used to submit a complaint for a specific application in the "Internship" status; the process includes validating the complaint and storing it within the system. Additionally, a notification mail will also be sent to the interested counterpart.
  It returns either the confirmation of submission or an error message (i.e. invalid input or system errors)

- **MarkComplaint(ApplicationID, ComplaintID)**
  This method is used to mark a previously submitted complaint as "Solved". Additionally, a notification mail will also be sent to the interested counterpart.
  It returns either the confirmation of the operation or an error message (i.e. the complaint has not been marked as solved)

2.5.8 Company Manager Component

The methods provided by the Company Manager Component are the followings:

- **RegisterCompany(CompanyName, TaxID, Address, WorkingField, CorporateEmail, Password)**
  This method is used to perform the company registration process, including all the necessary validation checks and storing the company's information in the system.
  It returns either a confirmation of success (i.e. if the registration process is completed without any issues) or an error message (i.e. if any of the checks fails)

- **AuthenticateCompany(CorporateEmail, Password)**
  This method is used to perform the company login process, including all the necessary validation checks.
  It returns either a confirmation of success (i.e. if the registration process is completed without any issues) or an error message (i.e. if any of the checks fails)

- **GenerateQuestions(Prompt)**
  This method is used to generate a set of questions by using the AI API, following the instructions specified by the company through the provided prompt.
  It returns either the list of generate questions or an error message (i.e. the questions have not been correctly generated)

- **SendQuestionnaire(Questions, ApplicationID)**
  This method is used to send a questionnaire, consisting of a set of questions, to a student whose application is in the "Selection Process" status. Additionally, a notification mail will also be sent to the student.
  It returns either the the confirmation of questionnaire submission or an error message (i.e. the questionnaire has not been successfully sent)

- **RetrieveInterviewProposal(InterviewID)**
  This method is used to retrieve a specific sent Interview Proposal.
  It returns either the Interview Proposal or an error message (i.e. if the Interview Proposal has not been retrieved)

- **CreateLinkInterview(InterviewID, Date, MeetingApp)**
  This method is used to generate an interview link using the API associated with the

chosen meeting application; the link is created for a specific interview identified by InterviewID and scheduled for the specified Date. Additionally, a notification mail will also be sent to the student.
It returns either the generated link or an error message (i.e. if the link has not been correctly generated)

- **SendInterviewProposal(ApplicationID, Date)**
  This method is used to send to the student related to an application in the "Selection Process" status an Interview Proposal for a specific date. Additionally, a notification mail will also be sent to the student.
  It returns either the the confirmation of Interview Proposal submission or an error message (i.e. if the interview proposal has not been correctly sent)

- **GetApplicationsList(InternshipID)**
  This method is used to retrieve the list of received application for a specific Internship Proposal.
  It returns either the list of received application or an error message (i.e. if no application proposals has been received for the given internship proposal)

- **SetApplicationUnderReview(ApplicationID)**
  This method is used to set a received application to the "Under Review" Status, performing all the checks to verify whether this operation is allowed or not. Additionally, a notification mail will also be sent to the student.
  It returns either the the confirmation of operation or an error message (i.e. if the status has not been changed)

- **RetrieveStudentCV(ApplicationID)**
  This method is used to retrieve the student's CV associated to the considered application.
  It returns either the student's CV or an error message (i.e. if the student's CV has not been correctly retrieved)

- **EvaluateApplication(ApplicationID, Status**
  This method is used to set a received application to one of the following statuses: "Accepted", "Rejected", "Selection Process" or "Internship". Additionally, a notification mail will also be sent to the student.
  It returns either the the confirmation of operation or an error message (i.e. if the status has not been changed)

- **RetrieveApplicationFeed(ApplicationID)**
  This method is used to retrieve all the feedback submitted for a specific application (i.e. the one in the "Internship" status) both by the student and the company.
  It returns either the list of submitted feedback or an error message (i.e. if feedback has been retrieved)

- **WriteComment(ApplicationID, Comment)**
  This method is used to submit a comment for a specific application in the "Internship" status; the process includes validating the comment and storing it within the system. Additionally, a notification mail will also be sent to the interested counterpart.

It returns either the confirmation of submission or an error message (i.e. invalid input or system errors)

- **WriteComplaint(ApplicationID, Complaint)**
  This method is used to submit a complaint for a specific application in the "Internship" status; the process includes validating the complaint and storing it within the system. Additionally, a notification mail will also be sent to the interested counterpart.
  It returns either the confirmation of submission or an error message (i.e. invalid input or system errors)

- **MarkComplaint(ApplicationID, ComplaintID)**
  This method is used to mark a previously submitted complaint as "Solved". Additionally, a notification mail will also be sent to the interested counterpart.
  It returns either the confirmation of the operation or an error message (i.e. the complaint has not been marked as solved)

2.5.9 FeedBack Manager Component

The methods provided by the FeedBack Manager Component are the followings:

- **Comment(ApplicationID, StudentType, ComplaintID)**
  This method is used to store the comment made by a user for a given application.
  It returns either the confirmation of the operation or an error message (i.e. the comment has not been correctly stored)

- **Complaint(ApplicationID, StudentType, ComplaintID)**
  This method is used to store the complaint made by a user for a given application.
  It returns either the confirmation of the operation or an error message (i.e. the complaint has not been correctly stored)

- **ComplaintSolved(ApplicationID, ComplaintID)**
  This method is used to mark an existing complaint as solved.
  It returns either the confirmation of the operation or an error message (i.e. the complaint has not been marked as solved)

2.5.10 WebApp Component

The methods provided by the WebApp Manager Component are the followings:

- **GetConfirmationCode()**
  This method is used to retrieve the confirmation code sent to the user during the registration phase via mail.
  It returns the confirmation code provided by the user

- **RequireDeleteConfirmation()**
  This method is used to prompt the student for confirmation regarding the deletion of a previously sent application.
  It returns a boolean value:
  - True: Indicates that the student confirmed the deletion operation
  - False: Indicates that the student declined the deletion operation

2.6 Selected Architectural Styles and Patterns

In this section, the architectural styles and design patterns selected for the S&C System will be analyzed.

2.6.1 3-Tier Architecture

As outlined in the previous sections, S&C System is designed with a 3-Tier Architecture.
This architectural choice has been made thanks to several significant advantages:

- **Clear Separation of Concerns**
  The 3-tier architecture divides the system into distinct layers, each with a specific responsibility.
  This separation enhances clarity and maintainability, ensuring that each layer focuses on its designated functionality without overlapping with the others
- **Scalability**
  As the S&C System gains in popularity and the user traffic increases, the 3-tier architecture allows each layer to scale independently:

  - The Presentation Layer can manage an increasing number of users by deploying additional web servers

  - The Application Layer can expand to handle more complex operations as new features may be introduced

  - The Data Layer can be optimized with advanced techniques like database clustering or sharing for faster data access and storage

  Thus, independent scaling ensures that resources can be allocated precisely where needed, promoting efficiency and flexibility
- **Modularity and Maintainability**
  Being the layers functioning independently, the updates or the changes in one layer do not disrupt the others: this modularity simplifies development, testing and maintenance
- **Security**
  This layered approach improves the overall system security and protects user information: indeed, by isolating sensitive data within the Data Layer, direct access vulnerabilities from the Presentation Layer are minimized

2.6.2 Model-View-Controller

The 3-tier architecture of the S&C system is complemented by the use of the Model-View-Controller (M.V.C.) pattern to implement the presentation layer (i.e. the web server in Figure 2.1). This pattern is composed of three interconnected components:

- **Model**: it is responsible for handling data-related operations, including retrieval, storage and updates.
- **View**: it manages the user interface, displaying information to the user and reflecting updates based on changes in the model

- **Controller**: it handles the user inputs, updates the model based on user actions and instructs the view to refresh following that event.

The choice of the MVC pattern has been made to ensure some advantages:

- **Maintainability**
  Updating the user interface (U.I.) can be achieved by modifying only the View component, leaving the data-handling and logic components (i.e. Model and Controller) unaffected
- **Scalability**
  Introducing new views, such as a mobile interface, becomes significantly easier: this can be done by simply creating a new View component without impacting the Model or Controller
- **Parallel Development**
  Different teams can work simultaneously on separate components: frontend developers can focus on the View, while backend developers work on the Model and Controller, guaranteeing high efficiency in development.

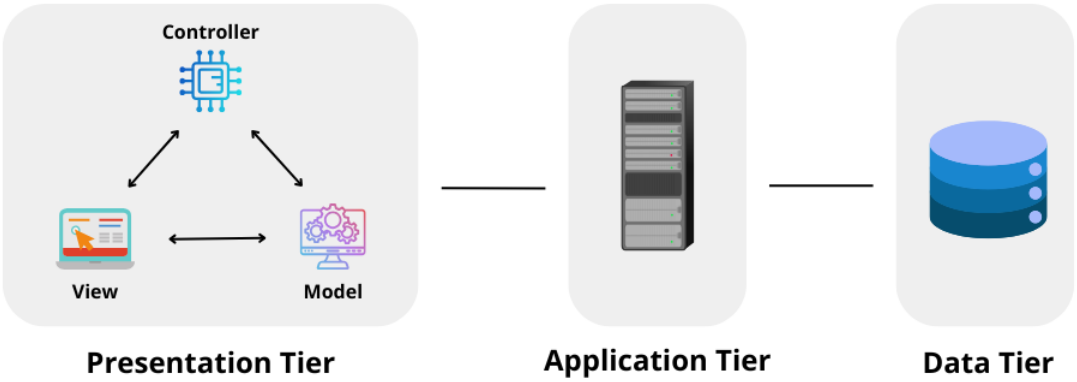

Figure 2.15: Model-View-Controller in the S&C System

2.7 Other Design Decisions

2.7.1 DataBase Structure

The S&C system will be based on a relational database for two primary reasons: first, it is designed to manage highly structured data, aligning perfectly with the capabilities of relational databases; second, relational databases enforce strict data integrity through mechanisms like primary keys, foreign keys and constraints.

2.7.2 RESTful APIs

A RESTful API serves as a communication interface between two computer systems, allowing them to securely exchange data over the internet.
In the S&C system, RESTful APIs are used to facilitate the transfer of information and objects in JSON format via the HTTP protocol, ensuring efficient and standardized communication.

Chapter 3

User Interface Design

This section presents the mockups designed for the S&C Platform, providing a detailed analysis of the individual views for each user type: student and company.

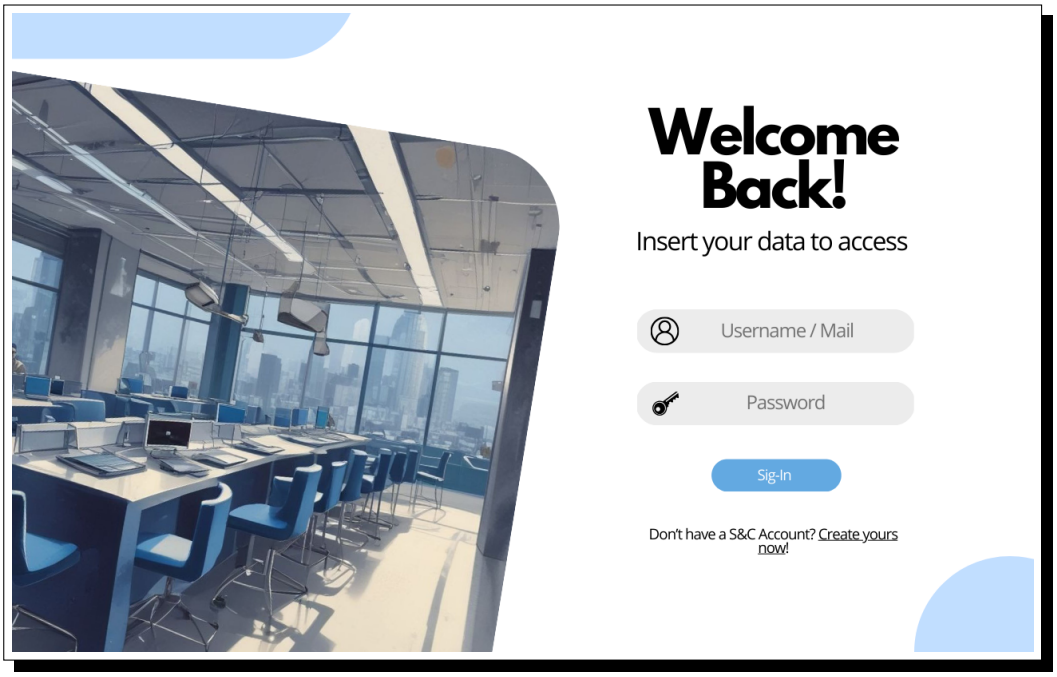

Figure 3.1: Login

3.1 Student View

This section presents the web pages designed for the student users of the platform.
The key pages for effective interactions with the platform are the followings:

- **Sign-Up**: figures 3.2 and 3.3 illustrate the two pages students interact with to complete their registration; these pages, indeed, guide students in providing all the necessary information to access the platform

- **Student Homepage**: upon logging in (see Figure 3.1), the student is redirected to his personalized homepage; this page displays the student's profile information and serves as a central hub for accessing other functionalities (Figure 3.4)
- **Internship Search Page**: figures 3.5 and 3.6 demonstrate how students can explore internship opportunities published on the platform; the search functionality even allows users to apply filters for a more targeted search. Figure 3.7 showcases the results of a sample search, displaying relevant internship listings
- **Sent Applications Page**: the page depicted in Figure 3.8 provides an overview of all applications submitted by the student, including the current status of each application so as to offer a clear summary of the progress for every internship applied to
- **Specific Application Page**: this page offers detailed insights into an individual application.
  Figure 3.9 shows a student reviewing an application in the "Selection Process" phase, allowing options to complete questionnaires and schedule interviews.
  Meanwhile, Figure 3.10 highlights an application in the "Internship" phase, where the student can provide feedback on his ongoing experience
- **Feedback Page**: Figure 3.11 presents the feedback section, which displays all feedback exchanged between students and companies, considering both comments and complaints.
  Complaints are categorized into "Solved" and "Pending" states, providing a clear view on their resolution status

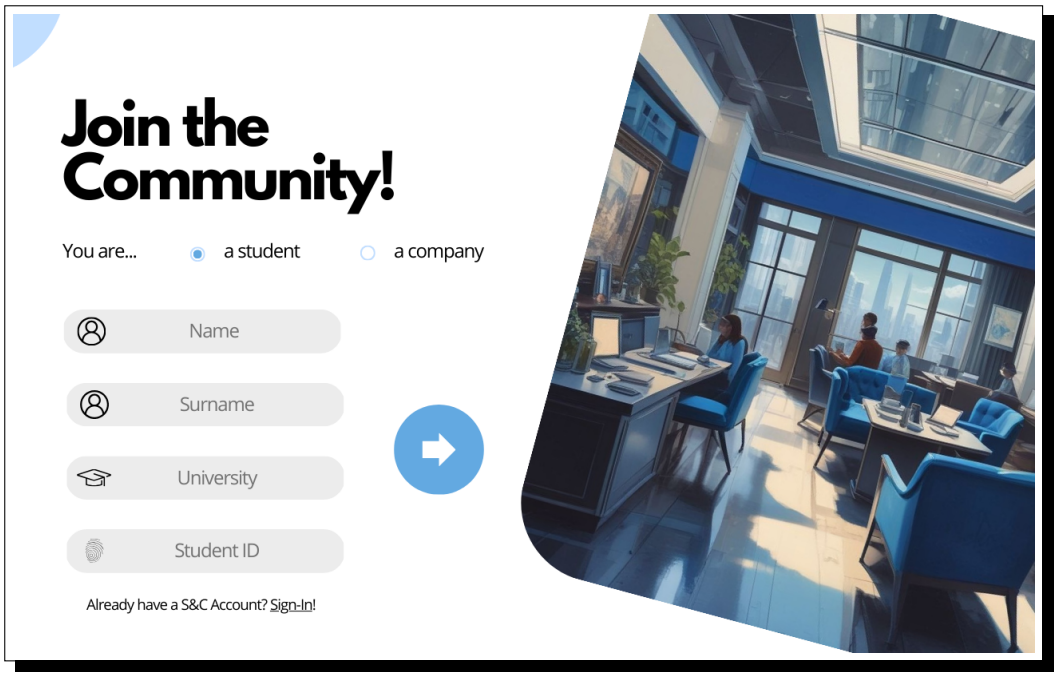

Figure 3.2: Student Sign-Up (A)

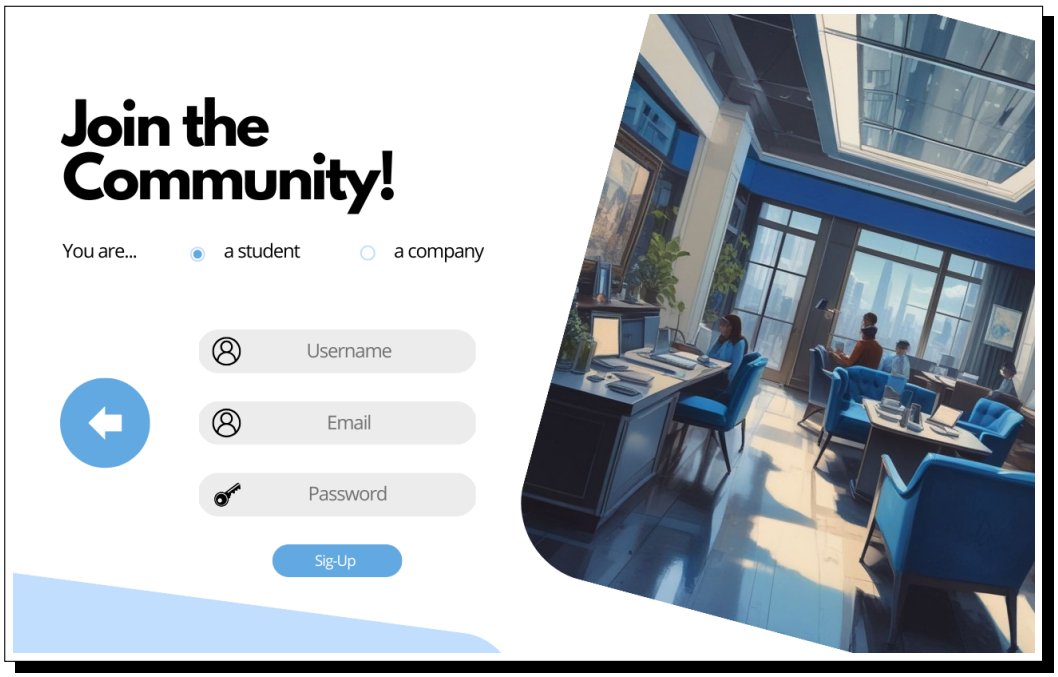

Figure 3.3: Student Sign-Up (B)

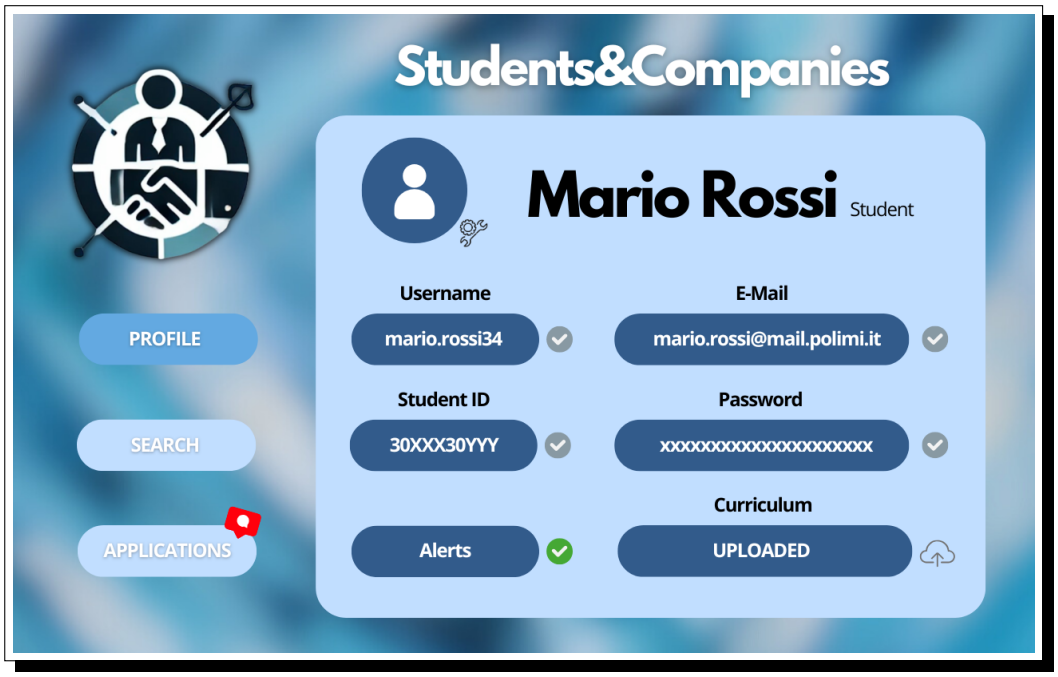

Figure 3.4: Student Profile

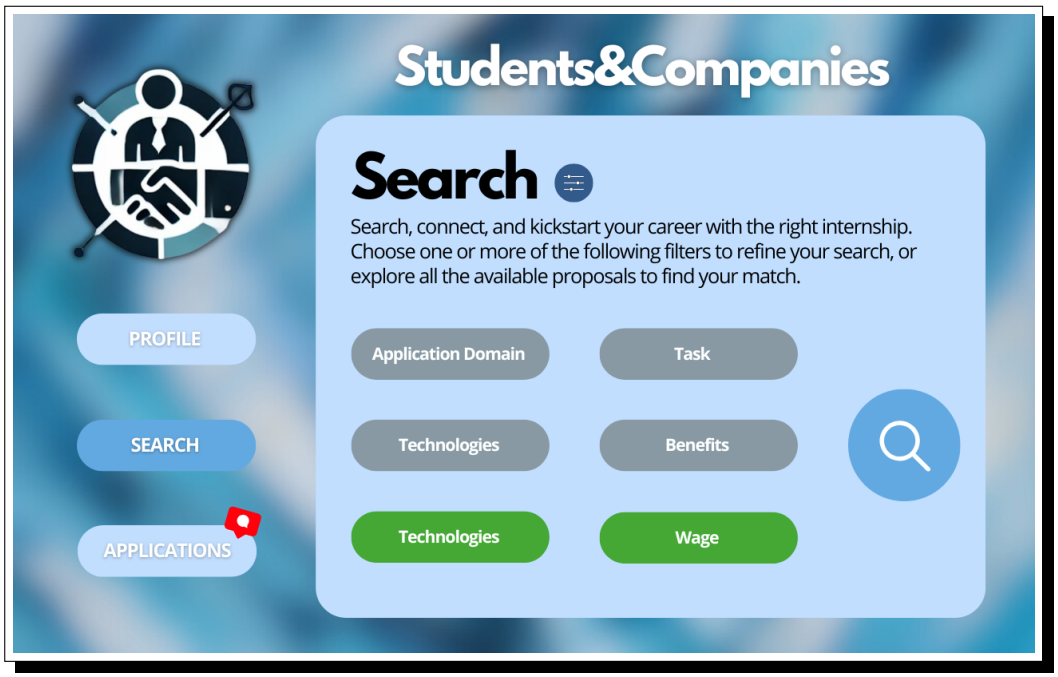

Figure 3.5: Student Internship Proposals Search (A)

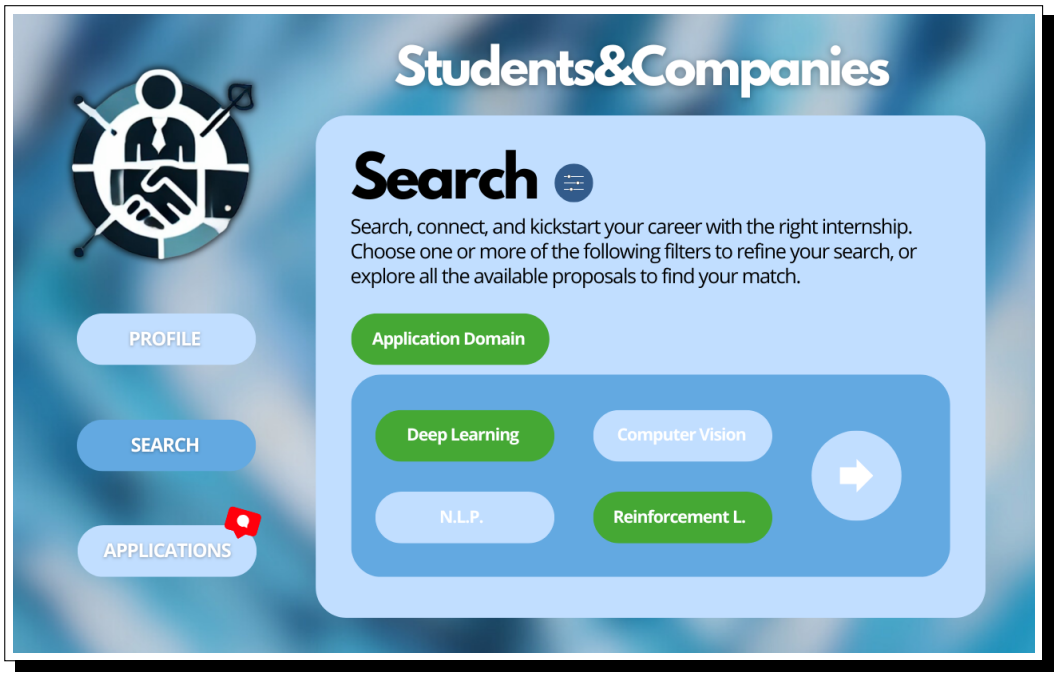

Figure 3.6: Student Internship Proposals Search (B)

Figure 3.7: Student Search Result

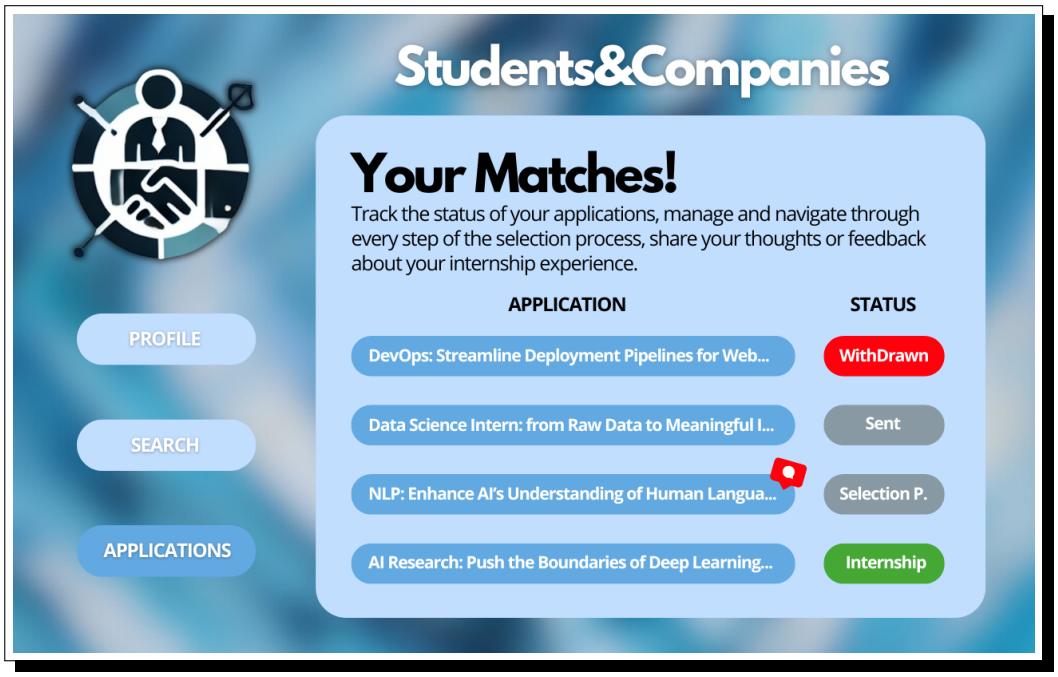

Figure 3.8: Student Sent Applications View

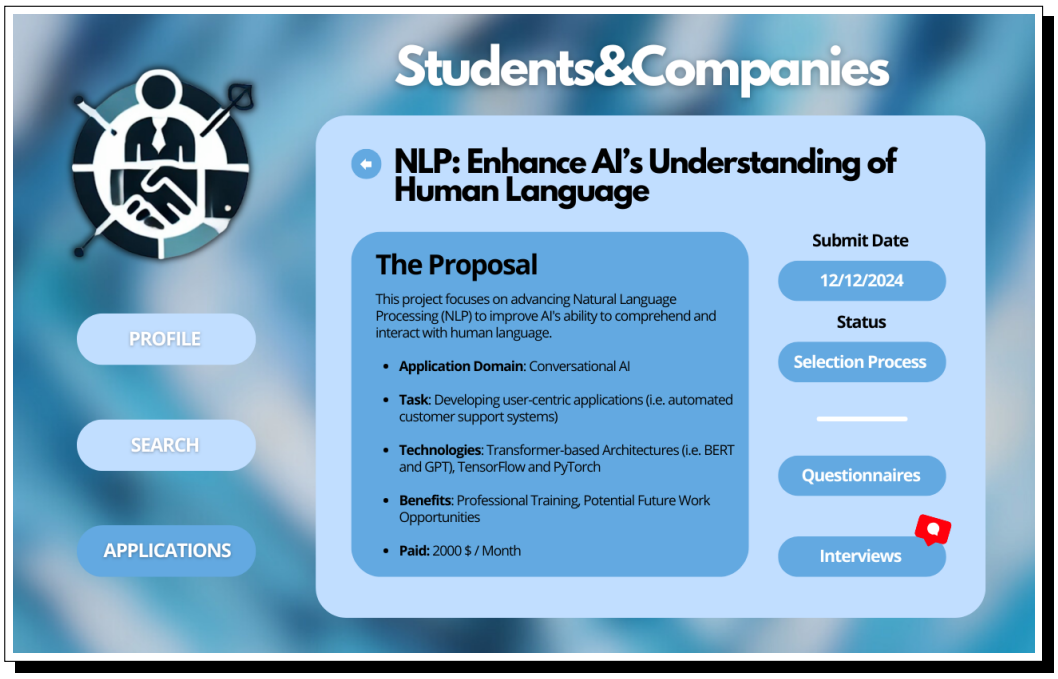

Figure 3.9: Student Selection Process Application

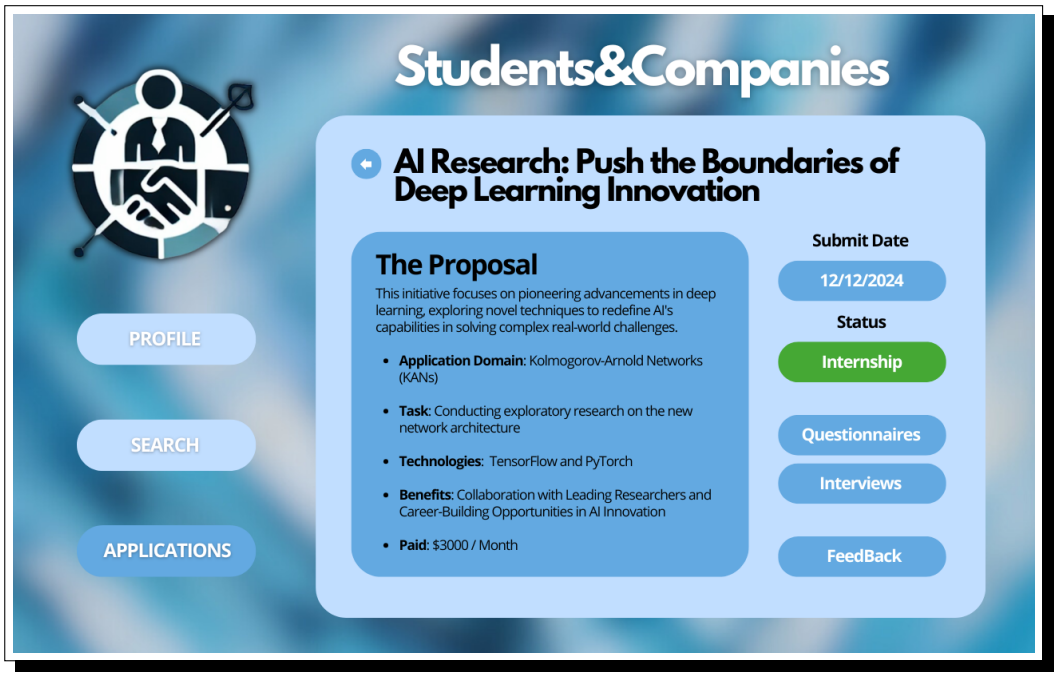

Figure 3.10: Student Internship Application

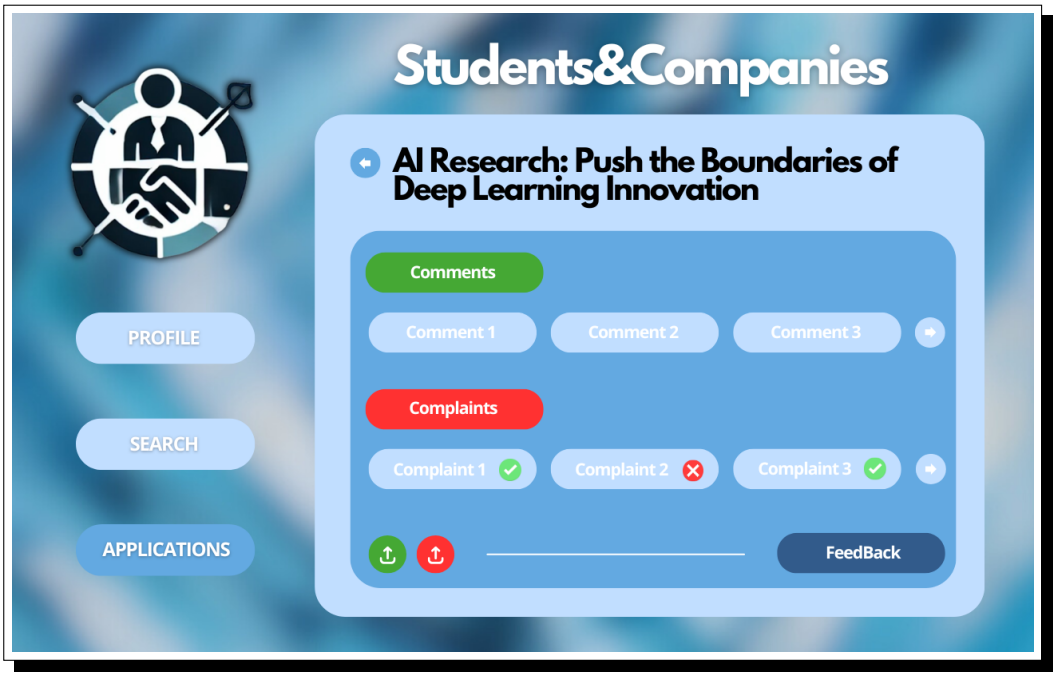

Figure 3.11: Student FeedBack Section

3.2 Company View

This section presents the web pages designed for the company users of the platform.
The key pages for effective interactions with the platform are the followings:

- **Sign-Up**: figures 3.12 and 3.13 illustrate the two pages companies interact with to complete their registration; these pages, indeed, guide companies in providing all the necessary information to access the platform
- **Student Homepage**: upon logging in (see Figure 3.1), the company is redirected to his personalized homepage; this page displays the company's profile information and serves as a central hub for accessing other functionalities (Figure 3.14)
- **Company Internship Proposals Page**: The page shown in Figure 3.15 offers a comprehensive overview of all internship proposals submitted by the company; each proposal will be displayed along with the corresponding number of applicants.
  Additionally, this page allows the company to publish new internship proposals
- **Applications List Page**: the page depicted in Figure 3.16 allows the company to view all the applicants for a specific internship proposal; for each applicant, details such as the submission date and associated university are displayed.
  Additionally, the page provides access to the description of the internship proposal itself and also offers filtering options, enabling the company to view all applications, only those in the "Selection Process" status or those in the "Internship" status
- **Application Evaluation Page**: when the company opens a specific application, the page shown in Figure 3.17 is displayed.

This page allows the company to review the student's CV, update the application's status and send questionnaires or interview proposals directly to the student.

- **Questionnaire Management Page**: figure 3.18 illustrates the page where the company can manage sent questionnaires; this includes viewing both pending and answered questionnaires, consulting the responses and sending new questionnaires to students as needed

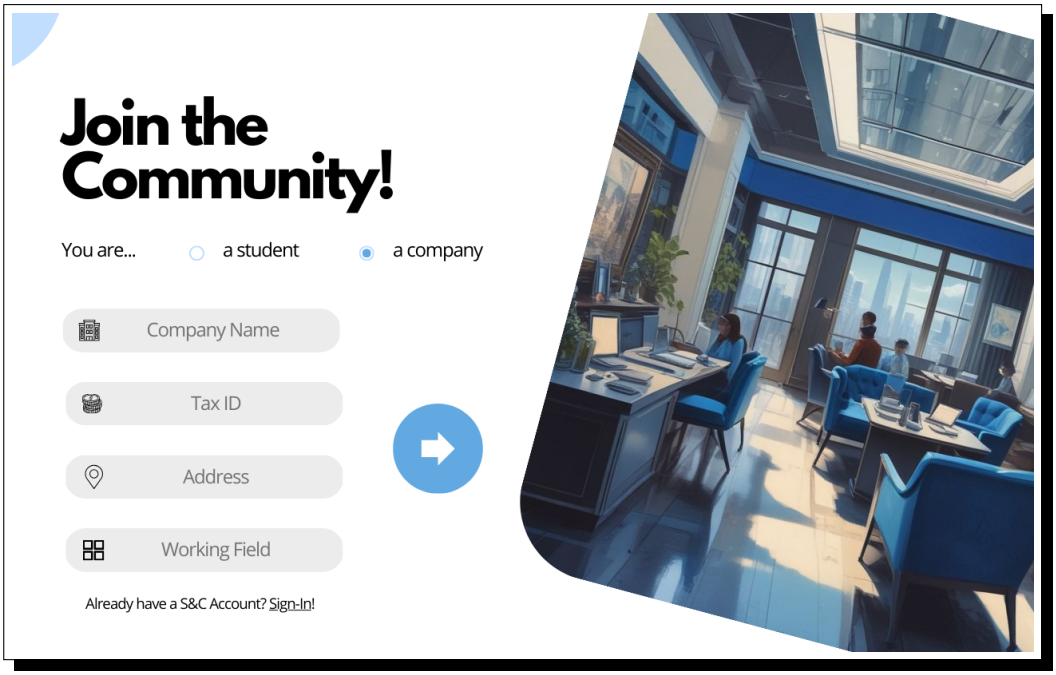

Figure 3.12: Company Sign-Up (A)

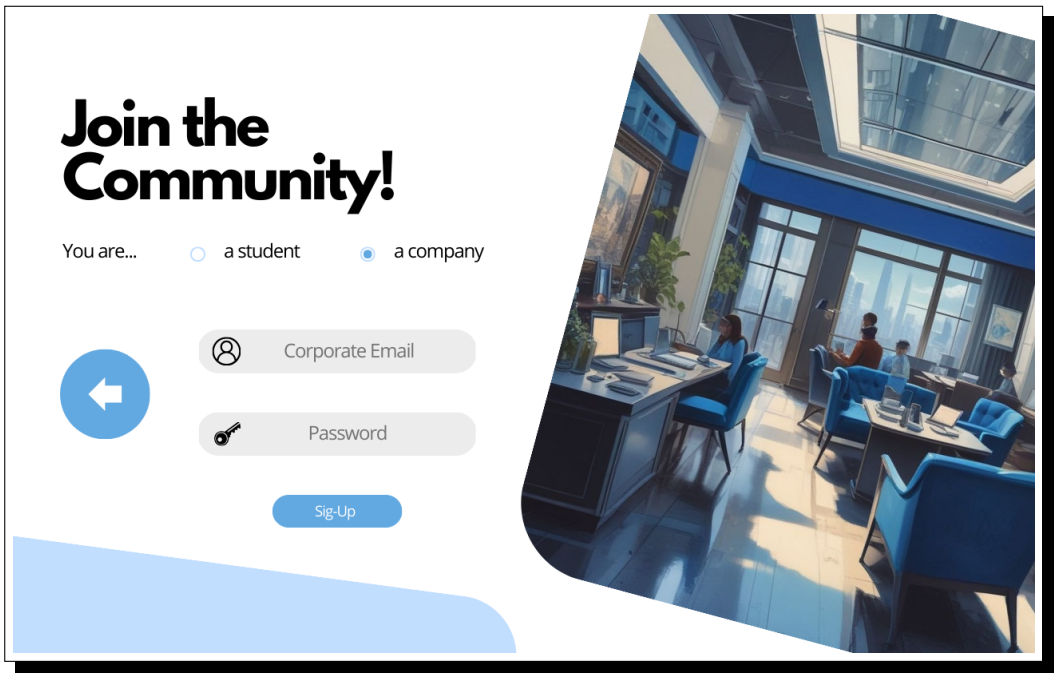

Figure 3.13: Company Sign-Up (B)

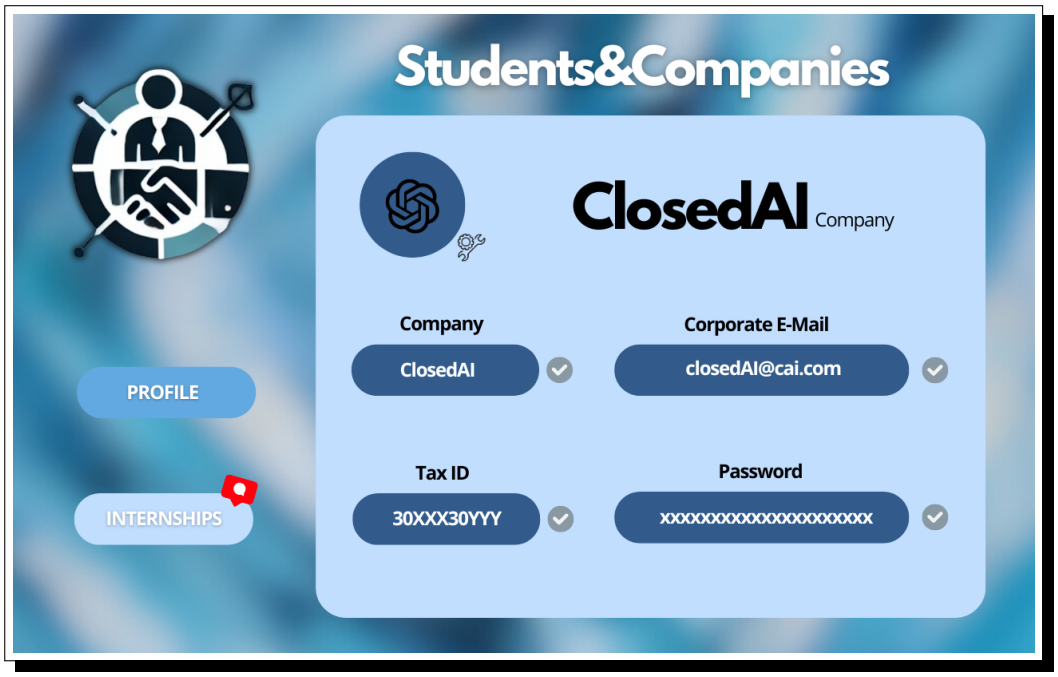

Figure 3.14: Company Profile

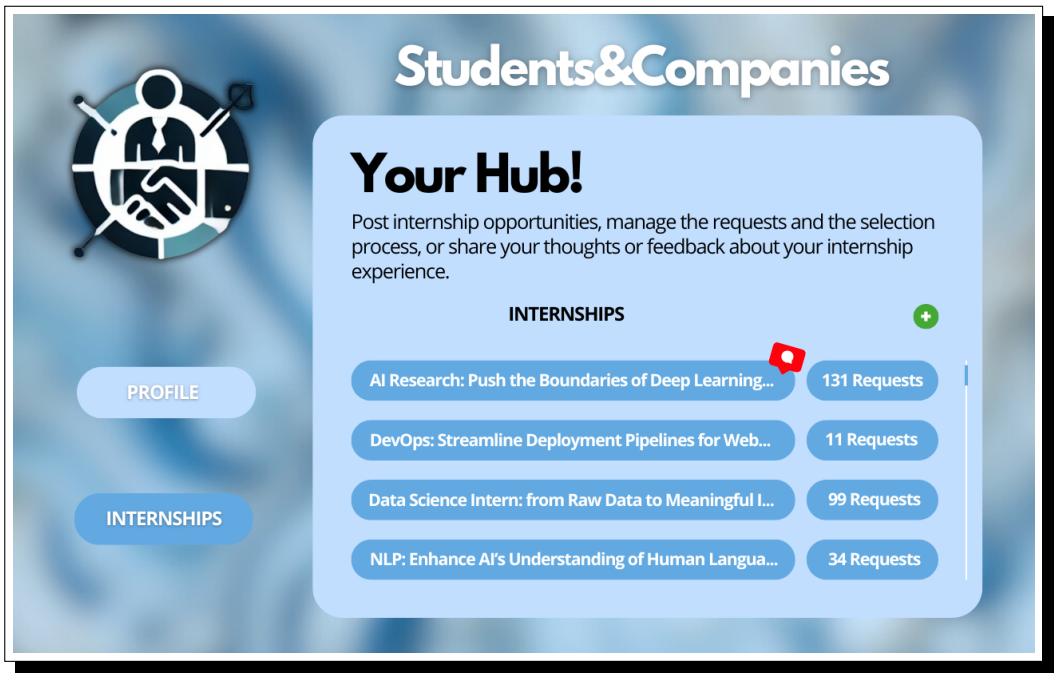

Figure 3.15: Company Internship Proposals View

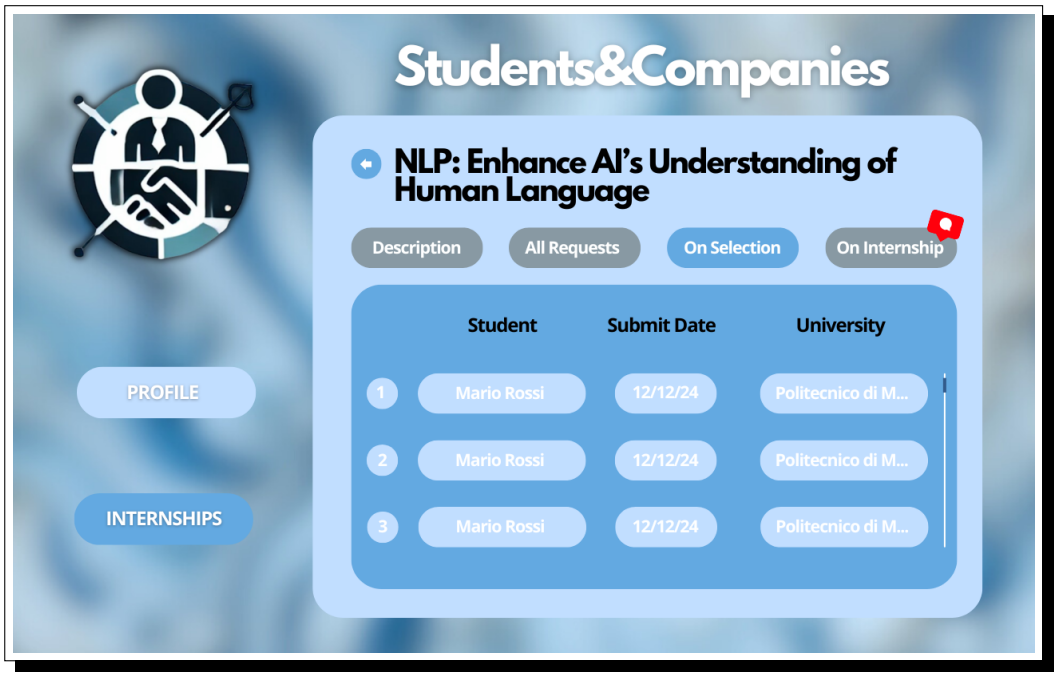

Figure 3.16: Company Application List Management

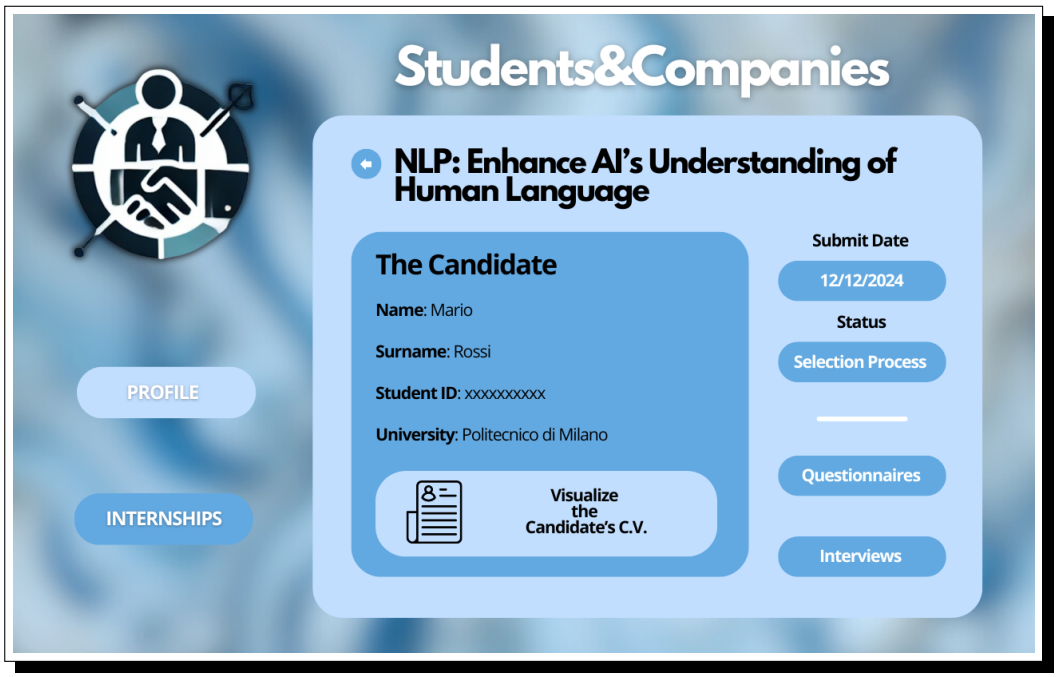

Figure 3.17: Company Application Evaluation

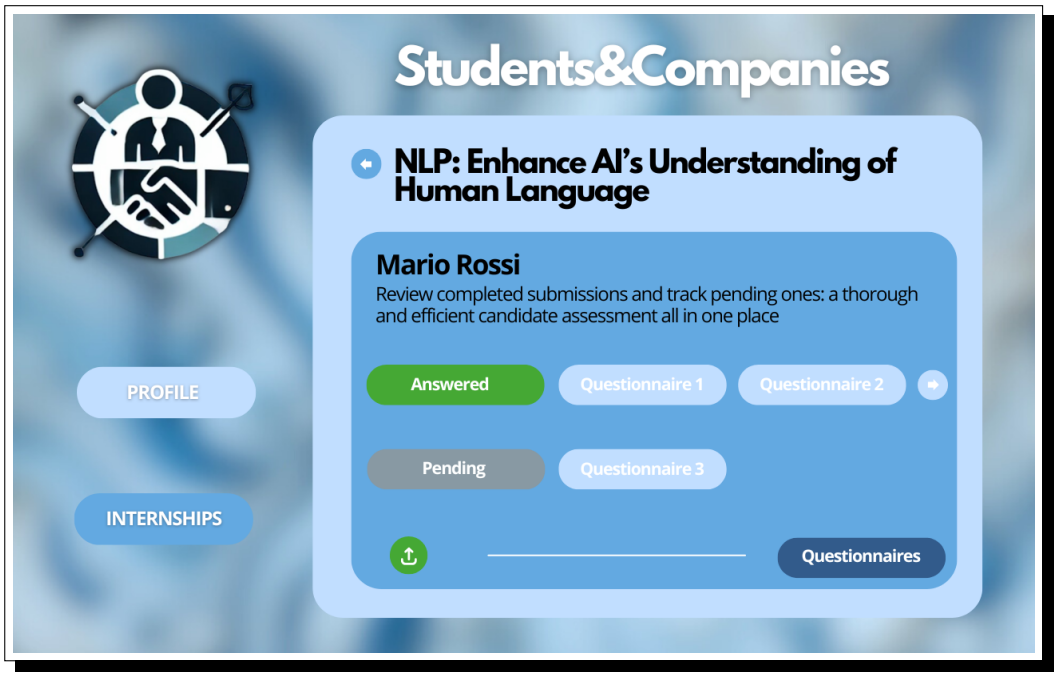

Figure 3.18: Company Sent Questionnaire

Chapter 4

Requirements Traceability

The table below illustrates the mapping of the system's components to their corresponding requirements, providing a clear and concise overview of which components contribute to fulfilling specific requirements.
The table shows a Requirement - Description - Components mapping, indicating a one-to-many relationship where a single requirement may even involve multiple components.
As evidenced by the table, every component plays a role in addressing at least one requirement, which highlights the importance of each component in meeting the system's overall objectives.

Requirement	Description	Component(s)
R.1	The system must allow the student to register into the system by providing all the mandatory information (i.e. Name, Surname, Student ID, etc...)	WebApp, StudentManager, AuthenticatorManager, DataBaseServer, MailManager, MailServer
R.2	The system must allow the company to register into the system by providing all the mandatory information (i.e. Name, Tax ID, Working Field, etc...)	WebApp, CompanyManager, AuthenticatorManager, DataManager, DataBaseServer, MailManager, MailServer
R.3	During the registration process, the system must validate the uniqueness of the Username and the Email provided by the student who is registering into the system	AuthenticatorManager, DataManager, DataBaseServer
R.4	During the registration process, the system must validate the uniqueness of the Name, Tax ID and the Corporate Email provided by the company who is registering into the system	AuthenticatorManager, DataManager, DataBaseServer
R.5	Once the student has entered his registration details, the system must send a verification code to the provided email address	StudentManager, MailManager, MailServer

R.6	Once the company has entered his registration details, the system must send a verification code to the provided corporate email address	CompanyManager, MailManager, MailServer
R.7	The system must allow the student to input the verification code sent to his email address	WebApp, StudentManager
R.8	The system must allow the company to input the verification code sent to its corporate email address	WebApp, CompanyManager
R.9	When a verification code is given in input by the user (either student or company), the system must verify whether the provided code matches the one sent to the user's email	StudentManager, CompanyManager
R.10	The system must allow the student to log into his account by entering his Username/Email and Password	WebApp, StudentManager, AuthenticatorManager, DataManager, DataBaseServer
R.11	The system must allow the company to log into its account by entering its Corporate Email and Password	WebApp, CompanyManager, AuthenticatorManager, DataManager, DataBaseServer
R.12	Once the log-in information has been validated, the system must redirect the user (either student or company) to his corresponding Profile Page	WebApp, StudentManager, CompanyManager
R.13	The system must allow the student to search for available internship proposals	WebApp, StudentManager, DataManager, DataBaseServer
R.14	When searching for internship proposals, the system must allow the student to apply specific filters to refine and limit the search results	WebApp
R.15	The system must allow the student to send an application for the chosen internship proposals	WebApp, StudentManager, DataManager, DataBaseServer
R.16	When the student submits an application, the system must attach the student's CV to the application itself	StudentManager, DataManager, DataBaseServer
R.17	When a new application is submitted, the system must set the application's status to "Sent"	StudentManager
R.18	The system must allow the student to view the list of the sent applications	WebApp, StudentManager, DataManager, DataBaseServer

R.19	The system must allow the student to view detailed information about a specific application (i.e. submission date, internship proposal details, application status, etc...)	WebApp, StudentManager, DataManager, DataBaseServer
R.20	The system must allow the student to delete his application for an internship proposal	WebApp, StudentManager, DataManager, DataBaseServer
R.21	When the student attempts to delete an application, the system must prevent this action if the application status has been set to "Under Review" by the company	StudentManager, DataManager, DataBaseServer
R.22	If the user is allowed to delete the application, the system must update the application's status to "Withdrawn"	StudentManager, DataManager, DataBaseServer
R.23	The system must allow the company to update the application's status in the following sequence: Sent $\rightarrow$ Under Review/Rejected $\rightarrow$ Selection Process $\rightarrow$ Internship/Rejected $\rightarrow$ Internship Completed	WebApp, CompanyManager, DataManager, DataBaseServer
R.24	The system must allow the student to upload his CV	WebApp, StudentManager, DataManager, DataBaseServer
R.25	The system must allow the student to upload a CV only if no CV has been previously uploaded or if the student explicitly agrees to replace the existing CV	StudentManager, DataManager, DataBaseServer
R.26	The system must allow the company to publish a new internship proposal	WebApp, CompanyManager, DataManager, DataBaseServer
R.27	The system must allow the company to view the list of the published internship proposal	WebApp, CompanyManager, DataManager, DataBaseServer
R.28	The system must allow the company to view the list of applications submitted for a published internship proposal	WebApp, CompanyManager, DataManager, DataBaseServer
R.29	The system must allow the company to view the details (e.g. student's CV, submission date, etc...) of a submitted application	WebApp, CompanyManager, DataManager, DataBaseServer
R.30	When the company attempts to change the application's status, the system must prevent this action if the status is set to "Withdrawn"	CompanyManager

R.31	When the application's status is updated, the system must notify the associated student via email	CompanyManager, MailManager, MailServer
R.32	When the company attempts to change the application's status from "Selection Process," the system must prevent this action unless at least one interview or one questionnaire has been completed	CompanyManager
R.33	The system must allow the company to visualize the list of sent questionnaire for a submitted application	WebApp, CompanyManager, DataManager, DataBaseServer
R.34	The system must allow the company to provide a prompt to an AI tool to generate a list of questions to include in the questionnaire	WebApp, CompanyManager, APIManager
R.35	The system must allow the company to view the questions generated by the A.I. Tool	WebApp
R.36	The system must allow the company to manage (i.e. modify, add and delete) the list of questions generated by the A.I. Tool	WebApp
R.37	The system must allow the company to create a questionnaire by submitting a list of questions	WebApp, CompanyManager, DataManager, DataBaseServer
R.38	When the company submits a list of questions, the system must automatically generate a structured questionnaire based on the provided questions	CompanyManager
R.39	After the structured questionnaire is generated, the system must send it to the associated student	CompanyManager, DataManager, DataBaseServer
R.40	The system must allow the company to view the student's answers associated to a sent questionnaire	WebApp, CompanyManager, DataManager, DataBaseServer
R.41	When the student receives a questionnaire, the system must notify him via email	CompanyManager, MailManager, MailServer
R.42	The system must allow the company to view the list of scheduled interviews of a submitted application	WebApp, CompanyManager, DataManager, DataBaseServer
R.43	The system must allow the company to send an Interview Proposal to a student in which it is specified a potential date for the interview	WebApp, CompanyManager, DataManager, DataBaseServer

R.44	When the student receives an Interview Proposal, the system must notify him via email	CompanyManager, MailManager, MailServer
R.45	The system must allow the student to manage (i.e. accept or propose a new date) an interview proposal	WebApp, StudentManager, DataManager, DataBaseServer
R.46	The system must allow the company to generate the Interview Link after the student has accepted the Interview Proposal	WebApp, CompanyManager, APIManager, DataManager, DataBaseServer
R.47	After the Interview Link has been generated, the system must send a mail notification to the associated student	CompanyManager
R.48	The system must allow the company to send a Questionnaire/Interview Proposal to the student only if the application's status has been set to "Selection Process"	WebApp
R.49	When the application's status is "Internship", the system must allow both the associated company and the associated student to submit feedback (i.e. comments or complaints) about the on-going experience	WebApp, StudentManager, CompanyManager, FeedBackManager, DataManager, DataBaseServer
R.50	The system must allow both the student and the company involved in an internship to view the feedbacks (i.e. comments or complaints) made by both of them	WebApp, StudentManager, CompanyManager, DataManager, DataBaseServer
R.51	When the user, either the student or the company, submits a complaint, the system must notify via email the counterpart of the new submission	FeedBackManager
R.52	When the user, either the student or the company, submits a comment, the system must notify via email the counterpart of the new submission	FeedBackManager
R.53	Once the user, either the student or the company, has submitted a complaint, the system must allow them to mark it as "Solved" if the issue has been addressed by the counterpart	WebApp, StudentManager, CompanyManager, FeedBackManager, DataManager, DataBaseServer
R.54	When the student receives a questionnaire, the system must allow the student to complete and submit the questionnaire	WebApp, StudentManager, DataManager, DataBaseServer

Chapter 5

Implementation, Integration and Test Plan

This section outlines the sequence for implementing the components that will make up the S&C system. Additionally, it also provides a detailed plan for the integration and testing phases, ensuring a comprehensive approach to the system development.

5.1 Implementation Plan

The system implementation follows a bottom-up approach, starting with the "leaf components": first the lowest-level components are implemented and, incrementally, they are used to develop the higher-level ones.
In the and, all components are integrated to form the final, complete system as designed.
This approach has been chosen for its several advantages:

- **Core Reliability**
  By focusing on implementing and testing the foundational core components first, the system ensures a stable and reliable base; this will minimize the risk of cascading failures as higher-level functionalities are built upon these components
- **Independent Component Development**
  Since the S&C system consists of multiple independent components, a bottom-up approach allows each component to be developed and validated individually
- **Early Validation of Critical Components**
  Early testing and validation of the components minimize the risk of wasting time and effort attempting to fix higher-level components when the root cause lies in lower-level ones.
  This approach will reduce rework and improves development efficiency
- **Parallel Development**
  Independent components can be developed in parallel without introducing dependencies that might complicate implementation timelines; this advantage obviously accelerates the overall development process

The implementation order for the components introduced in the previous chapters is outlined below:

1. **Database Server** and **Data Manager**: these elements form the core of the system, responsible for storing all data used during platform interactions, and then they must be implemented first
2. **Mail Server, Mail Manager** and **API Manager**: these components can be developed in parallel, as the mail and API functionalities are independent of each other
3. **Authenticator Manager**: once the foundational components are ready, the Authenticator Manager can be implemented; this module allows user registration and login, interacting with the previously developed components
4. **Feedback Manager**: this higher-level component is implemented next, building on the functionality of the earlier modules
5. **Student Manager** and **Company Manager**: these are higher-level components that leverage on all the previously implemented modules.
   Since they are independent of each other, they can be developed in parallel.
   Note: since these components interact with the WebApp during certain operations through the WebAppAPI, their integration will require a top-down approach using a stub to simulate the WebApp behavior during development
6. **WebApp**: the final component to be implemented is the user interface, tying together all functionalities into a cohesive platform

5.2 Integration Plan

This section provides a clear overview of the integration steps for the S&C system, detailing how its components will be combined to create a cohesive and fully functional platform.

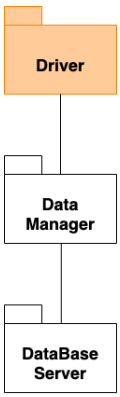

Figure 5.1: Integration Plan - Step 1

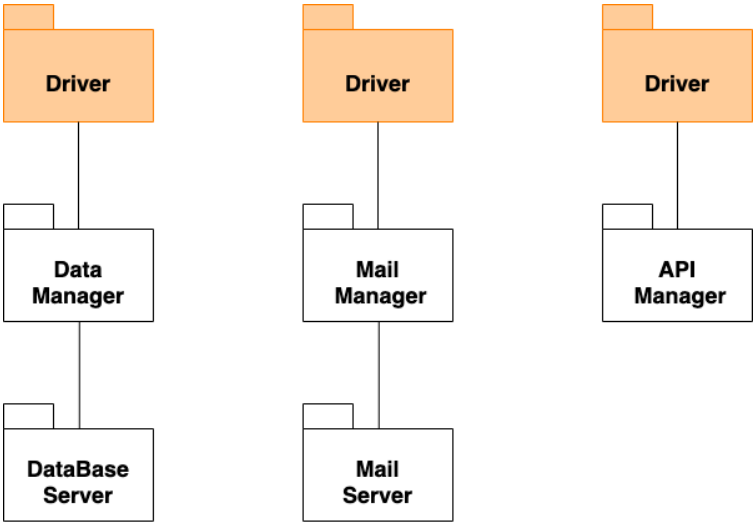

Figure 5.2: Integration Plan - Step 2

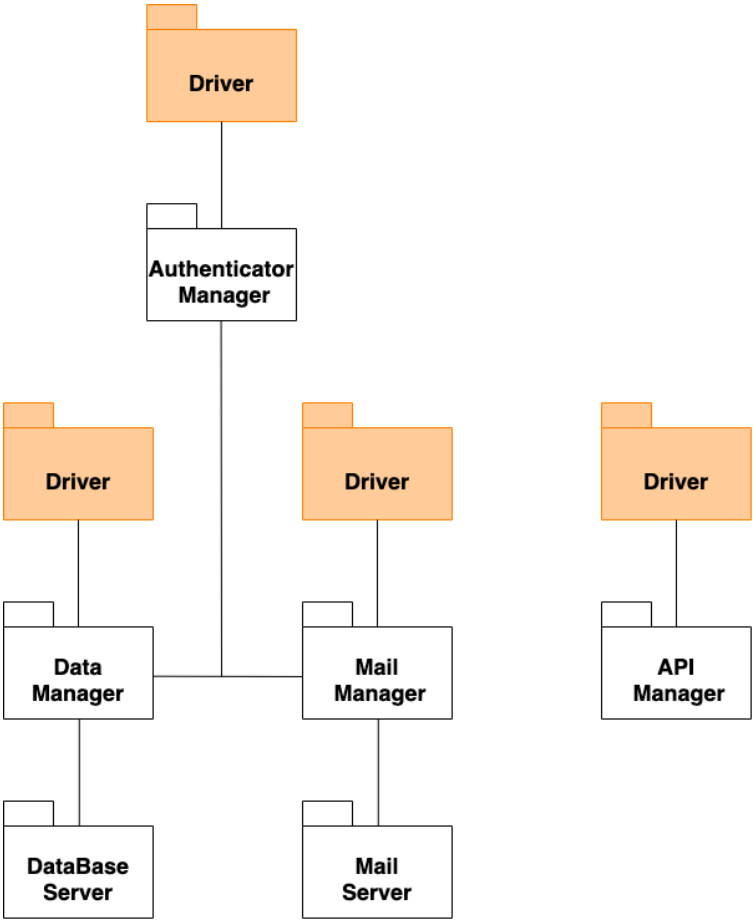

Figure 5.3: Integration Plan - Step 3

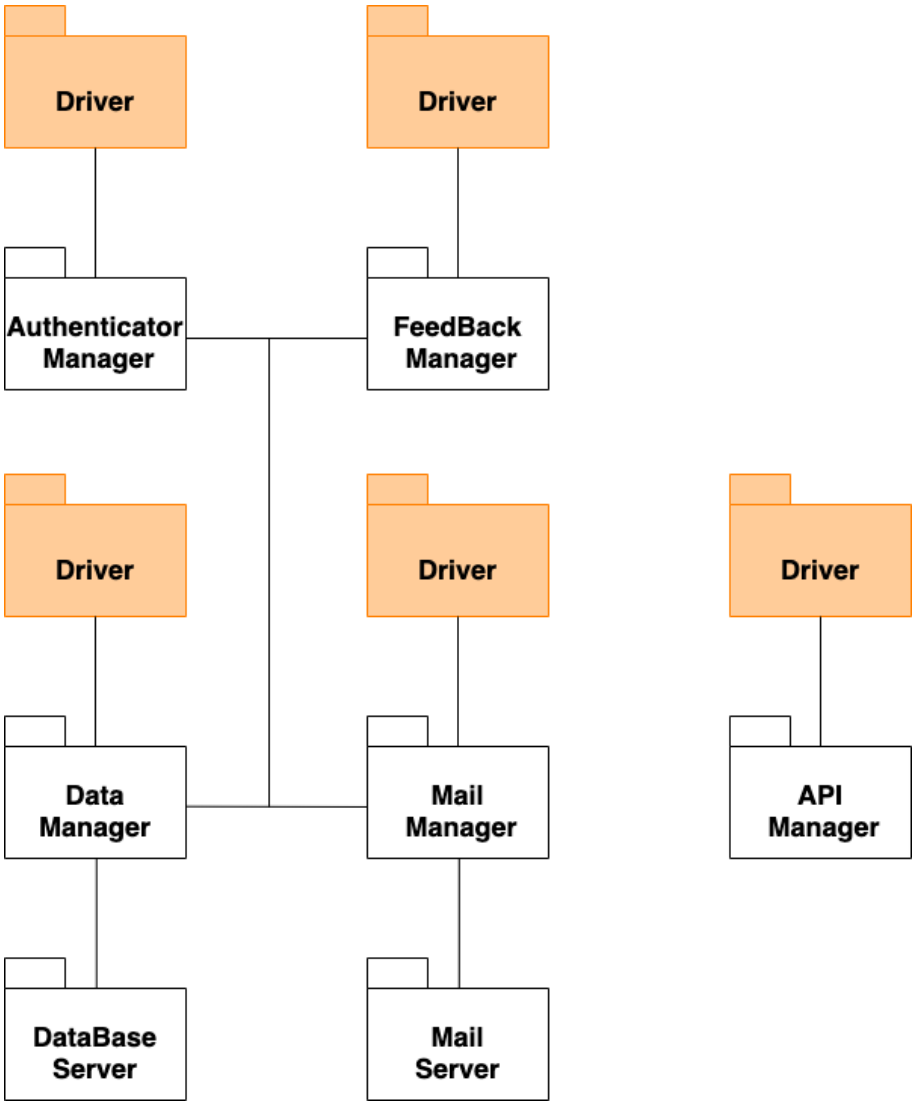

Figure 5.4: Integration Plan - Step 4

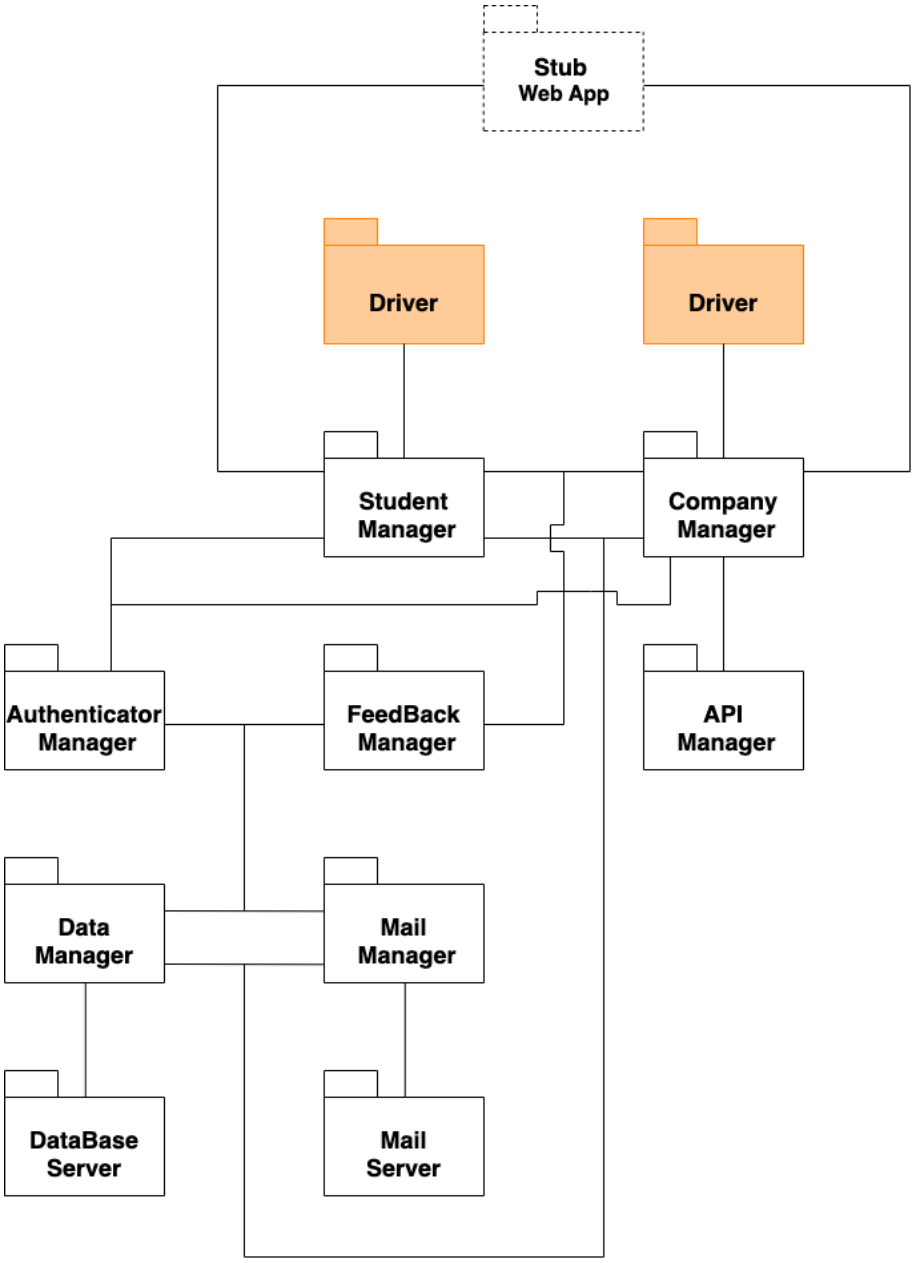

Figure 5.5: Integration Plan - Step 5

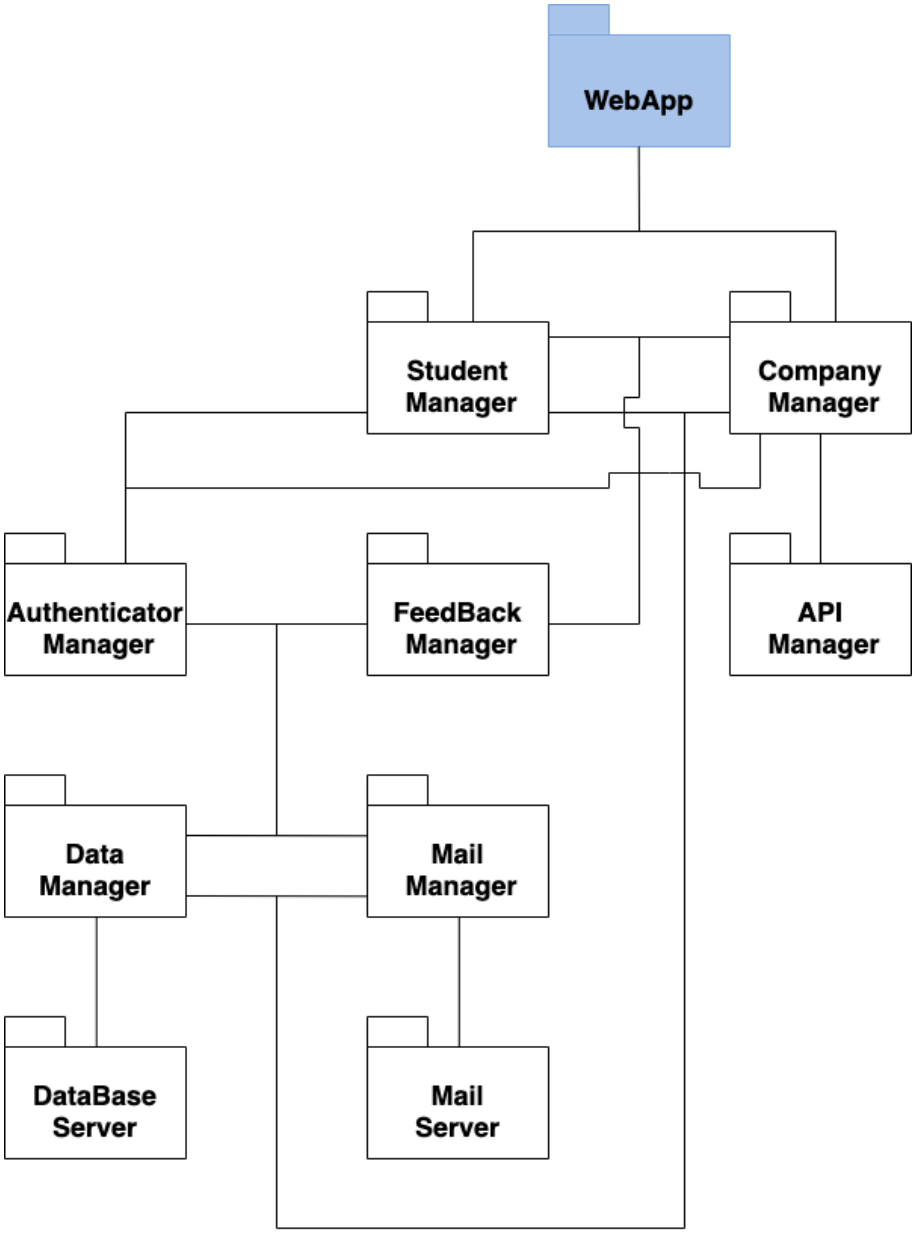

Figure 5.6: Integration Plan - Step 6

5.3 Testing Plan

In order to ensure the robustness and the reliability of the system, a structured testing approach will be applied at various stages of development. The testing phase will include the following methodologies:

1. **Unit testing**
   Unit testing will be carried out during the integration of each individual component, with the primary objective of verifying the correct functionality of each module in isolation, ensuring that all the functions, methods and interfaces operate as expected. For example, we can perform the testing of the Data Manager Component to ensure correct data storage and retrieval operations; validating the Mail Manager Component would allow us to analyze how the email delivery works under various scenarios, etc...

2. **End-to-End (E2E) Testing**
   E2E testing will be performed to validate the overall performance, functionality and integration of all components within the system.
   This approach will focus on the following aspects:

   a. **Performance Testing**: as outlined in the R.A.S.D. document, we have to meet some non-functional requirements related to the system performance.
   Therefore, through the performance testing we will evaluate the scalability (ensuring that the system can handle a growing number of users and data), response time (verifying that user operations, like login and application submission, execute within acceptable time limits under normal and peak loads) and reliability (testing for stability and system up-time under sustained usage)

   b. **Functional Testing**: it will ensure that all the required functionalities, as identified by stakeholders (students and companies), are implemented correctly and behave as expected. This testing will include verifying all operations listed in the system use cases, such as Internship Proposal Search, Apply to Internship Proposals, Evaluate Applications and so on

3. **Integration Testing**
   In addition to the unit and E2E testing, integration testing will be performed after the development of interconnected components.
   It will validate the correctness of data flow and communication between modules, such as between the Student Manager and Authenticator Manager, the compatibility of APIs and, in general, of the provided interfaces

In order to perform these operations appropriate testing frameworks and tools need to be used, such as:

- **JUnit**: to perform unit testing
- **Selenium**: to perform E2E testing
- **JMeter**: to perform performance testing

Chapter 6

References

- *UML Diagrams Examples*
- *Draw.io - Diagram Maker*
- *ChatGPT API - Documentation*
- *Google Meet API - Documentation*
- *Microsoft Teams API - Documentation*
- *3-Tier Architecture*
- *MVC Pattern*
- *JUnit 5 - Unit Testing Tool*
- *Selenium - E2E Testing Tool*
- *JMeter - Performance Testing Tool*