POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design Document (DD)

Students & Companies Problem

Authors:
Acquadro Patrizio
Colosio Giacomo
Drugman Tito Nicola

Course: Software Engineering 2, Computer Science and Engineering

Professor: Prof.ssa Elisabetta Di Nitto

Academic Year: 2024-25

Deliverable:	DD
Title:	Design Document
Authors:	Acquadro Patrizio, Colosio Giacomo, Drugman Tito Nicola
Version:	Version 1
Date:	December 25, 2024
Download page:	GitHub Repository
Copyright:	

Contents

1 | Introduction

As we discussed in the RASD (Requirement Analysis and Specification Document), finding suitable internships remains a challenge for university students with 60% in the U.S. having difficulty in locating opportunities as the main barrier [4]. Furthermore, internship availability has decreased significantly, with only 3,817 positions advertised in October 2024 compared to nearly 5,500 a year earlier [5]. Participation rates are also low, with only 21.5% of U.S. college students and 8.7% of UK students gaining formal work experience and just 19% at top universities [5].

Platforms like *LinkedIn* provide broad job listings but are not optimized for matching internships to student skills and interests. Additionally, companies often struggle to define projects and requirements, leading to mismatches and dissatisfaction. Despite U.S. students generally reporting high satisfaction with internships, 1 in 4 had negative experiences, highlighting the need for improved clarity and alignment between student expectations and company offerings. COVID-19 has also contributed to a sharp decline in internship rates, previously ranging from 50% to 60% in the U.S, now down to 21.5% [4].

1.1. Purpose

The primary objective of this Design Document (DD) is to detail the software design and architectural components of the Students & Companies (S&C) platform, as initially outlined in the Requirement Analysis and Specification Document, accessible at this link.

This document serves as a comprehensive guide to the design of the system, focusing on the following aspects:

- **High-Level Architecture:** An overview of the system's structure, emphasizing the relationships between its main components.
- **Component Design:** Detailed descriptions of the individual components that constitute the system.
- **Deployment View:** The mapping of software components onto hardware nodes, ensuring an effective deployment strategy.
- **Intercomponent Communication:** Analysis of the messages exchanged between components and the interfaces they provide.
- **Technological Choices and Patterns:** Justification of the technologies and design patterns employed to meet the system's requirements.

- **User Interface Design:** Specifications for the interfaces that facilitate interaction between users and the system.

In addition to defining the architectural and design aspects, this document includes an implementation, integration and testing plan to ensure the seamless realization of the platform. Furthermore this document provides an unambiguous description of the system's functionalities and constraints, enabling the verification of whether the platform meets its expected outcomes. This document have the general purpose of guiding the developers in the realization of the S&C platform. It is directed to the project manager, developers and testers but it could be useful for future development and maintenance. To maintain alignment with the RASD and provide a clear context for the design choices made, this document frequently references the RASD. It is highly recommended to read the RASD prior to this document for a complete understanding of the system's objectives and requirements.

1.2. Scope

This Design Document (DD) focus on the purpose and the problems solved by the S&C platform. As we wrote in the *RASD* document, the S&C platform was imagined to be a platform where students could find and apply to internships and companies could create new internships and select students. Lastly, the university tutor would be able to monitors the internships of the students.

The platform operates within the domains of education and professional recruitment, serving as a bridge between academia and industry. It promotes collaboration and interaction among three main categories of users:

- **Students**: who seek internship opportunities to gain professional experience.
- **Company Tutors**: responsible for creating and managing internships and selecting the most suitable candidates.
- **University Tutors**: overseeing the progress of internships and evaluating their outcomes.

The scope of the platform is to facilitate the matching between students and companies by assessing the student experiences, skills and attitudes (available in his/her CVs) and the projects and terms offered by the companies.

As detailed in this document, the S&C platform is built using a 3 tier architecture to ensure scalability, maintainability and efficiency. This architecture integrates the client-server communication paradigm for seamless integration, creating a platform that enhances the internship experience for students, improves recruiting processes for companies and ensures academic oversight by universities. This particular type of architecture will be discussed in greater detail later on.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- **User/actor**: A generic person who use the platform. Can be either a student, company tutor or a professor.
- **Student**: A primary actor representing a user who interacts with the platform to search for internships, submit applications and communicate with other types of users.
- **Company Tutor**: A primary actor representing a user who interacts with the platform to post internships, evaluate candidates and communicate with students and university tutors.
- **University Tutor/Academic Tutor**: A primary actor representing a user who monitors internship progress, evaluates reports and ensures alignment between the internship and university goals. He/she also have the ability to communicate with students and company tutors
- **Responsible Tutor**: A company or university tutor who is responsible for creating the profile either for the company or for the university, but not both.
- **Architectural Style:** An architectural style establishes the fundamental building blocks and rules that shape a software architecture. It determines the vocabulary of components and connectors that can be used, as well as the constraints on how they can be combined. Architectural styles also provide guidance for structuring solutions tailored to particular challenges or domains.
- **Client-Server Architecture**: The client-server style is one of the most familiar architectural approaches in software design, widely used in systems that require distributed access to shared resources. In the client-server model, the architecture is defined primarily by two roles:
  Client—The client component initiates requests and represents the users or user interfaces that require resources or services.
  Server—The server component awaits incoming requests and, upon receiving a request from the client, processes it and provides the necessary response.
- **Three-Tier Architecture:** A specific extension of the client-server architecture that divides the system into three layers. The first one is the Presentation Layer, also known as the Interface Layer, it manages interactions with the end users or external systems, typically through a Graphical User Interface (GUI) or an Application Programming Interface (API). The second is the application logic layer, which is the core processing layer responsible for executing business logic, coordinating tasks and acting as a mediator between the presentation and data layers. Lastly, the Data Layer handles the storage, retrieval and management of data necessary for the application logic layer to function. By separating concerns into these distinct layers, the three-tier architecture enhances scalability, maintainability and reusability. It builds upon the client-server model by introducing additional modularity, where the

"server" is often split into the application logic and data management components.

- **Thin Client:** In this configuration, the client only contains a minimal part of the system, often limited to the interface layer. Most of the application logic and data handling are located on the server. This setup is efficient for clients that need only basic interaction capabilities, as the bulk of processing is done server-side.
- **Thick (Fat) Client:** This configuration includes substantial portions of the application logic and possibly even some data on the client side. While it reduces the server's workload, it requires more resources on the client side and increases the complexity of client management.

1.3.2. Acronyms & Abbreviations

- **S&C**: Students and Companies
- **w.r.t.**: with respect to
- **i.e.**: *Id est*, that is
- **e.g.**: *Exempli gratia*, for example
- **CV**: Curriculum Vitae
- **RASD**: Requirements Analysis & Specification Document
- **DD**: Design Document
- **LLM**: Large Language Model
- **UI**: User Interface
- **API**: Application Programming Interface
- **AI**: Artificial Intelligence
- **NLP**: Natural Language Processing
- **HTTPS**: Hypertext Transfer Protocol Secure
- **UML**: Unified Modeling Language
- **GUI**: Graphical User Interface
- **cmp**: Component
- **REST**: Representational State Transfer
- **PK**: Primary Key
- **FK**: Foregin Key
- **MVC**: Model View Controller
- **CRUD**: Create, Read, Update and Delete

1.4. Revision History

1.5. Reference Documents

The document is based on the following materials:

- The specification of the RASD and DD assignment of the Software Engineering II course a.a. 2024/2025.
- Slides of the course on WeBeep.
- Course book created from notes: `https://drive.google.com/drive/u/1/folders/1dH-0IdPxUwhFMTnOr7UGTCkolvEL6g15`
- **IEEE Standard for RASD**: ISO/IEC/IEEE 29148 (Nov 2018). `https://doi.org/10.1109/IEEESTD.2018.8559686`
- **IEEE Standard for DD**: IEEE 1016-2009 (Jun 2009). `https://ieeexplore.ieee.org/document/5167255`
- RASD document link.

1.6. Document Structure

This Design Document (DD) is structured to provide a comprehensive overview of the design and implementation process for the Students & Companies (S&C) platform. Each section addresses a specific aspect of the system, as outlined below:

Section 1: Introduction.
This chapter provides the context, purpose and scope of the Design Document. It also clarifies terminology, outlines the intended audience and lists references. Readers will understand the document's goals and how it relates to the Requirement Analysis and Specification Document (RASD).

Section 2: Architectural Design
This chapter delves into the high-level and detailed architecture of the Students & Companies platform. It describes the chosen architectural styles, component interactions and deployment strategy. It also includes diagrams and rationales for the architecture decisions, ensuring clarity on how the system will be structured and operate.

Section 3: User Interface Design
Presents the user-facing elements of the system, focusing on usability and user experience. This section includes examples of the interface layout and interactions for critical functionalities.

Section 4: Requirements Traceability
This chapter ensures that every functional and non-functional requirement outlined in the RASD is directly mapped to the corresponding components or features in the system. It provides a comprehensive traceability table, ensuring that all requirements are addressed.

Section 5: Implementation, Integration and Test Plan
Outlines the plan for developing, integrating and testing the components of the system. It specifies the order of implementation and the strategies for verifying and validating the system's functionality and to ensure the system meets both functional and non-functional requirements.

Section 6: Effort Spent
This chapter list the contributions of each team member by recording the number of hours spent working on different parts of the document.

2 | Architectural Design

This chapter provides a detailed overview of the system's architecture, focusing on its high-level structure, core components, deployment setup, runtime interactions and applied architectural styles and patterns. As *Bass et al.* describe, *"software architecture is a set of structures needed to reason about the system"*. Including the software architecture in the Design Document (DD) is essential as it offers a framework to guide implementation and development, enabling early identification and resolution of design issues. It also facilitates communication among stakeholders by creating a shared understanding of the system's organization and behavior, while ensuring maintainability and scalability through well-defined components and relationships.

The architecture of the S&C platform is composed of three primary components. The first is the component-and-connector structures, which capture runtime interactions, communication pathways and dependencies, thereby highlighting the dynamic behavior of the system during execution. The second is the module structures, which define the static organization of the system by focusing on its logical units of code and data, aiding developers in implementation, planning and maintenance. Lastly, the allocation structures map the software elements to physical or organizational resources, such as deployment on hardware, file systems or team assignments.

To analyze these architectural components UML diagrams play a fundamental role. Component diagrams, as detailed in Section 2.2.1, illustrate the interactions between components, while sequence diagrams, discussed in Section 2.4.2, represent the flow of control and communication during execution. Deployment diagrams, covered in Section 2.3.2, provide insights into how components are mapped to the hardware environments. These diagrams ensure clarity and effective communication among stakeholders by offering a comprehensive view of the system's architecture.

2.1. Overview

The S&C platform is designed as a three-tier web-based system that connects, students, company tutors and university tutors. The three tier is one of the most popular implementation of a multi-tier architecture and consist of *Presentation Layer*, *Application Layer* and *Data Layer*. Thanks to this decomposition each layer can be developed or update independently by a different team and also it have great elasticity since a single layer can be scaled up regardless of the other layers. In a three-tier application, all communication goes through the application tier. The presentation tier and the data tier cannot communicate directly with one another.

- **Presentation Layer**: how the system interacts with the user. It is accessible to the user with a GUI. Its main purpose is to display information to and collect information from the user.
- **Application/Logic Layer**: handle the logic of the platform and provide all the functions available to the user. It receive the requests from the clients and handles them. It also communicate with the Data layer. It is the heart of the application. In this tier information that is collected in the presentation tier is processed.
- **Data Layer**: store and retried asked data. Does not implement any logic and it is only used for data storage. It is where the information that is processed by the application is stored and managed.

The thin-client approach minimizes client-side resource requirements while leveraging the server-side Application Layer for robust logic handling while the Data Layer guarantees secure and efficient management of user and system data.

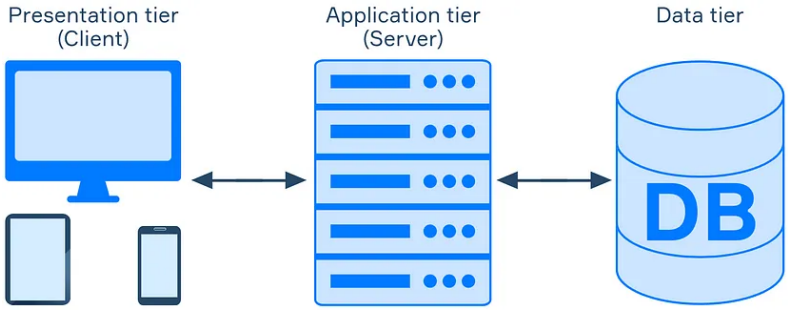

Figure 2.1: Three tier architecture. [1].

2.2. Component View

This section delves into the internal structure of the S&C platform, focusing on the relationships between its components and the interactions both internally and with external services. The component diagram in Figure 2.2 offers a high-level visual representation of the system's key components and their dependencies, providing a clear understanding of how the system operates and facilitating both implementation and maintenance.

The S&C platform is structured around two categories of components: external and

internal. External components interact with the system from outside its boundaries, integrating functionalities such as user interfaces, APIs and third-party services, they facilitate communication and integration with external entities. Key external components include the Web Application, which serves as the primary user interface for stakeholders, the DBMS for secure data storage, the Mail Server for communications and the External Calendar Service for synchronization of the calendars. External components also ensure that inputs can be received and outputs delivered to the external world.

Internal components, on the other hand, are entirely contained within the system and they manage the platform's core functionalities and logic. The Dashboard Manager acts as a central hub, orchestrating communication between users and various system modules. It routes requests to appropriate components. Model serves as the intermediary for database interactions, handling CRUD operations and ensuring secure and consistent data management. Internal modules like the Registration Manager, Login Manager, Profile Manager, Internship Manager, Matchmaking Manager and ActiveStage Manager handle specific functionalities, from user authentication and profile management to internship creation, matchmaking and ongoing monitoring. They will be discussed later on in greater detail in Section 2.2.2. Internal components manage the internal logic, process data and enable communication between various parts of the system, forming the backbone of the platform's operations.

2.2.1. Component Diagram

In Figure 2.2 it is possible to see the component diagram for the S&C platform. This figure is a high-level view that only show the main components, while the sub-components will be shown later in the Composite Structure Diagrams in Section 2.2.2. For a more clear description we decided to use colors to highlight specific categories of communications between different components:

- Yellow color: it is used to highlight communications with the *NotificationManager* component.
- Light blue color: it is used to highlight communications with the *Model* component.
- Purple color: it is used to highlight the communications with the *CalendarManager* component.
- Green color: it is used to highlight the communications with the *Chat Manager* component.

We define here the main external components:

- **WebAPP**: it is the external access point for all users. It allows the communication with the S&C platform thanks to the *Dashboard Interface*. The S&C platform can send notifications to users through the Notification Interface.
- **DBMS**: it is the storage repository for all data (about profiles, internships,...) of the S&C platform. It can communicate with the S&C platform via the DBMS API.

- **Mail Server**: it is responsible for sending registration confirmation email or to recover the user's password. It uses the Mail API interface to communicate with the S&C platform.
- **External Calendar Service**: it is used to connect with the users personal calendars. It uses the Calendar API to communicate with the S&C platform.

We define here the components inside the S&C platform:

- **Model**: it is a high-level component that represents the data on the server and acts as an interface to the database server. Every component needs to interface with *Model* to access data from the DBMS through the DBMS API.
- **Dashboard Manager**: it is a fundamental component that is in charge of orchestrates all communication between users and the S&C platform. All users interact with the S&C platform through the Dashboard Interface and the *Dashboard Manager* is used to direct all the requests of the users to the appropriate components. It servers as the central hub for all users interactions.
- **Registration Manager**: it is the component that handles the registration of new users. The idea behind is that when a new user wants to create an account on the platform, he/she communicates to the *Dashboard Manager* that send the user's request to the *Registration Manager*. This component communicates with the Mail Server to send a confirmation mail to the user and it also communicate with the *Model* component to add the new data of the user to the DBMS. This component is visible in greater detail in Figure 2.3.
- **Login Manager**: it is the component that is in charge of login users that are registered. When a registered user attempts to log in, the *Dashboard Manager* forwards the request to the *Login Manager*. It communicates to the *Model* through the model interface to retrieve the user's data from the DBMS. The login manager is also used for registered user to recover their password, at such it communicates to the *Mail Server* through the *Mail API* interface. This component is visible in greater detail in Figure 2.4.
- **Profile Manager**: this component servers general purposes and can be used by register users to visualize a profile, modify its own profile or delete their profile. It can be used for all profiles regardless of their role (student, company tutor or university tutor) and it can be used also for company and university profiles. When a user want to search a profile, or modify/delete its profile, the *Dashboard Manager* forwards the request to the *Profile Manager*. The *Profile Manager* is connect to the *Model* through the model interface to access all user's data. This component also communicates with others components that allows the user to visualize a profile, it can be seen in greater detail in Figure 2.5.
- **Internship Manager**: this component is used for several task related to internships. A company tutor (a particular category of users) can use *Internship Manager* to create or edit or delete an internship, save as draft, publish it or improve it with the help of an LLM. To do so he/she uses the *Dashboard Manager* component that

communicates to *Internship Manager* component. Also all registers users can visualize all the internship on the platform with the *Internship Manager* component. This component communicate also with the *Model* to save all the information about an internship in the DBMS and it is visible in greater detail in Figure 2.6.

- **Matchmaking Manager**: this component is used for the matchmaking process. It is used for recommending to students or companies their counterparts and to visualize their profiles (thus this component communicates with *Internship Manager* and *Profile Manager*) also it is used to invite users to the selection process. This component can be seen in greater detail in Figure 2.7.

- **Selection Manager**: this complex component it is used for several processes such as evaluating the invitation about an internship, plan meetings, fill up the questionnaires, pick up the university tutor and so on. When a user want to do either one of this task the *Dashboard Manager* forwards the request to the *Selection Manager*. Lastly, this component communicates with multiple components and is discussed in greater detail in Figure 2.8.

- **ActiveStage Manager**: it is the component that handles everything that can be done by users when an internship is active such as completing final evaluation, communications, planning events or reporting a complaint. When a user want to do either one of this task the *Dashboard Manager* forwards the request to the *ActiveStage Manager*. It is connected to several other components and is discussed in greater detail in Figure 2.9.

- **Complaint Manager**: this component it is used to user's tasks about complaint such as reporting one or managing one. When a user want to do so the request of the *Dashboard Manager* is forwarded to the *Complaint Manager* that will handle this request. Lastly this component can be used to delete a complaint (if resolved) or interrupt an internship. It is is discussed in greater detail in Figure 2.10.

- **Calendar Manager**: this component allows creation or scheduling of events or meetings. It communicates with *External Calendar Service* trough the *Calendar API* to allow users to save events inside the S&C platform also on their personal calendars such as *Google Calendar* or *Apple Calendar*. Lastly this component communicates also with *Chat Manager* since when an event is created then automatically a chat is created too.

- **Questionnaire Manager**: it is the component than handles the questionnaires created and filled in by the registered users. When a user needs to create a questionnaire or fill up one the *Dashboard Manager* forwards this request to the *Questionnaire Manager*. This component communicates with the *Model* component to save the questionnaires in the DBMS and with the *Notification Manager* to send some notifications to the users.

- **Chat Manager**: it is the component in charge of the communication between users. Its main job is to handle to send and receive messages on the platform and to notify users about new messages. As such this component needs to communicate with *Model* to save all the messages in the DBMS and with *Notification Manager* to

notify users about new messages. This component is also in charge of the chatbot assistance communication when a user needs help. The idea behind is to consider the interaction between a user and a chatbot as a particular kind of chat and thus handled by *Chat Manager* to avoid repeating creating a new almost-identical component just to handle the chatbot communication.

- **Notification Manager**: this component allows the user to receive notifications about some updates that happened in the platform (such as receiving a message). It also allows the user to customizes notification settings (such as frequency or type of notification).

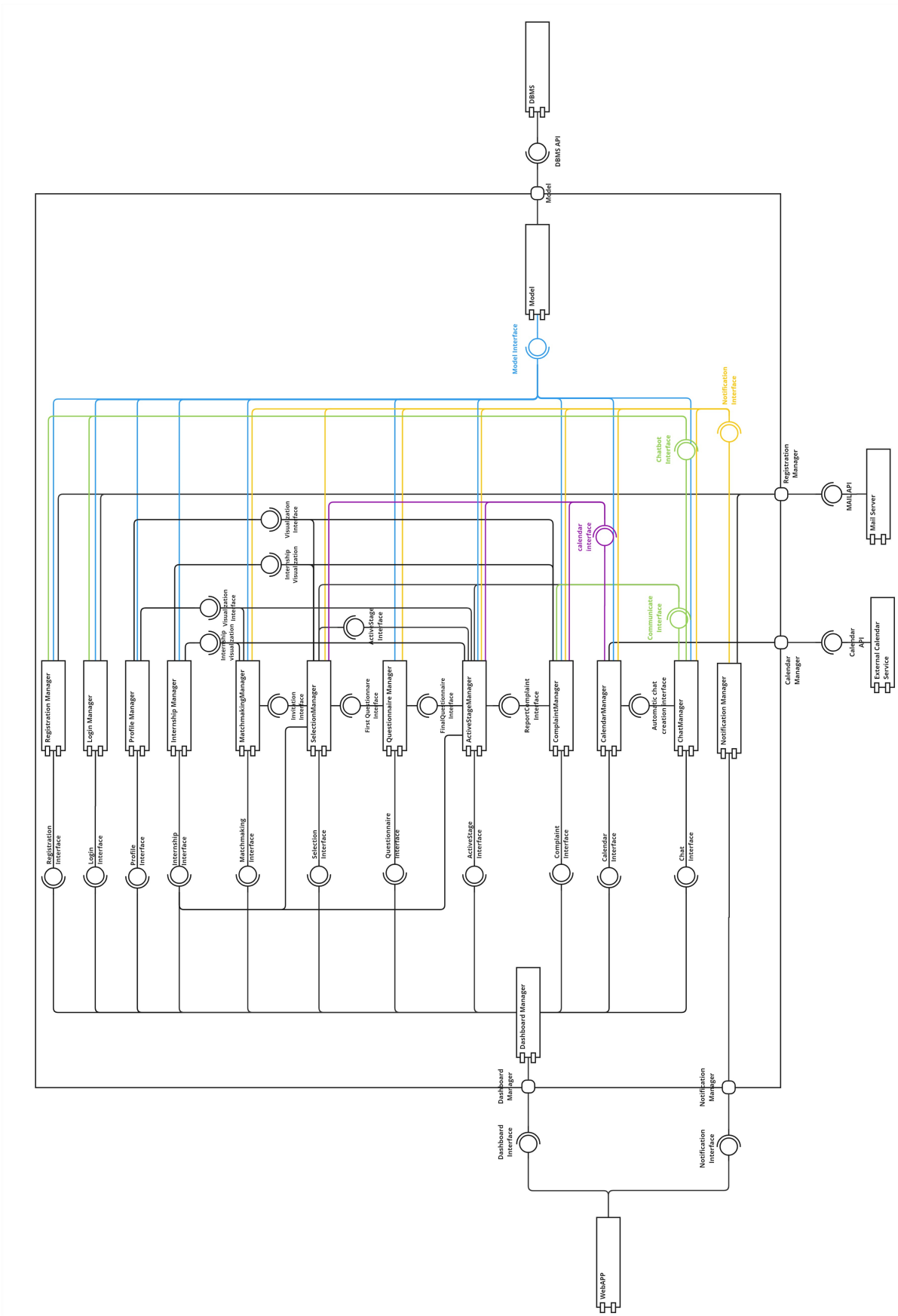

Figure 2.2: UML Component Diagram

2.2.2. Composite Structure Diagrams

Registration Manager

The *Registration Manager* component is composed of five sub-components that handles several processes. This component communicates with *Model* to add the users information to the DMBS and to check if the domain is already inside or not the database. *Registration Manager* also communicates with *(External) Mail Server* through the Mail API for the creation of the institutional profile and for notifying students when their university profile becomes available (as discussed in the RASD).

- **CV Uploader**: it handles the uploading of cv files inside the platform. It checks that the files are of the right size and format. Lastly it "reads" inside the uploaded file and communicate to *Form Handler* to auto-fill all possible fields extracted from the cv.
- **Form Handler**: it is the component that, during the registration phase, allows the user to fill the fields about his/her profile inside the platform. It also allows to edit what was automatically extracted from the cv and to also checks that the mandatory fields are filled and it communicates to the *Domain Checker* for validation. Lastly, this sub-component assist users in completing their registration and allows the user to communicate with the chatbot for assistance.
- **Domain Checker**: it is used to validate the email domain of the users. It checks if the domain exists or not in the database (at such it communicates with *Model*). If the domain does not exists the *Support Handler* comes into play.
- **Institution Profiler**: it sends the mail to the users for the creation of the company or university profile and allows the responsible tutors to actually create the profile for the company or university.
- **Support Handler**: it is used for handling issues related to the registration process. Such as when the email domain is non registered or to confirm that a user is a responsible tutor for the creation of the company or university profile.

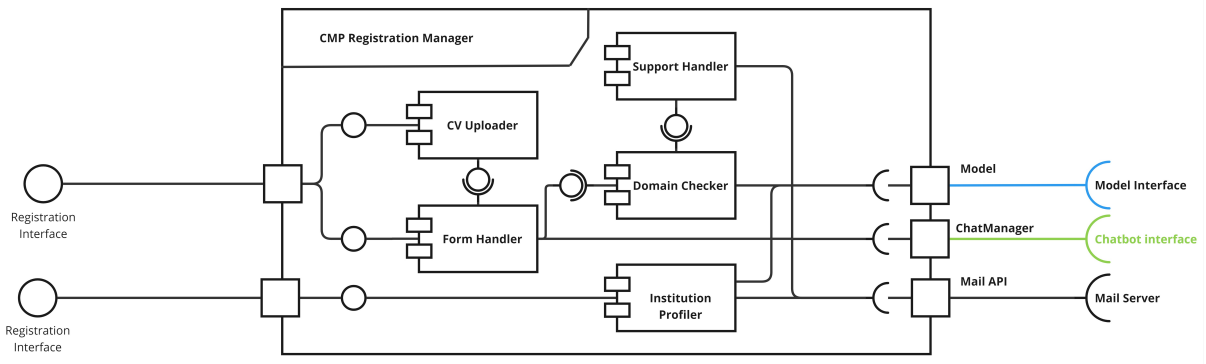

Figure 2.3: UML Component Diagram for *Registration Manager* Component.

Login Manager

Login Manager hands the login process for register Users. When a User attempts to log in the *Dashboard Manger* forwards the request to the *Login Manager* through the login interface. This component is made of two different sub-components.

- **Login Process**: allows a registered user to log in inside his/her profile. It communicates to the *Model* to check if the password and mail match the data saved and to log all the login attempts in the DBMS and with *Chat Manager* to handle the chatbot assistance.
- **Password Recovery**: handles the process of recovering a forgotten password of the profile of a registered users. It communicates with *Model* to access the data (such as the answer to the security questions selected by the user during the registration phase) and to the *Mail Server* to send to the temporary password to his/her email. Lastly, this sub-compontent communicates also with *Chat Manager* to handle the chatbot assistance if the user needs help to reset for the recovery process.

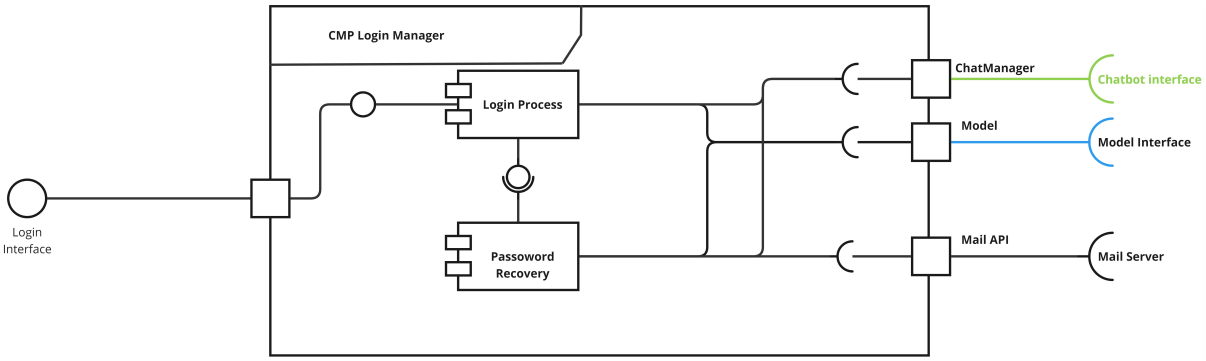

Figure 2.4: UML Component Diagram for *Login Manager* Component.

Profile Manager

This component is made up of three distinct sub-components and all of those sub-components needs to interact with *Model* to read or modify the data saved in the DBMS.

- **Visualize Profile**: allows a (registered) user to search a profile on the S&C platform based on some conditions (filter or keyword). It could be a profile of a student, company tutor, university tutor or either a company or university profile.
- **Modify Profile**: allows a user to modify his/her profile information, as such this sub-component interacts with *Model*.
- **Delete Profile**: allows a user to delete his/her profile from the platform. As for *Modify Profile*, this component interacts with *Model*. It is outside our scope to discuss here a particular process that happens when a responsible tutor deletes his/her profile, in this cause there is a process about moving the "title" of responsible tutor from a user to another user, this process would need to interact with the *Notification Manager* and *Mail Server* to ensure safety.

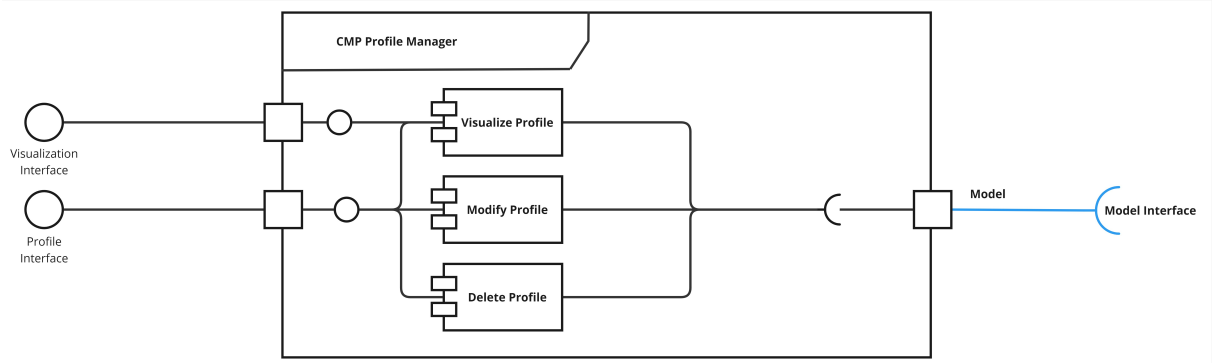

Figure 2.5: UML Component Diagram for *Profile Manager* Component.

Internship Manager

This particularly complex component is made of seven different sub-components.

- **Create/edit Internship**: this is the first step, it allows a category of users (company tutors) to create an internship on the platform or to modify the information of the internship.
- **Draft**: this components allows to save a draft of the internship, it is not needed that all the mandatory fields are completed. It communicates to *Model* to save the draft in the DBMS.
- **Complete Checker**: this component communicates with the two sub-components discussed before and it checks, before publishing an internship, that all the mandatory fields have been correctly filled in.
- **Improve Content**: this sub-component is a powerful tool that can be used to improve the content and description of the internship before publishing it on the platform.
- **Publish Internship**: this sub-component represent the final step and it is used to publish the internship and all its information on the platform to make it visible to all the users.
- **Visualize Internship**: this sub-component allows a user to visualize all the internship information that are available on the platform. As such it communicates with *Model* component.
- **Delete Internship**: this sub-component allow a user (company tutor) to delete an internship from the platform for any kind of reason. It communicates with *Model* to delete the data from the DBMS.

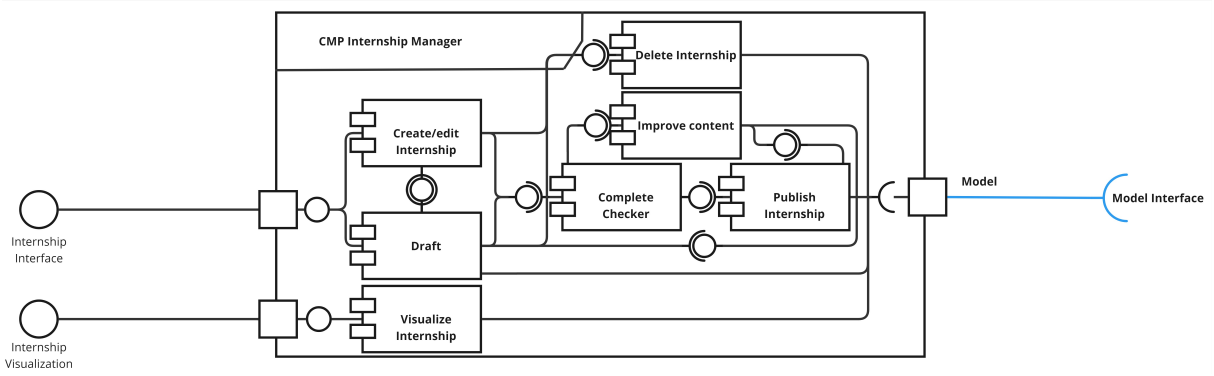

Figure 2.6: UML Component Diagram for *Internship Manager* Component.

Matchmaking Manager

The *Matchmaking Manager* is made of three sub-components.

- **Recommend**: this component for students show the recommended internships, while for company tutors it shows recommended students. It communicates with *Internship Manager*, *Profile Manager* and *Model*.
- **Invite**: it is used to invite a user for the *Selection Manager*. It also notify the invited user, so it communicates also with *Notification Manager*.
- **Search**: it is used to search the profile of a user, a company/university profile or the internship profile. Depending on the required profile it connects to *Internship Manager* or *Profile Manager* to visualize the specific profile.

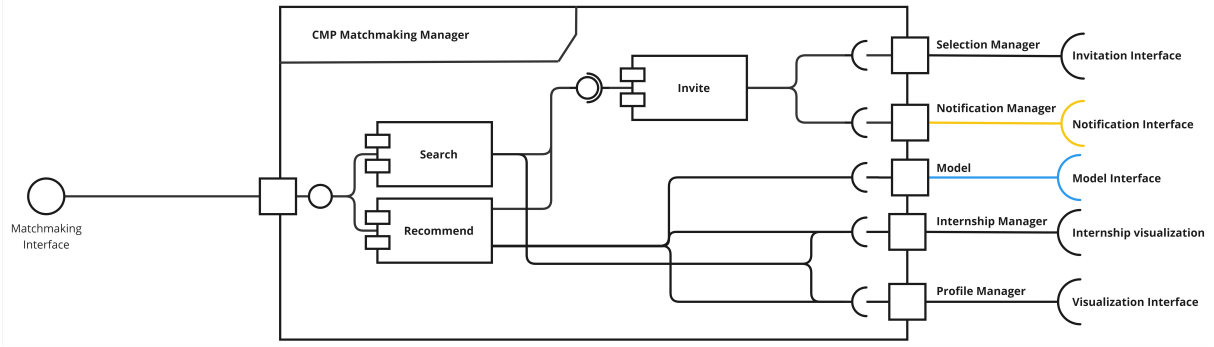

Figure 2.7: UML Component Diagram for *Matchmaking Manager* Component.

Selection Manager

This is one of most complex component since is made of five sub-components and it communicates with many other components. *Selection Manager* focus on the processes that happens after one users sended an initiation through the *Matchmaking Manager*.

- **Consult Invitation**: this is the first sub-component inside *Selection Manager*. Inside this sub-component the user who invited another user (either a student that

applied for an internship or a company tutor that invited a student) can see if the other user have seen or not the invitation. Inside *Consult Invitation* also the invited user can reject or accept the invitation.

- **Meeting Planner**: if the invitation was successful the *Meeting Planner* sub-component is used to plan the meeting and save it in the calendar thanks to *Calendar Interface*.
- **Communicate**: both user can also communicate to discuss further details.
- **Questionnaire**: after the meeting the company tutor use full up the first questionnaire to evaluate student's performance and suitability for the internship.
- **Decision**: this sub-component can be imagined to be made of three other components.
  - **Student Decision**: after the first meeting, this sub-component allows the student to refuse, accept or organize another meeting with the company tutor to further discuss other details related to the internship.
  - **Pick University Tutor**: if the student accept to go further in the internship process without rejecting it, this sub-component handles the process of picking a university tutor and communicate to him/her the student proposal. This sub-component checks that the university tutor belongs to the same university of the student and that he/she accept the student's invitation.
  - **Final Decision**: this sub-component (as the name explains) is the final one and can make an internship starts or not. The company tutor is asked to finalize the internship, if he/she decides to finalize it all the other applications related to the same internship are interrupted and the process communicates to *ActiveStage Manager* component.

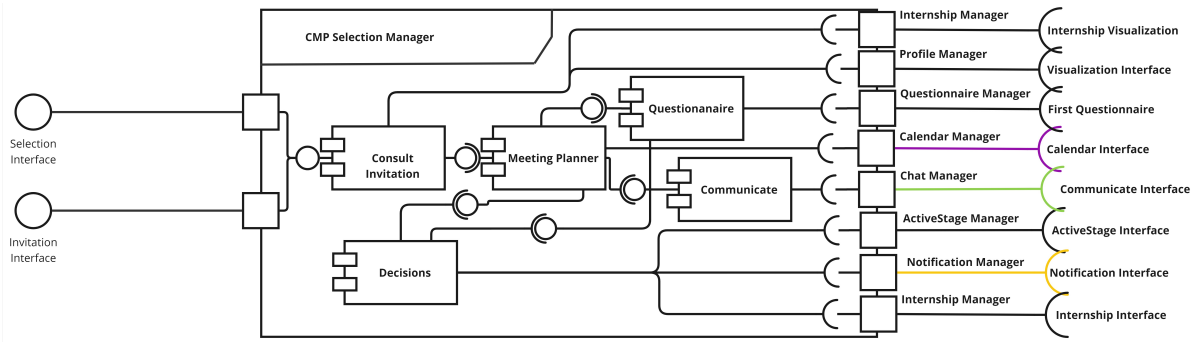

Figure 2.8: UML Component Diagram for *Selection Manager* Component.

ActiveStage Manager

ActiveStageManager is the component that handles everything that can be done by users when an internship is active. It is composed of six different sub-components.

- **Consult Active Stage**: use to check which stage are active. It also communicate with *Internship Manager* and *Profile Manager* to visualize the profiles of the users related to that internship and the internship informations.
- **Final Evaluation**: it is used for the end of an internship. Each participant (student, company tutor and academic tutor) completes a dedicated questionnaire assessing various aspects of the internship and the other participants.
- **End Stage**: set an end to the internship, it is the step after *Final Evaluation* when the questionnaire is completed.
- **Communicate**: this sub-component allows communication with other users while the internship is ongoing.
- **Events Planner**: it is used to plan events such as meetings. It communicates with *Notification Manager* and *Calendar Manager*.
- **Report Complaint**: it is used to report complaint related to the ongoing internship. The complaint than will be handled by another component called *Complain Manager*.

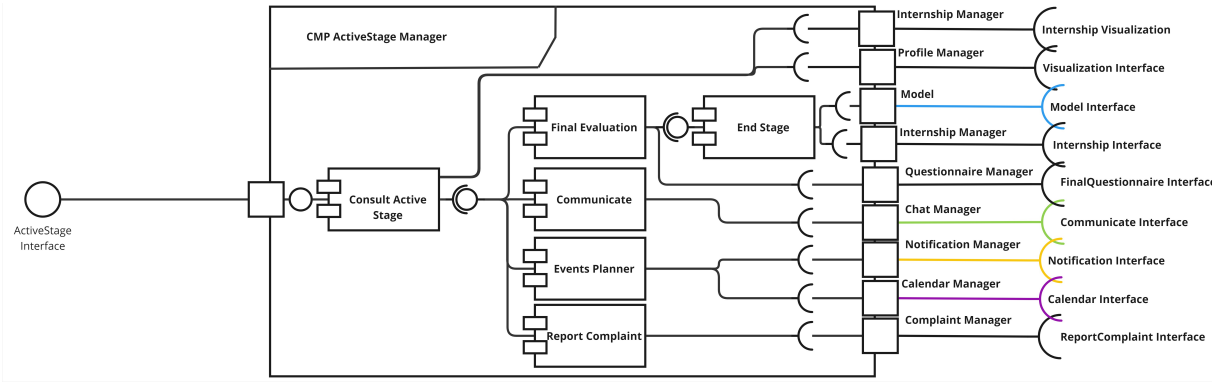

Figure 2.9: UML Component Diagram for *ActiveStage Manager* Component.

Complaint Manager

This component is in charge of handling all the processes related to complaints. It is made of six different sub-components.

- **Complaint Information**: this sub-component handles the creation of the complaint itself. It handles also the filling process of the user (so that he/she can describe the issue) it also assess the severity of the issue and check that all the mandatory information about the complaint have been inserted. Lastly it save and submit the complaint and so this sub-component interacts with *Model* and *Notification Manager* to notify also the other users of the internship.
- **Manage Complaint**: this sub-component allows a user (university tutor) to manage the complaint. It allows a user to load the saved information about a complaint, suggest actions and communicate. Lastly it will log all the actions in the issue resolution problem and can terminate, suspend or resume an internship.

- **Plan Event**: it is the sub-component used for planning events for solving the complaint.
- **Communicate**: allows users to communicate with each other to solve the complaint.
- **Delete complaint/active Stage**: is the sub-component that can put an end to either the complaint (if the complaint is resolved and the issue disappear) or to the active stage (if the complaint is "too big" and this requires the internship to end).
- **Visualize Complaint**: use to see which complaint are still open.

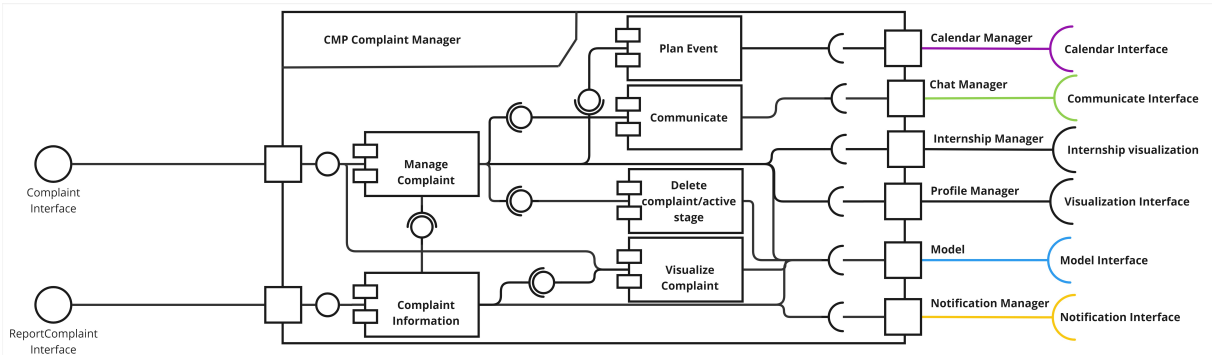

Figure 2.10: UML Component Diagram for *Complaint Manager* Component.

2.3. Deployment View

In this section it is show the allocation of the software components in the physical tiers of the system. It shows how the component previously described are actually deployed in different machines and how each tier is organized. It can be seen in Figure 2.11.

As we said before, we though that a three-tier architecture would be the best structure for the S&C platform. In Figure 2.11 we use different colors for clarity to express the different tier.

- **Presentation Layer**: purple.
- **Application Layer**: green.
- **Data Layer**: orange.

Users can access the S&C platform using any type of personal computer with the most popular operating system (such as *Windows, Mac OS* or *Linux OS* from the most used web browsers (such as *Safari, Google Chrome* or *Microsoft Edge*). Users are not limited to using personal computer and could access the platform also from phones or tables, but pc are preferred. For simplicity in Figure 2.11 we decided to represent only the personal computer.

Firewall: it limits potential attack of intruders by providing strict access rules and checking that the malicious request are not sent to the Reverse Proxy. One of the primary purposes

is to create a barrier between a trusted internal network and untrusted external network to protect against unauthorized access, malware and cyberattacks.

Reverse Proxy: it acts as an intermediary that forwards the client's requests to the S&C servers. It is used to increase scalability, performance, resilience and security. The idea behind is that a user "interacts" only with the reverse proxy (after the firewall) and so he/she remains unaware of the backend server's IP address. Lastly, a reverse proxy is also in charge of the load balancing of all the request received from internet that are filtered from the firewall.

The S&C servers manage all system functionalities. They are deployed as three identical nodes (three in the image, but could be more) to handle higher volumes of traffic and maintain availability even if one server fails. The Reverse Proxy load-balances incoming requests among these nodes. They communicate with the data layer in order to store, delete or modify useful data. All the application logic components are deployed here.

Lastly in the database server there is a relational DBMS that allows to store and retrieve all the data needed by the S&C server.

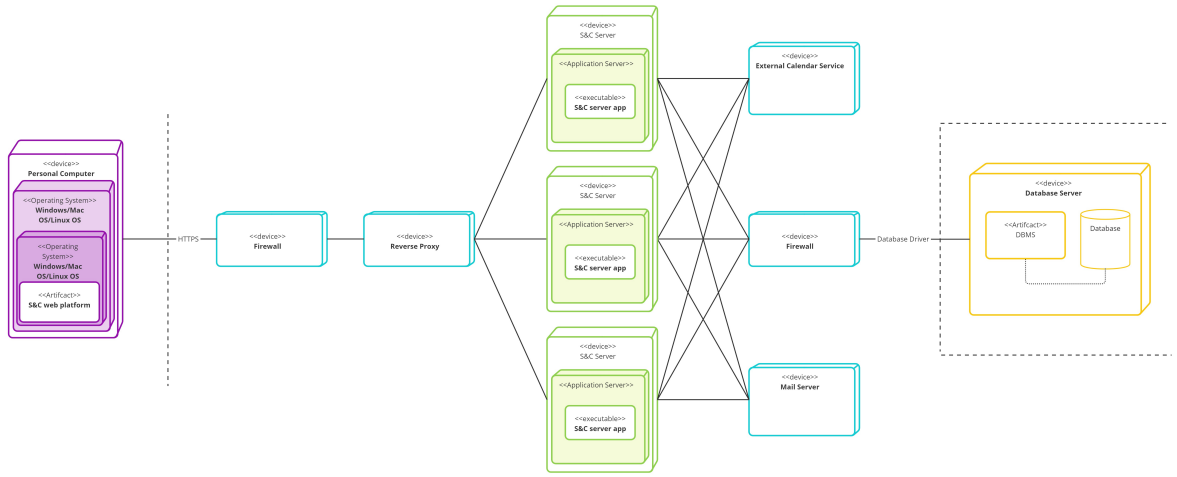

Figure 2.11: Deployment Diagram.

2.4. Runtime View

While the Component View in Section 2.2 focuses on the static behavior of the system, this section shifts the attention to the dynamic behavior and how components interact during their execution. It examines in detail how the system processes data or responds to events in real time, describing how components collaborate to fulfill functionalities such as authentication and access management, internship creation and publication, monitoring active internships, applications and updates.

2.4.1. Sequence Diagrams

The UML Sequence Diagrams presented in this section offer a detailed depiction of the system's dynamic behavior by illustrating the sequence of interactions between components over time. Each diagram corresponds to a specific use case defined in the RASD, capturing the flow of events and communication required to achieve the intended functionality. These diagrams not only demonstrate how the system processes user actions and external triggers under normal conditions but also provide insight into alternative and error-handling scenarios, ensuring a comprehensive understanding of the system's runtime processes.

[UC1]: Access the S&C Platform

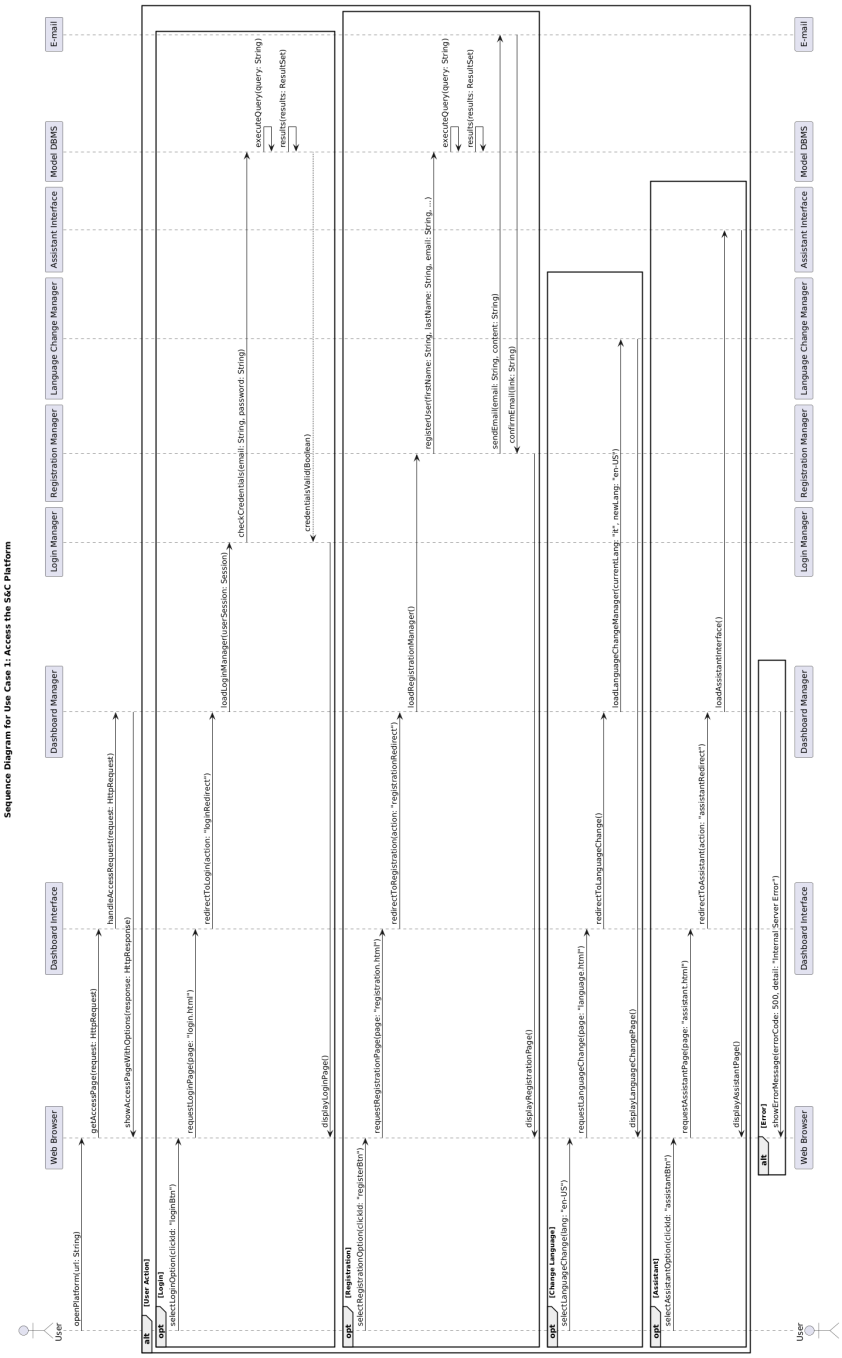

Figure 2.12: Sequence Diagram for Use Case 1: Accessing the S&C Platform.

[UC2]: Overview Page

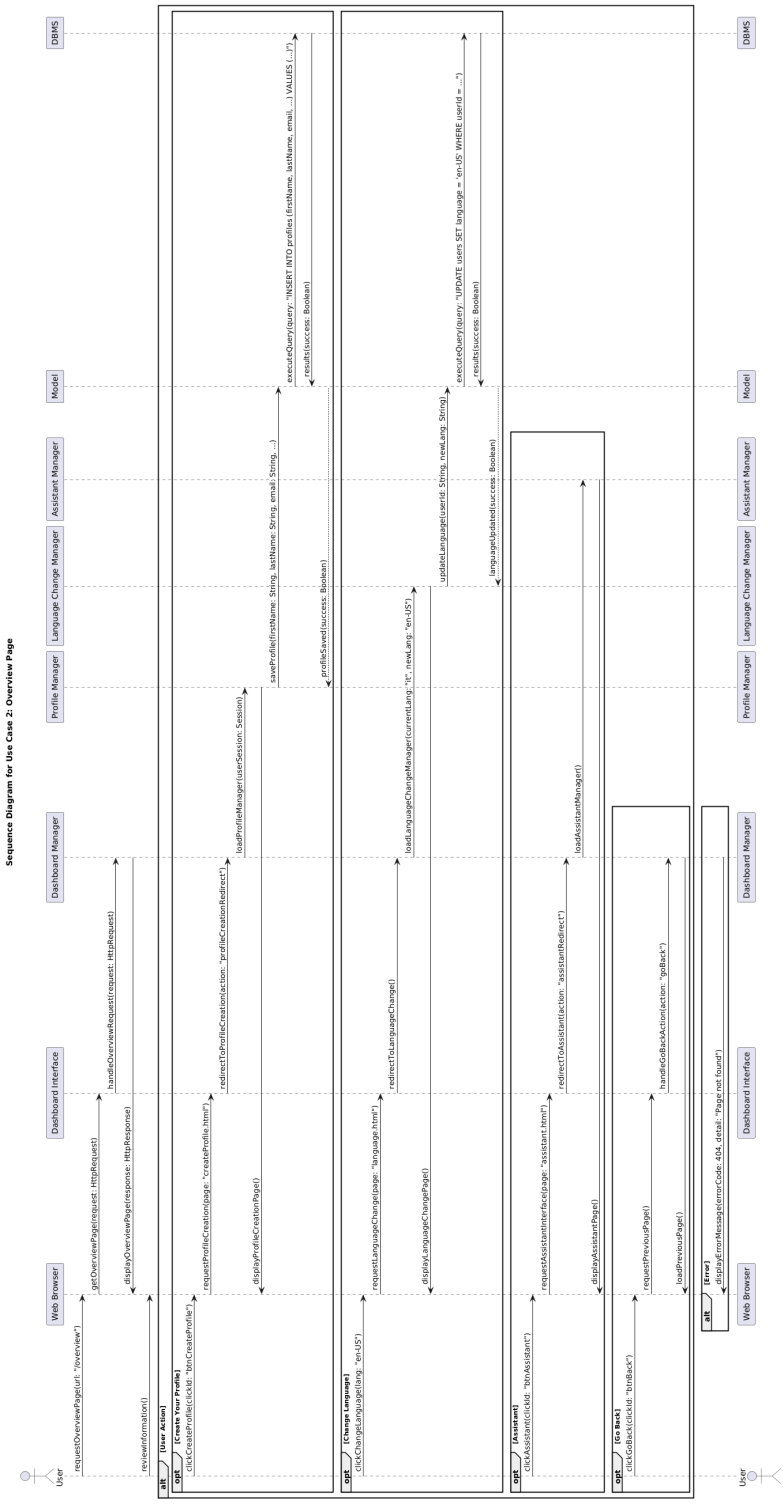

Figure 2.13: Sequence Diagram for Use Case 2: Overview Page.

[UC3]: CV Upload

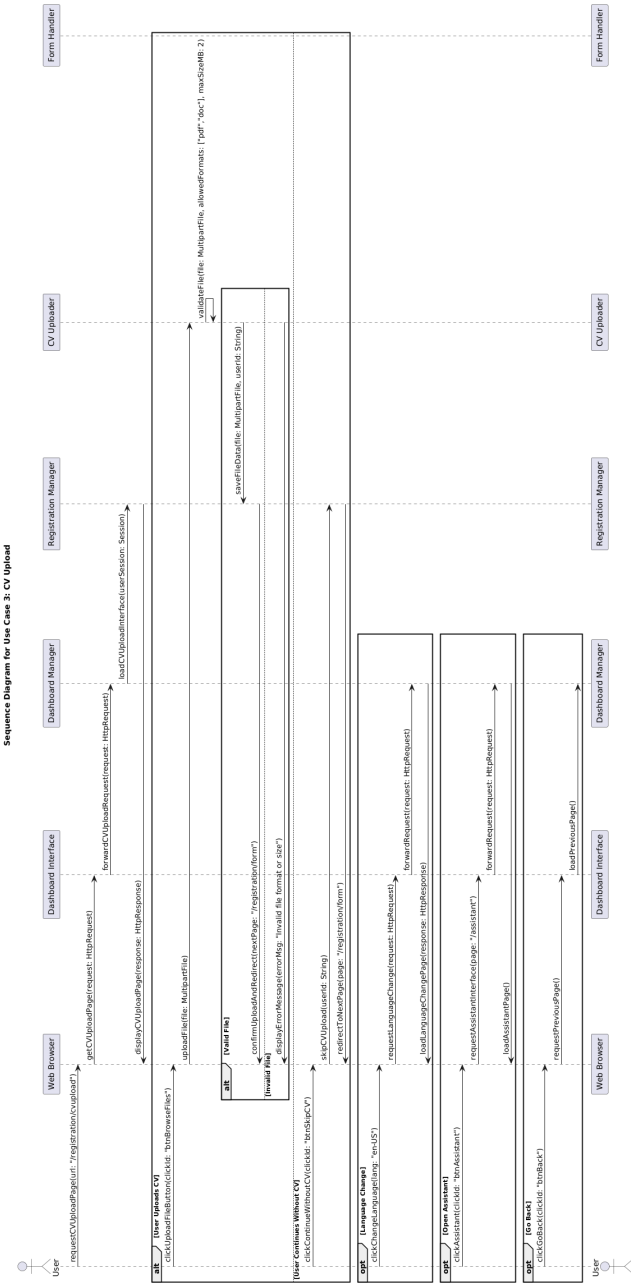

Figure 2.14: Sequence Diagram for Use Case 3: CV Upload.

[UC4]: User Profile Registration

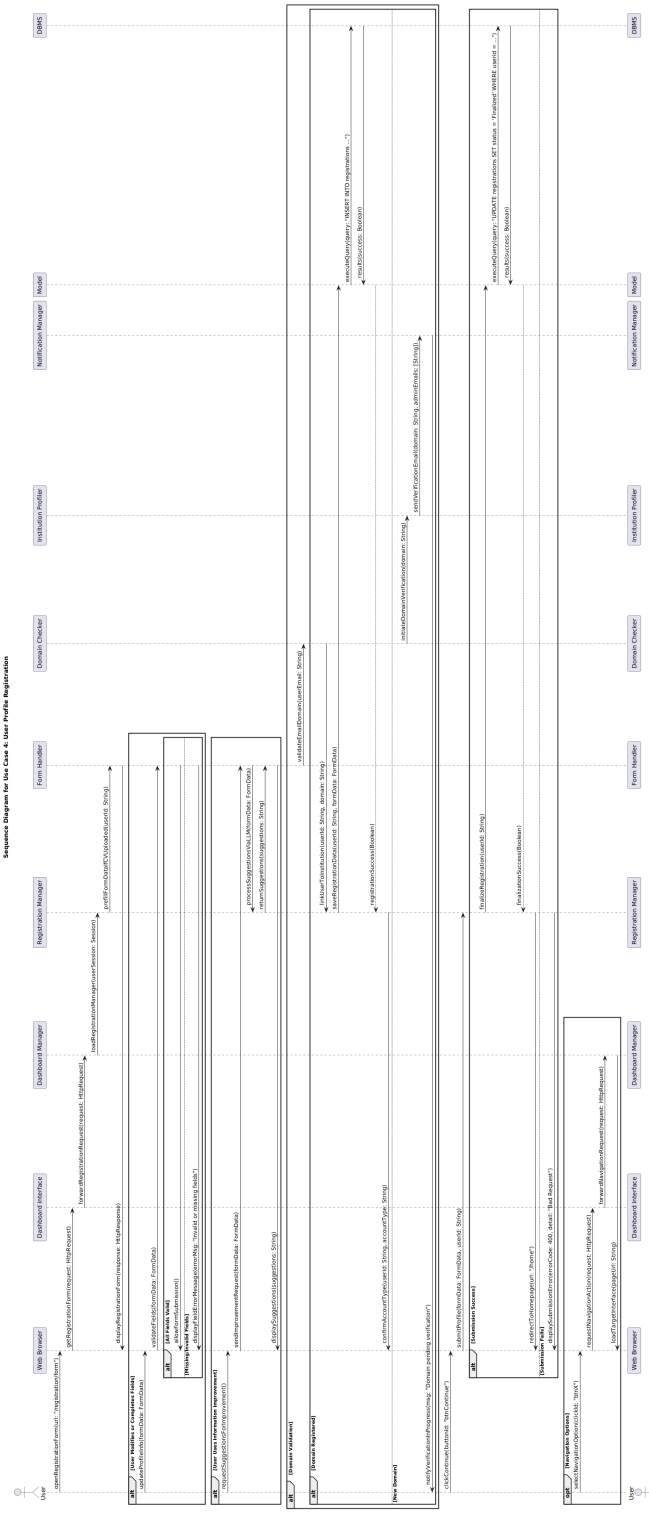

Figure 2.15: Sequence Diagram for Use Case 4: User Profile Registration.

[UC5]: Domain Verification

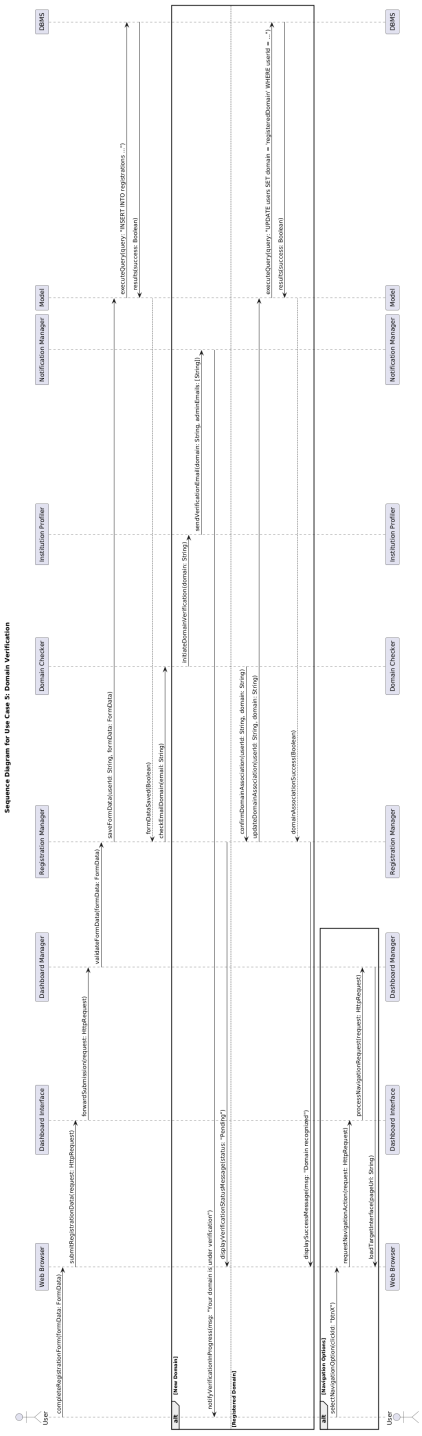

Figure 2.16: Sequence Diagram for Use Case 5: Domain Verification.

[UC6]: Institution Profile Registration

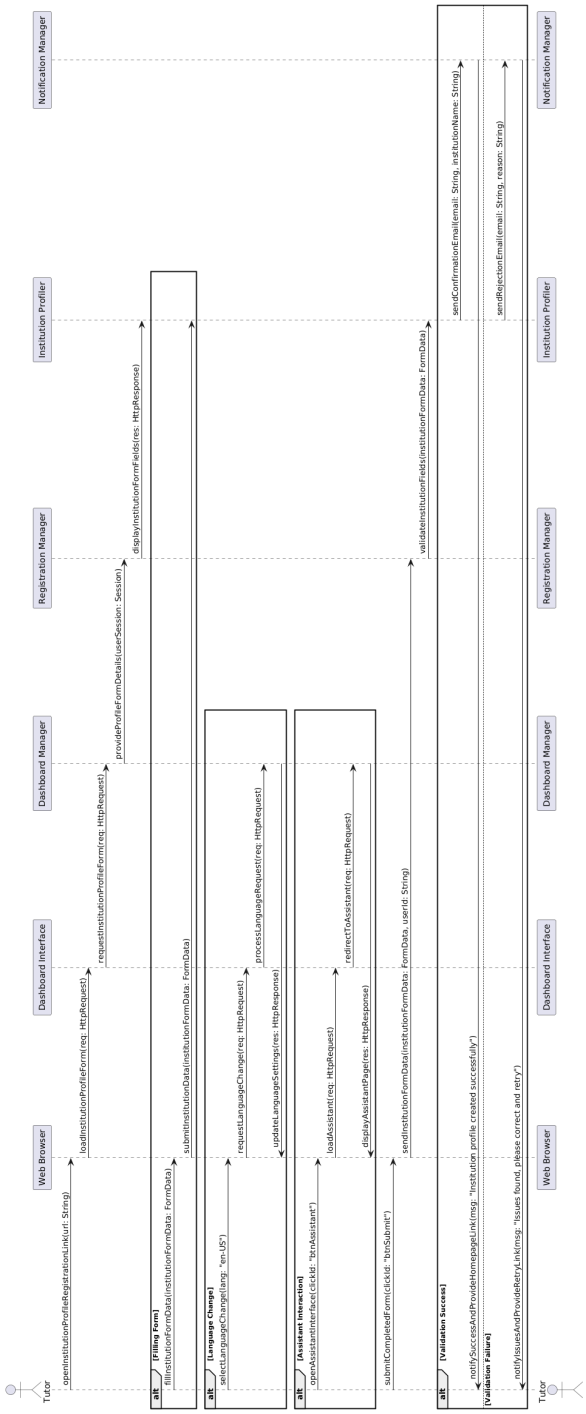

Figure 2.17: Sequence Diagram for Use Case 6: Institution Profile Registration.

[UC7]: User Login

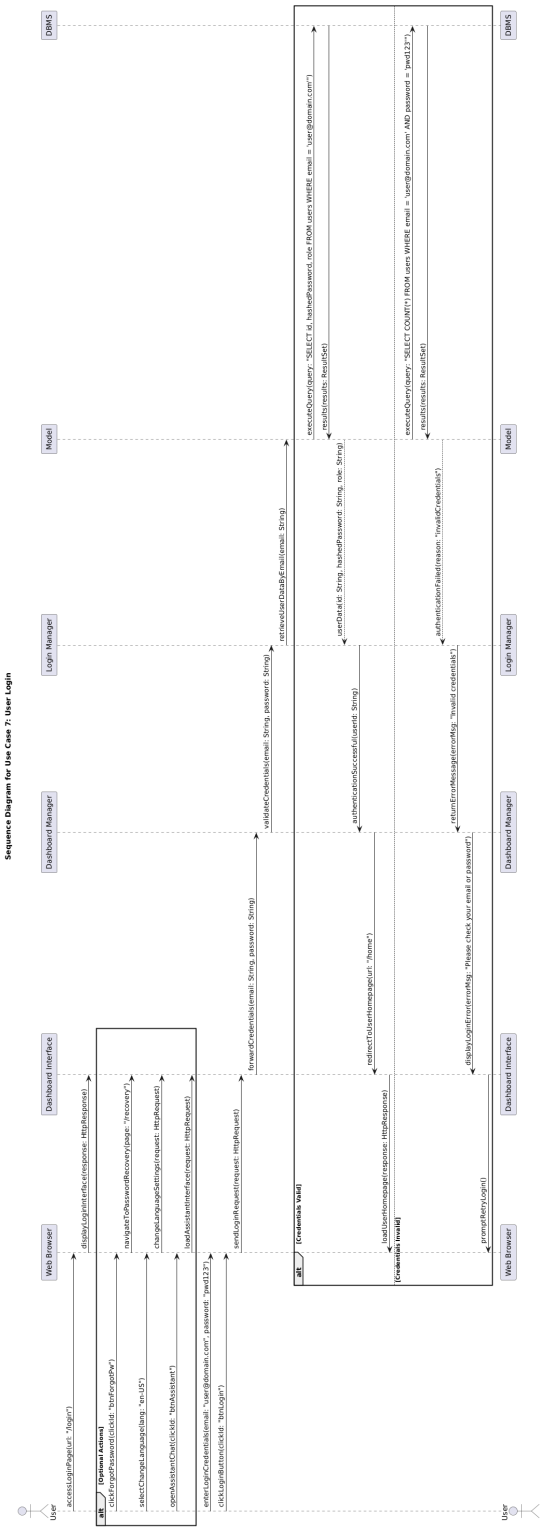

Figure 2.18: Sequence Diagram for Use Case 7: User Login.

[UC8]: Password Recovery

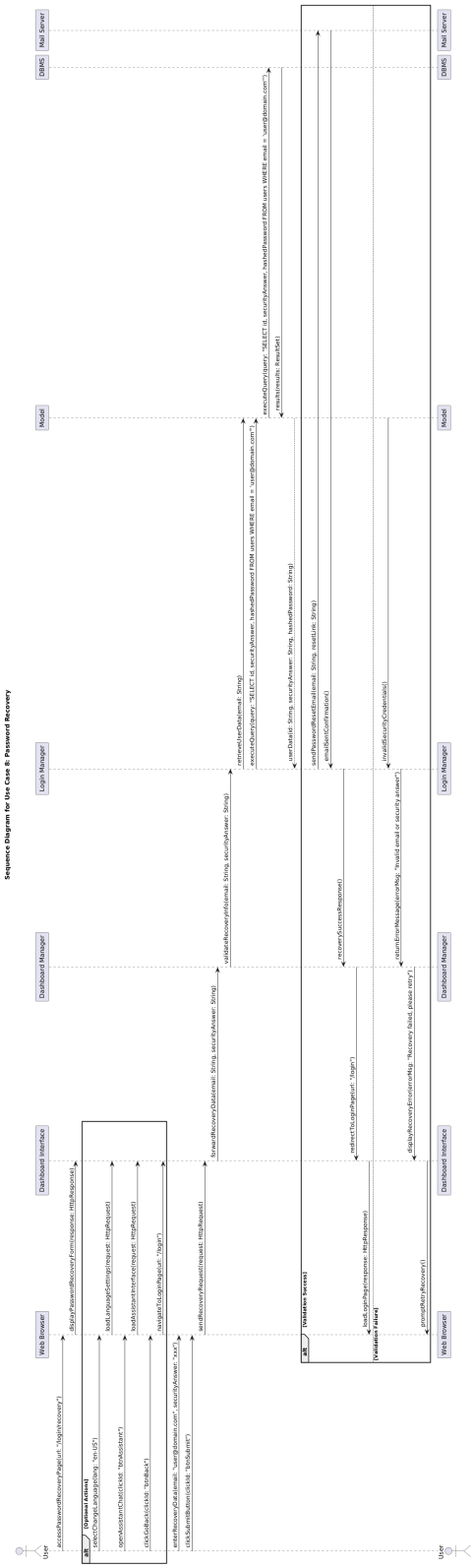

Figure 2.19: Sequence Diagram for Use Case 8: Password Recovery.

[UC9]: Homepage Access

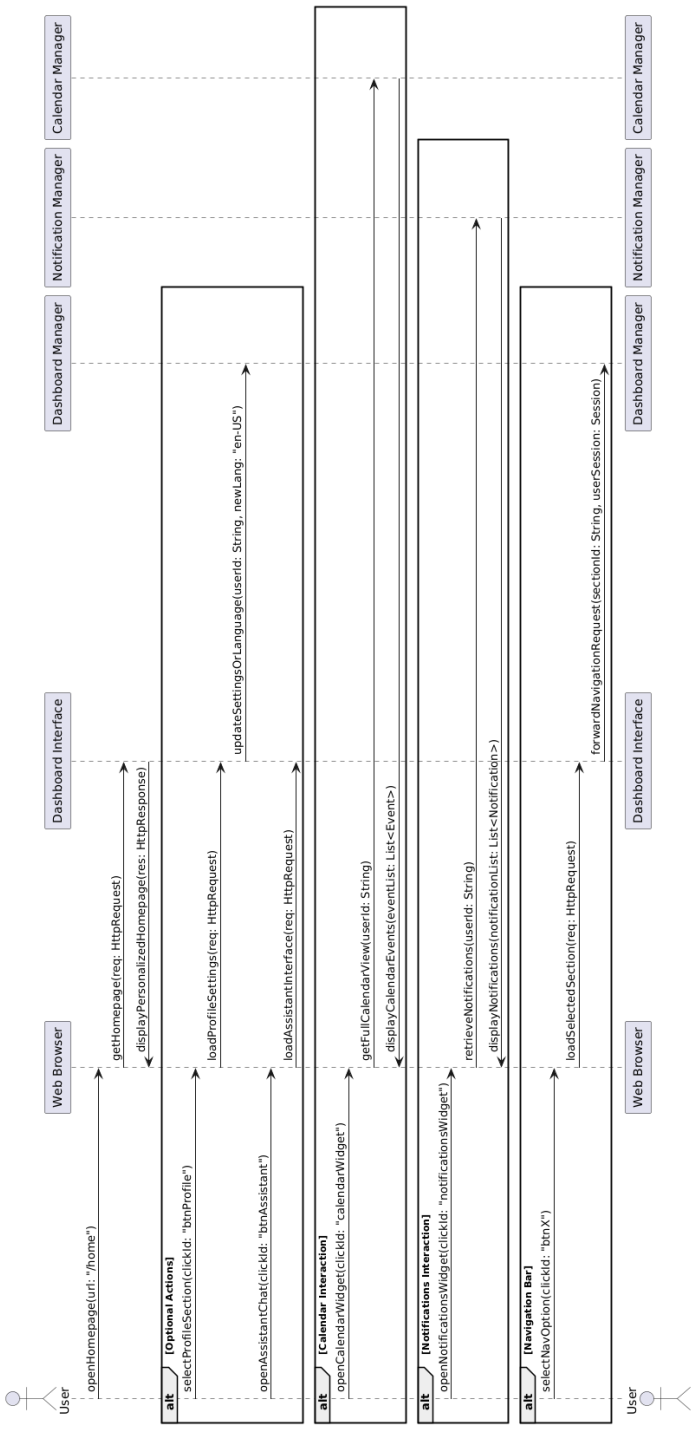

Figure 2.20: Sequence Diagram for Use Case 9: Homepage Access.

[UC10]: Change Language

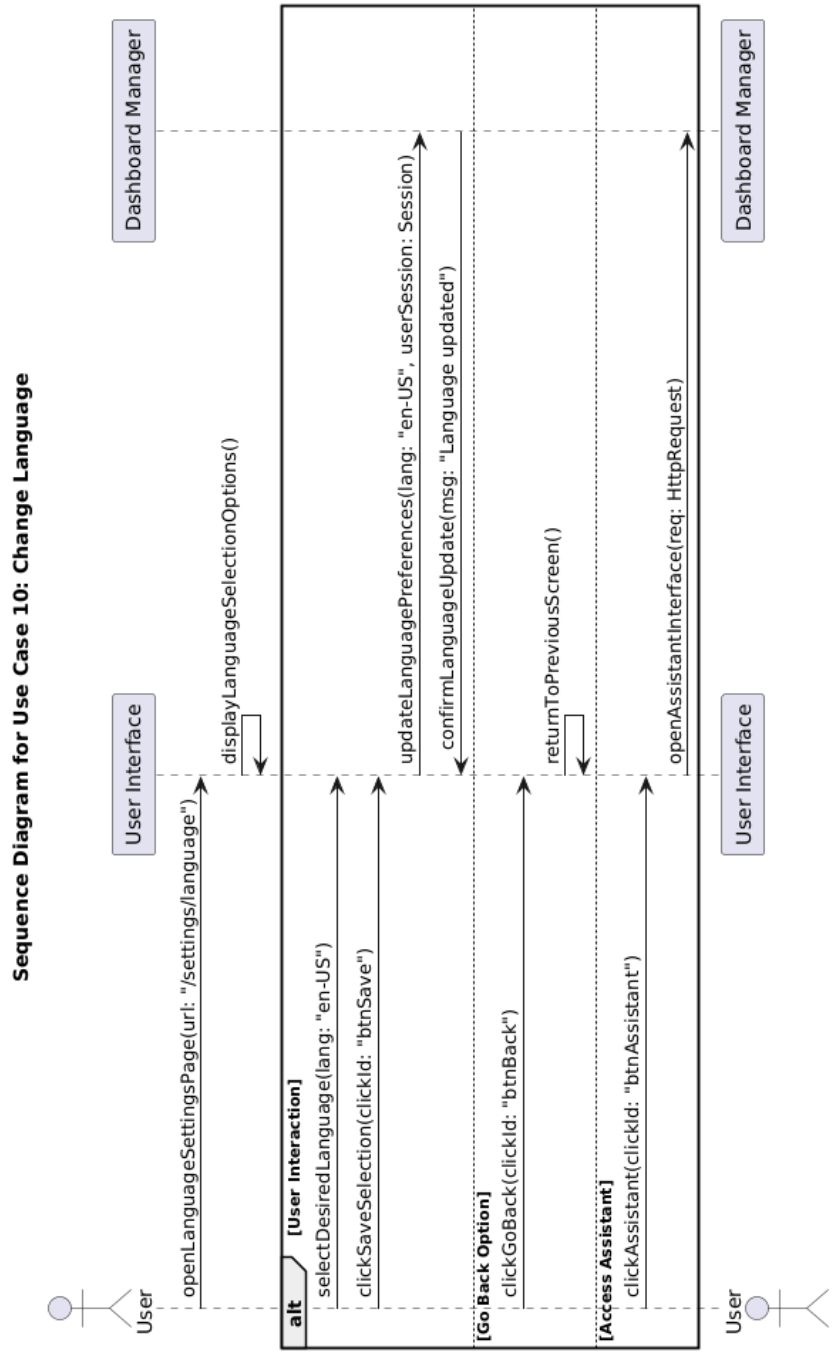

Figure 2.21: Sequence Diagram for Use Case 10: Change Language.

[UC11]: Virtual Assistant Interaction

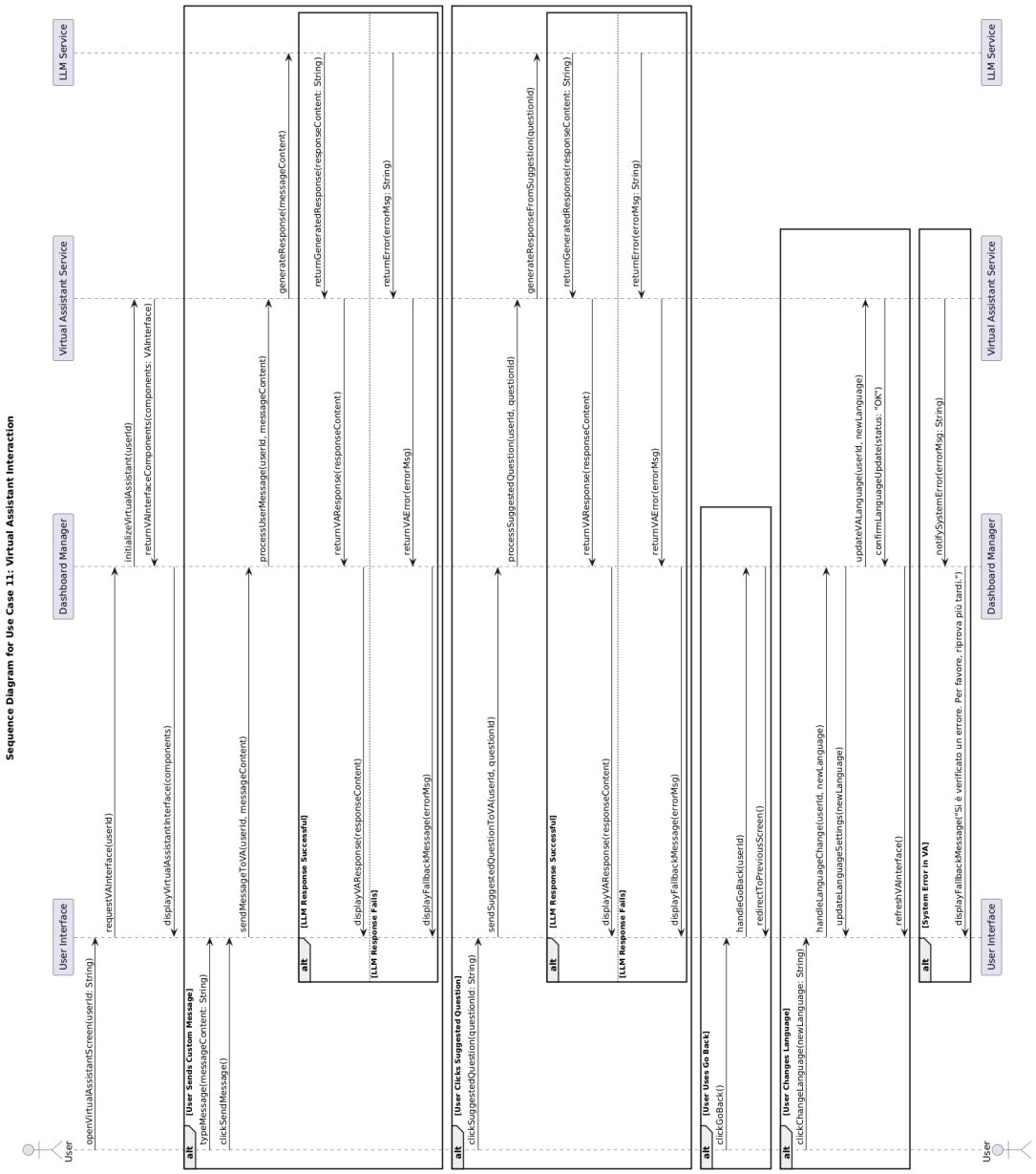

Figure 2.22: Sequence Diagram for Use Case 11: Virtual Assistant Interaction.

[UC12]: Settings Management

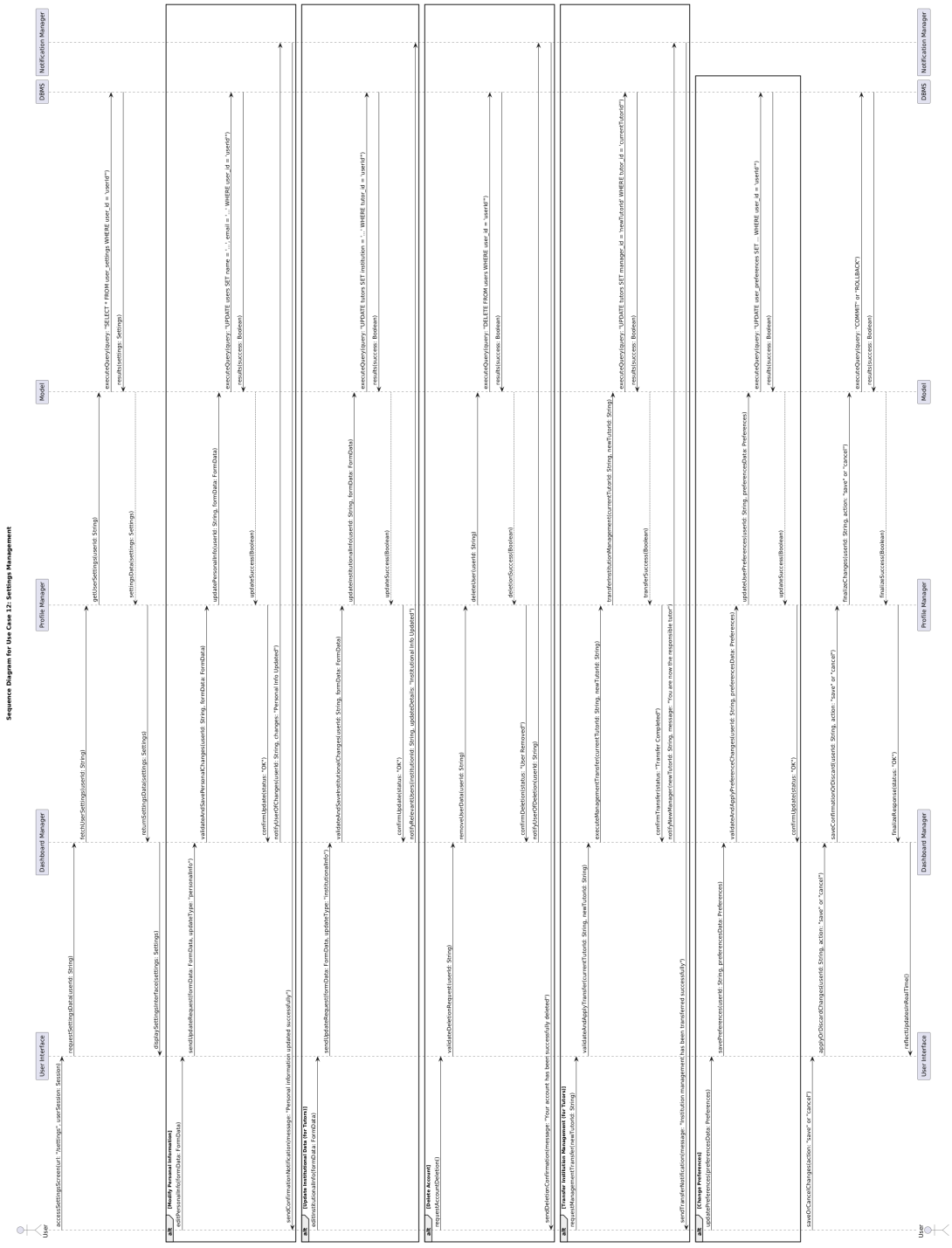

Figure 2.23: Sequence Diagram for Use Case 12: Settings Management.

[UC13]: Matchmaking

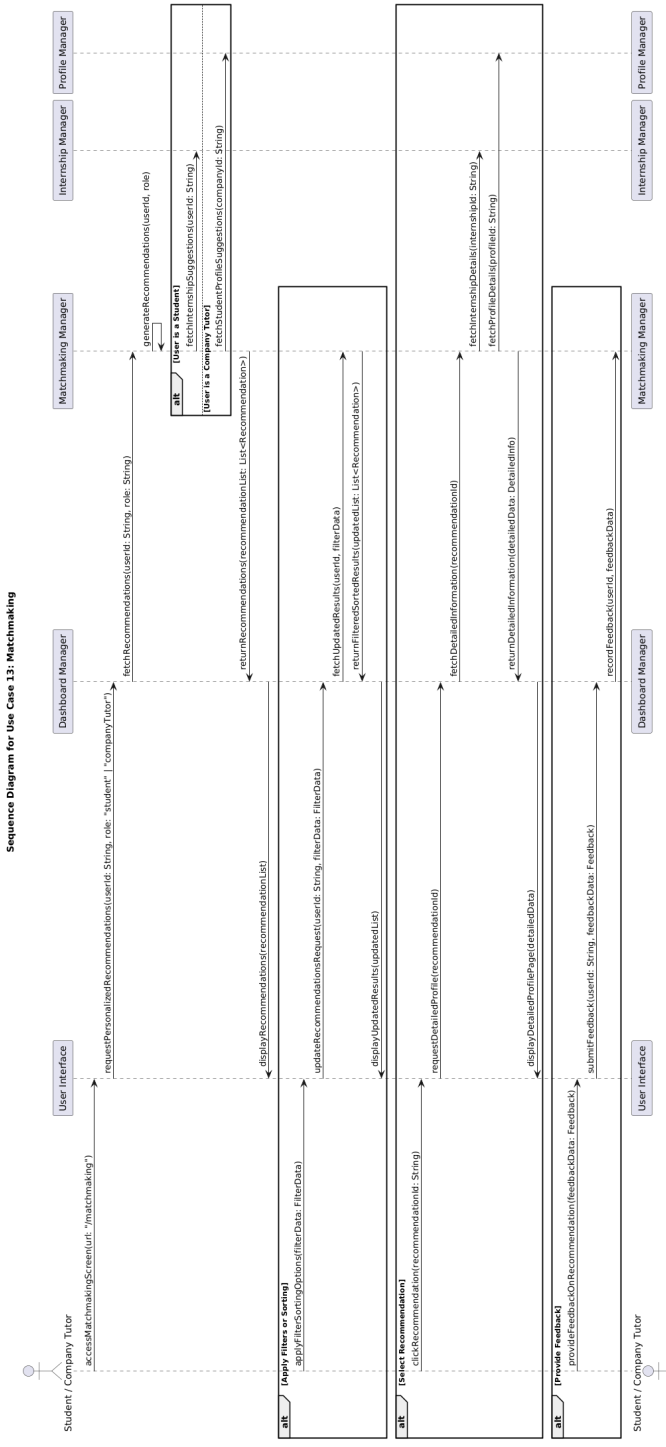

Figure 2.24: Sequence Diagram for Use Case 13: Matchmaking.

[UC14]: Calendar Management

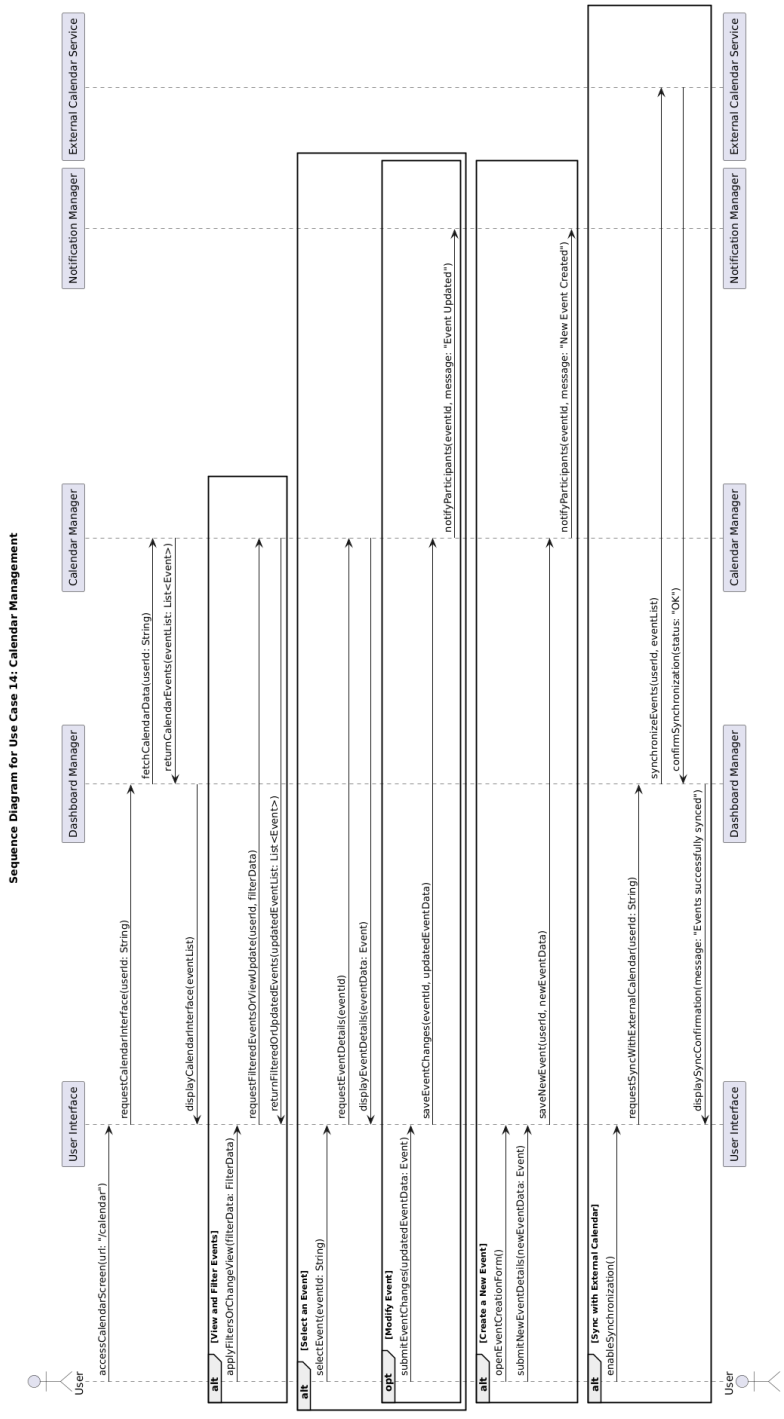

Figure 2.25: Sequence Diagram for Use Case 14: Calendar Management.

[UC15]: Event Viewing

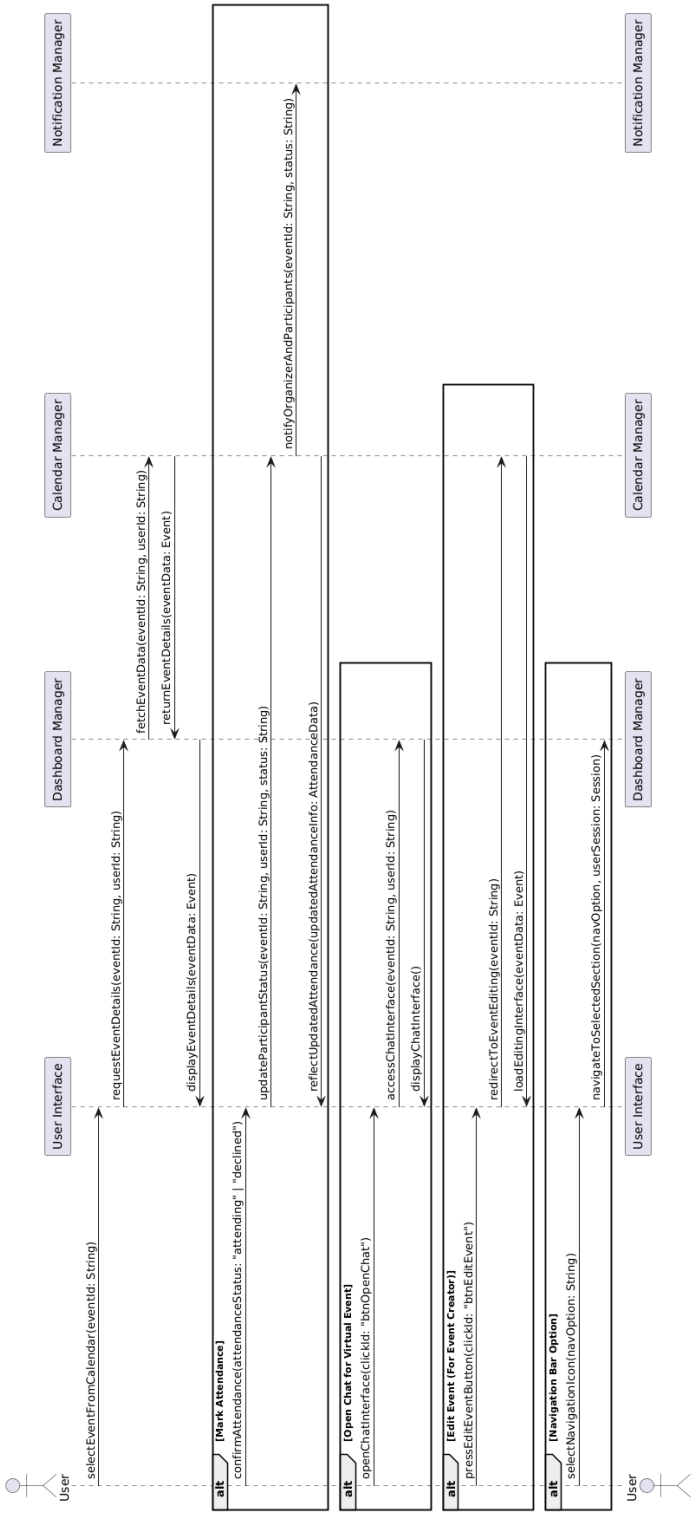

Figure 2.26: Sequence Diagram for Use Case 15: Event Viewing.

[UC16]: Event Creation and Management

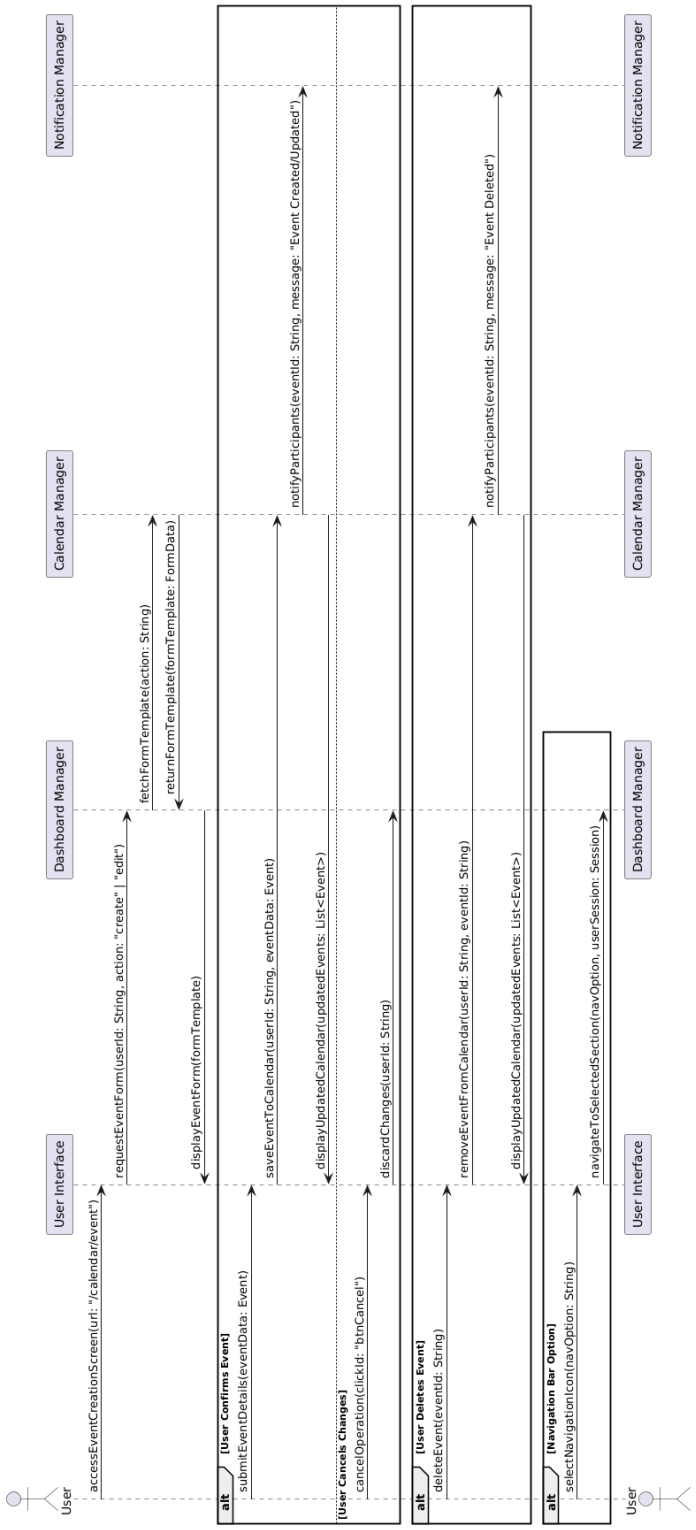

Figure 2.27: Sequence Diagram for Use Case 16: Event Creation and Management.

[UC17]: Messaging Interface

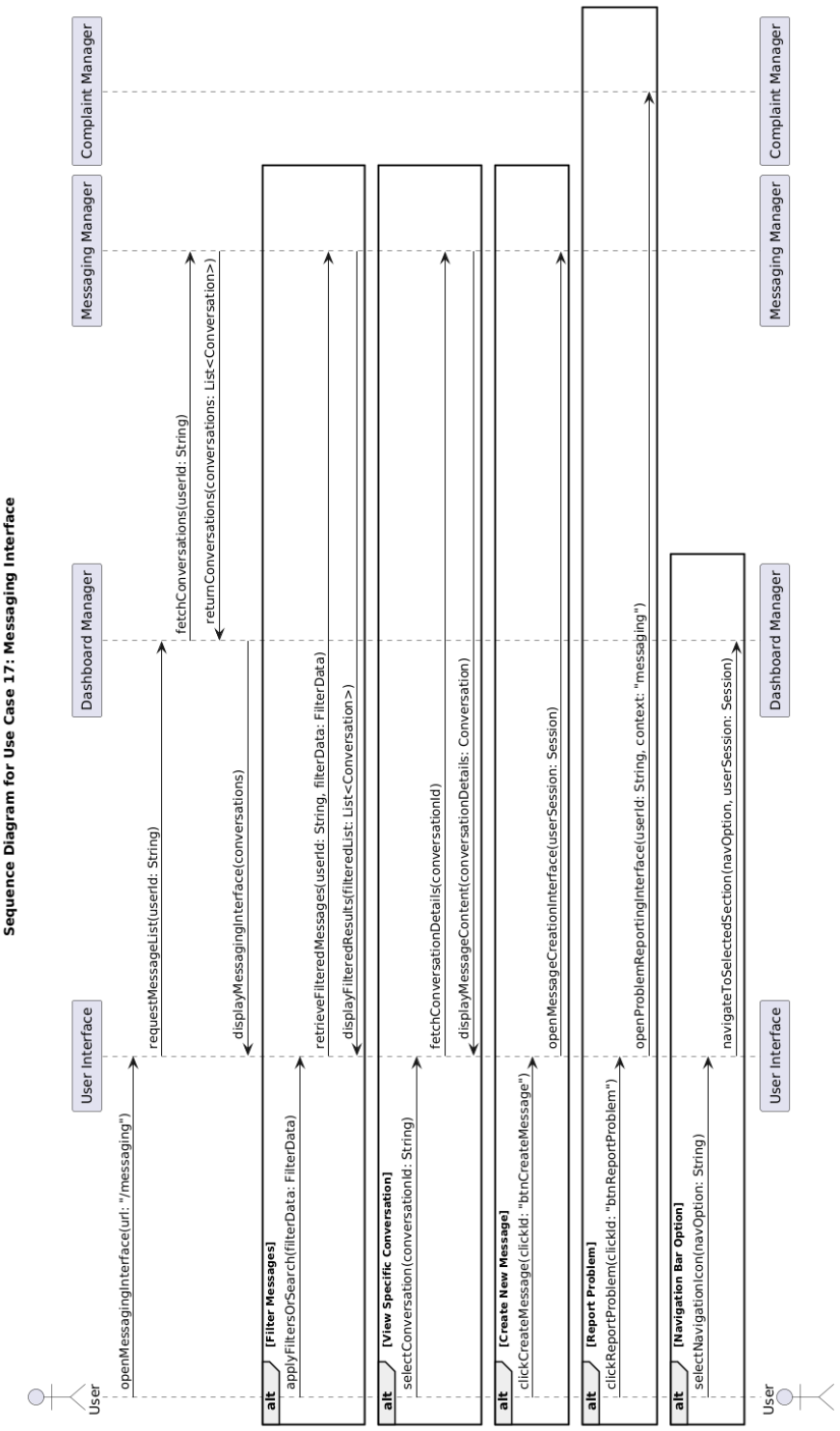

Figure 2.28: Sequence Diagram for Use Case 17: Messaging Interface.

[UC18]: Videocall Chat Interaction

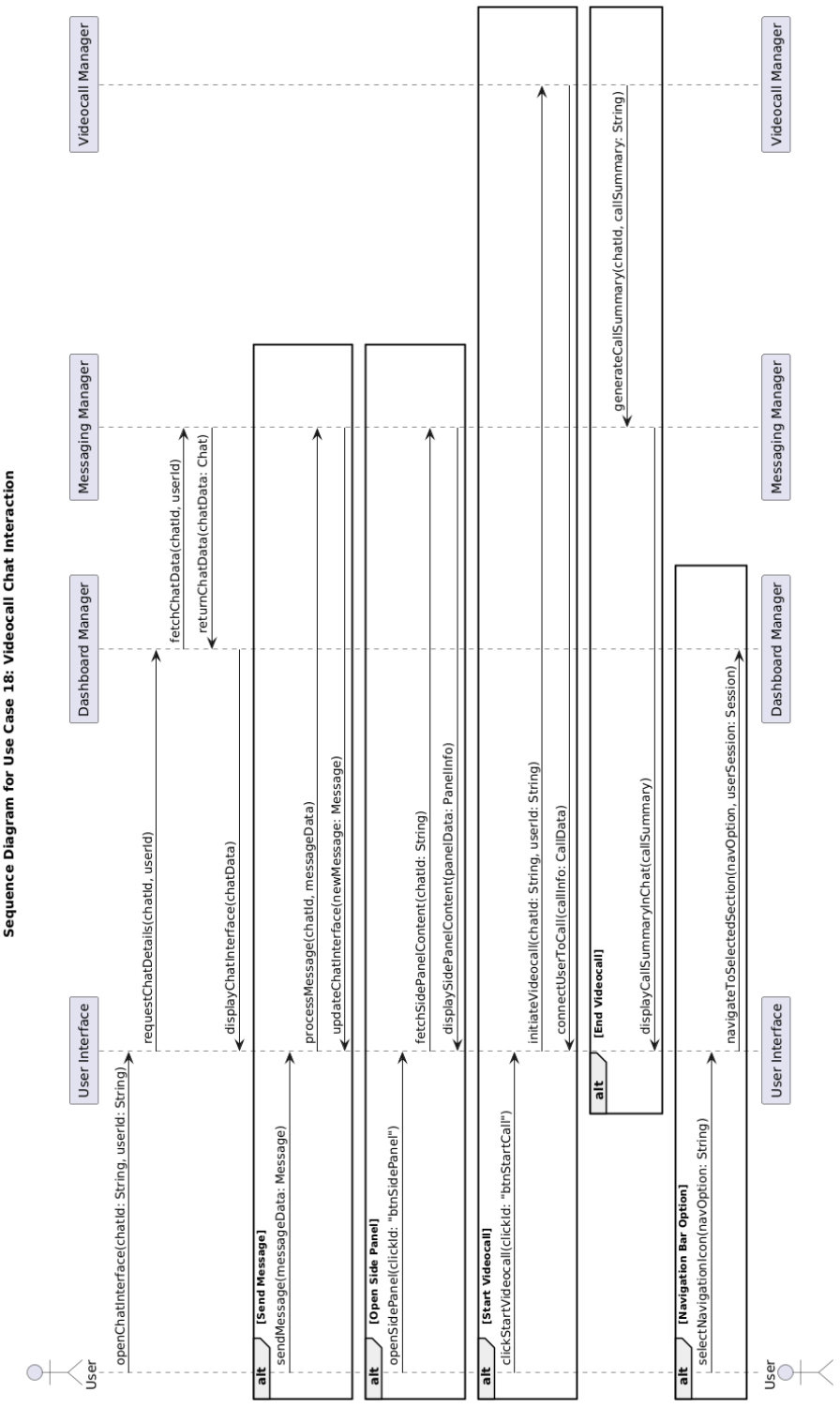

Figure 2.29: Sequence Diagram for Use Case 18: Videocall Chat Interaction.

[UC19]: Complaints Chat Interaction

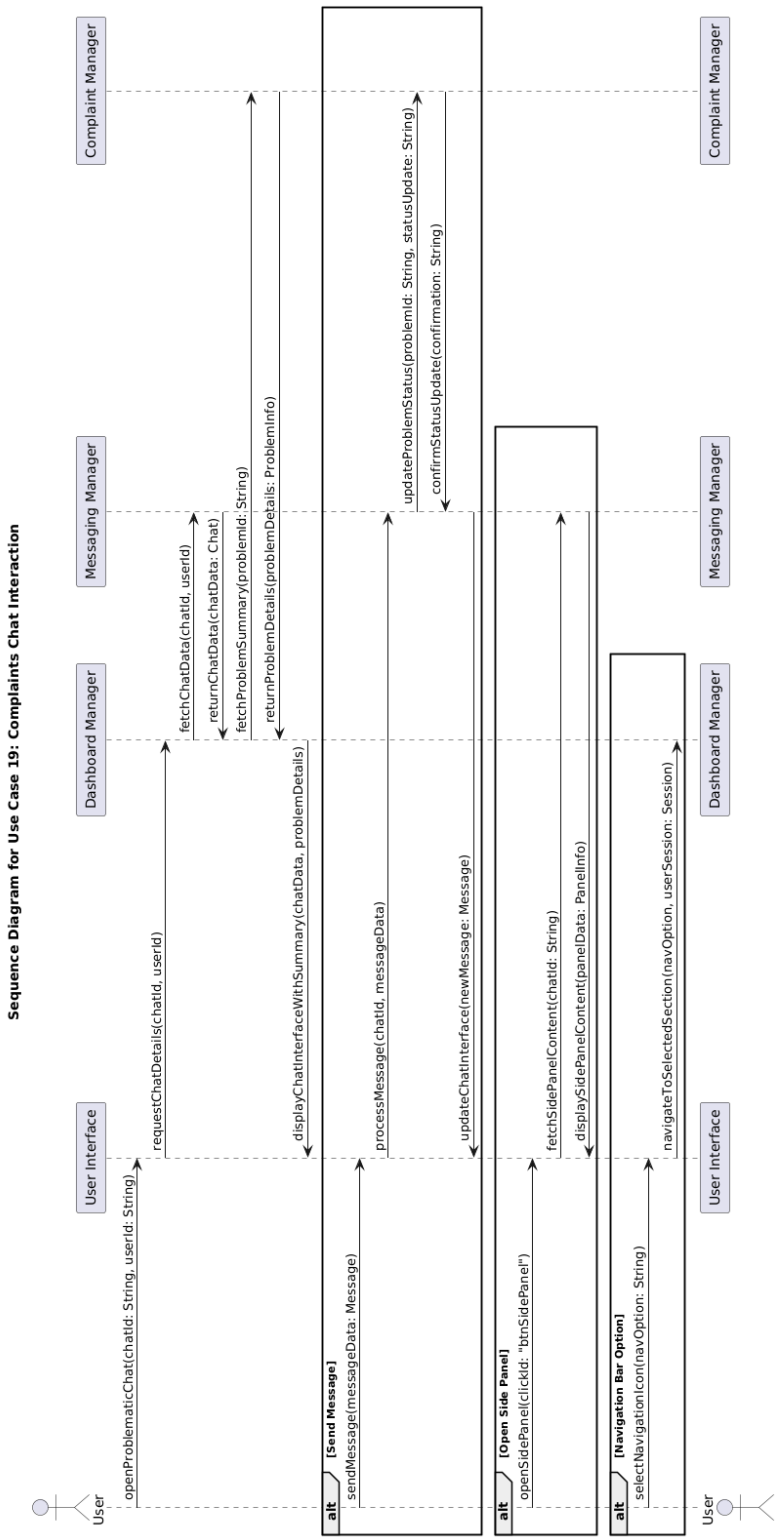

Figure 2.30: Sequence Diagram for Use Case 19: Complaints Chat Interaction.

[UC20]: Report Issues

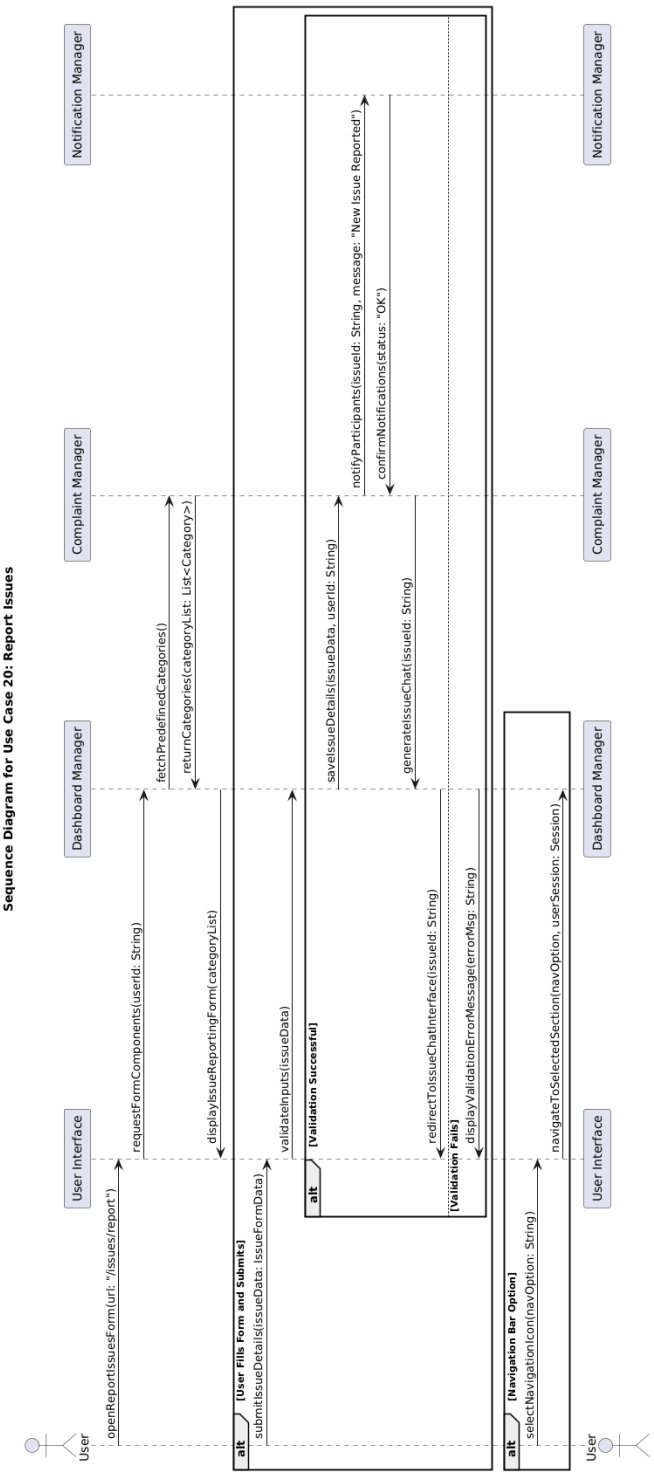

Figure 2.31: Sequence Diagram for Use Case 20: Report Issues.

[UC21]: Manage Issues

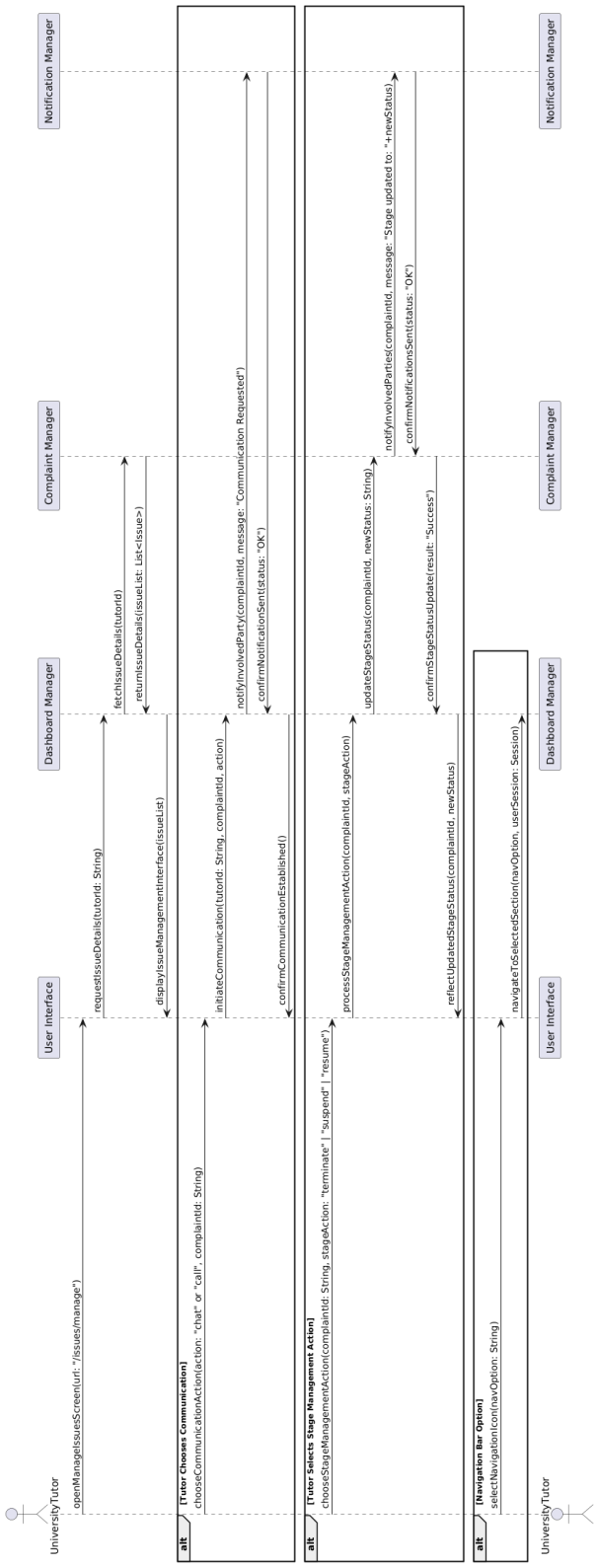

Figure 2.32: Sequence Diagram for Use Case 21: Manage Issues.

[UC22]: Selection Process Monitoring

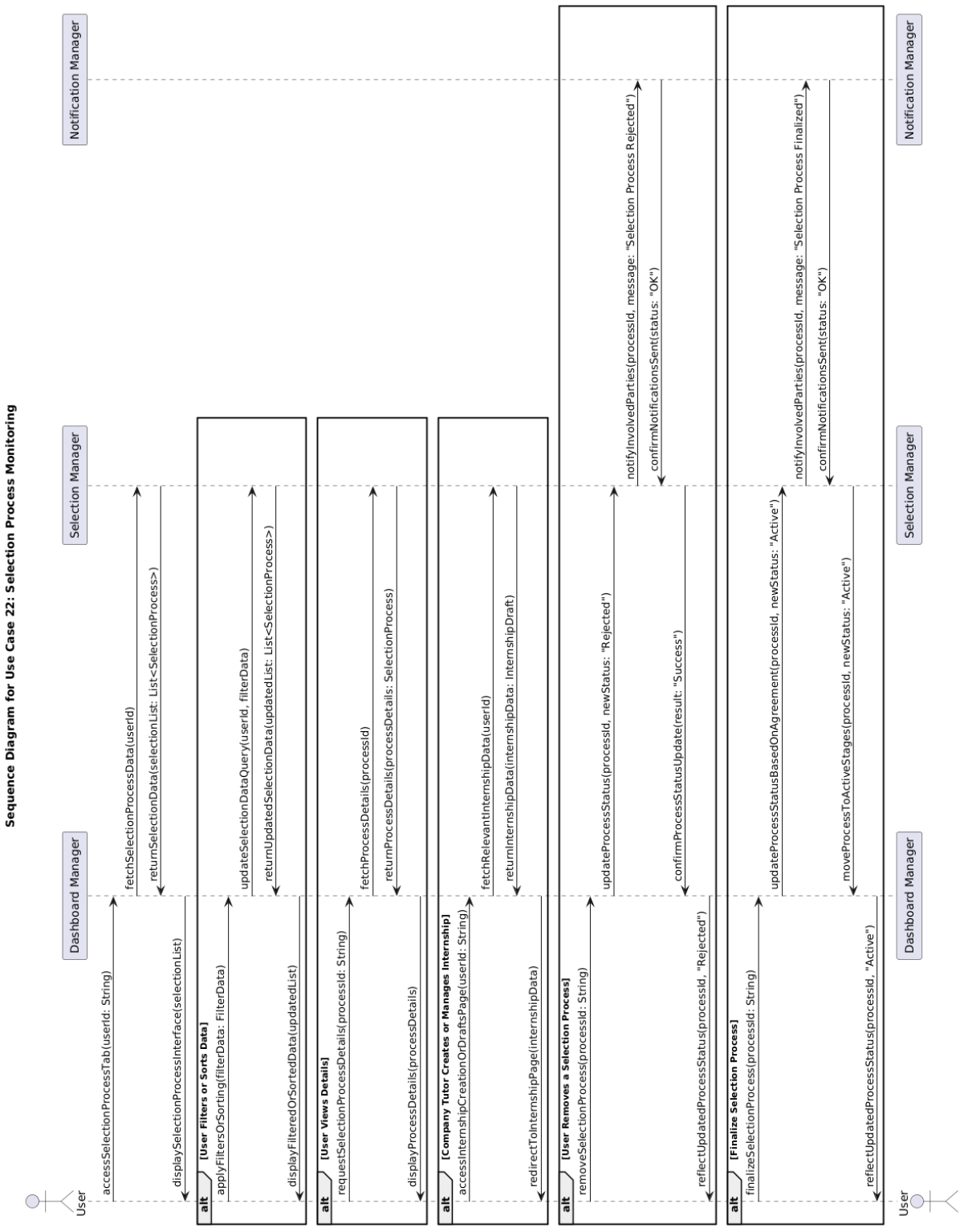

Figure 2.33: Sequence Diagram for Use Case 22: Selection Process Monitoring.

[UC23]: Active Stages Monitoring

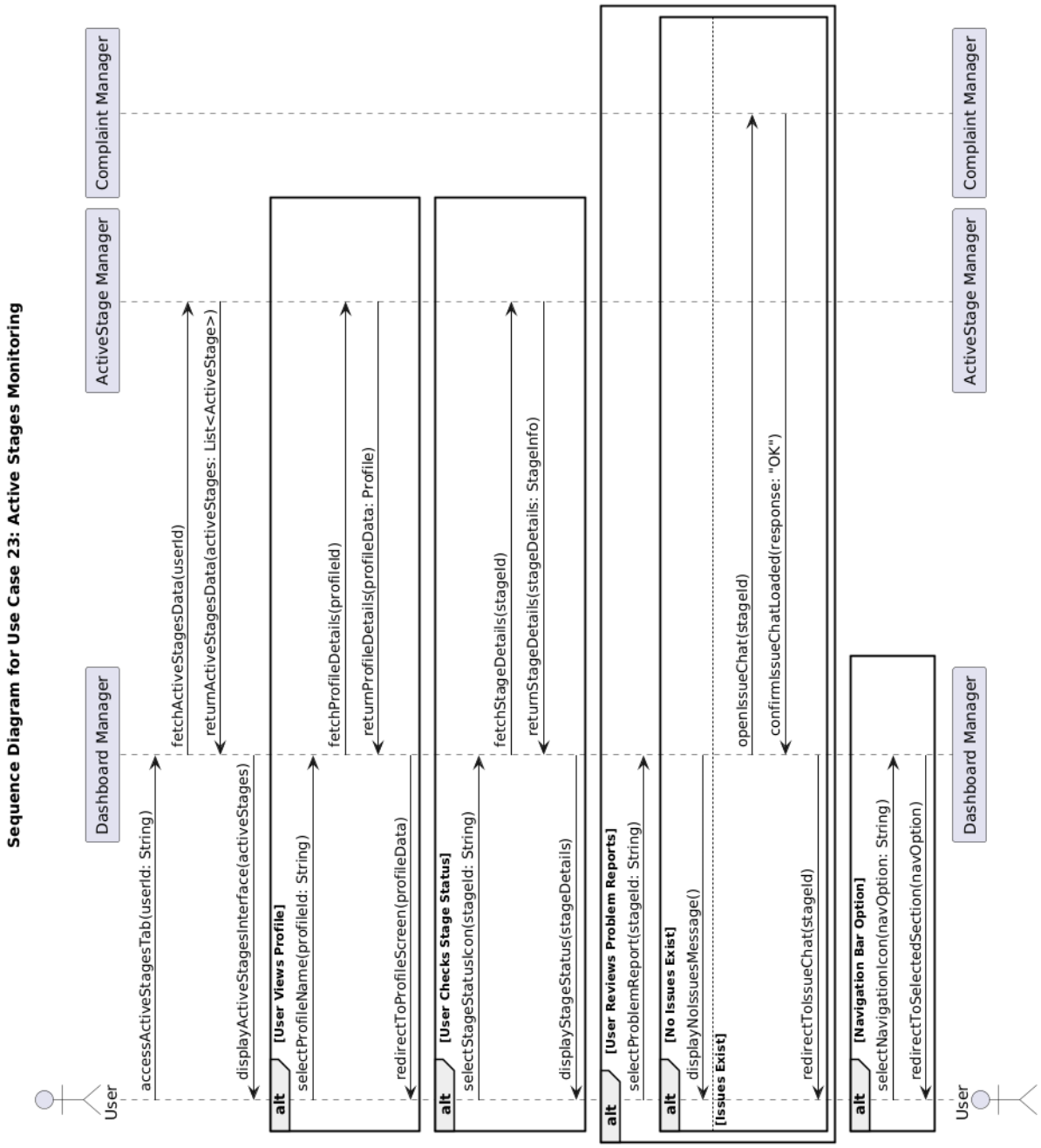

Figure 2.34: Sequence Diagram for Use Case 23: Active Stages Monitoring.

[UC24]: First Meeting Questionnaires Management

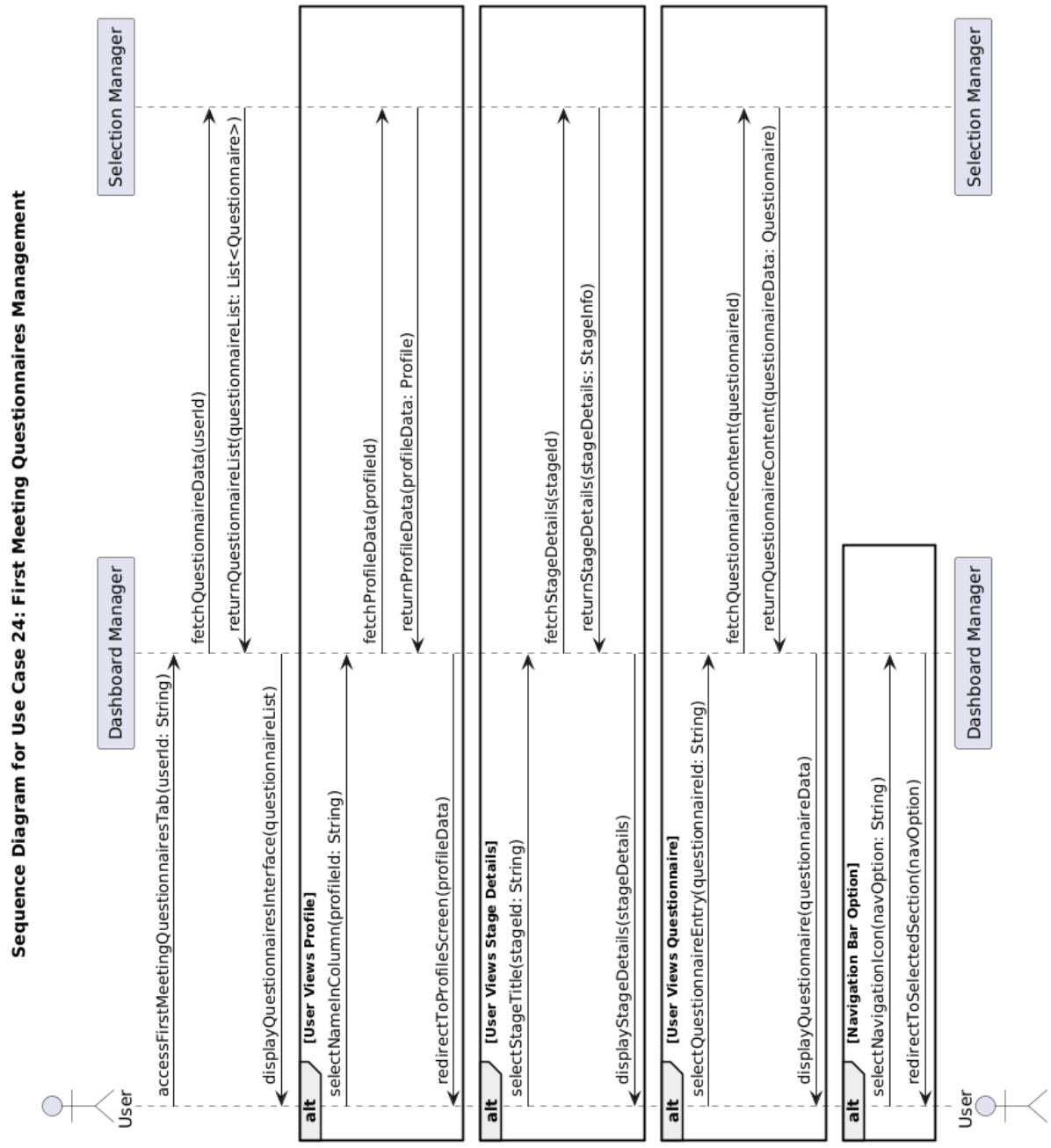

Figure 2.35: Sequence Diagram for Use Case 24: First Meeting Questionnaires Management.

[UC25]: Final Evaluations Management

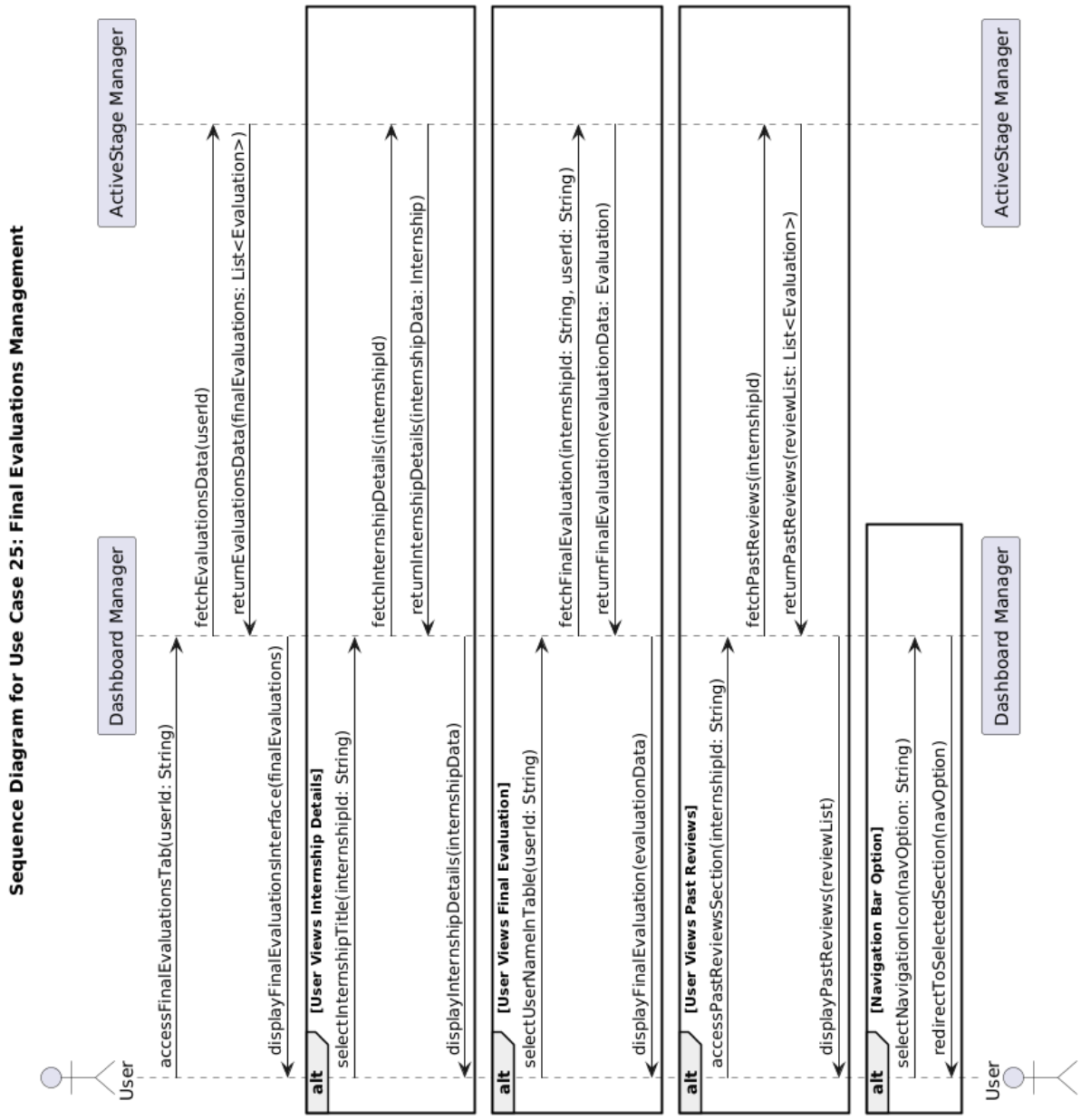

Figure 2.36: Sequence Diagram for Use Case 25: Final Evaluations Management.

[UC26]: Internship Creation and Management

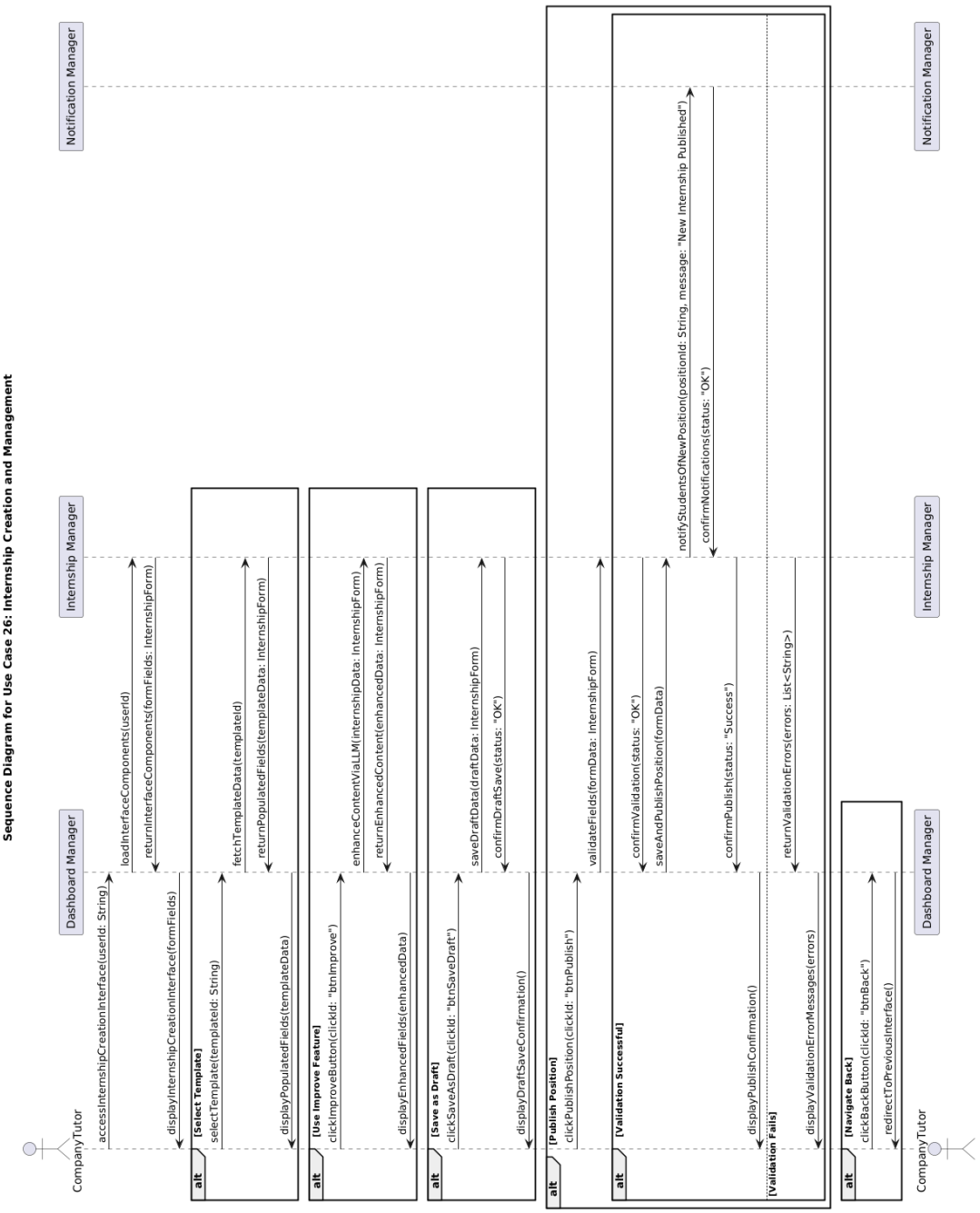

Figure 2.37: Sequence Diagram for Use Case 26: Internship Creation and Management.

[UC27]: Draft Management

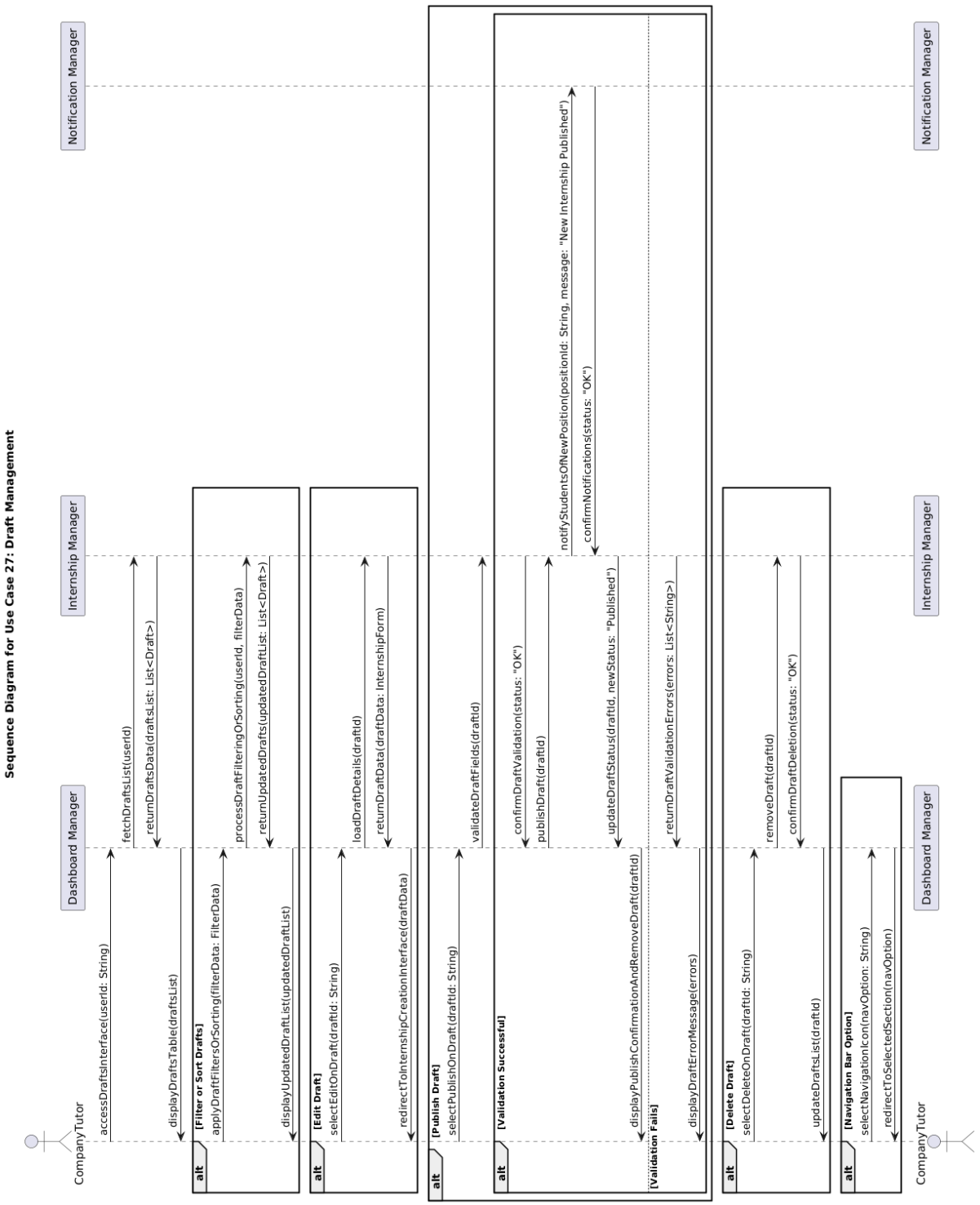

Figure 2.38: Sequence Diagram for Use Case 27: Draft Management.

[UC28]: First Meeting Questionnaire Completion

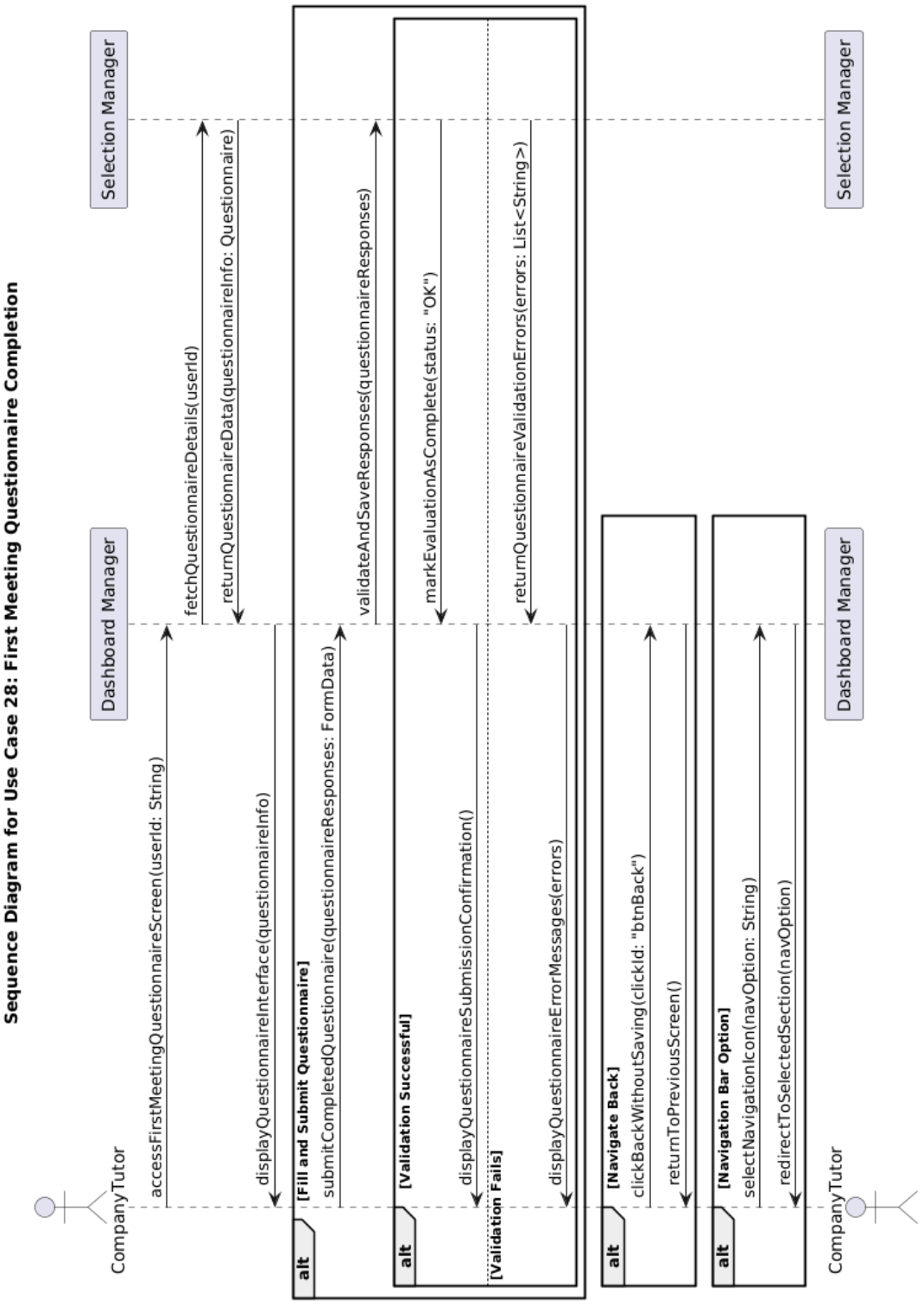

Figure 2.39: Sequence Diagram for Use Case 28: First Meeting Questionnaire Completion.

[UC29]: Final Evaluation Completion

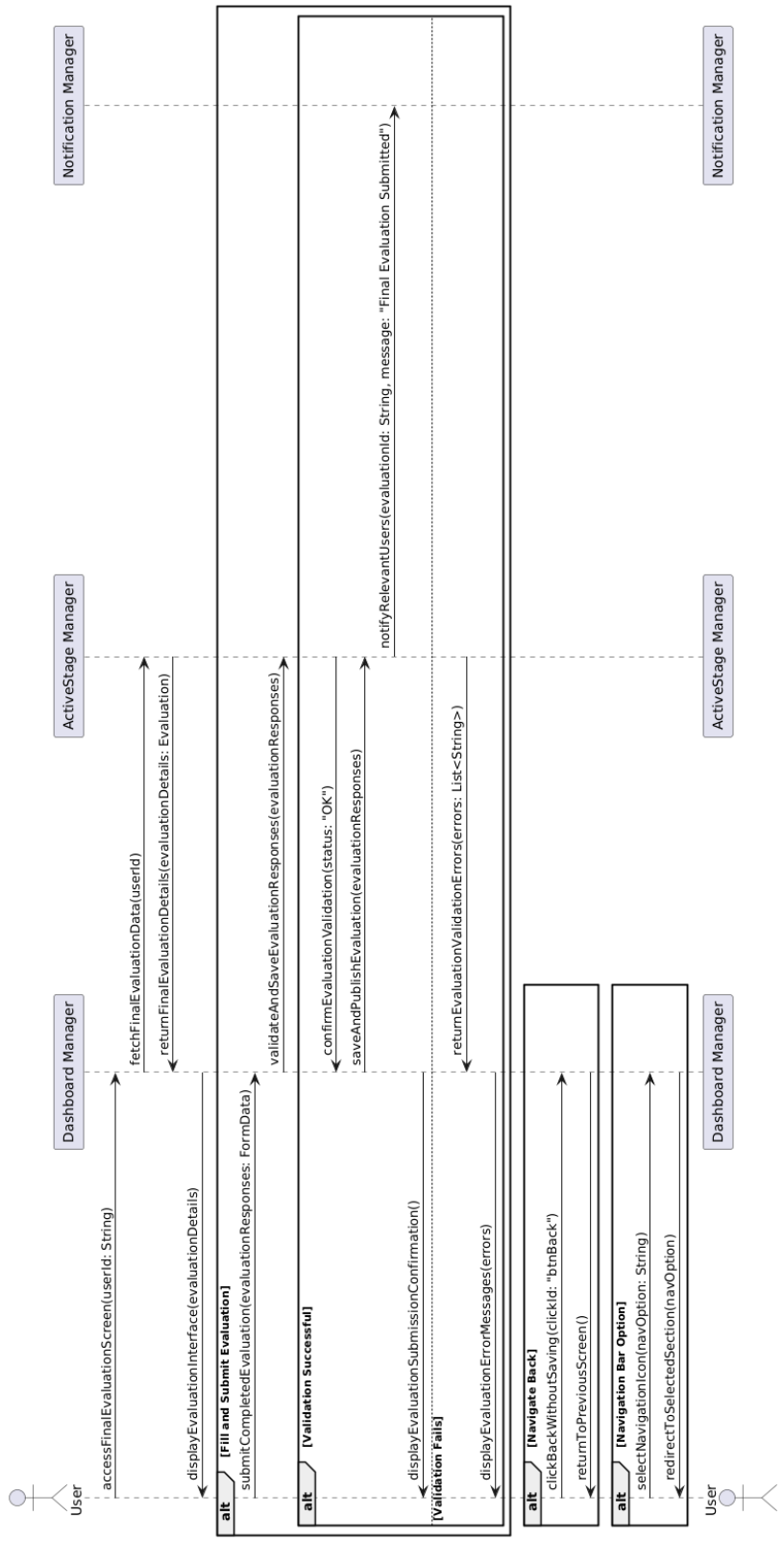

Figure 2.40: Sequence Diagram for Use Case 29: Final Evaluation Completion.

[UC30]: Viewing First Meeting Questionnaire

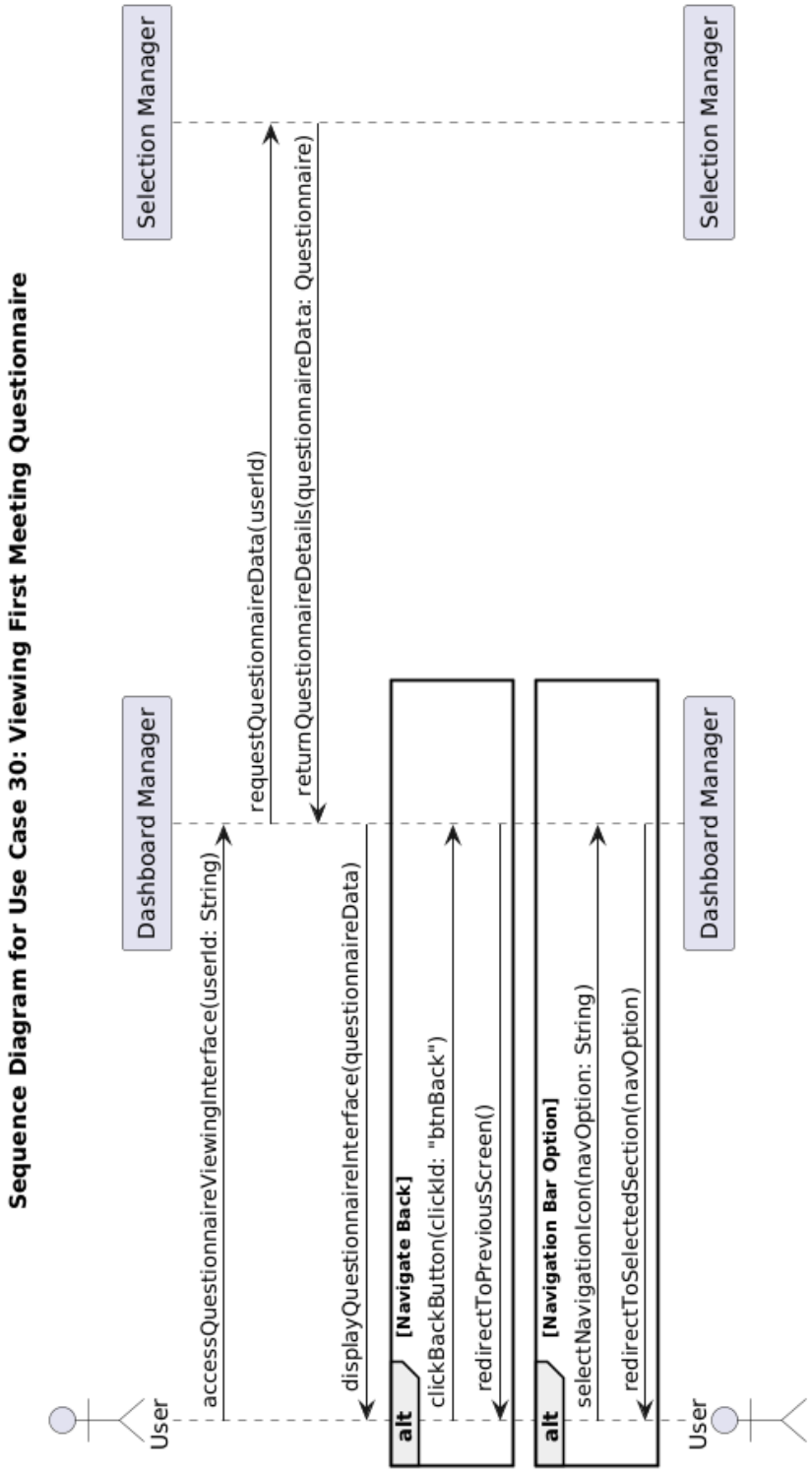

Figure 2.41: Sequence Diagram for Use Case 30: Viewing First Meeting Questionnaire.

[UC31]: Viewing Final Evaluation

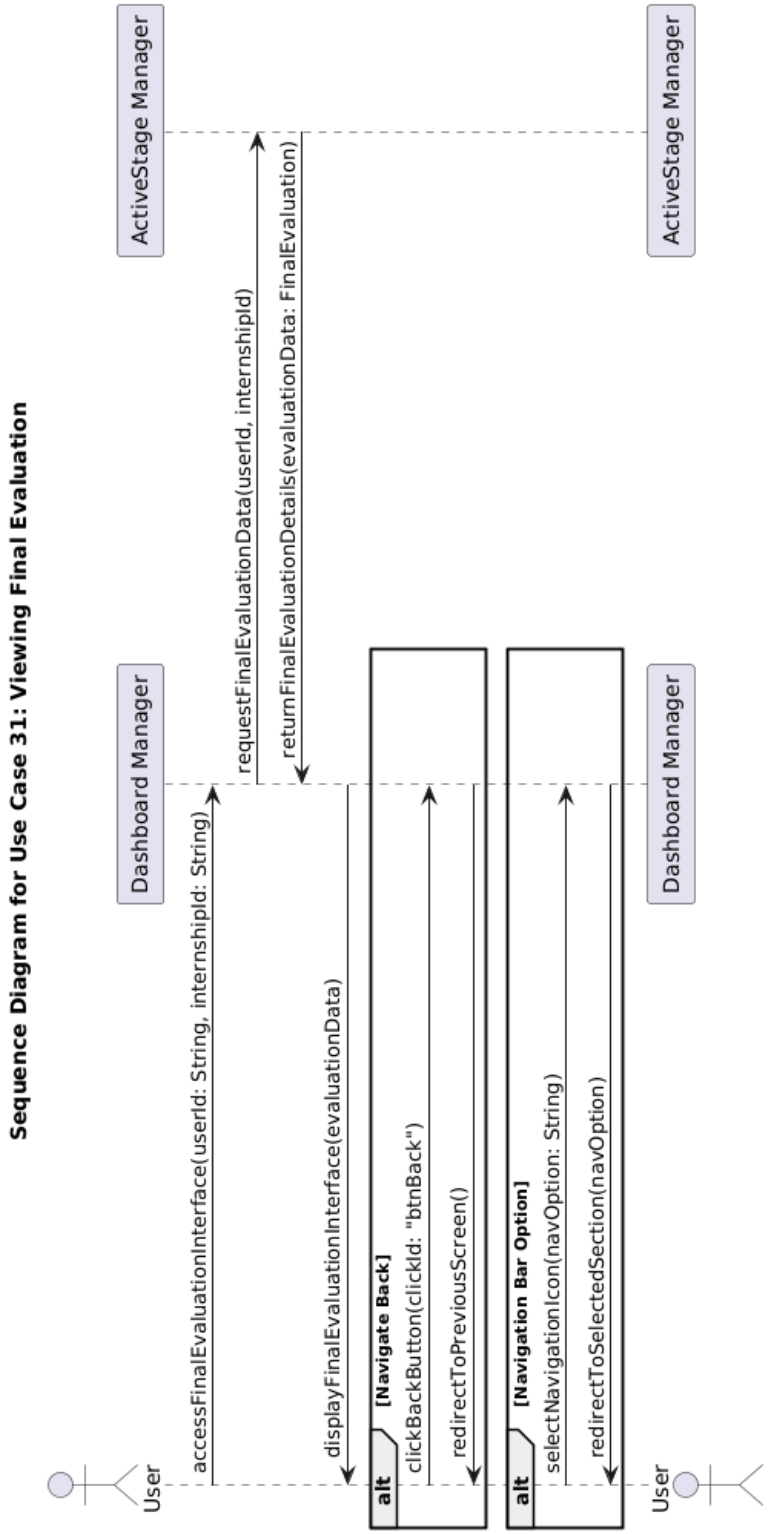

Figure 2.42: Sequence Diagram for Use Case 31: Viewing Final Evaluation.

[UC32]: Stage Status History

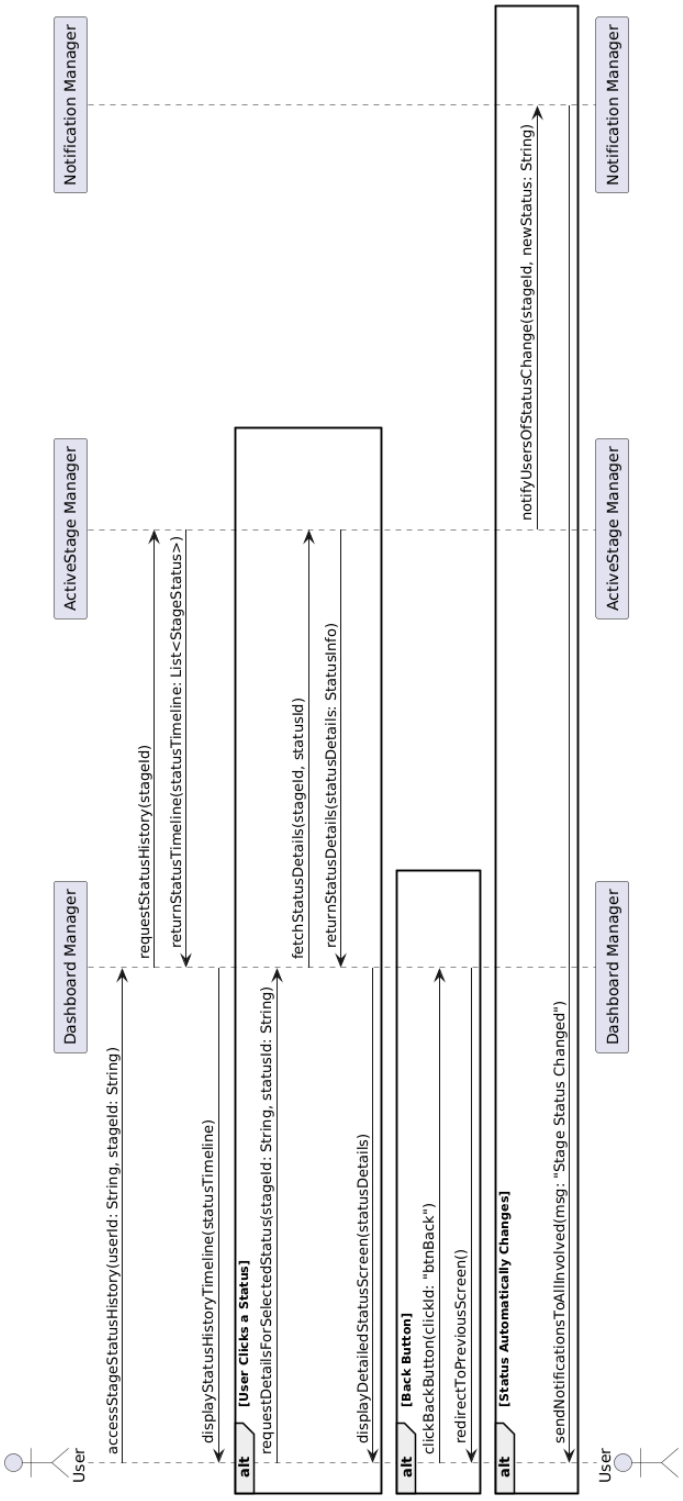

Figure 2.43: Sequence Diagram for Use Case 32: Stage Status History.

[UC33]: User Profile View

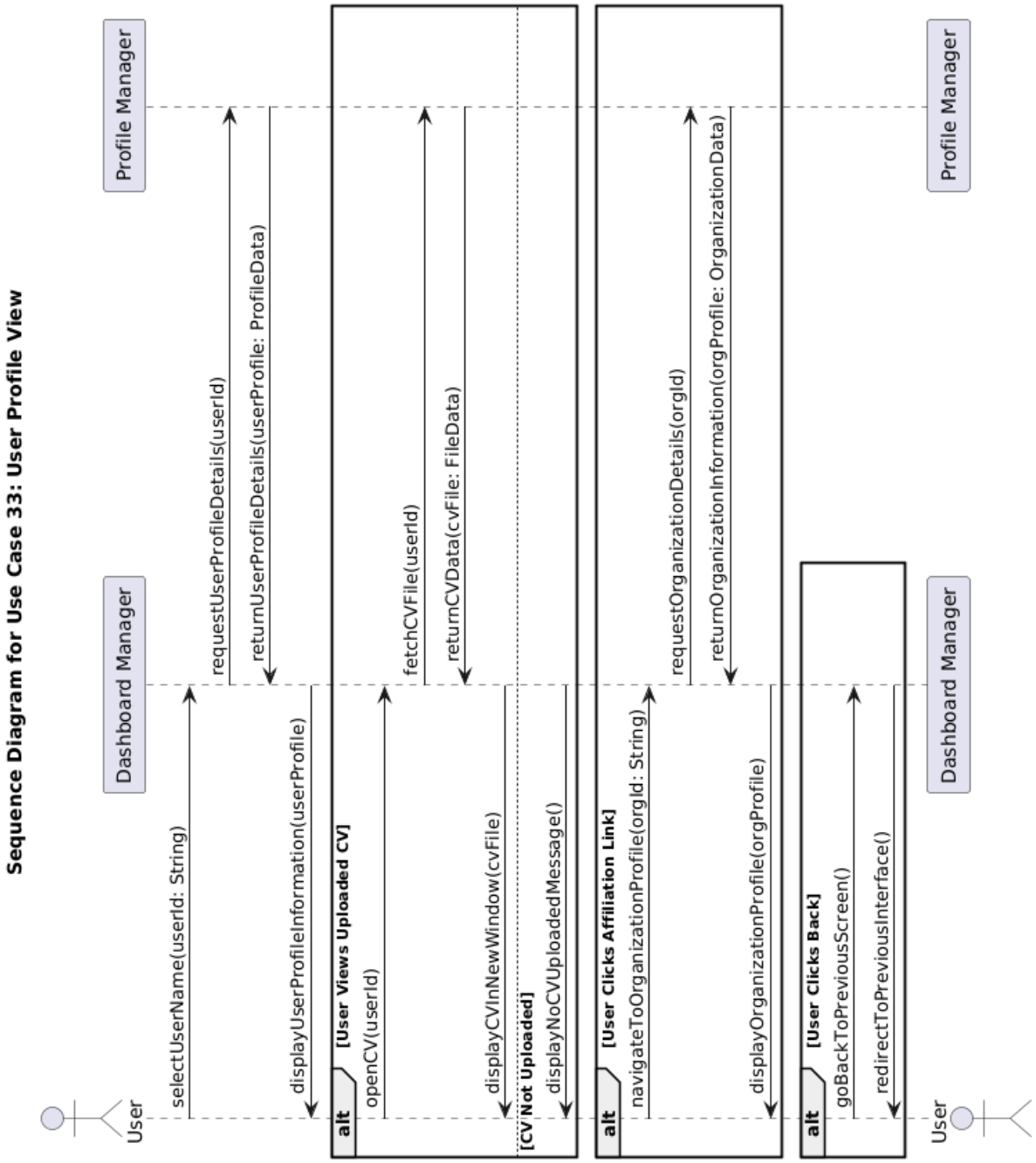

Figure 2.44: Sequence Diagram for Use Case 33: User Profile View.

[UC34]: Institution Profile View

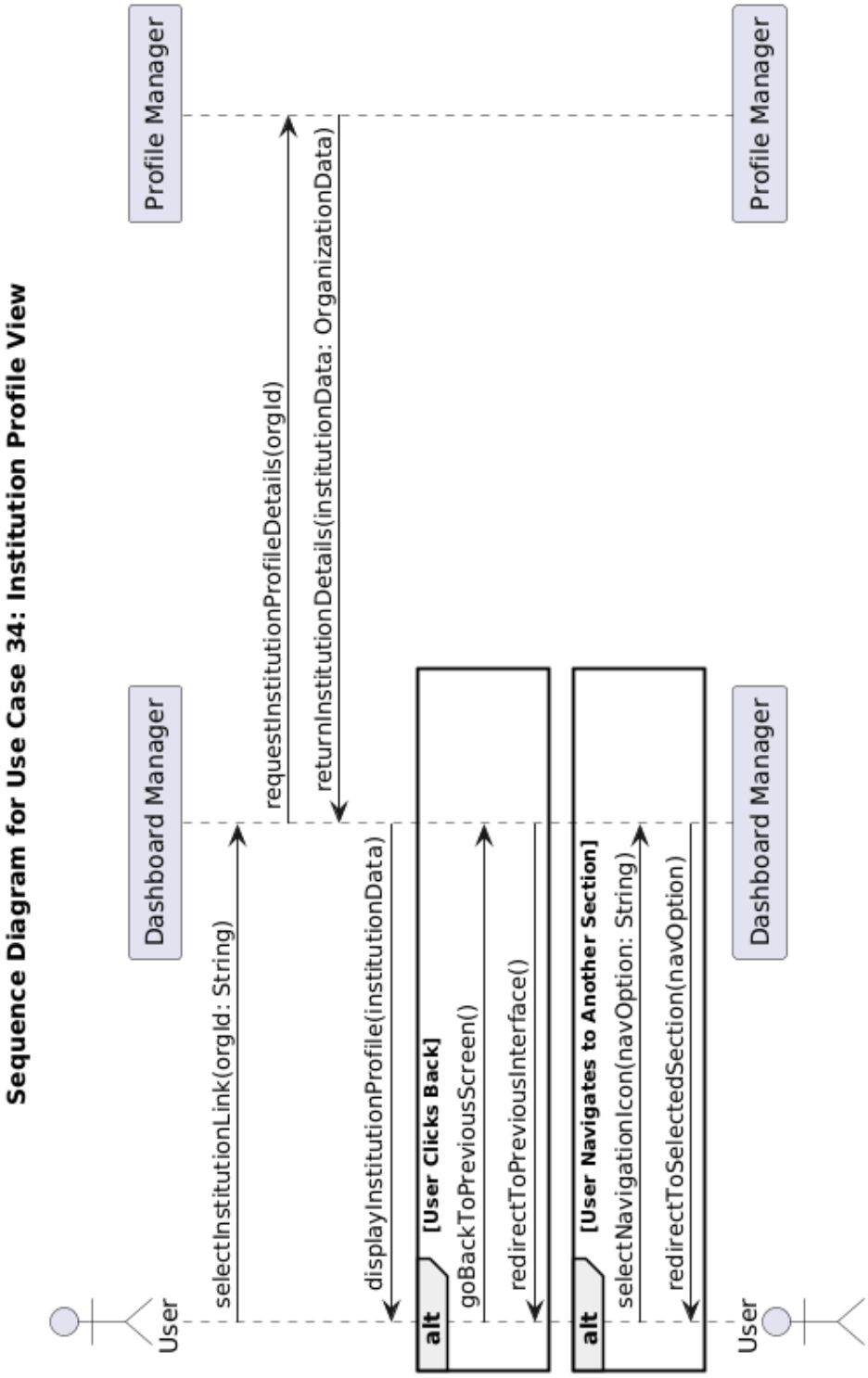

Figure 2.45: Sequence Diagram for Use Case 34: Institution Profile View.

[UC35]: Internship Profile View

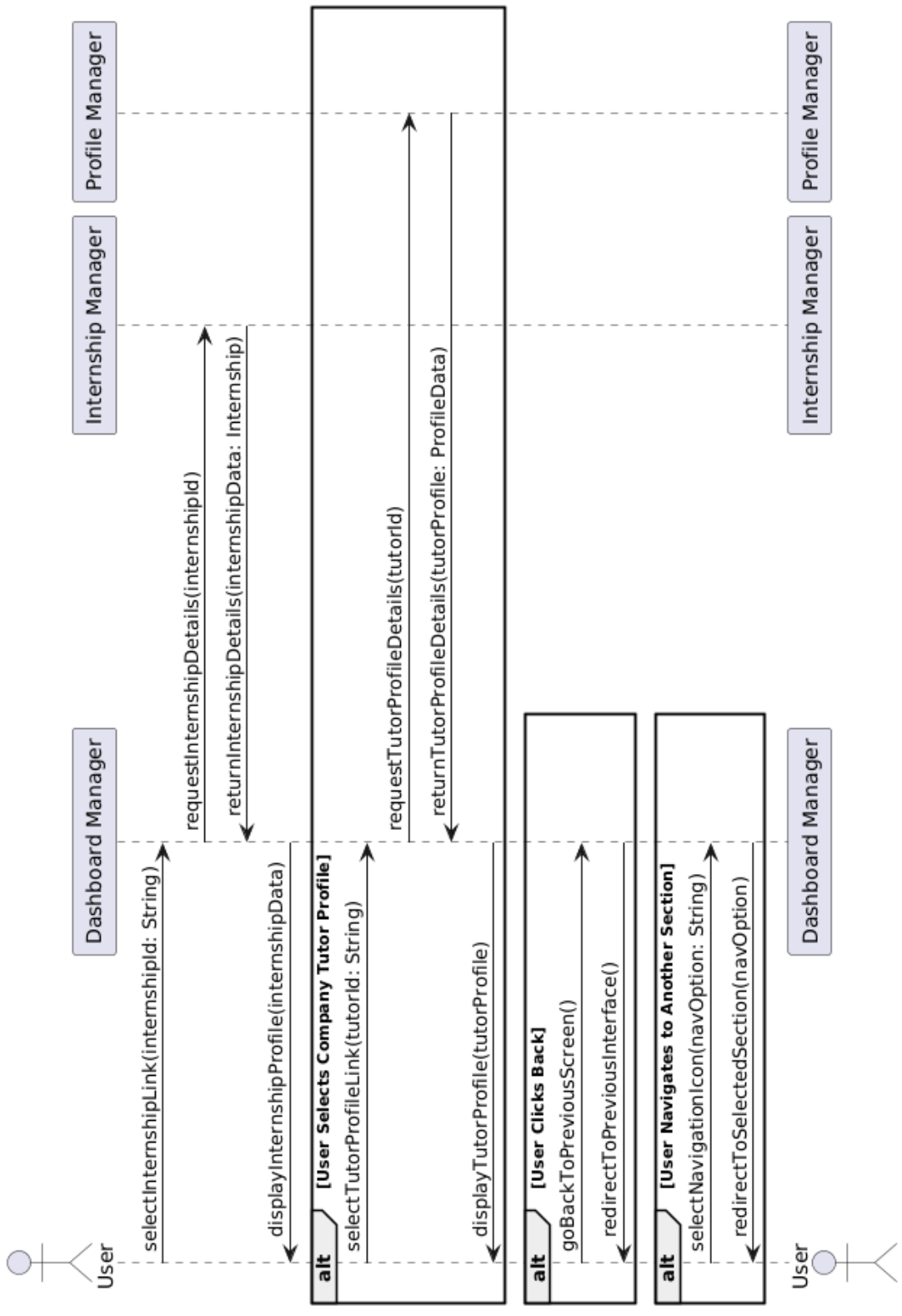

Figure 2.46: Sequence Diagram for Use Case 35: Internship Profile View.

[UC36]: Create Chat

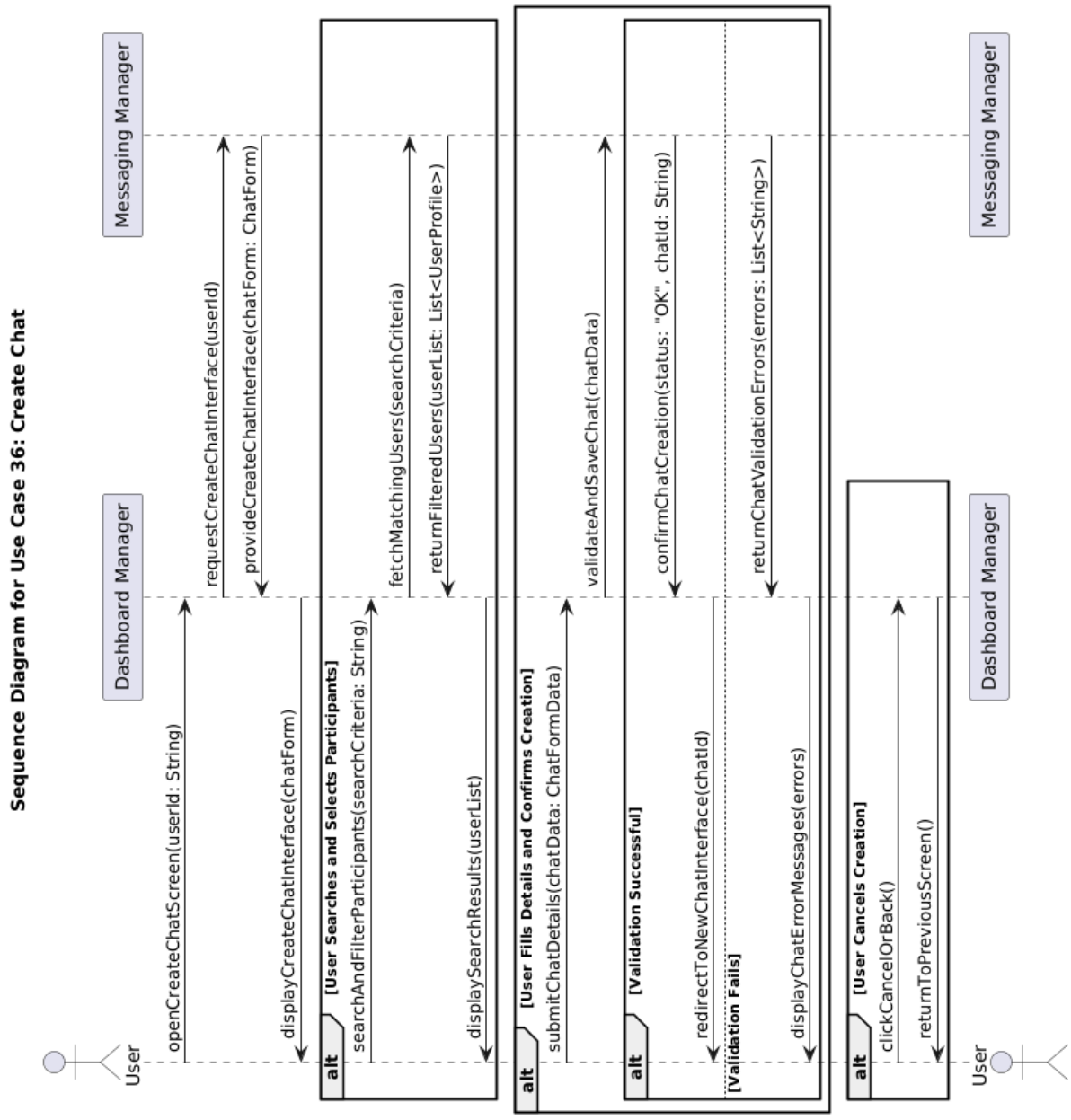

Figure 2.47: Sequence Diagram for Use Case 36: Create Chat.

2.5. Component Interface

2.5.1. Dashboard Manager

The **Dashboard Manager** acts as a mediator between the user interface (WebAPP, Browser, etc.) and the various internal Managers within the system. It handles redirections, page/resource loading and coordinates communications.

- `openPlatform(url: String)`: Opens the platform at the specified URL (e.g., "/home"), initializing the session and the main page.
- `handleAccessRequest(request: HttpRequest)`: Handles access requests (e.g., homepage, login page, registration page) by forwarding them to the appropriate components (Login Manager, Registration Manager, etc.).
- `showAccessPageWithOptions(response: HttpResponse)`: Displays the access page with various options (login, registration, assistant, language change, etc.).
- `redirectToLogin(action: String)`: Redirects the access request to the **Login Manager**.
- `redirectToRegistration(action: String)`: Redirects the access request to the **Registration Manager**.
- `redirectToAssistant(action: String)`: Redirects the request to the virtual assistant interface.
- `redirectToLanguageChange()`: Manages the language change logic by redirecting to the language settings interface.
- `forwardCredentials(email: String, password: String)`: Forwards the login credentials to the **Login Manager**.
- `forwardNavigationRequest(sectionId: String, userSession: Session)`: Forwards navigation requests (navigation bar, various links) to the appropriate component (Profile Manager, Internship Manager, etc.).
- `showErrorMessage(errorCode: int, detail: String)`: Displays a generic error message to the user.
- `displayErrorMessage(errorMsg: String)`: Shows an error message (e.g., in case of failed validation).
- `navigateToSelectedSection(navOption: String, userSession: Session)`: Manages navigation to another section of the app (e.g., "/profile", "/internship", etc.).
- `returnToPreviousScreen()` / `redirectToPreviousInterface()`: Allows returning to the previous interface (e.g., "Go Back" functionality).
- `displayXXXXPage(...)`: Generic methods (where "XXXX" is the page/resource) to load and display the view to the user, such as `displayLoginPage()`, `displayRegistrationPage()` `displayAssistantPage()`, etc.

2.5.2. Notification Manager

The **Notification Manager** is responsible for sending notifications to various users (e.g., emails, in-app notifications) whenever certain events occur (e.g., internship publication, status updates, selection invitations, etc.).

- `notifyUserOfChanges(userId: String, changes: String)`: Sends a notification to a single user regarding changes (profile, settings, etc.).
- `notifyRelevantUsers(entityId: String, message: String)`: Sends notifications to all "relevant" users associated with a specific entity (e.g., stage, selection, complaint, etc.).
- `notifyParticipants(entityId: String, message: String)`: Notifies participants of an internship, stage, meeting, or chat about an update.
- `notifyAdminsForVerification(domain: String)`: Alerts the app administrators to verify a new domain (during registration).
- `confirmNotifications(status: String)`: Returns the outcome of the notification sending process (e.g., "OK" if all notifications were sent successfully).
- `sendVerificationEmail(domain: String, adminEmails: [String])`: Sends domain verification emails to admins (part of domain checking logic).
- `notifyDomainVerificationInProgress(msg: String)`: Informs the user that the entered domain is under verification.
- `notifyUsersOfStatusChange(stageId: String, newStatus: String)`: Sends notifications to involved users when the status of a stage changes automatically.
- `sendAssistantNotification(...)`: *(Optional, if the assistant provides notifications).*

2.5.3. Calendar Manager

The **Calendar Manager** handles the creation, display and synchronization of events and meetings, optionally integrating with external services (Google, Apple, etc.).

- `fetchCalendarData(userId: String)`: Retrieves calendar data (events, meetings) related to the specified user.
- `getFullCalendarView(userId: String)`: Provides the complete calendar view (optional, e.g., for a widget).
- `requestFilteredEventsOrViewUpdate(userId: String, filterData: FilterData)`: Updates the list of events based on filters or view (daily, weekly, etc.).
- `saveNewEvent(userId: String, newEventData: Event)`: Saves a new event to the user's calendar.
- `saveEventChanges(eventId: String, updatedEventData: Event)`: Updates the data of an existing event and notifies participants.

- `synchronizeEvents(userId: String, eventList: List<Event>)`: Synchronizes events with an external service (e.g., Google Calendar).
- `confirmSynchronization(status: String)`: Confirms the outcome of the synchronization process (e.g., "OK").

2.5.4. Registration Manager

The **Registration Manager** handles registration processes, CV uploads, domain validation and profile creation (user, company, university).

- `prefillFormDataIfCVUploaded(userId: String)`: Pre-fills certain registration form fields using data extracted from the uploaded CV.
- `processSuggestionViaLLM(formData: FormData)`: Processes registration data (descriptions, fields, etc.) using an LLM to enhance/recommend improvements.
- `saveFileData(file: MultipartFile, userId: String)`: Saves the CV file (or attachment) associated with the user.
- `validateEmailDomain(email: String)`: Verifies if the email domain is registered/valid within the system.
- `linkUserToInstitution(userId: String, domain: String)`: Associates a user with a company or university based on the validated domain.
- `skipCVUpload(userId: String)`: Handles the case where the user opts not to upload a CV during registration.
- `submitProfile(formData: FormData, userId: String)`: Submits the final registration data to be saved in the database and completes account creation.
- `returnImprovementSuggestions(suggestions: String)`: Returns suggestions (e.g., fields to improve) for the registration phase.
- `notifyDomainVerificationInProgress(...)`: *(See Notification Manager).*

2.5.5. Login Manager

The **Login Manager** handles authentication and password recovery procedures.

- `validateCredentials(email: String, password: String)`: Compares credentials (email/password) with those stored in the database, returning the outcome.
- `retrieveUserData(email: String)`: Retrieves user data (id, hashedPassword, roles, etc.) from the database.
- `authenticationSuccessful(userId: String)`: Confirms a successful login (session creation, token generation, etc.).
- `authenticationFailed(reason: String)`: Returns an error for incorrect credentials, locked accounts, etc.

- `passwordRecovery(email: String, securityAnswer: String)`: *(Optional)* Handles the password recovery process using a secret answer or by sending a reset email.
- `showErrorMessage(errorMsg: String)`: *(Often managed by the Dashboard, but may have a local method).*

2.5.6. Profile Manager

The **Profile Manager** manages user profiles, including their modification, display and deletion.

- `requestUserProfileDetails(userId: String)`: Retrieves the details of a user profile (student, university/company tutor, etc.).
- `returnUserProfileDetails(userProfile: ProfileData)`: Returns the profile data (name, email, role, CV, affiliation, etc.).
- `fetchCVFile(userId: String)`: Retrieves the CV file associated with the user (if it exists).
- `displayCVInNewWindow(cvFile: FileData)`: *(Optional)* Logic to render the CV within the app (usually handled by the browser).
- `requestOrganizationDetails(orgId: String)`: Retrieves data of an organization (company/university) to display its profile.
- `returnOrganizationInformation(orgProfile: OrganizationData)`: Returns the organization's data (name, description, location, etc.).
- `modifyProfile(userId: String, updatedData: ProfileData)`: Updates profile fields (name, contact information, etc.).
- `deleteProfile(userId: String)`: Deletes the user profile.

2.5.7. Internship Manager

The **Internship Manager** is responsible for creating, modifying, publishing internships and managing their respective drafts.

- `requestInternshipDetails(internshipId: String)`: Retrieves internship data (title, description, associated tutor, status, etc.).
- `returnInternshipDetails(internshipData: Internship)`: Returns internship details to the requester (**Dashboard Manager**).
- `createOrEditInternship(userId: String, internshipData: InternshipForm)`: Handles the creation or modification of an internship (fields, requirements, deadlines, etc.).
- `saveDraftData(draftData: InternshipForm)`: Saves an internship draft in the database (incomplete).

- `publishDraft(draftId: String)`: Converts a draft into a published internship, pending validation.
- `validateDraftFields(draftId: String)`: Performs checks on mandatory fields before publication.
- `notifyStudentsOfNewPosition(positionId: String, message: String)`: *(See Notification Manager)*. Notifies students about the existence of a newly published internship.
- `improveContentViaLLM(internshipData: InternshipForm)`: Enhances the internship description using an LLM (feature "Improve").
- `removeDraft(draftId: String)`: Deletes a draft from the saved drafts list.

2.5.8. Matchmaking Manager

The **Matchmaking Manager** handles recommendation and matchmaking logic between students and companies.

- `fetchRecommendations(userId: String, role: String)`: Generates or retrieves a list of recommendations (internships for students, students for tutors, etc.).
- `generateRecommendations(userId: String, role: String)`: Generates personalized recommendations based on the user's role (student/company tutor).
- `inviteUserForSelection(processId: String, targetUserId: String)`: Sends an invitation for a selection process to a student or tutor.
- `searchProfilesOrInternships(...)`: Handles targeted searches of profiles or internships (stages) for matchmaking purposes.
- `recordFeedback(userId: String, feedbackData: Feedback)`: Records feedback on a recommended match or suggestion.

2.5.9. Selection Manager

The **Selection Manager** manages selection processes: invitations, meetings, initial questionnaires, university tutor selection, final decisions.

- `fetchSelectionProcessData(userId: String)`: Retrieves all selection processes (open, ongoing, etc.) related to a user.
- `updateProcessStatus(processId: String, newStatus: String)`: Updates the status of a selection process (Rejected, Active, Finalized, etc.).
- `fetchProcessDetails(processId: String)`: Provides details of a single selection process (meetings, invitations, tutors, etc.).
- `finalizeSelectionProcess(processId: String)`: Concludes a selection process positively and generates a transition to the "Active" stage.

- `fetchQuestionnaireData(userId: String)`: Retrieves questionnaires related to the first meeting (or selection phases) associated with a user.
- `validateAndSaveResponses(questionnaireResponses: FormData)`: Validates and saves responses to a selection questionnaire.
- `markEvaluationAsComplete(status: String)`: Marks the evaluation as completed with an "OK" outcome (in the context of the meeting questionnaire).
- `moveProcessToActiveStages(processId: String, newStatus: String)`: Moves the process from selection to the active stage (**ActiveStage Manager**).

2.5.10. ActiveStage Manager

The **ActiveStage Manager** handles everything that occurs during the active stage: communications, final evaluations, reports, events, etc.

- `fetchActiveStagesData(userId: String)`: Retrieves the list of active stages for a student or tutor.
- `fetchStageDetails(stageId: String)`: Retrieves details of a stage (roles, dates, status, etc.).
- `fetchFinalEvaluationData(userId: String)`: Retrieves final evaluation data associated with a user and a stage.
- `validateAndSaveEvaluationResponses(evaluationResponses: FormData)`: Validates and saves the final evaluation (completed by student/tutor/company, etc.).
- `saveAndPublishEvaluation(evaluationResponses: FormData)`: Makes the final evaluation effective and notifies the directly involved parties.
- `fetchPastReviews(internshipId: String)`: Retrieves any past reviews linked to the stage/internship.
- `returnActiveStagesData(...)` / `returnStageDetails(...)`: Methods to return data to the upper layer (**Dashboard Manager**).
- `notifyRelevantUsers(...)`: *(Often delegated to Notification Manager, but may have a wrapper method).*
- `updateStageStatus(stageId: String, newStatus: String)`: Updates the status of a stage (e.g., "Terminated", "Suspended", "Resumed") in case of issues.
- `moveProcessToActiveStages(...)`: *(Often related to Selection Manager).*

2.5.11. Complaint Manager

The **Complaint Manager** handles the creation, display and resolution of complaints (issues) raised by users during various phases (selection, stage).

- `saveIssueDetails(issueData: IssueFormData, userId: String)`: Saves the details of the issue (title, description, severity, etc.).

- `notifyInvolvedParty(complaintId: String, message: String)`: Notifies the party involved in a complaint (student, tutor, etc.).
- `openIssueChat(stageId: String)`: Opens (or loads) the dedicated chat for a specific issue associated with a stage.
- `updateProblemStatus(problemId: String, statusUpdate: String)`: Updates the status of the complaint (open, in management, resolved, etc.).
- `confirmStatusUpdate(confirmation: String)`: Confirms the successful update of the status.
- `fetchProblemSummary(problemId: String)`: Retrieves a summary of the problem to be displayed in the associated chat.
- `returnProblemDetails(problemDetails: ProblemInfo)`: Returns the details of the problem (description, attachments, chat, etc.).
- `manageComplaint(...)` / `planEvent(...)`: *(Optional methods for extended complaint management, if present in diagrams).*

2.5.12. Messaging Manager

The **Messaging Manager** coordinates chat logic, message sending, creating new conversations, managing attachments and the side panel (participants, shared files, etc.).

- `openMessageCreationInterface(userSession: Session)`: Loads the interface for creating a new chat (e.g., "Create Chat").
- `fetchConversations(userId: String)`: Retrieves the list of conversations (chats) for the user.
- `returnConversations(conversations: List<Conversation>)`: Returns the list of conversations to the **Dashboard Manager**.
- `fetchSidePanelContent(chatId: String)`: Retrieves additional information (participants, attached files) to be displayed in the side panel.
- `sendMessage(chatId: String, messageData: Message)`: Handles sending a new message and updating the chat interface.
- `updateChatInterface(newMessage: Message)`: *(Optional)* Makes the latest message available to all participants.
- `recordFeedback(...)`: *(If the chat includes feedback on recommendations, present in some matchmaking diagrams).*
- `validateAndSaveChat(chatData: ChatFormData)`: Validates the data for creating a new chat and saves it (name, participants, etc.).
- `confirmChatCreation(status: String, chatId: String)`: Returns the outcome of the chat creation and the corresponding ID.

- `searchAndFilterParticipants(searchCriteria: String)`: Allows searching/-filtering users to add to the chat.

2.5.13. General Observations

- The listed **methods** are those that appeared **recurringly** across the Sequence Diagrams. In a real implementation, some of them might:
  - Be **unified** or **split** further.
  - Not be "public" but internal to the component.
  - Have additional parameters (e.g., an `HttpRequest`, session parameters, more complex "DTO" objects, etc.).
- Each **Manager** publishes a set of "functions" that cover:
  1. **Data Retrieval** (`fetch / request / return ...`).
  2. **Validation and/or Saving** (`validate / save / update ...`).
  3. **Notifications or Redirections to External Components** (`notify / confirm ...`).
  4. **Domain-Specific Actions** (e.g., `publishDraft`, `finalizeSelectionProcess`, `moveProcessToActiveStages`, etc.).
- The presented **signatures** (function name + parameters + possible return value) constitute the **external view** of each component, i.e., the **interface** through which other components (or the **Dashboard Manager**) can interact.

2.6. Selected Architectural Styles and Patterns

2.6.1. Three-Tier Architecture

As we said in Chapter 2 we decided to use a three-tier architecture. We already said that a benefit of three-tier architecture is that each tier runs on its own infrastructure and so each tier can be developed simultaneously by a separate development team and can be updated or scaled as needed without impacting the other tiers. In a three-tier architecture each tier runs on its own infrastructure to reduce the complexity of the system and at the same time enhancing scalability and flexibility. We already discussed in greater detail how a three-tier architecture is made in Section 2.1 and Figure 2.1.

- **Presentation Layer**: is the layer where there is a user interaction. It focus on collecting user input and show the output to the user produced by the other two layers.
- **Application Layer**: receive the user request, perform the request and some computations and can retrieve, update or delete the data in the next layer.
- **Data Layer**: it retrieves, store, update or delete the data used or produced by the layer before.

2.6.2. Client-Server Architecture

The system also should follow a client-server architecture where the client is the front-end user and it uses the presentation layer, while the server is made up of the application layer and the data layer. The server component provides a function or service to one or many clients, which initiate requests for such services, thus clients initiate communication sessions with the server, which awaits incoming requests.

It is important to remember that in a sense the server is not only a passive component awaiting client requests but also executes autonomous operations, such as interacting with external services like the Mail Server or External Calendar Service, based on some triggers, thus making the server itself an active component and not just a passive one that process the request and send the response to the Client. As an example the server needs to interact with external software such as the Mail Server or the External Calendar Service.

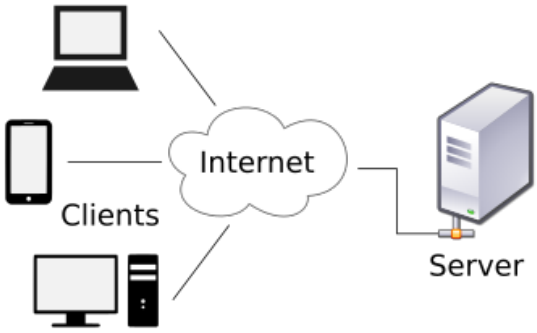

Figure 2.48: Client-Server Architecture. Source [2].

2.6.3. REST API

The REST API (Representational State Transfer Application Programming Interface) is used for the communication between the Client and the Server. REST API follows a the stateless protocol, so every transaction is treated independently and it does not rely on any information from previous transactions. The idea is that the system doesn't store any session information and every request must contain all the necessary data to be understood and processed.

REST API allows the server to be more scalable since it removes server load because the server does not have to retain past client request information. Also it allows for a greater independence between the client and the server since it is possible change the underlying technology on either side without affecting the communication.

2.6.4. Thin Client

We decided to have a thin client rather than a thick. With thick clients applications and data are stored on the machine itself and so they are far more vulnerable to cyberattacks and security breaches. On the other hand, with thin clients security updates can be rolled out automatically, without the need for user action.

By using a thin client the client does not need to know anything about the application logic: it just need to show the operation that a user can perform and the result of the request.

Thin clients have also an the advantage when it comes to scalability. Because everything is handled from a central server, any installations or updates can be deployed to all users automatically.

Lastly, this architecture enables users to access the platform even from older or less powerful devices with limited hardware resources, since the server handles the computational load.

2.6.5. Model-View-Controller Pattern

We decided to implement the S&C platform by following the Model-View-Controller architecture. The software is divided in three elements.

- **Model**: it contains methods to save, retrieve, delete or manipulate the data located in the database. It it responsible for getting and manipulating the data.
- **View**: it is the user interface, it is what the user sees when he/she interacts with the application.
- **Controller**: it is what link the model to the view. It requests data through the Model.

The Model-View-Controller architecture perfectly applies to the three-tier architecture we decided to implement. Presentation tier corresponds to view component, application tier corresponds to the controller and the data tier corresponds to the model.

MVC allows easy modification of the entire application since a single section is independent of the other sections thus accelerating the development process. Lastly MVC is also a great tool to help limit code duplication and allow easy maintenance of the application.

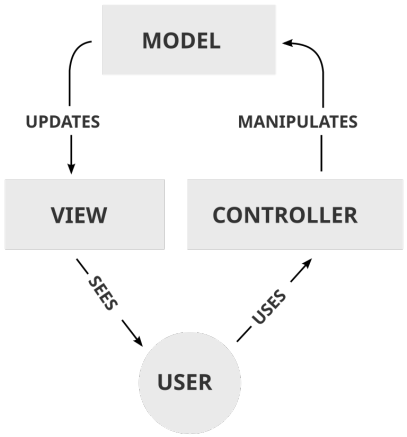

Figure 2.49: Model-View-Controller schema. Source [3].

2.7. Other Design Decisions

This section elaborates on additional design decisions that have been made to enhance the system's functionality, reliability and user experience. These choices complement the core architecture and ensure that the system operates efficiently under various conditions.

2.7.1. Availability

Ensuring high availability is crucial for maintaining uninterrupted service and providing a seamless user experience. To achieve this, the following strategies have been implemented:

- **Load Balancing**: A load balancing mechanism has been introduced to distribute incoming requests evenly across multiple server instances. This approach prevents any single server from becoming a bottleneck, thereby optimizing resource utilization and enhancing system responsiveness.
- **Replication**: Data replication is employed to create multiple copies of essential data across different servers. This redundancy ensures fault tolerance, allowing the system to maintain functionality even if one or more servers fail.

2.7.2. Scalability

The system is designed to accommodate growth and handle increasing loads without compromising performance. Key scalability considerations include:

- **Modular Architecture**: Adopting a modular architecture allows individual components or services to be scaled independently based on demand. This flexibility ensures that specific parts of the system can handle increased workloads without necessitating the scaling of the entire application.

2.7.3. Database

We decided that our data layer, should be a relational DBMS. The idea is to use MySQL which is open-source relational database management system (RDBMS). In Figure 2.50 it is possible to see how we imagine the database structure to be.

Please keep in mind that the password and the user's answers to the security questions will be encrypted. Also there should be a hierarchical structure to clearly distinguish between the different type of users (students, company tutor and university tutor). Lastly, some fields such as *languages, role* in the registration phase or *category* of the report complaint requires the user to select one element from a list. This would required an additional table for each of those selections with a foreign key, but to avoid over-complicating the representation of the database structure, we decided to use instead *VARCHAR*.

Figure 2.50: Database structure.

2.7.4. Notification Management

Effective notification management is essential for keeping users informed and engaged. The system incorporates a dedicated **Notification Manager** to handle all notification-related functionalities:

- **Real-Time Notifications**: Notifications are dispatched in real-time to inform users about critical events, such as profile updates, system changes, or important alerts.
- **Email Integration**: Leveraging the **Mail Server**, the system sends confirmation emails, password recovery links and other important communications.

2.7.5. Security

Protecting user data and ensuring system integrity are paramount. The following security measures have been implemented:

- **Authentication and Authorization**: Robust authentication mechanisms, such as JWT (JSON Web Tokens), are used to verify user identities and manage access rights.
- **Data Encryption**: Sensitive data is encrypted both in transit and at rest. Utilizing TLS/SSL protocols for data transmission and encryption algorithms for stored data safeguards against unauthorized access and data breaches.
- **Firewall Protection**: A firewall is deployed to filter incoming and outgoing traffic, mitigating potential cyber-attacks and preventing malicious activities.

3 | User Interface Design

The purpose of this chapter is to present the user interfaces designed for the system. These interfaces are crucial to provide an intuitive, efficient and user-friendly experience for all stakeholders involved.

The design of these interfaces has already been extensively detailed in the RASD, so this chapter will not delve into repetitive discussions but will instead provide an organized overview of the interface categories. The interfaces are presented in chronological order of user interaction and grouped by their respective categories.

Indeed the chapter is structured into distinct sections, each corresponding to a specific category of functionality that the interfaces support:

- **Authentication: Registration and Login:** This section outlines the interfaces enabling user authentication, including the processes for registration and login with password recovery.
- **Homepage with Settings, Change Language and Chatbot Assistance:** This section first introduces the homepage, accessible to all three types of users. From here, users can access customizable settings, language preferences and chatbot support for guidance through dedicated screens.
- **Matchmaking:** This category focuses on the interfaces dedicated to connecting students and companies through internships, based on system recommendations or predefined criteria, ensuring personalized and efficient matchmaking.
- **Monitoring: Selection Process, Active Stages and Questionnaires:** The monitoring interfaces include three tabs: Selection Process, guiding users to initiate an internship; Active Stages, tracking ongoing internships; and Questionnaires, gathering feedback from these two previous steps. In this section are also presented all the interfaces accessible from these tabs.
- **Calendar:** The calendar interfaces offer multiple views (daily, weekly and monthly) and include tools for creating and viewing calendar events. These features are common and accessible to all users, enabling seamless management of events.
- **Messaging with Issues and Video-calls:** This part encompasses communication interfaces, supporting messaging, video calls and issues reporting.

By categorizing the interfaces in this way, we aim to clarify their roles within the system and how they contribute to achieving the overarching objectives outlined in the RASD.

3.1. Authentication: Registration and Login

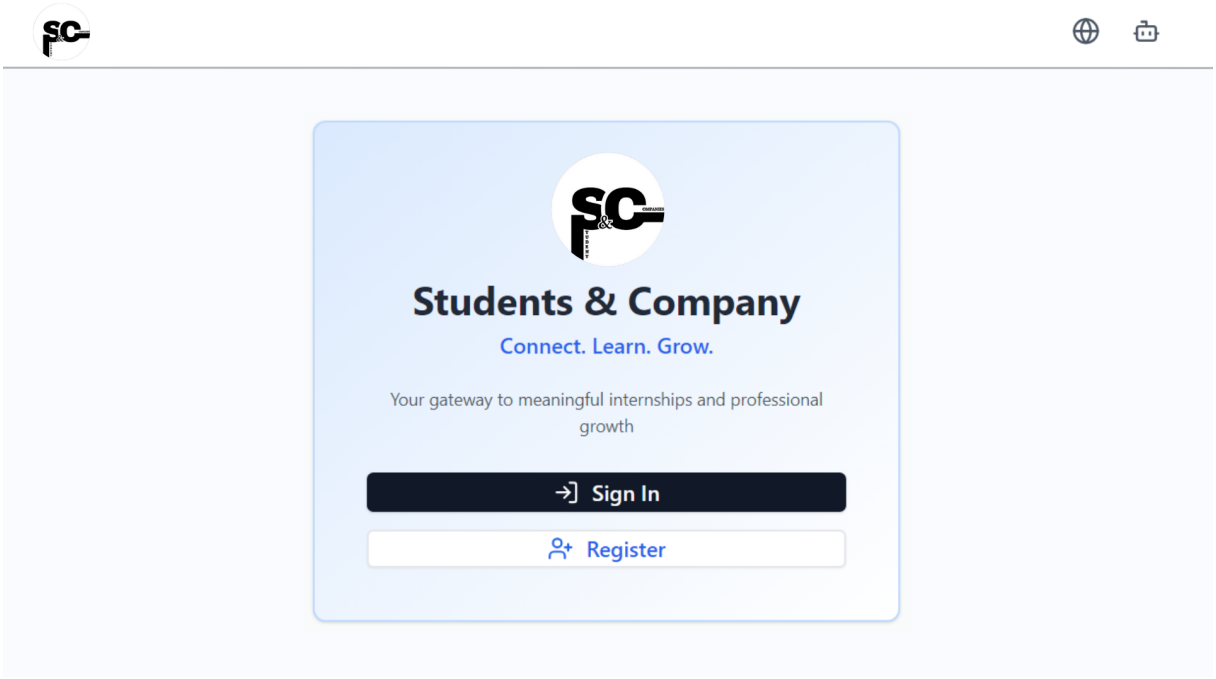

Figure 3.1: Authentication Interface of the Students & Companies platform.

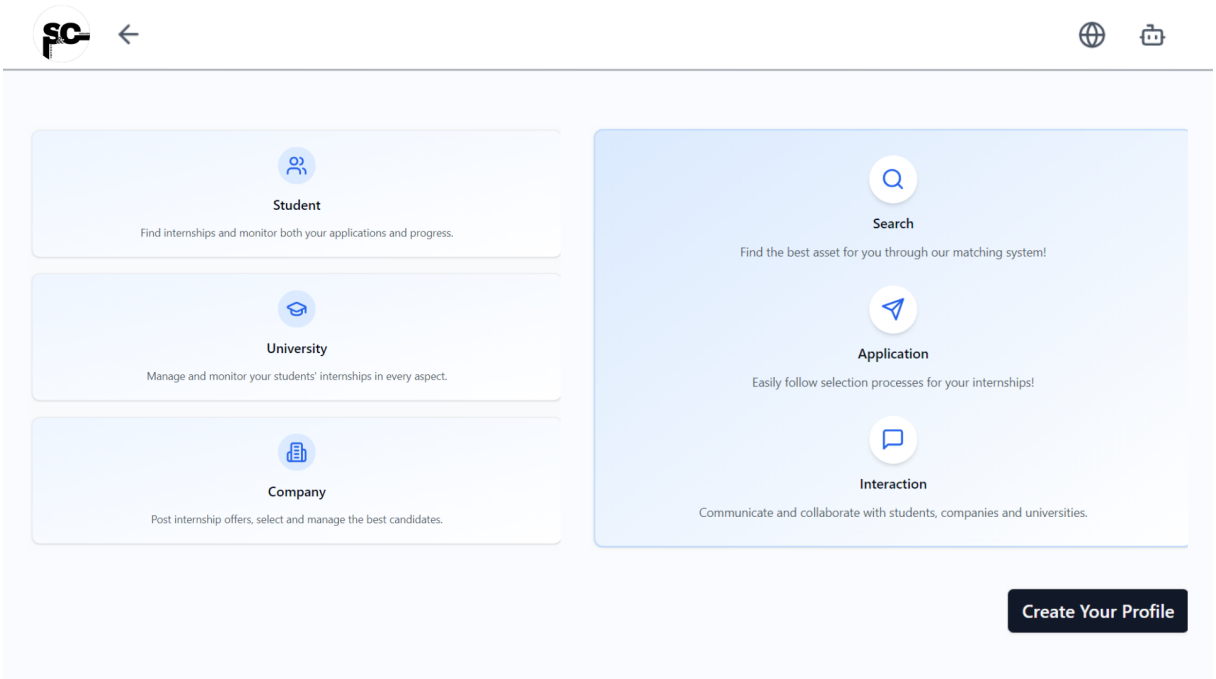

Figure 3.2: Registration - Introduction Interface.

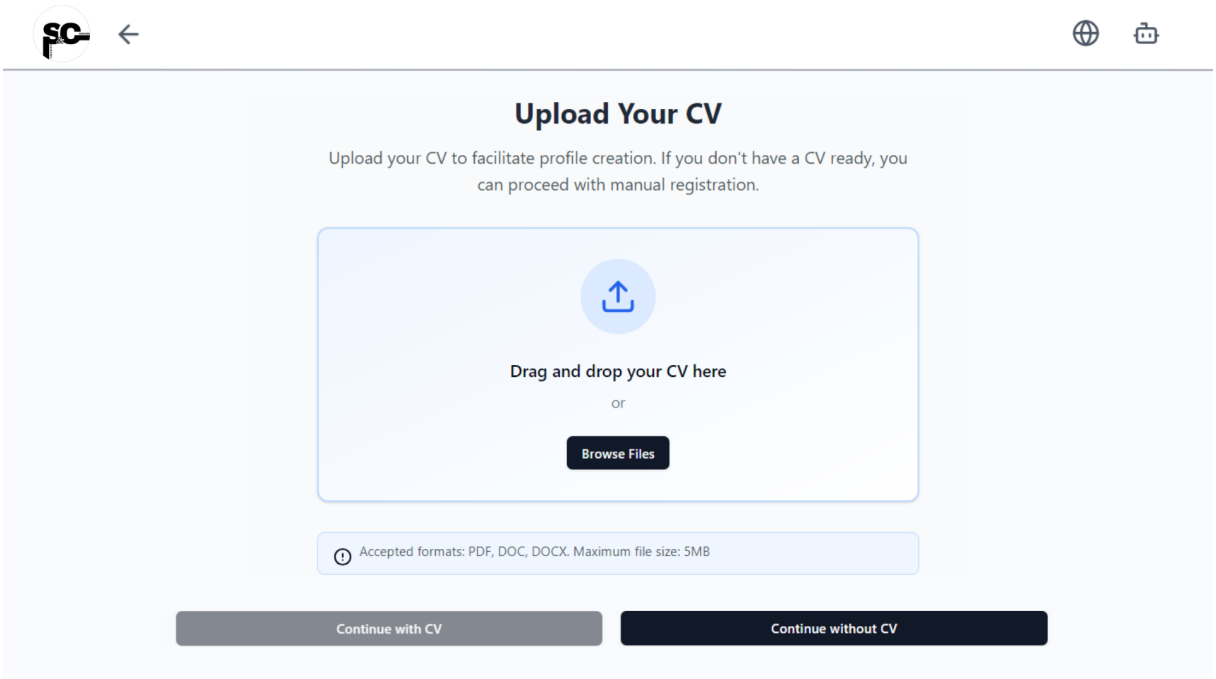

Figure 3.3: Registration - Upload CV Interface.

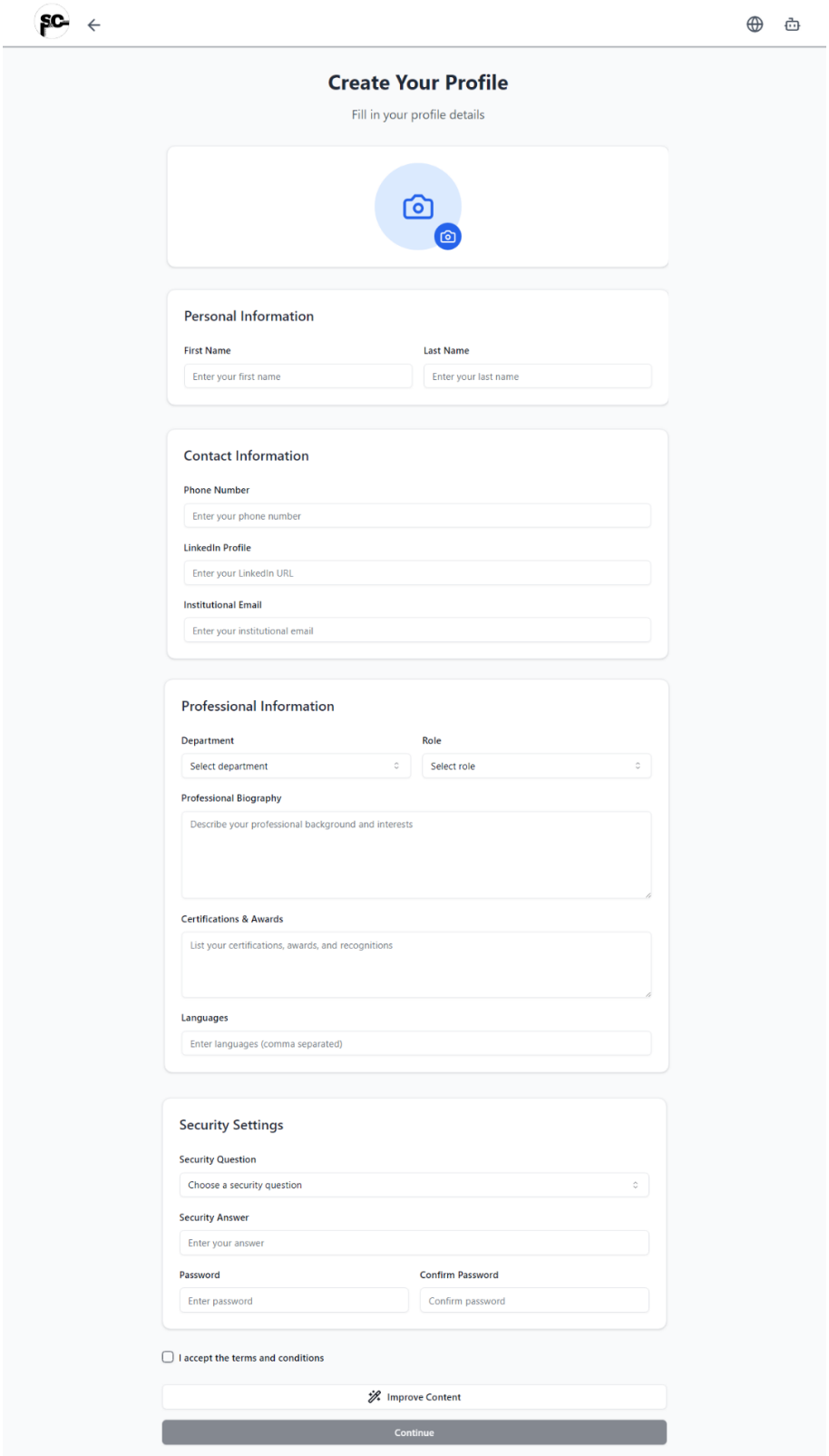

Figure 3.4: Registration - Personal Profile Creation Interface.

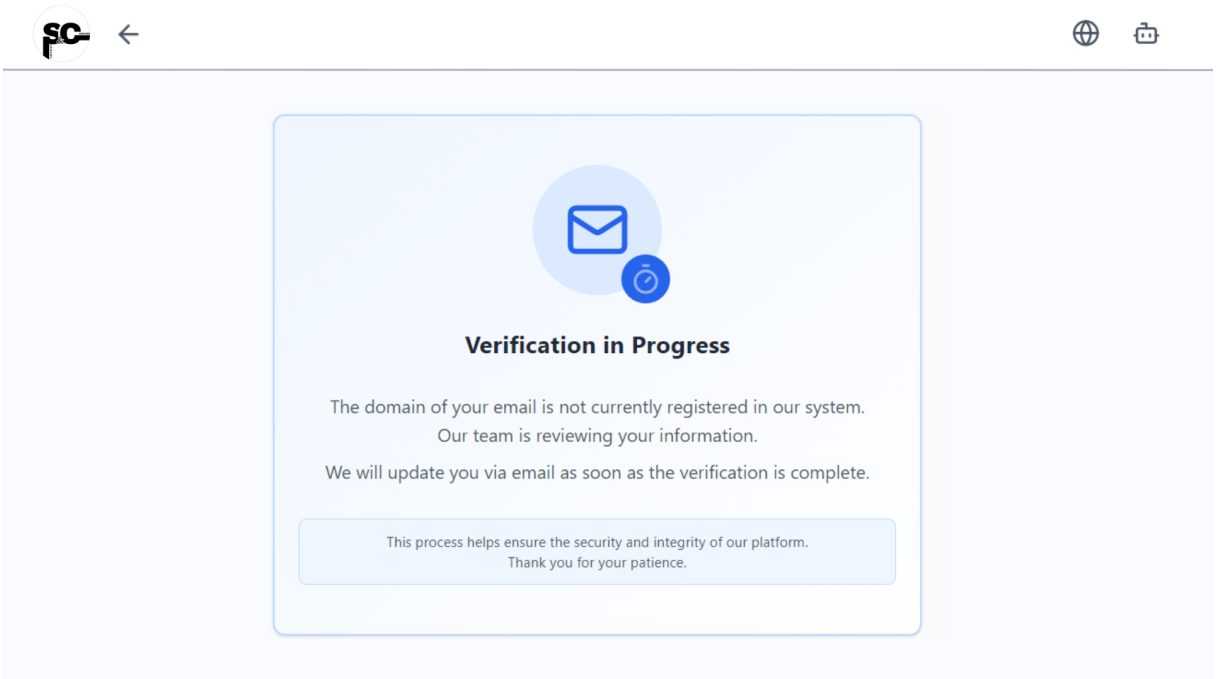

Figure 3.5: Registration - Verification in Progress Interface.

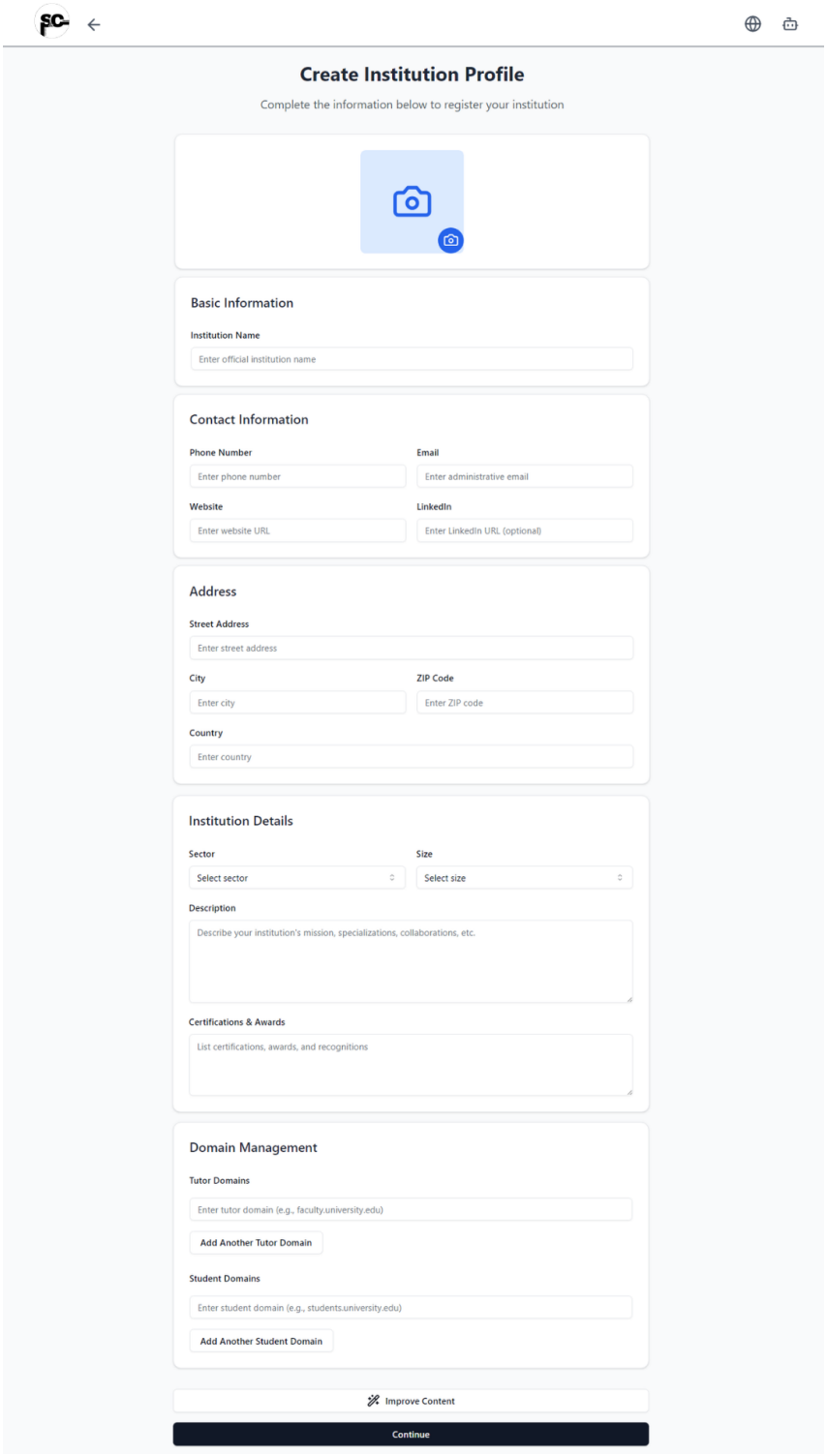

Figure 3.6: Registration - Institution Profile Creation Interface.

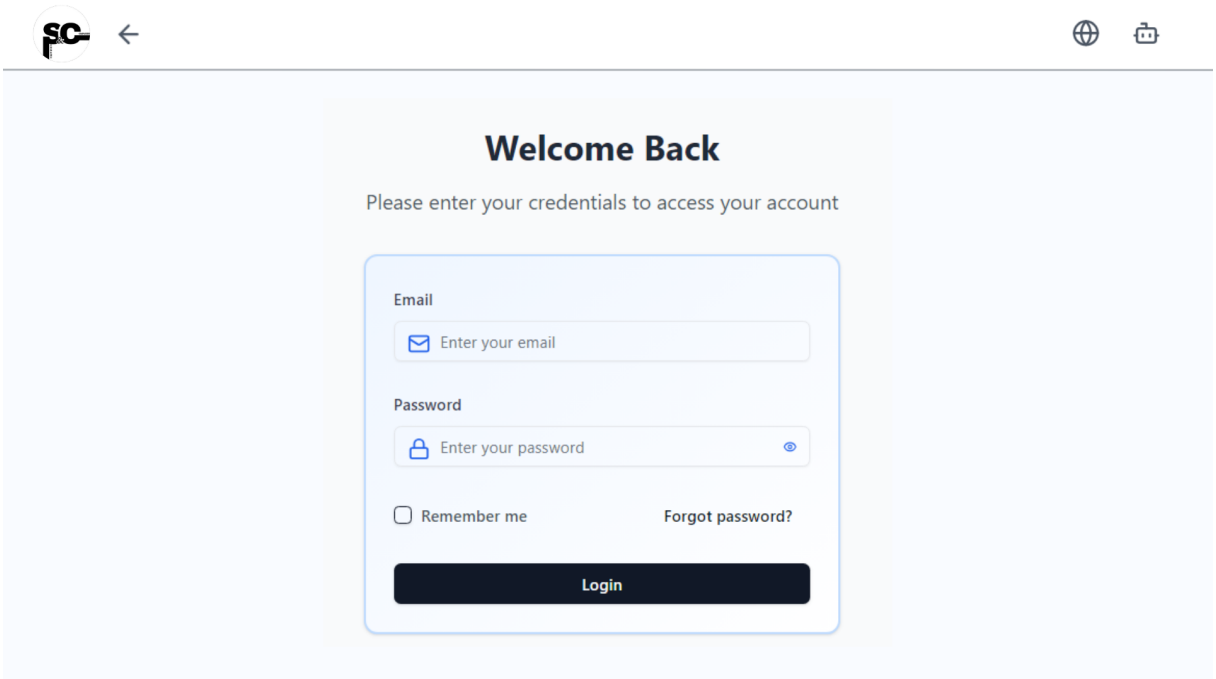

Figure 3.7: Login Interface of the Students & Companies platform.

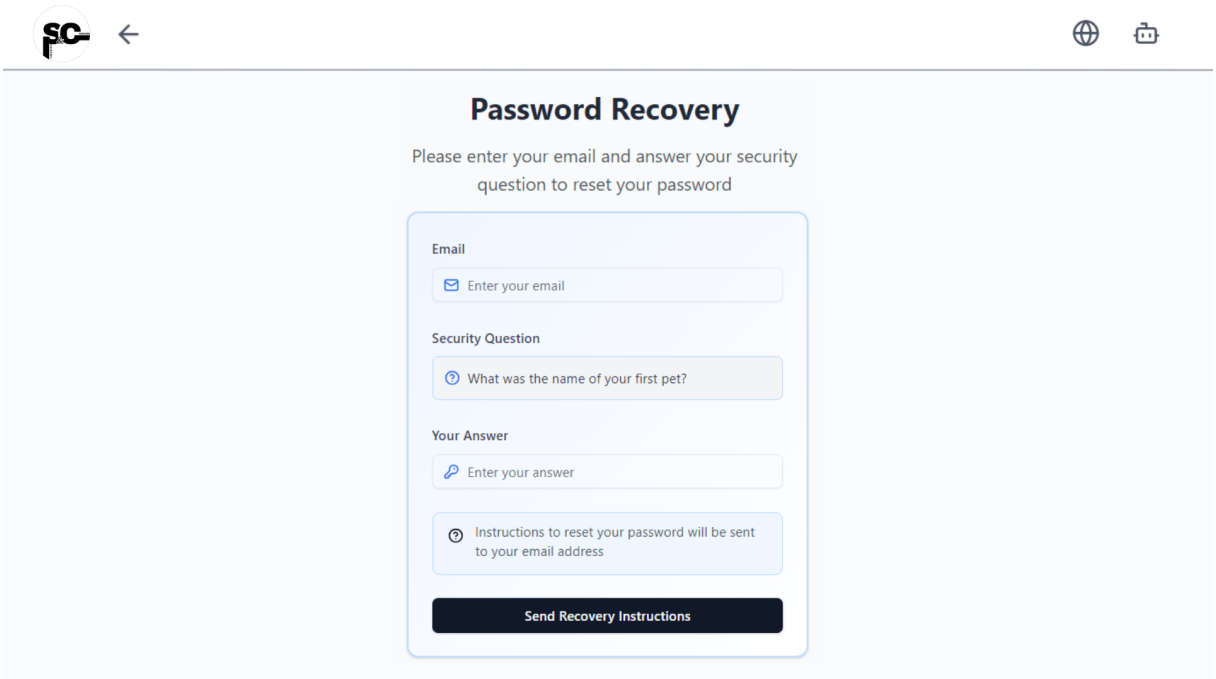

Figure 3.8: Login - Password Recovery Interface.

3.2. Homepage with Settings, Change Language and Chatbot Assistance

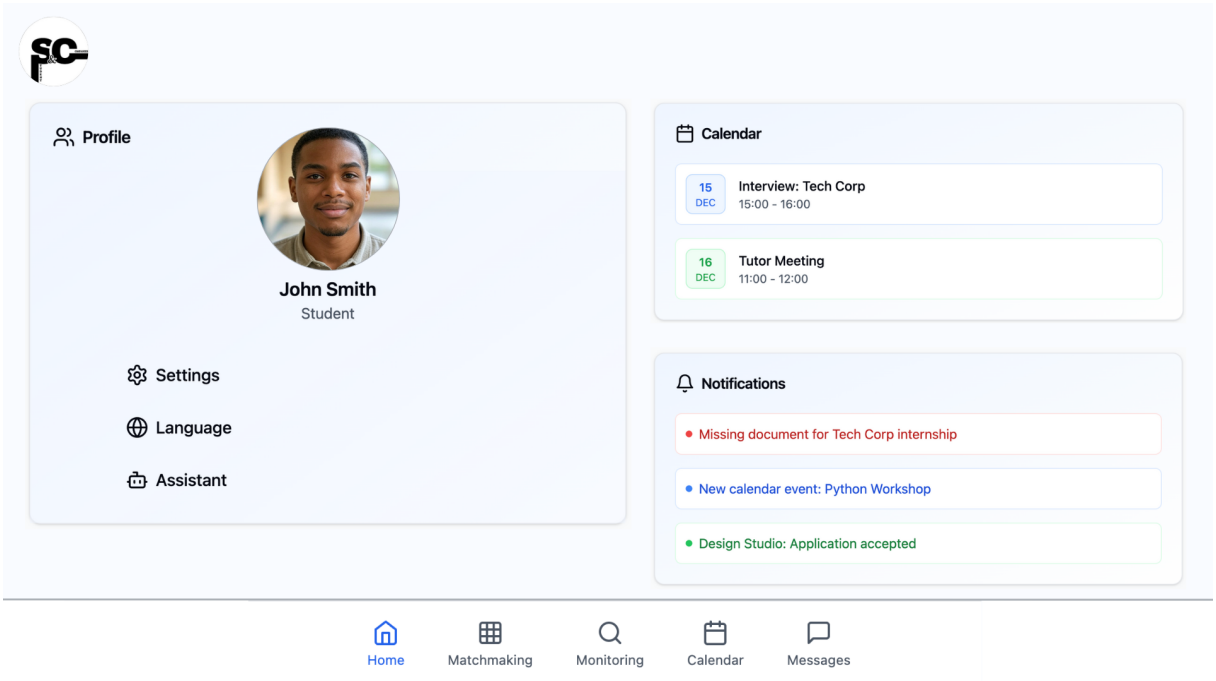

Figure 3.9: Homepage Interface for Students.

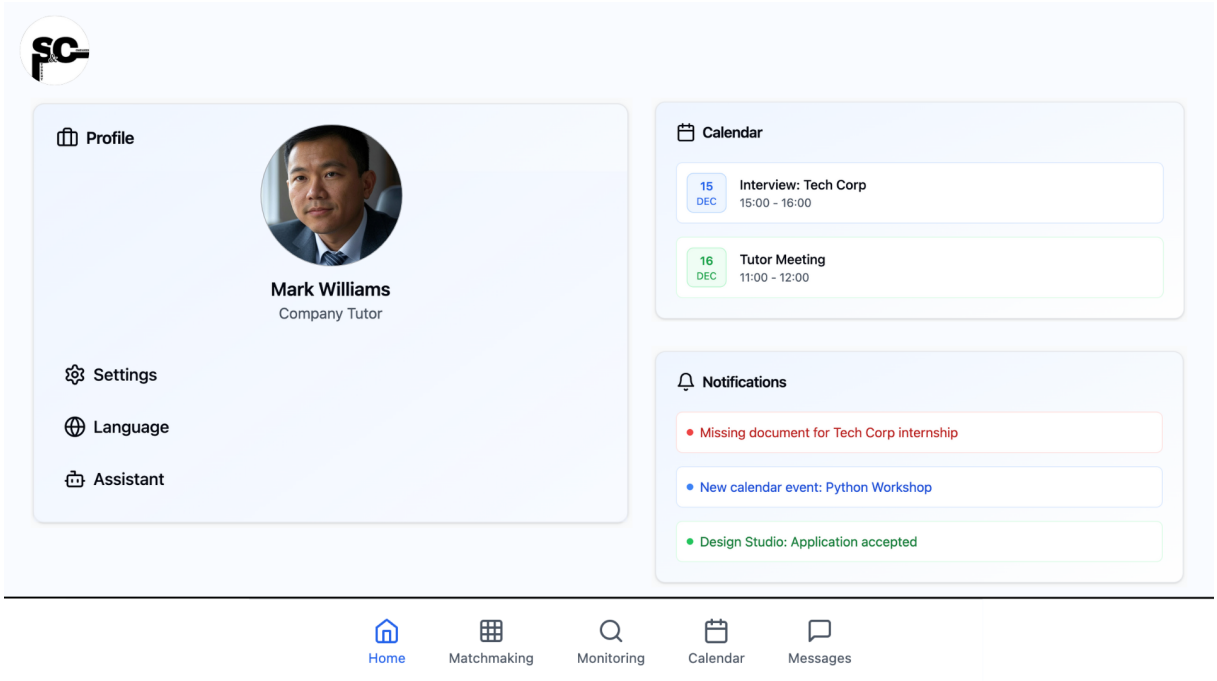

Figure 3.10: Homepage Interface for Company Tutors.

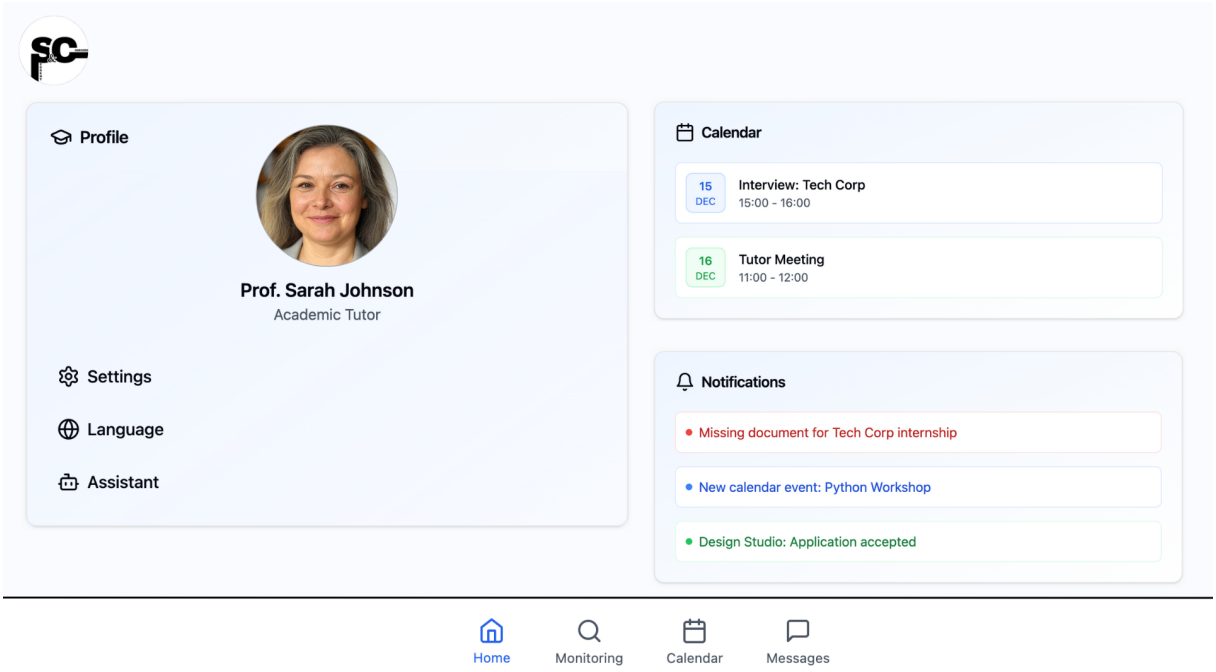

Figure 3.11: Homepage Interface for Academic Tutors.

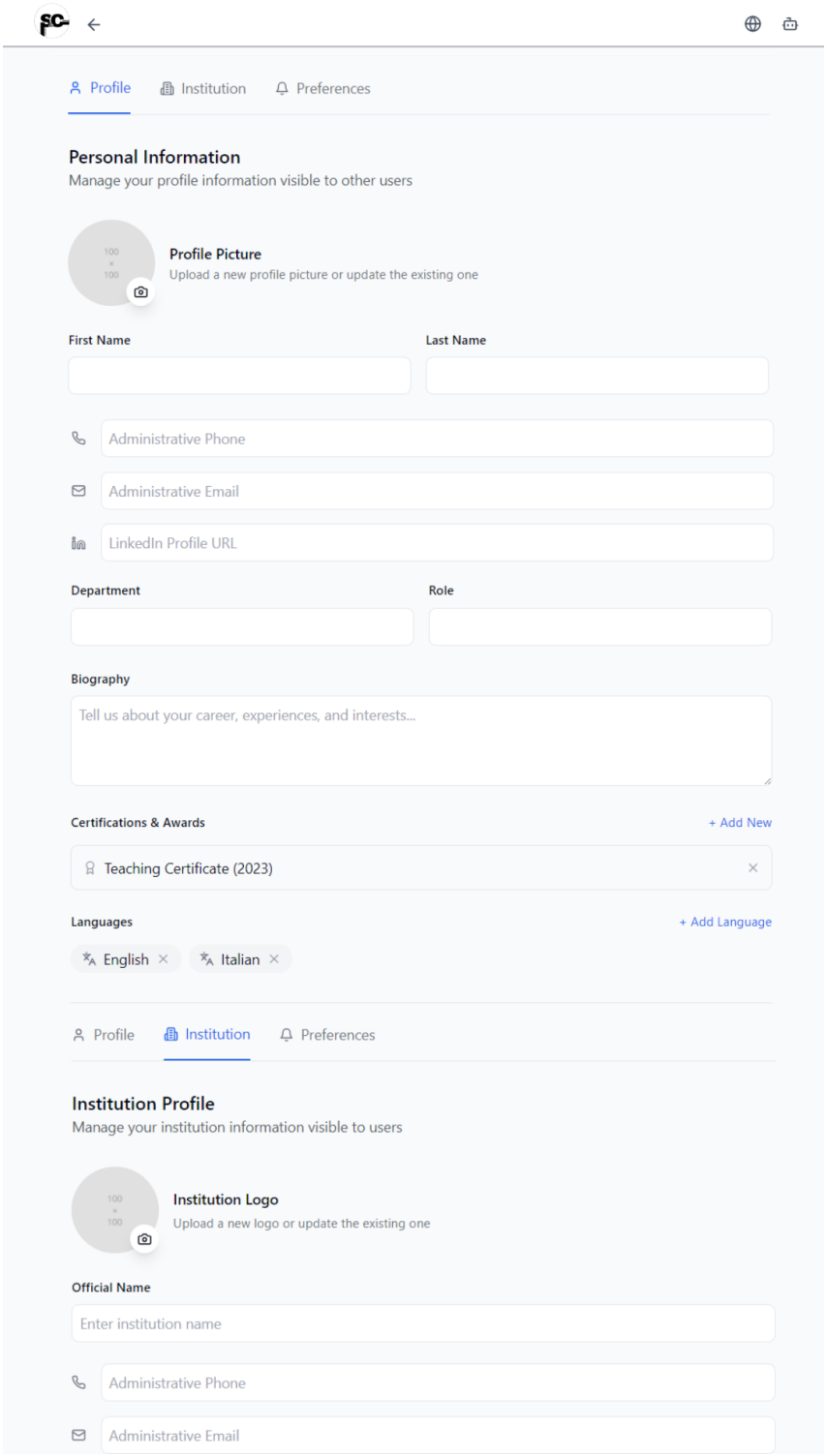

Figure 3.12: Settings Interface.

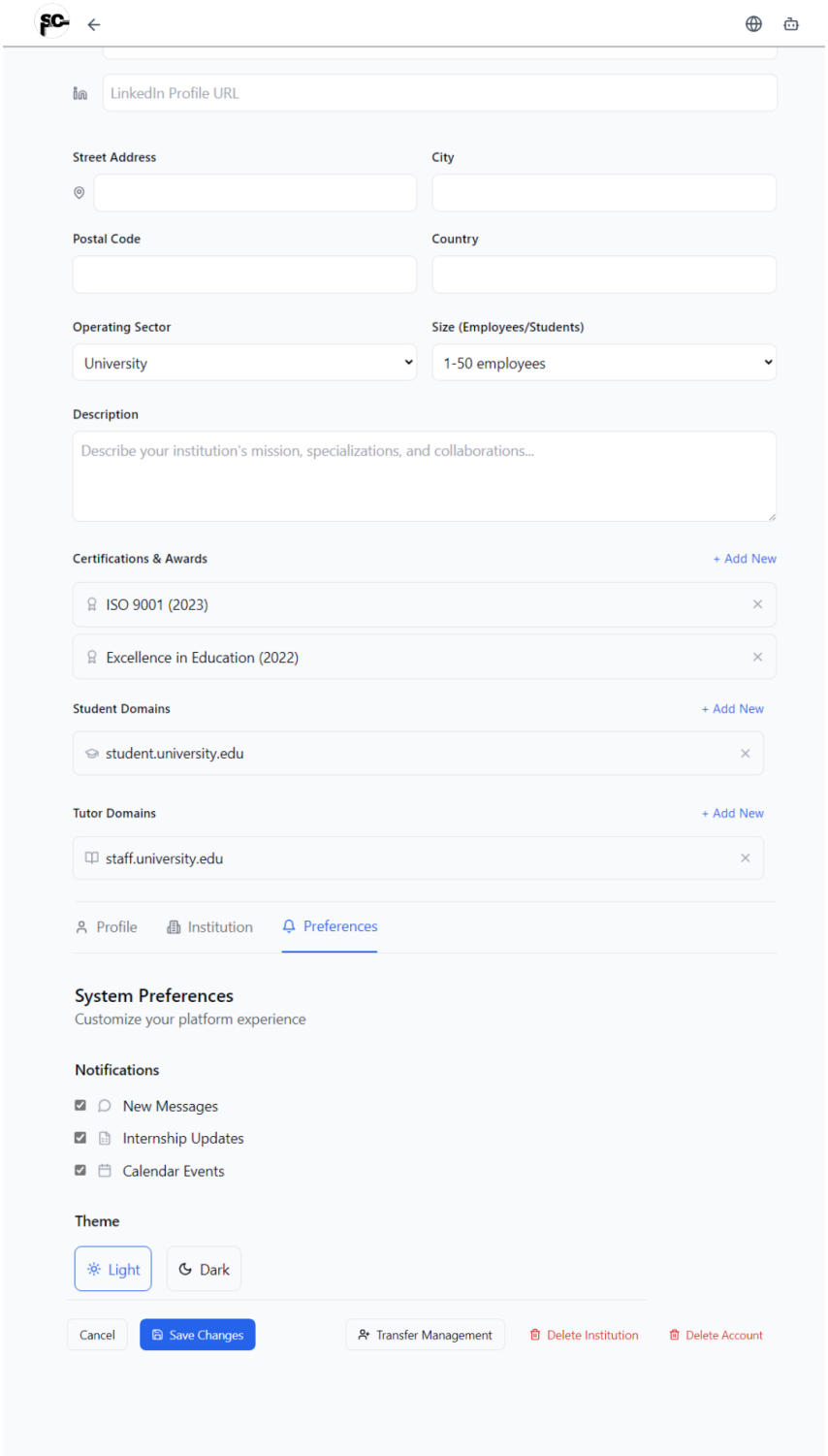

Figure 3.13: Settings Interface.

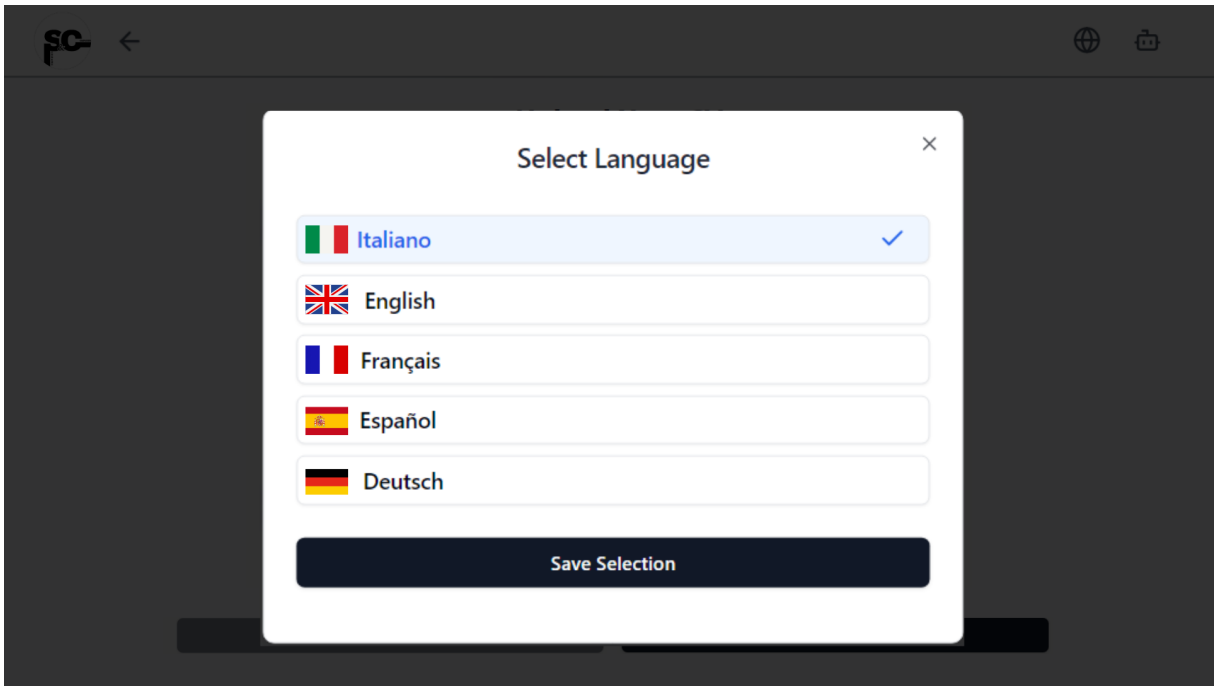

Figure 3.14: Change Language Interface.

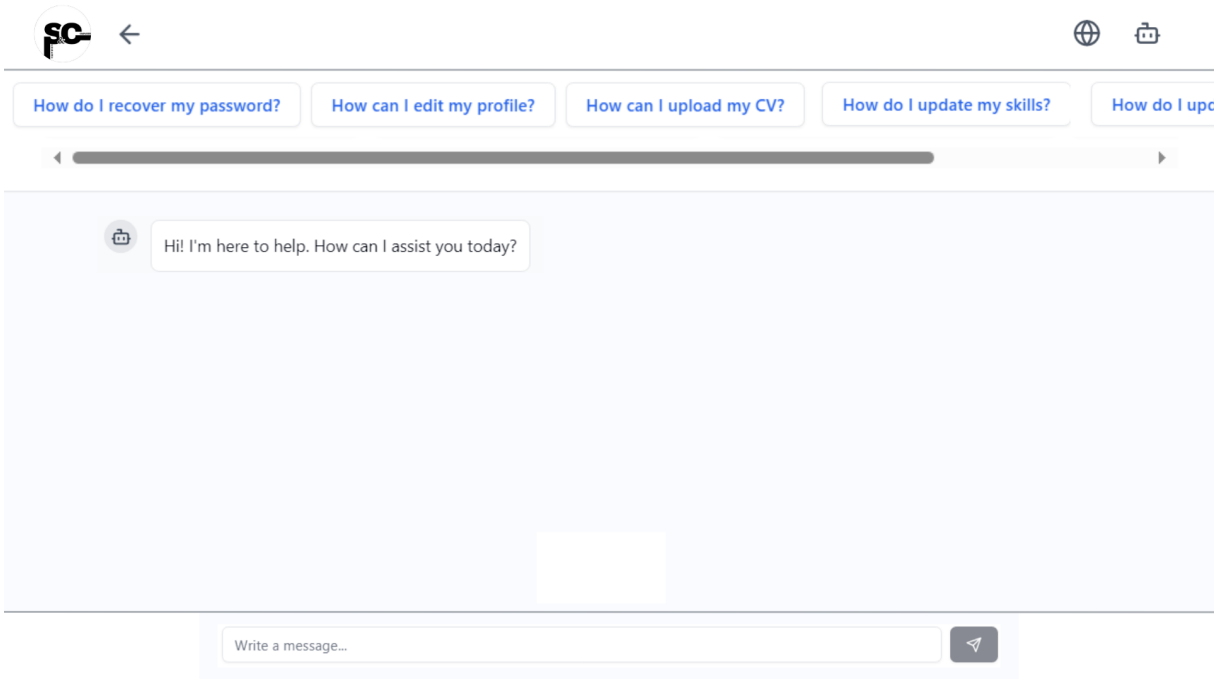

Figure 3.15: Chatbot Assistance Interface.

3.3. Matchmaking

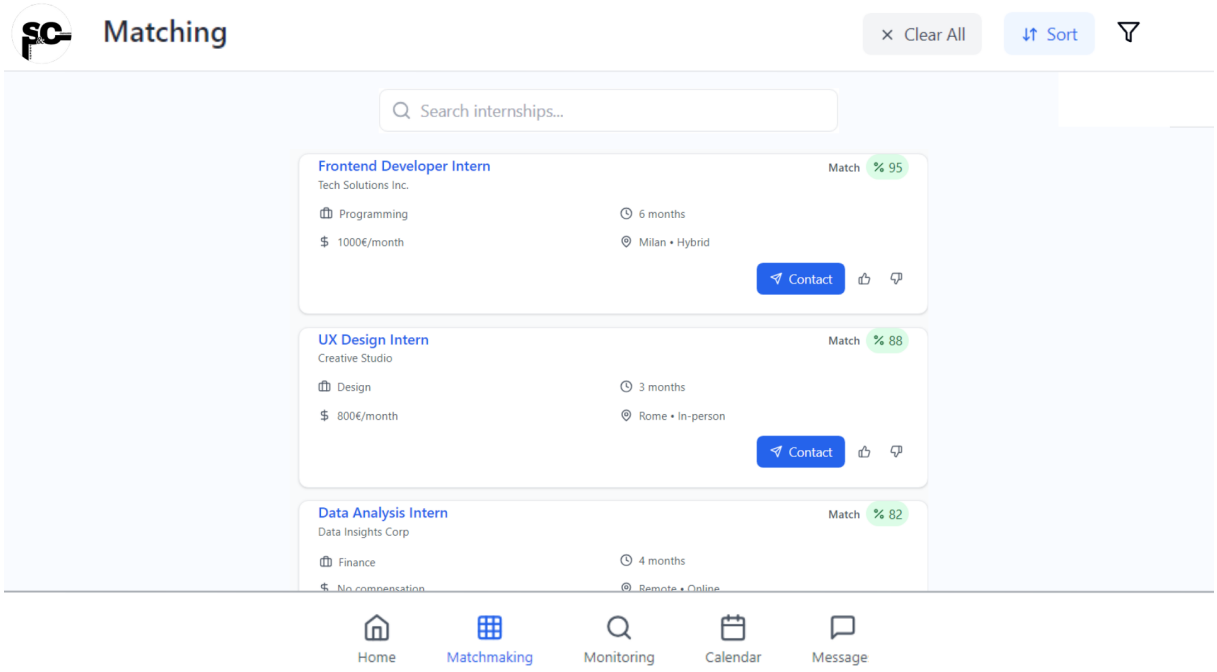

Figure 3.16: Matchmaking Interface for Students.

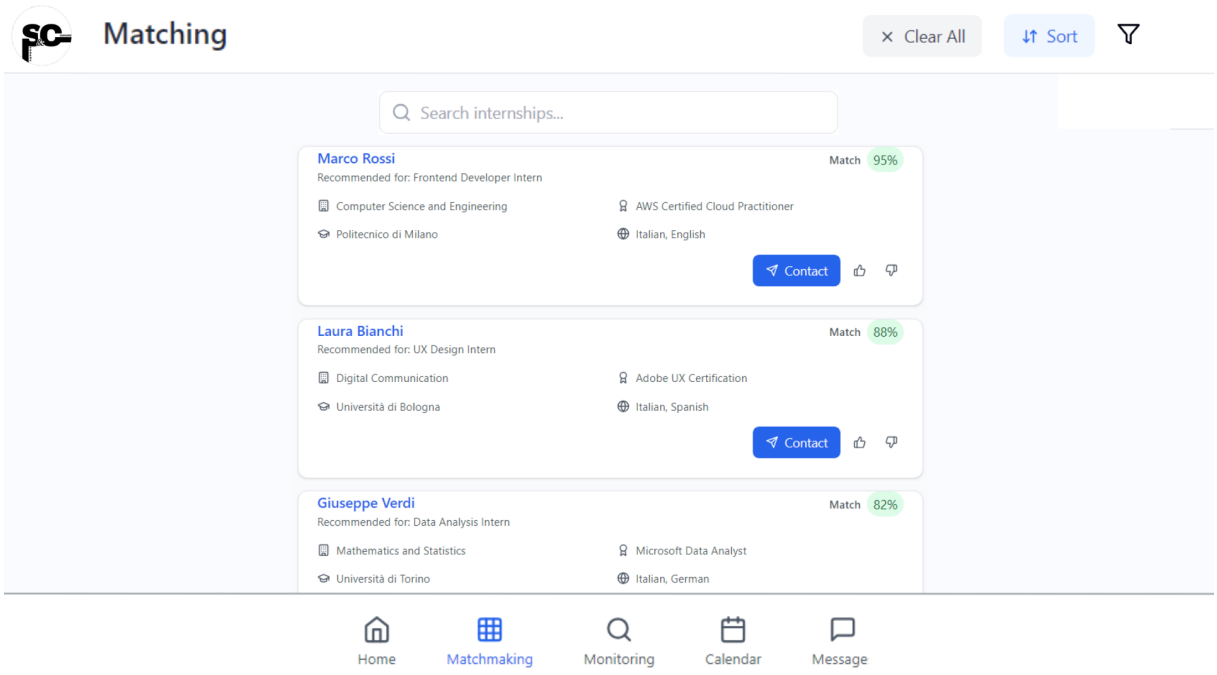

Figure 3.17: Matchmaking Interface for Company Tutors.

3.4. Monitoring: Selection Process, Active Stages and Questionnaires

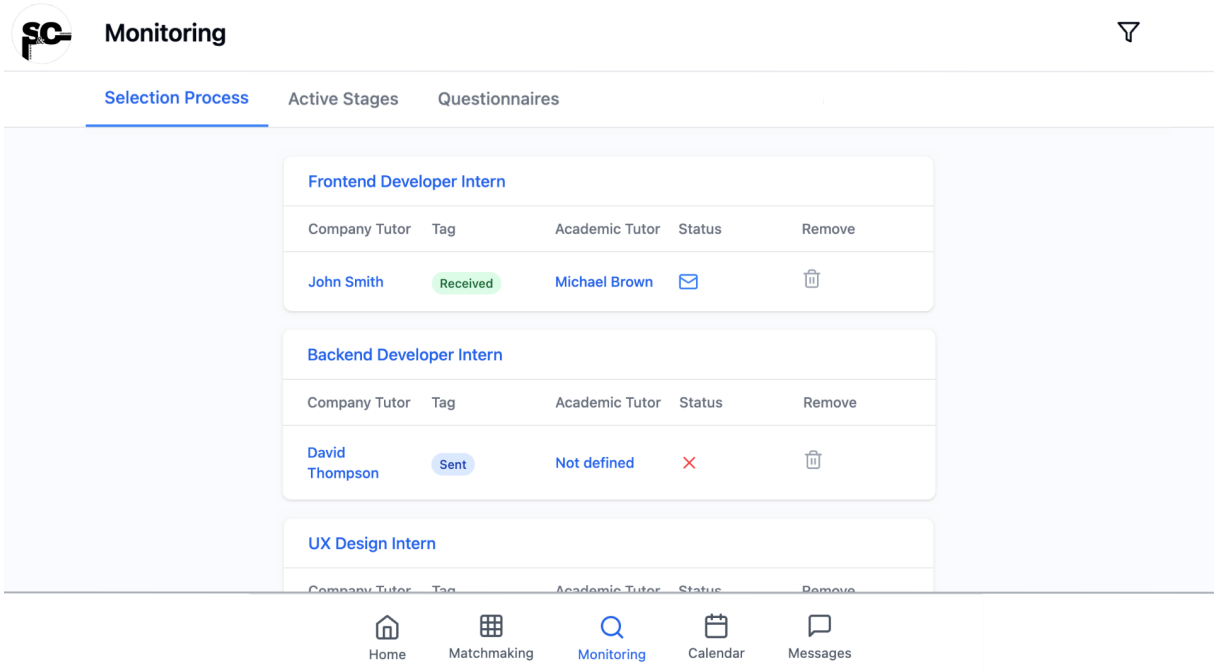

Figure 3.18: Selection Process Interface for Students.

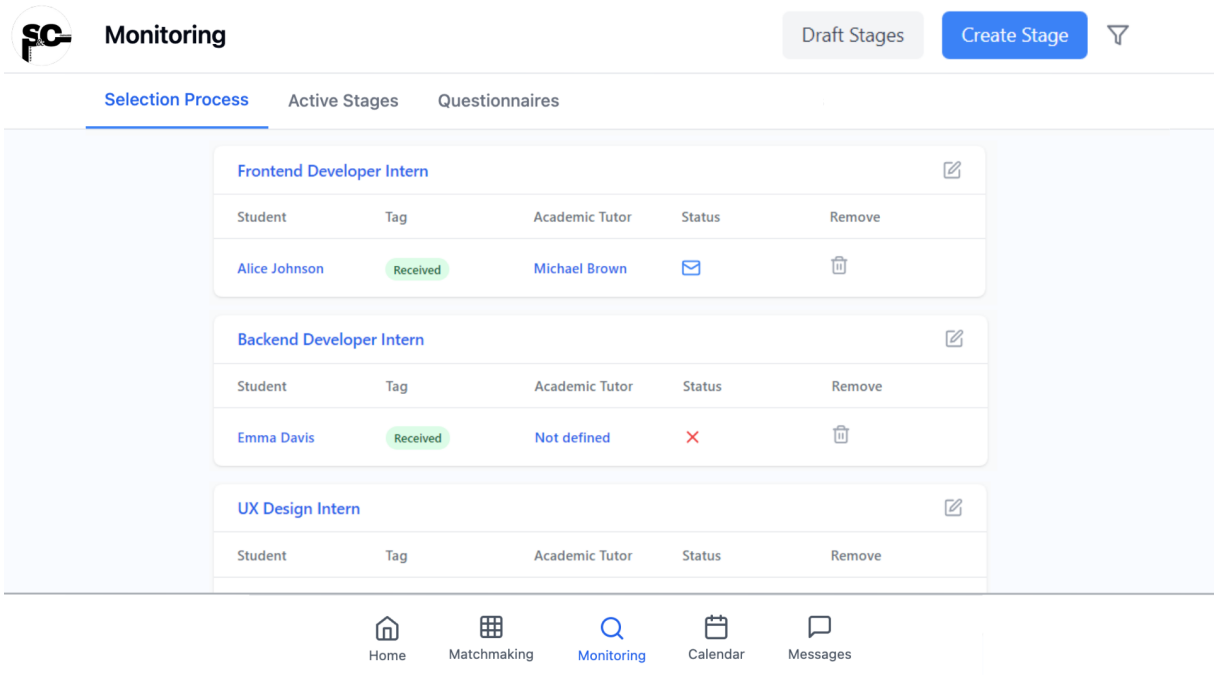

Figure 3.19: Selection Process Interface for Company Tutors.

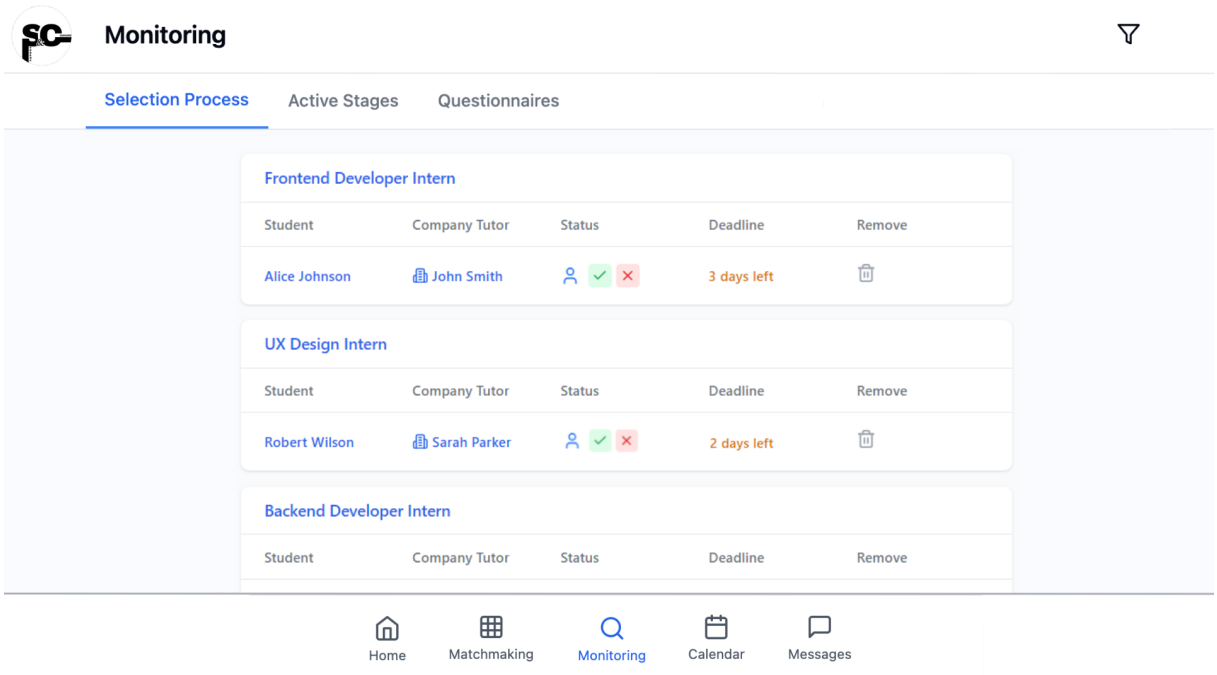

Figure 3.20: Selection Process Interface for Academic Tutors.

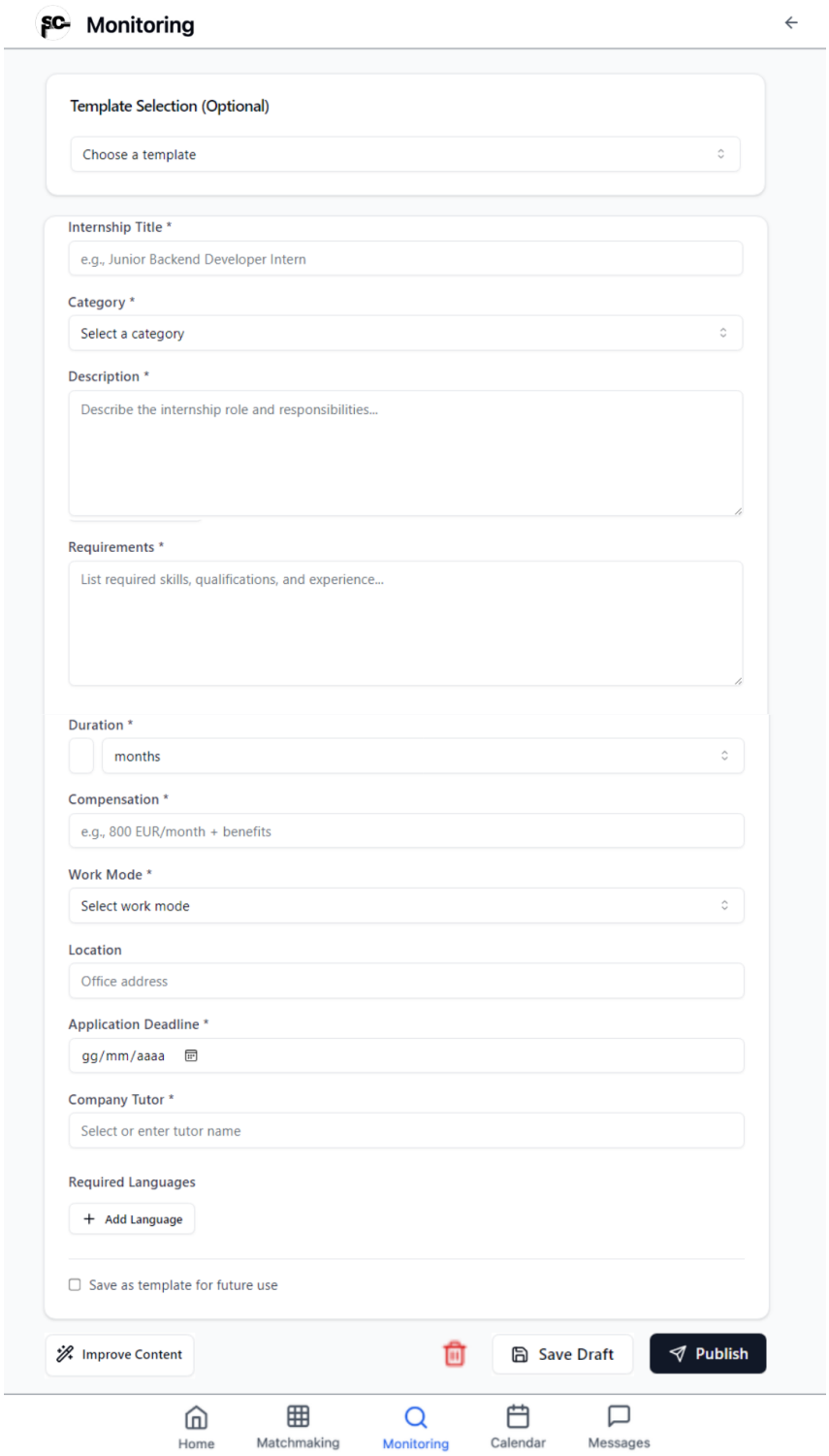

Figure 3.21: Selection Process - Internship Creation Interface for Company Tutors.

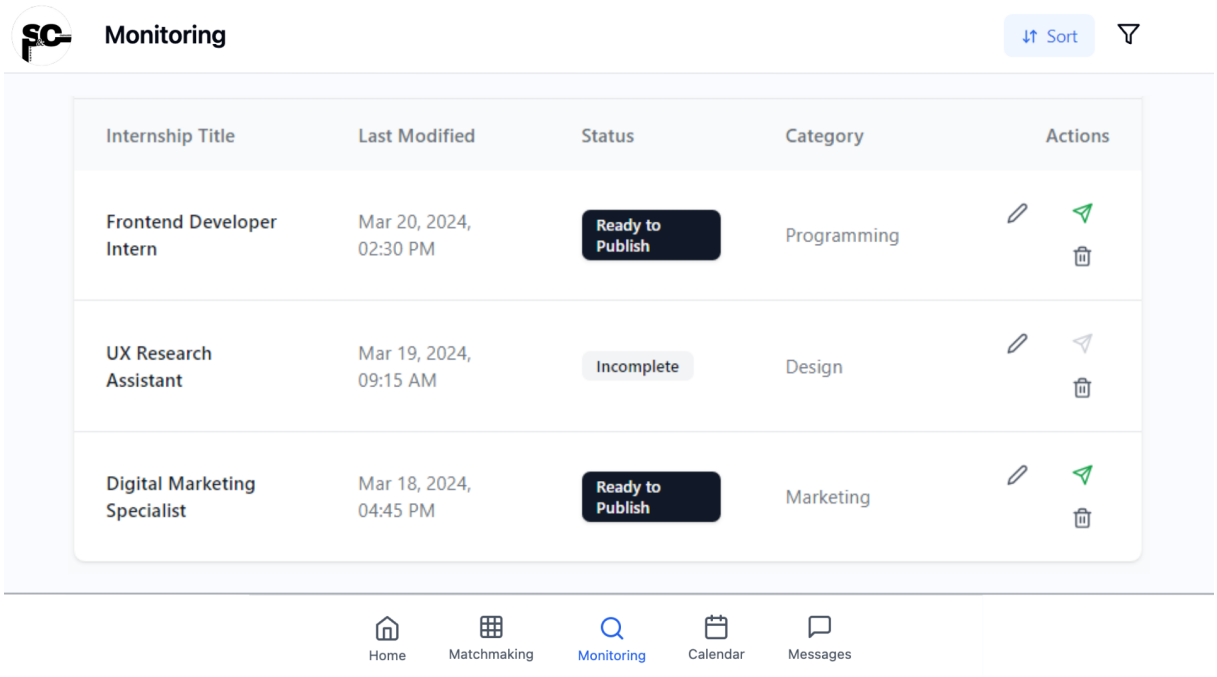

Figure 3.22: Selection Process - Drafts Interface for Company Tutors.

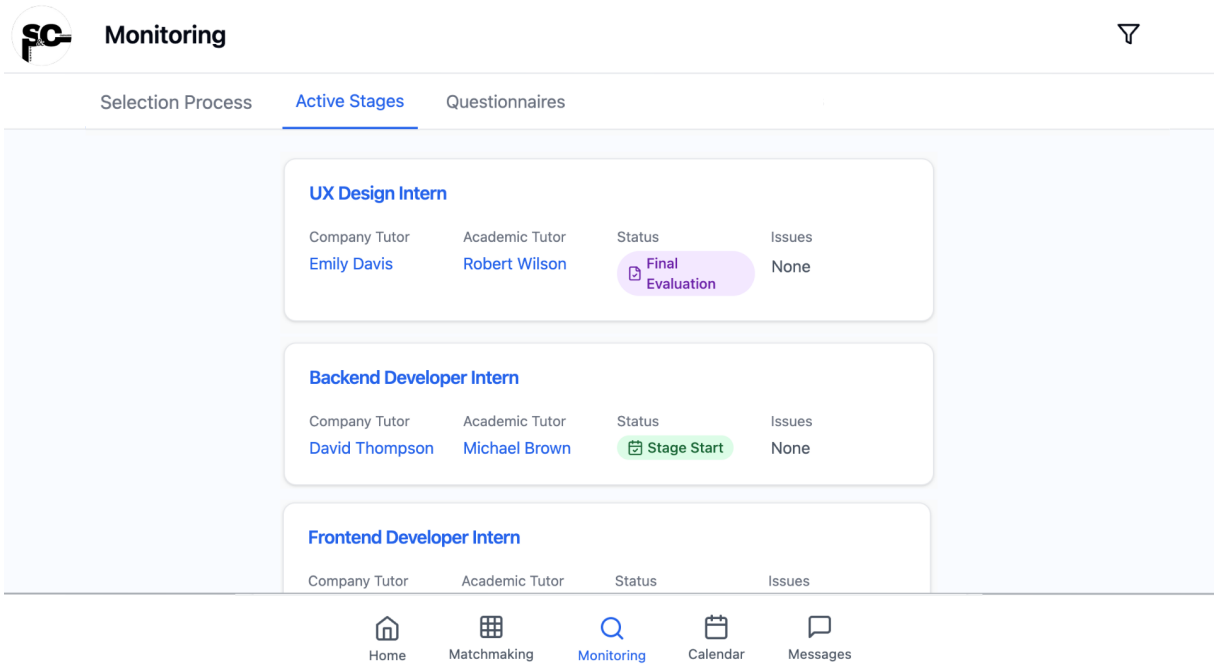

Figure 3.23: Active Stages Interface for Students.

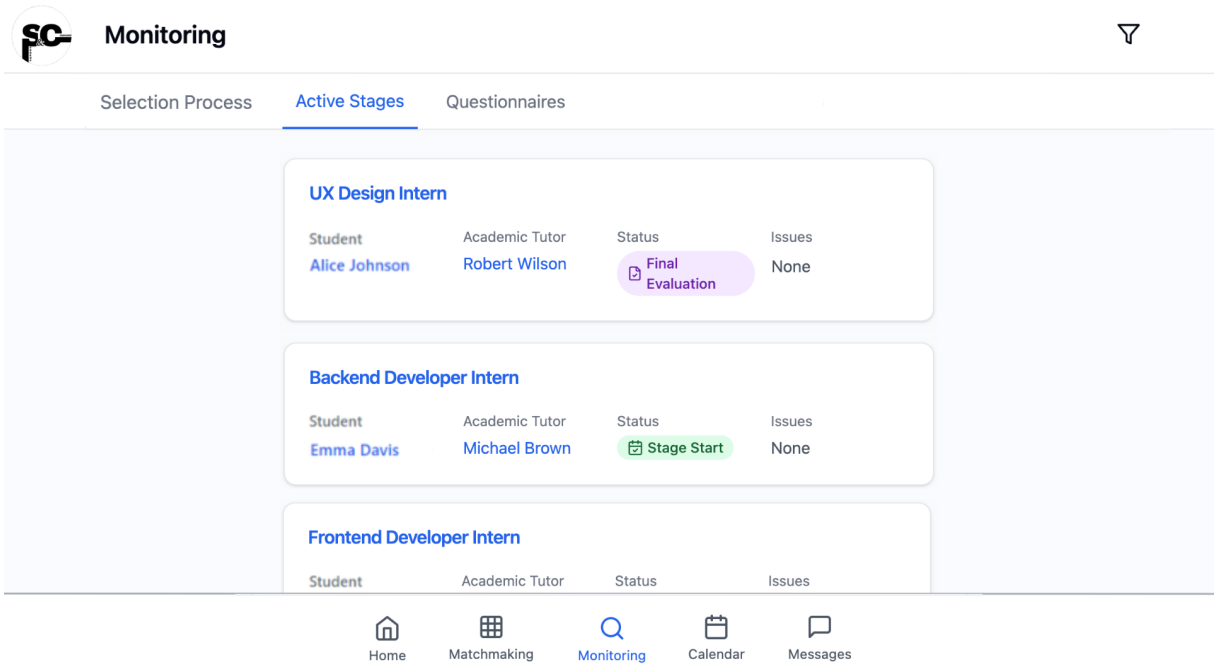

Figure 3.24: Active Stages Interface for Company Tutors.

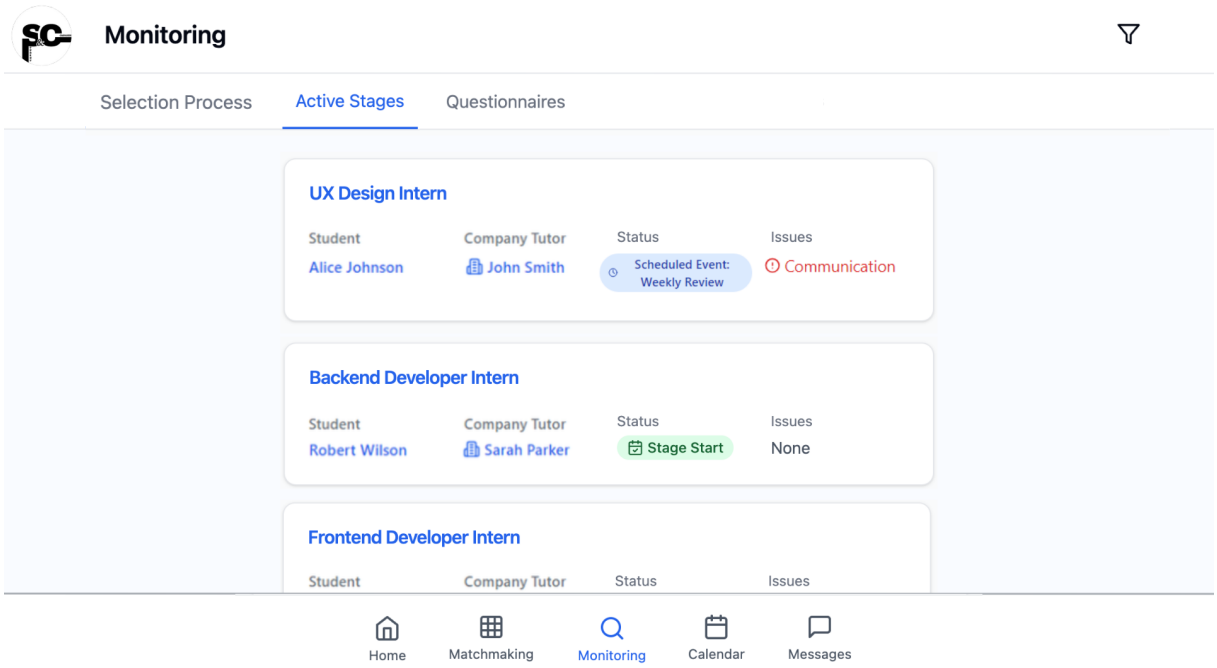

Figure 3.25: Active Stages Interface for Academic Tutors.

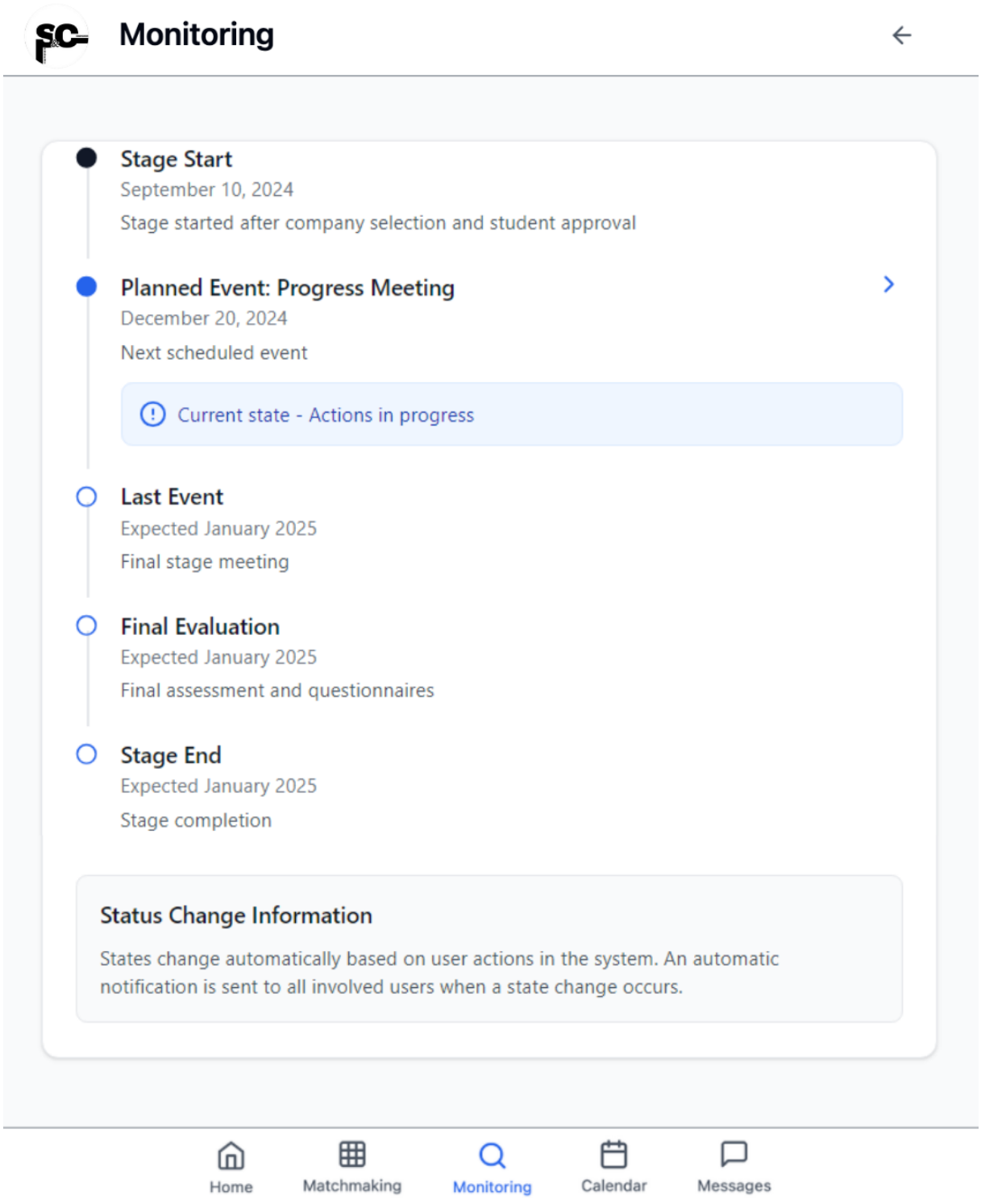

Figure 3.26: Active Stages - States History Interface.

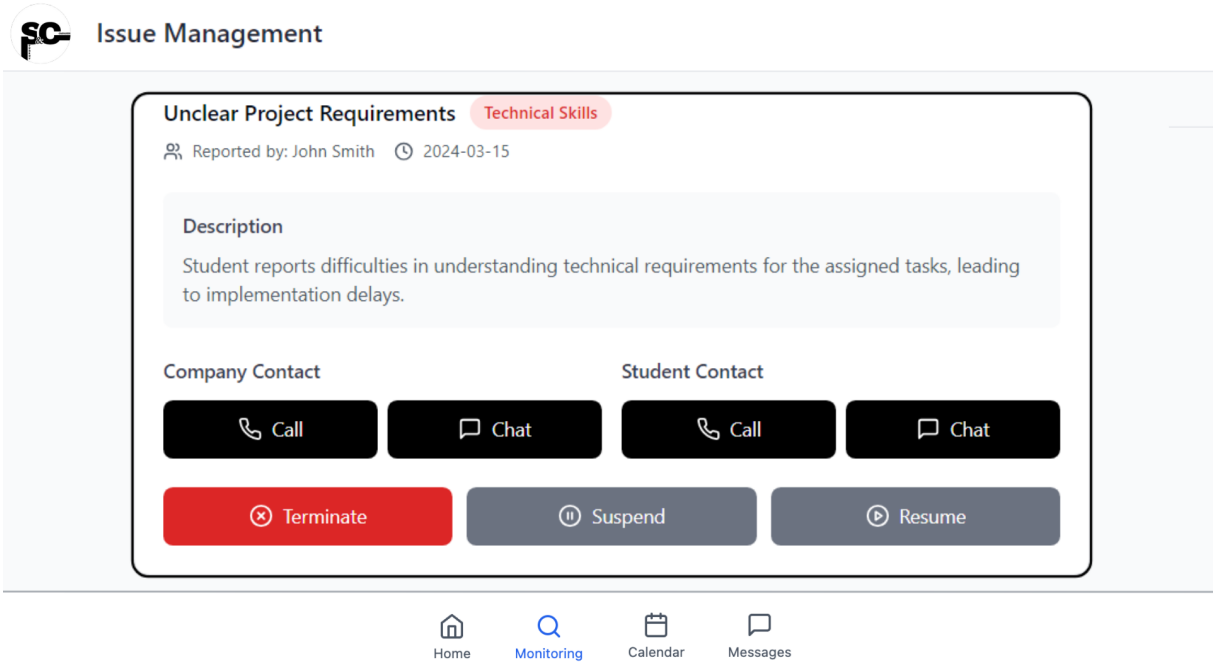

Figure 3.27: Active Stages - Issue Management Interface for Academic Tutors.

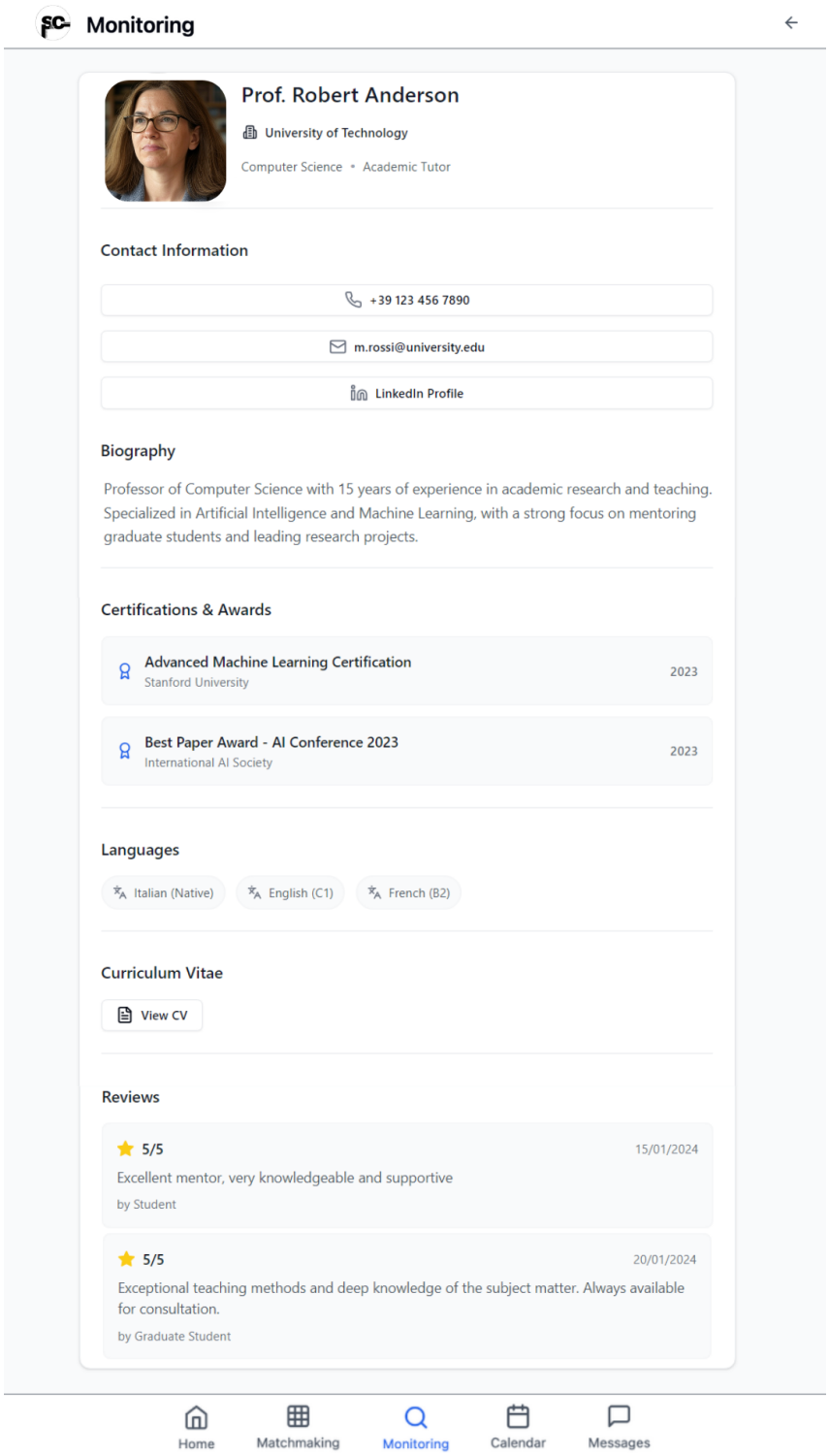

Figure 3.28: Selection Process/Active Stages - Personal Profile Visualization Interface.

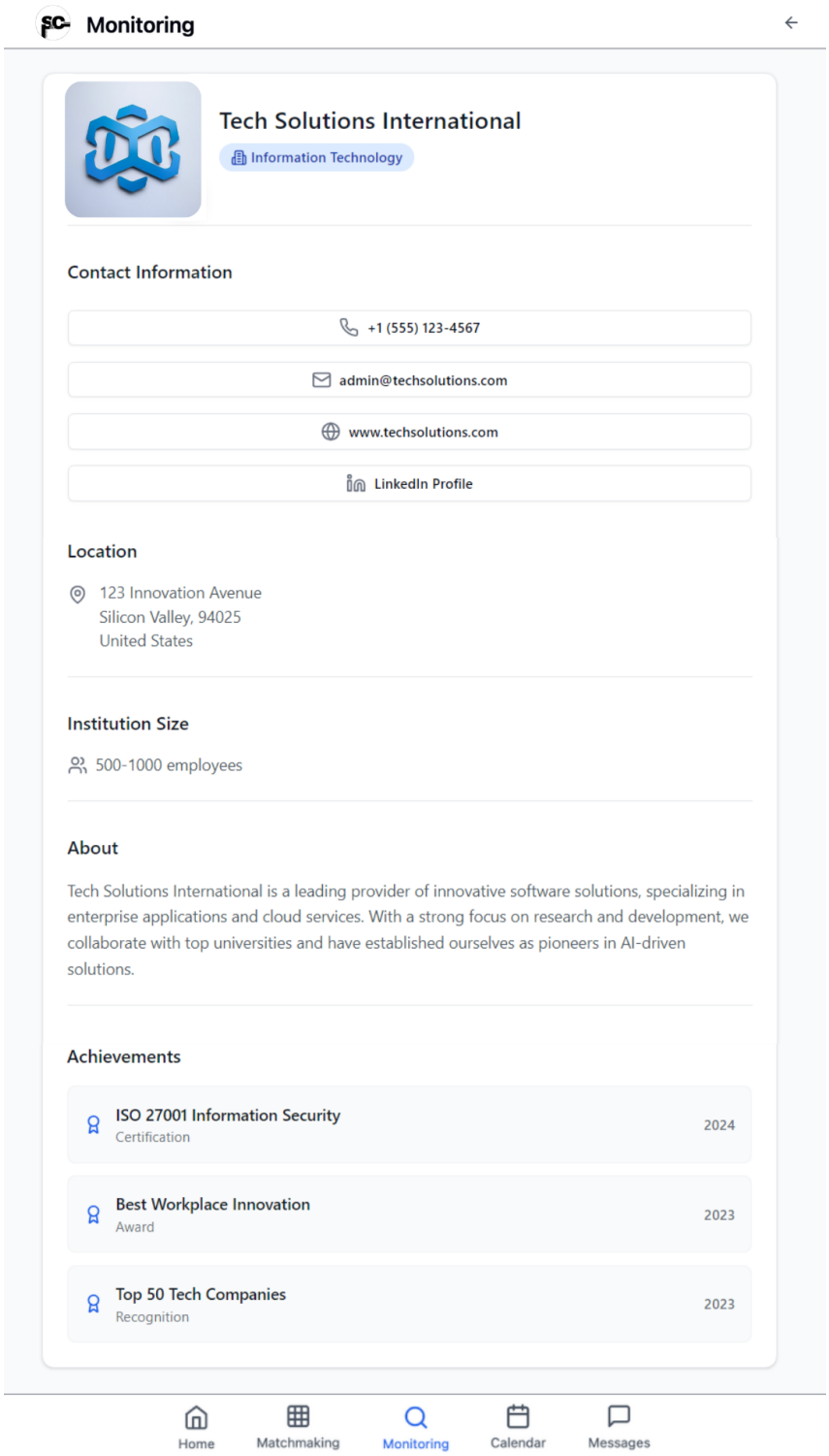

Figure 3.29: Selection Process/Active Stages - Institution Visualization Interface.

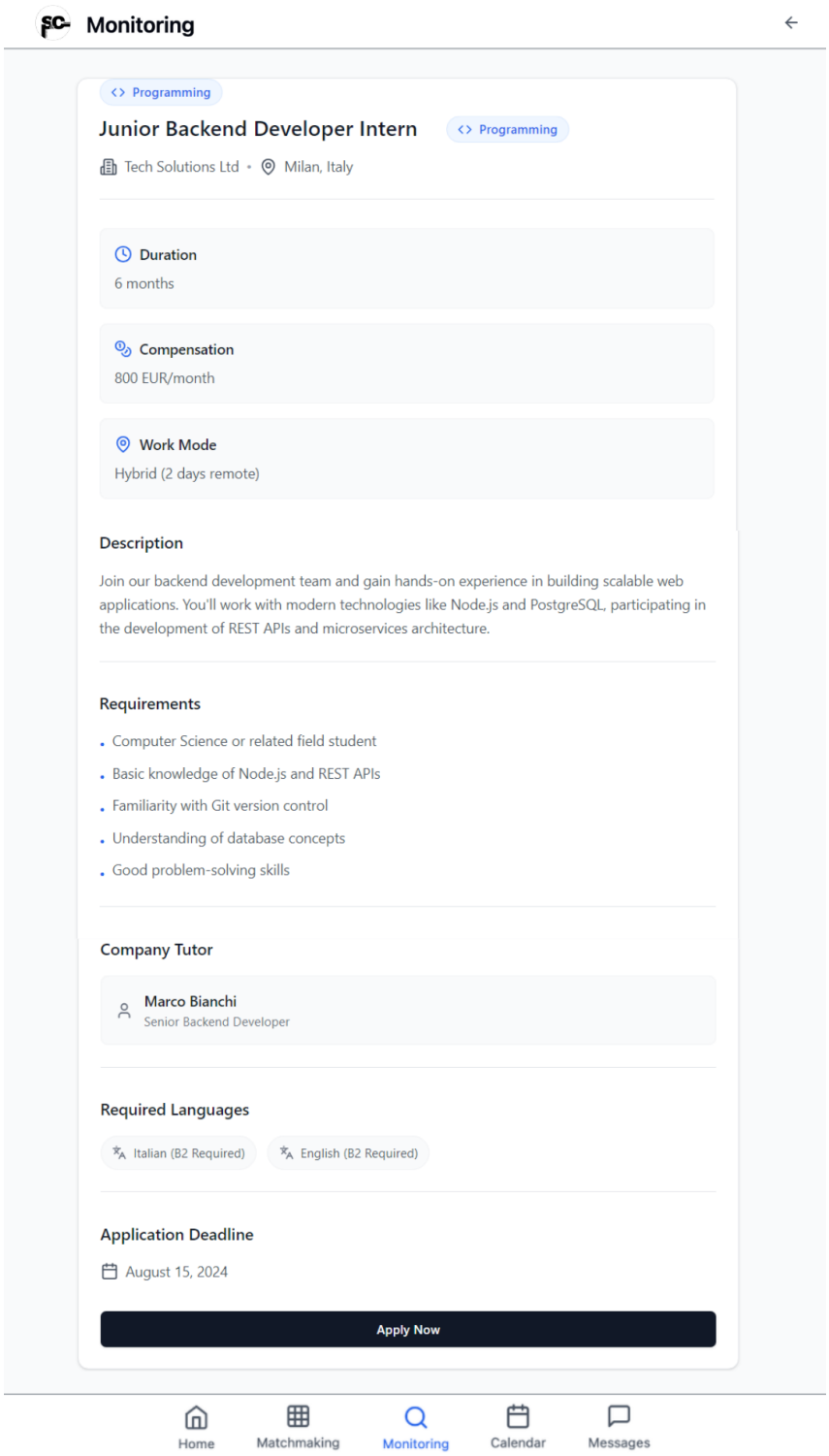

Figure 3.30: Selection Process/Active Stages - Internship Visualization Interface.

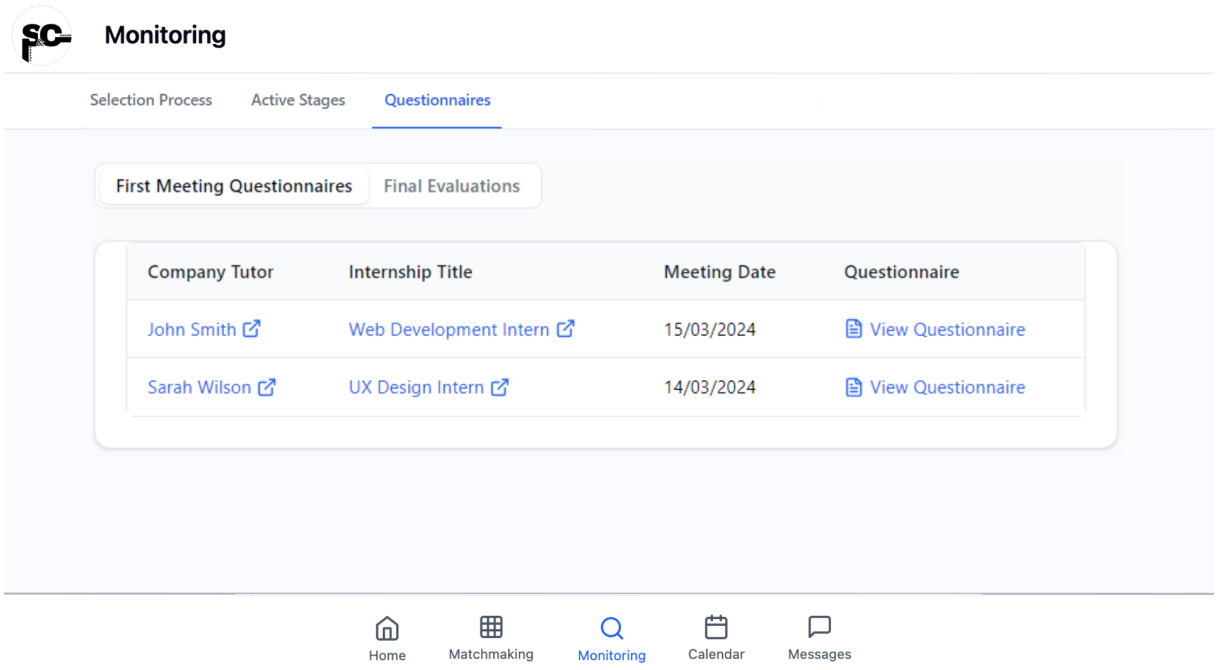

Figure 3.31: First Meeting Questionnaires Interface for Students.

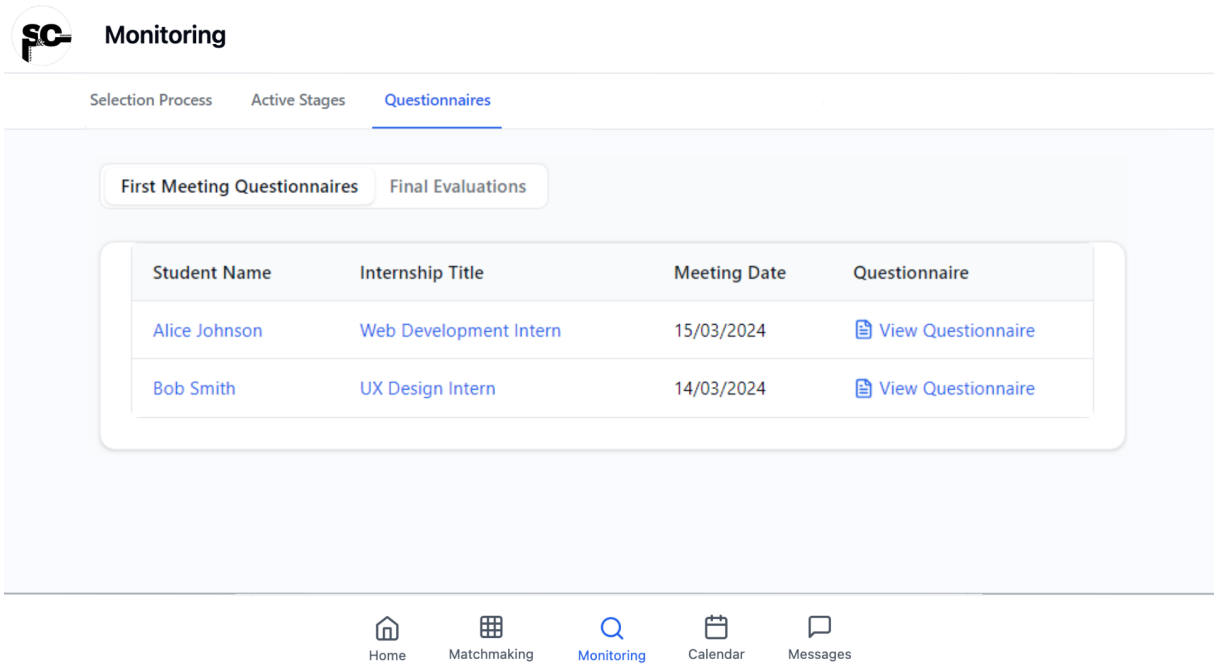

Figure 3.32: First Meeting Questionnaires Interface for Company Tutors.

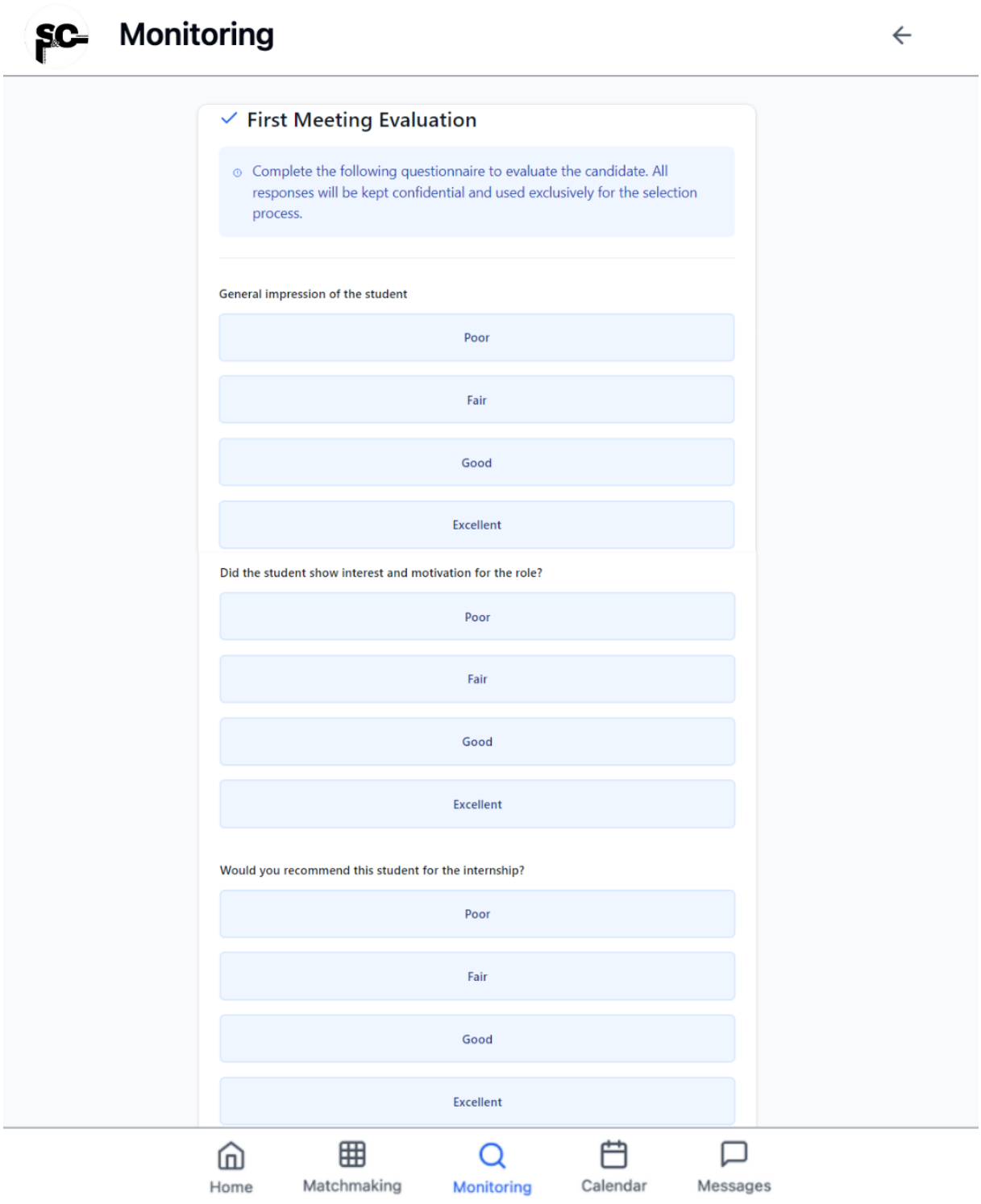

Figure 3.33: First Meeting Questionnaire Creation Interface.

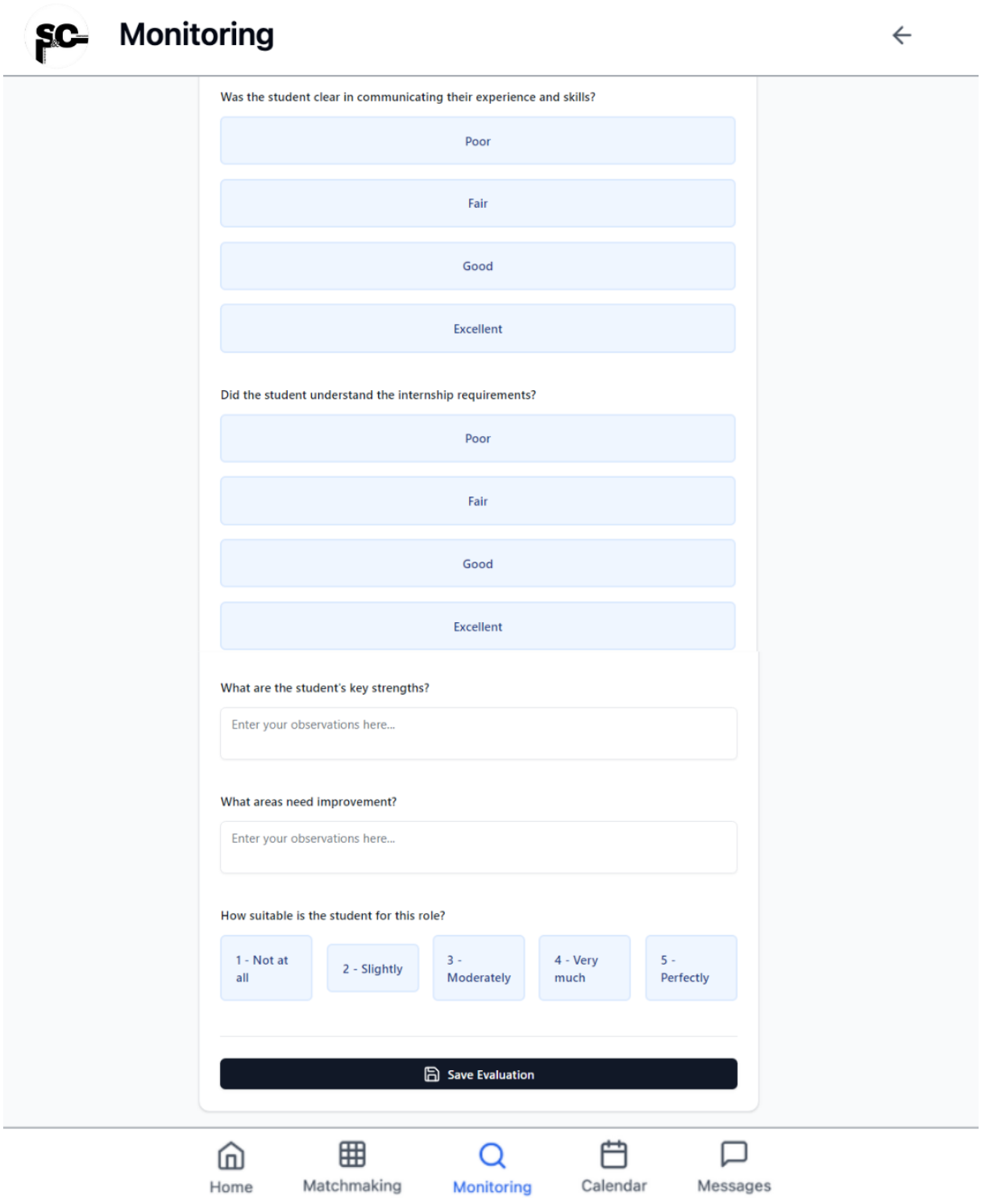

Figure 3.34: First Meeting Questionnaire Creation Interface.

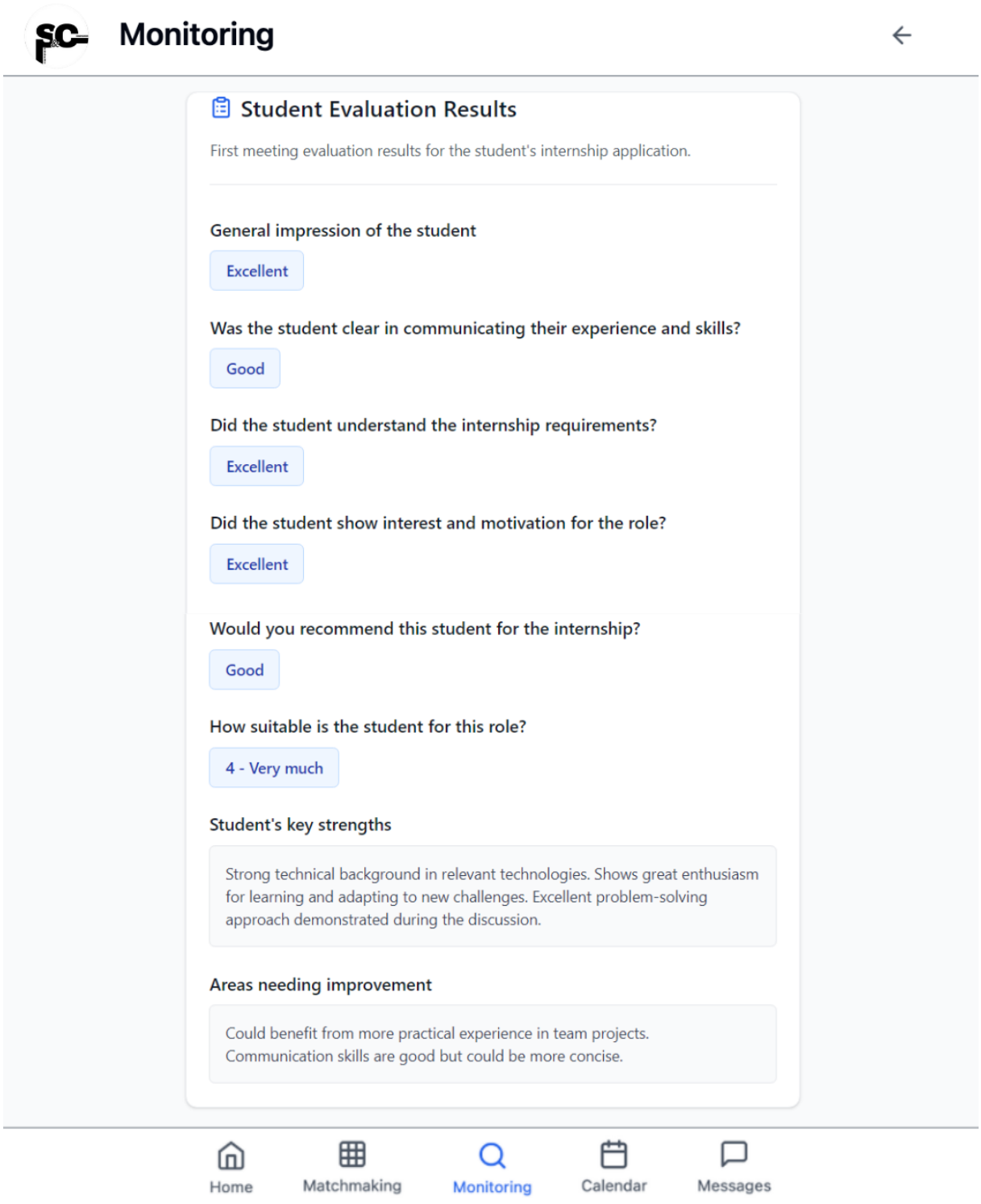

Figure 3.35: First Meeting Questionnaire Visualization Interface.

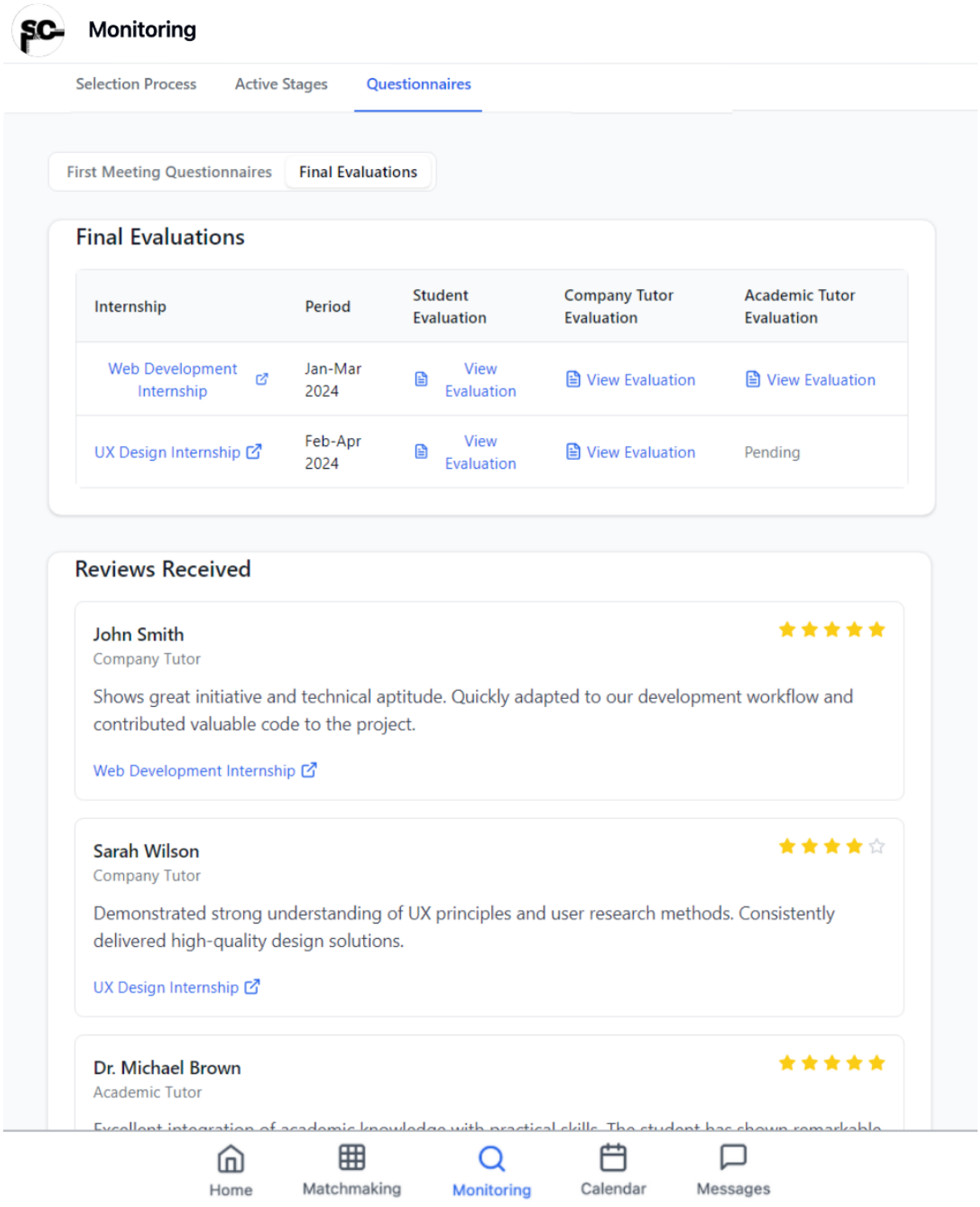

Figure 3.36: Final Evaluations Interface for Students.

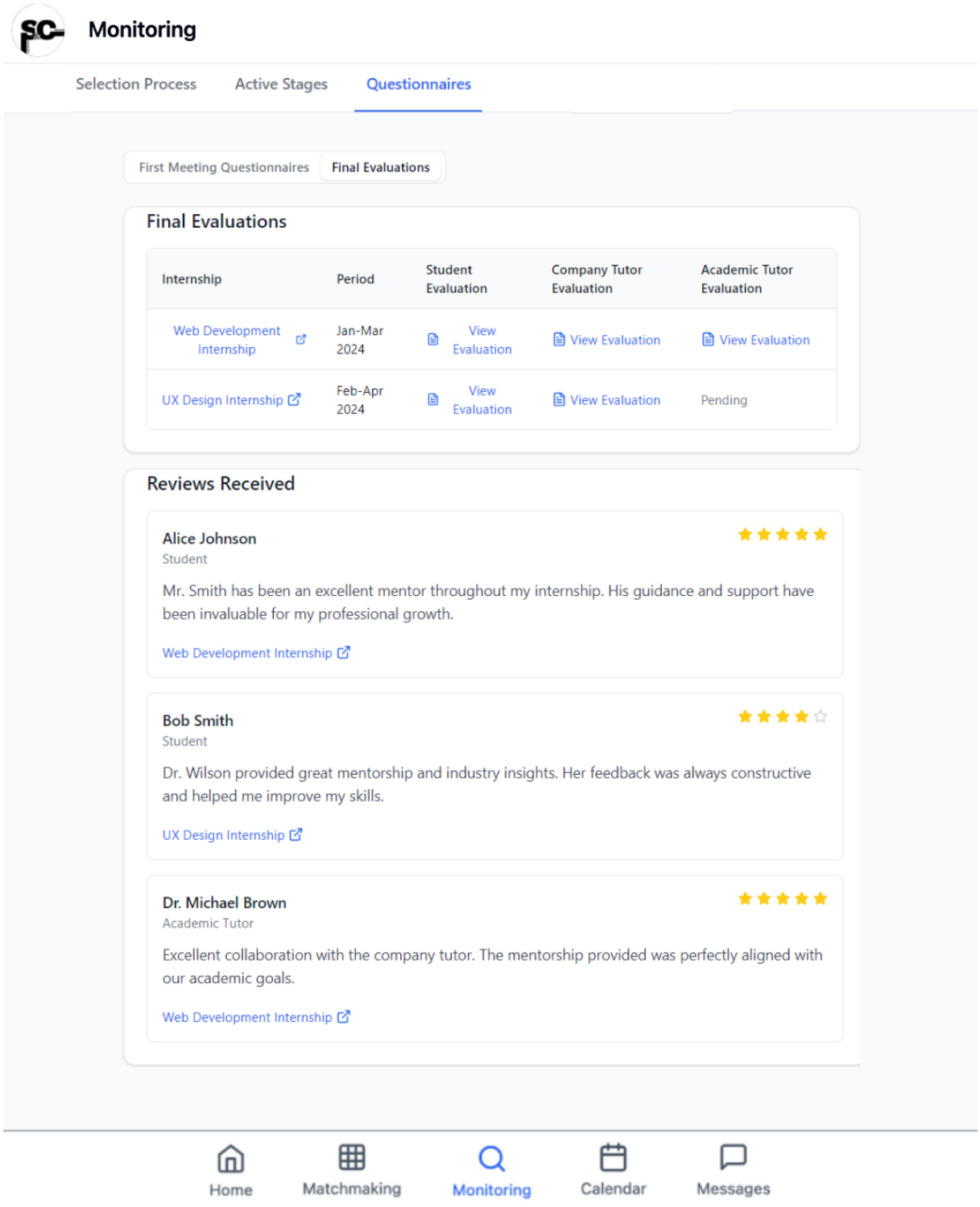

Figure 3.37: Final Evaluations Interface for Company Tutors.

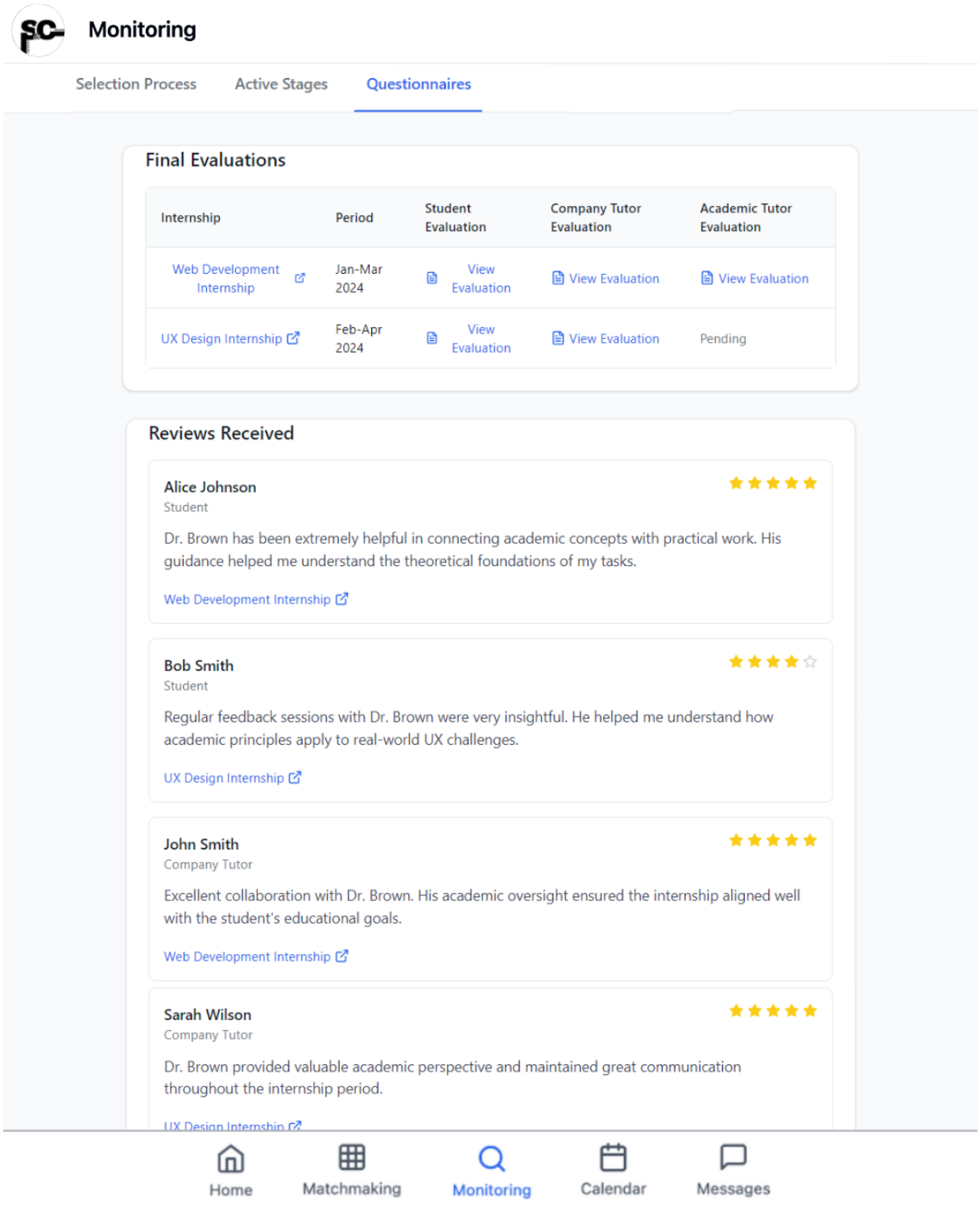

Figure 3.38: Final Evaluations Interface for Academic Tutors.

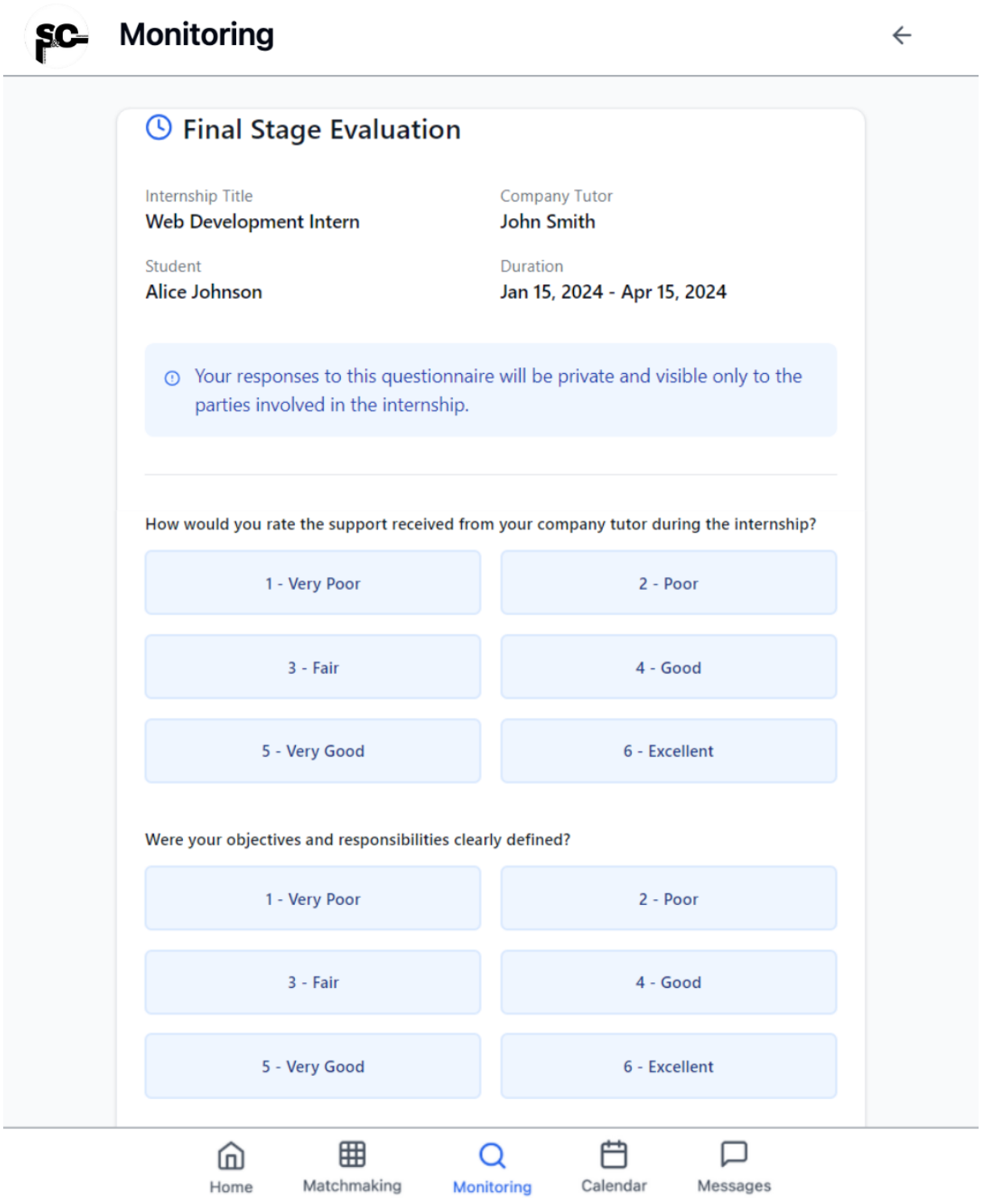

Figure 3.39: Final Evaluation Creation Interface.

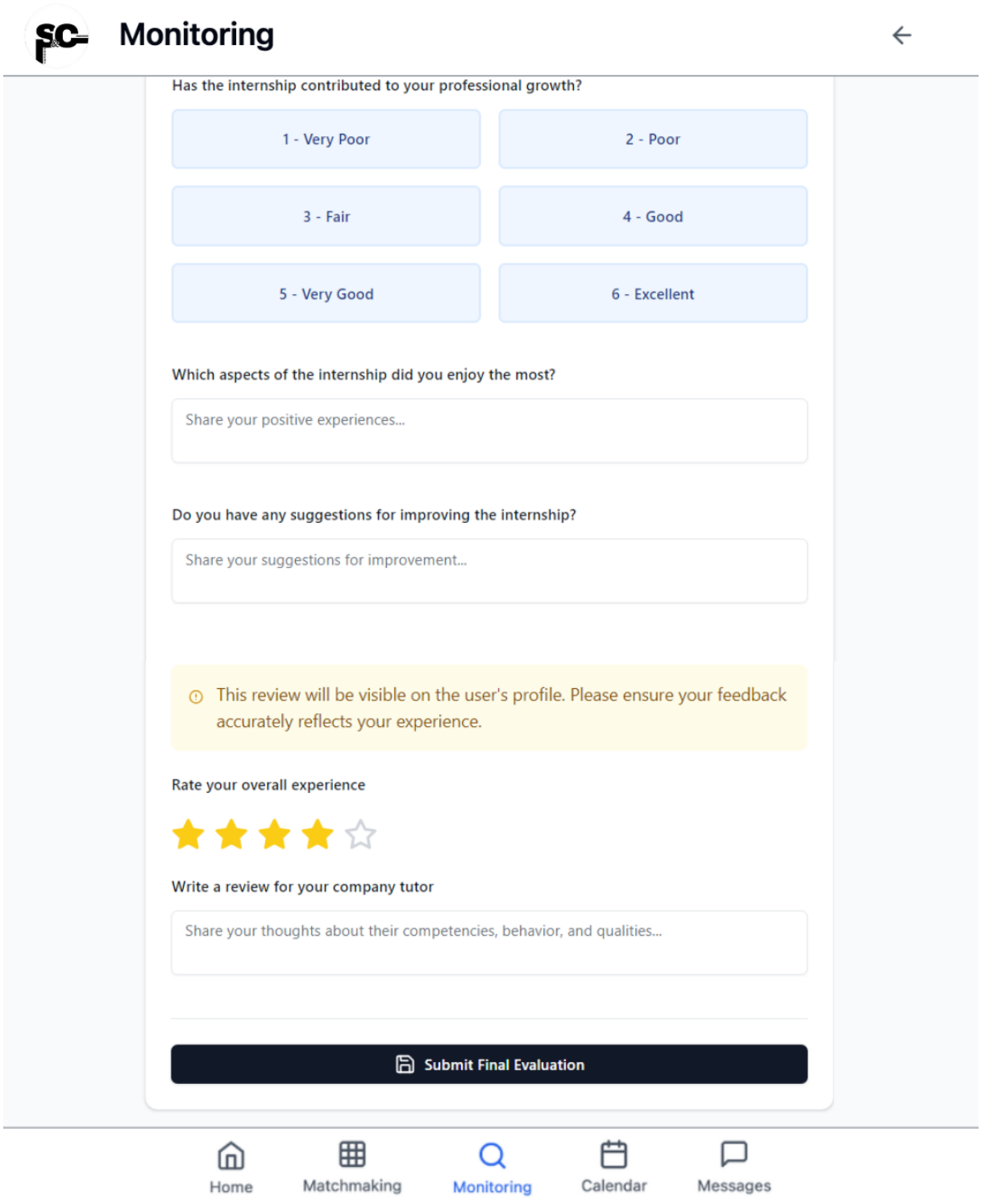

Figure 3.40: Final Evaluation Creation Interface.

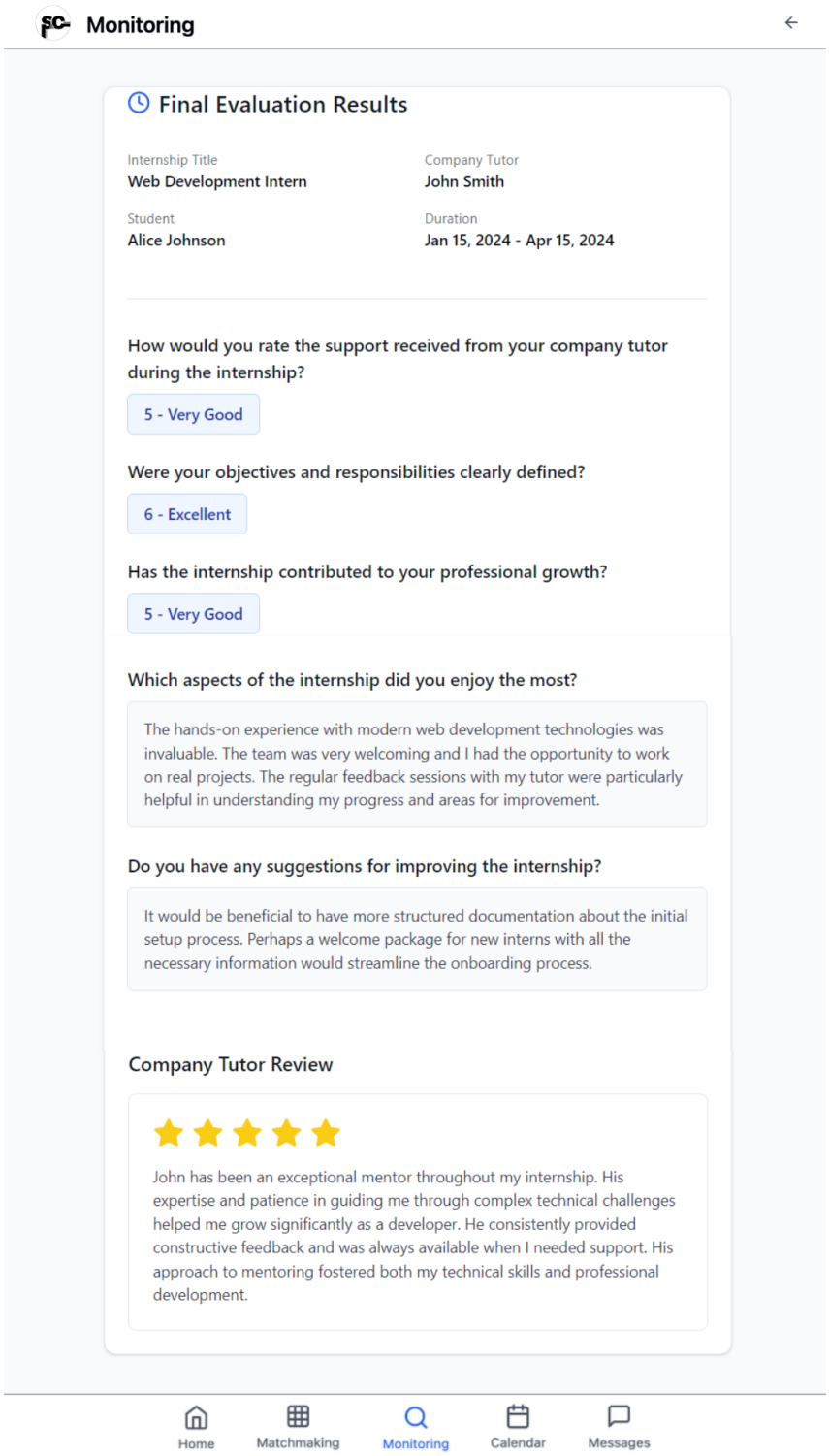

Figure 3.41: Final Evaluation Visualization Interface.

3.5. Calendar

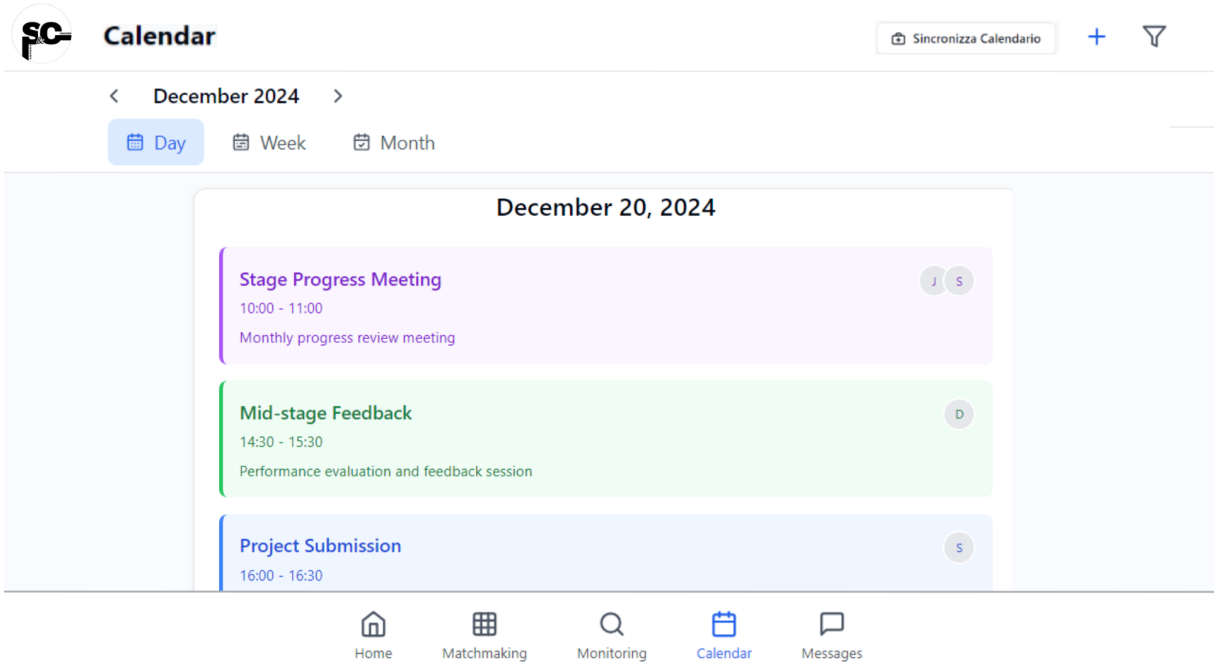

Figure 3.42: Calendar in "Day" Visualization Interfaces.

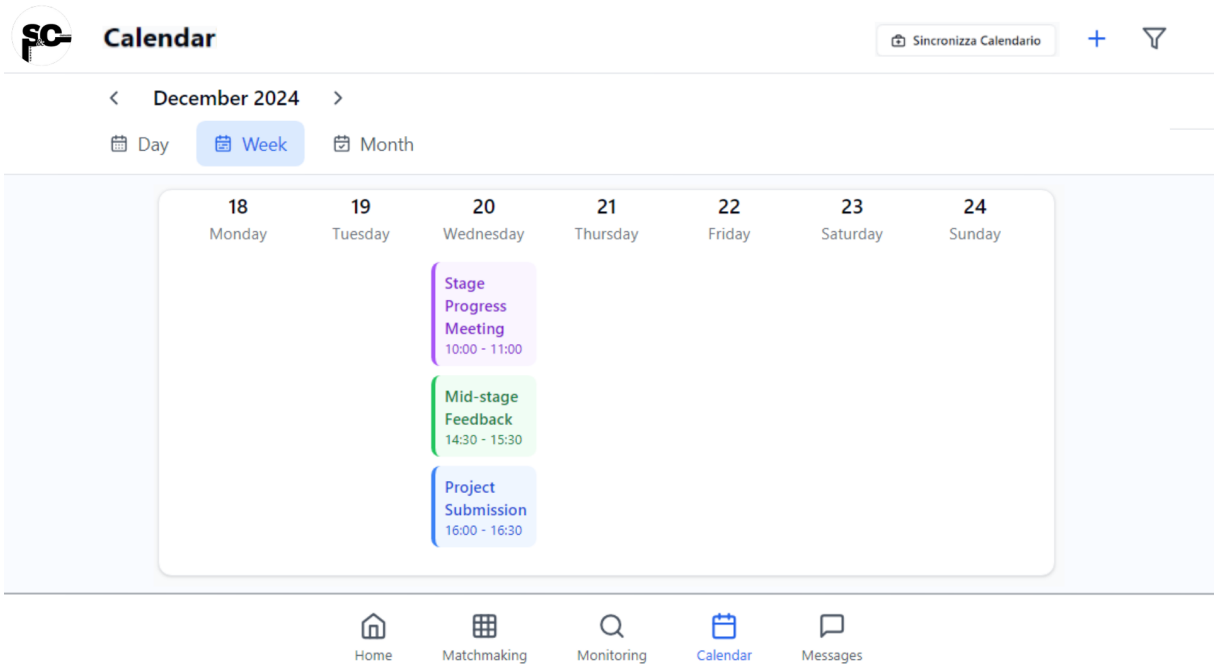

Figure 3.43: Calendar in "Week" Visualization Interface.

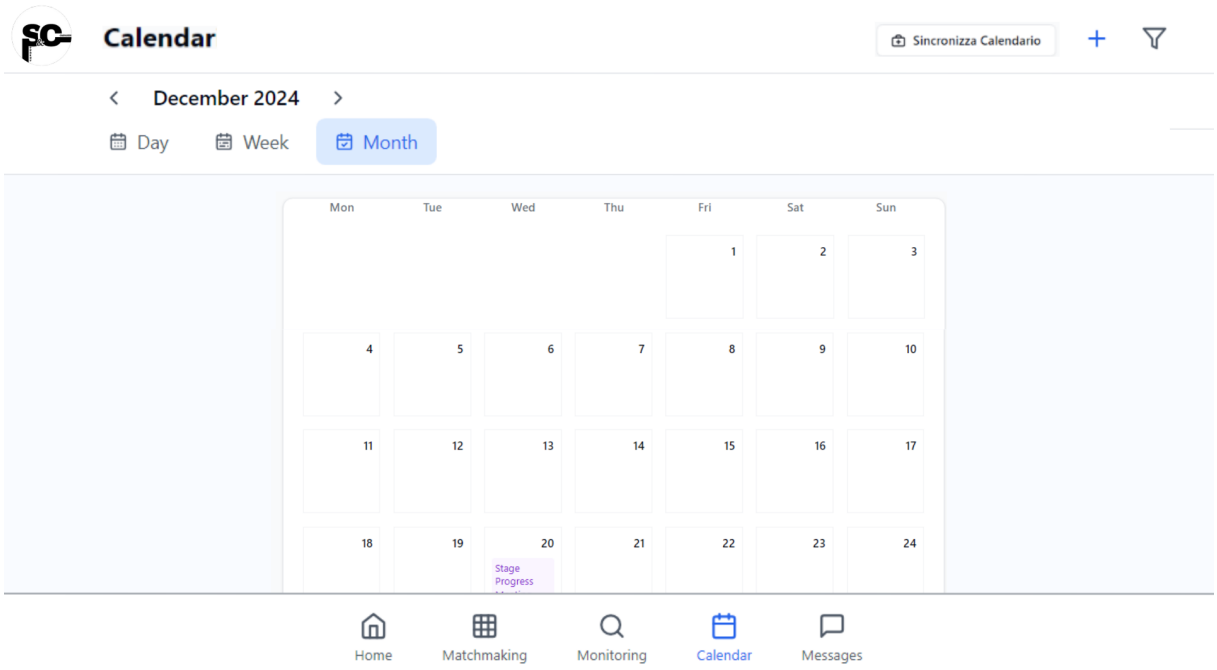

Figure 3.44: Calendar in "Month" Visualization Interface.

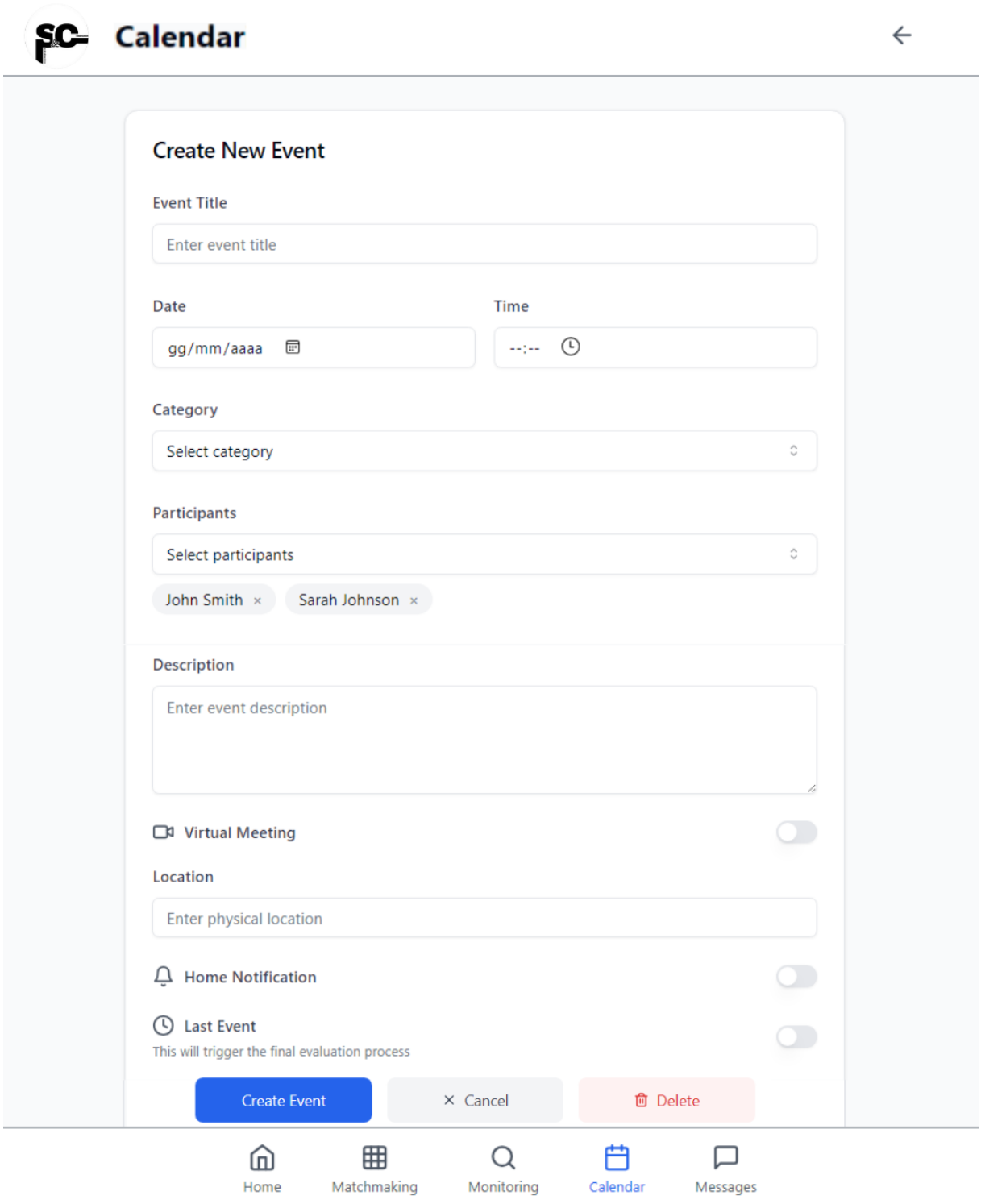

Figure 3.45: Calendar - Event Creation Interface.

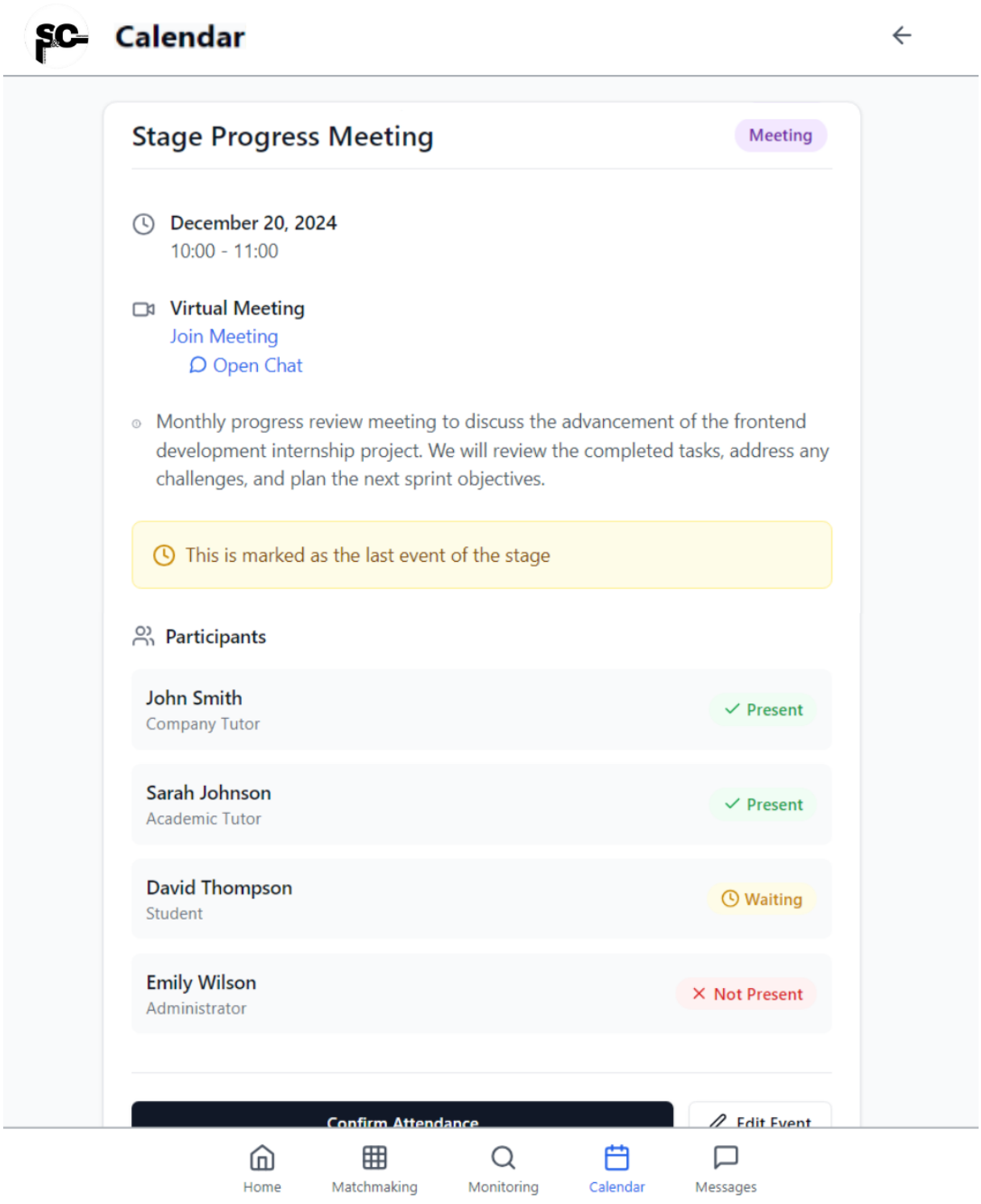

Figure 3.46: Calendar - Event Visualization Interface.

3.6. Messaging with Issues and Video-calls

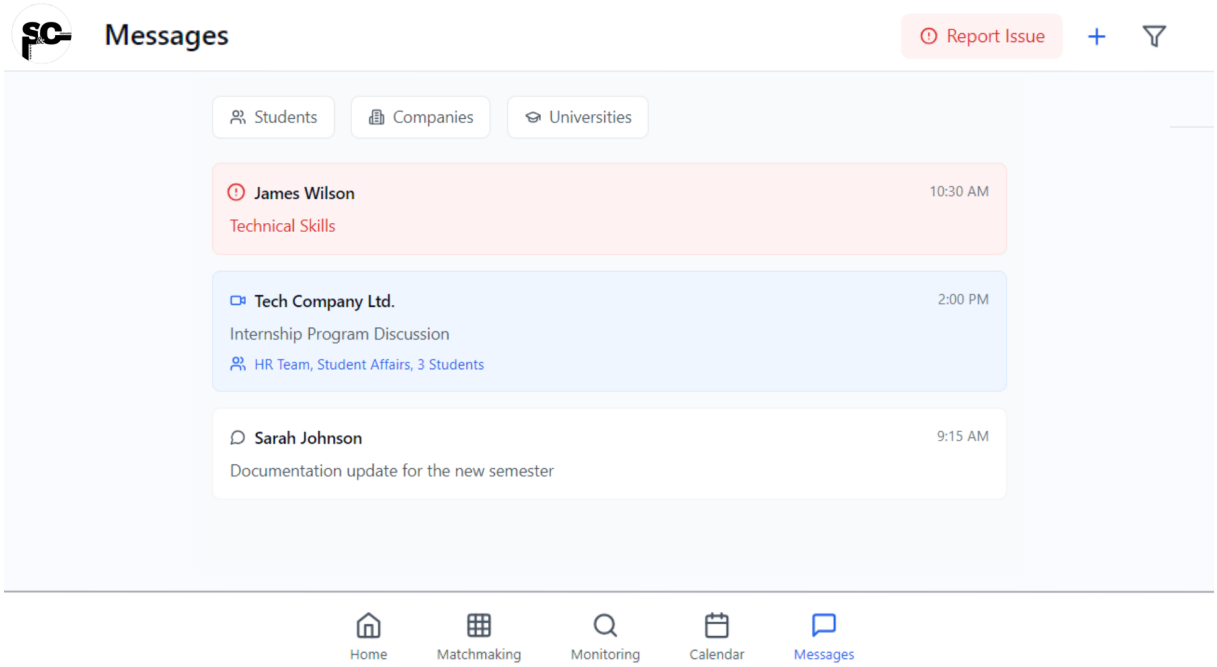

Figure 3.47: Messaging Interface.

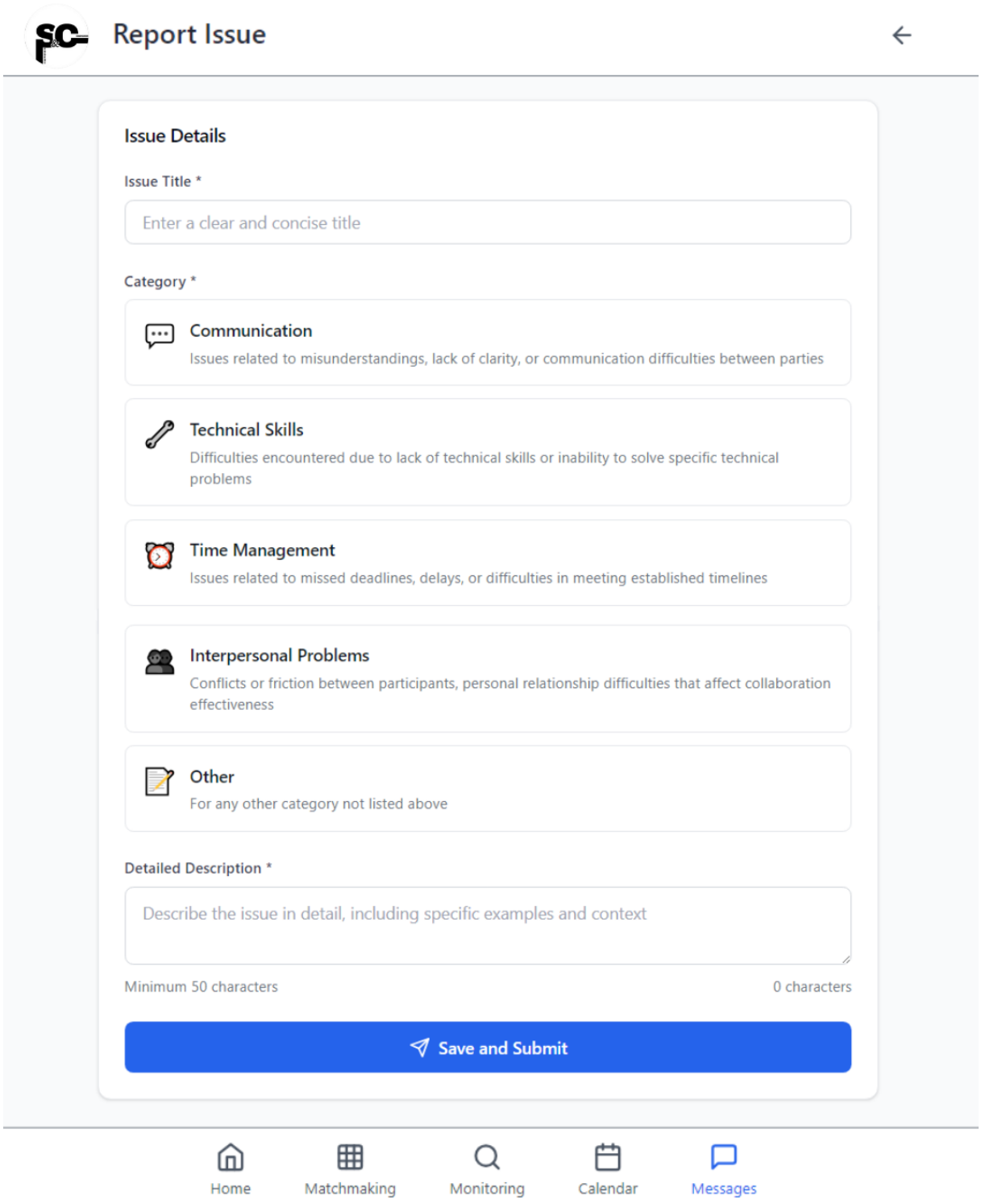

Figure 3.48: Messaging - Report Issue Interface for Students and Company Tutors.

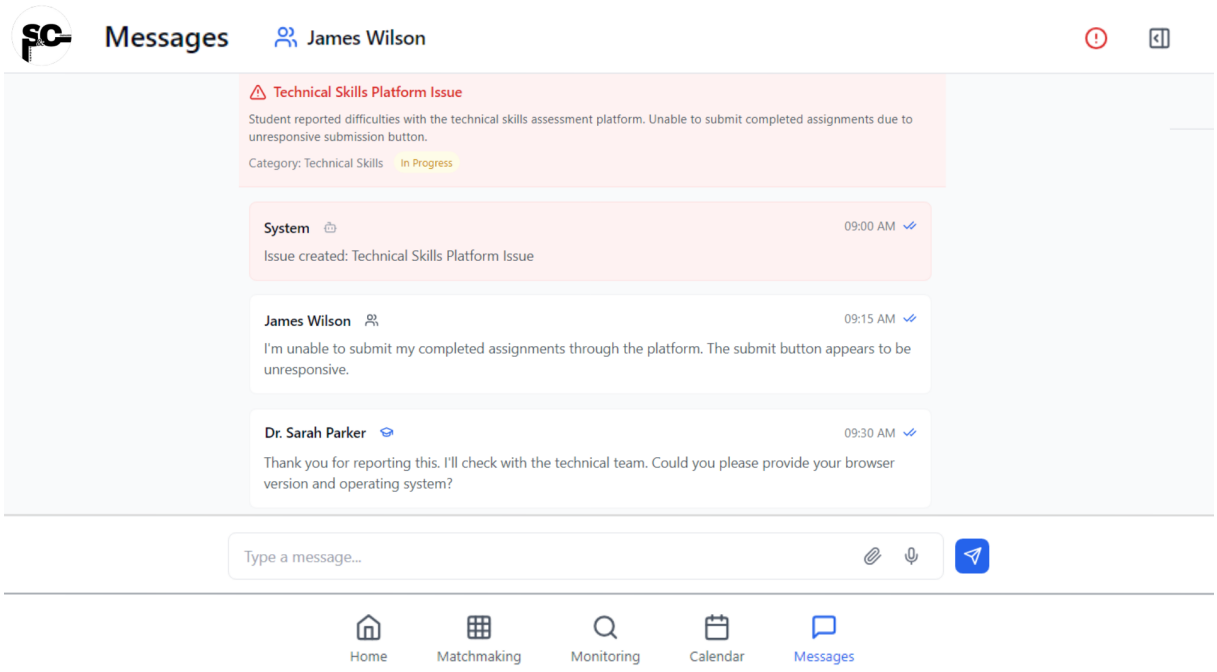

Figure 3.49: Messaging - Issue Chat Interface.

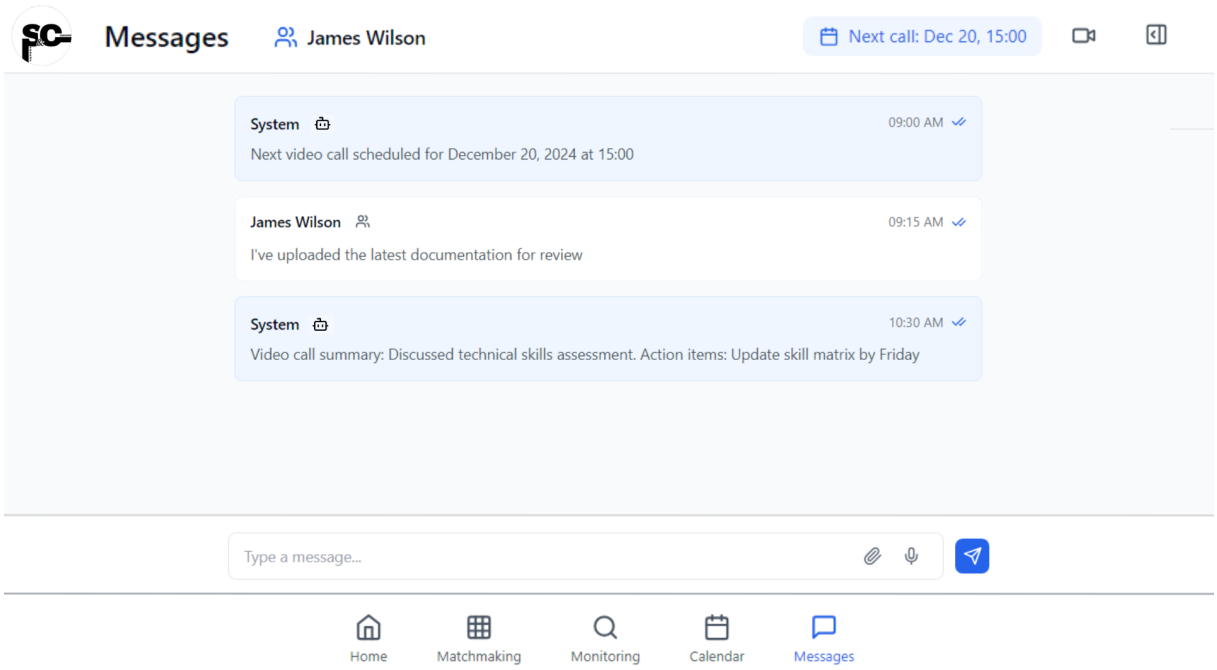

Figure 3.50: Messaging - Video-call Chat Interface.

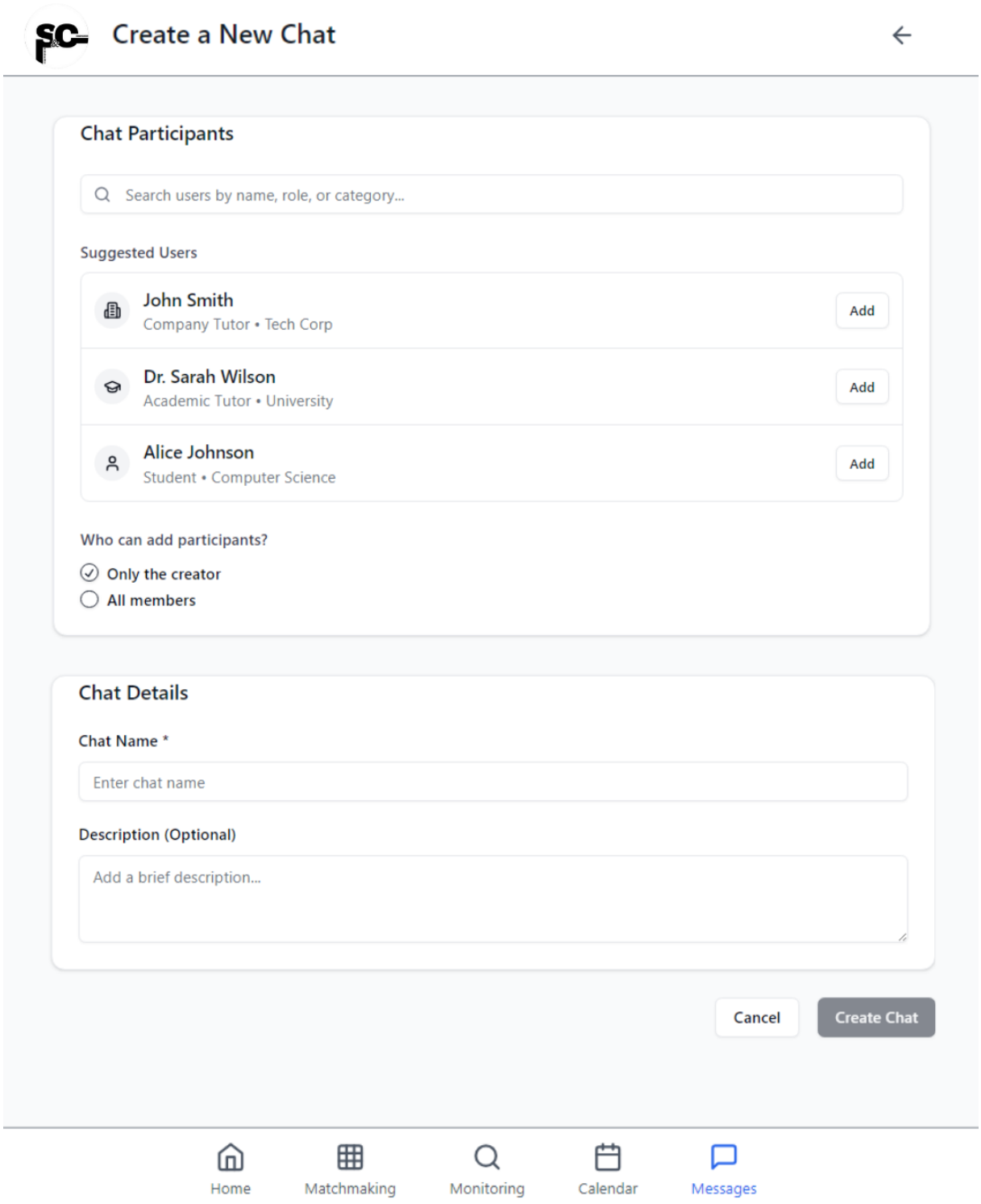

Figure 3.51: Messaging - Chat Creation Interface.

4 | Requirements Traceability

Sign-Up Requirements

FR1: The system must allow users to register by providing institutional email with registered domain, password, department and role.

- **Components:**
  - *Registration Manager: Form Handler*
  - *Registration Manager: Domain Checker*
  - *Mail Server*
  - *Model*

FR2: The system must provide a feature for users to upload their Curriculum Vitae (CV) in formats such as PDF or DOCX during the registration.

- **Components:**
  - *Registration Manager: CV Uploader*
  - *Model*

FR3: The system could extract key information from the uploaded CV to pre-fill the user's profile.

- **Components:**
  - *Registration Manager: CV Uploader*
  - *Registration Manager: Form Handler*

FR4: The system must allow users to edit and confirm the accuracy of the pre-filled information in their profile before final submission.

- **Components:**
  - *Registration Manager: Form Handler*
  - *Model*

FR5: The system must store uploaded CVs securely and ensure that all personal information is handled in compliance with applicable data protection regulations (GDPR).

- **Components:**
  - *Registration Manager: CV Uploader*
  - *Model*

FR6: The system must provide clear and helpful error messages if the CV upload fails due to an unsupported format or if the file size exceeds the allowed limit.

- **Components:**
  - *Registration Manager: CV Uploader*

FR7: The system should offer step-by-step guidance during the sign-up process to assist users in completing their registration and profile setup.

- **Components:**
  - *Registration Manager: Form Handler*

Log In Requirements

FR8: The system must allow users to log in by entering their registered email address and password.

- **Components:**
  - *Login Manager: Login Process*
  - *Model*

FR9: The system must authenticate the user's credentials against the stored data securely.

- **Components:**
  - *Login Manager: Login Process*
  - *Model*

FR10: The system must offer a password reset feature, where users can initiate a password reset through a link sent to their registered email address.

- **Components:**
  - *Login Manager: Password Recovery*
  - *Mail Server (via Mail API)*
  - *Model*

FR11: The system must require users to verify their identity via a security question or email verification during the password reset process.

- **Components:**
  - *Login Manager: Password Recovery*
  - *Model*
  - *Mail Server*

FR12: The system must log all login attempts, successful or failed and provide an administrative view for monitoring purposes.

- **Components:**
  - *Login Manager: Login Process*
  - *Model*

FR13: The system must redirect users to their respective dashboard based on their role (student, company tutor, university tutor) upon successful login.

- **Components:**
  - *Login Manager: Login Process*
  - *Dashboard Manager*
  - *Model*

FR14: The system must provide error messages that are clear and instructive if the login fails due to incorrect credentials or other issues.

- **Components:**
  - *Login Manager: Login Process*

Edit Profile Information Requirements

FR15: The system must allow all registered users to access and edit their profile information.

- **Components:**
  - *Dashboard Manager*
  - *Profile Manager: Modify Profile, Visualize Profile*
  - *Model*

FR16: The system must ensure that users can update critical profile fields (e.g., contacts, biography, department, role, certifications...).

- **Components:**
  - *Profile Manager: Modify Profile*
  - *Model*

FR17: The system must save changes made by users to their profile information immediately upon confirmation.

- **Components:**
  - *Profile Manager: Modify Profile*
  - *Model*

FR18: The system must require users to re-authenticate (e.g., confirm password) before allowing access to sensitive profile changes like email or password.

- **Components:**
  - *Login Manager: Login Process*
  - *Profile Manager: Modify Profile*
  - *Model*

FR19: The system should provide users with a preview of changes before final submission.

- **Components:**
  - *Profile Manager: Modify Profile*

Chatbot Communication and for Change Language Requirements

FR20: The system must offer a chatbot feature that is accessible from all main interfaces of the platform.

- **Components:**
  - *Dashboard Manager: Chatbot Interface*
  - *Chat Manager*

FR21: The system must ensure that the chatbot can handle frequently asked questions (FAQ) related to platform navigation, user account issues and general inquiries.

- **Components:**
  - *Dashboard Manager*
  - *Chat Manager: FAQ Handler*
  - *Model*

FR22: The system must allow the chatbot to guide users through common tasks such as resetting passwords, searching for internships and navigating to help sections.

- **Components:**
  - *Dashboard Manager*
  - *Chat Manager: Task Guidance*
  - *Login Manager: Password Recovery*
  - *Internship Manager*
  - *Model*

FR23: The system should enable the chatbot to escalate issues to a human agent when it cannot provide a sufficient solution or answer.

- **Components:**
  - *Dashboard Manager*
  - *Chat Manager: Escalation Handler*
  - *Support Handler (optional)*
  - *Mail Server / Notification Manager (optional)*
  - *Model*

FR24: The system must support multiple languages, allowing users to select their preferred language from a predefined list available within the platform settings.

- **Components:**
  - *Dashboard Manager*
  - *Settings Manager: Language Selector*
  - *Model*

FR25: The system must apply the selected language to all user interface elements, including menus, dialogues and help messages, without requiring a restart of the application.

- **Components:**
  - *Dashboard Manager*
  - *Model*

FR26: The system must remember the user's language preference for subsequent logins across different devices.

- **Components:**
  - *Dashboard Manager*
  - *Model*
  - *Login Manager*

Create Internships Requirements

FR27: The system must allow company tutors to create new internship listings via a dedicated interface within the platform.

- **Components:**
  - *Dashboard Manager: Dashboard Interface*
  - *Internship Manager: Create/edit Internship*
  - *Model*

FR28: The system must require that all necessary fields be completed before an internship can be published.

- **Components:**
  - *Internship Manager: Complete Checker*
  - *Internship Manager: Draft (optional)*
  - *Model*

FR29: The system must provide templates and guidelines to assist company tutors in creating detailed and effective internship descriptions.

- **Components:**
  - *Internship Manager: Create/edit Internship*
  - *Internship Manager: Improve Content*
  - *Model*

FR30: The system must validate the information entered by the user to ensure format and content standards before the internship is saved or published.

- **Components:**
  - *Internship Manager: Complete Checker*
  - *Dashboard Manager*
  - *Model*

FR31: The system must allow company tutors to preview the internship listing as it will appear to potential applicants before finalizing the publication.

- **Components:**
  - *Internship Manager: Create/edit Internship*
  - *Dashboard Manager*
  - *Model*

Save Internship Draft Requirements

FR32: The system must allow company tutors to save their internship listings as drafts before final publication.

- **Components:**
  - *Internship Manager: Draft*
  - *Internship Manager: Create/edit Internship*
  - *Model*

FR33: The system must ensure that drafts can be saved automatically at regular intervals.

- **Components:**
  - *Internship Manager: Draft*

FR34: The system should provide an option for company tutors to manually save a draft at any point.

- **Components:**
  - *Internship Manager: Draft*

FR35: The system must allow company tutors to access, review and continue editing their saved drafts.

- **Components:**
  - *Internship Manager: Draft*
  - *Internship Manager: Create/edit Internship*

FR36: The system must ensure that only authorized company personnel can access and edit saved drafts.

- **Components:**
  - *Login Manager / Profile Manager*
  - *Internship Manager: Draft*
  - *Model*

Internship Search Requirements

FR37: The system must provide a search interface for students to find internships using various filters.

- **Components:**
  - *Dashboard Manager*
  - *Internship Manager: Visualize Internship*
  - *Model*

FR38: The system must offer a recommendation engine that suggests internships based on the student's profile, past searches and other relevant criteria.

- **Components:**
  - *Matchmaking Manager: Recommend*
  - *Internship Manager: Visualize Internship*
  - *Profile Manager / Model*

FR39: The system should allow students to save their search criteria or specific internship listings for future reference.

- **Components:**
  - *Dashboard Manager*
  - *Internship Manager: Visualize Internship*
  - *Model*

FR40: The system must update the list of available internships in real-time as new opportunities are posted.

- **Components:**
  - *Internship Manager: Publish Internship*
  - *Internship Manager: Visualize Internship*
  - *Model*

Application for Internship Requirements

FR41: The system must allow students to apply for internships directly through the platform.

- **Components:**
  - *Dashboard Manager*
  - *Internship Manager: Visualize Internship*
  - *Matchmaking Manager: Invite*
  - *Model*

FR42: The system must confirm receipt of the application to the student via email or platform notification.

- **Components:**
  - *Notification Manager*
  - *Mail Server*
  - *Internship Manager*

FR43: The system should allow students to track the status of their applications within the platform.

- **Components:**
  - *Internship Manager*

  - *Dashboard Manager*
  - *Model*

Search for Students Requirements

FR44: The system must enable companies to search for potential candidates.

- **Components:**
  - *Dashboard Manager*
  - *Matchmaking Manager: Search*
  - *Profile Manager: Visualize Profile*
  - *Model*

FR45: The system should provide companies with tools to organize and manage the list of candidates based on search parameters.

- **Components:**
  - *Matchmaking Manager: Search*
  - *Dashboard Manager*

FR46: The system must allow companies to view detailed profiles of students, including application history and feedback from previous internships.

- **Components:**
  - *Matchmaking Manager: Search*
  - *Profile Manager: Visualize Profile*
  - *Model*

Accept/Reject Participation in the Selection Process (for Companies) Requirements

FR47: The system must allow companies to accept or reject applications from students and notify the students of the decision.

- **Components:**
  - *Selection Manager: Consult Invitation / Decisions*
  - *Notification Manager*
  - *Internship Manager*
  - *Model*

FR48: The system must enable companies to provide feedback or reasons for rejection (optional) when declining an application.

- **Components:**
  - *Selection Manager: Consult Invitation / Decisions*
  - *Notification Manager*
  - *Model*

Accept/Reject Participation in the Selection Process (for Students) Requirements

FR49: The system must allow students to accept or reject participation offers in the selection process from companies.

- **Components:**
  - *Selection Manager: Consult Invitation / Decisions*
  - *Dashboard Manager*
  - *Model*

FR50: The system should notify companies of the student's decision regarding their participation.

- **Components:**
  - *Selection Manager: Consult Invitation / Decisions*
  - *Notification Manager*

Communication Requirements

FR51: The system must provide a robust communication platform (messaging, forums, email notifications) among all users.

- **Components:**
  - *ActiveStage Manager: Communicate*
  - *Selection Manager: Communicate*
  - *Notification Manager*
  - *Model*

FR52: The system should ensure secure and private communication channels for all users.

- **Components:**
  - *ActiveStage Manager: Communicate*
  - *Selection Manager: Communicate*
  - *Notification Manager*
  - *Mail Server*
  - *Model*

FR53: The system must allow users to customize notification settings (frequency, type of notification) they receive.

- **Components:**
  - *Dashboard Manager*
  - *Notification Manager*
  - *Model*

Accept to be a Tutor Requirements

FR54: The system must allow university tutors to receive and respond to requests to become tutors for specific internships.

- **Components:**
  - *Selection Manager: Decisions*
  - *Notification Manager*
  - *Dashboard Manager*

FR55: The system must provide university tutors with internship details (responsibilities, duration, etc.) before accepting/rejecting.

- **Components:**
  - *Selection Manager: Decisions*
  - *Internship Manager: Visualize Internship*
  - *Model*

FR56: The system should notify the requesting party (company or university) of the tutor's decision.

- **Components:**
  - *Notification Manager*
  - *Selection Manager: Decisions*

Create and Manage Events Requirements

FR57: The system must allow company and university tutors to create and schedule events (interviews, deadlines...) related to internships.

- **Components:**
  - *Selection Manager: Meeting Planner*

- *ActiveStage Manager: Events Planner*
- *Calendar Manager*
- *Model*

FR58: The system must provide tools to manage event participants (invitations, tracking, reminders...).

- **Components:**
  - *Calendar Manager*
  - *Notification Manager*
  - *ActiveStage Manager / Selection Manager*

FR59: The system should integrate with common calendar applications (Google/Apple) to synchronize and notify.

- **Components:**
  - *Calendar Manager*
  - *External Calendar Service*

Compilation of the First Questionnaire Requirements

FR60: The system must enable company tutors to compile and customize a questionnaire for initial screening.

- **Components:**
  - *Selection Manager: Questionnaire*
  - *Model*

FR61: The system should automate the distribution of the questionnaire to candidates as soon as they apply or are pre-selected.

- **Components:**
  - *Selection Manager: Questionnaire*
  - *Notification Manager*
  - *Model*

FR62: The system must collect and organize questionnaire responses, allowing tutors to review them.

- **Components:**
  - *Selection Manager: Questionnaire*

  - *Model*
  - *Dashboard Manager*

Monitor Active Internships Requirements

FR63: The system must provide real-time monitoring for all active internships, allowing students, company tutors and university tutors to view status, deadlines and progress.

- **Components:**
  - *ActiveStage Manager*
  - *Model*
  - *Dashboard Manager*

FR64: The system must allow users to receive alerts and updates on every change or milestone in the internships they are involved in.

- **Components:**
  - *ActiveStage Manager*
  - *Notification Manager*
  - *Calendar Manager (optional)*

Report an Issue Requirements

FR65: The system must provide a feature for students and company tutors to report issues related to internships directly through the platform.

- **Components:**
  - *ActiveStage Manager: Report Complaint*
  - *Complaint Manager: Complaint Information*
  - *Notification Manager (optional)*

FR66: The system should allow users to categorize the type of issue (admin, ethical, logistical) and provide a detailed description.

- **Components:**
  - *ActiveStage Manager: Report Complaint*
  - *Complaint Manager: Complaint Information*
  - *Model*

Resolve Issues Requirements

FR67: The system must enable university tutors to access reported issues, review details and work on resolving them.

- **Components:**
  - *Complaint Manager: Manage Complaint*

FR68: The system should provide mechanisms for communication between the reporter (student/tutor) and resolver (university tutor) to discuss the issue.

- **Components:**
  - *Complaint Manager: Manage Complaint*
  - *Notification Manager / ActiveStage Manager: Communicate*

FR69: The system must log all actions in the issue resolution process and notify the reporter when the issue is resolved.

- **Components:**
  - *Complaint Manager: Manage Complaint*
  - *Notification Manager*
  - *Model*

Compilation of the Final Questionnaire Requirements

FR70: The system must allow for the compilation of a final questionnaire by students, company tutors and university tutors at the end of each internship.

- **Components:**
  - *ActiveStage Manager: Final Evaluation*

FR71: The system should automate the distribution of the final questionnaire and collect responses to evaluate the success of the internship.

- **Components:**
  - *ActiveStage Manager: Final Evaluation*
  - *Notification Manager*
  - *Model*

Real-Time Notifications Requirements

FR72: The system must provide real-time notifications for critical updates (e.g., internship status changes, new messages, documents, deadlines).

- **Components:**
  - *Notification Manager*

FR73: The system should allow users to customize the types of notifications and channels (email, SMS, platform alerts).

- **Components:**
  - *Notification Manager*
  - *Dashboard Manager*
  - *Mail Server (optional)*

Calendar Integration Requirements

FR74: The system must integrate with major calendar services (Google, Outlook) to facilitate scheduling and tracking of internship events.

- **Components:**
  - *Calendar Manager*
  - *External Calendar Service*

FR75: The system should ensure that all created events are automatically synchronized with the user's personal/professional calendar.

- **Components:**
  - *Calendar Manager*

FR76: The system must provide an option for users to view all internship-related events in an integrated calendar within the platform.

- **Components:**
  - *Calendar Manager*
  - *Dashboard Manager*

Traceability Table

Requirement	Components
FR1	Registration Manager: Form Handler, Domain Checker, Mail Server, Model
FR2	Registration Manager: CV Uploader, Model
FR3	Registration Manager: CV Uploader, Form Handler
FR4	Registration Manager: Form Handler, Model
FR5	Registration Manager: CV Uploader, Model
FR6	Registration Manager: CV Uploader
FR7	Registration Manager: Form Handler
FR8	Login Manager: Login Process, Model
FR9	Login Manager: Login Process, Model
FR10	Login Manager: Password Recovery, Mail Server, Model
FR11	Login Manager: Password Recovery, Model, Mail Server
FR12	Login Manager: Login Process, Model
FR13	Login Manager: Login Process, Dashboard Manager, Model
FR14	Login Manager: Login Process
FR15	Dashboard Manager, Profile Manager: Modify Profile, Visualize Profile, Model
FR16	Profile Manager: Modify Profile, Model
FR17	Profile Manager: Modify Profile, Model
FR18	Login Manager: Login Process, Profile Manager: Modify Profile, Model
FR19	Profile Manager: Modify Profile
FR20	Dashboard Manager: Chatbot Interface, Chat Manager
FR21	Dashboard Manager, Chat Manager: FAQ Handler, Model
FR22	Dashboard Manager, Chat Manager: Task Guidance, Login Manager: Password Recovery, Internship Manager, Model
FR23	Dashboard Manager, Chat Manager: Escalation Handler, Support Handler (optional), Mail Server / Notification Manager (optional), Model
FR24	Dashboard Manager, Settings Manager: Language Selector, Model
FR25	Dashboard Manager, Model
FR26	Dashboard Manager, Model, Login Manager
FR27	Dashboard Manager: Dashboard Interface, Internship Manager: Create/edit Internship, Model
FR28	Internship Manager: Complete Checker, Internship Manager: Draft (optional), Model
FR29	Internship Manager: Create/edit Internship, Internship Manager: Improve Content, Model

Table 4.1: Traceability Table Part 1

Requirement	Components
FR30	Internship Manager: Complete Checker, Dashboard Manager, Model
FR31	Internship Manager: Create/edit Internship, Dashboard Manager, Model
FR32	Internship Manager: Draft, Internship Manager: Create/edit Internship, Model
FR33	Internship Manager: Draft
FR34	Internship Manager: Draft
FR35	Internship Manager: Draft, Internship Manager: Create/edit Internship
FR36	Login Manager / Profile Manager, Internship Manager: Draft, Model
FR37	Dashboard Manager, Internship Manager: Visualize Internship, Model
FR38	Matchmaking Manager: Recommend, Internship Manager: Visualize Internship, Profile Manager / Model
FR39	Dashboard Manager, Internship Manager: Visualize Internship, Model
FR40	Internship Manager: Publish Internship, Internship Manager: Visualize Internship, Model
FR41	Dashboard Manager, Internship Manager: Visualize Internship, Matchmaking Manager:Invite, Model
FR42	Notification Manager, Mail Server, Internship Manager
FR43	Internship Manager, Dashboard Manager, Model
FR44	Dashboard Manager, Matchmaking Manager: Search, Profile Manager: Visualize Profile, Model
FR45	Matchmaking Manager: Search, Dashboard Manager
FR46	Matchmaking Manager: Search, Profile Manager: Visualize Profile, Model
FR47	Selection Manager: Consult Invitation / Decisions, Notification Manager, Internship Manager, Model
FR48	Selection Manager: Consult Invitation / Decisions, Notification Manager, Model
FR49	Selection Manager: Consult Invitation / Decisions, Dashboard Manager, Model
FR50	Selection Manager: Consult Invitation / Decisions, Notification Manager
FR51	ActiveStage Manager: Communicate, Selection Manager: Communicate, Notification Manager, Model
FR52	ActiveStage Manager: Communicate, Selection Manager: Communicate, Notification Manager, Mail Server, Model
FR53	Dashboard Manager, Notification Manager, Model

Table 4.2: Traceability Table Part 3

Requirement	Components
FR54	Selection Manager: Decisions, Notification Manager, Dashboard Manager
FR55	Selection Manager: Decisions, Internship Manager: Visualize Internship, Model
FR56	Notification Manager, Selection Manager: Decisions
FR57	Selection Manager: Meeting Planner, ActiveStage Manager: Events Planner, Calendar Manager, Model
FR58	Calendar Manager, Notification Manager, ActiveStage Manager / Selection Manager
FR59	Calendar Manager, External Calendar Service
FR60	Selection Manager: Questionnaire, Model
FR61	Selection Manager: Questionnaire, Notification Manager, Model
FR62	Selection Manager: Questionnaire, Model, Dashboard Manager
FR63	ActiveStage Manager, Model, Dashboard Manager
FR64	ActiveStage Manager, Notification Manager, Calendar Manager (optional)
FR65	ActiveStage Manager: Report Complaint, Complaint Manager: Complaint Information, Notification Manager (optional)
FR66	ActiveStage Manager: Report Complaint, Complaint Manager: Complaint Information, Model
FR67	Complaint Manager: Manage Complaint
FR68	Complaint Manager: Manage Complaint, Notification Manager / ActiveStage Manager: Communicate
FR69	Complaint Manager: Manage Complaint, Notification Manager, Model
FR70	ActiveStage Manager: Final Evaluation
FR71	ActiveStage Manager: Final Evaluation, Notification Manager, Model
FR72	Notification Manager
FR73	Notification Manager, Dashboard Manager, Mail Server (optional)
FR74	Calendar Manager, External Calendar Service
FR75	Calendar Manager
FR76	Calendar Manager, Dashboard Manager

Table 4.3: Traceability Table Part 4

5 | Implementation, Integration And Test Plan

5.1. Overview

This chapter outlines the structured approach to developing, integrating and testing the S&C system. It is divided into three main sections, each focusing on a critical aspect of the system's realization:

- **Implementation Plan (Chapter 5.2)**: This section details the strategies adopted for building the system components. A hybrid approach combining thread-based and bottom-up integration strategies is used. The thread-based strategy allows visible functionalities to be implemented and tested incrementally, offering immediate value to stakeholders. The bottom-up approach builds a stable application skeleton, progressively integrating more complex features. These strategies ensure modularity, parallelism, risk reduction and increased transparency.
- **Component Integration and Testing (Chapter 5.3)**: This section describes how the system's components are integrated and validated. Integration begins with the foundational DBMS and server infrastructure, progressing incrementally with the addition of features such as authentication, creation, visualization and matchmaking. Each feature is represented with a diagram, highlighting newly added components and their testing process. Drivers are used to verify each component individually before integration, ensuring stability and alignment with the system's workflows.
- **System Testing (Chapter 5.4)**: The final section focuses on validating the system as a whole. Comprehensive testing methodologies such as functional, performance, usability, load and stress testing are applied to ensure the system meets both functional and non-functional requirements. Continuous feedback loops, alpha testing and beta testing further refine the system, ensuring its robustness, usability and readiness for deployment.

This structured approach highlights the step-by-step progression from foundational system components to a fully integrated and validated system. By focusing on clear implementation strategies, comprehensive testing and iterative feedback, the chapter ensures that the system aligns with the specified requirements, addressing both functional and non-functional goals in a practical and methodical manner.

5.2. Implementation Plan

The purpose of this chapter is to outline the implementation strategies that will be adopted to develop the various components of the S&C system. First, the adopted integration philosophy is discussed, followed by the Feature Identification, which lists and plans the main functionalities (or "development threads") to be implemented.

5.2.1. Implementation Strategy

The implementation strategy used is the combination of two distinct strategies, leading to an hybrid approach:

Thread-based Strategy
We will adopt a "functional thread" (or "feature-based") approach where parts of functionalities are implemented and tested step by step. This allows immediate value to be shown to stakeholders, reducing the waiting time before the system becomes usable.

Bottom-Up with Incremental Integration
Starting from a "skeleton" of the application (where managers and main components are defined at a basic level), we will gradually add more complex features. This enables initial behavior simulation, such that foundational "building blocks" are tested and considered stable, before they are replaced or extended by higher-level components.

The advantages of this hybrid strategy can be summed up as:

- **Parallelism**: Multiple teams can work on different features (e.g., one on "Monitoring Selection," another on "Chat/Complaint"), integrating their results into a common workflow.
- **Risk Reduction**: If a thread proves more complex or encounters delays, other functionalities can continue advancing.
- **Increased Transparency**: At each partial release, stakeholders and users can verify the state of the system and provide early feedback.

5.2.2. Features Identification

The features to be implemented are identified based on the requirements and use cases outlined in the RASD and DD. Each feature corresponds to a well-defined set of domain functionalities, with concise title summarizing its primary function, followed by an overview of its key operations. We then explain the rationale for its development order, highlighting dependencies on preceding features.

[F1] Authentication Features This feature establishes the system's entry point, common to all the three type of users. Key functionalities include user registration, with CV uploading and domain verification and login management, which handles credential verification and password recovery. Developed as first, since it sets the groundwork for all subsequent features, as no other system module can function without a user profile.

[F2] Compose Features This feature enables creating and editing profiles (users/institutions), internships and complaints, with the option to save drafts exclusively for internships. The features can be developed in parallel across different teams: one working on profiles, another on internships and another on complaints, assuming the database structure and authentication (F2) are already implemented in the system.

[F3] Visualize Features This feature enables users to visualize profiles, internships and complaints, which come from the previous features F1 and F2. Moreover it allows to visualize the tables with these entities in the Selection Process, Active Stage and Issue Management. Parallel development is possible, with teams independently handling these different parts, progressively integrating their work, as done in previous features.

[F4] Matchmaking Features This feature includes the two main actions of Matchmaking, that are searching and recommending internships or candidates, respectively to company tutors and students. Dependencies include F2 and F3, as it requires entities to be created and they involved the visualization of these entities.

[F5] Calendar Features This feature supports planning and managing events such as selection process and issues management meetings, but also active stage events. It depends on F3, since the event creation can be performed from the tables in this feature, which they interface with the Calendar Manager. Moreover the events creation affect and update the current selection process or active stage state.

[F6] Messaging Features This feature facilitates communication through messages, video calls and chatbot assistance. It integrates with various modules like Selection Manager for student-company interactions, ActiveStages Manager for collaboration among all the three users, Complaint Manager for "PROBLEM" chats and Calendar Manager to create the chat when a video-call is setup. Development of this feature occur first independently and later with the integrated into its dependent contexts defined before.

[F7] Questionnaires Features This feature regards questionnaires composed at the end of the selection meetings, by the company tutor and of active stages, by all the three users. As for the messaging feature, it can be developed independently by first creating the Questionnaire Manager and then connecting it to selection and active stage processes and complaint management workflow, from F3.

[F8] Finalization Features This feature manages final steps such as inviting candidates to the selection process, making final decisions in it, deleting internships, profiles, issues reports, or selection processes and closing active stages. Therefore dependencies include F2, as some of these processes are in the editing/draft components, F3 and F4.

[F9] Notification Features This feature generates, sends and manages notifications for system events like new matching, complaints and questionnaires. This is the last possible set of features to be developed, since the proper functioning of this feature requires that every other kind of feature properly functions too.

5.3. Component Integration and Testing

In this section, we provide a detailed overview of the integration and testing process for the system's components, outlining their implementation at each stage of development. The integration process, as said before, adopts a bottom-up approach, starting with the foundational elements of the system and progressively building toward the full functionality of all features.

The integration begins with the DBMS and host server, which form the core infrastructure. Once these are operational, the system's components are integrated step-by-step. Connections to external systems are not required during the initial stages but will be integrated as the corresponding features are developed. This approach ensures that core functions are tested and validated before introducing dependencies on external services.

Diagram 0: Initial Integration This diagram shows the initial phase of integration, where the Model serves as the foundation of the system. Drivers are used to test and validate the Model in isolation, ensuring its proper functionality. This phase lays the groundwork for the integration of subsequent components.

Diagram 1: Authentication Features This diagram introduces the Authentication Features, including the Login Manager and Registration Manager. These modules provide the essential entry point for all users, enabling functionalities like credential verification, password recovery and domain-specific registration. Both components are tested with drivers, ensuring their reliability before moving to higher-level integrations.

Diagram 2: Compose Features This diagram adds the components for creating, editing of profiles, internships and complaints. Specific additions include the Modify Profile, Draft, Create/edit Internship and Complaint Information. Each of these components is tested using dedicated drivers to ensure they work independently and handle data consistently, before being combined with visualization and matchmaking processes.

Diagram 3: Visualize Features In this diagram, components like the Visualize Profile, Visualize Internship, Manage Complaint, Evaluate Invitation and Consult Active Stage are introduced. These components allow users to view the data created in the Compose Features, in different ways. Their testing ensures that data retrieval and presentation function correctly and are ready for integration into more complex workflows.

Diagram 4: Matchmaking Features The Matchmaking Features introduce components like the Search and Recommend, which facilitate finding internships or candidates. Testing ensures these components function efficiently and provide accurate results, focusing on the correctness of their independent logic.

Diagram 5: Calendar Features This diagram introduces components such as the Plan Event, Meeting Planner, Events Planner and Calendar Manager. These components enable users to view, plan and manage events. Testing ensures they correctly interact with other modules and handle event updates reliably.

Diagram 6: Messaging Features The Messaging Features add components like the three Communicate and the Chat Manager. These components are tested to validate independent communication functionalities before integrating them with scheduling, complaint and other workflows.

Diagram 7: Questionnaires Features This diagram adds the Questionnaire Manager, the Questionnaire and the Final Evaluation to handle forms filled out during selection meetings or active stages. Testing ensures these components collect and store data related to questionnaires accurately, verifying their standalone functionality before linking to other processes.

Diagram 8: Finalization Features Components such as Delete Profile, Delete Internship, Delete Complaint/Active Stage, Decisions, End Stage, Report Complaint and Invite are added in this phase. These manage end-stage processes such as inviting candidates, finalizing decisions and deleting records. Testing focuses on ensuring these actions are executed reliably and consistently.

Diagram 9: Notification Features This diagram introduces the Notification Manager, responsible for generating and managing alerts for events like new matches or complaints. Testing validates the delivery and accuracy of notifications. As can be noticed in the diagram, for simplicity the previous components are grouped in their respective Manager component.

Diagram 10: Dashboard Manager The last component to integrate and test is the Dashboard Manager. This final integration ensures seamless navigation and interaction for users. Testing focuses on verifying that all system components function cohesively within the dashboard, marking the system's readiness for usage.

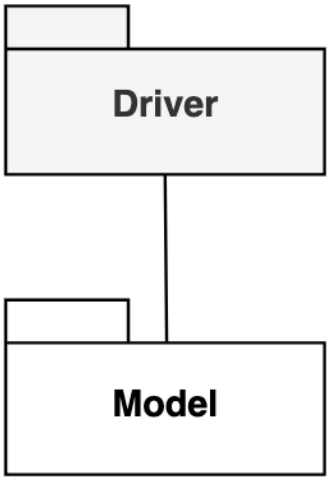

Figure 5.1: Diagram 0 - Initial Integration

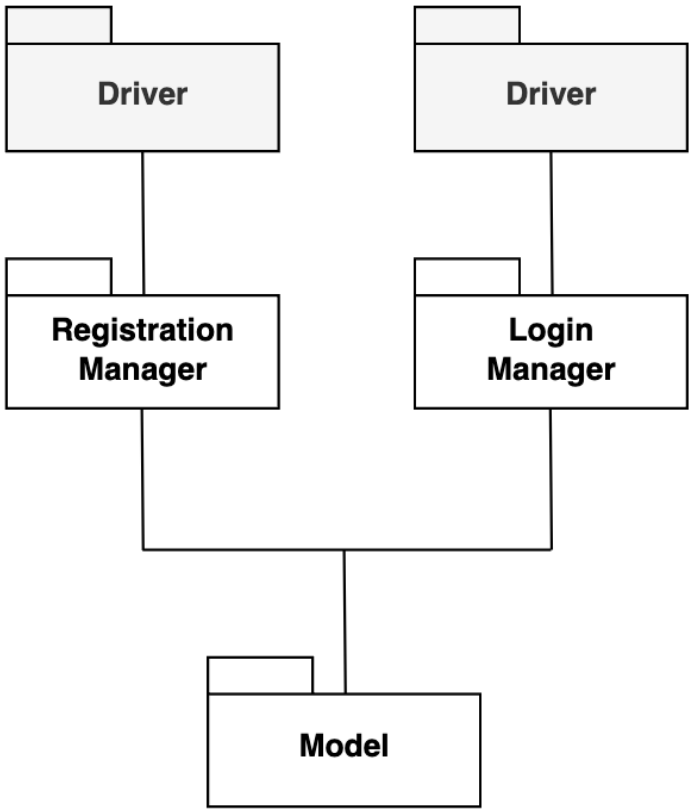

Figure 5.2: Diagram 1 - [F1] Authentication Features

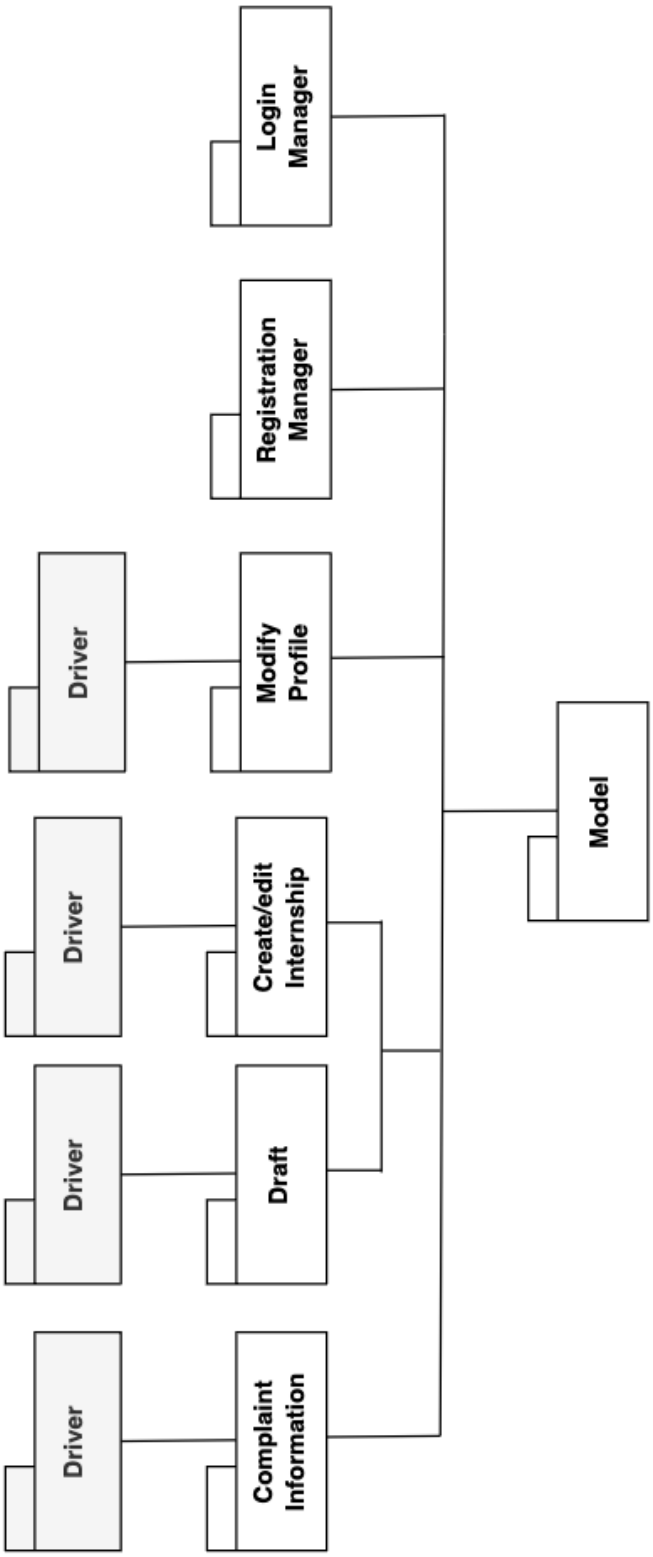

Figure 5.3: Diagram 2 - [F2] Compose Features

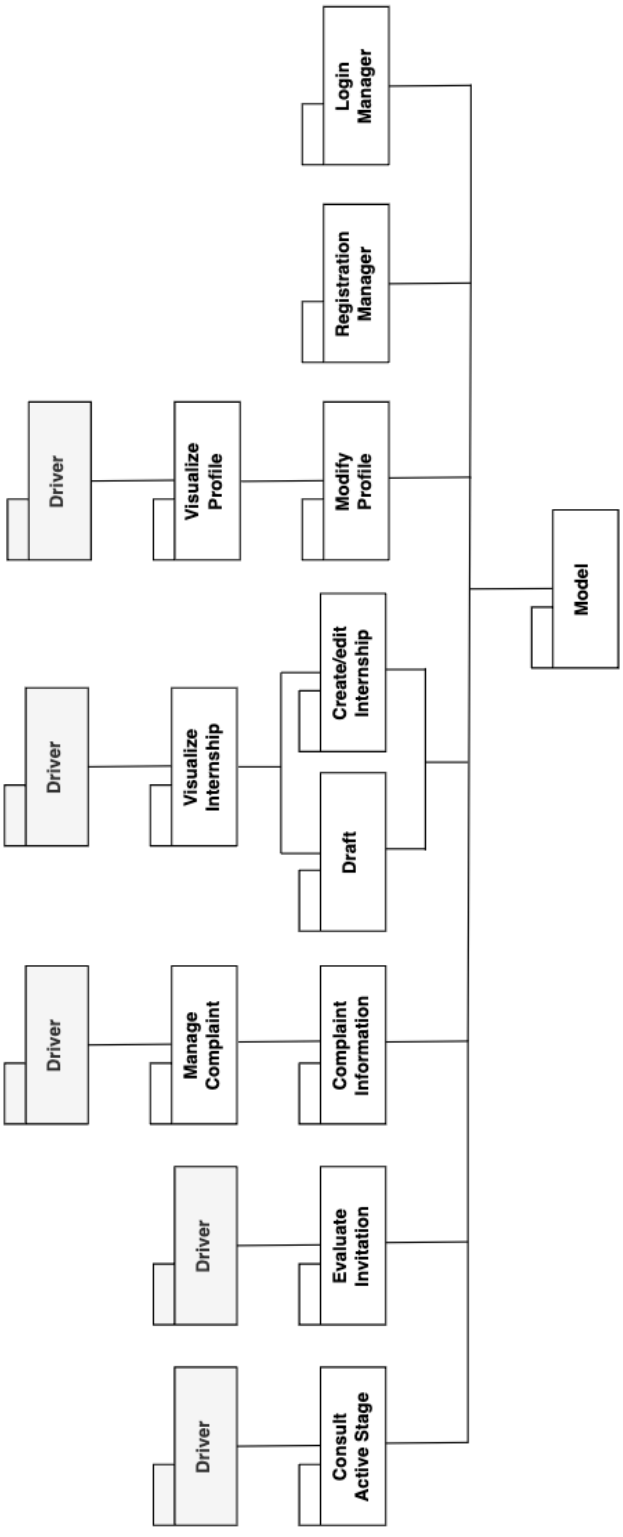

Figure 5.4: Diagram 3 - [F3] Visualize Features

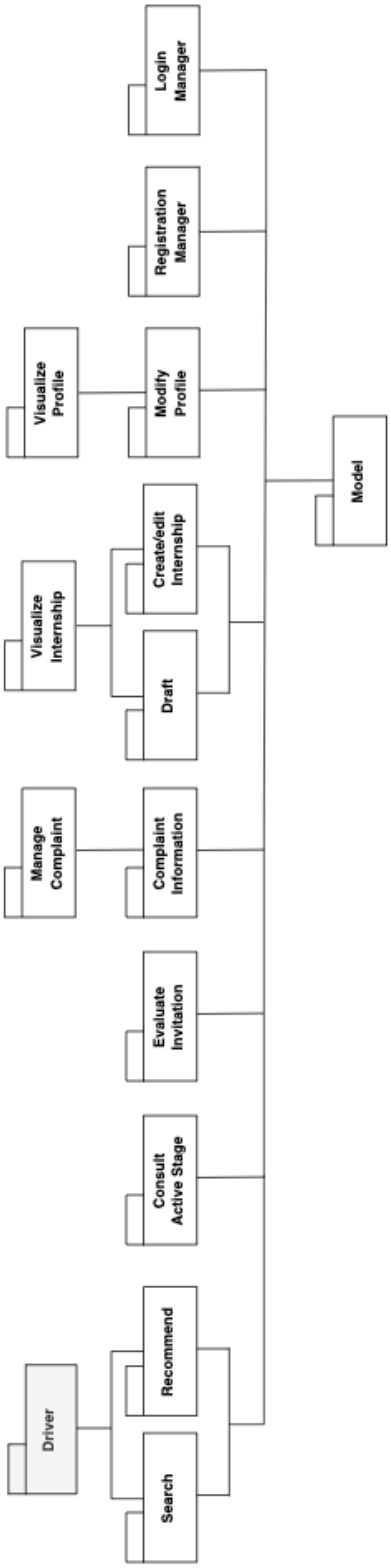

Figure 5.5: Diagram 4 - [F4] Matchmaking Features

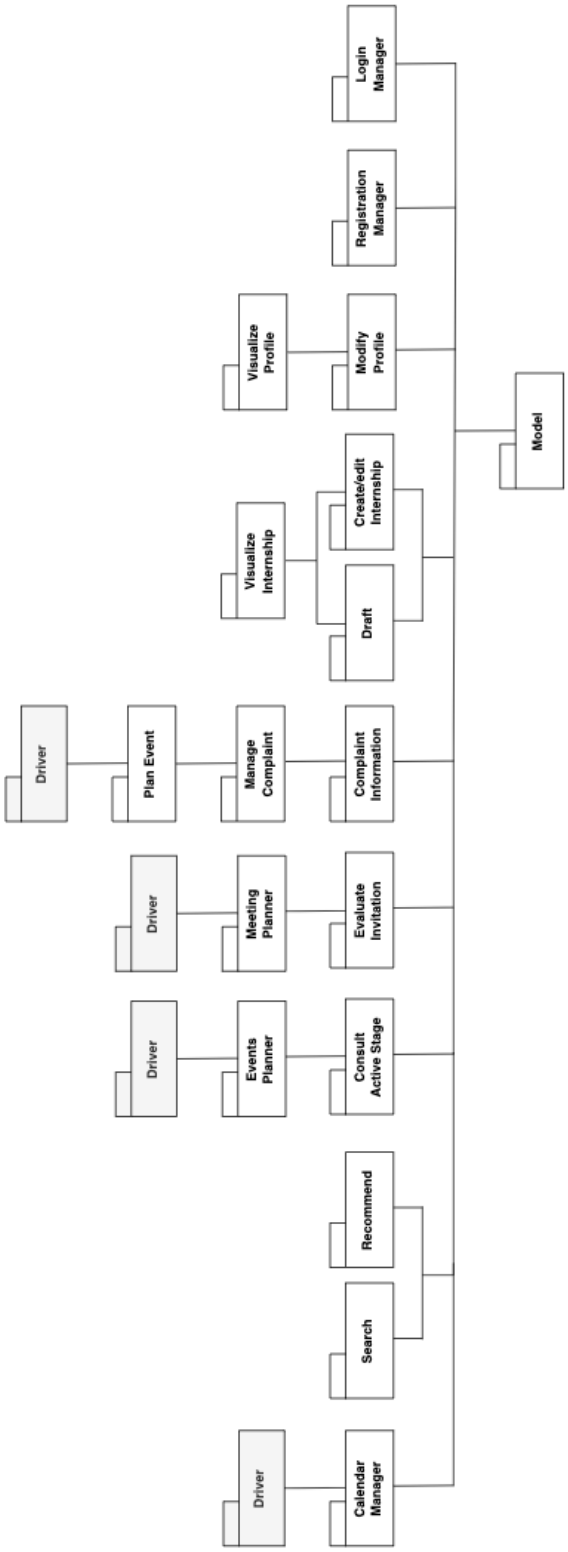

Figure 5.6: Diagram 5 - [F5] Calendar Features

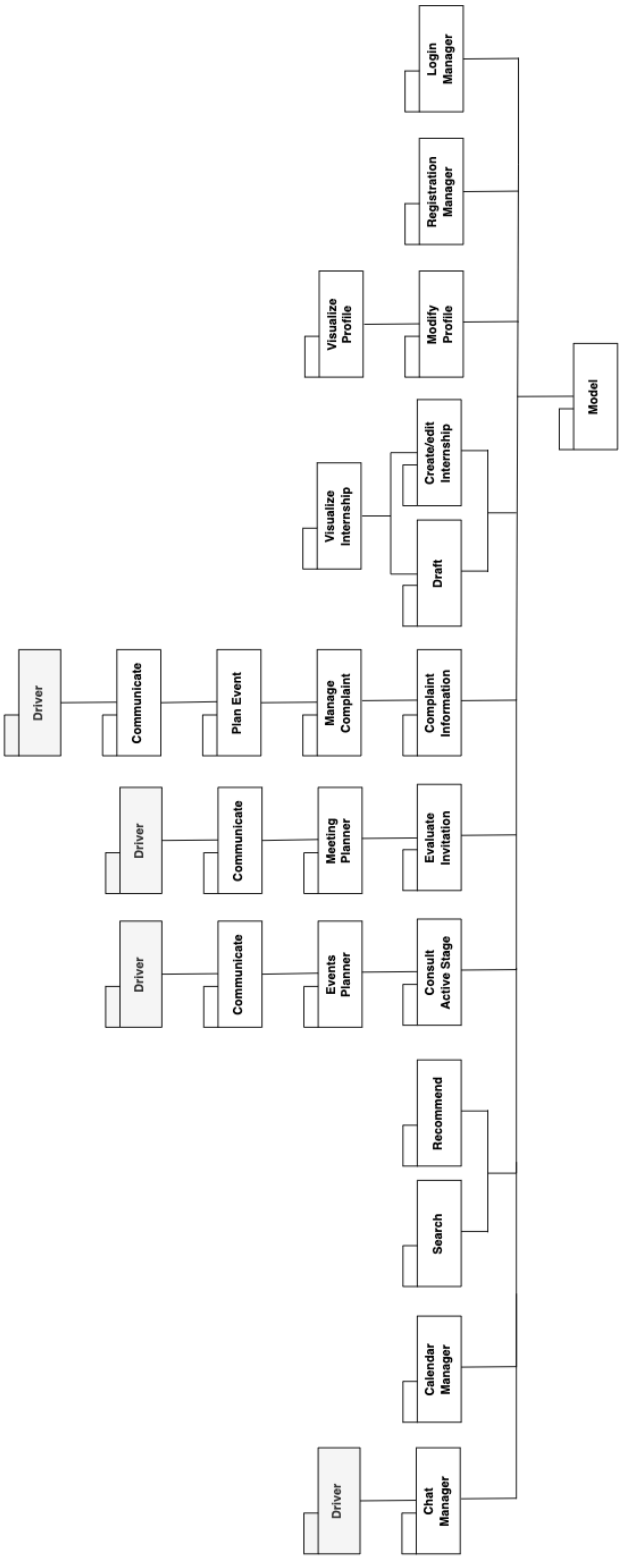

Figure 5.7: Diagram 6 - [F6] Messaging Features

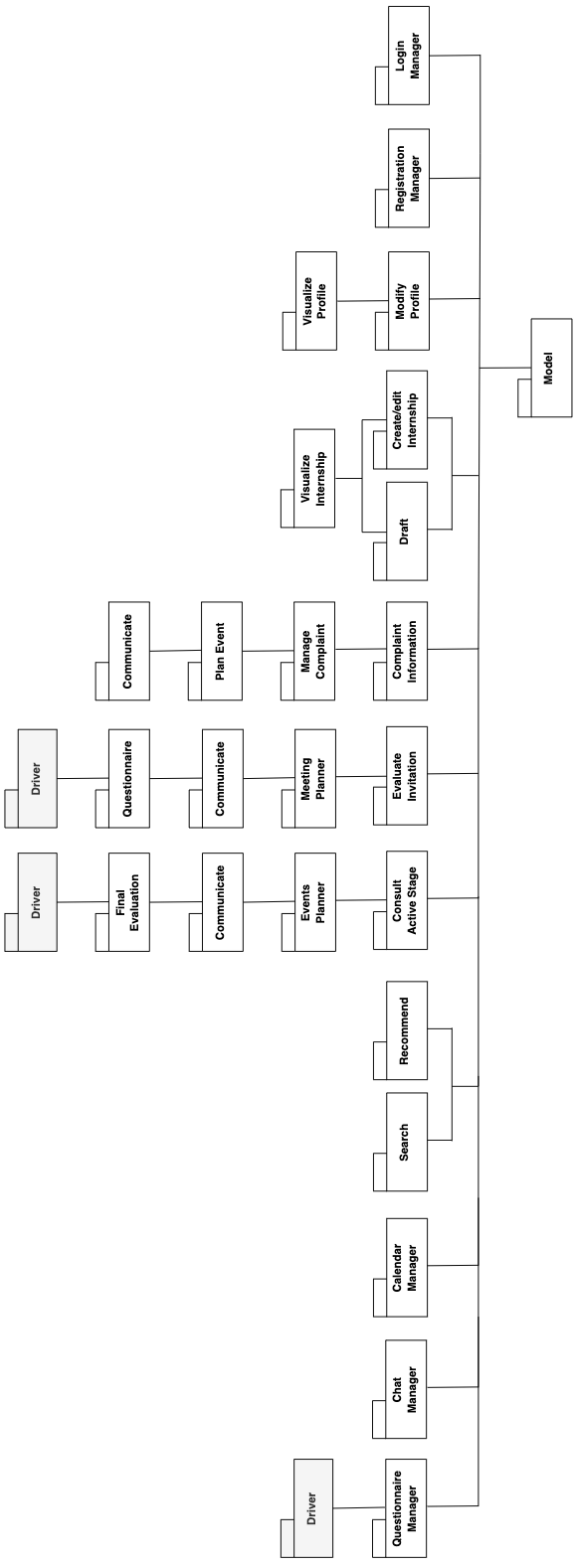

Figure 5.8: Diagram 7 - [F7] Questionnaires Features

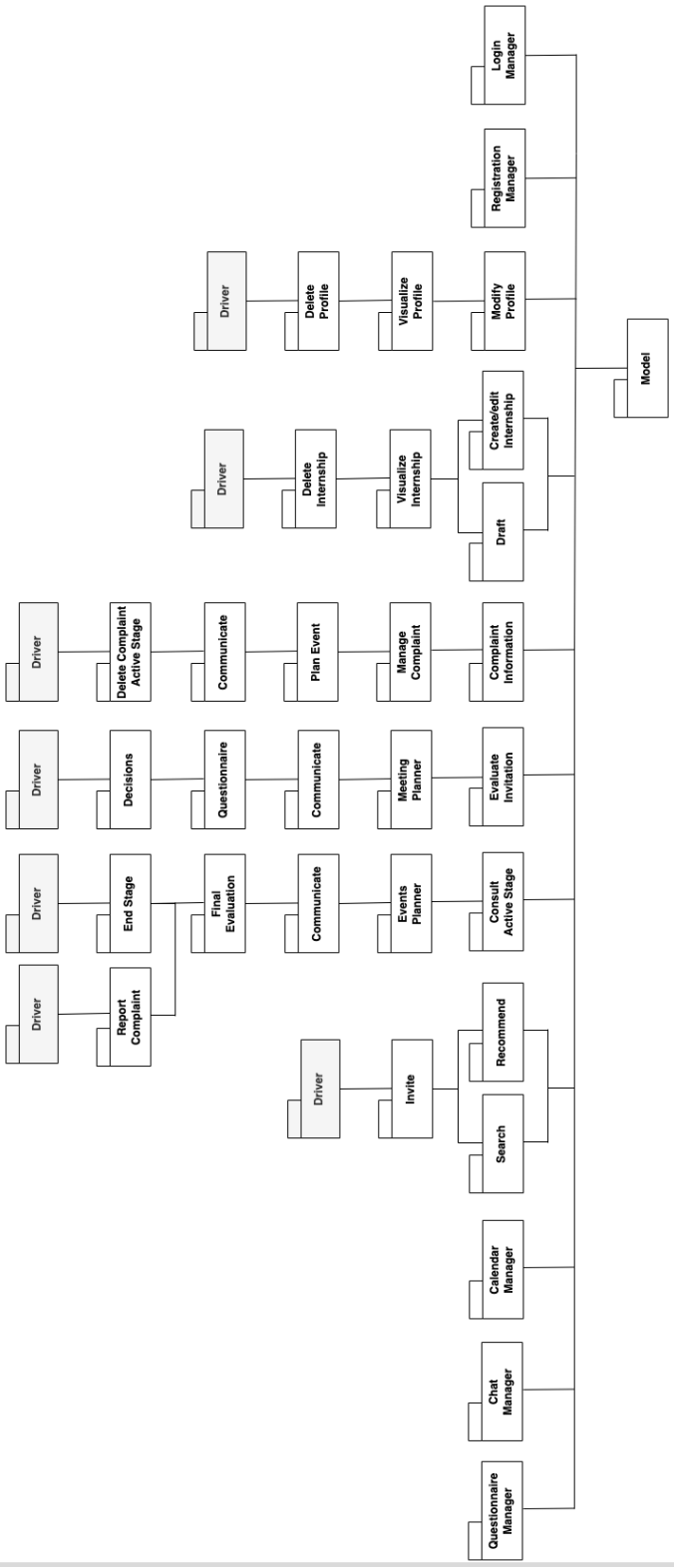

Figure 5.9: Diagram 8 - [F8] Finalization Features

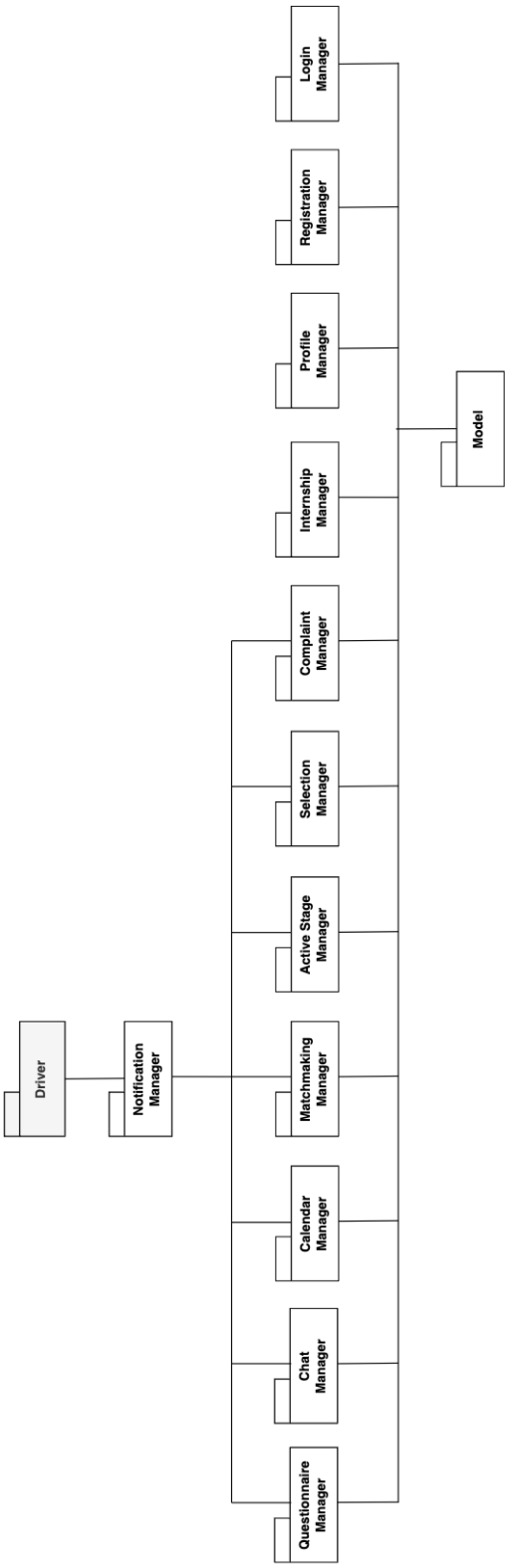

Figure 5.10: Diagram 9 - [F9] Notification Features

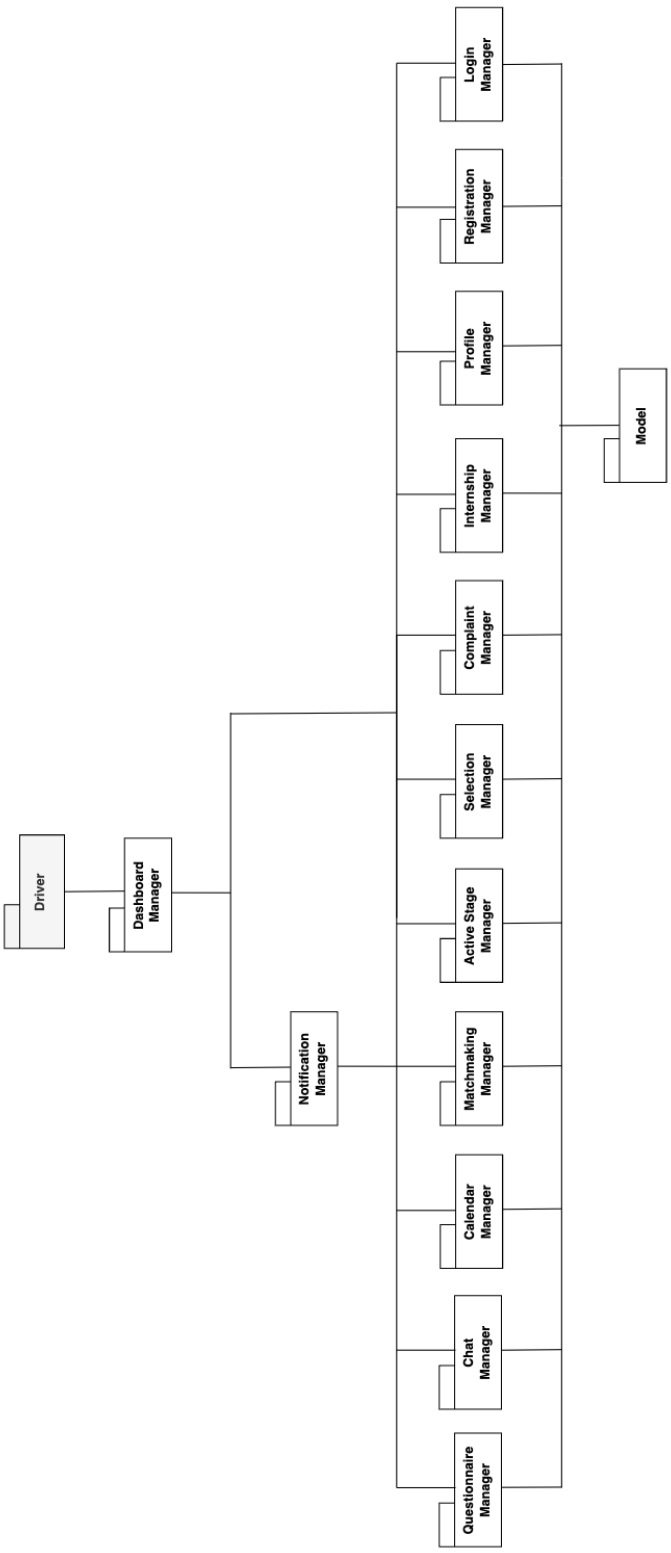

Figure 5.11: Diagram 10 - Dashboard Manager

5.4. System Testing

System testing represents the culmination of the integration process, aiming to ensure that the entire system operates as intended and meets the specifications defined in the RASD document, therefore validating the correctness of the software architecture.

During the development phase, each component is thoroughly tested to confirm its proper functionality. However, not all components can be tested in isolation from the entire system architecture. In such cases, drivers are used to simulate the higher-level modules not yet implemented, invoked by the component under test, while stubs are employed to replicate lower-level modules with which the component interacts. This allows for isolated testing of the component's functionality within a controlled environment. Once a single thread unit has been validated, it can be integrated into the broader software architecture. After integration, a new driver is used to verify that the newly integrated component works correctly with the existing system and that the overall workflow remains intact, ensuring no disruptions to previously tested functionalities. This process is repeated for each component that has to be developed. Finally, after the integration of all components, comprehensive system testing ensures the absence of bugs and verifies that the complete system adheres to the specified workflows.

5.4.1. Testing Methodologies

To achieve system testing, various testing methodologies are applied to ensure that every component, with its goals, requirements and outputs are achieved and integrated as intended in the workflow, while evaluating the performance, usability and robustness.

Functional Testing Functional testing ensures that the system follows the workflows and functionalities described in the RASD document. Each use case is simulated to confirm that all goals, requirements and expected outputs are achieved. By simulating real-world conditions, this testing validates the seamless operation of all integrated components.

Performance Testing Performance testing evaluates the system's resilience under heavy workloads, identifying bottlenecks and inefficiencies. This involves simulating concurrent user activity to measure response times, throughput and resource utilization, ensuring that the system can handle simultaneous user operations efficiently. By detecting hardware or software limitations, performance testing provides opportunities for algorithm optimization and ensures that the system meets the scalability requirements.

Usability and User Interface Testing This testing method examines how effectively users interact with the system's interface across various devices and browsers, addressing accessibility requirements. Real users are observed completing tasks, offering insights into the system's accessibility, intuitiveness and overall user experience. Key focus areas include verifying ease of navigation and consistency in user interactions for all actors defined in the system (students, company and academic tutors).

Load Testing Load testing determines the system's maximum operational capacity by exposing it to increasing workloads and demand, providing insights into its limits. This process helps identify critical issues such as memory leaks, buffer overflows and poor memory management. Additionally, it validates the system's stability under prolonged high loads, ensuring sustained performance over time.

Stress Testing Stress testing examines the system's resilience, robustness and recovery from extreme conditions and adverse situations, such as resource depletion or an overload of simultaneous users, without compromising core functionalities.

5.4.2. Continuous Feedback

Continuous feedback from stakeholders is an integral part of the testing process, providing valuable insights throughout system development, refining system's functioning and usability, and enabling quick resolutions and user satisfaction.

Alpha Testing Alpha testing is conducted in a controlled environment with a select group of users and psychologists, who propose questions to gather insights into how these users interact with the system. So this phase is crucial for identifying malfunctions and assessing usability, to then address issues and enhance user satisfaction, before exposing the system to broader usage.

Beta Testing This phase uncovers any issue that may arise under real-world conditions, providing critical data for debugging and refinement. Indeed, feedback from beta testers, combined with system logs, informs the final adjustments for the system's release.

5.4.3. Final System Validation

Once all components have been integrated and tested individually, the system undergoes a comprehensive validation process. This step ensures that workflows function correctly, all requirements are met and no critical bugs remain. The final validation prepares the system for deployment, with continuous monitoring planned to address potential issues post-launch, identified from logs. This ensures a smooth transition to full operational use while maintaining long-term reliability and performance.

6 | Effort Spent

In this section you will include information about the number of hours each group member has worked for this document.

Acquadro Patrizio

chapter	Effort (In hours)
1	0
2	0
3	0
4	0
5	0

Colosio Giacomo

chapter	Effort (In hours)
1	0
2	0
3	0
4	0
5	0

Drugman Tito Nicola

chapter	Effort (In hours)
1	0
2	0
3	0
4	0
5	0

Bibliography

[1] Three-tier architecture in web development, 2024. URL https://medium.com/@aasalex21c/three-tier-architecture-in-web-development-b2bf2e169ed0.

[2] Client–server model, 2025. URL https://en.wikipedia.org/wiki/Client%E2%80%93server_model.

[3] Model–view–controller, 2025. URL https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[4] U. o. W.-M. Center for research on College-Workforce Transitions (CCWT). National survey of college internships (nsci) 2021 report, 2021. URL https://ccwt.wisc.edu/wp-content/uploads/2022/04/CCWT_NSCI-2021-Report.pdf.

[5] T. T. Jane Hamilton. Tougher than ever to secure place on sought after internships, 2024. URL https://www.thetimes.com/article/tougher-than-ever-to-secure-place-on-sought-after-internships-8rrfj30rm.

List of Figures

List of Tables