DD Ricci, Paoli, Grisoni

POLITECNICO
MILANO 1863

DD
Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Lorenzo Ricci, Matteo Giovanni Paoli, Samuele Grisoni
Version:	1.0
Date:	7-01-2025
Download page:	https://github.com/Slaitroc/RicciPaoliGrisoni/
Copyright:	Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

0 Contents

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

0 List of Figures

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

0 List of Tables

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

1 Introduction

1.1 Purpose

The purpose of the Student&Company (S&C) platform is to enable students to enroll into internships that will enhance their education and strengthen their CVs, while letting companies publish internship offers and select the best candidates through interviews. More over, S&C allow students' universities to monitor each of their students' progress and intervene if needed. The platform support and aid the users throughout the entire process by provide suggestion to the uploaded CVs and internship offers, automatically matches students and companies thanks to a proprietary algorithm, manage the distribution and collection of interviews and provides a space for filing and resolving complaints. The reader can find more information about the platform in the RASD document. In the remaining part of this chapter we will present a summary of the technical choices made for the creation of the platform and different bullet point lists and table including the Goals that we are trying to accomplish with this software and the Definition, Acronyms, Abbreviations used in this document.

1.1.1 Goals

[G1] Companies would like to advertise the internships they offer.

[G2] Students would like to autonomously candidate for available internships.

[G3] Students would like to be matched with internships they might be interested in.

[G4] Companies would like to perform interviews with suitable students.

[G5] Students and companies would like to complain, communicate problems, and provide information about an ongoing internship.

[G6] Students and companies would like to be provided with suggestions about how to improve their submission

[G7] Universities would like to handle complaints about ongoing internships.

[G8] Students would like to choose which internship to attend from among those for which they passed the interview.

[G9] Companies would like to select students for the internship position among those who passed the interview.

1.2 Scope

This document, Design Document (DD), will provide a detailed description of the architecture of the S&C platform from a more technical point of view. In particular, it will provide a thorough description of the software with a special emphasis on its interfaces, system module, and architectural framework. This document will also discuss the implementation, integration and testing plan describing the tools and methodologies that will be used during the development of the platform.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

1.2.1 Main Architectural Choices

The chosen architectural style is a *microservices architecture*, as it enables a scalable and modular approach to development. The three main services are Presentation, Application, and Authenticator, which are responsible for the user interface, business logic, and authentication, respectively. Data-intensive service manage their data through autonomous databases, ensuring modularity and scalability. For now, the databases are not designed to scale horizontally, meaning that when services using them are duplicated, the database remains a single, centralized instance. The Presentation layer provides the client with a Single Page Application (SPA) for a smoother user experience. The Application layer contains modules that handle platform-specific logic services, which could be exported as independent services in the future. This setup facilitates reliability and fault tolerance, as each service is designed to be container-based and can be scaled vertically or horizontally using cloud orchestration tools during deployment.

1.3 Definitions, Acronyms, Abbreviations

This section provides definitions and explanations of the terms, acronyms, and abbreviations used throughout the document, making it easier for readers to understand and reference them.

1.3.1 Definition

The definition shared between this document and the RASD document are reported in the following list:

- **University**: A university that is registered on the S&C platform.
- **Company**: A company that is registered on the S&C platform.
- **Student**: A person who is currently enrolled in a University and is registered on the S&C platform.
- **User**: Any registered entity on the S&C platform.
- **Internship Offer**: The offer of an opportunity to enroll in an internship provided by a Company. The offer remains active on the platform indefinitely until the publishing Company removes it
- **Participant**:A Participant is an entity that interacts with the platform for the purpose of find or offering an Internship Position Offer, like Students and Companies
- **Recommendation Process**: The process of matching a Student with an Internship offered by a Company based on the Student's CV and the Internship's requirements made by the S&C platform.
- **Recommendation/Match**: The result of the Recommendation Process. It is the match between a Student and an Internship.
- **Spontaneous Application**: The process of a Student spontaneously applying for an Internship that was not matched through the Recommendation Process.
- **Interview**: The process of evaluating a Student's application for an Internship done by a Company through the S&C platform.
- **Feedback**: Information provided by Participant to the S&C platform to improve the Recommendation Process.
- **Internship Position Offer**: The formal offer of an internship position presented to a student who has successfully passed the Interview, who can decide to accept or reject it.
- **Suggestion**: Information provided by the S&C platform to Participant to improve their CVs and Internship descriptions.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- **Confirmed Internship**: An Internship that has been accepted by the Student and the offering Company.
- **Ongoing Internship**: A internship that is currently in progress. All Ongoing Internships are Confirmed Internships, but the vice versa is not always true.
- **Complaint**: A report of a problem or issue that a Student or Company has with an Ongoing Internship. It can be published on the platform and handled by the University.
- **Confirmed Match**: A match that has been accepted by both a Student and a Company.
- **Rejected Match**: A match that has been refused by either a Student or a Company.
- **Pending Match**: A match that has been accepted only by a Student or a Company, waiting for a response from the other party.
- **Unaccepted Match**: A match that has been refused by either a Student or a Company.

The definition specific to this document are reported in the following list:

- **Front-End**:The part of the software that is responsible for the presentation of the data and the interaction with the user. It is what the user sees and interacts with.
- **Back-End**: The part of the software that is responsible for the business logic of the platform and the storage and retrieval of data. It is composed of the servers and the database. It is what the user does not see.
- **RESTful API**: A set of rules that software engineers follow when creating an API that allows different software to communicate with each other.
- **3-tier architecture**: A software architecture that divides the software into three different layers: presentation layer that contains the logic for displaying data and retrieve input from the user, application layer where the main logic of the software is present, and data layer that contains the data and the logic to access it.
- **Proxy**: A server that acts as an intermediary for requests from clients seeking resources from other servers. It can redirect request based on different criteria.
- **Presentation Service**: The service that provides the user interface and experience to the client. It is responsible for delivering static content to the client upon connection to the platform's main domain.
- **Presentation Layer**: The layer of the software that is responsible for the visualization of the data and the retrieval of user inputs, offered by the Presentation Service.
- **Application Service**: The service that contains the platform's core functionalities, including platform logic, database interaction, and notification handling. It exposes various RESTful API endpoints for the different services it provides.
- **Application Layer**: The layer of the software that is responsible for the processing of the data, computation, and the logic of the platform, offered by the Application Service.
- **Authenticator Service**: The service that is responsible for every process concerning authentication and session validation.
- **Data Layer**: The layer of the software that is responsible for the storage and retrieval of the data.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- **Service**: A self-contained unit of functionality that can be independently deployed and scaled.
- **Container**: A container is a lightweight, standalone, and portable unit of software that isolates development environments, allowing developers to build, test, and deploy applications more efficiently without conflicts between different enviorments.
- **Notification Subsystem**: The system that is responsible for sending notifications to users when relevant events occur.
- **Middleware**: A software that acts as a bridge different applications, especially if they are on different network.

1.3.2 Acronyms

The acronyms shared between this document and the RASD document are reported in the following table:

Acronyms	Definition
RASD & Requirements Analysis & Specification Document	
CV	Curriculum vitae

Table 3: RASD Acronyms

The acronyms specific to this document are reported in the following table:

Acronym	Definition
DD	Design Document
UI	User Interface
UX	User Experience
DB	Database
API	Application Programming Interface
ORM	Object-Relational Mapping
DBMS	Database Management System
OLAP	Online Analytical Processing
SPA	Single Page Application
DMZ	Demilitarized Zone

Table 4: DD Acronyms

1.3.3 Abbreviations

The abbreviations shared between this document and the RASD document are reported in the following table:

Abbreviations	Definition
S&C	Students&Companies

Table 5: RASD Abbreviations

1.4 Revision History

1.5 Reference Documents

- Assignment RDD AY 2024-2025 [A]

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Revised on	Version	Description
7-1-2025	1.0	Initial release of the document

Table 6: Document Revision History

- Software Engineering 2 A.Y. 2024/2025 Slides “CreatingDD”

1.6 Document Structure

1. **Introduction**: This section provides a concise summary of the RASD document, explaining its purpose, definitions, and acronyms. Additionally, it includes a non-technical overview of the technical decisions made for the platform’s implementation.
2. **Architectural Design**: In this section, a top-down perspective of the S&C platform’s architectural design is presented. It begins with a high-level description of groups of components and their interactions, detailing the platform’s various areas and the design decisions behind them. A more detailed view follows, describing each component, their interfaces, and the architectural styles and patterns applied. Finally, deployment and runtime views of the system are represented using sequence diagrams.
3. **User Interface Design**: This section offers a technical description of the platform’s user interface design, supplemented with images and explanations. It builds upon the descriptions provided in the RASD to include more technical insights.
4. **Requirements Traceability**: This section contains a traceability matrix that links the requirements defined in the RASD to the system components responsible for implementing them.
5. **Implementation, Integration, and Test Plan**: This section outlines the tools and methodologies to be used during platform development. It describes plans for testing the software’s correctness and ensuring its quality.
6. **Effort Spent**: This section provides a summary table detailing the hours each group member spent developing this document.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

2 Architectural Design

The purpose of this chapter is to present a top-down description of the S&C architectural design, covering and justifying every design decision. First, we introduce the high-level components and their interactions. Next, we proceed with a detailed description of these components. Subsequently, we define the deployment strategy. After that, we address the runtime interactions of the components and provide a more detailed description of their interfaces. Finally, we conclude by outlining the main patterns adopted and other relevant design decisions.

2.1 Overview: High-level Components and their Interaction

The system employs a simple microservices architecture composed of the Presentation, Application, and Authenticator services, along with databases to manage the Application data. This microservices structure enables the system to scale and adapt to increasing demand. Additionally, it supports greater decoupling and modularization, facilitating the management of growing service complexity in the future.

Client access to server content is handled by a proxy, which routes requests to the appropriate service. When users navigate to the platform's main domain using a browser, the proxy directs them to the Presentation service, responsible for providing the user interface and experience.

The web interface communicates with the Application service via a RESTful API that handles the business logic. All service calls from the client-hosted presentation layer pass through the proxy, which analyzes and forwards the requests.

Requests requiring authentication are routed to the Authenticator service, which acts as middleware to handle all user authentication processes. If authentication succeeds, the request is forwarded to the Application service.

The Application service interacts with databases through APIs that manage its data.

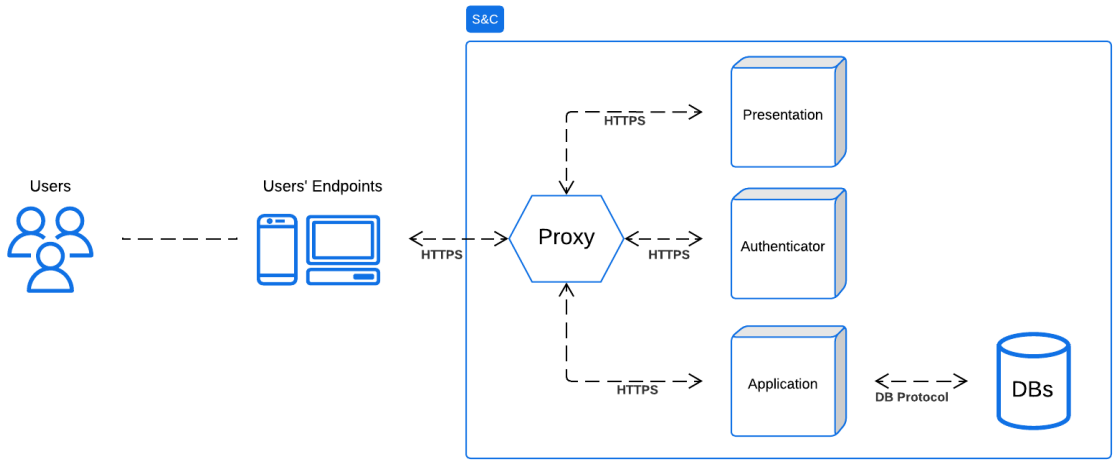

Figure 1: Architectural Design Overview

2.2 Component View

This section provides a more in-depth view of the software components that are part of the designed architecture, as well as the necessary interfaces between them.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

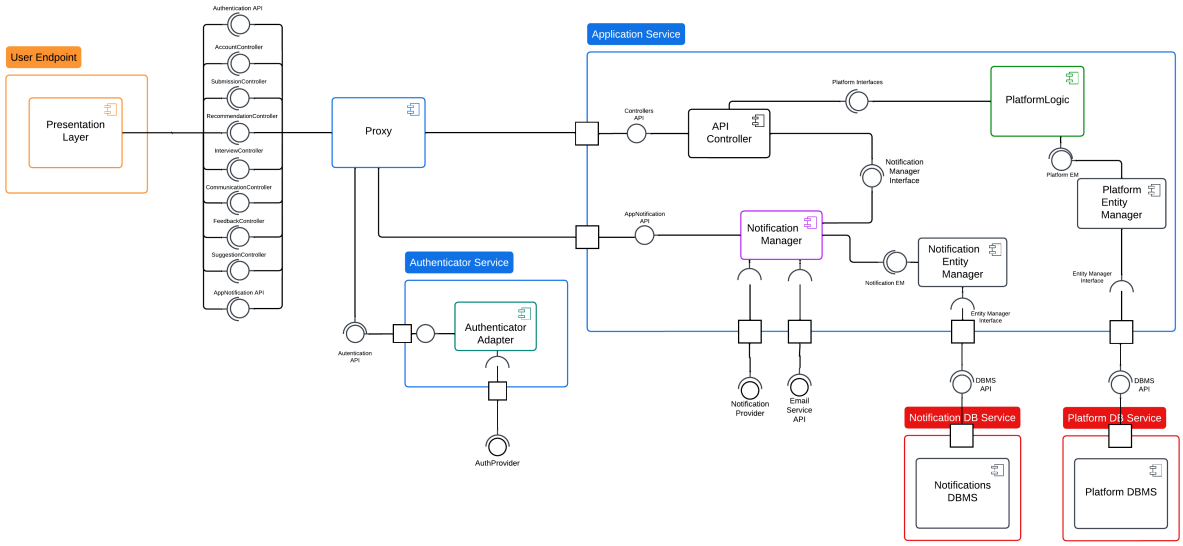

Figure 2: Architecture Components Diagram

2.2.1 Entity Manager

The Entity Manager acts as an API that enables communication with a DBMS, simplifying ORM, querying, and data life cycles. It provides standard methods, independent of the specific DBMS used to handle the data. As shown in the diagram, there are two Entity Managers. The Platform Entity Manager provides its interface to the Platform Logic component, enabling interaction with the Platform DBMS. The Notification Entity Manager works analogously with the Notification DBMS.

2.2.2 Platform Logic

The inner components of the Platform Logic encapsulate the logic of the main parts of the S&C environment. Each component autonomously handles its data management through the database, leveraging the Entity Manager interface.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

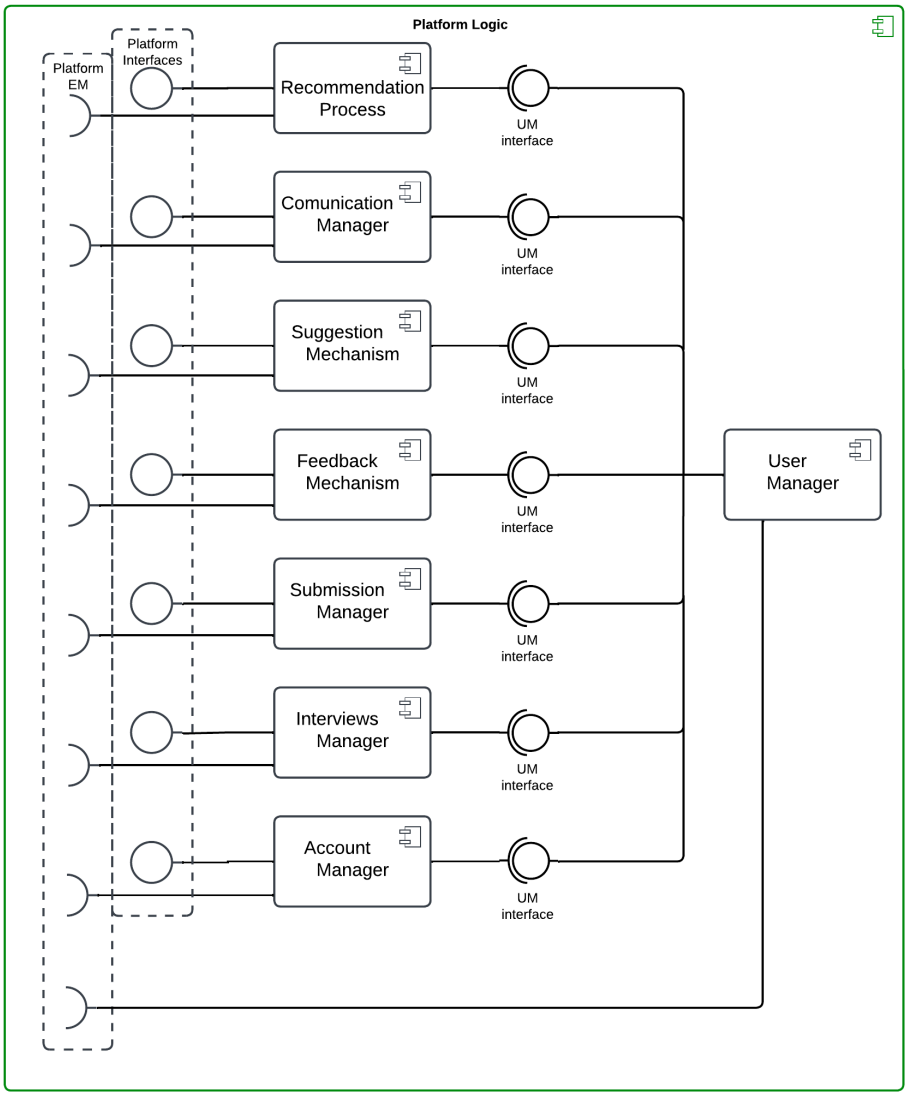

Figure 3: Platform Logic Inner Components Diagram

Component	Description
Recommendation Process	Handles recommendation logic and related data
Communication Manager	Manages communication logic and related data
Suggestion Mechanism	Handles suggestions logic and related data
Feedback Mechanism	Processes feedback logic and related data
Submission Manager	Manages submission logic and related data
Interview Manager	Handles interview-related logic and data
Account Manager	Manages account logic and related data
User Manager	Interface for querying and modifying user groups with specific characteristics

Table 7: Description of platform components.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

2.2.3 API Controller

The API Controller consists of a set of inner controllers whose methods, triggered by user calls, interact with and execute the logic of the Platform Logic inner components depicted above.

2.2.4 Notification Manager

This component handles all notification-related needs, regardless of their type. It functions as an adapter for external push notification providers and email services, offering an interface to seamlessly integrate these external features with other services. It also creates and manages corresponding in-app notifications, which can be fetched by users through a dedicated AppNotification API.

This component serves as a clear example of a service that could easily be exported to its own container in the future, for instance, by exposing its own RESTful API to the Platform Logic instead of using the current interface-based setup.

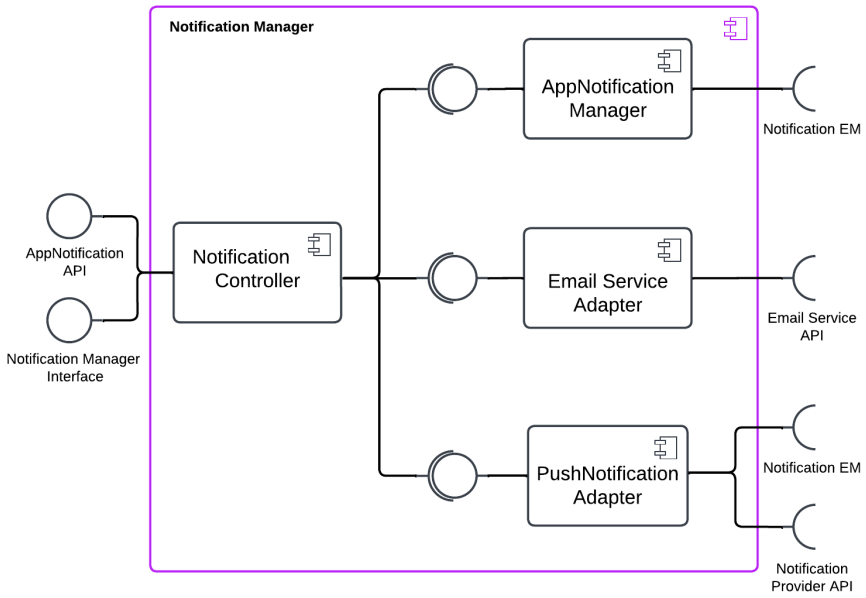

Figure 4: Notification Manager Inner Components Diagram

2.3 Deployment View

Each service will be hosted on its own container being able to run independently on the same or on different machines. Containers, in addiction to make the development and deployment easier, also provide a good level of isolation and security. The system will be hosted on the cloud and its container based nature allows to easily integrate orchestrator tools to decouple it from the cloud hosting provider and automatically manage scalability, reliability, fault tolerance and global security of the microservices cluster. A DMZ can be implemented in this design by placing the Proxy in a dedicated network segment isolated from both the external network and the internal services. A firewall between the DMZ and the external network would allow only HTTP/HTTPS traffic to the Proxy, while another firewall between the DMZ and the internal network would restrict traffic to authorized services. This setup enhances security by exposing only the Proxy to external access, keeping all the other services, along with the databases, fully protected within the internal network.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

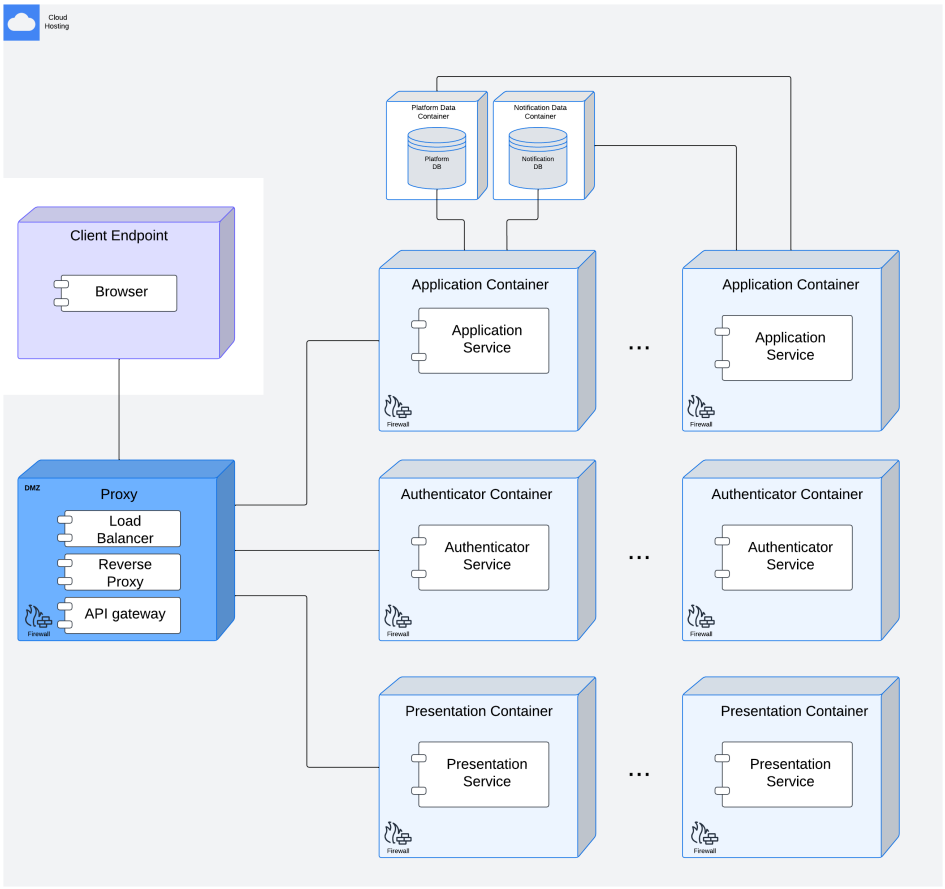

Figure 5: Deployment Diagram

2.4 Runtime View

Sequence Diagrams Descriptions

This section describes the interactions between system components during S&C runtime, depicting their call stacks at a high level of abstraction. For this purpose, sequence diagrams will be used.
The call stacks are often similar, as they follow the patterns established in the previous sections. All calls are triggered by the user through the Presentation layer, which sends API calls to the Proxy. The Proxy, in turn, forwards the requests to the appropriate service and, depending on the request, may add a middleware API call before reaching the final target.
For API requests that need to pass data to the services, the data type—always a JSON object—is not directly specified in the arrow representing the API call but can be inferred from the parameters of the subsequent call stack methods.
By analyzing the API endpoints, the Proxy determines where to redirect the requests, triggering middleware procedures such as those for authentication. These procedures will be described in advance, as they form the foundation for understanding the diagrams presented later in the section.
Every database interaction is performed through Entity Managers, which are responsible for querying the respective DBMS. All the steps described in purple are explained in detail in different runtime views.
Before proceeding to other diagrams, it is highly recommended to review the UserRegistration and UserLogin diagrams, as they are the most complex and thoroughly described ones. The other diagrams are more intuitive and, for this reason, less detailed, as they should be easily understood after gaining a general understanding of the first two.
Representing every return code or message for each call would be confusing. However, since failure

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

behaviors are often shared across components, we have distributed the possible alternatives among all the diagrams rather than including all possibilities in each one. Additionally, due to the system's complexity, we have chosen to represent the notification sending process as a single action performed by the Notification Manager, without detailing the actions of its internal components, which are fully shown in the component view. We also decided to simplify the DBMS response to a generic "result" when the query returns data, and a generic "success" when the query is successful but returns no data.

User Registration Diagram

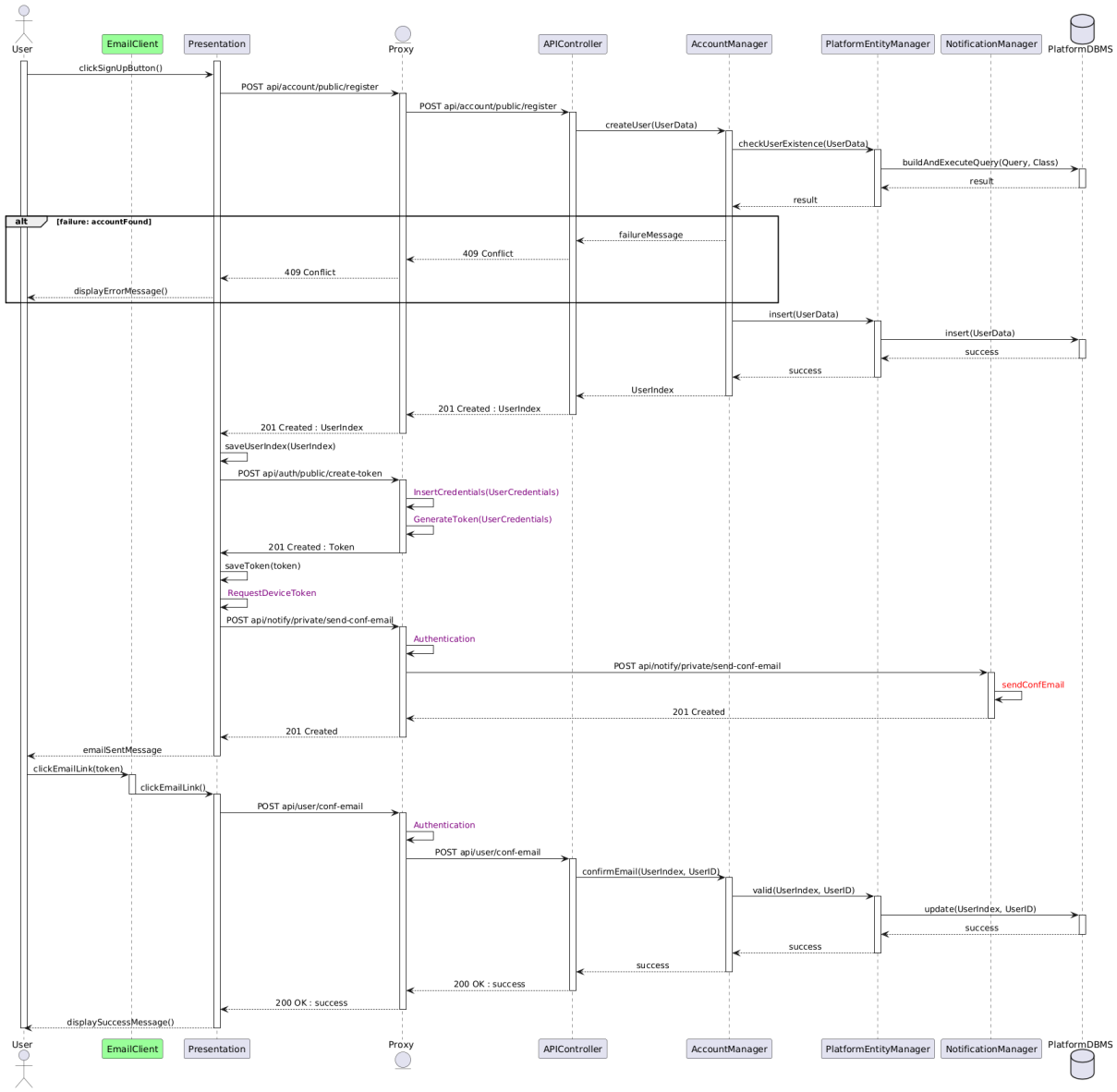

Figure 6: User Registration Sequence Diagram

The User Registration view is the most complex one, as it involves the initial procedure for all the features of our web application, such as authentication, session validation, and notifications.
When the User clicks the SignUp Button, the Presentation Layer sends a POST request to the Proxy with the UserData object in its body. The Proxy recognizes the request as public based on the endpoint and forwards it to the Application Service's API Controller.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

The API Controller accesses the AccountManager, which queries the databases through the Platform Entity Manager to check for user existence. If the User exists, the call stack proceeds backward, returning an error to the User. This could happen, for instance, if the user enters an email or a VAT that is already in use. If the UserData is unique and valid, the AccountManager creates a new User and stores it in the database.
A success API call code and a UserIndex object are returned to the Presentation Layer, which saves the object locally. The UserIndex is required to modify the UserData in the next step.
Then, the Presentation Layer sends a POST request with the UserCredential object to the Authenticator to generate the session token. The Proxy adds a middleware request to the Authenticator to insert the UserCredentials, a required step to allow the Authenticator to generate the token. For a detailed view see the *(fig:7) InsertCredentials* and *(fig:7) GenerateToken* diagrams
After the token is generated, the Authenticator returns it to the Presentation Layer, which saves it for future private calls. The Presentation Layer then sends a request to obtain a DeviceToken, which will be needed to notify its endpoint device. See for an in-depth view of the *(fig:9) RequestDeviceToken* step.
The final call triggered by the button click is a POST request responsible for sending the email confirmation. Since the request is to a private endpoint, the Proxy adds a middleware request to the Authenticator service to authenticate the user using the previously obtained token, provided in the header of every private API call. To understand how the Authentication works in detail, see the *(fig:8) Authentication* diagram.
After validation, the Proxy forwards the request to the NotificationManager, which sends the email by communicating with the EmailServiceProvider. The communication with the EmailServiceProvider is not shown in the diagram to avoid unnecessary complexity, as it consists of a simple call to an external service. For this reason, it is depicted in red text.
The EmailServiceProvider sends the email to the User and returns a success message to the NotificationManager, which forwards it to the Presentation Layer, ultimately returning control to the User.
The User receives the email and clicks the link to confirm their email using their EmailClient. The EmailClient link opens a Presentation Layer page, which sends a POST request to the Proxy. The Proxy authenticates the user as described earlier and forwards the request to the Application Service's API Controller, which accesses the AccountManager to activate the User. The AccountManager updates the User's status in the database and returns a success message to the Presentation Layer, which displays it to the User.
The User is now successfully registered and can use the web application features.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

InsertCredentials - GenerateToken

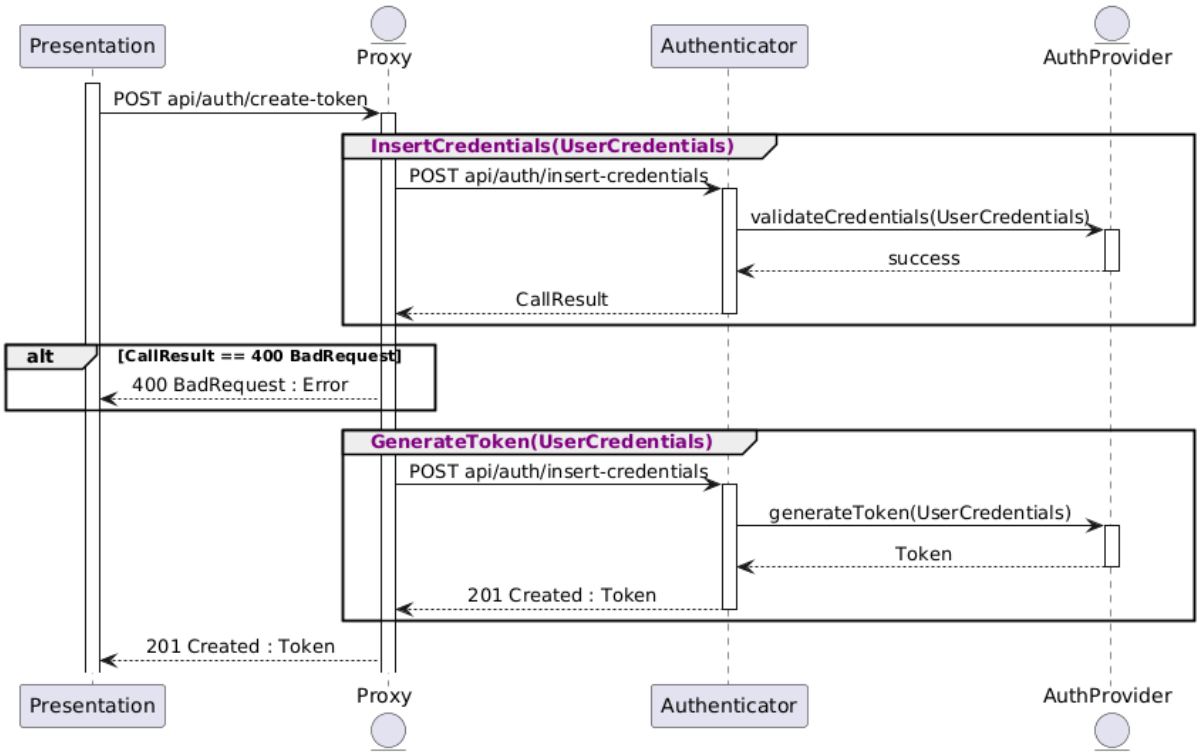

Figure 7: InsertCredentials - GenerateToken Sequence Diagram

By adding the InsertCredentials middleware call, the Proxy simply calls the POST method of the Authenticator service with the UserCredentials object in its body. The Authenticator communicates with the external AuthProvider, which adds the UserCredentials to its database.
Every call to an external service provider can generate errors that do not depend on our system. These errors typically correspond to response codes starting with 5 - - and should be handled properly. However, there can also be errors that depend on our system, such as a malformed UserCredentials object. These errors typically have codes starting with 4 - -, and this type of error is depicted in the diagram as a failure example.
If the call is successful, the AuthProvider returns a success message to the Authenticator, which forwards it to the Proxy. The Proxy then proceeds with the GenerateToken step, which consists of a POST call to another Authenticator endpoint responsible for generating the token associated with the UserCredentials object.
The Authenticator communicates with the AuthProvider to generate the token and returns it to the Proxy. The Proxy forwards the token to the Presentation Layer, which saves it for future private calls, as shown in the previous diagram.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Authentication

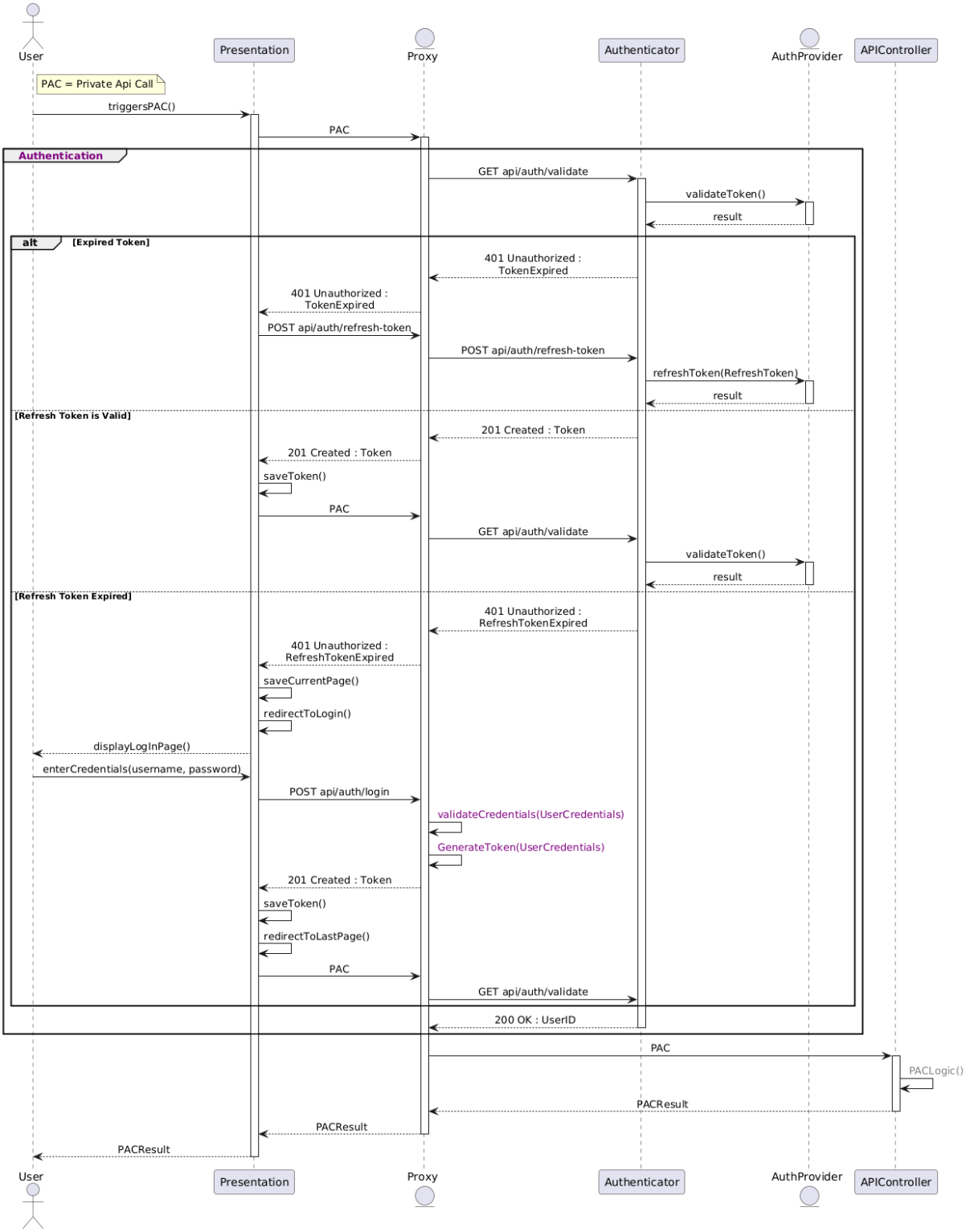

Figure 8: Authentication Sequence Diagram

When a private request is sent to the Proxy by the Presentation Layer, the Proxy adds a middleware call to the Authenticator to authenticate the user. The Authenticator receives a GET request and checks the token in the header.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

If the token is valid, the Authenticator returns a success message to the Proxy, which forwards the request to the Application service. If the token is invalid, the Authenticator returns an error message to the Proxy, which then forwards the error message to the Presentation Layer.
The Presentation Layer first attempts to refresh the token using the RefreshToken provided when the authentication token was originally issued. If the RefreshToken is valid, the Presentation Layer will request a new token from the Authenticator. If the RefreshToken is also invalid, the Presentation Layer will save the current page and redirect the user to the login page to re-authenticate and obtain a new authentication token and RefreshToken.
After the UserLogin process is completed, the Presentation Layer will save the newly obtained token and redirect the user to the previously saved page.
For more details about the ValidateCredentials step, see the UserLogin diagram.

RequestDeviceToken

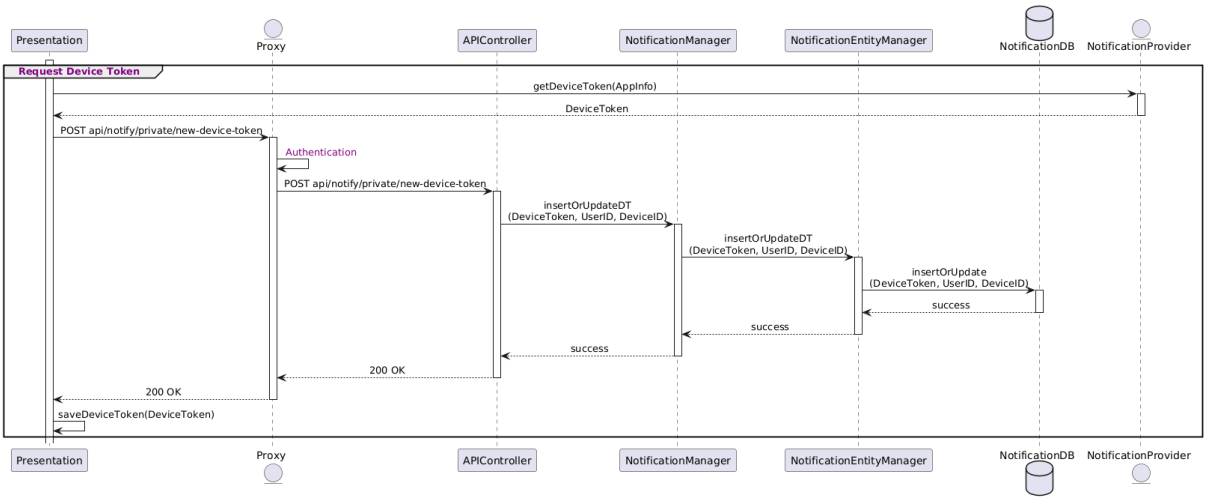

Figure 9: RequestDeviceToken Sequence Diagram

The RequestDeviceToken step begins with the Presentation Layer directly communicating with the NotificationProvider, which is responsible for handling the token associated with the user's endpoint device. To obtain the DeviceToken, some device information—retrievable through browser methods—is sent to the external service provider. The NotificationProvider returns the DeviceToken to the Presentation Layer, which sends it to the Application service so that the NotificationManager can store it in its database and bind it to the User.
Since a user can access S&C on multiple devices, a DeviceID object is required to inform the server which device the token is associated with. As a consequence, each device has its own DeviceToken, and a user can have multiple DeviceTokens, each bound to a unique DeviceID.
After the NotificationManager has stored the DeviceToken, it returns a success message to the Presentation Layer, which saves it locally.
The DeviceToken can expire or be invalidated by the external service provider. In that case is the Presentation Layer the one responsible to update the DeviceToken as shown next.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

CheckDeviceToken

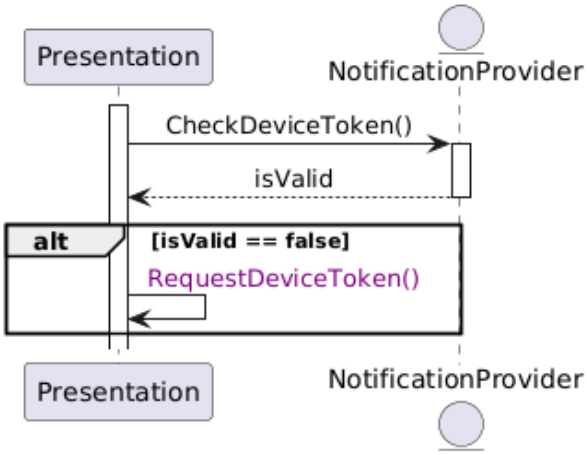

Figure 10: CheckDeviceToken Sequence Diagram

The CheckDeviceToken step is scheduled by the Presentation Layer. It consist of a direct communication to the NotificationProvider to check the actual locally saved token. If the token is invalid, the Presentation Layer will request a new one as shown in the RequestDeviceToken step. If the token is valid, the Presentation Layer will do nothing.

UserLogin - ValidateCredentials

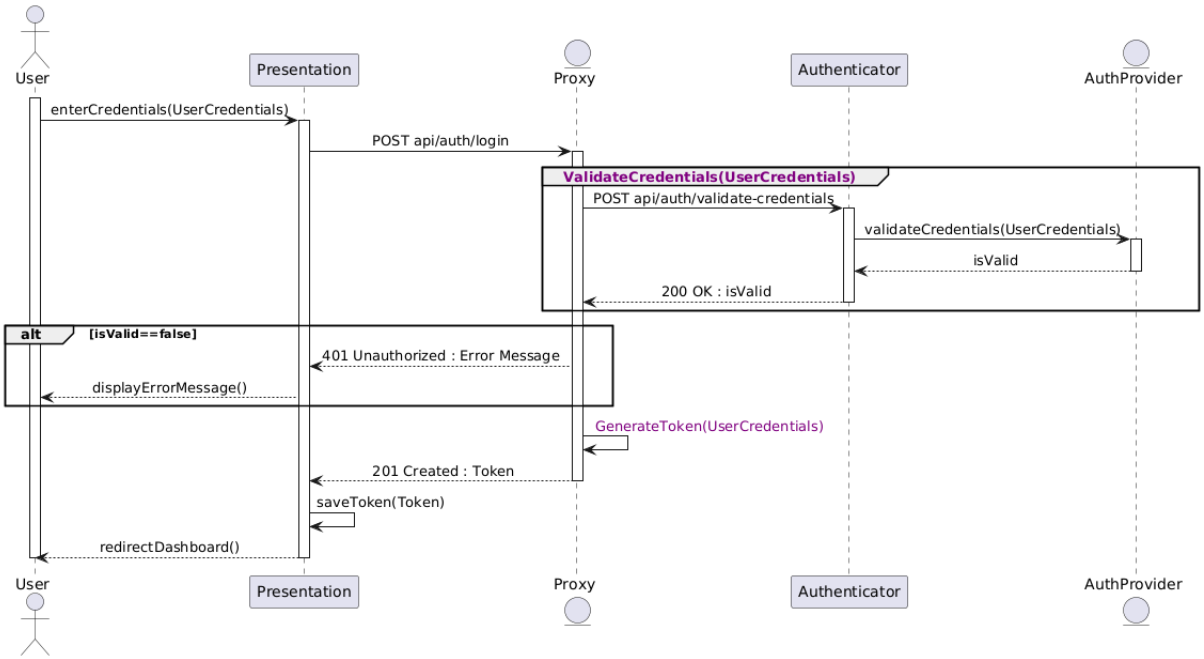

Figure 11: UserLogin ValidateCredentials Sequence Diagram

By clicking the LogIn button, the user sends the entered UserCredentials from the Presentation Layer to the Proxy. This triggers a POST public request that reaches the Proxy. The Proxy adds a middleware call to the Authenticator to validate the UserCredentials. The Authenticator receives the request and

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

checks the credentials in the AuthProvider. If the credentials are valid, the Proxy forwards the request again to the Authenticator, which generates the token associated with the provided credentials. The ValidateCredentials process differs from the InsertCredentials process because, for validation to succeed, the credentials must have been previously "inserted" into the AuthProvider. Once the token is generated, the Authenticator returns it to the Presentation Layer, which saves it for future private calls.

Participant Submission

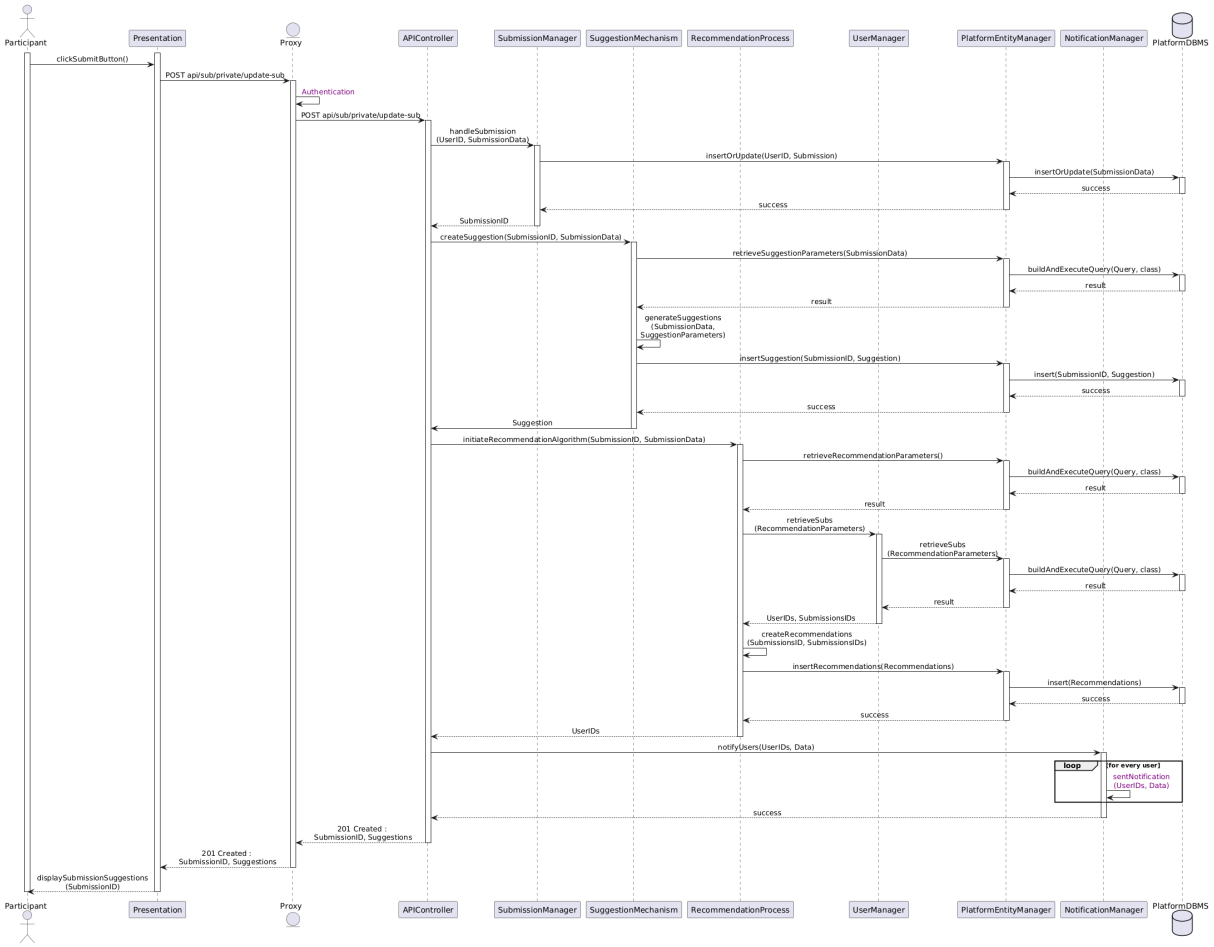

Figure 12: Participant Submissions Sequence Diagram

By clicking the Submit button, the Participant sends the submission data from the Presentation Layer to the Proxy. The triggered call is a POST private request, which the Proxy authenticates using the Authenticator. Once authenticated, the Proxy forwards the request to the APIController.
The APIController processes the submission by calling the SubmissionManager, which updates or inserts the data into the PlatformDBMS through the PlatformEntityManager. Afterward, the SuggestionMechanism generates suggestions for the submission, querying and updating the database as needed.
The APIController then triggers the RecommendationProcess, which identifies relevant users and submissions through the UserManager. Recommendations are generated, stored, and passed back to the APIController.
Finally, the NotificationManager notifies the identified users, and the APIController returns the SubmissionID and suggestions to the Presentation Layer, which displays them to the Participant.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Send Notification

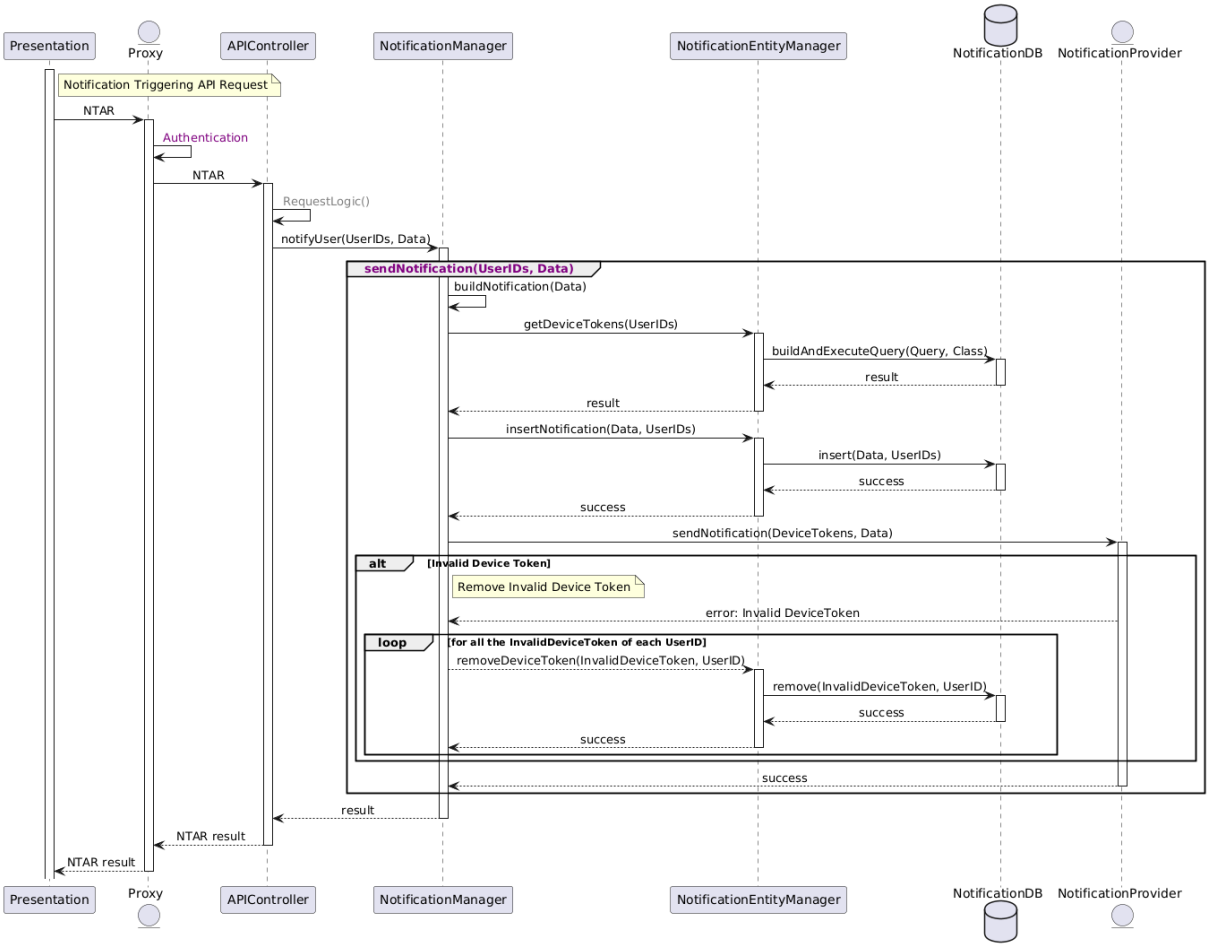

Figure 13: Send Notification Sequence Diagram

The process begins when the Presentation Layer triggers a Notification Triggering API Request (NTAR) to the Proxy. The Proxy authenticates the request and forwards it to the APIController.
The APIController processes the request with the required logic and calls the NotificationManager to notify the specified users. The NotificationManager first builds the notification data and retrieves the device tokens of the users from the NotificationDB through the NotificationEntityManager.
After retrieving the device tokens, the NotificationManager stores the notification data in the NotificationDB and sends the notifications to the users' devices via the NotificationProvider.
If any invalid device tokens are detected during this process, the NotificationProvider returns an error, prompting the NotificationManager to remove these invalid tokens from the NotificationDB for each associated user.
Once the notifications are successfully sent, or all invalid tokens are handled, the NotificationManager returns the result to the APIController. The APIController forwards the result back to the Proxy, which then returns it to the Presentation Layer.
Note that the NotificationManager simply removes the invalid tokens from the NotificationDB. Is the Presentation Layer the one responsible to update the DeviceToken as shown in the CheckDeviceToken step.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

User Opens Company Internship Offer

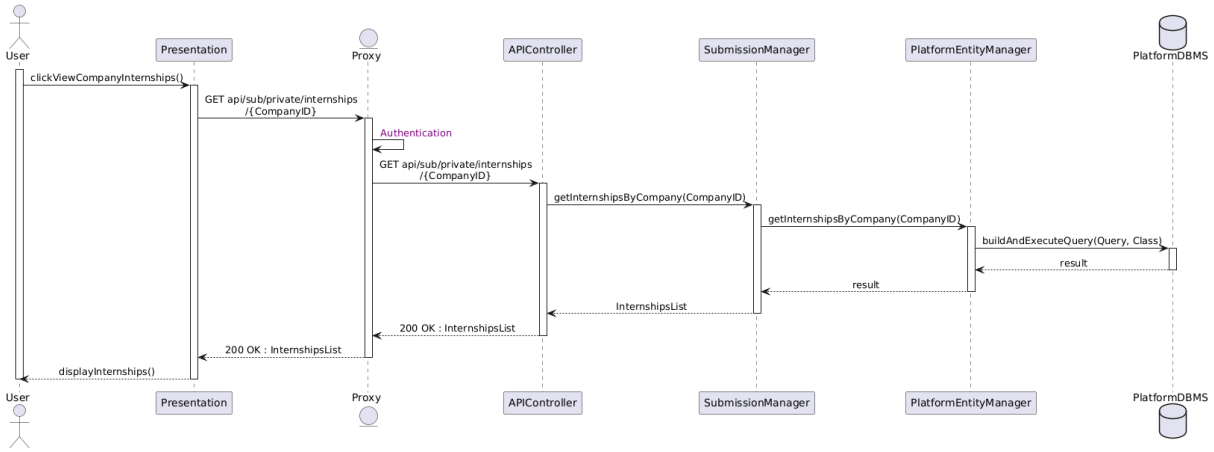

Figure 14: User Opens Company Internship Offer Sequence Diagram

By clicking the View Company Internships button, the User triggers a GET private request from the Presentation Layer to the Proxy. The Proxy authenticates the request using the Authenticator and forwards it to the APIController in the Application Service.
The APIController calls the SubmissionManager to retrieve the internships associated with the specified CompanyID. The SubmissionManager queries the database via the PlatformEntityManager, which executes the query in the PlatformDBMS and returns the results.
The SubmissionManager sends the list of internships back to the APIController, which forwards it to the Proxy. The Proxy then returns the response to the Presentation Layer, which displays the internships to the User.

User Opens Student CV

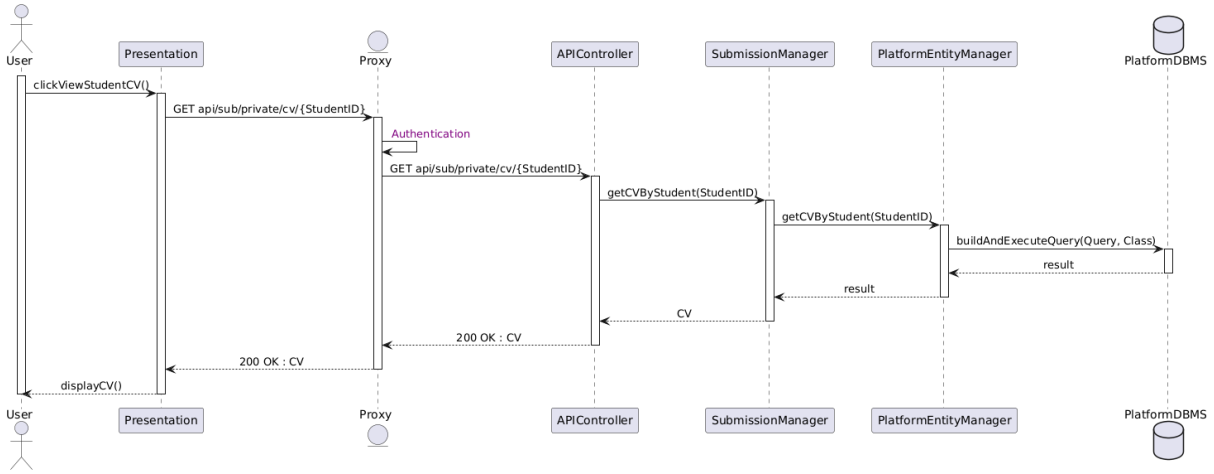

Figure 15: User Opens Student CV Sequence Diagram

By clicking the View Student CV button, the User triggers a GET private request from the Presentation Layer to the Proxy. The Proxy authenticates the request using the Authenticator and forwards it to the APIController in the Application Service.
The APIController invokes the SubmissionManager to retrieve the CV of the specified StudentID. The

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

SubmissionManager queries the database through the PlatformEntityManager, which executes the query in the PlatformDBMS and returns the result.
The SubmissionManager sends the CV back to the APIController, which forwards it to the Proxy. The Proxy then returns the CV to the Presentation Layer, which displays it to the User.

Participant Accepts Match

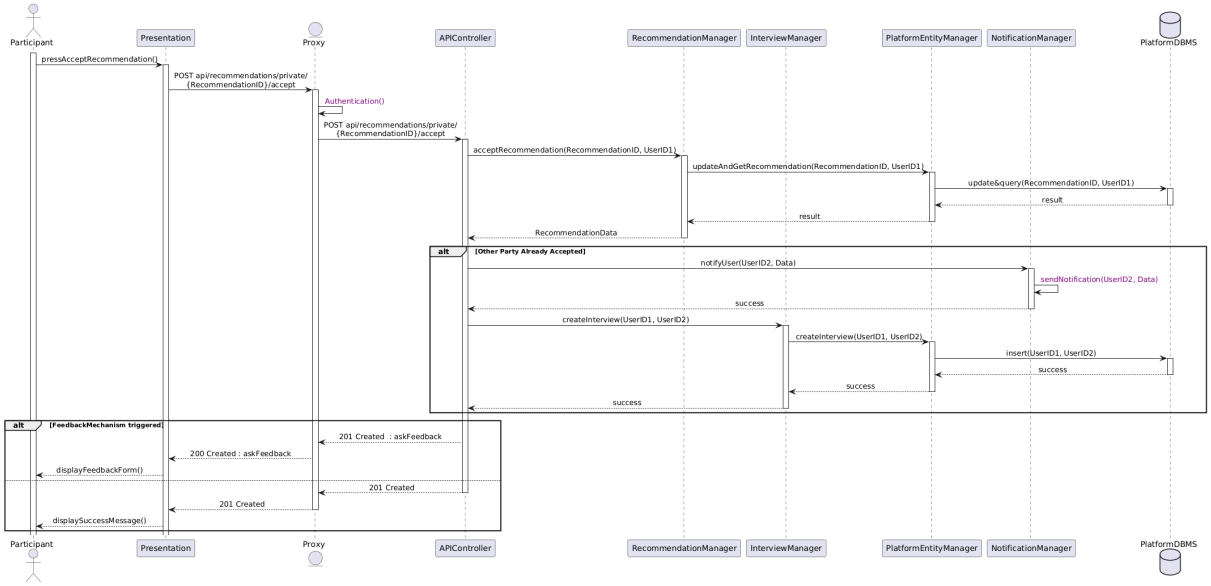

Figure 16: Participant Accepts Match Sequence Diagram

By pressing the Accept Recommendation button, the Participant triggers a POST private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController calls the RecommendationManager to process the acceptance of the recommendation. The RecommendationManager updates and retrieves the recommendation data by interacting with the PlatformEntityManager, which queries the PlatformDBMS.
If the other party has already accepted the recommendation, the APIController notifies the second user through the NotificationManager and initiates the creation of an interview using the InterviewManager. The InterviewManager inserts the interview data into the PlatformDBMS through the PlatformEntityManager.
If the FeedbackMechanism is triggered, the APIController returns a 201 Created response with a prompt for feedback, which the Presentation Layer displays to the Participant as a feedback form. Otherwise, the APIController simply returns a success message, which the Presentation Layer displays to the Participant.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Participant Submits Feedback

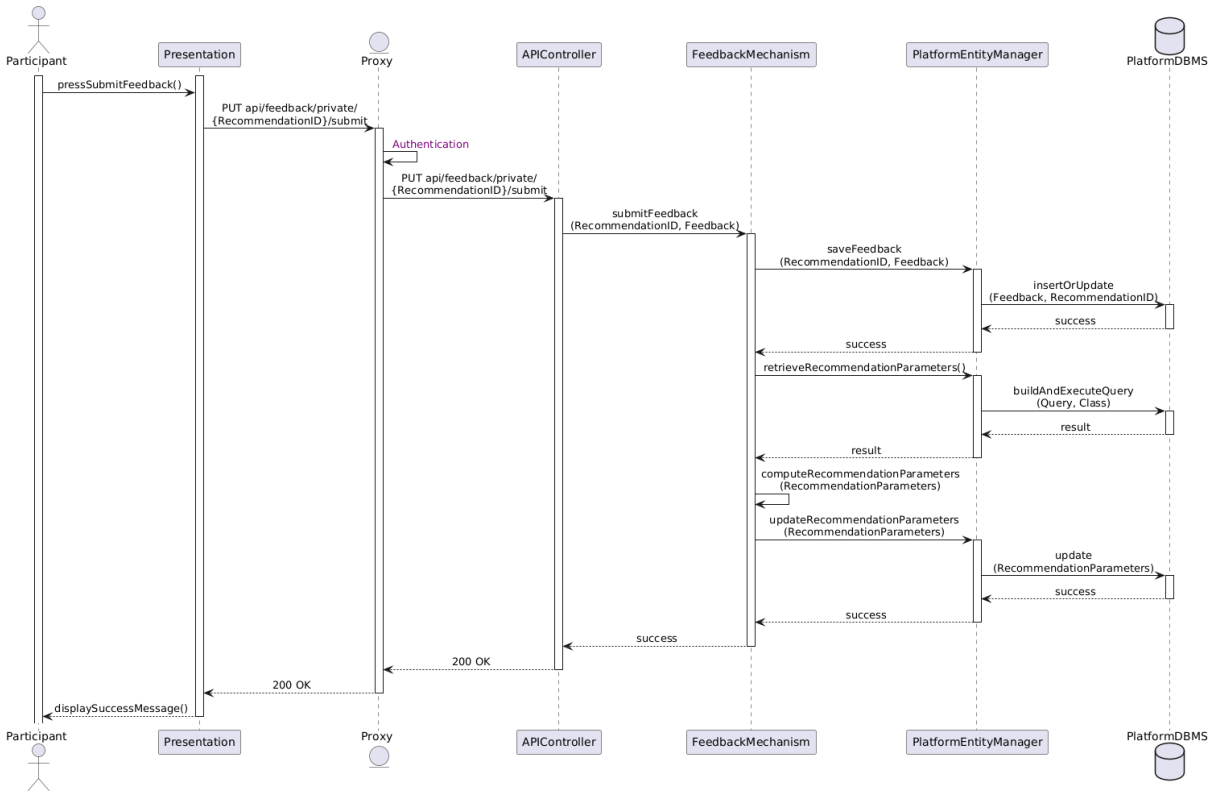

Figure 17: Participant Submits Feedback Sequence Diagram

When the Participant presses the Submit Feedback button, the Presentation Layer sends a PUT request with the RecommendationID and feedback data to the Proxy.
The Proxy authenticates the request and forwards it to the APIController. The APIController invokes the FeedbackMechanism to handle the feedback submission. The FeedbackMechanism saves the feedback data by interacting with the PlatformEntityManager, which updates the PlatformDBMS.
Once the feedback is saved, the FeedbackMechanism retrieves recommendation parameters from the database through the PlatformEntityManager. It computes updated recommendation parameters and saves them back into the database via the PlatformEntityManager.
After processing the feedback and updating the recommendation data, the FeedbackMechanism returns a success message to the APIController. The APIController forwards the response to the Proxy, which then sends it to the Presentation Layer. Finally, the Presentation Layer displays a success message to the Participant.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Student Sends Spontaneous Application

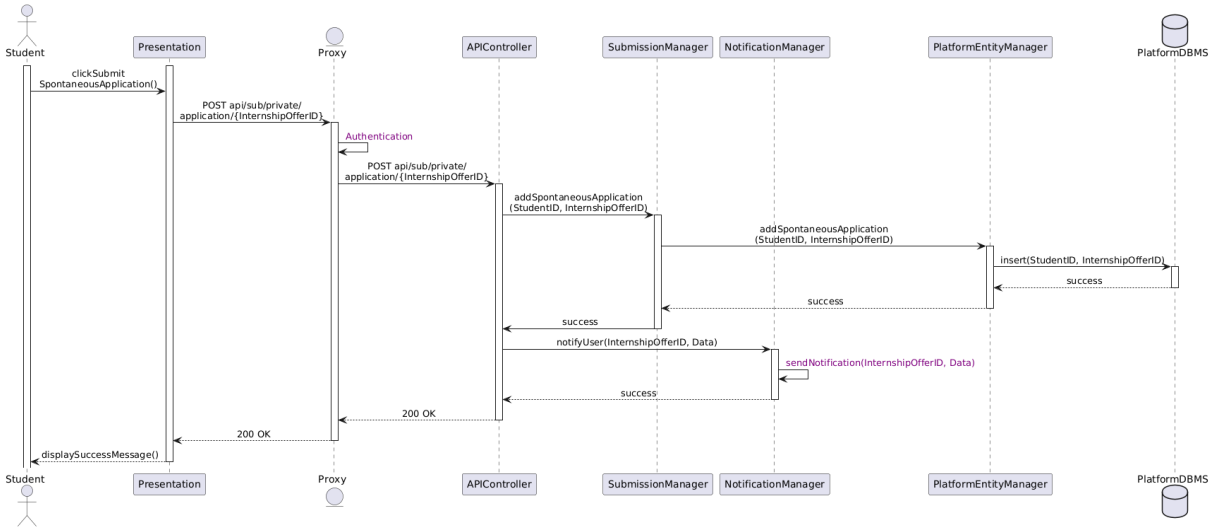

Figure 18: Student Sends Spontaneous Application Sequence Diagram

By clicking the Submit Spontaneous Application button, the Student triggers a POST private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController calls the SubmissionManager to handle the spontaneous application submission. The SubmissionManager insert the users' ID into the database, to represent the application, by interacting with the PlatformEntityManager, which performs the insertion in the PlatformDBMS.
Once the application is successfully stored, the APIController invokes the NotificationManager to notify the company about the new application. The NotificationManager sends the notification to the specified company and confirms the success of the operation.
Finally, the APIController returns a 200 OK response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a success message to the Student.

Participant Submits Feedback

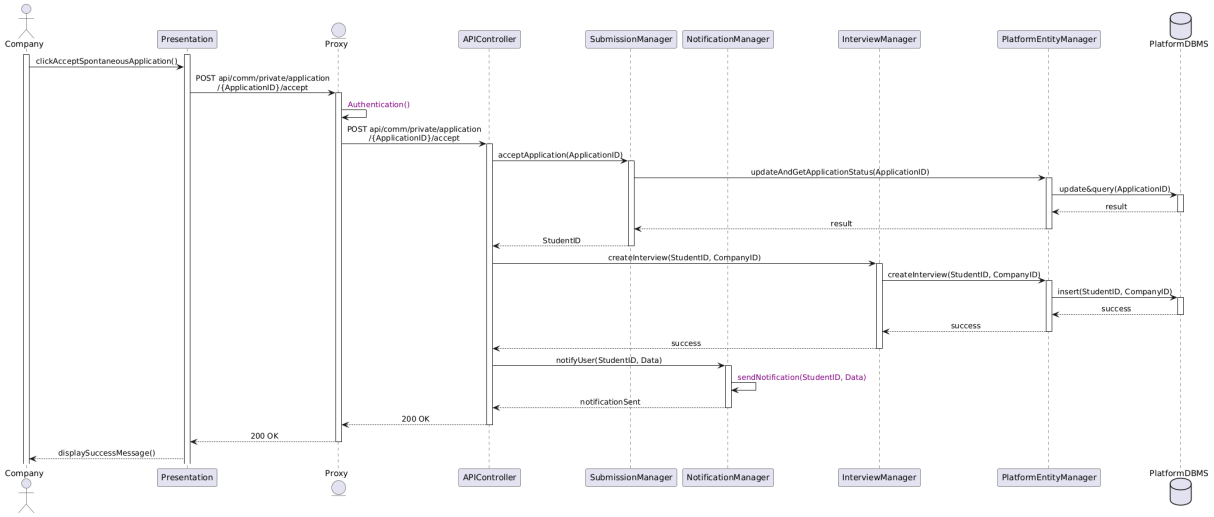

Figure 19: Company Accept a Spontaneous Application Sequence Diagram

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

By clicking the Accept Spontaneous Application button, the Company triggers a POST private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController calls the SubmissionManager to process the acceptance of the spontaneous application. The SubmissionManager updates the application status and retrieves the StudentID by interacting with the PlatformEntityManager, which queries and updates the PlatformDBMS.
With the StudentID retrieved, the APIController invokes the InterviewManager to create an interview for the student and company. The InterviewManager inserts the interview data into the PlatformDBMS through the PlatformEntityManager.
Once the interview is successfully created, the APIController calls the NotificationManager to notify the student about the acceptance of their application. The NotificationManager sends the notification and confirms the success of the operation.
Finally, the APIController returns a 200 OK response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a success message to the Company.

Student Answers Interview

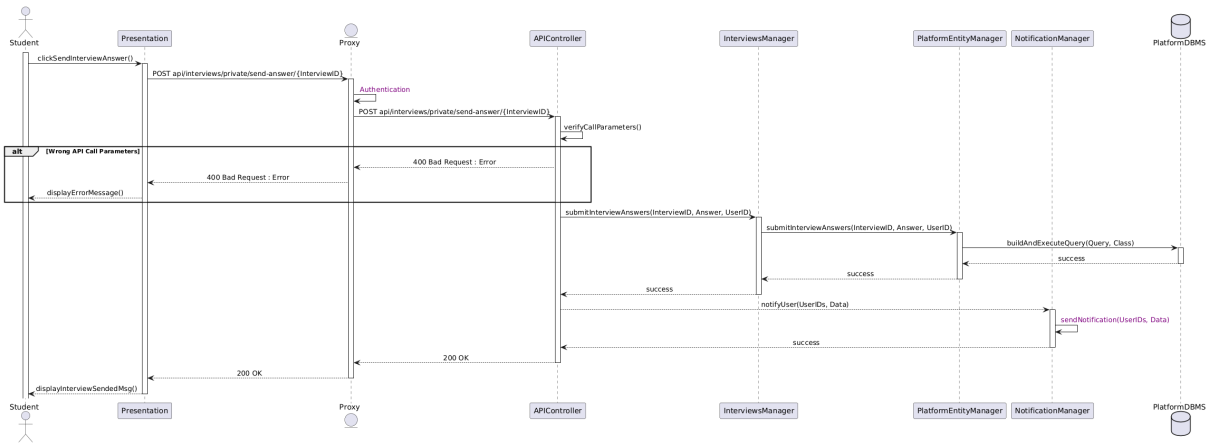

Figure 20: Student Answers Interview Sequence Diagram

By clicking the Send Interview Answer button, the Student triggers a POST private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController verifies the API call parameters. If the parameters are invalid, it returns a 400 Bad Request error to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer then displays an error message to the Student.
If the parameters are valid, the APIController calls the InterviewsManager to save the interview answers. The InterviewsManager updates the database by interacting with the PlatformEntityManager, which executes the required query in the PlatformDBMS.
After the interview answers are successfully stored, the APIController invokes the NotificationManager to notify relevant users about the submission. The NotificationManager sends the notification and confirms its success.
Finally, the APIController returns a 200 OK response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a success message to the Student, indicating that the interview answers have been submitted.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Company Submits Interview

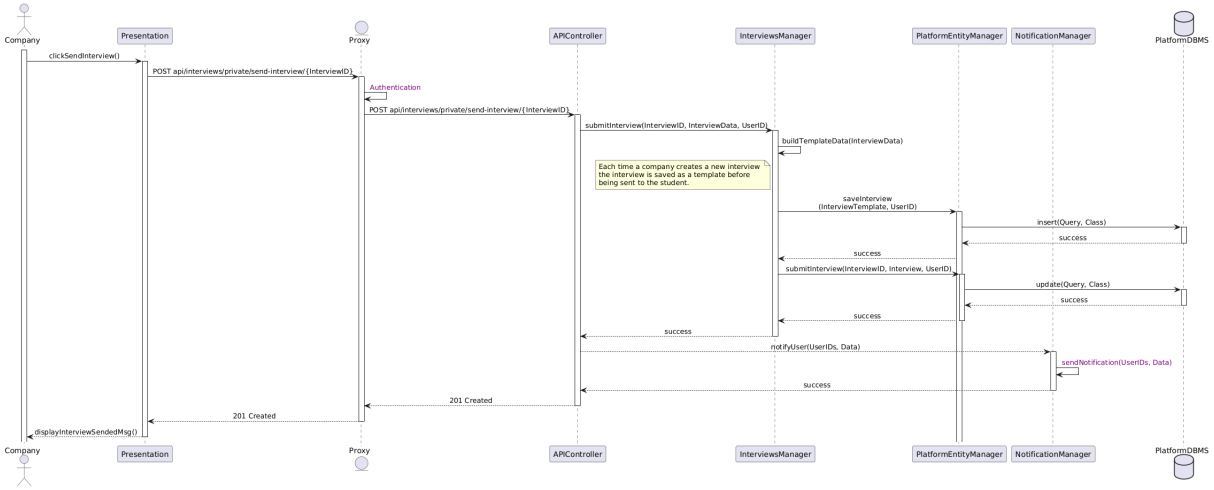

Figure 21: Company Submits Interview Sequence Diagram

By clicking the Send Interview button, the Company triggers a POST private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController invokes the InterviewsManager to handle the interview submission. The InterviewsManager first builds the interview template data from the provided InterviewData. Each time a company creates a new interview, it is saved as a template before being sent to the student.
The InterviewsManager saves the interview template to the database via the PlatformEntityManager, which performs the insertion in the PlatformDBMS. After successfully saving the template, the InterviewsManager submits the interview by updating the relevant data in the PlatformDBMS through the PlatformEntityManager.
Once the interview is successfully stored, the APIController calls the NotificationManager to notify the student about the new interview. The NotificationManager sends the notification and confirms its success.
Finally, the APIController returns a 201 Created response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a success message to the Company, indicating that the interview has been sent.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Company Creates Template Interview

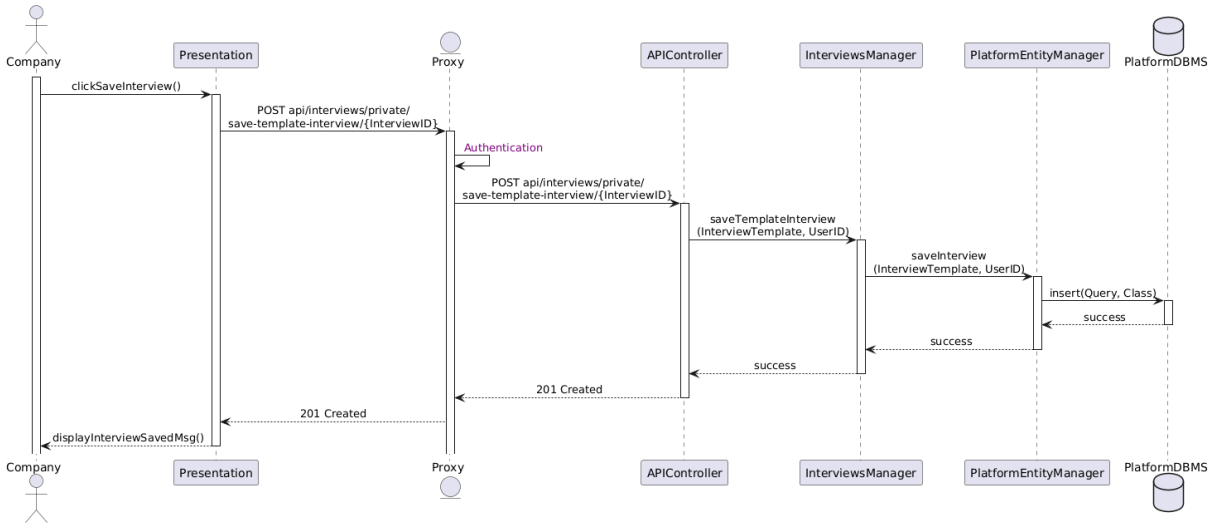

Figure 22: Company Creates Template Interview Sequence Diagram

By clicking the Save Interview button, the Company triggers a POST private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController invokes the InterviewsManager to handle the saving of the interview template. The InterviewsManager saves the provided InterviewTemplate to the database via the PlatformEntityManager. The PlatformEntityManager inserts the template data into the PlatformDBMS.
Once the interview template is successfully stored, the APIController returns a 201 Created response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a success message to the Company, indicating that the interview template has been saved.

Company Evaluates Interview

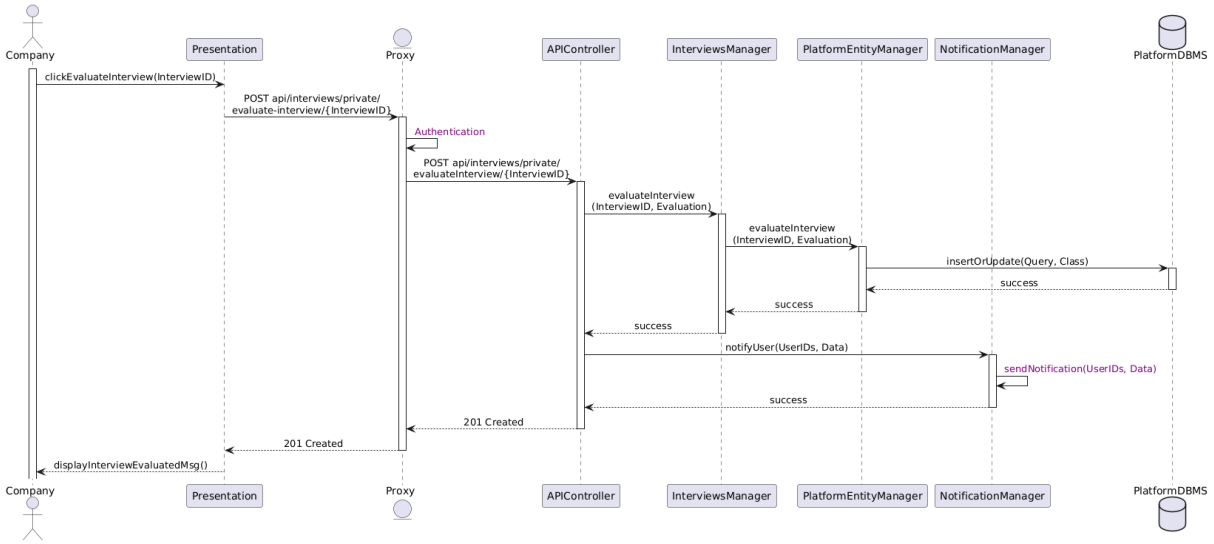

Figure 23: Company Evaluates Interview Sequence Diagram

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

By clicking the Evaluate Interview button, the Company triggers a POST private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController calls the InterviewsManager to handle the evaluation of the interview. The InterviewsManager processes the evaluation and updates the database through the PlatformEntityManager. The PlatformEntityManager performs an insert or update operation in the PlatformDBMS with the evaluation data.
After successfully saving the evaluation, the APIController triggers a notification to the student through the NotificationManager. The NotificationManager sends the notification and confirms its success.
The APIController returns a 201 Created response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a success message to the Company, confirming that the interview has been evaluated.
If the Company needs to evaluate individual questions within the interview, they can access the interview through the dashboard. By navigating to the Dashboard Interviews page, the Company sends a GET request to retrieve a list of interviews. Once the interviews are displayed, the Company can click on a specific interview to access the detailed evaluation page.

Student Sees Spontaneous Application

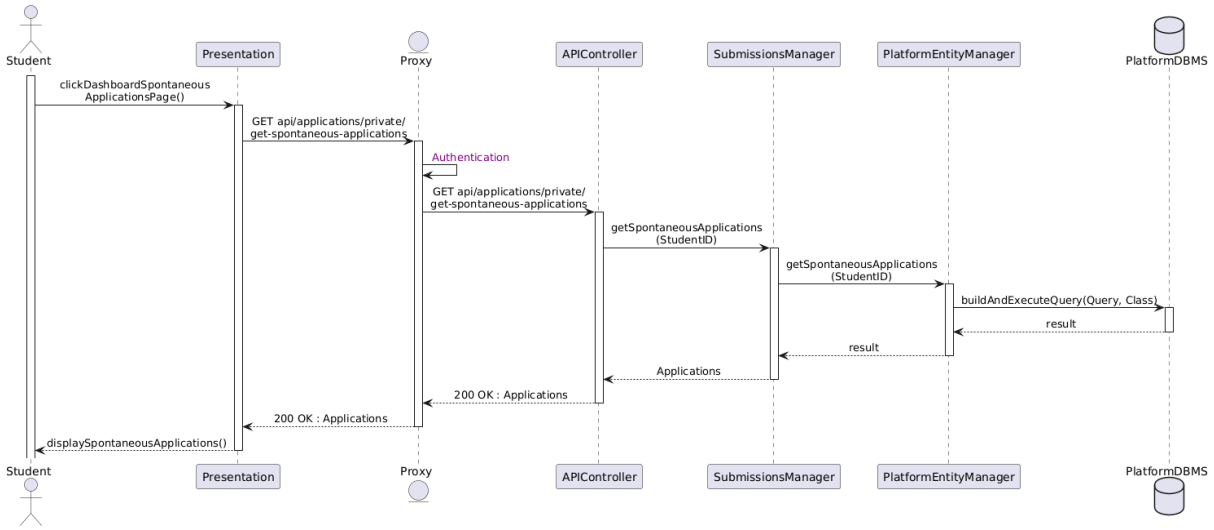

Figure 24: Student Sees Spontaneous Application Sequence Diagram

By clicking the Dashboard Spontaneous Applications page, the Student triggers a GET private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController calls the SubmissionsManager to retrieve the list of spontaneous applications related to the Student. The SubmissionsManager queries the database through the PlatformEntityManager. The PlatformEntityManager executes the query in the PlatformDBMS and retrieves the results.
The results are returned step-by-step: from the PlatformEntityManager to the SubmissionsManager, from the SubmissionsManager to the APIController, and finally to the Proxy. The Proxy sends a 200 OK response with the list of applications back to the Presentation Layer.
The Presentation Layer displays the retrieved spontaneous applications to the Student, allowing them to review their submissions.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Participant Sees Matches

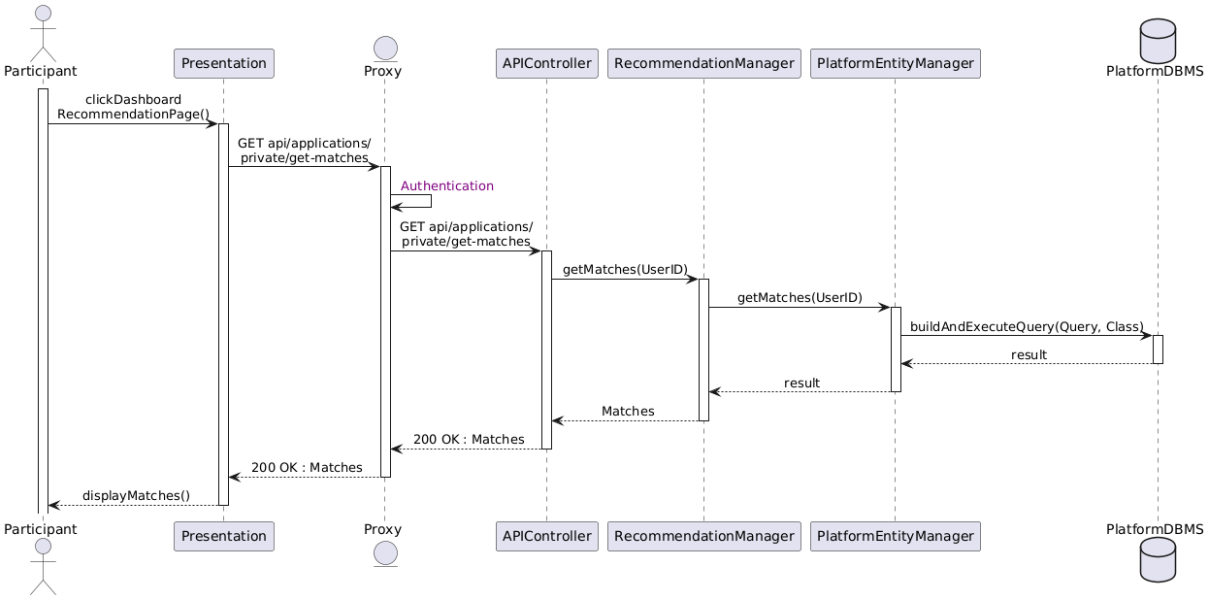

Figure 25: Participant Sees Matches Sequence Diagram

By clicking the Dashboard Recommendation page, the Participant triggers a GET private request from the Presentation Layer to the Proxy. The Proxy authenticates the request and forwards it to the APIController in the Application Service.
The APIController calls the RecommendationManager to retrieve matches associated with the Participant's UserID. The RecommendationManager queries the database through the PlatformEntityManager. The PlatformEntityManager executes the query in the PlatformDBMS and retrieves the results.
The results are returned step-by-step: from the PlatformEntityManager to the RecommendationManager, from the RecommendationManager to the APIController, and finally to the Proxy. The Proxy sends a 200 OK response with the list of matches back to the Presentation Layer.
The Presentation Layer displays the retrieved matches to the Participant, allowing them to review their recommendations.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

User Responds To Communication

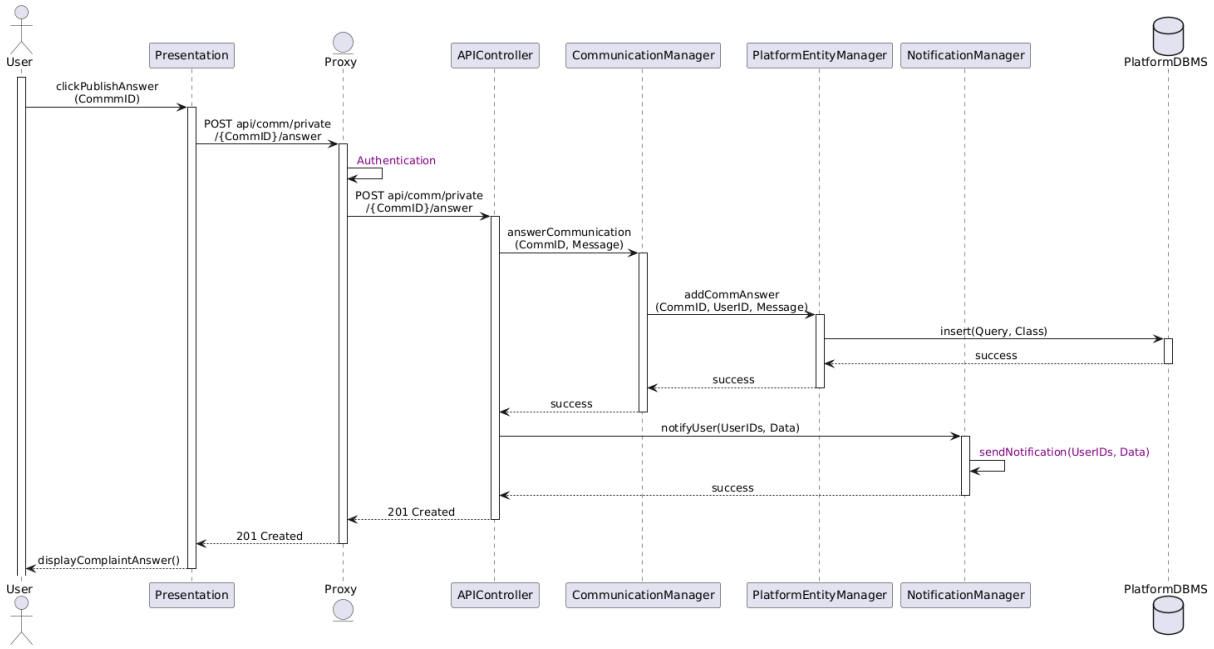

Figure 26: User Responds To Communication Sequence Diagram

By clicking the Publish Answer button, the User initiates a POST private API call from the Presentation Layer to the Proxy. The request contains the CommID, the User's message, and their authentication token.
The Proxy first validates the token using its authentication middleware and forwards the request to the APIController. The APIController processes the request by calling the CommunicationManager to handle the submission.
The CommunicationManager interacts with the PlatformEntityManager, which constructs a query to insert the answer (identified by CommID and associated with the UserID) into the database (PlatformDBMS). Upon successful insertion, a confirmation is passed back through the PlatformEntityManager and CommunicationManager to the APIController.
Subsequently, the APIController invokes the NotificationManager to notify relevant users about the new message. The NotificationManager sends notifications and confirms the operation's success.
Finally, the APIController returns a 201 Created response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a success message to the User, confirming the answer has been published successfully.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

User Opens a Complaint

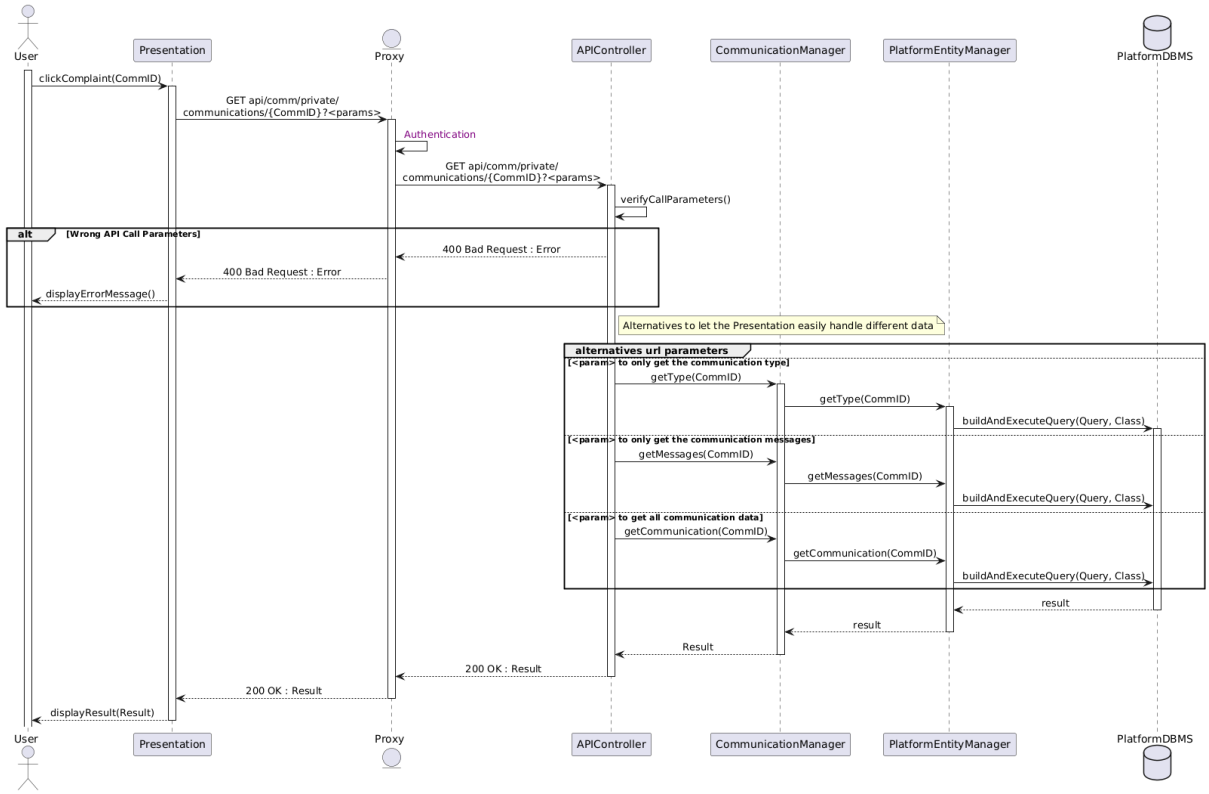

Figure 27: User Opens Complaint Sequence Diagram

When the User clicks on a complaint, a GET private API call is sent from the Presentation Layer to the Proxy. The request includes the CommID and optional URL parameters that specify the type of data to retrieve.
The Proxy validates the User's authentication token and forwards the request to the APIController. The APIController verifies the call parameters to ensure they are correct. If any parameter is invalid, the APIController responds with a 400 Bad Request error, which is forwarded by the Proxy to the Presentation Layer. The Presentation Layer displays an error message to the User.
If the parameters are valid, the APIController processes the request based on the specified URL parameter:

- If the parameter requests only the communication type, the APIController calls the CommunicationManager to retrieve the type of communication associated with the CommID.
- If the parameter requests only the communication messages, the APIController retrieves the messages through the CommunicationManager.
- If the parameter requests all communication data, the APIController retrieves the complete communication details via the CommunicationManager.

The CommunicationManager queries the PlatformDBMS through the PlatformEntityManager to fetch the requested data. Once the database query is executed successfully, the result is passed back to the APIController.
The APIController returns a 200 OK response containing the requested data to the Proxy. The Proxy forwards this response to the Presentation Layer, which displays the result to the User.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Participant Creates a Complaint

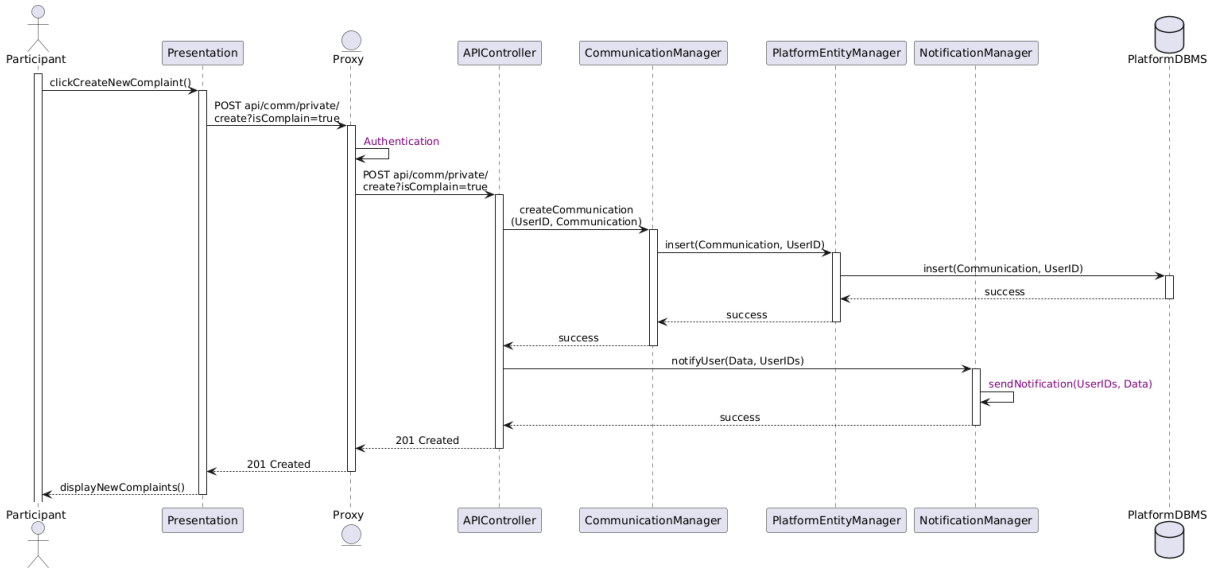

Figure 28: Participant Creates a Complaint Sequence Diagram

When the Participant initiates the creation of a new complaint by clicking the corresponding button, the Presentation Layer sends a POST request to the Proxy. The Proxy authenticates the request and forwards it to the APIController.
The APIController invokes the CommunicationManager to handle the creation of the new complaint. The CommunicationManager interacts with the PlatformEntityManager to insert the complaint data into the database. The PlatformEntityManager performs this operation by communicating with the PlatformDBMS.
Once the complaint is successfully stored in the database, the CommunicationManager returns a success response to the APIController. Subsequently, the APIController triggers the NotificationManager to notify relevant users about the new complaint. After the notifications are sent, a success response propagates back through the Proxy to the Presentation Layer.
Finally, the Presentation Layer displays the newly created complaint to the Participant.

User Sees his Communications Page

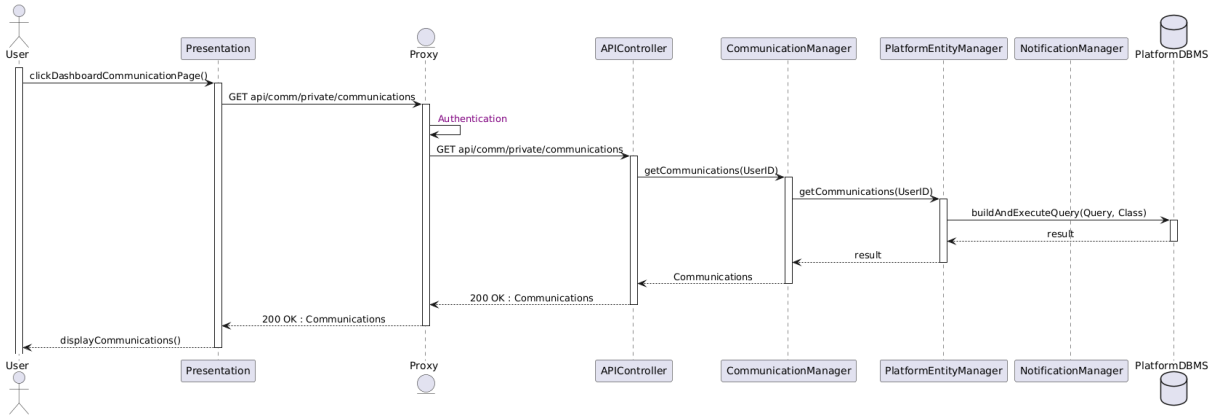

Figure 29: User Communications Page Sequence Diagram

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

When the User clicks the "Dashboard Communication Page" button, the Presentation Layer sends a GET private API call to the Proxy.
The Proxy validates the User's authentication token and forwards the request to the APIController. The APIController triggers the CommunicationManager to fetch the list of communications associated with the UserID.
The CommunicationManager sends a query to the PlatformEntityManager, which interacts with the PlatformDBMS to retrieve the requested data. Once the database returns the results, the PlatformEntityManager forwards them back to the CommunicationManager.
The CommunicationManager passes the fetched communications to the APIController, which returns a 200 OK response to the Proxy, including the list of communications. The Proxy forwards this response to the Presentation Layer.
Finally, the Presentation Layer displays the list of communications to the User.

University Interrupts an Ongoing Internship

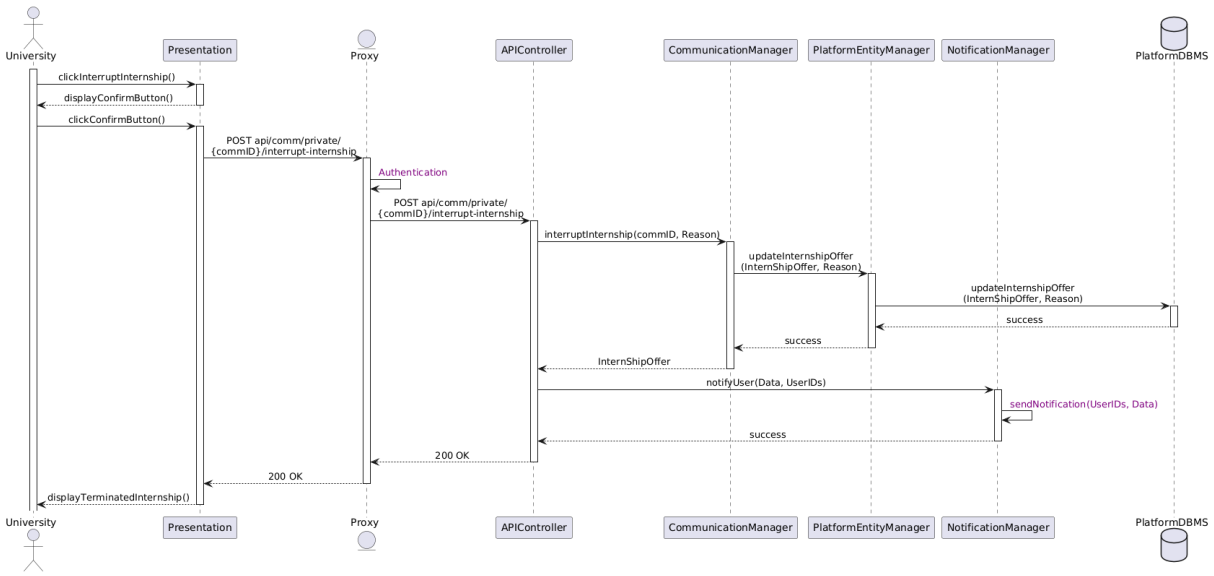

Figure 30: University Interrupts Internship Sequence Diagram

When the University initiates the process to interrupt an internship by clicking the appropriate button, the Presentation Layer displays a confirmation button. Upon confirmation, the Presentation Layer sends a POST request to the Proxy.
The Proxy authenticates the request and forwards it to the APIController. The APIController invokes the CommunicationManager, which interacts with the PlatformEntityManager to update the internship offer in the database. The PlatformEntityManager performs the update operation by communicating with the PlatformDBMS.
After successfully updating the internship data, the CommunicationManager returns the updated internship details to the APIController. The APIController then triggers the NotificationManager to notify relevant users about the interruption. Once the notifications are sent, a success response propagates back through the Proxy to the Presentation Layer.
Finally, the Presentation Layer displays the confirmation of the terminated internship to the University.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

User Terminates Communication

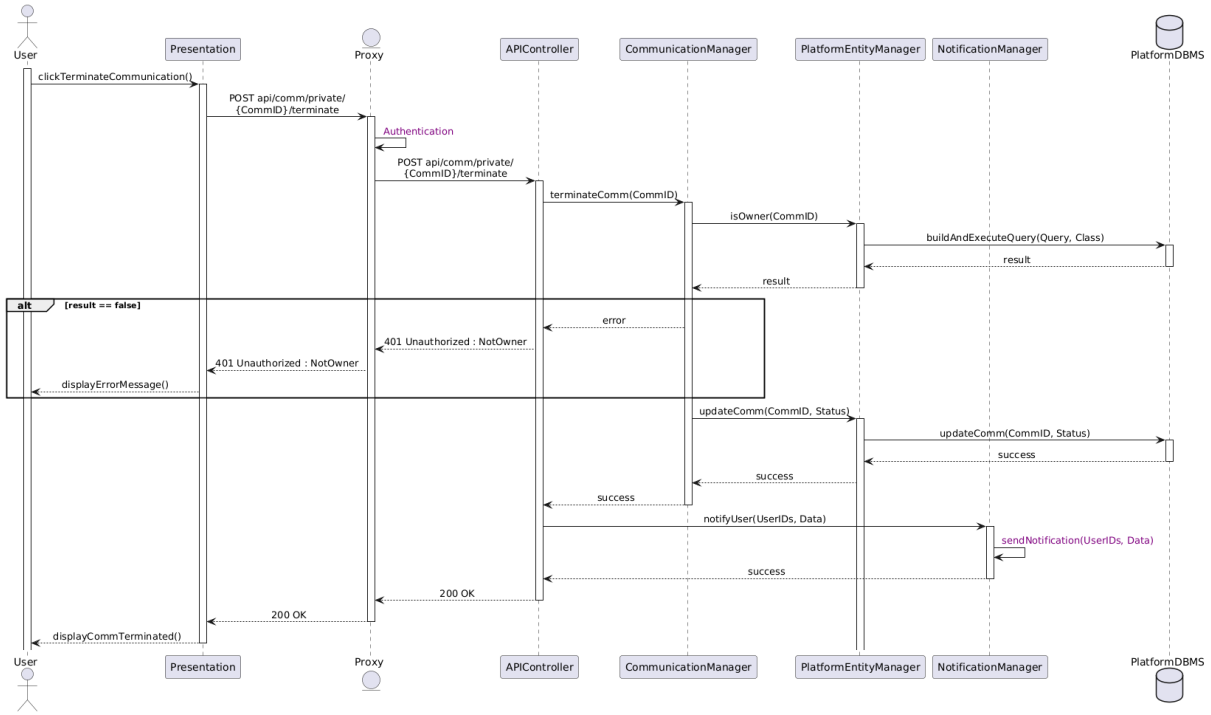

Figure 31: User Terminates Communication Sequence Diagram

When the User clicks the "Terminate Communication" button, the Presentation Layer sends a POST private API call to the Proxy.
The Proxy authenticates the request and forwards it to the APIController. The APIController calls the CommunicationManager to terminate the specified communication (CommID).
The CommunicationManager first verifies if the User is the owner of the communication by querying the PlatformEntityManager. The PlatformEntityManager checks the ownership in the PlatformDBMS and returns the result.
If the User is not the owner (result == false), an error message (401 Unauthorized: NotOwner) is returned through the chain to the Presentation Layer, which displays an error message to the User.
If the User is the owner, the CommunicationManager updates the communication's status to "terminated" by interacting with the PlatformEntityManager, which performs the update in the PlatformDBMS. Upon success, the update is confirmed back up the chain.
The APIController then triggers the NotificationManager to notify all relevant users about the termination of the communication. The NotificationManager sends notifications and confirms their delivery.
Finally, the APIController sends a 200 OK response to the Proxy, which forwards it to the Presentation Layer. The Presentation Layer displays a confirmation message to the User indicating that the communication has been successfully terminated.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Company Sends an Internship Position Offer

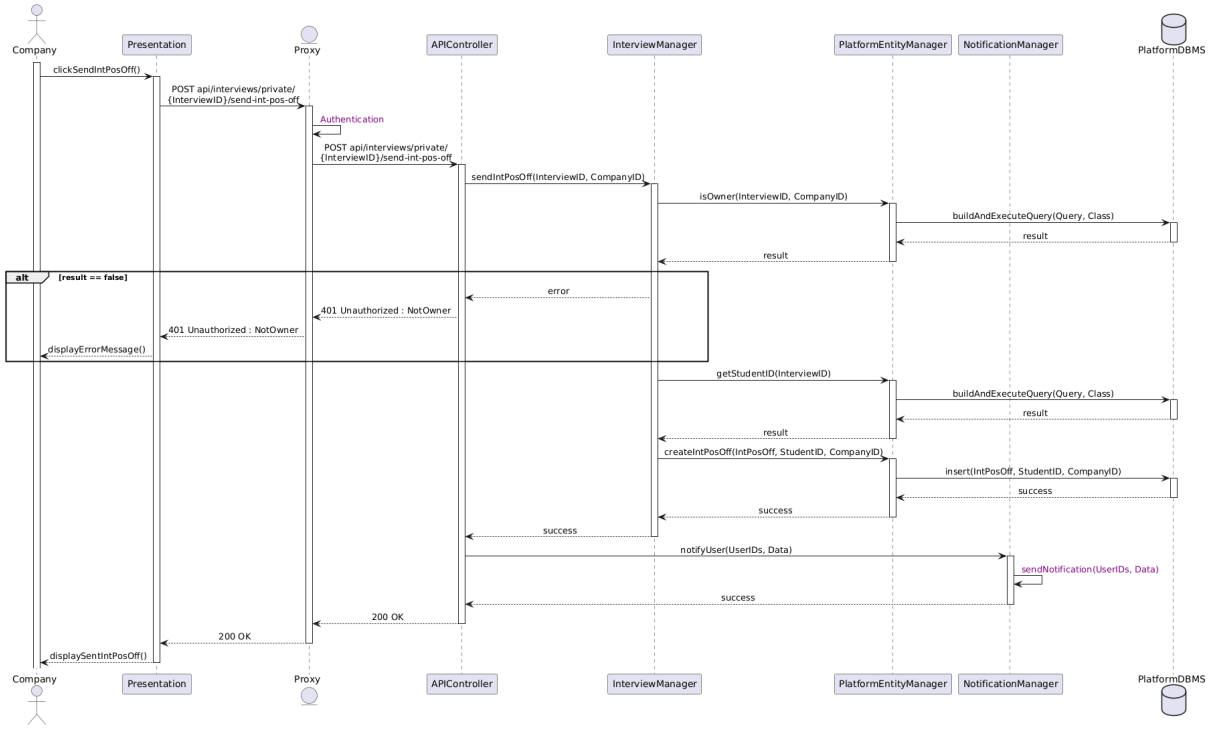

Figure 32: Company Sends Internship Position Offer Sequence Diagram

When the Company initiates the process to send an internship position offer by clicking the relevant button, the Presentation Layer sends a POST request to the Proxy.

The Proxy authenticates the request and forwards it to the APIController. The APIController invokes the InterviewManager, which first verifies if the Company is the owner of the specified interview by querying the PlatformEntityManager. The PlatformEntityManager communicates with the PlatformDBMS to perform the ownership check.

If the Company is not the owner, an error is returned through the APIController, Proxy, and Presentation Layer, displaying an error message to the Company. If ownership is confirmed, the InterviewManager retrieves the Student ID related to the interview and creates a new internship position offer by interacting with the PlatformEntityManager. The PlatformEntityManager inserts the offer into the database via the PlatformDBMS.

After successfully creating the internship position offer, the InterviewManager returns a success response to the APIController. The APIController then triggers the NotificationManager to notify relevant users about the new offer. Once notifications are sent, the success response propagates back through the Proxy to the Presentation Layer.

Finally, the Presentation Layer displays confirmation of the sent internship position offer to the Company.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Student Accepts Internship Position Offer

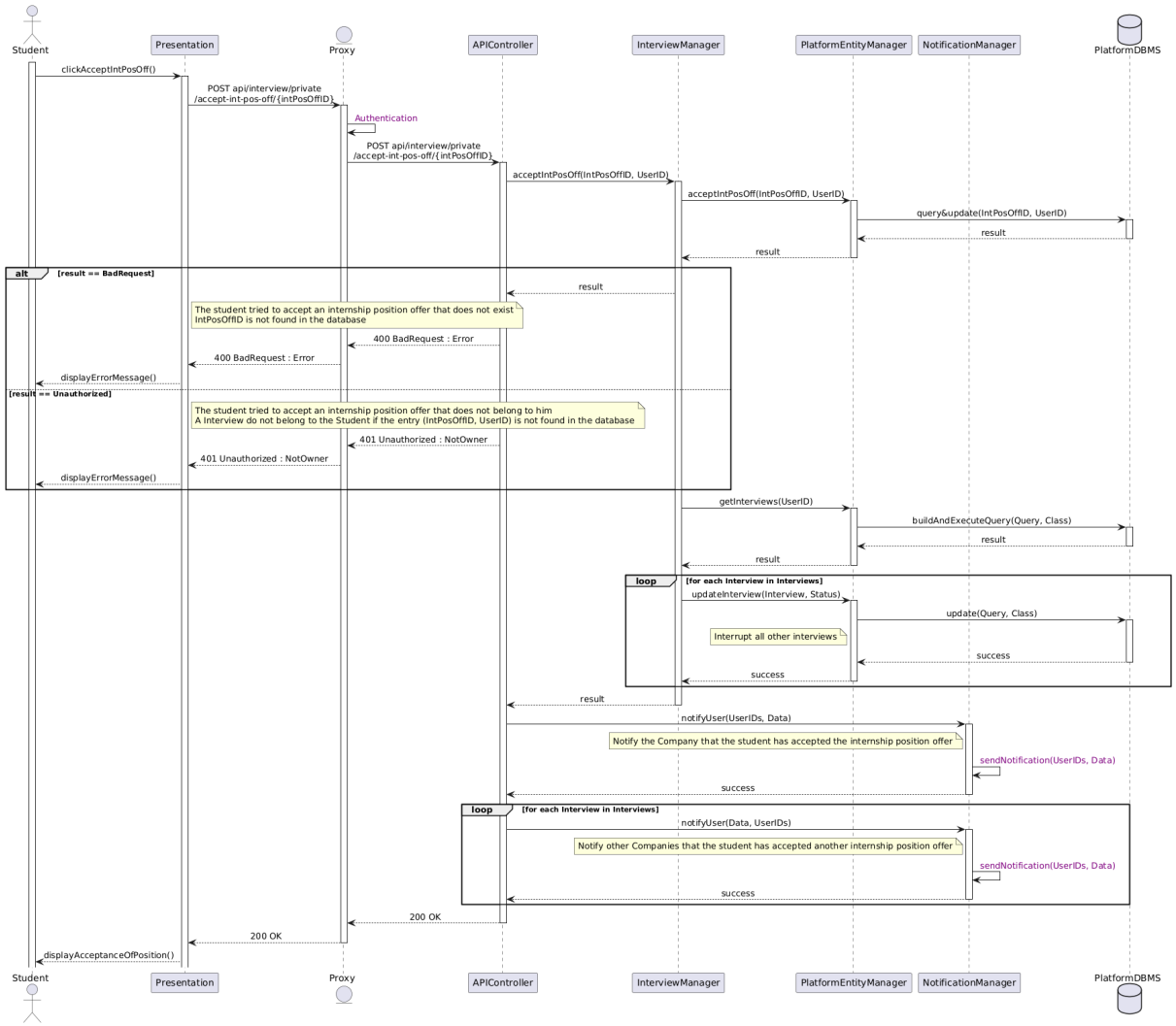

Figure 33: Student Accepts Internship Position Offer Sequence Diagram

When a Student clicks to accept an internship position offer, the Presentation Layer sends a POST request to the Proxy.
The Proxy authenticates the request and forwards it to the APIController. The APIController invokes the InterviewManager to handle the acceptance process. The InterviewManager checks the ownership of the offer by interacting with the PlatformEntityManager, which queries the PlatformDBMS. If the offer does not exist (BadRequest) or does not belong to the Student (Unauthorized), an error response is returned to the Presentation Layer, and an error message is displayed.
If the ownership is validated, the InterviewManager retrieves all interviews associated with the Student by querying the PlatformEntityManager. For each of these interviews, the InterviewManager interrupts them by updating their status in the PlatformDBMS.
Once all other interviews are updated, the InterviewManager confirms the acceptance of the internship position offer. The APIController triggers the NotificationManager to notify the relevant Company about the Student's acceptance. Additional notifications are sent to other Companies, informing them that the Student has accepted a different offer.
Finally, a success response is returned through the Proxy to the Presentation Layer, which displays a confirmation message to the Student.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Company Sends Template Interview

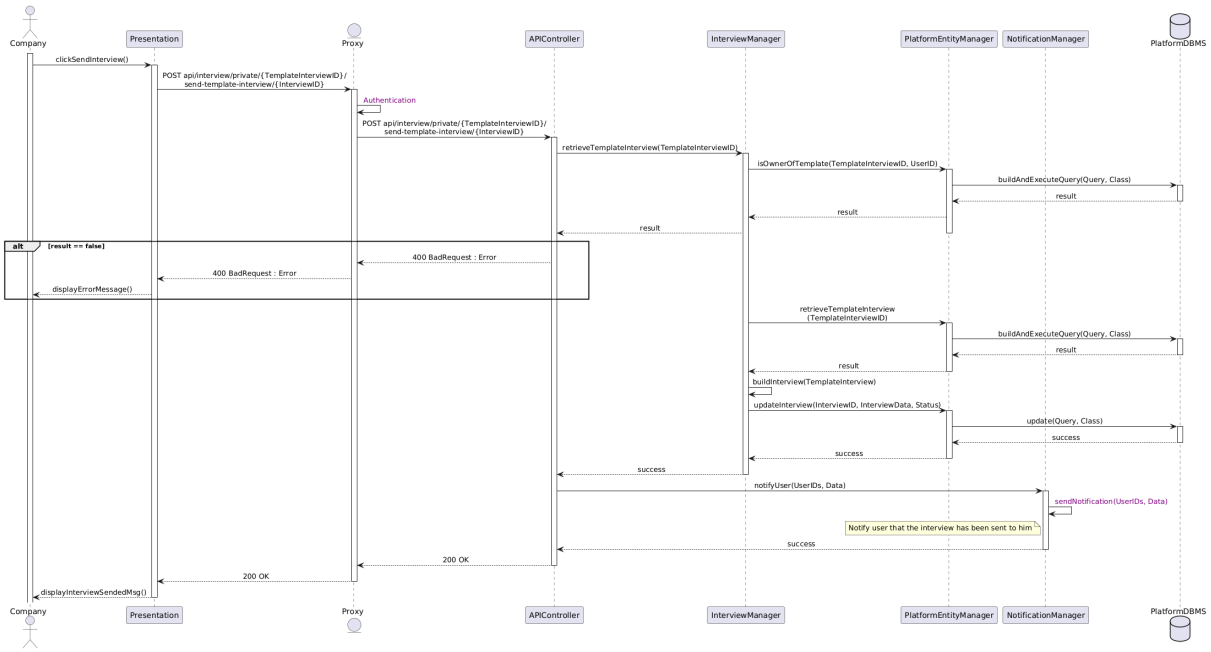

Figure 34: Company Sends Template Interview Sequence Diagram

When a Company clicks to send a saved interview, the Presentation Layer sends a POST request containing the InterviewID and TemplateInterviewID to the Proxy.
The Proxy authenticates the request and forwards it to the APIController. The APIController invokes the InterviewManager to retrieve the template interview. The InterviewManager checks if the Company owns the specified template by querying the PlatformEntityManager, which interacts with the PlatformDBMS. If the ownership check fails, an error response is returned, and an error message is displayed to the Company.
If the ownership check succeeds, the InterviewManager retrieves the template data from the database and constructs the interview based on the template. The updated interview data is then stored in the database by the PlatformEntityManager.
After the interview is successfully updated, the APIController triggers the NotificationManager to notify the Student that the interview has been sent. A success response is then returned through the Proxy to the Presentation Layer, which displays a confirmation message to the Company.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Company Closes Internship Offer

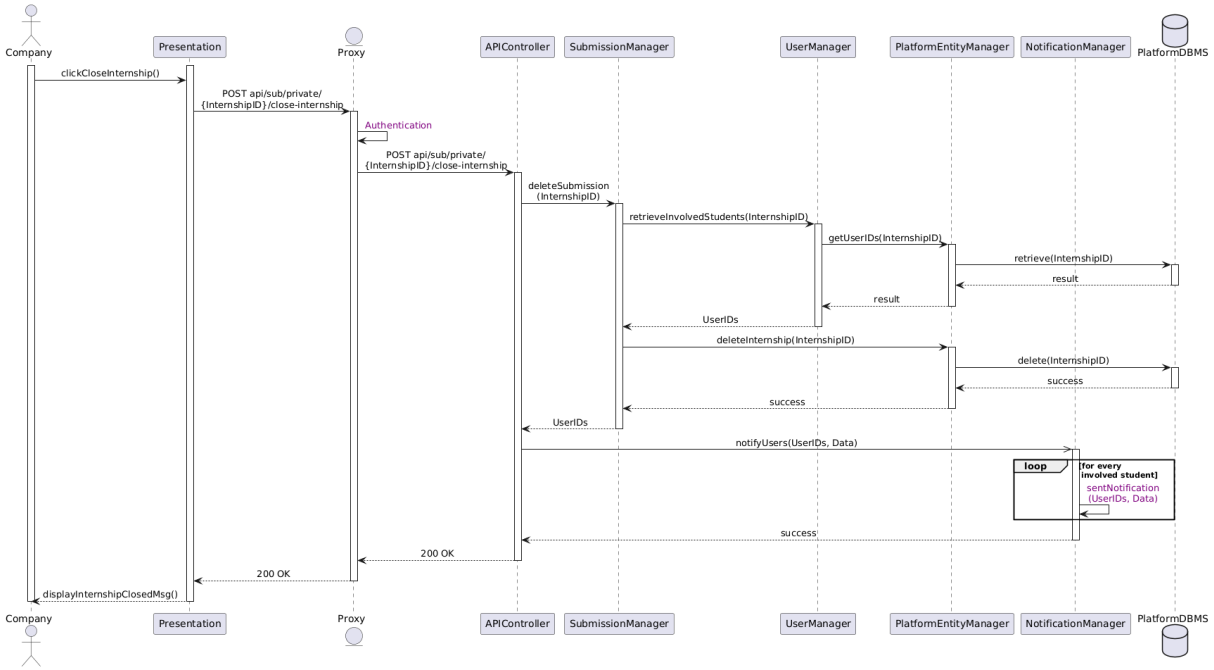

Figure 35: Company Closes Internship Offer Sequence Diagram

When a Company clicks the "Close Internship" button, the Presentation Layer sends a POST private API call to the Proxy. The request contains the InternshipID.
The Proxy authenticates the request and forwards it to the APIController. The APIController calls the SubmissionManager to delete the internship submission associated with the InternshipID.
The SubmissionManager retrieves the user IDs of all students involved in the internship by querying the UserManager. The UserManager interacts with the database through the PlatformEntityManager and PlatformDBMS to fetch the relevant user IDs.
Once the user IDs are retrieved, the SubmissionManager deletes the internship from the database using the PlatformEntityManager and PlatformDBMS.
After successfully deleting the internship, the APIController triggers the NotificationManager to notify all involved students about the closure of the internship. Notifications are sent for each user involved in the internship.
Finally, a 200 OK response is returned through the Proxy to the Presentation Layer, which displays a confirmation message.

2.5 Component Interfaces

In this section, we will outline the interfaces of each component of the S&C platform. We will detail the methods and parameters they expose, as well as the data they return. The first image provides a general overview of the components, resembling the component view but with a focus on the interfaces between them. The second image offers a more detailed representation of the interfaces within the Platform Logic components, while the third image highlights the interfaces exposed and utilized by the Notification Manager. In these images, we aimed to illustrate each method exposed by every component. This is why the number of methods depicted here is greater than in the sequence diagrams, where some methods were combined into a single one due to space constraints.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

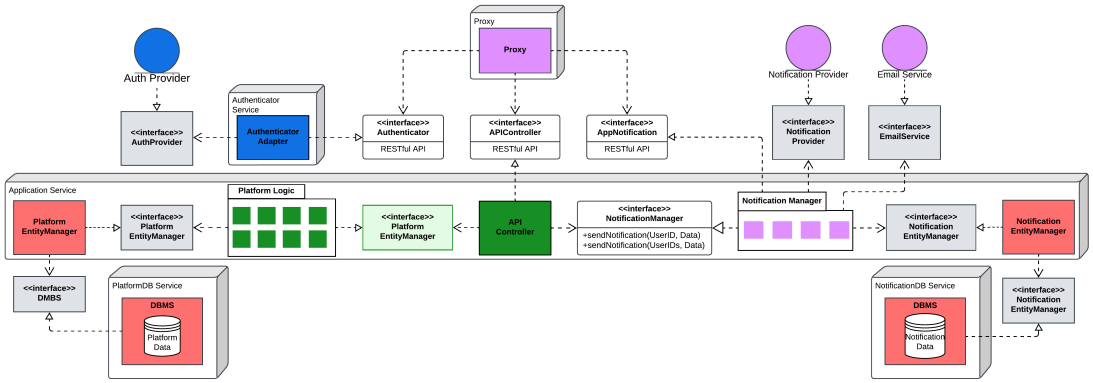

Figure 36: Components Interfaces Overview

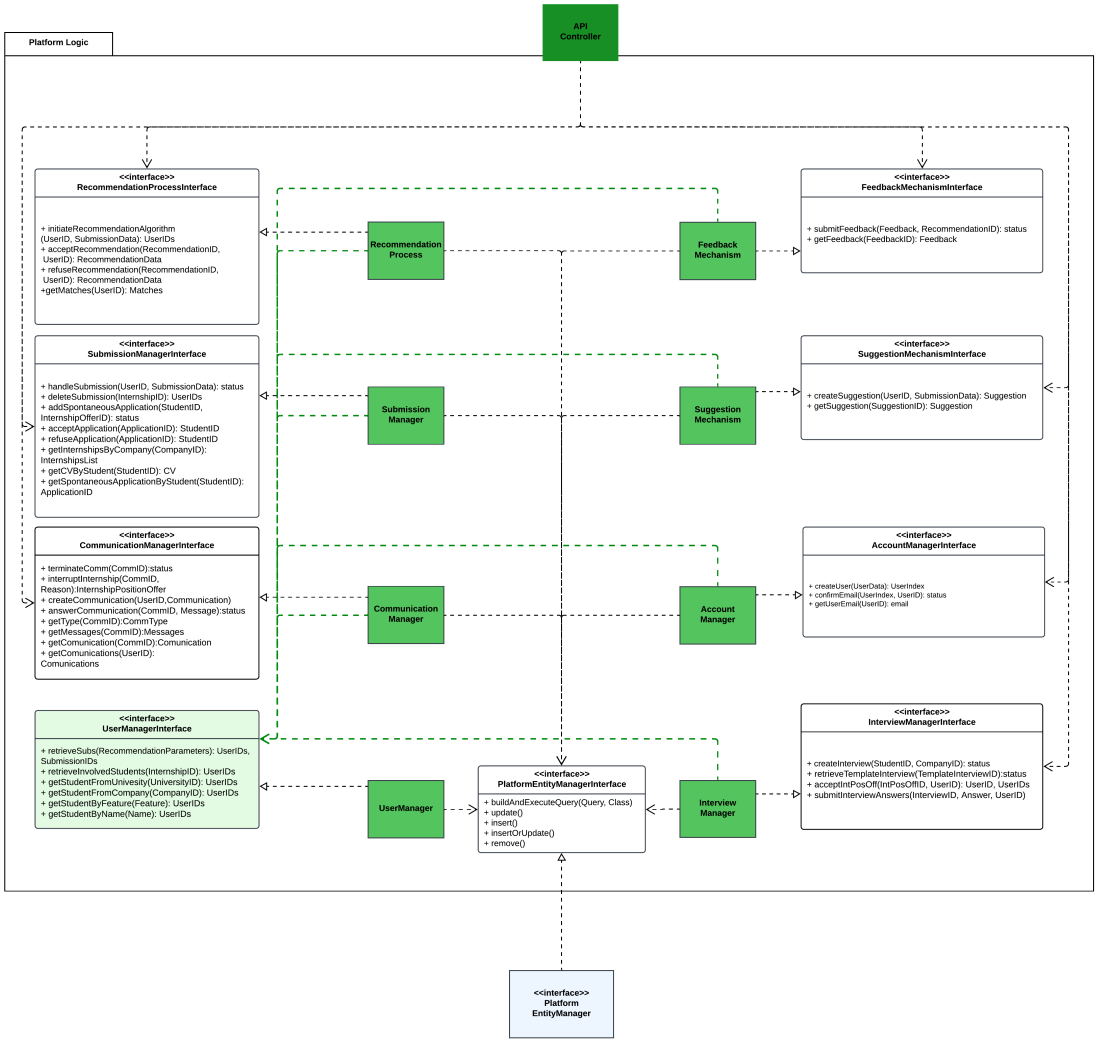

Figure 37: Platform Logic Components Interfaces

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

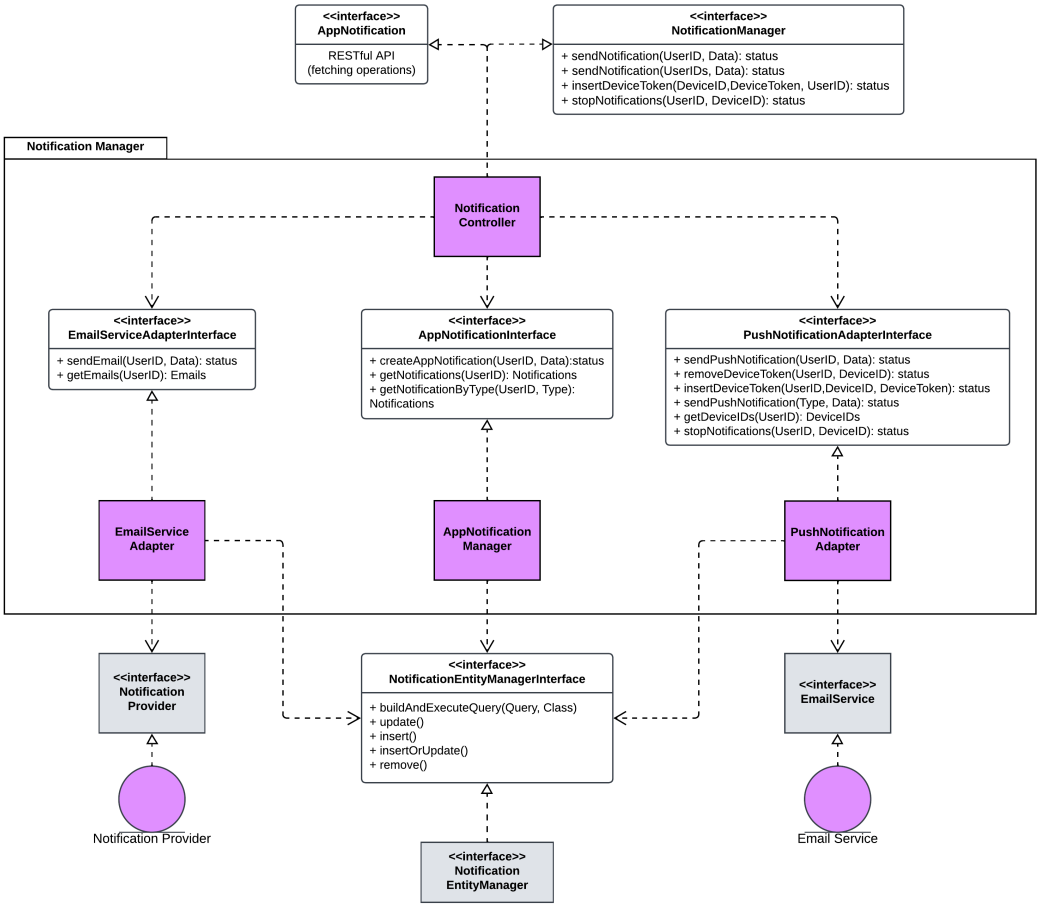

Figure 38: Notification Manager Components Interfaces

2.5.1 RESTful API endpoints

Proxy Endpoints

All the calls are routed to the Proxy that handles authentication redirecting private calls to the Authenticator service. That means that, in a sense, all the endpoints are Proxy endpoints. The Token to let the user authenticate is added to the header of the each private request. The Proxy will route the call to the Authenticator service that will validate the token and return the UserID to the Proxy that will then route the call to the right service. However there are some calls to which the Proxy add middleware endpoints that don't involve the simple token validation procedures, common to all the private calls. Those particular calls are the ones that follows:

POST api/auth/login

Request Body: UserCredentials : Object

Responses:

- 201 Created: Token : Object
- 400 Bad Request: InvalidError : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

Call Stack: Proxy -> api/auth/validate-credentials -> api/auth/validate-token

---

POST api/auth/create-token
Request Body: UserCredentials : Object
Responses:

- 201 Created: Token : Object
- 400 Bad Request: InvalidError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

Call Stack: Proxy -> api/auth/insert-credentials -> api/auth/generate-token

Application Endpoints

These calls are routed by the Proxy to the Application service that handles the business logic of the application. If the call has the private keyword in his address then the Proxy routes it to `api/auth/validate` middleware to validate the token. That means that every private call shall contain the Token Object in its body.

POST api/account/public/register
Request Body: UserData : Object
Responses:

- 201 Created: UserIndex : Object
- 400 Bad Request: InvalidError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/notify/private/send-conf-email
Request Body: UserIndex : Object
Responses:

- 201 Created: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

POST api/notify/private/conf-email
Responses:

- 200 OK
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/sub/private/update-sub
Request Body: SubmissionData : Object
Responses:

- 201 Created: SubmissionID, Suggestions : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

GET api/sub/private/internships/{CompanyID}
Responses:

- 200 OK: InternshipsList : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

GET api/sub/private/cv/{StudentID}
Responses:

- 200 OK: CV : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- 500 Internal Server Error: InternalServerError : Object

POST api/recommendations/private/{RecommendationID}/accept
Responses:

- 201 Created:
  - askFeedback : Object
  - Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

PUT api/feedback/private/{RecommendationID}/submit
Request Body: Feedback : Object
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/sub/private/application/{CompanyID}
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/comm/private/application/{ApplicationID}/accept
Responses:

- 200 OK: Message : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/interviews/private/send-answer/{InterviewID}
Request Body: Answer : Object
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/interviews/private/send-interview/{InterviewID}
Request Body: InterviewTemplate : Object
Responses:

- 201 Created: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/interviews/private/save-template-interview/{InterviewID}
Request Body: InterviewTemplate : Object
Responses:

- 201 Created: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

---

POST api/interviews/private/{TemplateInterviewID}/send-template-interview/{InterviewID}
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/interviews/private/evaluate-interview/{InterviewID}
Request Body: Evaluation : Object
Responses:

- 201 Created: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

GET api/applications/private/get-spontaneous-applications
Responses:

- 200 OK: Applications : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

GET api/applications/private/get-matches
Responses:

- 200 OK: Matches
- 400 Bad Request: InvalidError
- 401 Unauthorized: UnauthorizedError
- 500 Internal Server Error: InternalServerError

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

---

POST api/comm/private/{commID}/answer
Request Body: Message : Object
Responses:

- 201 Created: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

GET api/comm/private/communications/{CommID}
Responses:

- 200 OK: Result : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/comm/private/create
Request Body: Communication : Object
Responses:

- 201 Created: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

GET api/comm/private/communications
Responses:

- 200 OK: Communications : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

---

POST api/comm/private/{commID}/interrupt-internship
Request Body: Reason : Object
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/comm/private/{commID}/terminate
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
  - NotOwner
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/interviews/private/{InterviewID}/send-int-pos-off
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
  - NotOwner
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/interview/private/accept-int-pos-off/{intPosOffID}
Responses:

- 200 OK: Message : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
    - NotOwner
- 409 Conflict: ConflictError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/notify/private/new-device-token
Request Body: DeviceToken, DeviceID : Object
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/sub/private/{InternshipID}/close-internship
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

Authenticator Endpoints

These calls are routed by the Proxy to the Authenticator service that handles the authentication and token generation.

POST api/auth/insert-credentials
Request Body: UserCredentials : Object
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 409 Conflict: ConflictError : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- 500 Internal Server Error: InternalServerError : Object

POST api/auth/validate-credentials
Request Body: UserCredentials : Object
Responses:

- 200 OK: Message : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/auth/generate-token
Request Body: UserCredentials : Object
Responses:

- 201 Created: Token : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
- 500 Internal Server Error: InternalServerError : Object

POST api/auth/refresh-token
Request Body: RefreshToken : Object
Responses:

- 201 Created: Token : Object
- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
  - RefreshTokenExpired
  - InvalidToken
- 500 Internal Server Error: InternalServerError : Object

GET api/auth/validate
Responses:

- 200 OK: UserID : Object

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

- 400 Bad Request: InvalidError : Object
- 401 Unauthorized: UnauthorizedError : Object
  - TokenExpired
  - InvalidToken
- 500 Internal Server Error: InternalServerError : Object

2.6 Selected Architectural Styles and Patterns

In this subsection, we will describe the architectural styles and patterns that have been selected for the S&C platform, how they work, and the reason behind their choice from a non-technical point of view.

- **Microservices Architecture**: The S&C platform is designed as a set of loosely coupled services, each responsible for a specific set of functionalities. This architecture allows for a high degree of modularity, scalability, and fault tolerance by enabling the independent development, deployment, and scaling of each service and the division of the platform into smaller, more manageable components that can be developed by different teams.
- **RESTful API**: The platform's services communicate with the front-end through a RESTful API, which provides a standardized method for different services to interact with one another. This API utilizes the stateless HTTP protocol using methods such as *GET*, *POST*, *PUT*, and *DELETE* to perform CRUD (Create, Read, Update, and Delete) operations on the data stored by the platform. By being stateless, the API allows for better scalability and reliability, as it eliminates the requirement for a client to connect to the same server for every request. This is because each request is independent and does not rely on or maintain context from previous requests.
- **Lightweight Client-Server Architecture**: The platform follows a lightweight client-server architecture, where the client (front-end) is only responsible for handling the Presentation layer, consisting of the visualization of the data and the retrieval of user inputs. The server (back-end) is responsible for the implementation of the Business Logic and the Data Layer, where the former is responsible for the processing of the data, computation, and the logic of the platform, while the latter is responsible for the storage and retrieval of the data.
  It is important to note that while this architecture seems to follow the 3-tier architecture pattern, each tier is not necessarily composed of or limited to a single server, but it can be composed of multiple servers and containers, following more of a microservices architecture.

2.7 Other Design Decisions

In this final part of the architectural design chapter, we will describe all the other design decisions that have been made for the S&C platform, such as the choice of the DBMS, the notification handling, the authentication and validation process, and the scalability of the platform.

Database Management

For the Platform database, we have chosen to use a relational database management system (DBMS) to store the data. This choice was made because the data structure of the platform is well-suited to a relational database, as it consists of structured data with clear relationships between different entities and the typology of query and analysis make on the data did not justify the complexity or the performance

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

cost of a OLAP database. The DBMS will be used to store all the data required by the platform including but not limited to: user information, internship offers, CVs, and recommendations. The data will be structured in a way that allows for efficient querying and retrieval, ensuring that the platform can provide fast and reliable access to the information stored in the database through the Entity Manager interface and the JPA framework.

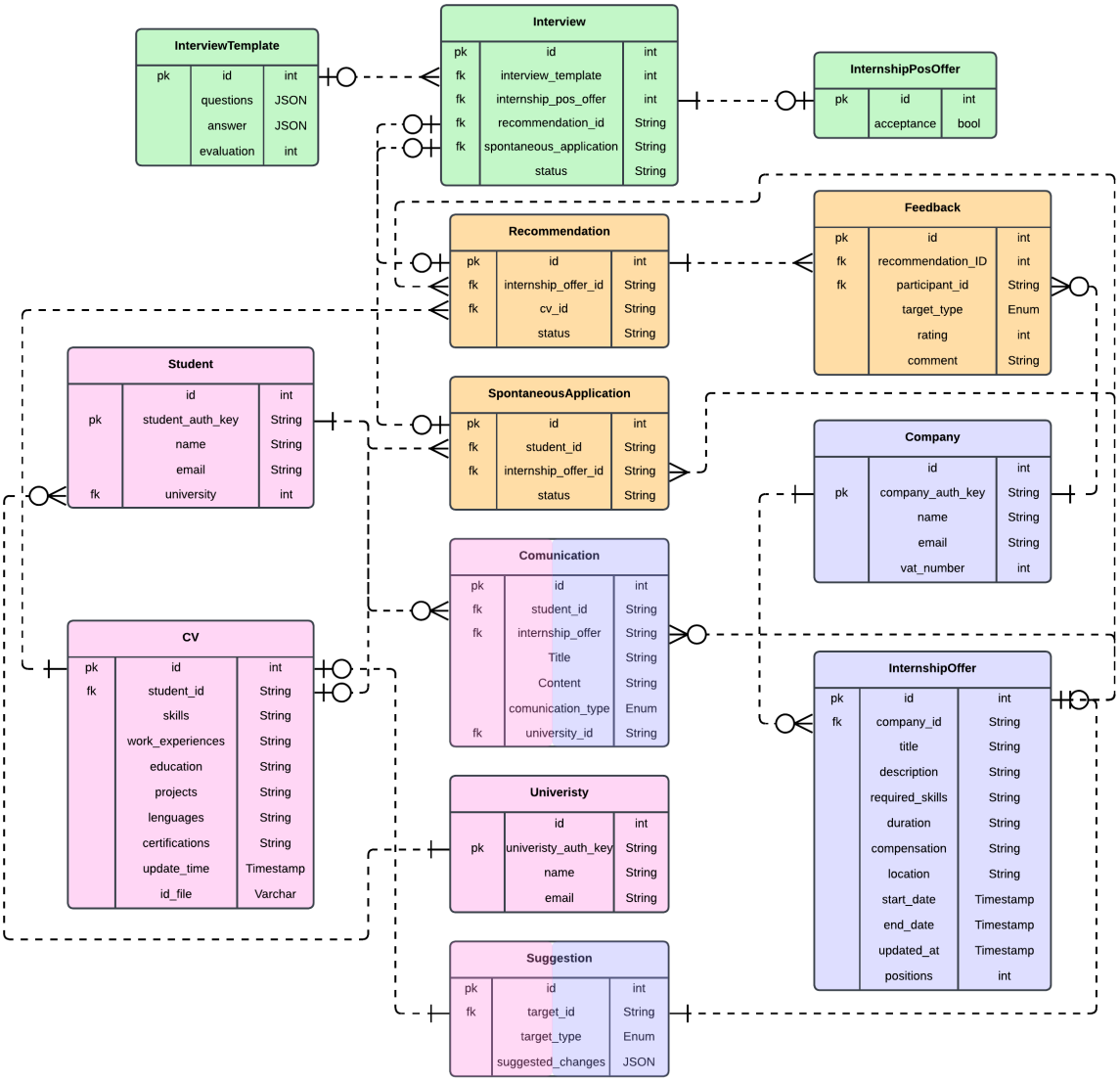

Figure 39: Platform Database Schema

Here we have the initial idea of the database structure, representing the main entities and their relationships as seen in the sequence diagram. This structure is subject to change as the development of the platform progresses and the need for new entities or relationships arises. Please note that we have not included the Notification DBMS in this diagram, as it is a separate entity from the Platform DBMS and is used exclusively by the Notification Manager. Moreover, the type used in each entity field is not specified according to a strict DBMS standard , instead more general and descriptive types, such as String, JSON, and Enum, have been used intentionally to better convey the concept and structure of the data, making it easier to understand the purpose of each field.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Notification Handling

Notifications of all type, both in-app and push, are handled by the Notification Manager component who is responsible to send notifications and sign-up mail confirmation to all users of the platform. This component acts both as an adapter for external push notification and email service providers and as a controller for in-app notification fetch requests. The Notification Manager is designed to do this by providing a simple interface *sendNotification* that can be called to send a notification to one or more users. This is done to hide the complexity of the underlying notification system, show in sequence diagrams Sequence diagram *(fig:13) Send Notification* and Sequence Diagram *(fig:9) RequestDeviceToken*.
Due to the nature of this component and the high level of work it has to do, we expect it to be one of the first to be exported to its own container in the future to handle scalability concerns effectively. By decoupling the Notification Manager into its own container, the platform can better allocate resources to this high-demand service, ensuring consistent performance even under heavy load. This is why the Notification Manager in the Architecture Components Diagram *(fig:2)* is represented as a separate entity from the other components, having its own interface and DB, even though it could be considered part of the Platform Logic.

Authentication and Validation

The user authentication process is handled by the Authenticator service, which is responsible for validating user credentials and generating tokens for authenticated users. The reason behind the use of a token-based authentication system is done because we want to limit the number of times a user has to enter their credentials while ensuring that only logged-in users can interact with *private* API call.
The token is generated by the Authenticator service and is stored in the Presentation layer after the user login, and it is sent every time a private request is made. The token is validated by the Authenticator service for every request, look at the Sequence Diagram *(fig:8)Authentication*, validating and refreshing the token or requesting a new login by the user, ensuring that only recent authenticated users can access the platform's functionalities.

2.7.1 Scalability

The S&C platform is designed from the beginning to be scalable, meaning that it can handle an increasing number of users and requests without compromising performance. This is achieved through the use of a microservices architecture, which allows the platform to be divided into smaller, more manageable services that can be independently scaled as needed. Each service can be deployed in its own container, which can be easily replicated to handle additional load. This ensures that the platform can scale horizontally by adding more containers to distribute the load across multiple instances of the service. Additionally, the use of a Stateless RESTful API ensures that each request is independent and does not rely on or maintain context from previous requests, allowing the platform to scale more easily and reliably without the need for complex session management.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

3 User Interface Design

The Presentation Node serves the static files and scripts to run a single page application (SPA) on the client's browser. The SPA web interface allows a wide number of interaction without the need of refreshing the page, providing a smoother user experience. The platform root page is the Home Page from which every non-registered user can find information about S&C such as the latest news. The Home page is linked to other pages, such as the Dashboard, Contacts, and About page, using an app bar. In the Contacts page, the user can find useful links to get in touch with S&C.

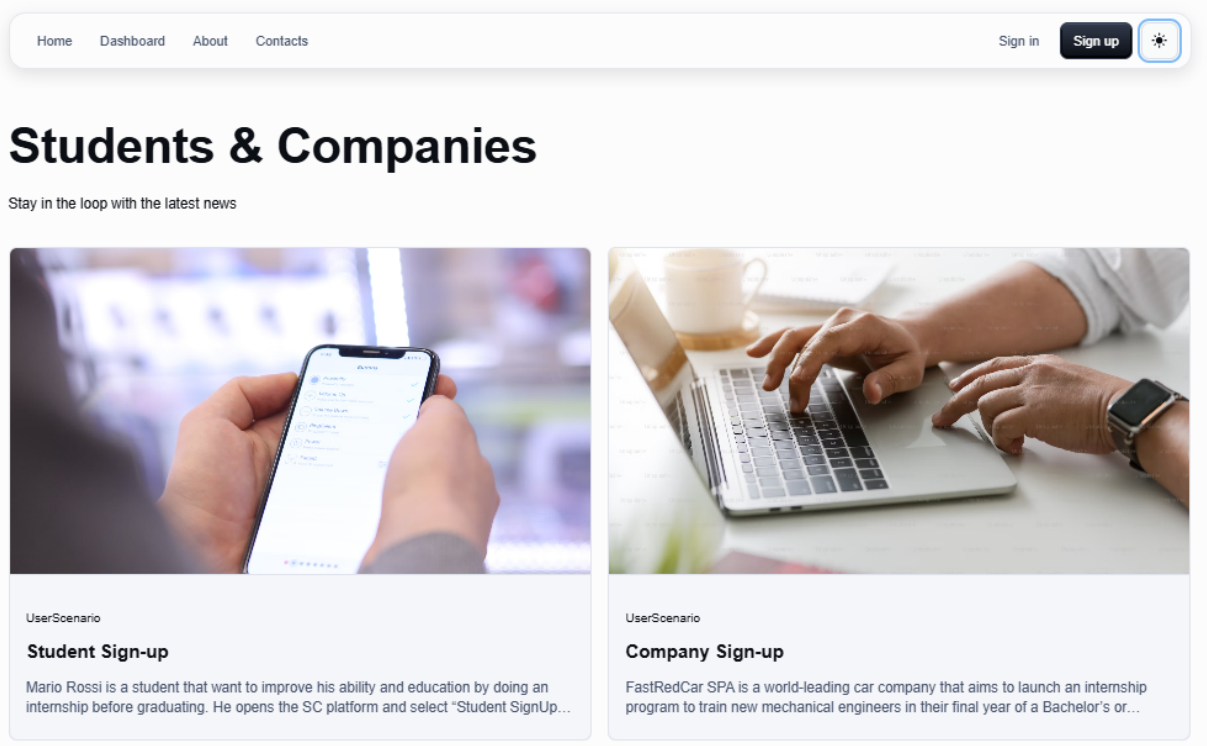

Figure 40: UI Home Page: as an example, UI cards containing some user scenarios described in the **??** are shown

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

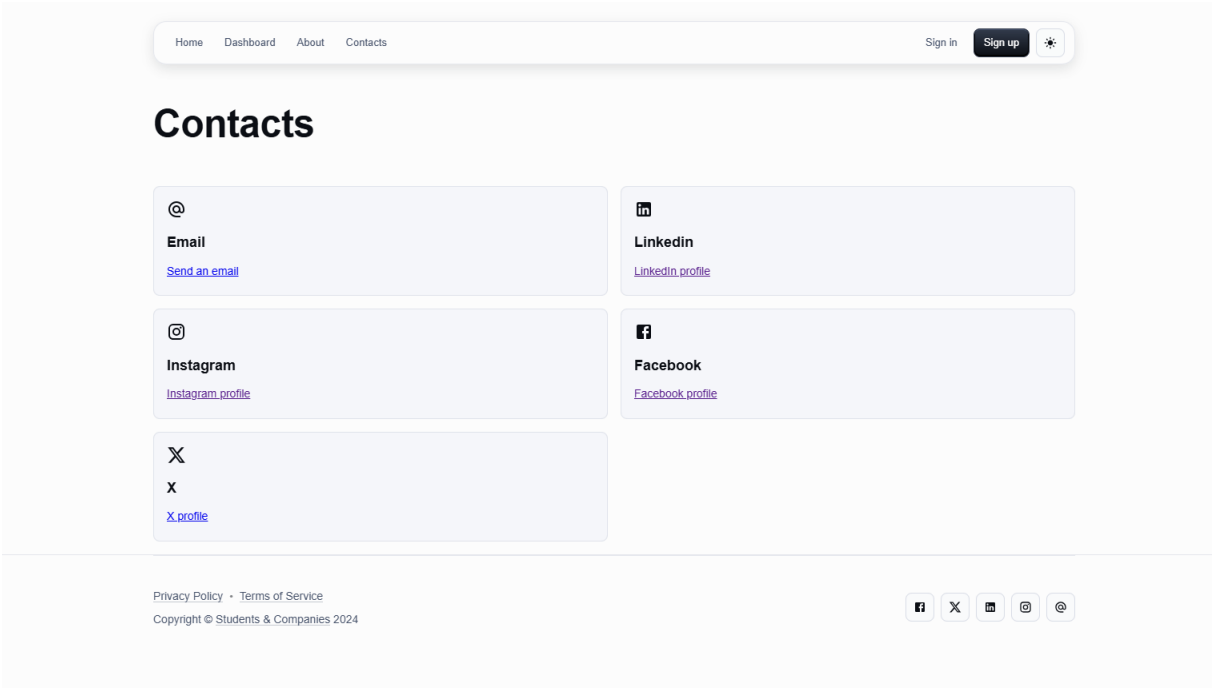

Figure 41: UI Contacts Page

Thanks to the app bar link buttons, users can also reach the Sign-Up and Sign-in pages. The Sign-Up page allows for different types of sign-up according to the new user type. This allows the user to provide the platform with the correct information.

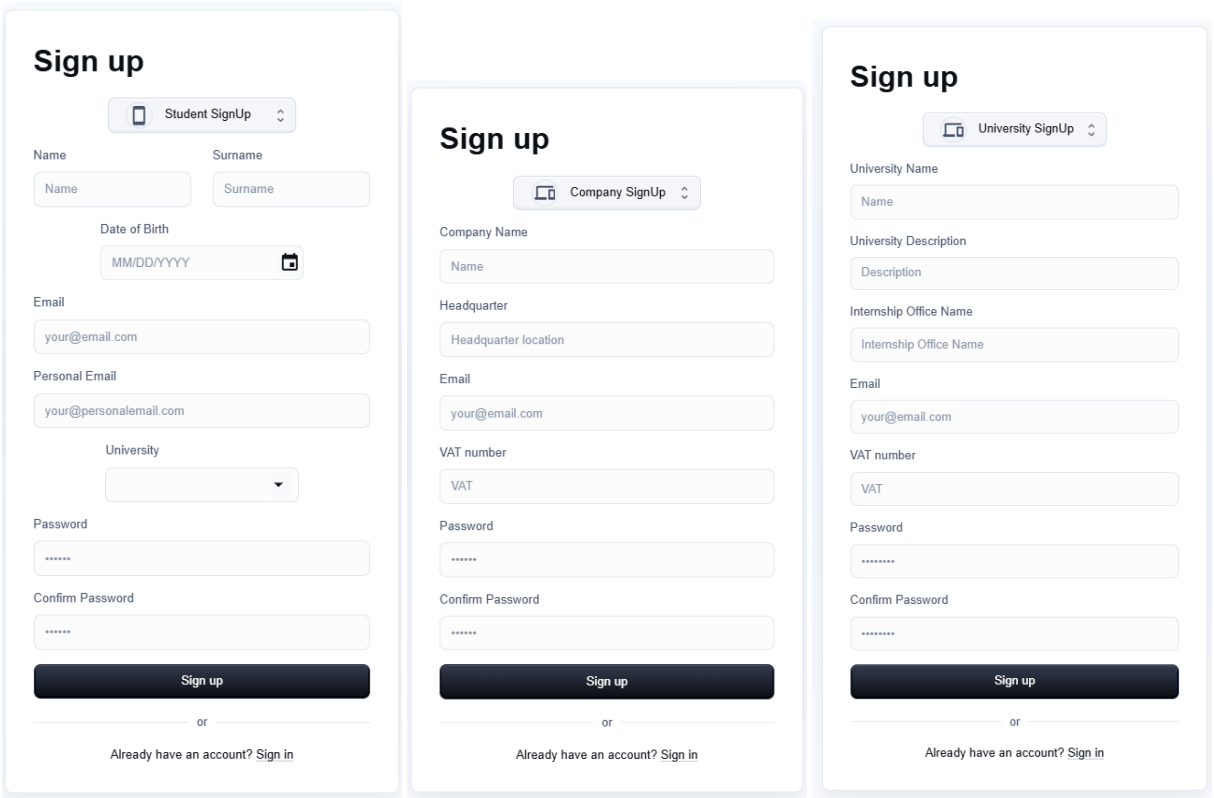

Figure 42: UI Sign-Up Page

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

To be able to log into the platform, the user shall provide his email and password.

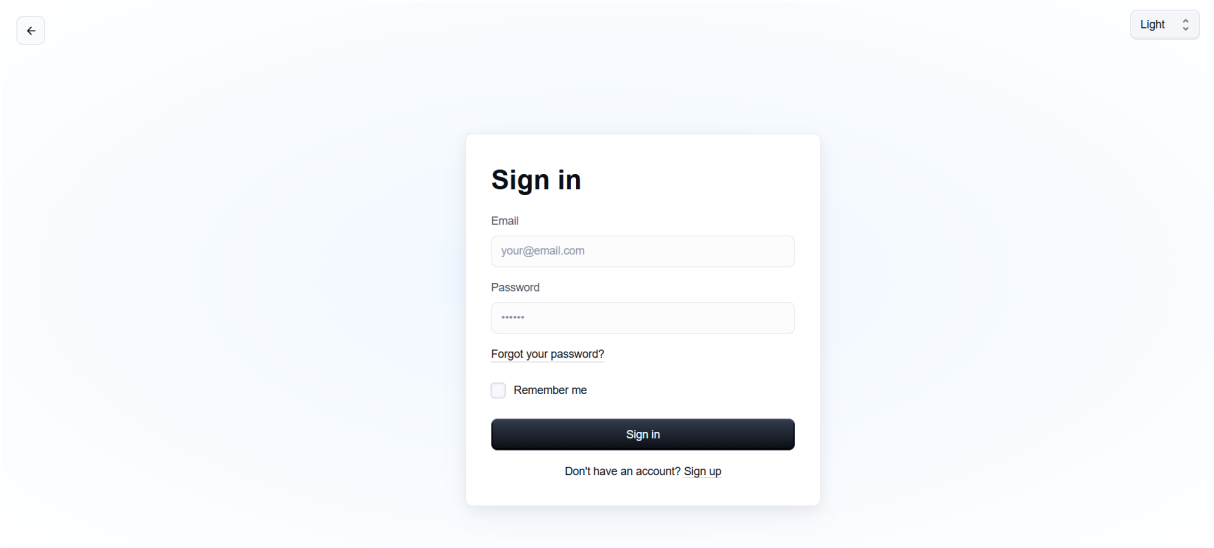

Figure 43: UI Sign-In Page

The Dashboard page will be the central hub for logged-in users. From the left-hand panel of the dashboard, all the pages associated with the core functionalities of the platform are reachable. Therefore, this page is provided after a successful log-in. The side panel contains different elements according to the user needs.

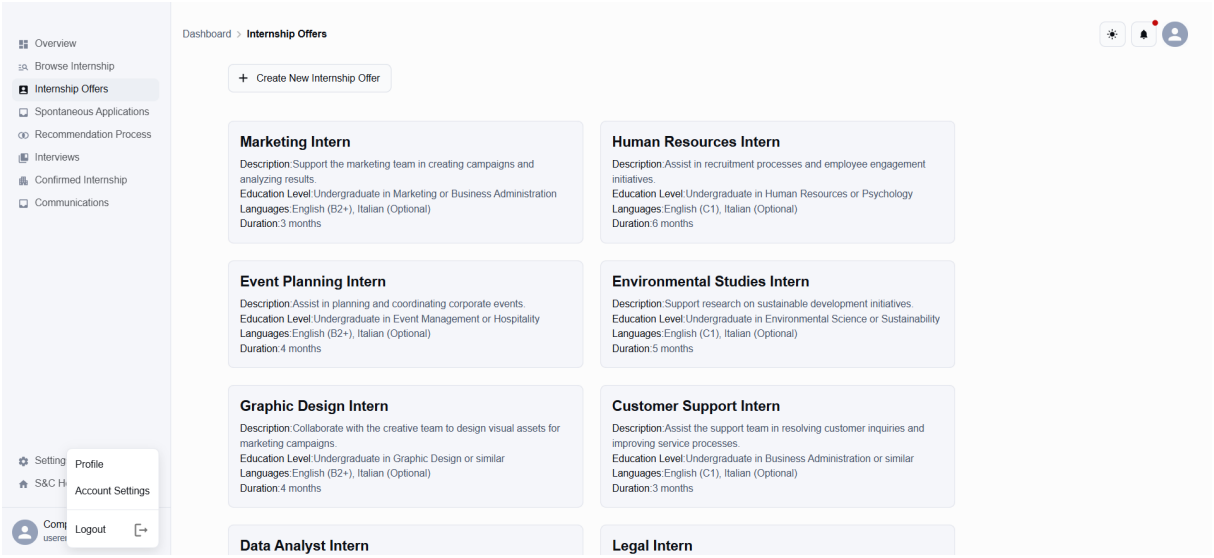

Figure 44: UI Company Dashboard Page

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

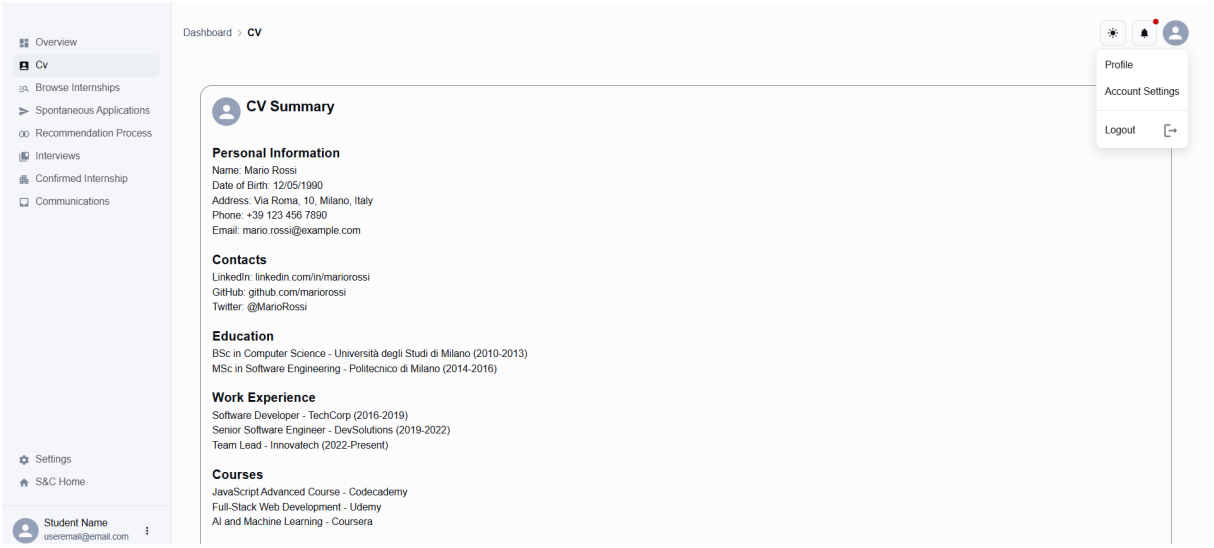

Figure 45: UI Students Dashboard Page

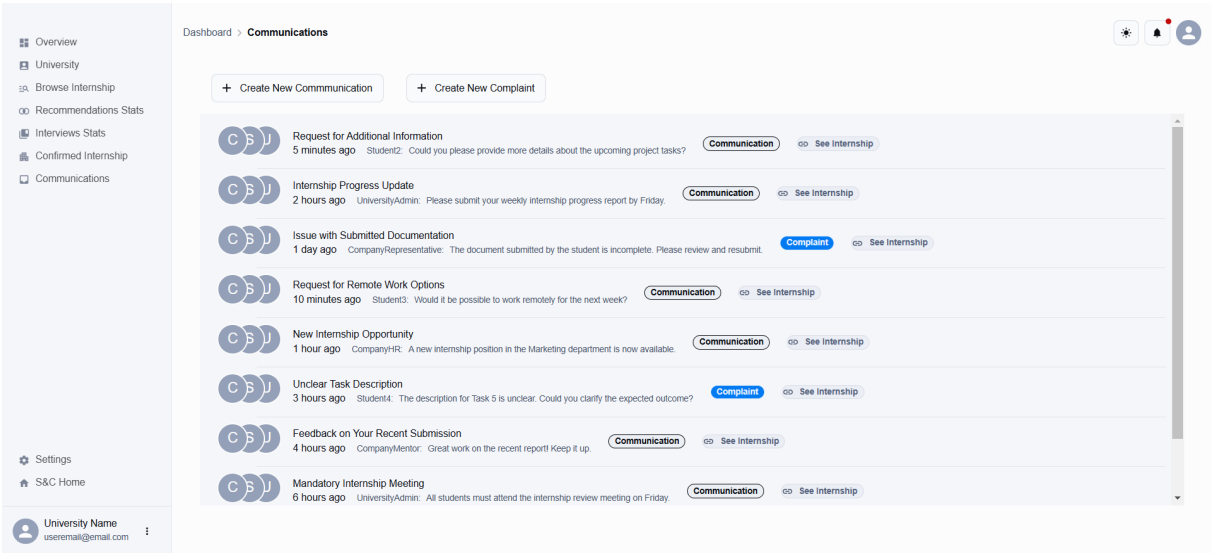

Figure 46: UI University Dashboard Page

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

4 Requirements Traceability

Internal Components

[C1] API Controller

[C2] Account Manager

[C3] User Manager

[C4] Recommendation Process

[C5] Submission Manager

[C6] Interviews Manager

[C7] Communication Manager

[C8] Feedback Mechanism

[C9] Suggestion Mechanism

[C10] Platform Entity Manager

[C11] Authenticator Adapter

[C12] Notification Manager

[C13] Notification Entity Manager

External Components

[E1] Notification Provider

[E2] Email Service API

[E3] Authentication Provider

[E4] DBMS

[R1]	**The platform shall allow any unregistered students to register by providing personal information and selecting their University.**
[C1]	API Controller
[C2]	Account Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[C12]	Notification Manager
[C13]	Notification Entity Manager
[E1]	Notification Provider
[E2]	Email Provider API
[E3]	Authentication Provider
[E4]	DBMS

Table 8: Requirement R1: Traceability for Student Registration Process

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R2]	**The platform shall allow any companies to register by providing company information.**
[C1]	API Controller
[C2]	Account Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[C12]	Notification Manager
[C13]	Notification Entity Manager
[E1]	Notification Provider
[E2]	Email Provider API
[E3]	Authentication Provider
[E4]	DBMS

Table 9: Requirement R2: Traceability for Company Registration Process

[R3]	**The platform shall allow any universities to register by providing university information.**
[C1]	API Controller
[C2]	Account Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[C12]	Notification Manager
[C13]	Notification Entity Manager
[E1]	Notification Provider
[E2]	Email Provider API
[E3]	Authentication Provider
[E4]	DBMS

Table 10: Requirement R3: Traceability for University Registration Process

[R4	**The platform shall allow Users to log in using their email and password.**
[C3]	User Manager
[C11]	Internal Authenticator Adapter
[E3]	Authentication Provider

Table 11: Requirement R4: Traceability for User Login Functionality

[R5]	**The platform shall send notifications to Users when relevant events occur.**
[C12]	Notification Manager
[C13]	Notification Entity Manager
[E1]	Notification Provider
[E4]	DBMS

Table 12: Requirement R5: Traceability for Notification Functionality

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R6]	**The platform shall allow Companies to create and publish Internship offers specifying details.**
[C1]	API Controller
[C3]	User Manager
[C5]	Submission Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 13: Requirement R6: Traceability for Internship Offer Creation

[R7]	**The platform shall allow Companies to terminate their Internship offers at their own discretion.**
[C1]	API Controller
[C3]	User Manager
[C5]	Submission Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 14: Requirement R7: Traceability for Internship Termination

[R8]	**The platform shall provide Students with Matches automatically obtained by the Recommendation Process.**
[C1]	API Controller
[C3]	User Manager
[C4]	Recommendation Process
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 15: Requirement R8: Traceability for Recommendation Matching

[R9]	**The platform shall allow Students to view and navigate all available Internships.**
[C1]	API Controller
[C5]	Submission Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 16: Requirement R9: Traceability for Viewing Available Internships

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R10]	**The platform shall enable Students to submit Spontaneous Applications to Internships they choose.**
[C1]	API Controller
[C5]	Submission Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 17: Requirement R10: Traceability for Spontaneous Applications

[R11]	**The platform shall allow Students to submit their CV.**
[C1]	API Controller
[C5]	Submission Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 18: Requirement R11: Traceability for CV Submission

[R12]	**The platform shall allow Students to modify their CV.**
[C1]	API Controller
[C5]	Submission Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 19: Requirement R12: Traceability for CV Modification

[R13]	**The platform shall allow Students to monitor the status of their Spontaneous Applications.**
[C1]	API Controller
[C5]	Submission Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 20: Requirement R13: Traceability for Monitoring Spontaneous Applications

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R14]	**The platform shall allow Students to monitor the status of their Recommendation.**
[C1]	API Controller
[C4]	Recommendation Process
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 21: Requirement R14: Traceability for Monitoring Recommendations

[R15]	**The platform shall display to Students all the Internships found by the Recommendation Process.**
[C1]	API Controller
[C4]	Recommendation Process
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 22: Requirement R15: Traceability for Displaying Recommended Internships

[R16]	**The platform shall display to Companies all the CVs of Matched Students obtained by the Recommendation Process.**
[C1]	API Controller
[C4]	Recommendation Process
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 23: Requirement R16: Traceability for Displaying Matched Student CVs

[R17]	**The platform shall allow Students and Companies to accept a Recommendation.**
[C1]	API Controller
[C4]	Recommendation Process
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 24: Requirement R17: Traceability for Accepting Recommendations

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R18]	**The platform shall allow Companies to accept a Spontaneous Application.**
[C1]	API Controller
[C4]	Recommendation Process
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 25: Requirement R18: Traceability for Accepting Spontaneous Applications

[R19]	**The platform shall start a Selection Process only if both the Company and the Student have accepted the Recommendation.**
[C1]	API Controller
[C4]	Recommendation Process
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 26: Requirement R19: Traceability for Starting Selection Process via Recommendation

[R20]	**The platform shall start a Selection Process only if the Company has accepted the Spontaneous Application.**
[C1]	API Controller
[C5]	Submission Manager
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 27: Requirement R20: Traceability for Starting Selection Process via Spontaneous Application

[R21]	**The platform shall allow Companies to create Interviews.**
[C1]	API Controller
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 28: Requirement R21: Traceability for Creating Interviews

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R22]	**The platform shall allow Companies to submit Interviews to Students they have initiated a Selection Process with.**
[C1]	API Controller
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 29: Requirement R22: Traceability for Submitting Interviews

[R23]	**The platform shall allow Students to answer Interview questions and submit them.**
[C1]	API Controller
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 30: Requirement R23: Traceability for Answering and Submitting Interviews

[R24]	**The platform shall allow Companies to manually evaluate Interview submissions.**
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 31: Requirement R24: Traceability for Evaluating Interview Submissions

[R25]	**The platform shall allow Students and Companies to monitor the status of their Interviews.**
[C1]	API Controller
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 32: Requirement R25: Traceability for Monitoring Interview Status

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R26]	**The platform shall enable Companies to complete the Interview process by submitting the final outcome to each candidate.**
[C1]	API Controller
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 33: Requirement R26: Traceability for Submitting Interview Outcomes

[R27]	**The platform shall enable Companies to send an Internship Position Offer to a Student only if he previously passed the relative Interview.**
[C1]	API Controller
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 34: Requirement R27: Traceability for Sending Internship Position Offers

[R28]	**The platform shall enable Students to accept or reject an Internship Position Offer sent by a Company only if he previously passed the relative Interview.**
[C1]	API Controller
[C6]	Interviews Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 35: Requirement R28: Traceability for Accepting or Rejecting Internship Position Offers

[R29]	**The platform shall collect Feedback from both Students and Companies regarding the Recommendation Process.**
[C1]	API Controller
[C8]	Feedback Mechanism
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 36: Requirement R29: Traceability for Collecting Feedback on Recommendation Process

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R30]	**The platform shall provide Suggestions to Students on improving their CVs.**
[C1]	API Controller
[C9]	Suggestion Mechanism
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 37: Requirement R30: Traceability for Providing CV Improvement Suggestions

[R31]	**The platform shall provide Suggestions to Companies on improving Internship descriptions.**
[C1]	API Controller
[C9]	Suggestion Mechanism
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 38: Requirement R31: Traceability for Improving Internship Descriptions

[R32]	**The platform shall allow registered Universities to access and monitor Internship Communications related to their Students.**
[C1]	API Controller
[C7]	Communication Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 39: Requirement R32: Traceability for Monitoring Internship Communications by Universities

[R33]	**The platform shall provide a dedicated space for Students and Companies to exchange Communications about the current status of an Ongoing Internship.**
[C1]	API Controller
[C7]	Communication Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 40: Requirement R33: Traceability for Communication Space for Ongoing Internships

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

[R34]	**The platform shall allow registered Universities to handle Complaints and to interrupt an Internship at their own discretion.**
[C1]	API Controller
[C7]	Communication Manager
[C10]	Platform Entity Manager
[C11]	Authenticator Adapter
[E3]	Authentication Provider
[E4]	DBMS

Table 41: Requirement R34: Traceability for Handling Complaints and Interrupting Internships

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

The following matrix provides a compact overview of the previous tables that facilitates the identification of the key components that manage the core functions of the system.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	E1	E2	E3	E4
R1	X	X								X	X	X	X	X	X	X	X
R2	X	X								X	X	X	X	X	X	X	X
R3	X	X								X	X	X	X	X	X	X	X
R4			X								X					X	
R5												X	X	X			X
R6	X		X		X					X	X					X	X
R7	X		X	X						X	X					X	X
R8	X		X							X	X					X	X
R9	X				X					X	X					X	X
R10	X				X					X	X					X	X
R11	X				X					X	X					X	X
R12	X				X					X	X					X	X
R13	X				X					X	X					X	X
R14	X			X						X	X					X	X
R15	X			X						X	X					X	X
R16	X			X						X	X					X	X
R17	X			X						X	X					X	X
R18	X			X						X	X					X	X
R19	X			X						X	X					X	X
R20	X				X	X				X	X					X	X
R21	X					X				X	X					X	X
R22	X					X				X	X					X	X
R23	X					X				X	X					X	X
R24	X					X				X	X					X	X
R25	X					X				X	X					X	X
R26	X					X				X	X					X	X
R27	X					X				X	X					X	X
R28	X									X	X					X	X
R29	X							X		X	X					X	X
R30	X								X	X	X					X	X
R31	X								X	X	X					X	X
R32	X						X			X	X					X	X
R33	X						X			X	X					X	X
R34	X						X			X	X					X	X

Table 42: Requirements Traceability Matrix

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

5 Implementation, Integration and Test Plan

This section provides a detailed plan for the implementation, integration, and testing of the S&C platform. The first section will describe the main concepts and ideas behind the Implementation, Integration and Test plan and the following sections will provide a description for each step of it.

5.1 Plan Overview

The platform will be implemented, integrated and tested following a mix between a *thread* and *bottom-up* approach thanks to the aforementioned microservices architecture that allows different service to be implemented and tested in parallel with others.
The main idea is to develop one feature of the Platform Logic at a time, whenever possible, by implementing the front-end user interface (UI) and the back-end corresponding components and the notifications they will generate. Each feature will undergo unit testing before being integrated with other components. By adopting this thread-based implementation, we can begin testing component integration early in the development process, instead of waiting for the entire Platform Logic to be completed. We believe that this approach allows us to identify and resolve integration issues at an earlier stage, minimizing the risk of significant problems arising later on.
However, due to dependency constraints, not all components can be developed using this thread-based approach. For example, the Recommendation Process and Suggestion Mechanism relay to the User Manager and Platform Entity Manager. To address this, the "Implementation, Integration and Test Plan" is organized into several steps, following a bottom-up approach dividing components into groups that can be developed, tested, and integrated independently.

5.2 Plan Stage

5.2.1 Stage 1: User Manager and Notification Manager

In the first stage, we will develop the User Manager and part of the Notification Manager, enabling each component to store and retrieve data from its respective database. This process will also involve developing the Platform Entity Manager and the Notification Entity Manager so that each can interact with its own database. To test these two components, we will create an API controller DRIVER and a Manager DRIVER to simulate API controller calls and the various invocations from other components within the Platform Logic.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

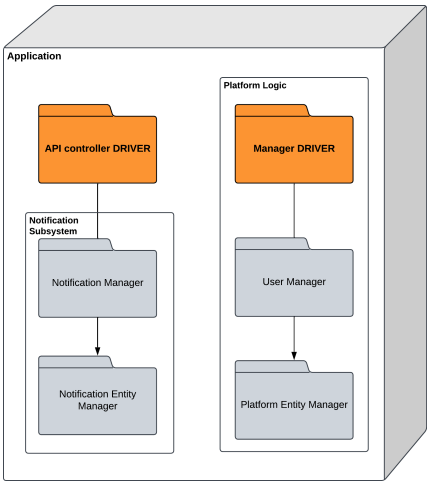

Figure 47: User Manager and Notification Manager testing

5.2.2 Stage 2: Platform Logic Components and User Interface

In the second stage, we will develop the remaining Platform Logic components, including the Recommendation Process, Mechanism components, and the remaining Managers. During this phase, we will also implement the User Interface (UI) and update the Notification Manager to handle notifications sent by these components, following the idea behind the *thread* approach. To test these back-end components, we will create an API Controller DRIVER to simulate their invocation, while creating a Rest API STUB to simulate the front-end calls.

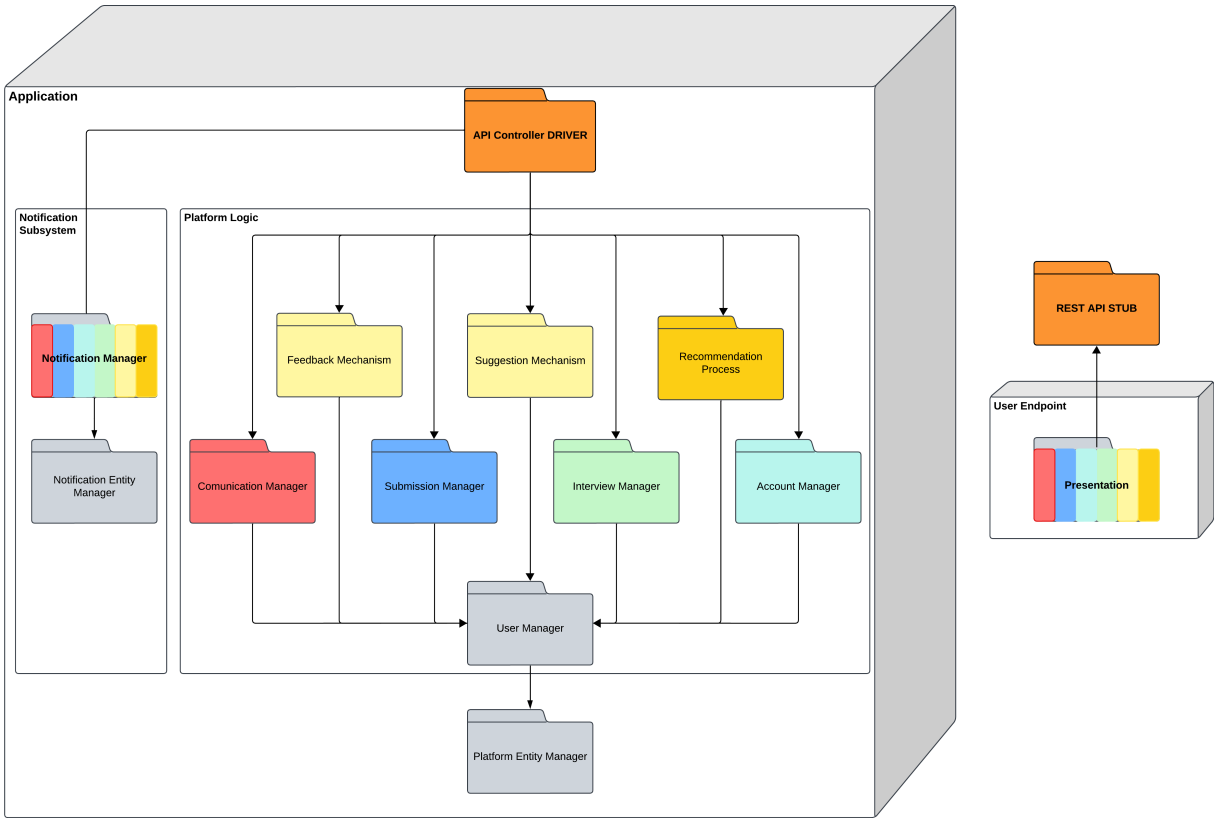

Figure 48: Platform Logic Components and User Interface testing

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

5.2.3 Stage 3: API Controller and Front-End Integration

In the third stage, we will develop and integrate the API Controller with the Platform Logic components, the Notification Manager, and the User Interface that should have been completed in the previous stages. This will allow us to test the API Controller and the Platform Logic components together, ensuring that they interact correctly and that the API Controller can handle requests from the front-end and route them to the appropriate components.
We will also develop the Authenticator Adapter to handle user authentication and token generation. For testing purpose we will create a Proxy DRIVER to simulate the API Controller and Authenticator Adapter calls and the front-end when it receives different responses.

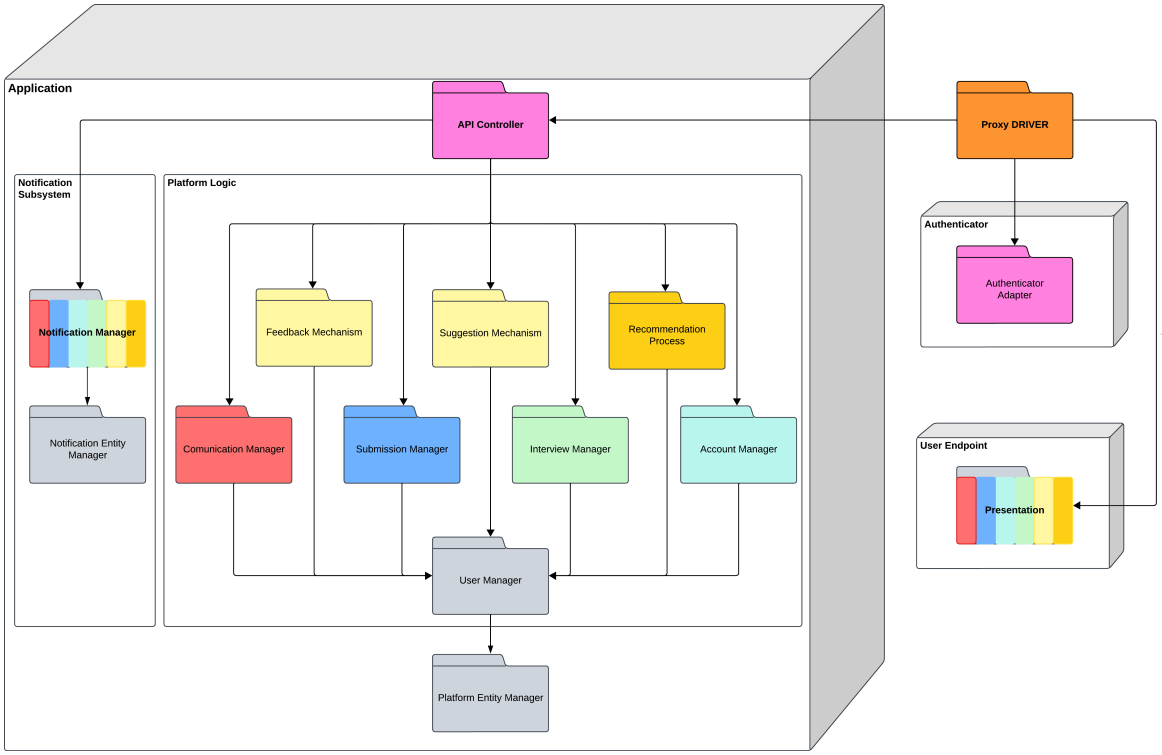

Figure 49: API Controller and Front-End Integration testing

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

5.2.4 Stage 4: Full Integration and Testing

In the final stage, we will integrate all components of the platform thanks to the development of the Proxy and the Authenticator Adapter. This allows us to test the platform as a whole, ensuring that all components work together as expected. We will also conduct end-to-end testing to verify that the platform meets all requirements and functions correctly.

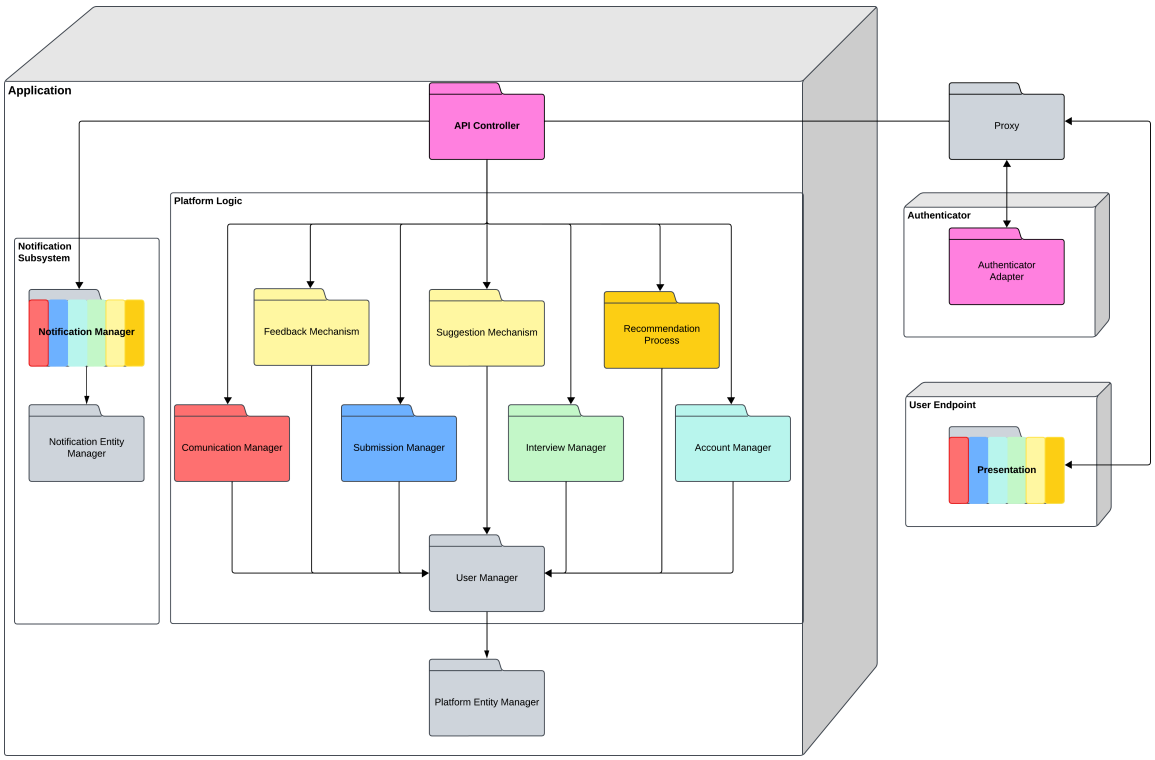

Figure 50: Full Integration and Testing

5.3 Technologies Used

In this last paragraph, we will describe the technologies used for the implementation, integration, and testing of the S&C platform, discussing the reasons behind their choice and how they will be used in the development process.

5.3.1 Implementation Technologies

- **Front-End**: The front-end of the platform will be developed using React, a popular JavaScript library for building user interfaces. React was chosen for its ease of use, flexibility, and performance, and wide support and documentation given the large community of developers that use it. The front-end will be styled using the MUI library to ensure a consistent and modern design across all pages and animated using the Framer Motion library.

- **Back-End**: The back-end of the platform will be developed using Java and Spring Boot, a popular framework for building Java-based web applications. Spring Boot was chosen robustness, and scalability, and the familiarity of the team with the Java language.

- **Database**: The platform's database will be established with MariaDB, a widely-used and open-source relational database management system. To engage with the database, we will utilize the

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

Java Persistence API (JPA) alongside Hibernate, a object-relational mapping (ORM) framework for Java applications, which enables us to handle the data without manually crafting SQL queries.

- **Authentication**: Firebase Authentication, a reliable and popular service from Google, will be utilized to implement the authentication system for the platform. Firebase Auth was selected due to its easy of integration, scalability, and the range of authentication options it offers, such as email/-password and logins via popular social networks like Facebook or Google.
  Moreover, Firebase Auth offers inherent security functionalities, including token-based authentication and secure user management, which correspond with our platform's requirement to manage sensitive information securely and efficiently

- **Notifications**: The platform's notification system will be built with Firebase Cloud Messaging (FCM), a cross-platform messaging service that enables us to deliver push notifications to users on Android, iOS, and the web. FCM was selected for its dependability, scalability, and seamless integration with Firebase Auth, enabling us to send notifications securely and efficiently to platform users.

5.3.2 Integration and Testing Technologies

- **Unit Testing**: The platform's components of the back end will be tested using JUnit, a popular unit testing framework for Java applications.
  Moreover, JUnit was chosen for its simplicity, ease of use, compatibility with the Spring Boot framework and the team's familiarity with the tool.

- **Back-end Mock**: Mockito, a widely-used mocking framework for Java applications, will be used to create stub and mock objects for testing purposes.
  Moreover, Mockito was selected for its flexibility, ease of use, and compatibility with JUnit, allowing us both to simulate the behavior of external dependencies and to verify the interactions between components.

- **Front-end Testing**: The front-end of the platform will be tested using React Testing Library, a popular testing utility for React applications.
  Moreover, React Testing Library was chosen for the same reasons as React, and it allows us to write tests that closely resemble how users interact with the application, ensuring that the UI functions as expected.

- **Front-end Mock**: MSW (Mock Service Worker) will be used to mock the API calls made by the front-end during testing.
  Moreover, MSW was chosen for its ease of use, flexibility, and compatibility with React Testing Library, enabling us to simulate the behavior of the back-end components and test the front-end in isolation.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

6 Effort Spent

Lorenzo Ricci

Section	Hours
Introduction	3
Architectural Design	20
User Interface Design	2
Requirements Traceability	1
Implementation, Integration, and Test Plan	2
Misc Activities	15

Table 43: Effort - Lorenzo Ricci

Matteo Giovanni Paoli

Section	Hours
Introduction	2
Architectural Design	24
User Interface Design	0.5
Requirements Traceability	8
Implementation, Integration, and Test Plan	4
Misc Activities	0

Table 44: Effort - Matteo Giovanni Paoli

Samuele Grisoni

Section	Hours
Introduction	6
Architectural Design	20
User Interface Design	1
Requirements Traceability	1
Implementation, Integration, and Test Plan	5
Misc Activities	3

Table 45: Effort - Samuele Grisoni

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

7 References

7.1 Used Tools

[1] Lucid Software Inc. *Lucidchart - Online Diagram Software*. Purpose: used to realize all the diagrams along the documents. 2024. URL: https://www.lucidchart.com.

[2] MermaidJS. *Mermaid - Markdownish Syntax for Generating Diagrams and Flowcharts*. Purpose: Used for generating diagrams and flowcharts directly from text or markdown-like syntax. 2024. URL: https://mermaid-js.github.io/.

[3] Meta (Facebook Inc.) *React - A JavaScript Library for Building User Interfaces*. Purpose: used to create the UI prototype. 2024. URL: https://reactjs.org.

[4] MUI Team. *Material-UI: React Components for Faster and Easier Web Development*. Purpose: used to create the UI prototype. 2024. URL: https://mui.com.

[5] PlantUML Team. *PlantUML - Create UML Diagrams from Plain Text*. Purpose: used to create UML diagrams throughout the documentation. 2024. URL: https://plantuml.com/.

7.2 Reference Tools

[6] React Testing Library Team. *React Testing Library - Simple and Complete Testing Utilities for React*. Purpose: used for testing React components. 2024. URL: https://testing-library.com/docs/react-testing-library/intro/.

[7] Framer Team. *Framer Motion - Production-Ready Motion Library for React*. Purpose: used for adding animations to the UI prototype. 2024. URL: https://www.framer.com/motion/.

[8] Spring Team. *Spring Boot - Framework for Building Java Applications*. Purpose: used for back-end development. 2024. URL: https://spring.io/projects/spring-boot.

[9] MariaDB Foundation. *MariaDB - Open Source Database*. Purpose: used as the database for the application. 2024. URL: https://mariadb.org/.

[10] Hibernate Team. *Hibernate - Java Framework for ORM and Persistence*. Purpose: used for ORM and database interaction in the back-end. 2024. URL: https://hibernate.org/.

[11] Firebase Team. *Firebase Authentication - Authenticate and Manage Users*. Purpose: used for user authentication. 2024. URL: https://firebase.google.com/products/auth.

[12] Firebase Team. *Firebase Cloud Messaging - Cross-Platform Messaging Solution*. Purpose: used for sending notifications to devices. 2024. URL: https://firebase.google.com/products/cloud-messaging.

[13] JUnit Team. *JUnit - Testing Framework for Java*. Purpose: used for unit testing Java components. 2024. URL: https://junit.org/.

[14] Mockito Team. *Mockito - Java Mocking Framework*. Purpose: used for creating mock objects in tests. 2024. URL: https://site.mockito.org/.

[15] Mock Service Worker Team. *Mock Service Worker - API Mocking Library*. Purpose: used to mock API calls during testing. 2024. URL: https://mswjs.io/.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved

A Assignement RDD AY 2024-2025

Students&Companies (S&C) is a platform that helps match university students looking for internships and companies offering them. The platform should ease the matching between students and companies based on:

- the experiences, skills and attitudes of students, as listed in their CVs;
- the projects (application domain, tasks to be performed, relevant adopted technologies-if any etc.) and terms offered by companies (for example, some company might offer paid internships and/or provide both tangible and intangible benefits, such as training, mentorships, etc.).

The platform is used by companies to advertise the internships that they offer, and by students to look for internships. Students can be proactive when they look for internships (i.e., they initiate the process, go through the available internships, etc.). Moreover, the system also has mechanisms to inform students when an internship that might interest them becomes available and can inform companies about the availability of student CVs corresponding to their needs. We refer to this process as "recommendation". Recommendation in S&C can employ mechanisms of various levels of sophistication to match students with internships, from simple keyword searching, to statistical analyses based on the characteristics of students and internships. When suitable recommendations are identified and accepted by the two parties, a contact is established. After a contact is established, a selection process starts. During this process, companies interview students (and collect answers from them, possibly through structured questionnaires) to gauge their fit with the company and the internship. S&C supports this selection process by helping manage (set up, conduct, etc.) interviews and also finalize the selections. To feed statistical analysis applied during recommendation, S&C collects various kinds of information regarding the internships, for example by asking students and companies to provide feedback and suggestions. Moreover, S&C should be able to provide suggestions both to companies and to students regarding how to make their submissions (project descriptions for companies and CVs for students) more appealing for their counterparts. In general, S&C provides interested parties with mechanisms to keep track and monitor the execution and the outcomes of the matchmaking process and of the subsequent internships from the point of view of all interested parties. For example, it provides spaces where interested parties can complain, communicate problems, and provide information about the current status of the ongoing internship. The platform is used by students at different universities. Universities also need to monitor the situation of internships; in particular, they are responsible for handling complaints, especially ones that might require the interruption of the internship.

Copyright © 2025, Ricci, Paoli, Grisoni – All rights reserved