POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design Document

Computer Science and Engineering

Author(s): **Mattia Brianti**

Alex Hataway

Mattia Rainieri

Academic Year: 2023-24

Deliverable:	DD
Title:	Design Document
Authors:	Mattia Brianti, Alex Hathaway, Mattia Rainieri
Version:	1.0
Date:	07-01-2025
Download page:	https://github.com/MattiaBrianti/BriantiHathawayRainieri/
Copyright:	

Contents

1 Introduction

1.1. Purpose

The purpose of this document is to present a technical description of the S&C platform. It is intended for developers that have to implement requirements and can serve as a contractual agreement between the customer and the contractors. Additionally, this document aims to provide the customer with a clear and precise explanation of the system's functionalities and constraints.

1.2. Scope

The Students&Companies platform is a tool that allows students and companies to facilitate their research for internships and interns. It offers students the possibility to join an internship project and companies to share the projects they are offering. For this software system a 3-tier architecture is the best choice, since there are three main levels such as a presentation level for users, an application tier to manage the logic of the internships, the request and communication and a data layer that copes with databases.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- **OAuth Access Token**: is a temporary credential that allows a client to access protected resources on behalf of a user or an app.
- **OAuth Refresh Token**: is a credential used to obtain a new access token without re-authenticating.

1.3.2. Acronyms

SSO: Single Sign On

API: Application Programming Interface

CV: Curriculum Vitae

S&C: Students&Companies

IDE: Integrated Development Environment

RASD: Requirements Analysis & Specification Document

1.3.3. Abbreviations

e.g.: For example

w.r.t.: With reference to

ID: Identifier

1.4. Revision history

- **1.0** (07th January 2025) - Initial release

1.5. Reference Documents

Specification document: *"Assignment RDD AY 2024-2025"*

UML official specification: `https://www.omg.org/spec/UML/`

Requirements Analysis and Specification Document: *"RASD"*

1.6. Document Structure

1. **Section 1: Introduction**
   This section gives a brief description of the project. It analyzes summarily the goals and the purpose of the platform described in the RASD. It also contains lists of definitions, acronyms and abbreviations used in the document.
2. **Section 2: Architectural Design**
   This section firstly contains an high level description of the components and their interactions. It uses diagrams to describe the architecture of the system. Also, at the end of the section, it describes design choices, styles, patterns and paradigms.
3. **Section 3: User Interface Design**
   This section provides an overview on how the UI of the system will look like.
4. **Section 4: Requirements Traceability**
   This section maps the requirements that have been defined in the RASD to the design elements defined in this document.
5. **Section 5: Implementation, Integration and Test Plan**
   This section shows the order in which the subcomponents of the system will be implemented as well as the order in which subcomponents will be integrated and how to test the integration.
6. **Section 6: Effort Spent** In the sixth section are included information about the number of hours each group member has worked for this document.

2 | Architectural Design

2.1. Overview

Here we represent an high level overview of the S&C architecture:

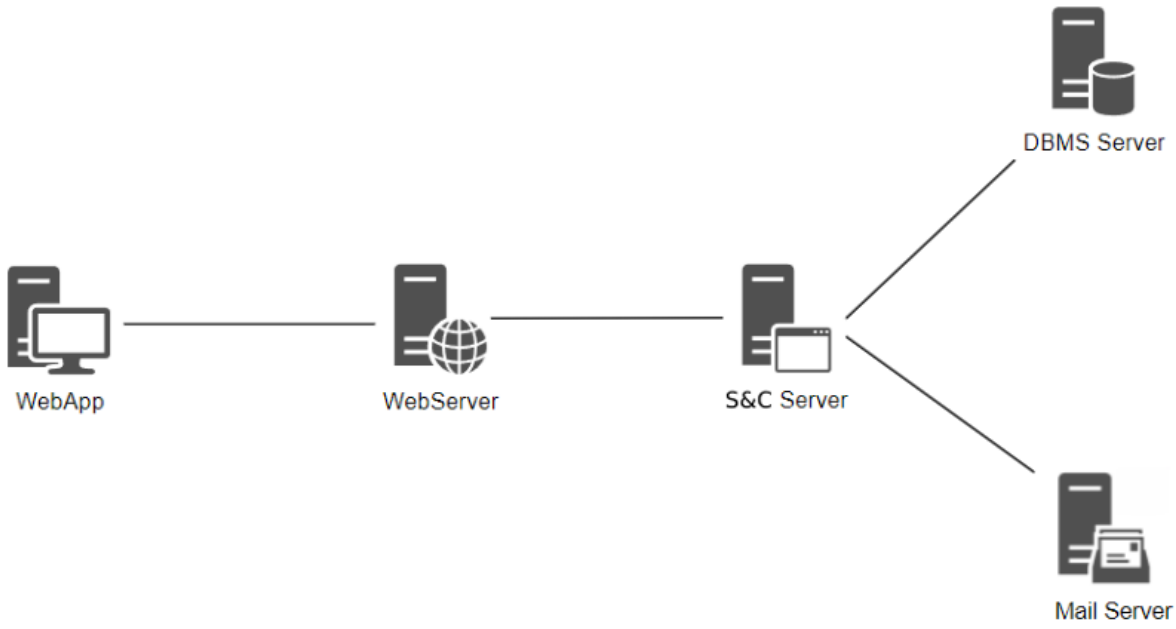

Figure 2.1: S&C overview

Client side:

- **WebApp:** allows all users to access the S&C platform. It enables user to perform various operations such as registration, login in, creating an internship or a CV etc.

Server side:

- **Web Server:** handles the communication with users by receiving and processing their commands, which are then directed to the S&C server.
- **S&C server:** manages most of the platform functionalities by using different components which are created to do a specific job. It facilitates the communication between the Web Server and Databases/APIs. It is also replicated across multiple machines to handle a high volume of requests.
- **DBMS Server:** contains data of students, universities, companies, CVs and various information obtained through questionnaires compiled by students.
- **Mail Server:** is responsible for sending confirmation eMails and all the notification regarding the reccomendation process.

2.2. Component view

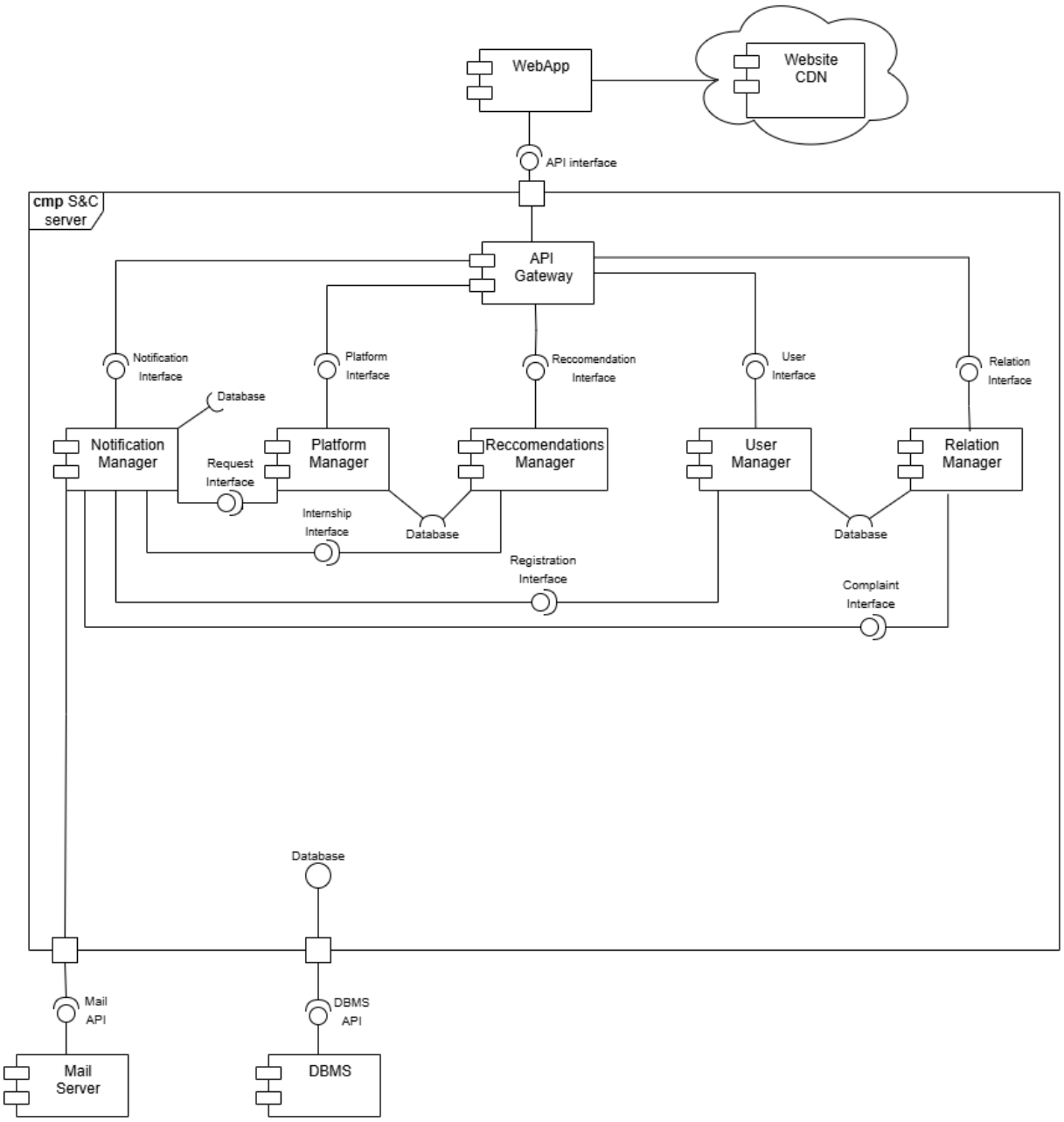

Figure 2.2: Component View Diagram

In the figure above it's shown the component view diagram of the platform, where it is represented how the external components of S&C communicate with the S&C server. Looking at the high level part of the diagram the components are:

- **WebApp:** which is an external access point for users, allowing the communication with the S&C server through an API interface.
- **Website CDN:** which is responsible for improving the distribution of web content to end users.

- **API interface:** which is responsible for connecting the presentation layer (WebApp) with the application layer (S&C server), which is coordinated by an API Gateway.
- **Mail Server:** is responsible for sending notification to the users such as registration confirmation eMails, new student Cv notification to companies or new internship posted notifications to students.
- **DBMS:** which is the data warehouse of the system containing data from the questionnaires, user's CV, company's data and internship related data.

Looking the diagram at a lower level, it shows all the components inside the server:

- **API Gateway:** is a must have component which manages all the communication between the user and the platform. Users interact with S&C through the API interface, and the API Gateway directs the requests to the appropriate component.
- **Notification Manager:** handles all the notifications that are to be sent to users. In particular it sends a message to: companies every time a student who is fit to their needs has registered, to students every time a company posts an interesting internship or to universities every time one of the two parties complains. To know which user to send the message to, the notification manager needs to interact with the DBMS API.
- **Platform Manager:** it manages all the aspects related to users necessities. For example, it allows students to provide their own information to create their personalized CV, allows student to join internships or allows all the parties to chat and communicate.
- **Reccomendations Manager:** is a component that handles the reccomendation process where through the use of statistical analysis and word searching mechanism recommends to student the most fit internships available, or recommends to companies new students that added their CV information.
- **User Manager:** this component handles the registration and log in of users. Whenever a new user wants to create an account a request is forwarded to this component by the API Gateway. Then the User Manager handles the request by sending through the registration interface and the notification manager a confirmation email. Once the email has been confirmed the component through the model interface adds the new user's information to the DBMS, including his topic preferences. For the login process the API Gateway forwards the request to the User manager, which communicates with the DBMS to retrieve the user's data.
- **Relation Manager:** this component manages the interactions between users letting them complain about problems or informing other parties about the current status of the internship.

2.3. Deployment view

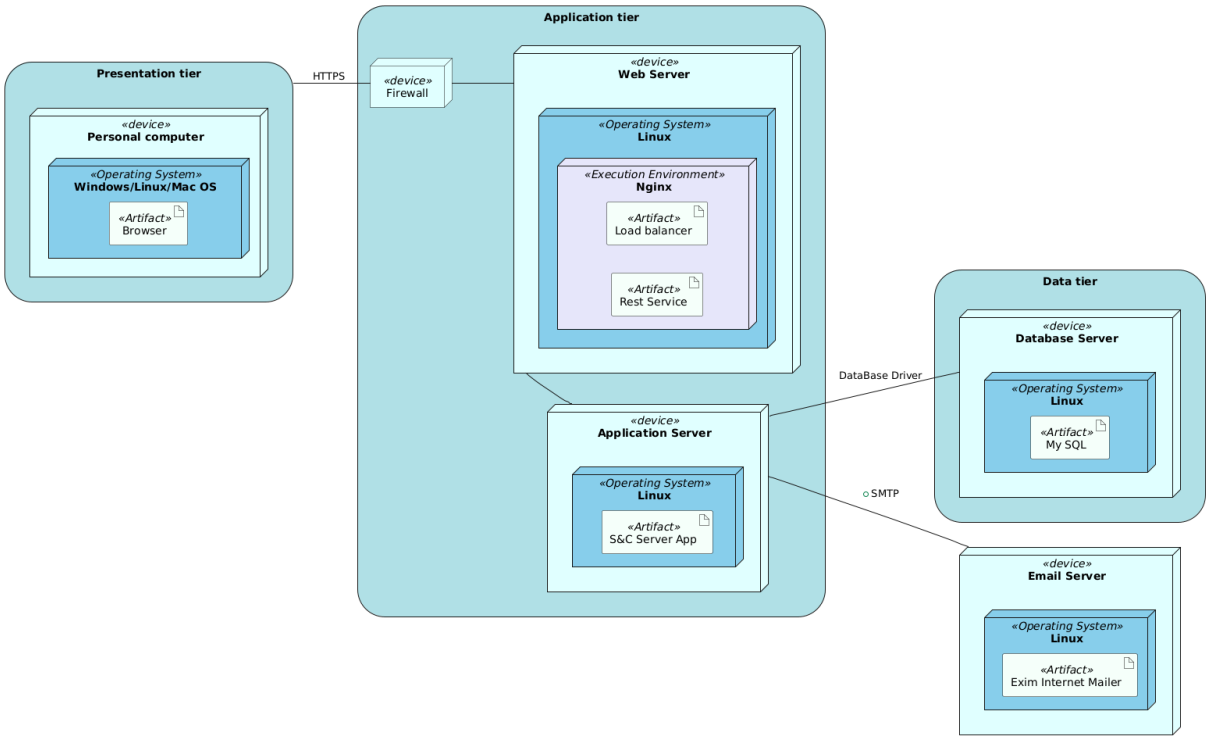

Figure 2.3: Deployment Diagram

Personal computer: Students and companies can access the platform using any browser installed on any computer. Access is also granted via any device that allows access to a web browser, such as a smartphone, tablet or smart TV. The browser will communicate with the Web Server

Web Server: The Web Server serves as the entry point for users accessing the Application Server's services via a web browser. It does not process business logic itself; rather, it balances incoming requests among various Application Servers to efficiently handle high volumes of user traffic. Additionally, it provides the client's browser with the necessary HTML, JSON, JavaScript, and CSS files to render the pages.

Firewall: Security system that monitors and filters incoming and outgoing network traffic based on predefined rules. It acts as a barrier between a trusted internal network and untrusted external networks, helping to block malicious traffic and unauthorized access.

Application Server: The Application Server hosts the system's entire business logic. It communicates with clients over HTTPS, facilitated by the Web Server. The Dashboard Manager directs incoming requests from the Web Server to the appropriate module, and the model gateway manages communication with the Database Server. This server node is replicated to accommodate high user traffic.

Database Server: All data concerning users, companies and universities are stored in the Database Server and managed by MySQL. The various Application Servers can retrieve information on this node via the model module and the database driver.

Email Server: The Email Server handles all notifications that are sent by e-mail. When there is a notification, such as that of a compatible internship offer, the Application Server contacts the Email Server through SMTP protol to send the email to the recipient user

2.4. Runtime view

The following sequence diagrams represent the dynamics of interaction between components.
Sequence diagrams from RW1 to RW9 are the realizations of the corresponding use cases in the RASD document.

RW1:

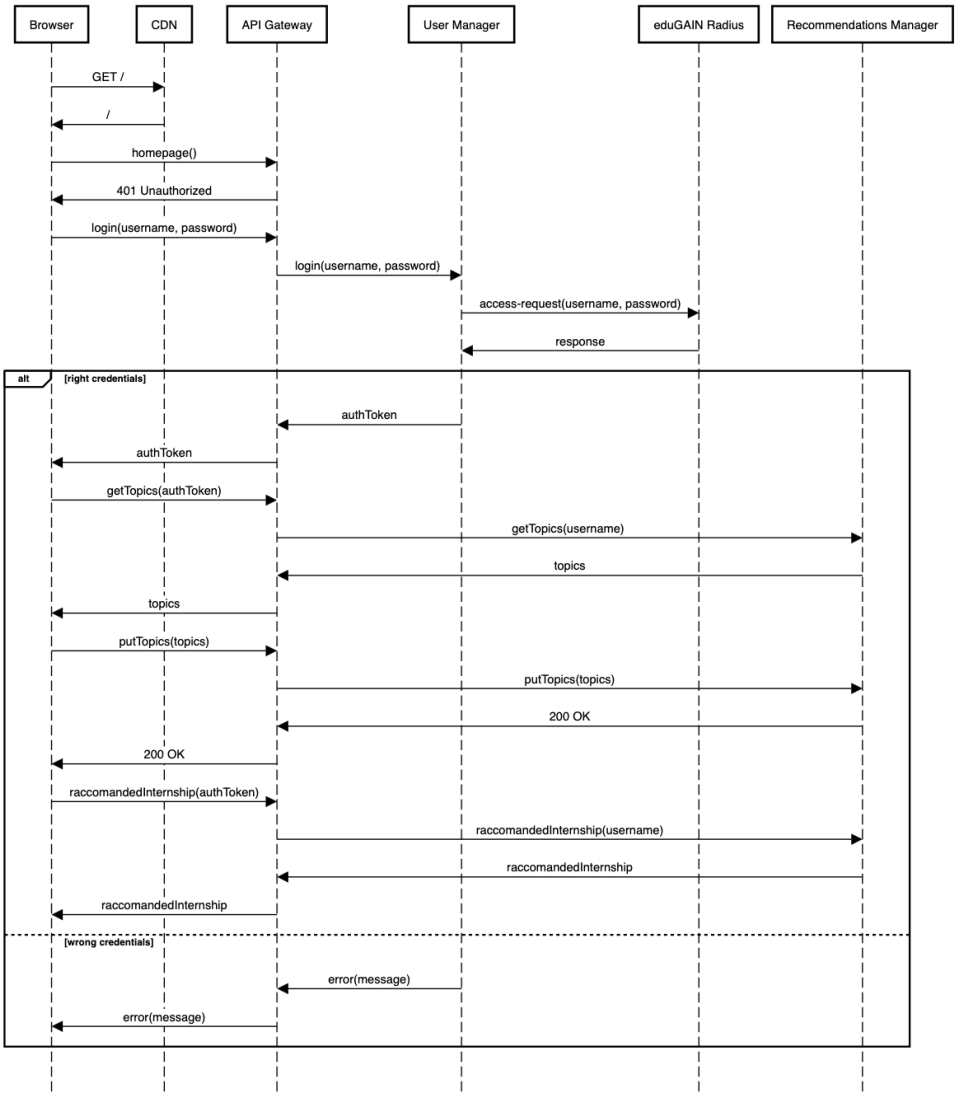

Figure 2.4: Student's first platform access.

RW2:

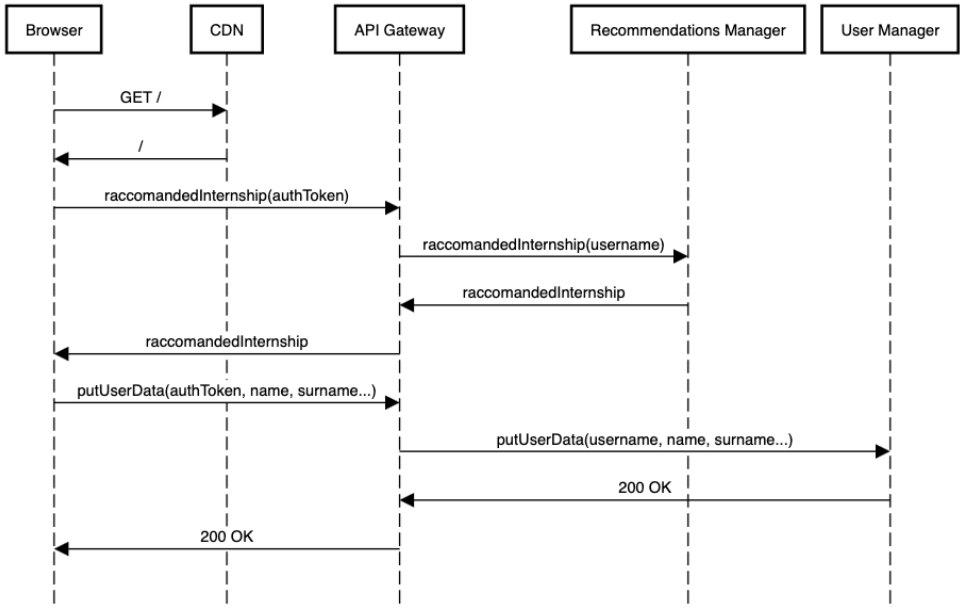

Figure 2.5: Student inserts his CV information in the InitialForm.

RW3:

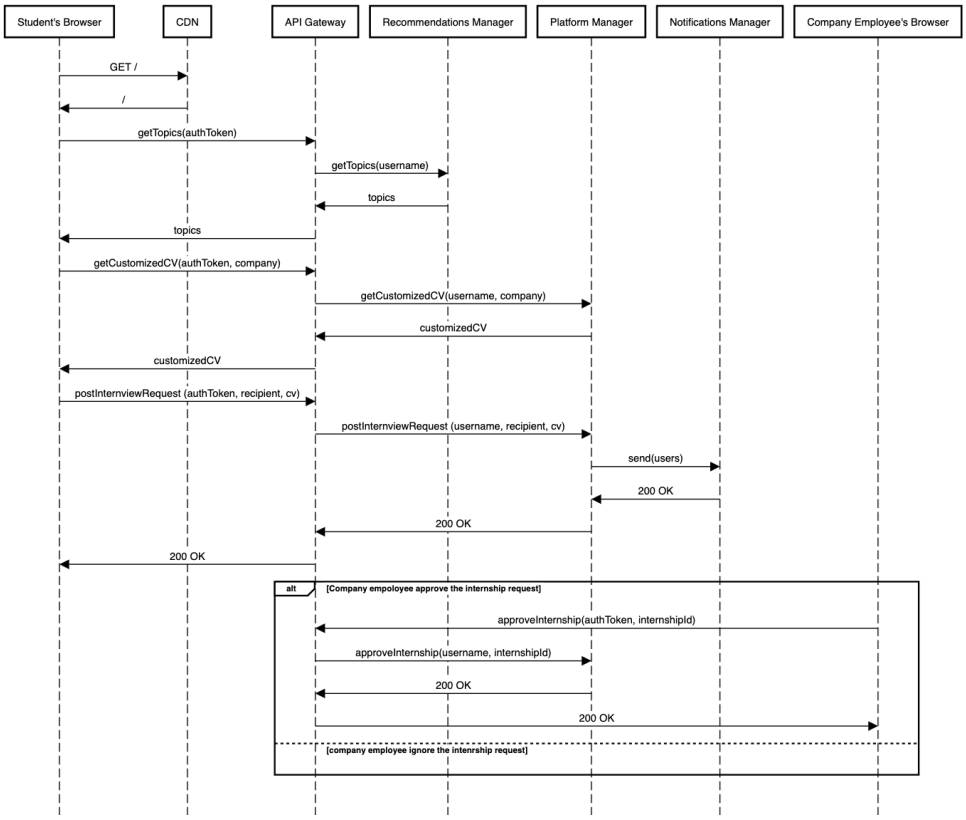

Figure 2.6: Student search through the internships and contact the company.

RW4:

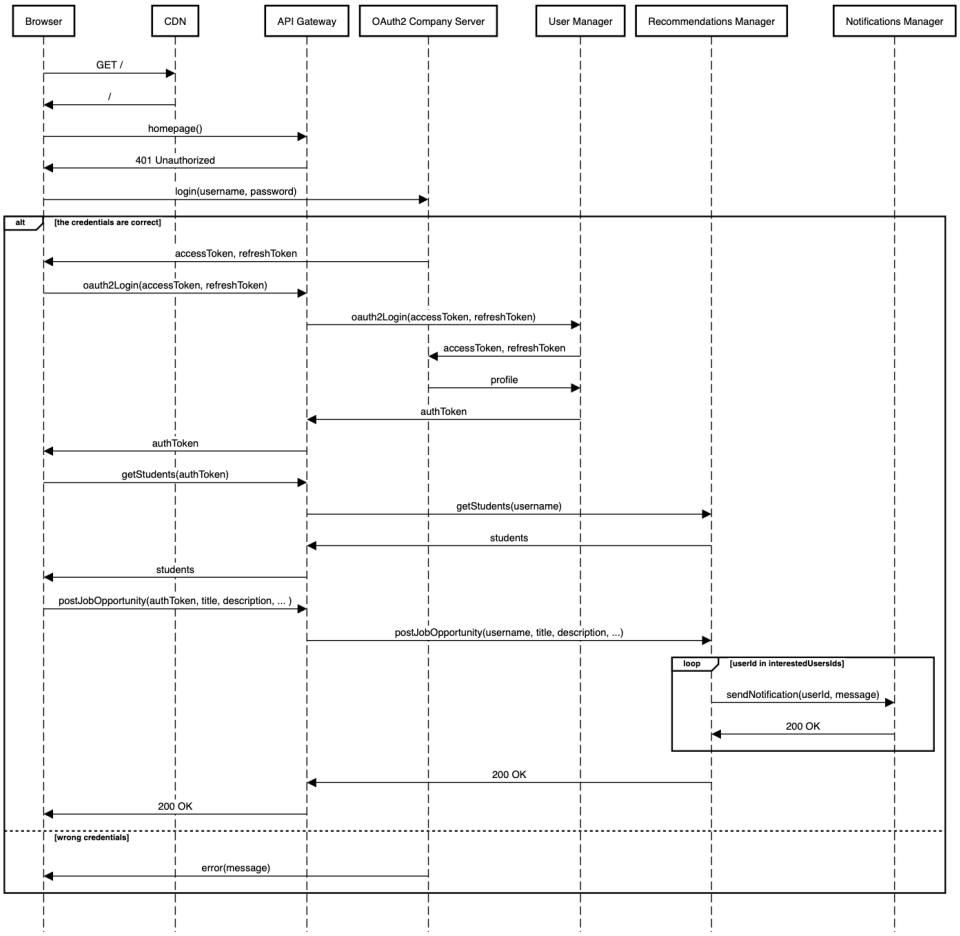

Figure 2.7: A company publishes an advertisement about the internships they are offering.

RW5:

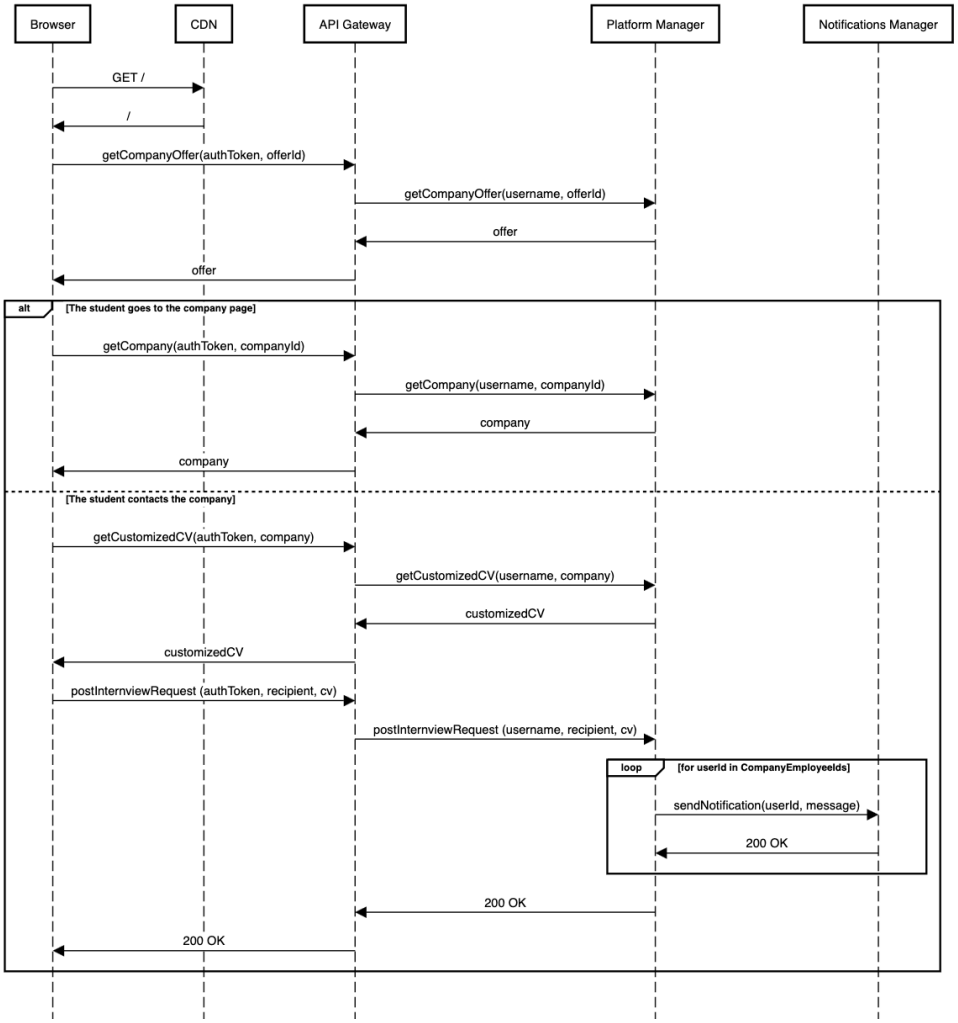

Figure 2.8: A student receive a notification about the availability of an internship that might interest her.

RW6:

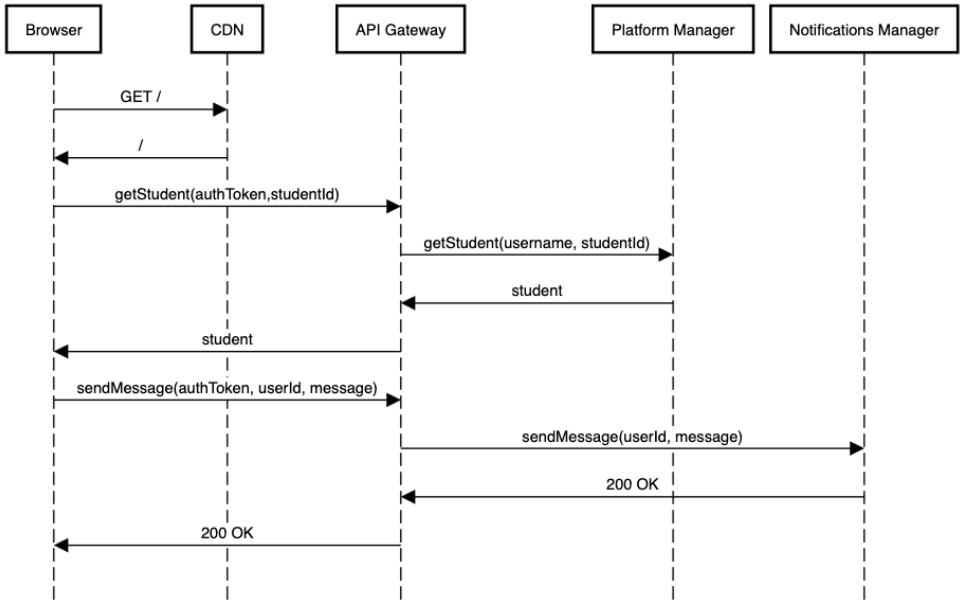

Figure 2.9: A company receives a notification about the availability of a student CV corresponding to their needs.

RW7:

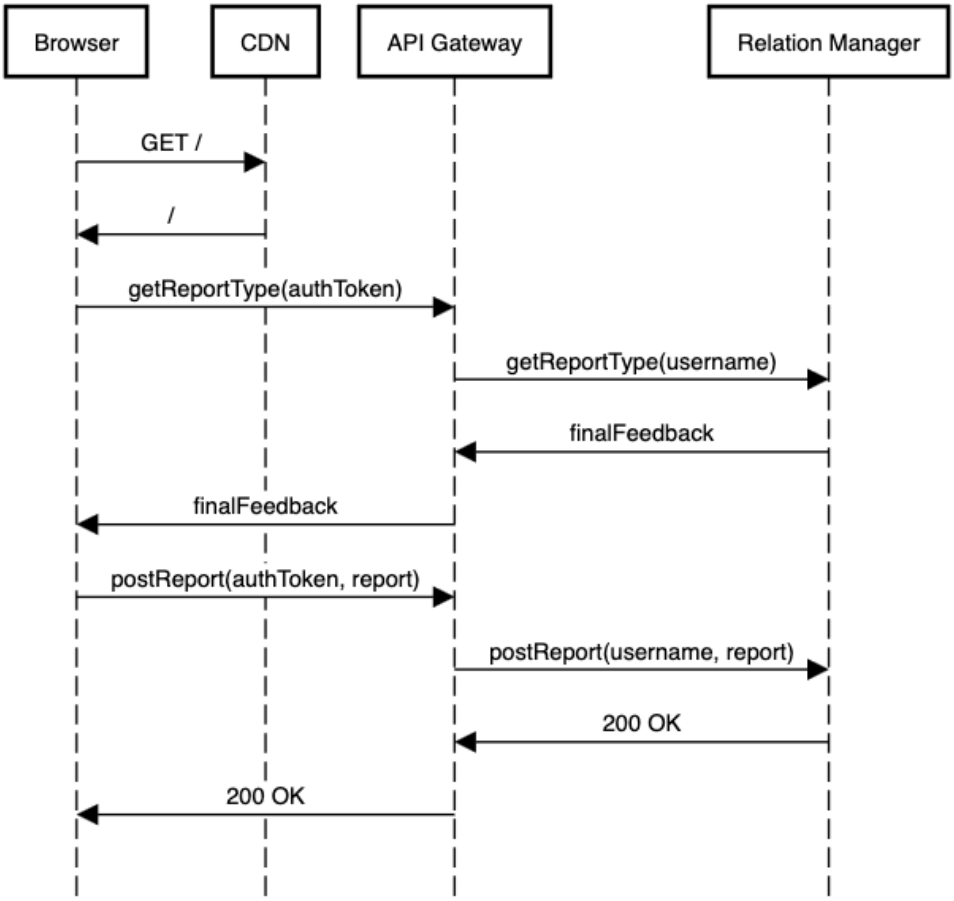

Figure 2.10: Student gives final feedback about the internship.

RW8:

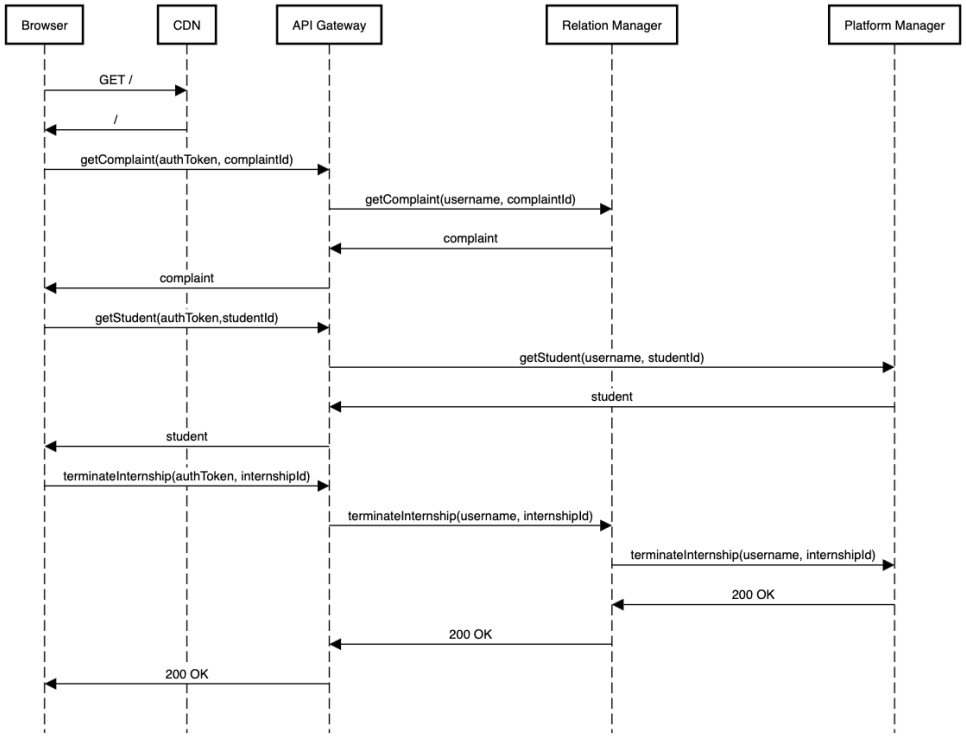

Figure 2.11: The University receives the request to end an internship from a student and contacts the company to end it.

RW9:

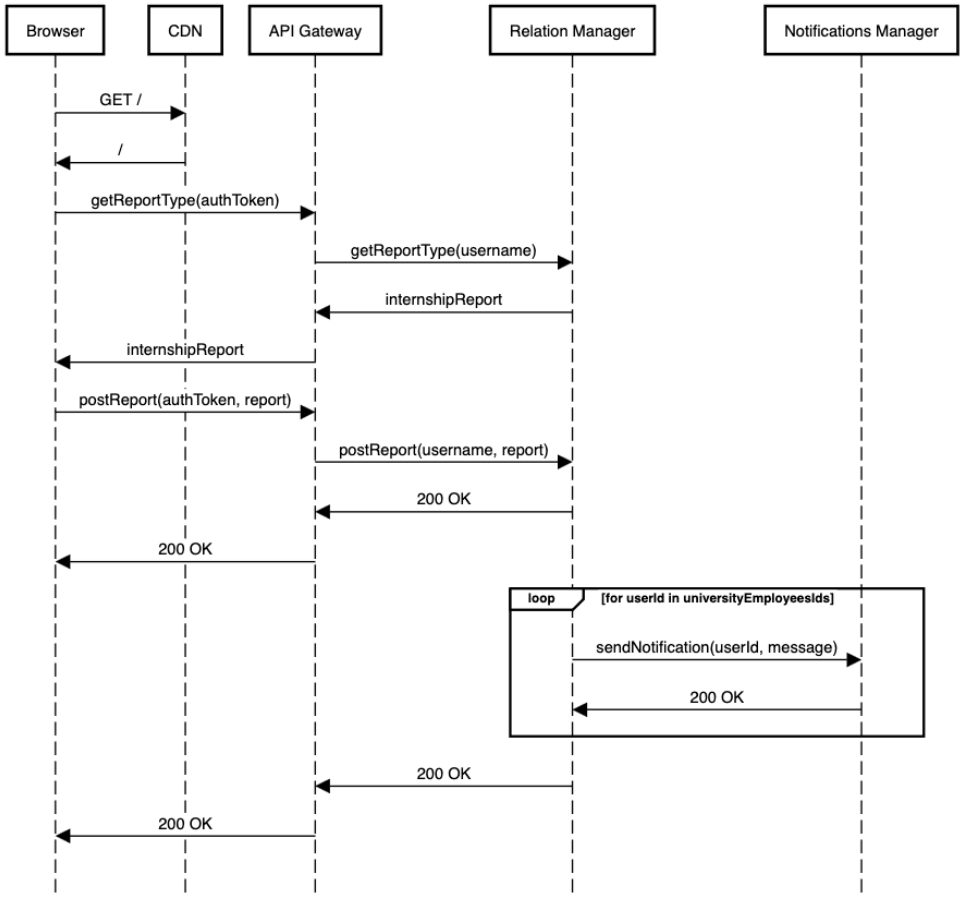

Figure 2.12: Student complains with the university on the "Report Area" about his ongoing internship.

RW10:

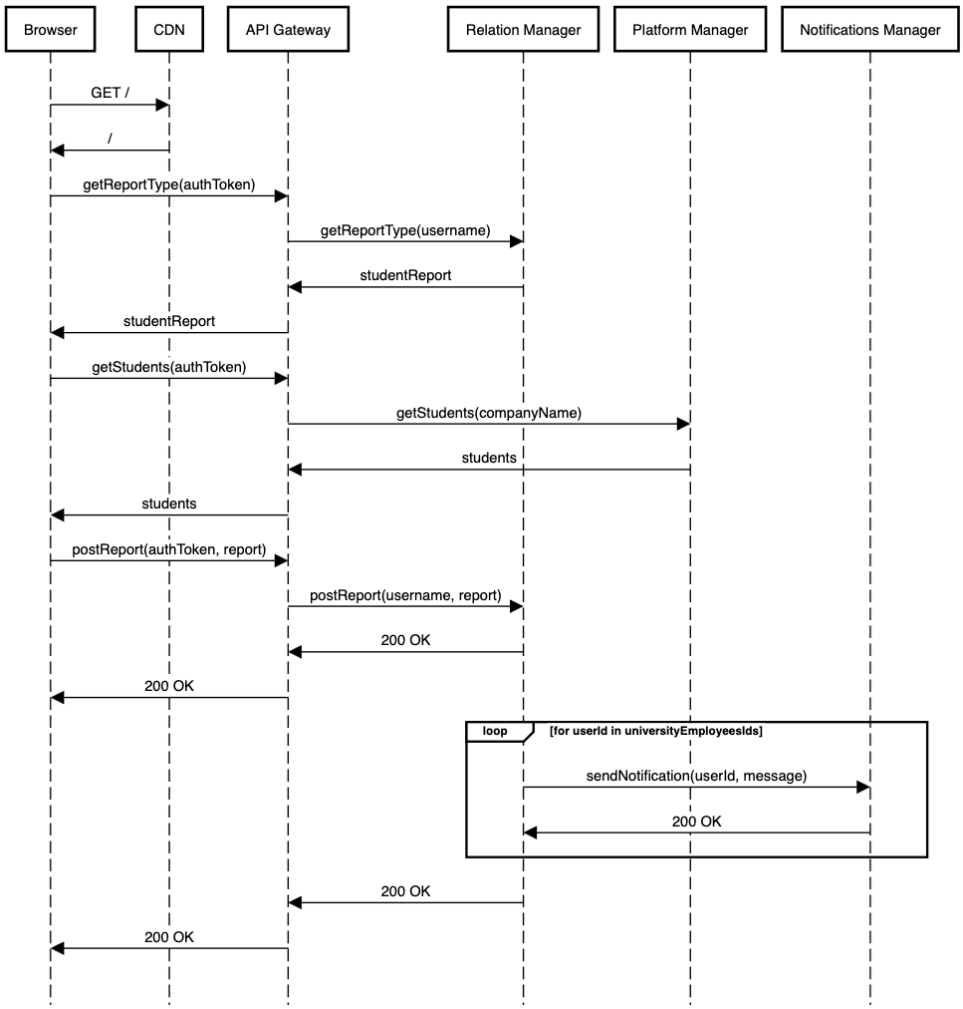

Figure 2.13: The company complains about the student taking the internship.

2.5. Component interfaces

Here the most relevant interfaces exposed by components are described, including all the operations seen in the previous diagrams:

- **API Gateway**
  - **login(username: String, password: String): String?**
  - **getTopics(authToken: String): List<Topic>**
  - **putTopics(authToken: String): Void**
  - **raccomandedInternship(authToken: String): List<Internship>**
  - **putUserData(authToken: String, name: String, surname: String, ...): Void**
  - **oauth2Login(accessToken: String, refreshToken: String): Void**
  - **getStudents(authToken: String): List<Student>**
  - **postJobOpportunity(authToken: String, title: String, description: String, ...): Void**
  - **getCustomizedCV(authToken: String, company: String): CV**
  - **postInterviewRequest (authToken: String, recipient: String, cv: CV): Void**
  - **approveInternship(authToken: String, internshipId: ID): Void**
  - **getCompanyOffer(authToken: String, offerId: ID): Offer?**
  - **getCompany(authToken: String, companyId: ID): Company?**
  - **getStudent(authToken: String, studentId: String): Student?**
  - **sendMessage(authToken: String, userId: ID, message: String): Void**
  - **getReportType(authToken: String): ReportType**
  - **postReport(authToken: String, report: Report): Void**
  - **getComplaint(authToken:String, complaintId: ID): Complaint?**
  - **terminateInternship(authToken: String, internshipId: ID): Void**
- **User Manager Interface**
  - **login(username: String, password: String): String?**
  - **oauth2Login(accessToken: String, refreshToken: String)**
  - **putUserData(username: String, name: String, surname: String, ...): Void**

- **Recommendations Manager Interface**
  - **getTopics(username: String): List<Topic>**
  - **putTopics(username: String): Void**
  - **raccomandedInternship(username: String): List<Internship>**
  - **getStudents(username: String): List<Students>**
  - **postJobOpportunity(username: String, title: String, description: String, ...): Void**
- **Notifications Manager Interface**
  - **sendMessage(userId: ID, message: String): Void**
  - **sendNotification(userId: ID, message: String): Void**
- **Platform Manager Interface**
  - **getCustomizedCV(username: String, company: String): CV**
  - **postInterviewRequest (username: String, recipient: String, cv: CV): Void**
  - **approveInternship(username: String, internshipId: ID): Void**
  - **getCompanyOffer(username: String, offerId: ID): Offer?**
  - **getCompany(username: String, companyId: ID): Company? getStudent(username: String, studentId: String): Student?**
  - **terminateInternship(username: String, internshipId: ID): Void**
- **Relation Manager Interface**
  - **getReportType(username: String): ReportType**
  - **postReport(username: String, report: Report): Void**
  - **getComplaint(username:String, complaintId: ID): Complaint?**
  - **terminateInternship(username: String, internshipId: ID): Void**

Types:

- **ID** - a unique identifier, auto-incremented by the DBMS
- **Topic**

```
{
    id: ID,
    name: String,
    category: Tech | Extra | Design,
    type: String
```

```
    priority: Int
}
```

- **Internship**

```
{
    companyId: ID,
    companyName: String,
    companyLogoUrl: String
    start: DateTime,
    end: DateTime,
    description: String
}
```

- **Student**

```
{
    id: ID,
    name: String,
    surname: String,
    birthday: Date,
    status: available | busy,
}
```

- **Company**

```
{
    id: ID,
    name: String,
    description: String,
    logoUrl: String,
    category : Tech | Extra | Design
}
```

- **Offer**

```
{
    title: String,
    condition: String,
}
```

- **Complaint**

```
{
    id: ID,
    sender: {
        userId: Id,
```

```
        name: String,
        surname: String
        companyId: ID?
        isStudent: y | n,
        }
    description: String,
    status: Pending | inProgress | Closed
}
```

- **CV**

```
{
    id: ID,
    user: {
        id: ID,
        name: String,
        surname: String,
        birthday: Date
    },
    contact: {
        phone: String,
        email: String,
    },
    ambitions: String,
    education:
        {
           organizationName: String
           start: Date,
           inProgress: y | n,
           end: Date?,
           evaluation: Int?,
           laudeo: (y | n)?
        }[],
    experiences:
        {
            organizationName: String
            start: Date,
            inProgress: y | n,
            end: Date?,
            type: (project | research)?
        }
    skils:
    {
```

```
        name: String,
        description: String,
        type: Soft | Technical
    },
    extracurricularActivities: {
        name: Steing,
        organizazionName: String,
        event: String?,
        Achievments: String
    },
    linguage: {
        linguage: String,
        certificate: {
            name: String,
            organizationName: String,
            writeLevel: A1 | A2 | B1 | B2 | C1 | C2,
            readLevel: A1 | A2 | B1 | B2 | C1 | C2,
            listenLevel: A1 | A2 | B1 | B2 | C1 | C2,
            speackLevel: A1 | A2 | B1 | B2 | C1 | C2,
        }[]
    }[],
    availability: {
        type: paid | free,
        period: {
            start: DateTime,
            end: DateTime,
        }[],
    },
    additionalInformation: String?
}
```

- **ReportType**

```
{
    type: internshipReport | studentReport | finalFeedback
}
```

- **Report**

```
{
    id: ID,
    senderId: ID,
    name: String,
    description: String
```

```
    status: Pending | InProgress | Closed,
    type: Complaint | Evaluation
}
```

2.6. Selected architectural styles and patterns

Students&Companies will be developed on a **3-tier architecture**, a software design model that organises the platform into 3 distinct and interconnected layers. This style is widely used, as each tier runs its own infrastructure, ensuring scalability and maintainability. The model is organised into the following 3 tiers:

Presentation Tier: The tier designed for users, responsible for managing the interactions between the application and its users. This tier provides the browser-accessible graphical interface where users can enter data and display results. The objective of this tier is to ensure an intuitive and easy-to-access user experience.

Application Tier: The application tier is the brain of the system. This tier processes user requests coming from the Presentation Tier, executes commands and can communicate with the Data Tier, to save data or complete certain actions requested by users. This tier is also capable of handling multiple requests simultaneously while guaranteeing the security and integrity of the stored data.

Data Tier: The Data Tier is fundamental for data management. It stores and retrieves data securely, ensuring its integrity and availability. This tier includes databases (such as MySQL) and storage systems that interact with the logical tier.

The system is structured with a **Client-Server architecture**, in which the Client represents the user interface (Front-End), while the Server represents the backend unit that processes the various requests. Usually, it is always the Client that sends the requests and the Server takes care of handling and answering them. In some cases, however, such as when sending notifications, the Server will perform actions without being invoked.

The implementation of Students&Companies will follow the **Model-View-Controller pattern**, which is a software architecture pattern that separates an application into three main components: Model, View, and Controller, making it easier to manage and maintain the codebase.

Model: It is responsible for managing the application's data, processing business rules, and responding to requests for information from other components, such as the View and the Controller.

View: Displays the data from the Model to the user and sends user inputs to the Controller. It is passive and does not directly interact with the Model. Instead, it receives data from the Model and sends user inputs to the Controller for processing.

Controller: Controller acts as an intermediary between the Model and the View. It

handles user input and updates the Model accordingly and updates the View to reflect changes in the Model.

2.7. Other design decisions

Relational DBMS: it's been chosen to use relational database because of their guarantees given by ACID properties enforcing consistency and durability of the data.

Load balancing: to optimize the resource utilization by distributing requests evenly across servers, preventing bottlenecks.

Firewall: to enhance the protection of the user's data.

Notifications: they are sent via email with attached a link from which the user is able to open the platform. This approach guarantees that users are instantly informed about key events.

3 | User Interface Design

The user, when logging in, simply enters his institutional/work email, and will be redirected to his university/company login page. After logging in, the S&C platform will recognise whether the user is a company or a student and redirect him/her to its main page.

Figure 3.1: Login

Figure 3.2: Sidebar

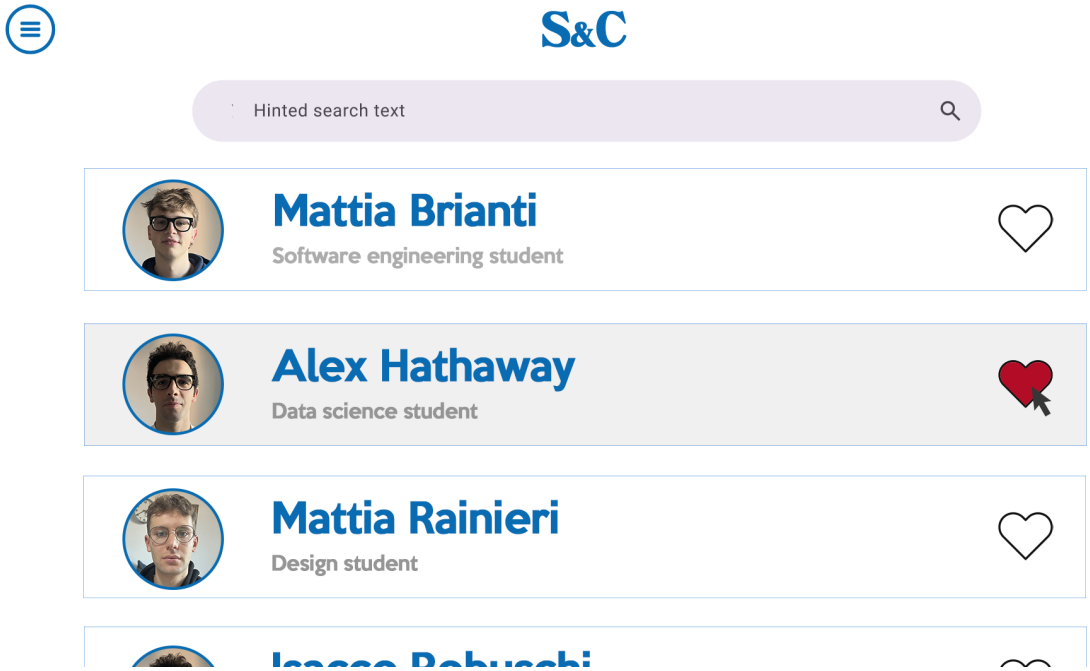

Figure 3.3: HomePage Company

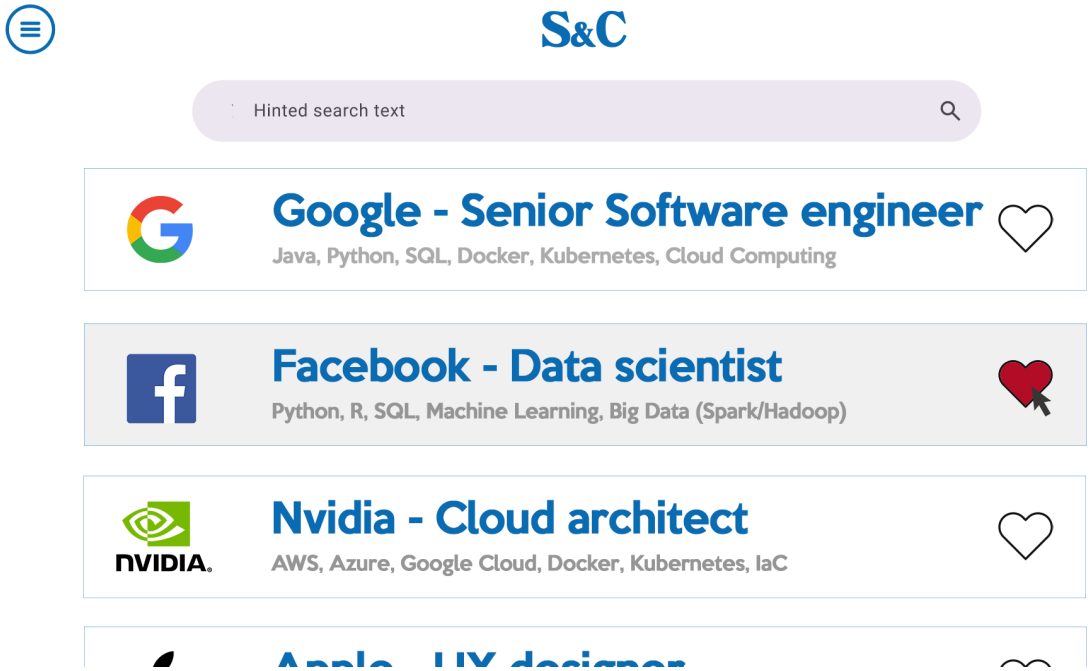

Figure 3.4: HomePage Student

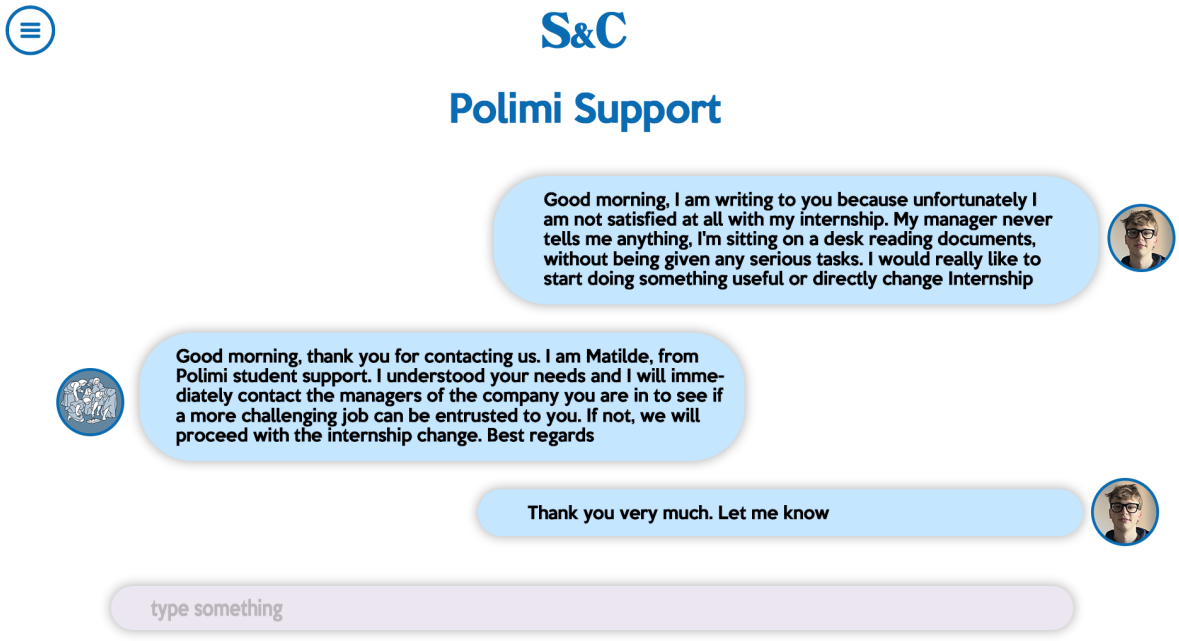

Figure 3.5: Chat

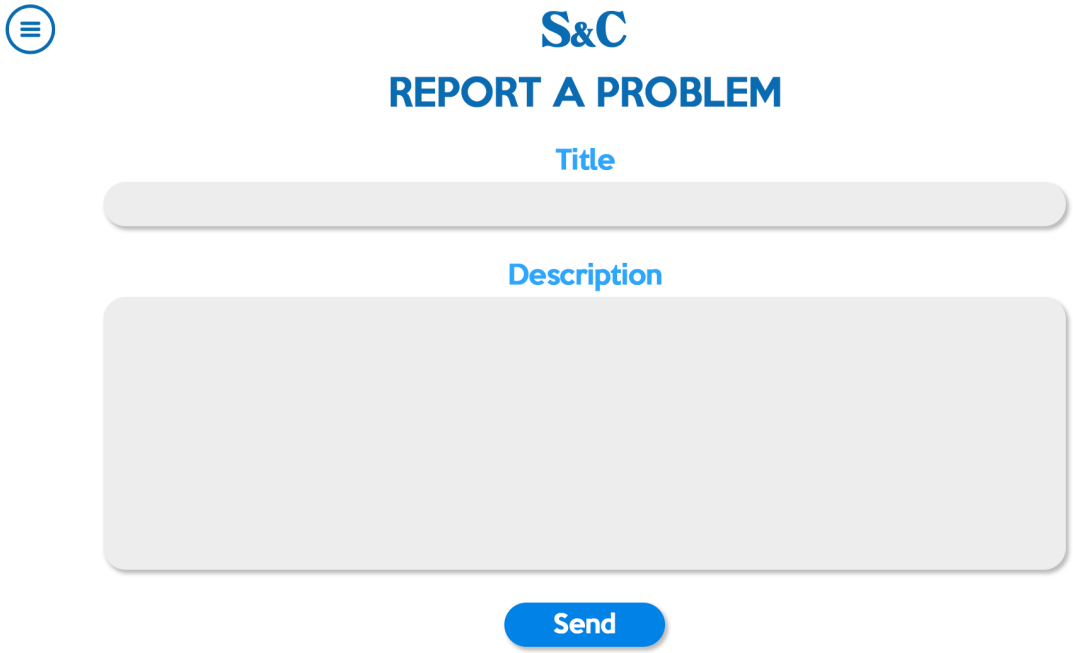

Figure 3.6: Report

4 | Requirements Treceability

In this section the requirements specified in the RASD are mapped to the components defined above. For brevity, the API Gateway is omitted as, since it mediates most of the operations, it would need to be specified in almost all requirements.

	Description	**Components**
R1	The system shall allow users to log in with SSO.	**User Manager**
R2	The system shall allow the student to provide information for their CVs.	**Platform Manager**
R2.1	The system shall allow the students to specify their personal information, like name, contact and personal ambition.	**Platform Manager, User Manager**
R2.2	The system shall allow the students to specify their education experience, including their academic background.	**Platform Manager**
R2.3	The system shall allow the students to specify their past work experience, specifying job title, company name, duration and technologies used.	**Platform Manager**
R2.4	The system will allow students to specify their technical and soft skills.	**Platform Manager**
R2.5	The system shall allow the students to specify their project and research, including the title, duration and description.	**Platform Manager**
R2.6	The system shall allow the students to specify their extracurricular activities, including the name, organization and achievements.	**Platform Manager**
R2.7	The system will allow students to specify their knowledge of languages, including the level and any certifications.	**Platform Manager**
R2.8	The system shall allow students to specify their availability.	**Platform Manager**
R2.9	The system will allow the students to add additional information.	**Platform Manager**
R3	The system shall allow the students to join an internship.	**Platform Manager**

	Description	Components
R3.1	The system shall help the students to create a customized CV for each company.	**Platform Manager**
R4	The system shall allow the students to be notified when a new applicable internship becomes available.	**Reccomendations Manager, Notification manager**
R5	The system shall allow the company employee to create an internship.	**Reccomendations Manager**
R5.1	The system shall allow the company employee to specify the title of the internship.	**Reccomendations Manager**
R5.2	The system shall allow the company employee to specify the description, helped by an AI, of the internship.	**Reccomendations Manager**
R5.3	The system shall allow the company employee to specify the requirement of the internship.	**Reccomendations Manager**
R5.4	The system shall allow the company employee to specify the duration of the internship.	**Reccomendations Manager**
R5.5	The system shall allow the company employee to specify the availability of the internship.	**Reccomendations Manager**
R5.6	The system shall allow the company employee to specify other information about the internship.	**Reccomendations Manager**
R6	The system shall allow the company employee to be notified when a new potentially interesting student becomes available.	**Reccomendations Manager, Notification Manager**
R7	The system shall allow the student to view a personalized homepage after inserting his CV's information.	**Reccomendations Manager**
R8	The system shall allow the company employee to view a personalized homepage after publishing an internship.	**Reccomendations Manager**
R9	The system shall allow the company to write a complaint to the university.	**Relation Manager**
R10	The system shall allow the students to write a complaint to the university.	**Relation Manager**
R11	The system shall allow students to chat with his company.	**Platform Manager**

	Description	Components
R12	The system shall allow companies employee to chat with his trainee.	**Platform Manager**
R13	The system shall allow universities employee to chat with his students.	**Platform Manager**
R14	The system shall allow the student to insert initial information.	**Reccomendations Manager**
R15	The system shall allow the student and the company to arrange a meeting.	**Platform Manager, Notification Manager**
R16	The system shall send a meeting link for the interview to the student and the company.	**Platform Manager, Notification Manager**
R17	The system shall allow the student to write on a logbook to inform the university on the status of his internship.	**Relation Manager**

Table 4.1: Components traceability matrix

5 | Implementation, Integration and Test Plan

The following chapter aims to show the strategies implemented in order to better integrate and test the various components. The main purpose of the tests conducted is to try to induce and recreate, as early as the development phase, situations that might give rise to malfunctions or inate behaviors, so that they can be resolved before the final version intended for the public.

5.1. Feature Identification

Feature	**Importance**	**Difficulty**
Login	High	Medium
AI CV Generation	High	High
Publish internship ADV	High	Low
Custom notification	Medium	High
Send reports	High	Low
Chat	Low	High
AI Homepage Feed	High	Low

Table 5.1: Importance and difficulty of features

5.2. Development and Test Plan

The components were developed using a bottom-up approach, so that the work could be divided into several independent parts, each assigned to a different team. Subsequently, after completing the equal parts as well as those of high importance, we began to integrate them into the system, as described below. The tests were divided into the following categories:

- **Functional Testing** - in order check the presence of bugs.
- **Performance Testing** - in other to check if the system is responsive.
- **Stress Testing** - in order to simulate intensive use in a real-world scenario

5.3. Component integration and testing

The following section shows how the individual components were implemented in the system. In the following section ne interaction between components are shown using graphs.

5.3.1. Login

The only part that need to be tested is the interaction of the system with the interfaces provided by the two different authentication method (via OAuth2 for companies and via eduGAIN radius connection for students). The authentication services used are considered reliable by definition.

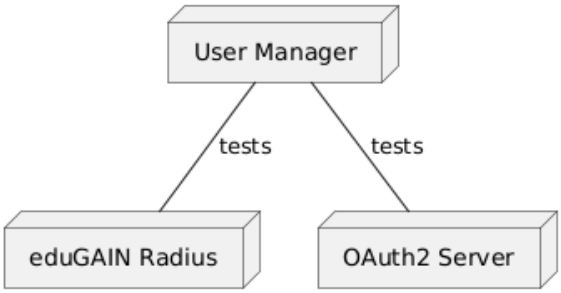

Figure 5.1: User manager.

5.3.2. Platform Manager

This component, in addition to allowing automatic CV generation, enables the addition of new job offers. For this purpose, given the criticality and confidentiality of the information contained, it was decided not to use external services, so all CVs are generated within the Platform Manager.

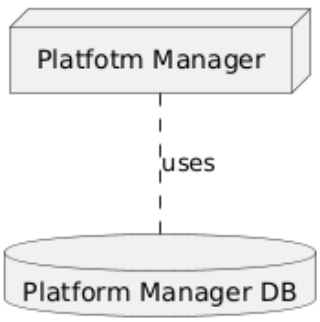

Figure 5.2: Platform manager.

5.3.3. Relation Manager

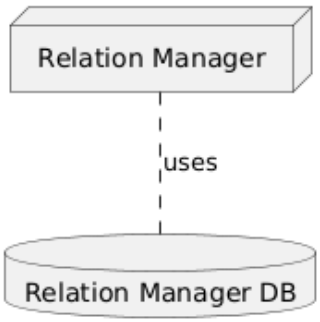

Figure 5.3: Relation manager.

5.3.4. Recommendation Manager

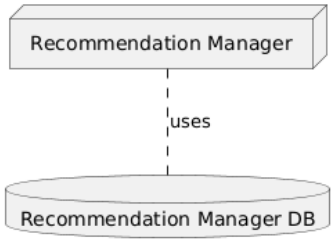

Figure 5.4: Relation manager.

5.3.5. Notification Manager

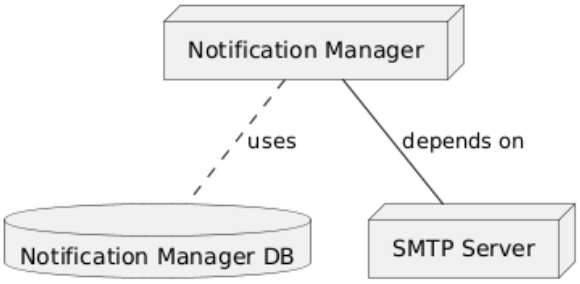

Figure 5.5: Notification manager.

At this point all subsystems have been implemented and tested, so the next step is performing integration testing between the subsystems and the API Gateway

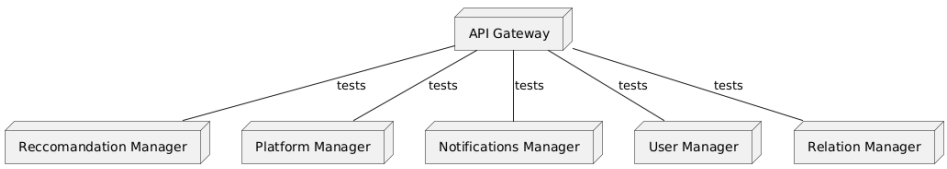

Figure 5.6: API Gateway

Since all the mangers have few iterations between them, except for the notification manager, the testing phase was carried out in a very focused mode on individual components.

Therefore, the only iterations tested separately were with the "Notification Manager" component and the interaction between "Relation Manager" and "Platform Manager" upon receipt of internship interruption request.

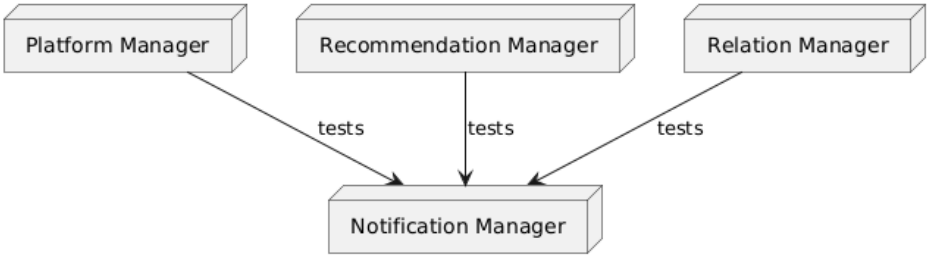

Figure 5.7: Notification manager tests.

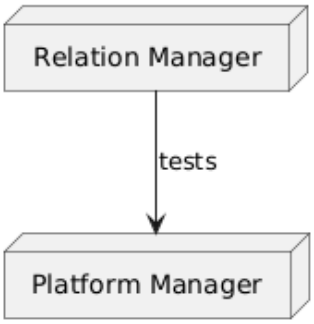

Figure 5.8: Testing between the "Relation Manager" and the "Platform Manager".

6 | Effort Spent

Member of group	Chapter	Time spent
Mattia Brianti	Introduction	2h
	Architectural Design	8.5h
	User Interface Design	3h
	Requirements Traceability	1h
	Implementation, Integration and Test Plan	1h
Alex Hathaway	Introduction	1h
	Architectural Design	9.5h
	User Interface Design	1h
	Requirements Traceability	1h
	Implementation, Integration and Test Plan	2h
Mattia Rainieri	Introduction	1.5h
	Architectural Design	10h
	User Interface Design	1h
	Requirements Traceability	1h
	Implementation, Integration and Test Plan	3h

Table 6.1: Time spent by each member of group.

Bibliography

[1] B.Liskov and J. Guttag. *Program Development in Java.* Pearson education, 2000.

[2] A. S. Tanenbaum and M. V. Steen. *Distributed Systems: Principles and Paradigms (2nd Edition).* Prentice Hall, 2007.

List of Figures

List of Tables