POLITECNICO
MILANO 1863

SOFTWARE ENGINEERING II

Students&Companies - Design Document

Authors
Priuli Elia
Viceconti Veronica
Zuccoli Marco

6 January 2025
Version 1

Contents

1 Introduction

1.1 Purpose

This document presents all the information regarding the design and architectural choices made to develop the S&C software, its component and how they interact with each other to satisfy the requirements listed in the RASD document.
In addition information about the testing, integration and implementation phases are provided.

1.2 Scope

The S&C application aims to solve the problem of matching university students with companies that offer internships. Moreover the students' universities can stop an internship if relevant complaints are received.
Different types of users means different devices with different hardware and power limitation, running different operating systems.
As a consequence, a lightweight application with a shared codebase is desirable.
Given these constraints a 3-tier, web-based application, with a thin client is the best solution (to address the hardware/power limitation).
In order to simplify the implementation, the notification system will be implemented via the Google Firebase CloudMessaging.

1.3 Definitions, Acronyms, Abbreviations

- DB: Database
- DBMS: Database Management System
- S&C: Students&Companies
- IOS: Iphone Operating System
- SOA: Service Oriented Architecture
- RASD: Requirements Analysis and Specification Document
- HTTPS: Hypertext Transfer Protocol over Secure Socket layer

1.4 Revision History

- Version 1 (6/01/2025) : First release of the document

1.5 Reference Documents

- previous RASD document
- Assignment RDD AY 2024-2025.pdf on the webeep page of the course

1.6 Document Structure

The rest content of this document is divided into the 6 sections below:

Introduction: In the first section, we outline the project's purpose and aim, and also incorporates a glossary of abbreviations and definitions necessary for understanding the issue.

Architectural Design: the key components and their interaction are shown here. This section also explains the main design and architectural choices.

User Interface Design: in this section are presented some mockups of the S&C interfaces.

Requirements Traceability: Here is shown the mapping between requirements and design elements.

Implementation, Integration and Test plan: Order in which you plan to implement subsystems and components as well as plan of the integration and test of the integration.

Effort Spent: here is explained the individual contributions of each team member in creating this document.

References: All the referenced documents used to draft this one.

2 Architectural Design

2.1 Overview

We have decided to develop the software through a WebApp and consequently to implement a 3-tier architecture, because:

1. On the client side, a light and non-constant computation is required.
2. Exploitation of existing protocols such as HTTPS, servlets, DBMS, and REST APIs for a modular and rapid implementation.
3. Maintainability: The application is easily maintainable and updatable.
4. Platform independent: it is not necessary to write different applications for different devices (PC, Mac, Android, iOS ...). By using web technologies, a single codebase that is accessible and usable by the various browsers already on the market and widely used is sufficient.
5. Scalability: as the number of users increases, it is possible to scale the infrastructure by adding more servers in parallel (scale out) to handle more requests simultaneously.

6. Reusability: the logic, for example of matchmaking, can also be reused for future projects.

7. Greater throughput: increases the ability to perform more work in parallel, reducing user wait time and increasing the usability of the WebApp.

The first tier is an interface level with a thin client, the second is the S&C server which manages both the web service and the actual computation (including match calculation), and finally, there is a server that hosts the DBMS and the actual database.

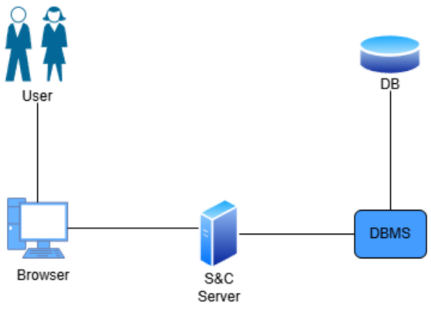

Figure 1: Three-tier architecture

2.2 Component View

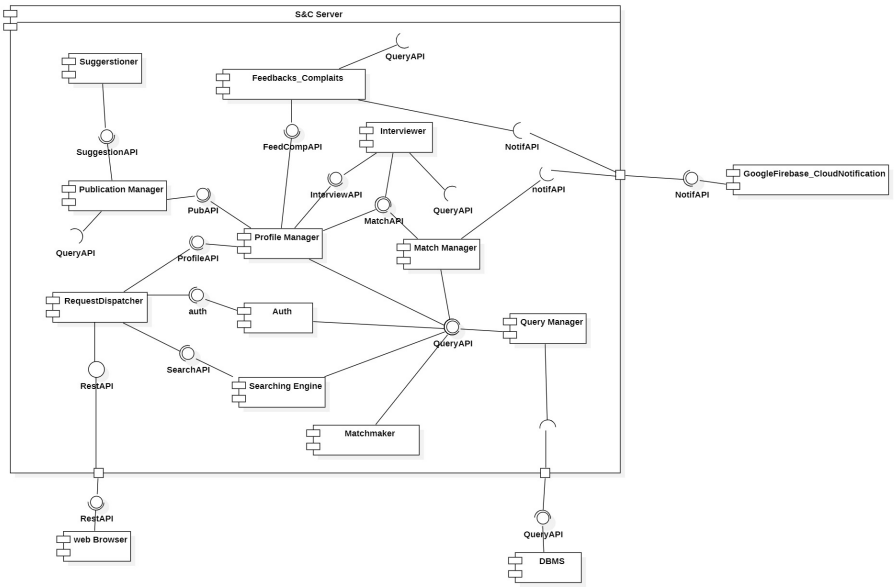

Figure 2: Component view

2.2.1 Request Dispatcher

This component is responsible for receiving various requests from the web app and has the task of routing them to the various other components based on the specific request. For this reason, it interacts with Profile Manager, Auth, and Searching Engine.

2.2.2 Profile Manager

This component allows universities to manage complaints, enabling the removal of ongoing internships, if necessary.
Furthermore, it communicates with other components that manage the publication of students and companies, feedback/complaints, and interviews, and with the Query Manager to obtain the data requested by the user.

2.2.3 Publication Manager

This component is responsible for managing the publications of students and companies. It allows these users to view existing publications, as well as create, modify, and delete them.
To carry out these activities, the component communicates with the Query Manager to obtain the necessary data and make the corresponding changes. It

also communicates with the Suggestioner component to allow users who want to improve their publication to receive suggestions.

2.2.4 Feedbacks_Complaints

This component is responsible for managing user feedback and complaints. It then communicates with the Query Manager to insert or obtain the necessary data.

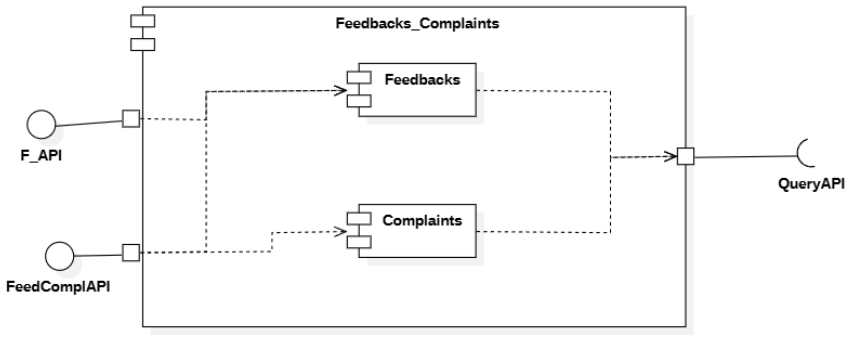

Figure 3: Component Feedbacks_Complaints

The sub-component Feedbacks manages everything related to user feedback, including creation and visualization, as well as the use of this data to allow other components to create suggestions. The sub-component Complaints manages everything related to user complaints, including creation and visualization.

2.2.5 Interviewer

This component is responsible for managing the interviews conducted with the various students accepted by companies. It allows the creation of the form, formatting it, recording the students' responses, and saving everything. Furthermore, following the interviews, it communicates with the Match Manager to handle the acceptance or rejection of the students.

2.2.6 Match Manager

This component is responsible for managing the matches found by the MatchMaker component. It allows viewing all matches created for a particular user and accepting or rejecting them. Additionally, it communicates with the Query Manager to request the necessary data and with the GoogleFirebase_CloudNotification API to send notifications of new matches and acceptance/rejection to various users or new complaints to universities.

2.2.7 Auth

This component is responsible for managing user registration and authentication. It is therefore composed of two sub-components, Registrator and Authenticator.

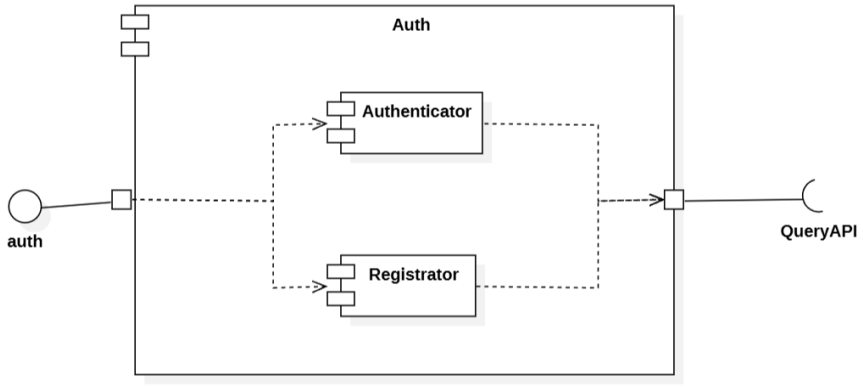

Figure 4: Component Auth

The sub-component Authenticator is responsible for managing user authentication, whether it is a Student, Company, or University, by checking the data and, if possible, allowing access to the system. The sub-component Registrator is responsible for managing user registration, whether it is a Student, Company, or University, by checking the data and communicating with the Query Manager to update the database.

2.2.8 Searching Engine

This component is responsible for managing the searches performed in the system by users (students can search for and view company profiles, and companies can request to view the profile of a specific user with whom they have a match). It interacts with the Query Manager to obtain the data requested by the user. Additionally, this component is responsible for searching and making visible to students all possible internships, so they can manually search for the one that interests them.

2.2.9 MatchMaker

This component is responsible for finding matches between companies and students, using different levels of sophistication, thanks to communication with the Query Manager to obtain all the data useful for creating matches (all companies, all students, ...).

2.2.10 Query Manager

This component is responsible for interfacing with the external DBMS component through queries that request specific data to be extracted from the database.

2.2.11 Suggestioner

This component is responsible for enabling users to improve their visibility thanks to some suggestions (for CVs and internships) created by this component and allowing the users to visualize them, through the interface, by the Publication Manager component.

2.2.12 GoogleFirebase_CloudNotification

This is an external component to the system that handles the notifications sent from the system to users (new matches, accepted or rejected matches, new complaints).

2.3 Deployment View

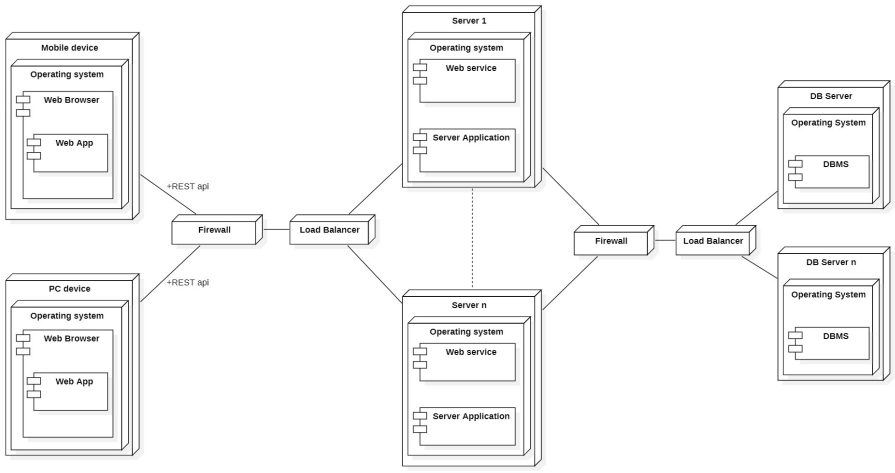

Figure 5: Three-tier architecture

- **Mobile/Pc Device** : the devices user could use to access the service. Every device runs on its operating system and executes a web browser. It then connects with the web server via Rest APIs. They host the presentation layer.
- **firewall** : This component prevents the system to be attacked

- **load balancer** : It forwards the requests to different servers to distribute the workload
- **Server 1..n** : The web servers that host the logic layer of the 3-tier architecture. It proved the S&C service to all the clients
- **DB servers** : the DB server handles data storage, retrieval, and management, acting as the middle tier between the application server and the data. It ensures efficient access to data and supports queries from the application layer.

2.4 Runtime View

In this section we are going to illustrate all the sequence diagrams that are useful for understanding the interactions within different components.

2.4.1 Login

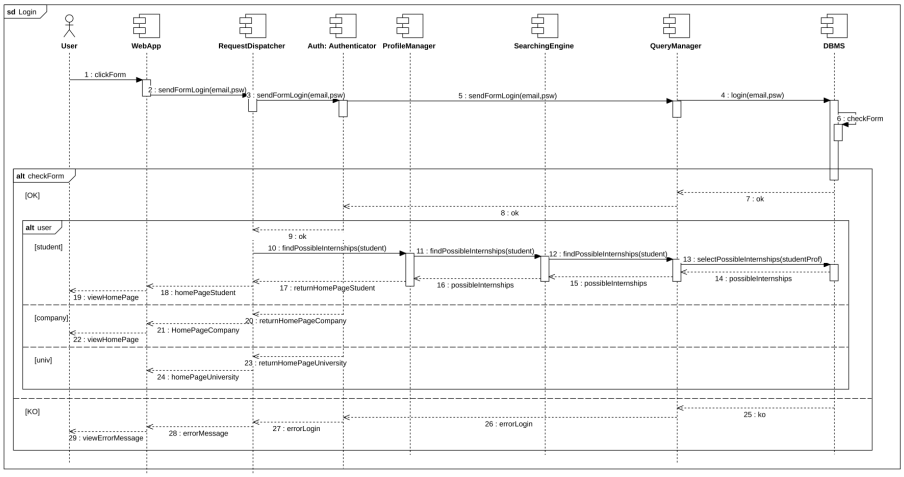

Figure 6: Login Sequence Diagram

This sequence diagram allows the user to log in, thus entering the application. To log in, the user must correctly enter their email and password, so that the system allows them to access their homepage. If the user logging in is a student, then the system, through the SearchingEngine component, searches for all company internships and makes them visible on the student's homepage.

2.4.2 Registration

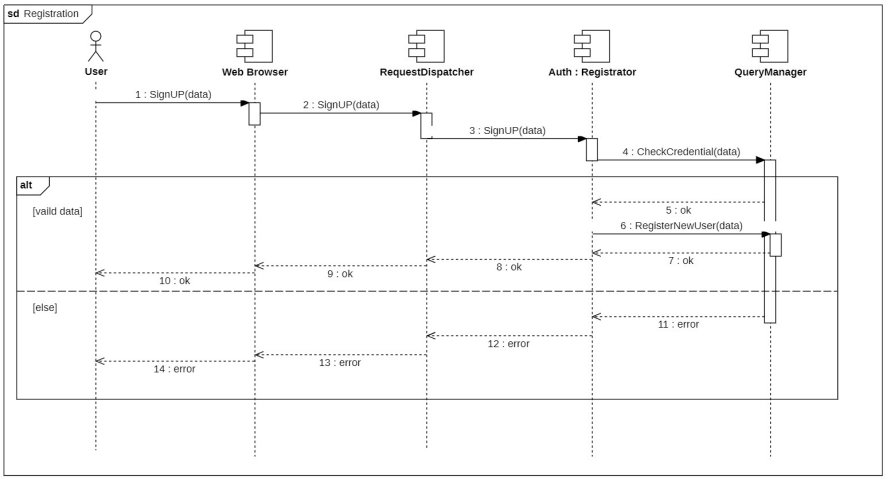

Figure 7: Registration Sequence Diagram

The sequence diagram above describes the event's flow that leads to the registration of a new user. The flow starts with the user who provides its information to the web browser. The information is then forwarded through the displayed components and finally reaches the QueryManager. Here the email's validity is checked and a proper answer is forwarded back to the user

2.4.3 CV & Preferences publication

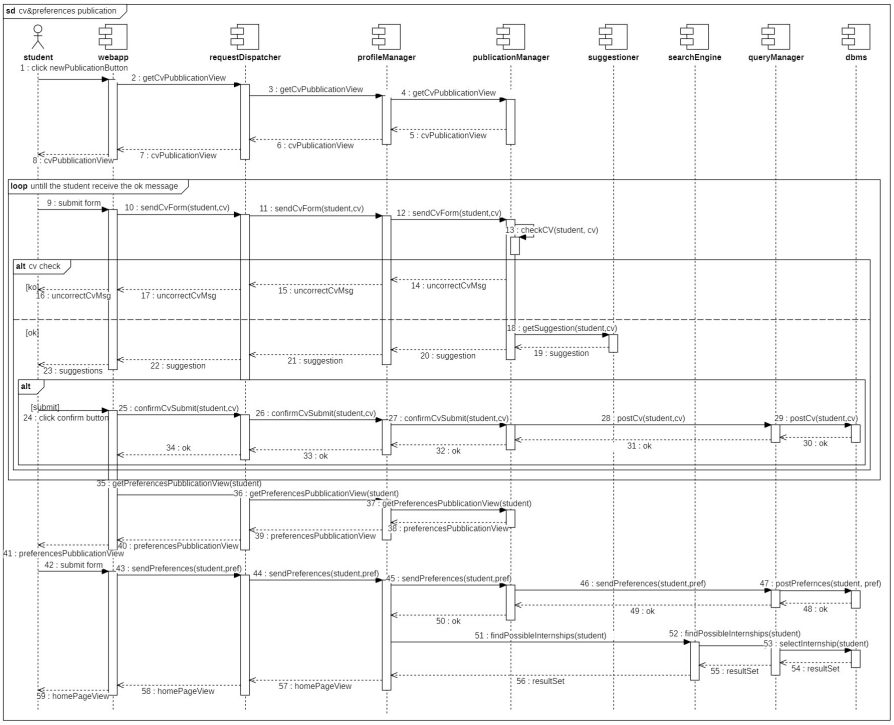

Figure 8: CV & Preferences publication Sequence Diagram

This sequence diagram allows the student to upload their CV, with the possibility of accepting the application's suggestions, and publishing it with their preferences.

2.4.4 Manage CV

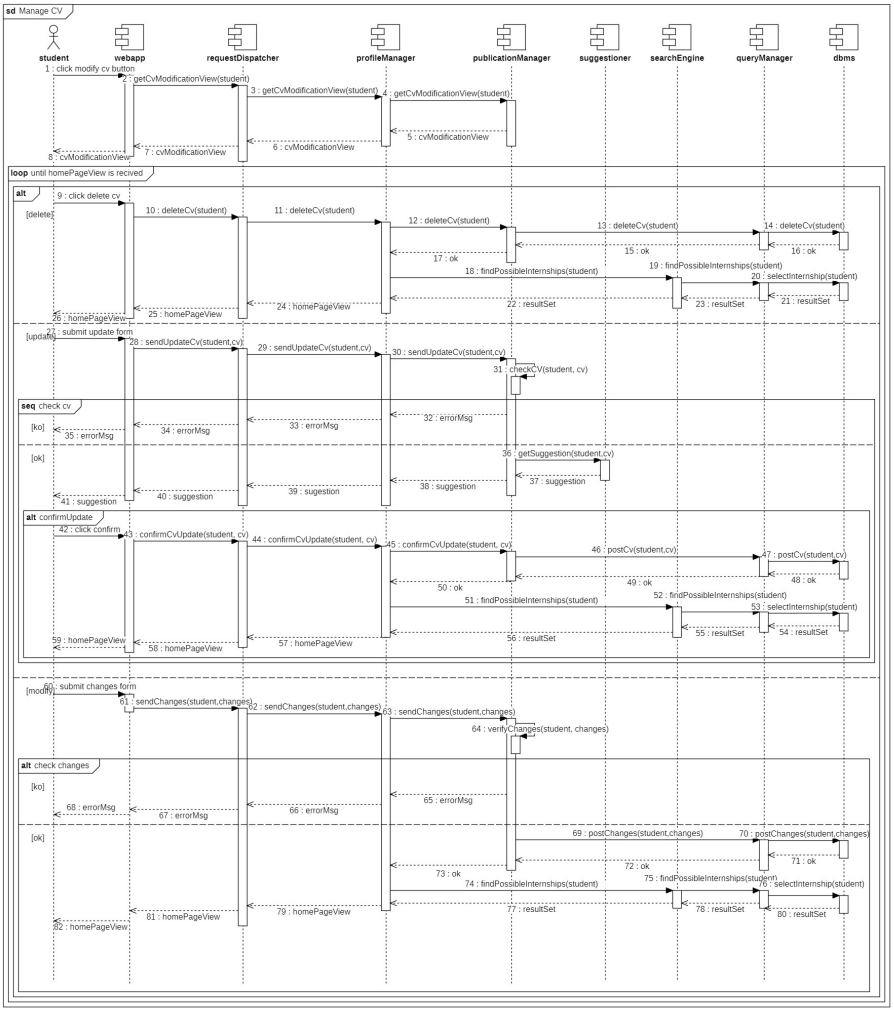

Figure 9: Manage CV Sequence Diagram

This sequence diagram allows the student to modify, update, or delete their CV. The student after receiving the modification page can choose which operation performs, after the end of the operation the student will be redirect to the home page.

2.4.5 Manage student publication

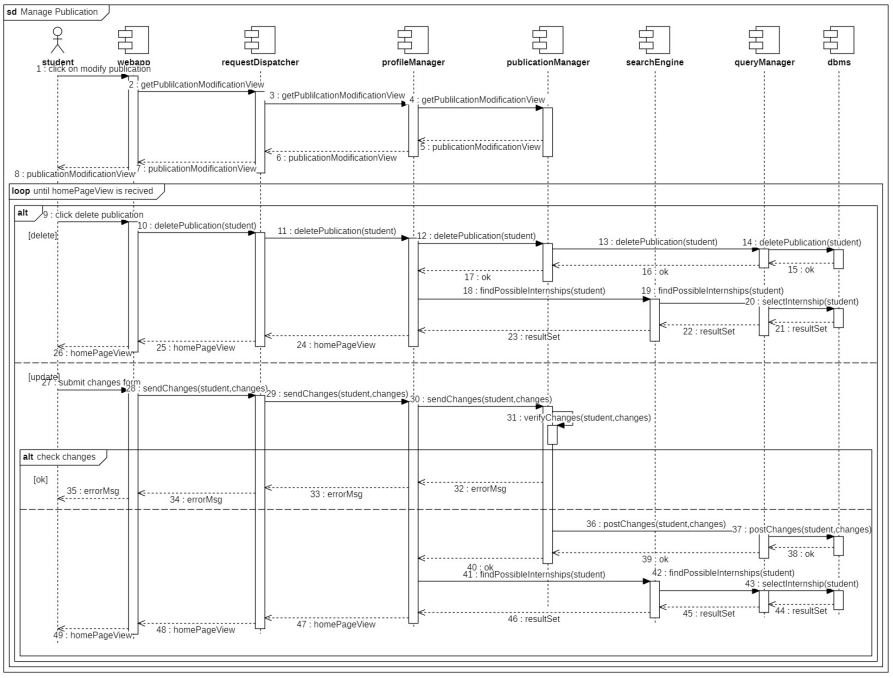

Figure 10: Manage student publication Sequence Diagram

This sequence diagram allows the students to modify or delete their publication. The student after receiving the modification page can choose which operation performs, after the end of the operation the student will be redirect to the home page.

2.4.6 Search Internship

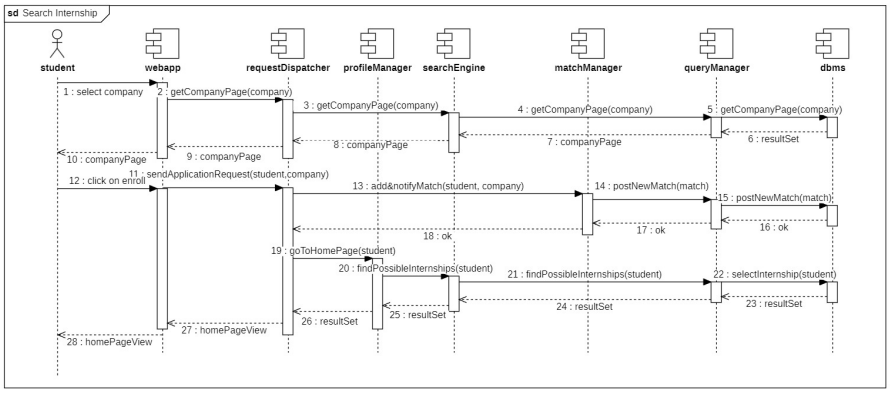

Figure 11: Search Internship Sequence Diagram

This sequence diagram allows the user to select a company and enroll, the system will create a new match and will send a notification to the company.

2.4.7 AcceptDeclineInternships

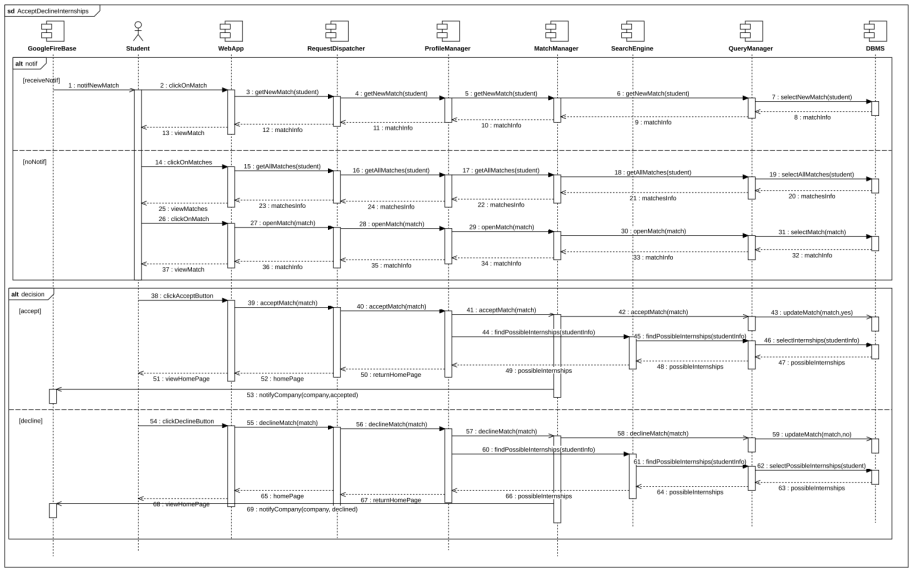

Figure 12: AcceptDeclineInternship Sequence Diagram

This sequence diagram allows students to accept or reject a new match. The interaction can begin either via notification, where the student directly opens the found match, or manually, within their profile, they access the matches, select one, and decide whether to accept or reject it. Following the student's choice, the corresponding company is notified.

2.4.8 PublishComplain student

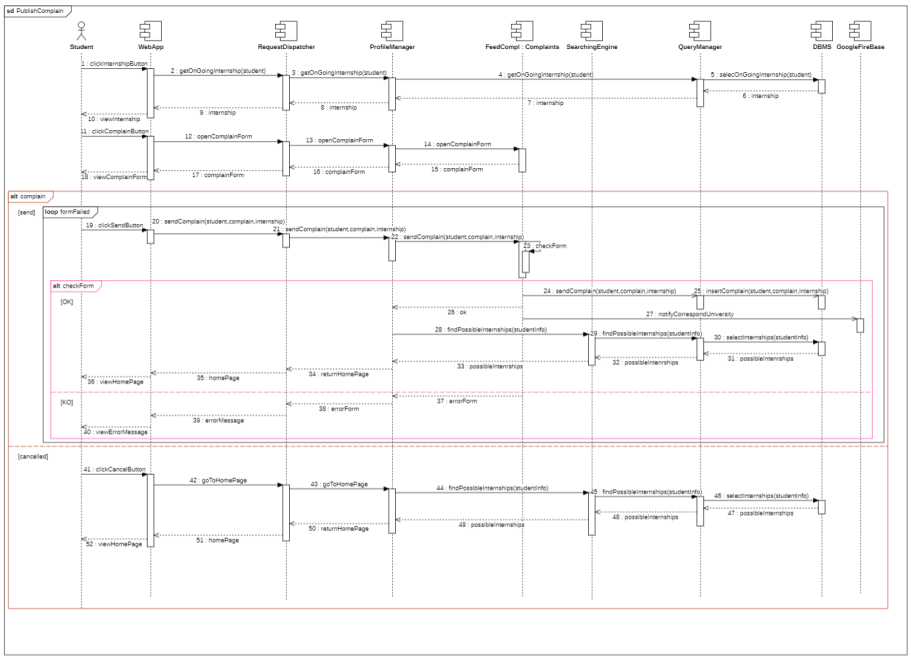

Figure 13: PublishComplain Sequence Diagram

This sequence diagram allows the student to interact with the system to write a complaint about their current internship. To do this, they first request the internship data and the corresponding form to write the complaint, and then either submit it or delete it if they change their mind. Until the form is correctly submitted, the system returns an error message. The system then notifies the corresponding university of the new complaint.

2.4.9 PublishFeedBack student

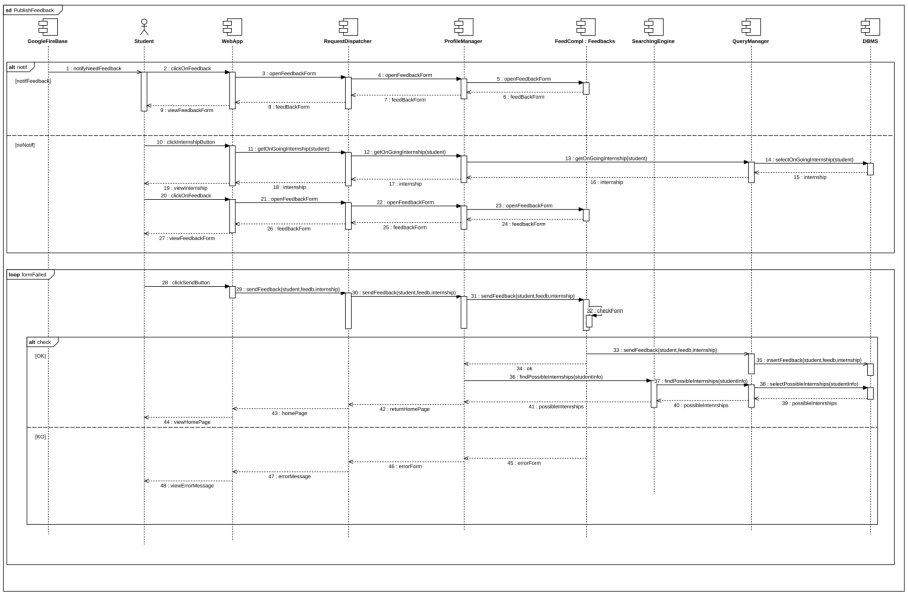

Figure 14: PublishFeedBack Sequence Diagram

This sequence diagram allows the student to interact with the system to insert feedback at the end of their internship. To do this, they either receive a notification and open the form to write and submit the feedback, or they can manually open the internship and write the feedback. Until the form is correctly submitted, the system returns an error message.

2.4.10 ManageComplains university

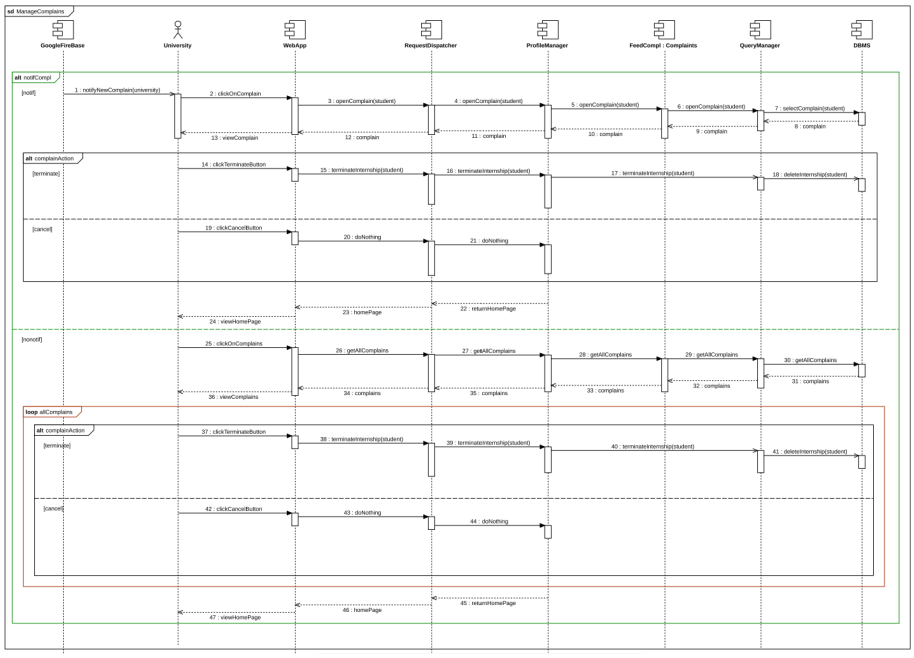

Figure 15: ManageComplains Sequence Diagram

This sequence diagram allows the university to interact with the system to view complaints from its students regarding various internships. To do this, they either receive a notification from the system, or they manually open their homepage and then decide whether to close the internship or not.

2.4.11 Internship publication

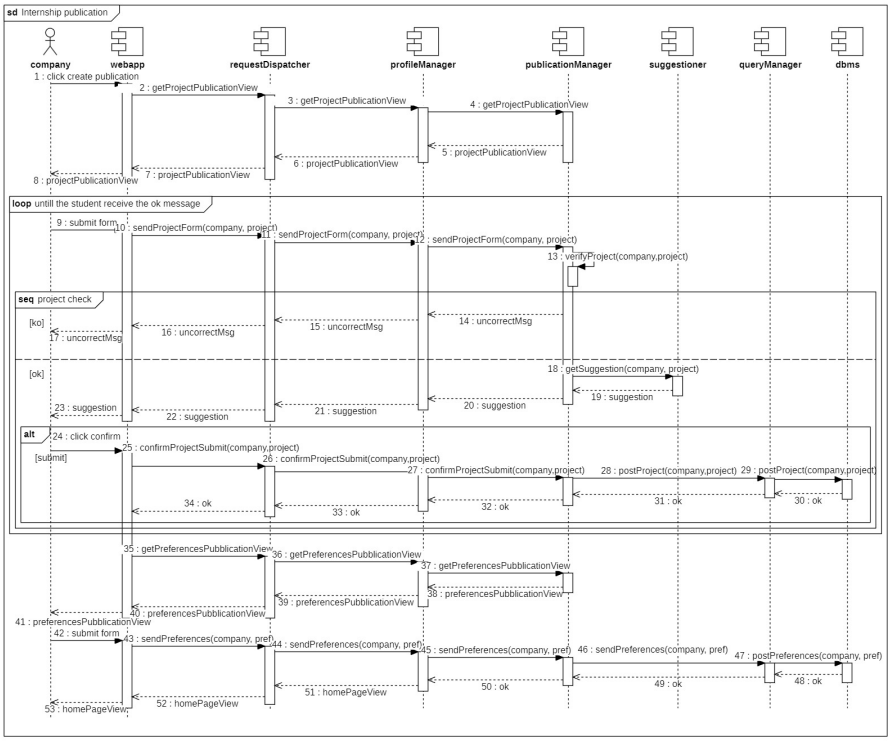

Figure 16: Internship publication diagram

This sequence diagram allows the company to upload the project, with the possibility of using the software to obtain suggestions, and to publish the necessary requirements.

2.4.12 ManagePublication company

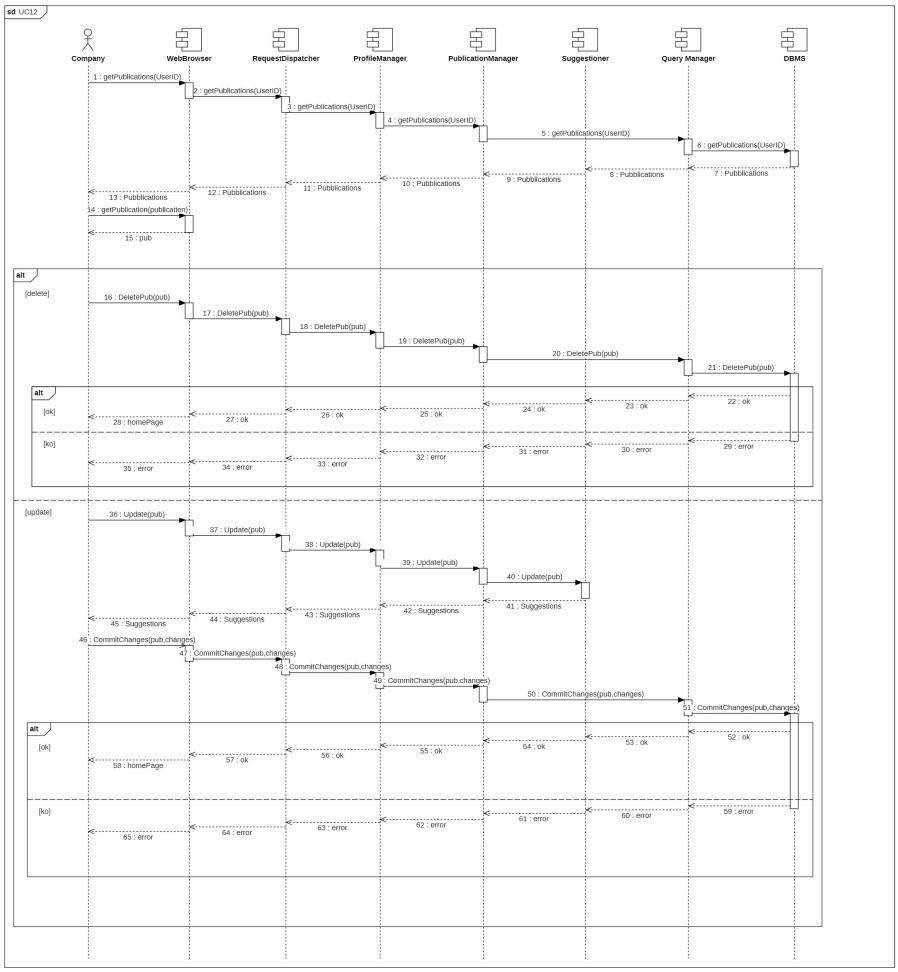

Figure 17: ManagePublication Sequence Diagram

The above sequence diagram describes the process of updating an internship publication carried out by a company. After getting all its publications and selecting the wanted one, the company can choose whether to update or delete it.

2.4.13 AcceptDeclineMatch company

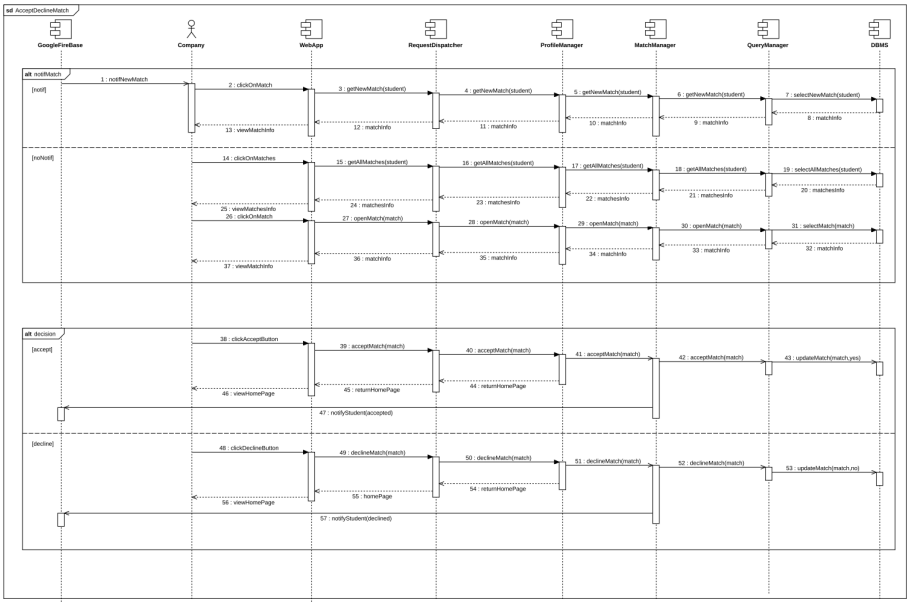

Figure 18: AcceptDeclineMatch Sequence Diagram

This sequence diagram allows the company to interact with the system to accept or reject a new match. To do this, they either receive a notification from the system, or they manually open the section with the found matches, and then decide whether to accept or reject the match. The system then notifies the corresponding student of the choice.

2.4.14 AcceptDeclineStudent

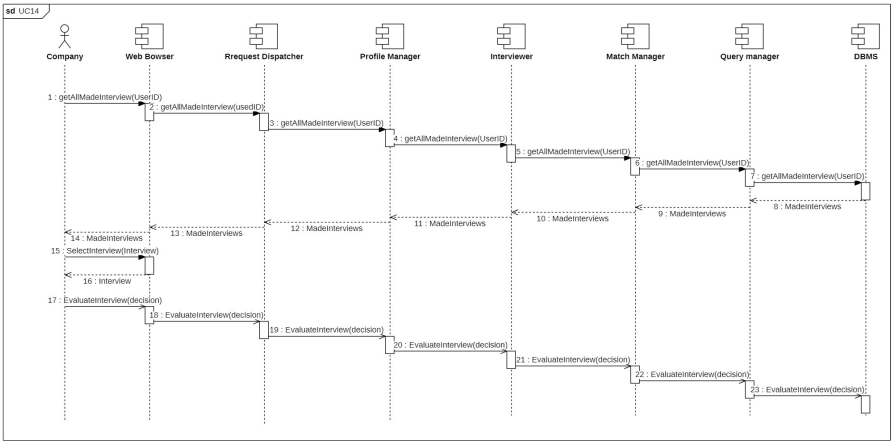

Figure 19: AcceptDeclineStudent Sequence Diagram

The sequence diagram in figure 19 shows how a company can accept or decline a student after the interview has been made. Once the wanted interview is selected, using the function EvaluateInterview, the decision (accepted/rejected) is passed as a parameter. The system will be then updated with that information and the user will eventually see the company decision

2.4.15 PublishComplaint company

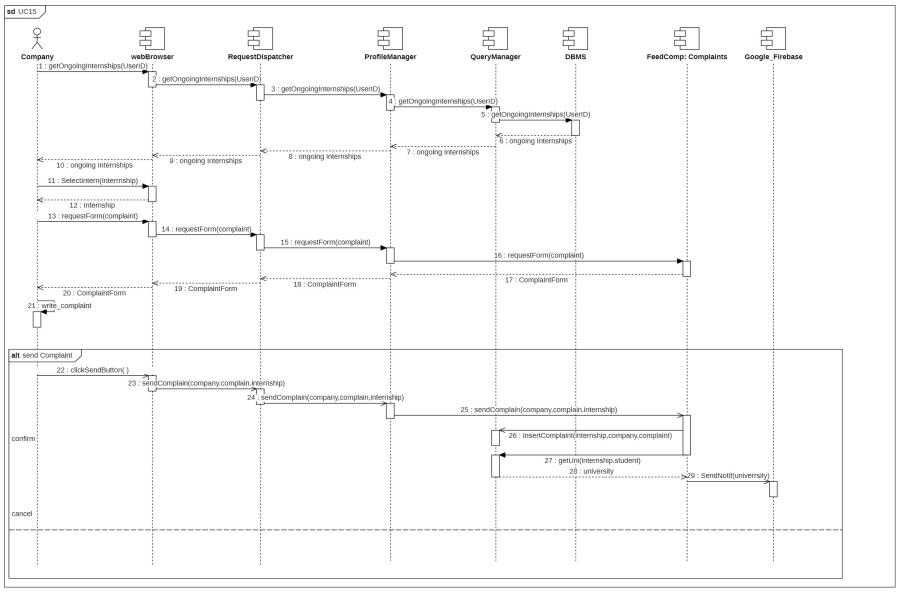

Figure 20: PublishComplaint Sequence Diagram

If a company has a complaint regarding an internship, it can select the needed one among all the others and start a complaint procedure (the function requestForm). Once the complaint is written, it's sent to the system which will upload it in the DB and send a notification to the university which will eventually take actions.

2.4.16 PublishFeedback company

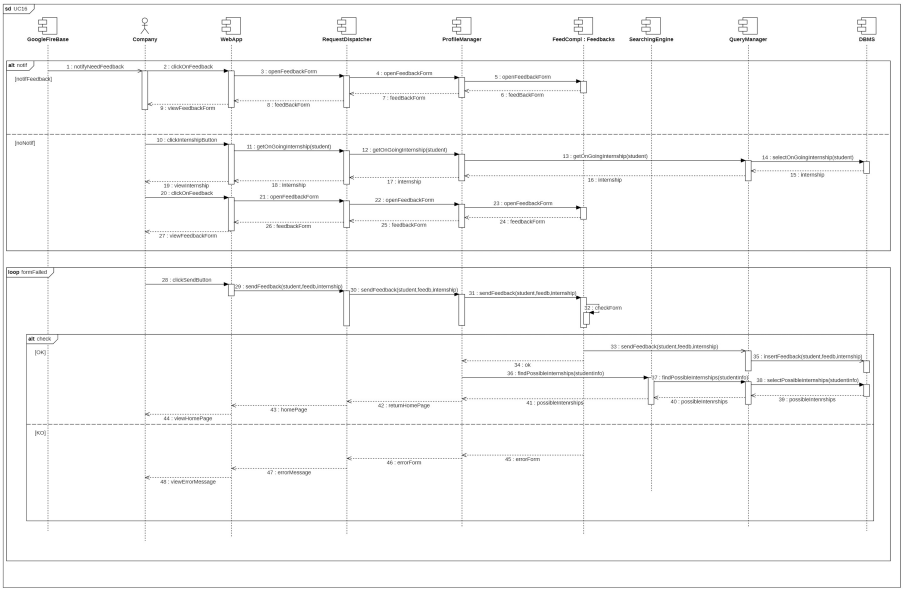

Figure 21: PublishFeedback Sequence Diagram

Once an internship is finished, the system will send a notification to both the user and the company, asking them to provide feedback about it. The "publishFeedback" procedure can begin by clicking the respective notification or by selecting the specific internship and calling the OpenFeedbackForm. Once the feedback is written, it is sent to the queryManager, which is in charge of uploading it to the DB. Simultaneously, the system uses the new information to update the internship suggestions.

2.4.17 MakeInterviews

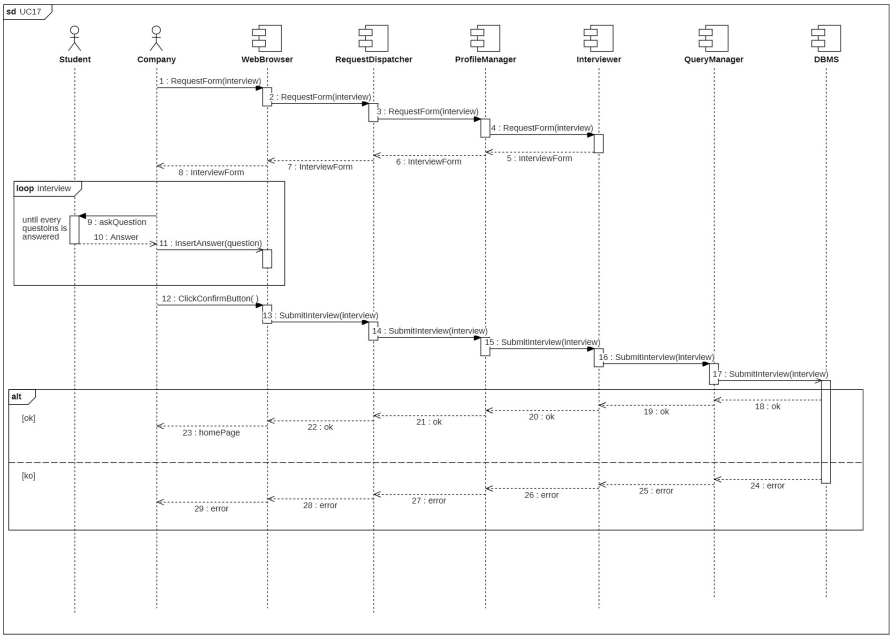

Figure 22: MakeInterviews Sequence Diagram

The interview process is described in the above diagram. Once the company has generated the interview form, it asks the form's questions to the user and fills out the form with the student's answers. Once the form is complete, it is sent to the query manager, which will save it to the DB.

2.4.18 Match making

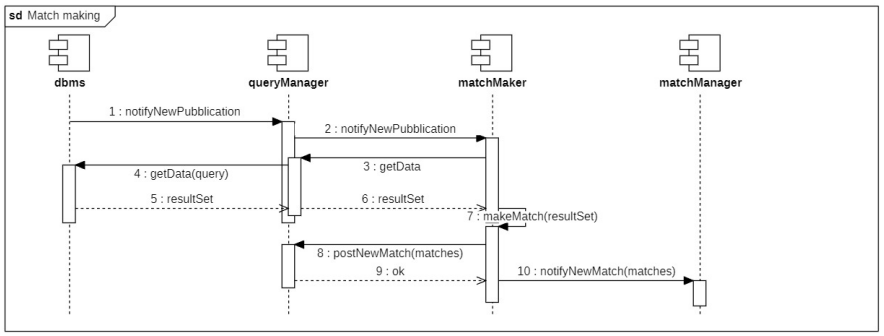

Figure 23: Match making Sequence diagram

This component is responsible for initiating matchmaking after a publication is inserted or modified in the database.

2.5 Component Interfaces

2.5.1 RequestDispatcher

1. sendFormLogin(email,psw)
2. getNewMatch(student)
3. getAllMatches(student)
4. openMatch(match)
5. acceptMatch(match)
6. declineMatch(match)
7. getOnGoingInternship(student)
8. openComplainForm()
9. sendComplain(user,complain,internship)
10. openComplain(student)
11. getAllComplains()
12. terminateInternship(student)
13. doNothing()
14. goToHomePage()

15. openFeedbackForm()
16. sendFeedback(student,feedb,internship)
17. getPublicationView()
18. getPublication(UserID)
19. sendCvForm(student,cv)
20. confirmCvSubmit(student,cv)
21. getPreferencesPublicationView()
22. sendPreferences(company, pref)
23. getCvModificationView(student)
24. deleteCv(student)
25. sendUpdateCv(student,cv)
26. sendChanges(student,changes)
27. getProjectPublicationView()
28. sendProjectForm(company, project)
29. getCompanyPage(company)
30. sendApplicationRequest(student,company)
31. getPublilcationModificationView()
32. deletePub(pub)
33. signUp(data)
34. update(pub)
35. commitChanges(pub,changes)
36. getAllMadeInterview(userID)
37. evaluate(interview)
38. getOngoingInternships(userID)
39. requestForm(formType)
40. SubmitInterview(interview)
41. DeletePub(pub)
42. Update(pub)

43. EvaluateInterview(decision)
44. getOngoingInternships(UserID)
45. confirmCvUpdate(student, cv)
46. confirmProjectSubmit(company,project)
47. getCvPubblicationView()

2.5.2 Auth:Registrator

1. SignUP(data)

2.5.3 Auth:Authenticator

1. sendFormLogin(email,psw)

2.5.4 Searching Engine

1. findPossibleInternships(student)
2. getCompanyPage(company)

2.5.5 Profile Manager

1. findPossibleInternships(student)
2. getNewMatch(student)
3. getAllMatches(student)
4. openMatch(match)
5. acceptMatch(match)
6. declineMatch(match)
7. getOnGoingInternship(student)
8. terminateInternship(student)
9. doNothing()
10. openComplainForm()
11. sendComplain(user,complain,internship)
12. openComplain(student)
13. getAllComplains()
14. openFeedbackForm()

15. sendFeedback(student,feedb,internship)
16. goToHomePage()
17. getPublicationView()
18. sendCvForm(student,cv)
19. confirmCvSubmit(student,cv)
20. getPreferencesPublicationView()
21. sendPreferences(company, pref)
22. getCvModificationView(student)
23. deleteCv(student)
24. sendUpdateCv(student,cv)
25. sendChanges(student,changes)
26. getProjectPublicationView()
27. sendProjectForm(company, project)
28. getPublilcationModificationView()
29. deletePublication(student)
30. getPublications(UserID)
31. DeletePub(pub)
32. Update(pub)
33. CommitChanges(pub,changes)
34. getAllMadeInterview(UserID)
35. EvaluateInterview(decision)
36. getOngoingInternships(UserID)
37. requestForm(formType)
38. submitInterview(interview)
39. confirmCvUpdate(student, cv)
40. confirmProjectSubmit(company,project)
41. getCvPubblicationView()

2.5.6 Publication Manager

1. getPublicationView()
2. sendCvForm(student,cv)
3. confirmCvSubmit(student,cv)
4. getPreferencesPublicationView()
5. sendPreferences(company, pref)
6. checkCV(student, cv)
7. getCvModificationView(student)
8. deleteCv(student)
9. sendUpdateCv(student,cv)
10. sendChanges(student,changes)
11. verifyChanges(student, changes)
12. getProjectPublicationView()
13. sendProjectForm(company, project)
14. verifyProject(company,project)
15. getPublilcationModificationView()
16. deletePublication(student)
17. getPublications(UserID)
18. DeletePub(pub)
19. Update(pub)
20. CommitChanges(pub,changes)
21. confirmCvUpdate(student, cv)
22. confirmProjectSubmit(company,project)
23. getCvPubblicationView()

2.5.7 Suggestioner

1. getSuggestion(student,cv)
2. getSuggestion(company, project)
3. Update(pub)

2.5.8 MatchMaker

1. notifyNewPubblication()
2. makeMatch(resultSet)

2.5.9 MatchManager

1. getNewMatch(student)
2. getAllMatches(student)
3. openMatch(match)
4. acceptMatch(match)
5. declineMatch(match)
6. add&notifyMatch(student, company)
7. notifyNewMatch(matches)
8. getAllMadeInterview(UserID)
9. evaluateInterview(decision)

2.5.10 QueryManager

1. sendFormLogin(email,psw)
2. findPossibleInternships(student)
3. getNewMatch(student)
4. getAllMatches(student)
5. openMatch(match)
6. acceptMatch(match)
7. declineMatch(match)
8. getOnGoingInternship(student)
9. terminateInternship(student)
10. sendComplain(student,complain,internship)
11. sendFeedback(student,feedb,internship)
12. openComplain(student)
13. getAllComplains()
14. postCv(cv)

15. postPreference(preference)
16. deleteCv(student)
17. postProject(company,project)
18. getCompanyPage(company)
19. deletePublication(student)
20. notifyNewPubblication()
21. getData()
22. postNewMatch(matche)
23. signUP(data)
24. registerNewUser(data)
25. getPublications(UserID)
26. DeletePub(pub)
27. CommitChanges(pub,changes)
28. getAllMadeInterview(UserID)
29. EvaluateInterview(decision)
30. getOngoingInternships(UserID)
31. InsertComplaint(internship,company,complaint)
32. getUni(internship.student)
33. submitInterview(interview)
34. postChanges(student,changes)

2.5.11 Feedbacks_Complaints:Feedbacks

1. openFeedbackForm()
2. sendFeedback(student,feedb,internship)

2.5.12 Feedbacks_Complaints:Complaints

1. openComplainForm()
2. sendComplain(user,complain,internship)
3. openComplain(student)
4. getAllComplains()
5. requestForm(formType)

2.5.13 Interviewer

1. getAllMadeInterview(UserID)
2. evaluatInterview(decision)
3. createInterviewForm()
4. requestForm(formType)
5. submitInterview(interview)

2.6 Selected architectural styles and patterns

2.6.1 3-tier Architecture

We decided to use a 3-tier architecture because:

1. it facilitates the separation of the system into independent layers (such as presentation, business logic, and data access) and tiers (including client, application server, and DBMS), making the system more modular and simpler to update or manage.
2. the distinction between layers and tiers simplifies scaling the system by allowing the distribution of workloads across several servers.
3. the architecture supports the implementation of load balancing and caching methods to enhance performance.

2.6.2 Client-Server Architecture

The system's behavior will primarily follow a Client-Server architecture, where the Client acts as the front-end user interface while the Server functions as the back-end platform, processing user requests, performing computations, and providing responses.
In a traditional Client-Server architecture, the Client always initiates the requests, while the Server remains a passive entity that processes these requests and returns the corresponding responses. However, in certain scenarios, such as interacting with the Google Firebase component to send notifications, the Server must execute actions without being prompted by the Client, making it an active component with a more Event-Driven approach.

2.6.3 REST API

We decided to implement a REST API to simplify communication between the client and server, as it provides numerous advantages. The API leverages standard web technologies (HTTPS), making it straightforward to use and integrate with other systems. Its stateless nature ensures that each request operates independently of others. Furthermore, using a REST API enhances scalability, as the separation between Client and Server allows for independent modifications or updates to each component.

2.6.4 Model-View-Controller pattern

The implementation of S&C will adopt the Model-View-Controller pattern, a software design approach that divides the software into three interconnected components:

- **Model**: includes the methods responsible for data management. It provides functionality for saving, retrieving, and modifying data in the database.
- **View**: focuses on the visual presentation of data to the end user.
- **Controller**: serves as the intermediary between the view and the model. It manages user interactions with the view and processes the corresponding operations.

2.7 Other Design Decisions

2.7.1 Availability

The incorporation of load balancing and replication mechanisms greatly improves the reliability and availability of our system. Load balancing maximizes resource efficiency by evenly distributing incoming requests among servers, thereby avoiding performance bottlenecks. At the same time, replication enhances fault tolerance by duplicating critical services. This redundancy reduces the effects of potential failures, ensuring that our system can sustain service availability even under adverse conditions.

2.7.2 Scalability

The Service-Oriented Architecture (SOA) promote scalability through several key principles. SOA structures an application as a collection of services, which can be independently deployed and managed. This modularity allows for selective scaling of individual services based on demand. For instance, if a specific service, such as order processing, experiences a surge in traffic, only that service needs to be scaled up, rather than the entire application.

2.7.3 Notification Handling

To send notifications to users of the app, we use an external tool called "GoogleFireBase". This tool allows us to send push messages directly to users when needed, without the app having to manage everything internally. For example, we can notify users about new match created or new complains. By integrating with GoogleFireBase, the process becomes much simpler and more efficient, enabling us to send targeted notifications quickly without building a complex solution from scratch.

2.7.4 Relational Database

We chose a relational database for our system design because it excels at storing structured data, maintaining data integrity, and delivering fast query performance. It is also scalable, capable of handling large volumes of data and supporting many simultaneous users. The database enables us to store and retrieve information efficiently, while ensuring that the data remains accurate and consistent.

2.7.5 Ease of Deployment

Adopting a SOA architecture offers a significant advantage in deployment ease. This approach allows changes to be made to individual services independently, enabling smooth deployments without affecting the entire system. This flexibility in deployment not only speeds up updates but also improves the system's resilience and agility, making troubleshooting and maintenance much more efficient.

3 User Interface Design

3.1 Overview

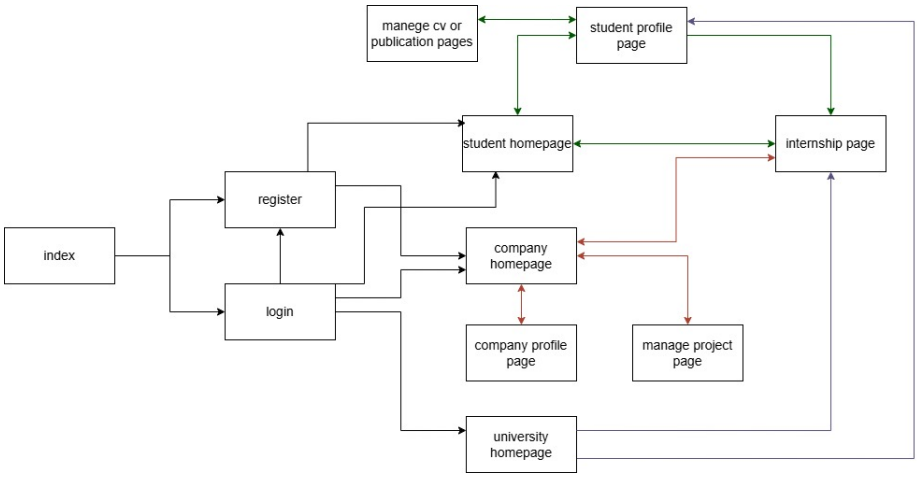

Figure 24: overview of the user interface

The figure 24 show how a user, with the correct rights, can navigate through the pages. The rights are showd by usign differents color, green for the student, red for the companies and then purple for the university.

3.2 Mockups

In this section there are the most important pages that show how the user interfaces will be.

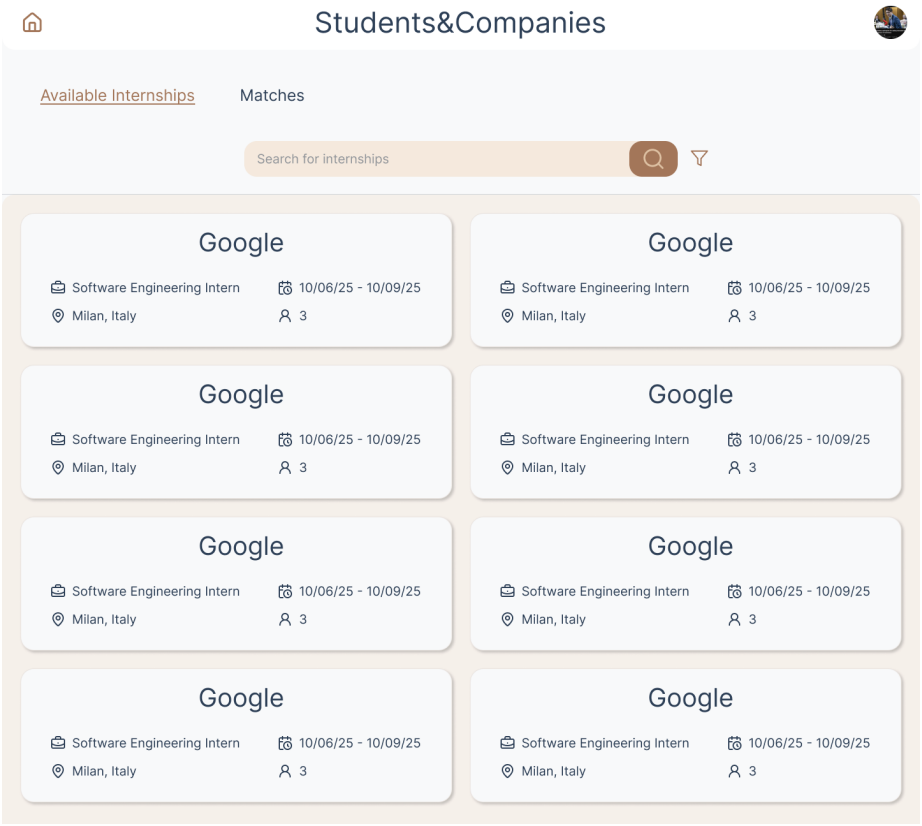

Figure 25: Student home page available internships

This page shows all available internships, not just those found through matches, but any internship published by companies. The student can also search a particular internship, for example if he is looking for a specific company. Clicking on his profile image, he can go through it's profile page, while clicking on one internship he can go through the internship page.

Figure 26: Student home page match sections

This page shows all available matches. The student can also search a particular internship, for example if he is looking for a specific company. Clicking on his profile image, he can go through it's profile page, while clicking on one internship he can go through the internship page.
The 'new matches' show all the matches found and not accepted by anyone yet. The 'Waiting for response' show all the matches accepted by the student but not accepted by the company yet. The 'Waiting for interview' show the match accepted by the company and by the student, but the company is making the interview and can still accept or decline the student.

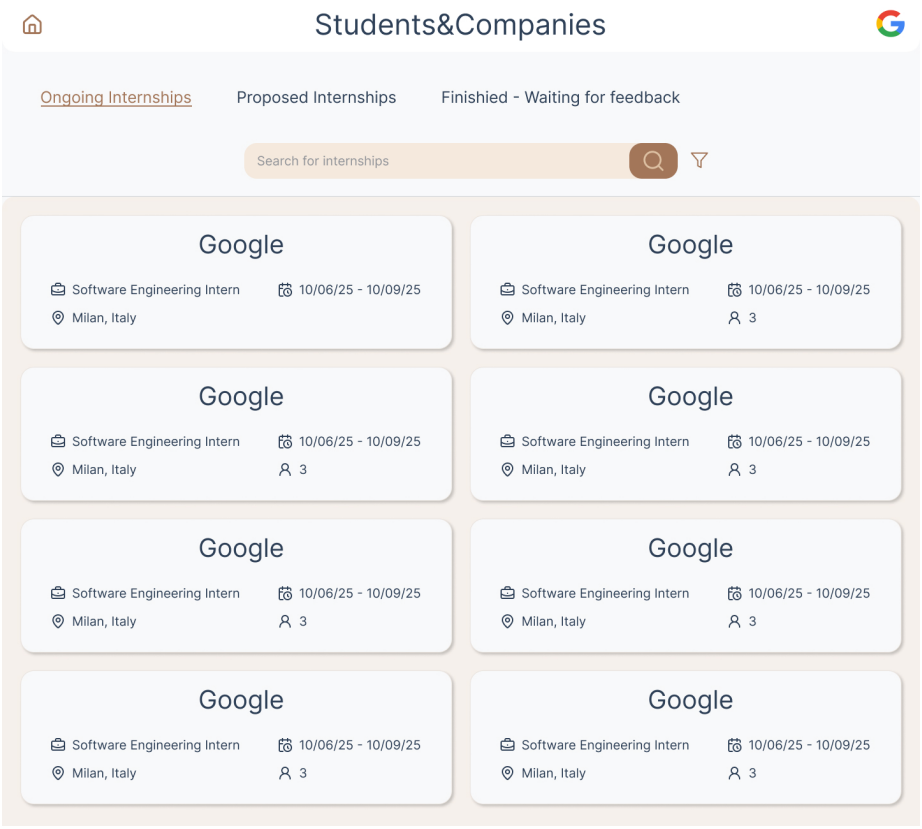

Figure 27: Company home page ongoing internships

This page shows all the ongoing internships. Clicking on an internship the company will see the page with all the information about that ongoing internship. It can also goes to the 'Proposed Internships' and 'Finished - Waiting for feedback' sections.

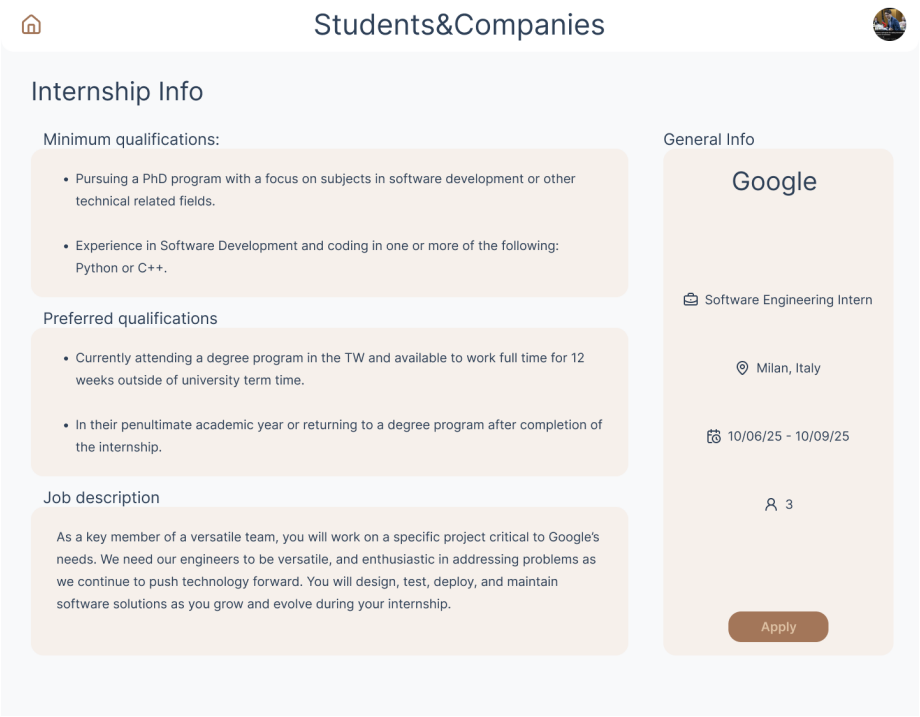

Figure 28: Internship view seen by a student

This page shows the info about an internship published by a company and give to the student the opportunity to apply to it.

4 Requirements Traceability

Auth:Registrator

R1 The system shall allow unregistered users to create an account on S&C only if their university already has an account on the platform.

R3 The system shall allow the universities to request an account on the platform.

Auth:Authenticator

R2 The system shall allow registered users to log in to their accounts.

Searching Engine

R18 The system shall allow the students to browse through the available internships

Profile Manager

R16 The system shall allow the universities to stop the on going internships involving their students

R32 The system shall allow the universities to see their own students and their ongoing internships

Publication Manager

R4 The system shall allow the companies to publish their internships

R5 The system shall allow the companies to modify their internships

R6 The system shall allow the companies to delete their internships

R7 The system shall allow the students to publish their CVs and their internship's preferences

R8 The system shall allow the students to modify their CVs and their internship's preferences

R9 The system shall allow the students to delete their CVs and their internship's preferences

R17 The system shall allow the students to update their CVs

Suggestioner

R27 The system shall suggests students on how to improve their CVs

R28 The system shall suggest companies on how to improve their internships proposal

MatchMaker

R29 The system shall analyse the students pubblications in order to find a match with the right companies

R30 The system shall analyse the companies pubblications in order to find a match with the right students

R31 The system shall create matches using different level of sophis tication, from simple keyword-search to statistical analyses based final feedbacks made by students and companies

MatchManager

R10 The system shall allow the students to view all the matches found

R11 The system shall allow the companies to view all the matches found

R12 The system shall allow the companies and the students to accept or decline the proposed matches

R19 The system shall allow the students to directly make an application to the available internships

Feedbacks_Complaints:Feedbacks

R22 The system shall allow the students to give the final feedback

R23 The system shall allow the companies to give the final feedback

Feedbacks_Complaints:Complaints

R15 The system shall allow the universities to manage the complaints which involves their students

R20 The system shall allow the students to send complaints

R21 The system shall allow the companies to send complaints

Interviewer

R13 The system shall allow the companies to create the interview form.

R14 The system shall allow the companies to add responses of the students during the interview, to the form.

GoogleFireBase

R24 The system shall notify the students whenever a new match is found

R25 The system shall notify the universities whenever a new complaint involving one of its students is submitted

R26 The system shall notify the companies whenever a new match is found

5 Implementation, Integration and Test Plan

5.1 Overview

The implementation, integration and test strategies are realized by exploiting the bottom up approach, which allows to start the implementation from the "leaves" modules, which are the ones that don't use other modules to work, and this allows also to integrate and test in an incremental way which is useful to find in early stage the bugs deriving from the integration of the single modules and to test each one individually and after the integration.

5.2 Implementation

The application is composed by 3 tiers, so the implementation can be realized in parallel on each of them. The following sections will be focused on the implementation of application and view tiers.

5.2.1 Application tier

The tier will be implemented starting from the "leaves" up to the "root", some of the modules can be implemented in parallel if they are on same level which means that all the modules that they use are already implemented, integrated and tested. The order of implementation will be divided in layer, each one has to be implemented in order and inside all the modules can be implemented in parallel or sequentially in any order:

- First layer:
  - Suggestioner
  - Query Manager
- Second Layer:
  - Match Manager
  - Match Maker
  - Search Engine
  - Auth
  - Publication Manager
  - Interviewer
  - Feedbacks and Complaints
- Third layer:
  - Profile Manager
- Fourth layer
  - Request Dispatcher

5.2.2 View tier

The view can be implemented in parallel by using some REST mockup in order to simulate some state of the application, thanks to the rest mockup each page can be implemented in parallel or by using any sequential order.

5.3 Integration

The integration of components should start as soon as the component of the first layer are ready. As explained before the integration will follow a bottom-up approach.

5.3.1 First layer

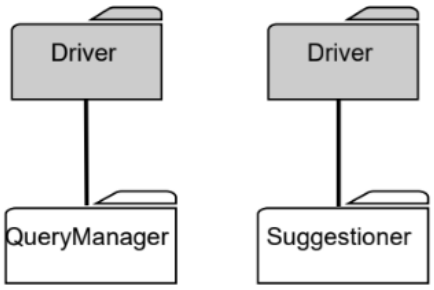

Figure 29: First layer integration

As soon as the DB is ready, we can implement the first layer that will focus on the communication with the DB and the creation of suggestions based on data coming from the drivers.

5.3.2 Second layer

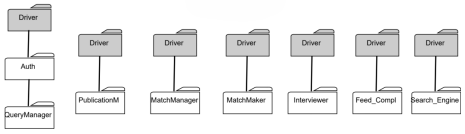

Figure 30: Second layer integration

The integration of the second layer allow to test all the basic functions of the system, using a different driver for each component in order to obtain and test the right actions triggered by the right data.

5.3.3 Intermediate step

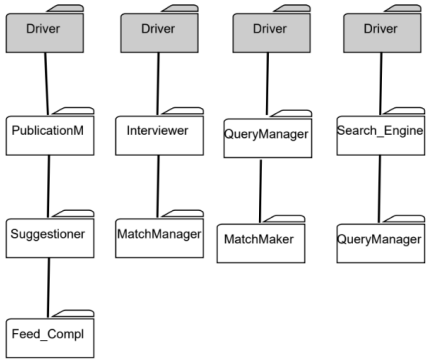

Figure 31: Intermediate integration step

Before going on with the third layer integration, it's possible to integrate some components in order to test other functions that couldn't have been tested before, like the entire mechanism of creating the CV and internship suggestions or the mechanism of making interviews.

5.3.4 Third layer

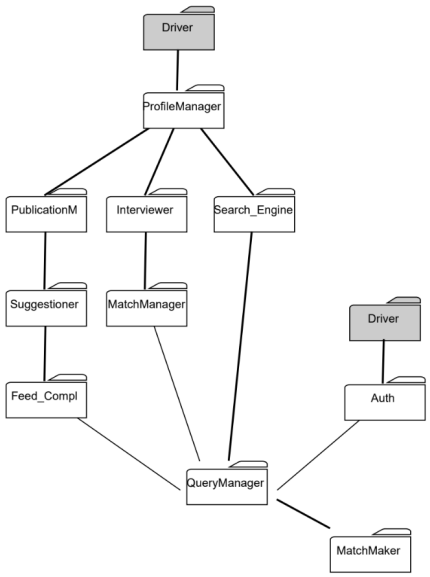

Figure 32: Third layer integration

This layer is used to integrate the ProfileManager component, allowing to test the remaining part of the system and also to integrate almost all the system with just two drivers.

5.3.5 Fourth layer

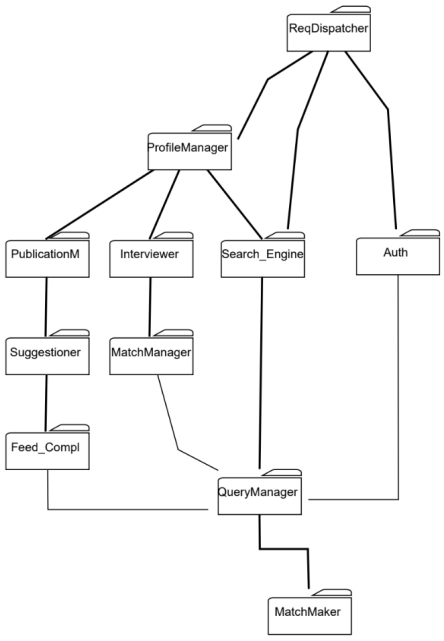

Figure 33: Fourth layer integration

This is the last layer to integrate, RequestDispatcher, the component used to dispatch all the requests from the user. So, integrating this component means all the system is integrated and ready to be tested entirely.

5.4 Testing Strategy

To ensure the correctness of the software down to its fundamental components a **bottom-up** strategy will be adopted. As soon as a new component is developed and integrated, it is tested to ensure that it works properly. Once all the components have been integrated, an additional series of tests will be done. In particular:

- Functional testing : every functional requirement is tested to hold
- Load testing : The software is meant to be used by a large number of users, making it crucial to ensure its functionality

6 Effort Spent

The table below show the number of hours that each member of the group worked for the RASD document.

Member	Hours
Elia Priuli	20h
Veronica Viceconti	21h
Marco Zuccoli	24h

7 References

1. Software Engineering II course slides
2. Draw.io
3. Figma to create User Interface Design
4. StarUML