

Students & Companies

Design Document

Francesco Ostidich, Matteo Salari, Francesco Rivitti

February 10, 2025

Contents

1	Introduction	2
1.1	Purpose	2
1.2	Scope	2
1.3	Definitions, acronyms, abbreviations	2
1.3.1	Definitions	2
1.3.2	Acronyms	3
1.3.3	Abbreviations	3
1.4	Revision history	3
1.5	Reference documents	3
1.6	Document structure	3
2	Architectural design	5
2.1	Overview	5
2.2	Components view	5
2.2.1	Logical description of data	7
2.3	Deployment view	8
2.4	Runtime view	9
2.5	Component interfaces	19
2.5.1	Application APIs	19
2.5.2	Interface methods	26
2.6	Architectural styles and patterns	30
2.7	Other design decisions	31
3	User interface design	32
3.1	Home pages	32
3.2	Profile pages	34
3.3	Notifications pages	36
3.4	Help pages	38
4	Requirements traceability	40
5	Implementation, integration and test plan	48
5.1	Development process and approach	48
5.2	Implementation and integration plan	48
5.2.1	Application server	48

5.2.2	Web server	51
5.2.3	Final test	51
5.3	Development technologies	52
5.3.1	Web server	52
5.3.2	Application server	52
5.3.3	Database	52
5.3.4	Architectures and patterns	53
5.4	Technologies used for testing	53
5.4.1	Backend unit testing	53
5.4.2	Frontend unit testing	53
5.4.3	REST APIs requests fabrication	53
5.4.4	REST APIs responses simulation	53
5.4.5	Web pages requests fabrication	53
6	Effort spent	54
7	References	55

1 Introduction

1.1 Purpose

This document contains the design description of the Students&Companies system. It includes the architectural design, the user interface design, and the descriptions of all the operations that the system will perform. It also shows how the requirements and the use cases detailed in the RASD document are satisfied by the design of the system. This document is intended to be read by the developers, the testers and the project managers of the system. It is also intended to be used as a reference for any future maintenance.

1.2 Scope

Students&Companies (S&C) is a platform designed to connect university students with companies offering internships. It simplifies the internship searches of students and the projects advertisement for companies. The platform employs recommendation mechanisms to match students and companies based on experience, skills, and project requirements. S&C also supports the selection process by managing interviews and collecting feedbacks. Additionally, it provides suggestions for improving CVs and project descriptions.

A more detailed description of the system can be found in the RASD document. This document provides a detailed description of the design that is to implement the requirements and the use cases found in the RASD document.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- **Internship project:** the description of the skills, technologies and roles the student will be working with during the internship, along with the set of tasks that will be covered
- **Internship advertisement:** the public post created by companies to promote available internships on the platform, aimed at attracting suitable candidates by highlighting its key aspects
- **Internship information:** general data about the (ongoing) internship, including the elapsed and remaining time, the compensation and the description of the project the student is working on
- **Enrollment request:** the submission of a student to indicate interest in a specific internship, initiating the selection process by formally applying

- **Enrollment suggestion:** the recommendation made by the platform to guide students in finding projects that best suit them
- **Custom questionnaire:** the tailored set of questions used by companies during interviews to assess a candidate fit for the internship
- **Candidate student:** a student who has applied for an internship and is currently under consideration by a company, moving forward in the selection process
- **Eligible student:** a student who meets the qualifications for an internship, making them viable candidates for recommendation and application
- **Suitable student:** a student who meets the qualifications for an internship, making them potential candidates to be recommended in the companies feed
- **Complaint:** a report submitted by a student or company to the university, regarding issues during the internship, such as unmet expectations, mistreatments, or procedural problems
- **Feedback form:** a structured form for students and companies used to provide feedback on their internship experience, enabling the platform to gather data for analysis, improvements, and recommendations

1.3.2 Acronyms

- **S&C:** Students&Companies

1.3.3 Abbreviations

- **Rn:** n-th requirement
- **Cn:** n-th component
- **RVn:** n-th runtime view

1.4 Revision history

- **Revised on:** February 10, 2025
- **Version:** 2.0
- **Description:** updated APIs, interfaces, DB schema and runtime views

1.5 Reference documents

- **Polimi Software Engineering 2 AY 2024/2025 assignment document:** goal, schedule and rules of the requirement engineering and design project
- **Polimi Software Engineering 2 AY 2024/2025 course slides:** the lecture slides provided during the course

1.6 Document structure

- **Chapter 1:** this section provides a brief description of the purpose and the scope of the system; moreover it contains the definitions, acronyms and abbreviations used in the document.

- **Chapter 2:** this section provides a description of the architecture of the system, including the components and the interfaces between them; it also includes the runtime view of the most important operations of the system, along the deployment view and the architectural styles and patterns used.
- **Chapter 3:** here are included the mockups of the user interfaces.
- **Chapter 4:** this section shows how the requirements described in the RASD document are satisfied by the design implementation.
- **Chapter 5:** here is included a step-by-step plan for the implementation and testing of the system.
- **Chapter 6:** here is found an estimation of the effort spent by each group member.
- **Chapter 7:** here is provided a list of the references used in this document.

2 Architectural design

2.1 Overview

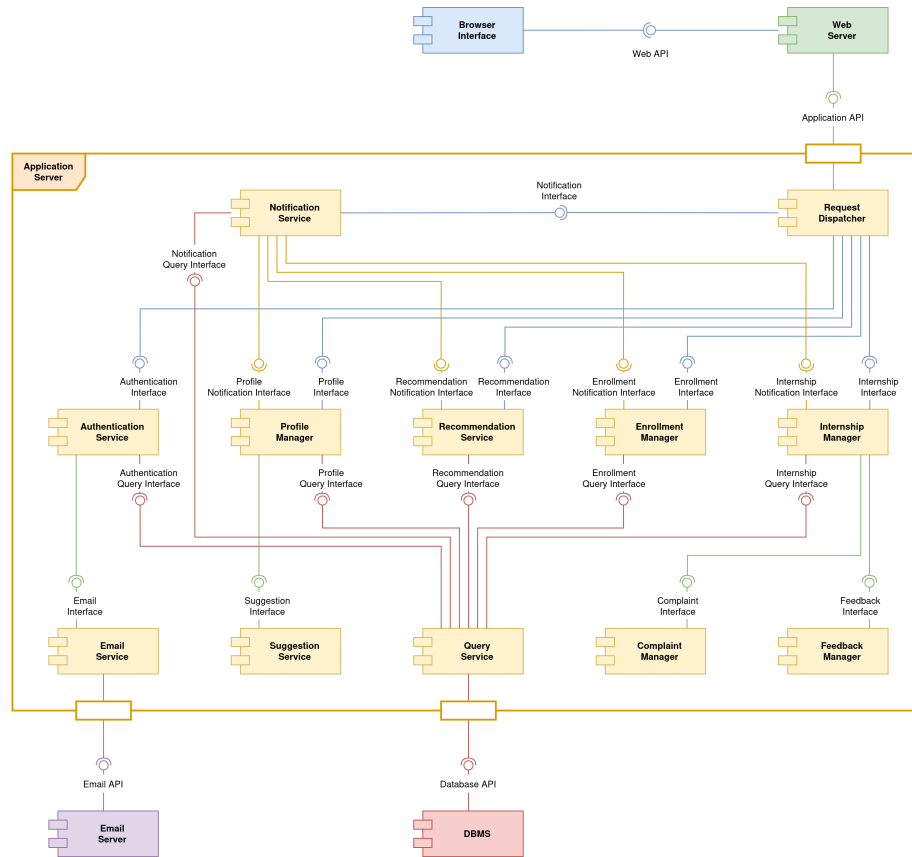
In the S&C application, five high-level components can be distinguished: the web server, the application server, the DBMS, the email service, and the browser interface.

The web server main objective is to deliver web pages upon user requests. The application server is the main component of the system, as it contains every piece of business logic required to satisfy the goals specified in the RASD document. The role of the DBMS is the usual, which is providing interaction with a database by executing command operations. The email service is used to validate the user account at sign up time, since the verification link is sent via email. Lastly, the browser interface is the application, in the form of web pages, that is reachable by the user from a browser.

2.2 Components view

- C1 - Browser interface** - This is the the web application used by the client, which is reachable by any browser. The full range of operations that it provides can only be used after the authentication process, performed by the authentication service, results successful. It allows both students and companies to perform a certain set of actions based on the type of user.
- C2 - Web server** - This is the intermediary between the frontend and the backend. Its purpose is to manage the browser UI pages independently and transparently from the point of view of the other servers components. The web server main tasks are the static resources delivery and the web pages content retrieval and forwarding. After receiving an HTTP request, the web server retrieves from the application server the data needed to fill the corresponding HTML template, which are then both returned to the user, along with the static style assets.
- C3 - DMBS** - The DBMS, which manages the DB containing all the data that the platform utilizes, stays on a separate machine. It is then the query service component that, by providing the necessary SQL queries to other components, allows the application server to reach for the stored data.
- C4 - Email server** - The email server represents the external machines that the email service reaches in order to send messages. Those machines are property of the actual email providers, but they show how the application server email service component can utilize them via their APIs.

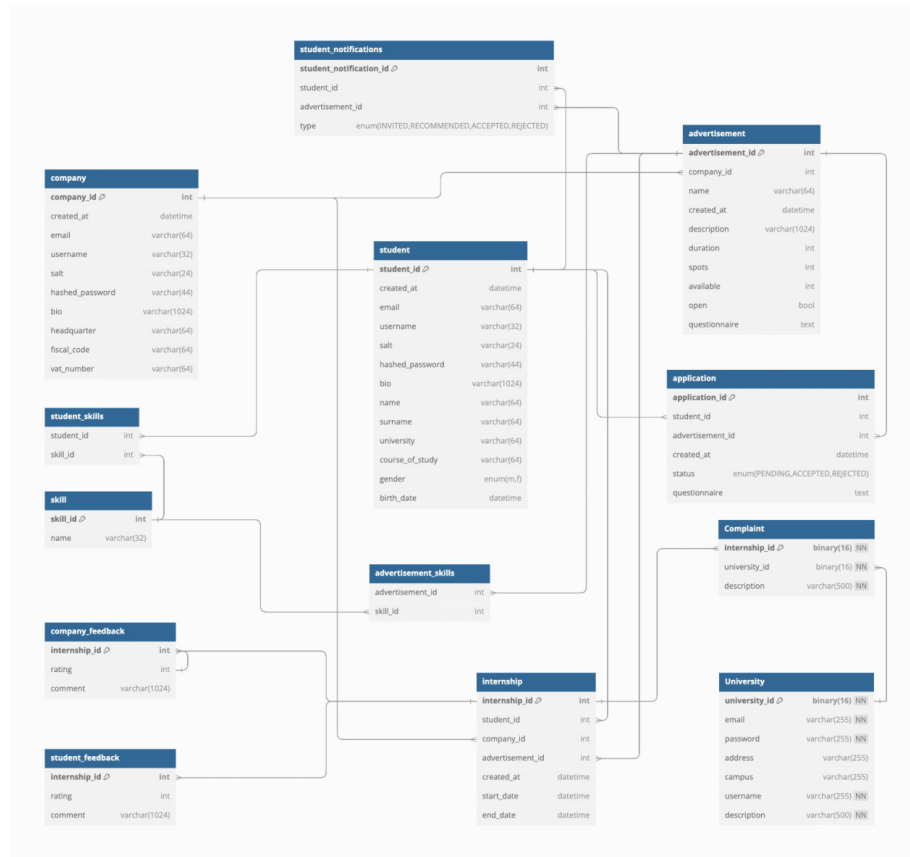
- C5 - Request dispatcher** - The web server processes incoming HTTP requests by calling the appropriate application server component. The components are responsible for retrieving the necessary data that the web server uses to populate the HTML template, which is then returned to the user. Since different web pages may require data from different components, the request dispatcher ensures that each request is routed to the correct application server component.
- C6 - Authentication service** - This component provides the set of procedures required to handle the authentication of a user into the system. It manages the log in and sign up phases.
- C7 - Email service** - The main goal of this component is to implement all the procedures that the server requires in order to send email messages to the user. Mainly, this component allows the authentication service to send a verification email when handling the sign up of a user.
- C8 - Notification service** - The purpose of this component is to collect and manage all the notifications that a user receives. It defines the notification types and allows other components to easily send them.
- C9 - Query service** - This service acts as a mediator between the server components and the DBMS. It uses the DBMS APIs to implement a set of functions which have the sole purpose of manipulating the database or retrieving information from it.
- C10 - Recommendation service** - This service provides the algorithms necessary for finding suitable advertisements and candidates. These are called when the user opens the feed in the home page, or when looking for strong matches that are to be notified.
- C11 - Suggestion service** - This service provides the algorithms necessary for evaluating the user profiles, in order to propose enhancements. These are called by the profile manager component when the user provides new profile information.
- C12 - Profile manager** - Users must be able to insert and edit their profile information. Validity checks and DB updates are performed by this component. The query service and the suggestion service APIs are used when data is to be updated.
- C13 - Enrollment manager** - This component manages all the selection process phases, from the student application request to the start of the internship. It also handles the questionnaires that companies send to students to fill in.
- C14 - Internship manager** - This component handles the operations that the internship may allow or require when it is in the ongoing status. Along initializing and interrupting the internship, it permits users to visualize its annex information.
- C15 - Complaint manager** - This component allows students and companies to send complaints regarding an ongoing internship, hence allowing the university to visualize and handle them. Since complaints must have a valid reference to an ongoing internship, the internship manager component APIs are here used.
- C16 - Feedback manager** - When an internship ends, both parties are requested to fill in a feedback form. This component handles this operation, by sending, receiving and acting accordingly to the feedback forms content. Since feedback forms must have a valid reference to an internship that has concluded, the internship manager component APIs are here used.



2.2.1 Logical description of data

The following diagram presents the logical design of the database, illustrated through an entity-relationship (ER) model. The diagram defines the entities, their attributes, and the relationships between them, providing a structured and coherent representation of the system data organization.

The design ensures clarity in how information is managed, emphasizing integrity, consistency, and minimal redundancy. It serves as a comprehensive framework for understanding the logical structure of the data and its interactions within the system.



2.3 Deployment view

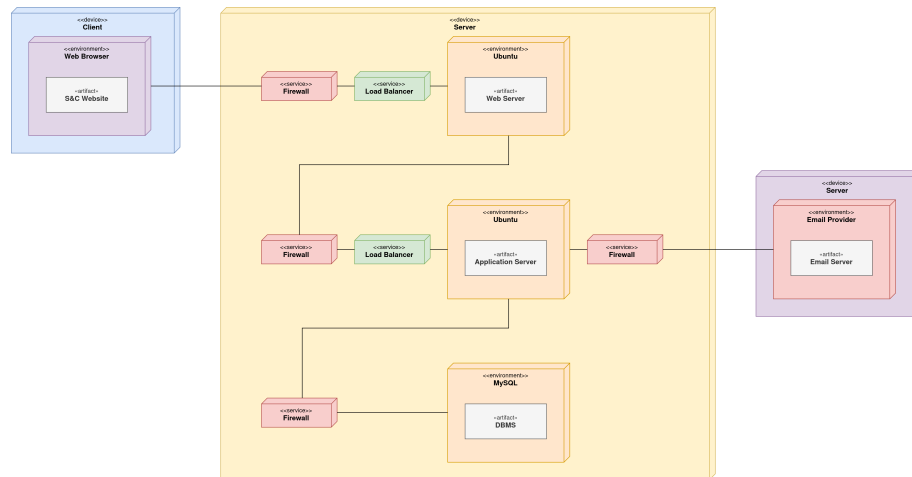
The following diagram shows the deployment view of the system. It illustrates the distribution of the components on the different nodes, and how they communicate between each others.

The system is composed of four tiers, one for the client frontend and three for the server full-stack processes: the client tier, the web tier, the application tier and the data tier. The client tier corresponds to the web browser, which is required by the user to access the application. The web tier represents the web server, which is used to handle incoming user requests and to return web pages and their content. The application tier corresponds to the application server, that is used to execute the business logic and to communicate with the database. The data tier is composed of the database, which is used to store all the data that the application server may require.

The email service is a set of external APIs used by the application to send emails to the users.

Moreover, since the platform will potentially need to manage a lot of concurrent users, the implementation of load balancers can help distribute the load among multiple servers. This allows the application to be more scalable, since it eases the addition of other servers to handle an increasing load. Those are placed between the client and the web server, and between the web server and the application server.

Lastly, to protect the application from external attacks, multiple firewalls are in use to filter the traffic. In particular, those are placed between the client and the web server, between the web servers and the application server, before the database, and between the application server and the provider's email server. This allows the application to execute in a more secure environment, since the firewalls will filter the traffic before it reaches the designated component.

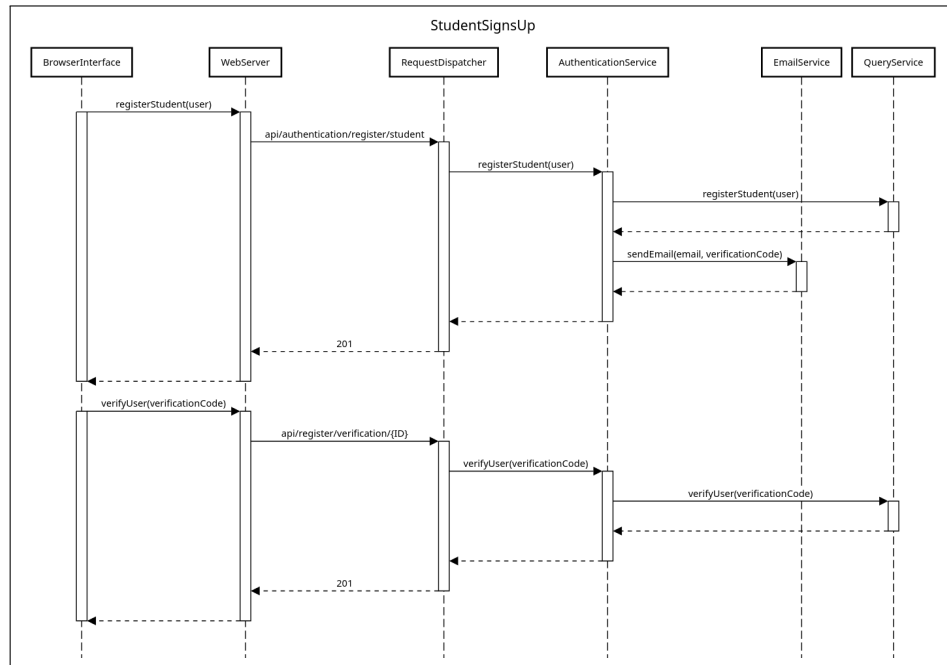


2.4 Runtime view

This section contains the sequence diagrams of the most important operations of the system. The diagrams include the component described in the previous sections, and the external components that are involved in the operations.

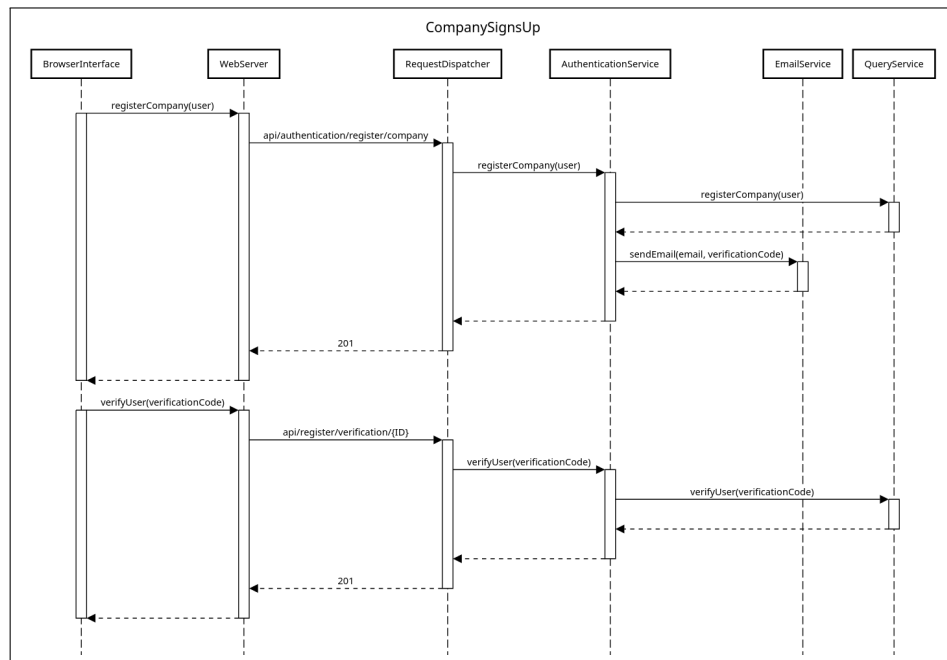
RV1 - StudentSignsUp

To register into the system, the user fills in the registration form and submits it. The registration form fields vary based on whether signing up it is a student or a company. The whole process is mainly handled by the authentication service component, that interacts with the query service to validate the information and to insert the new user into the DB. The application server checks that information is valid, and that the user is not already registered. Then, the system will insert the user into the DB and send an email to the user. The email contains a link with the verification code. The user, by clicking on the link, confirms the registration.



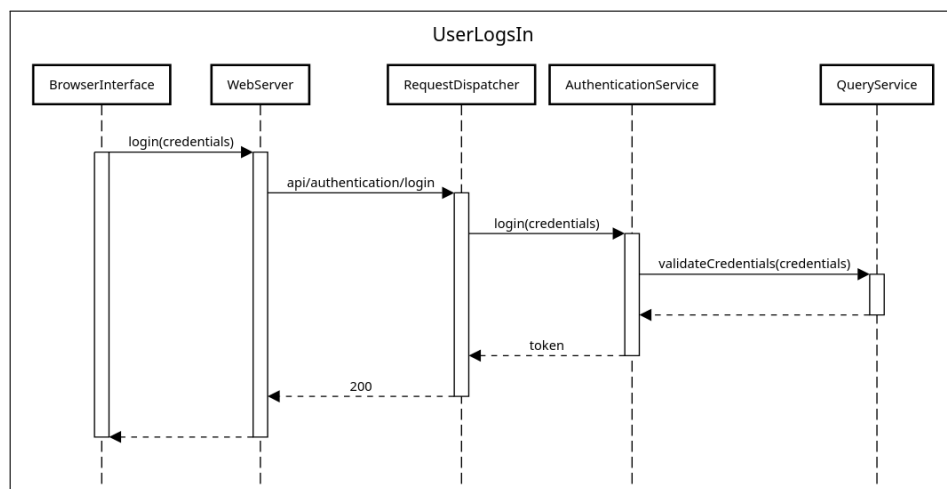
RV2 - CompanySignsUp

To register into the system, the user fills in the registration form and submits it. The registration form fields vary based on whether signing up it is a student or a company. The whole process is mainly handled by the authentication service component, that interacts with the query service to validate the information and to insert the new user into the DB. The application server checks that information is valid, and that the user is not already registered. Then, the system will insert the user into the DB and send an email to the user. The email contains a link with the verification code. The user, by clicking on the link, confirms the registration.



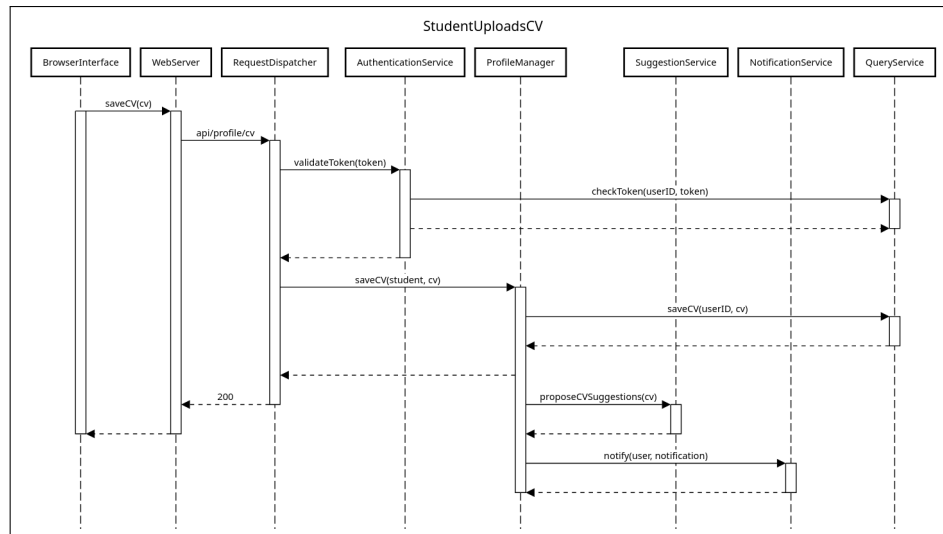
RV3 - UserLogIn

To log in into the system, the user has to fill in the credentials fields and submit it. This process is the same for both students and companies. The whole process is handled by the authentication service component, that interacts with the query service to validate the information. Once the user is logged in, the authentication service will generate a token, which is sent to the user. With the token, the user can access the other website pages reserved for logged users.



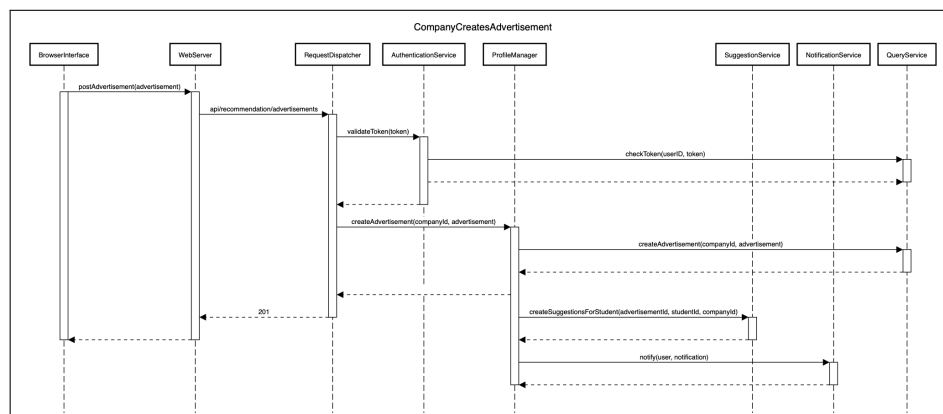
RV4 - StudentUploadsCV

To upload the CV, the student is before hand authenticated by means of the token found in the header of the request. Then, via the profile manager component, the CV is accepted, validated, and inserted into the file storage of the system. The suggestion service is called afterwards, and if valid suggestions are found for the CV, they are notified to the student.



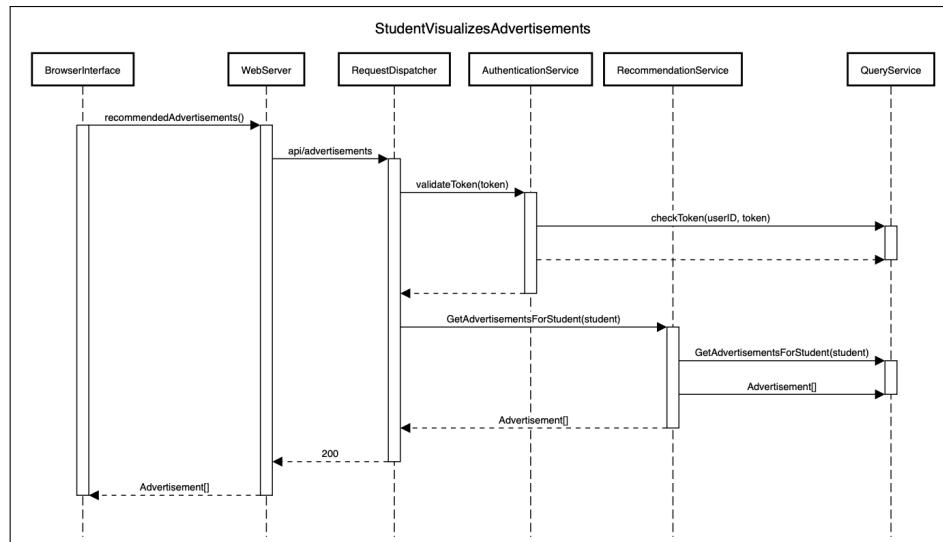
RV5 - CompanyCreatesAdvertisement

To create an advertisement, the company is before hand authenticated by means of the token found in the header of the request. Then, via the profile manager component, the advertisement is accepted, validated, and inserted into the DB. The suggestion service is called afterwards, and if valid suggestions are found for the advertisement, they are notified to the company.



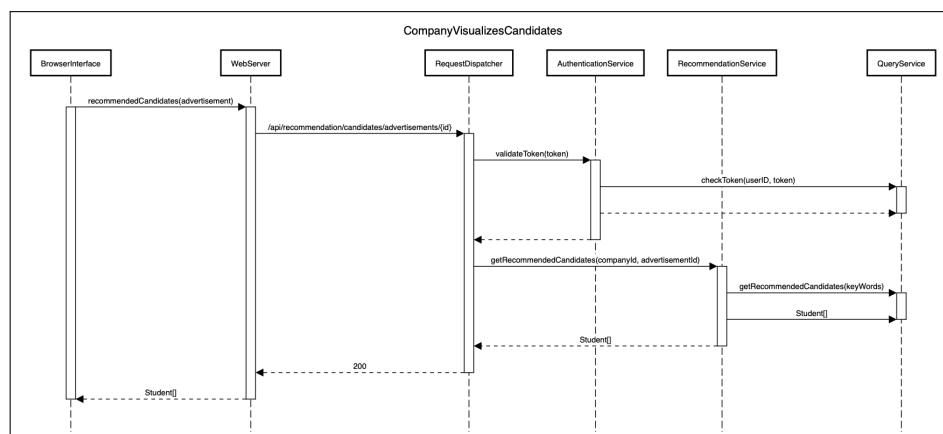
RV6 - StudentVisualizesAdvertisements

To visualize advertisements in the feed page, the student is before hand authenticated by means of the token found in the header of the request. Then, the recommendation service uses keywords found in the student profile to find matching advertisements in the DB. Those are returned to the student, which finds the feed page populated with the advertisements.



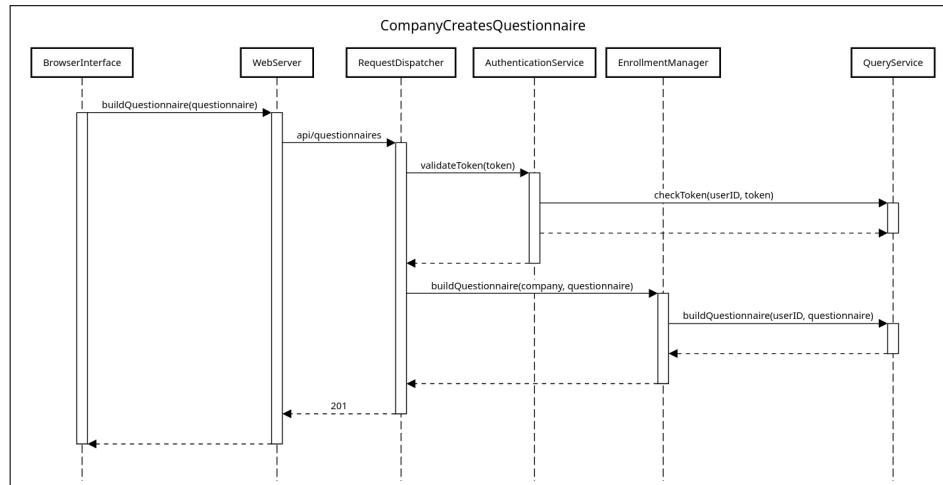
RV7 - CompanyVisualizesCandidates

To visualize eligible students in the feed page, the company is before hand authenticated by means of the token found in the header of the request. Then, the recommendation service uses keywords found in the company profile and in the selected advertisement to find matching student candidates in the DB. Those are returned to the company, which finds the feed page populated with the students information.



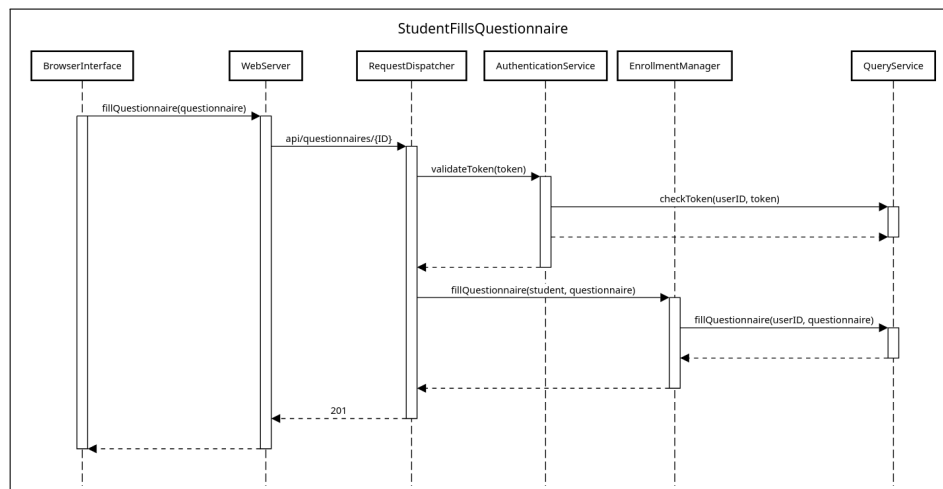
RV8 - CompanyCreatesQuestionnaire

To build a new questionnaire, the company is before hand authenticated by means of the token found in the header of the request. Then, via the enrollment manager component, the new questionnaire is accepted, validated, and inserted into the DB.



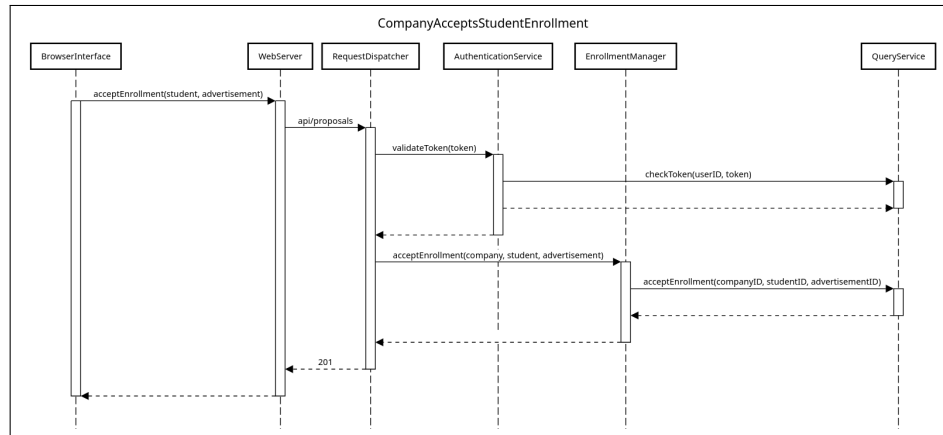
RV9 - StudentFillsQuestionnaire

To fill in a questionnaire, the student is before hand authenticated by means of the token found in the header of the request. Then, via the enrollment manager component, the filled in questionnaire is accepted, validated, and inserted into the DB.



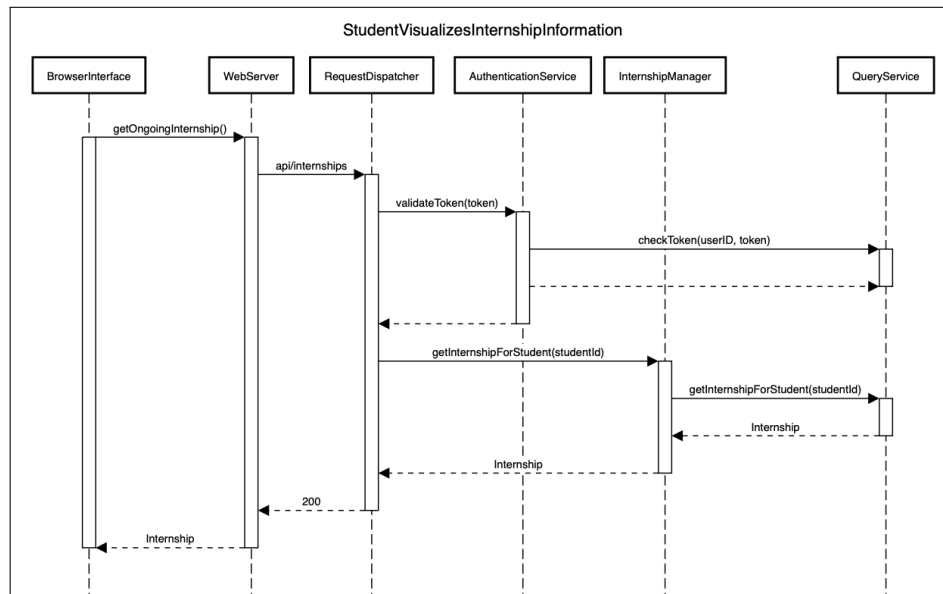
RV10 - CompanyAcceptsStudentEnrollment

To accept the enrollment request made by a user, the company is before hand authenticated by means of the token found in the header of the request. Then, via the enrollment manager component, the internship application, relative to a specific advertisement, is accepted, validated, and inserted into the DB.



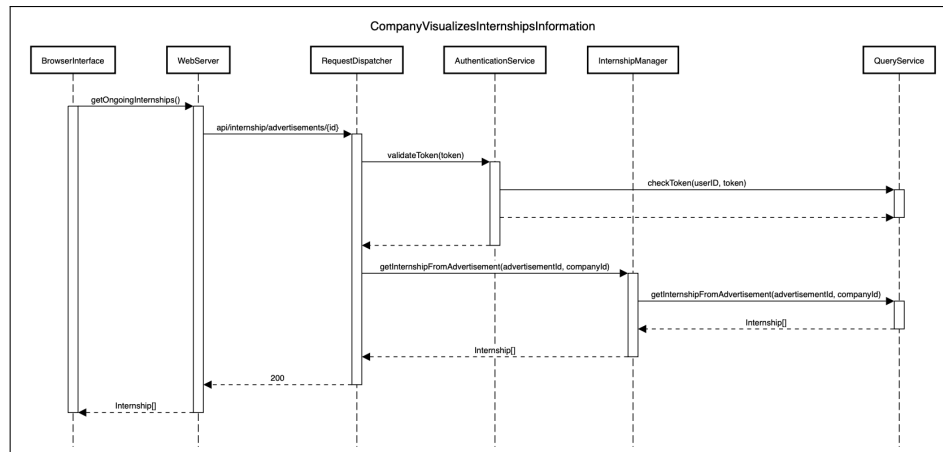
RV11 - StudentVisualizesInternshipInformation

To visualize information about its ongoing internship, the student is before hand authenticated by means of the token found in the header of the request. Then, via the internship manager component, the information about the internship is queried and returned to the student.



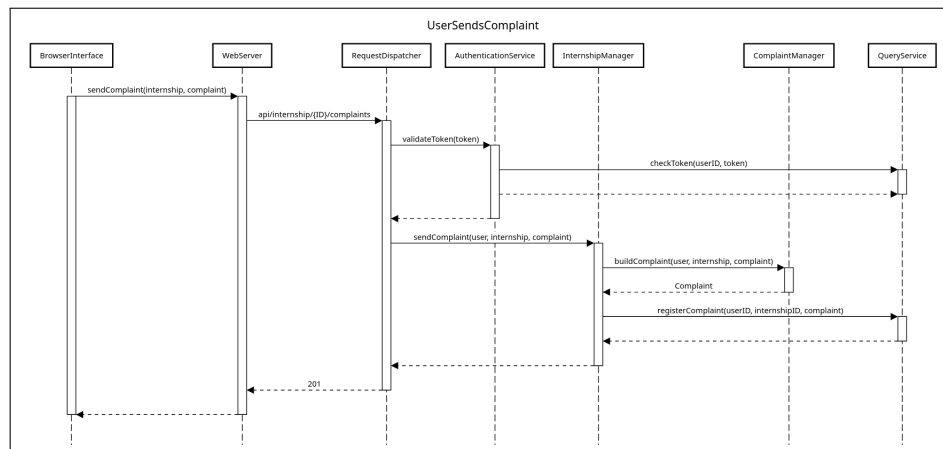
RV12 - CompanyVisualizesInternshipsInformation

To visualize information about its ongoing internships, the company is before hand authenticated by means of the token found in the header of the request. Then, via the internship manager component, the information about the internships is queried and returned to the company.



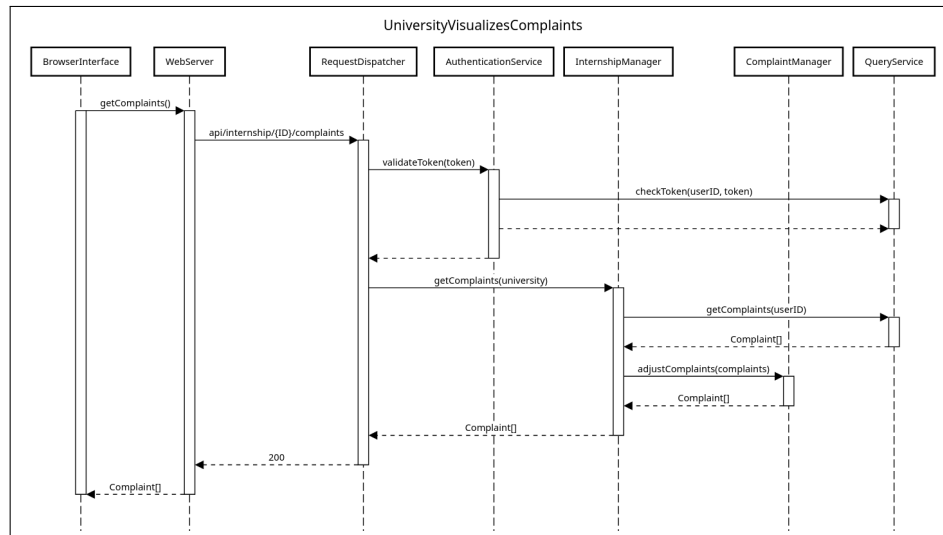
RV13 - UserSendsComplaint

To send a complaint relative to an ongoing internship, the participant is before hand authenticated by means of the token found in the header of the request. Then, via the internship manager and the complaint manager components, the complaint is accepted, validated and inserted into the DB. The university will then be notified.



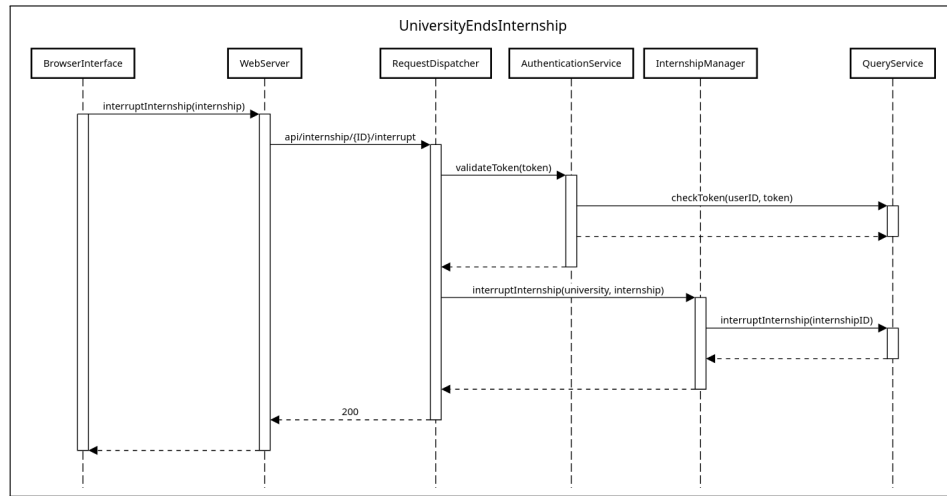
RV14 - UniversityVisualizesComplaints

To visualize the complaints, the university is before hand authenticated by means of the token found in the header of the request. Then, via the internship manager and complaint manager components, the complaints relative to its students internships are queried from the DB. Those are returned to the university, which finds the complaints page populated with the complaints messages.



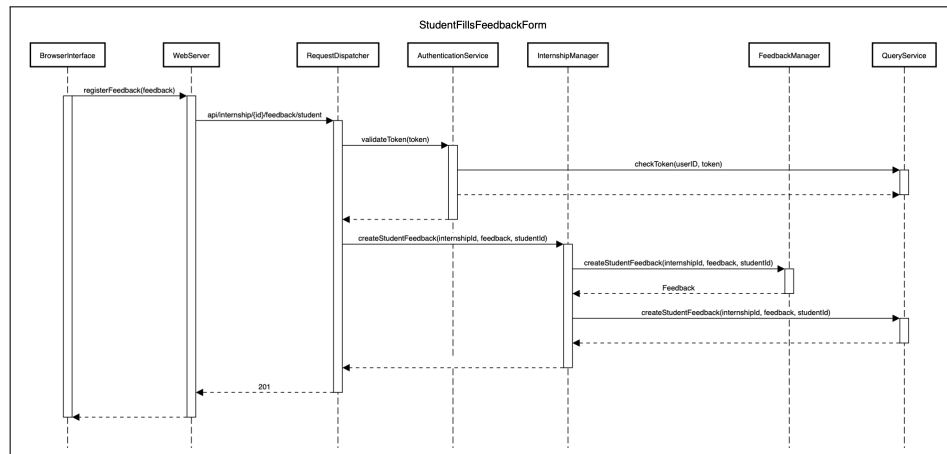
RV15 - UniversityEndsInternship

To end an internship, the university is before hand authenticated by means of the token found in the header of the request. Then, via the internship manager and complaint manager components, the internship of its student can be interrupted, if complaints have arose. If that is the case, the DB is updated accordingly.



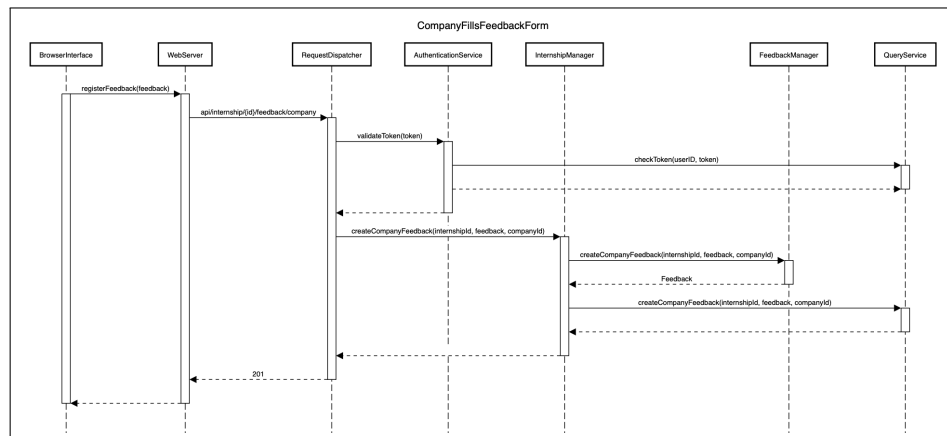
RV16 - StudentFillsFeedbackForm

To provide a feedback about a finished internship, the student is before hand authenticated by means of the token found in the header of the request. Then, via the internship manager and the feedback manager components, the feedback form is accepted, validated and inserted into the DB.



RV17 - CompanyFillsFeedbackForm

To provide a feedback about a finished internship, the company is before hand authenticated by means of the token found in the header of the request. Then, via the internship manager and the feedback manager components, the feedback form is accepted, validated and inserted into the DB.



2.5 Component interfaces

2.5.1 Application APIs

POST /api/authentication/register/company

- Description: register a company
- Request body: `RegistrationFormCompany`
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

POST /api/authentication/register/student

- Description: register a student
- Request body: `RegistrationFormStudent`
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

POST /api/authentication/login

- Description: log in a user from credentials
- Request body: `Credentials`
- Response 200: OK
- Response 400: Bad Request
- Response 401: Unauthorized

POST api/authentication/verification/{id}

- Description: user is verified through the verification code
- Response 200 - Ok

- Response 400 - Bad Request

GET /api/authentication

- Description: check log in token validity
- Response 200: OK
- Response 401: Unauthorized

POST /api/enrollment/applications/{id}

- Description: make an application request
- Request body: ApplicationRegistration
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

GET /api/enrollment/applications/{id}

- Description: get pending applications
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found
- Response 500: Internal Server Error

POST api/enrollment/questionnaires

- Description: a custom questionnaire is created
- Request body - Questionnaire
- Response 200 - Ok
- Response 400 - Bad Request

GET api/enrollment/questionnaires/{id}

- Description: the questionnaire is returned
- Response 200 - Ok Questionnaire
- Response 400 - Bad Request

POST api/enrollment/questionnaires/{id}

- Description: the filled in questionnaire is stored
- Request body - Questionnaire
- Response 200 - Ok
- Response 400 - Bad Request

GET api/enrollment/proposals

- Description: proposals are returned
- Response 200 - Ok Proposal []
- Response 400 - Bad Request

POST api/enrollment/proposals

- Description: a proposal is registered
- Request body - Proposal
- Response 200 - Ok
- Response 400 - Bad Request

POST api/enrollment/proposals/{id}

- Description: the proposal is accepted
- Response 200 - OK
- Response 400 - Bad Request

POST /api/enrollment/accept/{id}

- Description: accept an application
- Request body: Date
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found
- Response 500: Internal Server Error

POST /api/enrollment/reject/{id}

- Description: reject an application
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found
- Response 500: Internal Server Error

GET /api/internship

- Description: get internships for a student
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

POST /api/internship/{id}/feedback/student

- Description: create feedback for a student
- Request body: Feedback
- Response 200: OK
- Response 400: Bad Request

GET /api/internship/{id}/feedback/student

- Description: get the student feedback
- Response 200: OK
- Response 400: Bad Request

POST /api/internship/{id}/feedback/company

- Description: create feedback for a company
- Request body: Feedback
- Response 200: OK
- Response 400: Bad Request

GET /api/internship/{id}/feedback/company

- Description: get the company feedback
- Response 200: OK
- Response 400: Bad Request

GET /api/internship/advertisements/{id}

- Description: get internships from an advertisement
- Response 200: OK
- Response 400: Bad Request

POST /api/internship/delete/{id}

- Description: only for test, don't use
- Response 200: OK
- Response 404: Not Found

GET api/internship/{id}/complaints

- Description: complaints about an internship are returned
- Response 200 - OK Complaint[]
- Response 400 - Bad Request

POST api/internship/{id}/complaints

- Description: a complaint about an internship is created
- Request body - Complaint
- Response 200 - Ok
- Response 400 - Bad Request

POST api/internship/{id}/interrupt

- Description: the internship is interrupted
- Response 200 - Ok
- Response 400 - Bad Request

POST /api/internship/delete/feedback/{id}

- Description: delete your feedback for the internship
- Response 200: Ok
- Response 404: Not Found

GET /api/notification

- Description: get notifications for a student
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found
- Response 500: Internal Server Error

POST /api/notification/delete/{id}

- Description: delete a notification for a student
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

GET /api/profile/company

- Description: get the company profile information
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

POST /api/profile/company

- Description: update the profile information of the company
- Request body: ProfileUpdateCompany
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

GET /api/profile/student

- Description: get the student profile information
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

POST /api/profile/student

- Description: update the profile information of the student
- Request body: ProfileUpdateStudent
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

GET /api/profile/company/{id}

- Description: get a company profile information

- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

GET /api/profile/student/{id}

- Description: get a student profile information
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

GET /api/profile/cv

- Description: download the CV of the student
- Response 200: OK
- Response 404: Not Found

POST /api/profile/cv

- Description: upload the CV of a student
- Request body: CV
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

GET /api/profile/cv/{id}

- Description: download the CV of a student
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

POST /api/profile/cv/delete

- Description: delete the CV of a student
- Response 200: OK
- Response 404: Not Found

POST /api/profile/delete

- Description: delete the user
- Response 200: OK
- Response 404: Not Found

POST /api/profile/skills

- Description: add a skill into the student profile
- Request body: SkillRegistration

- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

GET /api/profile/skills

- Description: get the skills of the student
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found
- Response 500: Internal Server Error

GET /api/profile/skills/{id}

- Description: get the skills of a student
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found
- Response 500: Internal Server Error

POST /api/profile/skills/delete/{id}

- Description: delete a skill from profile
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

GET /api/recommendation/advertisements

- Description: get distinct advertisements for a student or for a company
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

POST /api/recommendation/advertisements

- Description: create a new advertisement
- Request body: AdvertisementRegistration
- Response 200: OK
- Response 400: Bad Request
- Response 500: Internal Server Error

GET /api/recommendation/advertisements/{id}

- Description: get a specific advertisement
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

GET /api/recommendation/candidates/advertisements/{id}

- Description: get recommended students for specific advertisements
- Response 200: OK
- Response 400: Bad Request

POST /api/recommendation/suggestions/advertisement/{id}/student/{id}

- Description: create a new suggestion by a company for one student
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

POST /api/recommendation/delete/{id}

- Description: delete an advertisement
- Response 200: OK
- Response 400: Bad Request
- Response 404: Not Found

2.5.2 Interface methods

IEnrollmentQueries

- Entity.Application GetApplication(int userId, int advertisementId)
- Entity.Application GetApplication(int applicationId)
- bool CreateApplication(int userId, int advertisementId, string questionnaireAnswer)
- List<Entity.Application> GetPendingApplications(int id)
- bool AcceptApplication(int id)
- bool RejectApplication(int id)
- bool CreateInternship(int studentId, int companyId, int advertisementId, DateTime start, DateTime end)
- bool NotifyStudent(int studentId, int advertisementId, bool accepted)
- bool RejectAllApplications(int id)
- bool UpdateAdvertisementSpots(int id)

IRecommendationQueries

- List<Entity.Advertisement> GetAdvertisementsOfCompany(int companyId)
- List<Entity.Advertisement> GetAdvertisementsForStudent(int studentId)
- int? CreateAdvertisement(int companyId, Entity.Advertisement advertisement, List<Entity.Skill> skills)
- void MatchAdvertisementForStudent(int advertisementId)
- Entity.Advertisement GetAdvertisement(int advertisementId)
- List<Entity.Student> GetRecommendedCandidates(int companyId, int advertisementId)

- bool CreateSuggestionsForStudent(int advertisementId, int studentId, int companyId)
- bool DeleteAdvertisement(int advertisementId, int companyId)

IProfileService

- Company GetCompany(int id)
- Student GetStudent(int id)
- bool UpdateProfileCompany(int id, DTO.ProfileUpdateCompany updateForm)
- bool UpdateProfileStudent(int id, DTO.ProfileUpdateStudent updateForm)
- bool IsCompanyUpdateFormValid(DTO.ProfileUpdateCompany updateForm)
- bool IsStudentUpdateFormValid(DTO.ProfileUpdateStudent updateForm)
- bool CheckCvValidity(IFormFile file)
- bool StoreCvFile(int id, IFormFile file)
- IFormFile RetrieveCvFile(int id)
- bool DeleteCv(int id)
- bool DeleteUser(UserType type, int id)
- bool AddSkill(int id, string name)
- List<Skill> GetSkills(int id)
- bool DeleteSkill(int skillId, int studentId)

IRecommendationService

- List<Advertisement> GetAdvertisementsOfCompany(int companyId)
- List<Advertisement> GetAdvertisementsForStudent(int studentId)
- bool CreateAdvertisement(int companyId, DTO.AdvertisementRegistration advertisement)
- Advertisement GetAdvertisement(int advertisementId)
- List<Student> GetRecommendedCandidates(int companyId, int advertisementId)
- bool CreateSuggestionsForStudent(int advertisementId, int studentId, int companyId)
- bool DeleteAdvertisement(int advertisementId, int companyId)

IAuthenticationQueries

- bool RegisterCompany(Entity.Company user)
- bool RegisterStudent(Entity.Student user)
- Entity.Company FindCompanyFromUsername(string username)
- Entity.Student FindStudentFromUsername(string username)
- Entity.Company FindCompanyFromEmail(string email)
- Entity.Student FindStudentFromEmail(string email)

IFileService

- string GetCvFilePath(string fileName)
- bool SaveFile(string filePath, byte[] fileData)
- byte[] RetrieveFile(string filePath)
- bool DeleteFile(string filePath)

IEmailService

- boolean sendEmail(email: Email, content: String)

ISuggestionService

- List<Suggestion> proposeProfileSuggestions(user: User)
- List<Suggestion> proposeCVSuggestions(cv: File)
- List<Suggestion> proposeAdvertisementSuggestions(
 advertisement: Advertisement)

IComplaintService

- Complaint buildComplaint(user: User, internship: Internship,
 complaint: Complaint)
- List<Complaint> adjustComplaints(complaints: List<Complaint>)

IEnrollmentService

- Application GetApplication(int userId, int advertisementId)
- Application GetApplication(int applicationId)
- bool CreateApplication(int userId, int advertisementId,
 string questionnaireAnswer)
- Advertisement GetAdvertisement(int id)
- List<Application> GetPendingApplications(int id)
- bool CheckStartDateValidity(DateTime date)
- bool AcceptApplication(int id)
- bool RejectApplication(int id)
- bool CreateInternship(int studentId, int companyId,
 int advertisementId, DateTime start)
- bool NotifyStudent(int studentId, int advertisementId, bool accepted)
- Internship GetInternship(int id)
- bool RejectAllApplications(int id)
- bool UpdateAdvertisementSpots(int id)

IInternshipQueries

- Entity.Internship GetInternshipForStudent(int studentId)
- List<Entity.Internship> GetInternshipFromAdvertisement(
 int advertisementId, int companyId)
- bool CreateStudentFeedback(int internshipId,
 Entity.StudentFeedback feedback, int studentId)
- bool CreateCompanyFeedback(int internshipId,
 Entity.CompanyFeedback feedback, int companyId)
- Entity.StudentFeedback GetStudentFeedback(int internshipId,
 int userId, string role)
- Entity.CompanyFeedback GetCompanyFeedback(int internshipId,
 int userId, string role)

- bool DeleteInternship(int internshipId, int userId, string role)
- bool DeleteStudentFeedback(int internshipId, int userId)
- bool DeleteCompanyFeedback(int internshipId, int userId)

IDataService

- MySqlConnection GetConnection()
- List<Entity.Student> MapToStudents(IDataReader reader)
- List<Entity.Company> MapToCompanies(IDataReader reader)
- List<Entity.Skill> MapToSkills(IDataReader reader)
- List<Entity.Advertisement> MapToAdvertisements(IDataReader reader)
- List<Entity.StudentNotifications> MapToStudentNotifications(IDataReader reader)
- List<Entity.Application> MapToApplications(IDataReader reader)
- List<Entity.Internship> MapToInternships(IDataReader reader)
- List<Entity.StudentFeedback> MapToStudentFeedback(IDataReader reader)
- List<Entity.CompanyFeedback> MapToCompanyFeedback(IDataReader reader)
- List<string> MapToStrings(IDataReader reader, string fieldName)
- List<int> MapToInts(IDataReader reader, string fieldName)

IIternshipService

- Internship GetInternshipForStudent(int studentId)
- List<Internship> GetInternshipFromAdvertisement(int advertisementId, int companyId)
- bool CreateStudentFeedback(int internshipId, DTO.Feedback feedback, int studentId)
- bool CreateCompanyFeedback(int internshipId, DTO.Feedback feedback, int companyId)
- DTO.Feedback GetStudentFeedback(int internshipId, int userId, string role)
- DTO.Feedback GetCompanyFeedback(int internshipId, int userId, string role)
- bool DeleteInternship(int internshipId, int userId, string role)
- bool DeleteFeedback(int internshipId, int userId, string role)

INotificationService

- List<StudentNotifications> GetStudentNotifications(int studentId)
- bool DeleteNotification(int notificationId, int studentId)

IAuthenticationService

- bool IsCompanyRegistrationValid(DTO.RegistrationFormCompany registrationForm)
- bool IsStudentRegistrationValid(DTO.RegistrationFormStudent registrationForm)
- bool RegisterCompany(DTO.RegistrationFormCompany registrationForm)
- bool RegisterStudent(DTO.RegistrationFormStudent registrationForm)
- User ValidateCredentials(DTO.Credentials credentials)
- string GenerateToken(User user)

- string HashPassword(string salt, string password)
- string GenerateSalt()
- bool IsValidEmail(string email)

INotificationQueries

- List<Entity.StudentNotifications> GetStudentNotifications(int studentId)
- bool DeleteNotification(int notificationId, int studentId)

IProfileQueries

- Entity.Company FindCompanyFromId(int id)
- Entity.Student FindStudentFromId(int id)
- bool UpdateSaltAndPassword(UserType type, int id, string salt, string hash)
- bool UpdateUsername(UserType type, int id, string username)
- bool UpdateEmail(UserType type, int id, string email)
- bool UpdateBio(UserType type, int id, string bio)
- bool UpdateHeadquarter(int id, string headquarter)
- bool UpdateFiscalCode(int id, string fiscalCode)
- bool UpdateVatNumber(int id, string vatNumber)
- bool UpdateName(int id, string name)
- bool UpdateSurname(int id, string surname)
- bool UpdateUniversity(int id, string university)
- bool UpdateCourseOfStudy(int id, string courseOfStudy)
- bool UpdateGender(int id, char gender)
- bool UpdateBirthDate(int id, DateTime birthDate)
- bool DeleteUser(UserType type, int id)
- int FindSkill(string name)
- bool AddSkill(string name)
- bool AddSkillToStudent(int studentId, int skillId)
- List<Entity.Skill> GetSkills(int id)
- bool DeleteSkill(int skillId, int studentId)

2.6 Architectural styles and patterns

Three-tier architecture - Since the S&C platform is offered through the web, the client-server architecture is the most suitable for this kind of application. In particular, a three-tier architecture comes handy when dividing the system into the three main logical layers: presentation (web server), business logic (application server), and data storage (database). Each tier has its specific functions, ensuring separation of concerns and efficient management of the system. It allows scalability, simplifies maintenance, and enhances control and security.

REST - The communication between the web server and the application server is based on the REST architectural style. REST is stateless, meaning that the server does not need to store any information about the session, allowing higher scalability. Moreover, it provides a uniform and easy interface via the HTTP methods (GET, POST, PUT, DELETE).

MVC - The MVC pattern is used to separate the presentation layer from the business logic. In particular, the model contains the definitions of all the elements of the application, the view is the browser interface component, and the controller comprehends all the other application server services and managers. This pattern allows a more maintainable application, since changes in one layer do not affect other layers.

2.7 Other design decisions

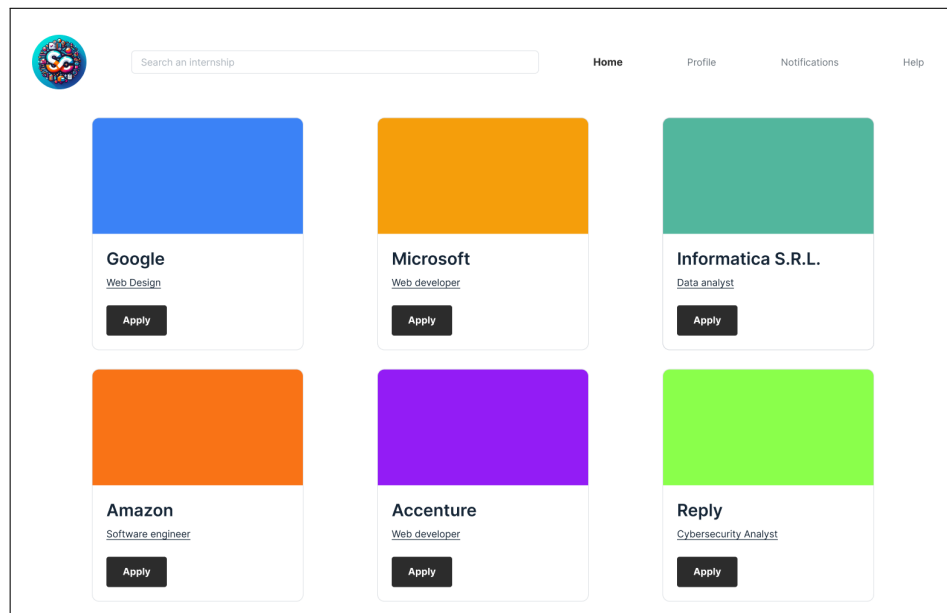
Scale out - Using a scale out design, when implementing the software, can highly enhance availability, by avoiding the need of downtimes. This design approach enables the system to expand its capacity to cope with an increasing demand, by adding more resources when needed. Moreover, it can be more cost effective than upgrading individual components. The scale out design also improves reliability, by providing redundant resources that can take over when other components fails. Flexibility is also augmented, as resources can be fastly added or removed, as required. Lastly, the scale out design improves the system performance, since workloads are processed in parallel rather than sequentially.

Relational database - Due to its effectiveness at storing structured data, a relational database was chosen for the system design. It also enforces data integrity while providing fast query performance. A relational DB is also able to handle large amounts of data, while supporting many concurrent users.

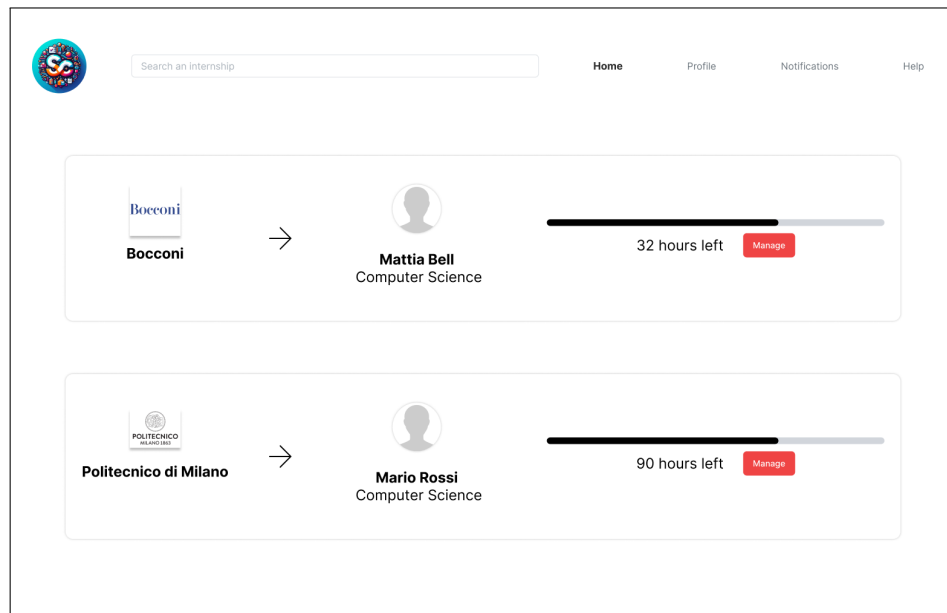
Token-based authentication - The authentication and authorization is implemented using a token-based process. These tokens are sent to the users each time they log in, and they must be included within each request that requires authentication or authorization.

3 User interface design

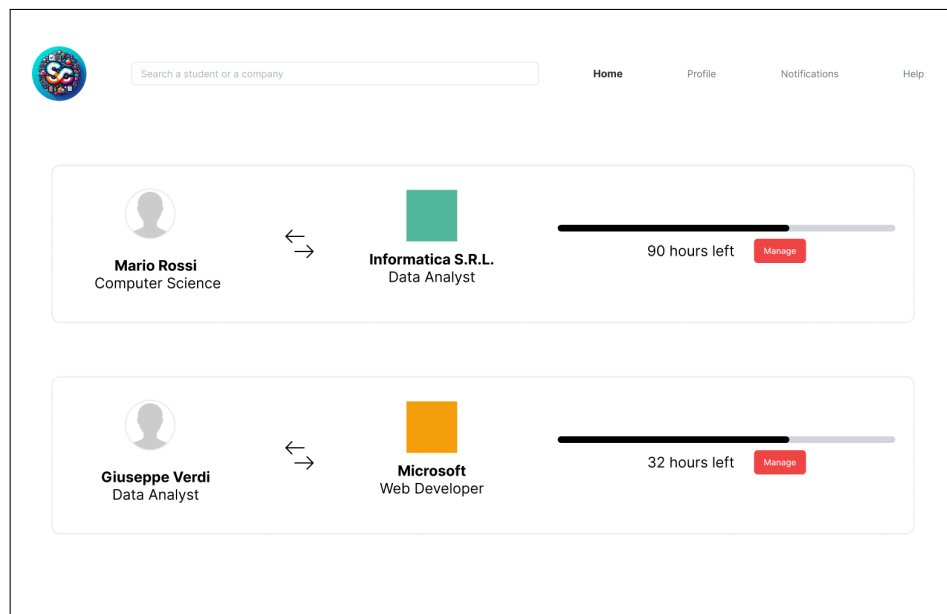
3.1 Home pages



Student home page showing the advertisements feed

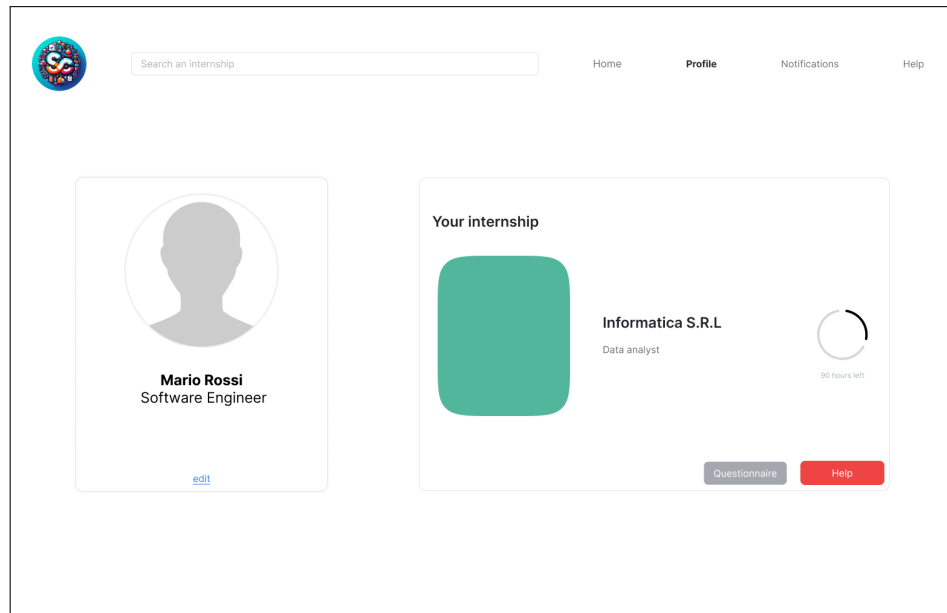


Company home page showing its ongoing internships




University home page showing its students ongoing internships


3.2 Profile pages



Student profile page



HomeProfileNotificationsHelp



Informatica S.R.L.

[edit](#)

Create a new internship

Title


Skills you need

Description


Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

[Continue](#)

Company profile page



HomeProfileNotificationsHelp

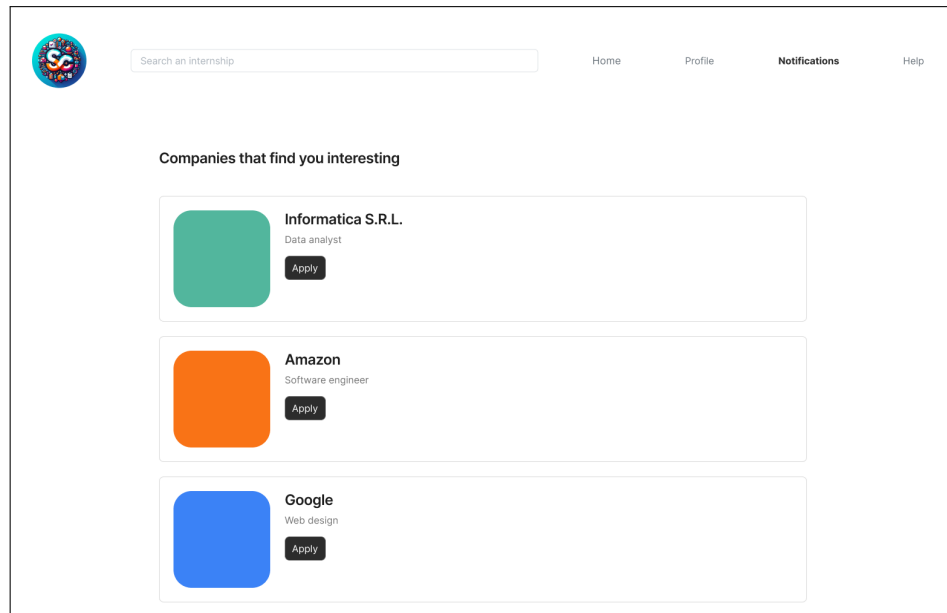


Università Luigi Bocconi

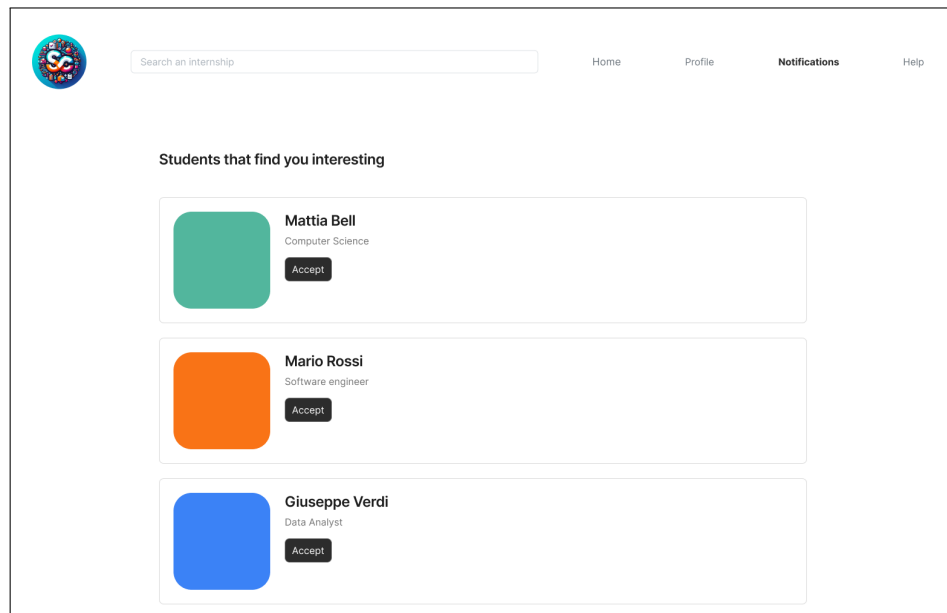
[edit](#)

University profile page

3.3 Notifications pages

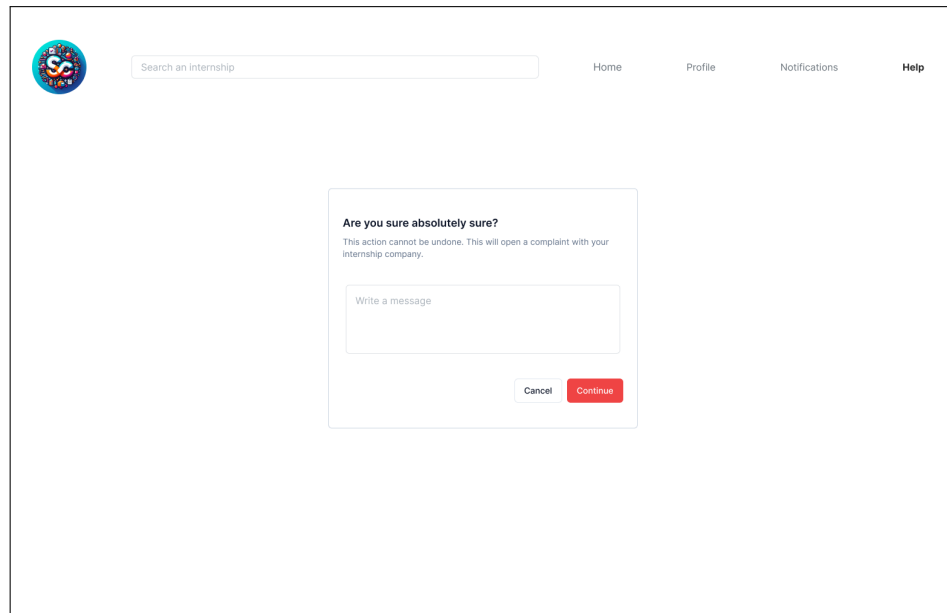


Student notifications page



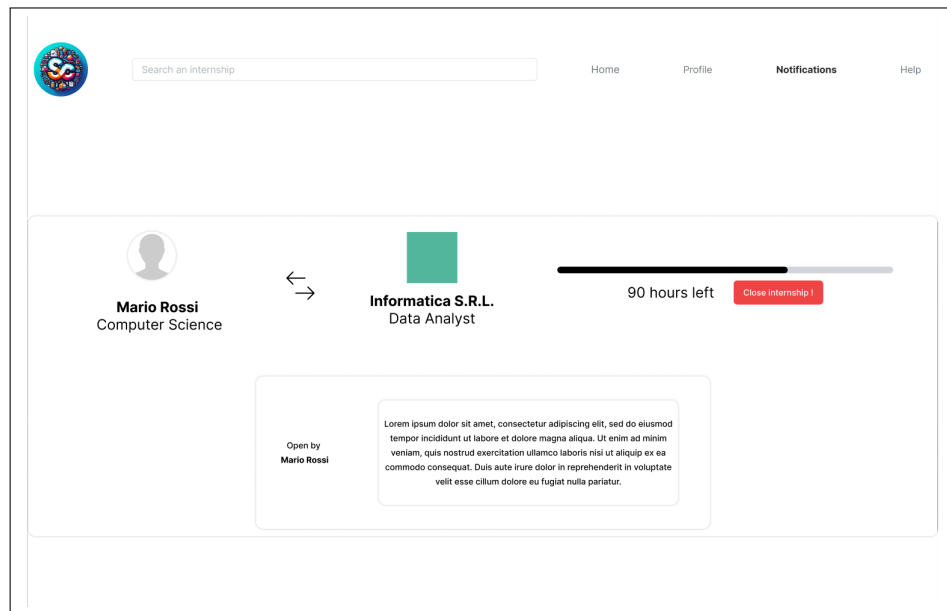
Company notifications page

3.4 Help pages



The screenshot displays a web application interface. At the top left is a circular logo with the letters 'Sc' and a colorful background. To its right is a search bar with the placeholder text 'Search an internship'. Further right are navigation links: 'Home', 'Profile', 'Notifications', and 'Help'. The 'Help' link is highlighted. In the center of the page, a modal dialog box is open. The dialog has a title 'Are you sure absolutely sure?' and a warning message: 'This action cannot be undone. This will open a complaint with your internship company.' Below the message is a text input field with the placeholder 'Write a message'. At the bottom right of the dialog are two buttons: 'Cancel' and 'Continue'.

Help page for student and company



Help page for university

4 Requirements traceability

This chapter provides detailed traceability description tables that maps each system requirement to the corresponding components used to fulfill it. These descriptions ensure that all requirements are addressed by the system architecture, by helping in tracking the implementation and verification of each requirement. Each table below lists a requirement and the components involved in its realization.

R1	The system must allow an unregistered student to sign up
C1	Browser interface
C2	Web server
C3	DMBS
C4	Email server
C5	Request dispatcher
C6	Authentication service
C7	Email service
C11	Suggestion service
C12	Profile manager

R2	The system must allow an unregistered company to sign up
C1	Browser interface
C2	Web server
C3	DMBS
C4	Email server
C5	Request dispatcher
C6	Authentication service
C7	Email service
C11	Suggestion service
C12	Profile manager

R3	The system must allow an unregistered university to sign up
C1	Browser interface
C2	Web server
C3	DMBS
C4	Email server
C5	Request dispatcher
C6	Authentication service
C7	Email service
C11	Suggestion service
C12	Profile manager

R4	The system must allow a registered user to log in
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C6	Authentication service
C9	Query service
C10	Recommendation service
C11	Suggestion service

R5	The system must allow a registered user to fill in and edit its personal information
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C11	Suggestion service
C12	Profile manager

R6	The system must allow a registered student to upload its CV
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C11	Suggestion service
C12	Profile manager

R7	The system must allow a registered company to post an internship project
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C12	Profile manager

R8	The system must allow a registered student to visualize a list of open internship projects
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C10	Recommendation service
C11	Suggestion service

R9	The system must allow a registered company to visualize a list of eligible students
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C10	Recommendation service

R10	The system must allow a registered student to make an enrollment request to an internship project
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C13	Enrollment manager

R11	The system must allow a registered company to build custom made questionnaires
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C14	Internship manager
C16	Feedback manager

R12	The system must allow a registered company to send questionnaires to students
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C14	Internship manager
C16	Feedback manager

R13	The system must allow a registered student to fill in the questionnaire
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C14	Internship manager
C16	Feedback manager

R14	The system must allow a registered company to accept students enrollment requests
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C8	Notification service
C9	Query service
C13	Enrollment manager
C14	Internship manager

R15	The system must allow a registered student to see their ongoing internship information
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C12	Profile manager
C14	Internship manager

R16	The system must allow a registered company to see their ongoing internships information
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C12	Profile manager
C14	Internship manager

R17	The system must allow a registered university to see their students ongoing internship information
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C12	Profile manager
C14	Internship manager

R18	The system must allow a registered student to send complaints to the university
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C14	Internship manager
C15	Complaint manager

R19	The system must allow a registered company to send complaints to the university
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C14	Internship manager
C15	Complaint manager

R20	The system must allow a registered university to visualize complaints it received
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C8	Notification service
C9	Query service
C14	Internship manager
C15	Complaint manager

R21	The system must allow a registered university to end an ongoing internship of its student
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C12	Profile manager
C14	Internship manager
C16	Feedback manager

R22	The system must allow a registered student to fill in a feedback form when the internship ends
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C8	Notification service
C9	Query service
C14	Internship manager
C16	Feedback manager

R23	The system must allow a registered company to fill in a feedback form when the internship ends
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C14	Internship manager
C16	Feedback manager

R24	The system must allow a registered student to visualize a list of suggested internships
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C10	Recommendation service

R25	The system must allow a registered company to visualize a list of suggested students
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C9	Query service
C10	Recommendation service

R26	The system must allow a registered student to be notified about recommended internship
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C8	Notification service
C9	Query service
C10	Recommendation service

R27	The system must allow a registered company to be notified about recommended students
C1	Browser interface
C2	Web server
C3	DMBS
C5	Request dispatcher
C8	Notification service
C9	Query service
C10	Recommendation service

5 Implementation, integration and test plan

5.1 Development process and approach

The system will be implemented, integrated, and tested following a bottom-up approach, starting from the model layer and progressively adding and testing the other components of the server architecture. This approach allows for the early validation of low-level functionalities, ensuring a solid foundation for higher-level components.

Both server-side and client-side components will be developed and tested simultaneously, but the focus will be on server-side components initially, due to their backbone role of the system. Incremental integration testing will be applied, aiming to identify and resolve bugs as soon as they emerge during the development process, minimizing the impact on subsequent stages.

To facilitate testing at different levels of the architecture, drivers will be employed to simulate higher-level components that are not yet implemented, while stubs will be used to mock the behavior of lower-level dependencies, such as external services or the database. This strategy ensures that individual components can be tested in isolation, while also validating their integration as part of the broader system.

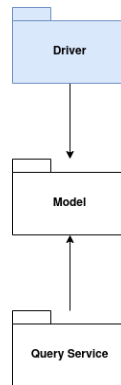
This testing approach ensures that dependencies between components are carefully managed, leading to a robust and well-tested system at every level.

5.2 Implementation and integration plan

This section describes the implementation and integration plan of each part of the system.

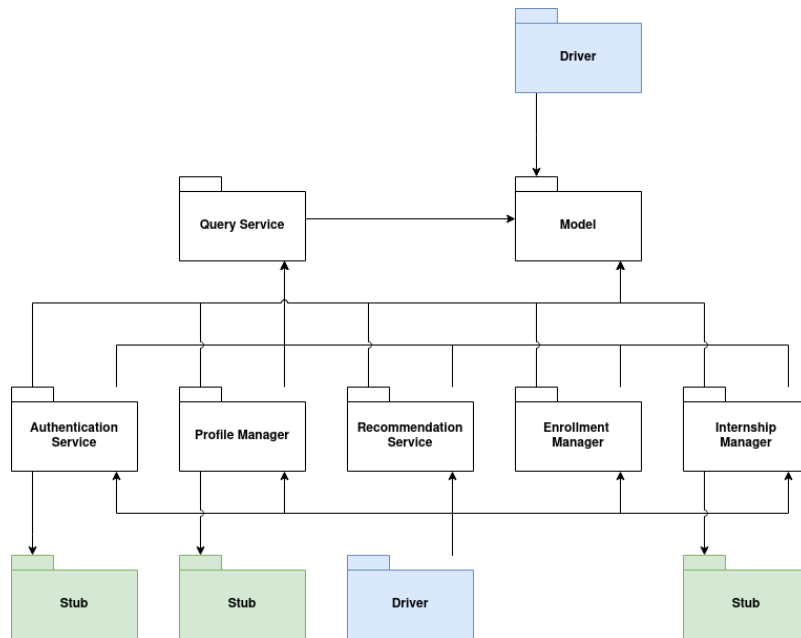
5.2.1 Application server

Firstly, the model and the query service will be implemented and unit tested with a driver, which will substitute components not yet implemented.

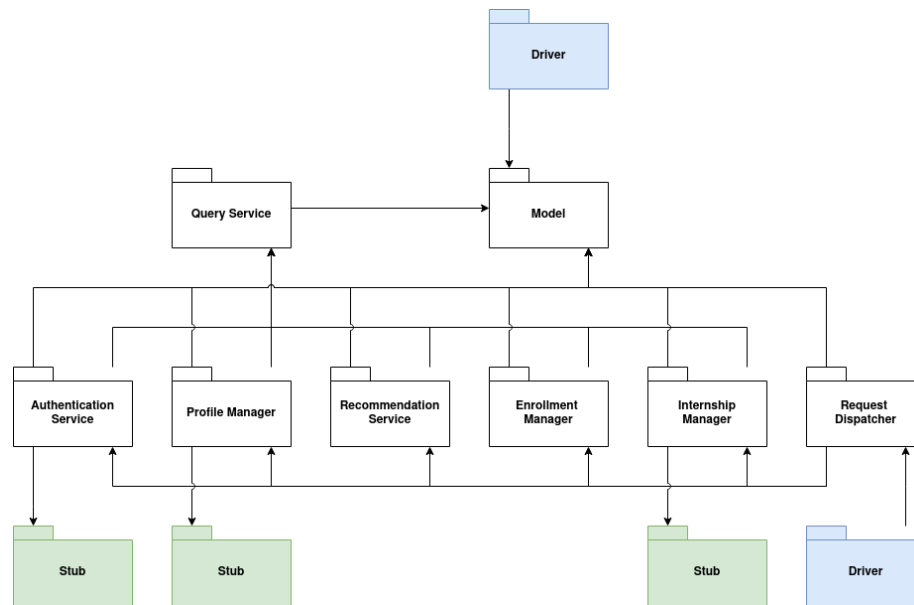


As the second step, the authentication service, profile manager, recommendation service, enrollment manager and internship manager will be implemented and tested with a driver, which will substitute the request dispatcher.

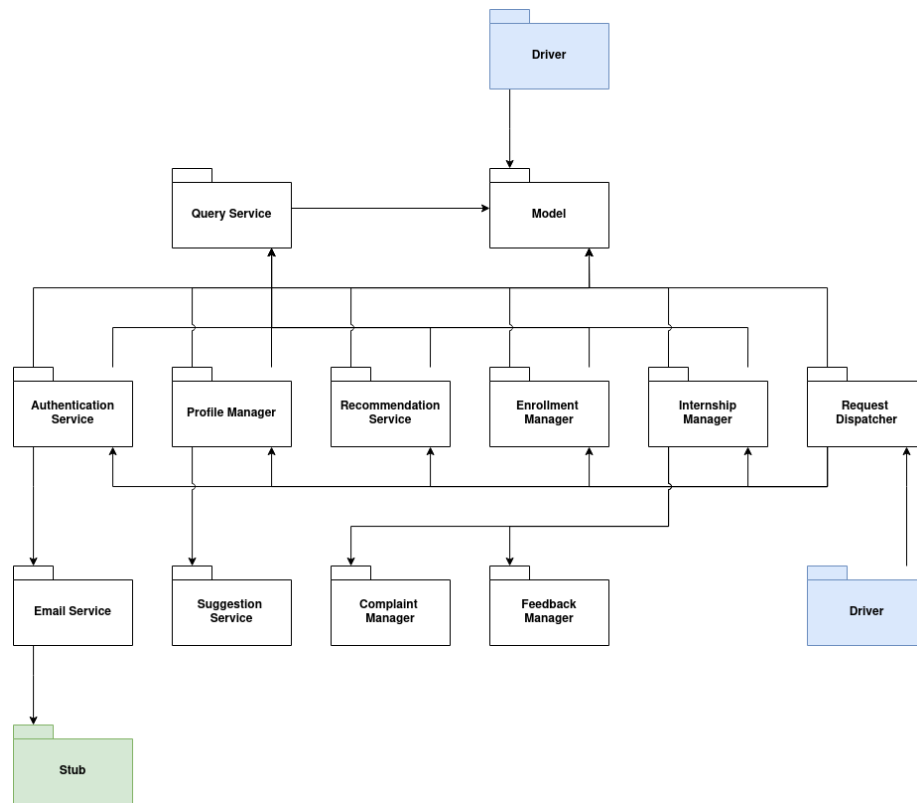
There will also be 3 stubs, which will substitute, respectively, the email service for the authentication service, the suggestion service for the profile manager, and the complaint manager and feedback manager for the internship manager.



Then, the request dispatcher will be implemented and tested with a driver substituting the web server.

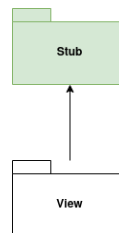


the last components of the server that will be implemented will be the email service, the suggestion service, the complaint manager and the feedback manager. A stub will be used to simulate the behavior of the email server.



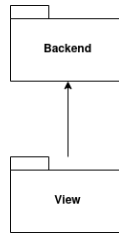
5.2.2 Web server

Each component of the view is rigorously unit tested using a stub for the REST APIs, enabling parallel development of the frontend and backend.



5.2.3 Final test

Once the implementations of the backend and of the frontend are finished, final tests can take place.



5.3 Development technologies

The selection of technologies for our project was the result of a careful analysis, with the goal of ensuring a system that is performant, scalable, and easily maintainable. In this section, we will describe in detail the choices made for the web and application servers and database, highlighting the reasons behind each decision.

5.3.1 Web server

The frontend, hosted on the web server, is built using JavaScript and the React framework, along with Vite and Tailwind. This approach allows to create responsive and easily maintainable user interfaces. React, combined with Vite and Tailwind, offers the possibility of developing complex and dynamic user interfaces, with excellent component management and a high-level user experience.

5.3.2 Application server

The backend of the system, deployed on the application server, is developed in C#, with the support of the .NET runtime. This allows to build a solid, scalable, and reliable system. The choice of C# and .NET was guided by their robustness, high performance, and the vast range of tools and libraries available for the development of web applications and services.

For the development of the RESTful APIs, we use the ASP.NET web framework. This technology allows to create high-performance, efficient web services that comply with industry standards. ASP.NET provides a flexible development environment, with a high level of performance and the ability to create well-documented APIs, which are essential for the interaction between servers.

5.3.3 Database

Data persistence is entrusted to MySQL. This choice provides a robust database management system that is well-suited to our needs. MySQL is a reliable and widely supported DBMS, ideal for managing relational data and its capacity to handle complex queries and large volumes of data.

The responses to database queries are provided by our MySQL system, hosted on a dedicated AWS machine. Initially, the DBMS holds a single DB used for tests, but the architecture theoretically includes a dedicated database for production when system is ready. The MySQL database provides a stable environment for data storage, while the logical separation between test and production ensures data integrity during development.

5.3.4 Architectures and patterns

Our system is developed following the distributed MVC pattern, with the model, view and controller components distributed between the web server, application server, and database. We also adopt the clean architecture pattern, dividing the code into application, domain, infrastructure, and presentation modules. The first three are found in the application server, whereas the last is on the web server. The adoption of these patterns allows to keep the code organized, modular, and easily maintainable, promoting a clear separation of responsibilities and greater flexibility in development.

5.4 Technologies used for testing

To ensure the quality and reliability of our system, we have implemented a complete testing strategy using various tools and technologies. In this section, we describe the choices made for unit and integration tests, as well as tools for API testing and response simulation.

5.4.1 Backend unit testing

For unit testing of the backend, we use xUnit. This framework allows to test the components of the application server in an isolated and precise manner. xUnit provides a flexible and complete testing environment, ideal for testing the various functionalities and logic of our system.

5.4.2 Frontend unit testing

For testing the JavaScript code of the frontend, we rely on Jest. This framework is essential to verify the correct implementation of the user interface. Jest provides an effective and complete testing environment for JavaScript code, allowing us to test each individual component of the frontend in isolation.

5.4.3 REST APIs requests fabrication

For creating REST requests, we use cURL and Swagger. These tools allow to simulate and test requests efficiently. cURL is a versatile tool for testing from the command line, while Swagger offers a graphical interface for documenting and testing APIs.

5.4.4 REST APIs responses simulation

To simulate REST responses during the frontend development, we use MSW (Mock Service Worker). This tool allows to work independently from the backend. MSW allows to create mock APIs without having to depend on a functioning backend, speeding up the frontend development and testing process.

5.4.5 Web pages requests fabrication

For creating web page requests, we use a web browser. This allows to verify the interaction with the user interface. The direct use of the browser ensures a real user experience and allows to test the functionalities of the web interface.

6 Effort spent

Unit	Member	Hours
Setup	Ostidich	2
Introduction	Ostidich	1
Architectural design	Ostidich, Salari	10
Component diagram	Ostidich	2
ER diagram	Salari	3
Deployment diagram	Ostidich	1
Sequence diagrams	Ostidich	5
User interface design	Rivitti	10
Requirements traceability	Rivitti	2
Implementation, integration and test plan	Salari	5

7 References

1. The UI mockups have been created using figma.com
2. The document has been written using latex-project.org
3. The sequence diagrams have been created using sequencediagram.org
4. The other diagrams have been created using draw.io
5. The project repository has been uploaded on github.com