SOFTWARE ENGINEERING II

DESIGN DOCUMENT

Students&Companies

POLITECNICO DI MILANO

COMPUTER SCIENCE AND ENGINEERING

Authors
Travaini Alessandro

Academic year
2024-2025
Version 1.0

Professor
Camilli Matteo

Deliverable: DD

Title: Design Document

Authors:

Alessandro Travaini 10742196

Date: 21/12/2024

Download page: GitHub

Table of contents

1 Introduction 1
1.1 Scope 1
1.1.1 Product domain 1
1.1.2 Main architectural choices 1
1.2 Definitions, acronyms, abbreviations 2
1.2.1 Definitions 2
1.2.2 Acronyms 2
1.2.3 Abbreviations 2
1.3 Reference documents 3
1.4 Overview 3
2 Architectural design 4
2.1 Overview: high level components and interactions 4
2.2 Component view 5
2.3 Deployment view 7
2.4 Component interfaces 8
2.4.1 Account Manager 8
2.4.2 Internship Manager 9
2.4.3 Application Manager 9
2.4.4 Questionnaire Manager 10
2.4.5 Interview Manager 11
2.4.6 Comment Manager 12
2.4.7 Mail Manager 12
2.5 Runtime view 13
2.6 Selected architectural styles and patterns 22
3 User interface design 23
4 Requirements traceability 24
5 Implementation, integration and test plan 27
6 Effort spent 29
7 References 30

1 Introduction

1.1 Scope

1.1.1 Product domain

The scope of the project will cover the users that interact with the system, the user-generated actions that influence the system and the system-generated actions that have an effect on the outside world.

For the project the following users that interact with the system have been found:

- Students.
- Companies.

Students will be able to upload relevant information (experiences, skills and attitudes of students, as listed in their CVs) and will browse available internship offerings. After an internship is found, the student can apply to it and wait for the company to accept his request, meanwhile the student can browse for other internships. Once the company accepts the request, the student is notified and a contact is established, after which a selection process starts. During this process the student can be asked to fill questionnaires or have interviews with the company. If the student passes the selection phase he can start the internship; after this point any other pending internship request from the student is automatically cancelled. During the internship, the student can use the platform to provide information about the internship, like complaints, problems or other relevant information.

Companies will be able to open internship positions specifying the projects, number of open positions and other relevant information. Companies can get in touch with students that satisfy their needs. Companies will be able to accept incoming requests from students that apply for available internships after which the selection process starts. During this process the company can create questionnaires and host interviews with the student. If the student is hired, the internship position is removed from the available listings. During the internship, the company can use the platform to provide information about the internship, like complaints, problems or other relevant information.

1.1.2 Main architectural choices

The system is to be implemented using a microservices oriented architecture. This allows for independent scaling of individual components based on their requirement and eases the process of implementation and testing, moreover microservices satisfy the needs of many nonfunctional requirements such as availability, resilience and maintainability.

In those cases, the chosen architecture has a lead compared to others thanks to the fact that if a component fails or needs to be updated the whole system is not necessarily affected, instead only a subset of features is rendered unavailable.

Lastly, this architecture allows for faster parallel development since different teams can work on different services at the same time considering the other parts of the system as black boxes.

1.2 Definitions, acronyms, abbreviations

This section contains the definitions for people that may not know what a specific concept is, acronyms and abbreviations used throughout the document.

1.2.1 Definitions

- **Contact**: phase of the internship process in which an internship offer has been accepted by a student and that same student has been accepted by the company. After this phase, both parties can get in touch with each other.
- **Selection**: phase of the internship process where students are interviewed with questionnaires and virtual meetings.

1.2.2 Acronyms

- **S&C**: Students & Companies
- **CV**: Curriculum Vitae
- **UI**: User Interface
- **MS**: Microsoft
- **API**: Application Programming Interface
- **DBMS**: DataBase Management System

1.2.3 Abbreviations

- **R***: Requirement

1.3 Reference documents

This document is based on the following materials:

- The specification of the RASD and DD assignment of the Software Engineering II course a.y. 2024/25.
- Course slides shared on WeBeep.
- Past Design Documents.

1.4 Overview

1. **Introduction**: this section introduces the project. It contains a high level description of the system, including its architectural style and architectural choices.
2. **Architectural design**: this section is very broad and contains the description of the various interfaces of the system, its deployment and an in-depth description of the use cases, opening the black box that is the system.
3. **User interface design**: this section focuses on the user interface design, expanding, if necessary, what was shown in the RASD.
4. **Requirements traceability**: this section contains a mapping between the requirements and design elements.
5. **Implementation, integration and test plan**: this section describes the order in which the various components and subsystems must be developed and tested.
6. **Effort spent**: this section shows the time spent on each section of the document, for each member of the group.
7. **References**: this section contains all the various references used to write this document.

2 Architectural design

2.1 Overview: high level components and interactions

The system will serve responses to clients connected through the web application.

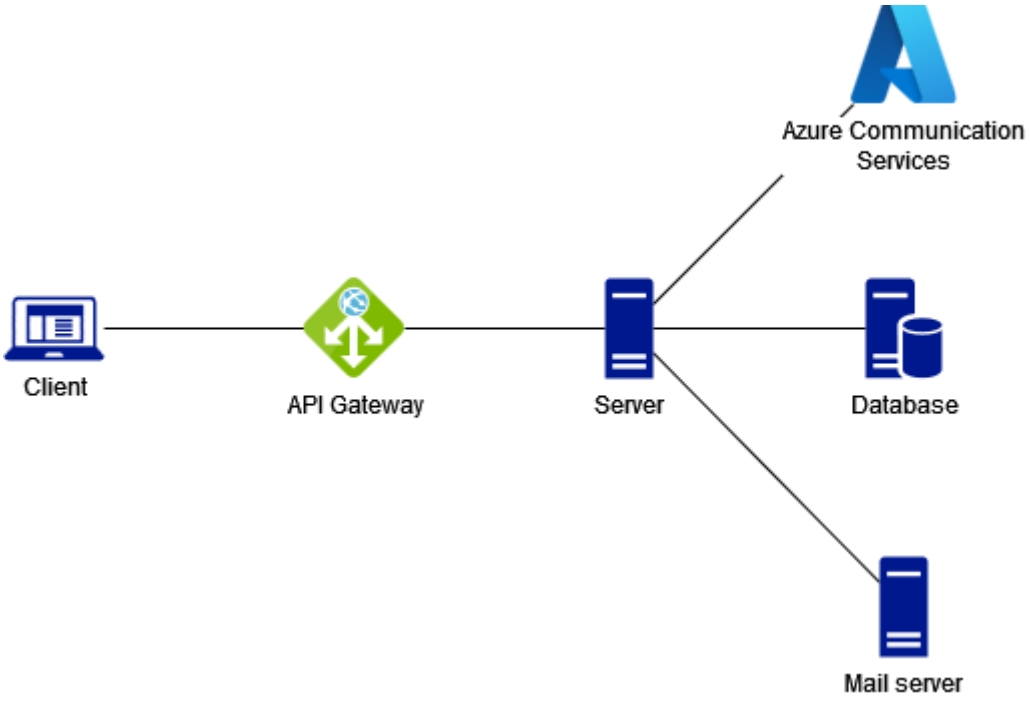


Figure 2-1: high level architecture diagram

All client requests are processed via an API gateway which redirects the request to the correct microservice, the server will then interact with the microservice in different ways, depending on the request.

The possible actions of the server can be the following:

- The server may query the database after a request.
- The server may exploit an API call to MS servers in the following scenarios:
    - MS Forms: creation and submission of a questionnaire.
    - MS Teams: creation and participation in a virtual meeting.
- The server may use a mail server for notifications to the users.

2.2 Component view

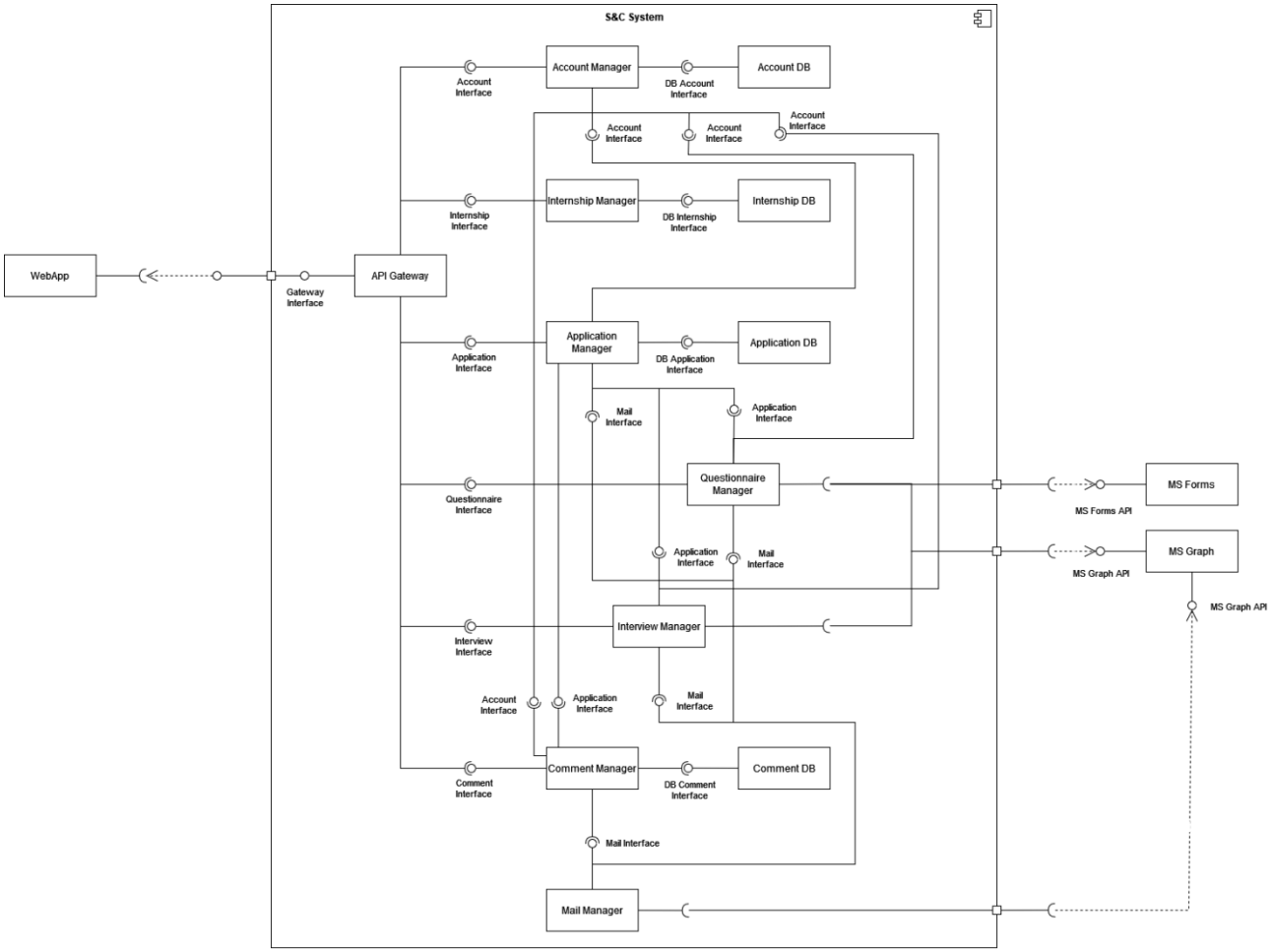


Figure 2-2: Component diagram

The components in the diagram are explained in detail as follows:

- **Webapp**: it represents the website used by each type of user of the system. By using the **Gateway Interface** it sends requests to the **API Gateway** that will then forward the request to the correct service.
- **API Gateway**: this component is a dispatcher of requests for the microservices. All requests coming from the end-user are handled first by this component which will then redirect the request to the correct service. It needs to interact with all the services interfaces.
- **Account manager**: it handles the information about the account of each user. It also handles the login of an user. It receives requests from the **API Gateway** and uses the **DB Account Interface** to interact with its database where all the information of the users is stored.
- **Internship manager**: it handles the creation of an internship. It receives requests from the **API Gateway** and uses the **DB Internship Interface** to interact with its database where all the information of the internship is stored.
- **Application manager**: it handles the life cycle of the internship application, from creation to acceptance/rejection and lastly the various questionnaires/interviews available during the selection phase. It receives requests from the **API Gateway** and uses the **DB Application Interface** to interact with its database where all the information of the application is stored.
- **Questionnaire manager**: it handles the creation and submission of questionnaires. It receives requests from the **API Gateway** and uses **Azure API** to create/submit forms via MS Forms.

- **Interview manager**: it handles the creation and participation to of virtual meetings. It receives requests from the **API Gateway** and uses **Azure API** to create/participate to videoconferences via MS Teams.
- **Comment manager**: it handles the creation of comments during an internship. It receives requests from the **API Gateway** and uses the **DB Comments Interface** to interact with its database where all the information of the comment is stored.
- **Mail manager**: it handles the notifications that are sent via mail. It receives requests from all the services that may send notifications and uses the **Mail Server API** to send mail to the mail server.
- **Account DB**: it's the database for **Account Manager** and it stores information about the users.
- **Internship DB**: it's the database for **Internship Manager** and it stores information about the internships.
- **Application DB**: it's the database for **Application Manager** and it stores information about the applications to internships.
- **Comment DB**: it's the database for **Comment Manager** and it stores information about the comments to internships.

2.3 Deployment view

This section describes the deployment for Students&Companies. This view describes the execution environment of the system, together with the physical distribution of the hardware components that execute the software.

Figure 2-3: Deployment diagram

Since the architecture chosen is based on microservices, every service has an independent database if needed. All the information between services is exchanged with API calls.

Since both Application Manager and Internship Manager may have a great number of requests for the same things (such as a popular internship, that may be viewed hundreds of times), a cache has been provided in both these microservice to store the frequently accessed elements, in order to increase the performance and efficiency.

2.4 Component interfaces

In this section there's a summary of all methods that each component provides to the rest of the system, including names, return types and required arguments.

2.4.1 Account Manager

URI	/api/account/signup
HTTP request method	POST
Request parameter(s)	• void signUp(String userType, String name, String surname, String email, String password) • void signUp(String userType, String companyName, String companyIdentification, String email, String password)
Response	• 201: User has been created • 400: Empty fields in the form • 409: Email already in use
Purpose	Register an user to the platform.

URI	/api/account/login
HTTP request method	POST
Request parameter(s)	• void login(String email, String password)
Response	• 200: Correct credentials • 401: Incorrect credentials
Purpose	Login an user to the platform.

URI	/api/account/settings
HTTP request method	POST
Request parameter(s)	• void updateSettings(Long userId, String name, String surname, String email, String password) • void updateSettings(Long userId, String companyName, String companyIdentification, String email, String password)
Response	• 200: User has been updated • 302: Updated email or password • 400: Empty fields in the form • 409: Email already in use
Purpose	Update user's information.

URI	/api/account/publicinfo
HTTP request method	POST
Request parameter(s)	• void updatePublicInfo(Long userId, String experiences, List<Skill> skills, String attitudes)
Response	• 200: User info has been updated
Purpose	Update user's work information.

URI	/api/account/cv
HTTP request method	POST
Request parameter(s)	• void uploadCV(Long userId, File cv)
Response	• 200: User info has been updated • 413: File is too large • 415: File format is not PDF
Purpose	Update user's CV.

URI	/api/account/getuser
HTTP request method	POST
Request parameter(s)	• user getUser(Long userId)
Response	• 200: User has been retrieved
Purpose	Get an user by its ID.

2.4.2 Internship Manager

URI	/api/internship/create
HTTP request method	POST
Request parameter(s)	• void createInternship(Long userId, String scope, int openPositions, bool isPaid, String otherBenefits)
Response	• 201: Internship has been created • 400: Empty fields
Purpose	Create a new internship.

URI	/api/internship/search
HTTP request method	POST
Request parameter(s)	• List<Internship> searchInternships(String keyword, List<String> filters)
Response	• 200: Return list of internships that match the criteria.
Purpose	Search an internship.

2.4.3 Application Manager

URI	/api/application/create
HTTP request method	POST
Request parameter(s)	• void createNewApplication(Long userId, Long intershipId, String applicationStatus)
Response	• 201: Create a new application
Purpose	Search an internship.

URI	/api/application/getapplications
HTTP request method	POST
Request parameter(s)	• List<Application> getApplications(Long userId)
Response	• 200: Retrieve all applications of user.
Purpose	Get all applications of user.

URI	/api/application/getapplication
HTTP request method	POST
Request parameter(s)	• Application getApplication(Long applicationId)
Response	• 200: Application has been retrieved.
Purpose	Get an application by its ID.

URI	/api/application/updatestatus
HTTP request method	POST
Request parameter(s)	• void updateApplicationStatus(Long userId, Long intershipId, String applicationStatus)
Response	• 200: Application status is updated
Purpose	Update the status (Accepted, Declined, Ongoing) of an internship.

URI	/api/application/savequestionnaire
HTTP request method	POST
Request parameter(s)	• void saveNewQuestionnaire(Long applicationId, Long questionnaireId)
Response	• 201: The questionnaire ID is saved.
Purpose	Save the ID of a questionnaire in the application.

URI	/api/application/getquestionnaire
HTTP request method	POST
Request parameter(s)	• long getQuestionnaire(Long applicationId)
Response	• 200: The questionnaire ID is retrieved
Purpose	Get a questionnaire ID.

URI	/api/application/saveinterview
HTTP request method	POST
Request parameter(s)	• void saveNewInterview(Long applicationId, Long eventId, Long interviewId)
Response	• 201: The questionnaire ID is saved.
Purpose	Save the ID of a questionnaire in the application.

URI	/api/application/getinterview
HTTP request method	POST
Request parameter(s)	• long getInterview(Long applicationId)
Response	• 200: The interview ID is retrieved
Purpose	Get a interview ID.

2.4.4 Questionnaire Manager

URI	/api/questionnaire/create
HTTP request method	POST
Request parameter(s)	• void newQuestionnaire(Long userId, Long applicationId)
Response	• 201: Questionnaire is created • 403: User has no permission to create a questionnaire • 503: MS Forms is not available
Purpose	Create a new questionnaire.

URI	/api/questionnaire/read
HTTP request method	POST
Request parameter(s)	• Questionnaire getQuestionnaire(Long questionnaireId)
Response	• 200: Questionnaire is retrieved • 503: MS Forms is not available
Purpose	Read a questionnaire.

URI	/api/questionnaire/submit
HTTP request method	POST
Request parameter(s)	• void submitQuestionnaire(Long questionnaireId)
Response	• 200: Questionnaire is submitted • 400: Empty fields • 503: MS Forms is not available
Purpose	Submit a questionnaire.

2.4.5 Interview Manager

URI	/api/interview/create
HTTP request method	GET
Request parameter(s)	• void newInterview(Long userId, Long applicationId)
Response	• 201: Interview is created • 400: Date is in the past • 403: User has no permissions to create an interview • 503: MS Teams is not available
Purpose	Create a new interview.

URI	/api/interview/read
HTTP request method	POST
Request parameter(s)	• List<Interview> getInterviews(Long userId)
Response	• 200: Interview is retrieved • 503: MS Teams is not available
Purpose	Read an interview.

URI	/api/interview/join
HTTP request method	POST
Request parameter(s)	• void joinInterview(Long interviewId)
Response	• 200: Interview is joined • 503: MS Teams is not available
Purpose	Join an interview.

2.4.6 Comment Manager

URI	/api/comment/create
HTTP request method	POST
Request parameter(s)	• void createComment(Long userId, Long applicationId, String commentNature, String commentBody)
Response	• 201: New comment is created • 400: Emtpy fields • 403: Internship is not in ongoing status
Purpose	Create a new comment in an internship.

URI	/api/comment/read
HTTP request method	POST
Request parameter(s)	• List<Comment> readComments(Long userId, Long applicationId)
Response	• 200: Comments are retrieved • 403: User has no permissions to read comments
Purpose	Read all the comments of an internship.

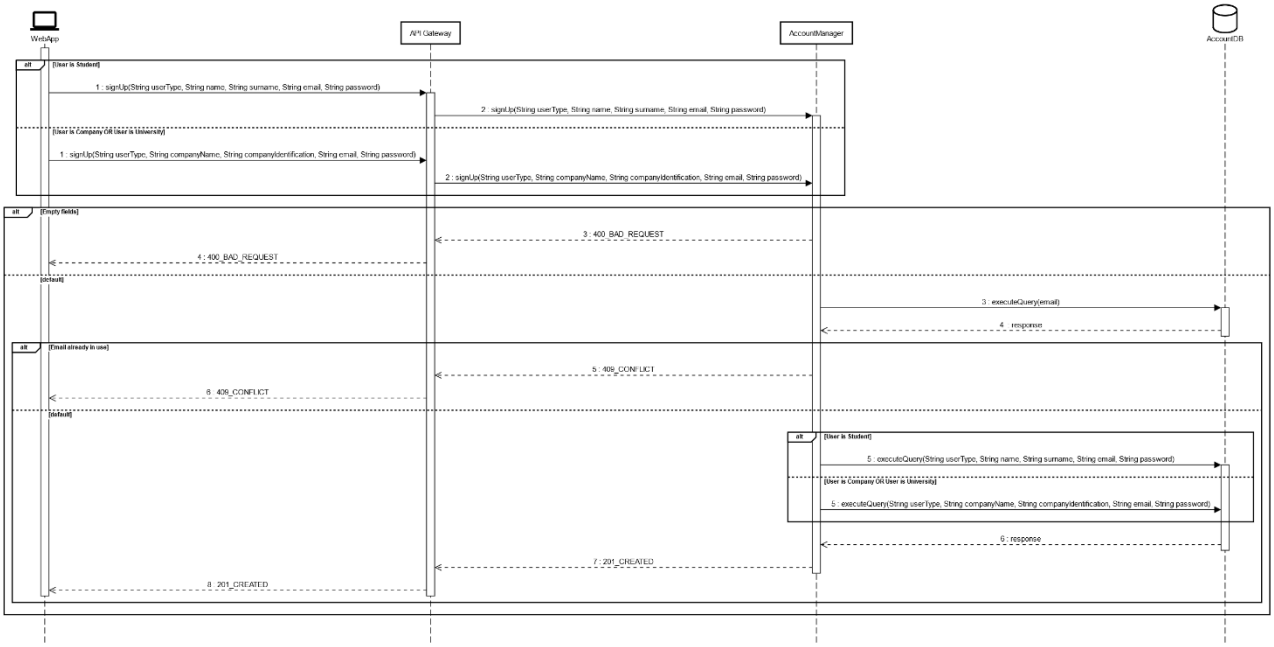
2.4.7 Mail Manager

URI	/api/mail/create
HTTP request method	POST
Request parameter(s)	• void createMail(String recipientEmail, String body)
Response	• 202: Mail has been delivered.
Purpose	Send an email to the user.

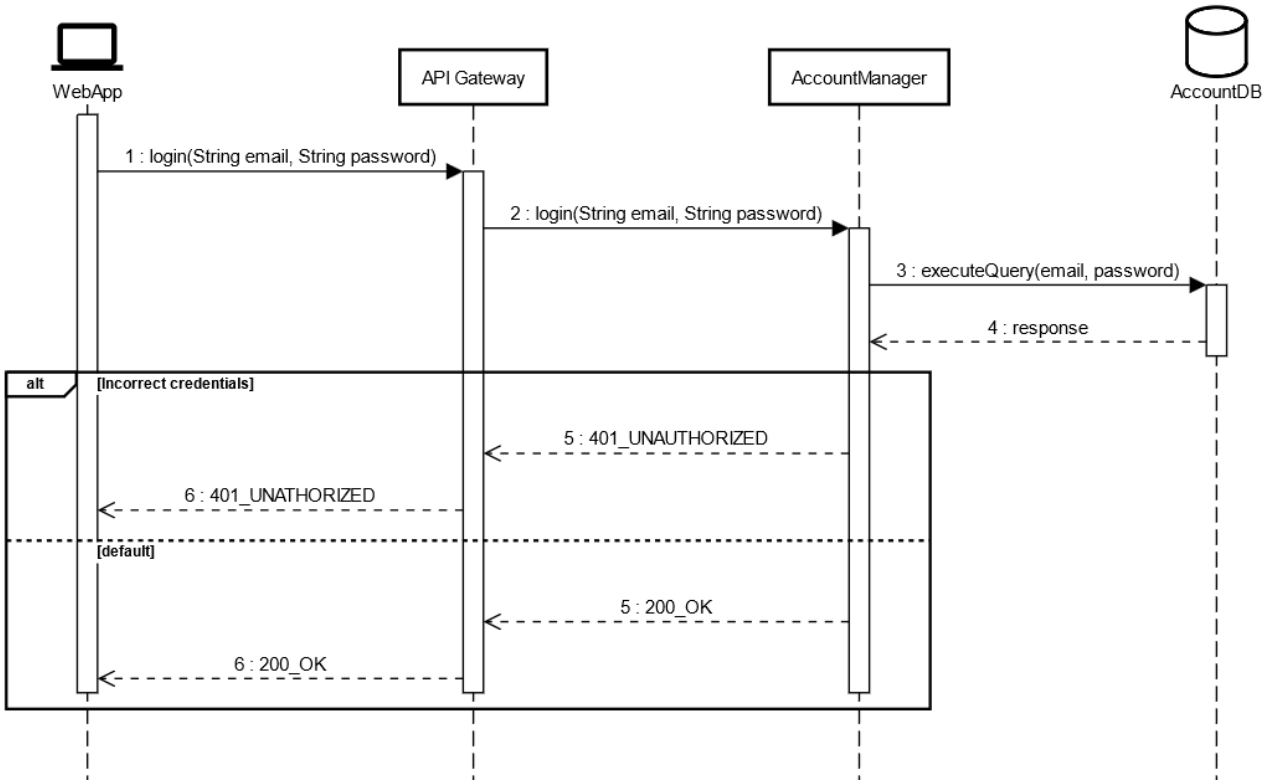
2.5 Runtime view

In this section there are all the sequence diagrams for all use cases found in the RASD.

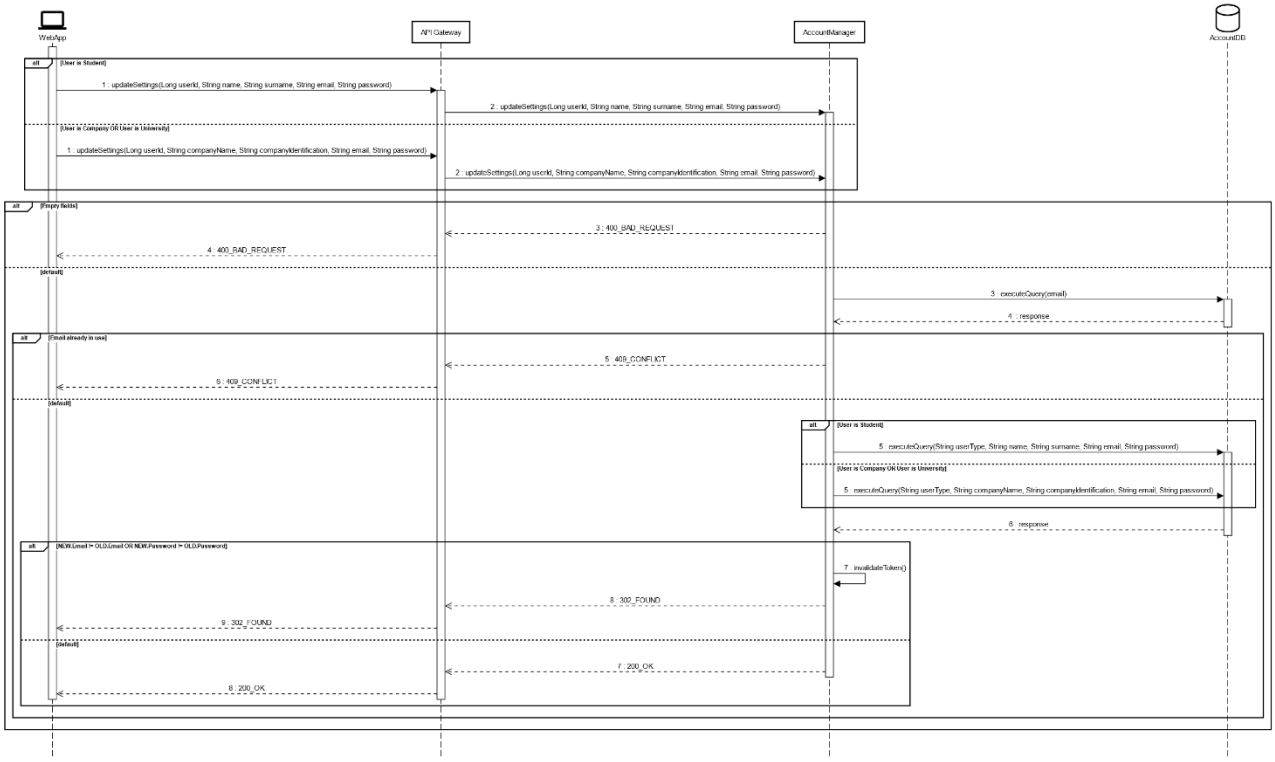
[User Registration]



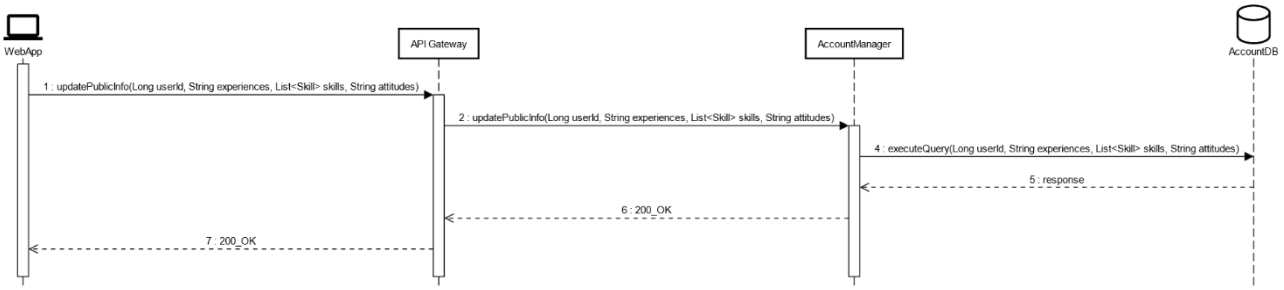
[User Login]



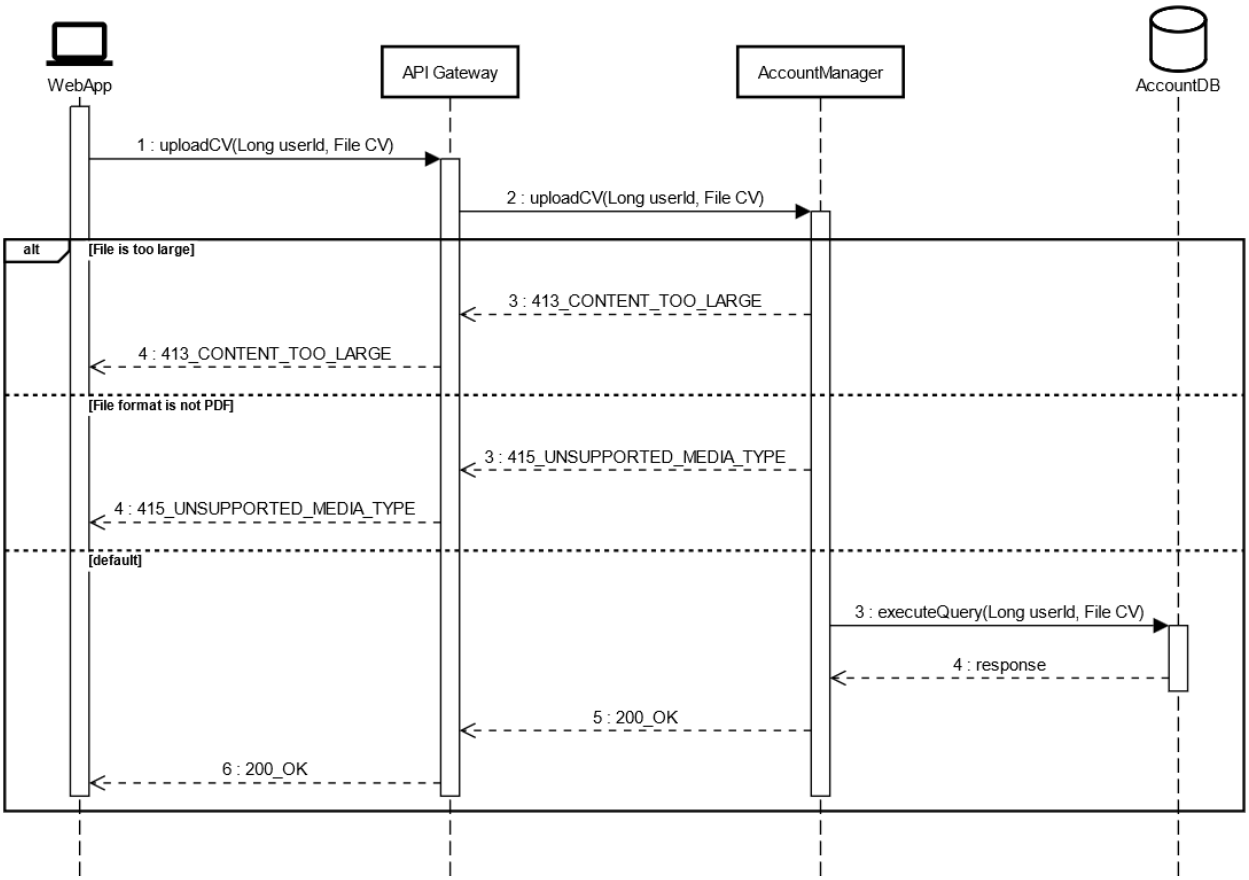
[Manage User Settings]



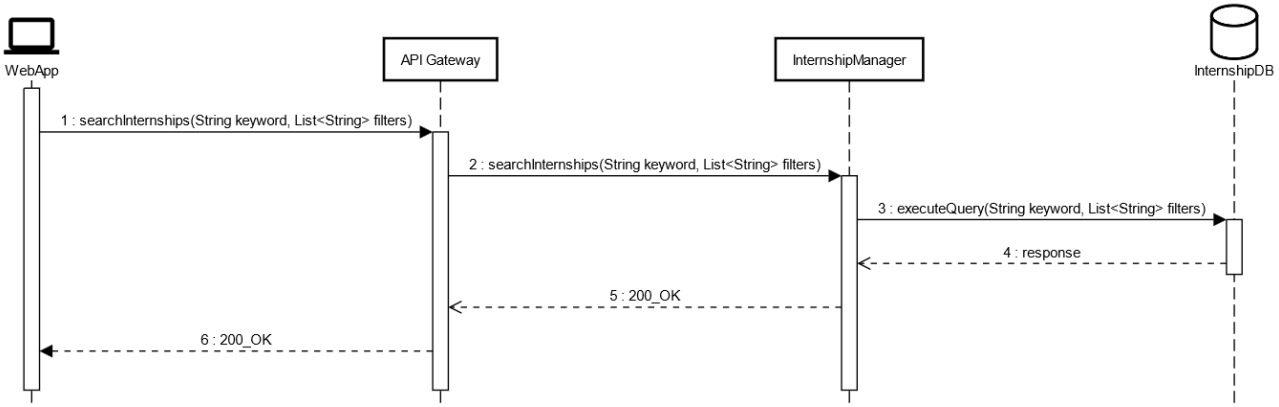
[Share work-related information]



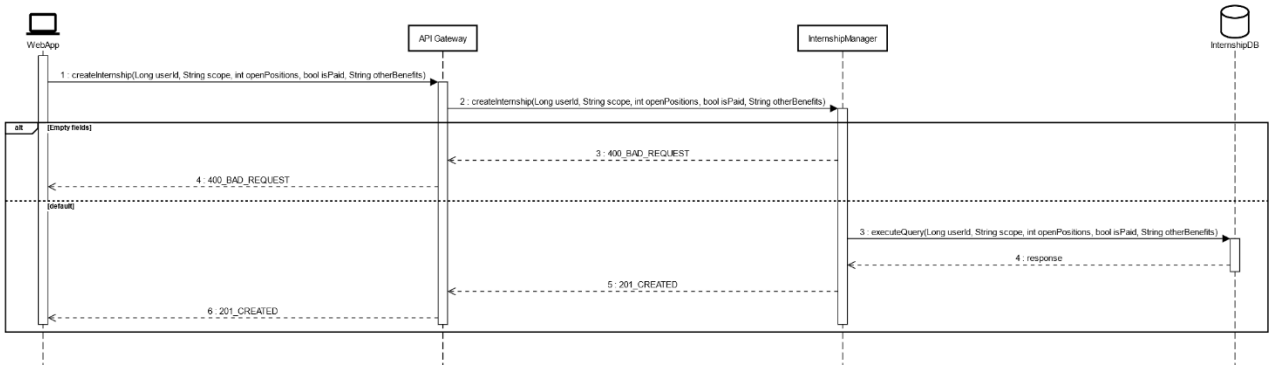
[Upload CV]



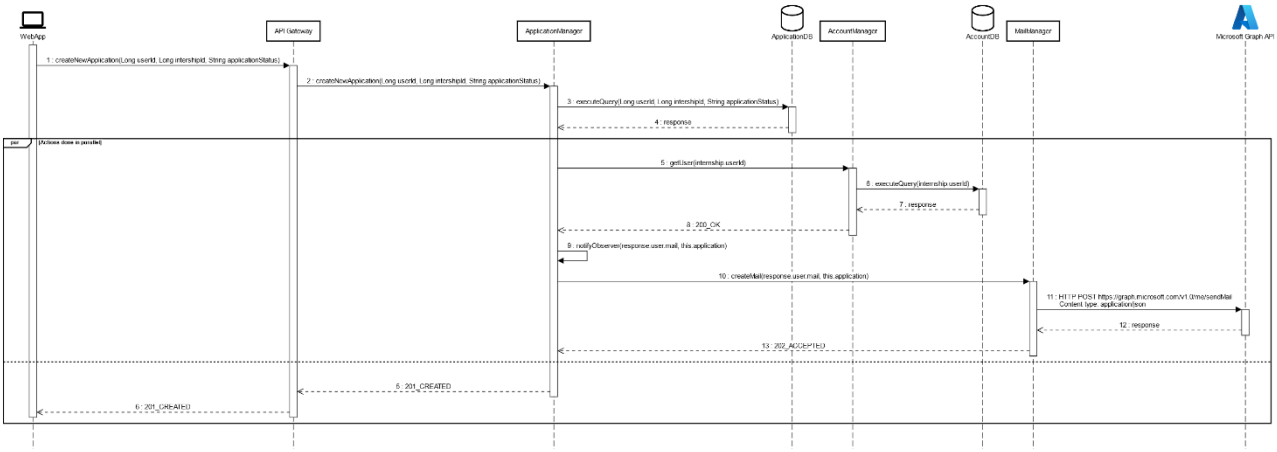
[Search for internship]



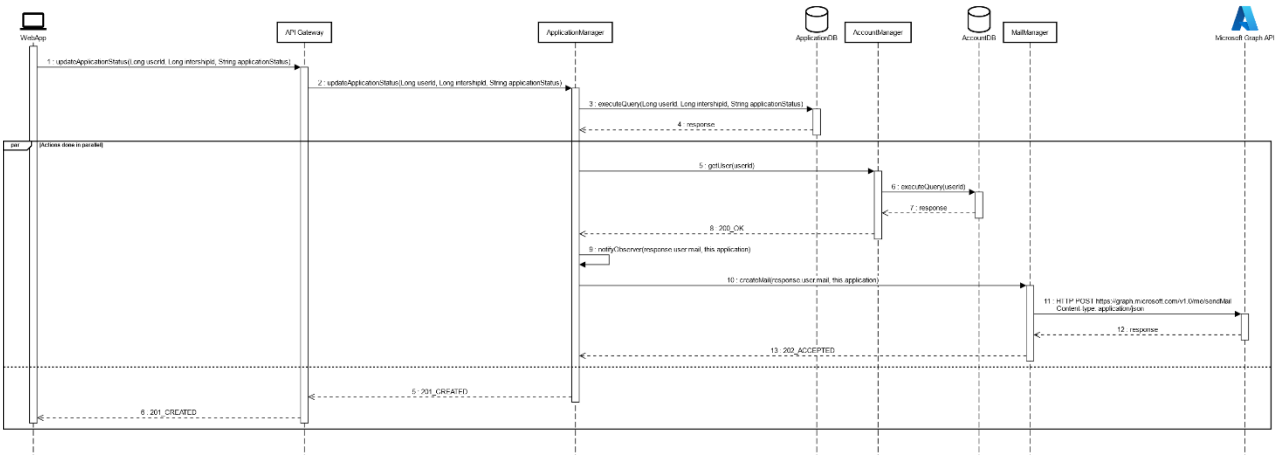
[Create internship]



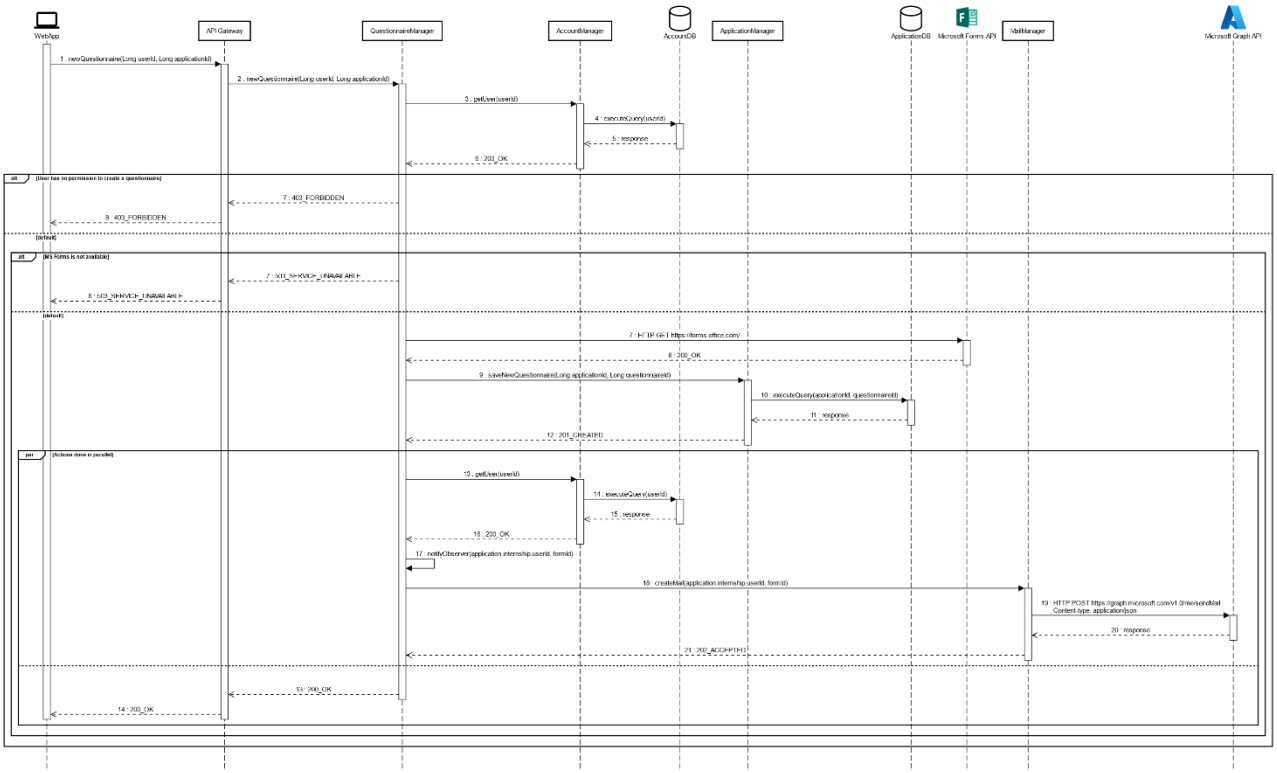
[Apply to internship]



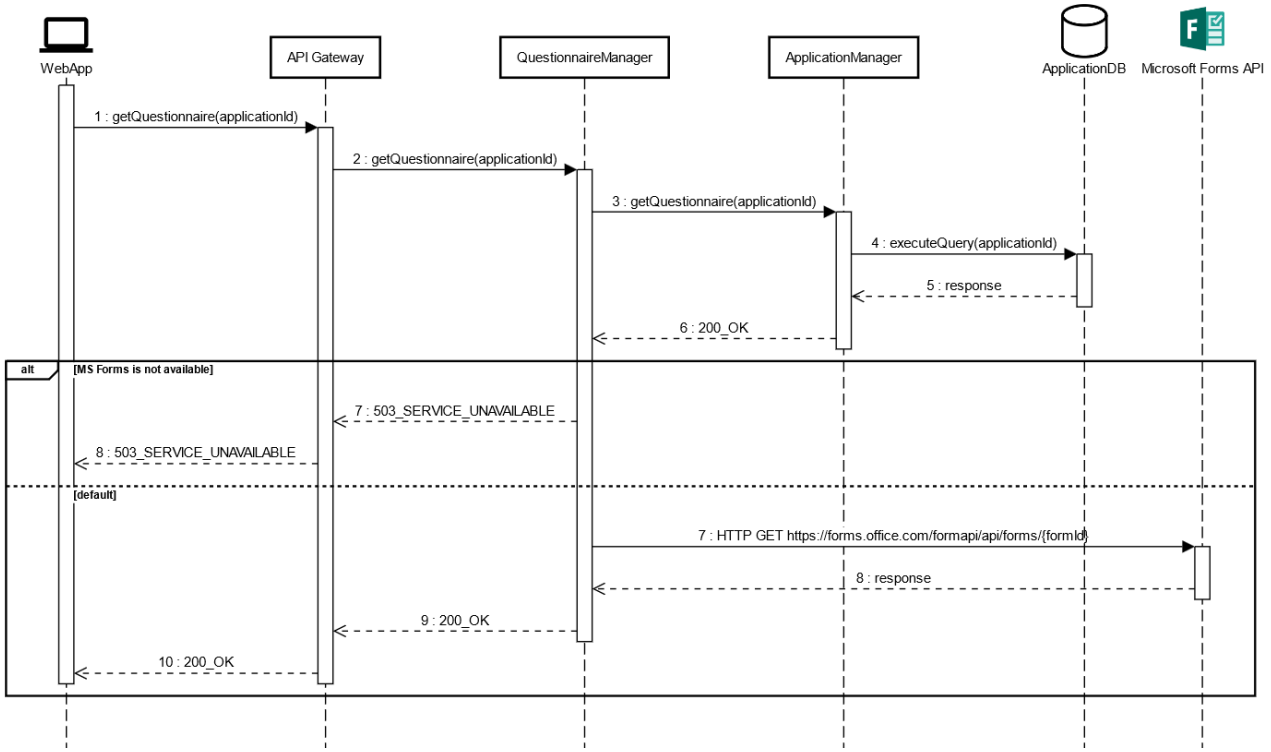
[Accept and Decline request (company to student)]



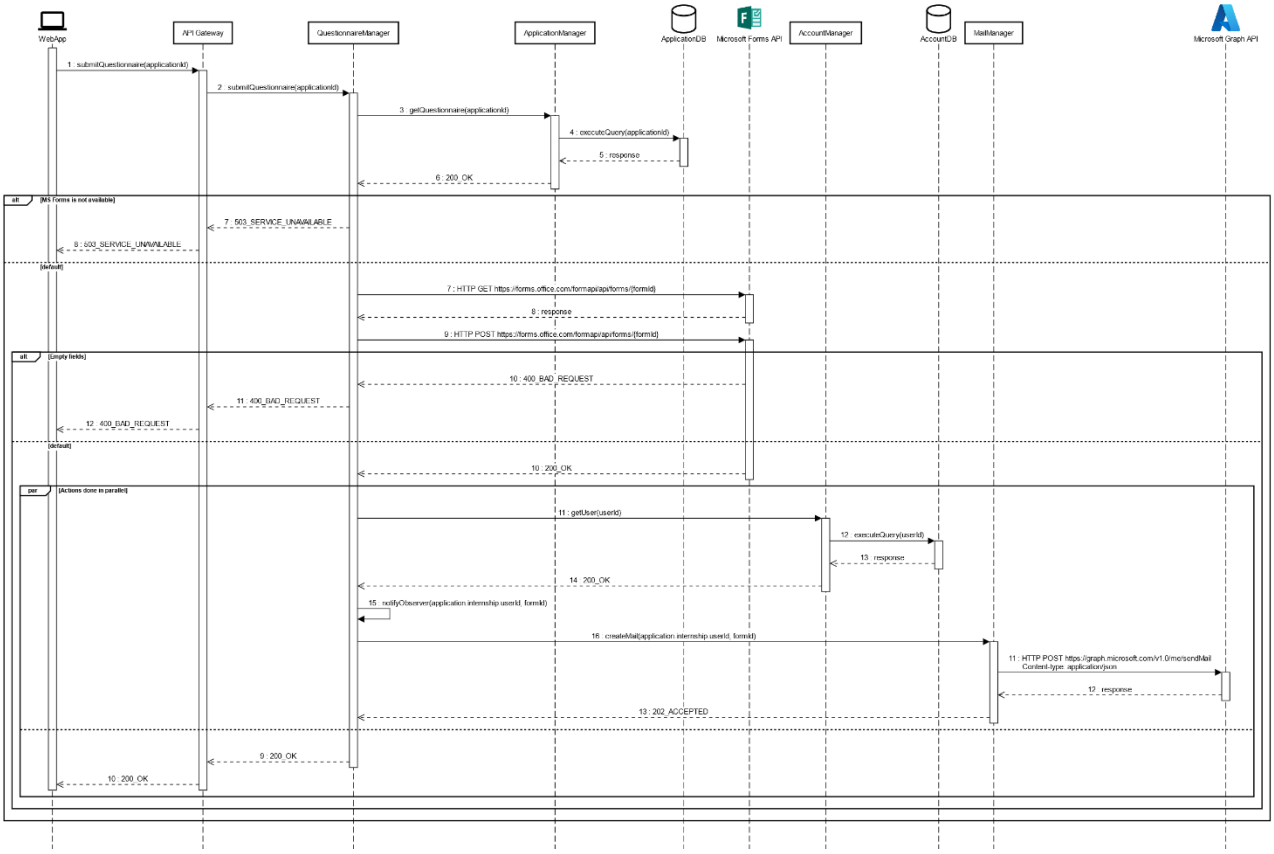
[Create questionnaires]



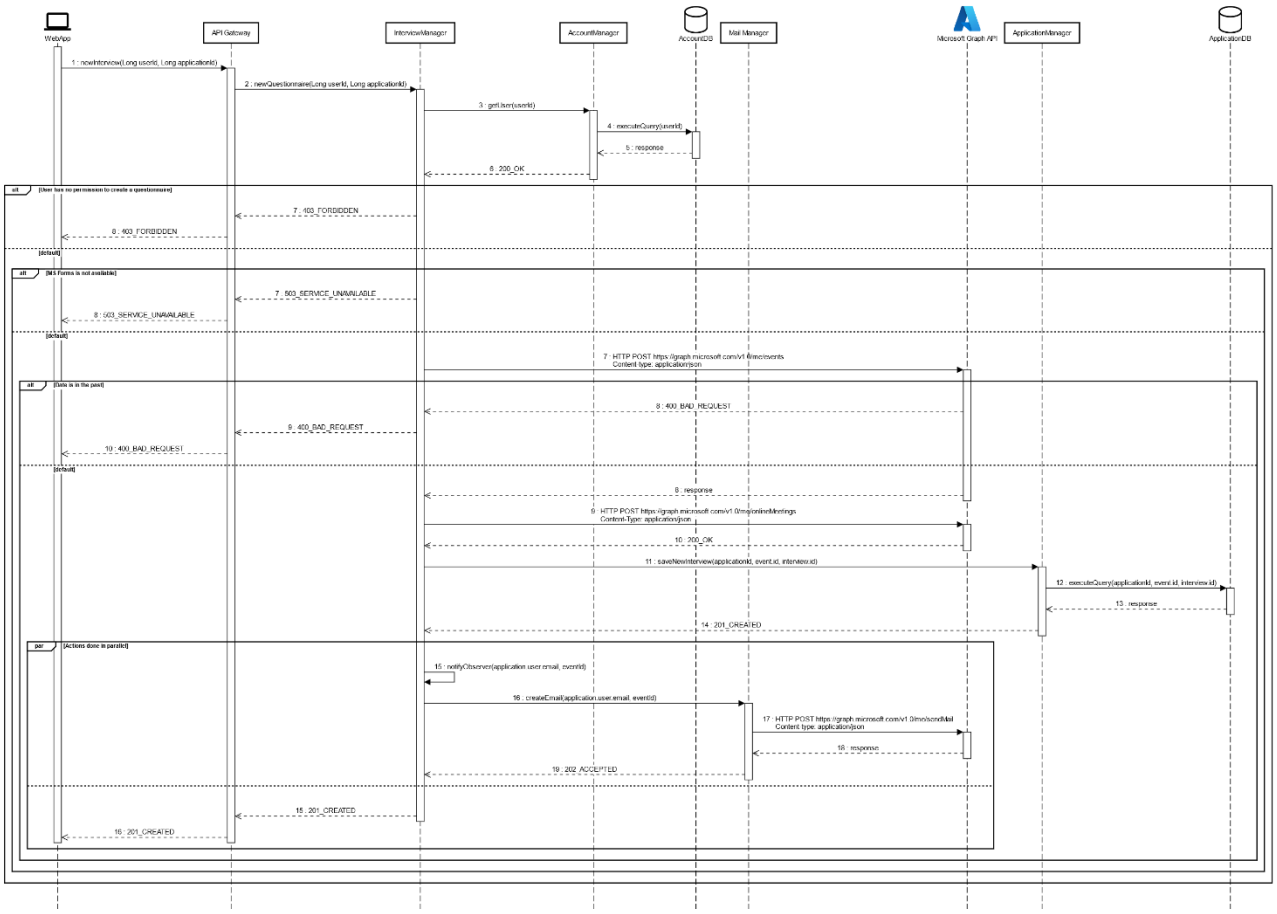
[Read questionnaires]



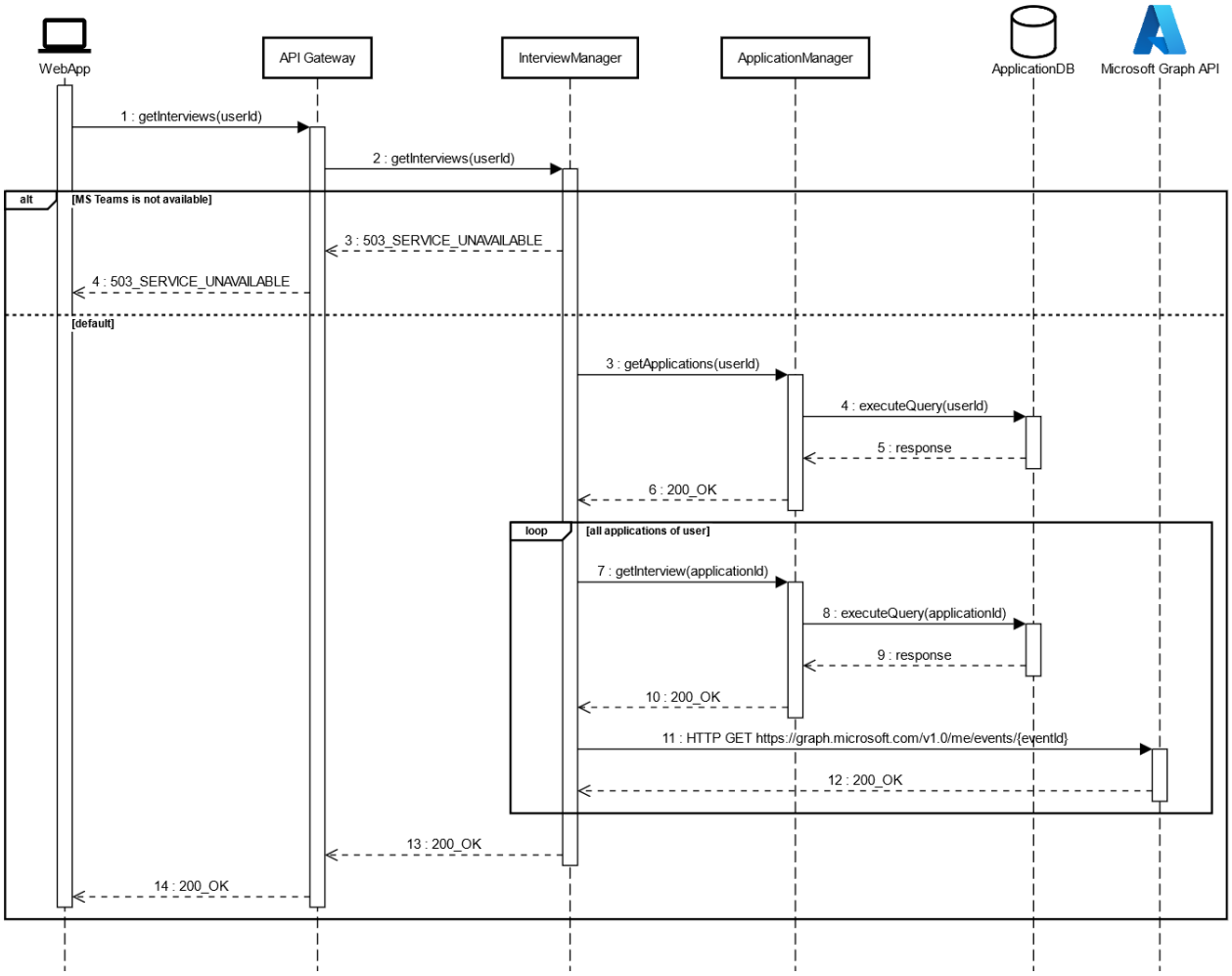
[Submit questionnaires]



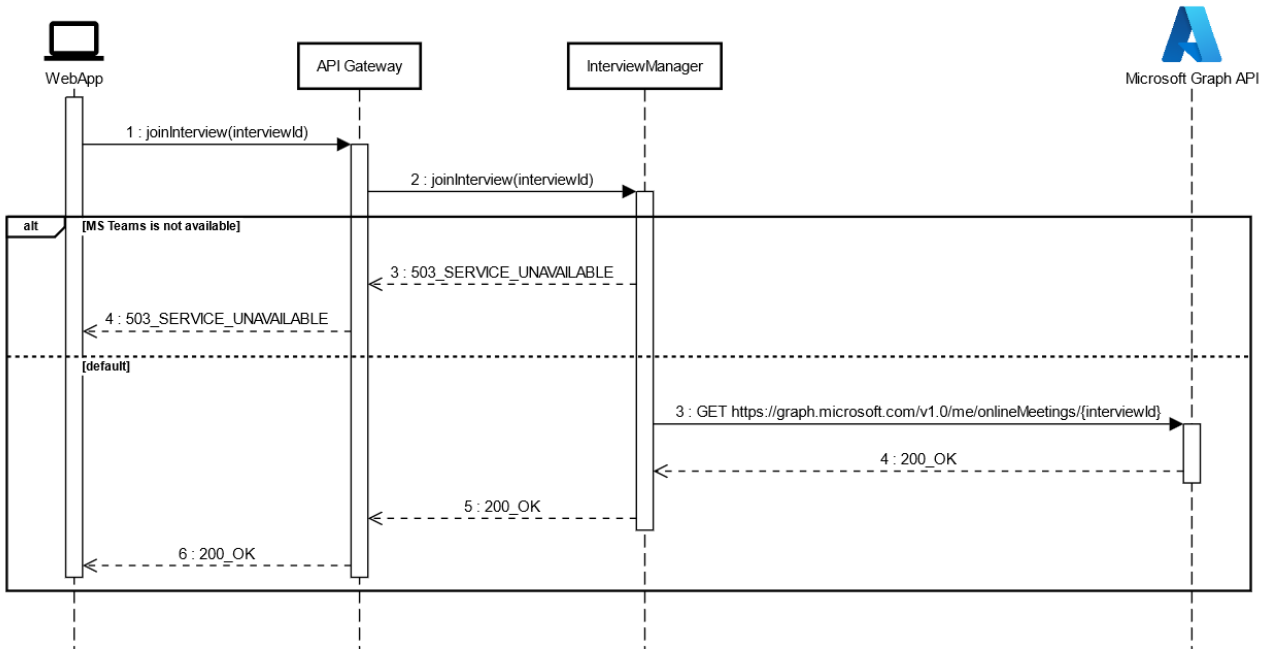
[Schedule meetings]



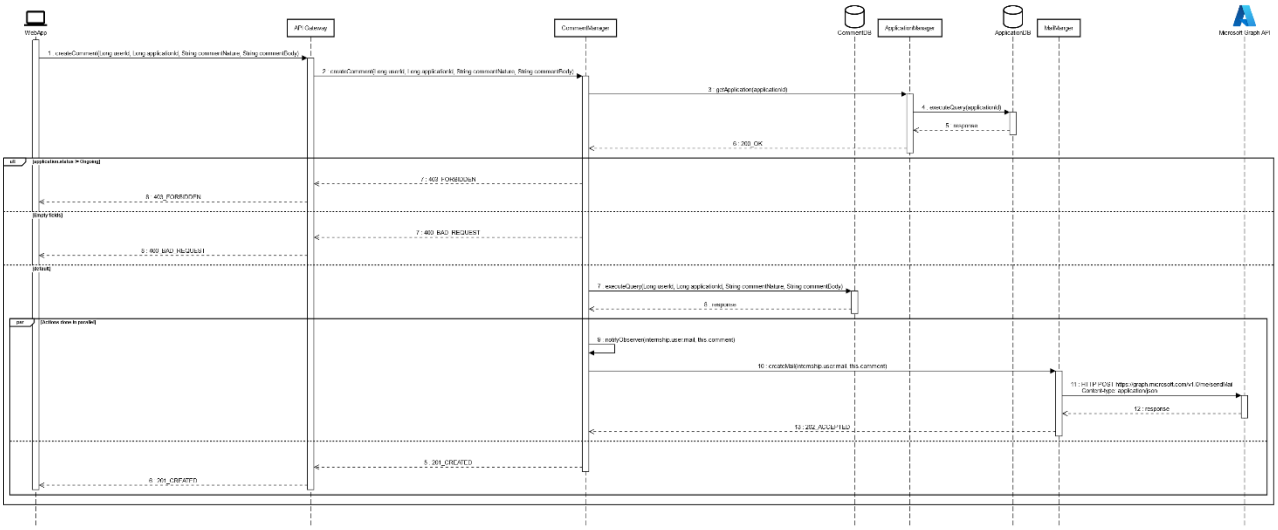
[Read scheduled meetings]



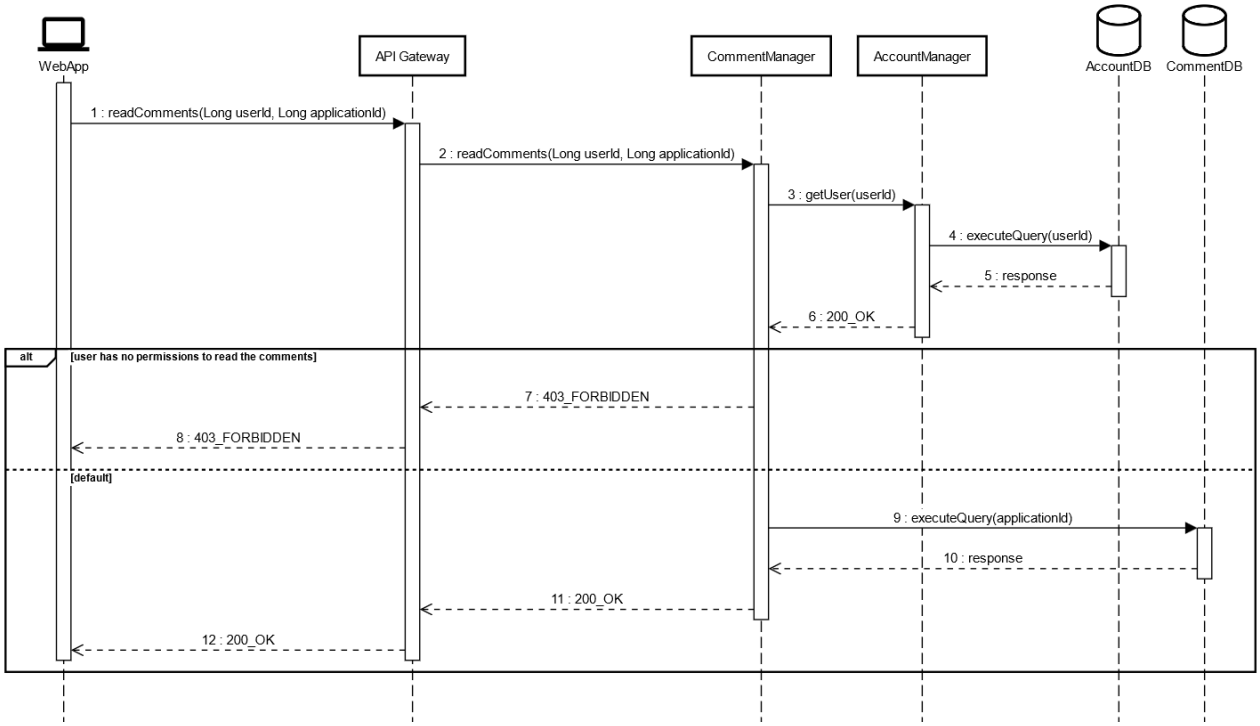
[Participate in meetings]



[Create comments]



[Read comments]



2.6 Selected architectural styles and patterns

In this section the various architectural styles are presented:

- **Microservices architecture**: the system is designed using microservices which provide high scalability and a high degree of decoupling between components. Based on the *divide et impera* principle, this architecture splits the system into smaller services, each specialized in satisfying a smaller set of requirements; this helps speeding up the development of the system by reducing communication overhead between developers and allowing easier testing of the "atomic" components.
- **REST API**: the system provides a RESTful API for lightweight communication that users can exploit when interacting with the system. This choice works well with microservices because it provides a technology-neutral base making the technical implementation of the single services irrelevant.
- **API Gateway**: this design is popular with microservices because it abstracts the complexity of the system to a single interface that the end-user interacts with. This gateway can also act as a load balancer, forwarding requests equally to the various instances of the services.
- **Server-side service discovery**: all microservices contact the discovery service as soon as they start to communicate what service they offer, what IP address and port they use. When a service has to talk to another one, it first contacts the discovery service which returns the IP/port tuple of the requested service, after the first contact all the communications are done directly. Thanks to this discovery service, there's no need to hardcode IP addresses in the various services, which greatly improves scalability. The discovery service can also act as a load balancer since the cost of adding a new instance of a service is just the cost of the initial contact with the discovery server.
- **Observer-Observable pattern**: in order to automatically start the process of sending notifications, the services that may create notifications push to their subscribers the information with a standard function, such as `notifyObserver(Subject, Subscriber)`. The subscriber's implementation of the function will then create both a new request to the Mail Service for the email notification and an internal notification for the WebApp in case the user has disabled mail notifications.

3 User interface design

All the significative user interface mock-ups are already present in the RASD, section 3.3.1 User interface. Please refer to the RASD.

4 Requirements traceability

This section shows which system components concur to the satisfaction of each requirement defined in the RASD:

Requirements	• R1: The system allows students to register to the platform by providing their personal information. • R2: The system allows companies and universities to register to the platform by providing business information. • R3: The system allows registered users to login using the specified credentials. • R4: The system allows students to list their experiences, skills and attitudes. • R5: The system allows students to upload their CV.
Components	• WebApp • Account Manager • DBMS

Requirements	• R6: The system allows students to filter internships according to their preference. • R8: The system allows companies to create internships by writing a description of the job and other relevant information.
Components	• WebApp • Internship Manager • DBMS

Requirements	• R7: The system allows students to create an application with a company if the internship is interesting. • R9: The system allows companies to accept an application from a student if the student's CV corresponds to the company's need. • R10: The system allows companies to decline an application from a student if the student's CV does not correspond to the company's need. • R11: The system allows students and companies to get in touch with each other with a selection process, after the initial contact is made.
Components	• WebApp • Application Manager • Account Manager • Mail Manager • MS Graph API • DBMS

Requirements	• R12: The system allows companies to collect information from students during the selection process, with questionnaires using MS Forms.
Components	• WebApp • Questionnaire Manager • Account Manager • Application Manager • MS Forms API • Mail Manager • MS Graph API • DBMS

Requirements	• R13: The system allows companies to collect information from students during the selection process, with interviews using MS Teams.
Components	• WebApp • Interview Manager • Account Manager • Application Manager • MS Graph API • Mail Manager • DBMS

Requirements	• R14: The system allows students and companies to keep track of the ongoing internship, by providing spaces where both parties can write their opinions, problems and other relevant information regarding the internship. • R15: The system allows universities to monitor the comments for ongoing internships of its students.
Components	• WebApp • Comment Manager • Application Manager • MS Graph API • Mail Manager • DBMS

The following traceability matrix sums up the relations:

	Account Manager	Internship Manager	Application Manager	Questionnaire Manager	Interview Manager	Comment Manager	Mail Manager	WebApp	DBMS	MS Forms API	MS Graph API
R1	X							X	X		
R2	X							X	X		
R3	X							X	X		
R4	X							X	X		
R5	X							X	X		
R6		X						X	X		
R7	X		X				X	X	X		X
R8		X						X	X		
R9	X		X				X	X	X		X
R10	X		X				X	X	X		X
R11	X		X				X	X	X		X
R12	X		X	X			X	X	X	X	X
R13	X		X		X		X	X	X		X
R14			X			X	X	X	X		X
R15			X			X	X	X	X		X

5 Implementation, integration and test plan

Since the system is based on microservices, all the different services can be implemented and tested in parallel by different teams.

The critical modules strategy has been chosen for the project. This means that the “riskiest” modules should be developed before anything else. In the case of S&C, the riskiest module is the Application Manager since it’s the service which the whole platform revolves around. The API Gateway is equally important, otherwise the end-user will not be able to use the platform, but it’s less custom than Application Manager so it’s not as risky to develop. Broadly speaking, each service must be tested separately with the use of stubs and/or drivers, then tested again with the services it may come in contact after deployment to ensure optimal functionality and inter-operability.

The following table contains the order of service development, based on the risk that component has:

Component	Impact of possible failure	Order of development
Application Manager	5	1
API Gateway	5	1
Account Manager	4	2
Internship Manager	3	3
Questionnaire Manager	2	4
Interview Manager	2	4
Mail Manager	1	5

The following table maps the integer value of impact of failure to a brief description:

Value	Impact severity
5	**Very high**. The system is not reachable, or the core functionality is not available rendering the platform useless.
4	**High**. Some important functionality is not available and some users may not be able to use the platform.
3	**Moderate**. Some important functionality is not available and some users may have limited actions.
2	**Low**. Some functionality is not available because of an offline dependency, but the platform works in all its core components.
1	**Very low**. Some functionality is not available, but the platform works in all its core components.

The various services have dependencies on each other, which need to be evaluated so that the correct tests are carried out to check interactions. The following graph shows the possible dependencies:

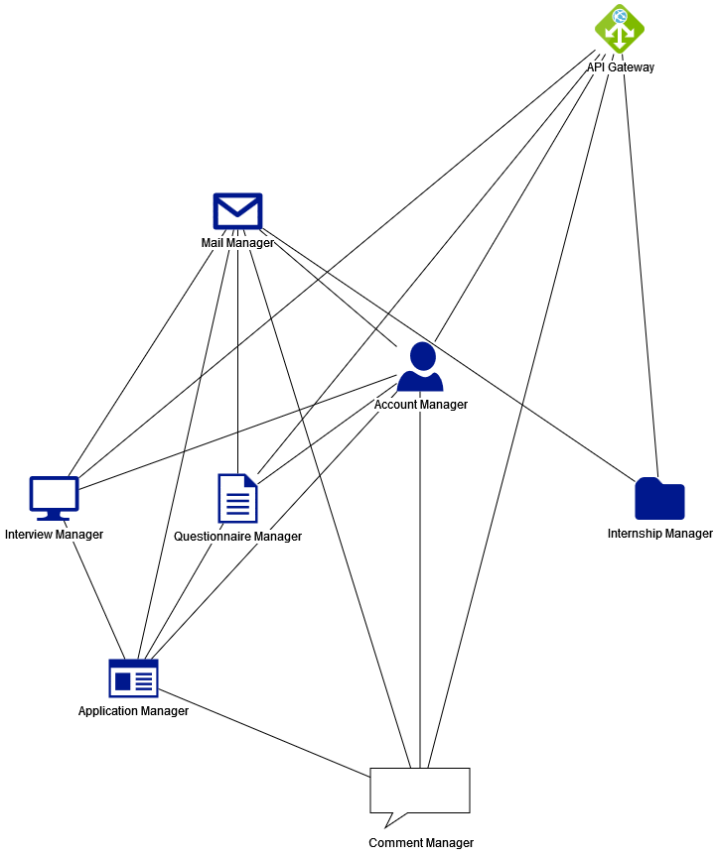


Figure 5-1: Dependency graph

The testing between the interoperability of two elements should not be carried out until the dependency between both is resolved.

6 Effort spent

Alessandro Travaini

Section	Time spent [h]
Introduction	1
Architectural design	20
User interface design	6
Requirements traceability	2
Implementation, integration and testing	2

7 References

- HTTP return status codes: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
- MS Forms API: https://techcommunity.microsoft.com/discussions/microsoftforms/api-to-access-ms-forms/1463830/replies/2896868#M10042
- MS Graph API reference: https://learn.microsoft.com/en-us/graph/api/overview?view=graph-rest-1.0