Tunisian Republic
Ministry of Higher Education and Scientific Research
University of Sousse

Higher Institute of Applied Sciences and Technology of Sousse

DEPARTEMENT OF COMPUTER SCIENCE

GRADUATION PROJECT REPORT

In order to obtain the:
degree of engineering in Computer Science
Option: Software Engineering

Communication agency platform

Elaborated by:

Fedi Kouzana

Hosting Company

University year: 2020/2021

Subject code: FI-GL21-045

Contents

List of Figures

List of Tables

Acronyms

RAD Rapid application development

XP eXtreme Programming

FDD Feature driven development

CD Continuous delivery

CI Continuous integration

UML Unified Modeling Language

Introduction

Chapter 1

General context

Introduction

This first chapter offers an overview of the project. First of all, we describe the project framework. The host organization will be presented next. Following that, we will disclose the problem and then emphasize the existing solutions in order to suggest a suitable ones. Finally, we introduce the process of development.

1.1 Project context

This project is named **Digital Communications Agency**; a project management tool for communications agencies. It was designed in order to prepare the end of the study project presented with a view to obtaining the national diploma of computer engineer in Software Engineering at the Higher Institute of Applied Sciences and Technology of Sousse, for the 2020/2021 academic year. The work is carried out within the Comguru agency.

1.2 Hosting company

Figure 1.1: Comguru agency logo

This section will introduce the hosting company, as well as its various sections and their missions.

1.2.1 Presentation

Comguru is a 360° agency which can take responsibility and assist any project at all phases of growth and development. Two partners established the agency in Sousse in 2013, **M. Sendy Zardi** and **M. Fadhel Naïja**.

1.2.2 Different departments

The Comguru agency consists of a number of departments:

- **Administrative Division**: employee and company management.
- **Marketing Division**: Constantly tries to establish and enhance the company's and its customers' marketing strategy, and collaborates with all other departments to clearly describe the needs based on the types of requests.
- **Graphic Division**: Creates and implements graphic identities, as well as produces media, whether printed (flyer, urban display, leaflet, stand, catalog...) or virtual (electronic flyer, online mock-up, or applications...). Because every sort of project is born in the graphics department, it is termed a launch department.
- **Technical Division**: This is the key innovation department; it is constantly on the lookout for new technologies in all sectors in order to provide the best and most up-to-date tools to all departments, therefore boosting efficiency. as well as the company's work quality He is also in charge of the whole internal technology infrastructure (local network, internet, computer hardware). Not to mention that this department handles any development request, whether it is Web, Mobile, Application, or other.
- **Audio-Visual Division** : Is in charge of all audiovisual creations, including video clips, institutional videos, and filming (products, person or location). This department is required for the development of bespoke graphics or the completion of an e-commerce site...
- **Content management department Division** : This is where the project valuation department is carried out; it is there to generate tailor-made content. His employment is mostly filling websites and producing catchy social media postings.

1.2.3 Clients

Comguru's clientele is varied, comprising both Tunisians and foreigners.

Figure 1.2: Comguru clients

- **Hookahvar**: Online shop specializing in the sale of chichas.
- **BNA bank**: National agriculture bank.
- **Graïet**: Electronics retailer.
- **TuniBet**: Tunisian sport betting site.
- **L'epi d'or**: Tunisian pasta maker.
- **VanDutch France**: Location and sale of luxury boats.

1.3 Problematic

Team members of the agency employ different tools during the work process, from first talks to project fulfillment.

The methods for processing the request, its conceptual analysis and technical implementation will appear in the middle. The whole process necessitates the involvement of many softwares and programmes, sometimes tandem in order to oversee the development of the project (Slack, Trello, emails, etc.) or to contact the client, share project references and documents.

Along with this plethora of applications comes a multitude of authentication or login accounts. The diversification of software and programs scatters and poorly organizes the various parts, files, elements, and folders related to the project, even if it means losing some data or creating a problem of dialogue within the team, which will disrupt the smooth running of the work process within the upstream and downstream agency.

Indeed, lack of organization is a problem even on the client's side, who, after receiving the rendering from the agency (image / video / specific file), finds himself making his returns or sending related directives to his project, either directly (he travels on site within the agency and the remarks are then dictated orally and the manager must take notes), or remotely through e-mail or over the phone (again we find ourselves collecting information from both sides to send it to the team).

Consequently, we have a communication problem with a loss of time between the customer and the agency. Incomplete and/or poorly communicated information will interfere with the timetable previously agreed between the two parties.

In the face of such a load and lack of structure, some agencies may use freelancers to consolidate their workforce and provide customers with good outcomes. Though this approach is efficient in providing customers in a timely manner, the customer and the agency may lack follow-up. A non-team Member who is not or does not refer to the agency, cannot offer the agency with a guarantee that a trail of email exchanges or telephone conversations will be provided. Consequently, due to lack of control over the external members freed from a manager to oversee and supervise his job, we will have a follow-up difficulty.

To summarize, the creative process inside a communications agency necessitates the involvement of a number of internal and external stakeholders, clients and team members, as well as a number of programs and software. In addition to the team's expertise and competence, communicating and establishing a continuous fluid line from the start of a project to its completion may encounter particular excitement at various places along this line, whether upstream or downstream. Gathering all of the required resources to launch the project, manage demand, allocate duties, oversee work, maintain the smooth operation of client / team exchanges, and eventually deliver the project would be beneficial to the agency and the team's performance.

1.4 Analyzing existing solutions

The examination of what currently existing is a necessary stage at the beginning of every endeavor. Indeed, this study enabled us to identify the strengths and limitations of current solutions in order to minimize gaps and give insight into the project's relevance, practicality, and continuity. In this example, we will seek for a pre-existing solution that combines the major and required functionality of the program stated in the 1.3 section and addresses the difficulties that it brings.

1.4.1 Existing solutions presentation

We based our search for existing ones on the presence of the following functionalities: messaging, project management, permission management, client-specific area, invoicing, file sharing, and administration section. And it was only after lengthy investigation that we discovered these answers. Here's a high-level overview of each solution.

- **Slack** [13] : Slack is a business messaging tool that links individuals to the information they require. Slack changes organizational communication by bringing individuals together to work as a united team.

Figure 1.3: Slack logo

- **Discord** [5]: Discord is another another messaging program that is largely used by video game communities. However, in the last two years, numerous other groups have begun to adopt it as well. It is extremely customizable, which is what encouraged this diversity of communities.

Figure 1.4: Discord logo

- **Trello** [11]: Trello is a collaborative application that organizes tasks into boards and cards using the Kanban[7] technique. It displays the status of tasks, who is working on what, and the process's present state.

Figure 1.5: Trello logo

- **ClickUp** [3]: ClickUp is software for project management and collaboration. It is an application that attempts to increase the productivity of its users, regardless of which team they are a part of. As a result, persons from various domains may simply collaborate. They can handle projects and tasks in a variety of methods that consumers want.

Figure 1.6: ClickUp logo

- **ActiveCollab** [1]: ActiveCollab is a project management software that combines project management, time tracking, and invoicing. Organize your work in a single tool, measure the time spent on each job, and create and send invoices. ActiveCollab distinguishes between an investment and a cost.

Figure 1.7: ActiveCollab logo

1.4.2 Benefits and drawbacks

In the table 1.1, we will give the benefits and drawbacks of the solutions that we have discovered based on our previous experience with the solutions or on the trial period spent on paid ones.

Table 1.1: The benefits and drawbacks of existing solutions

Solution	Benefits	Drawbacks
Slack	• The ability to integrate additional apps. • Discussion history. • File download. • Unlimited number of users.	• The absence of the client actor. • The initial setup is complex.

Table 1.1: The benefits and drawbacks of existing solutions

Solution	Benefits	Drawbacks
Discord	• Allows private one-to-one messaging as well as group communication. • Allows easy configuration of servers for specific groups. • Unlimited number of users. • Screen sharing.	• Lack of traceability. • The absence of the client actor.
Trello	• Recognize when a deadline is approaching in instantaneously. • Trello follows the Kanban system. • Flexible.	• Difficult to manage large projects. • Very limited download size. • No chatting system. • Lack of traceability.
ClickUp	• Intuitive graphical interface. • Multitude of views. • Calendar integration. • Invoice management.	• Functional changes occur frequently. • A proper setup is required in order to make the most of the functionality.
ActiveCollab	• Invoice management. • An area devoted to the client. • Configurable and customizable.	• Basic report structure. • Cluttered user interface.

1.4.3 Comparative between existing solutions

We will compare the fundamental features and exclude extensions and third-party software that may be installed, because the installation of third-party software requires the acquisition of an extra account, which can be free or paid. Assuming that money is not a limitation, we are faced with the previously noted dilemma of a plethora of authenticating accounts.

In the table 1.2, we will specify the features discovered in the solutions found from the previously given list of features. We will also list the free programs discovered; applications that provide both free and paid packages or a free trial period will be classified as paid.

Table 1.2: Comparative between existing solutions

Solution	Slack	Discord	Trello	ClickUp	ActiveCollab
Project management	✗	✗	✓	✓	✓
Permission management	✓	✓	✓	✓	✓
Client devoted area	✗	✗	✗	✓	✓
Invoicing	✗	✗	✗	✓	✓
Administration	✓	✓	✓	✓	✓
Chatting system	✓	✓	✓	✓	✓
File share	✓	✓	✗	✓	✓
Free	✗	✓	✗	✗	✗

1.4.4 Pricing

In the table 1.3, for each solution, we will select the best package for our needs inside the agency and provide a monthly and yearly pricing. It is also said that the agency has about fifteen employees since the majority of the offers are provided by user.

Table 1.3: Price comparisons between existing solutions

Solution	Slack[10]	Discord	Trello[12]	ClickUp[4]	ActiveCollab[2]
Offer	Business+	Basic	Business Class	Business	Pro
Per month	€225	Free	€187,5	€285	€93,75
Per year	€2.250	Free	€1.800	€1.620	€1.125

1.4.5 Decision

Finally, the solutions that we discovered and examined do not entirely satisfy the needs that the agency is looking for.

1.5 Proposed solution

Our approach entails creating and implementing a web platform that includes all of the tools required for a remote virtual communication agency. This platform enables the creation of a management system for projects, employees, and clients, with the goal of implementing methods of communication between users, including the ability to exchange digital data and comment on the results provided, in order to ensure accurate feedback and reliable, simple, and efficient communication.

1.6 Adopted methodology

The nature and scale of the project influence the choice of development process model. When it comes to a project where data is not obtained from the start, needs are insufficient or even ambiguous, he recommends using an iterative or prototype-oriented strategy.

1.6.1 Agile methodologies

Agile methodology is a project management technique best recognized for its use in software development, in which requirements and solutions are created collaboratively by self-organized teams. The Agile Manifesto ideals and principles were developed to bridge holes in traditional development approaches such as the waterfall process. Agile project management is a popular approach for delivering complicated projects due to its versatility. It places a premium on cooperation, adaptability, continual development, and high-quality output. It seeks to provide clarity and measurability by tracking progress and creating products using six primary "deliverables."
We list the most well-known agile methods:

- Scrum
- Kanban
- Rapid application development (RAD)
- eXtreme Programming (XP)
- Feature driven development (FDD)

1.6.2 Kanban

Kanban[8] is concerned with visualizing your job, minimizing the amount of work in process, and increasing efficiency (or throughput). The Kanban team is committed to decreasing the time it takes to execute a project (or use case) from beginning to end. They utilize Kanban to do this and are constantly striving to enhance their process.

1.6.2.1 Kanban vs Scrum

Kanban is frequently confused with Scrum, although there is a significant distinction between the two. Table 1.4 depicts these distinctions.

Table 1.4: Comparative between Kanban and Scrum

	Kanban	Scrum
Cadence	Continuous flow	Regular fixed length sprints (ie, 2 weeks)
Release methodology	Continuous delivery	At the end of each sprint
Roles	No required roles	Product owner, scrum master, development team
Key metrics	Change can happen at any time	Teams should not make changes during the sprint

1.6.2.2 Kanban board

Kanban is all about visualizing your work; to do so, we use a Kanban board, which is a project management tool that utilizes cards (user stories / tasks) and columns (Kanban process stages) to keep everything organized and transparent for everyone.

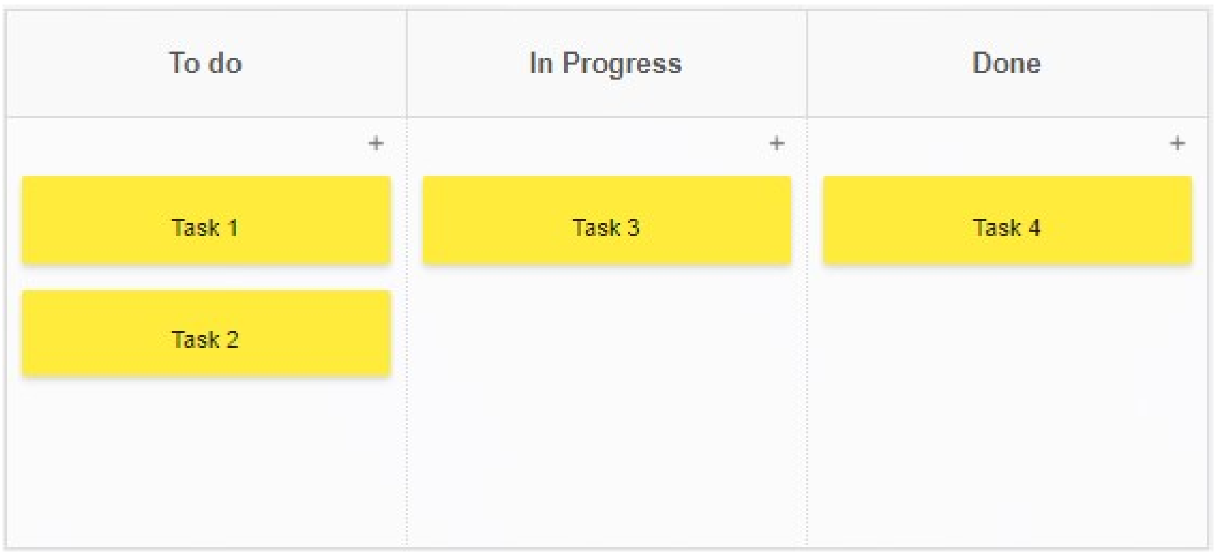


Figure 1.8: Kanban board

1.6.2.3 Kanban WIP limit

WIP[9] limitations in agile development define the minimum and maximum amount of work permitted for each state of the process. Limiting the quantity of work in progress might help discover inefficiencies in the team's process. The bottleneck in the team's delivery pipeline is plainly evident before the situation deteriorates.

1.6.2.4 Continuous Delivery

Continuous delivery (CD) is the practice of frequently delivering work to clients. Continuous integration (CI) is the process of progressively developing and testing code throughout the day. They create a CI/CD pipeline, which is critical for development teams (particularly DevOps teams) to ship software quicker while maintaining good quality.
Kanban and CD work well together because both approaches emphasize just-in-time (and one-at-a-time) value delivery. The sooner a team can bring innovation to market, the more competitive their product will be. Kanban teams, on the other hand, are laser-focused on this: enhancing the flow of work to consumers

1.6.3 Unified Modeling Language

The Unified Modeling Language (UML) was developed to establish a standard, semantically and syntactically rich visual modeling language for the structural and behavioral design and execution of large software systems. Beyond software development, UML has applications such as process flow in manufacturing. It is similar to blueprints used in other industries and comprises of several sorts of diagrams. UML diagrams, in aggregate, define the boundaries, structure, and behavior of the system and the items included inside it.

Figure 1.9: UML logo

1.7 Chronogram

A Gantt chart is a popular graphical representation of a project timeline. It's a form of bar chart that displays the start and end dates of project aspects including resources, planning, and dependencies.

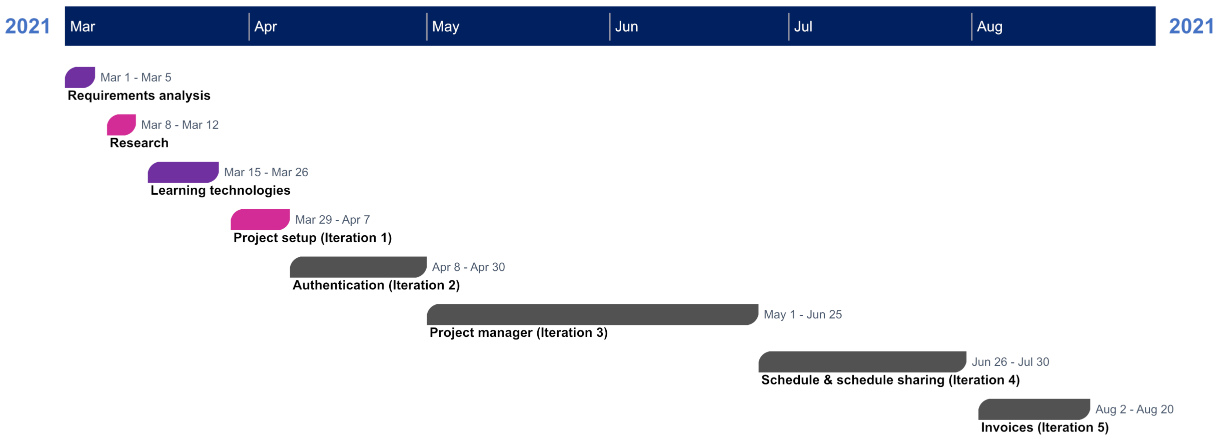


Figure 1.10: Gantt chart for the development process

Figure 1.10 depicts the project timeline.

Conclusion

In this first chapter, we introduced the hosting organization and established the context for our project. We then displayed the problem as well as the existing solution, allowing us to establish a recommended solution. We also chose and adopted a development methodology. In the next chapter, we will do a requirement analysis to gain a comprehensive understanding of the project's needs.

Chapter 2

Requirement specification

Introduction

The first stage of the software development cycle is the requirements specification phase. It is used to identify the system's reactive actors and link each of them to the set of actions with which it intervenes in order to get the best possible result and satisfy our objectives. So, in this chapter, we will first identify the system's actors before defining their functional and non-functional needs. And we'll wrap off this chapter with several use case illustrations.

2.1 Actors

Identifying the actors in the application is one of the initial tasks in system design. An actor is a system-independent entity. It depicts an user or other computer system that anticipates receiving one or more services through an access interface. Our application admits three actors.

- **Admin**: This actor has all of the access privileges necessary to run the system. Its primary role is to manage users, permissions, and program settings as needed.
- **Client**: This actor has well-defined functions.
- **Employee**: This actor will have access to a set of features, selected from a global list of functions devoted to employees, based on the permissions granted by the administrator.
- **Visitor**: This actor will be a potential customer, he can just create a customer account in the platform.

2.2 Functional requirements

Functional requirements are features that satisfy the demands of users. This platform should primarily meet the following functional requirements:

- Admin can:

  - ⋆ Manage users
  - ⋆ Manage projects configuration

- Employee can :
  - ⋆ Communicate instantly with other employees
  - ⋆ Respond to customer feedback
  - ⋆ Manage his tasks
  - ⋆ manage projects and tasks

- Client can :
  - ⋆ Import files to tasks
  - ⋆ Leave notes in tasks and projects
  - ⋆ View projects and tasks
  - ⋆ Print your invoices
  - ⋆ Request the creation or modify a project
  - ⋆ Create personalized requests and validate an invoice

- Visitor can :
  - ⋆ Create a customer account
  - ⋆ Visit the website

2.3 Non-functional requirements

It is critical to consider some restrictions when designing the solution, such as ergonomic, technological, and even aesthetic constraints, in order to ensure that our final product functions properly.

- **Ergonomics** : Our solution must have an ergonomic interface that includes all of the functionality available. In order to adapt to the users' computer expertise, the interface must be straightforward and obvious to use.

- **Security** : Our application must ensure the integrity of the backed-up data. Furthermore, it must safeguard the privacy of user identities.

- **Performance** : Given the huge volume of data being maintained, it is critical to ensure that execution time is kept to a minimum. Even in the case of heavy use, the provided product must be expandable and efficient.

- **Efficiency** : The application must always be functional and free of errors.

- **Extensibility** : It is the possibility of adding or modifying new functionalities.

- **Availability** : The application must be available at all times.

2.4 Use case diagrams

A use case diagram is a graphical representation of how a user could interact with a system. A use case diagram depicts numerous use cases and different sorts of system users. Following the definition of our system actors and their functional needs, we will present a global use case diagram in this part that depicts the actors and their interactions with the system.

2.4.1 General use case

The figure below is a simplified representation of the intended functionality.

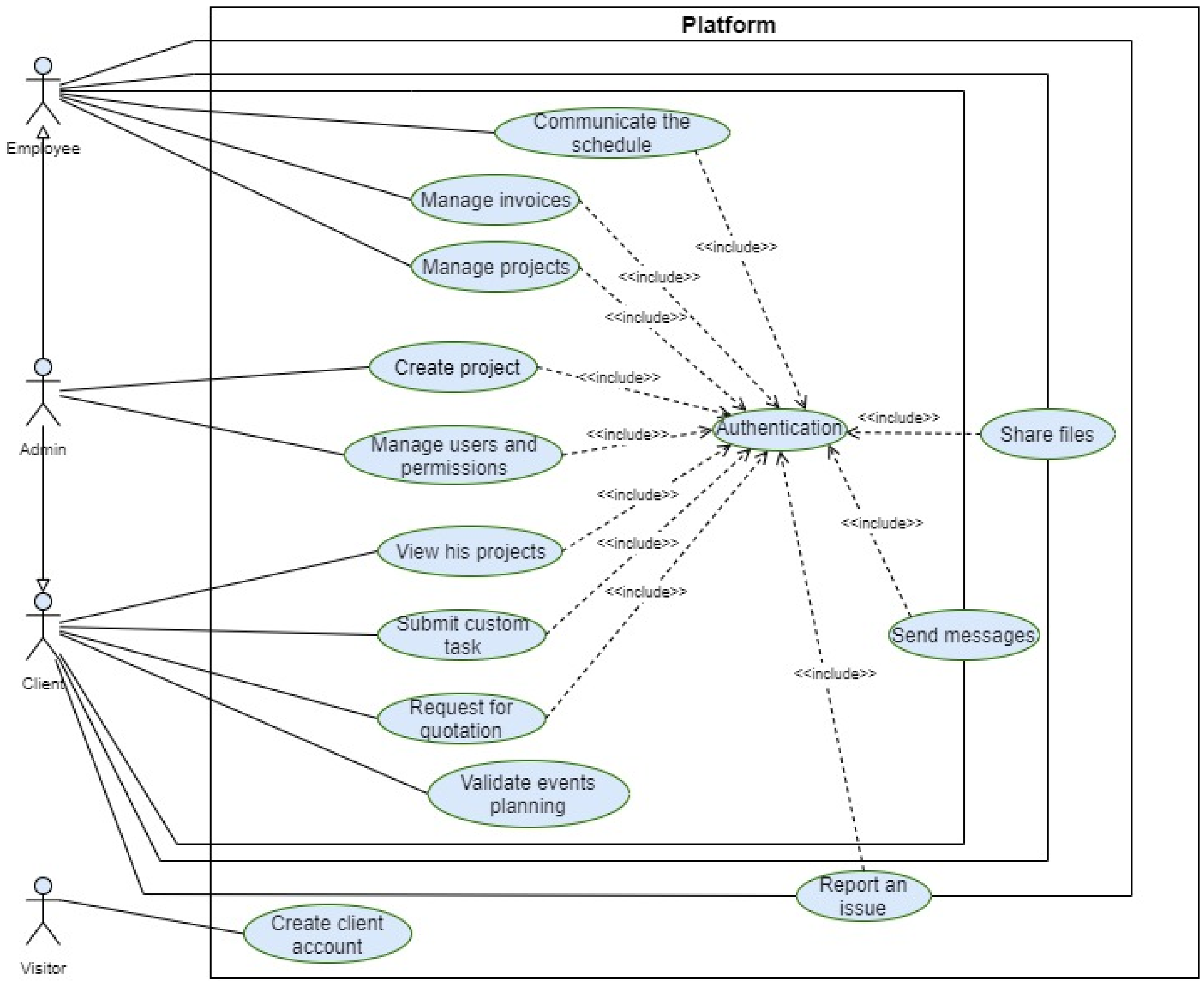


Figure 2.1: General use case diagram

2.4.2 Project manager use case

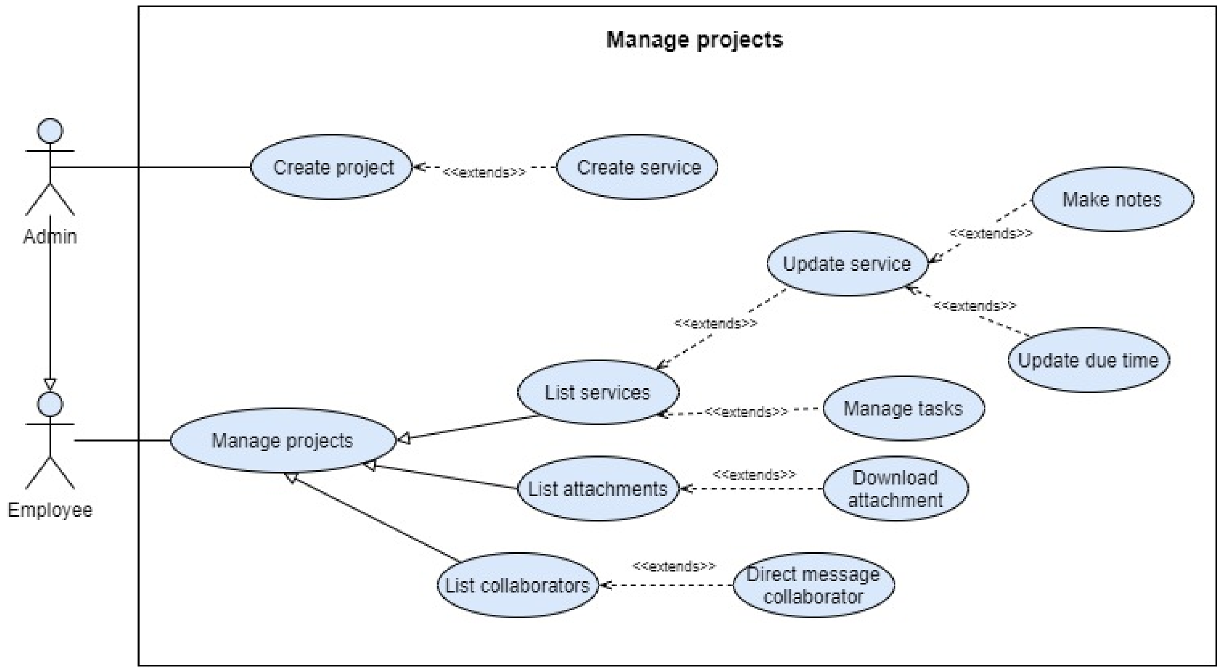


Figure 2.2: Manage projects use case diagram

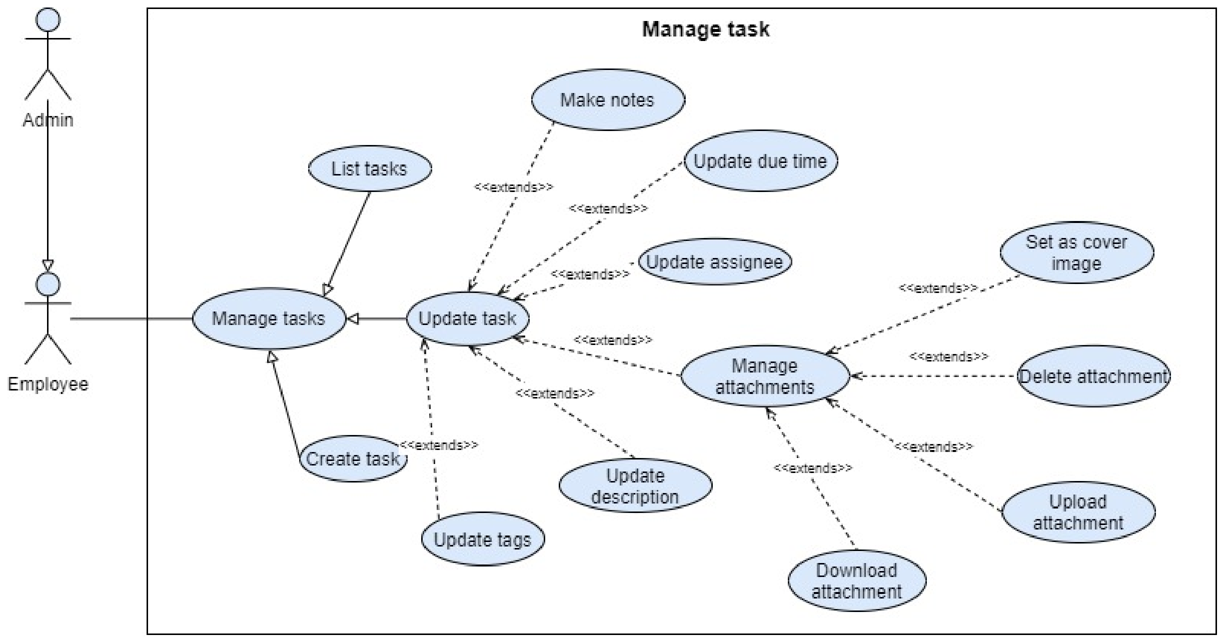


Figure 2.3: Manage tasks use case diagram

Table 2.1: Description of the update task use case

Title	Update tasks
Actor	Employee or admin
Pre-conditions	• Logged in user with role "admin" or "employee". • The task has already been created. • The service that includes the task already exists. • The project that includes the service that contains the task.
Post-conditions	Task information updated.
Default scenario	1. The actor sets due time. 2. The actor upload files as attachment. 3. The actor leaves a note.

2.4.3 Invoice use case

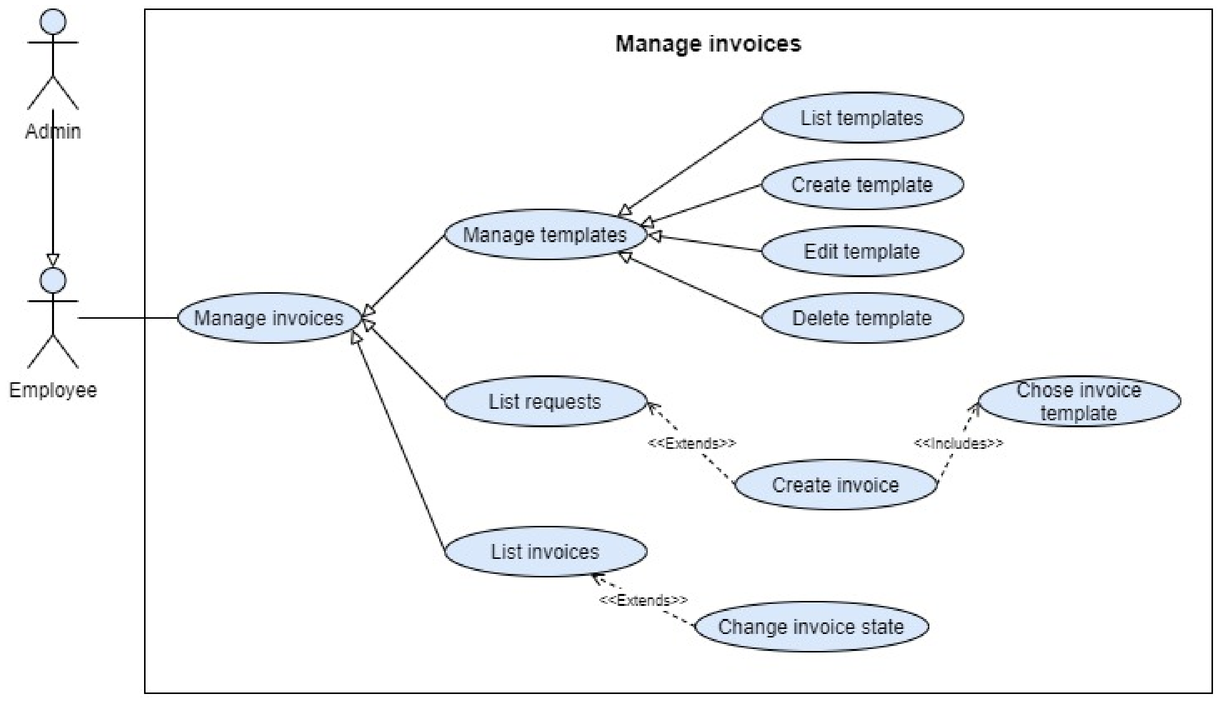


Figure 2.4: Manage invoices use case diagram

Table 2.2: Description of the create invoice use case

Title	Create invoice
Actor	Employee or admin
Pre-conditions	• Logged in user with role "admin" or "employee" • Request was chosen from the list of requests. • Selecting an invoice template.
Invoice created for request.	
Default scenario	1. The actor selects a request. 2. The actor chooses an invoice template. 3. The actor creates the invoice.

2.4.4 Schedule sharing use cases

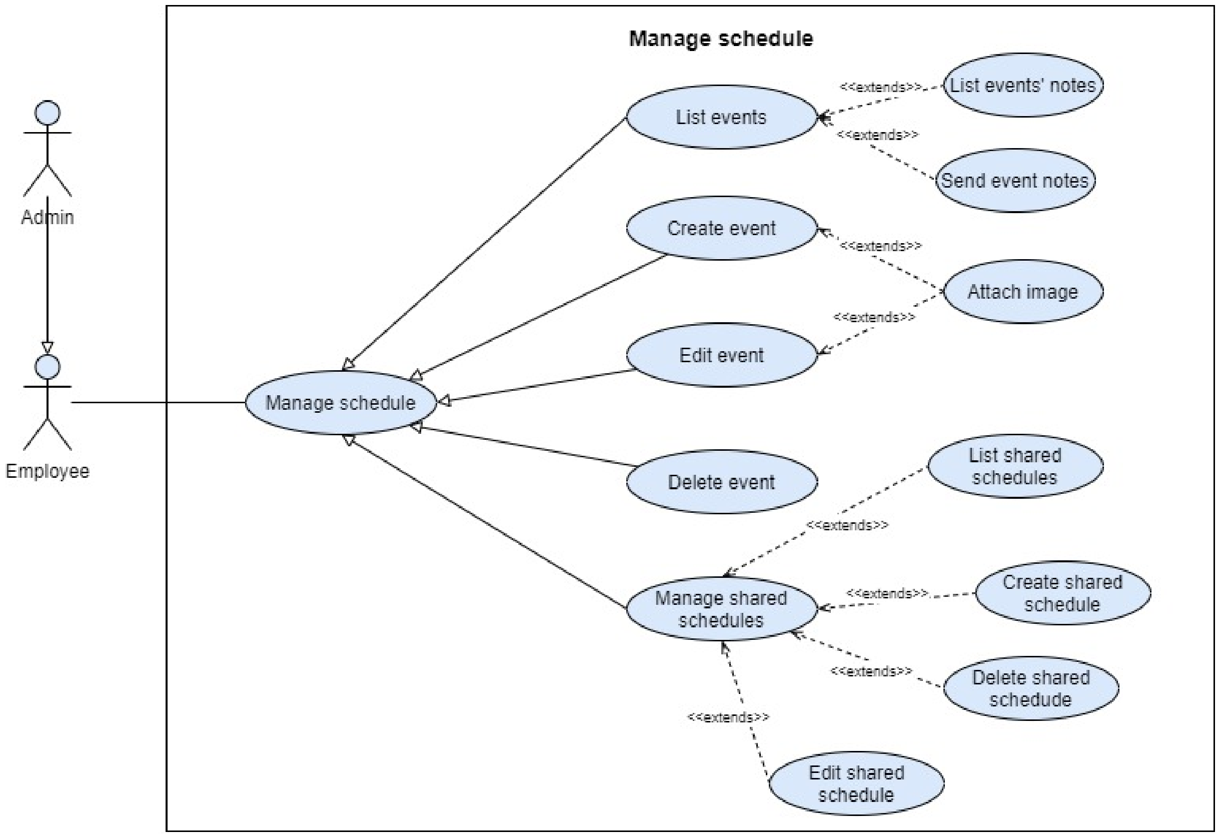


Figure 2.5: Manage schedule use case diagram

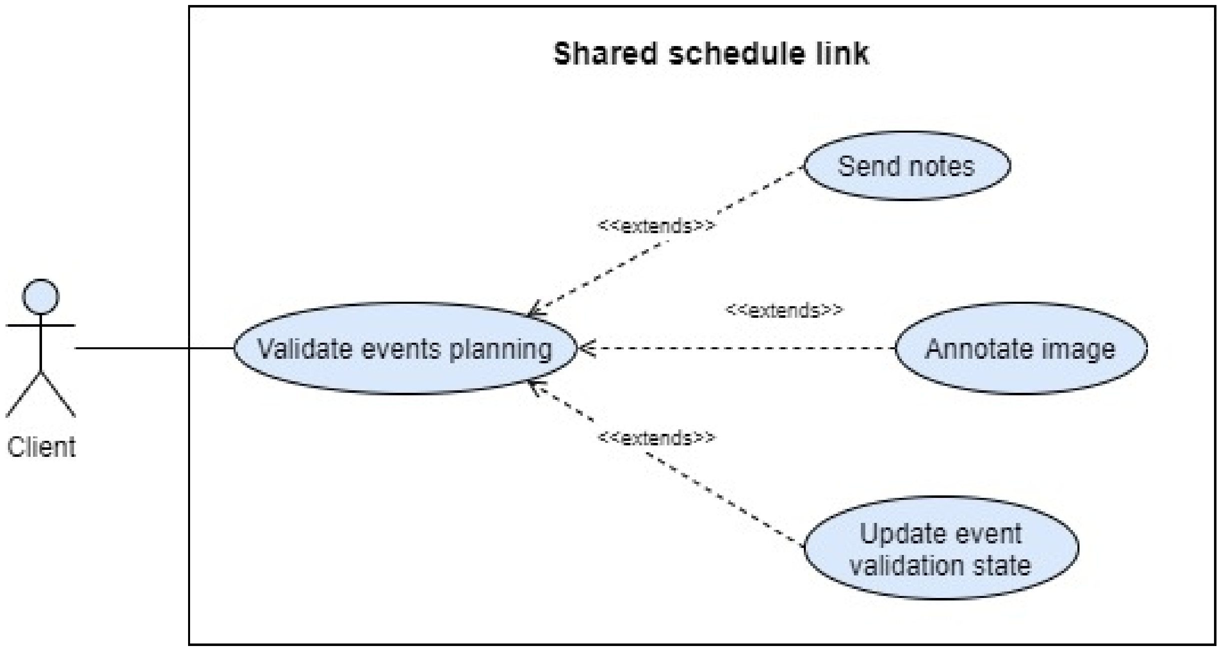


Figure 2.6: Validate events planning use case diagram

Table 2.3: Description of the validate events planning use case

Title	Validate events planning
Actor	Client
Pre-conditions	• A shared URL for event planning has been provided.
Post-conditions	Event planning verified by the client.
Default scenario	1. The actor selects event. 2. The actor leaves notes. 3. The actor annotates image. 4. The actor either validates or rejects the event.

2.4.5 Manage users and permissions use case

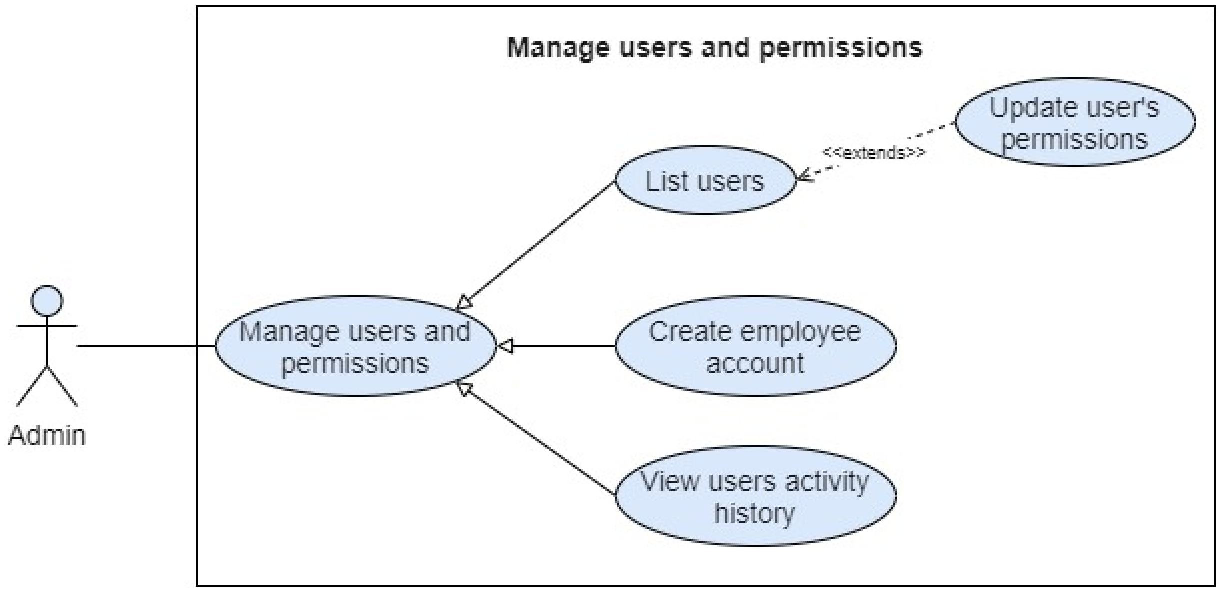

Manage users and permissions
Update user's permissions
<<extends>>
List users
Manage users and permissions
Create employee account
View users activity history
Admin

Figure 2.7: Manage users and permissions use case diagram

Conclusion

We defined the actors and their functional needs in this chapter and expressed them in use case diagrams. We also discussed non-functional needs, which are just as essential as functional requirements. In the next chapter, we will show our development environment, as well as the physical and logical architectures.

Chapter 3

System architecture and technologies

Introduction

To depict the design of software, the program requires an architectural design. The process of identifying a collection of hardware and software components, as well as their interfaces, in order to build the framework for the development of a computer system is referred to as architectural design. During this chapter, we will first show the development environment, including the hardware and software. Following that, we will present the technologies selected for this project as well as the reasons behind our selection. Finally, we shall present the system architectures, both physical and logical.

3.1 Development environment

In this part, we will present the hardware that was utilized throughout the project's phases, as well as the software that was used for the development process or other associated activities.

3.1.1 Hardware

During the research and development phase, we utilized a laptop that had the following characteristics:

- **Description**: LENOVO ideapad 300
- **Processor**: Intel Core i5-6200U 2.30GHz
- **Operating System**: Windows 10 Pro 64-bit
- **RAM**: 8 Go DDR3
- **Storage**: 1To HDD & 480Go SSD
- **Graphics Card**: Intel HD Graphics 520

3.1.2 Software

Throughout the development period, we mentioned the use of several software that made work easier and more productive:

- **Visual Studio Code**:

Figure 3.1: Visual Studio Code logo

Microsoft's Visual Studio Code (often known as VS Code) is a free open source IDE. Windows, Linux, and macOS are all supported by VS Code. Although the editor is small, it contains several powerful features that have helped VS Code become one of the most popular development environment tools in recent years.
We will be writing code with VS Code because it is the most comfortable IDE for us and meets our demands.

- **Ubuntu on WSL**:

Figure 3.2: Ubuntu on WSL logo

WSL (Windows Subsystem for Linux) is a compatibility layer that allows Linux binary executables to operate natively on Windows 10.
We used WSL to install Ubuntu so that we could operate our development servers and database on a Linux system.

- **Altair GraphQL Client**:

Figure 3.3: Altair GraphQL Client logo

Altair is a visually appealing, feature-rich GraphQL Client IDE for all platforms. It allows you to interact with any GraphQL server that you are permitted to visit from any platform.
Altair will be used to test our GraphQL queries and mutations.

- **Robo 3T**:

Figure 3.4: Robo 3T logo

Robo 3T is a well-known resource for MongoDB hosting setups. It provides a graphical user interface (GUI) to interact with data bricks rather than a text-based interface. It is both free and light.
Robo 3T will be utilized to monitor and administer our Mongo database.

- **GitKraken**:

Figure 3.5: GitKraken logo

GitKraken is a graphical user interface (GUI) Git client that allows for the efficient and reliable use of Git on a desktop while also providing most command line functions. GitKraken will be used to interact with our Git repository.

- **Draw.io**:

Figure 3.6: Draw.io logo

Draw.io is a free and open source application for generating charts, flowcharts, and diagrams.
All of our diagrams and system architecture figures were created with Draw.io.

3.2 Technologies used

In this section, we will describe the technologies that were chosen and employed during the development phase, as well as the thinking process and reasoning behind why that technology was picked above others.

3.2.1 JavaScript

Figure 3.7: JavaScript Logo

Considering our solution is a web-based platform, it should come as no surprise that our main programming language will be JavaScript.
JavaScript is a dynamic programming language for computers. It is lightweight and is most often used as a part of web pages, where its implementations allow client-side script to interact with the user and create dynamic sites. It is an object-oriented programming language that may be interpreted. It is also utilized on the server side due to the NodeJs runtime environment.

3.2.2 React

Since we have selected JavaScript as our primary programming language, there are primarily four well-known frameworks or libraries from which to pick from; Angular, React, Vue and recently emerged Svelte.

Table 3.1: Comparative between technologies

Technology	Angular	React	Vue	Svelte
Easy learning curve	✗	✓	✓	✓
Performant and fast	✓	✓	✓	✓
Community contribution	large	large	medium	small

Table 3.1 was created to summarize this article [6] that was written by an expert in the field. He evaluated these four key technologies based on a variety of factors.

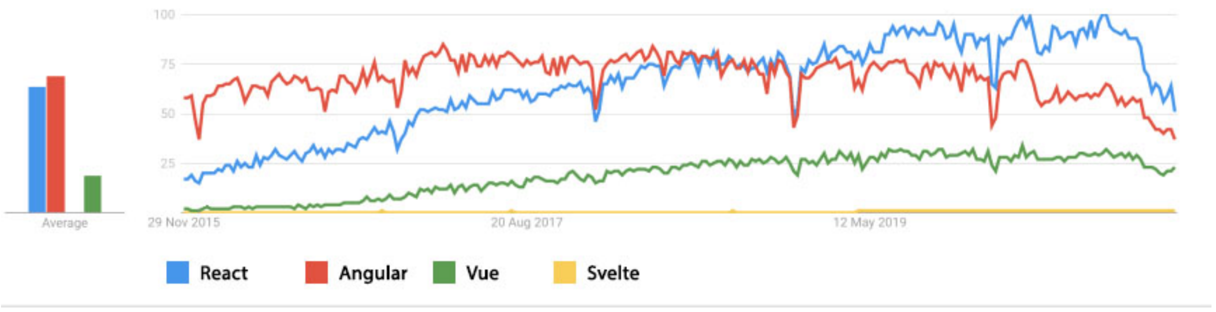


Figure 3.8: Graph displaying interest over time

Figure 3.8 depicts the evolution of interest in the four major frontend technologies over the last five years: React, Vue, Angular, and Svelte.

We chose React because it is the greatest fit for our needs, taking into account the learning curve, popularity, and community support.

Figure 3.9: React Logo

React is a free and open-source JavaScript front-end library for creating user interfaces or UI components. It is maintained by Facebook and the community.

3.2.3 Redux

Working with React as a front-end technology in a large project like ours will necessitate the use of a state management library.
State management is just a method of encouraging communication and data sharing among components. It generates a tangible data structure to describe the State of your application, that you can read and write.

There are primarily two major libraries for this: MobX and Redux. We shall compare them so that we can determine which one to choose.

Table 3.2: Comparison between MobX & Redux

The basis of comparison	Mobx	Redux
Definition	It is a testing library that uses TFRP to create basic state management.	It is the Javascript library that is used to manage the application's state.
Developed	Michel Weststrate develops it	Dan Abramov and Andrew Clark develop it.
Datastore	There are several data storage locations.	There is just one large data storage storage.

Table 3.2: Comparison between MobX & Redux

The basis of comparison	Mobx	Redux
Application	Primarily used for small and straightforward applications.	Primarily used for complex and large applications.
Scalable	Is less scalable comparatively.	Primarily used for scalable applications.

We chose Redux to handle a large portion of the state and its community support. The extensive boilerplate that Redux is known for was first a nuisance. We discovered Redux-toolkit, a package that removes almost all of the boilerplate what made working with Redux more pleasant.

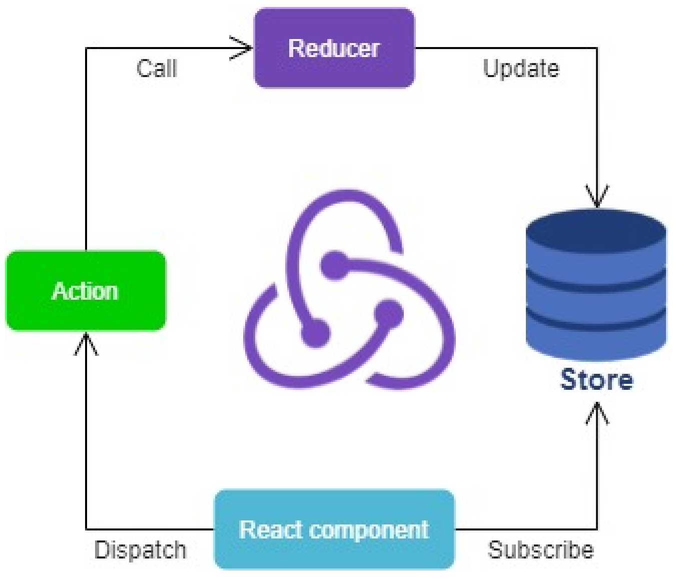


Figure 3.10: Redux data flow

- **Store**: the source of truth that drives our app and it's immutable.
- **Component**: a declarative description of the UI based on the current state.
- **Action**: the events that occur in the app based on user input, and trigger updates in the state.
- **Reducer**: is a function that receives the current state and an action object, decides how to update the state if necessary, and returns the new state.

The unidirectional data flow that Redux follows is seen in Figure 3.10. That means our program will have a one-way binding data flow. Redux minimizes code complexity by setting restrictions on how and when state updates may occur. It is much easier to manage updated states this way.

3.2.4 GraphQL

We were given the option of building the API using REST or GraphQL. So we compared the two so that we could determine which one to go with.

Table 3.3: Comparison between GraphQL & REST

Architecture	GraphQL	REST
Organized in terms of	schema & type system	endpoints
Operations	Query, Mutation, Subscription	Create, Read, Update, Delete
Data fetching	specific data with single API call	fixed data with multiple API calls.
Community	Growing	Large
Performance	fast	multiple network calls take up more time
Learning curve	difficult	moderate
Web caching	via libraries built on top	✓

Figure 3.11 depicts the main difference between the two approaches, which is the number of accessible endpoints. While GraphQL attempts to consolidate all requests into a single location, REST is designed such that each resource is hidden behind a particular endpoint.

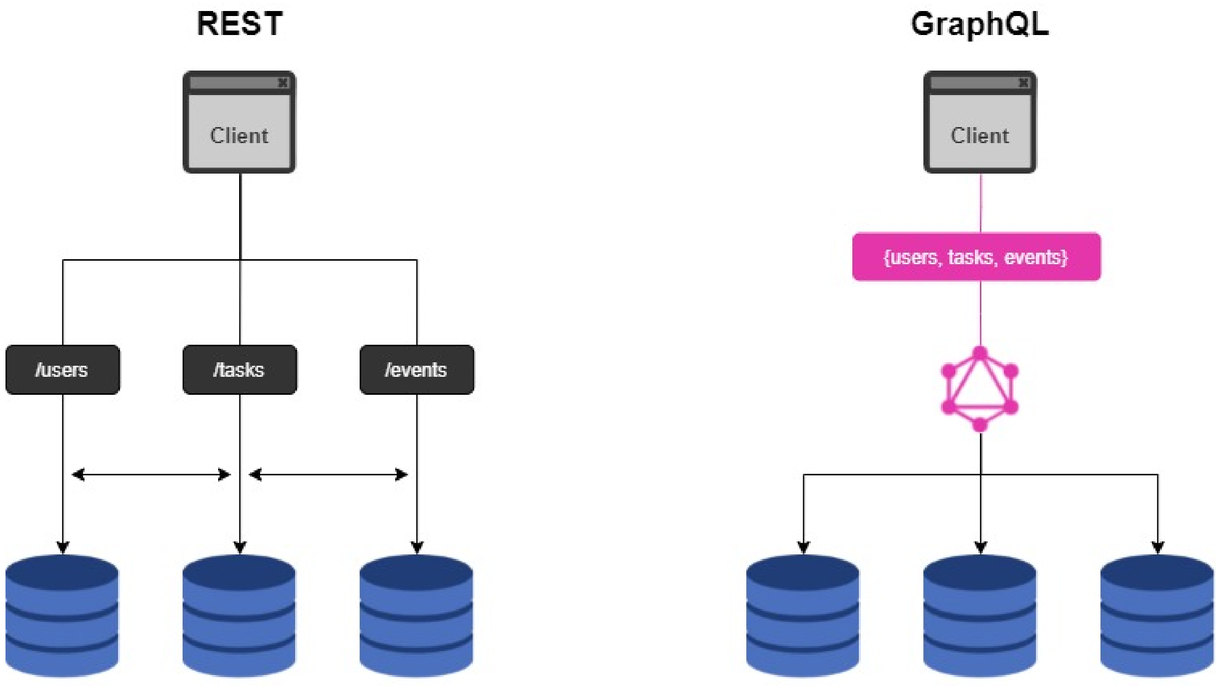


Figure 3.11: Structural difference between REST and GraphQL

So, after researching and learning about both approaches, we decided on GraphQL since it is fast and their data fetching mechanism is appropriate for our use case because it avoids two key drawbacks with the REST approach: over- and under-fetching. This is due to the fact that the only option for a client to download data is to connect to endpoints that deliver defined data structures. It is quite tough to build the API in such a manner that it can offer customers with the exact data they want. It's worth mentioning that GraphQL was developed internally by Facebook.

3.2.5 Apollo GraphQL

Because GraphQL is merely a query language, we need to utilize a platform that will perform all of the heavy lifting for us when we decide on it as the approach for developing the backend API. Apollo GraphQL will be used for this.

Figure 3.12: Apollo GraphQL Logo

The Apollo platform is a GraphQL implementation that can transport data from the cloud (server) to the UI of the application. In reality, Apollo's environment is designed in such a manner that we can use it to handle GraphQL on both the client and server sides of the application.

- **Apollo Client**: Apollo Client is a robust state management library that uses GraphQL to handle both local and remote data. It may be used to fetch, cache, and change application data while also automatically updating the UI.
- **Apollo Server**: Apollo Server is a Node.js package that allows you to connect a GraphQL schema to an HTTP server. It connects with backend data sources in order to get and change data as needed.

3.2.6 MongoDB

When it comes to databases, we must first decide whether to use a SQL database or a NoSQL database. So let us begin by comparing them.

Table 3.4: Comparison between SQL & NoSQL

Parameter	SQL	NoSQL
Type	Relational	Non-Relational
Data	Stored data stored in Tables	Semi-structured stored in JSON files
Schema	Static	Dynamic
Scalability	Vertical	Horizontal
Flexible	Rigid schema bound to relationship	Non-rigid schema and flexible
Performance	Slower then NoSQL	Fast

NoSQL datastores are built to handle a lot more data than SQL databases. The data has no relational restrictions and does not even need to be tabular. NoSQL provides higher-level speed at the expense of robust consistency. And for that, we chose NoSQL databases.

Figure 3.13: MongoDB Logo

MongoDB is a database that is structured as a collection of documents. The documents look like JSON and may be customized with schemas. It is, without a doubt, the most popular NoSQL database. Its features and advantages include the following:

- It is completely free to use.
- Schema that is dynamic
- Horizontal scalability
- Excellent response time with basic inquiries
- Add additional columns and fields without affecting your existing rows or the performance of the application.

Mongoose will be used for MongoDB interactions; It is an Object Data Modeling (ODM) library for MongoDB and Node.js. It handles data associations, does schema validation, and is used to translate between objects in code and their representation in MongoDB.

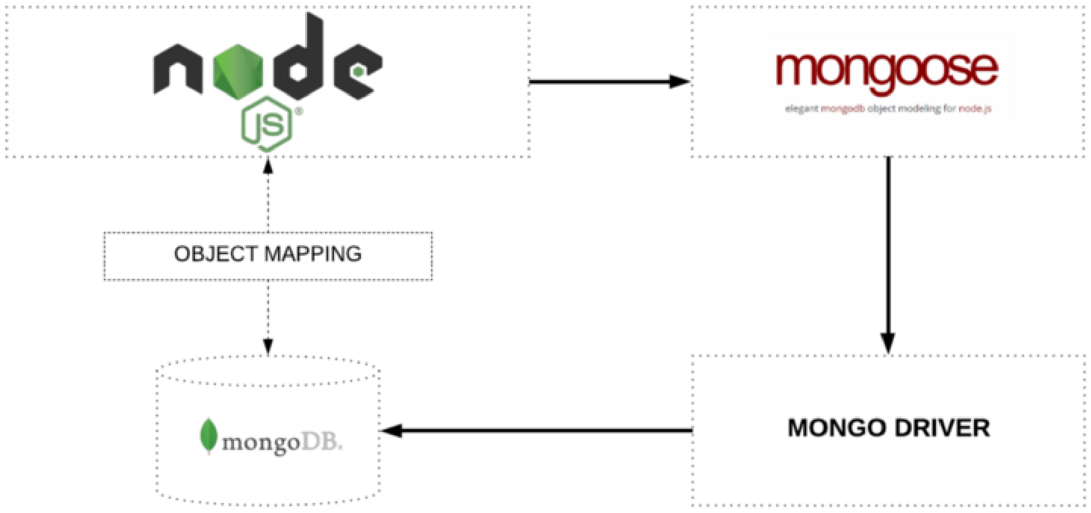


Figure 3.14: Object Mapping between Node and MongoDB managed via Mongoose

3.3 Physical architecture

For client and server applications, as well as the necessity for the server to be centralized (no Peer-to-Peer), only the third tier architecture can be used, since it is the most mature and has overcome the problems of its predecessor, the second tier.

3.3.1 Three-tier architecture

Three-tier architecture is a well-known software application architecture that divides applications into three logical and physical processing tiers; the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.

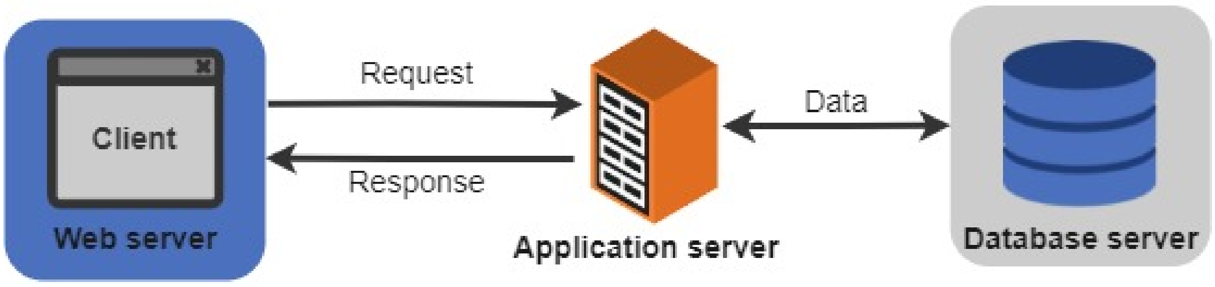


Figure 3.15: Three-tier architecture for web development

3.3.2 The three tiers in detail

Here are the three layers of the three-tier architecture.

- **Presentation tier**: is the program's user interface and communication layer, where the end user interacts with the application. Its primary function is to show information to and gather data from the user. This top-level layer can be accessed via a web browser, as a desktop program, or through a graphical user interface (GUI).
- **Application tier**: also known as the logic tier or middle tier, is the application's beating heart. In this layer, information gathered in the presentation tier is processed - often in conjunction with data gathered in the data tier - using business logic, or a specific set of business rules. The data tier can also be added, deleted, or modified by the application tier.
- **Data tier**: also known as the database tier, data access tier, or back-end, is where the application's data is kept and maintained.

All communication in a three-tier application passes through the application tier. The presentation and data tiers are not able to communicate directly with one another.

3.3.3 Benefits of three-tier architecture

The main advantage of three-tier design is the logical and physical separation of functions. Each tier can run on a different operating system and server platform, such as a web server, application server, or database server, depending on its functional needs. Furthermore, each tier operates on at least one dedicated server hardware or virtual server, allowing each layer's services to be modified and optimized without affecting the other tiers.

Other advantages (over single- or two-tier design) include:

- **More rapid development**: Because separate teams can work on each layer at the same time.

- **Scalability enhanced**: Any tier can be scaled regardless of other tiers if necessary.
- **Reliability enhanced**: The availability or performance of other tiers is less likely to be affected by a fault at one tier.
- **Security enhanced**: Due to the absence of communication between the presentation and the data tier, the well built application tier can act as some kind of internal firewall to prevent SQL injections and other harmful attacks.

3.4 WebSocket

Our program has functionalities that will need the transmission of data from and to the server in real time. We shall take the use of the WebSocket protocol into account right from the beginning.

A WebSocket is a persistent bi-directional connection between a client and a backend service. Websockets can transmit any number of protocols in contrast with HTTP request/response connections and provide the message server-to-client without polling.

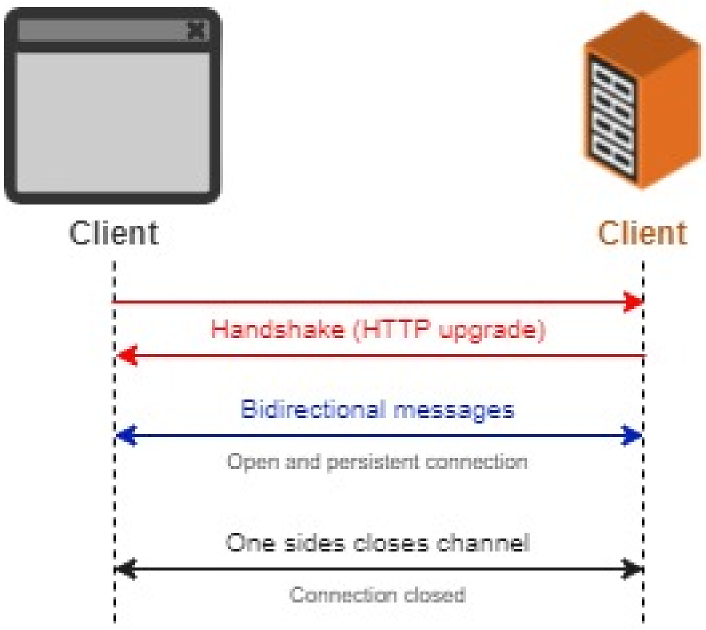


Figure 3.16: Connection using WebSocket

Figure 3.16 depicts the handshake and connection process for WebSocket. The handshake begins with an HTTP request/response, which allows servers to handle HTTP and WebSocket connections on the same port. Once the connection is established, communication shifts to a bidirectional binary mode that is not compliant with the HTTP protocol.

3.5 Logical System Architecture

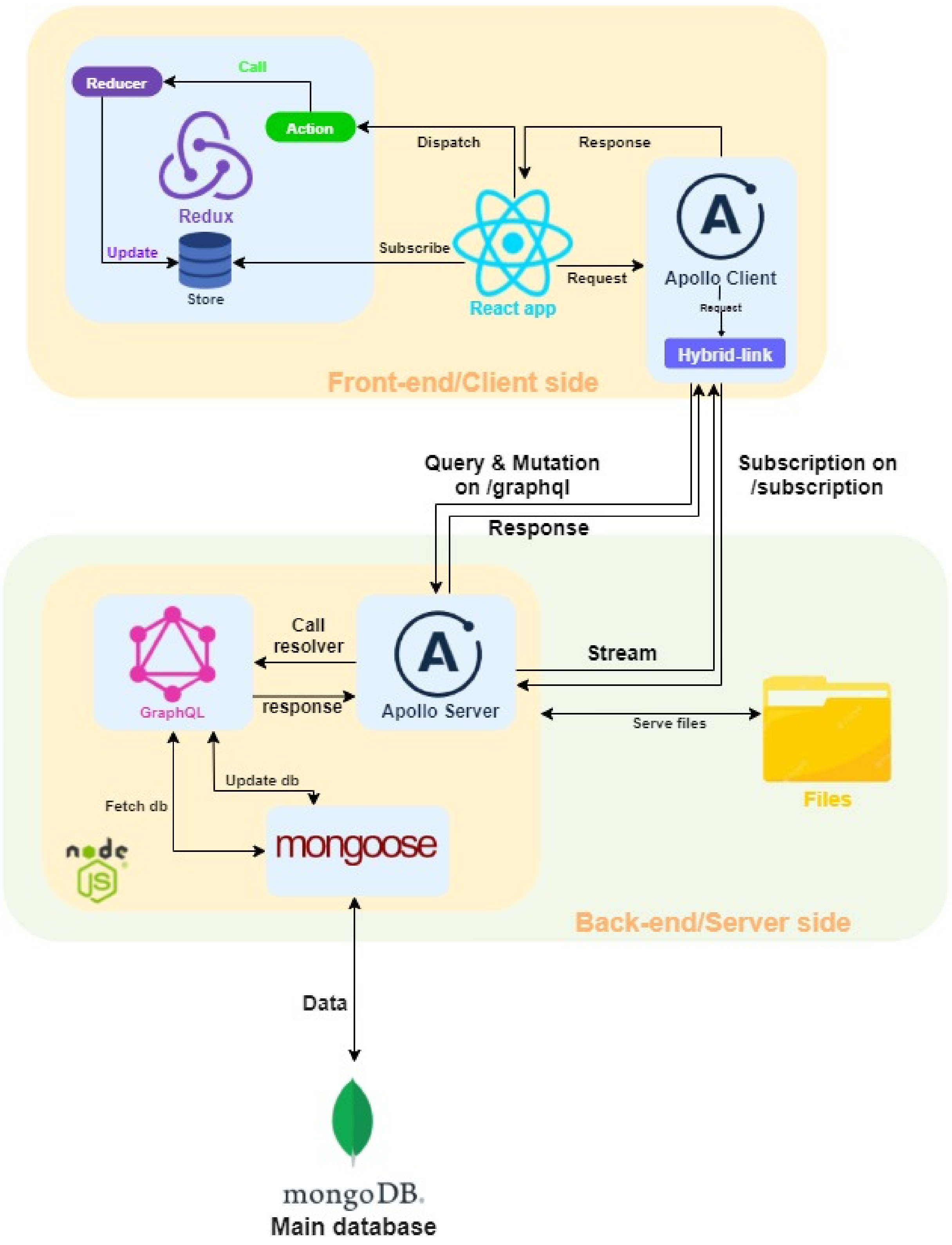


Figure 3.17: Logical System Architecture

Each component of the logical system architecture is described in depth here.

- **Front-end/Client side**: The front-end context will be powered by React and will have two distinct and significant scopes.
    - ⋆ **Redux**: The Redux scope will handle the front-end context's state management; it will contain a single store as the single source of truth, and it will offer actions that will trigger reducers to update the state.
    - ⋆ **Apollo Client**: The Apollo Client scope will act as a gateway, sending and receiving GraphQL queries and answers. The **Hybrid-link** identifies the request provided to the Apollo Client as an HTTP request or a WS (WebSocket) request before passing it to the appropriate endpoint; /graphql or /subscription.
- **Back-end/Server side**: The back-end context will be powered by the NodeJs runtime and will contain three distinct and significant scopes.
    - ⋆ **Apollo Server**: The Apollo server scope will act as a gateway, taking multiple types of queries (HTTP requests and WS requests) and providing basic replies for HTTP requests and a stream for WS requests.
    - ⋆ **GraphQL**: The GraphQL scope will serve as the orchestrer, defining the resolvers that the Apollo server will be able to call.
    - ⋆ **Mongoose**: The Mongoose scope will serve as an intermediate for the MongoDB database and the GraphQL. It will make its methods available to the resolvers.

  The back-end will be able to host and serve files.
- **MongoDB database**: It will include the database itself as well as the drivers required to administer the database.

Conclusion

We spoke a lot about technical aspects and designs in this chapter, and we detailed the logical architecture of our system as well as the physical architecture that we had to utilize in depth. We discussed the technologies utilized throughout this project previously, and we went over them in depth and compared them to similar ones so we could back up our decisions. In the following chapter, we begin the development process by establishing the project and database.

Chapter 4

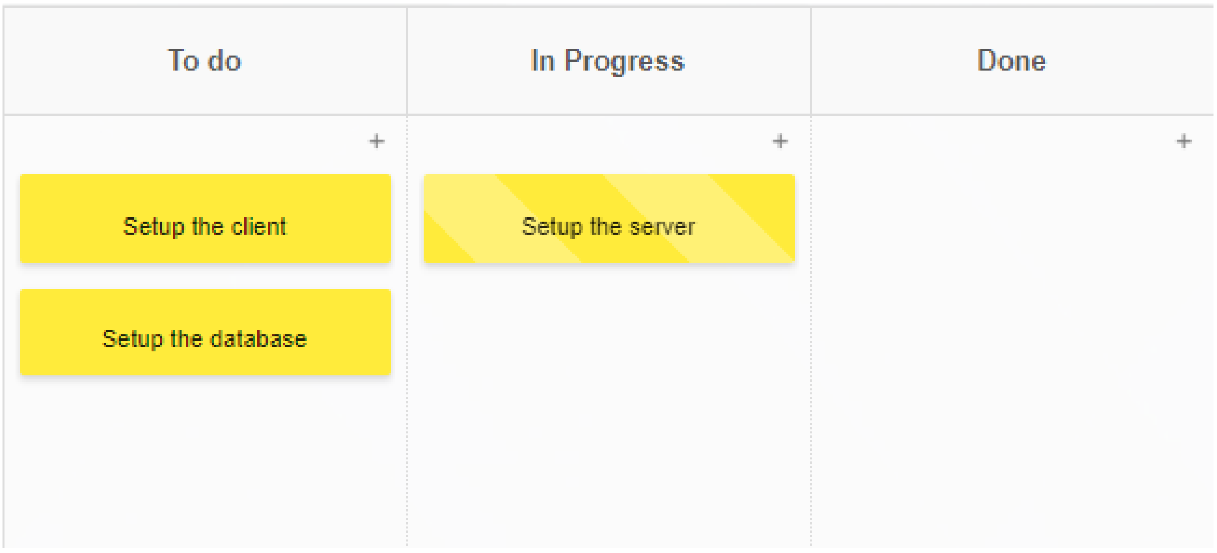
Iteration 1: Project and Database Setup

Introduction

After doing a requirement analysis, determining the logical and physical architectures, and selecting technologies, we are ready to begin the development process. Before we begin developing the platform features, we must first set up the project's various components and connect them so that everything works and communicates. Therefore, in this chapter, we'll go over the requirements and implementation of each project component, from the client to the database, and then to the server configuration.

4.1 Backlog

The figure 4.1 depicts the status of our Backlog throughout the first iteration.



To do	In Progress	Done
Setup the client	Setup the server	
Setup the database		

Figure 4.1: Iteration 1 Backlog

4.2 Client setup

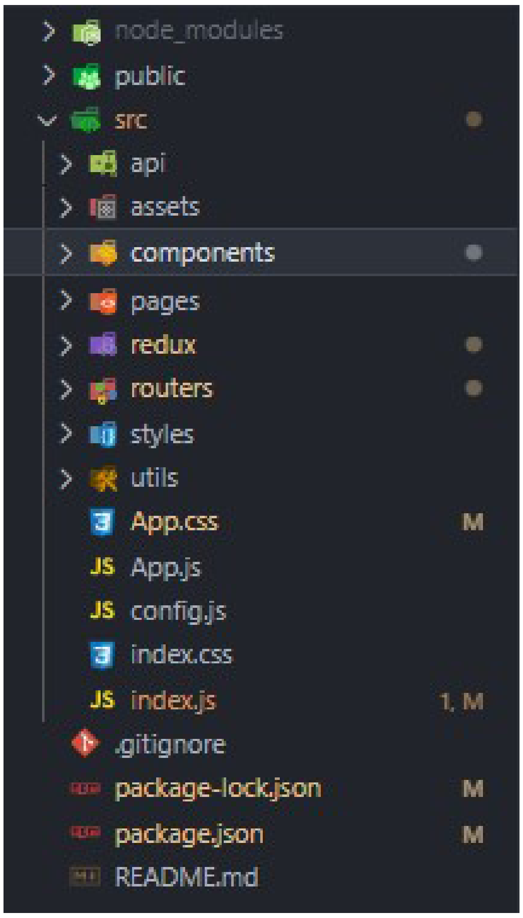
In this part, we will provide the requirements that will describe the objectives that we want to achieve throughout the client setup, as well as how we were able to complete these requirements.

4.2.1 Requirements

We want a fully setup Client from the start, thus we have defined a list of requirements and they are as follows:

- Initialize the React application, as it is the heart of the Client.
- Add Redux for state management so that we can interact with and update the store from the very first feature we create.
- Install and setup the Apollo Client so that we can communicate with our graphql backend by sending queries and mutations.
- Add a UI framework so we do not have to create basic UI components from scratch and therefore can speed up development.

4.2.2 Implementation



```
> node_modules
> public
v src
  > api
  > assets
  > components
  > pages
  > redux
  > routers
  > styles
  > utils
    App.css              M
    App.js
    config.js
    index.css
    index.js             1, M
  .gitignore
  package-lock.json      M
  package.json           M
  README.md
```

Figure 4.2: Client file structure

Figure 4.2 depicts the end result of establishing the React project and installing the necessary packages to get the Redux store and the Apollo Client up and running. Material-UI was chosen as the UI framework. The following is the file structure:

- **api**: Will hold the definition of GraphQL queries.
- **assets**: Will include all of the assets required locally (images, icons, fonts..).
- **components**: Will contain all of the React components, which will be arranged into folders.
- **redux**: Will contain the store, the actions and the reducers.
- **routers**: Will include both the generic and nested routes definitions.
- **index.js**: Is the React application's starting point, where we initialize and configure the Apollo Client. And connecting the root React component to the Redux store.

4.3 Server setup

In this section, we will offer the requirements that outline the goals we want to achieve throughout the server configuration, as well as how we were able to complete these needs.

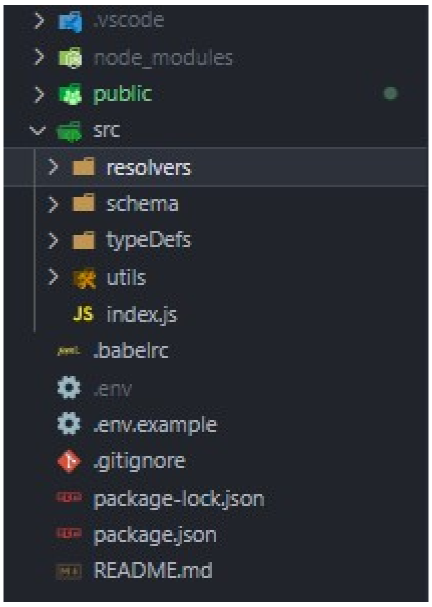
4.3.1 Requirements

Setting up the server so that it is ready to go and begin creating our API, we have established a list of prerequisites:

- Initialize the NodeJs server, as it is the heart of the backend.
- Add and configure the Apollo Server so that we can begin developing our queries and mutations to reply to the Client request.
- Set up and configure the database connection so that it will be ready when the database server is deployed.

4.3.2 Implementation

Figure 4.4 illustrates the end result of creating the Node.js project and installing the required packages, as well as setting up the Apollo Server, configuring it as a middleware, and adding it to the Express application. Configure the server to serve the static files that we will receive and save from the Client. As well as preparing the database connection configuration.



```
> .vscode
> node_modules
> public
v src
  > resolvers
  > schema
  > typeDefs
  > utils
    index.js
  .babelrc
  .env
  .env.example
  .gitignore
  package-lock.json
  package.json
  README.md
```

Figure 4.3: Server file structure

- **public**: Will store any static files received from the client.
- **resolvers**: Where all GraphQL resolvers will be defined.
- **schema**: All of our Mongoose schemas and model definitions will be stored here.
- **typeDefs**: Will include all of the GraphQL type definitions.
- **index.js**: It serves as the application's primary entry point. This is where we start the Express application and connect it to the GraphQL middleware. Create a public endpoint where we may deliver static files.

4.4 Database setup

In this part, we will provide the criteria that describe the goals we want to achieve throughout the database configuration, as well as how we were able to complete these requirements.

4.4.1 Requirements

Setting up a database is typically rather simple:

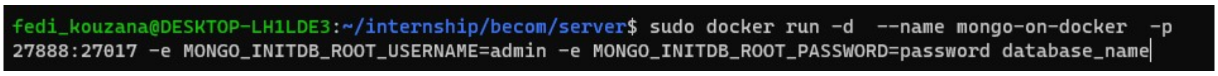
- Installing the MongoDB database, so that we have a management system for the database and the set of tools that come with it.
- Making a database user, so that we may manage and safeguard the database's access control.
- Making the database, so that we can begin creating documents and saving data in them.

4.4.2 Implementation

To avoid installing the entire MongoDB database locally and managing the dependencies that come with it, we resorted to utilizing a **Docker container** in this situation.

- **Containerization** is the process of packaging software code with just the operating system (OS) libraries and dependencies needed to run the code in order to generate a single lightweight executable, known as a container, that operates consistently on any infrastructure. Virtual machines are less portable and use less resources (VMs).
- **Docker** is a containerization platform that is open source. It allows developers to bundle programs into containers, which are standardized executable components that combine application source code with the operating system (OS) libraries and dependencies necessary to run that code in any environment. Containers make it easier to provide distributed programs.

We will use the Docker official Image for Mongo. After downloading it, we will run the following command to launch a Docker container constructed with the Mongo Image, create the database, and create a user to that database.



```
fedi_kouzana@DESKTOP-LH1LDE3:~/internship/becom/server$ sudo docker run -d  --name mongo-on-docker  -p
27888:27017 -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=password database_name
```

Figure 4.4: Command to initialize MongoDB container

Conclusion

Throughout this chapter, we went through the process of configuring all of the project's components, beginning with the client and ending with the database. We are now ready to begin creating the platform features after connecting them and testing that they communicate successfully. In the following chapter, we will begin a new iteration and concentrate on the authentication system.

Chapter 5

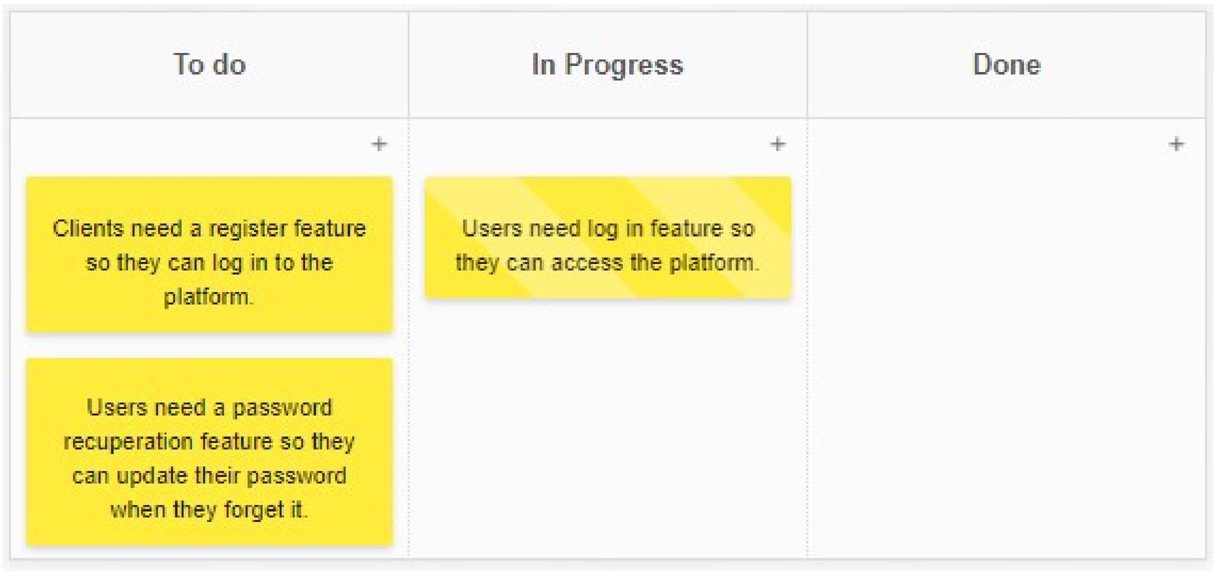
Iteration 2: Authentication

Introduction

In the previous chapter, we explained the procedures we took to set up the development environment, configure the project, and launch it. The foundation has been constructed and is ready for us to begin constructing the features. The authentication feature will be designed and implemented in this iteration.

5.1 Backlog

The figure 5.1 depicts the status of our Backlog throughout the second iteration.



To do	In Progress	Done
Clients need a register feature so they can log in to the platform.	Users need log in feature so they can access the platform.	
Users need a password recuperation feature so they can update their password when they forget it.		

Figure 5.1: Iteration 2 backlog

5.2 Design phase

During the design phase, we had to convert our business needs for this feature, which we defined in the second chapter requirement definition, to feature design and user interfaces.

In this section, we will exhibit several sorts of UML diagrams that will highlight our system design, as well as mockups as a UI prototype.

5.2.1 Class diagram

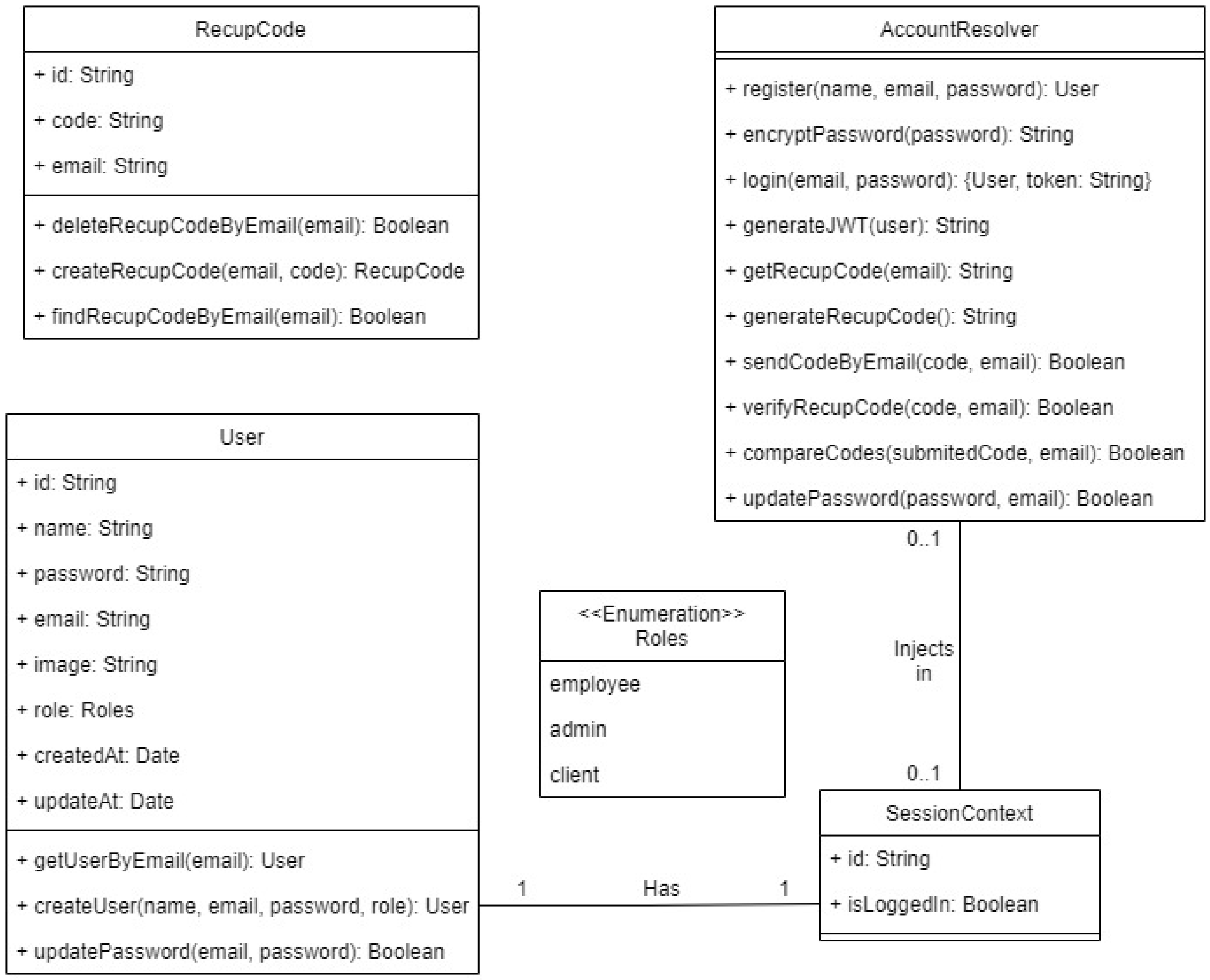

RecupCode
+ id: String
+ code: String
+ email: String
+ deleteRecupCodeByEmail(email): Boolean
+ createRecupCode(email, code): RecupCode
+ findRecupCodeByEmail(email): Boolean
AccountResolver
+ register(name, email, password): User
+ encryptPassword(password): String
+ login(email, password): {User, token: String}
+ generateJWT(user): String
+ getRecupCode(email): String
+ generateRecupCode(): String
+ sendCodeByEmail(code, email): Boolean
+ verifyRecupCode(code, email): Boolean
+ compareCodes(submitedCode, email): Boolean
+ updatePassword(password, email): Boolean
0..1
Injects in
0..1
User
+ id: String
+ name: String
+ password: String
+ email: String
+ image: String
+ role: Roles
+ createdAt: Date
+ updateAt: Date
+ getUserByEmail(email): User
+ createUser(name, email, password, role): User
+ updatePassword(email, password): Boolean
<<Enumeration>> Roles
employee
admin
client
1
Has
1
SessionContext
+ id: String
+ isLoggedIn: Boolean

Figure 5.2: Authentication class diagram

Table 5.1: Descriptive table of authentication class diagram classes

Class	Description
User	This class represents the user object and will manage and interact with the Users database
RecupCode	This class represents the recupCode object and will manage and interact with the RecupCode database
SessionContext	This class will contain the session context for a given user and will be injected into the AccountResolver so that it can access its attributes
AccountResolver	This class will handle all of the client's account requests and will provide the methods required to fulfill these requests.

5.2.2 Activity diagram

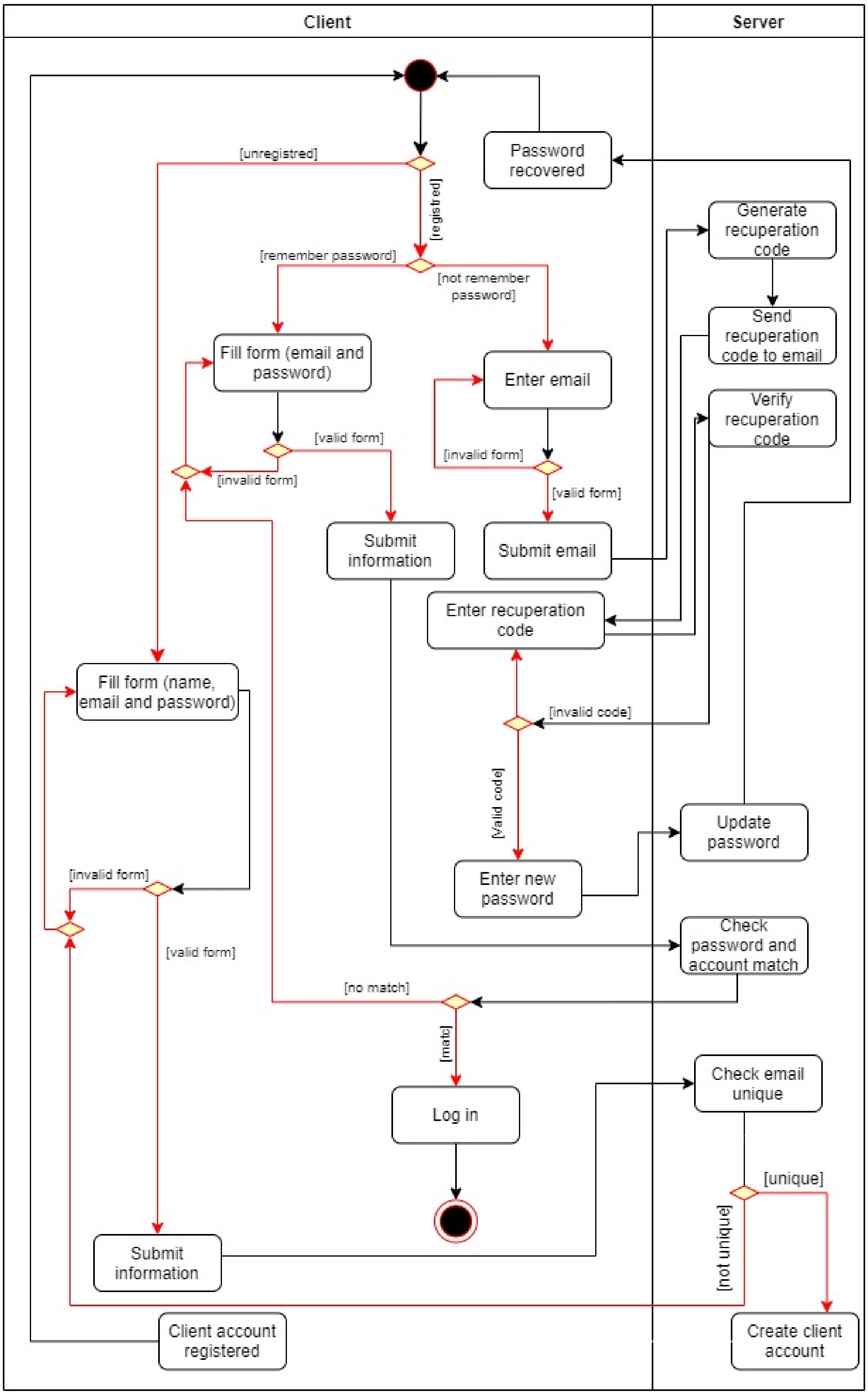


Figure 5.3: Authentication activity diagram

The activity diagram in figure 5.3 depicts the process of transitioning from not logged in to logged in while taking into account numerous circumstances, such as the client not having an account or having forgotten his account's password.

5.2.3 Sequence diagram

In this part, we will go through three scenarios in depth. A sequence diagram will be used to depict each situation.

5.2.3.1 Register sequence diagram

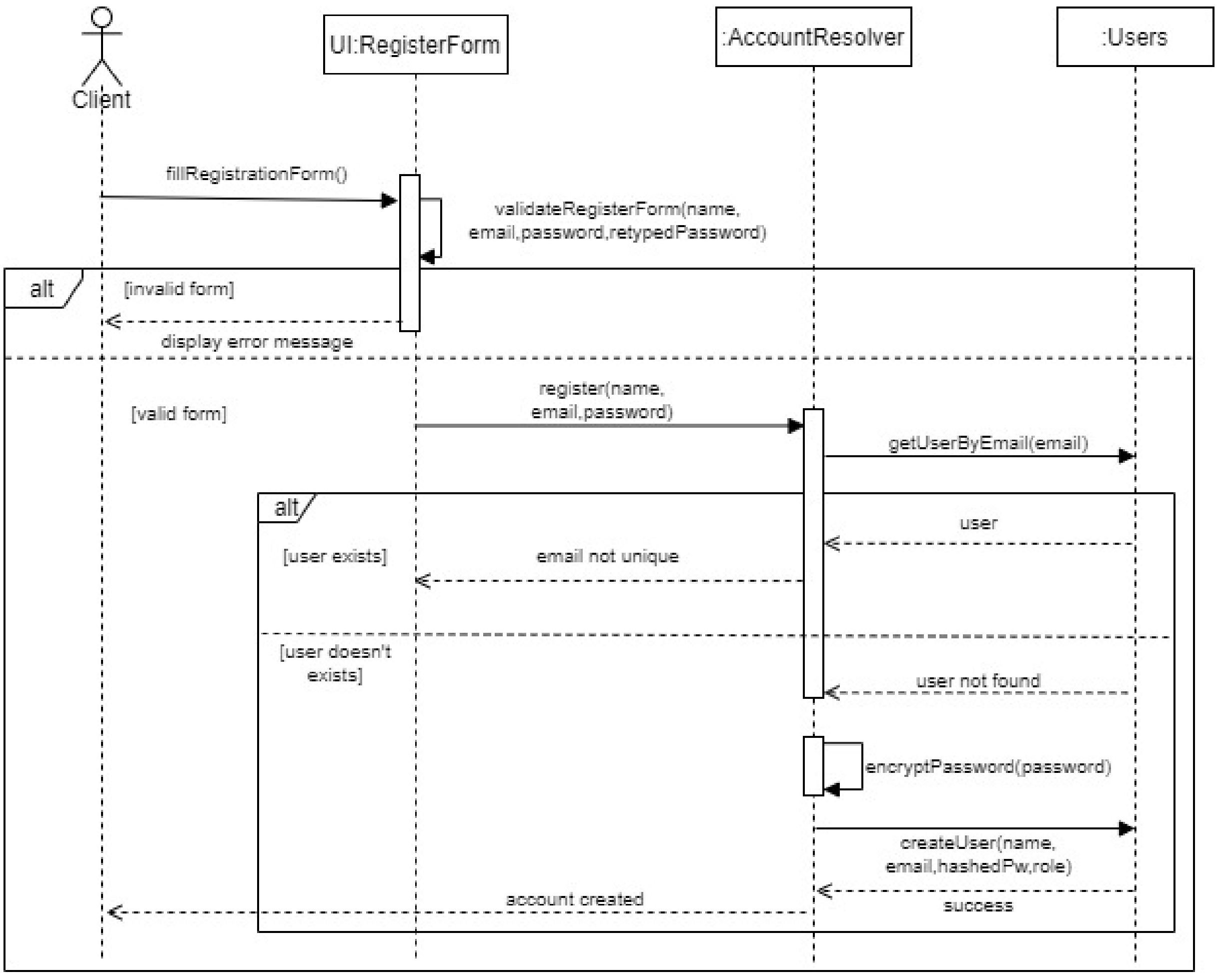


Figure 5.4: Register sequence diagram

The sequence diagram 5.4 describe the process of registering a client account.

1. The client fills out the registration form fields, while the form validates the client's input to ensure that it conforms to specific expected formats.
2. Submits the form values to the system only if it is valid; otherwise, an error notice is displayed.

3. The system processes the date received (hashes the password and checks to see whether the email address has not previously been used) and then creates a user and stores it in the database.

5.2.3.2 Login sequence diagram

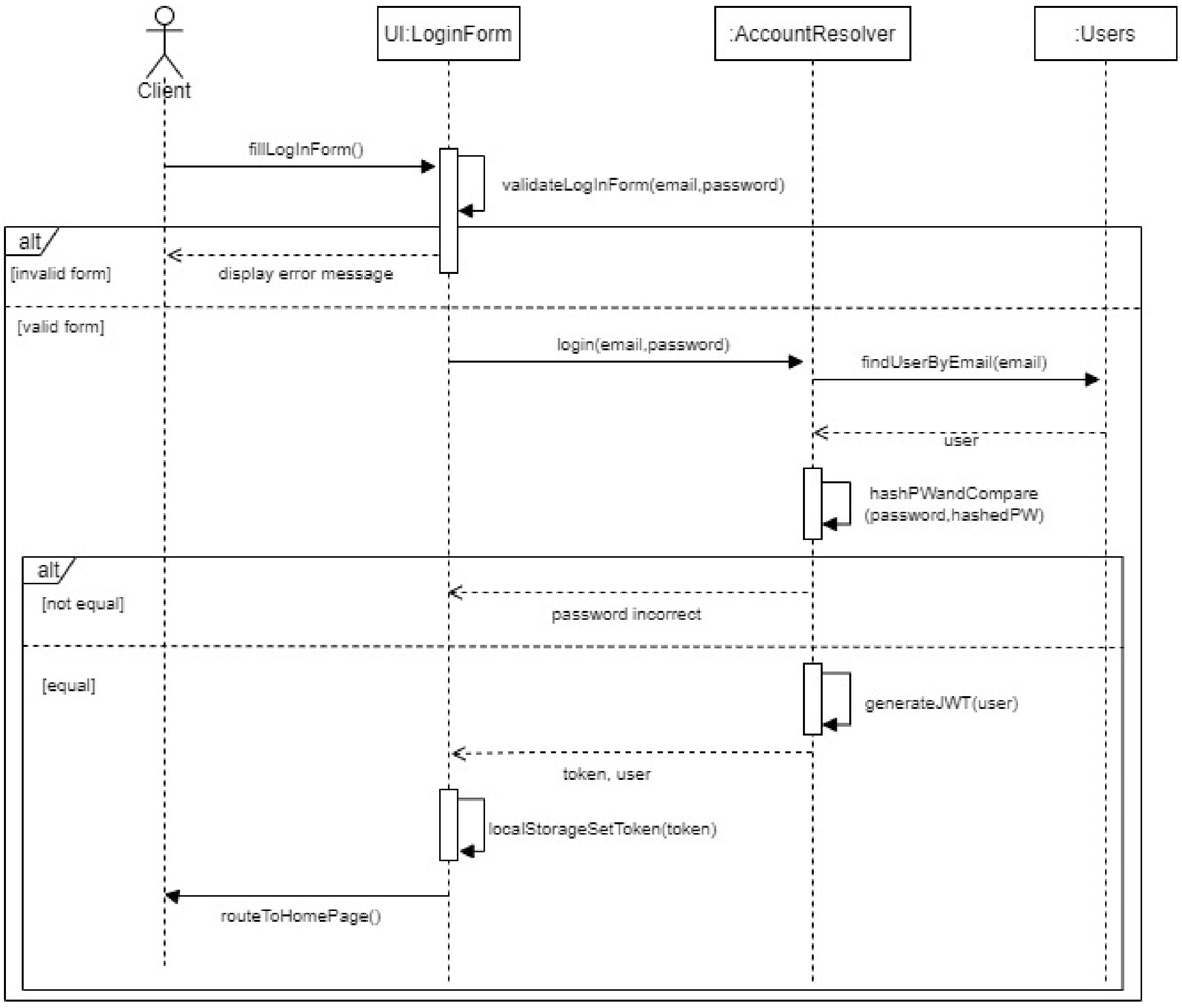


Figure 5.5: Login sequence diagram

The sequence diagram 5.5 describe the process of logging in into an account.

1. The user fills out the login form fields, while the form validates the client's input to ensure that it conforms to specific expected formats.
2. Submits the form values to the system only if it is valid; otherwise, an error notice is displayed.
3. When the system receives the data, it validates it and generates a JWT token if it is legitimate.
4. If the data given is legitimate, the token and user information are returned to the user; otherwise, an error message is displayed.
5. The JWT token is saved in the browser's LocalStorage.

5.2.3.3 Password recuperation sequence diagram

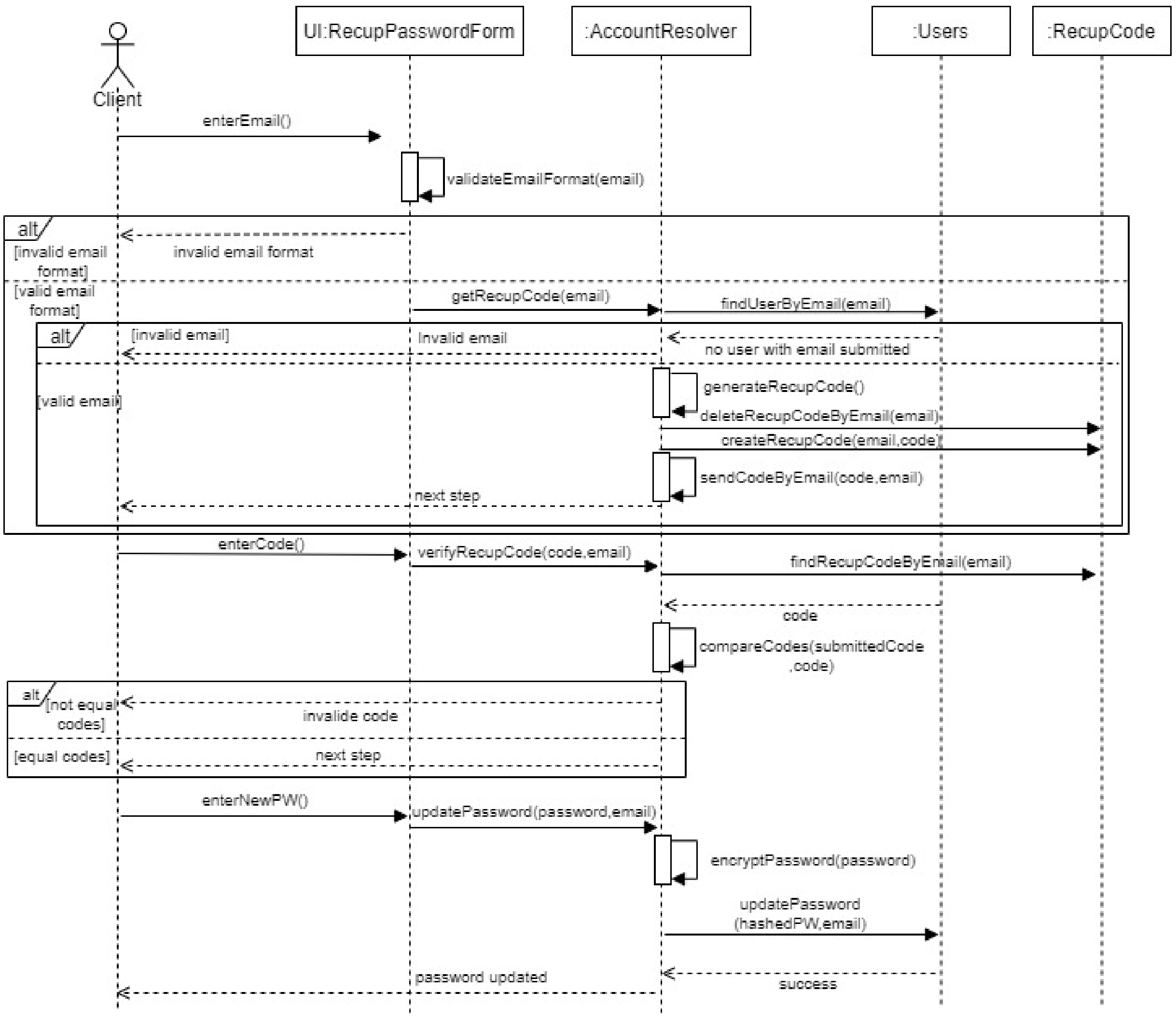


Figure 5.6: Password recuperation sequence diagram

The sequence diagram 5.6 describe the process of password recovery.

1. The user provides his or her account email address, which is subsequently validated and submitted.
2. The system receives the email and then checks to see if an account with that email already exists; if so, it creates a recovery code, stores it in the database, and then sends it by email to the email address supplied.
3. The user provides the recovery code and submit it to the system.
4. The system will then deliver a message indicating whether or not the recovery code is legitimate.
5. When the recovery code is genuine, the user can retype a new password and transmit it to the system.
6. The system will then update and save the new password.

5.3 Implementation

During the implementation phase, we had to rely on specific notions to fulfill our objectives. JWT is the token we utilized to securely send data from the client to the server side and complete the authentication procedure. We also utilized hashing to safeguard our consumers' passwords.

5.3.1 JWT

During the authentication, we used JWT to ensure that the user was authenticated and we using to secure some private GraphQL queries that are only available for logged in users.

5.3.1.1 What's JWT

We utilized JWT during authentication to verify that the user was authenticated, and we used it to protect some private GraphQL queries that are only visible to logged in users.

5.3.1.2 JWT composition

JSON Web Tokens are composed of three parts separated by dots (.), which are as follows:

- **Header**: typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.
- **Payload**: It includes the claims. They consist of assertions about an entity (usually the user) as well as supplementary data.
- **Signature**: is used to ensure that the message has not been altered along the route, and in the case of tokens signed with a private key, it may also ensure that the sender of the JWT is who they claim to be.

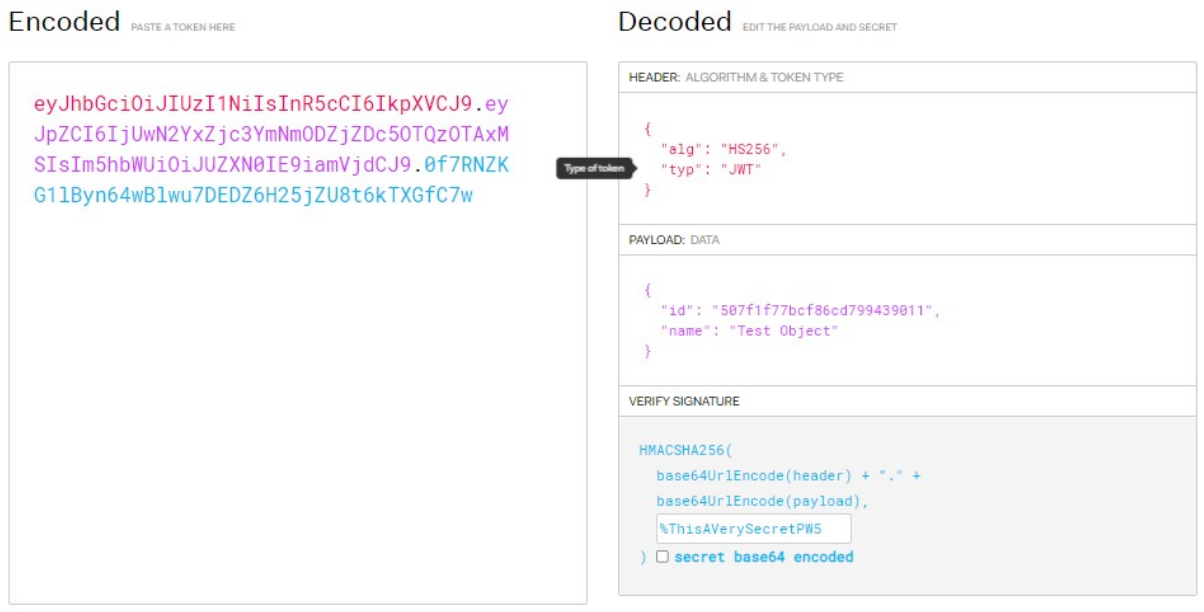
5.3.2 Password hashing

Storing plain passwords in the database is a major security flaw if our database is hacked and the attacker has access to it. He has quick access to all of the accounts on the system since he can view their passwords.

5.3.2.1 Hashing

We are left with two options: hash the password or encrypt it and save it in the database so it is no longer in plain text. Encryption necessitates the usage of an encryption key and a decryption key, both of which must be stored someplace on the server, which does not make this solution the best option.

A hashing algorithm is a mathematical technique that takes a specified input of data and produces a fixed-length result known as a hash value. The hash value serves as a condensed version of the original value.



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JpZCI6IjUwN2YxZjc3YmNmODZjZDc5OTQzOTAxM
SIsIm5hbWUiOiJUZXN0IE9iamVjdCJ9.0f7RNZK
G1lByn64wBlwu7DEDZ6H25jZU8t6kTXGfC7w
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "id": "507f1f77bcf86cd799439011",
  "name": "Test Object"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  %ThisAVerySecretPW5
) ☐ secret base64 encoded
```

Figure 5.7: JWT encoding example

5.3.2.2 Collision

Hashing algorithms are one-way functions. Hashing the same keys will result in the same hash, which we term Collision. We will use this phenomenon to determine whether or not the password given by the user is correct by hashing it and comparing it to the previously saved hashed password.

5.3.3 Interfaces

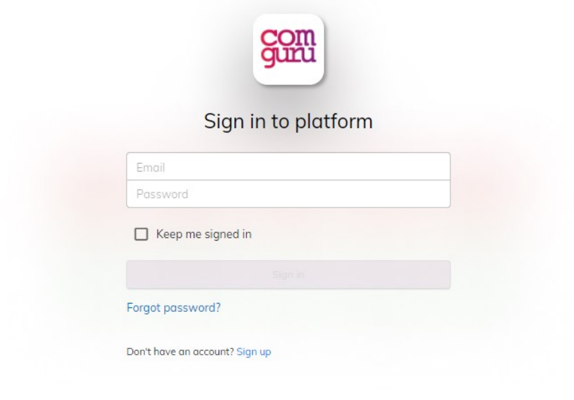


Figure 5.8: Log in interface

Figure 5.8 demonstrates our application's log in form.

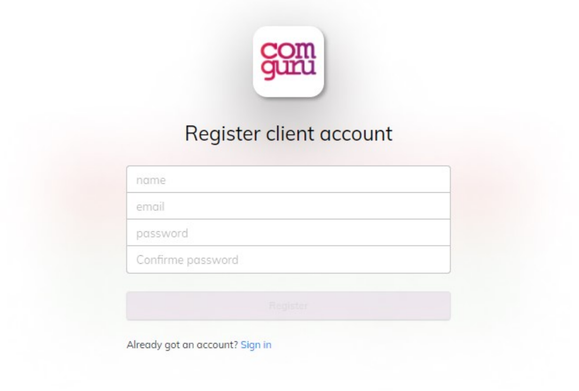


Figure 5.9: Client register interface

Figure 5.9 demonstrate our application's register form.

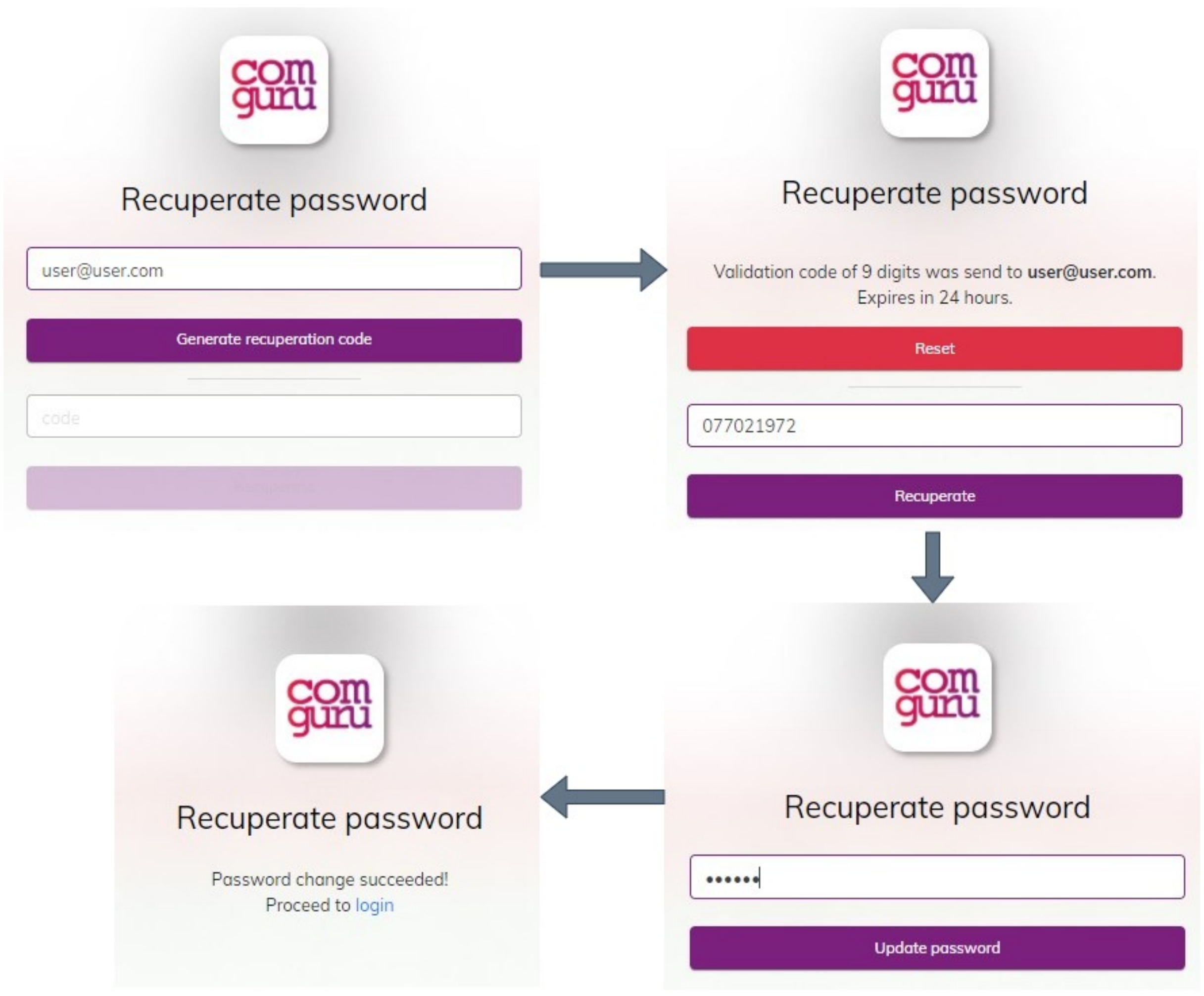
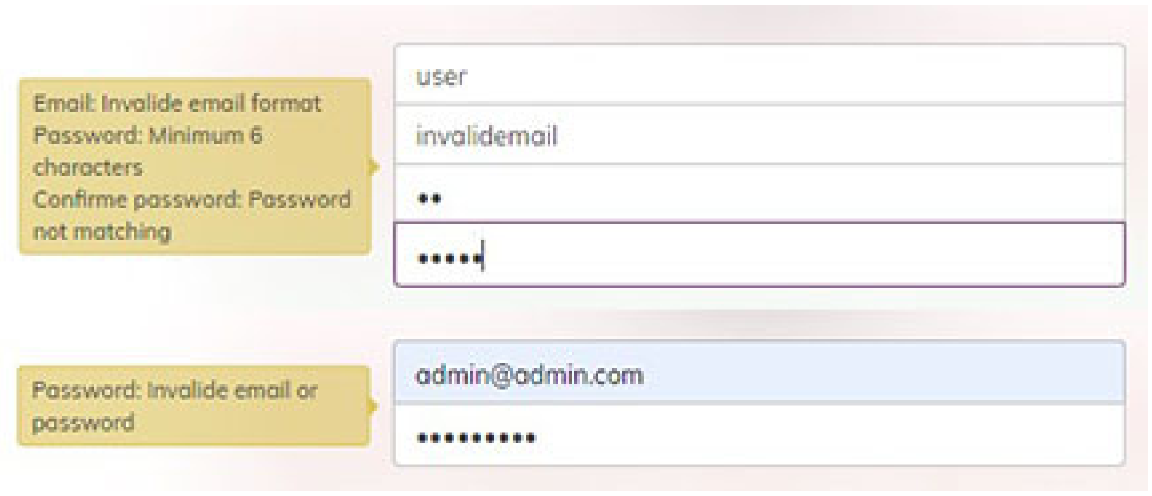


Figure 5.10: Password recovery interfaces step by step

Figure 5.10 displays the interfaces for our application's password recovery procedures.


Email: Invalide email format
Password: Minimum 6 characters
Confirme password: Password not matching

user
invalidemail
••
•••••

Password: Invalide email or password

admin@admin.com
•••••••••

Figure 5.11: Invalid form & invalid log in information errors

Figure 5.11 displays the error messages that the register and log in form displays when an error occurs.

Conclusion

In this chapter, we looked through all of the processes that we encountered when developing the authentication functionality, from concept to implementation. Now that it is finished and working, we will go on to developing one of our application's main features, the project management tool.

Chapter 6

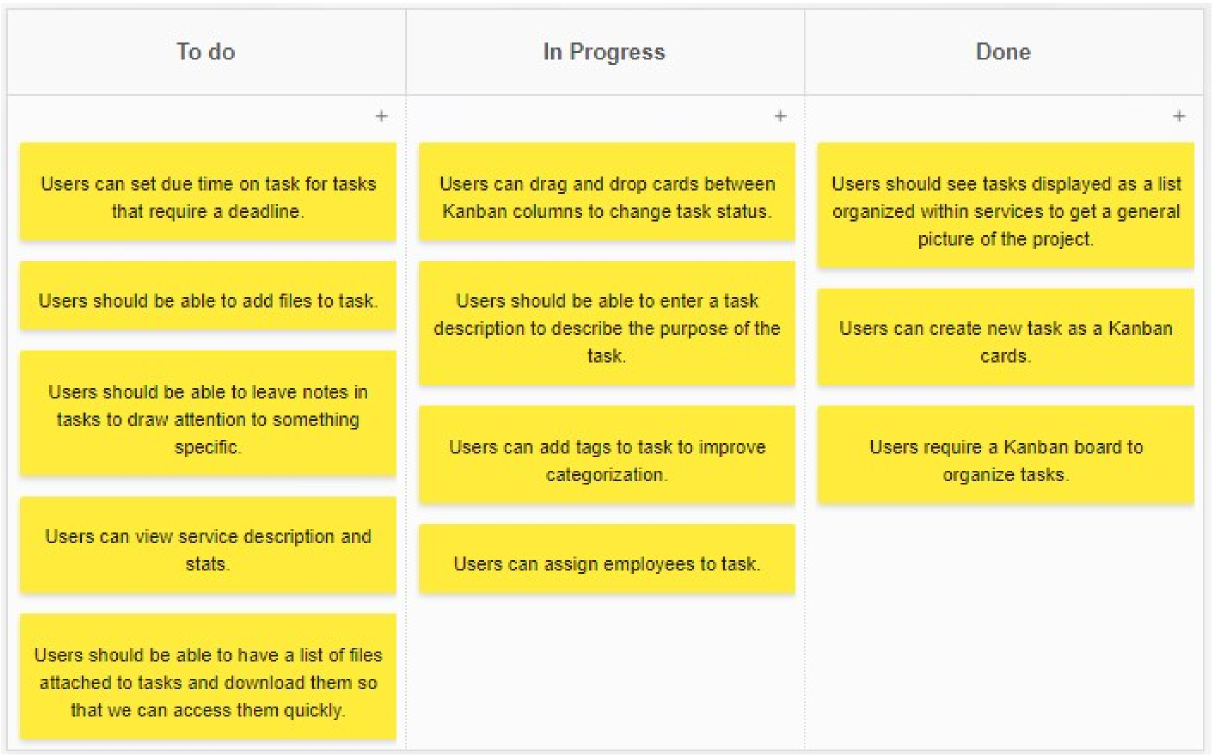
Iteration 3: Project manager

Introduction

In the previous chapters, we set up the project and built a working authentication mechanism on top of it, allowing us to manage access to our platform; now we move on to developing the major features for our application. During this iteration, we will discuss the project management features and detail the methods we took to build this capability.

6.1 Backlog

The figure 6.1 depicts the status of our Backlog throughout the third iteration.



To do	In Progress	Done
Users can set due time on task for tasks that require a deadline.	Users can drag and drop cards between Kanban columns to change task status.	Users should see tasks displayed as a list organized within services to get a general picture of the project.
Users should be able to add files to task.	Users should be able to enter a task description to describe the purpose of the task.	Users can create new task as a Kanban cards.
Users should be able to leave notes in tasks to draw attention to something specific.	Users can add tags to task to improve categorization.	Users require a Kanban board to organize tasks.
Users can view service description and stats.	Users can assign employees to task.	
Users should be able to have a list of files attached to tasks and download them so that we can access them quickly.		

Figure 6.1: Iteration 3 backlog

6.2 Design phase

6.2.1 Class Diagram

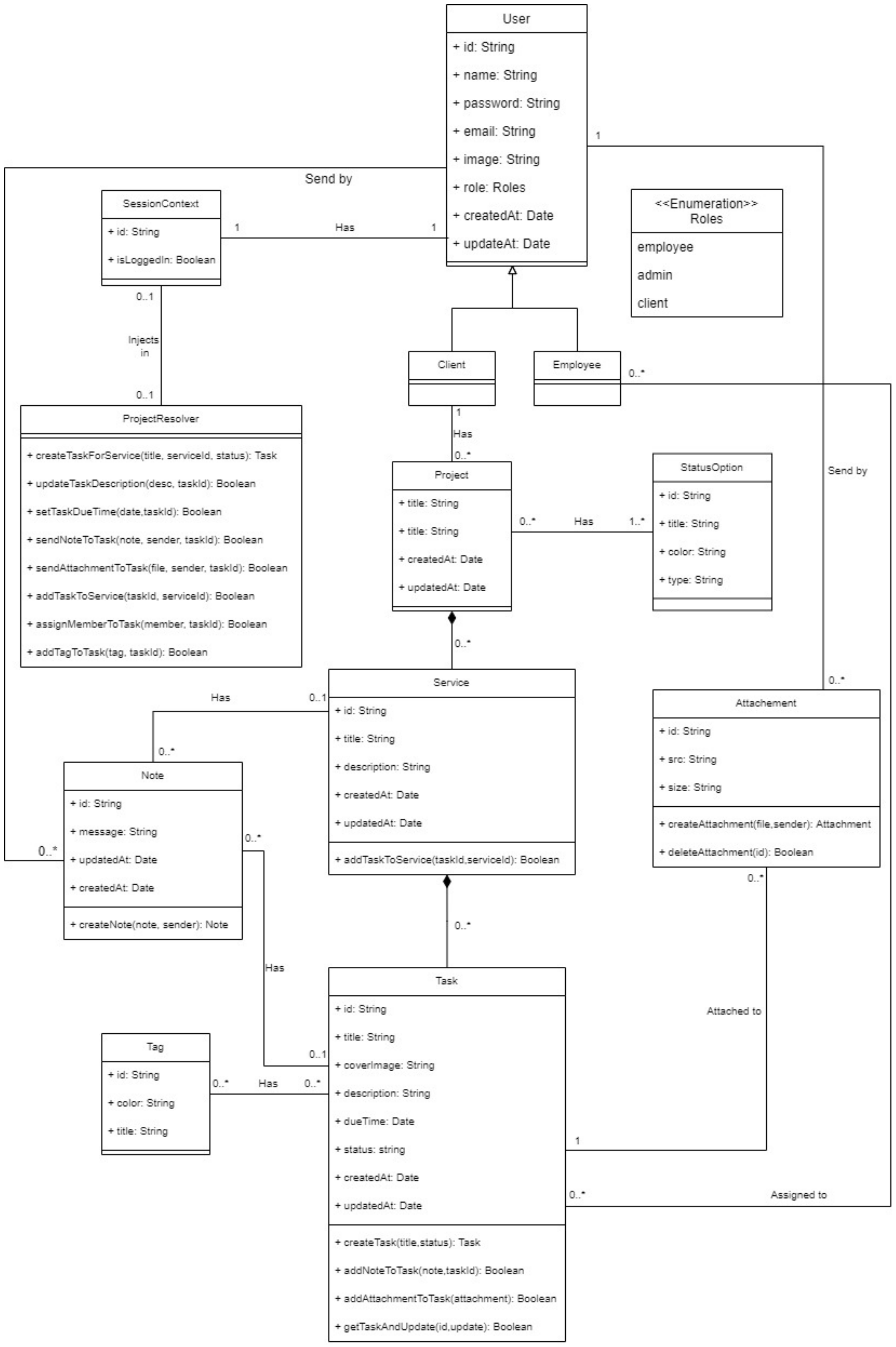


Figure 6.2: Project manager class diagram

Table 6.1: Descriptive table of project manager class diagram classes

Class	Description
User	This class represents the user object and will manage and interact with the Users database.
Client	This class is a subclass of the class User and it represents users with role "client".
Employee	This class is a subclass of the class User and it represents users with role "employee".
SessionContext	This class will contain the session context for a given user and will be injected into the ProjectResolver so that it can access its attributes.
ProjectResolver	This class will handle all of the client's project manager requests and will provide the methods required to fulfill these requests.
Project	It is a bundle of services devoted to a client and contains the project's business logic to handle it.
Service	It is a bundle of tasks and contains the service's business logic to handle it.
Task	This class provides a collection of attributes that will assist in tracking progress and completing the job and the business logic for managing it.
StatusOption	This class contains the definition of status, which will be handed on to the project for the task's status choices and the business logic for managing it.
Attachment	This class store the definition a saved file to a task and the functions for handling it.
Note	This class contains the definition of notes sent to tasks or services, as well as the functions for processing them.
Tag	This class will hold the definition of tags that will be added to task to categorize and the business logic for managing it.

6.2.2 Navigation diagram

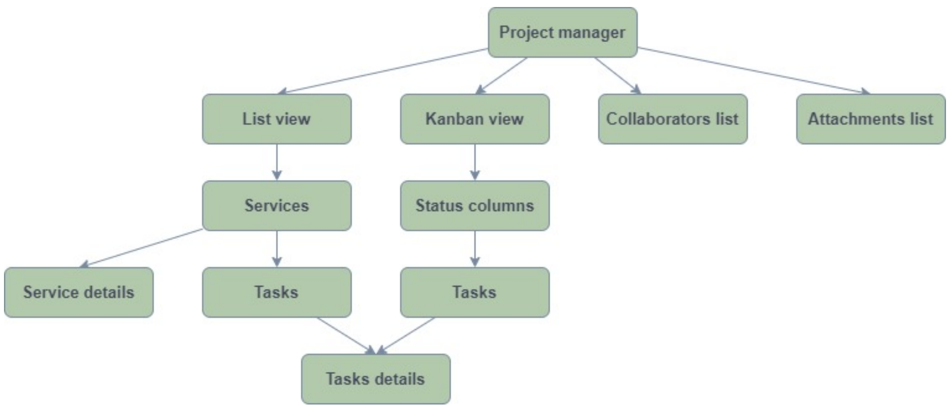


Figure 6.3: Project manager navigation diagram

Figure 6.3 illustrates the different tabs for the project management section, as well as their composition.

6.2.3 Sequence Diagram

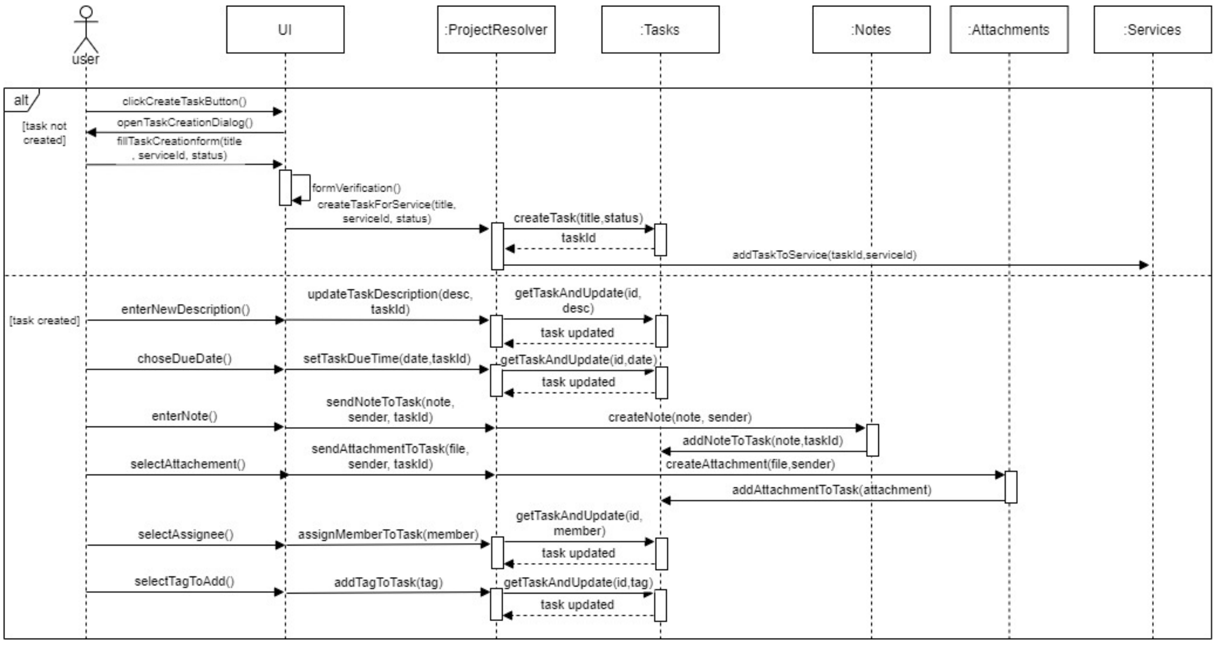

user
UI
:ProjectResolver
:Tasks
:Notes
:Attachments
:Services
alt
[task not created]
clickCreateTaskButton()
openTaskCreationDialog()
fillTaskCreationform(title , serviceId, status)
formVerification()
createTaskForService(title, serviceId, status)
createTask(title,status)
taskId
addTaskToService(taskId,serviceId)
[task created]
enterNewDescription()
updateTaskDescription(desc, taskId)
getTaskAndUpdate(id, desc)
task updated
choseDueDate()
setTaskDueTime(date,taskId)
getTaskAndUpdate(id,date)
task updated
enterNote()
sendNoteToTask(note, sender, taskId)
createNote(note, sender)
addNoteToTask(note,taskId)
selectAttachement()
sendAttachmentToTask(file, sender, taskId)
createAttachment(file,sender)
addAttachmentToTask(attachment)
selectAssignee()
assignMemberToTask(member)
getTaskAndUpdate(id, member)
task updated
selectTagToAdd()
addTagToTask(tag)
getTaskAndUpdate(id,tag)
task updated

Figure 6.4: Task management sequence diagram

The sequence diagram of figure 6.4 describe the process of managing a task. If task is not created :

1. The user click on create task button and the creation form appears.
2. Fill out the form fields, and the UI will validate the entries.
3. The system receives the data, generates the task, and stores it before connecting it to its service.

If task is created :

1. User defines a description for task and sends it to the system, which receives it, retrieves the task, and updates its description.
2. User selects deadline time for a task, the system receive it, retrieve that specific task and updates its due date.
3. The user enters a note that wants to leave on this task and sends it to the system. When the latter gets it, it creates the note object, stores it, and then associates it with the task.
4. The user attaches a file to the task and submits it to the system. The system hosts it, generates an attachment object with the file's definition, and then saves it. Finally, it connects it to the task at hand.

5. The user chooses one employee to assign to the task, submits their credentials to the system, and the system appoints the person to the task.

6. The user selects a tag from the list and enters it into the system. The latter fetches the task and associates it with a tag.

6.2.4 Mockups

In this part, we'll show you some of the mockups we created for the project management functionality. They will serve as a guidepost during the development process, making interface design and implementation very simple. We kept the UX in mind when creating the mockups.

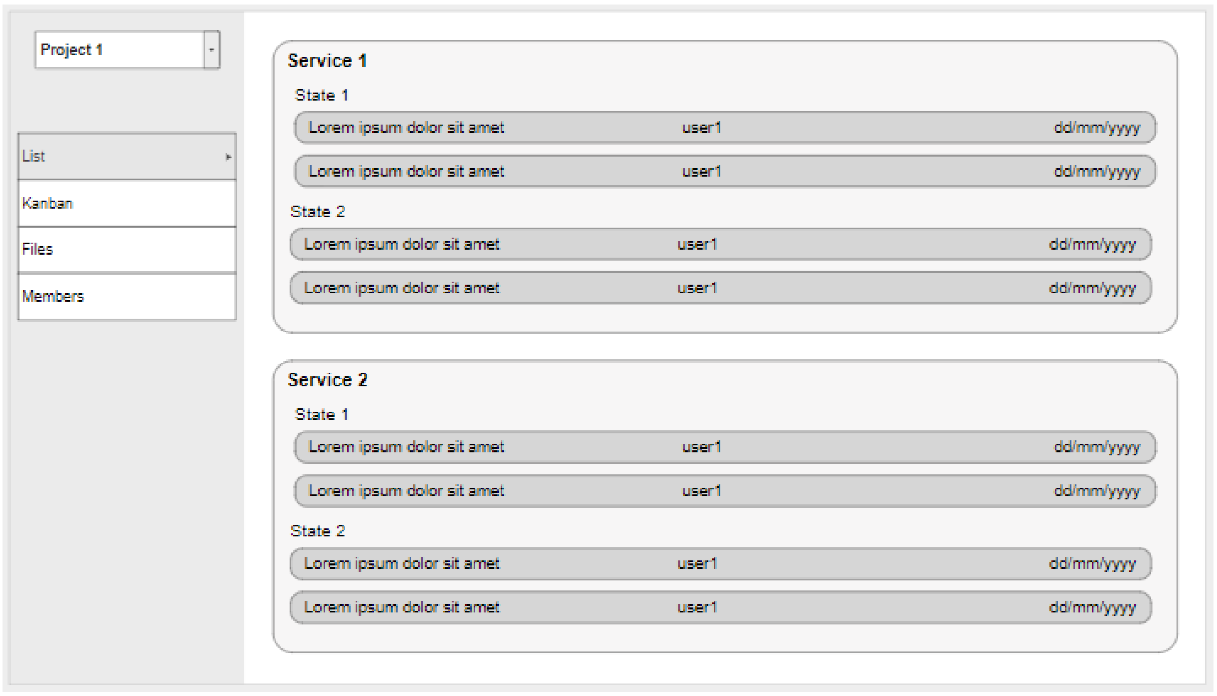

Project 1
List
Kanban
Files
Members
Service 1
State 1
Lorem ipsum dolor sit amet user1 dd/mm/yyyy
Lorem ipsum dolor sit amet user1 dd/mm/yyyy
State 2
Lorem ipsum dolor sit amet user1 dd/mm/yyyy
Lorem ipsum dolor sit amet user1 dd/mm/yyyy
Service 2
State 1
Lorem ipsum dolor sit amet user1 dd/mm/yyyy
Lorem ipsum dolor sit amet user1 dd/mm/yyyy
State 2
Lorem ipsum dolor sit amet user1 dd/mm/yyyy
Lorem ipsum dolor sit amet user1 dd/mm/yyyy

Figure 6.5: Tasks list mockup

In the figure 6.5 the overall layout of the project management function is shown. On the left side, we have the navigation section, which includes navigation between projects as well as navigation between project parts. On the other hand, we will show the duties of each service, organized by state.

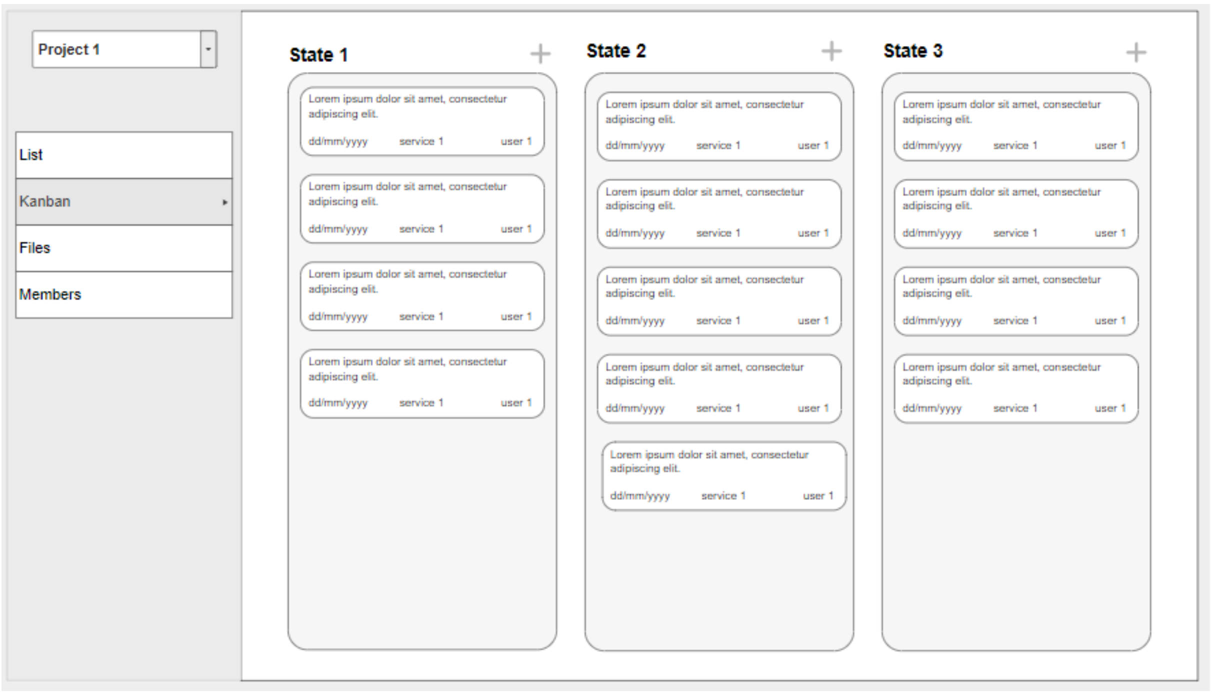
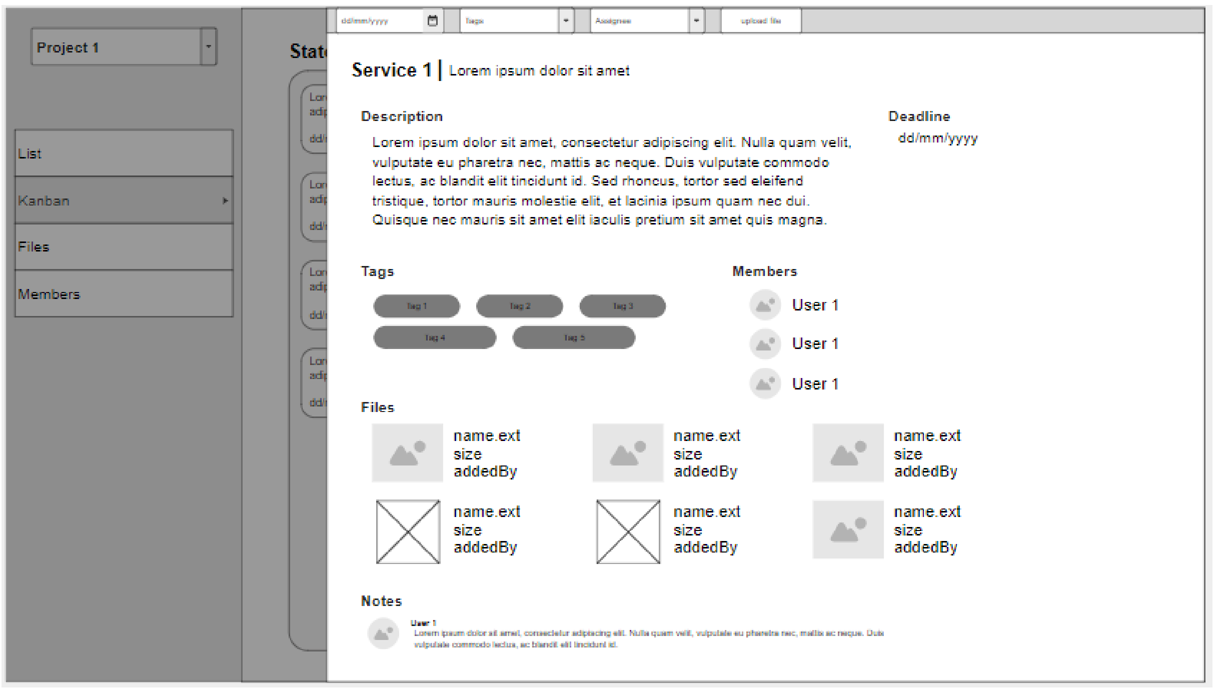


Figure 6.6: Kanban board mockup

On the left side of figure 6.6, we see the same navigation panel as in figure 6.5. On the right side, we'll have a Kanban board having columns of statuses that will contain the tasks as cards.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id.

Figure 6.7: Task detail mockup

Figure 6.7 shows a sliding from the right modal that displays all of the task information.

6.3 Implementation

Conclusion

Chapter 7

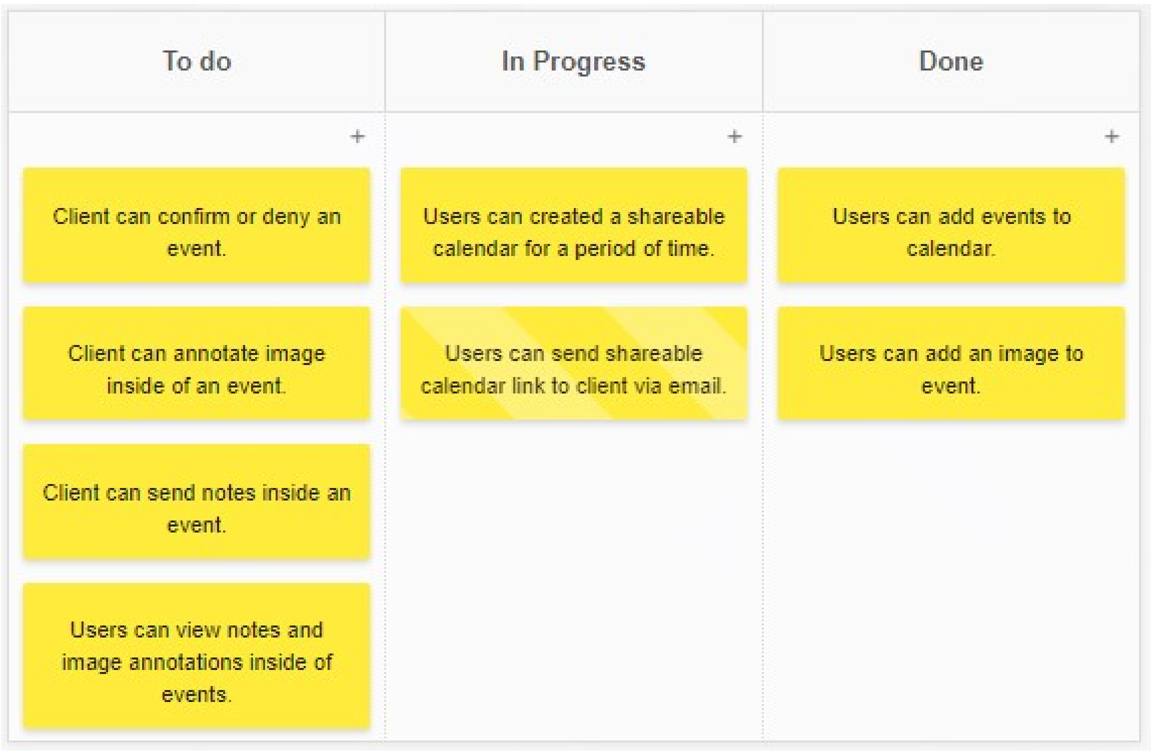
Iteration 4: Schedule & schedule sharing

Introduction

We started building the basic elements of our platform in the previous chapter and completing the project management tool, we moved on to the next critical component, which is the schedule and schedule sharing. We will discuss the design and implementation phases of this feature throughout this iteration.

7.1 Backlog

The figure 7.1 depicts the status of our Backlog throughout the fourth iteration.



To do	In Progress	Done
Client can confirm or deny an event.	Users can created a shareable calendar for a period of time.	Users can add events to calendar.
Client can annotate image inside of an event.	Users can send shareable calendar link to client via email.	Users can add an image to event.
Client can send notes inside an event.		
Users can view notes and image annotations inside of events.		

Figure 7.1: Iteration 4 backlog

7.2 Design phase

7.2.1 Class Diagram

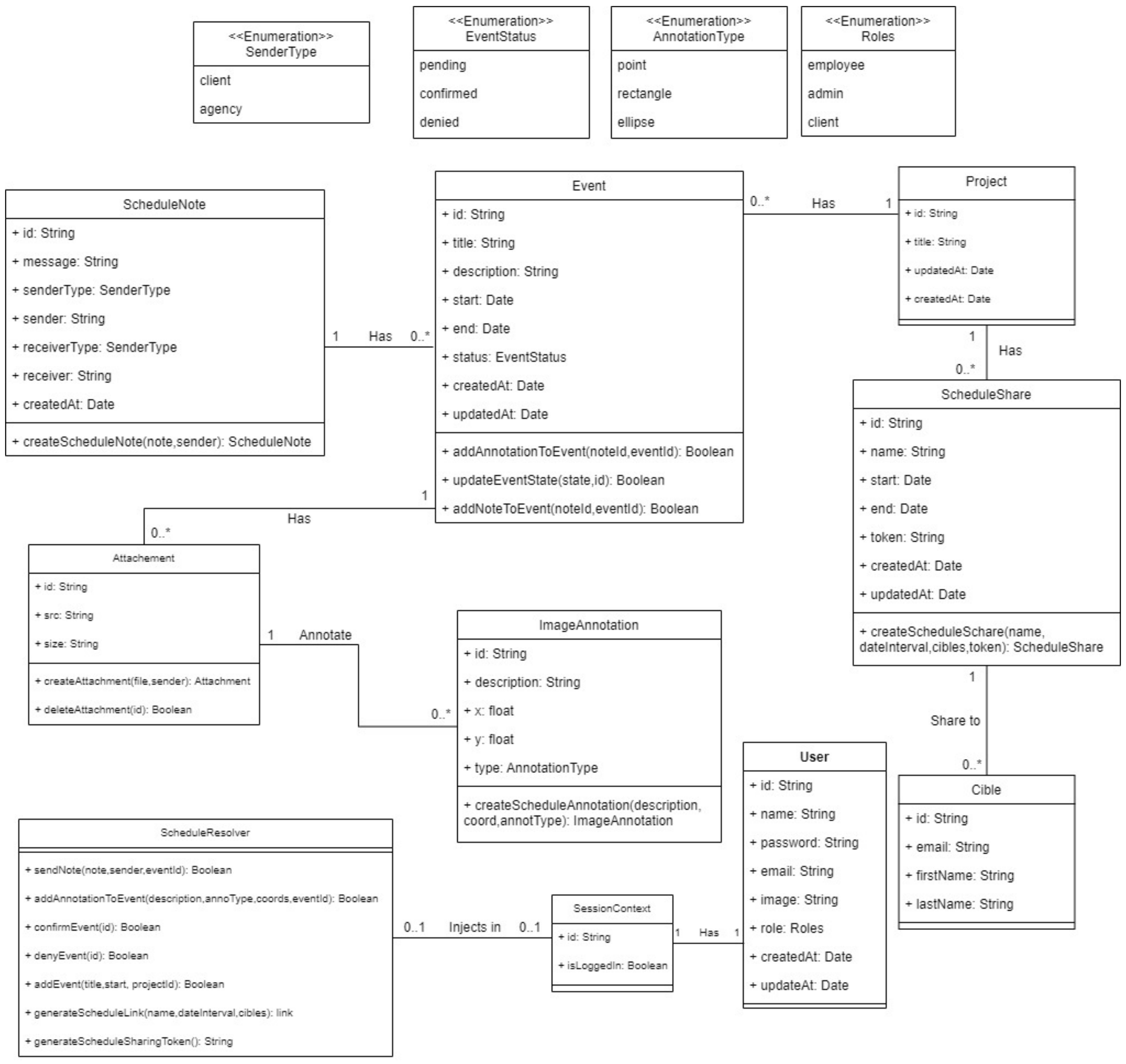


Figure 7.2: Schedule & schedule sharing class diagram

Table 7.1: Descriptive table of schedule & schedule sharing class diagram classes

Class	Description
User	This class represents the user object and will manage and interact with the Users database.
SessionContext	This class will contain the session context for a given user and will be injected into the ProjectResolver so that it can access its attributes.
ScheduleResolver	This class will handle all of the client's schedule requests and will provide the methods required to fulfill these requests.
Event	This class will be the definition of a calendar event and will offer the functionalities needed to manage it.
Project	This class will contain the definition of a project that is associated with a Client.
ScheduleShare	This class will cover the time intervals of a shared calendar as well as the functions for maintaining these schedules.
Cible	This class will contain the credentials for ScheduleShare targets that will be using these shared calendars.
Attachment	This class holds the specification of a stored picture that will be associated with an event and the functions needed to manage them.
ImageAnnotation	This class provides the data for an annotation of an Attachment of type image, as well as the operations for processing it.
ScheduleNote	This class contains the definition of a note delivered to an event as well as the business logic for managing it.

7.2.2 Sequence Diagram

In this part, we will explain two key situations for this feature using a sequence diagram to show how our system components interact. The scenarios will be the creation of a schedule sharing link and the verification of planning.

7.2.2.1 Schedule sharing creation sequence diagram

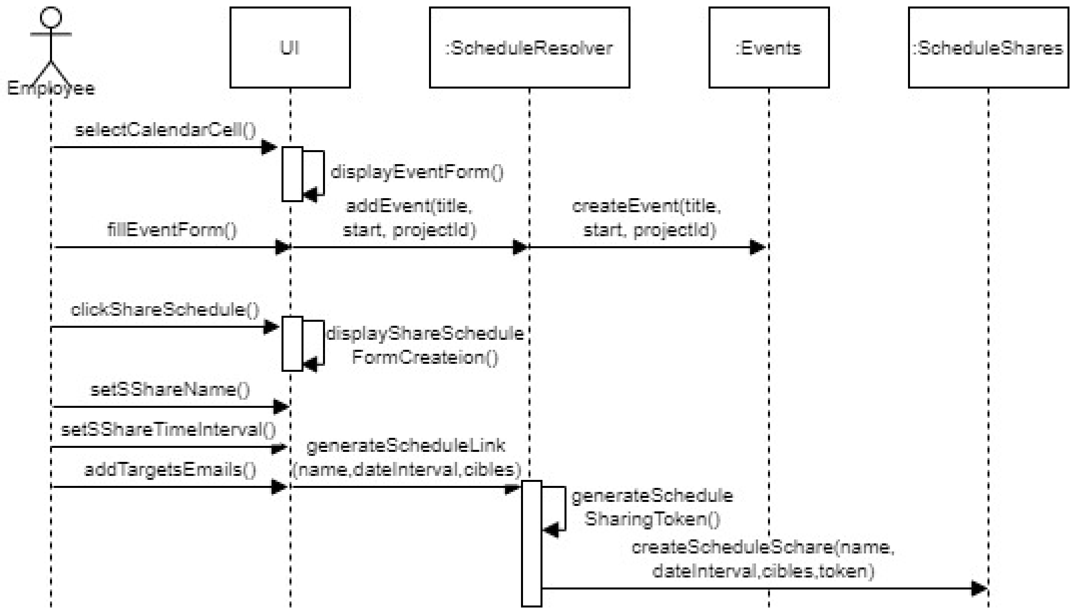

Employee
UI
:ScheduleResolver
:Events
:ScheduleShares
selectCalendarCell()
displayEventForm()
fillEventForm()
addEvent(title, start, projectId)
createEvent(title, start, projectId)
clickShareSchedule()
displayShareSchedule FormCreateion()
setSShareName()
setSShareTimeInterval()
addTargetsEmails()
generateScheduleLink (name,dateInterval,cibles)
generateSchedule SharingToken()
createScheduleSchare(name, dateInterval,cibles,token)

Figure 7.3: Schedule sharing creation sequence diagram

The sequence diagram of figure 7.3 describe the process of creating a schedule share.

1. The user selects a calendar cell and the event form appears.
2. Fill in the form fields, send it to the system, and the event is created.
3. The user clicks on share schedule, a form for share schedule then opens.
4. The user enters the name of the schedule share as well as the interval to dates. Adds the target's information and email addresses, which are then sent to the system. The latter produces a token for that schedule share and builds and stores the schedule share object.

7.2.2.2 Planning verification sequence diagram

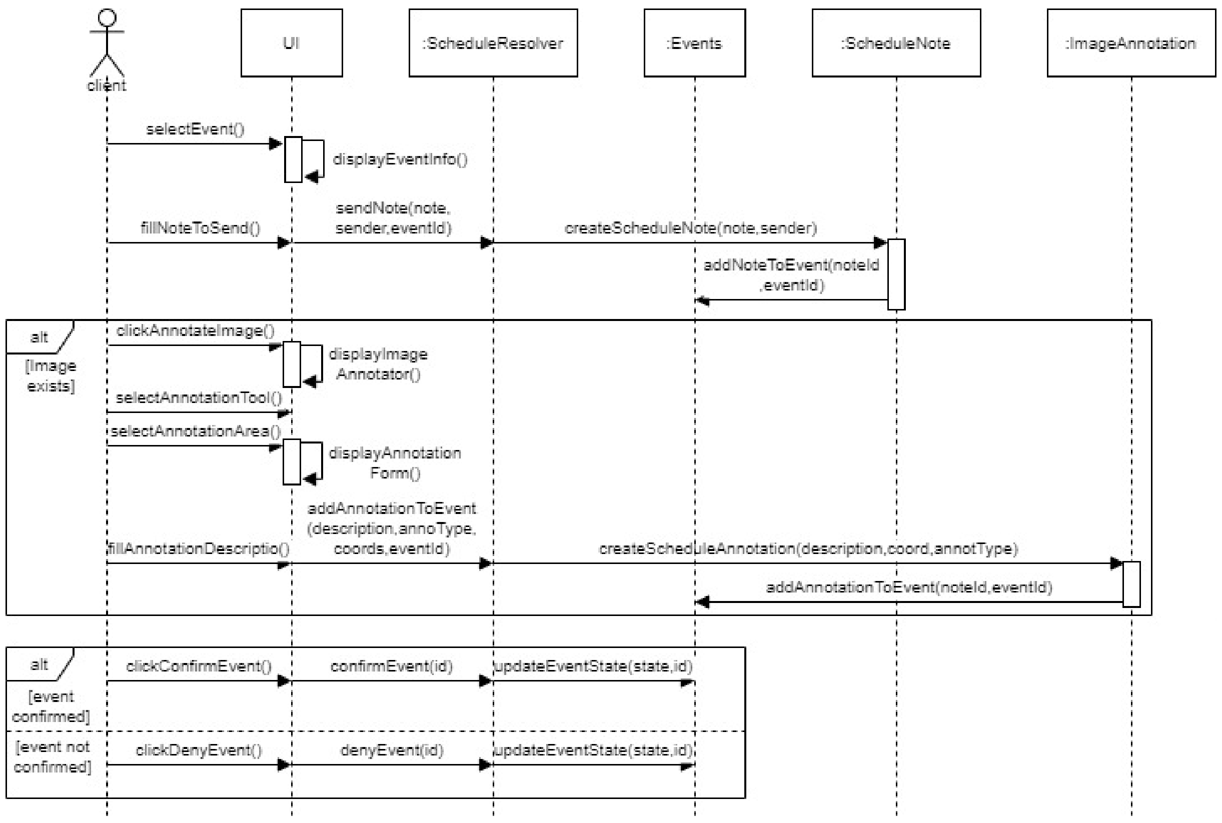


Figure 7.4: Planning verification sequence diagram

Figure 7.4's sequence diagram depicts the process of a customer verifying the planning.

1. The consumer selects the event to check, and the event information appears.
2. The customer fills out the note that they wish to leave for this event, which is subsequently sent to the system, saved, and linked with the event in question.
3. If the event has an image, the customer may click on it to bring up the image annotator, pick an annotation tool, select the area to annotate, write a remark, and submit.
4. The system gets the picture annotation, stores it, and then associates it with the event.
5. The customer confirms or denies the event, the system retrieves the event status and changes the event accordingly.

7.2.3 Mockups

In this section we will show mockups designed for the schedule feature.

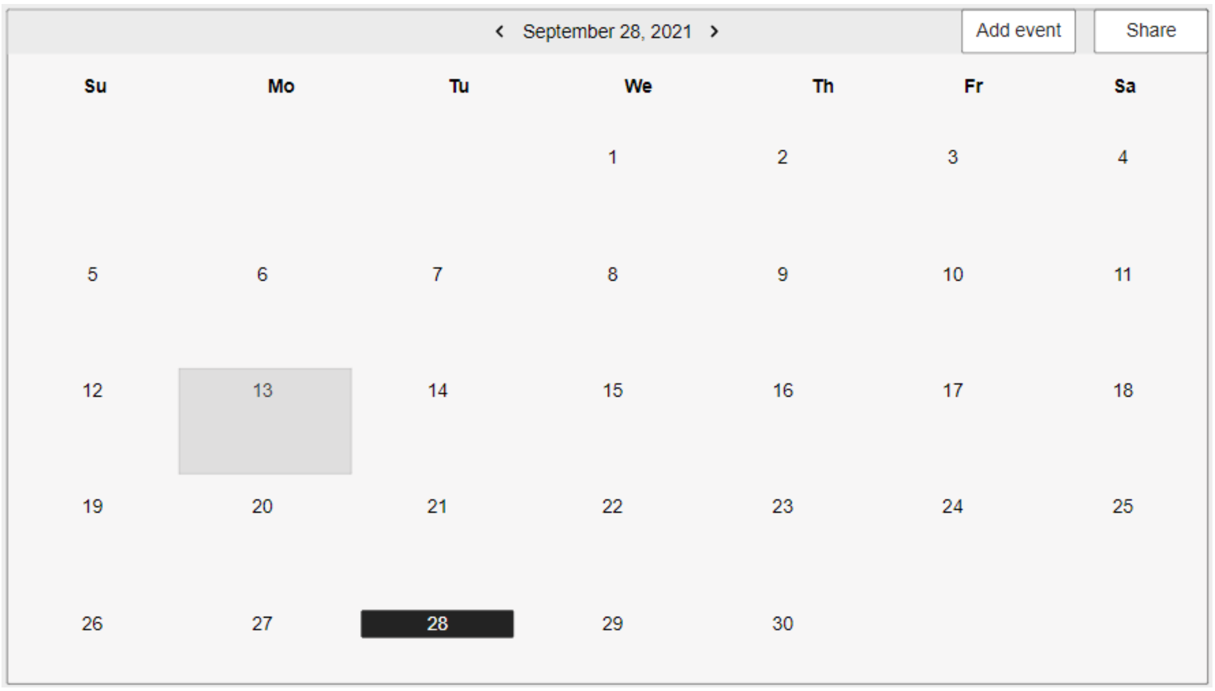


Figure 7.5: Schedule calendar view mockup

Figure 7.5 shows a calendar with a couple of action buttons on the upper right side. The generated events will be displayed in the calendar.

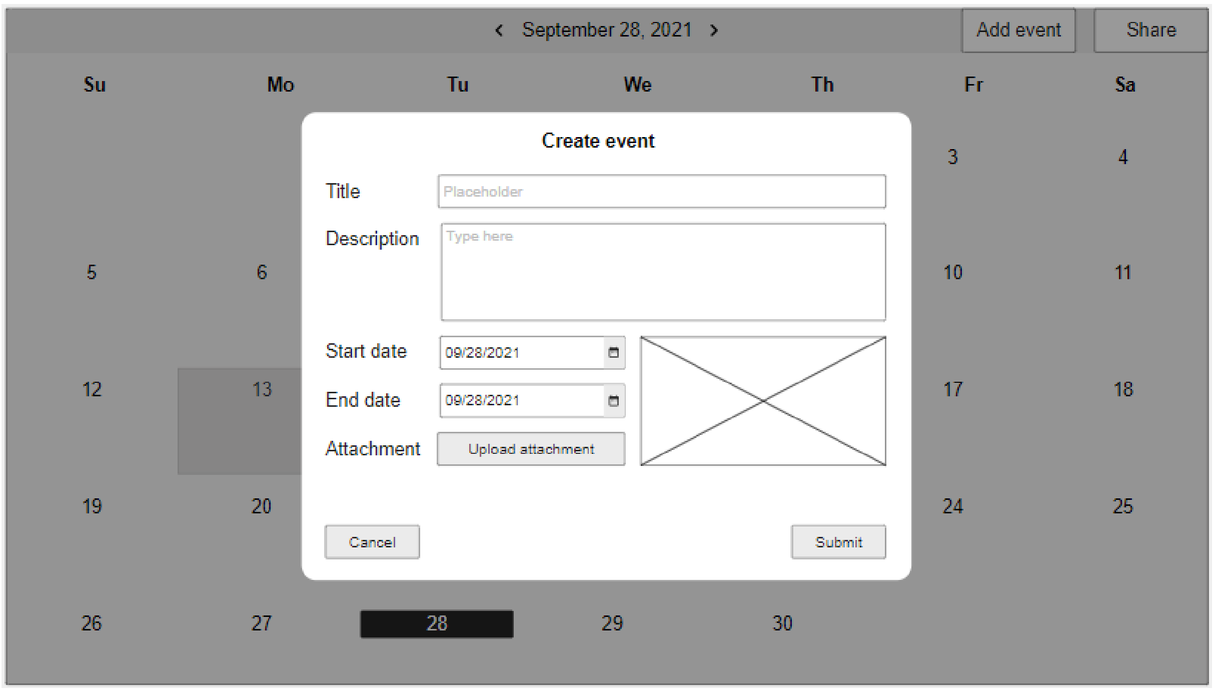


Figure 7.6: Schedule event creation mockup

Figure 7.6 show the event creation modal with the necessary fields

7.3 Implementation

Conclusion

Chapter 8

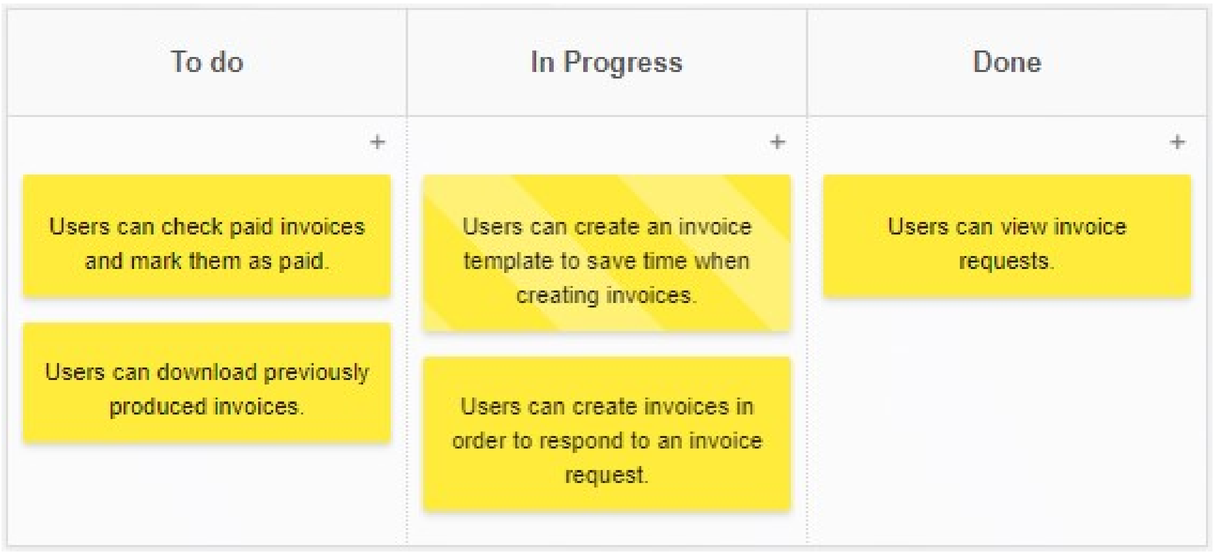
Iteration 5: Invoices

Introduction

We will discuss the processes we took from designing to developing the invoicing functionality in this iteration.

8.1 Backlog

The figure 8.1 depicts the status of our Backlog throughout the fifth iteration.



To do	In Progress	Done
Users can check paid invoices and mark them as paid.	Users can create an invoice template to save time when creating invoices.	Users can view invoice requests.
Users can download previously produced invoices.	Users can create invoices in order to respond to an invoice request.	

Figure 8.1: Iteration 5 backlog

8.2 Design phase

8.2.1 Class Diagram

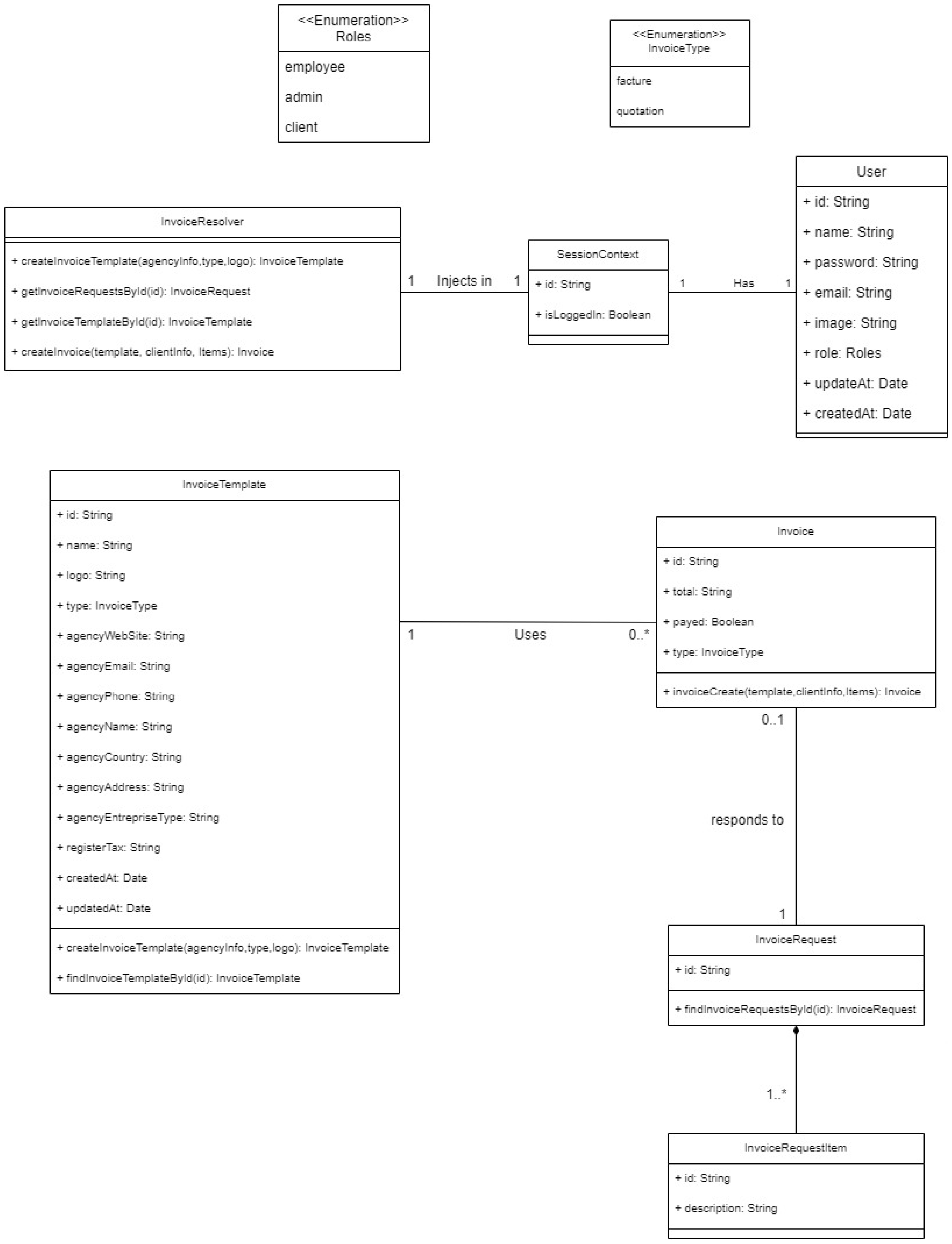


Figure 8.2: Invoice class diagram

Table 8.1: Descriptive table of schedule & schedule sharing class diagram classes

Class	Description
User	This class represents the user object and will manage and interact with the Users database.
SessionContext	This class will contain the session context for a given user and will be injected into the ProjectResolver so that it can access its attributes.
InvoiceResolver	This class will handle all of the client's invoice requests and will provide the methods required to fulfill these requests.
InvoiceTemplate	This class will include the specification of an invoice template, which will aid in the creation of invoices by loading non-changeable data. It also has the necessary functionality for managing these templates.
InvoiceRequest	Is the class that will include a list of InvoiceRequestItem that indicate exactly what items are needed in the invoice.
InvoiceRequestItem	This class will hold the description of an invoice request item and will contain the necessary methods for managing them.

8.2.2 Sequence Diagram

8.2.2.1 Invoice creation sequence diagram

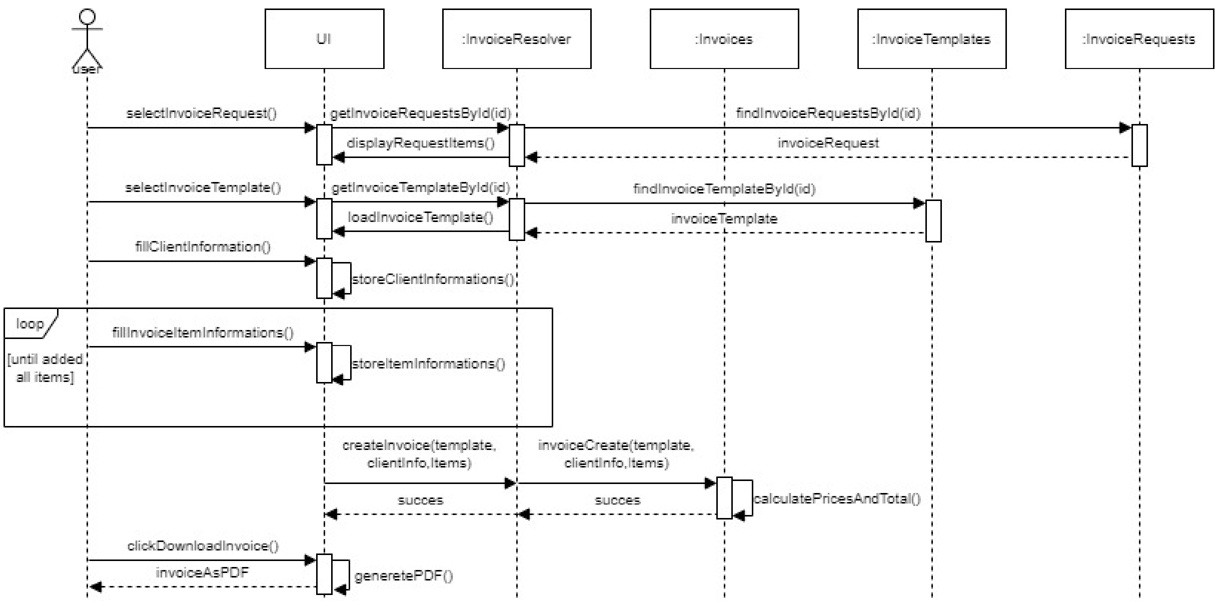


Figure 8.3: Invoice creation sequence diagram

The sequence diagram in Figure 8.3 shows the process of producing an invoice.

1. The The user picks the invoice request to which want to respond; the request items will then display when they have been requested from the system.

2. Selects an invoice template depending on the invoice type then the system retrieves and loads it.
3. Fills out the client information, which is first saved locally.
4. The user continues to enter invoice item information such as the description and unit pricing, and each item is initially saved locally.
5. The user submits all locally stored data to the system, which processes it and creates the invoice object, calculates the total amount, stores it, and returns it.
6. The user clicks the download button, the invoice is downloaded as a PDF.

8.2.2.2 Template creation sequence diagram

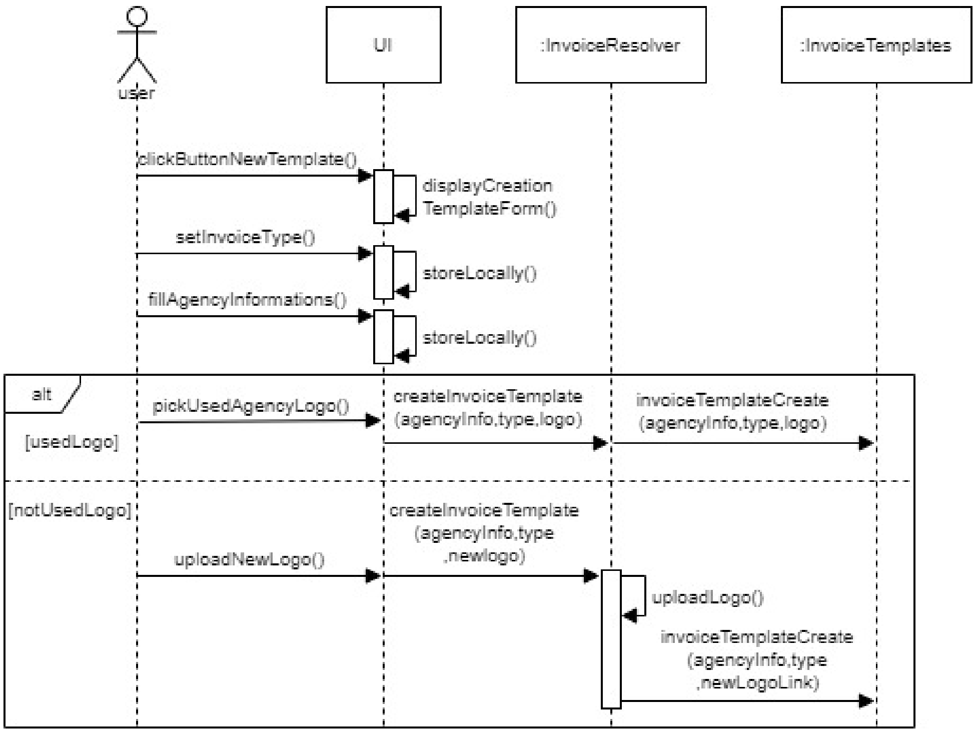

user
UI
:InvoiceResolver
:InvoiceTemplates
clickButtonNewTemplate()
displayCreation TemplateForm()
setInvoiceType()
storeLocally()
fillAgencyInformations()
storeLocally()
alt
[usedLogo]
pickUsedAgencyLogo()
createInvoiceTemplate (agencyInfo,type,logo)
invoiceTemplateCreate (agencyInfo,type,logo)
[notUsedLogo]
uploadNewLogo()
createInvoiceTemplate (agencyInfo,type ,newlogo)
uploadLogo()
invoiceTemplateCreate (agencyInfo,type ,newLogoLink)

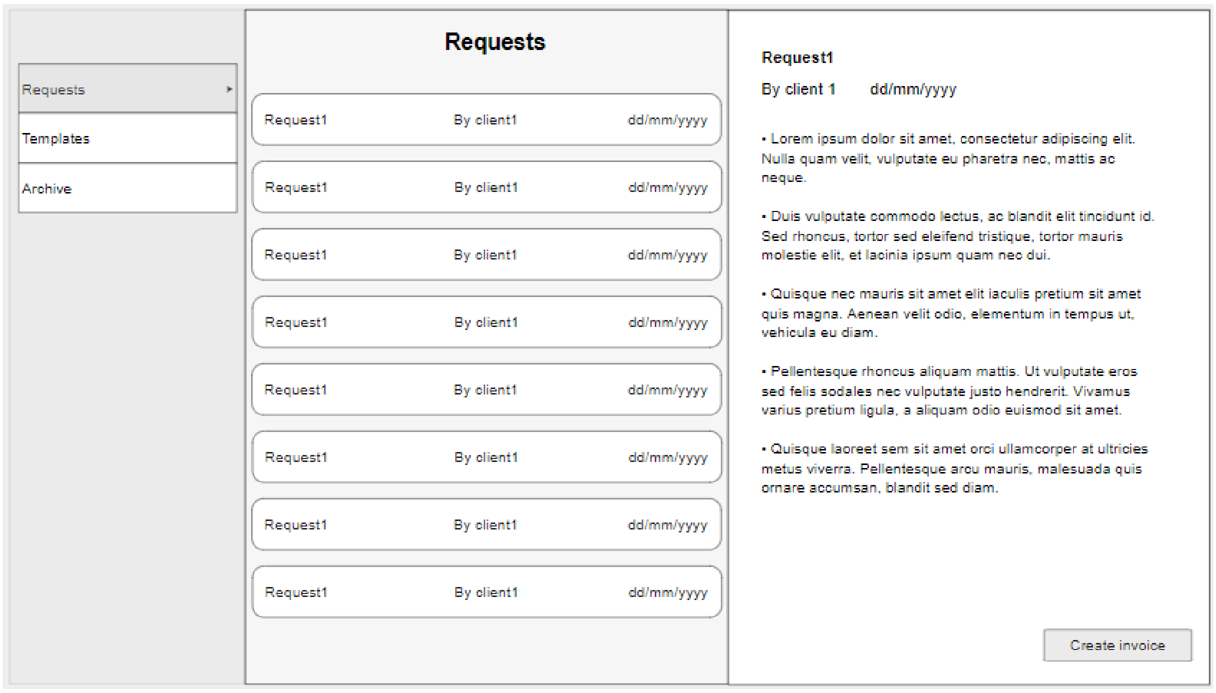
Figure 8.4: Template creation sequence diagram

The sequence diagram in Figure 8.4 shows the process of producing an invoice template.

1. Clicks on new invoice template button and the invoice template form will appear.
2. Sets the invoice type, it gets stored locally.
3. Fills out the agency information, which is first saved locally.
4. If the user wishes to utilize previously used logos, just selects them and sends them to the system together with previously saved data locally. The latter will obtain them, construct the invoice object, and save it.
5. If the user wants to use a new logo, will choose the logo file and submit it to the system together with previously saved data. The latter will initially host the logo before creating and storing the invoice object.

8.2.3 Mockups

In this section we will show mockups designed for the invoice feature.



Request1	By client1	dd/mm/yyyy
Request1	By client1	dd/mm/yyyy
Request1	By client1	dd/mm/yyyy
Request1	By client1	dd/mm/yyyy
Request1	By client1	dd/mm/yyyy
Request1	By client1	dd/mm/yyyy
Request1	By client1	dd/mm/yyyy
Request1	By client1	dd/mm/yyyy

Request1

By client 1 dd/mm/yyyy

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque.
- Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui.
- Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam.
- Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet.
- Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.

Figure 8.5: Invoice requests mockup

Figure 8.5 shows the navigation menu for browsing through the invoicing feature tabs on the left side. The invoice request part is depicted in this figure. The list of requests will be shown in the center, with the request information on the right side.

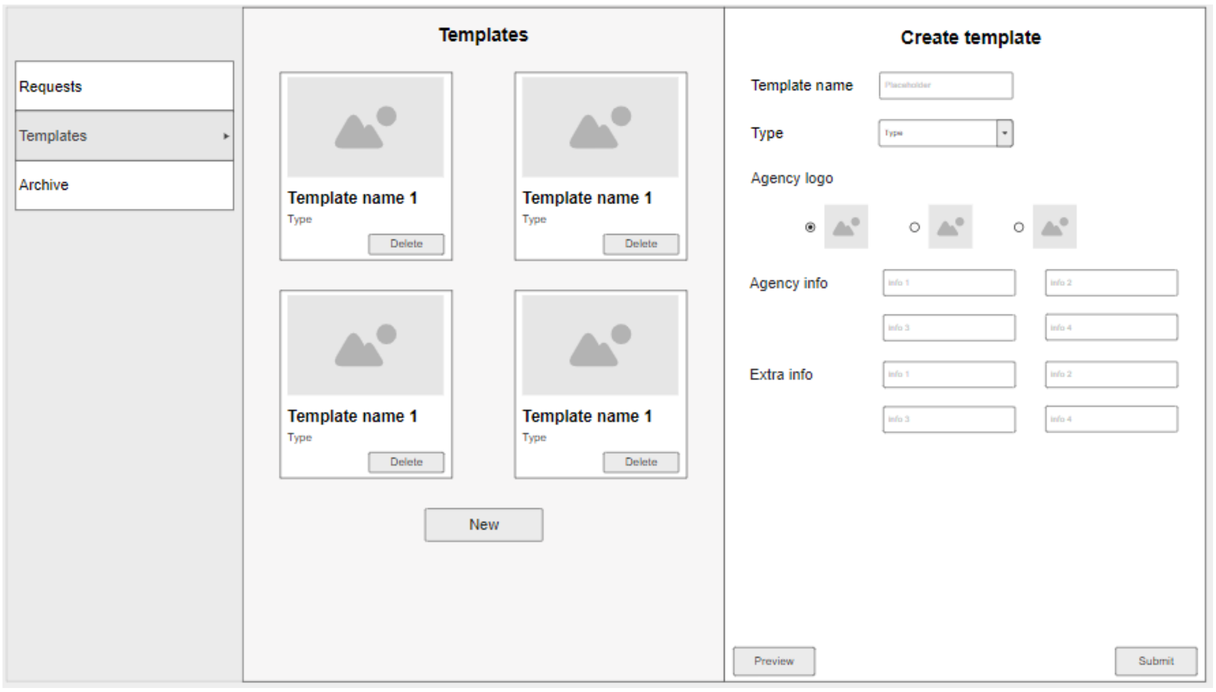
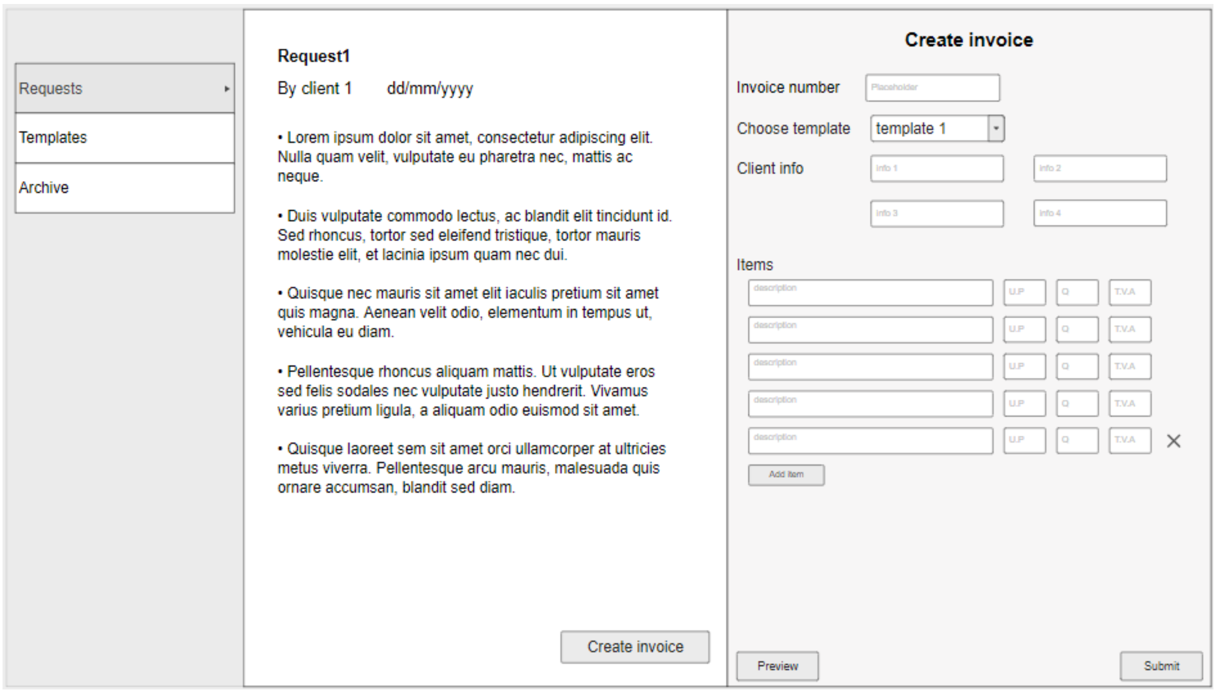


Figure 8.6: Invoice template mockup

Figure 8.6 depicts a list of templates produced as cards in the middle and the template creation form on the right side.



Request1

By client 1 dd/mm/yyyy

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque.
- Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui.
- Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam.
- Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet.
- Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.

Figure 8.7: Invoice creation mockup

Figure 8.7 displays the invoice creation feature, the request information in the middle, and the create invoice form on the right.

8.3 Implementation

Conclusion

Conclusion

Bibliography

[1] *ActiveCollab*. URL: `https://activecollab.com/`. (accessed: 08.04.2021).

[2] *ActiveCollab Pricing*. URL: `https://activecollab.com/pricing`. (accessed: 15.05.2021).

[3] *Clickup*. URL: `https://clickup.com/`. (accessed: 08.04.2021).

[4] *ClickUp Pricing*. URL: `https://clickup.com/pricing`. (accessed: 15.05.2021).

[5] *Discord home page*. URL: `https://discord.com/`. (accessed: 05.04.2021).

[6] *FrontEndFrameWorkComparaison*. URL: `https://dev.to/hb/react-vs-vue-vs-angular-vs-svelte-1fdm`. (accessed: 08.05.2021).

[7] *Kanban board*. URL: `https://www.atlassian.com/agile/kanban/boards`. (accessed: 06.04.2021).

[8] *Kanban vs Scrum*. URL: `https://www.atlassian.com/fr/agile/kanban/kanban-vs-scrum`. (accessed: 06.04.2021).

[9] *limit WIP*. URL: `https://www.atlassian.com/fr/agile/kanban/wip-limits`. (accessed: 06.04.2021).

[10] *Slack Pricing*. URL: `https://slack.com/intl/en-tn/pricing`. (accessed: 05.04.2021).

[11] *Trello home page*. URL: `https://trello.com/`. (accessed: 05.04.2021).

[12] *Trello Pricing*. URL: `https://trello.com/en/pricing`. (accessed: 15.05.2021).

[13] *What is Slack?* URL: `https://slack.com/intl/en-tn/help/articles/115004071768-What-is-Slack-`. (accessed: 05.04.2021).

Abstract

This project entitled "Communication agency platform" is part of the end of studies project which has just concluded our engineering training at the Higher Institute of Applied and Technological Sciences of Sousse.
It is an internal project developed inside the agency Comguru.

This project would give the agency and its clients a platform to manage and see their projects, as well as create and share schedules. This platform is meant to reduce customer visits to the agency premises and to provide a direct contact mechanism with the agency's staff.

To carry out this project, we mainly used React, GraphQL and a MongoDB database. We adopted a Kanban methodology for our development process.
Keywords : React, GraphQL, MongoDB, web platform.

Résumé

Ce projet intitulé "Plateforme d'agence de communication" s'inscrit dans le cadre du projet de fin d'études qui vient de conclure notre formation d'ingénieur à l'Institut Supérieur des Sciences Appliquées et Technologiques de Sousse.

C'est un projet interne développé au sein de l'agence Comguru.

Ce projet donnerait à l'agence et à ses clients une plateforme pour gérer et voir leurs projets, ainsi que créer et partager des plannings. Cette plate-forme est destinée à réduire les visites des clients dans les locaux de l'agence et à fournir un mécanisme de contact direct avec le personnel de l'agence.

Pour bien mener à ce projet, nous avons principalement utilisé React, GraphQL et une base de données MongoDB. Nous avons adopté une méthodologie Kanban pour notre processus de développement.

Mots clés : React, GraphQL, MongoDB, platforme web.