National School of Engineers of Tunis

Department of Information and Communication Technologies

End of Year Project II

DEVELOPMENT OF CUSTOMER SERVICE AND SUPPORT SYSTEM

Prepared by:

Iheb Ben Soltane **Amrou Majdoub**

Class:

2A info 1

Supervised by:

Mrs Sonia Alouane

Academic Year 2023/2024

Acknowledgments

We would like to express our deep gratitude to Mrs **Sonia Alouane**, for the considerable efforts made in the success of this project. We greatly appreciated her support and involvement throughout this period, which allowed us to achieve this result. We also wish to thank the entire jury for accepting to review our project.

Abstract

As part of our End of Year Project II, we present a web application aimed at providing a comprehensive solution for customer support. The application includes a front-office for clients to report issues with purchased products or services and access after-sales services, as well as a back-office for agents to manage and resolve customer concerns efficiently.

Keywords: Helpdesk, Ticket, Node Js, React Js, MySQL, Ollama.

Table of Contents

List of Figures

List of Tables

General introduction

E-commerce is the buying and selling of goods or services via the internet. Today, and especially after the global pandemic of COVID-19, such commerce has become highly needed and valuable. Indeed, we need to know what differs an E-commerce business from another: is it only the quality of products, or the shipping speed or users also look for other criteria?

By now, everyone looks to not only get new customers but also guarantee their old clients, as a result they seek out solutions that can be integrated with their e-commerce industry and achieve their needs. As a pertinent solution, sellers are heading to implement a good after-sales system that guarantees customer-care, where clients can express their problems and have someone respond to their needs and fix their issues. Such systems organize customer communication, provide a management of agents who can handle clients' complaints, problems and answer them. In addition, these systems can contain knowledge bases where clients can find answers to common problems or documentations for products or services.

This project is being conducted in this context, which consists of creating a support system composed of Front office directed for clients and a back office for the agent and other staff to work on and complete their tasks. We begin the report by a study of the existing followed by a proposal for a solution responding to the needs detected by this study and the methodology to be used for its realization, this in as a pre-study announcing the first chapter. After, four chapters are proposed: A chapter that will be devoted to the specification of needs, pattern and style architectural and project management with Scrum method. After, we advance the product backlog and the breakdown of functionalities.

Three chapters relating to the development itself which interferes: The first focuses on authentication, user management and role and permission management. The second will have the most important functionalities so it focuses on tickets management, ticket items manipulation as well as client's management. The third and last chapter relates to the chatbot and consulting dashboard. Finally, we concluded our project with a general conclusion that presents a summary of the work, the main results, inputs and the perspectives acquired during this project. this is the introduction to our project

Chapter 1

Project Study

Introduction

In this chapter, we will introduce the general situation and aspects of our final year project. We start by an important review of the software programs and applications that solve the same problems in this project. Then, we will propose a suggested solution to solve the existing problem.

1.1 The context of the project

Due to the economic growth, especially in e-commerce and online services, it has become difficult to meet the needs of customers and help them solve their problems with the products and get the best services. Many of them complain about products for the simple reason that they don't know how to use it, or it wasn't as expected. In addition, they complain to support agents that they can't help them to solve their problems. Such complaints and misunderstanding make companies lose their clients. In this project, our goal is to create a web application for customer services and support systems, that manage clients requests and organize the support system staff. This Web application will be used for companies providing after-sales service.

1.2 Existent Study

In order to develop our project that responds to our needs, we must first go through the stage of studying the existing solutions, which will help us to identify the constraints to be respected during the realization and find the main strengths that allow us to improve the project.

1.2.1 Current Solutions

1. **Liberum Help Desk** Liberum Help Center is a complete help desk solution for IT departments and service providers. The software provides a simple and easy-to-use Web interface for managing and tracking technical support issues. This web application provides configurable e-mail notification, time and activity reports.[1]

Figure 1.1: Librum home page

2. **HelpDeskZ** HelpDeskZ is one of the most convenient and simplistic PHP-based software that allows users to use a web-based support ticket system. The software comes with a very simplified dashboard, multilingual capability, and allows personalization features.[2]

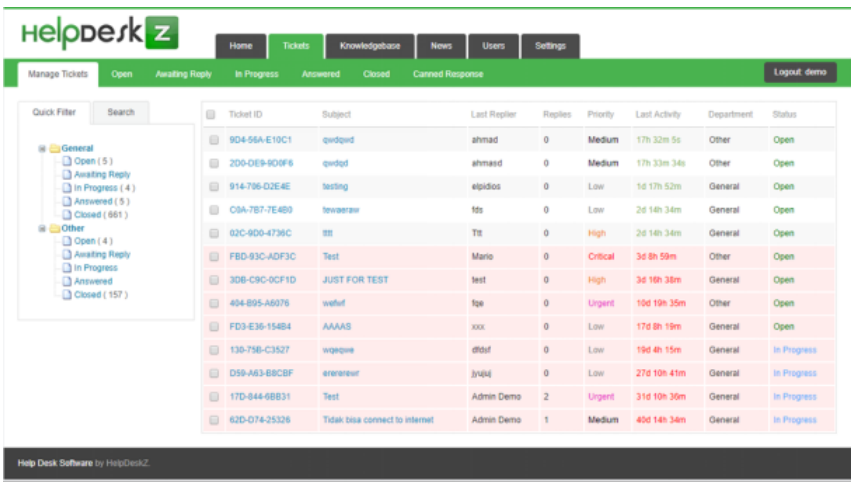


Figure 1.2: HelpDeskZ home page

3. **SpiceWorks** SpiceWorks is a powerful help desk software tool for start-up enterprises. It is equipped with a powerful ticket management system, which is fast and responsive, and has a huge knowledge base.[3]

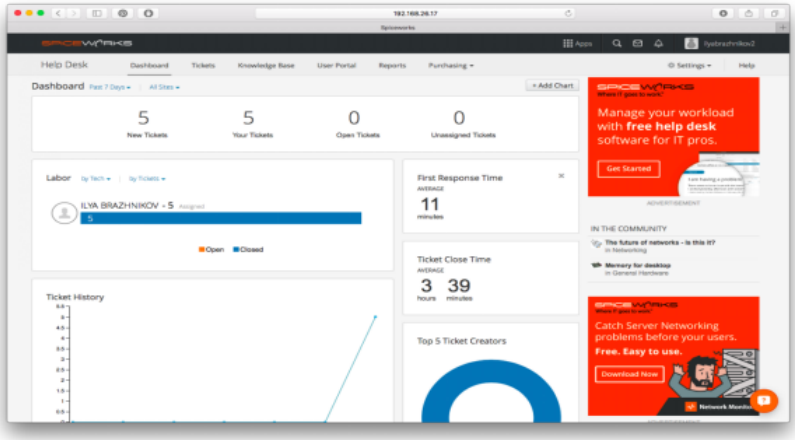


Figure 1.3: SpiceWorks home page

4. **Katak-Support** Katak-support is an online help desk software solution for managing the flow of customer needs and requests. It has email and message pop-up functions, and due to its simple and efficient design, it is very suitable for start-ups and small businesses.[4]

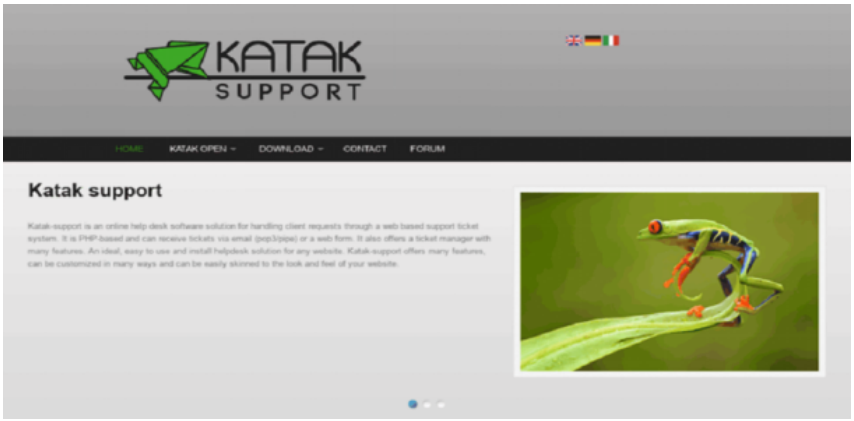


Figure 1.4: Katak home page

1.3 Comparative Study of the Existing Solutions

At the end of the study of the existing situation, we have determined the criteria to be considered in the comparison and application evaluation, and we were able to establish

an analytical table 1.1 to provide an easier way to compare available products. The comparison is based on the needs and features we are trying to develop in the application, which are:

- ⋆ **Ticket Management:** All Tickets are well organized with the ability to modify their properties.
- ⋆ **Knowledge Base:** Users can find solutions to their problems on their own by looking in the research base.
- ⋆ **Dashboard:** Agents and supervisors can view all ongoing processes at once.
- ⋆ **Escalation:** Tickets can be assigned and reassigned to other agents for more efficiency and to avoid agents collusion.
- ⋆ **Role Management:** Control roles permissions and manage their activities.
- ⋆ **Point-to-Point Contact:** Customers and agents are in direct contact (messaging, calls).
- ⋆ **Client Tracking:** Users can track the ticketing history.

Table 1.1: Comparative table for available solutions

Features	**Librum**	**HelpDeskZ**	**KataSupport**	**SpiceWorks**
Ticket Management	+	+	+	+
Knowledge Base	+	+	-	+
Dashboard	-	-	-	+
Escalation	+	+	+	+
Role Management	-	-	-	-
Point-to-Point Contact	+	+	+	+
Client Tracking	-	-	-	-

As shown in the table above, the study allowed us to identify several requirements expected by users but not fully respected by existing solutions. For this, we have thought of a more satisfactory solution by integrating more features.

1.4 Proposed Solution

After conducting a deep study of the previous comparison table, we aim to develop a product that can meet the needs of the customer and other users (agents, admins). For that reason, we propose in this work to develop a web application that provides two spaces: A Front office and a Back office. Our application will contain, initially, the following features:

Front Office:

The client space where the client can:

- Consult ticketing history.
- Send requests to the support agent.
- Access the Knowledge base by asking a chatbot.

Back Office:

Dedicated to support agents to:

- Manage roles and permissions.
- Provide valid management for client's tickets and requests.
- Manage tickets items.

1.5 Methodology

Before executing our project, it is necessary to choose a working methodology and a follow-up process in order to obtain reliable software. The chosen methodology will help us to build, plan and control the application development while respecting the rules of computer engineering.

1.5.1 Scrum

Scrum is one of the main agile frameworks, which has a series of guidelines for managing the development of product. The methodology is iterative, measurable and incremental because it focuses on tightening up the development cycle.

The agile Scrum method is based on project decomposition in iterations called sprints; these iterations follow each other and last for 1 to 4 weeks. Every iteration Start with a planning meeting called ”Sprint Planning” to define the goals and decompose each requirement into several tasks. The sprint ends with a demonstration that evaluates and verifies what has been achieved. Before starting a new iteration, the team conducted a retrospective to analyze what was developed during the last sprint, in order to improve the next sprint and increase the added value of the product.

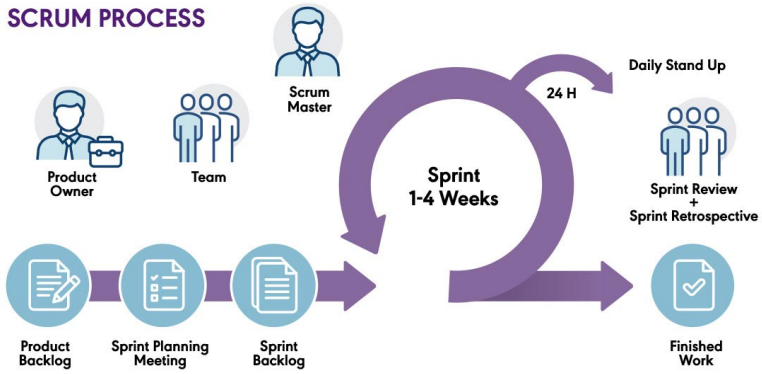


Figure 1.5: Scrmum process

1.5.2 Avantages of Scrum

- **Iterative and Incremental Development:** Scrum allows for the iterative and incremental development of the website, enabling us to deliver working increments of the product at the end of each sprint. This approach allows for early and continuous delivery of valuable features, which is important for a customer support website where user needs may evolve.

- **Flexibility and Adaptability:** Scrum is known for its flexibility and adaptability to changing requirements. In the context of a customer support website, where requirements may evolve based on user feedback or market changes, Scrum provides a framework that allows us to respond to these changes quickly and effectively.

- **Continuous Improvement:** Scrum promotes a culture of continuous improvement through regular retrospectives. This allows us to reflect on our processes and practices after each sprint and make adjustments to improve efficiency and effectiveness.

1.5.3 Design language

Most languages are object-oriented, the shift from functional programming to Object-oriented is not easy, and so methods for modeling languages need to be used. As part of the project, we chose to use the unified modeling language to model the system. UML aims at graphical modeling based on pictograms. Therefore, UML provides us with a diagram that represents the application to be developed.

conclusion

In this chapter, we introduced our project study. We established a study on the current situation to clarify what to achieve, finally we have introduced research on some existing development methods to select the most adopted one to use in this project. In the next chapter, we'll examine and discuss our system's requirements.

Chapter 2

Project Analysis

Introduction

In this chapter, we will analyze and explain the requirements of our system, which is an important stage for the success of the project. Actually, a detailed description of the requirements is helpful for work development.

2.1 Needs Analysis

Our application must meet the requirements of its users, so in the following we will present the actors who will intervene in the application, the functional needs of all them as well as the non- functional needs of the system.

2.1.1 Identification of Actors

An actor corresponds to a role and which interacts directly with the system. In our application,we have four major actors:

- **Client:** A customer of the application who can access his account, create a new one, and interact with system services and agents.
- **Agent:** A qualified person who can log into his working account, where he can act according to the permissions he has (reply to clients' tickets,manage items, etc...).
- **Admin:** In addition to agents' functionalities, the admin can assign tickets to an agent.
- **Webmaster:** The person who has complete control over the system.

2.1.2 Functional Needs

The drafting of functional needs is based on the actions that the system must perform and the expression of the customer's needs. Therefore, our application must respond to the following requirements:

- **Client Authentication and Sign Up:** Every client has the ability to connect to his account or create a new one.
- **Edit Account:** The system allows the user to change and update his personal information.
- **Tickets Manipulation:** The system allows users to perform manipulation (create, reply, delete, assign) on tickets based on the permissions they have.
- **Role Management:** super admin has the ability to manage roles and control their roles and permissions.
- **Real-Time Chat:** Clients and agents can communicate in real-time.
- **asking chatbot :** The system provides a chatbot that will help client find solution to some of their problems rapidly .

2.1.3 Non-Functional Needs

Non-functional needs are constraints that are assumed to ensure the quality and proper functioning of the system such as:

- **Extensibility:** Allowing the possibility to add or modify new features to the application.
- **Security:** The application should be highly secure, allowing authenticated and authorized access to sensitive and private information only through a login and password.
- **Interface:** User interfaces must respect the principles of human/machine interfaces.
- **User-Friendliness:** The application should be simple and easy to handle and work with. A good UX (User experience) consists of making the end user journey more pleasant, intuitive, fluid, and optimal.

2.2 Translation of Needs

The use case diagram serves as a streamlined model depicting the functionality of the application, specifying both the actors utilizing it and the services it provides to them.

2.2.1 Global Use Case Diagram

The diagram 2.1 illustrates the role of each actor and the scope of their responsibilities.

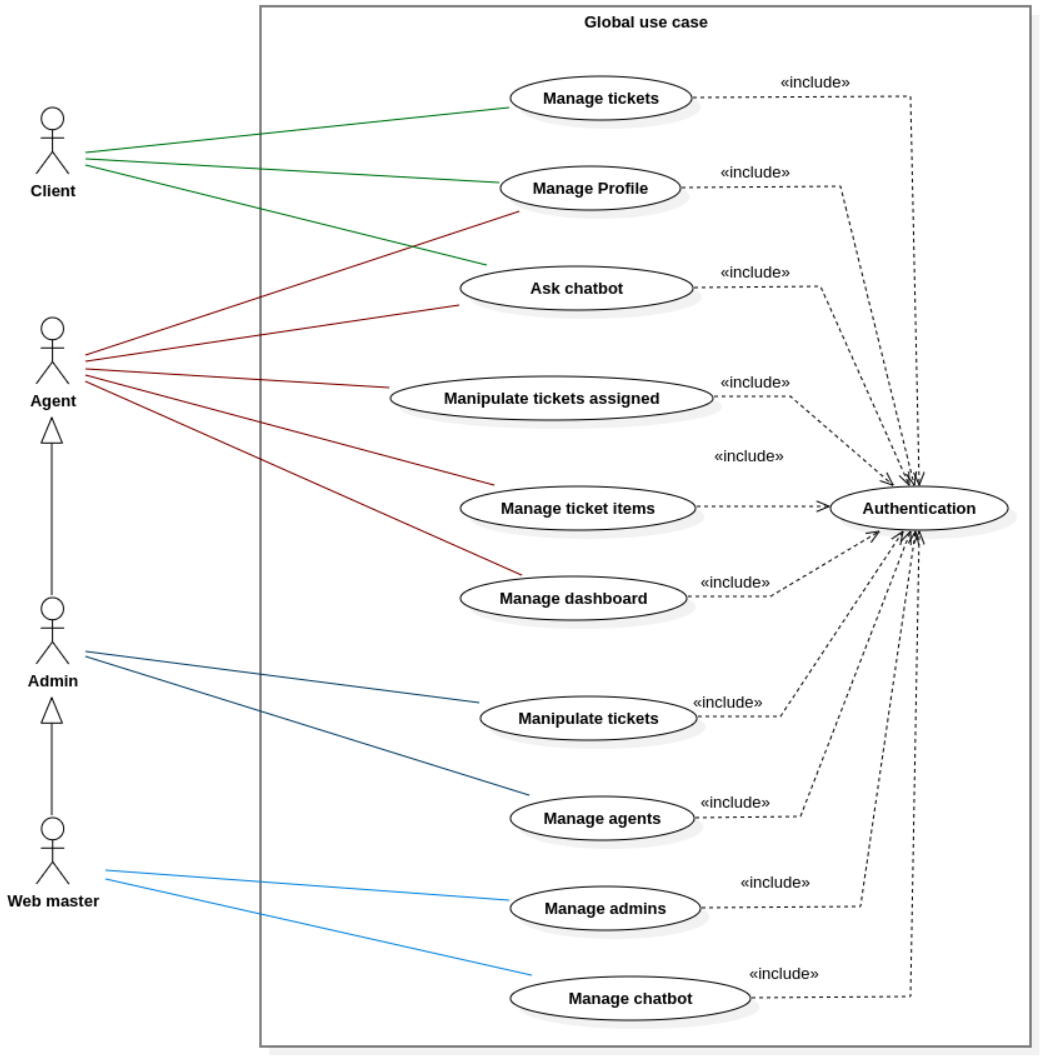


Figure 2.1: Global use case diagram

2.3 Pattern and Architectural Style

This section details the initialization of our project where we will describe the architectural model of our application.

2.3.1 Architectural style

An architectural style is a logical model of application architecture which aims to model the application; it helps to get an overview of the system before it is developed. Each system requires 3 levels or layers for it to be achieved:

Presentation layer.

Treatment layer (or service layer).

Data access layer.

In order to place our application, we adopted a three-tier architecture which consists in separating the realization of the three parts (data storage, application logic, presentation) this separation means that each part can be deployed on a standalone server. The establishment of this type of architecture allows in all cases greater scalability and more security to the system. This architecture is composed of three levels, detailed as following:

- **The client:** or the web browser installed on the client workstation which is responsible for the display and maintains dialogue with the user and the application server: it triggers a request to the processing layer and receives a response.
- **Application server:** also called middleware, is responsible for the functional part of application and data processing in order to offer these services (functions) to the presentation layer based on customer requests.
- **The database server:** This server will be used to store and restore data from the application.

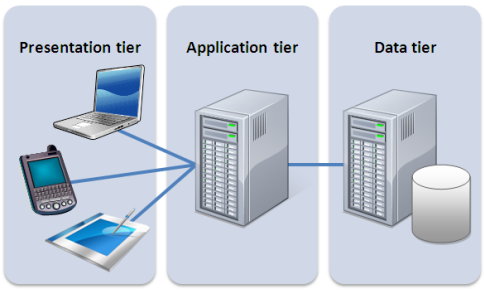


Figure 2.2: Thee tier architecture

2.3.2 Architectural pattern

The **RESTful API** architectural pattern is a design approach that emphasizes the use of HTTP methods to perform operations on resources. This pattern is particularly well-suited for web services and applications that need to communicate over the internet, providing a standardized way to structure and interact with data. Unlike the Model-View-Controller (MVC) pattern, which is primarily concerned with the user interface and interaction layer of an application, RESTful APIs focus on the data layer, making them ideal for building scalable and flexible back-end systems. Incorporating the main logical components of a system that utilizes the RESTful API architectural pattern, we can identify four key components:

- **Client:** Initiates interactions with the server by sending REST requests. This could be a web browser, mobile app, or any application capable of making HTTP requests.
- **Server:** Acts as the central component that receives REST requests from clients, processes them (often involving database interactions), and sends back responses. The server's architecture supports concurrent request handling, ensuring efficient and scalable operation.
- **Database:** Serves as the persistent storage layer, where the application's data is stored and managed. The server interacts with the database to perform CRUD operations based on the REST requests it receives.
- **RESTful API:** Defines the interface between the client and the server, specifying how requests are made and how responses are structured. It outlines the endpoints (URLs) for accessing resources, the HTTP methods for different operations, and the data exchange format. The RESTful API is designed to be stateless, requiring each request to contain all necessary information for processing.

2.3.3 Advantages of using the RESTful API architectural pattern

These components work together to enable the development of scalable, flexible, and efficient web services and applications.

- **Scalability:** The stateless nature of RESTful APIs, combined with the ability to cache responses, makes it easier to scale applications horizontally by adding more servers to handle increased load.

- **Flexibility:** RESTful APIs can be used to build a wide range of applications, from simple web services to complex, distributed systems. The flexibility of the pattern allows developers to tailor the architecture to meet the specific needs of their project.
- **Layered System:** The architecture is composed of multiple layers, each with a specific role. This separation of concerns allows for easier maintenance and updates to individual components without affecting the entire system.
- **Uniform Interface:** RESTful APIs use a consistent and predictable interface, making it easier for developers to understand and use the API. This uniformity reduces the learning curve and increases the efficiency of development.

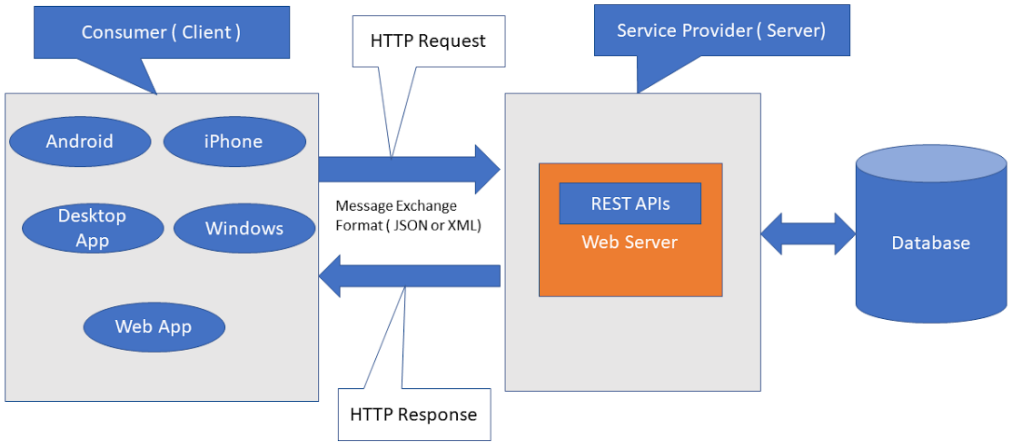


Figure 2.3: RESTful API architecture

2.4 Project management

To effectively transition from a global use case diagram to a product backlog using Jira for agile project management, it's crucial to understand the relationship between these elements and how they contribute to the development of a user-centric application. Here's a structured approach to guide you through this process:

Analyzing the Global Use Case Diagram

The global use case diagram provides a high-level view of the interactions between the system and its users. It outlines the various functionalities and features that the system needs to support to fulfill its purpose. Each use case in the diagram represents a specific goal that a user can achieve with the system.

Identifying User Needs

From the use case diagram, identify the needs of each user group. This involves understanding what actions users want to perform, what information they need to access, and how they expect the system to behave. These needs will guide the development of features and functionalities in your product backlog.

Creating the Product Backlog

The product backlog is a prioritized list of features, enhancements, and fixes that need to be implemented in your product. It serves as a roadmap for the development team, outlining what needs to be done to achieve the product's goals. Here's how to create the product backlog:

- Prioritize Features: Based on the user needs identified from the use case diagram, prioritize the features that are most critical to the success of your application. Consider the value these features bring to the user and the business.
- Estimate Effort: For each feature, estimate the effort required to implement it. This can be done using story points, hours, or any other estimation method that your team finds useful.

After these steps we finally got our three backlogs:

This is the first sprint backlog.

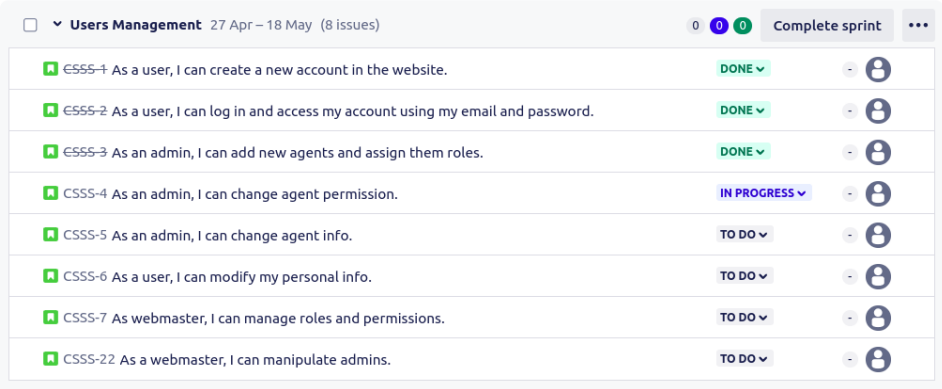


Figure 2.4: First sprint backlog

The figure 2.5 represents the second srpint backlog.

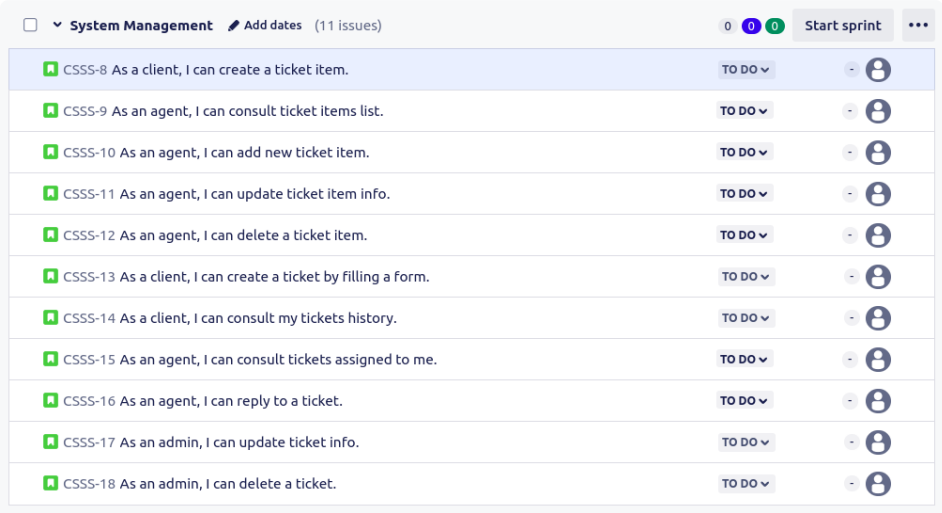


Figure 2.5: Second sprint backlog

The issues of the third sprint are shown in this figure

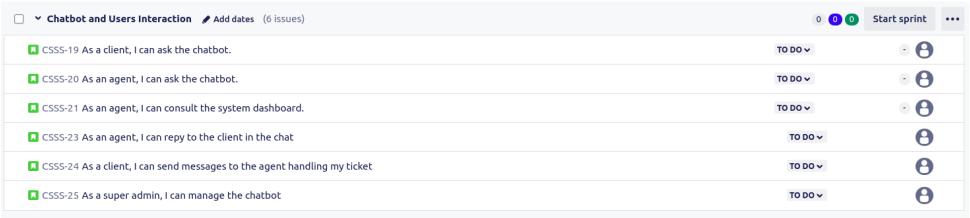


Figure 2.6: Last sprint backlog

2.5 Software environment

2.5.1 Design tools

Star UML

is a software engineering tool for system modeling published by MKLabs, that enables us to create and customize UML 2 diagrams, Entity-relationship, Data flow diagram, Flowchart, Mindmapand and more.[5]

2.5.2 Development environment

Visual Studio Code

is a free, powerful, lightweight code editor for Windows, macOS and Linux. Based on open source, it is highly customizable with over 25,000 extensions, for every developer and every programming language.[6]

MySQL Workbench

is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X.[7]

Figure 2.7: MySQL Workbench logo

Ollama

Ollama facilitates the local execution of open-source large language models like Llama2 by consolidating model weights, configuration, and data into a unified package called a Modelfile, streamlining setup and configuration, including GPU utilization.[8]

Figure 2.8: Ollama logo

2.5.3 Development technologies

Node.js

Node.js is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.[9]

Express

is a versatile Node.js web application framework, offering a minimalistic yet powerful foundation for building web and mobile applications. With its extensive HTTP utility methods and middleware support, creating robust APIs becomes straightforward and efficient.[10]

React

is the library for web and native user interfaces. Build user interfaces out of individual pieces called components written in JavaScript.[11]

Tailwind CSS

is a utility-first CSS framework for rapidly building modern websites without ever leaving your HTML.[12]

conclusion

In this chapter, we have specified the specifications and requirements of the application to be established in addition to that we have presented the pattern and architectural style that we are going to apply in our application following that, we released the product backlog, which included a list of the application's supported features. In the next chapter, we will discuss in detail the first sprint which contains the system functionalities to be developed.

Chapter 3

Sprint 1: Users Management

Introduction

Following the presentation of the project specification and the development of our application backlog, it is time to build our scrum board who is the backbone of this agile methodology and who will keep us on track to meet our requirements. Therefore, we will begin our first functionalities which are authentication, agent management, admin management and role control.

3.1 Functional specifications

The use case diagram is derived from the functional specification, which is considered as the most important step for each iteration. These diagrams are useful for describing the project's functionalities, which are triggered by the actors in order to accomplish their goals.

3.1.1 Use case diagram

Use case diagrams are a type of UML diagram that represents the functional behavior of a software system. Actually, a use case is a discrete unit of interaction between a system and a user (human or machine). As part of our analyses of the first sprint functions we have prepared a global use case shown in the figure 3.1.

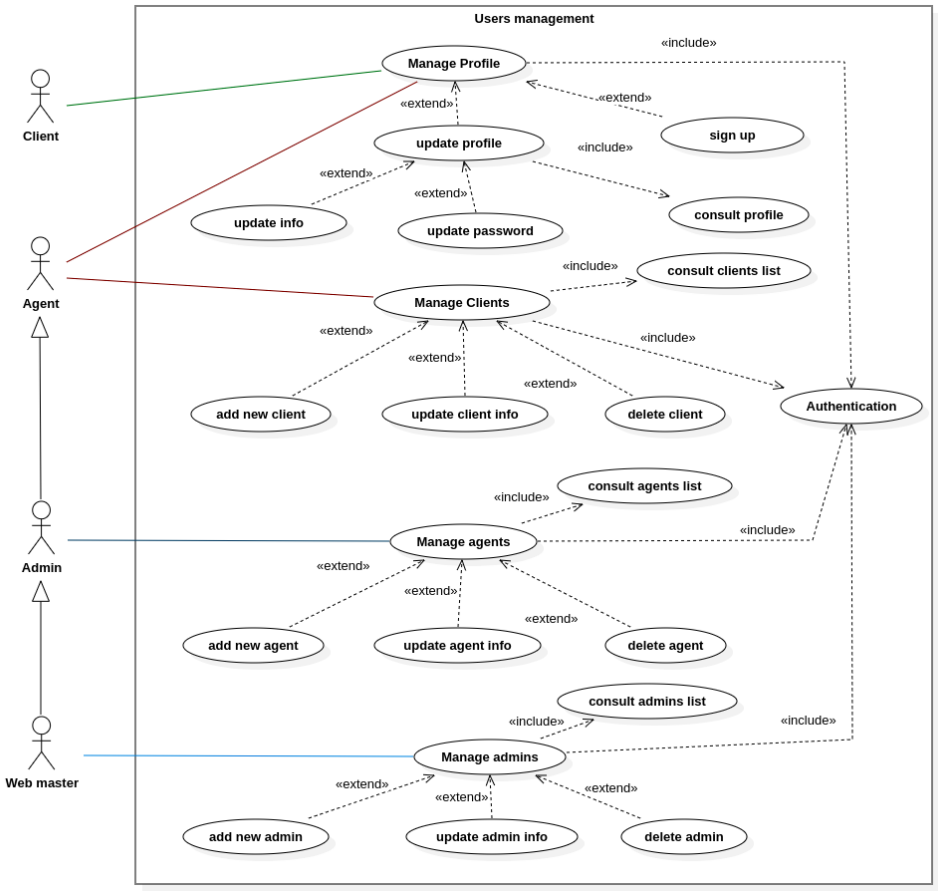


Figure 3.1: Users management use case diagram

3.1.2 Description of use cases

We improved the use cases to include an overview of the various scenarios that might occur.

3.1.2.1 Text description of Authentication use case

The use case Authentication is illustrated in detail in the following table.

Use case	Authenticate
Main Actor	Client / agent
Summary	Every client or an agent can log in to the system if he is registered.
Pre-condition	- The logger must have an account saved in the system base. - The Email and the Password must be correct and compatible.
Post condition	Client or agent is authenticated
Main scenario	1- User asks for the login interface. 2- Login interface is displayed. 3- Client or agent puts his email address and password. 4- The logger confirms his input by clicking on the “Login” button. 5- The system checks the entered parameters. 6- The system displays the wanted interface with the logged user in.
Alternative scenario	- The system displays an error message if the user parameters are false. - The system won’t execute if there is invalid input.

Table 3.1: Text description of the Authentication use case

3.1.2.2 Text description of the Sign up use case

To further describe the use case Sign up, we have created the following table.

Use case	Sign up
Main Actor	User
Summary	Every User can create a new account in the system by filling a form with appropriate information.
Pre-condition	Fill all the wanted inputs with correct information.
Post condition	- Client is created and saved in the system database.
Main scenario	1- User asks for the Sign up interface. 2- Sign up interface is displayed with a form to fill. 3- User fills all the input fields. 4- User confirms his inputs by submitting on the Sign up button. 5- The system checks the input validation and saves the new client.
Alternative scenario	- The system won’t save the new client if there is an empty or invalid input.

Table 3.2: Text description of the Sign up use case

3.1.2.3 Text description of the Update profile use case

The use case Update profile is illustrated in detail in the following table.

Use case	Update profile
Main Actor	Client / agent
Summary	Every registered user can update his personal info related to his account in the system by filling a form with the new appropriate information.
Pre-condition	Client, agent, or admin must be authenticated in the system.
Post condition	Client, agent, or admin info are updated in the system database.
Main scenario	1- Client, agent, or admin access the settings interface. 2- The system presents the account info in a form. 3- Client, agent, or admin choose either to change personal info or the account password. 4- The registered user changes the wanted data then saves the new info by clicking on the “Save” button. 5- The system verifies the info then it displays the updated info.
Alternative scenario	- The system won’t save the new info if there is empty or invalid input. - Password won’t change if the old password is invalid. - The system resets to old info when the Client, agent, or admin clicks on the “Cancel” button.

Table 3.3: Text description of the Update profile use case

3.1.2.4 Text description of the Manage agents use case

The use case Manage agents is detailed in detail in the table 3.4 below.

Use case	Manage agents
Main Actor	Admin
Summary	Admin can manage the system agents.
Pre-condition	Admin must be authenticated.
Post condition	An agent can be added, disabled, updated, or changed its permissions.
Main scenario	1- Admin asks for the Agents interface. 2- The system prepares the list of all the registered agents. 3- Admin chooses to: 3.1- Add a new agent by filling the form. 3.2- Update a selected agent info where the admin can update its personal info or change its password. 3.3- Change the permission given to the chosen agent by adding new ones or deleting existing ones.
Alternative scenario	- The system won't save the input values if there is a missing or invalid field.

Table 3.4: Text description of the Manage agents use case

3.1.2.5 Text description of the Manage admins use case

The use case Manage admins is illustrated in detail in the following table.

Use case	Manage admins
Main Actor	Webmaster
Summary	Webmaster can manage the system admins.
Pre-condition	Web master must be authenticated.
Post condition	An admin can be added, disabled, or modified.
Main scenario	1- Webmaster asks for the Admins interface. 2- The system prepares the list of all the registered admins. 3- Webmaster chooses to: 3.1- Add a new admin by filling the form. 3.2- Update a selected admin info where the webmaster can update its personal info or change its password.
Alternative scenario	- The system won't save the inputs if there is a missing or invalid field.

Table 3.5: Text description of the Manage admins use case

3.1.2.6 Text description of the Update Role Permissions use case

The use case **Update Role Permissions** is a critical functionality that allows a Super Admin to manage the permissions associated with specific roles within the system. This process is essential for maintaining the security and functionality of the system by ensuring that each role has the appropriate permissions to perform its designated tasks.

Use case	Update Role Permissions
Main Actor	Super Admin
Summary	Super Admin can update the permissions associated with a specific role.
Pre-condition	Super Admin must be authenticated and have the necessary permissions to modify role permissions.
Post condition	A role's permissions can be updated by adding new permissions or removing existing ones.
Main scenario	1- Super Admin navigates to the Roles interface. 2- The system displays a list of all roles and their current permissions. 3- Super Admin selects a role to update. 4- The system displays the current permissions for the selected role. 5- Super Admin chooses to: 5.1- Add a new permission by selecting from a list of available permissions. 5.2- Remove an existing permission by deselecting it from the list. 6- Super Admin submits the changes. 7- The system saves the changes and redirects the Super Admin to the all roles page.
Alternative scenario	- The system won't save the changes if there is a missing or invalid selection.

Table 3.6: Text description of the Update Role Permissions use case

3.2 Conception

In this section, we'll present the detailed sequence diagram for the functionalities presented early.

3.2.1 Detailed system sequence diagram

he sequence diagram is an object interaction diagram that focuses on chronologically ordering messages while the system is operating.

3.2.1.1 System sequence diagram of the Sign up case

Each new client must fill out a registration form, if data is missing or doesn't match the required field the system displays an error message.

The diagram of Sign up system sequences is depicted in this figure.

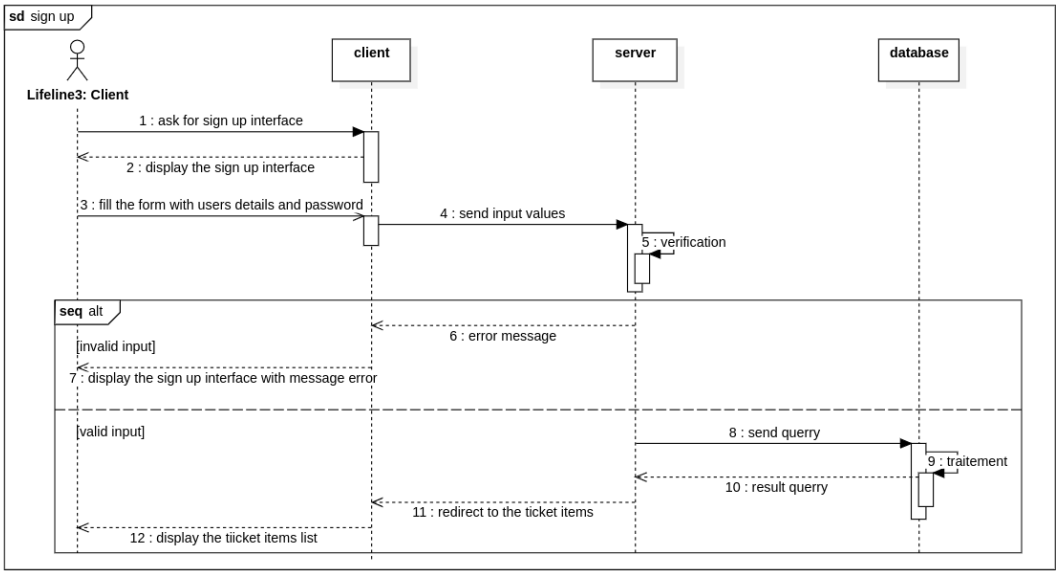


Figure 3.2: System sequence diagram of the Sign up

3.2.1.2 System sequence diagram of the Authentication case

This flowing graphic demonstrates the Authentication system sequences.

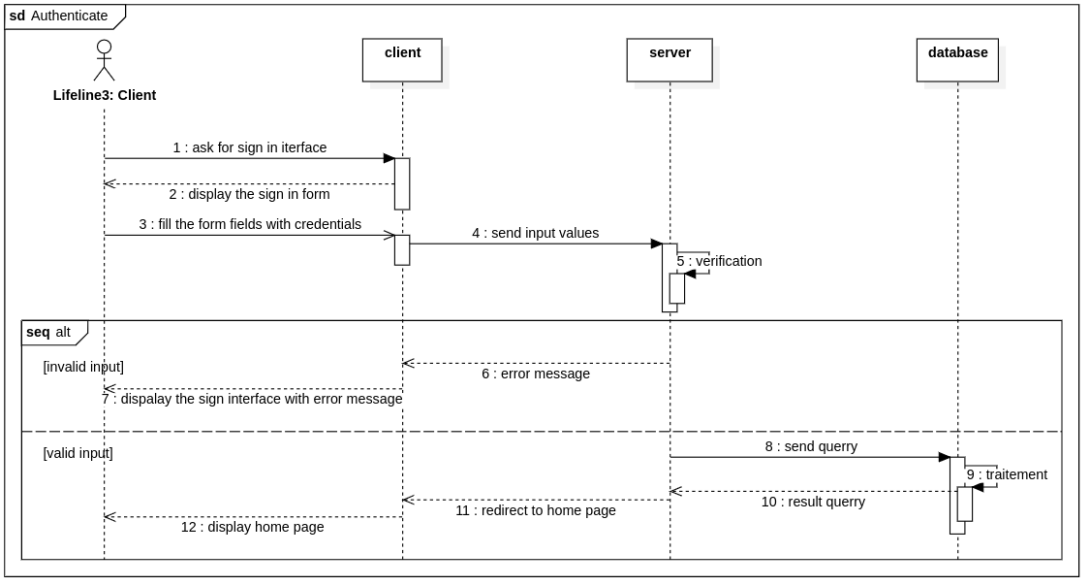


Figure 3.3: System sequence diagram of the Authentication

3.2.1.3 System sequence diagram of the update agent case

The admin has the ability to change an agent's information in order to do so he chooses an agent from agents list that appears to him then he chooses the update info, a form will be shown where he can choose to change personal info or change the agent's password. The system displays an error message, if data is empty or don't match the required type.

The system sequences for Update agent info are shown in this diagram.

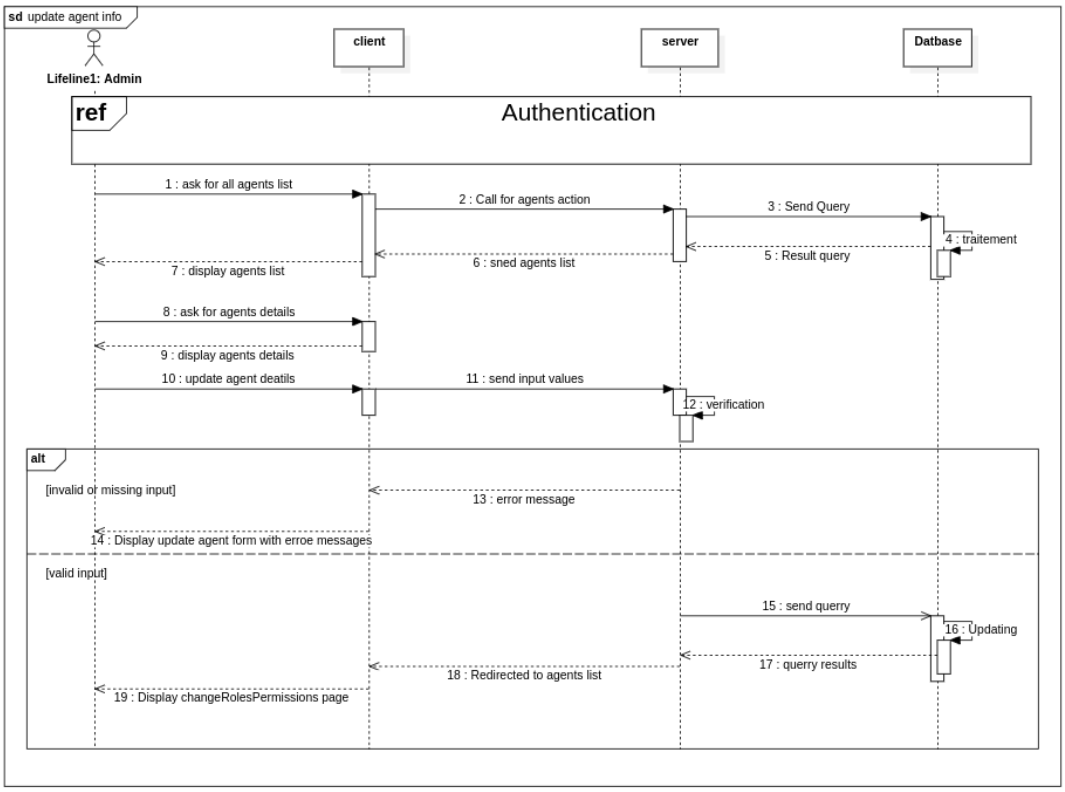


Figure 3.4: System sequence diagram of the update agent

3.2.1.4 System sequence diagram of the Change role permission case

The Change agent permission system sequences are represented in this diagram.

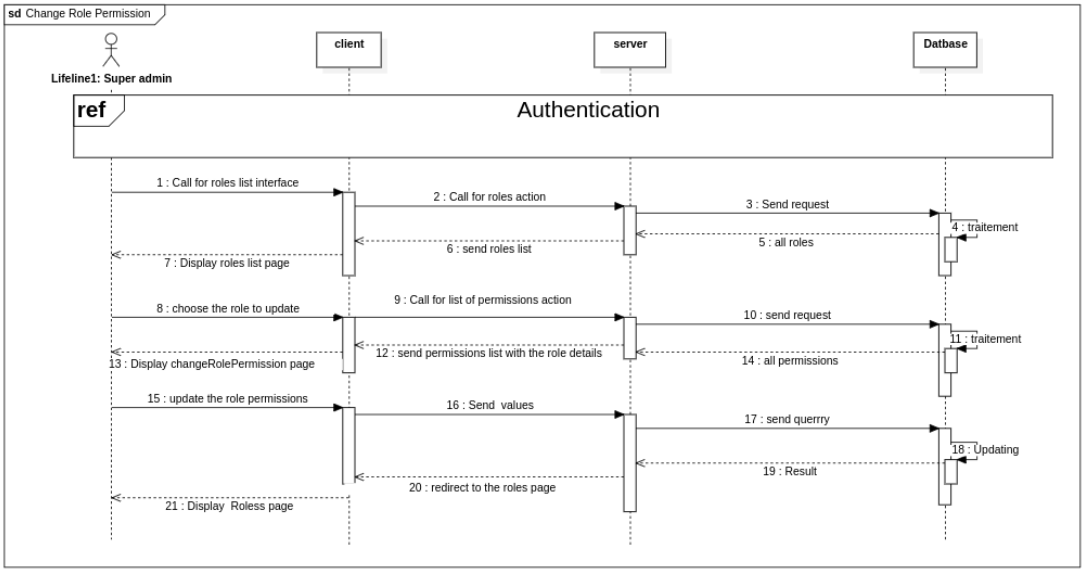


Figure 3.5: System sequence diagram of the Change role permission

3.2.2 Global class diagram of the first Sprint

The final class diagram for the first Sprint is shown in the diagram below.

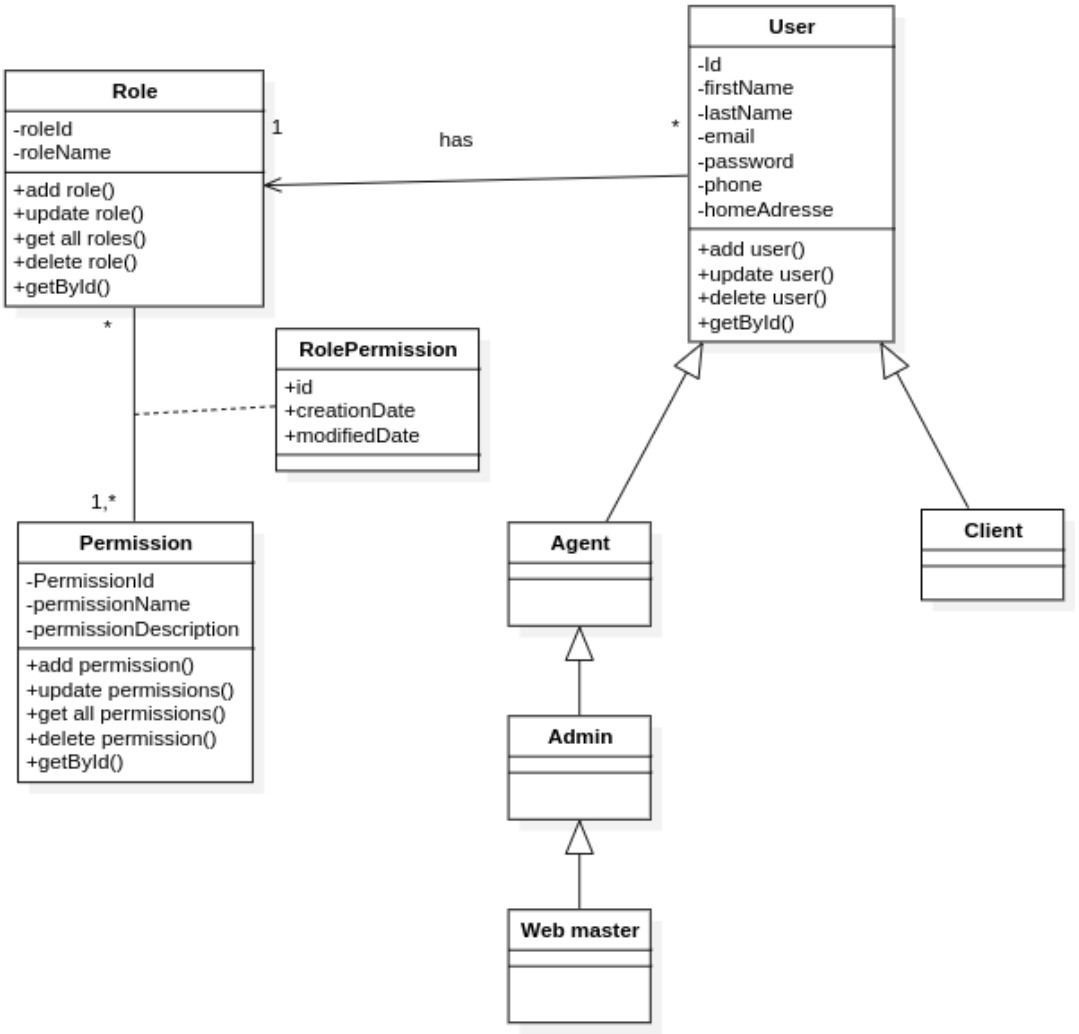


Figure 3.6: Users management class diagram

3.2.3 Detailed sequence diagram

The sequence diagram is an object interaction diagram that focuses on chronologically ordering interactions between objects.

3.2.3.1 Sequence diagram of update profile case

The update profile sequences are represented in this diagram.

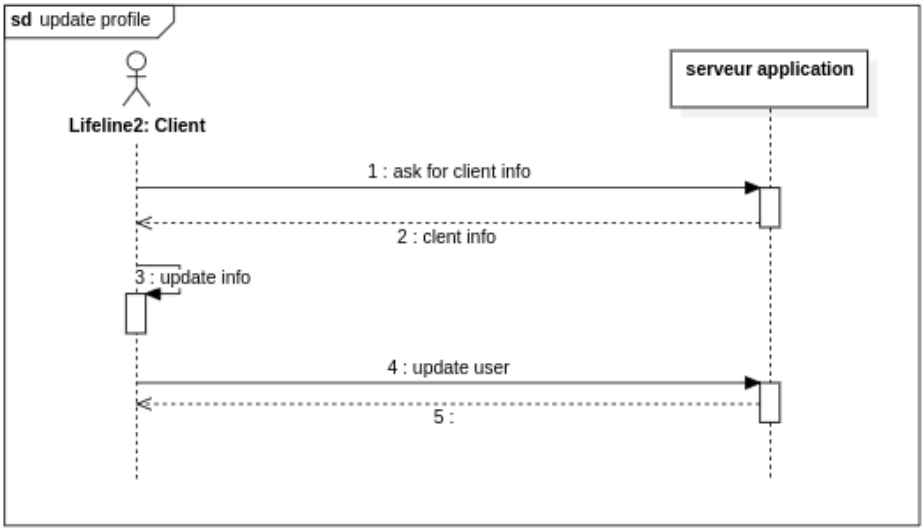


Figure 3.7: Update profile sequence diagram

3.2.3.2 Sequence diagram of Add permission to role case

The Add permission to role sequences are represented in the diagram 3.8 .

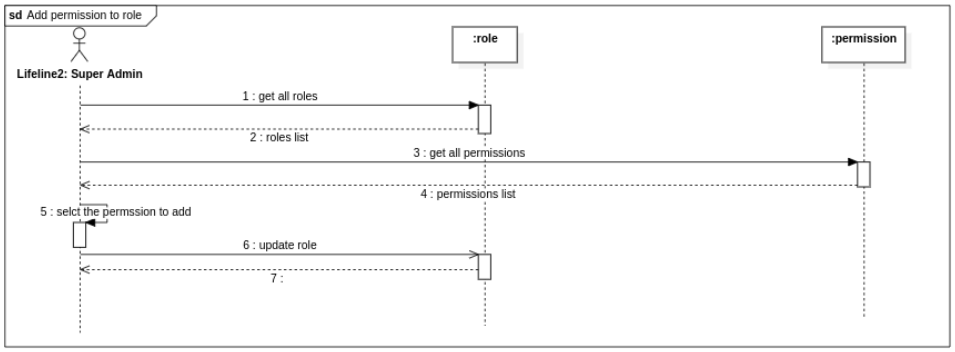


Figure 3.8: Add permission to role sequence diagram

3.3 Realization

To demonstrate our implementation's outcomes, we'll first present the database structure, thenwe will display some screenshots of the application's graphical interfaces.

3.3.1 First sprint scrum table

the next figure shows our Scrum board status and the work in progress.

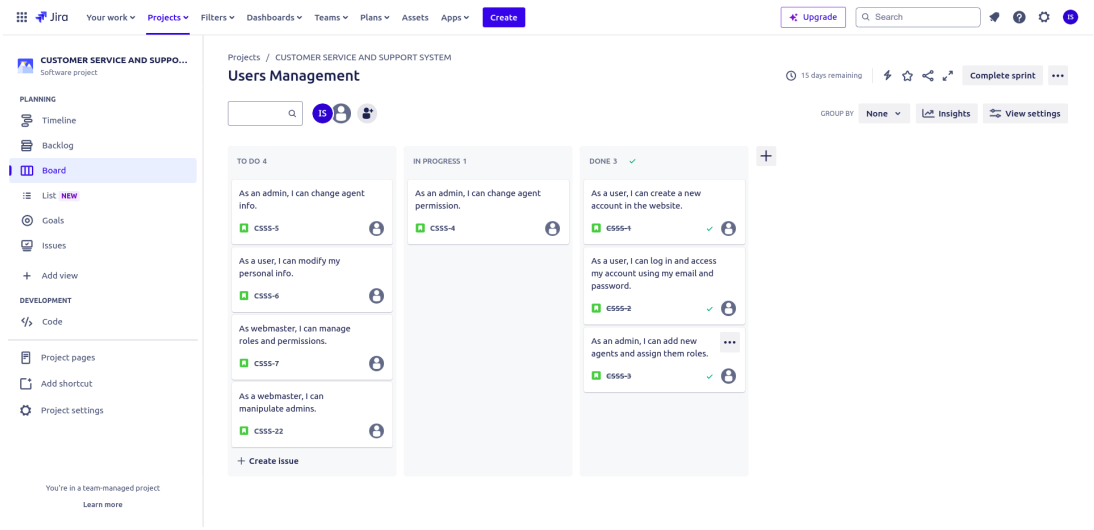


Figure 3.9: First spring scrum board

3.3.2 Coding

The coding phase, often known as development, is responsible for putting the User stories that were studied and planned in the preceding two stages into action. We'll provide the database structure for this first increment in this part.

This figure shows the users table structure in our application.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	Field	Type	Null	Key	Default	Extra
1	id	int	NO	PRI	NULL	auto_increment
2	firstName	varchar(255)	NO		NULL	
3	lastName	varchar(255)	NO		NULL	
4	email	varchar(255)	NO	UNI	NULL	
5	password	varchar(255)	NO		NULL	
6	homeAddress	varchar(255)	NO		NULL	
7	createdAt	datetime	NO		NULL	
8	updatedAt	datetime	NO		NULL	
9	RoleId	int	YES	MUL	NULL	

Figure 3.10: Users table schema screenshot

Roles table structure is shown in the figure below.

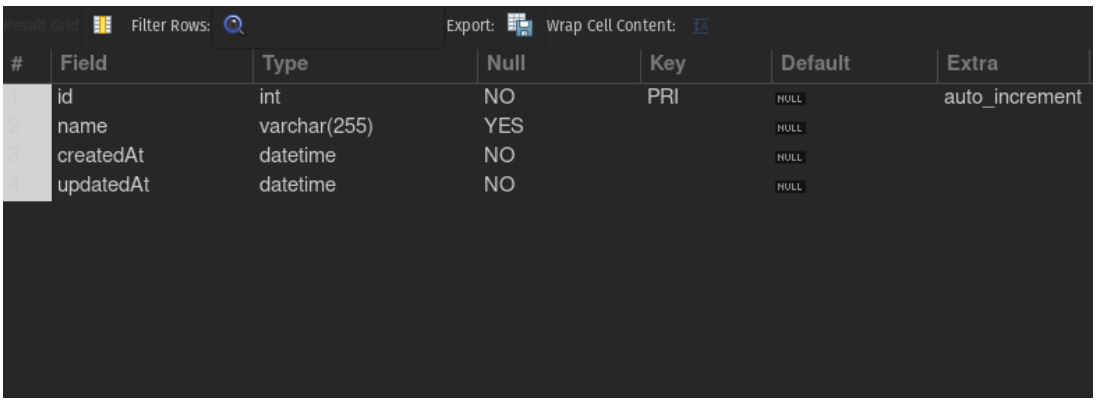


Figure 3.11: Roles table schema screenshot

All the system permissions is saved in permission table which this figure describes

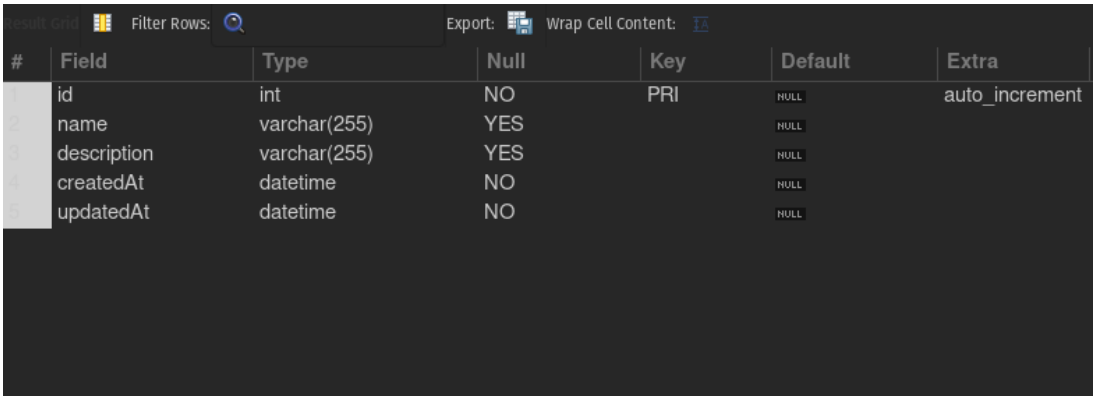


Figure 3.12: Permissions table schema screenshot

The table RolePermissions table save the permissions of each role.

Figure 3.13: RolePermissions table schema screenshot

3.3.3 The interfaces

Now lets see some interfaces of the first sprint:

The registration form for our app's users is depicted in the interface below:

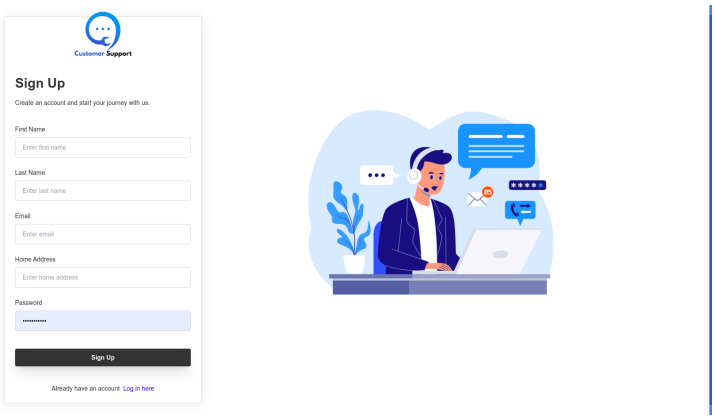


Figure 3.14: Sign up interface screenshot

The log in fields of the system is shown in the interface below.

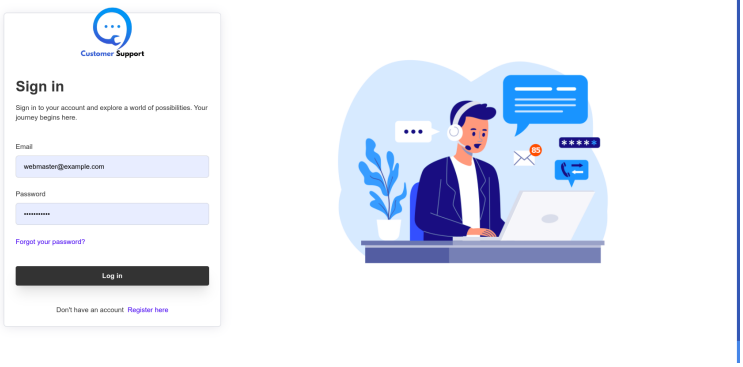


Figure 3.15: Sign in iterface screenshot

All users interface is shown in the interface 3.16.

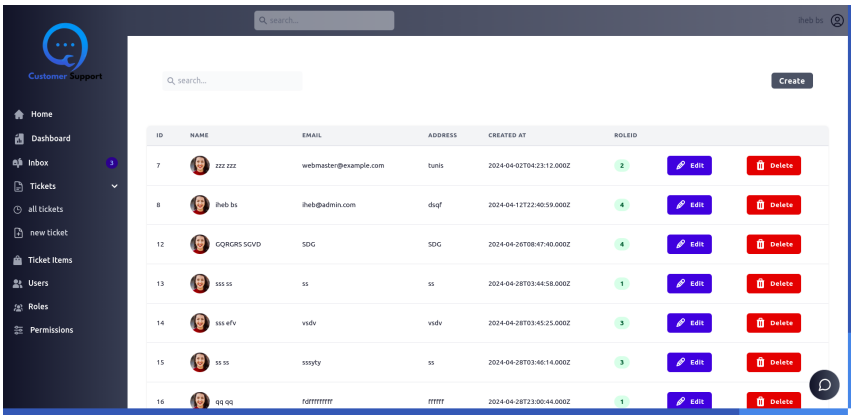


Figure 3.16: Users interface screenshot

The edit Role interface is shown below.

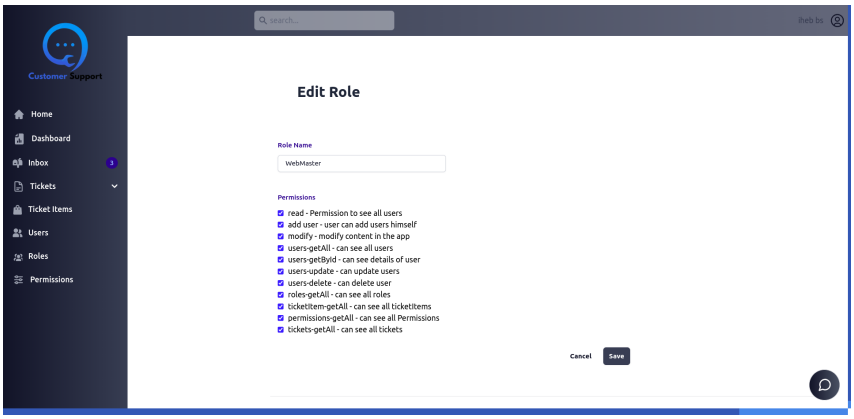


Figure 3.17: Edit Role screenshot

When a A super admin try to delete a role he should confim the action in the interface below.

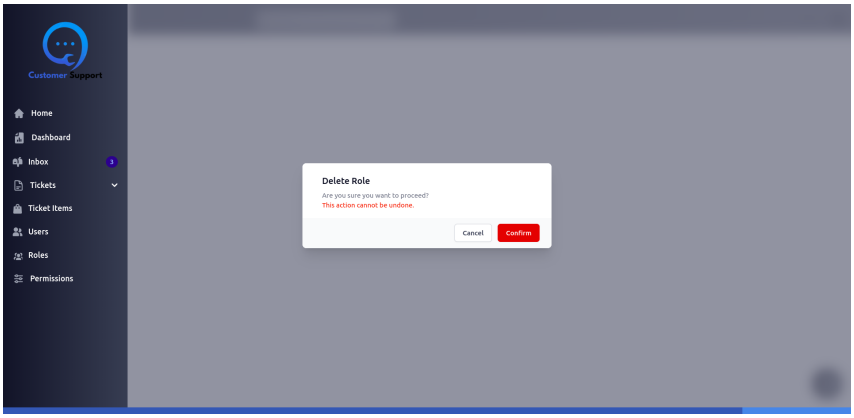


Figure 3.18: RolePermissions table schema screenshot

Below the profile edit interface for a user.

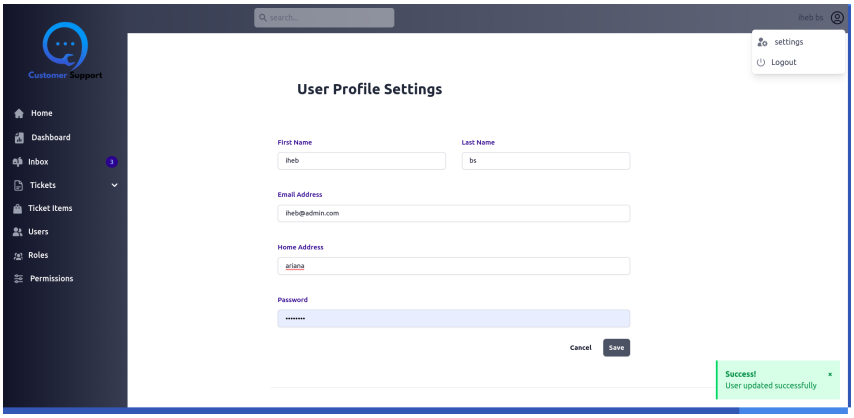


Figure 3.19: RolePermissions table schema screenshot

conclusion

In this chapter, we have managed to evaluate, create, and develop the first sprint of our customer support project up to this point. Our efforts in the next section will be focused onachieving the second sprint.

Chapter 4

Sprint 2: System Management

Introduction

In the previous chapter, we presented the first sprint which is based on authentication, agent management, admin management and role control. In this chapter we will continue our work and begin the second sprint which have Ticket item manipulation, Ticket manipulation and functionalities.

4.1 Functional specifications

4.1.1 Use case diagram

The primary use case diagram for second sprint is represented in this diagram:

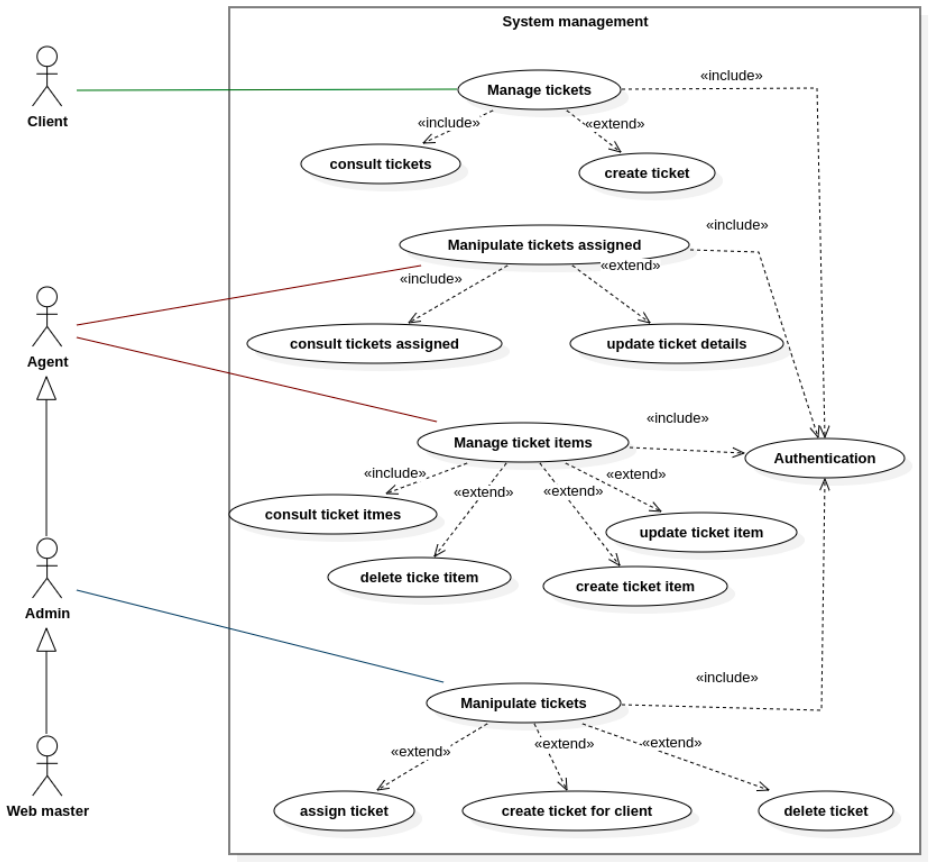


Figure 4.1: System management use case diagram

4.1.2 Description of use cases

We improved the use cases to include an overview of the various scenarios that might occur.

4.1.2.1 Text description of The Manipulate ticket item case

The use case Manipulate ticket item use case is illustrated in detail in the following table.

Use case	Manipulate ticket item
Main Actor	Agent/Admin/Super Admin
Summary	Every agent with permission, admin, or super admin can manipulate the ticket items.
Pre-condition	Agent, admin, or super admin must be authenticated in the system.
Post condition	-
Main scenario	1- System displays a list of ticket items. 2- Agent, admin, or super admin can choose to perform a specific operation concerning ticket items: 2.1- Add ticket item by filling the form. 2.2- Update ticket item information by updating the field values and submitting the new information. 2.3- Delete a ticket item after submitting the confirmation dialogue appears. 2.4- Display ticket item detailed information. 3- Agent, admin, or super admin can look for specific ticket items in the search field.
Alternative scenario	- The system won't save the new info if there is empty or invalid input. - The system resets the operation when the agent, admin, or super admin clicks on the "Cancel" button.

Table 4.1: Text description of the Manipulate ticket item use case

4.1.2.2 Text description of The Manage tickets case

The use case Manage tickets is illustrated in detail in the following table.

Use case	Manage tickets
Main Actor	Client
Summary	Every client can submit, reply, consult, or close his tickets.
Pre-condition	Client must be authenticated.
Post condition	A ticket is added or replied.
Main scenario	1- Client opens the "All Tickets" interface. 2- The system displays a list of all tickets sent by the client. 3- Client looks into one of the tickets or submits a new one by filling the form with necessary info. 4- The system displays the ticket details interface. 5- Client checks the replies from the back office.
Alternative scenario	- The system won't send the ticket if there is a missing or invalid field.

Table 4.2: Text description of the "Manage tickets" use case

4.1.2.3 Text description of The Manipulate tickets case

The use case Manipulate ticket is illustrated in detail in the following table.

Use case	Manipulate ticket
Main Actor	Admin/Super Admin
Summary	Every admin can get tickets to work on, consult ticket details, reply to tickets, or assign tickets to other agents.
Pre-condition	Admin or super admin must be authenticated in the system.
Post condition	-
Main scenario	1- System displays a list of all available tickets in the system. 2- Admin can choose tickets to add to his working list. 3- Admin replies to tickets from his working list. 4- Admin can assign tickets to agents.
Alternative scenario	- The system won't save the new info if there is empty or invalid input. - The system resets the operation when the agent, admin, or super admin clicks on the "Cancel" button.

Table 4.3: Text description of the Manipulate ticket use case

4.2 Conception

In this section, we'll present the detailed sequence diagram for the functionalities presented early.

4.2.1 Detailed system sequence diagram

4.2.1.1 System sequence diagram of the create ticket item case

Every agent can add a new ticket item to the system by filling the form with valid input.

The following graphic demonstrates the ticket item system sequences.

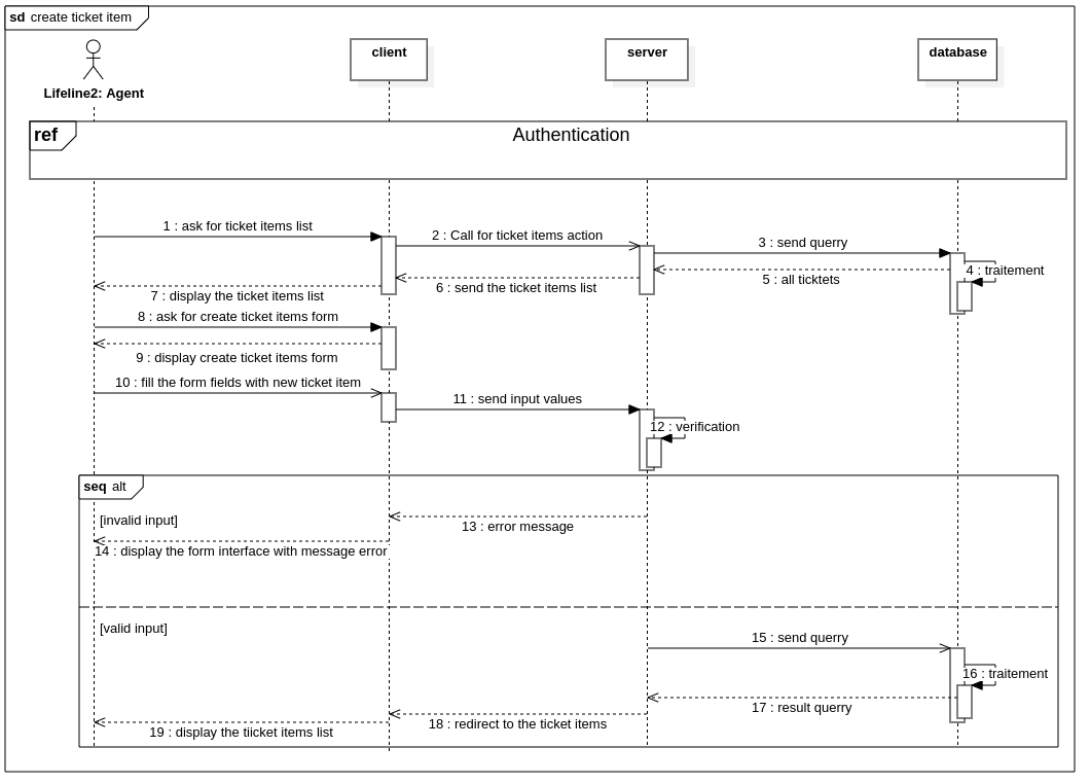


Figure 4.2: System sequence diagram of the create ticket item

4.2.1.2 System sequence diagram of the create a ticket case

To resolve an issue, any customer can file a ticket connected to a product or service that he purchased. After the customer has been authenticated, he is led to the submit ticket interface, where he selects a product or service from the list of items, and then fills out the rest of the form with the required information before submitting his ticket.

The create ticket system sequences are represented in this diagram.

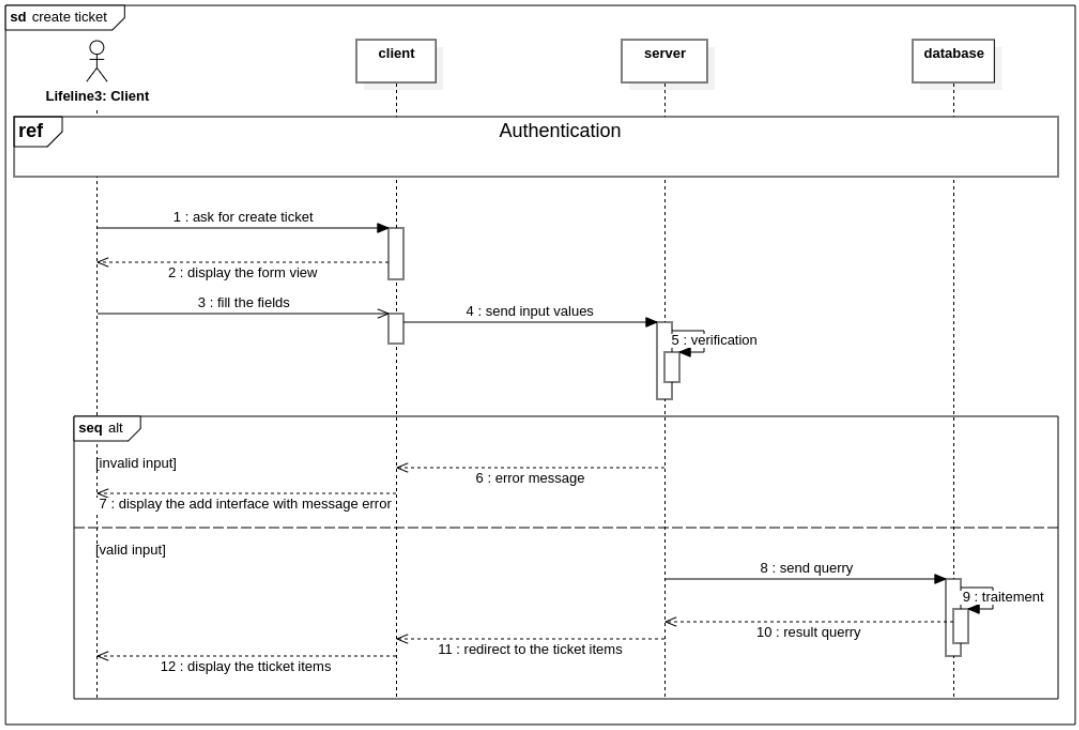


Figure 4.3: System sequence diagram of the create a ticket

4.2.1.3 System sequence diagram of the assign ticket case

An admin can assign tickets to agents by selecting the ticket information and selecting the assigned ticket action to access the assign ticket interface, where he can choose the agent to whom the ticket should be assigned.

This flowing graphic demonstrates the assign ticket system sequences.

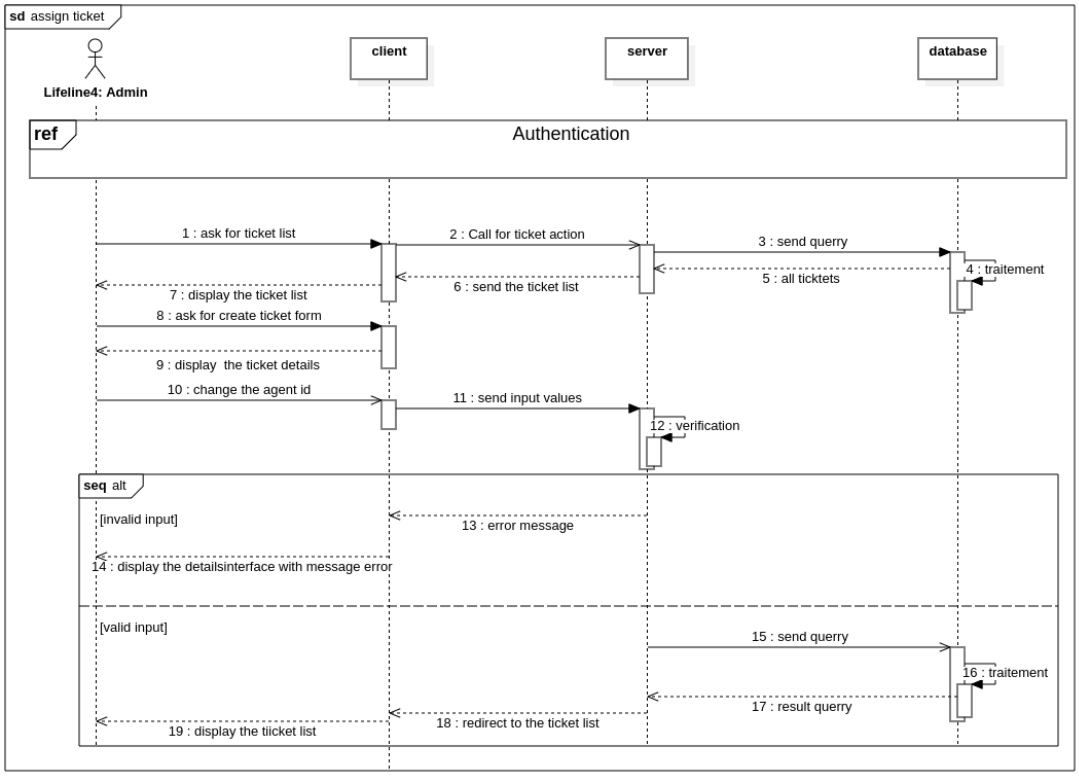


Figure 4.4: System sequence diagram of the assign ticket

4.2.1.4 System sequence diagram of the update a ticket case

After dealing with a ticket, an agent can update its state,priority or orther details to oragenize his work.

The update a ticket system sequences are represented in diagram below.

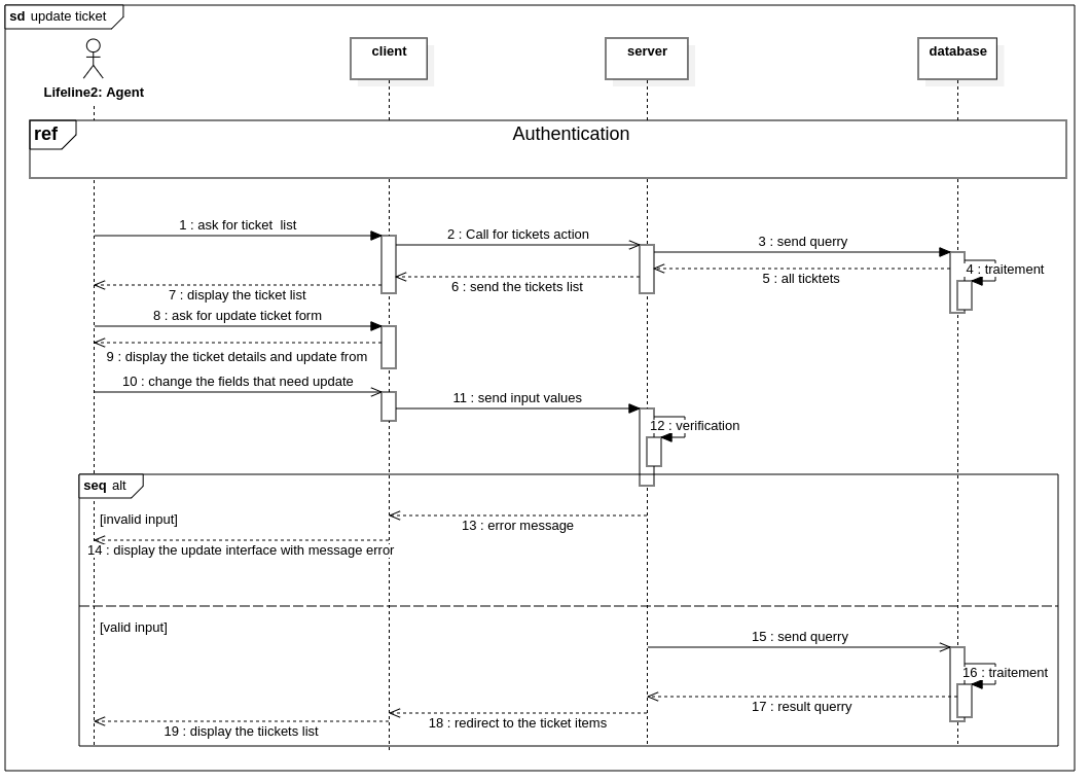


Figure 4.5: System sequence diagram of the update a ticket

4.2.2 Global class diagram of the second Increment

The diagram below shows the final class diagram for the second sprint.

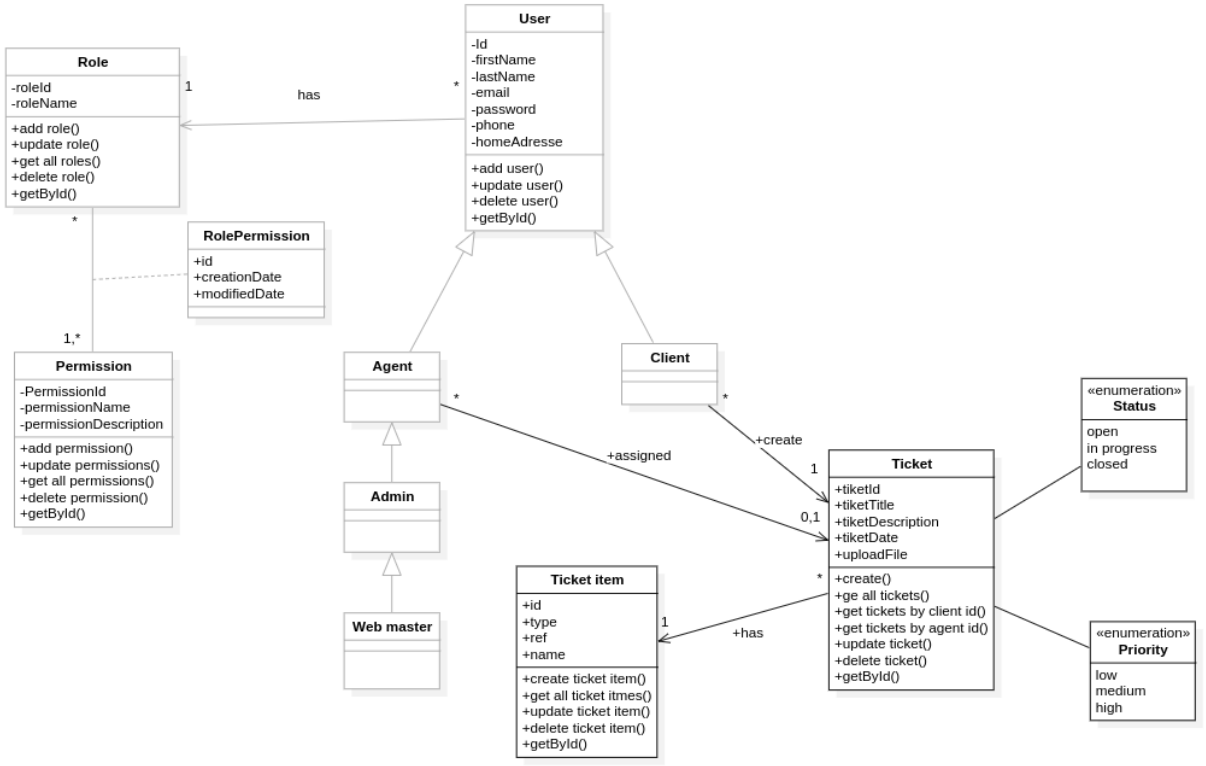


Figure 4.6: System management class diagram

4.2.3 Detailed sequence diagram

The sequence diagram is an object interaction diagram that focuses on chronologically ordering interactions between objects.

4.2.3.1 Sequence diagram of create ticket case

The create ticket sequences are represented in this diagram.

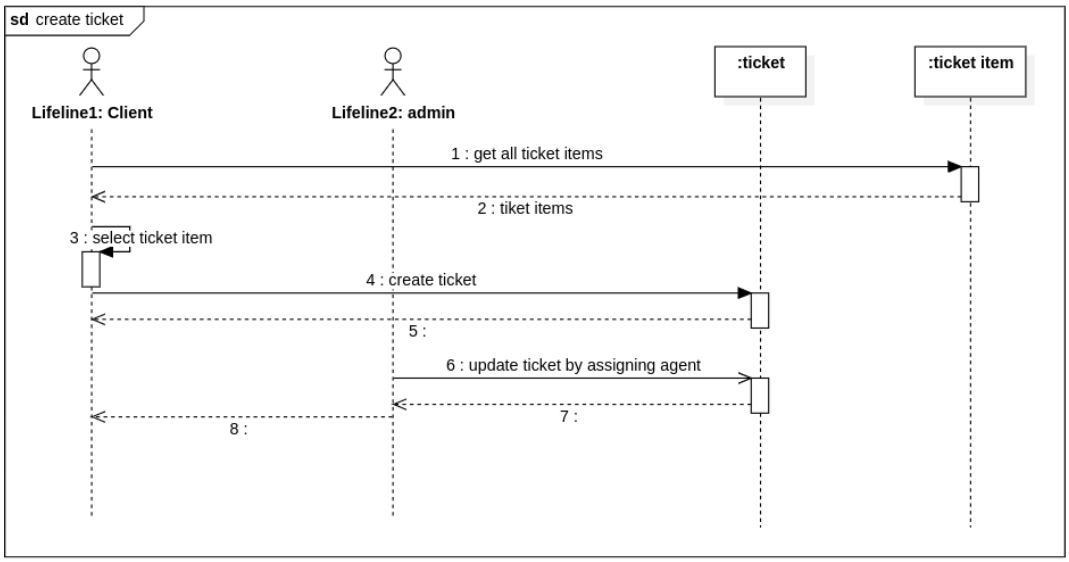


Figure 4.7: Create ticket sequence diagram

The diagram below shows how an admin can manipulate tickets.

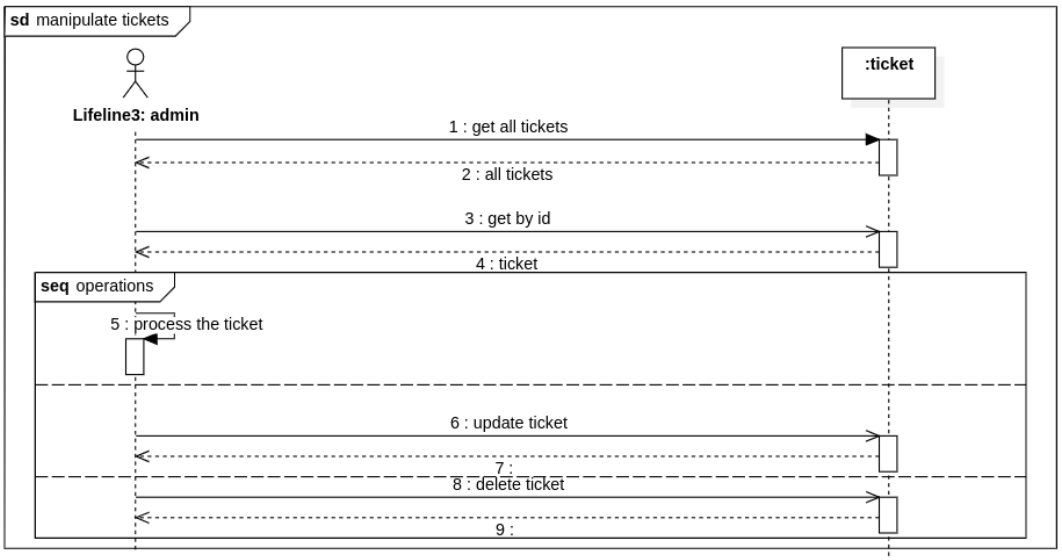


Figure 4.8: Manipulate tickets sequence diagram

The sequence diagram 4.9 shows the interaction between agent and ticket.

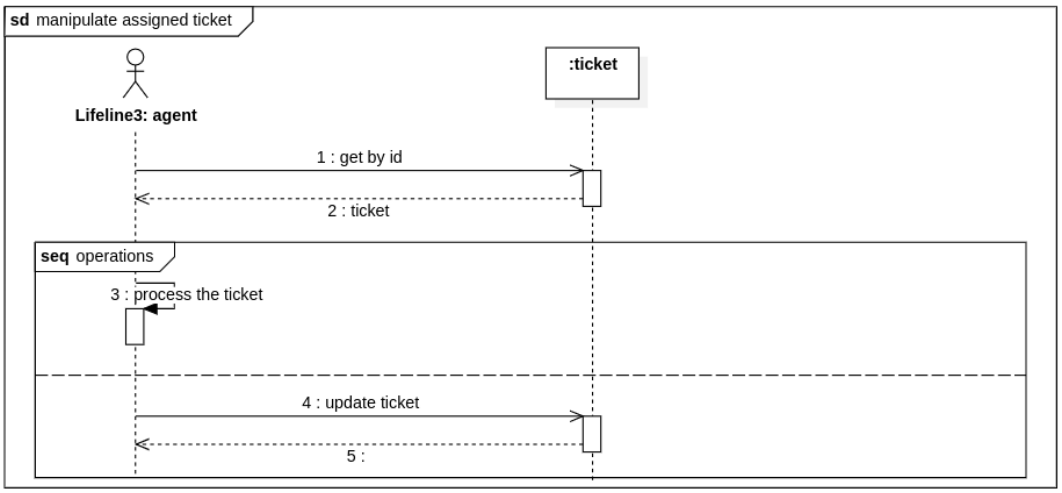


Figure 4.9: Manipulate assigned tickets sequence diagram

4.3 Realization

4.3.1 Second sprint scrum table

As we can see in the following figure, our Scrum board shows the create ticket is now finished, update ticket and delete ticket are now in the in progress column.

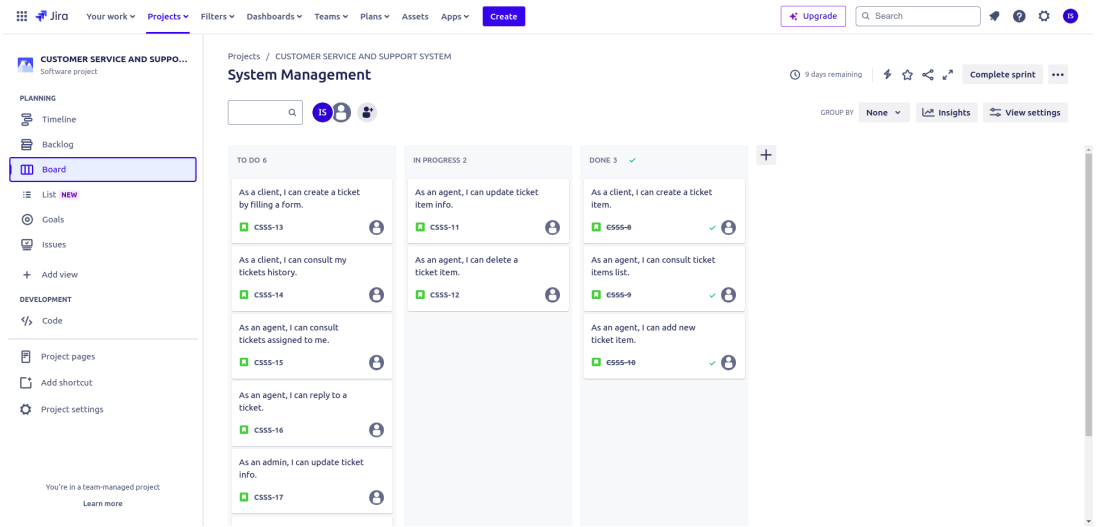
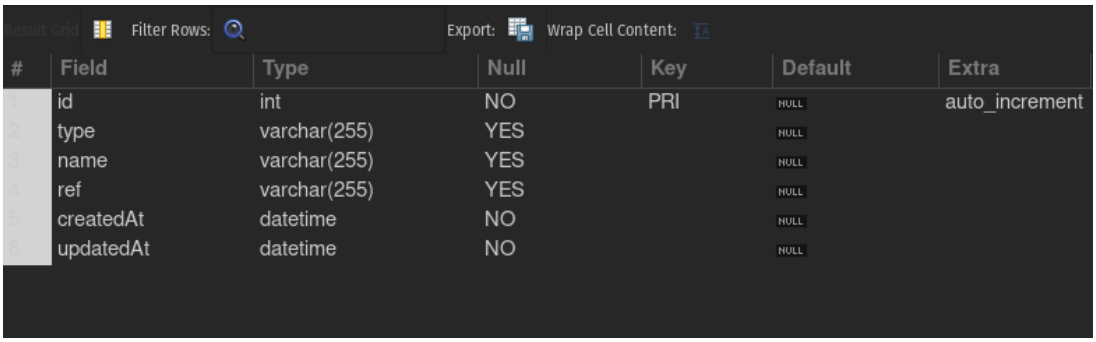


Figure 4.10: Second spring scrum board

4.3.2 Coding

All Tickets Items details and information are saved in the TicketsItems table. The structure of this table is represented on Figure below.



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	Field	Type	Null	Key	Default	Extra
1	id	int	NO	PRI	NULL	auto_increment
2	type	varchar(255)	YES		NULL	
3	name	varchar(255)	YES		NULL	
4	ref	varchar(255)	YES		NULL	
5	createdAt	datetime	NO		NULL	
6	updatedAt	datetime	NO		NULL	

Figure 4.11: TicketsItems table schema screenshot

The system saves all the tickets sent by client into table tickets.
4.12 represents thestructure of the tickets table.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	Field	Type	Null	Key	Default	Extra
1	id	int	NO	PRI	NULL	auto_increment
2	title	varchar(255)	YES		NULL	
3	description	varchar(255)	YES		NULL	
4	status	varchar(255)	YES		NULL	
5	priority	varchar(255)	YES		NULL	
6	agentId	int	YES		NULL	
7	date	datetime	YES		NULL	
8	uploadFile	varchar(255)	YES		NULL	
9	createdAt	datetime	NO		NULL	
10	updatedAt	datetime	NO		NULL	
11	TicketItemId	int	YES	MUL	NULL	
12	UserId	int	YES	MUL	NULL	

Figure 4.12: Tickets table schema screenshot

4.3.3 The interfaces

The screenshot below show the Create ticket interface.

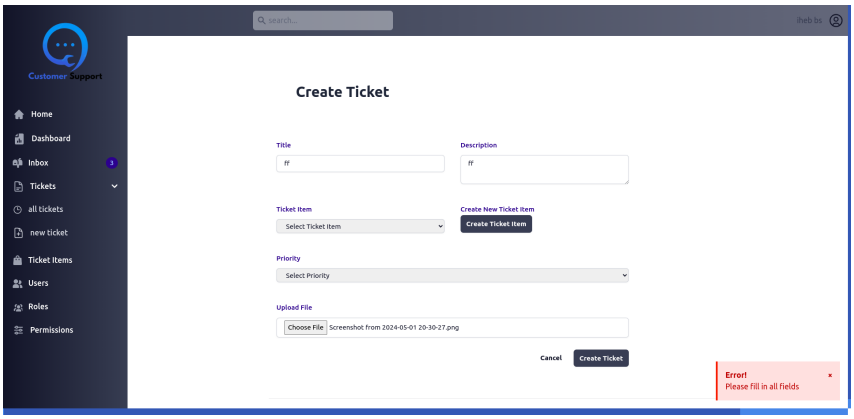


Figure 4.13: Create ticket interface screenshot

The interface below is showing the tickets list.

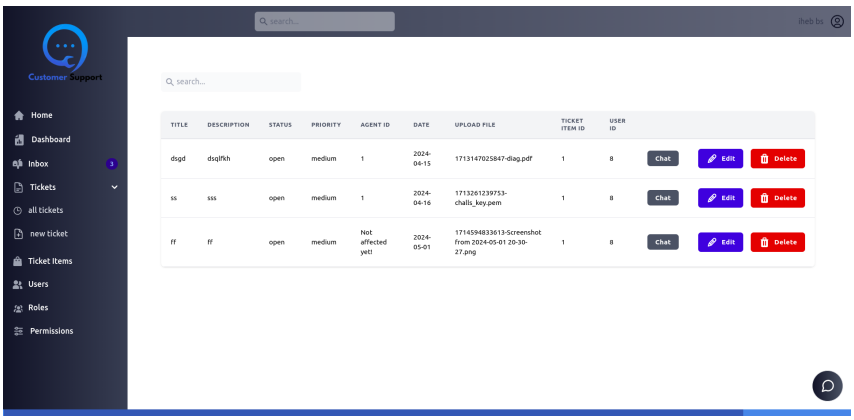


Figure 4.14: Tickets table schema screenshot

The figure below show the create ticket item interface.

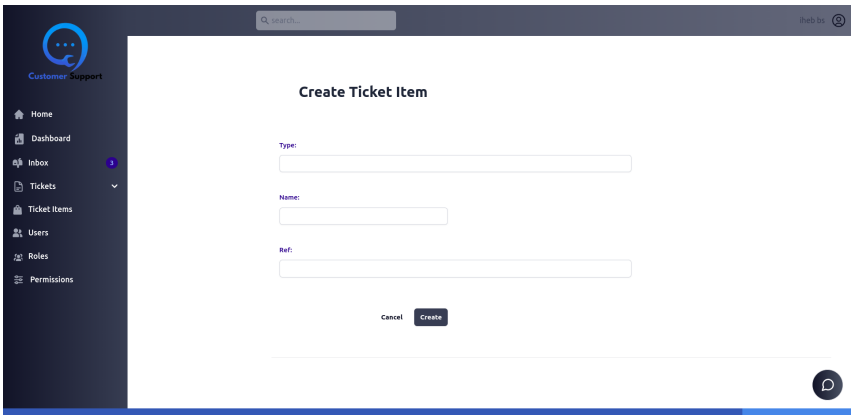


Figure 4.15: Tickets table schema screenshot

conclusion

We evaluated, created, and developed the second sprint of our project in this chapter, which included the system's most critical functionalities and took a lot of effort to produce.

Chapter 5

Sprint 3: Chatbot and Users Interaction

Introduction

In the previous chapter, we have presented the second increment which is based on managing our system main entities so it handled: ticket item Manipulation, Ticket Manipulation and Client manipulation functionalities. In this chapter, we will proceed to analyze and develop the last functionalities of our system which are: chatbot manipulation, Manage dashboard, and chat.

5.1 Functional specifications

5.1.1 Use case diagram

This diagram in figure 5.1 shows the principal use case diagram for the last sprint.

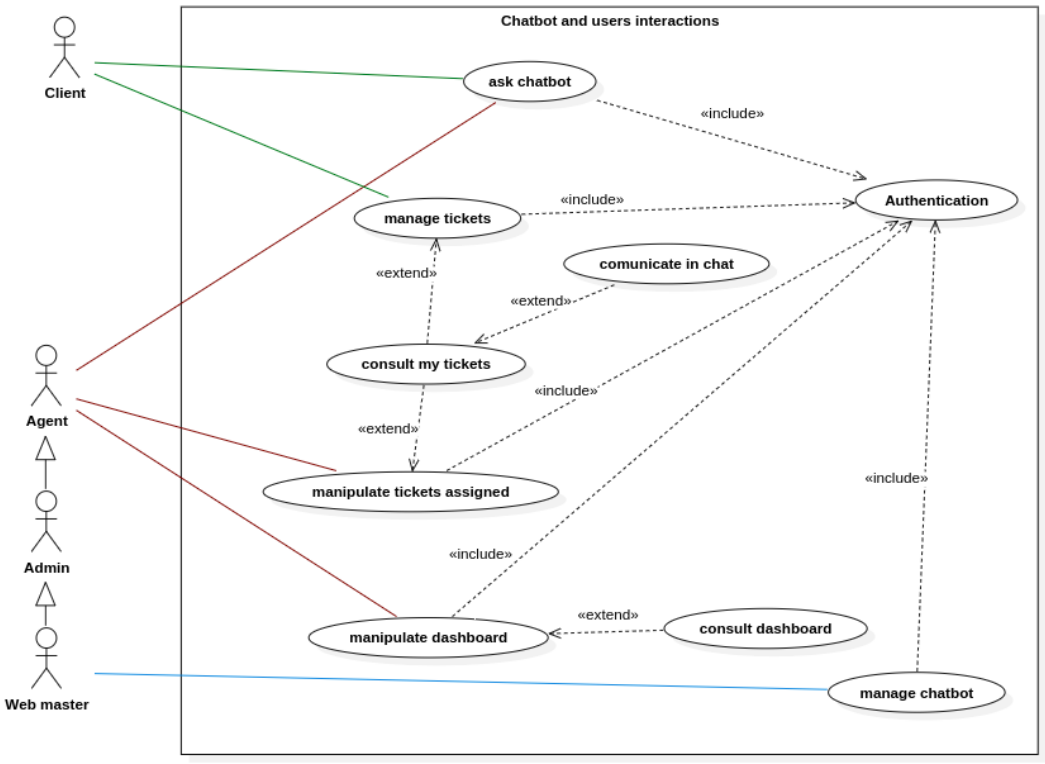


Figure 5.1: Chatbot and users interactions use case diagram

5.1.2 Description of use cases

5.1.2.1 Text description of the ask chatbot use case

The use case ask chatbot is detailed in detail in the table below

Use case	Ask chatbot
Main Actor	Client/Agent
Summary	Client or agent can ask the chatbot to get responses from the knowledge base.
Pre-condition	Client/agent must be authenticated.
Post condition	-
Main scenario	1- Client/agent asks for the chatbot interface. 2- Receive the response from the chatbot. 3- Interact with the response.
Alternative scenario	-

Table 5.1: Text description of the Ask chatbot use case

5.1.2.2 Text description of the Manage dashboard use case

The use case Manage dashboard is represented in detail in the table below.

Use case	Manage dashboard
Main Actor	Agent/Admin/Super Admin
Summary	Every agent, admin, or super admin can consult the system dashboard.
Pre-condition	Agent, admin, or super admin must be authenticated in the system.
Post condition	-
Main scenario	- Agent asks for the Dashboard interface. - The system gets the necessary data. - The system displays charts with the statistics information about tickets.
Alternative scenario	-

Table 5.2: Text description of the "Manage dashboard" use case

5.1.2.3 Text description of the Manage chatbot use case

The use case Manage chatbot is illustrated in detail in the following table.

Use case	Manage chatbot
Main Actor	Super Admin
Summary	Super admin is able to fine-tune the AI model.
Pre-condition	Super admin must be authenticated in the system.
Post condition	The dataset is ready in a specific format.
Main scenario	- Super admin asks for the Manage chatbot interface. - The super admin feeds the system with dataset. - The system gets the necessary data. - Update the chatbot database.
Alternative scenario	-

Table 5.3: Text description of the "Manage chatbot" use case

5.2 Conception

5.2.1 Detailed system sequence diagram

5.2.1.1 System sequence diagram of the ask chatbot case

Every user to the application can consult the knowledge base by asking the chatbot. This flowing graphic demonstrates the Ask chatbot system sequences.

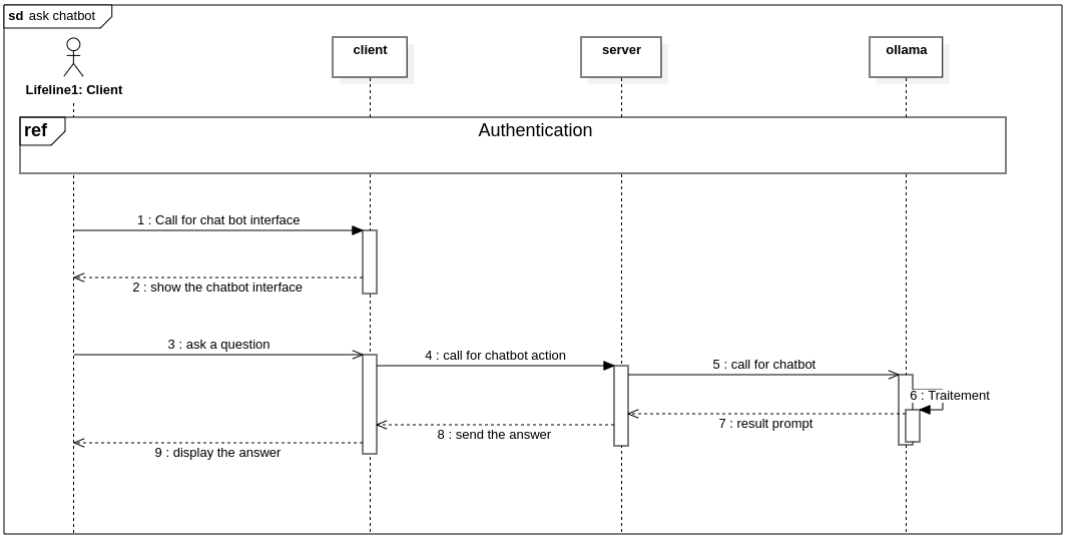


Figure 5.2: System sequence diagram of the ask chatbot

5.2.1.2 System sequence diagram of the reply to client in chat case

An agent can reply to the client question or problems in the chat assigned the the ticket to help him get answers or solutions for his problem.

The diagram of Reply to client system sequences is depicted in this figure below.

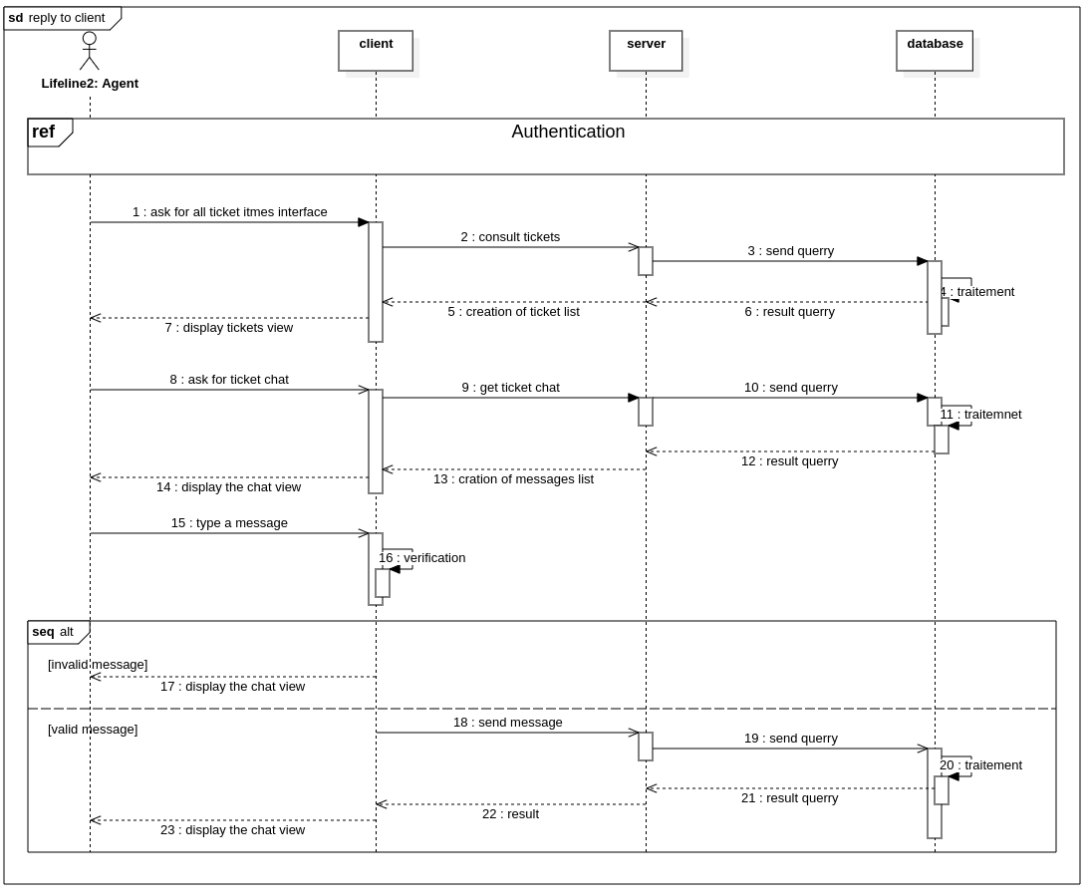


Figure 5.3: System sequence diagram of the reply to client in chat

5.2.1.3 System sequence diagram of the consult dashboard case

The back-office staff are allowed to consult the system dashboard and look into the application statistics.

This flowing graphic demonstrates the Consult dashboard system sequences.

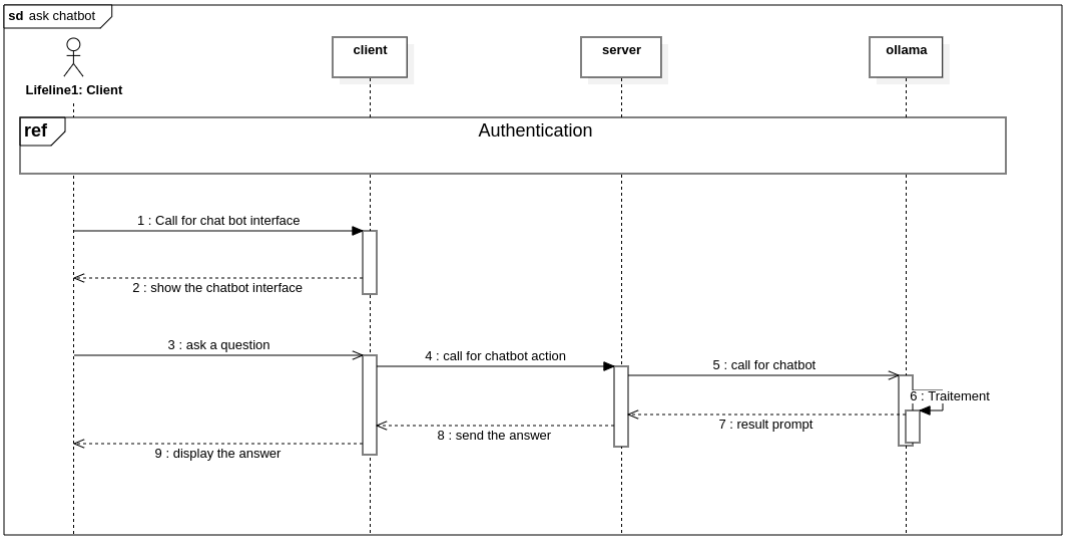


Figure 5.4: System sequence diagram of the consult dashboard

5.2.2 Global class diagram of the last sprint

The final class diagram for the last sprint is shown in the diagram below.

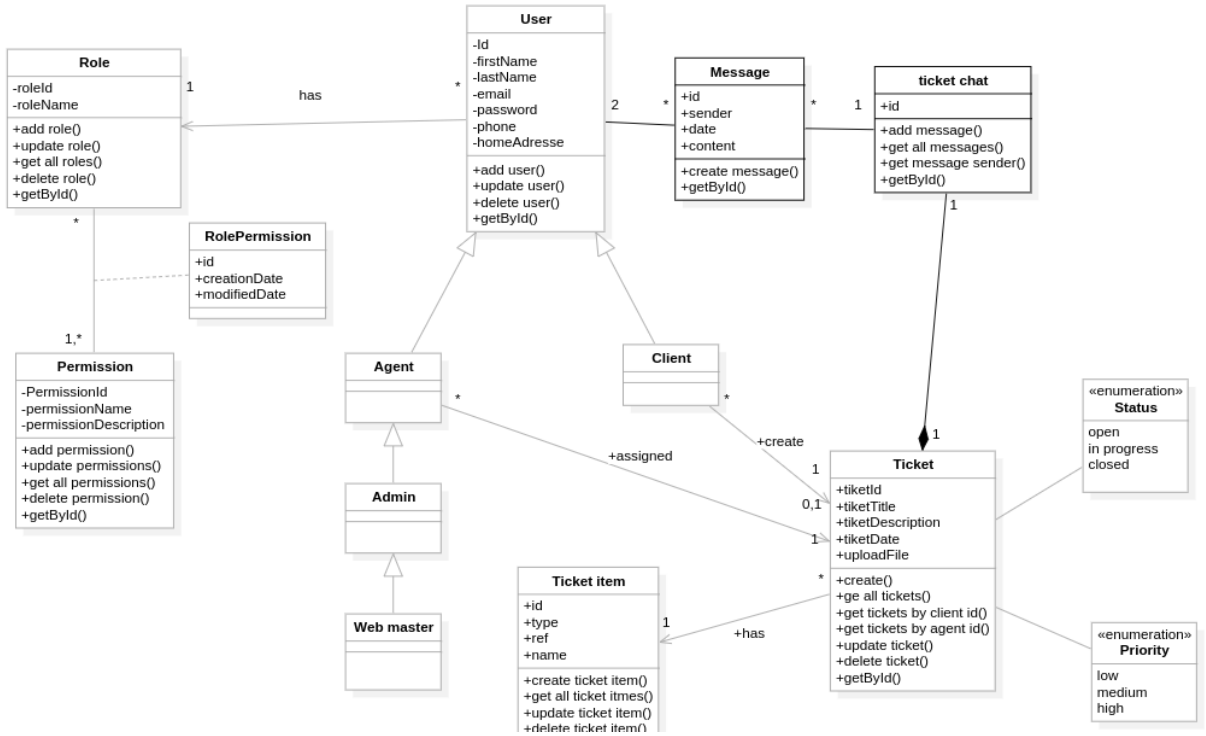


Figure 5.5: Sprint theree class diagram

5.2.3 Detailed sequence diagram

The sequence diagram is an object interaction diagram that focuses on chronologically ordering interactions between objects.

5.2.3.1 Sequence diagram of Reply to ticket case

The Reply to ticket sequences are represented in this diagram.

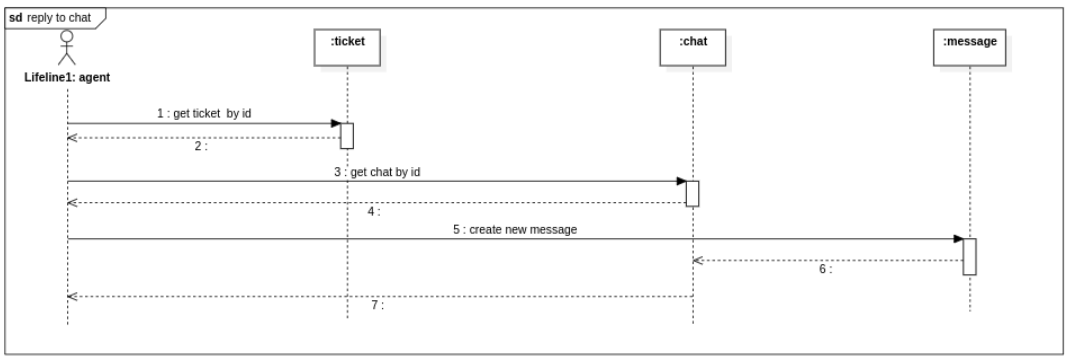


Figure 5.6: Reply to ticket sequence diagram

Below the sequence diagram of the consult dashboard.

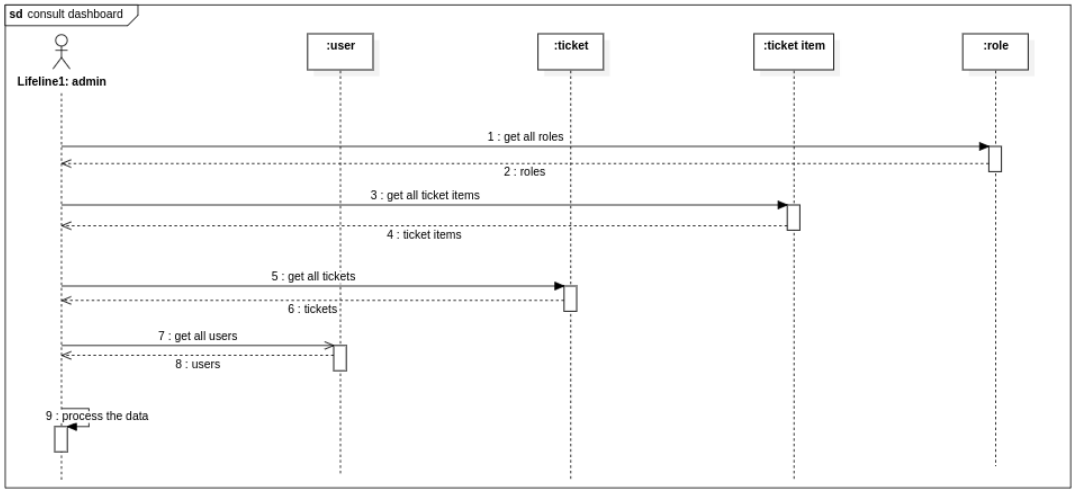


Figure 5.7: Consult dashboard sequence diagram

5.3 Realization

5.3.1 Second sprint scrum table

The figure below shows the Scrum board of our final sprint.

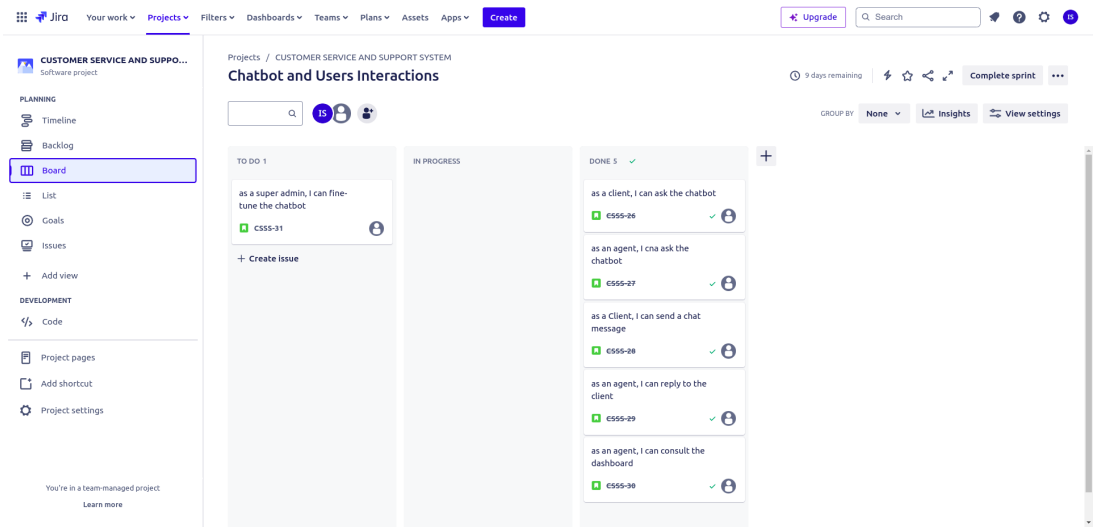


Figure 5.8: Last spring scrum board

5.3.2 Coding

for the chatId we used the ticket id to insure that every message is linked to its ticket. All Messages details and information are saved in the Messages table.

The structure of this table is represented on Figure below.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	Field	Type	Null	Key	Default	Extra
1	id	int	NO	PRI	NULL	auto_increment
2	message	text	NO		NULL	
3	createdAt	datetime	NO		NULL	
4	updatedAt	datetime	NO		NULL	
5	senderId	int	YES	MUL	NULL	
6	ticketId	int	YES	MUL	NULL	

Figure 5.9: Messages table schema screenshot

llama2

For the chatbot we've integrate the llama2 model wich is a collection of foundation language models ranging from 7B to 70B parameters. Llama 2 is released by Meta Platforms, Inc. This model is trained on 2 trillion tokens, and by default supports a context length of 4096.

This is a screenshot showin a lacal run of the model to test it.

Figure 5.10: Running llama2 with Ollama

Due to time constraints, we haven't been able to implement fine-tuning for the Llama2 model, which would enable the model to provide more specific responses regarding products or services.

5.3.3 The interfaces

The screenshot below show the Dashboard interface.

Figure 5.11: Dashboard interface screenshot

The Chatbot interface is shown in the interface 5.12.

Figure 5.12: chatbot interface screenshot

Below the screenshot of a chat between an agent and a client.

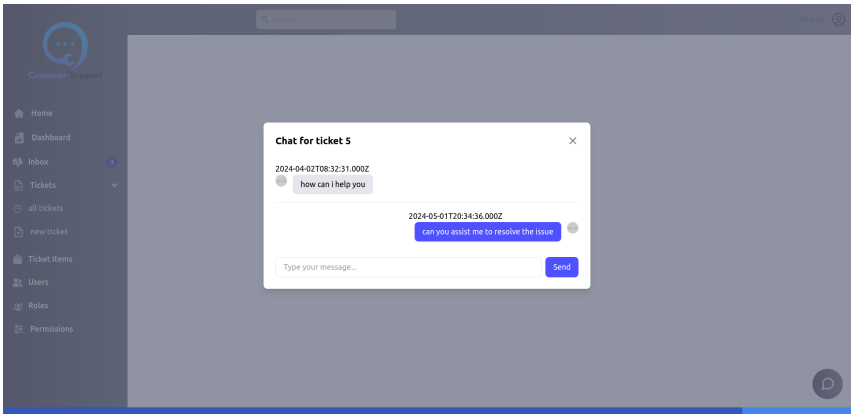


Figure 5.13: Agent-client chat interface screenshot

conclusion

At this level, we completed the third and final sprint of the project, in which we produced the following functionalities:Manage dashboard, Chatbot and Chat. This chapter marks the completion of our project development, which included the construction of a web application for client services and support.

General conclusion

In conclusion, the development of our Customer Support app has been a rewarding journey, offering us a chance to apply our theoretical knowledge practically. Through this project, we've not only built a help desk web application but also cultivated valuable skills in teamwork, problem-solving, and project management.

One of the most significant achievements of this project is the integration of various functionalities catering to both the front office for clients and the back office for agents and staff. From authentication and user management to ticket handling and product manipulation, our app provides a comprehensive solution for managing customer support efficiently.

Moreover, the implementation of a chatbot and consulting dashboard adds an extra layer of convenience and accessibility for both clients and support agents. These features streamline communication and empower users to find quick resolutions to their queries and concerns.

Looking ahead, we recognize the potential for further enhancements and refinements to our Customer Support app. Future iterations could focus on expanding the knowledge base, refining the user interface, and integrating additional communication channels for greater flexibility.

In conclusion, we are proud of the Customer Support app we have developed, and we believe it has the potential to make a positive impact in the realm of e-commerce customer service. We look forward to continuing our journey of learning and innovation, leveraging the experiences gained from this project to tackle new challenges and pursue future opportunities.

Bibliography

[1] Liberum Organization : Liberum - product presentation, 2024. Accessed on April 27, 2024.

[2] HelpDeskZ : Helpdeskz. Accessed on April 27, 2024.

[3] Spiceworks : Spiceworks free cloud help desk software, 2024. Accessed on April 27, 2024.

[4] Katak Support : Katak support. Accessed on April 27, 2024.

[5] StarUML : Staruml. Accessed on April 2024.

[6] Microsoft : Visual studio code. Accessed on April 2024.

[7] Oracle Corporation : Mysql workbench. Accessed on April 2024.

[8] Langchain Python Documentation : Ollama - llms and python, Year of publication. Accessed on April 28, 2024.

[9] Node.js Contributors : Node.js. Accessed on April 2024.

[10] Express.js Contributors : Express.js. Accessed on April 2024.

[11] Facebook : React. Accessed on April 2024.

[12] Tailwind CSS Authors : Tailwind css. Accessed on April 2024.