Private University
Horizon School of Digital Technologies
*****
N° agrément : 2020/02
*****

N° d'ordre :

End Of Studies Project Report

Presented in order to obtain the

National Bachelor's Degree in Computer Science

Major :

Software Engineering and Information Systems

By

Fedi Hmaidi

SellPoint Application

Defended on 09/10/2024 in front of the jury composed of :

Ms. : Nawel Bayar	**President**
Ms. : Houda Bechir	**Reporter**
Ms. : Leila Gazzah	**Academic Supervisor**
Mr. : Abderraouf ghrissi	**Professional Supervisor**

Academic Year: 2023 / 2024

DEDICATION

First and foremost, we would like to express our deepest gratitude to our supervisor, Mr. Mehrez Boulares, for his attentive guidance, exemplary professionalism, and the wealth of advice he provided throughout this project. His continuous commitment and availability were a true source of inspiration and played a major role in the success of our work.

We would also like to warmly thank all of our teachers for the knowledge and support they have given us throughout this academic year. Their guidance, patience, and passion have played a fundamental role in our personal and academic development.

Our sincerest thanks also go to the jury members for agreeing to evaluate our work with rigor, kindness, and professionalism. Their feedback is a valuable recognition for us and a source of motivation for our future endeavors.

Finally, we extend a heartfelt thank you to all the individuals who, directly or indirectly, contributed to the completion of this project. Their moral support, encouragement, and invaluable assistance were essential at every step of this journey.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABREVIATIONS

- **SQL**: Structured Query Language
- **API**: Application Programming Interface
- **MVC**: Model View Controller
- **HTTP**: Hypertext Transfer Protocol
- **2TUP**: 2 Tracks Unified Process
- **HTML**: Hypertext Markup Language
- **DB**: Database
- **UML**: Unified Modeling Language
- **CDM**: Conceptual Data Model
- **LDM**: Logical Data Model
- **REST**: Representational State Transfer
- **ORM**: Object Relational Mapping
- **JSON**: JavaScript Object Notation

INTRODUCTION

In today's digital age, software engineering plays a key role in transforming traditional ways of working by automating manual processes, improving collaboration, and making systems more centralized, flexible, and accessible. Whether in business, education, or administration, the aim is always to reduce friction, eliminate redundancy, and empower users with tools that simplify complex workflows.

Within the academic world, particularly in higher education institutions, the management of end-of-year (PFA) and end-of-studies (PFE) projects remains a major pain point. Despite being such a critical part of a student's academic journey, the current process is often disorganized and fragmented. Information is scattered across multiple channels—Excel spreadsheets, email threads, and even informal platforms like Messenger or WhatsApp. This lack of centralization creates confusion, miscommunication, and inefficiencies that affect both students and supervisors.

Our project was born out of the need to solve this issue through a modern, web-based platform that centralizes and automates the entire PFA/PFE management process. The platform is designed to serve as a digital workspace for students, supervisors, and jury members. It includes features for managing project details, assigning and tracking tasks, setting deadlines, creating student-supervisor pairings, and handling jury assignments.

Beyond task and user management, the platform also supports document uploads, project progress tracking, and dashboard views that give supervisors and administrators a bird's-eye view of all ongoing projects. This solution not only simplifies the administrative burden but also encourages clearer communication, more structured workflows, and ultimately better outcomes for final-year academic projects.

This report is organized into five chapters. In the first chapter, we introduce the study of existing tools and methods. We conclude the chapter by defining the methodology adopted for carrying out this project.

The second chapter presents the planning phase in detail. This includes the formation of the Scrum team, an overview of our preliminary analysis, definition of functional requirements, development environment setup, and detailed sprint planning.

The remaining chapters are each dedicated to an individual sprint. These chapters all follow a common structure: analysis of the sprint objectives, implementation of the planned features, and a realization section where we present the outcomes, challenges faced, and improvements made throughout each sprint.

CHAPTER 1

GENERAL FRAMEWORK OF THE PROJECT

1.1 Introduction

In this chapter, we will introduce our project by first studying its context. Then, we will elaborate on the problem analysis, analyze the existing solutions, present the proposed solution, and finally, explain the development methodology used.

1.2 Project Context

The management of university projects (PFA1, PFA2, PFE) within our National School of Engineering of Tunis (Ensit) represents a major challenge for administrators, teachers, and students. Indeed, with the increase in the number of projects and the diversity of departments (computer science, mechanical, electrical, etc.), it becomes essential to have a centralized tool to facilitate their organization, monitoring, and evaluation. In this context, the digitization of processes plays a key role in simplifying coordination between the different stakeholders: students can be assigned to specific projects and tasks, supervisors can effectively monitor their work, and jury members can easily access reports and evaluations.

As part of the second End-of-Year Project, we propose to develop a web application dedicated to the management of university projects. This solution will allow administrators to easily classify and manage projects by department, assign students to supervisors and juries, and track report deadlines. Teachers will find an optimized tool to supervise multiple projects simultaneously, while students will benefit from a structured environment to collaborate and view their tasks. By modernizing this process, our application aims to improve academic efficiency and simplify the lives of all stakeholders involved.

1.3 Study of Existing Solutions

The study of existing solutions is a fundamental step before designing our application.

It will allow us to analyze the solutions already implemented for the management of university projects (PFA/PFE), identify their strengths and limitations, and thus define the features that will add value to our tool.

In this section, we will present several existing platforms currently available on the market. Then, we will compare their key features to guide us towards an optimal solution tailored to the specific needs of ENSIT.

1.3.1 Existing Solutions

- **Trello.com [1]:** It is a project management tool based on the Kanban methodology, organized into boards, lists, and cards. It allows for clear task visualization and real-time collaboration among team members. The home interface of this platform is illustrated in Figure 1.1.

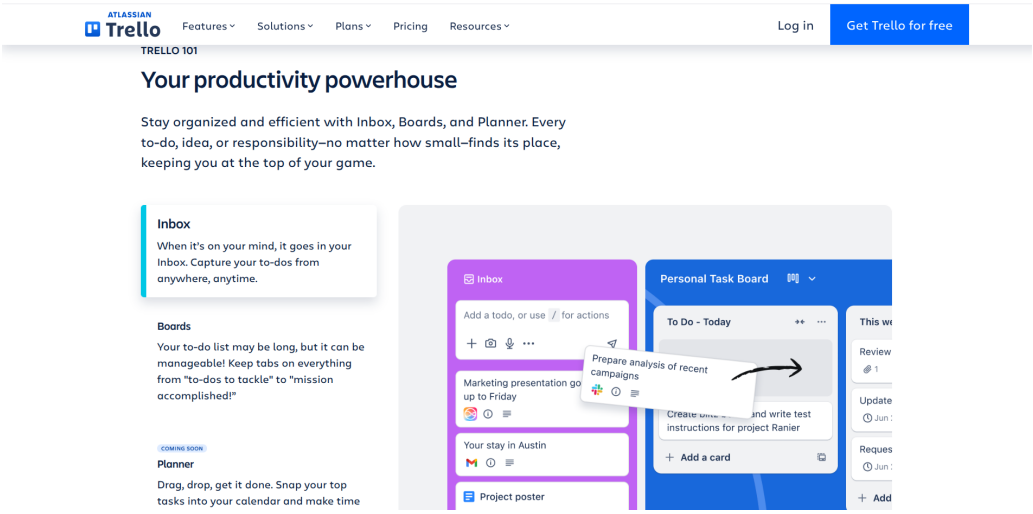


Figure 1.1: Home interface of Trello

- **Jira (Atlassian) [2]:** It is a professional project management platform initially designed for software development (Agile/Scrum methods). Figure 1.2 below presents the platform's home interface.

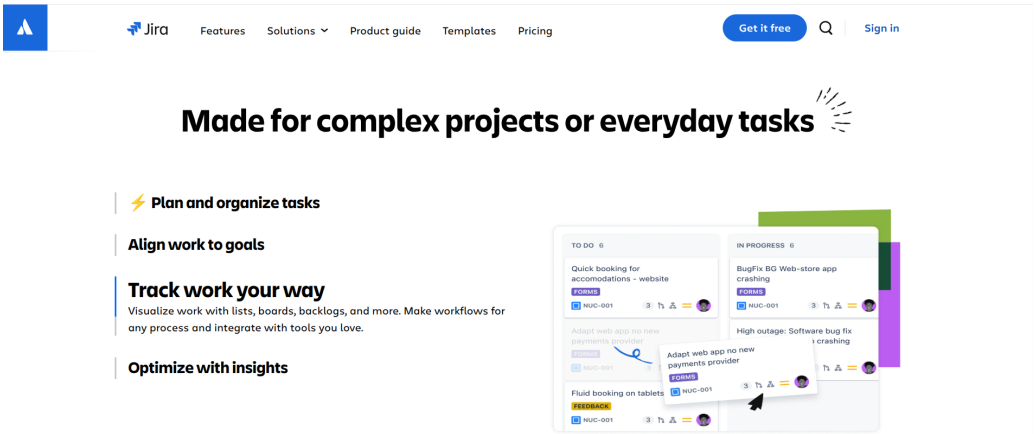


Figure 1.2: Home interface of Jira

- **Moodle [3]:** It is an open-source learning management system (LMS) widely used in educational institutions. The main interface of the platform is shown in Figure 1.3.

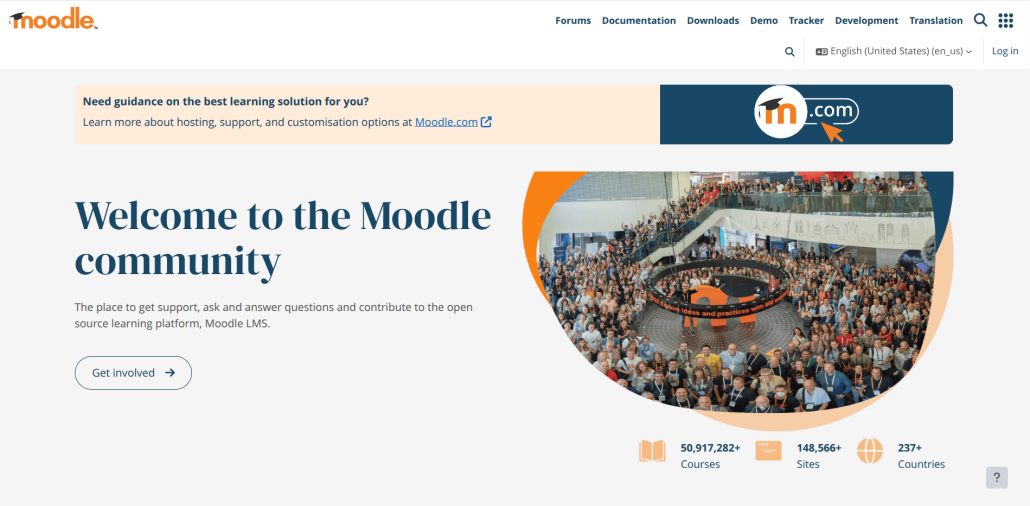


Figure 1.3: Main interface of Moodle

1.3.2 Limitations of Existing Project Management Tools for Academic Use

While Trello, Jira, and Moodle are widely used in professional and educational settings, they fall short in the context of managing university-specific academic projects, such as PFA (First-Year Projects) or PFE (Final-Year Projects). Below is a breakdown of the main limitations.

1. Not Designed for Academic Workflows

Trello and Jira were initially built for corporate teams, especially in software development, and not for managing academic projects led by students. Their design is centered around business workflows and professional team collaboration, which often doesn't align with the structure or requirements of student projects. As a result, essential academic features like report submission management, supervisor follow-up, and jury coordination are either entirely absent or require complicated, manual configuration. This makes these platforms poorly suited for handling the full scope of project management needs in a university environment.

2. Complexity for Beginners

For many first-year students or those new to project management, tools like Trello and Jira can feel overwhelming. Their interfaces are often cluttered with features and workflows designed for experienced users in professional environments, not students just beginning to learn how to organize academic tasks. As a result, students frequently abandon these platforms after a short time, feeling lost or unsure of their usefulness. These tools also operate on the assumption that users are already familiar with methodologies like Kanban or Scrum—concepts that aren't always introduced in early stages of university education. This creates a learning barrier that can discourage students from engaging with the tools altogether.

3. Missing Critical Features

Despite their popularity, platforms like Trello, Jira, and Moodle lack several essential features needed for effective academic project supervision. Important components such as report management, supervisor assignment and tracking, and jury member coordination are either entirely missing or require cumbersome manual setups. These gaps make the platforms poorly suited for managing the full lifecycle of university projects like PFA or PFE.

4. Not Centralized or Scalable for Universities

One of the major drawbacks of using platforms like Trello, Jira, or Moodle in an academic setting is the lack of integration between them. Students and professors are often forced to juggle multiple tools to manage different aspects of their projects, which leads to confusion, missed deadlines, and unnecessary administrative overhead. Moreover, these tools were not built with scalability in mind for university-wide deployment. High licensing costs, combined with the need for continuous IT support, make them impractical for widespread use across academic institutions.

5. Relies Too Much on Student Initiative

Another significant limitation of platforms like Trello, Jira, and Moodle is that they rely heavily on student initiative. These tools expect students to explore features on their own, set up workflows, and organize tasks—something that many students may not be ready for, especially those in their first year of university. Without a built-in academic structure tailored to the university's needs, these platforms can quickly become irrelevant and difficult to maintain engagement with, causing students to abandon them altogether.

Table 1.1: Comparison of Existing Solutions

	Trello	**Jira**	**Moodle**
Designed for academic workflows (PFA/PFE)	✗	✗	✗
Ease of use for students (beginner friendly)	✓	✗	✗
Task tracking with visual boards (Kanban)	✓	✓	✗
Academic deadline management (PFA/PFE)	✗	✓	✓
Supervisor and jury tracking	✗	✗	✗
Report submission and evaluation	✗	✗	✓
Centralized collaboration (all in one place)	✓	✓	✓
Role assignment (student, supervisor, admin)	✗	✓	✓
Scalable and customizable for university use	✗	✗	✓

As shown above in the Table 1.1 , although each tool has some strengths, none of them are specifically designed to support PFA/PFE processes at ENSIT. The lack of clarity, complexity of use, licensing limitations, and missing academic-specific flows make them suboptimal for university use.

1.4 Proposed Solution

Our solution stands out because it is specifically designed with universities in mind. It's simple, clear, and focused on the unique needs of academic project management, especially for tasks like PFA/PFE projects. Unlike other tools, it includes:

- Projects are divided by departments, making access easier and clearer for everyone involved.
- Task tracking that is directly adapted to the academic project flow.
- Role-based dashboards, ensuring that students, supervisors, and jury members each have a tailored experience.
- Integrated report submission, feedback, and evaluation tools—all in one easy-to-use platform.

Our solution is built to be user-friendly, even for students with no prior experience in project management tools. Furthermore, it can be deployed across an entire university without the need for complex configurations or costly paid licenses, making it accessible and scalable for academic institutions.

1.5 Adopted Methodology

Choosing an appropriate methodology is crucial for the success of any project, as it directly impacts how efficiently the team collaborates and adapts to changes. Agile was particularly suitable for our project due to its flexibility, iterative nature, and ability to accommodate evolving requirements.

1.5.1 Agile Methodology

Agile is a very popular project management methodology in IT companies. Among the many existing approaches, Agile focuses on setting short-term goals. We used the Agile methodology for this project. [4]

1.5.2 Scrum

Scrum is a framework derived from Agile that addresses complex and evolving problems while delivering the highest possible value in a productive and creative manner. [5]

1.5.2.1 Scrum Roles for Academic Projects

In the context of an academic project with a supervisor and two students, the Scrum team structure can be adapted to reflect the different roles and responsibilities involved in the project. The roles are as follows:

- **Supervisor (Product Owner):** The supervisor represents the academic goals and project requirements. They are responsible for expressing the needs of the project, prioritizing tasks, and validating the completed work. They guide the students and provide feedback at each stage of the project.

- **Project Manager (Scrum Master):** This role can be assumed by one of the students or the supervisor, depending on the structure of the project. The Project Manager ensures that the Scrum process is being followed, facilitates the team's meetings (e.g., daily scrums), and removes any obstacles that might hinder progress.
- **Students (Scrum Team):** The two students form the Scrum Team, responsible for executing the tasks and delivering the academic outputs of the project. They work on the research, development, or writing tasks based on the priorities set by the supervisor. They collaborate, self-organize, and are accountable for delivering the Sprint Goal.

1.5.2.2 Scrum Process for Academic Projects

Scrum, as an iterative methodology, fits well with academic projects, where progress can be incrementally developed, and tasks can be completed in manageable, focused periods. In this context, each iteration is called a *Sprint*, usually lasting between two to four weeks. The Scrum process for small academic projects, such as one with a supervisor and two students, includes the following ceremonies, illustrated in Figure 1.4:

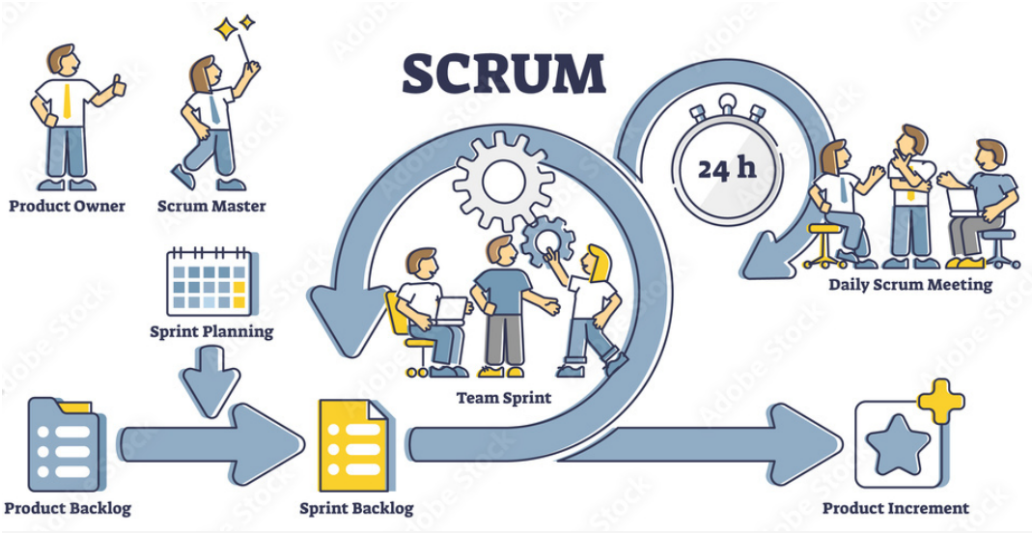


Figure 1.4: The Scrum Process adapted for Academic Projects

- **Sprint**: A time-boxed iteration (usually 2-4 weeks) where a piece of the academic project is developed or a milestone is achieved. The goal is to create a tangible output such as a report, research findings, or prototype.
- **Sprint Planning**:
  - Time-boxed meeting (max 4 hours for a 2-week Sprint)
  - The students, guided by the supervisor, select tasks or goals from the project backlog to complete during the Sprint, such as writing a section of the report, conducting research, or implementing a feature.
  - The Sprint Backlog is created, and the Sprint Goal is defined, ensuring everyone is aligned with what is to be achieved.
- **Daily Scrum** :
  - Daily 10-15 minute meeting
  - Each team member answers: What did I do yesterday? What will I do today? Are there any obstacles in my way?
  - Focus on keeping the project on track and making sure all members are aligned with the Sprint Goal.

- **Sprint Execution (Development Work)**:
  - The period during which the students work on the tasks defined in the Sprint Backlog
  - This could include research, coding, analysis, or writing the project documentation
  - The team works autonomously, with regular check-ins with the supervisor to ensure the project is progressing correctly.
- **Sprint Review**:
  - Time-boxed meeting (max 2 hours)
  - The team demonstrates the completed work to the supervisor, who provides feedback
  - The completed work is reviewed, and adjustments to the project or tasks can be made based on the supervisor's input.
  - The goal is to gather feedback and improve the project incrementally.
- **Sprint Retrospective**:
  - Time-boxed meeting (max 1 hour)
  - The team reflects on the past Sprint, identifies what went well, and discusses areas for improvement
  - Focus is placed on improving the working relationship between the students and supervisor, and on refining the project process.
  - The team creates a plan for applying improvements to the next Sprint.
- **Project Backlog**:
  - An ordered list of everything known to be needed for the academic project
  - Maintained and prioritized by the supervisor, with input from the students
  - This backlog evolves throughout the project, and items may be added or removed based on progress and feedback.
- **Sprint Backlog**:
  - A set of tasks selected from the Project Backlog for completion during the Sprint, along with a plan for delivering them
  - Owned and worked on by the two students, who are responsible for tracking their progress
  - It provides a real-time view of the work the team plans to complete and helps in monitoring progress toward the Sprint Goal.
- **Increment**:
  - The sum of all completed tasks from the Sprint Backlog during a Sprint
  - The result must be in a usable condition, even if the supervisor decides not to release it at the moment
  - Meets the academic team's definition of "Done," such as completing a chapter of a report or finalizing a feature.

1.6 Conclusion

This chapter outlined the project context and the limits of existing tools in academic settings. To overcome these, we proposed a solution focused on better collaboration and task tracking. We adopted Agile (Scrum) for its flexibility and iterative nature, fitting well with academic project needs.

CHAPTER 2

PLANIFICATION

2.1 Introduction

In this chapter, we will present our Scrum team. Then, we will conduct a preliminary analysis of the project by identifying the actors of our system, the product backlog, and the functional and non-functional requirements of our project.

In the second part, we will represent these requirements in a general use case diagram. Finally, we will present the various architectures of our system and the development environment.

2.2 Scrum Team

In the first chapter, we introduced the different roles in Scrum. Table 2.1 below represents the members of the Scrum team who participated in the development of this project.

Role	**Team Member**
Scrum Master	Issra Brahmi
Product Owner	Mehrez Boulares
Scrum Team	Issra Brahmi Rawia Ghrairi

Table 2.1: Scrum Team Members

2.3 Preliminary Analysis

A preliminary analysis will allow us to better understand the various features and characteristics of our application. In this section, we will describe the actors of our module, the product backlog, and then we will present the functional and non-functional requirements.

2.3.1 System Actors

The system actors are the users who will benefit from the services provided by our application. We can identify the following actors:

- **Student:** The student uses the application to view available projects, request a supervisor, receive project assignments, and track task completion. They can also upload deliverables and communicate with their supervisor.
- **Supervisor:** The supervisor assigns projects to students, creates and manages tasks, tracks student progress, and provides feedback at different stages of the project.
- **Chair of Department:** The chair oversees the entire thesis/project workflow. They approve project topics, assign supervisors, ensure fair distribution of workload, and handle escalation if needed.
- **Jury Member:** The jury member evaluates the final project submission, reviews assigned tasks and progress reports, and participates in the student's final defense session to provide a comprehensive evaluation.

2.3.2 Product Backlog

The "Product Backlog" is defined as a list of "user stories" that represent the different functionalities that characterize our product. To better organize our backlog, we define two properties:

- **Priority:**
  This is classified into 3 categories:
  - High
  - Medium
  - Low
- **Estimation:**
  This represents the complexity of the task to be performed:
  1. Easy
  2. Medium
  3. Difficult

The following table Table 2.2 summarizes the key user stories, their priorities, and estimated complexity for the university project management system:

User	User Story	Priority	Estimation
Student	As a student, I want to view my assigned project and tasks so I can organize my work.	High	Medium
Supervisor	As a supervisor, I want to assign tasks to students so I can monitor their progress.	High	Medium
Administrator	As an admin, I want to assign supervisors and jury members to projects so that responsibilities are clearly defined.	High	Difficult
Student	As a student, I want to upload my project report before the deadline so that the jury can evaluate it.	High	Medium
Jury Member	As a jury member, I want to access student reports easily so that I can prepare for the defense.	Medium	Medium
Supervisor	As a supervisor, I want to track all my supervised projects in one place so that I stay organized.	Medium	Easy
Admin	As an admin, I want to filter projects by department so that I can manage them efficiently.	Medium	Easy
Student	As a student, I want to receive notifications for deadlines so I don't miss them.	Low	Medium

Table 2.2: Product Backlog for University Project Management System

2.4 Requirements Gathering

Requirements gathering helps us identify the necessary functionalities that we will implement throughout this project in order to meet the needs of the end users.

2.4.1 Functional Requirements

The functional requirements define the essential features that must be implemented in the system:

- **The Administrator** can:
  - Manage student project records:
    - Create, edit, and delete student projects.
    - Assign students to projects.
    - Assign supervisors and jury members to projects.
    - Organize projects by department (Computer Science, Mechanical, etc.).
  - Monitor deadlines and submission statuses.
  - Filter and search projects by department or supervisor.

- **The Supervisor** can:
  - View and manage all assigned student projects.
  - Create and assign tasks for each student.
  - Track student progress and report completion.
  - Provide feedback or validation on project submissions.
- **The Student** can:
  - View their assigned project and related tasks.
  - Upload project reports and deliverables before the deadline.
  - Receive notifications for upcoming deadlines or supervisor feedback.
  - Track progress and see feedback from their supervisor.
- **The Jury Member** can:
  - Access assigned student reports and materials.
  - Evaluate submissions and provide assessment during the defense.
  - View schedules and related documentation for jury sessions.

2.4.2 Non-Functional Requirements

Non-functional requirements define the qualities and constraints the system must meet in order to provide a robust and high-value product:

- **Ergonomics:** The application must have a simple, clear, and user-friendly interface that is easy to navigate for all types of users (students, supervisors, administrators, and jury members).
- **Flexibility:** The system should be adaptable, allowing the addition or removal of new functionalities without impacting existing features.
- **Performance:** The application must ensure short response times and fast page loading, even when handling a large number of users and projects.
- **Maintainability and Scalability:** The codebase should be clean, modular, and well-documented to facilitate future development, bug fixes, and scalability as project needs evolve.

2.5 Functional Specification

In this section, we illustrate the needs of each actor through a use case diagram, followed by a discussion of the system architecture we have adopted.

2.5.1 Use Case Diagram

Figure 2.1 presents the general use case diagram that summarizes the functional requirements described in the previous section.

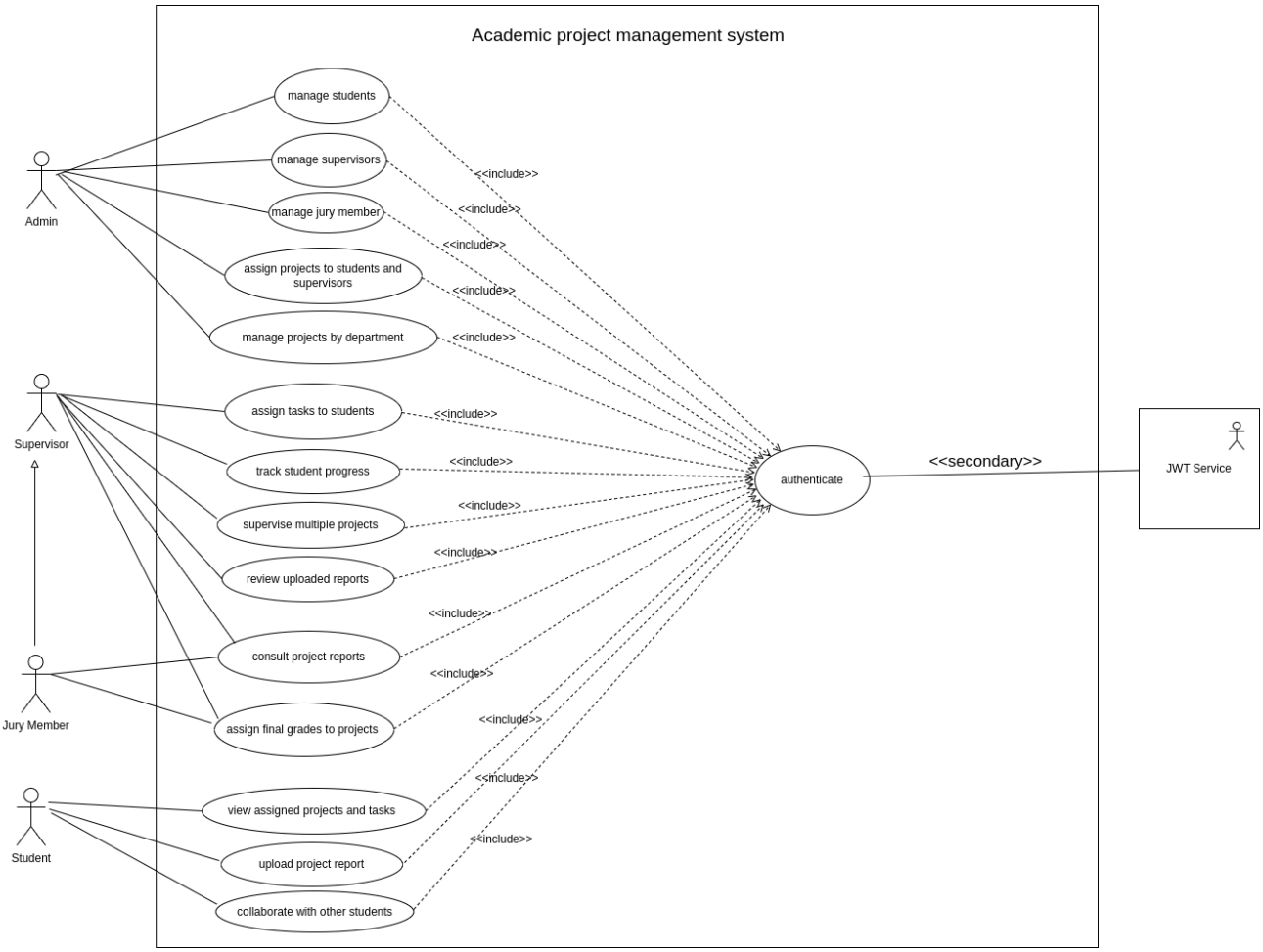


Figure 2.1: General Use Case Diagram

2.6 Global Architecture

The architecture of our application follows a **layered and modular approach** that ensures clear separation of concerns, scalability, and maintainability. This structure enhances collaboration between the frontend and backend teams, promotes clean code organization, and simplifies future updates or testing. As shown in the Figure 2.2 The system is composed of distinct but interacting layers: **presentation**, **business**, and **data**, each responsible for specific tasks in the application's flow.

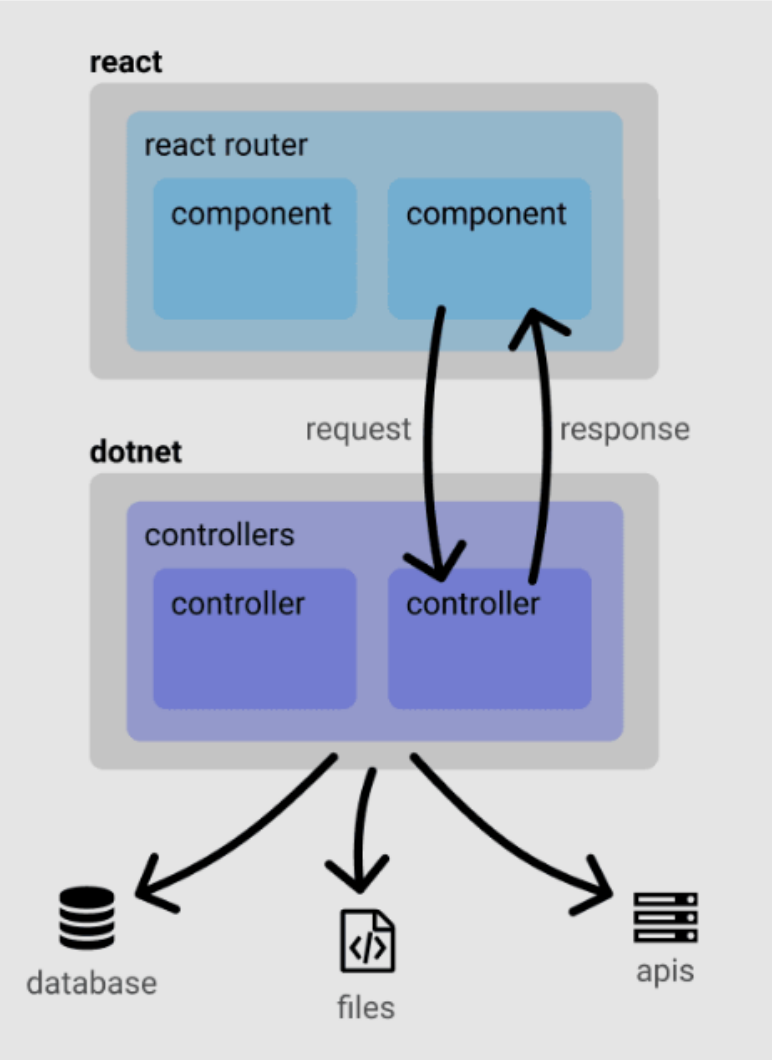


Figure 2.2: Global Architecture [6]

2.6.1 Main Components

- **React Frontend**:
  - Built with ReactJS, a JavaScript library for dynamic user interfaces.
  - Uses *React Router* for client-side routing.
  - Composed of reusable components for modular UI design.
  - Communicates with backend via RESTful APIs (e.g., Axios).
- **.NET Backend**:
  - Built on ASP.NET Core, following Clean Architecture principles.
  - Controllers handle incoming HTTP requests and responses.
  - Uses DTOs, repositories, and interfaces to organize logic and data flow.
  - Ensures separation between infrastructure, domain logic, and presentation.

- **Data Layer**:
  - Uses PostgreSQL for relational data persistence.
  - Manages file uploads and storage.
  - Supports external service integration via APIs.

2.6.2 REST Architecture:

Our Backend service uses REST, an HTTP-based architectural style where resources are accessed via URIs, and responses are often in JSON format. Key principles include:

- **Uniform Interface:** All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI).
- **Client-server decoupling:** In REST API design, client and server applications must be completely independent of each other. The only information that the client application should know is the URI of the requested resource;
- **Stateless:** REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it.
- **Cacheability:** When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource.
- **Layered system architecture:** In REST APIs, the calls and responses go through different layers. As a rule of thumb, don't assume that the client, and server applications connect directly to each other.

In summary, adhering to these REST API principles—such as uniform interfaces, client-server decoupling, statelessness, cacheability, and layered system architecture—ensures that the API is scalable, efficient, and easy to maintain. By following these guidelines, developers can create APIs that are flexible and can evolve independently on both the client and server sides without breaking functionality. Figure 3.3 illustrates the Rest architecture.

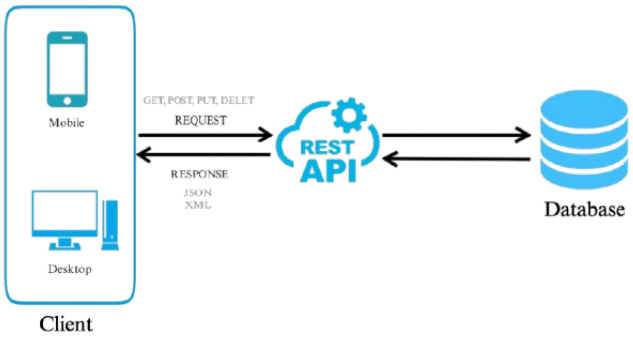


Figure 2.3: Rest Architecture

2.7 Development Environment

This section presents the development environment used for this project, starting with the hardware specifications followed by the software tools and technologies.

2.7.1 Hardware Environment

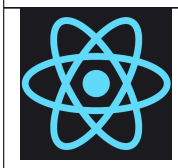
The development work was carried out on a machine with the following specifications:

- **Processor:** 11th Gen Intel Core i7-1165G7 @ 2.80GHz
- **RAM:** 8 GB
- **Storage:** 119.2 GB SSD
- **Operating System:** Ubuntu/Linux

2.7.2 Software Environment

Table 2.3 summarizes the languages and technologies used to develop our project.

Table 2.3: Languages and Technologies

Logo	Description
	**Csharp [7]** is a modern, object-oriented programming language developed by Microsoft, commonly used for building desktop applications, games, and enterprise solutions.
	**.NET Framework [8]** is a software development platform by Microsoft that supports building and running applications on Windows. It includes a large class library and supports multiple languages.
	**React Native[9]** is an open-source framework developed by Facebook for building mobile apps using JavaScript and React, allowing code sharing between Android and iOS.
	**PostgreSQL[10]** is a powerful, open-source relational database known for reliability, feature robustness, and standards compliance.
	**Swagger[11]** is a set of open-source tools for designing, building, documenting, and consuming REST APIs using OpenAPI specification.

Logo	Description
	**Cursor IDE[12]** is a modern AI-powered IDE optimized for productivity and focused on software development using advanced features like inline suggestions.
	**GitHub[13]** is a cloud-based platform for version control using Git, enabling collaboration, code review, and CI/CD workflows.
	**Trello[14]** is a visual collaboration tool used for task and project management, popular for organizing work in a kanban-style board.

2.8 Sprint Planification

This section outlines the breakdown of user stories across three development sprints. Each sprint focuses on a specific module of the system: project and task management, user management, and report management. The table below lists the user stories, the sprint they belong to, and the corresponding actors involved.

Sprint	User Story	Actor (Role)
Sprint 1 – Project and Task Management		
Sprint 1	Admin can create new project	Admin
Sprint 1	Admin can update a project	Admin
Sprint 1	Admin can delete a project	Admin
Sprint 1	Admin assigns supervisor to a project	Admin
Sprint 1	Admin can add students to a project	Admin
Sprint 1	Add tasks to a project	Supervisor, Student
Sprint 1	Delete tasks of a project	Supervisor, Student
Sprint 1	Update tasks of a project	Supervisor, Student
Sprint 1	Assign one student to a task	Supervisor
Sprint 1	Upload/download attachments inside a task	Student

Sprint	**User Story**	**Actor (Role)**
Sprint 1	Change priority of a task	Student
Sprint 1	Collaborate with others inside a task	Student
Sprint 2 – User Management		
Sprint 2	Add new supervisor to list	Admin
Sprint 2	Delete a supervisor	Admin
Sprint 2	Update a supervisor	Admin
Sprint 2	Add a department	Admin
Sprint 2	Add a department chair	Admin
Sprint 2	Edit a department	Admin
Sprint 2	Add a jury member	Admin
Sprint 2	Delete a jury member	Admin
Sprint 2	Update a jury member	Admin
Sprint 2	Add a student	Admin
Sprint 2	Update student details	Admin
Sprint 2	Delete a student	Admin
Sprint 3 – Report Management		
Sprint 3	Create a report field with project ID	Admin
Sprint 3	Assign a jury member to a report	Admin
Sprint 3	Upload a report	Student
Sprint 3	Download a report	Student, Supervisor, Jury Member

2.9 Conclusion

This chapter was dedicated to the project planning phase, during which we began by introducing the Scrum team, followed by a preliminary analysis detailing the various actors of our system, the functional and non-functional requirements, and finally concluded with the presentation of the different architectures and the project's development environment. We also included the sprint planning, outlining the tasks and objectives to be achieved during each sprint to ensure a smooth and efficient project development process.

CHAPTER 3

SPRINT 1 : "PROJECT/TASK MANAGEMENT"

3.1 Introduction

In this chapter, we will first address the analysis and requirements specification part of the first sprint, starting with the presentation of the sprint backlog, the functional requirements, the use case diagram, and the use case scenarios. Then, we will move on to the second part, which will be dedicated to the design phase, during which we will briefly describe the class and sequence diagrams for the various features associated with this sprint. Finally, we will conclude with the presentation of some interfaces of the implemented features.

3.2 Analysis and Requirements Specification

The specification of requirements is considered a crucial phase in project planning, as it allows for identifying and defining the client's needs.

3.2.1 Sprint 1 Backlog

Table 3.1 presents the various features related to project and task management that were implemented during the first sprint.

Table 3.1: Sprint 1 – Project and Task Management

Sprint	**User Story**	**Actor(s)**
Sprint 1	Admin can create new project	Admin
Sprint 1	Admin can update a project	Admin
Sprint 1	Admin can delete a project	Admin
Sprint 1	Admin assigns supervisor to a project	Admin
Sprint 1	Admin can add students to a project	Admin
Sprint 1	Add tasks to a project	Supervisor, Student
Sprint 1	Delete tasks of a project	Supervisor, Student

Sprint	**User Story**	**Actor(s)**
Sprint 1	Update tasks of a project	Supervisor, Student
Sprint 1	Assign one student to a task	Supervisor
Sprint 1	Upload/download attachments inside a task	Student
Sprint 1	Change priority of a task	Student
Sprint 1	Collaborate with others inside a task	Student

3.3 Functional Requirements Gathering

During this sprint, we will focus on the functional requirements related to project and task management, which are listed as follows:

- The Administrator can:
  - Manage projects:
    - Create a project.
    - Update a project.
    - Delete a project.
    - Assign a supervisor to a project.
    - Add students to a project.
- The Supervisor or Student can:
  - Manage project tasks:
    - Add tasks.
    - Delete tasks.
    - Update tasks.
- The Supervisor can:
  - Assign a student to a task.
- The Student can:
  - Upload and download attachments within a task.
  - Change the priority of a task.
  - Collaborate with other students within a task.

3.4 Use Case Diagram

The use case diagram in Figure 3.1 below describes the different actions that the admin can perform to manage projects.

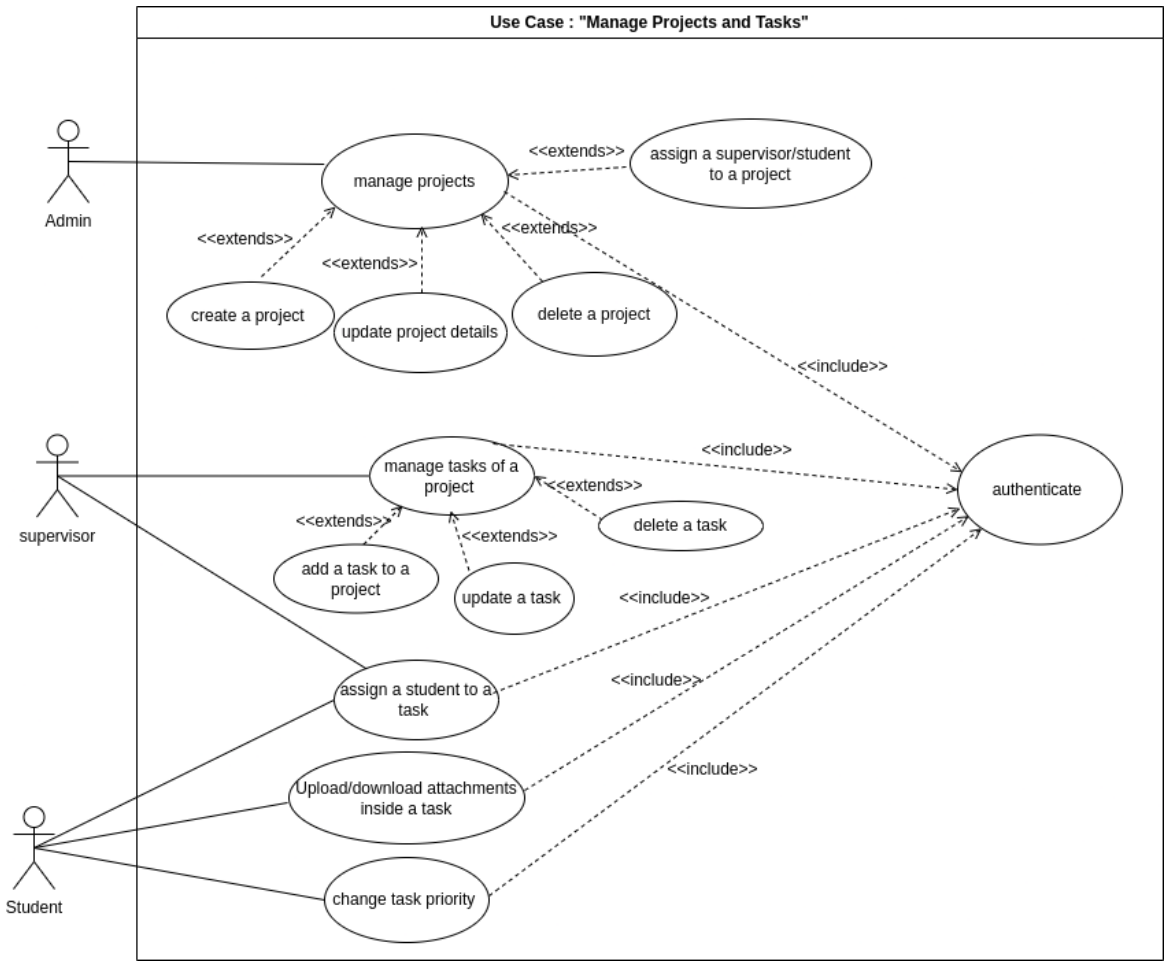


Figure 3.1: Use Case Diagram for Project/Task Management

3.5 Use Case Scenarios

After introducing the use case diagram *Manage Projects/Tasks*, this section details each use case scenario through a textual description.

3.5.1 Use Case: "Create a New Project"

Table 3.2 presents the textual description of the use case *Create a New Project*.

Table 3.2: Textual Description of the Use Case "Create a New Project"

Element	**Description**
Primary Actor	Project Manager
Objective	To manage tasks effectively, the project manager must first create different projects to organize the work structure.

Element	**Description**
Basic Flow of Events	1. The project manager clicks on the “Add Project” button. 2. The system displays the project creation form. 3. The project manager fills in the project name, supervisor’s first and last name, level, department, status, start date, and end date, then clicks the “Submit” button. 4. The system updates the database and refreshes the list of existing projects.
Post-condition	The database is updated with the new project.

3.5.2 Use Case: "Update a Project"

Table 3.3 presents the textual description of the use case *Update a Project*.

Table 3.3: Textual Description of the Use Case "Update a Project"

Element	**Description**
Primary Actor	Project Manager
Objective	To modify an existing project’s information when necessary to ensure data accuracy and up-to-date project details.
Basic Flow of Events	1. The project manager selects a project from the project list. 2. The system displays the project’s existing details. 3. The project manager edits fields such as the project name, supervisor’s name, level, department, status, or dates, then clicks the “Update” button. 4. The system updates the database and refreshes the list of projects.
Post-condition	The selected project’s information is updated in the database.

3.5.3 Use Case: "Delete a Project"

Table 3.4 presents the textual description of the use case *Delete a Project*.

Table 3.4: Textual Description of the Use Case "Delete a Project"

Element	**Description**
Primary Actor	Project Manager
Objective	To remove an outdated or unnecessary project from the system to maintain a clean and relevant project list.
Basic Flow of Events	1. The project manager selects a project from the list. 2. The system displays a confirmation dialog. 3. The project manager confirms the deletion. 4. The system removes the project from the database and refreshes the list of projects.
Post-condition	The selected project is permanently deleted from the database.

3.6 Design

The design phase consists of refining the descriptions made during the analysis. It aims to build a stable architecture for the system to be implemented. This is a crucial and essential step to move toward the implementation phase, as it defines a structured approach to developing a reliable and scalable product.

In this section, we will present the class diagrams and sequence diagrams of the first sprint.

3.6.1 Class Diagram

The class diagram represents the static structure of the system in terms of classes and their relationships.

After conducting a full analysis of all use cases, we derived the class diagram shown in Figure 3.2. This diagram includes the following core models:

- **Project**: This class represents a project created and managed by 1 or many supervisors, and may involve multiple students and tasks.
- **Task**: Represents the tasks associated with a specific project. Each task may include a description, status, due date. We can assign one student per task.
- **Attachment**: This class is used to store files or resources related to tasks.
- **Student**: Represents the students who participate in one type of project based on their level.
- **Supervisor**: Represents the supervisor who manages and oversees projects.

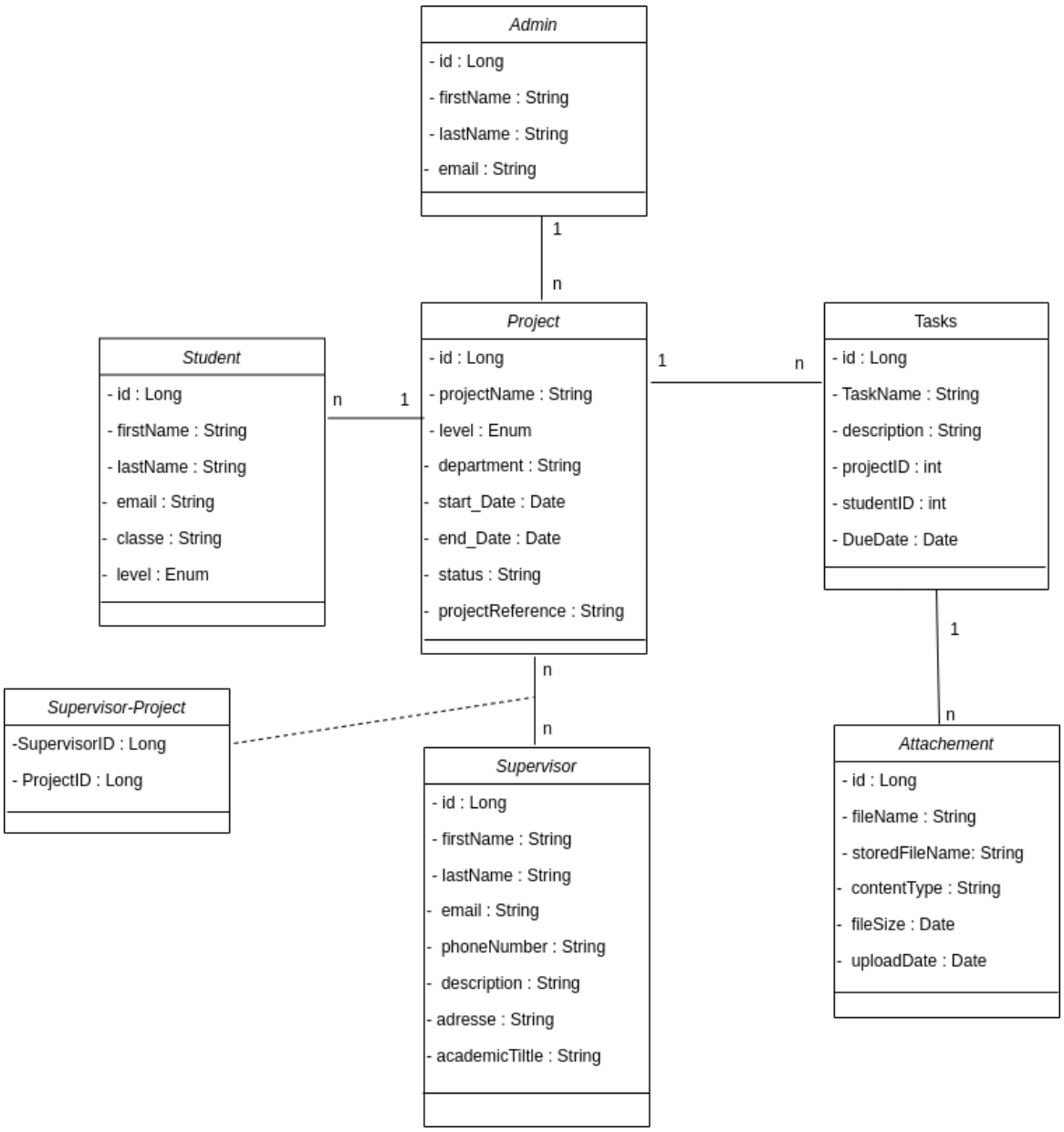


Figure 3.2: class Diagram of sprint 1

3.6.2 Sequence Diagram

Sequence diagrams are UML diagrams used to describe the chronological flow of interactions between objects in a software system. They help visualize how components communicate over time to complete a specific functionality.

The following section presents the refined sequence diagrams from the first sprint.

3.6.3 Sequence Diagram: "Create a Project"

Figure 3.3 illustrates the sequence diagram for the functionality *Create a Project*.

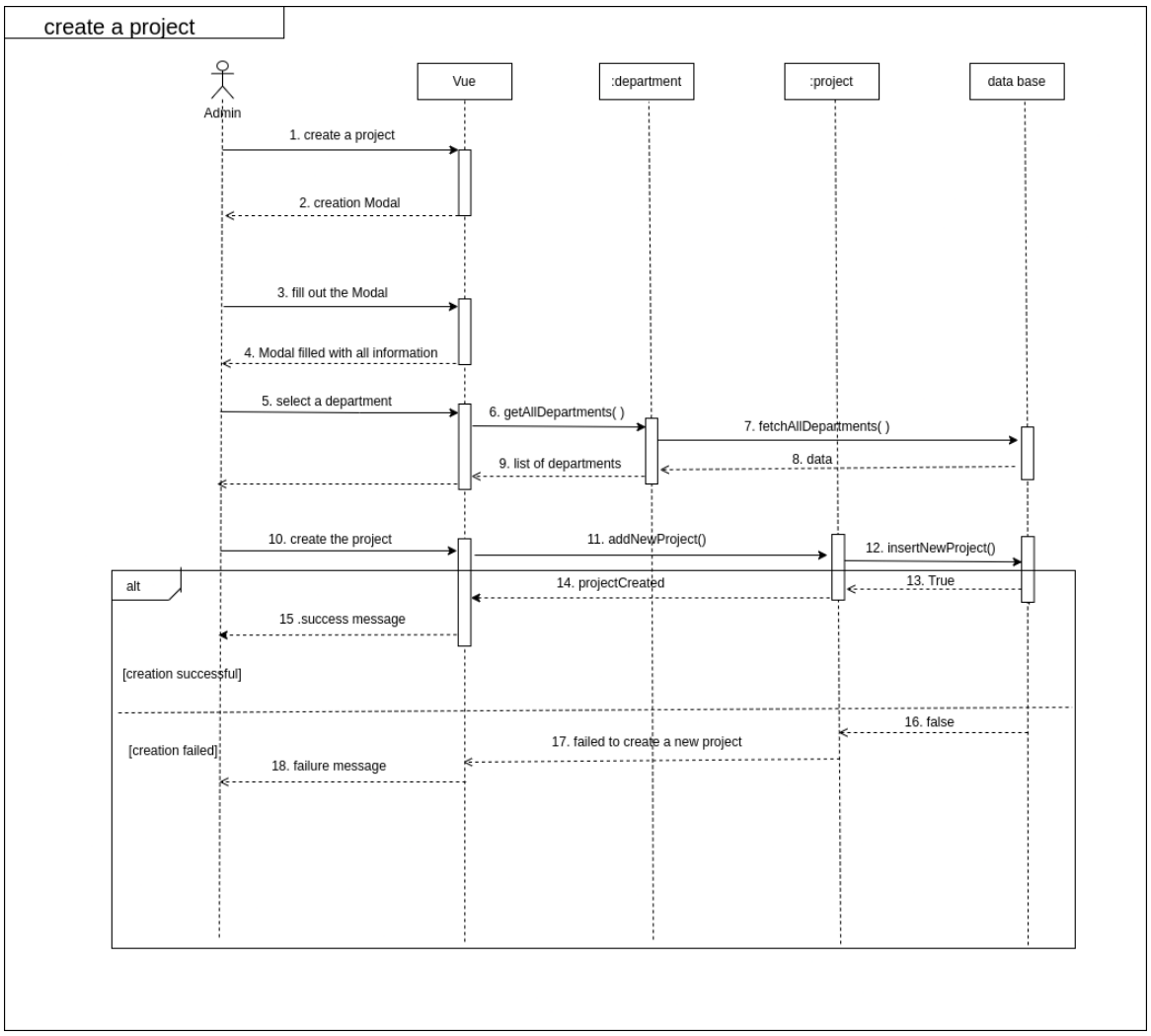


Figure 3.3: sequence diagram "create new project"

3.7 Implementation

After analyzing the various requirements from the first sprint and briefly describing each part of our system, this section presents some of the most relevant user interfaces that make up the application.

3.7.1 Create New Project

As shown in Figure 3.4 ,this interface allows the project manager to add a new project using a modal form. When the "Add Project" button is clicked, a modal appears with a form that includes the following fields:

- **Project Name** – the name or title of the project.
- **Project Reference** – the reference of the project.
- **Supervisor First Name and Last Name** – identifying the supervisor responsible for the project.
- **Start Date and End Date** – the duration of the project.
- **Status** – selected from one of the following: *Active, Suspended, Waiting, Paused.*
- **Department** – the department to which the project belongs.
- **Level** – the academic level of the project, chosen from: *PFA1, PFA2, PFE.*

The form ensures that all required information is captured before submission. Once submitted, the project is added to the system, and the list of existing projects is updated.

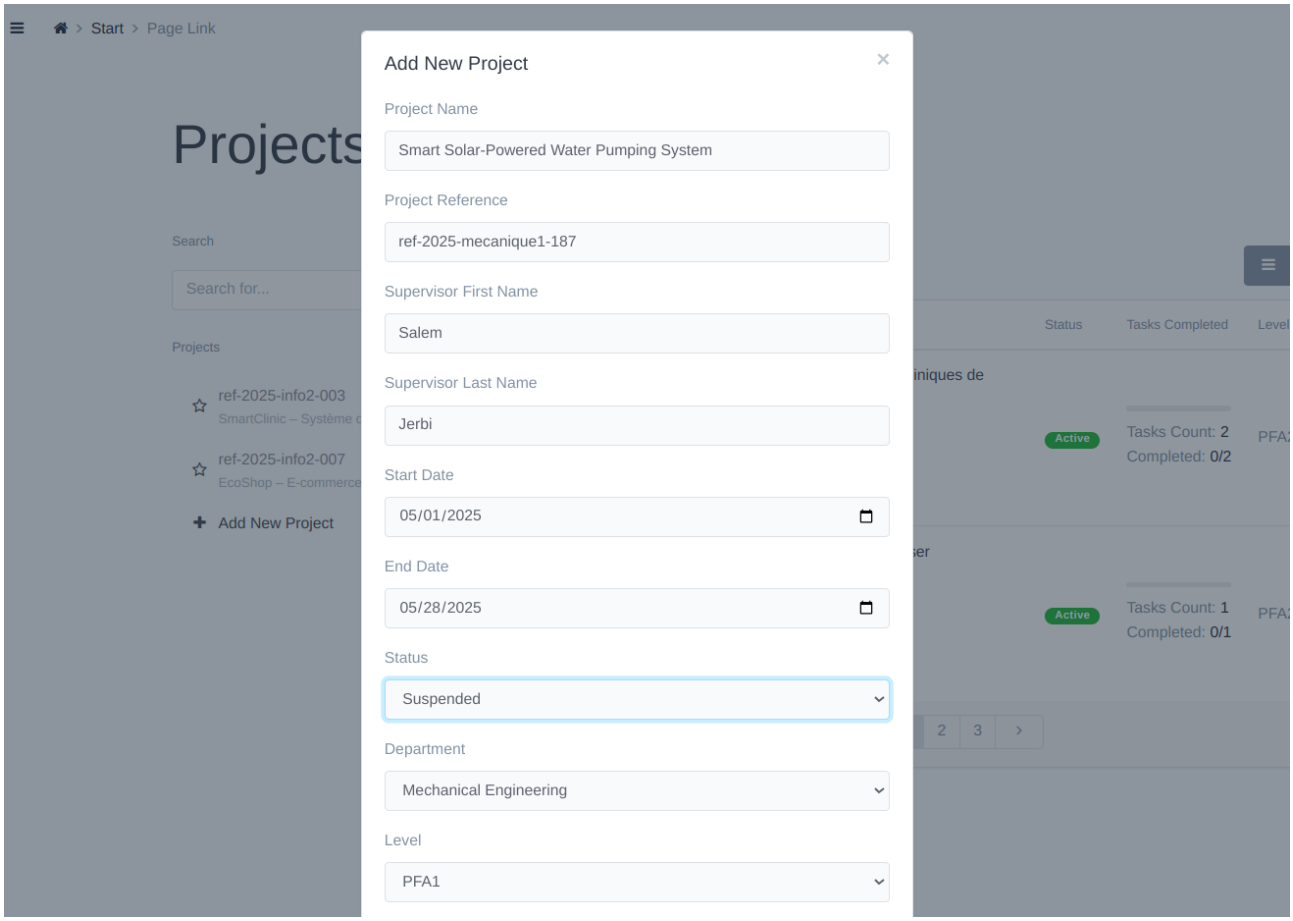


Figure 3.4: Create New Project Interface

3.7.2 List of projects interface

this interface shows the list of projects created by the admin we can add tasks and we can see the number of overall tasks and the data that we got from api call .

Figure 3.5 illustrates the interface

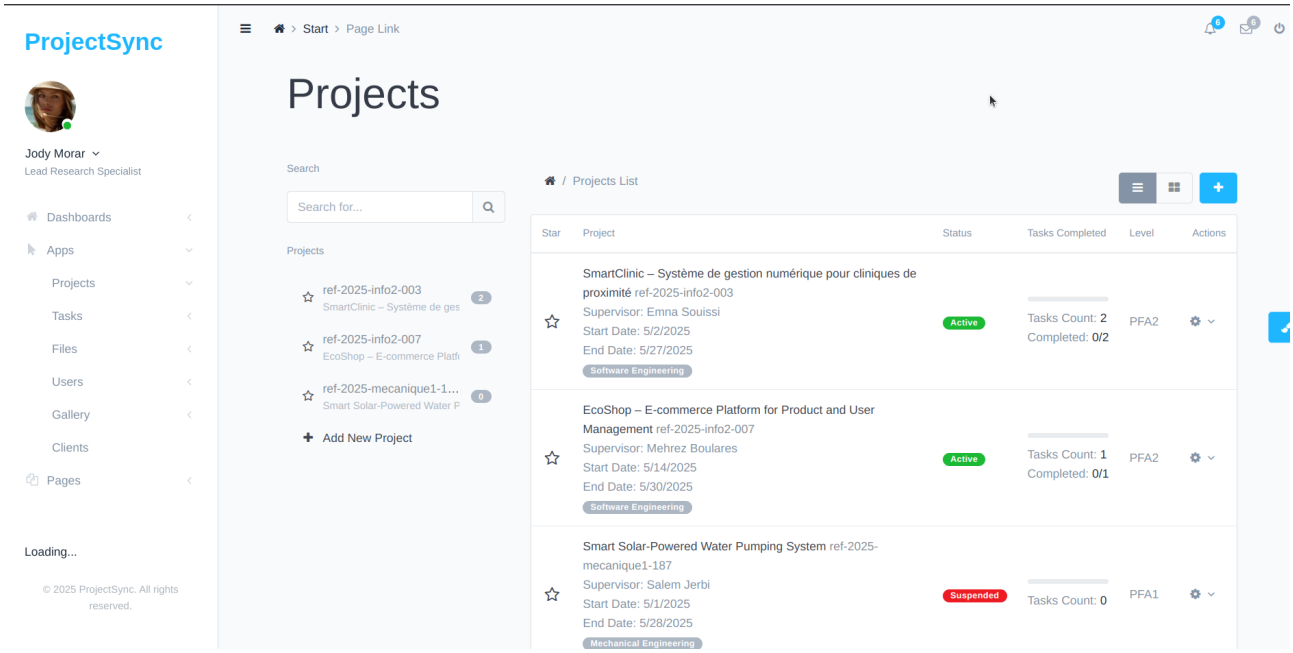


Figure 3.5: List of projects Interface

3.7.3 Add new Task to a project interface

we can add a task into a project by clicking the gear icon it open a dropdown and we can update a project or add a task or delete the project, we add a task by setting up the task title , due date , and description

Figure 3.6 illustrates the interface.

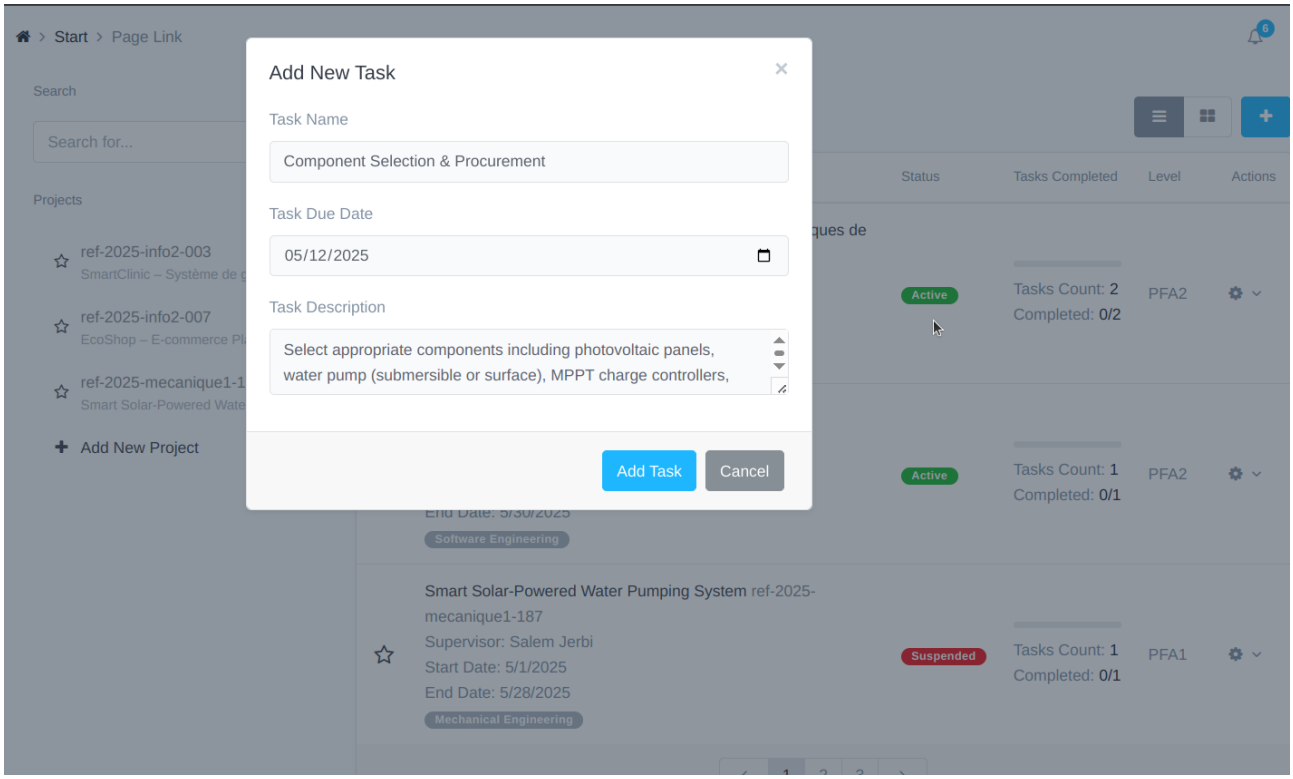


Figure 3.6: List of projects Interface

3.7.4 Add new Task to a project interface

we can add a task into a project by clicking the gear icon it open a dropdown and we can update a project or add a task or delete the project, we add a task by setting up the task title , due date , and description

Figure 3.7 illustrates the interface.

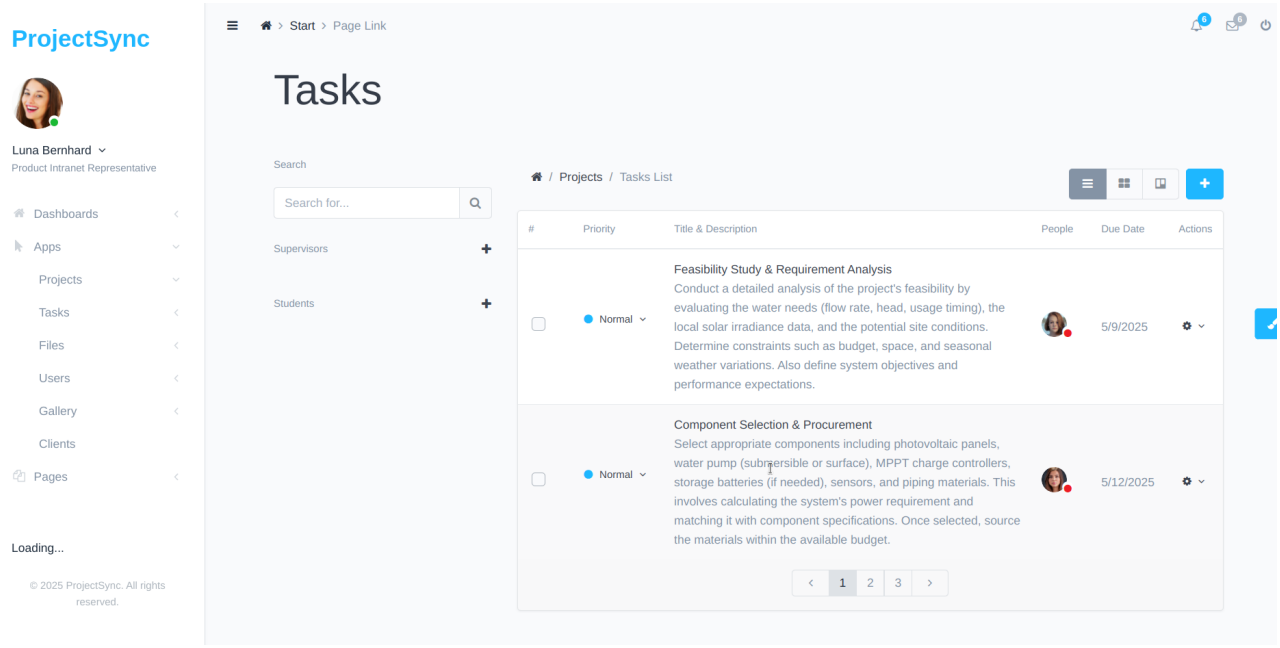


Figure 3.7: List of project tasks Interface

3.7.5 Add Supervisors and Assign Students to a Project

To assign supervisors or students to a project,we can just click the "Add" button. A list of available supervisors or students will appear, and you can select the ones you want to assign to the given project.

Figure 3.7 illustrates the interface.

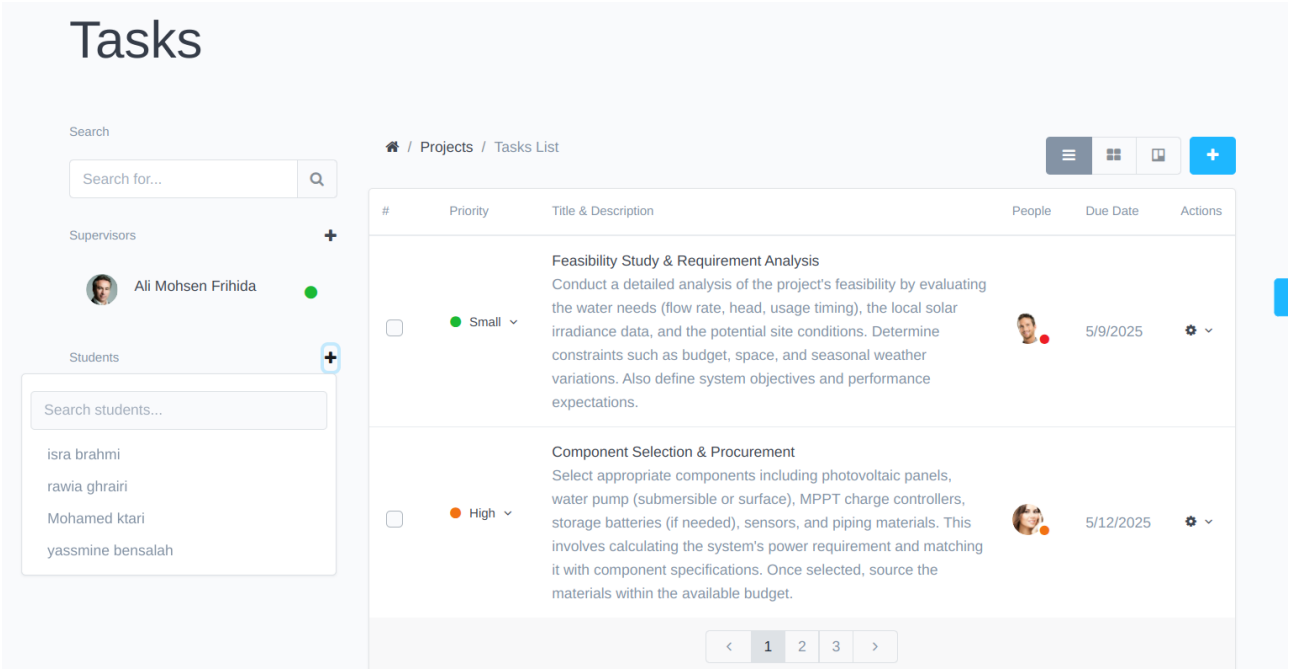
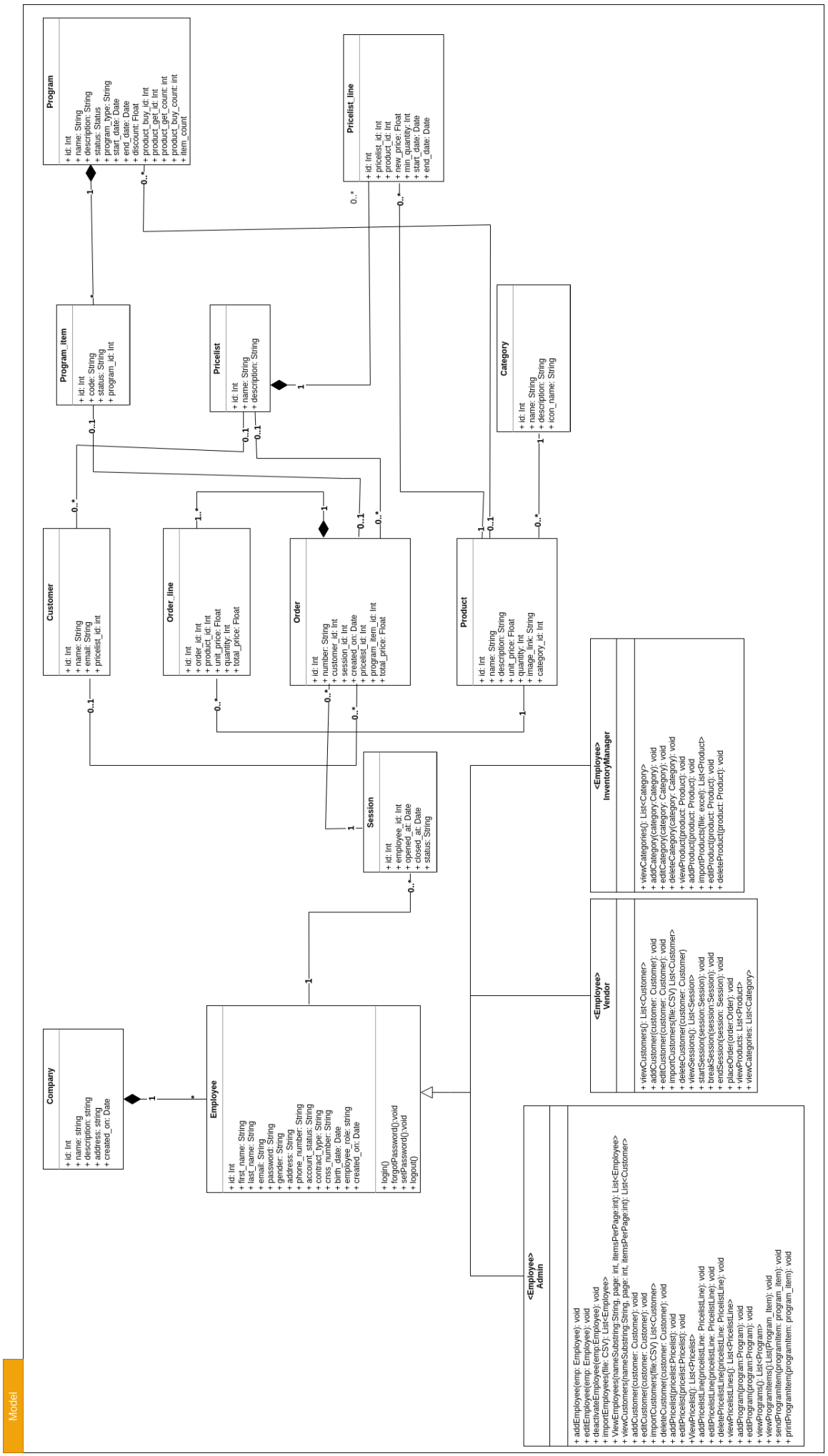


Figure 3.8: List of project tasks Interface


Model
Program
+ id: Int
+ name: String
+ description: String
+ status: Status
+ program_type: String
+ start_date: Date
+ end_date: Date
+ discount: Float
Pricelist_line
+ id: Int
+ pricelist_id: Int
+ product_id: Int
+ new_price: Float
+ min_quantity: Int
+ start_date: Date
+ end_date: Date
Program_item
+ id: Int
+ code: String
+ status: String
+ program_id: Int
Pricelist
+ id: Int
+ name: String
+ description: String
Category
+ id: Int
+ name: String
+ description: String
+ icon_name: String
Customer
+ id: Int
+ name: String
+ email: String
+ pricelist_id: int
Order_line
+ id: Int
+ order_id: Int
+ product_id: Int
+ unit_price: Float
+ quantity: Int
+ total_price: Float
Order
+ id: Int
+ number: String
+ customer_id: Int
+ session_id: Int
+ created_on: Date
+ pricelist_id: Int
+ program_item_id: Int
+ total_price: Float
Product
+ id: Int
+ name: String
+ description: String
+ unit_price: Float
+ quantity: Int
+ image_link: String
+ category_id: Int
Session
+ id: Int
+ employee_id: Int
+ opened_at: Date
+ closed_at: Date
+ status: String
Company
+ id: Int
+ name: string
+ description: string
+ address: string
+ created_on: Date
Employee
+ id: Int
+ first_name: String
+ last_name: String
+ email: String
+ password: String
+ gender: String
+ address: String
+ phone_number: String
+ account_status: String
+ contract_type: String
+ cnss_number: String
+ birth_date: Date
+ employee_role: string
+ created_on: Date
+ login()
+ forgotPassword():void
+ setPassword():void
+ logout()
<Employee>
InventoryManager
+ viewCategories(): List<Category>
+ addCategory(category:Category): void
+ editCategory(category: Category): void
+ deleteCategory(category: Category): void
+ viewProduct(product: Product): void
+ addProduct(product: Product): void
+ importProducts(file: excel): List<Product>
+ editProduct(product: Product): void
+ deleteProduct(product: Product): void
<Employee>
Vendor
+ viewCustomers(): List<Customer>
+ addCustomer(customer: Customer): void
+ editCustomer(customer: Customer): void
+ importCustomers(file:CSV) List<Customer>
+ deleteCustomer(customer: Customer)
+ viewSessions(): List<Session>
+ startSession(session:Session): void
+ breakSession(session:Session): void
+ endSession(session: Session): void
+ placeOrder(order:Order): void
+ viewProducts: List<Product>
+ viewCategories: List<Category>
<Employee>
Admin
+ addEmployee(emp: Employee): void
+ editEmployee(emp: Employee): void
+ deleteEmployee(emp: Employee): void
+ importEmployees(file: CSV): List<Employee>
+ ViewEmployees(nameSubstring:String, page: int, itemsPerPage:int): List<Employee>
+ viewCustomers(nameSubstring:String, page: int, itemsPerPage:int): List<Customer>
+ addCustomer(customer: Customer): void
+ editCustomer(customer: Customer): void
+ importCustomers(file:CSV) List<Customer>
+ deleteCustomer(customer: Customer): void
+ addPricelist(pricelist:Pricelist): void
+ editPricelist(pricelist:Pricelist): void
+ViewPricelist(): List<Pricelist>
+ addPricelistLine(pricelistLine: PricelistLine): void
+ editPricelistLine(pricelistLine: PricelistLine): void
+ deletePricelistLine(pricelistLine: PricelistLine): void
+ viewPricelistLines(): List<PricelistLine>
+ addProgram(program:Program): void
+ editProgram(program:Program): void
+ viewPrograms(): List<Program>
+ viewProgramItems():List(Program_Item): void
+ sendProgramItem(programItem: program_item): void
+ printProgramItem(programItem: program_item): void

- **Component**: Components are essential elements that construct the user interface of an application. Each component defines a segment of the interface and manages the rendering of views and user interactions. Components often utilize services to carry out operations, such as retrieving data from a server.

- **Service**: Services are used to manage business logic and data operations that can be shared across multiple components. They interact with backend systems to obtain or modify data, which is then provided to components for use.

- **Controller**: Controllers act as intermediaries between the services and the models. They receive requests from the services, apply business logic, and interact with the models to retrieve or update data. Controllers ensure that the right data is processed and returned to the services.

- **Model**: Models represent the data structures and business entities in the application. When using SQLAlchemy with PostgreSQL, models are defined using SQLAlchemy's ORM (Object-Relational Mapping) capabilities. SQLAlchemy allows you to define data schemas, manage relationships between entities, and interact with the PostgreSQL database using Python classes and SQL queries.

3.7.6 Dynamic View: Design Sequence Diagram

3.7.6.1 General Interaction Model

A sequence diagram illustrates how objects in a system interact over time, showing the sequence of messages exchanged between them. It helps visualize the flow of control and data in dynamic scenarios.

Figure 3.7 illustrates the interaction model between the frontend, backend, and database.

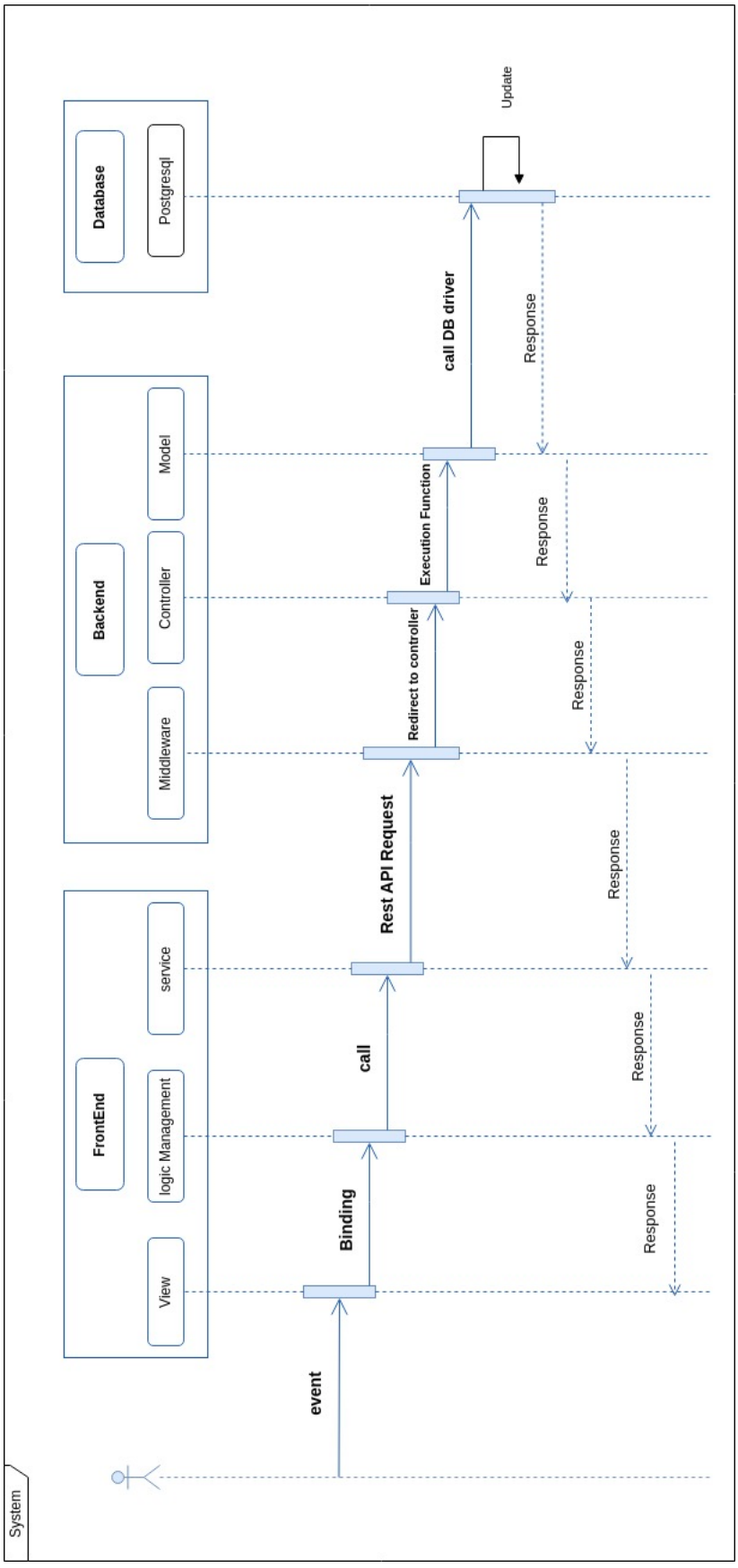


Figure 3.9: Interaction model

The interaction between the layers represented in Figure 3.7 takes place as follows:

- **Event**: Represents user interactions with the application's graphical interface, such as clicks, key presses, or other forms of input. These events trigger specific actions or updates within the application, enabling it to respond dynamically to user behavior.

- **Binding**: Establishes a connection between the data in a view-model and the values presented in the view. This mechanism ensures that any changes in the data are automatically reflected in the view, maintaining synchronization without manual updates.

- **Call**: Refers to the view-model utilizing a functionality provided by a service. The service, implemented on the frontend, handles technical aspects of communication, allowing the view-model to focus on data manipulation logic without dealing with the complexities of communication.

- **Rest API Request**: The frontend consumes services offered by the web API using the HTTP protocol and four main operations: GET (read), POST (create), PUT (update), DELETE (delete). The data exchange between the frontend service and the web API controller is in JSON format.

- **Redirect ToController**: The middleware invokes functionalities provided by the server's application layer, where the platform's business logic is implemented.

- **Call DB Driver**: The application layer needs to access the database to update data. It utilizes services provided by the database server, which abstracts database manipulations and offers basic CRUD methods (push, set, update, remove).

3.7.6.2 Sequence Diagram "Authentication"

The sequence diagram presented in Figure 3.8 illustrates the user authentication scenario, involving two verification tests where the user enters their email and password. The system first checks to ensure all input fields are valid and not empty. If any input errors are detected, an error message is displayed on the user interface. If there are no input errors, the **lookingForUser**() method is executed to validate the email and password against the data stored in the database. If the validation is successful, the system redirects the user to the dashboard page. However, if validation fails, an error message is displayed on the interface, notifying the user of incorrect credentials.

The authentication system of the application is based on JWT (Json Web Token). This allows for the secure exchange of tokens between multiple parties. The security of this exchange is ensured through the verification of the token's integrity and authenticity. The figure below represents the authentication sequence diagram. Figure 3.8 illustrates the authentication process.

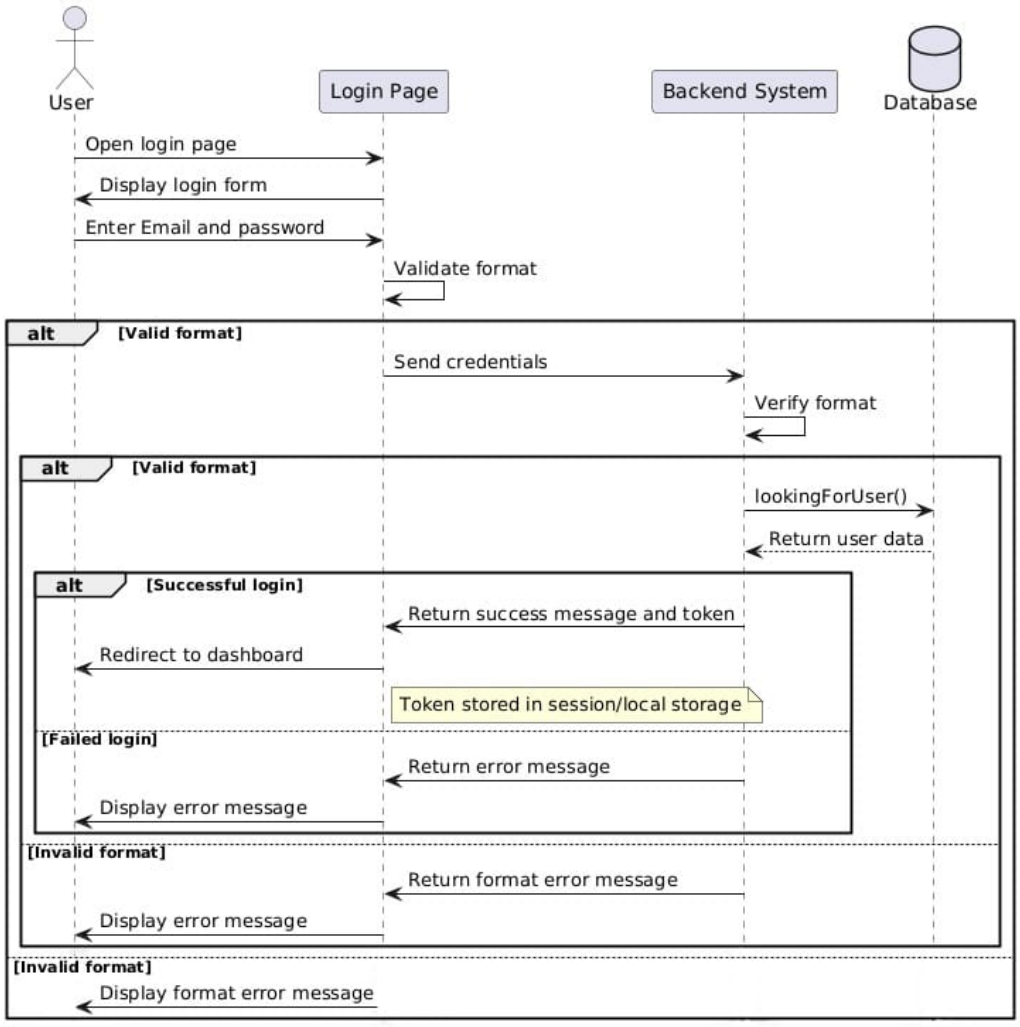


Figure 3.10: Authentication sequence diagram

3.7.6.3 Sequence Diagram: POS System User and Employee Setup

The sequence diagram in Figure 3.9 depicts the process of setting up a Point of Sale (POS) system for a store and managing employee accounts. Initially, the store's administrator contacts the POS Software Company to request a license. The POS company handles administrative tasks and negotiations. After finalizing the deal, the company generates the store's credentials using an internal tool, stores these credentials in the database, and delivers them to the store administrator. Once the store has the credentials, the administrator logs into the system via the web interface. The backend system authenticates the store's credentials by validating them against the stored data in the database. After successful authentication, the administrator can add new employees to the system. Employee accounts are created with details such as username, email, and role, but without a password. A token is generated, and the employee's information is stored in the database. The backend sends a password setup link to the employee's email.

When the employee clicks on the password setup link, they are directed to set their password and confirm their account. The backend system verifies the token's validity by checking if it exists in the database and whether it is within the valid time frame (less than 60 minutes). If valid, the system updates the employee's account with the new password and confirms the account. The employee is notified that their account has been successfully created. If the token is expired, already used, or invalid, the employee is informed accordingly. This process ensures secure authentication, proper user management, and employee account setup in the POS system.

Figure 3.9 illustrates the store employee and set password process.

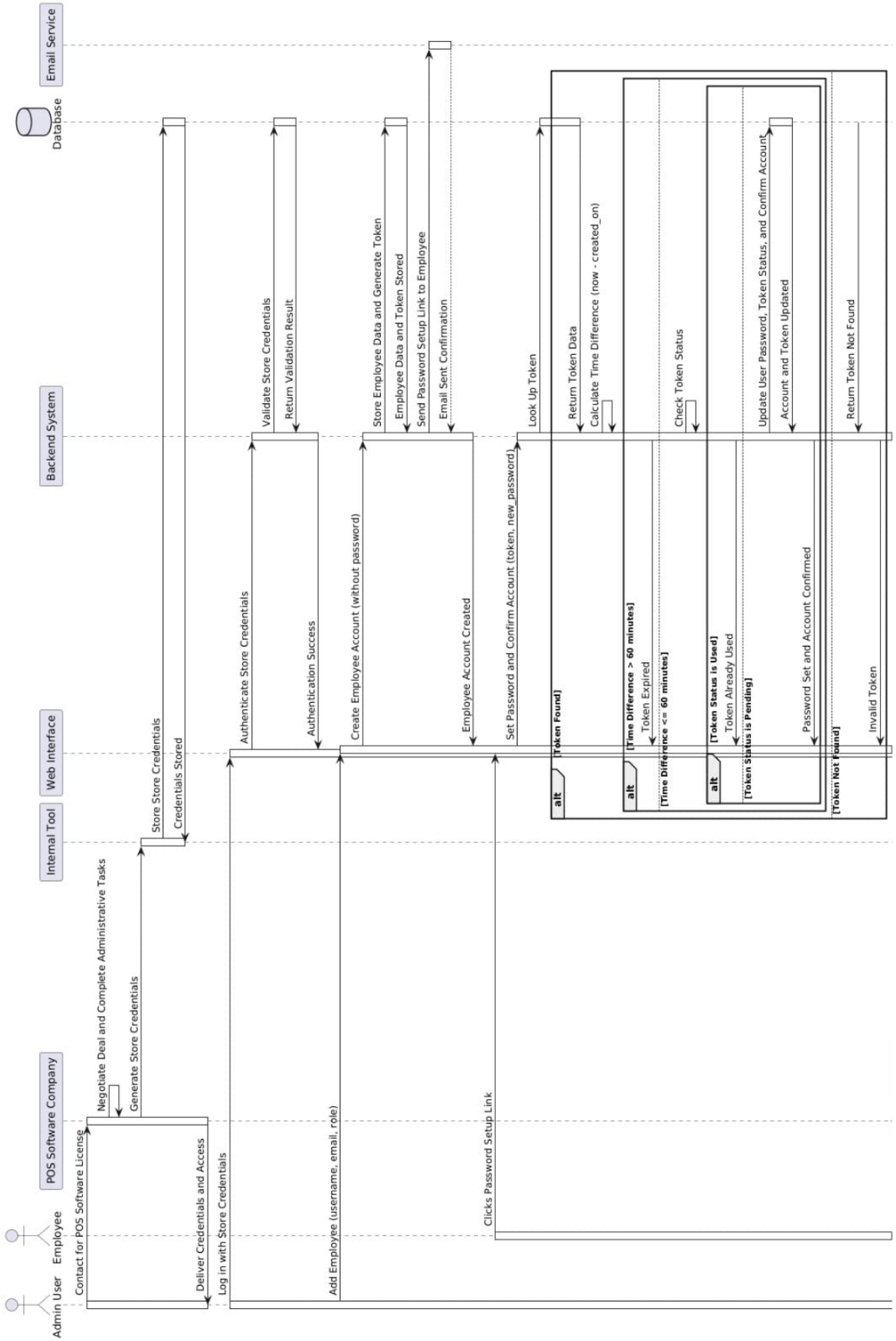


Figure 3.11: Store Employee And Set Password

3.7.6.4 Sequence Diagram: Password Reset

The sequence diagram provided in Figure 3.10 illustrates the password reset scenario. In this process, a user submits a password reset request via the web interface by providing their email address. The web interface forwards the request to the backend system, which checks the database for a corresponding user. If a user is found, the backend generates a unique reset token, stores it in the database along with the user's ID and a timestamp, and sends a password reset email with the token link through the email service. The user is then notified to check their email. If no matching email is found, the user is informed accordingly.

When the user clicks the reset link in the email, the web interface sends the token and new password to the backend system. The backend validates the token by checking if it exists in the database and calculates the time difference between the token's creation and the current time. If the token has expired (after more than 60 minutes), the backend notifies the user that the token has expired. If the token is valid and unused, the backend updates the user's password and marks the token as used. Finally, the user is informed that the password reset was successful. If the token is invalid or has already been used, the user is notified of the issue. Figure 3.10 illustrates the reset password process.

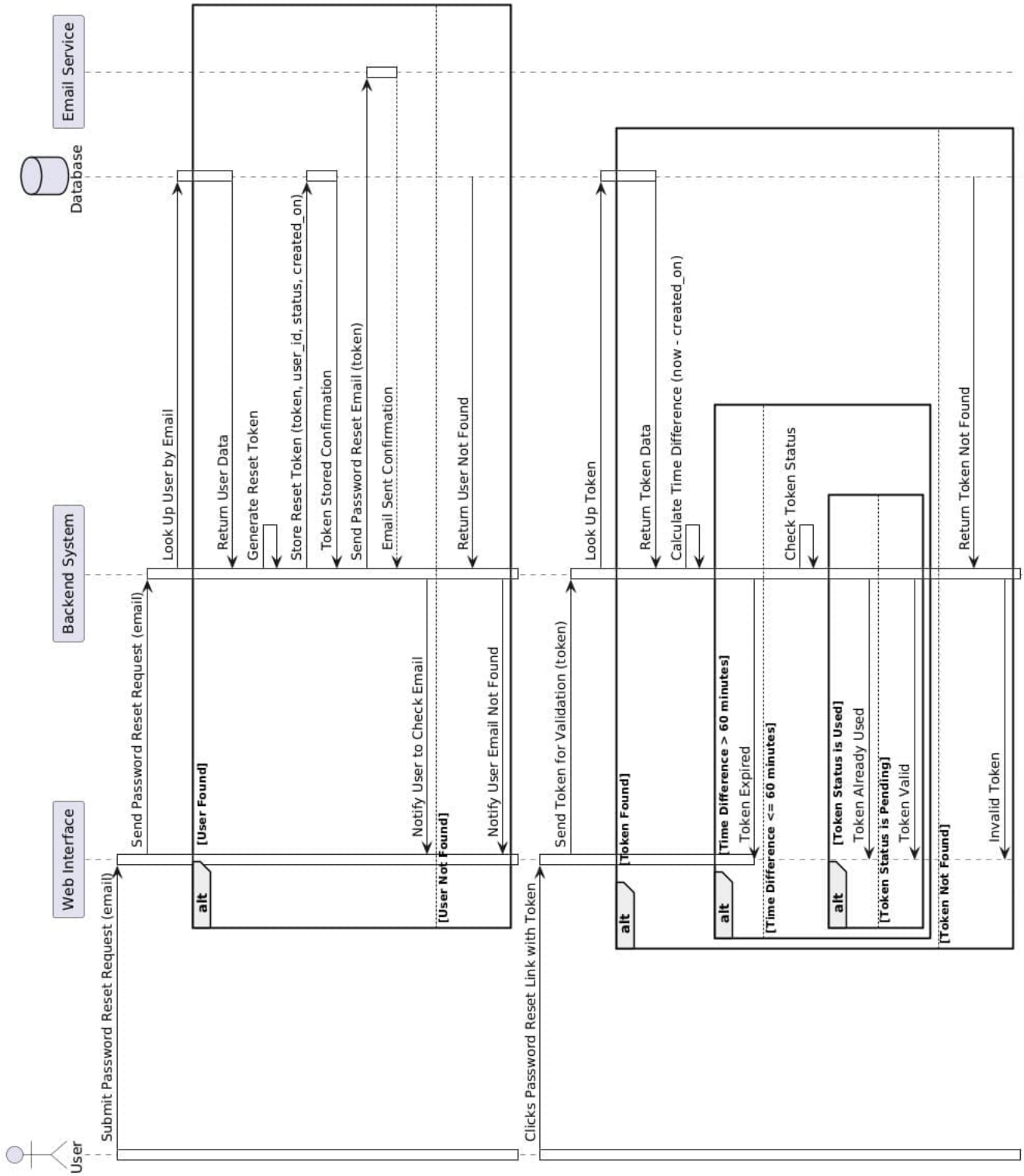


Figure 3.12: Password Reset Sequence Diagram

3.7.6.5 Sequence Diagram: Product Creation with Optional Image Upload

The sequence diagram in Figure 3.11 represents the process of creating a product through a web interface, with the option to upload an image.

First, the user enters the product details, and optionally selects an image to upload. The web interface validates the form inputs to ensure they are correct.

If the validation fails, the system returns the errors to the user for correction. If the validation succeeds, the system proceeds to the next step.

If the user has selected an image, the web interface uploads the image to Cloudinary, which returns the image URL and public ID. This image URL is then attached to the product data. If no image is selected, the process continues without an image URL.

The product data, including the image URL if provided, is submitted to the backend service, which saves the product details to the database.

If the save operation is successful, the database confirms that the product has been saved, and the backend returns a success response to the web interface, which informs the user with a success message.

However, if saving the product fails, the backend service retrieves error details from the database and informs the web interface. If an image was uploaded, the system deletes the image from Cloudinary using the public ID. The user is then notified of the error. Figure 3.11 illustrates the creation of product with optional image upload

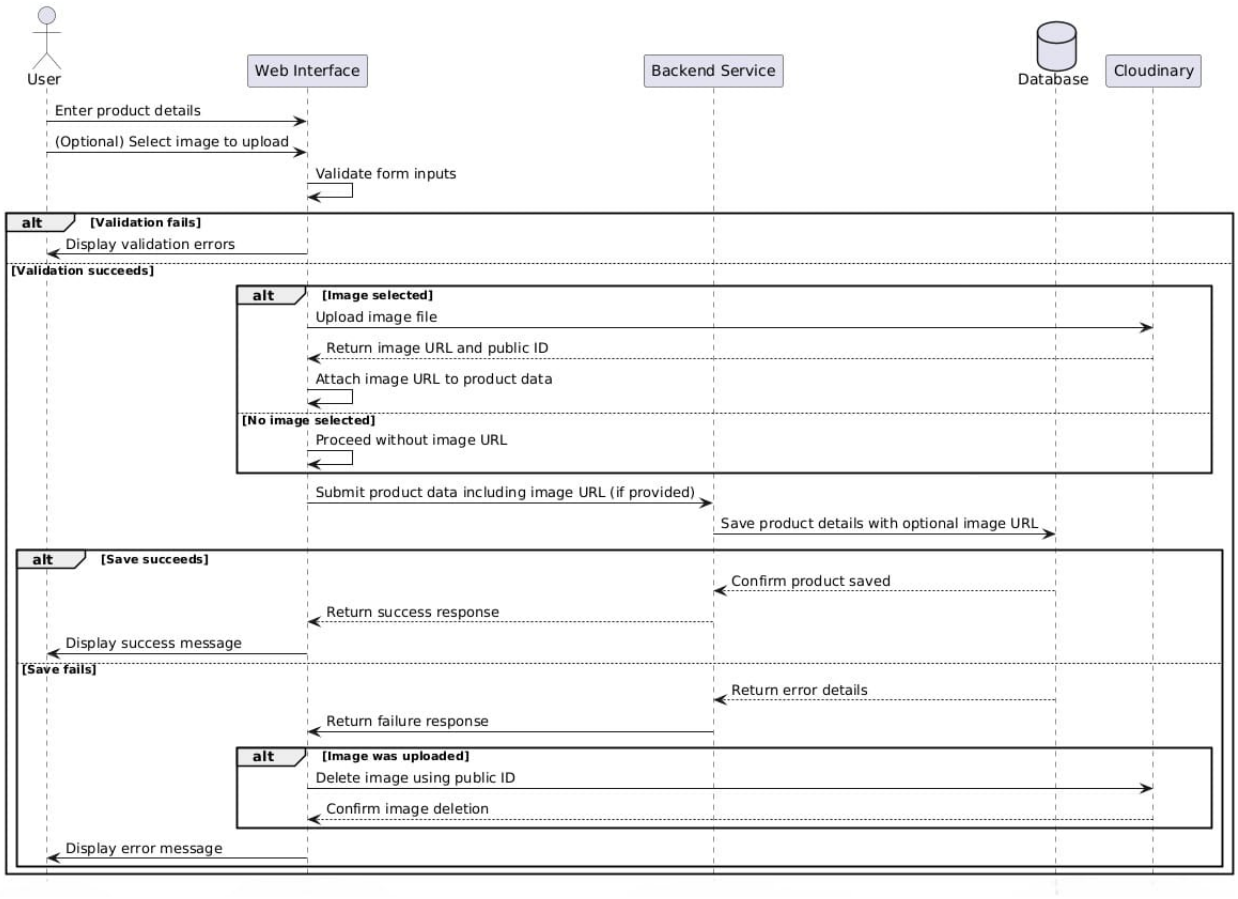


Figure 3.13: Product Creation with Optional Image Upload Sequence Diagram

3.8 Conclusion

In this chapter, we examined the design of the application. We began by presenting the overall architecture, which outlines the structure and interaction between various components of the system. This provided a foundational understanding of how the application is organized and how different elements work together. Next, we introduced class diagrams that detail the static structure of the application. These diagrams illustrate the key classes, their attributes, methods, and relationships, offering a clear view of the system's object-oriented design. Following this, we explored sequence diagrams that capture the dynamic interactions between components during specific scenarios. These diagrams depict the sequence of operations and messages exchanged between objects, highlighting the flow of data and control within the application. Overall, this chapter provided a comprehensive overview of the application's design, establishing a solid foundation for understanding its structure and behavior. This knowledge will facilitate the subsequent implementation and further development of the system.

CHAPTER 4

SPRINT 2: "USER MANAGEMENT"

4.1 Introduction

User Management is a core component of the application, enabling the administration of the system's various user roles. In this chapter, we will discuss the functionalities related to user account management, including the addition, modification, and deletion of supervisors, students, jury members, as well as the management of departments and their assigned leaders.

4.2 Analysis and Requirements Specification:

As with the previous chapter, this section will explore what's needed for user management, identifying all key requirements that will ensure the system serves its users effectively and helps us meet our objectives.

4.2.1 Sprint 2 Backlog:

In our second sprint, we'll roll out the various user management features for the system. Table 4.1 below shows what's planned in our sprint backlog.

Table 4.1: Sprint 2 – User Management

Sprint	**User Story**	**Actor(s)**
Sprint 2	Add new supervisor to list	Admin
Sprint 2	Delete a supervisor	Admin
Sprint 2	Update a supervisor	Admin
Sprint 2	Add a department	Admin
Sprint 2	,Add a department chair	Admin
Sprint 2	Edit a department	Admin
Sprint 2	Add a jury member	Admin
Sprint 2	Delete a jury member	Admin
Sprint 2	Update a jury member	Admin
Sprint 2	Add a student	Admin

Sprint	**User Story**	**Actor(s)**
Sprint 2	Update student details	Admin
Sprint 2	Delete a student	Admin

4.2.2 Functional Requirements Gathering

Managing our system's different users involves defining the functional requirements listed below:

- The Administrator can:
  - Manage supervisors:
    - Add new supervisor to supervisor's list.
    - Delete a supervisor.
    - Update a supervisor.
  - Manage departments :
    - Add a department.
    - Add a department chair.
    - Edit a department.
    - Delete a department.
  - Manage jury member :
    - Add a jury member .
    - Delete a jury member.
    - Update a jury member.
  - Manage students :
    - Add a student.
    - Update student details.
    - Delete a student.

4.3 Use Case Diagram

The figure 4.1 below shows the use case diagram for Sprint 2, which focuses on user management. This module lets us handle all user profiles (students, supervisors, jury members, etc.) and their access rights, building on the system interactions we described earlier.

4.4 Use Case Scenarios

This section elaborates on each use case scenario from the 'Manage Users' diagram through textual descriptions.

4.4.1 Use Case: "Create a New Supervisor"

Similarly, Table 4.2 provides the textual description of the 'Create a New Supervisor' use case.

Table 4.2: Textual Description of the Use Case "Create a New Project"

Element	**Description**
Primary Actor	Project Manager
Objective	To manage users effectively, creating supervisors is necessary to implement the system's management structure.
Basic Flow of Events	1. The project manager clicks on the "Add Supervisor" button. 2. The system displays the Supervisor creation form. 3. The project manager fills in the supervisor's first and last name,email, department,phone number, description, address,and the academic Title, then clicks the "Submit" button. 4. The system updates the database and refreshes the list of existing Supervisor.
Post-condition	The database is updated with the new supervisor.

4.4.2 Use Case: "Update a Supervisor"

Table 4.3 presents the textual description of the use case *Update a Supervisor*.

Table 4.3: Textual Description of the Use Case "Update a Supervisor"

Element	**Description**
Primary Actor	Project Manager
Objective	To modify an existing supervisor's information when necessary to ensure data accuracy and up-to-date supervisor details.

Element	**Description**
Basic Flow of Events	1. The project manager selects a supervisor from the supervisor list and he clicks on the “Edit Supervisor” button. 2. The system displays the supervisor’s existing details. 3. The project manager edits fields including the supervisor’s first name, last name, email, phone number, description, address, and academic title, then selects the department (specifically the department to which the supervisor belongs) before clicking the ’Update’ button. 4. The system updates the database and refreshes the list of supervisors.
Post-condition	The selected supervisor’s information is updated in the database.

4.4.3 Use Case: "Delete a Supervisor"

Table 4.4 presents the textual description of the use case *Delete a Supervisor*.

Table 4.4: Textual Description of the Use Case "Delete a Supervisor"

Element	**Description**
Primary Actor	Project Manager
Objective	To remove an outdated or unnecessary supervisor from the system to maintain a clean and relevant supervisor list.
Basic Flow of Events	1. The project manager selects a supervisor from the list. 2. The project manager clicks on "Delete". 3. The system removes the supervisor from the database.
Post-condition	The selected supervisor is permanently deleted from the database.

4.5 Design

Building on our previous chapter's structure, we'll now explore the user management design phase.

4.5.1 Class Diagram

The class diagram represents the static structure of the system in terms of classes and their relationships.

After conducting a full analysis of all use cases, we derived the class diagram shown in Figure 4.2. This diagram includes the following core models:

4.5.2 Sequence Diagram

Sequence diagrams are UML diagrams used to describe the chronological flow of interactions between objects in a software system. They help visualize how components communicate over time to complete a specific functionality.

In this section, we will present the refined sequence diagrams from sprint 2.

4.5.3 Sequence Diagram: "Create a Supervisor"

Figure 13 illustrates the sequence diagram for the functionality *Create a Supervisor*.

4.5.4 Sequence Diagram: "Update a Supervisor"

Figure 13 illustrates the sequence diagram for the functionality *Update a Supervisor*.

4.5.5 Sequence Diagram: "Delete a Supervisor"

Figure 13 illustrates the sequence diagram for the functionality *Delete a Supervisor*.

4.6 Realisation

4.7 Conclusion

Throughout this chapter, we outlined the design and implementation phases for all user management features (supervisors, students, jury members) and department management in our application. The next chapter will present our final sprint: Report Management.

CHAPTER 5

SPRINT 3 : “ REPORT MANAGEMENT ”

5.1 Introduction

This chapter details the development process for report management functionalities within our system. We begin by presenting the sprint backlog and associated functional requirements, which define the core tasks such as report creation, jury assignment, and document handling. Next, we analyze the system interactions through a use case diagram and scenario descriptions, capturing how students, supervisors, and jury members engage with the reporting features. The design phase then follows, where we outline the underlying structure using class and sequence diagrams to demonstrate how data flows during report uploads, downloads, and evaluations. Finally, we showcase key user interfaces that bring these technical designs to life, illustrating the practical implementation of each feature.

5.2 Analysis and Requirements Specification

As with previous chapters, this section will focus on gathering and analyzing the requirements for report management within the system.

5.2.1 Sprint 3 Backlog

Table 5.1 presents the various features related to report management that were implemented during the first sprint.

Table 5.1: Sprint 3 – Report Management

Sprint	**User Story**	**Actor(s)**
Sprint 3	Create a report field with project ID	Admin
Sprint 3	Assign a jury member to a report	Admin
Sprint 3	Upload a report	Student
Sprint 3	Download a report	Student, Supervisor, Jury Member

5.3 Functional Requirements Gathering

In this section, we will develop these specified report management features:

- The Administrator can:
  - Create a report field with project ID.
  - Assign a jury member to a report.
- The Supervisor or Student or jury member can:
  - Download a report.
- The Student can:
  - Upload a report Student.

5.4 Use Case Diagram

The use case diagram in Figure **??** below describes the different actions that the admin can perform to manage report

5.5 Use Case Scenarios

After presenting the *Manage Report* use case diagram, we now describe each scenario in detail to explain how the system works

5.5.1 Use Case: "Create a report field with project ID"

Table 5.2 presents the textual description of the use case *Create a report field with project ID*.

Table 5.2: Textual Description of the Use Case "Create a New Report field"

Element	**Description**
Primary Actor	Project Manager
Objective	To manage report effectively, the project manager must first a report field to organize the report by project ID.
Basic Flow of Events	1. 2. 3. 4.
Post-condition	The database is updated with the new report field.

5.5.2 Use Case: "Upload a report"

Table 5.3 presents the textual description of the use case *Upload a report*.

Table 5.3: Textual Description of the Use Case "Upload a report"

Element	**Description**
Primary Actor	Student
Objective	To upload a report, the student must submit their project report before the deadline.
Basic Flow of Events	1. 2. 3. 4.

Element	**Description**
Post-condition	The database is updated with the uploaded report.

5.6 Design

The design phase consists of refining the descriptions made during the analysis. It aims to build a stable architecture for the system to be implemented. This is a crucial and essential step to move toward the implementation phase, as it defines a structured approach to developing a reliable and scalable product.

In this section, we'll explore the class and sequence diagrams that bring our third sprint's functionality to life.

5.6.1 Class Diagram

The class diagram represents the static structure of the system in terms of classes and their relationships.

After analyzing all the use cases, we've created the class diagram in Figure 5.2 - it captures our system's core building blocks, including:

5.6.2 Sequence Diagram

In this section, we'll use sequence diagrams to map out how users interact with the report management system—showing the step-by-step workflows for students, supervisors, and jury members.

5.6.3 Sequence Diagram: "Create a report field with project ID"

Figure 5.3 presents the sequence diagram for the functionality *Create a report field with project ID*.

5.6.4 Sequence Diagram: "Upload Report"

Figure 5.4 presents the sequence diagram for the functionality *Upload Report*.

5.7 Realisation

5.8 Conclusion

Through this chapter, we’ve seen how Sprint 3 brings report management to life,from students submitting work to juries evaluating it.

CONCLUSION

The development of SellPoint was aimed at creating an efficient, user-friendly tool that allowed us to apply our academic knowledge in a practical context, ultimately delivering a valuable resource to the educational community. Throughout this project, we successfully designed and implemented a robust web application, utilizing modern technologies such as Angular for the frontend and FastAPI for the backend. This technology stack has provided an intuitive and efficient user experience for managing employees, customers, orders, and inventory. This project not only enhanced our practical skills but also honed our critical thinking, research capabilities, and system design proficiency. We utilized use cases, class diagrams, and interface design to shape the architecture of the application. Furthermore, the collaborative nature of this project allowed us to strengthen our communication and teamwork skills. Looking ahead, there are promising opportunities for further development. A key priority is the integration of a chat application to streamline communication between departments within the company. This feature would enhance internal collaboration, contributing to a more connected and efficient workplace.g

Establishing a continuous feedback loop from users will be crucial for refining the system and ensuring it adapts to evolving market demands. As technology continues to transform the retail landscape, the adaptability of this POS system will be key to helping businesses thrive. This project not only lays a solid foundation for advancing POS technology but also underscores the critical importance of innovation in meeting the dynamic needs of retailers and their customers. The insights gained throughout this project will serve as a valuable resource for future initiatives aimed at revolutionizing retail management practices.

REFERENCES

- [1] *Trello - Project Management Tool:* **https://trello.com**
- [2] *Jira - Agile Project Management Tool:* **https://www.atlassian.com/software/jira**
- [3] *Moodle - Learning Management System (LMS):* **https://moodle.org**
- [4] *Agile Methodology:* **https://www.agilealliance.org/agile101/**
- [5] *Scrum Framework:* **https://www.scrum.org**
- [6] *React and ASP.NET MVC Web App:* **https://dev.to/thisdotmedia_staff/adding-react-to-your-asp-net-mvc-web-app-4ilm**
- [7] *Csharp Language:* **https://learn.microsoft.com/en-us/dotnet/csharp/**
- [8] *.NET Framework:* **https://learn.microsoft.com/en-us/dotnet/framework/**
- [9] *React Native:* **https://reactnative.dev**
- [10] *PostgreSQL:* **https://www.postgresql.org**
- [11] *Swagger (OpenAPI):* **https://swagger.io**
- [12] *Cursor IDE:* **https://www.cursor.so**
- [13] *GitHub:* **https://github.com**