Perfumery

Martín Alejandro Castro Álvarez

martincastro.10.5@gmail.com

Universidad Internacional de Valencia (VIU)

Calle Pintor sorolla, 21 46002, Valencia (España)

May 2024

Abstract. This report outlines the development of an intranet system for a perfumery, aiming to enhance operational efficiency through modern software engineering principles and cloud-based technologies. The system, built using Java Spring Boot for backend development and ReactJS for the user interface, is hosted on AWS. It adopts a hexagonal architecture to ensure clean separation of concerns. PostgreSQL is selected as the database service. The most important features include user registration, session management, role-based access control, logging, and tracing for reliability and maintainability. The application also implements a RESTful API following commonly accepted design patterns. Deployment is managed using Docker Compose.

Keywords: Java, Spring Boot, ReactJS, AWS, Hexagonal Architecture, RESTful API, PostgreSQL, Docker Compose, Cloud Computing, Role-Based Security, Exception Handling, Logging

1. Introduction

1.1. Goal

This report describes the development of an intranet system for a perfumery, focusing on planning, design, and implementation using Java Spring Boot. It details the project's approach, including division of labor, timelines, and roles. In addition, it covers the selection of AWS for cloud-based data storage and illustrates the system's architecture with class diagrams, use cases, and database schemas. It also describes the graphical user interface implemented in ReactJS and how it uses the hexagonal architecture. Key features such as user registration, time tracking, and HR communications are included, along with elegant exception handling.

1.2. Division of Labor

In an ideal project, backend engineers would focus on Java Spring Boot for server-side components, while frontend engineers would handle user interface development using React. At the same time, DevOps engineers would manage AWS services and manage the deployment process, and QA Engineers would be responsible for testing. Additionally, the Project Manager would orchestrate these activities, ensuring efficient collaboration, adherence to timelines, and removing blockers (very much like the scheduler of a CPU does). Additionally, a Software Architect or Senior Engineer, in collaboration with the Project

Manager, would develop the Entity-Relationship Diagram (ERD) and Use Case Diagrams based on requirements provided by the Product Manager. Unfortunately, in this project, the diverse responsibilities of all these roles are managed and executed by a single individual. Tasks are assigned to each individual role as described by Table 5.

2. Background Theory

2.1. Use Case Diagram

It provides a graphical representation of how users (actors) interact with a system to achieve specific goals. It describes various system interactions using symbols such as ovals for use cases, stick figures for actors, and lines for relationships. [2].

2.2. Entity Relationship Diagram (ERD)

This diagram illustrates how "entities" such as people, objects, or concepts relate within a system. It is commonly used in designing relational databases and employs standardized symbols like rectangles for entities, diamonds for relationships, and ovals for attributes. [3].

2.3. PostgreSQL

It is a powerful open-source relational database management system (RDBMS) known for its strong compliance with SQL standards and support for advanced queries, foreign keys, triggers, materialized views, transactions, and concurrency control. It is highly scalable both in data management and user support. [4].

2.4. Java Spring Boot

It is an extension of the Spring framework that simplifies the development of new Java applications, providing built-in solutions for common application use cases such as backend servers, metric monitoring, health checks, and flexible configuration. [5].

2.5 Hexagonal Architecture

This design pattern organizes the application into a central core containing business logic, surrounded by ports and adapters that manage input and output. Each port represents a primary or secondary operation of the application, connected to an adapter that translates between the port and external components. [6]. This architecture is represented in Fig. 1.

2.6 REST API

It is a set of guidelines for creating web services, emphasizing stateless communication between consumers and providers of services, where each call from the client to the server contains all necessary information to understand the request. [7].

2.7 ReactJS

It is a declarative JavaScript library for building user interfaces, enabling developers to create large web applications that can change data without reloading the page [8]. The sign up page is shown in Fig. 11 whereas the login page is shown in Fig. 12. The users management page is shown in Fig. 13.

2.8 AWS

It is a cloud platform providing infrastructure services like computing power, storage options, and networking, as well as features like machine learning, artificial intelligence, data lakes, and big data analytics. [9].

2.9 Docker Compose

It uses a YAML file format to configure application services, networks, and volumes, simplifying the management of multi-container Docker applications [10].

3. Methodology

3.1. Planning

The development of the application begins with planning tasks as outlined in this section. These tasks include defining use cases, presented in Table 1 and illustrated by Figures 2 to 9. The use cases are derived directly from the requirements provided by the Product Manager. Additionally, the Entity-Relationship Diagram (ERD) is displayed in Figure 10 and further detailed in Table 2. Table 3 lists the API endpoints, designed following a RESTful approach, based on the use cases in Table 1.

3.2. Data Layer

PostgreSQL was selected as the database system for its robustness, scalability, and capability to manage complex queries effectively. It is supported by extensive documentation and a strong community. Security and privacy are ensured through a role-based security model, detailed in Table 4, which controls access by assigning specific roles to users, thereby protecting sensitive data and functionalities from unauthorized access and ensuring a secure operational environment.

3.4. Application Layer

The presentation layer is crafted using ReactJS, focusing on creating a user-friendly interface that supports single-page interactions and efficient state management. The choice of ReactJS leverages its declarative nature to optimize updates and rendering of components that change, ensuring a responsive user experience. For end-to-end testing, Cypress is used to verify the functional integrity and performance of the frontend by simulating real user interactions.

3.3. Presentation Layer

The application relies on Java Spring Boot to facilitate the construction of robust server-side components. The design follows a hexagonal architecture model to enhance modularity and maintainability by separating the core logic from external interfaces. Custom exceptions are used for exception handling to ensure reliability and service quality, while logging helps track operations, aiding in debugging and performance tuning. The development lifecycle includes unit tests to verify consistent behavior of components across updates.

3.8. Deployment

The deployment of the perfumery system is streamlined using Docker Compose, which manages multi-container Docker applications with a single command shown in (1). This method ensures consistent and reliable deployment of all system components, including backend services, databases, and frontend applications, across various environments. The application is hosted on AWS EC2 instances for scalable cloud computing, and AWS Elastic IP provides a static IPv4 address, while AWS Route53 makes the application accessible online.

```
docker-compose up
```
(1)

4. Results

4.1 Resources

The application is available both as open-source code and as a live deployment. The source code is accessible on GitHub at (2), inviting developers and interested parties to review, fork, and contribute to its continuous enhancement. The live site can be temporarily visited at (3), demonstrating the fully functional intranet system implemented with hexagonal architecture and built using the outlined technologies.

```
https://github.com/MartinCastroAlvarez/hexagonal-spring-boot
```
(2)

```
https://hex.martincastroalvarez.com/
```
(3)

4.2 Conclusion

During this project, I gained extensive insights into the entire software development process. From planning with use cases and diagrams to implementing databases with PostgreSQL and programming with Java Spring Boot and ReactJS, each phase honed my technical skills. Deploying with Docker Compose and using AWS services gave me practical cloud computing experience. Additionally, learning about hexagonal architectures was a new and valuable aspect, enhancing my skills for future challenges in software development.

5. References

[1] Clotet, R. (2024). Proyectos de programación. Grado en Ingeniería Informática, Programación de Computadores.

[2] Lucidchart. (2024). "UML Use Case Diagram Tutorial". https://www.lucidchart.com/pages/uml-use-case-diagram

[3] Lucidchart. (2024). "What is an Entity Relationship Diagram (ERD)?". https://www.lucidchart.com/pages/er-diagrams

[4] PostgreSQL Global Development Group. (2024). Concepts. https://www.postgresql.org/docs/current/tutorial-concepts.html

[5] VMware Tanzu. (2024). Spring Boot. https://spring.io/projects/spring-boot

[6] Huseynov, M. (2022). Hexagonal Architecture With Spring Boot. Medium. https://muraddev.medium.com/hexagonal-architecture-with-spring-boot-342dc7a17d48

[7] Red Hat. (2020). What is a REST API?. https://www.redhat.com/en/topics/api/what-is-a-rest-api

[8] React. (n.d.). React: A JavaScript library for building user interfaces. https://legacy.reactjs.org

[9] Amazon Web Services, Inc. (n.d.). Computación en la nube con AWS. https://aws.amazon.com/es/what-is-aws/

[10] Docker, Inc. (n.d.). Docker Compose overview. https://docs.docker.com/compose/

6. Appendix

6.1 Tables

Code	Use Case	Roles
1	**Session Management Use Case**	
1.1	Sign Up with username and password.	Anonymous User
1.2	Login with username and password.	Anonymous User
1.3	View session profile details.	All Users
1.4	Update session profile details.	All Users
1.5	Recover password from email.	Anonymous User
2	**Account Management Use Case**	
2.1	List users by name, signup date, and last login date	Admin, Manager
2.2	Search users by name, email, and role	Admin, Manager
2.3	Update user details	Admin
2.4	Deactivate user	Admin
2.5	Activate user	Admin
3	**Role Management Use Case**	
3.1	List managers by name, signup date, and last login date	Admin
3.2	Name new manager	Admin
3.3	Remove Manager	Admin
3.4	List salesmen by name, signup date, and last login date	Manager
3.5	Name a new salesman	Manager
3.6	Remove a salesman	Manager
3.7	List providers by name, signup date, and last login date	Manager
3.8	Name a new provider	Manager
3.9	Remove a provider	Manager
4	**Schedule Management Use Case**	

4.1	List schedule rules by user	All Users
4.2	Add a new schedule to a user	Manager
4.3	Remove a schedule from a user	Manager
4.4	List time off by user and day range	All Users
4.5	Add a new time off to a user	Manager
4.6	Remove a time off from a user	Manager
4.7	Check-in	All Users
4.8	Check-out	All Users
4.9	List checked-in time by user, and date range	All Users
5	**Meeting Management Use Case**	
5.1	List meetings by date	All Users
5.2	Create a new meeting	Manager
5.3	Delete an existing meeting	Manager
5.4	Update existing meeting	Manager
5.5	Upload a file to a meeting	Manager
5.6	List invited users	All Users
5.7	Invite users to a meeting	Manager
5.8	Remove users from a meeting	Manager
6	**Message Management Use Case**	
6.1	List Messages	All Users
6.2	Create a new message	Admin, Manager
6.3	Update existing message	Admin, Manager
6.4	Add recipient user to an unsent message	Admin, Manager
6.5	Add recipient role to an unsent message	Admin, Manager
6.6	List message recipients	All Users
6.7	Send an unsent message	Admin, Manager
6.8	Delete unsent messages	Admin, Manager
7	**Product Management Use Case**	

7.1	List products by code, name, and quantity	All Users
7.2	Search products by name, and code	All Users
7.3	Create a product	Manager
7.4	Delete a product	Manager
7.5	Update product details	Manager
8	**Stock Management Use Case**	
8.1	Sell a product	Salesman
8.2	List sales by product, date range, price, and salesman	Salesman, Manager
8.3	Add stock to an existing product	Provider
8.4	List purchases by product, date, cost, range and provider	Provider, Manager

Table 1: Use cases in the Perfumery application.

Attribute	Type	Visibility
Product		
id	Integer	private
name	String	private
isActive	Bool	private
price	Double	private
Transaction		
id	Integer	private
amount	Double	private
datetime	DateTime	private
product	Product	protected
Sale (extends Transaction)		
cost	Double	private
salesman	User	private
Purchase (extends Transaction)		
cost	Double	private
provider	User	private
Schedule		
id	Integer	private
startTime	Time	private
endTime	Time	private
dayOfWeek	Enum {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}	private
user	User	private
Pto		
id	Integer	private
day	Date	private

type	Enum {VACATION, SICKNESS, PERSONAL}	private
user	User	private
Work		
id	Integer	private
startTime	DatetTime	private
endTime	DatetTime	private
user	User	private
User		
id	Integer	private
name	String	private
email	String	private
passwordHash	String	private
lastLoginDate	DateTime	private
signUpDate	DateTime	private
isActive	Boolean	private
role	Enum {USER, ADMIN, MANAGER, SALESMAN, PROVIDER}	private
Message		
id	Integer	private
subject	String	private
text	String	private
sender	User	private
creationDate	DateTime	private
sentAt	DateTime	private
recipients	List<User>	private
Meeting		
id	Integer	private
title	String	private

`date`	`DateTime`	`private`
`files`	`List<File>`	`private`
`invitees`	`List<User>`	`private`
`File`		
`id`	`Integer`	
`fileName`	`String`	
`fileHash`	`String`	

Table 2: Entities, attributes, and attribute types in this application. Models implement the getter and setter design pattern.

Method	Endpoint	Description	Roles
POST	/auth/signup	Sign up with username and password	Anonymous User
POST	/auth/login	Login with username and password.	Anonymous User
GET	/auth/profile	View session profile details.	All Users
GET	/auth/schedule	List session profile schedule rules	All Users
GET	/auth/work	List session user work time.	All Users
GET	/auth/sales	List session user sales.	All Users.
GET	/auth/purchases	List session user purchases	All Users.
GET	/auth/meetings	List session user meetings	All Users.
GET	/auth/files	List session user files.	All Users.
GET	/auth/pto	List session profile paid time off	All Users
PUT	/auth/profile	Update session profile details.	All Users
POST	/auth/reset	Recover password via email.	Anonymous User
GET	/users	List users by name, signup date, and last login date. Search users by name, email, and role.	Admin, Manager
PUT	/users/{userId}	Update user details by id.	Admin
DELETE	/users/{userId}/status	Deactivate user by id.	Admin
POST	/users/{userId}/status	Activate user by id	Admin
GET	/managers	List managers by name, signup date, and last login date.	Admin

POST	/managers/{userId}	Name new manager by user id.	Admin
DELETE	/managers/{userId}	Remove manager by user id.	Admin
GET	/salesmen	List salesmen by name, signup date, and last login date.	Manager
POST	/salesmen/{userId}	Name new salesman by user id.	Manager
DELETE	/salesmen/{userId}	Remove salesman by user id.	Manager
GET	/providers	List providers by name, signup date, and last login date.	Manager
POST	/providers/{userId}	Name new provider by user id.. (Manager).	Manager
DELETE	/providers/{userId}	Remove provider by user id.	Manager
GET	/schedule	List schedule rules.	All Users
GET	/pto	List PTO	All Users
GET	/work	List Work	All Users
GET	/users/{userId}/schedule	List schedule rules by user.	All Users
POST	/users/{userId}/schedule	Add a new schedule rule to a user.	Manager
DELETE	/users/{userId}/schedule/{ruleId}	Delete schedule rule from user.	Manager
GET	/users/{userId}/pto	List time off by user, and date range.	All Users
POST	/users/{userId}/pto	Add time off to an user.	Manager
DELETE	/users/{userId}/pto/{ptoId}	Delete time off from an user.	Manager
POST	/checkin	Check-in	All Users
POST	/checkout	Check-out	All Users

GET	/meetings	List meetings by date, and title.	All Users
POST	/meetings	Create a new meeting	Admin, Manager
DELETE	/meetings/{meetingId}	Delete an incoming meeting	Admin, Manager
PUT	/meetings/{meetingId}	Update incoming meeting	Admin, Manager
GET	/meetings/{meetingId}/files	List meeting files.	All Users
POST	/meetings/{meetindId}/files	Upload a file to an incoming meeting.	Manager
DELETE	/meetings/{meetingId}/files/{fileId}	Delete file from an incoming meeting.	Manager
GET	/meetings/{meetingId}/users	List invited users.	All Users
POST	/meetings/{meetingId}/users/{userId}	Invite an user to an incoming meeting	Manager
DELETE	/meetings/{meetingId}/users/{userId}	Delete an user from an incoming meeting	Manager
GET	/messages	List messages by date, and title.	All Users
POST	/messages	Create a new message	Admin, Manager
PUT	/messages	Update an unsent message	Admin, Manager
POST	/messages/{messageId}	Send an unsent message	Admin, Manager
DELETE	/messages/{messageId}	Delete an unsent message	Admin, Manager
PUT	/messages/{messageId}	Update unsent message.	Admin, Manager
GET	/messages/{messageId}/files	List message files.	Manager
POST	/messages/{messageId}/files	Upload a file to an unsent message.	Manager
DELETE	/messages/{messageId}/files/{fileId}	Delete file from an unsent message.	Manager

GET	/messages/{messageId}/users	List destination users.	Manager
POST	/messages/{messageId}/users	Add an user to an unsent message.	Manager
DELETE	/messages/{messageId}/users/{userId}	Delete an user from an unsent message.	Manager
GET	/products	List products by code, name, and quantity. Search by name, and code.	All Users
POST	/products	Create a product (Manager).	Manager
PUT	/products/{productId}	Update product details (Manager).	Manager
DELETE	/products/{productId}	Delete product	Manager
GET	/sales	List sales by date range, price.	Salesmen, Manager
GET	/products/{productId}/sales	List sales by product, date range, and price.	Salesmen, Manager
GET	/users/{userId}/sales	List sales by salesman, date range, and price.	Salesmen, Manager
POST	/sales/{saleId}	Sell a product.	Salesman
GET	/purchases	List purchases by product, date range, price, and provider	Provider, Manager.
GET	/products/{productId}/purchases	List purchases by date range, price, and provider	Provider, Manager
GET	/users/{userId}/purchases	List purchases by provider, price, and provider	Provider, Manager
POST	/purchases/{purchaseId}	Add stock to an existing product	Provider

Table 3: This table shows the REST API endpoints in this application.

Role	Authorized Action
`Anonymous User`	
	Can sign up with a username and password.
	Can log in with a username and password.
	Can recover their password via email.
`User`	
	Can view and update their session profile details.
	Can list schedule lines.
	Can check-in and check-out.
	Can list meetings by date.
	Can list invited users to meetings.
	Can list messages and their recipients.
	Can list products by code, name, and quantity.
	Can search products by name and code.
	Can list checked-in time by user, and date range.
`Admin`	
	Can list, search, update, and deactivate users.
	Can list and manage Managers (name new, remove).
	Can create, update, and delete messages, including adding recipient users and roles.
	Can send and delete unsent messages.
`Manager`	
	Can list, search, update, and deactivate users.
	Can list Salesmen and Providers by name, signup date, and last login date.
	Can add and remove Salesmen and Providers.
	Can add and remove schedules and time-offs for users.
	Can create, delete, update meetings, upload files to meetings, invite and remove users from meetings.

	Can create, update, delete messages, including adding recipient users and roles.
	Can create, delete, and update product details.
	Can list sales by product, date range, price, and salesman.
	Can list purchases by product, date, cost, range, and provider.
`Salesman`	
	Can sell products
	Can list sales by product, date range, price, and themselves.
`Provider`	
	Can add stock to an existing product.
	Can list purchases by product, date, cost, range, and themselves.

Table 4: This table describes the actions that each role is authorized to execute in the application.

Task #	Description	Assignee	Date
`Milestone 1: Setup project`			
`001`	Create a GitHub repository.	`DevOps`	`03 May`
`002`	Install Java Spring Boot.	`DevOps`	`03 May`
`Milestone 2: Create domain classes`			
`003`	Create a User model.	`Backend Engineer`	`03 May`
`004`	Create a Product model.	`Backend Engineer`	`03 May`
`005`	Create a Message model.	`Backend Engineer`	`03 May`
`006`	Create a Meeting model.	`Backend Engineer`	`03 May`
`007`	Create a File model.	`Backend Engineer`	`03 May`
`008`	Create a Sale model.	`Backend Engineer`	`03 May`
`009`	Create a Purchase model.	`Backend Engineer`	`03 May`
`010`	Create a Schedule model.	`Backend Engineer`	`03 May`
`011`	Create a Pto model.	`Backend Engineer`	`03 May`

012	Create a Work model.	Backend Engineer	03 May
013	Create a User repository.	Backend Engineer	03 May
014	Create a Product repository.	Backend Engineer	03 May
015	Create a Message repository.	Backend Engineer	03 May
016	Create a Meeting repository.	Backend Engineer	03 May
017	Create a File repository.	Backend Engineer	03 May
018	Create a Sale repository.	Backend Engineer	03 May
019	Create a Purchase repository.	Backend Engineer	03 May
020	Create a Schedule repository.	Backend Engineer	03 May
021	Create a Pto repository.	Backend Engineer	03 May
022	Create a Work repository.	Backend Engineer	03 May
Milestone 3: Create database adapters			
023	Add Docker Compose to the project.	DevOps	04 May
024	Add a PostgreSQL database.	DevOps	04 May
025	Install JPA in the Spring Boot application.	DevOps	04 May
026	Create a User JPA entity.	Backend Engineer	04 May
027	Create a Product JPA entity.	Backend Engineer	04 May
028	Create a Message JPA entity.	Backend Engineer	04 May
029	Create a Meeting JPA entity.	Backend Engineer	04 May
030	Create a File JPA entity.	Backend Engineer	04 May
031	Create a Sale JPA entity.	Backend Engineer	04 May
032	Create a Purchase JPA entity.	Backend Engineer	04 May
033	Create a Schedule JPA entity.	Backend Engineer	04 May
035	Create a Pto JPA entity.	Backend Engineer	04 May
036	Create a Work JPA entity.	Backend Engineer	04 May
037	Create a User JPA entity.	Backend Engineer	04 May
038	Create a Product JPA repository.	Backend Engineer	04 May
039	Create a Message JPA repository.	Backend Engineer	04 May

040	Create a Meeting JPA repository.	Backend Engineer	04 May
041	Create a File JPA repository.	Backend Engineer	04 May
042	Create a Sale JPA repository.	Backend Engineer	04 May
042	Create a Purchase JPA repository.	Backend Engineer	04 May
043	Create a Schedule JPA entity.	Backend Engineer	04 May
044	Create a Pto JPA entity.	Backend Engineer	04 May
045	Create a Work JPA entity.	Backend Engineer	04 May
Milestone 5: Create core business logic			
046	Setup logging.	DevOps	05 May
047	Create User service.	Backend Engineer	05 May
048	Create Product service.	Backend Engineer	05 May
049	Create Message service.	Backend Engineer	05 May
050	Create Meeting service.	Backend Engineer	05 May
051	Create Schedule service.	Backend Engineer	05 May
052	Create User application.	Backend Engineer	05 May
053	Create Product application.	Backend Engineer	05 May
054	Create Message application.	Backend Engineer	05 May
055	Create Meeting application.	Backend Engineer	05 May
056	Create Schedule application.	Backend Engineer	05 May
Milestone 6: Protect the application			
057	Install security dependencies in Spring Boot	DevOps	06 May
058	Create Authentication service.	Backend Engineer	06 May
059	Create Authentication application.	Backend Engineer	06 May
060	Create Authentication controller.	Backend Engineer	06 May
Milestone 6: Create API adapter			
060	Setup custom exceptions.	DevOps	07 May
061	Create User API controller.	Backend Engineer	07 May
062	Create User API controller.	Backend Engineer	07 May

063	Create User API controller.	Backend Engineer	07 May
064	Create User API controller.	Backend Engineer	07 May
065	Create User API controller.	Backend Engineer	07 May
Milestone 7: Setup Frontend			
066	Install React	DevOps	08 May
067	Install TypeScript	DevOps	08 May
068	Install React Router	DevOps	08 May
069	Install Tailwind	DevOps	08 May
070	Create main layout	Frontend Engineer	08 May
071	Create navigation bar	Frontend Engineer	08 May
Milestone 8: Connect Frontend and Backend			
072	Install React Redux	DevOps	09 May
073	Install Axios	DevOps	09 May
074	Setup Authentication Store	Frontend Engineer	09 May
075	Add an Axios interceptor for the JWT token.	Frontend Engineer	09 May
076	Add an Axios interceptor for API exceptions.	Frontend Engineer	09 May
077	Create login form	Frontend Engineer	09 May
078	Create sign up form	Frontend Engineer	09 May
079	Setup authentication using Local Storage.	Frontend Engineer	09 May
080	Add toast messages.	Frontend Engineer	09 May
081	Add logout button.	Frontend Engineer	09 May
Milestone 8: Homepage			
082	Create a Homepage.	Frontend Engineer	10 May
083	Load User profile.	Frontend Engineer	10 May
084	Load User schedule.	Frontend Engineer	10 May
085	Load User PTO.	Frontend Engineer	10 May
086	Load User work history.	Frontend Engineer	10 May
087	Add Check-In button.	Frontend Engineer	10 May

088	Add Check-Out button.	Frontend Engineer	10 May
089	Add exception handling	Frontend Engineer	10 May
090	Add messages list.	Frontend Engineer	10 May
091	Add meetings list.	Frontend Engineer	10 May
092	Add button to update name.	Frontend Engineer	10 May
Milestone 9: Schedule page			
093	Add schedule list.	Frontend Engineer	11 May
094	Add schedule creation form.	Frontend Engineer	11 May
095	Add schedule deletion button.	Frontend Engineer	11 May
097	Add PTO list.	Frontend Engineer	12 May
098	Add PTO creation form.	Frontend Engineer	12 May
099	Add PTO deletion button.	Frontend Engineer	12 May
Milestone 10: Users Page			
099	Add Users list.	Frontend Engineer	13 May
100	Add User details modal.	Frontend Engineer	13 May
101	Add button to activate a User.	Frontend Engineer	13 May
102	Add button to deactivate a User	Frontend Engineer	13 May
103	Add button to promote to Manager.	Frontend Engineer	13 May
104	Add button to promote to Salesman.	Frontend Engineer	13 May
105	Add button to promote to Provider.	Frontend Engineer	13 May
106	Add button to demote a Manager.	Frontend Engineer	13 May
107	Add button to demote a Salesman.	Frontend Engineer	13 May
108	Add button to demote a Provider.	Frontend Engineer	13 May
109	Add user schedule list.	Frontend Engineer	14 May
110	Add user PTO list.	Frontend Engineer	14 May
111	Add user work list.	Frontend Engineer	14 May
112	Add user sales list.	Frontend Engineer	14 May
113	Add user purchases list.	Frontend Engineer	14 May

114	Add user messages list.	Frontend Engineer	14 May
115	Add user meetings list.	Frontend Engineer	14 May
Milestone 11: Meetings Page			
116	Add meetings list.	Frontend Engineer	15 May
117	Add button to delete a meeting.	Frontend Engineer	15 May
118	Add meeting creation form.	Frontend Engineer	15 May
119	Add button to upload a file to a meeting.	Frontend Engineer	15 May
120	Add meeting invitees list.	Frontend Engineer	15 May
121	Add form to invite a user.	Frontend Engineer	15 May
122	Add button to remove a user from a meeting.	Frontend Engineer	15 May
Milestone 12: Messages Page			
123	Add messages list	Frontend Engineer	16 May
124	Add format to create a new message draft	Frontend Engineer	16 May
125	Add a button to delete a message.	Frontend Engineer	16 May
127	Add a button to send a message.	Frontend Engineer	16 May
128	Add a list of recipients.	Frontend Engineer	16 May
129	Add form to add a user as recipient.	Frontend Engineer	16 May
130	Add button to remove a recipient.	Frontend Engineer	16 May
Milestone 12: Products Page			
131	Add products list	Frontend Engineer	17 May
132	Add form to create a product.	Frontend Engineer	17 May
133	Add form to update a product name,	Frontend Engineer	17 May
134	Add button to delete a product.	Frontend Engineer	17 May
135	Add form to update a product price.	Frontend Engineer	17 May
136	Add list of sales per product.	Frontend Engineer	17 May
137	Add a list of purchases per product.	Frontend Engineer	17 May
138	Add form to create a sale.	Frontend Engineer	17 May
139	Add form to create a purchase.	Frontend Engineer	17 May

Milestone 13: Deployment			
140	Create an AWS account.	DevOps	18 May
141	Create an AWS EC2 instance.	DevOps	18 May
142	Create an AWS ElasticIP.	DevOps	18 May
143	Create an AWS Route53 Domain.	DevOps	18 May
144	Create an AWS Route53 CNAME.	DevOps	18 May
145	Create an AWS S3 bucket.	DevOps	18 May
146	Create an AWS CloudFront distribution.	DevOps	18 May
147	Configure an AWS CloudFront origin.	DevOps	18 May
148	Build and publish the React application.	DevOps	19 May
149	Build a Dockerfile.	DevOps	19 May
150	Install Docker Compose in the EC2 instance.	DevOps	19 May
152	Deploy the Java Spring Boot application to AWS.	DevOps	19 May
153	Execute user acceptance tests	QA Engineer	19 May

Tabla 5: This table shows the REST API endpoints in this application.

6.2 Figures

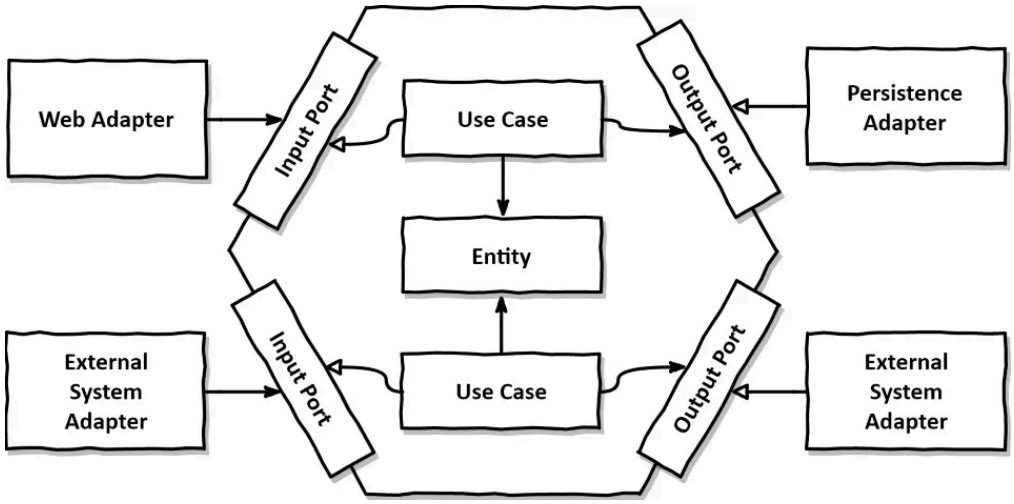

Fig. 1: Visual representation of an hexagonal architecture. Source: [6].

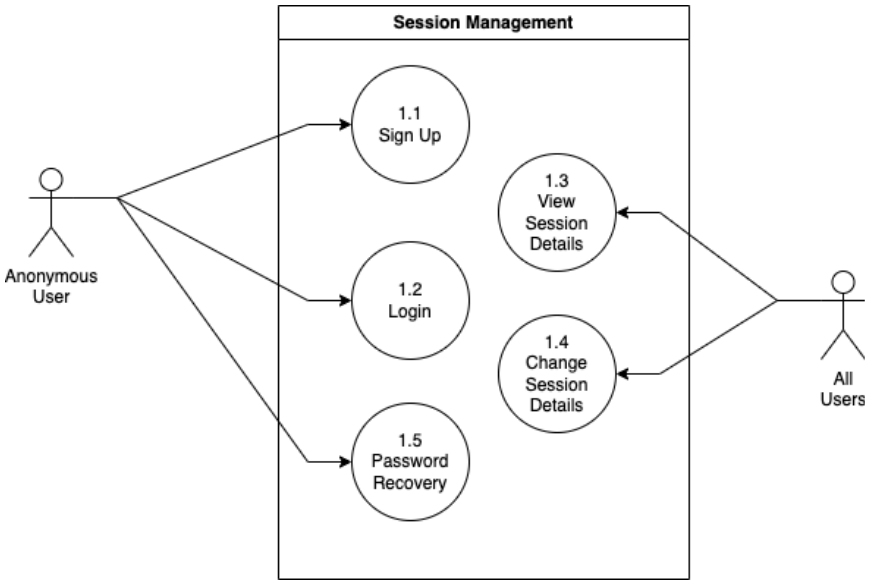

Fig. 2: Session Management Use case diagram.

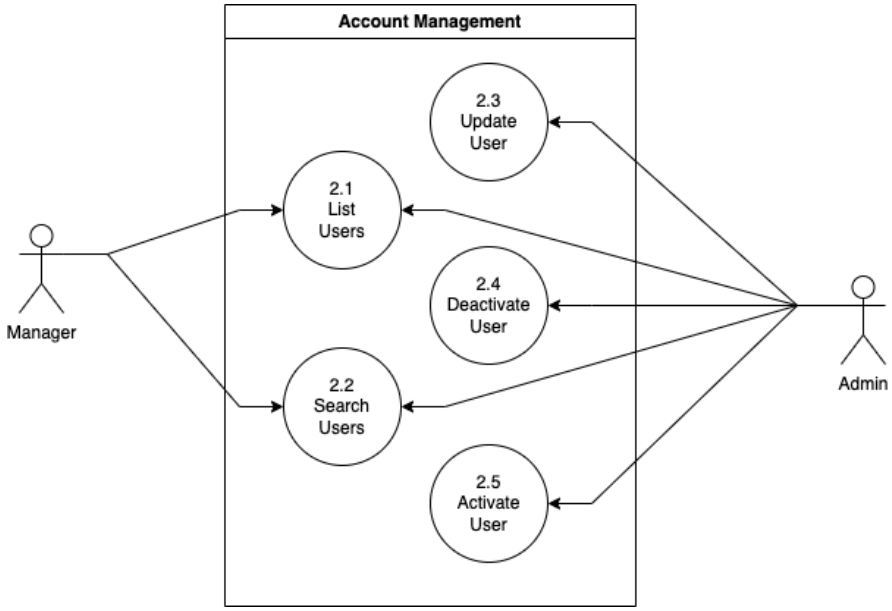

Fig. 3: Session Management Use case diagram.

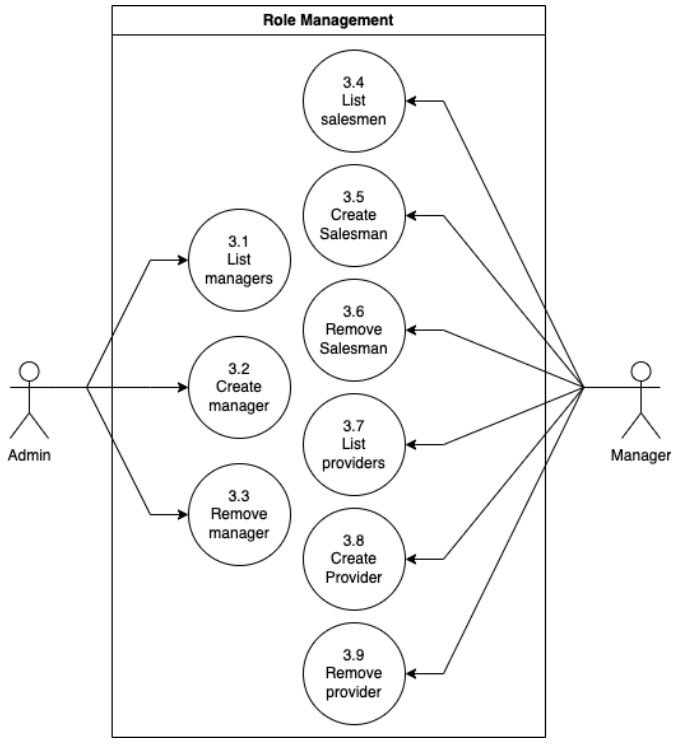

Fig. 4: Role Management Use case diagram.

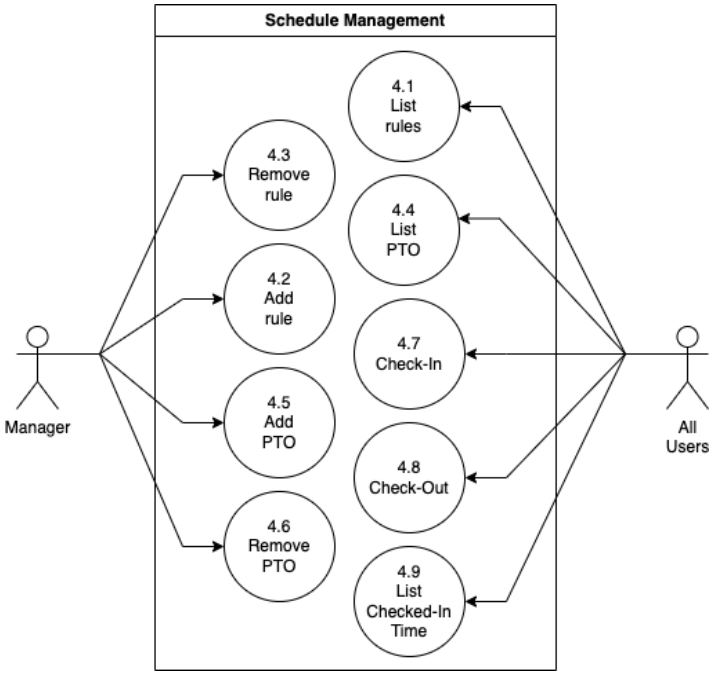

Fig. 5: Schedule Management Use case diagram.

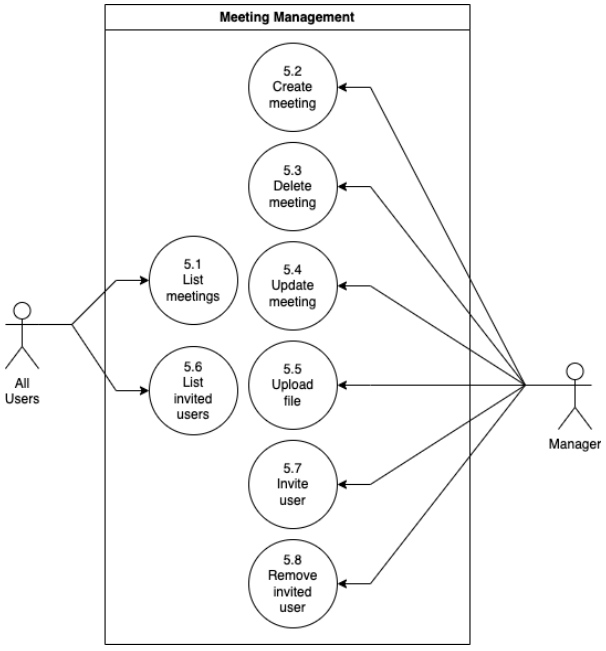

Fig. 6: Meeting Management Use case diagram.

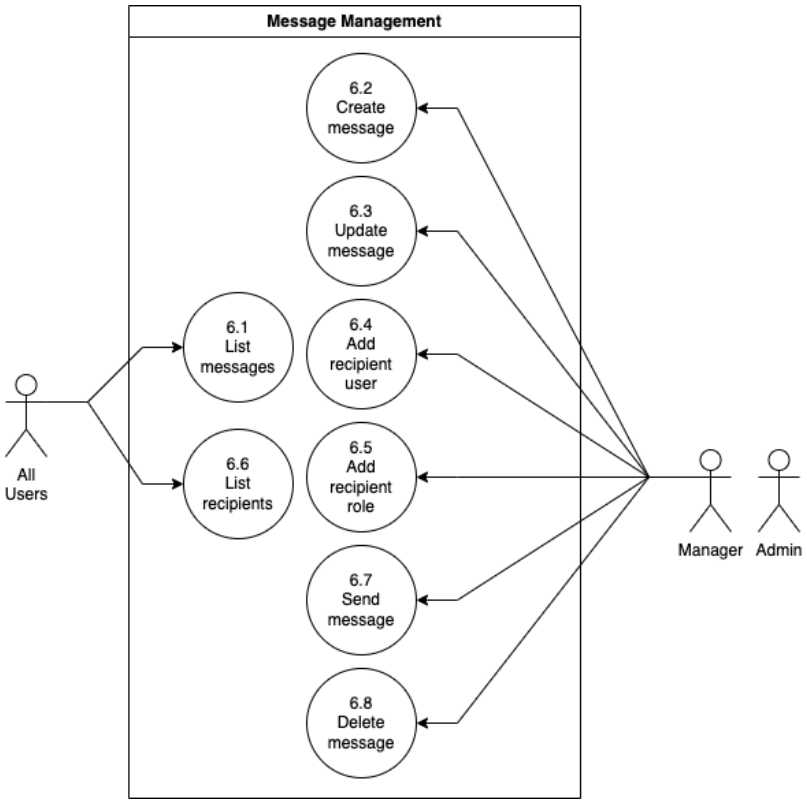

Fig. 7: Message Management Use case diagram.

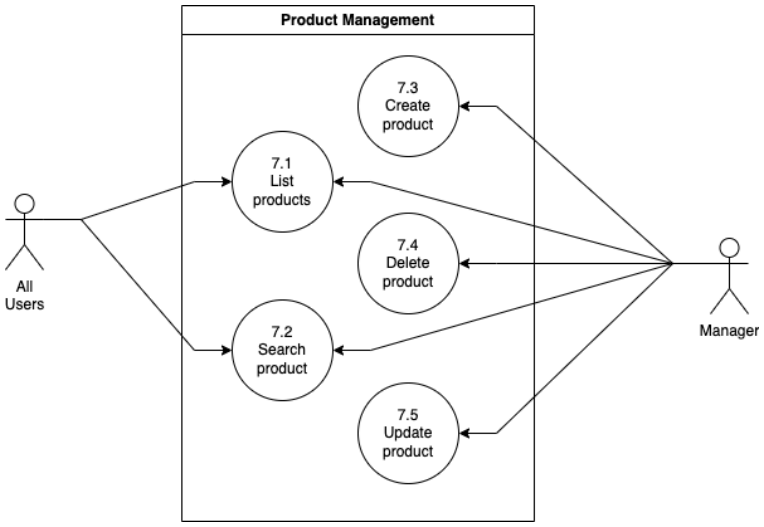

Fig. 8: Product Management Use case diagram.

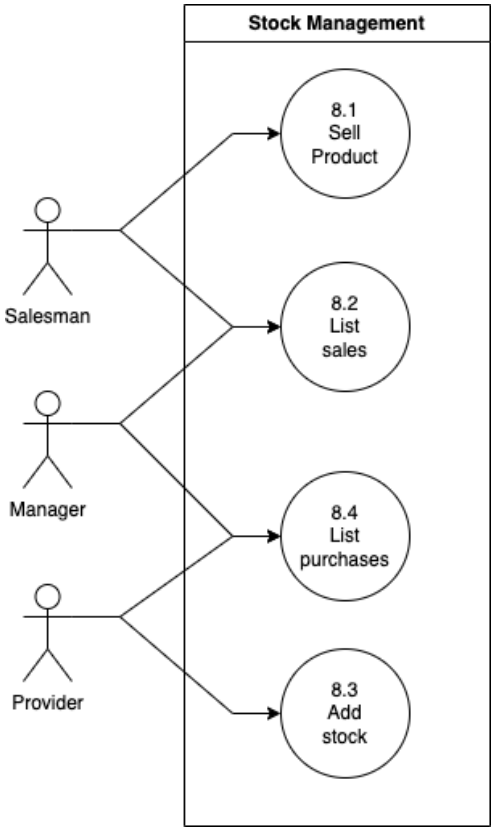

Fig. 9: Stock Management Use case diagram.

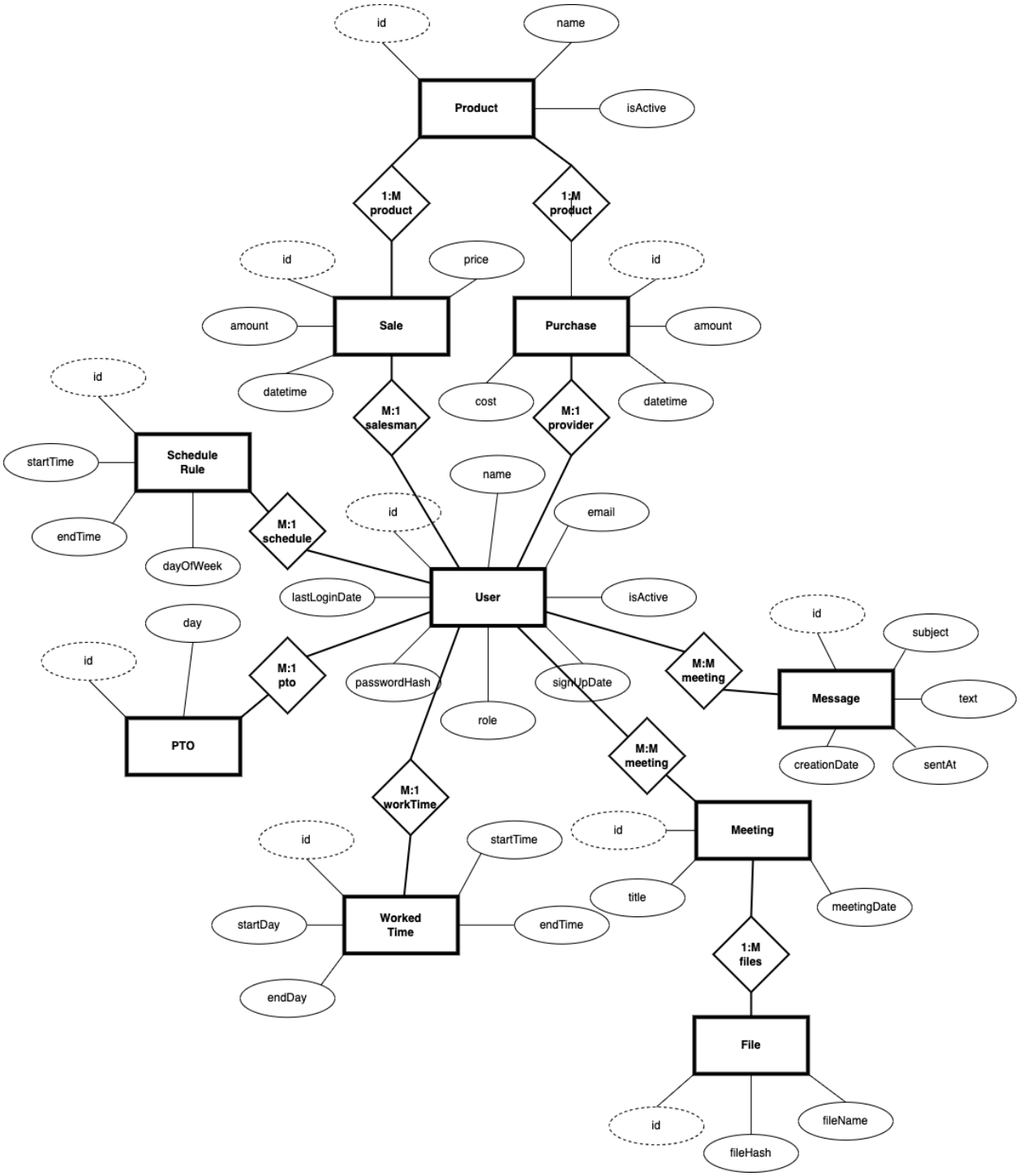

Fig. 10: Entity-Relationship Diagram.

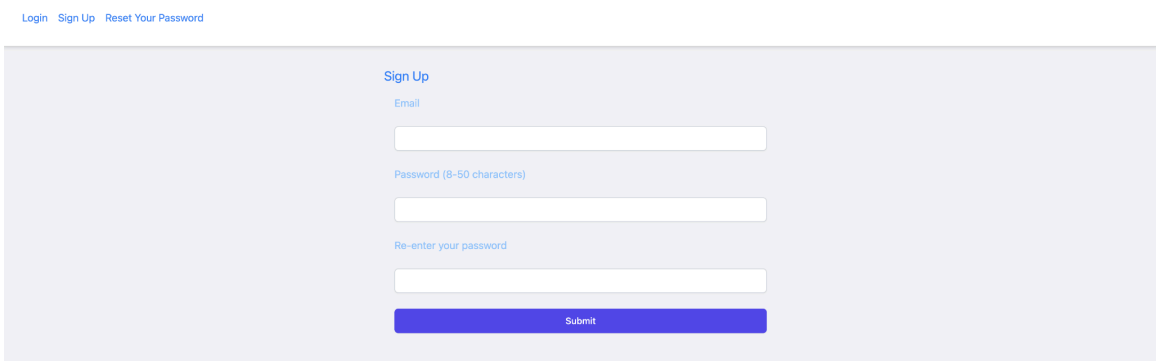

Fig. 11: Sign Up Page

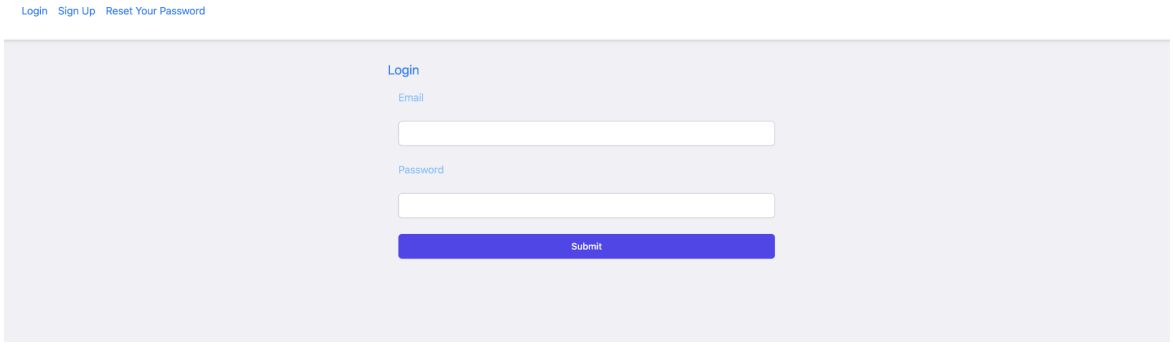

Fig. 12: Login Page

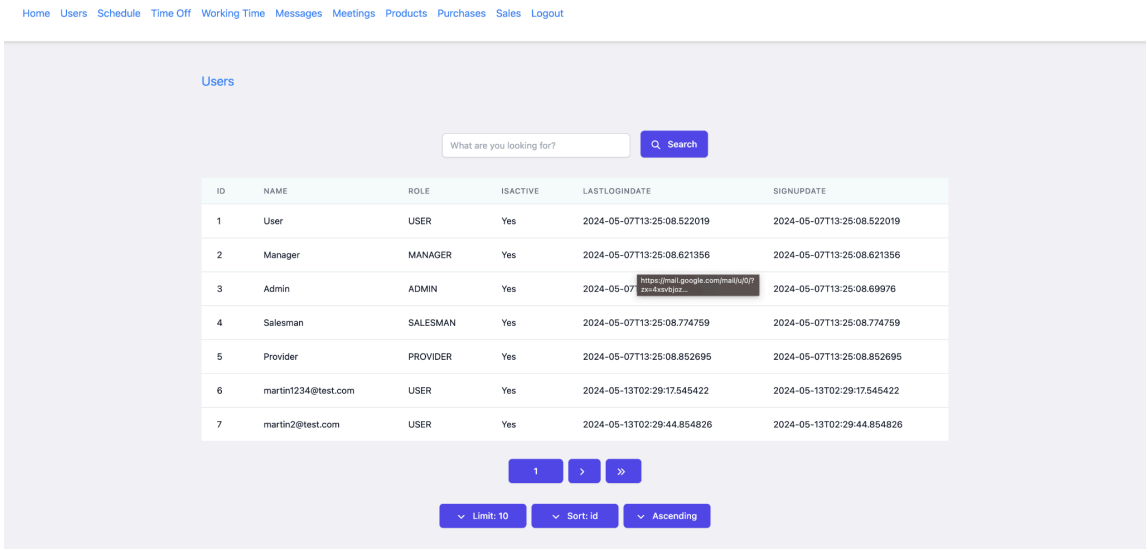

Fig. 13: Users Management Page.