|  |  |  |
| --- | --- | --- |
|  |  |  |

Appointment scheduling Platform 2025

**Graduation Project Documentation**



**Appointment scheduling Platform**

**A Project Submitted in partial fulfilment of the requirements for the Degree of Bachelor of Science in Systems and Computers Engineering**

**Submitted By**

**Eslam Samy Abdul-Qader 404019**

**Ahmad Sobhy Nassar 404011**

**Sayed Ahmad Mahmoud 404023**

**Eslam Mohammed Saleh 404021**

**Mohammed Mustafa Ramadan 404082**

**Supervised by**

**Dr. Abdulrahman Halawa**

**2025**

|  |  |  |
| --- | --- | --- |
|  |  |  |

Appointment scheduling Platform 2025

**Graduation Project Documentation**



**Appointment scheduling Platform**

**A Project Submitted in partial fulfilment of the requirements for the Degree of Bachelor of Science in Systems and Computers Engineering**

**Submitted By**

**Eslam Samy Abdul-Qader 404019**

**Ahmad Sobhy Nassar 404011**

**Sayed Ahmad Mahmoud 404023**

**Eslam Mohammed Saleh 404021**

**Mohammed Mustafa Ramadan 404082**

**Supervised by**

**Dr. Abdulrahman Halawa**

**2025**

|  |  |  |
| --- | --- | --- |
| **Examiner Committee** | | |
| **Name** | **Rule** | **Signature** |
| Prof. Ashraf Maddcore | President |  |
| Dr. Abdulrahman Halawa | Supervisor |  |
| Dr. Mohammed Rayan | Member |  |

ABSTRACT

Time management and scheduling are critical challenges for professionals and service providers who rely on client appointments to deliver their services efficiently. This project, Timease, presents a full-stack scheduling and booking platform designed to streamline event management and appointment scheduling. The platform enables professionals—such as doctors, lawyers, and consultants—to create, update, and manage event listings, while allowing users to seamlessly book or cancel appointments. To ensure secure and scalable interaction, the backend is built using Java Spring Boot with RESTful API design and implements robust authentication via JWT tokens and refresh mechanisms. Role-based access control distinguishes between regular users and administrators, ensuring proper authorization for sensitive operations.

The frontend is developed as a mobile application using Flutter, ensuring a cross-platform and user-friendly experience. PostgreSQL serves as the primary relational database, integrated with the backend using Spring Data JPA and Hibernate for efficient data persistence. The entire system is containerized with Docker and deployed via Railway, enabling a streamlined DevOps pipeline and facilitating easy deployment and scalability. API endpoints are documented and exposed through Swagger UI for testing and development purposes.

This project demonstrates the practical integration of modern web technologies, cloud deployment practices, and secure API design to address real-world scheduling challenges. The results validate the system’s usability, functionality, and maintainability. Future extensions include the implementation of a notification system and enhanced input validation, which aim to improve user interaction and reliability. Overall, Timease contributes an effective and extensible solution for time management in professional settings.

**KEYWORDS** Scheduling; Spring Boot; Flutter; JWT; Docker;

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to **Dr. Abdulrahman Halawa**, our project supervisor, for his unwavering support, expert guidance, and thoughtful feedback throughout every stage of this graduation project. His mentorship played a critical role in helping us navigate technical challenges and refine our vision for Timease. His dedication and encouragement greatly contributed to the successful completion of our work.

We are also sincerely thankful to **Prof. Ashraf Maddcore**, President of the Graduation Committee, for his valuable insights and for fostering a rigorous academic environment. Our appreciation extends to **Dr. Mohammed Rayan**, Committee Member, whose observations and suggestions enriched the development and final presentation of our project.

  We gratefully acknowledge the technical and academic resources provided by our institution, which enabled us to develop and test our platform effectively. While we did not receive external financial assistance, the availability of institutional support and infrastructure was vital to our progress.

  We would also like to recognize the informal contributions of colleagues and peers who assisted us during development—particularly in testing, debugging, and providing feedback. Lastly, we extend heartfelt thanks to our families and friends for their continuous encouragement and support throughout this journey.

TABLE OF CONTENTS

ABSTRACT…………………………………………….………. Iv

ACKNOWLEDGEMENTS …………………………….……….v

**TABLE OF CONTENTS ……………………………….……….vi**

**CHAPTER 1 INTRODUCTION ………………….………… 1**

1.1 [Background](#_bookmark2)

1.2 [Motivation](#_bookmark3)

1.3 [Problem description](#_bookmark4)

1.4 [Aims and Objectives](#_TOC_250024)

1.5 [Methodology](#_bookmark5)

**CHAPTER 2** [**LITERATURE REVIEW.**](#_TOC_250023) **………….………… 1**

2.1 Introduction

2.2 Previous work

2.3 Summary

**CHAPTER 3 SYSTEM DESIGN.………….……….……… 1**

3.1 Introduction

3.2 System Architecture

3.3 Requirements

3.3.1 User Requirements

3.3.2 System Requirements

3.4 Sequence Diagrams

3.5 Use case Diagrams

3.6 Database Design

**CHAPTER 4 IMPLEMENTATION …….………….………… 1**

4.1 Introduction

4.2 Tools & Technologies

4.3 UI Design

**CHAPTER 5 CONCLUSION AND FUTURE WORK.…...…… 1**

**CHAPTER 1**

**INTRODUCTION**

The aim of the work described in this document is to develop a full-stack software system that simplifies and streamlines the scheduling process between service providers and their clients. The result of this effort is Timease - a name that reflects the platform’s core purpose: making time management easy. It is a combination of the words time and ease, representing our goal of easing the process of booking, managing, and attending appointments through a unified, user-friendly solution.

  Professionals across various industries—such as healthcare, legal consultation, and freelance services—depend on precise and reliable scheduling systems to manage their availability and client interactions. However, existing solutions often fall short in terms of flexibility, role-specific control, and ease of integration, especially for professionals seeking simple, mobile-first solutions. Timease addresses these limitations by offering a robust appointment scheduling platform that supports both one-on-one sessions and group events, built around a mobile-friendly architecture with role-based access, secure authentication, and effortless deployment.

  Our solution is designed as a monolithic Spring Boot backend connected to a Flutter-based mobile frontend, supported by a PostgreSQL relational database. The application is containerized with Docker and deployed via Railway, providing a production-ready architecture that is easy to replicate and scale. The system also supports secure user authentication through JWT and refresh tokens, along with administrative tools for event creation and moderation.

  This document outlines the rationale behind the development of Timease, discusses its implementation, and evaluates its potential to serve as a lightweight yet powerful scheduling platform. The sections that follow provide background context, articulate our motivations and objectives, define the specific problem we addressed, and detail the methodology followed throughout the project life-cycle.

**1.1** [**Background**](#_bookmark2)

Effective time management is a critical component of productivity, particularly for professionals whose work depends on client appointments, meetings, or service sessions. In fields such as medicine, law, consulting, education, and freelance services, maintaining an organized schedule is essential not only for operational efficiency but also for client satisfaction. With the increasing reliance on digital solutions, scheduling platforms have become integral tools for managing such time-dependent interactions.

While several commercial tools exist—such as Calendly, Doodle, and Google Calendar—they are often constrained by rigid configurations, limited role-specific customization, or require integration with external ecosystems that may not align with every user's needs. Additionally, many platforms impose paywalls for essential features, or lack support for easy deployment by independent professionals or small organizations without dedicated IT infrastructure.

  Recognizing these challenges, we set out to design and implement *Timease*, a comprehensive and accessible scheduling system that can be easily adopted by professionals without technical overhead. By combining a mobile-first user interface with a flexible and secure backend, *Timease* supports a range of appointment types—including both individual and group events—while ensuring that administrative control, user management, and session security are handled efficiently.

  Furthermore, we observed that many existing systems are fragmented across multiple services or rely heavily on cloud-based automation that is not always transparent or controllable by end users. With *Timease*, we aimed to deliver a system where the entire stack is clear, contained, and customizable—giving full ownership to the service provider or organization that deploys it.

  The development of *Timease* was also motivated by a desire to produce a system that could serve as a complete learning experience in full-stack development, deployment, and secure application design, reflecting real-world software engineering practices.

### 1.2 Motivation

Our motivation for developing Timease stemmed from a combination of technical curiosity, real-world demand, and a desire to create a practical and scalable solution to a common problem. As students of software engineering preparing for professional careers, we sought to build a system that not only demonstrated our full-stack development capabilities, but also delivered meaningful value to users beyond academic boundaries.

Throughout our academic and personal experiences, we noticed that professionals often rely on ad hoc or overly complex solutions to manage their appointments. Many are forced to use generalized tools that lack critical features such as role differentiation, mobile accessibility, or secure user authentication. This observation highlighted a gap in the availability of simple, customizable platforms that cater to small-scale service providers—individuals or teams who require robust functionality without the overhead of enterprise systems.

We were particularly motivated to build a mobile-first system because of the increasing dependence on smartphones for daily management tasks. A mobile-friendly design ensures that users—both professionals and clients—can interact with the platform anytime and anywhere, increasing the practicality and relevance of the system. In addition, incorporating role-based access control, session management via JWT, and Dockerized deployment allowed us to explore real-world technologies used in modern software systems.

By creating Timease, we aimed to produce a solution that is both educational and applicable, demonstrating how thoughtful design and strategic technology choices can solve common scheduling challenges. The project also allowed us to apply software engineering principles in a structured and meaningful way, with the potential for further development, real-world use, and academic presentation.

### 1.3 Problem Description

Scheduling and time coordination are persistent challenges for professionals and service providers who rely on client appointments to deliver their services effectively. In many cases, these professionals lack access to tailored tools that allow them to manage bookings, prevent overlaps, and adapt to both structured and dynamic availability. While commercial scheduling platforms exist, they often fail to meet the specific needs of individual practitioners or small teams due to limited conformability, overcomplicated interfaces, or high subscription costs for essential features.

The core problem lies in the absence of a lightweight, flexible, and mobile-oriented scheduling system that offers **secure role-based access**, **event customization**, and **client-friendly booking interfaces**, all in one integrated platform. Professionals may require different levels of control—such as the ability to create, modify, and monitor appointments—while clients should be able to book or cancel meetings easily without compromising security or performance. Existing solutions rarely provide such role-aware flexibility while also supporting both **individual and group meeting structures**.

Moreover, deployment complexity and platform dependence can act as barriers to adoption. Many tools assume continuous internet connectivity, cloud platform integration, or third-party plugin reliance, which may not be suitable for all contexts. For developers or teams seeking to own and operate their own scheduling system, these dependencies introduce unnecessary friction.

Timease addresses these issues by providing a full-stack, self-contained scheduling and booking platform that is mobile-first, easy to deploy, and secure by design. By simplifying both the user experience and the underlying technical infrastructure, we aim to close the gap between professional needs and accessible technology. The problem we set out to solve, therefore, is the **lack of a complete, customizable, and user-friendly solution for managing time-dependent professional services**.

### 1.4 Aims and Objectives

The primary aim of this project is to design and develop a fully functional, secure, and user-friendly scheduling platform—Timease—that enables professionals to create and manage appointments while allowing clients to book, cancel, or view events with ease. The system is intended to simplify time management processes by offering a mobile-first experience backed by a reliable backend infrastructure that can be easily deployed and maintained.

To achieve this aim, we established a set of specific objectives that guided the development of Timease throughout its lifecycle:

• **Develop a responsive mobile application** using Flutter that enables end users to browse available events and book appointments efficiently.

• **Design and implement a RESTful backend API** using Spring Boot that supports secure user authentication, event and user management, and booking functionalities.

• **Integrate role-based access control** to distinguish between general users and administrators, ensuring appropriate permissions for each action within the system.

• **Implement secure authentication using JWT** and refresh tokens to maintain session integrity and prevent unauthorized access.

• **Support both one-on-one and group event scheduling**, giving administrators the flexibility to define different event types and capacities.

• **Enable users to view, book, and cancel appointments**, with backend logic to ensure data consistency and event availability.

• **Containerize the backend application using Docker** to ensure reproducibility, ease of deployment, and environment consistency.

• **Deploy the platform on Railway**, leveraging a cloud-based environment for hosting the backend and database in production settings.

• **Follow modern software engineering practices**, including version control with GitHub, modular code design, and clear documentation.

Together, these objectives ensure that Timease is not only a technically robust application, but also a practical tool for real-world use, aligning with our academic goals and professional aspirations.

**1.5** [**Methodology**](#_bookmark5)

The development of Timease followed a structured, iterative methodology inspired by agile software development principles. Our goal was to ensure continuous progress through frequent testing, clear task segmentation, and close coordination across all stages of the project—from requirements gathering to deployment. The methodology emphasized adaptability, allowing us to incorporate feedback and adjust features based on evolving priorities and technical constraints.

We began by defining the system requirements and drafting a set of user stories to identify the core functionalities needed for both administrators and end users. These included booking workflows, authentication mechanisms, event creation, and role-based access control. The system architecture was then planned with a clear separation of concerns between the frontend and backend, both of which were developed in parallel to ensure seamless integration.

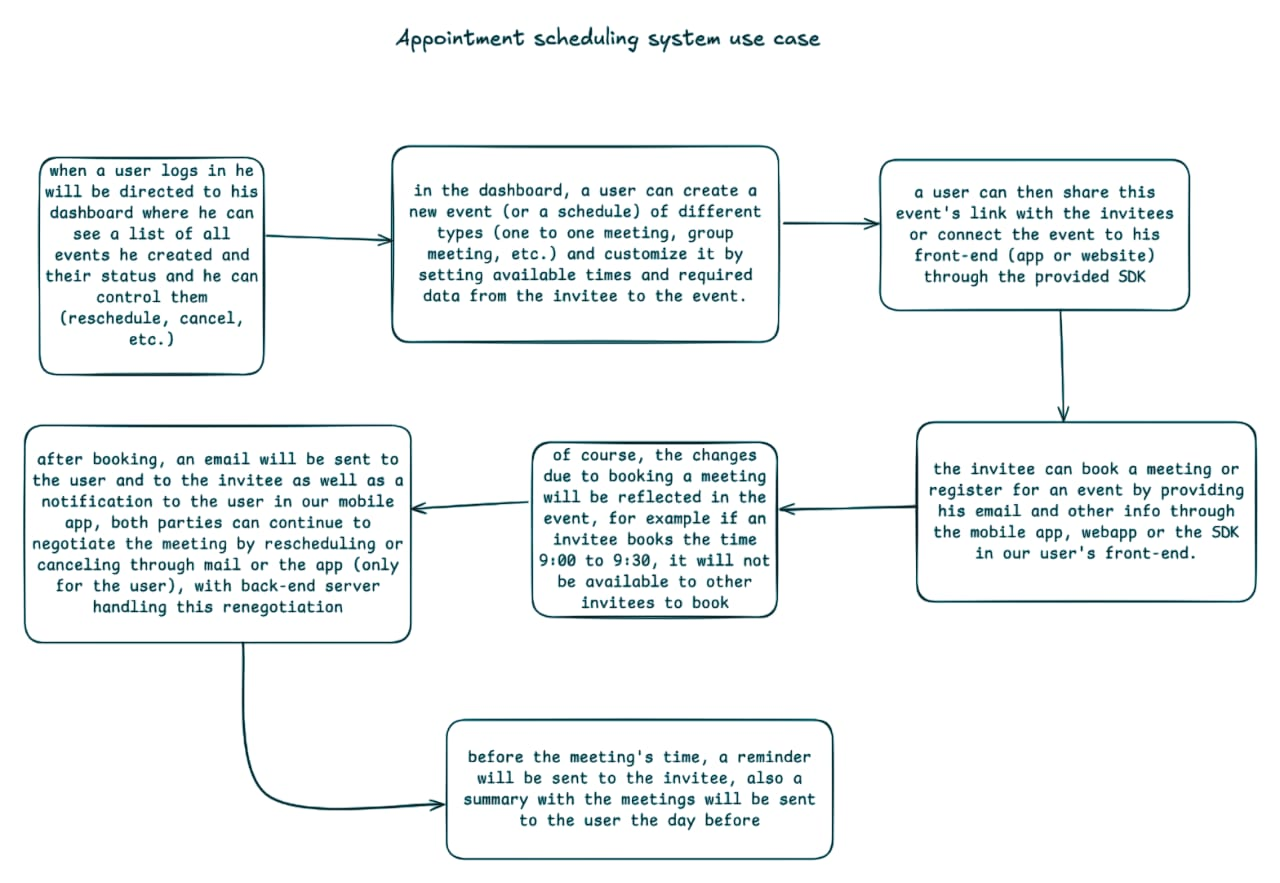
Figures 1-1 and 1-2 show early design dreafts and user stories from the design phase

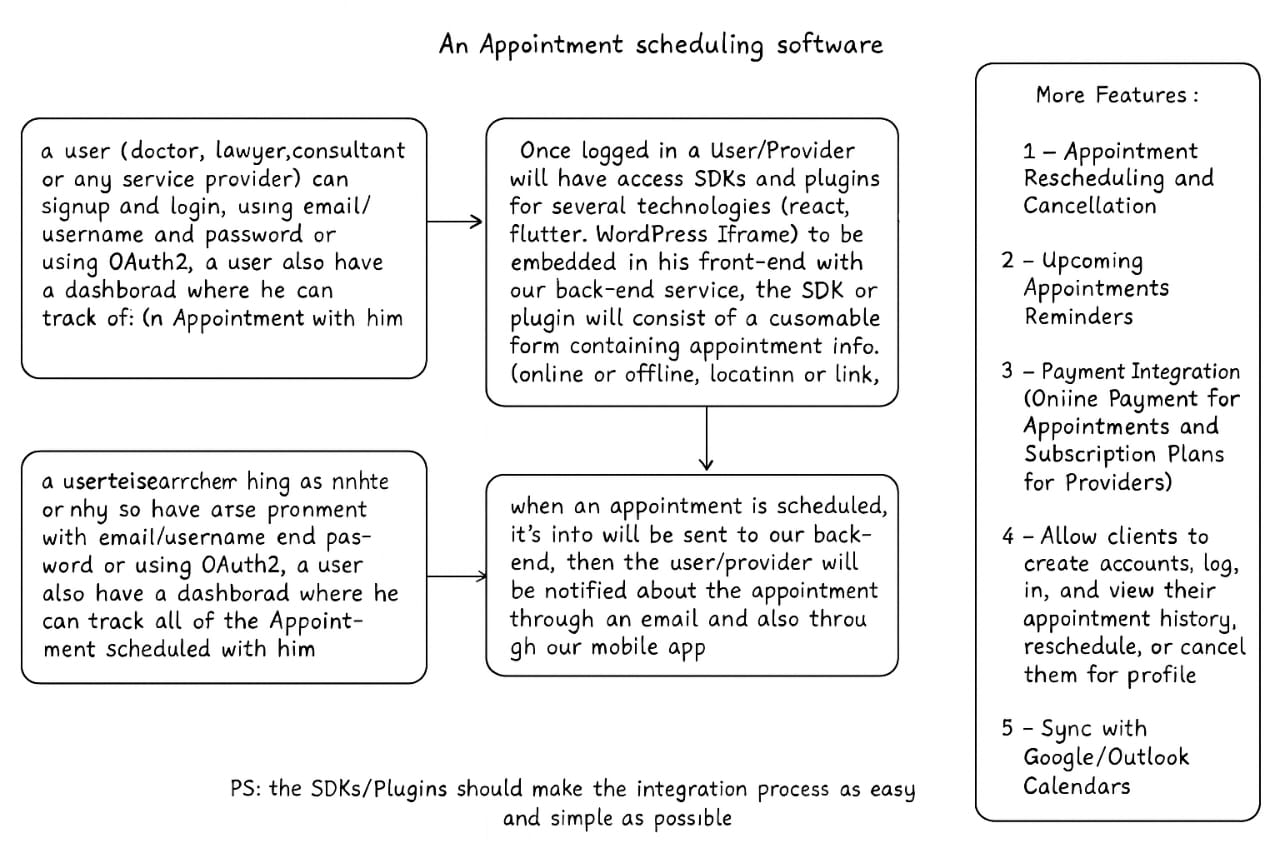
On the backend, we used Spring Boot to implement a RESTful API that handles all business logic and communicates with a PostgreSQL database using Spring Data JPA and Hibernate. Secure authentication was implemented using JWT tokens and refresh tokens, while access rights were enforced through role-based authorization. The backend was fully containerized using Docker, and a Docker Compose file was created to manage service dependencies for local development and production.

The frontend was built using Flutter, chosen for its cross-platform capabilities and responsive design features. The mobile application connects to the backend API and provides interfaces for booking, cancellation, and event browsing. We conducted regular testing on both the UI and backend services to ensure functional correctness, security, and performance.

Finally, we deployed the backend and database on Railway, a cloud platform that enabled fast and reliable deployment. Version control and collaboration were maintained through GitHub, allowing for systematic tracking of changes and contribution management.

This methodology ensured that Timease was developed with clear milestones, technical rigor, and a focus on usability—resulting in a robust and deployable scheduling platform.

Figure 1-1

Figure 1-2

**CHAPTER 2**

[**LITERATURE REVIEW**](#_TOC_250023)

**2.1 Introduction**

  Scheduling systems have become an essential component in modern digital infrastructure, enabling professionals and clients to coordinate meetings, appointments, and events with greater efficiency. These tools serve a broad spectrum of users, from individuals managing personal calendars to businesses automating complex booking workflows. In recent years, the demand for intuitive, accessible, and mobile-friendly scheduling platforms has grown significantly, driven by the shift toward remote work, freelance services, and time-sensitive professional engagements.

  The primary role of scheduling applications is to streamline time coordination, reduce administrative overhead, and eliminate conflicts through real-time availability management. They often include features such as calendar synchronization, automated reminders, client booking links, and integrations with third-party platforms like email or video conferencing tools. Many of these systems are delivered through cloud-based platforms that offer subscription tiers based on functionality, scalability, and automation capabilities.

  In this chapter, we provide an overview of several widely used scheduling tools that have shaped user expectations and industry standards. While the design and capabilities of these platforms vary, they all aim to simplify appointment management and improve user experience. The following sections introduce popular examples of such tools, providing context for understanding the space in which Timease operates and the general expectations users may have when engaging with a scheduling solution.

## 2.2 Previous work

  A number of established scheduling platforms are widely used across various professional fields, each offering its own approach to appointment management. These tools have shaped the landscape of digital scheduling and serve as important reference points for understanding user needs and technical features. Below are brief overviews of several notable systems:

### Calendly

  Calendly is a widely adopted cloud-based scheduling platform that allows users to create booking links that reflect their availability. Clients can use these links to schedule meetings without back-and-forth messaging. Calendly integrates with popular calendar services such as Google Calendar and Outlook, and it supports features like automated reminders, time zone detection, and meeting buffers. The platform is especially popular among consultants, sales teams, and educators due to its user-friendly interface and automation capabilities.

### Doodle

  Doodle focuses on group scheduling and polling, allowing users to propose multiple time options for a meeting and gather input from participants to determine the most suitable time. It is often used in academic, administrative, and team collaboration contexts where coordination among multiple people is required. While less focused on individual appointment booking, Doodle remains effective for collective time management.

### SimplyBook.me

  SimplyBook.me is a comprehensive appointment scheduling solution tailored to service-based businesses. It provides customizable booking pages, support for payment processing, and a range of client communication tools. The platform also offers integrations with CMS systems and social media platforms, making it suitable for small businesses that require both functionality and brand presentation.

### Setmore

  Setmore offers a free and premium booking platform designed for small businesses, healthcare providers, and beauty services. It supports appointment reminders, staff logins, and integrations with tools like Zoom and Slack. Its simple interface and mobile app make it accessible for teams that need to manage multiple calendars and client interactions.

  These systems demonstrate the diversity of features and design strategies in the digital scheduling domain. Each platform has contributed to the broader understanding of usability, automation, and deployment in appointment management, and they offer valuable insight into the key functionalities users expect from such solutions.

## 2.3 Summary

  The landscape of scheduling applications is populated with diverse solutions that address a variety of professional and organizational needs. Tools such as Calendly, Doodle, SimplyBook.me, and Setmore have introduced users to streamlined appointment workflows, calendar integration, and digital accessibility. These platforms have successfully demonstrated the value of automated scheduling in reducing time conflicts, improving service efficiency, and enhancing user experience.

  Despite their wide adoption, existing systems vary significantly in focus and complexity. Some prioritize individual bookings, while others target group coordination or enterprise-level needs. Additionally, certain features—such as mobile-first design, customizable user roles, or self-hosted deployment—are either missing or locked behind premium tiers. These variations reveal opportunities for alternative solutions that cater to specific user groups with greater flexibility and autonomy.

  By examining the strengths and limitations of these platforms, we gained insight into key design principles and user expectations. This understanding informed the development of Timease, which seeks to offer a simplified, mobile-accessible, and developer-friendly scheduling experience. The platform aims to fill the usability and deployment gaps identified in the literature, while maintaining core features such as secure authentication, role-based access, and responsive user interfaces.

  In the following chapters, we present the technical design and implementation of Timease, demonstrating how the platform builds upon and adapts these foundational ideas to meet real-world scheduling needs.

**CHAPTER 3**

**SYSTEM DESIGN**

## 3.1 Introduction

  This chapter presents the system design of Timease, a full-stack scheduling and appointment management platform developed to address the need for a flexible, scalable, and mobile-accessible booking solution. The design of Timease emphasizes clean separation of concerns, security in authentication, and seamless integration between components through a service-oriented approach. The system architecture reflects the use of modern development tools and practices, aligning with the best standards of contemporary software engineering.

  The design process followed a modular structure, wherein the system was decomposed into core functional layers: the frontend interface, the backend service layer, and the persistence layer. The mobile frontend, developed using Flutter, ensures cross-platform accessibility. The backend, built with Spring Boot, offers a RESTful API that facilitates communication with the mobile client and interacts with a PostgreSQL relational database for data persistence.

  Security is enforced through robust role-based access control (RBAC), implemented using Spring Security with JWT-based authentication and refresh token support. This ensures secure, stateless session management and protects user data across the platform. The backend is containerized using Docker and deployed on Railway, enabling rapid iteration, ease of deployment, and scalability under load. The PostgreSQL database is hosted on Neon and structured around normalized relational models to support efficient querying and data integrity.

  The rest of this chapter elaborates on the high-level architecture, requirement specifications, and functional flow of the application. It also includes visual representations through sequence diagrams, use case diagrams, and database design models, providing a comprehensive view of the system's technical blueprint.

## 3.2 System Architecture

  Timease is designed using a layered, service-oriented architecture that ensures modularity, maintainability, and scalability. The system is composed of three primary components: the frontend client (Flutter mobile app), the backend server (Spring Boot RESTful API), and the relational database (PostgreSQL). These components communicate over secure HTTP connections and are orchestrated using Docker containers to ensure consistent deployments across environments.

  At a high level, the system architecture is organized into the following layers:

### 3.2.1. ****Presentation Layer****

  This layer is implemented using Flutter, enabling the creation of a responsive, cross-platform mobile application. The frontend handles user interaction, input validation, and rendering of dynamic content. It communicates with the backend using HTTP requests and consumes JSON-formatted RESTful APIs.

### 3.2.2. ****Application and Service Layer****

  The core of the backend system is developed using Spring Boot. It exposes REST endpoints that encapsulate the business logic of the platform. This layer includes modules for:

* **Authentication & Authorization**: Based on Spring Security, it uses JWT tokens for stateless session management and refresh tokens for prolonged user sessions.
* **Event Scheduling**: Handles event creation, availability management, and booking workflows.
* **Notifications**: Manages sending event-based notifications to users based on triggers.

  All services are protected by role-based access control (RBAC) mechanisms, supporting multiple user roles such as ROLE\_USER, ROLE\_ADMIN, and ROLE\_MODERATOR.

### 3.2.3. ****Persistence Layer****

  The system relies on PostgreSQL for data storage, with schemas designed for normalization, data integrity, and referential consistency. JPA and Hibernate are used to interact with the database, enabling object-relational mapping and simplifying data access logic.

### 3.2.4. ****Infrastructure Layer****

  Timease is containerized using Docker, allowing isolated environments for the backend and database services. This setup supports portability and environment consistency. The application is deployed using Railway for managed hosting of the backend and Neon for hosting the PostgreSQL database. These tools facilitate continuous deployment, scaling, and monitoring of system health in production.

  This multi-layered architecture not only provides a clear separation of responsibilities but also enhances the platform's ability to evolve, scale, and integrate new features with minimal disruption.

## 3.3 Requirements

  The requirements of Timease were gathered and refined based on an analysis of typical use cases for scheduling systems, current limitations in existing solutions, and the project’s aim to deliver a robust, user-friendly, and scalable application. These requirements are categorized into **User Requirements**, which describe the system from the perspective of the end users, and **System Requirements**, which outline the technical specifications needed to support the platform's functionality.

### 3.3.1 User Requirements

  The following user-oriented requirements define the functionalities expected by different user roles in Timease, including professionals (service providers), end users (clients), and administrators:

* **Account Management**:
  + Users must be able to register, log in, and log out securely.
  + Passwords must be encrypted and account activation should be required.
  + Forgotten passwords should be recoverable through a secure process.
* **Role-Based Access**:
  + The system should distinguish between at least three roles: regular users, administrators, and moderators.
  + Administrators should have access to event management features and user oversight.
  + Users should only access functionality authorized to their role.
* **Event Creation and Management**:
  + Service providers must be able to create, edit, and delete events.
  + Providers should specify event details including title, location, duration, scheduling range, and maximum attendees.
* **Availability Scheduling**:
  + Providers must be able to define time slots for their events, either on a specific date or recurring weekly.
  + Users must be able to view available slots before booking.
* **Booking and Participation**:
  + End users must be able to book meetings within available time slots.
  + Users should be allowed to cancel their bookings within valid timeframes.
  + The system should prevent overbooking beyond defined attendee limits.
* **Notifications**:
  + Users must receive in-app notifications for booking confirmations, cancellations, and event changes.
* **Mobile Accessibility**:
  + All user interactions must be available via a mobile-first application interface, designed for performance and responsiveness.

### 3.3.2 System Requirements

  The system requirements define the technical specifications and operational conditions under which Timease must function. These are categorized into **functional** and **non-functional** requirements to ensure both feature completeness and quality attributes like performance, reliability, and maintainability.

#### **Functional Requirements**

* **Authentication and Authorization**:
  + Support for secure login and logout using JWT-based authentication.
  + Use of refresh tokens to maintain session continuity without requiring repeated logins.
  + Enforcement of role-based access control (RBAC) using predefined roles: ROLE\_USER, ROLE\_ADMIN, and ROLE\_MODERATOR.
* **Event and Scheduling Management**:
  + Endpoints for creating, updating, retrieving, and deleting events.
  + Support for adding availabilities based on specific dates or recurring days of the week.
  + Functionality to define constraints like maximum attendees and scheduling windows.
* **Meeting Booking Workflow**:
  + REST API endpoints to handle booking creation and cancellation.
  + Automatic conflict prevention for overlapping bookings or overcapacity.
* **Notification Delivery**:
  + Asynchronous notification system for informing users of key actions (e.g., booking success, cancellation, event updates).
* **API Design**:
  + RESTful API endpoints with consistent response formatting.
  + Metadata support in list endpoints (e.g., total count, pagination support).

#### **3.3.3 Non-Functional Requirements**

* **Scalability**:
  + The system must scale to support multiple concurrent users and high-frequency event interactions.
  + Docker-based containerization must support horizontal scalability during deployment.
* **Security**:
  + All communication must be secured using HTTPS.
  + User data must be stored securely with hashed passwords and restricted access control.
  + JWT and refresh tokens must be securely signed and validated.
* **Performance**:
  + The system should respond to requests within acceptable time thresholds (<500ms for most operations).
  + Backend services must support concurrent requests efficiently.
* **Reliability and Availability**:
  + Deployed backend services must have high uptime and resilience to failure, supported by Railway’s managed infrastructure.
  + The PostgreSQL database on Neon must ensure ACID compliance and automatic failover.
* **Maintainability**:
  + Source code should be modular and well-documented.
  + APIs must be versioned and easily extendable to accommodate future features.
* **Portability**:
  + Docker containers should ensure consistent behavior across different environments (development, testing, production).
  + System setup must support local and cloud-based deployment with minimal configuration.

**3.4 Sequence Diagrams**

  Sequence diagrams describe how objects interact in a particular scenario of a use case. In Timease, sequence diagrams were used to visualize the step-by-step flow of communication between the system components for core functionalities. These diagrams help clarify the logic, ordering, and dependencies of operations in our system.

  The following are key interactions represented using sequence diagrams:

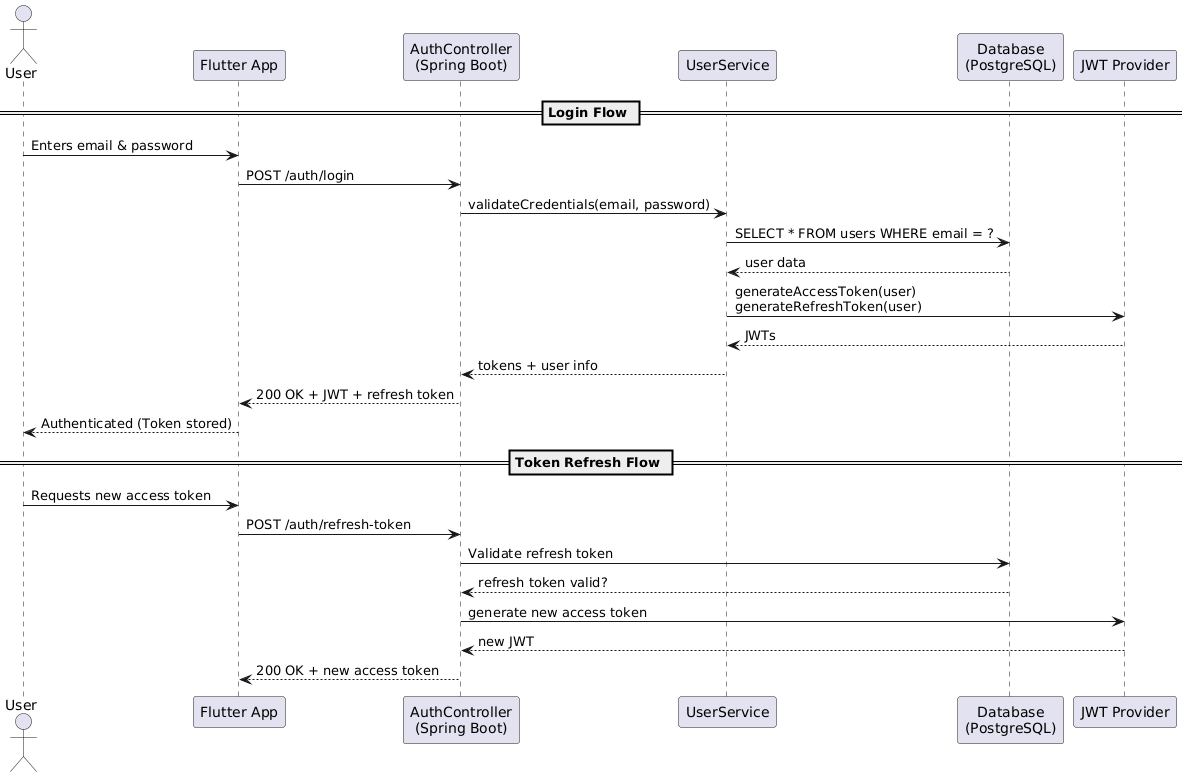
**3.4.1 User Authentication Sequence**

**Scenario:** A user logs into the system using email and password to obtain access tokens.

**Actors:** User, Authentication Controller, Authentication Service, JWT Utility, Database

**Flow:**

1. User submits login credentials via the mobile app.
2. The Authentication Controller receives the request and passes it to the Authentication Service.
3. The service validates credentials against hashed passwords in the database.
4. If credentials are valid, a JWT access token and a refresh token are generated.
5. The refresh token is stored in the database and returned to the user with the access token.

Figure 3.4.1 - User Authentication Sequence

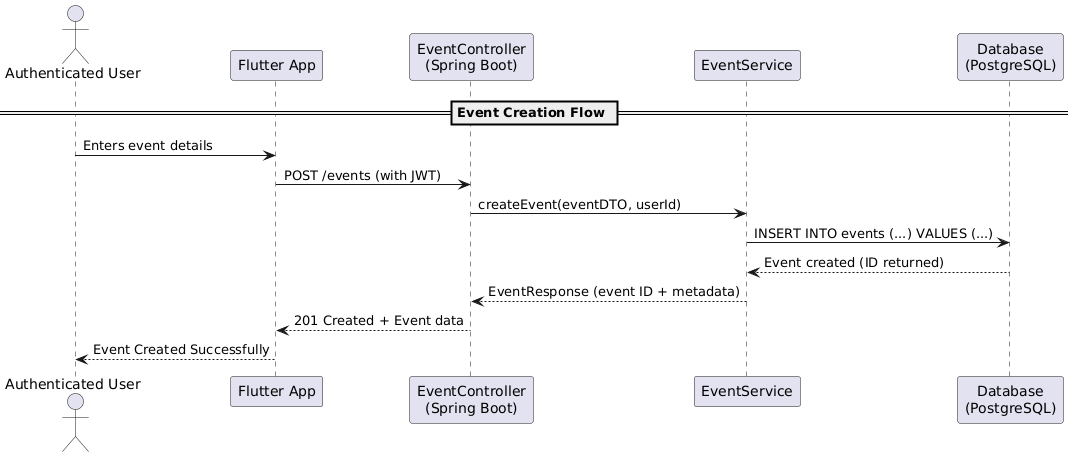
**3.4.2 Event Creation Sequence**

**Scenario:** A professional user creates a new event.

**Actors:** User (Provider), Event Controller, Event Service, Database

**Flow:**

1. User submits event data through the frontend (e.g., title, duration, max attendees).
2. The Event Controller receives the request and calls the Event Service.
3. The Event Service verifies the user’s role and creates the event record in the database.
4. A success response with event details is returned to the user.

Figure 3.4.2 - Event Creation Sequence

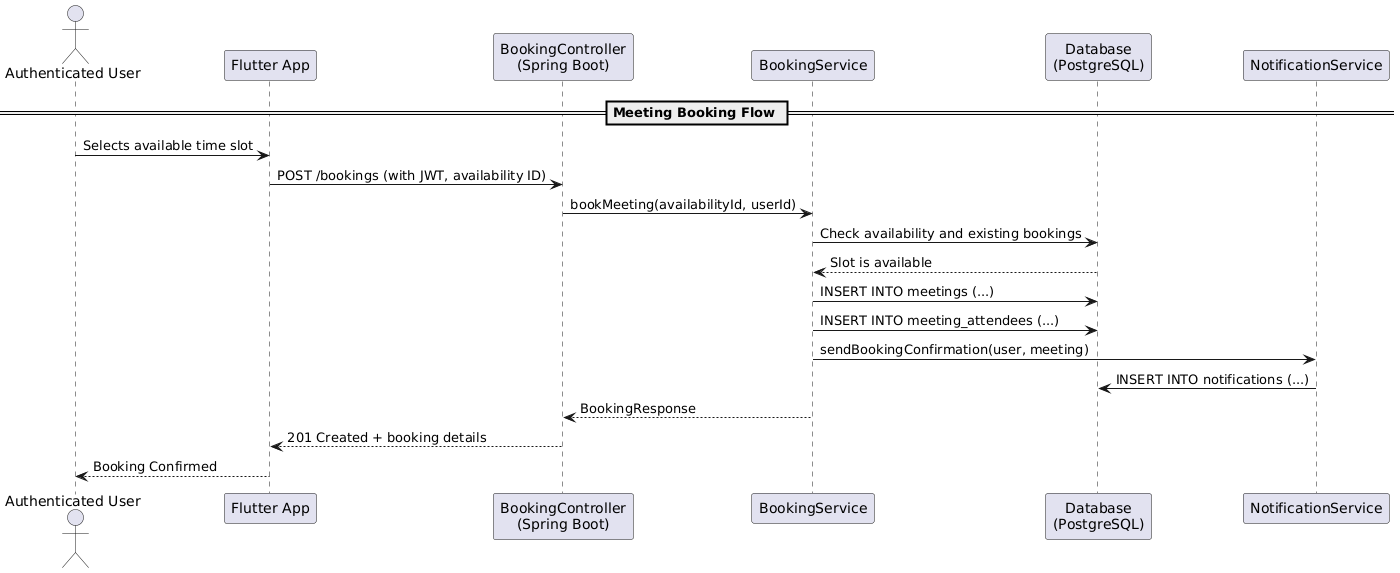
**3.4.3 Meeting Booking Sequence**

**Scenario:** A user books a meeting based on an available time slot.

**Actors:** User, Meeting Controller, Meeting Service, Availability Service, Database

**Flow:**

1. User selects a time slot and submits a booking request.
2. The Meeting Controller verifies the user’s identity and calls the Meeting Service.
3. The service checks if the availability is valid and not fully booked.
4. The meeting is created, and the user is registered as an attendee.
5. A confirmation notification is generated and sent to the user.

Figure 3.4.3 - Meeting Booking Sequence

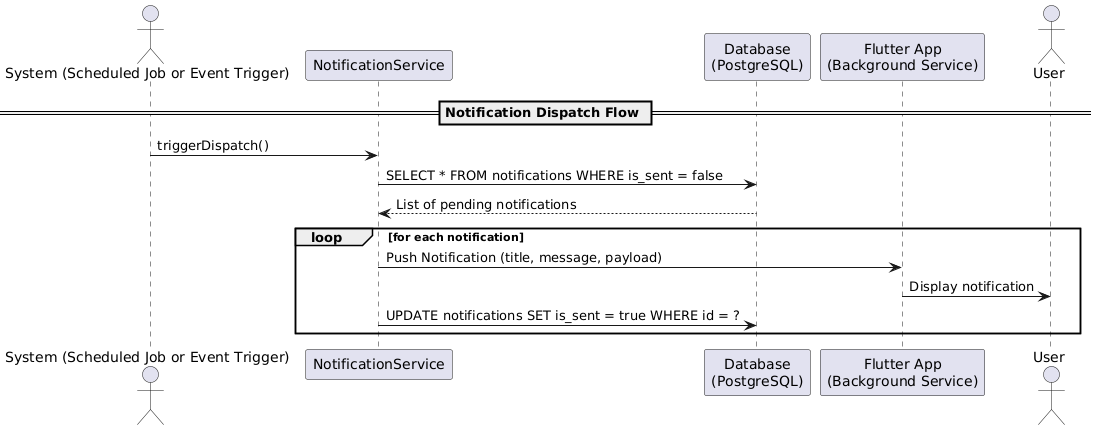
**3.4.4 Notification Dispatch Sequence**

**Scenario:** A notification is sent to a user after booking a meeting.

**Actors:** System, Notification Service, Notification Controller, Database

**Flow:**

1. A new meeting triggers the notification module.
2. The Notification Service formats and queues the notification.
3. Notification is stored in the database.
4. The notification is sent to the user via the mobile frontend.

Figure 3.4.4 - Notification Dispatch Sequence

**3.5 Use Case Diagrams**

  Use case diagrams provide a high-level graphical representation of the interactions between users (actors) and the system. They help to visualize the functionality offered by Timease from the user's perspective and are crucial during the design phase to ensure all user needs are captured and appropriately addressed.

  In Timease, we identify two primary actor groups:

* **Regular Users**: Individuals who use the application to find professionals and schedule or book meetings.
* **Service Providers (Professionals)**: Users who offer services (e.g., doctors, consultants) and manage event availability.

**3.5.1 Use Case Diagram – Regular User**

**Actors**: Regular User  
**Use Cases**:

* Register an account
* Log in with email and password
* View available events
* Book a meeting
* Cancel a booking
* Receive notifications
* View personal schedule

This diagram captures all the actions a typical user can perform within the mobile app, focusing on discovery, scheduling, and managing appointments.

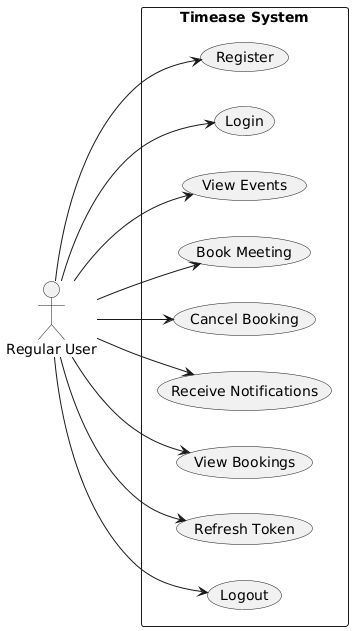


Figure 3.5.1 Use Case Diagram – Regular User

**3.5.2 Use Case Diagram – Service Provider**

**Actors**: Service Provider (Professional User)  
**Use Cases**:

* Register and complete profile
* Log in securely
* Create and manage events
* Define availability (daily/weekly/specific dates)
* View meeting bookings
* Send or trigger notifications
* Edit or delete events

This diagram highlights the control professionals have over scheduling parameters and engagement with their clients.

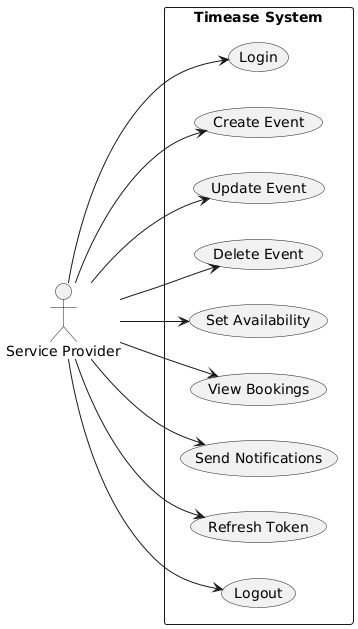


Figure 3.5.2 Use Case Diagram – Service Provider

**3.6 Database Design**

  The database design of *Timease* plays a critical role in ensuring data integrity, scalability, and seamless interaction between users, events, and scheduling logic. The system uses **PostgreSQL**, a powerful open-source relational database, hosted on **Neon**, which integrates smoothly with our Spring Boot backend. The database schema was designed to support modularity, role-based access control, secure authentication, and efficient scheduling.

**3.6.1 UML Diagram**

  The UML (Unified Modeling Language) diagram provides a conceptual overview of the major entities and their relationships. The main components represented in the UML diagram include:

* **User**: Core entity representing individuals in the system (either regular users or professionals).
* **Roles**: Defines user access levels (ROLE\_USER, ROLE\_ADMIN, ROLE\_MODERATOR).
* **Events**: Created by service providers to define the nature and scope of the meetings.
* **Availabilities**: Time slots when an event can occur, based on provider availability.
* **Meetings**: Instances of confirmed appointments between users and professionals.
* **Meeting Attendees**: Maps users to meetings they are attending.
* **Refresh Tokens**: Used to manage session continuity via JWT authentication.
* **Notifications**: Messages delivered to users for event confirmations, updates, or alerts.

Each class in the UML diagram encapsulates relevant attributes and outlines foreign key relationships to enforce referential integrity and cascade behaviors.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3.6.1 – UML diagram

**3.6.2 Entity Diagrams**

  Entity diagrams represent the physical implementation of the database schema. Each table aligns with a specific entity:

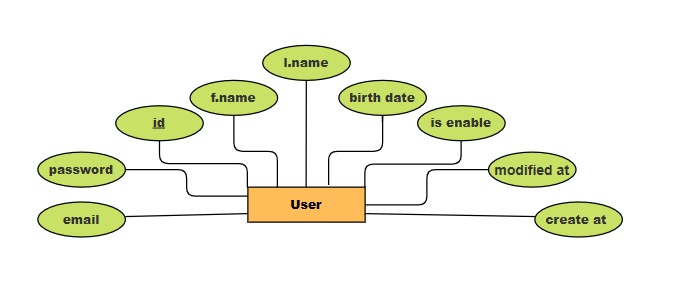
* User: Stores user details like names, birthdate, email, and account status.

Figure 3.6.2.1 - User Entity

* roles and user\_roles: Manage role-based access using a many-to-many relationship.

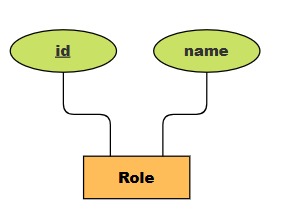


Figure 3.6.2.2 - Role Entity

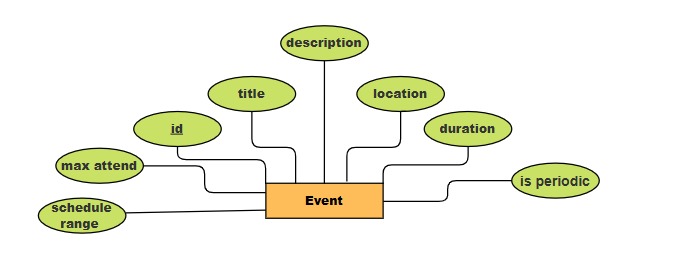
* events: Stores metadata for scheduled services.

Figure 3.6.2.3 - Event Entity

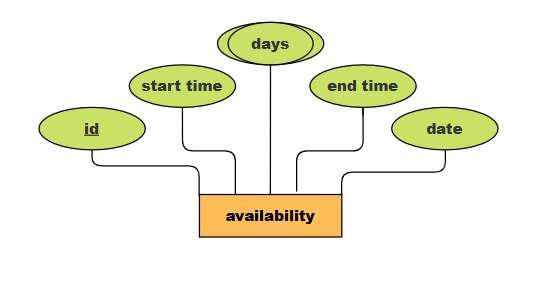
* availabilities: Encodes recurring or date-specific time slots tied to events.

Figure 3.6.2.4 - Availability Entity

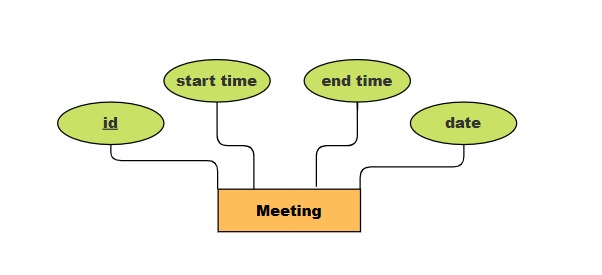
* meetings: Represents confirmed time allocations based on availability.

Figure 3.6.2.5 - Meeting Entity

* refresh\_tokens: Supports secure login sessions.

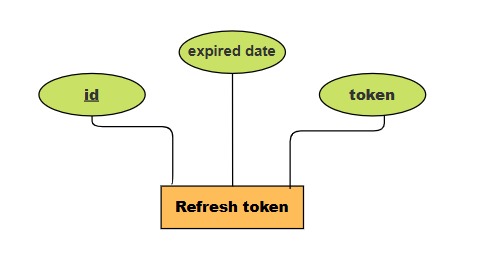


Figure 3.6.2.6 – Refresh token Entity

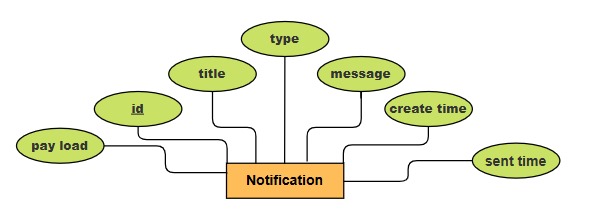
* notifications: Stores system-generated alerts for user interaction.

Figure 3.6.2 .7- Notification Entity

Primary keys are universally defined using UUIDs for uniqueness and distributed scalability. Foreign keys with cascading behavior ensure data consistency during deletions and updates.

**3.6.3 Relationship Diagrams**

  The relationship diagrams depict how entities are connected. Some key relationships include:

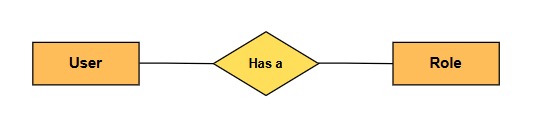
* A **user** can have multiple **roles**, and vice versa (many-to-many via user\_roles).

Figure 3.6.3.1 – User - role relation

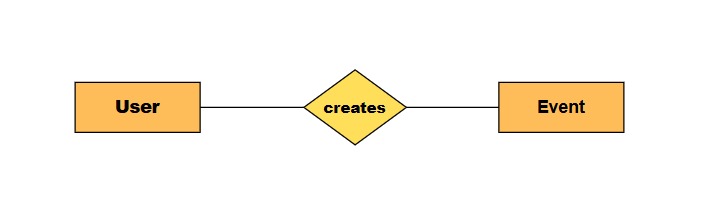
* A **user** (professional) can own multiple **events**.

Figure 3.6.3.2 – User - events relation

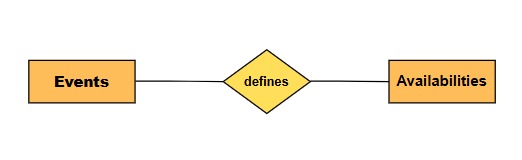
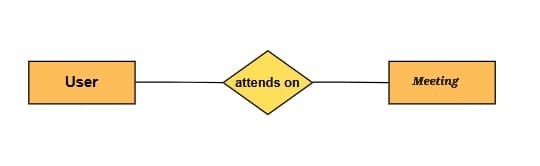
* Each **event** can have multiple **availabilities**, which in turn can have multiple **meetings**.

Figure 3.6.3.3 – Events - Availabilities relation

* Each **meeting** can include multiple **users** as **attendees**.

Figure 3.6.3.4 – User - Meeting relation

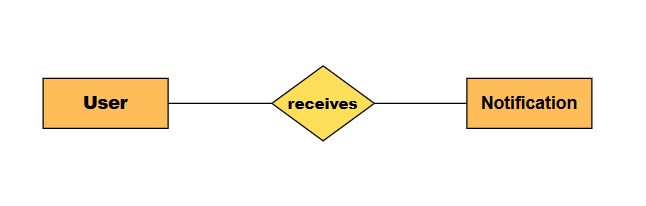
* Each **user** may receive multiple **notifications** related to meetings or events.

Figure 3.6.3.5 – User - Notification relation

These relationships were carefully structured to allow for:

* Fast retrieval of booking and availability data.
* Secure and efficient user-role authorization.
* Clean separation between scheduling, authentication, and communication components.

  Overall, the database schema supports a robust, secure, and scalable structure that meets the business logic requirements of Timease. It enables flexible expansion, efficient querying, and clean integration with our backend and frontend layers.

**CHAPTER 4**

**IMPLEMENTATION**

### ****4.1 Introduction****

  This chapter provides a detailed account of the implementation phase of the Timease platform, highlighting how the system was developed to transform design concepts into a fully functional software solution. The primary goal of this phase was to build a robust, scalable, and user-friendly scheduling and booking system that meets the needs of both service providers and their clients. The development process involved close alignment with the requirements outlined in earlier stages of the project and aimed to ensure the correct integration of all critical functionalities including user authentication, event management, availability scheduling, and real-time notifications.

  Timease was implemented as a full-stack solution, with the frontend developed using Flutter to ensure cross-platform mobile accessibility, and the backend implemented using Spring Boot, a powerful Java-based framework for building secure, production-grade APIs. PostgreSQL was chosen as the relational database due to its reliability, extensibility, and native support for structured queries. The backend and database were containerized using Docker, ensuring consistency between development and production environments. Deployment was handled through Railway, enabling a seamless CI/CD workflow and simplified cloud hosting.

  The development team adopted a modular and iterative approach during implementation, making use of version control tools such as GitHub for code management and collaboration. Continuous integration practices ensured that features were tested and integrated systematically. Each core module of the system—from authentication using JWT and refresh tokens, to the event and meeting management system, and finally the notification dispatch mechanism—was implemented incrementally and verified through real-time testing.

  The following sections delve deeper into the technologies and tools utilized during implementation, the rationale behind UI design decisions, and the integration strategies employed to connect the various system components into a cohesive whole.

### ****4.2 Tools and Technologies****

  The successful implementation of Timease relied on a carefully selected technology stack that ensured performance, security, scalability, and ease of development. This section outlines the core technologies, frameworks, and tools used across the backend, frontend, and DevOps layers of the system.

#### **Backend Technologies**

  The backend of Timease was developed using **Spring Boot**, a Java-based framework that simplifies the development of RESTful APIs with minimal configuration. It provided built-in support for dependency injection, robust security integration, and seamless database interaction through Spring Data JPA and Hibernate.

  To secure user authentication and authorization, **Spring Security** was employed, integrating **JWT (JSON Web Tokens)** for stateless access control along with **refresh tokens** to ensure session continuity. **Role-based access control** was enforced through enumerated roles such as ROLE\_USER and ROLE\_ADMIN, governing user permissions throughout the platform.

**PostgreSQL** was used as the relational database management system due to its strong support for ACID transactions, complex queries, and extensibility features. Database interaction was streamlined through **JPA annotations**, and schema changes were managed efficiently using native SQL and auto-generated entity mappings.

#### **Frontend Technologies**

  The client-facing interface of Timease was developed as a **Flutter mobile application**, allowing for a cross-platform experience on both Android and iOS devices using a single codebase. Flutter’s widget-based architecture provided full control over UI design, responsiveness, and performance. The app communicates with the backend via secure HTTP requests using token-based authentication headers.

#### **DevOps and Deployment**

  For development consistency and deployment efficiency, both the backend application and database were **containerized using Docker**. This ensured an isolated and reproducible environment across development, testing, and production stages. A custom **Docker Compose** setup was created to manage service dependencies such as PostgreSQL during local development.

  Deployment was handled through **Railway**, a platform-as-a-service that supports containerized application hosting with minimal configuration. Railway also provided seamless integration with **Neon**, a cloud-hosted PostgreSQL service. Continuous Integration and Deployment (CI/CD) pipelines were designed to build, test, and deploy the backend automatically upon updates to the main branch on GitHub.

  In summary, the combination of these technologies enabled the creation of a secure, modular, and maintainable system that could be deployed reliably across various environments.

### ****4.3 UI Design****

  The user interface of Timease was designed with a **mobile-first approach**, focusing on clarity, responsiveness, and user accessibility. As a scheduling and booking platform serving both professionals and clients, the UI needed to accommodate multiple user roles with differing functionalities, while maintaining a consistent and intuitive user experience. The decision to use **Flutter** as the frontend framework enabled the development of a cohesive, platform-agnostic mobile application, delivering a native-like performance on both Android and iOS devices.

  The design process emphasized usability and minimalism. The color scheme, typography, and layout were chosen to create a clean and modern aesthetic, reinforcing user trust and reducing visual clutter. Icons and components were chosen to enhance navigation and clearly communicate functionality, such as creating events, viewing available time slots, and managing bookings.

  Key UI elements include:

* **Authentication Screens**: Sign up, login, and password recovery pages that handle user onboarding securely with proper input validation and feedback.
* **Home Dashboard**: Provides quick access to upcoming meetings, event creation shortcuts, and notifications. Content is dynamically rendered based on the user's role.
* **Event Management Screens**: For professionals, the app allows creating, editing, and deleting events. Time slot availabilities can be set using an interactive calendar and time picker interface.
* **Booking Interface**: For end users, a calendar view enables easy browsing of available slots, while the booking flow is designed to be frictionless and efficient.
* **Notification Panel**: Users are notified about successful bookings, cancellations, or system messages through a dedicated notification center.

  The UI was iteratively improved during development, informed by user testing and feedback. Particular attention was paid to accessibility by ensuring adequate contrast, legible fonts, and intuitive navigation patterns. The Flutter widget tree structure also allowed for easy reuse of components and a modular UI system.

  Overall, the UI design of Timease complements its functionality by delivering a seamless user experience, making scheduling and time management straightforward and efficient.

**CHAPTER 5**

**CONCLUSION AND FUTURE WORK**

## Conclusion

Give a brief statement of how the solution that you have provided addresses the problem stated in the introduction. Provide an evaluative statement based on the results. You should not introduce new material.

## Future Work

Future Work: here you will recommend what is needed to be done in the future to your work. This is done in GP2.

**REFERENCES**

Cerny, V. 1985.Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm, *J. Opt. Theory Appl.*, volume 45, number1, pages.41-51.

References: for your Final Year project, it is required that you cite and reference work to which you owe an intellectual debt. It is required that you cite and reference work that provides supporting evidence. It is required that you cite and reference work so that the reader can find the sources that have been quoted.

In other words, the purposes of a reference are to acknowledge the contributions of other authors and to enable readers to locate source easily. In this section, you can use alphabetically or numerically system (e.g. APA style referencing).

**General guidelines:**

**APA Reference List Examples**

* **Book with Single Author:**

Gore, A. (2006). *An inconvenient truth: The planetary emergency of global warming and what we can do about it.* Emmaus, PA: Rodale.

**In-text reference:** (Gore, 2006)

* **Book with Two Authors:**

Michaels, P. J., & Balling, R. C., Jr. (2000).*The satanic gases: Clearing the air about global warming*. Washington, DC: Cato Institute.

**In-text reference:** (Michaels & Balling, 2000)

* **Book with Editor as Author:**

Galley. K. E. (Ed.). (2004). *Global climate change and wildlife in North America.*Bethesda, MD: Wildlife Society.

**In-text reference:** (Galley, 2004)

* **An Anonymous Book:**

*Environmental resource handbook*. (2001). Millerton, NY: Grey House.

**In-text reference:** (Environmental Resource Handbook, 2001)

* **Articles in Reference Books (unsigned and signed):**

Greenhouse effect.(2005). *American heritage science dictionary*. Boston, MA: Houghton Mifflin.

Schneider, S. H. (2000). Greenhouse effect.*World book encyclopedia* (Millennium ed. Vol. 8, pp. 382-383). Chicago, IL: World Book.

**In-text references:** (Greenhouse effect, 2005)

(Schneider, 2000)

* **Journal Article when each issue begins with p.1:**

Bogdonoff, S., & Rubin, J. (2007). The regional greenhouse gas initiative: Taking action in Maine. *Environment, 49*(2), 9-16.

**In-text reference:** (Bogdonoff& Rubin, 2007)

* **Website:**

United States Environmental Protection Agency. (2007, May 4). *Climate Change*. Retrieved From the Environmental Protection Agency website: http://www.epa.gov/climatechange

**In-text reference:** (United States Environmental, 2007)

Gelspan, R. (2007). *The Heat Is Online*. Lake Oswego, OR: Green House Network. Retrieved from The Heat Is Online website: http://www.heatisonline.org

In-text reference: (Gelspan, 2007)

**How to Cite an Website in APA**

**Structure:** Last, F. M. (Year, Month Date Published). Article title.*Website Title*.Retrieved Month Date, Year, from URL.

**Example:**

* Satalkar, B. (2010, July 15). Water aerobics. *Buzzle*.com. Retrieved July 16, 2010, from http://www.buzzle.com.
* Cain, K. (2012, June 29). The Negative Effects of Facebook on Communication. *Social Media Today RSS*. Retrieved January 3, 2013, from [http://socialmediatoday.com](http://socialmediatoday.com/).

OR numerically as below

All reference items must be in 10pt font. Please use Regular and Italic styles to distinguish different fields. Number the reference items consecutively in square brackets (e.g. [1]).

When referring to a reference item, please simply use the reference number, as in [2]. Do not use “Ref. [3]” or “Reference [3]” except at the beginning of a sentence, e.g. “Reference [3] shows …”. Multiple references are numbered with one bracket and separated with comas (e.g. [2], [2, 3], [4 – 6]).

Structure: First Author Last Name, First Author First Name.

[1]**Author. (Date published if available; n.d.--no date-- if not). Title of article/book. *Title of web site* . Retrieved date. From URL.**

[2]Author. (Date published if available; n.d.--no date-- if not). Title of article/book. Title of web site . Retrieved date. From URL.

**APPENDICES**

**Appendix A:**

Appendices: the Report must contain an appendix explaining file structure on thedata CD submitted with it. The appendix must also contain information on how thecode should be run. Other appendices may include documents such as: the projectproposal; a selection of experimental data; schedules; testing strategy; riskmanagement plans; glossary; manual; etc. Don't include the source code as anappendix (submit it on CD; see below). Don't include voluminous appendices (theseshould also be submitted on a CD). A report template can be found in the collegewebsite.

**Timeline/Milestones (Gantt Chart)**

This course is similar to self-study/research. Weekly meetings are scheduled with the supervisor for the project. Each student’s group will meet together weekly, keeping detailed minutes of the meetings.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Task |  | Weeks | | | | | | | |
|  |  | 1-2 | 3-4 | 5-6 | 7-8 | 9-10 | 11-12 | 13-14 | 15 |
|  |  |  |  |  |  |  |  |  |  |
| 1. …….. |  |  |  |  |  |  |  |  |  |
| 2. …….. |  |  |  |  |  |  |  |  |  |
| 3. …….. |  |  |  |  | |  |  |  |  |
| 4. …….. |  |  |  |  |  |  | |  |  |
| 5. …….. |  |  |  |  |  |  |  |  |  |
| ……….. |  |  |  |  |  |  |  |  | |