Assignment #2 – Project Demo - PACT

Course code: 24F\_CST8319\_450

Facilitator: Moe Osman

Group Assignment

Project Proposal: PACT

***Group 11***

|  |  |
| --- | --- |
| **Name** | **Student number** |
|  |  |
| Abou-El-Kheir, Adam | 041139787 |
| Concepcion Perez, Michelle | 041044464 |
| Dinesh Ranaweera | 041090901 |
| Laperrière, François | 041125096 |
| Qingyi Zhang | 040545120 |

Date: November 5th , 2024.

# Assignment #2 - Planning & Calendar Tool (PACT)

Table of Contents

[Assignment #2 - Planning & Calendar Tool (PACT) 1](#_Toc480412168)

[Introduction 2](#_Toc1264264920)

[1. System and Services Architecture 3](#_Toc1306079327)

[2. UML Diagrams 8](#_Toc1100283772)

[3. Entity-Relationship Diagram 10](#_Toc1964268921)

[3.1. Users Table 11](#_Toc1436382429)

[3.2. Appointments Table 12](#_Toc1587954294)

[3.3. User\_Appointments Table 12](#_Toc397154657)

[3.4. Calendars Table 12](#_Toc895441579)

[3.5. User\_Logins Table 12](#_Toc251184790)

[4. Class Diagram 12](#_Toc1587849565)

[Classes and Relationships 13](#_Toc2117016715)

[Conclusion 16](#_Toc15647174)

# Introduction

PACT (Planning A Calendar Tool) is a web application designed to streamline the scheduling process. It allows clients to create, manage, and book appointments effortlessly, reducing the need for back-and-forth emails and improving time management for both personal and business use. In this phase, we focus on developing a detailed system architecture, creating refined UML and Class Diagrams, and integrating backend services that enhance scalability, reliability, and performance. The objective is to build a robust structure that supports PACT’s functionality and provides a seamless user experience.

## 1. System and Services Architecture

The **System Architecture** of PACT is designed in layers to separate different functionalities and responsibilities, providing a clear structure for managing user interactions, business logic, and data storage.

1. **Client/Presentation Layer (Frontend)**:
   * This is the user-facing part of the application, accessible via a web interface. Users can perform actions like registration, login, and booking appointments, which are sent as requests to the backend.
2. **Application Layer (Backend)**:
   * This layer handles the core business logic, including user authentication, booking management, and notification functions. Components like Java Servlets and JSP generate dynamic content and process client requests.
3. **Data Access Layer**:
   * Manages database interactions using JDBC and optionally Hibernate/JPA for object-relational mapping. It facilitates CRUD (Create, Read, Update, Delete) operations on the database and maintains efficient data retrieval and storage.
4. **Database Layer**:
   * A MySQL database stores all essential data, including user profiles, schedules, and booking information. It is optimized with indexing to improve query performance and supports reliable data storage.
5. **External Services**:
   * Optional services, such as an email API for notifications and HTTPS for secure communication, are used to enhance functionality and security without overloading internal resources.

Figure #1 – System Architecture Diagram  
A diagram of a software development process

Description automatically generated

**Services Architecture**

The **Services Architecture** outlines the key backend services that support PACT’s main features. This modular structure allows each service to operate independently, promoting scalability and easier maintenance.

1. **Client/Presentation Layer (Frontend)**:
   * Users access PACT through a web browser interface, which sends requests to the backend. This layer handles all user interactions and visual elements.
2. **Load Balancer**:
   * Distributes incoming requests evenly across backend services, improving scalability and reliability by preventing server overload and ensuring consistent performance.
3. **Core Backend Services**:
   * **Authentication Service**: Manages user login, registration, and session handling.
   * **Notification Service**: Sends appointment reminders and confirmations via an external email API.
   * **Availability Management Service**: Allows users to manage their availability with calendar integration.
   * **Appointment Booking Service**: Handles appointment scheduling, including caching for faster access.
   * **Dashboard Service**: Displays user schedules and notifications in a user-friendly interface.
4. **Data Persistence Service**:
   * Manages communication with the database, using tools like Hibernate/JPA and connection pooling to improve efficiency.
5. **Database Layer**:
   * The MySQL database securely stores user data, schedules, and booking information.
6. **External Email API**:
   * An optional service to handle notification delivery, reducing the load on internal resources.

Figure #2 – Service Architecture Diagram  
A close-up of a computer screen

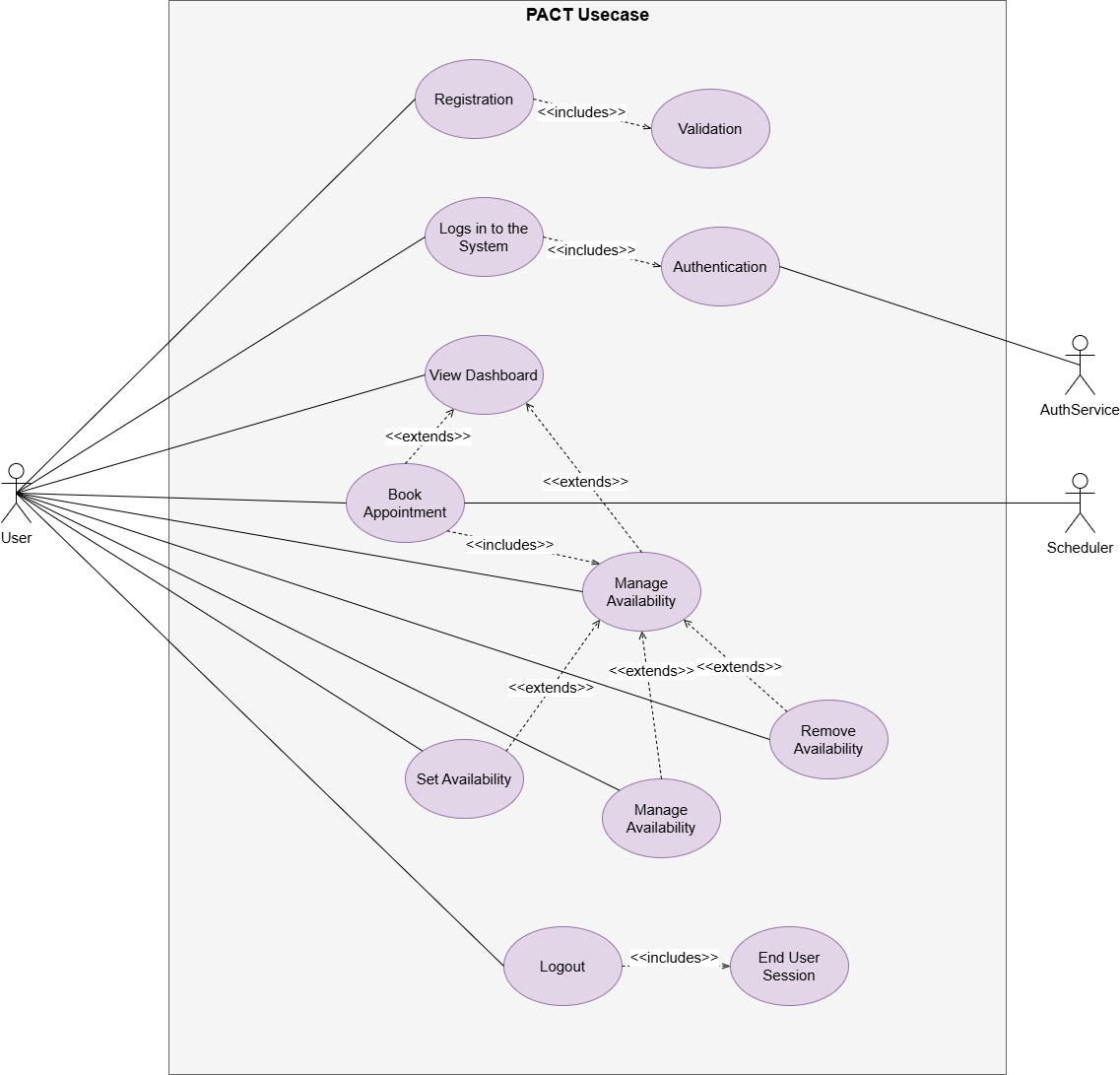
Description automatically generated

## 2. UML Diagrams

2.1 Use Case Diagram

The Use Case Diagram shows the interactions between the User, AuthService and Scheduler.

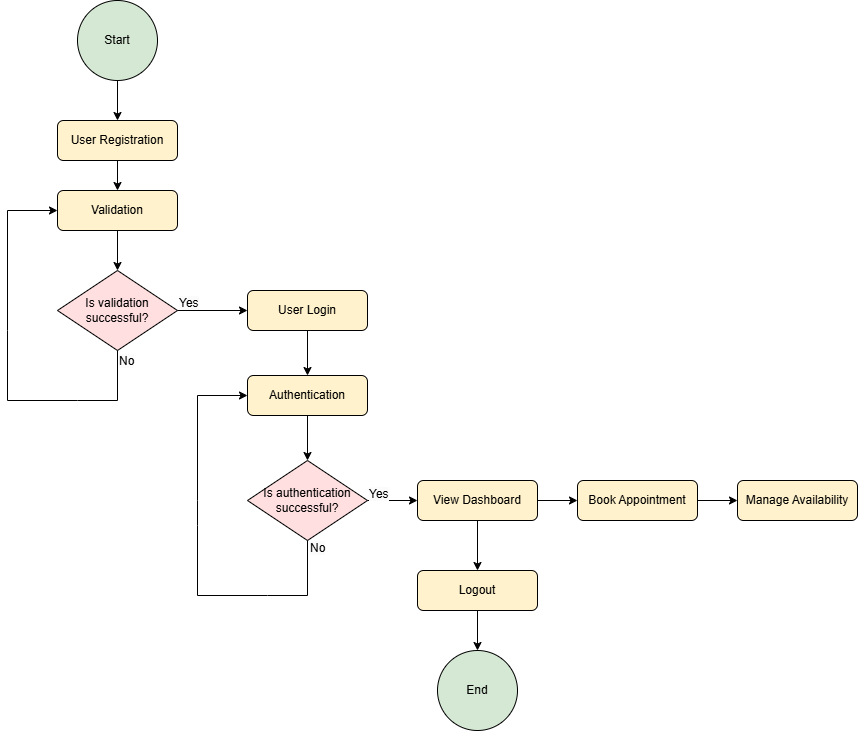
Key use cases include Registration, Login, Book Appointment and Mange Availability. The diagram uses ‘includes and ‘extends’ relationships to clarify dependencies and optional actions.



#### Figure #3. PACT Use Diagram

2.2 Activity Diagram

The Activity Diagram represents the user flow through the system, starting from Registration to Logout. The flow includes decision points for validation and authentication checks, leading to action such as Dashboards, Book Appointment and Mange Availability.



#### Figure #4. PACT Activity Diagram

## 3. Entity-Relationship Diagram

A screenshot of a computer

Description automatically generated

#### Figure #5 Entity-Relationship Diagram

### 3.1. Users Table

Stores the information about each user in the scheduling application, including authentication details, contact information, and account status.

### 3.2. Appointments Table

Manages details for each appointment, including timing, location, and associated calendar. This table records individual appointments, each of which can involve multiple users.

### 3.3. User\_Appointments Table

Acts as an associative table to convert logical many-to-many relationship between users and appointments into one-to-many physical relationships. This allows multiple users to participate in each appointment with specific roles and statuses.

### 3.4. Calendars Table

Stores each user’s calendar information, allowing users to organize appointments by different categories (e.g., work, personal). Each user can have multiple calendars.

### 3.5. User\_Logins Table

Tracks each login event for each user, recording timestamps for recent logins to provide user activity statistics on the dashboard.

## 4. Class Diagram

The class diagram for the PACT application represents the core structure and relationships between different components of the system. It illustrates how users interact with the system, manage their profiles, set availabilities, book appointments, and receive notifications.

### Classes and Relationships

**User**

* Attributes: userID, email, password, role
* Methods: register(), login(), logout()
* Relationships:
* One-to-One with Profile
* One-to-One with Dashboard
* One-to-Many with Availability, Appointment, Notification
* One-to-One with Calendar

**Profile**

* **Attributes**: profileID, name, phoneNumber
* **Methods:** updateProfile()
* **Relationships:**
* One-to-One with User

**Availability**

* **Attributes:** availabilityID, availableDate, startTime, endTime
* **Methods:** setAvailability(), updateAvailability(), removeAvailability()
* **Relationships:**
* Many-to-One with User
* One-to-Many with Appointment

**Appointment**

* **Attributes:** appointmentID, appointmentDate, startTime, endTime, status
* **Methods**: createAppointment(), updateAppointment(), cancelAppointment()
* **Relationships:**
* Many-to-One with User
* Many-to-One with Availability
* Many-to-One with Calendar

**Dashboard**

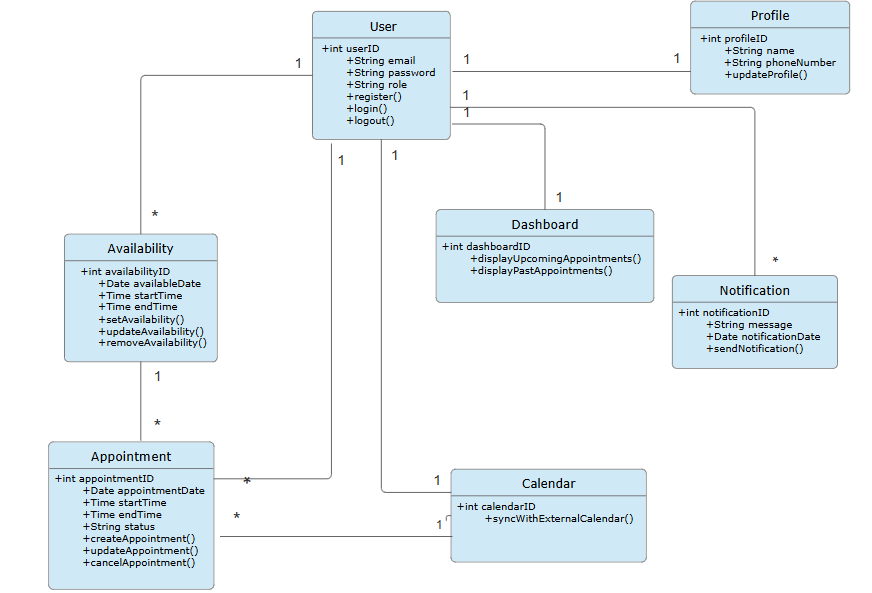
* **Attributes:** dashboardID
* **Methods:** displayUpcomingAppointments(), displayPastAppointments()
* **Relationships:**
* One-to-One with User

**Calendar**

* **Attributes:** calendarID
* **Methods:** syncWithExternalCalendar()
* **Relationships:**
* One-to-One with User
* Many-to-One with Appointment

**Notification**

* **Attributes:** notificationID, message, notificationDate
* **Methods:** sendNotification()
* **Relationships:**
* Many-to-One with User

 Figure #6 Class Diagram

# Conclusion

In this phase of the PACT project, we created detailed documentation to show how the system works and how different parts connect. The **Use Case Diagram** maps out the main actions users can take, like registering, booking appointments, and setting availability, giving an overview of user interactions. The **Activity Diagram** breaks down the steps involved in core tasks, such as registration and scheduling, showing how each action flows in the system. The **Class Diagram** (if applicable) shows how key components like User, Task, and Notification relate to each other, making it clear how data is organized. The **Entity-Relationship Diagram (ERD)** illustrates the relationships between main data entities, like Users, Appointments, and Availability slots, helping us understand how data is stored and connected in the database. Finally, the **Architectural Diagram** provides an overall look at the system’s structure, highlighting the presentation, business logic, and data layers. Together, these diagrams help us understand how PACT is built and supports efficient scheduling and appointment management.