MedSync – A centralized platform for medical resource scheduling

Course Project: Software System Design

Vazquez, Eli

Penn State University SWENG837

Table of Contents

[Problem Statement & Requirements 2](#_Toc196680699)

[Problem Statement 2](#_Toc196680700)

[Business Requirements (Functional Requirements) 2](#_Toc196680701)

[Non-Functional Requirements 2](#_Toc196680702)

[System Design Using Domain Modeling 3](#_Toc196680703)

[Use Case Diagram 3](#_Toc196680704)

[Domain Model 4](#_Toc196680705)

[Class Diagram 0](#_Toc196680706)

[Sequence Diagrams 0](#_Toc196680707)

[Schedule Resource Sequence Diagram 0](#_Toc196680708)

[Modify/Cancel Booking Sequence Diagram 0](#_Toc196680709)

[Manage Resources Sequence Diagram 1](#_Toc196680710)

[Manage Users Sequence Diagram 1](#_Toc196680711)

[Search and Filter Resources Sequence Diagram 2](#_Toc196680712)

[State Diagram 2](#_Toc196680713)

[Booking State 2](#_Toc196680714)

[Activity Diagram 3](#_Toc196680715)

[Scheduling Activity Diagram 3](#_Toc196680716)

[Component Diagram 3](#_Toc196680717)

[Deployment Diagram 0](#_Toc196680718)

[Skeleton Classes & Table Definitions 0](#_Toc196680719)

[Design Patterns 0](#_Toc196680720)

# Problem Statement & Requirements

## Problem Statement

“Small and medium-sized clinics often face fragmented processes when scheduling shared resources such as exam rooms, specialized equipment, and provider time slots. Communication between nurses, physicians, and administrative staff relies heavily on ad-hoc methods such as hallway conversations, sticky notes, and disconnected calendars. This lack of a cohesive system architecture leads to double-bookings, underutilized resources, and delays in patient care.

The design challenge lies in developing a centralized platform that streamlines resource scheduling, role-based access, and real-time communication across clinic staff. The system must address issues of event coordination, calendar synchronization, user interface clarity for diverse roles (e.g., nurse vs. admin), and secure access control.”

## Business Requirements (Functional Requirements)

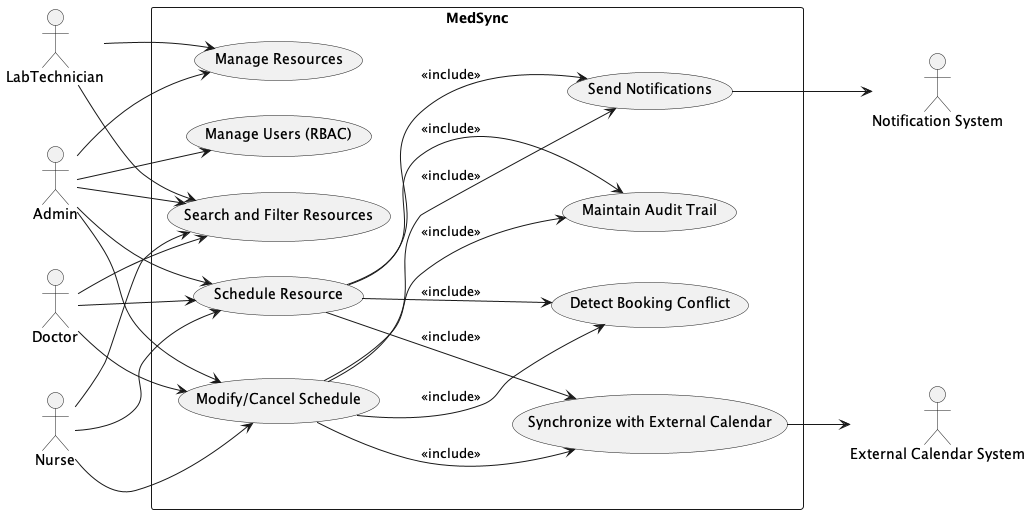
* FR1: The system must allow clinic staff to schedule resources in a centralized scheduling system.
* FR2: Support role-based access control (RBAC) to ensure users access only relevant functionalities and data.
* FR3: Enable real-time conflict detection for overlapping bookings.
* FR4: Provide notifications (email or in-app) to users when:
  + Bookings are created, modified, or canceled.
  + Conflicts arise.
* FR5: Integrate with external calendars (Google Calendar, Outlook) for synchronization.
* FR6: Maintain an audit trail of scheduling changes (who did what and when).
* FR7: Provide search and filter functionalities for available resources.

## Non-Functional Requirements

* NFR1: Scalability:
  + Must handle up to 100 concurrent users without performance degradation.
* NFR2: Security:
  + Role-based access:
    - admin
    - nurse
    - doctor
  + Data encryption at rest (AES-256) and in transit (TLS 1.2+).
  + Multi-factor authentication (MFA) for administrative users.
  + Audit logging for key operations.
* NFR3: Performance:
  + Response time for scheduling operations < 2 seconds.
  + Calendar updates should reflect in real-time.
* NFR4: Maintainability:
  + Use of modular microservices to separate concerns (scheduling, user management, notifications).
  + CI/CD pipeline with automated testing (unit, integration).
  + Documentation for APIs and deployment.
* NFR5: Availability:
  + System availability of 99.9% uptime.
* NFR6: Compliance:
  + Ensure HIPAA-compliance for storing sensitive patient-related data.

# System Design Using Domain Modeling

## Use Case Diagram



## Domain Model

A screenshot of a computer

AI-generated content may be incorrect.

## Class Diagram

A diagram of a company

AI-generated content may be incorrect.

## Sequence Diagrams

### Schedule Resource Sequence Diagram

A screenshot of a project

AI-generated content may be incorrect.

### Modify/Cancel Booking Sequence Diagram

A screenshot of a computer program

AI-generated content may be incorrect.

### Manage Resources Sequence Diagram

A screenshot of a computer

AI-generated content may be incorrect.

### Manage Users Sequence Diagram

A diagram of a software program

AI-generated content may be incorrect.

### Search and Filter Resources Sequence Diagram

A diagram of a workflow

AI-generated content may be incorrect.

## State Diagram

### Booking State

A diagram of a business

AI-generated content may be incorrect.

## Activity Diagram

### Scheduling Activity Diagram

A diagram of a system

AI-generated content may be incorrect.

## Component Diagram

A diagram of a software application

AI-generated content may be incorrect.

## Deployment Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## Skeleton Classes & Table Definitions

Refer to the following repository for the skeleton classes and table definitions under the scripts/ directory: <https://github.com/swevazquez/sweng837CourseProject>

## Design Patterns

The MedSync system leverages a combination of design patterns, microservices architecture, and GRASP principles to ensure scalability, maintainability, and modularity.

* GoF (Gang of Four) Design Patterns
  + Observer Pattern: Applied for the *NotificationService*, where observers (*EmailNotification*, *SMSNotification*, *InAppNotification*) are notified when a booking event occurs.
  + Abstract Factory Pattern: Used for resource creation (*ExamRoomFactory*, *LabFactory*, *MedicalEquipmentFactory*) to allow flexible and scalable management of resource types. This encapsulates the creation logic, allowing future expansion without altering core booking logic.
  + Adapter Pattern: Implemented for external calendar integration (Google Calendar, Outlook). The *CalendarAdapter* interface allows MedSync to integrate with various calendar systems without tightly coupling the internal logic to external APIs.
* Microservices Architecture
  + Decomposition of services: The system is divided into independent microservices, including:
    - Booking Service
    - User Management Service
    - Notification Service
    - Authentication Service
  + Independent Databases: Each microservice has its own isolated database (PostgreSQL), promoting loose coupling and data autonomy as per microservices best practices.
* GRASP (General Responsibility Assignment Software Patterns)
  + Controller: The *MedSyncController* acts as the system’s entry point for handling requests, coordinating between different services (Booking, User Management) to achieve high cohesion and low coupling principles.
  + Creator: Responsibility for creating Booking objects resides with the service classes that aggregate and manage their lifecycle.
  + Information Expert: Each domain class (Booking, User, Resource) is assigned responsibilities based on the information it controls. For example, the Booking class manages scheduling details and related state transitions.
  + Low Coupling: To promote independent scalability and ease of maintenance, we introduce the Observer and Adapter patterns reduce dependencies between components (booking and notifications, calendar integrations).
  + High Cohesion: Services and components are focused on a single responsibility.