CSAI 203 - Fall 2025

Introduction to Software Engineering

Medibook

Team:

Ahmed Ragheb 202301566

Ammar Yasser 202400963

Ahmed Ibrahem 202402177

Yossef Ahmed 202300216

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed description of the requirements for the MediBook: Doctor Appointment System. This document will serve as the foundational agreement between the project stakeholders and the development team.

It will describe the system's intended purpose, its scope, the primary features it will provide, the constraints under which it must operate, and the criteria for its success. This SRS will be used to guide the design, implementation, and testing (verification) phases of the project.

1.2 Scope

This document specifies the requirements for MediBook, a web-based application designed to connect patients with doctors and facilitate the process of booking medical appointments.

The system will provide the following primary capabilities:

1) **User Account Management: The ability for users (Patients and Doctors) to register, log in, and manage their profiles.**
2) **Advanced Doctor Search: The ability for Patients to search for doctors based on medical specialty (e.g., Cardiology, Dermatology) and geographic location (city or area).**
3) **Doctor Profile Management: The ability for Doctors to create a detailed profile, including their specialty, qualifications, clinic address(es), and a profile picture.**
4) **Appointment Management System:**
    - ✓ **The ability for Doctors to define their available time slots.**
    - ✓ **The ability for Patients to view available slots and book an appointment.**
    - ✓ **The ability for Doctors to confirm or decline booking requests.**
5) **Administrator Dashboard: The ability for an Admin to verify doctor accounts and manage the system's list of medical specialties.**

Out of Scope:

To clarify the project's boundaries, the following features will not be included in this version:

- **An integrated online payment system for consultation fees.**
- **Real-time video consultation (telehealth) functionality.**
- **An e-prescription or pharmacy integration system.**
- **Native iOS or Android mobile applications.**

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
SRS	Software Requirements Specification
Patient	A registered user seeking to book a medical appointment.
Doctor	A registered and verified medical professional offering appointments.
Admin	A privileged user responsible for system maintenance and verification.
Slot	A specific block of time (e.g., 30 minutes) that a Doctor makes available for booking.

1.4 References

1. **[CSAI 203 Course Project Requirements, Fall-2025. Dr. Mohamed Sami Rakha.]**
2. **[IEEE Std. 830-1998, IEEE Recommended Practice for Software Requirements Specifications.]**

1.5 Overview

This SRS document is structured as follows:

1) **Section 1 (Introduction) provides an overview of the project's purpose and scope.**

2) **Section 2 (Overall Description) describes the product perspective, key functions, user characteristics, and operational environment.**

3) **Section 3 (Specific Requirements) contains the detailed functional requirements, use case models, domain model, and testable non-functional requirements.**

4) **Section 4 (Appendices) contains supplementary information.**

2. Overall Description

2.1 Product Perspective

The MediBook Doctor Booking Application is a mobile and web-based platform designed to connect patients with healthcare providers (doctors, clinics, and medical centers). The system acts as an intermediary that enables users to search for doctors, book medical appointments, make payments, and submit reviews.

The system follows a **client–server architecture** and uses a centralized database to store user accounts, doctor information, appointment schedules, payments, and reviews.

Main System Components:

Component	Description
Frontend (User Interface)	Mobile App (Android/iOS) and Web App for browsing doctors and booking appointments
Backend (Server API)	RESTful API handling login, search, booking, payment, and notification services
Database (Model Layer)	Stores users, doctors, appointments, specialties, locations, reviews, and payments
External Services	SMS gateway, email service, and online payment API

The application requires an active internet connection and does not depend on any external internal systems except for communication services (SMS, email) and online payment providers.

2.2 Product Functions

The main functions of the system include:

1-User Account Management

a) Account registration (Patient / Doctor)

b) Login using email or phone number

c) Password reset and profile editing

2- Doctor Search and Appointment Booking

1) Search doctors by specialty, location, insurance, fees, and rating
2) View doctor profile (bio, experience, clinic info, schedule, ratings)
3) Book, cancel, or reschedule appointments

3-Payments and Notifications

I. Pay at clinic or online payment (credit card / wallet)
II. Automated SMS/Email notifications for appointment confirmation and reminders

4- Review and Rating System

1. Patients can rate doctors after completed visits
2. Doctors can view and respond to reviews

5-Doctor Dashboard

- Manage appointment schedule
- Edit clinic details, consultation fees, availability
- View patient visits and received ratings

6-Admin Dashboard

A. Approve doctor accounts (identity and license verification)
B. Manage users, appointments, and system data
C. Review complaints, flags, or report issues

2.3 User Classes and Characteristics

User Class	Description	Technical Expertise	Key Responsibilities
Patient	Regular user booking medical appointments	Basic	Register, search, book, pay, review
Doctor	Licensed medical provider listed in the app	Moderate	Manage schedule, view bookings, edit profile
Admin	System operator supervising platform operations	Advanced	Approve doctors, manage users and data
Clinic Assistant (optional)	Staff member managing appointments for a clinic	Moderate	Confirm patient attendance, modify booking status

---

2.4 Operating Environment

1-Client Environment

a) Android 8.0+ / iOS 13+ (Mobile App)

b) Any modern web browser (Chrome, Safari, Firefox, Edge)

2-Server Environment

1) Backend: Django / Laravel / Node.js (depending on implementation)
2) Database: MySQL or PostgreSQL
3) Deployed on cloud hosting (AWS, Azure, DigitalOcean)

External Dependencies

a. SMS Gateway (e.g., Twilio, Vodafone SMS API)

b. Email Notification System

c. Payment Gateway (e.g., Paymob, Stripe, PayPal)

2.5 Design and Implementation Constraints

Constraint Type	Description
Technology Constraint	Must support both mobile and web clients through REST API
Security Constraint	Patient medical data must be encrypted and access-controlled
Payment Constraint	Relies on third-party payment gateway
Privacy Constraint	Must comply with medical data protection policies (HIPAA-like)
Performance Constraint	Search results must load in < 2 seconds

---

2.6 User Documentation

The system will provide the following documentation:

1. **User Guide (PDF / In-App Help Section)**
    I. Account creation
    II. Searching and booking doctors
    III. Payment and cancellation policies
2. **Doctor Manual**
    1. Profile setup and verification
    2. Managing schedules and appointments
    3. Viewing ratings and patient feedback
3. **Admin Manual**
    1) Approving doctor registrations
    2) Managing users, records, and system databases
    3) Monitoring reports and analytics

2.7 Assumptions and Dependencies

1-Assumptions

a) Users have access to a stable internet connection
b) Doctors are licensed and verified before activation

c) Patients understand basic app navigation and appointment booking

d) All data (doctor info, fees, locations) is stored in the platform database

2-Dependencies

a) SMS/Email services for notifications

b) Payment gateway for online transactions

c) Cloud hosting for backend system availability

d) Mobile app store approval (Google Play / App Store)

3.Specific Requirements

3.1 Functional Requirements

The system shall provide the following functional requirements. Each requirement includes a brief scenario to clarify behavior.

ID	Requirement	Description	Example Scenario
FR-01	User Registration	Allow new patients to register with name, email, phone, password, ensuring email uniqueness.	A visitor enters details and is registered if valid.
FR-02	User Login	Allow users to log in securely using email and password.	User enters valid credentials, system logs them in.
FR-03	Doctor Search	Allow users to search/filter doctors by name, specialty, location, or price.	User searches for cardiologist in Cairo and gets results.
FR-04	View Doctor Profile	Display doctor's info: bio, clinic location, price, schedule.	User clicks a doctor and views full details.
FR-05	Book Appointment	Allow user to select an available slot and book.	User selects time slot, system confirms booking.
FR-06	Cancel Appointment	Allow users to cancel bookings.	User cancels appointment, slot becomes available.

FR-07	View Booking History	Allow users to view upcoming and past appointments.	User views appointment history.
FR-08	Doctor Registration	Allow doctors to register with clinical & professional info.	Doctor submits details, admin approves.
FR-09	Manage Schedule	Allow doctors to manage available slots.	Doctor adds slots for weekdays 4pm–8pm.
FR-10	Manage Appointments	Allow doctors to accept/reject bookings.	Doctor accepts booking, user notified.
FR-11	Admin Manage Users	Admin can view/suspend/delete users.	Admin suspends spam account.
FR-12	Admin Manage Clinics	Admin approves or blocks doctors/clinics.	Admin reviews and approves new doctor.

3.2 Use Case Model

Describes the system's functional interactions between users and the system.
The Use Case Model represents the actions that each system actor can perform and how the system responds to those actions.

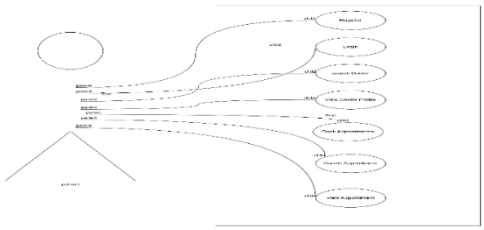
The system includes three main actors:

1. **Patient** – searches for doctors, views profiles, and books appointments.
2. **Doctor/Clinic** – manages profile, schedule, and appointment requests.
3. **Admin** – manages platform users and approves doctors/clinics.

The following subsections contain the system use case diagrams and detailed use case descriptions.

3.2.1 Use Case Diagrams

https://tinyurl.com/nhja532j



3.2.2 Use Case Descriptions

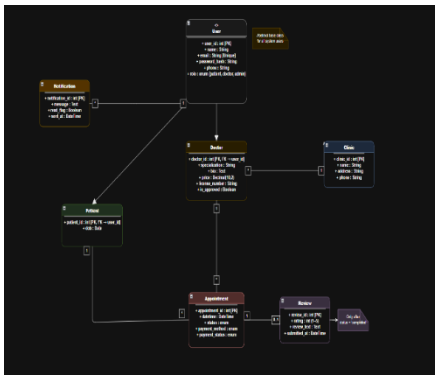
Example Use Case Template (IEEE Standard):

Use Case ID	UC-01
Name	Book Appointment
Actor	Patient
Preconditions	User is authenticated and viewing doctor profile.
Main Flow / Alternative Flow	1. User selects slot 2. Confirms booking 3. System saves appointment Alternative: Time slot unavailable → system prompts user to select another

Additional Use Cases to be filled similarly: Login, Register, Cancel Appointment, Manage Schedule, Admin Approvals.

3.3 Domain Model

https://tinyurl.com/457wusfx



The conceptual domain model outlines the main system entities and how they relate to each other.
It identifies core classes such as *User*, *Doctor*, *Patient*, *Clinic*, and *Appointment*, describing their key attributes and interactions.
For instance, a clinic can have many doctors, and each doctor can manage multiple appointments with patients.
This model provides a clear overview of the system's structure before moving to detailed design.

3.3.1 Conceptual Classes

Class	Attributes
User	user_id, name, email, password_hash, role
Patient	patient_id, phone, dob
Doctor	doctor_id, specialization, bio, price, clinic_id
Clinic	clinic_id, name, address, phone
Appointment	appointment_id, patient_id, doctor_id, datetime, status
Notification	notif_id, user_id, message, read_flag

3.3.2 Class Descriptions

UML Class Diagram to be inserted.
Classes and multiplicity relationships:
- Clinic 1..* Doctor
- Doctor 1..* Appointment
- Patient 1..* Appointment

3.4 Non-Functional Requirements

The system must satisfy several non-functional requirements to ensure quality, performance, and maintainability. Each requirement below includes a short description and testing criteria.

ID	Requirement	Description	Testing / Evaluation Criteria
NFR-01	**Performance**	The system should respond to key operations (search, booking, login) within **2 seconds** under normal load conditions.	Measure the average response time for several requests; the test passes if the average is below 2 seconds.
NFR-02	**Security**	User passwords must be stored using secure hashing (e.g., bcrypt or Werkzeug security). Access to admin or doctor dashboards must be role-restricted.	Inspect the database to confirm hashed passwords and attempt unauthorized access to verify that it is blocked.
NFR-03	**Usability**	The interface should be simple, intuitive, and usable on both desktop and mobile browsers.	Manually test the UI on different screen sizes and confirm that all major functions are accessible.
NFR-04	**Reliability**	The system should handle invalid inputs or server errors gracefully without crashes.	Perform stress tests and check that error messages appear instead of system failures.
NFR-05	**Maintainability**	The code should follow a modular MVC structure with clear documentation and comments for future updates.	Review the codebase to ensure MVC separation and adequate inline documentation.

3.5 External Interface Requirements

This section describes the external interfaces that connect the system with users, hardware, software, and communication components.

3.5.1 User Interface

The system will provide a simple web-based user interface accessible through standard web browsers.
The interface will be developed using **HTML and basic CSS** for structure and presentation.
Main screens include:

a) **Home/Search Page:** Allows patients to search for doctors by name, specialty, or location.

b) **Doctor Profile Page:** Displays doctor details, clinic information, and available time slots.

c) **Booking Page:** Enables patients to select and confirm appointment slots.

d) **Patient Dashboard:** Shows user's upcoming and past appointments.

e) **Doctor Dashboard:** Allows doctors to manage schedules and bookings.

f) **Admin Dashboard:** Enables administrators to manage users and system data.

3.5.2 Hardware Interface

No special hardware is required.
The system can run on any device with a web browser—desktop, laptop, or smartphone.
The server only needs a normal computer capable of running a **Python Flask** application.

3.5.3 Software Interface

The project uses the following software components:

1) **Backend:** Python Flask

2) **Frontend:** HTML and CSS

3) **Database:** SQLite (for testing) / PostgreSQL (for deployment)

4) **Operating Systems:** Windows, macOS, or Linux

5) **Email Service:** SMTP for sending confirmation messages.

3.5.4 Communication Interface

The website communicates through **HTTP / HTTPS** between users and the server.
Email notifications (for booking confirmations or cancellations) are sent using an **SMTP service**.
All connections will be secure to protect user data.

4.1 Appendix A: Data Dictionary

This section lists all the main data fields used in the MediBook Doctor Booking App. It explains what each thing is, what type of data it stores, and any rules or limits that apply. The goal is to make sure everything stays constant when building, and managing the database.

Data Element	Description	Data Type	Format / Range	Constraints	Source / Destination
user_id	A unique number that identifies every user in the system, whether they're a patient, doctor, or admin.	Integer	Auto-incremented, ≥ 1	Primary Key, Not Null	System-generated
email	The user's email used for logging in and getting messages.	String	Must look like user@domain.com	Unique, Not Null, Max 100 characters	User input → DB
password_hash	The user's password, but stored safely in a hashed form	String	60–255 characters (common limit)	Not Null, must follow password rules	System (hashing) → DB
name	The full name of the user.	String	3–100 characters	Not Null	User input → DB
phone	User's phone number for SMS notifications and login.	String	number format (e.g., +201234567890)	Must be valid and unique for each user type	User input → DB
role	Shows what type of user it is. patient, doctor, admin, or clinic assistant.	Enum	patient, doctor, admin, clinic_assistant	Not Null, Default: patient	System/DB
patient_id	A unique ID for each patient, connected to their user_id.	Integer	≥ 1	Foreign Key → user_id, Not Null	System-generated
dob	The patient's date of birth.	Date	YYYY-MM-DD	Must be before today and after 1900	User input → DB
doctor_id	A unique ID for each doctor, also linked to their user_id.	Integer	≥ 1	Foreign Key → user_id, Not Null	System-generated

specialization	The doctor's medical specialty	String	3–50 characters	Not Null, must be from a set list	Doctor input → DB
bio	Short professional bio for the doctor.	Text	Max 1000 characters	Optional	Doctor input → DB
price	The consultation fee charged by the doctor per visit.	Decimal(10,2)	≥ 0.00	Not Null	Doctor input → DB
clinic_id	Links the doctor to the clinic they work at.	Integer	≥ 1	Foreign Key → clinic.clinic_id, Not Null	Doctor input → DB
clinic_name	The name of the clinic.	String	3–150 characters	Not Null	Clinic input → DB
address	The clinic's full address.	String	Max 200 characters	Not Null	Clinic input → DB
appointment_id	A unique number for each appointment.	Integer	Auto-incremented, ≥ 1	Primary Key, Not Null	System-generated
datetime	The date and time of the appointment.	DateTime	YYYY-MM-DD HH:MM:SS	Must be in the future when booked, in 15-min intervals	System/User → DB
status	The current state of the appointment (like pending, confirmed, or cancelled).	Enum	pending, confirmed, cancelled, completed, no_show	Not Null, Default: pending	System/User → DB
payment_method	How the patient pays. online or at the clinic.	Enum	online, at_clinic	Not Null	User selection → DB
payment_status	Shows if the payment went through or not.	Enum	pending, paid, failed, refunded	Default: pending	Payment Gateway → DB
rating	The score (1–5) the patient gives the doctor after the visit.	Integer	1–5	Optional, only after completed appointment	Patient input → DB
review_text	Optional written feedback from the patient.	Text	Max 500 characters	Optional	Patient input → DB
notification_id	Unique number for each system notification.	Integer	Auto-incremented	Primary Key	System-generated

message	The content of an SMS or email notification.	Text	Max 160 (SMS), 500 (email)	Not Null	System → External Service
read_flag	Shows whether the user has seen the notification or not.	Boolean	0 or 1	Default: 0	System → DB

4.2 Appendix B: Glossary

This glossary gives short and clear meanings for terms used in the MediBook system. It's meant to help anyone reading the document understand what each technical word means.

Term	Definition
Actor	A user or system that interacts with the MediBook app, like a patient, doctor, or admin.
Appointment Slot	A 15-minute time block in a doctor's schedule that can be booked.
Booking	The act of reserving a time slot for a doctor's appointment. Same as "appointment" in user terms.
Cancellation Policy	The rules for when and how patients can cancel their bookings, usually up to 2 hours before the appointment.
Clinic	A place (like a private practice or hospital) where doctors see patients.
Conceptual Class Diagram	A UML diagram that shows the main entities (like Patient, Doctor, Appointment) and how they're connected.
Doctor Dashboard	The part of the web app that doctors use to manage appointments, schedules, and reviews.
External Interface	Any outside service the system connects with, like SMS, email, or payment systems.
Functional Requirement	Something the system must do, for example, "a user should be able to book an appointment."
IEEE Use Case Template	A standard layout for writing detailed use cases, including the actor, steps, and conditions.
Multiplicity	A UML term showing how many instances of one class relate to another (like one-to-many or one to one).
Non-Functional Requirement	System qualities like speed, security, or ease of use, not what it does, but how well it does it.
Patient	A person who uses the app to find and book appointments with doctors.
Payment Gateway	A third-party service (like Paymob or Stripe) that processes online payments.
RESTful API	The backend interface that allows the app to communicate using HTTP methods like GET and POST.
SRS	Stands for Software Requirements Specification. the formal document that describes what the system should do.

Use Case	A detailed description of how a user interacts with the system to reach a goal, like booking an appointment.
<extend>	A UML relationship showing an optional extension of another use case.
<include>	A UML relationship showing a use case that's always part of another.