# Title Page

**SimpleCMS Documentation**

Author:

**CARDOZA, MALBERT C.**

**NOJA, ARIEL A.**

Table of Contents

[Title Page 1](#_Toc204608813)

[Introduction 2](#_Toc204608814)

[Project Overview 2](#_Toc204608815)

[Objectives 3](#_Toc204608816)

[Scope 3](#_Toc204608817)

[System Analysis 4](#_Toc204608818)

[1. Current System Analysis 4](#_Toc204608819)

[Problem Definition 4](#_Toc204608820)

[2. Requirements Analysis 4](#_Toc204608821)

[Functional Requirements 4](#_Toc204608822)

[Non-Functional Requirements 5](#_Toc204608823)

[3. Stakeholder Analysis 6](#_Toc204608824)

[4. Use Case Analysis 6](#_Toc204608825)

[System Design 9](#_Toc204608826)

[System Architecture 9](#_Toc204608827)

[Database Design 12](#_Toc204608828)

[Software Development Life Cycle 15](#_Toc204608829)

[1. Analyze & Plan 15](#_Toc204608830)

[2. Design 16](#_Toc204608831)

[3. Build 16](#_Toc204608832)

[4. Test 16](#_Toc204608833)

[5. Review 16](#_Toc204608834)

[6. Launch 16](#_Toc204608835)

[User Interface Design 17](#_Toc204608836)

[Data Flow Diagrams (DFDs) 18](#_Toc204608837)

[Class Diagrams 22](#_Toc204608838)

[Controllers 22](#_Toc204608839)

[Models 24](#_Toc204608840)

[Implementation 25](#_Toc204608841)

[Appendices 26](#_Toc204608842)

[References 27](#_Toc204608843)

# Introduction

## Project Overview

SimpleCMS is a web-based system made to help clinics and healthcare professionals manage patient appointments in a smooth and organized way. It’s built using ASP.NET Core MVC and is designed for both patients and doctors to easily access important features like booking appointments, logging in securely, and viewing schedules.

This system aims to make the healthcare process more convenient by providing a clear platform where patients can find clinics, choose doctors, and book visits, while doctors can manage their schedules and patient information easily.

## Objectives

The main goals of SimpleCMS are:

* To make it easier for patients to find clinics and book appointments online.
* To help doctors manage their appointment schedules efficiently.
* To provide a secure and user-friendly portal for both patients and doctors.
* To create a simple and smooth experience for everyone using the system.

## Scope

SimpleCMS includes the following main features:

* **Homepage Navigation:** Easy-to-use homepage with links to sections like About, Services, Doctors, and a search tool to help patients find clinics and doctors.
* **Appointment Booking:** Patients can search for clinics, view doctor availability, and make or cancel appointments with ease.
* **Patient Portal:** Patients can log in, view their dashboard, book appointments, check profiles, and manage their accounts.
* **Doctor Portal:** Doctors can log in to their own dashboard, view appointment bookings, and manage their schedule.
* **Admin Features:** Secure login portals for managing users and roles, if needed later (can be extended).
* **Responsive Design:** Works well on desktops, tablets, and mobile phones so users can access it from anywhere.

# System Analysis

## 1. Current System Analysis

Right now, most clinics and patients rely on manual methods like walk-ins, text messages, or phone calls to manage appointments. This way of handling bookings can cause confusion, delays, and poor communication between patients and doctors. Without a system in place, it’s hard to manage schedules, monitor appointments, or keep track of patient history.

## Problem Definition

The traditional system brings several problems that affect both patients and healthcare staff:

* **Time-Consuming:** Booking and managing appointments manually takes a lot of time and often causes delays.
* **Lack of Organization:** It’s difficult to keep track of appointments, which can lead to double bookings or missed visits.
* **No Real-Time Access:** There’s no quick way for patients or doctors to see appointment updates or availability.
* **Limited Control for Clinics:** Clinic staff can't easily manage doctors' schedules, patient flow, or appointment stats.
* **Human Error:** Mistakes in writing down or managing appointments manually are common and can affect service quality.
* **Poor Patient Experience:** Patients may become frustrated due to unclear schedules and poor communication.

## 2. Requirements Analysis

## Functional Requirements

1. User Management:
   * Patients and doctors can register and manage their own accounts.
   * Doctors have access to their own portal.
   * Role-based access: Patients and doctors see only what’s relevant to them.
2. Appointment System:
   * Patients can search for clinics, view available doctors, and book appointments.
   * Doctors can view and manage their upcoming appointments.
   * Prevent double bookings and maintain clear schedules.
3. Authentication and Security:
   * Secure login for both patients and doctors.
   * Password reset and registration workflows for patients.
   * Only authorized users can access their assigned portal.
4. Profile Management:
   * Patients can update their personal info and view their bookings.
   * Doctors can view their profiles and appointment records.
5. Search and Clinic Details
   * Patients can search for clinics/doctors based on location and specialty.
   * Display clinic and doctor details, availability, and booking options.
6. Statistics and Monitoring:
   * Show counts of registered clinics, doctors, and appointments.
   * Admin or future enhancements can include reports or performance tracking.

## Non-Functional Requirements

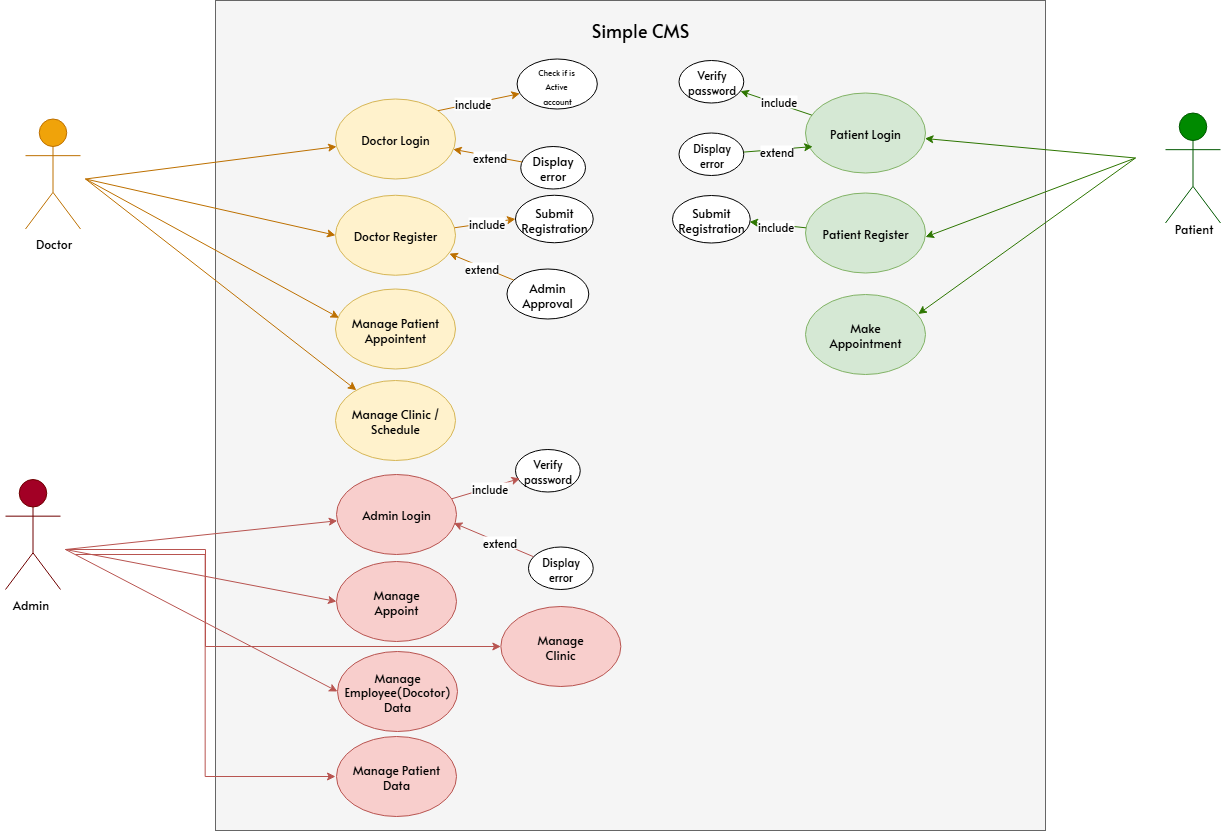
1. Usability:
   * The system should be easy to use for both patients and doctors.
   * Pages and buttons should be clear, and the layout should make sense.
   * The application should work well on phones, tablets, and desktops.
2. Performance:
   * The system should be able to support multiple users at the same time, such as patients booking appointments and doctors viewing their schedules.
   * Pages should load quickly, and real-time features like appointment updates should respond without delays.
3. Security:
   * User accounts should be protected with secure login (username and password).
   * Patient and doctor information should be kept safe and private.
   * Role-based access should ensure that only patients access patient features, and only doctors access doctor tools.
4. Reliability:
   * The system should work properly and be available with minimal downtime.
   * Information, such as appointments and user data, should always be correct and consistent.
5. Scalability:
   * As more clinics, doctors, and patients use the system, it should continue to work smoothly without slowdowns.
   * The system should be built in a way that allows for new features and expansions in the future.

## 3. Stakeholder Analysis

* Administrators (Future Role or Add-on Feature): Responsible for managing the system, such as overseeing clinic data, user roles, and performance. They need access to management tools and dashboards.
* Doctors: They use the system to log in, check appointments, manage their schedule, and view patient bookings. They need a reliable and organized interface to do their job efficiently.
* Patients: They are the main users who search for clinics, view doctors, book appointments, and manage their profiles. They expect a smooth, quick, and reliable experience.
* IT Support / Developers: They ensure the system is running smoothly, maintain security, fix bugs, and update the application when needed.

## 4. Use Case Analysis

1. **Use Case: User Login**
   * **Actors:** Patient, Doctor
   * **Description:** The user logs in with their username and password.
   * **Preconditions:** The user must have a registered account.
   * **Postconditions:** The user is successfully logged in and redirected to their dashboard (Patient or Doctor Portal).
2. **Use Case: Book Appointment**
   * **Actors:** Patient
   * **Description:** The patient selects a region, clinic, doctor, and preferred date, then books an appointment.
   * **Preconditions:** The patient is logged in.
   * **Postconditions:** The appointment is saved in the system and appears in the patient’s dashboard.
3. **Use Case: Manage Appointments**
   * **Actors:** Doctor
   * **Description:** The doctor views upcoming appointments and checks patient details.
   * **Preconditions:** The doctor is logged in.
   * **Postconditions:** The doctor can track and manage appointment schedules in real time.
4. **Use Case: Patient Registration**
   * **Actors:** New User (Patient)
   * **Description:** A patient registers by filling out personal and account details.
   * **Preconditions:** The user is not yet registered.
   * **Postconditions:** A new patient account is created and stored in the database.
5. **Use Case: Forgot Password**
   * **Actors:** Patient
   * **Description:** The patient enters their username to reset their password.
   * **Preconditions:** The user must provide a valid username.
   * **Postconditions:** The system verifies the username and allows the user to set a new password.
6. **Use Case: Search for Clinics and Doctors**
   * **Actors:** Visitor or Patient
   * **Description:** The user searches for clinics or doctors using filters and location.
   * **Preconditions:** The search page is accessed.
   * **Postconditions:** A list of matching clinics/doctors is displayed, and users can proceed to book.
7. **Use Case: View Doctor Details and Availability**
   * **Actors:** Patient
   * **Description:** Patients view doctor profiles, schedules, and available appointment slots.
   * **Preconditions:** The user accesses the clinic-doctor detail page.
   * **Postconditions:** The system displays the doctor’s availability and booking option.



*Figure 1: Use Case Diagram*

# System Design

## System Architecture

* + High-Level Architecture

SimpleCMS follows a **three-tier architecture**, designed to be clean, scalable, and easy to maintain:

1. Presentation Layer (Frontend):

* This is the part users interact with — the web pages.
* It includes forms, navigation menus, appointment booking pages, and login portals.
* Technologies Used:
  + HTML, CSS, JavaScript.
  + Bootstrap (for responsive design).
  + Razor Views (from ASP.NET Core MVC).

1. Application Layer (Backend / Business Logic):

* This layer manages how the system behaves when users perform actions (like booking an appointment or logging in).
* It connects the front end to the database, handles user inputs, applies business rules, and manages sessions.
* Technologies Used:
  + ASP.NET Core MVC.
  + C# for controllers and services.
  + Entity Framework Core for handling data access.

1. Data Layer (Database):

* This layer stores all application data such as user accounts, appointments, clinic and doctor details.
* It manages the saving, retrieving, and updating of data.
* Technologies Used:
  + Microsoft SQL Server or MySQL (depending on your choice/config).
  + Entity Framework Core ORM for interacting with the database.
  + Detailed Design

1. Authentication and Authorization:

* Patients and doctors log in using their credentials.
* The system uses role-based access:
* The system uses role-based access:
  + - * **Patients** can only access booking and profile features.
      * **Doctors** can access appointments and schedule management.
* Passwords are securely stored using hashing.

1. Appointment and Profile Management:

* Patients can:
  + - * Search for clinics/doctors based on location and specialty.
      * Book, cancel, or view appointments.
* Doctors can:
  + - * View appointment schedules.
      * Manage appointment statuses.
* Functions for starting, serving, and ending queue sessions.

1. Clinic & Doctor Details Display:

* A detailed page shows:
  + - * Doctor information.
      * Available schedule.
      * A button to book appointments.
* Only patients can proceed to booking from here.

1. Search and Filter System:

* Patients can search for clinics/doctors with filters (location, specialization, etc.).
* Results are displayed in a carousel/list with direct access to book appointments.

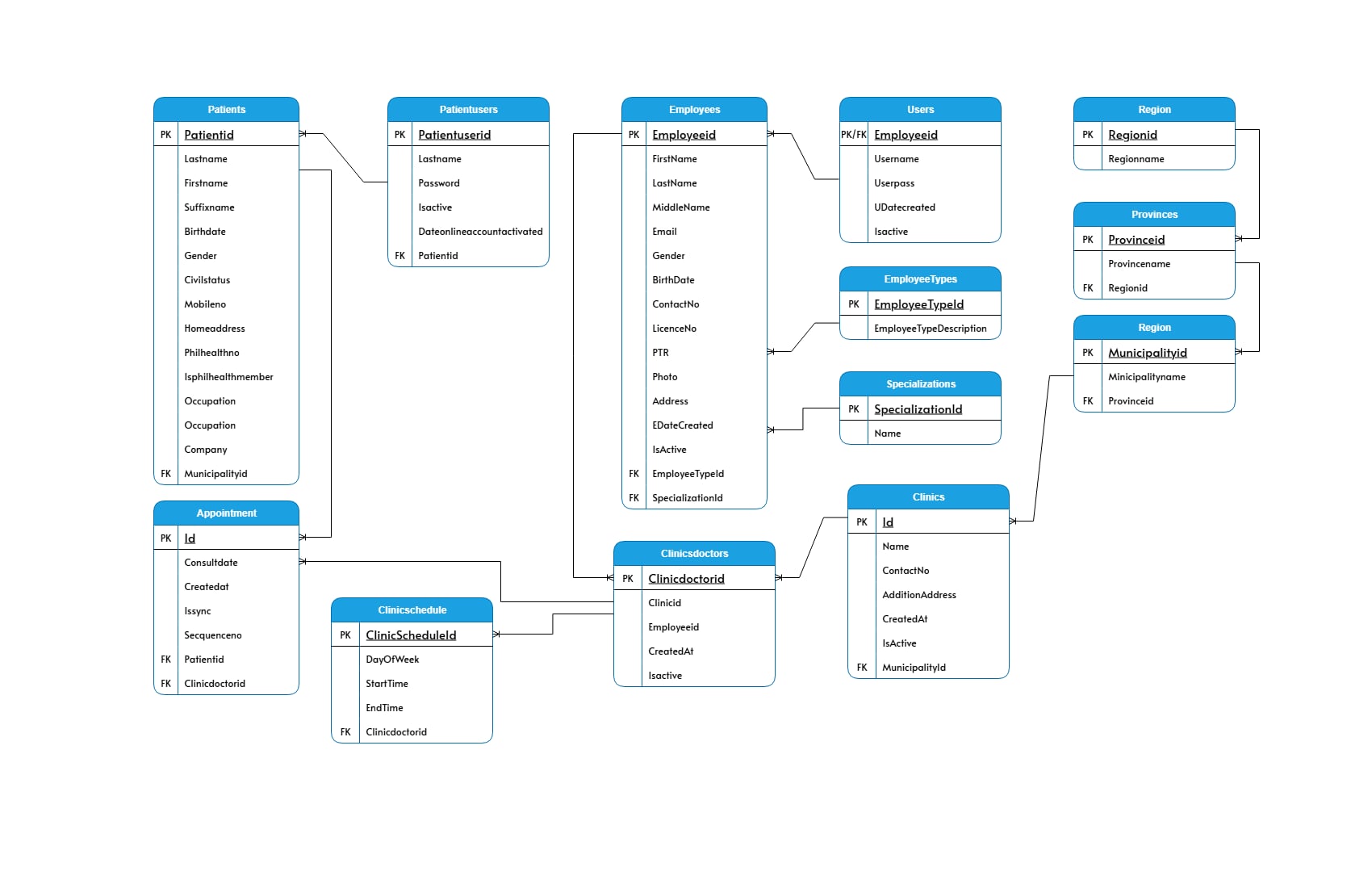
1. Real-Time Stats and Dashboard Info:

* Count of doctors, clinics, and appointments are shown dynamically on the homepage.
* Testimonials, FAQs, and gallery sections are also included for user engagement.

1. Future Reporting Module (Optional Admin Role):

* As the system scales, an admin role could be added to generate:
  + - * Appointment activity reports.
      * Patient statistics.
      * Doctor performance summaries.

## Database Design

* ER Diagrams****

*Figure 2: ER diagram*

* Schema

The application's database consists of several tables related to offices, terminals, queues, and users. Below are the main tables and their structures:

1. Patients Table

* Patientid: Primary key.
* Lastname: Patient’s last name.
* Firstname: Patient’s first name.
* Suffixname: Patient’s suffix (e.g., Jr., Sr.).
* Birthdate: Date of birth.
* Gender: Gender of the patient.
* Civilstatus: Marital status.
* Mobileno: Contact number.
* Homeaddress: Permanent address.
* Philhealthno: PhilHealth number (if applicable).
* Isphilhealthmember: Indicates if a member of PhilHealth.
* Occupation: Job title or role.
* Company: Name of the employer.
* Municipalityid: Foreign key to Municipality table.

1. Patientusers Table

* Patientuserid: Primary key.
* Lastname: Patient user’s last name.
* Password: Encrypted password.
* Isactive: Indicates if the account is active.
* Dateonlineaccountactivated: Date when online account was activated.
* Patientid: Foreign key linking to Patients.

1. Employees Table

* Employeeid: Primary key.
* Firstname: First name.
* Lastname: Last name.
* Middlename: Middle name.
* Email: Email address.
* Gender: Gender of the employee.
* BirthDate: Date of birth.
* ContactNo: Mobile number.
* LicenceNo: Medical or professional license number.
* PTR: Professional Tax Receipt number.
* Photo: Profile picture path.
* Address: Address of the employee.
* EDateCreated: Date employee record was created.
* IsActive: Indicates if the employee is currently active.
* EmployeeTypeId: Foreign key to EmployeeTypes.
* SpecializationId: Foreign key to Specializations.

1. **Users Table**

* Employeeid: Primary and foreign key linking to Employees.
* Username: Login username.
* Userpass: Login password.
* UDatecreated: Date user account was created.
* Isactive: Indicates if the user account is active.

1. EmployeeTypes Table

* EmployeeTypeId: Primary key.
* EmployeeTypeDescription: Description of the type (e.g., doctor, nurse, etc.).

1. Specializations Table

* SpecializationId: Primary key.
* Name: Specialization name (e.g., Cardiology, Pediatrics).

1. Clinics Table

* Id: Primary key.
* Name: Name of the clinic.
* ContactNo: Clinic's contact number.
* AdditionAddress: Additional address info.
* CreatedAt: Date the clinic record was created.
* IsActive: Indicates if the clinic is active.
* Municipalityid: Foreign key linking to Municipality.

1. ClinicDoctors Table

* Clinicdoctorid: Primary key.
* Clinicid: Foreign key to Clinics.
* Employeeid: Foreign key to Employees.
* CreatedAt: Record creation timestamp.
* Isactive: Indicates if the assignment is active.

1. ClinicSchedule Table

* ClinicScheduleId: Primary key.
* DayOfWeek: Day the schedule applies to.
* StartTime: Time the shift starts.
* EndTime: Time the shift ends.
* Clinicdoctorid: Foreign key to ClinicDoctors.

1. Appointment Table

* Id: Primary key.
* Consultdate: Scheduled date of consultation.
* Createdat: Date the appointment record was created.
* Issync: Indicates if data is synced with another system.
* Sequenceno: Number/order in the appointment queue.
* Patientid: Foreign key to Patients.
* Clinicdoctorid: Foreign key to ClinicDoctors.

1. Region Table

* Regionid: Primary key.
* Regionname: Name of the region.

1. Provinces Table

* Provinceid: Primary key.
* Provincename: Name of the province.
* Regionid: Foreign key to Region.

1. Municipality Table

* Municipalityid: Primary key.
* Municipalityname: Name of the municipality.
* Provinceid: Foreign key to Provinces.

# Software Development Life Cycle

### 1. **Analyze & Plan**

* **Tasks**:
  + Meet with stakeholders (e.g., clinic staff, doctors, developer team) to gather ideas.
  + Conduct simple workshops or meetings to understand user needs and pain points.
  + Create a clear vision for the project and list features for the first version (initial sprint backlog).

### 2. **Design**

* **Tasks**:
  + Create wireframes/mockups for key pages like homepage, login pages, dashboard, and appointment form.
  + Design the **database schema**, including tables for:
    - Patients.
    - Doctors.
    - Admins.
    - Regions.
    - Provinces.
    - Municipalities.
    - Clinics.
    - Appointment.
  + Plan how future features (e.g., AI-based doctor recommendations) can be integrated.

### 3. **Build**

* **Tasks**:
  + Implement **authentication** and **role-based access** for patients and doctors.
  + Build user interfaces for:
  + Integrate Bootstrap for styling and modals for CRUD operations.
    - Login, Registration, Forgot Password.
    - Patient dashboard and appointment booking.
    - Doctor dashboard and schedule view.
  + Use **Bootstrap** for styling and layout to make the UI responsive.
  + Add **modals and forms** for data input and CRUD operations (like booking/canceling appointments).

### 4. **Test**

* **Tasks**:
  + Write and run unit tests for key functions like login, booking, and canceling appointments.
  + Perform integration testing to check that the frontend, backend, and database work together.
  + Let the senior and junior developers or testers go through the app for **User Acceptance Testing (UAT)**.

### 5. **Review**

* **Tasks**:
  + Present the system to key users (patients, doctors, project team) and explain features.
  + Collect feedback on usability, missing features, or bugs.
  + Update the backlog and plan next tasks or improvements based on what users say.

### 6. **Launch**

* **Tasks**:
  + Prepare the production environment (e.g., server, domain, database setup).
  + Do a final test run and fix any last-minute issues.
  + Deploy the system live and monitor usage for errors, user feedback, or performance issues.

## User Interface Design

* + Wireframes or Mockups

1. Login Page:

* Input fields for **Username** and **Password.**
* **Login** button to submit credentials.
* Links for:
  + - * **Register** (for new patient users).
      * **Forgot Password** (for password recovery).

2. Patient Dashboard:

* Summary of appointments (upcoming/past).
* Navigation links:
  + - * Book Appointment.
      * Profile.
      * Logout.

3. Doctor Dashboard:

* Appointment list with:
  + - * Patient name.
      * Date.
      * Status (e.g., upcoming, canceled).
      * Action (view, mark as done and cancel).
      * Logout option.
* Profile

4. Appointment Booking Page:

* Dropdowns or filters:
  + - * Region > Province > Municipality > Clinic > Doctor.
* Doctor’s available schedule display.
* **Book Appointment** button.

5. Clinic/Doctor Detail Page:

* Doctor name, specialty, and working days.
* **Book Now** button that redirects to login if not authenticated.

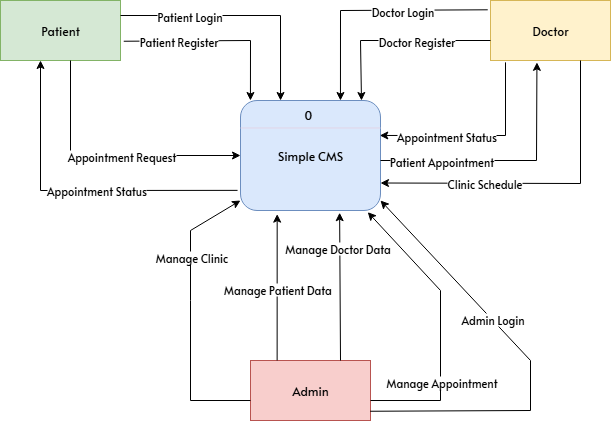
6. Profile Page:

* Read-only or editable form with patient data.
* Button to update profile info.

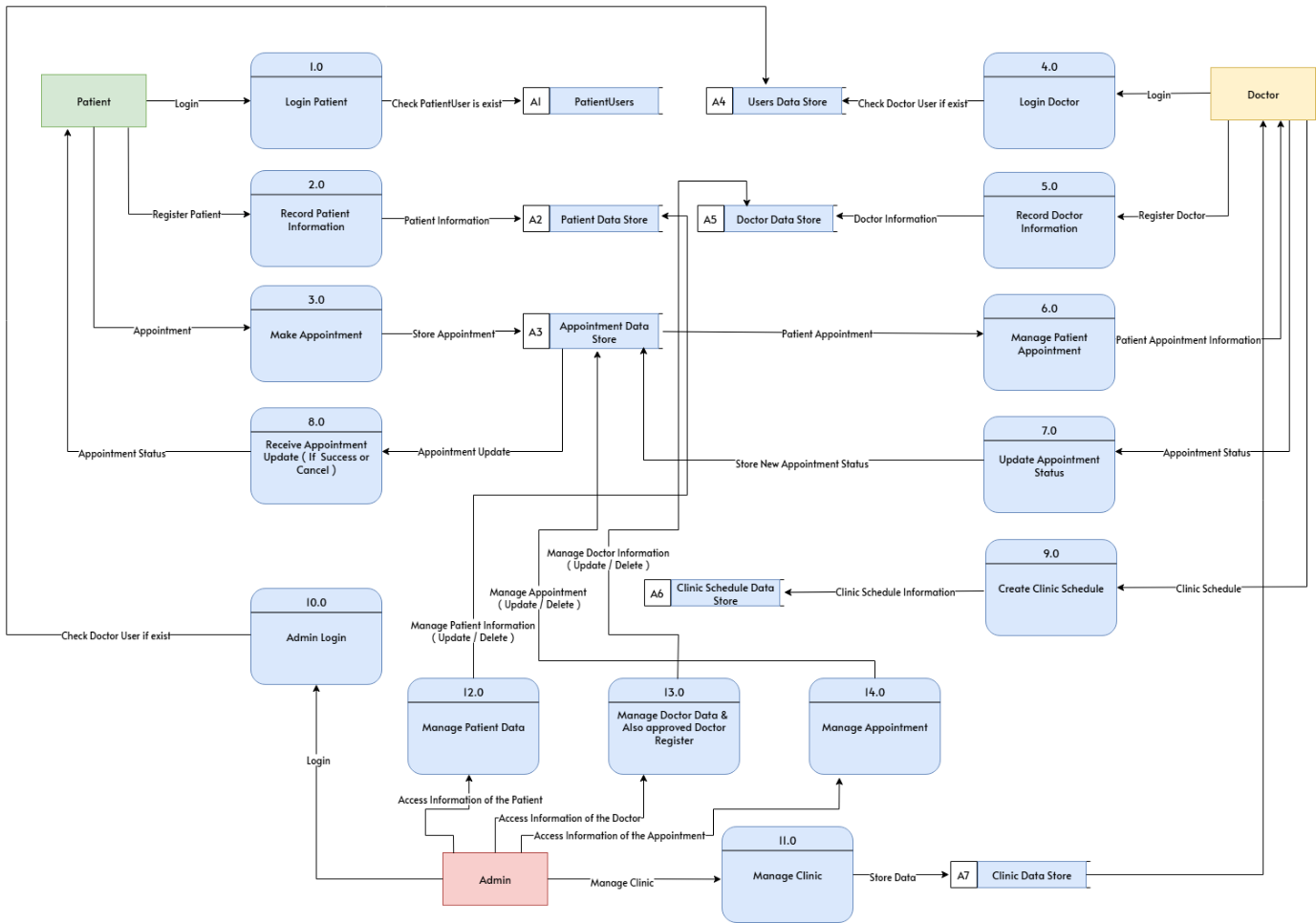
1. Admin Dashboard:

* Form to **Add/Edit/Delete** Doctor, Employee, Appointment, Clinics and Patient.
* Button to update profile info.
* Table showing:
  + - * Names (e.g., Doctor, Employee and Patient).
      * Location.
      * Status.
      * Action (view, mark as done and cancel).
      * 0Logout button.

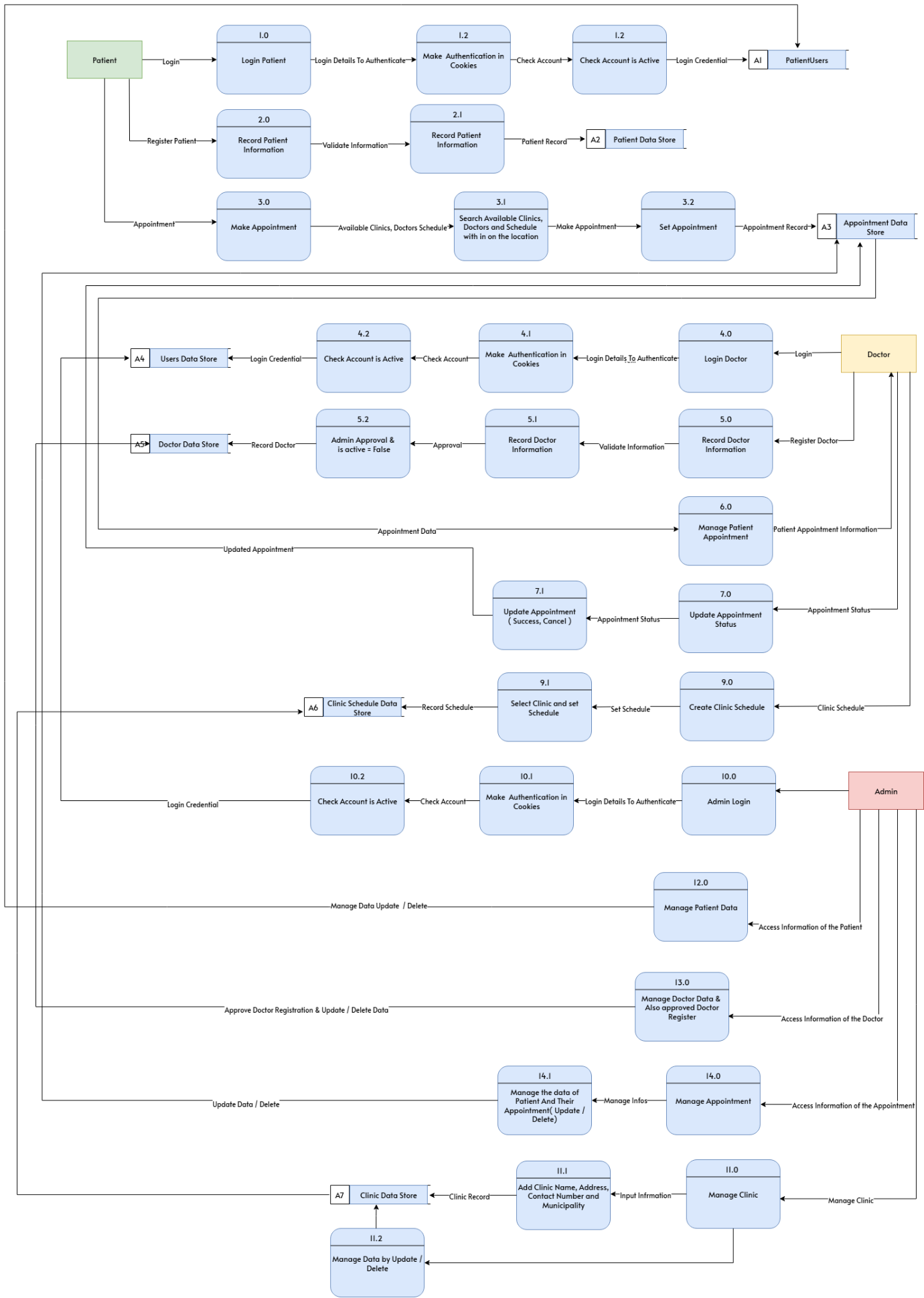
Data Flow Diagrams (DFDs)

****

*Figure 3: Level 0 DFD*

**

*Figure 4: level 1 DFD*

**

*Figure 5: level 2 DFD*

## Class Diagrams

****

*Figure 6: class diagram*

## Controllers

**AppointmentController**

* **GetRegions()**: Returns a list of regions as JSON.
* **GetProvinces(regionId)**: Returns provinces under a selected region.
* **GetMunicipalities(regionId, provinceId)**: Returns municipalities within a selected province.
* **GetClinics(municipalityId)**: Returns clinics located in a given municipality.
* **GetDoctors(clinicId)**: Returns a list of doctors assigned to a specific clinic.
* **Autocomplete(term)**: Performs a search across clinics, municipalities, provinces, and regions for a given input term.

**EmployeeController**

* **EmployeeView(searchString, page)**: Displays paginated list of employees with optional search filtering.
* **Details(id)**: Shows detailed info for a selected employee via partial view.
* **logout():** Logs the user out of the application.
* **DeleteEmployee(employeeId)**: Deletes the specified employee.
* **Edit(id)**: Returns a form for editing an employee's profile.
* **UpdateEmployee(model)**: Saves updates to an employee's record via form post.

**ErrorHandlerController**

* **BadGateWay()**: Displays a 502 Bad Gateway error page.
* **Maintenance()**: Displays a maintenance mode page.

**HomeController**

* **Index()**: Displays homepage with system stats (e.g., total doctors, clinics).
* **Choices()**: Displays a view for public user choices (e.g., patient or admin portal).
* **ClinicDoctorsDetail(clinicId, municipalityId)**: Displays doctors and their schedules for a selected clinic and municipality.
* **Error()**: Returns an error page with diagnostic info (used for unhandled exceptions).

**PatientAccountController**

* **Register()**: Handles patient registration form (GET & POST).
* **Login() / Logout()**: Authenticates patients or logs them out.
* **IsPatientLoggedIn()**: Checks if patient session is active.
* **Dashboard()**: Displays patient's dashboard with appointments and account info.
* **BookAppointment(doctorId, scheduleId, appointmentDate)**: Handles appointment creation and validation.
* **GetDoctorSchedules(doctorId)**: Fetches a doctor’s available schedule.
* **GetFullyBookedDates(doctorId)**: Returns dates on which the doctor has no availability.
* **Profile() / UpdateProfile(model)**: Displays and updates patient's profile.
* **ForgotPassword()**: Handles patient password reset.
* **IsUsernameTaken(username)**: Checks if the username is already used during registration.

### Models

**AppointmentModel**

* **Autocomplete(string term)**:Retrieves all clinic data assigned to the user.

**ClinicModel**

* **GetClinics(int municipalityId):** Retrieves all clinic data based on the specified municipality ID.

**ClinicDoctorModel**

* **GetDoctors(int clinicId)**: Retrieves all doctors based on the specified clinic ID.

**PatientUserModel**

* **IsUsernameAvailable(string username)**: Checks if the specified username is available by querying existing username data.

# Implementation

**1. Technologies Used**

* **Programming Language**: C#
* **Framework**: asp.net core 8 mvc
* **Database**: SQL
* **Frontend**: HTML, CSS (Bootstrap), JavaScript (jQuery)
* **Version Control**: Git
* **Web Server**: Kestrel / IIS
* **IDE**: Visual Studio

**2. Modules and Components**

**User Management Module**

* **Patient Registration:** Patients can sign up through a registration form.
* **User Authentication:** Login system for both patients and doctors using secure credentials.
* **Role-Based Access:** Controls what patients and doctors can access in the system.

**Appointment Management Module**

* **Book Appointment:** Patients can select a clinic, doctor, and time slot.
* **View Appointments:** Patients and doctors can view upcoming appointments.
* **Cancel or Reschedule:** Patients have the option to cancel if needed.

**Clinic & Doctor Management Module**

* **Doctor Profiles:** Displays information such as name, specialty, and schedule.
* **Clinic Info Display:** Shows available clinics and their doctors for patient selection.
* **Availability Display:** Shows which days and times a doctor is available.

**Search & Navigation Module**

* **Clinic Search:** Patients can search by region, municipality, or doctor name.
* **Filters:** Location-based filtering helps patients narrow down options.
* **Doctor Detail Pages:** Display key information and a booking button.

**Dashboard Module**

* **Patient Dashboard:** Shows booked appointments and basic profile information.
* **Doctor Dashboard:** Displays a list of upcoming appointments and patient names.

**Forgot Password & Profile Module**

* **Password Reset Flow:** Verifies username and allows password changes securely.
* **Profile Management:** Patients can view and update their personal information.

# Appendices

#### 1. Glossary

* **Admin (Future Role):** A user with full access to the system, potentially responsible for managing users, clinics, appointments, and settings (may be added in future versions).
* **Patient:** A registered user who can search for clinics and doctors, book appointments, and manage their own profile.
* **Doctor:** A medical professional who can log in, view scheduled appointments, and manage their availability.
* **Appointment:** A scheduled visit between a patient and a doctor, including date, time, and clinic location.
* **Clinic:** A healthcare facility listed in the system where doctors are available for appointments.
* **Dashboard:** A personalized page shown after login, where patients or doctors can view relevant summaries (e.g., upcoming appointments, profile info).
* **MVC (Model-View-Controller):** A software design pattern that separates application logic (Model), user interface (View), and user interactions (Controller), used in ASP.NET Core MVC.
* **Bootstrap:** A front-end framework that helps design clean, responsive, and mobile-friendly user interfaces.
* **Entity Framework Core (EF Core):** An object-relational mapper (ORM) for .NET that simplifies data access and database interactions.
* **jQuery:** A JavaScript library used to simplify HTML DOM manipulation, events, and AJAX requests (used as needed).

# References

* **ASP.NET Core MVC Documentation:**  
  <https://learn.microsoft.com/en-us/aspnet/core/mvc/>
* **Entity Framework Core Documentation:**  
  https://learn.microsoft.com/en-us/ef/core/
* **SQL Server Documentation:**  
  <https://learn.microsoft.com/en-us/sql/sql-server/>
* **Bootstrap 5 Documentation:**  
  <https://getbootstrap.com/docs/5.0/getting-started/introduction/>  
  <https://www.w3schools.com/bootstrap5/>
* **JavaScript Documentation:**  
  https://developer.mozilla.org/en-US/docs/Web/JavaScript  
  <https://www.w3schools.com/jquery/>
* **C# Language Documentation:**  
  <https://learn.microsoft.com/en-us/dotnet/csharp/>
* **Visual Studio (IDE) Documentation:**  
  https://learn.microsoft.com/en-us/visualstudio/