**Course Submission Cover Sheet** 

**Module: CC6012 Data and Web Application**

**Deadline:**

**Module Leader: Student ID:**

PLAGIARISM

You are reminded that there exist regulations concerning plagiarism. Extracts from these regulations are printed below. Please sign below to say that you have read and understand these extracts:

Extracts from University *Regulations on*

Cheating, Plagiarism and Collusion

Section 2.3: "The following broad types of offence can be identified and are provided as indicative examples.

1. Cheating including taking unauthorised material into an examination; consulting unauthorised material outside the examination hall during the examination; obtaining an unseen examination paper in advance of the examination; copying from another examinee; using an unauthorised calculator during the examination or storing unauthorised material in the memory of a programmable calculator, which is taken into the examination; copying coursework.
2. Falsifying data in experimental results.
3. Personation, where a substitute takes an examination or test on behalf of the candidate. Both candidate and substitute may be guilty of an offence under these Regulations.
4. Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.
5. Collusion to present joint work as the work solely of one individual.
6. Plagiarisms, where the work or ideas of another are presented as the candidate's own.
7. Other conduct calculated to secure an advantage on assessment.

(viii) Assisting in any of the above.

Some notes on what this means for students:

1. Copying another student's work is an offence, whether from a copy on paper or from a computer file, and in whatever form the intellectual property being copied takes, including text, mathematical notation and computer programs.

2. Taking extracts from published sources *without attribution* is an offence. To quote ideas, sometimes using extracts, is generally to be encouraged. Quoting ideas is achieved by stating an author's argument and attributing it, perhaps by quoting, immediately in the text, his or her name and year of publication, e.g. " e = mc2 (Einstein 1905)". A *references* section at the end of your work should then list all such references in alphabetical order of authors' surnames. (There are variations on this referencing system, which your tutors may prefer you to use.) If you wish to quote a paragraph or so from published work, then indent the quotation on both left and right margins, using an italic font where practicable, and introduce the quotation with an attribution.

(signature:)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This header sheet should be attached to the work you submit. No work will be accepted without it.

# Introduction

The coursework assignment is an individual assessment weighted 40% of the marks for the module. It is designed mainly to assess students’ practical problem-solving skills and critical thinking/evaluation on the design and development of database systems. It requires the student to analyse, design and implement a web-based database application based on a given business case study. You are asked to provide a software solution as well as appropriate documentation detailing the design and implementation of the system.

# Case Study - Online Healthcare Appointment System

**MediCare Pvt Ltd** is a medium-sized private healthcare provider in Sri Lanka. They manage several clinics and medical centers that offer services in areas such as general medicine, pediatrics, cardiology, and dermatology. Currently, patients book appointments via phone calls or by visiting the clinics physically, which often leads to long waiting times, booking errors, and difficulty in managing patient records.

To improve efficiency and provide better patient care, the management has decided to develop an **Online Healthcare Appointment System**.

## System Requirements

**Home Page**

1. Home page with company (clinic/hospital) information.
2. Sign-in links for **Admin**, **Doctors**, and **Patients**.

**Admin**

* Can sign in.
* Can add/delete **doctor profiles and specialties**.
* Can edit/update **doctor availability, consultation fees, and schedules**.
* Can view all **registered patients and doctors**.
* Can generate reports (appointments, payments, patient statistics).

**Doctor**

* Can sign in.
* Can update their **profile and availability**.
* Can view **upcoming appointments**.
* Can confirm, reschedule, or cancel appointments.
* Can update **patient consultation notes/prescriptions**.

**Patient (Guest/Registered User)**

* Can search for doctors by **specialty, availability, or location**.
* Can register and sign in.
* Can book and reserve an **appointment**.
* Can pay consultation fees online.
* Can view **appointment history and medical records**.
* Can leave feedback or messages for doctors/services.

You have been **sub-contracted by MediCare Pvt Ltd** as a **Junior Database Developer** to design and implement a **prototype of the web-based database system** for the **Online Healthcare Appointment System**.

On completion of the system, you are required to provide a **full set of documentation** of the system to MediCare Pvt Ltd.

Your prototype of the system will be developed using **Oracle SQL Developer (or SQL Server)** for database implementation and **ASP.NET using C#** for the front-end and business logic.

# Requirements of the Coursework

Marks are awarded for producing a working and properly documented system that meets the requirements as specified below:

## Contents Page

A list of sections/subsections of the document, including page numbers.

## Database generation

**Specification of database relations – Using Oracle**

* Relational ER diagram
* Data dictionary (attribute list and constraints for each table)
* Use *SQL Developer Data Modeler* to convert Entity Relationship Diagram into DB tables

**Generation of Database – Using SQL Server/MongoDB**

* Data Models of the CW tables and generated table structures.
* Input data into DB tables (INSERT)
* Display data in the tables (SELECT)

## Implementation of the Online Healthcare Appointment System

Implementation of a Online Healthcare Appointment System, which includes the following using MVC Technology: ASP.NET using C#:

## Documentation of the developed system

**For each Requirement:**

**Implementation document**

* Provide a set of screen dumps for all the functionalities you have produced.

**Testing Document**

* Provide a copy of the initial data (table contents) in your system.
* For each requirement, list the individual tests that have been carried out together with their results.

**FOR THE APPLICATION**

MVC application should be connected with SQL Server/Mongo database system

**User Manual (up to 3 pages)**

* The User Manual should have a contents page and separate sections for each form provided
* The User Manual should contain clear instructions on how to use the system and how to run each of the forms available to it.

## Further Discussion

Your discussion should summarise your experience in undertaking this coursework.

**Your work must be submitted in a single word document, with all output including diagrams, tables, classes, forms, SQL scripts clearly labelled and presented.**

# Introduction

* 1. Background

In today’s digital healthcare environment, many hospitals and clinics still rely on manual processes to manage patient appointments, resulting in scheduling conflicts, long waiting times, and inefficient record-keeping. To solve these problems, the Online Healthcare Appointment System was developed to streamline communication between patients, doctors, and administrators through a centralized web platform.

* 1. Project Purpose

The goal of this system is to allow users to easily book, amend, or cancel appointments online, while doctors and administrators can efficiently handle schedules, payments, and patient data. The solution lowers human error, increases data security, and enhances the entire patient experience.

1.3 System Overview

This project follows the Model-View-Controller (MVC) architecture using **ASP.NET MVC (C#)**, **SQL Server** as the relational database, and **Entity Framework** for ORM. Bootstrap is used for responsive design, ensuring accessibility across devices. The system includes three main roles:

* **Admin:** Manages doctors, appointments, payments, and reports.
* **Doctor:** Reviews patient details and manages appointment schedules.
* **Patient:** Registers, logs in, books appointments, and makes online payments.

1.4 Project Objectives

The primary goals of the system are to:

* Create a safe, user-friendly healthcare appointment platform.
* Reduce administrative workload and scheduling conflicts.
* Provide quick access to doctors' availability and patient records.
* Create summary reports for management and statistical analysis.

1.5 GitHub Repository

The complete source code and project documentation can be accessed at:

https://github.com/Moses2004/Online-Healthcare-Appointment-System

# Database Generation

Add ERD, relation specs, and data dictionary here. Export diagram images and paste them below with captions.

**ERD Diagram**

This is the conceptual view of the database. It focuses on how the entities relate to one another rather than the exact column types.

* Admin manages doctors in the system.
* Doctor belongs to a specific Specialization and has details such as name, email, phone, consultation fee, and availability.
* Patient can register, book appointments, and give feedback.
* Appointment acts as the core link between patients and doctors, recording the date, status, and notes.
* Payment is generated after an appointment and stores payment amount, date, method, and status.
* Prescription is created by the doctor for each appointment, containing doctor notes and medicine details.
* Feeback is written by patients for doctors, including comments and ratings.

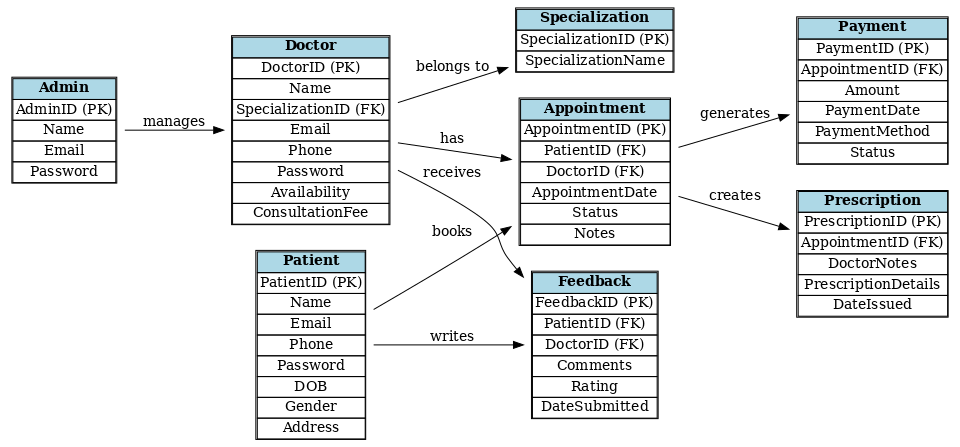
****

Figure: ERD Diagram

A diagram of a computer

AI-generated content may be incorrect.

Figure: ER Diagram

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Engineer to Relational Model

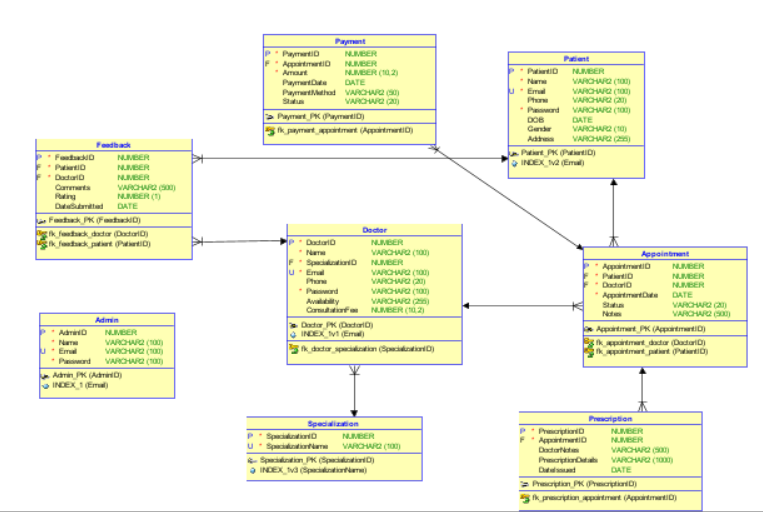


Figure: Relational Model

A computer screen shot of a computer

AI-generated content may be incorrect.

Figure: Database Diagram in Microsoft SQL Server Management Studio

**Data Dictionary**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Table | Column | |  | | --- | |  |  |  | | --- | | Data Type | | Constraints / Description |
| Admin | AdminId | Number | Primary Key, Auto Increment |
|  | Name | |  | | --- | |  |  |  | | --- | | VARCHAR2(100) | | |  | | --- | |  |  |  | | --- | | Not Null | |
|  | Email | VARCHAR2(100) | Unique, Not Null |
|  | Password | VARCHAR2(100) | Not Null |
| Specialization | SpecializationID | NUMBER | Primary Key, Auto Increment |
|  | SpecializationName | VARCHAR2(100) | |  | | --- | |  |  |  | | --- | | Unique, Not Null | |
| Doctor | DoctorID | NUMBER | Primary Key, Auto Increment |
|  | Name | VARCHAR2(100) | Not Null |
|  | SpecializationID | NUMBER | FK → Specialization.Specialization |
|  | Email | VARCHAR2(100) | Unique, Not Null |
|  | Phone | VARCHAR2(20) | Nullable |
|  | Password | VARCHAR2(100) | Not Null |
|  | |  | | --- | |  |  |  | | --- | | Availability | | |  | | --- | | VARCHAR2(255) |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | Nullable | |
|  | ConsultationFee | NUMBER(10,2) | Nullable |
| Patient | PatientID | NUMBER | Primary Key, Auto Increment |
|  | Name | VARCHAR2(100) | Not Null |
|  | Email | VARCHAR2(100) | Unique, Not Null |
|  | Phone | VARCHAR2(100) | Nullable |
|  | Password | VARCHAR2(100) | |  | | --- | |  |  |  | | --- | | Not Null | |
|  | DOB | DATE | Nullable |
|  | Gender | |  | | --- | |  |  |  | | --- | | VARCHAR2(10) | | Nullable |
|  | Address | VARCHAR2(255) | Nullable |
| Appointment | AppointmentID | NUMBER | Primary Key, Auto Increme |
|  | PatientID | NUMBER | FK → Patient.PatientID |
|  | DoctorID | NUMBER | FK → Doctor.DoctorID |
|  | AppointmentDate | DATE | |  | | --- | |  |  |  | | --- | | Not Null | |
|  | Status | VARCHAR2(20) | |  | | --- | |  |  |  | | --- | |  | |
|  | Notes | |  | | --- | |  |  |  | | --- | | VARCHAR2(500) | | |  | | --- | |  |  |  | | --- | | Nullable | |
| Payment | PaymentID | NUMBER | Primary Key, Auto Increment |
|  | AppointmentID | NUMBER | FK → Appointment.AppointmentID |
|  | Amount | NUMBER(10,2) | |  | | --- | |  |  |  | | --- | | Not Null | |
|  | PaymentDate | DATE | Default SYSDATE |
|  | PaymentMethod | VARCHAR2(50) | Nullable |
|  | Status | VARCHAR2(20) | Nullable |
| Prescription | PrescriptionID | NUMBER | Primary Key, Auto Incremen |
|  | AppointmentID | NUMBER | FK → Appointment.AppointmentI |
|  | DoctorNotes | |  | | --- | | VARCHAR2(500) |  |  | | --- | |  | | Nullable |
|  | PrescriptionDetails | VARCHAR2(1000) | Nullable |
|  | DateIssued | DATE | |  | | --- | |  |  |  | | --- | | Default SYSDATE | |
| Feedback | FeedbackID | NUMBER | Primary Key, Auto Increment |
|  | PatientID | NUMBER | FK → Patient.PatientID |
|  | DoctorID | NUMBER | FK → Doctor.DoctorID |
|  | Comments | VARCHAR2(500) | Nullable |
|  | Rating | |  | | --- | |  |  |  | | --- | | NUMBER(1) | | Nullable |
|  | |  | | --- | |  |  |  | | --- | | DateSubmitted | | DATE | Default SYSDATE |

**Connecting Microsoft SQL Sever Management studio and SELECT Displays in the database**

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure: Connecting to the database

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Display all data in doctor table

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Display all data in appointment table

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Display all data in Patient table

**Architecture & Tech Stack**

The Online Healthcare Appointment System is created using the ASP.NET MVC architecture, which separates the application into three communicating layers — Model, View, and Controller. This design pattern facilitates the development of more scalable, maintainable, and modular applications. The Models articulate data and business logic, the Controllers handle requests and orchestrate the flow, and the Views provide the users with Razor-based interfaces for information through their interactions. The use of Entity Framework and Identity is to manage database communication and role-based authentication respectively.



**Model Layer (Data and Business Logic)**

* Represents the data structure and logic of the application.
* Includes classes like Doctor, Patient, Appointment, and Payment inside the Models folder.
* Uses Entity Framework for ORM (Object-Relational Mapping), handling database operations through C# objects instead of raw SQL.
* The model defines relationships, validation, and database context (ApplicationDbContext.cs).

**View Layer (User Interface)**

* Contains Razor View (.cshtml) files organized under the Views folder.
* Each folder (e.g., Views/Doctor, Views/Appointment, Views/Patient) corresponds to a controller.
* Responsible for displaying data to users using HTML, Bootstrap, and Razor syntax such as @Model and @Html.DisplayFor.

**Controller Layer (Application Logic)**

* Found in the Controllers folder (e.g., DoctorController.cs, AppointmentController.cs, PaymentController.cs).
* Handles incoming HTTP requests, interacts with Models, and returns Views or JSON responses.
* Example: The DoctorController retrieves doctor records from the database and passes them to the Index view.

**Supporting Components**

* Entity Framework Core: Manages database communication via LINQ queries.
* ASP.NET Identity: Provides authentication and authorization for Admin, Doctor, and Patient roles.
* SQL Server: Used as the primary database for storing relational data.
* Bootstrap & jQuery: Handle front-end layout and dynamic interactivity.
* Dependency Injection (DI): Registers and manages services such as ApplicationDbContext within Startup.cs or Program.cs.

# Controllers

## HomeContoller

The HomeController.cs is the starting point of the website. It controls what users see when they open the system. This controller has three main actions: Index(), Privacy(), and Error(). The Index() method loads the homepage, Privacy() shows the privacy policy, and Error() displays error details using the ErrorViewModel. It helps keep the website organized and ensures users are directed to the correct pages or shown an error message when something goes wrong.

**A computer screen shot of a program

AI-generated content may be incorrect.**

Figure: HomeController.cs

## AppointmentController

The **AppointmentsController** manages all actions related to appointments between doctors and patients. It makes sure that each user can only see or change what they are allowed to. It uses the database context \_context to get or save data.

The **Index()** method shows all appointments.

* If the user is **Admin**, they can see every appointment and search by doctor or patient name.
* If the user is a **Doctor**, they only see their own appointments.
* If the user is a **Patient**, they only see their personal appointments.

The **Details()** method shows full information about one appointment. It checks the user role to make sure only the doctor, the patient, or the admin can view it.

The **Create()** methods let a patient make a new appointment. The system automatically fills in the patient’s information and sets the status to “Pending.” Only approved doctors appear in the dropdown list.

The **Edit()** methods allow doctors or admins to edit appointments, such as changing the date, status, or notes. Patients are not allowed to edit.

The **Delete()** methods let doctors or admins remove appointments from the system. Patients are not allowed to delete.

Finally, the **AppointmentExists()** method checks if an appointment still exists in the database.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure: AppointmentContoller.cs

## DoctorController

The **DoctorsController.cs** manages everything related to doctors in the system. It connects to the database using **ApplicationDbContext** to display, add, edit, or delete doctor information.

* The **Index()** method shows all doctors and allows searching by name.
* The **Details()** method shows full information of one doctor.
* The **Create()** methods let the admin add new doctors with their specialization, availability, and consultation fee.
* The **Edit()** methods are used to update doctor details.
* The **Delete()** methods remove a doctor from the system.
* There are also **Approve()** and **Remove()** methods to approve or delete a doctor easily.

It helps the admin manage doctor accounts and keeps the data organized in the database.

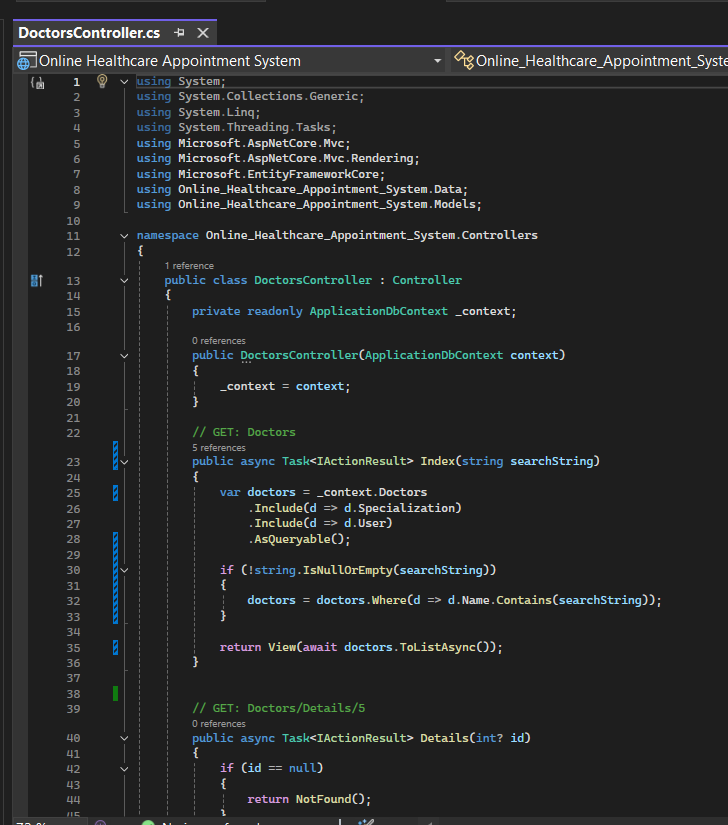


Figure: DoctorController.cs

## PatientController

The **PatientsController.cs** manages all actions related to patients in the system

* The **Index()** method shows a list of all patients, which only admins and doctors can see.
* The **Details()** method shows full details of one patient.
* The **Create()** methods allow the admin to add a new patient with their name, gender, date of birth, and address.
* The **Edit()** methods let the admin update patient information.
* The **Delete()** methods remove a patient from the system.
* There’s also a **ManageProfile()** method that redirects patients to their profile management page.

It helps the admin and doctors manage patient records safely and easily.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure: PatientController.cs

## FeedBackController

The **FeedbacksController.cs** manages all feedback actions in the system. It connects to the database to show, add, edit, or delete patient feedback for doctors.

The **Index()** method displays feedback based on user roles: admins can see all, doctors can see feedback about them, and patients can see only their own.  
The **Create()** methods let patients write and submit feedback for approved doctors.  
The **Edit()** methods allow patients to update their own feedback.  
The **Delete()** methods remove a feedback entry from the system.

It helps manage doctor reviews and ensures each user only sees or edits their allowed data.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure: Feedbackcontroller.cs

# Models

## Appointment Model

The Appointment model is used to store each booking between a patient and a doctor in healthcare system.

* **AppointmentId** → This is the main ID (primary key). It gives each appointment a unique number.
* **PatientId** → This connects the appointment to the patient who made it.
* **DoctorId** → This connects the appointment to the doctor who will meet the patient.
* **AppointmentDate** → This stores the date and time when the appointment happens.
* **Status** → Shows if the appointment is “Pending”, “Confirmed”, “Cancelled”, etc.
* **Notes** → Optional. The doctor or admin can write extra information about the appointment.
* **Patient** and **Doctor** → These are navigation properties. They help the system link the appointment with real patient and doctor details.
* **Payments** → This shows that one appointment can have one or more payments.
* **Prescription** → One appointment can have one prescription

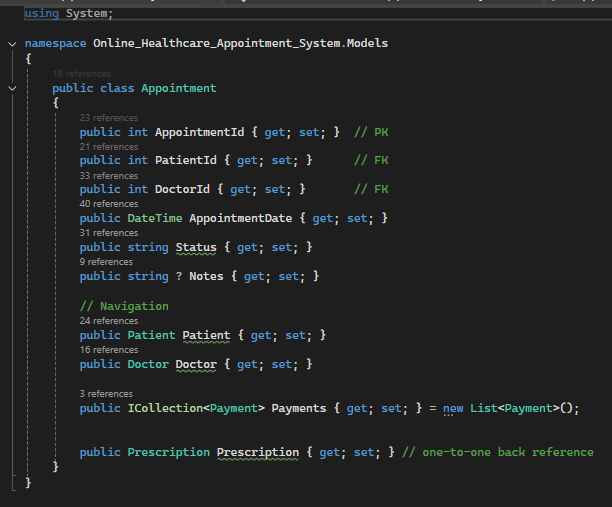


Figure: Appointment Model

## Doctor Model

The Doctor model keeps all the main information about each doctor in the system.

* **DoctorId** → The unique ID for each doctor.
* **Name** → The doctor’s full name.
* **SpecializationId** → Connects the doctor to their medical field .
* **Availability** → Shows if the doctor is available or not.
* **ConsultationFee** → The fee that patients must pay for an appointmen
* **UserId** → Links the doctor’s account to the main login system (**AspNetUsers table**).
* **User** → Connects the doctor with the user profile (email, password, etc.).
* **Specialization** → Connects the doctor to their medical department or skill area.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure: Doctor.cs

## Patient Model

The **Patient** model stores personal information about each patient who uses the system.

* **PatientId** → The unique ID for each patient.
* **Name** → The patient’s full name.
* **Email** → The patient’s email address.
* **Phone** → The patient’s phone number.
* **DOB** → The patient’s date of birth.
* **Gender** → The patient’s gender (Male/Female).
* **Address** → The patient’s home address.
* **UserId** → Links the patient account to the login system (**AspNetUsers** table).
* **User** → Stores login details and connects the patient to their user profile.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure: Patient.cs

## FeedBack Model

The **Feedback** model stores comments and ratings that patients give to doctors after their appointments.

* **FeedbackId** → The unique ID for each feedback.
* **PatientId** → Connects the feedback to the patient who wrote it.
* **DoctorId** → Connects the feedback to the doctor who received it.
* **Comments** → The message or opinion the patient writes.
* **Rating** → A number that shows how satisfied the patient is (for example, 1–5).
* **DateSubmitted** → The date when the feedback was sent.
* **Patient** → Connects the feedback to the patient’s information.
* **Doctor** → Connects the feedback to the doctor’s information.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure: FeeBack.cs

## Payment Model

The Payment model stores details about the money a patient pays for an appointment.

* **PaymentId** → The unique ID for each payment.
* **AppointmentId** → Connects the payment to the specific appointment.
* **Amount** → The total money the patient paid.
* **PaymentDate** → The date when the payment was made.
* **PaymentMethod** → The way the payment was done (like cash, card, or online).
* **Status** → Shows if the payment is “Completed”, “Pending”, or “Failed”.
* **Appointment** → Connects the payment record to its related appointment.

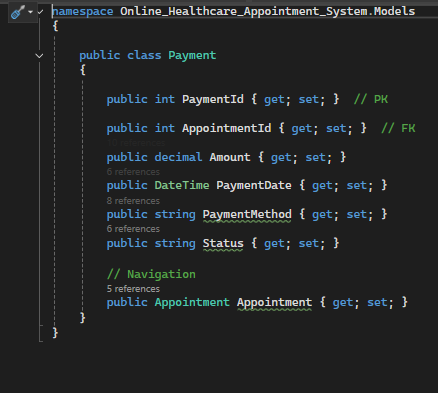


Figure: Payment.cs

# User Manual

## Admin User Guide

### Doctor Account Management Page

There are three roles in the system. They are Admin, Doctor and Patient roles. Each role has their own privileges and restrictions. Admin default login email is **“admin@healthcare.com”** and password is “Admin123!@#”.

A computer screen with text

AI-generated content may be incorrect.

Figure: Creating default adminEmail and Pw

A screenshot of a login form

AI-generated content may be incorrect.

Figure: Login Page

After logging in as an admin, the browser will lead to you to the admin dashboard. In admin dashboard, admin can view general information of the system such as , how many patients, doctors and appointments. An admin can also go to page and can edit, create, delete processes. An can go to the pages by using the tags top bar or buttons in the page.

A screenshot of a computer

AI-generated content may be incorrect.  
Figure: Admin Dashboard

In this dashboard, admin can create new doctors and edit doctors’ information such as Specilization, consulation fee etc. An admin can also delete and edit doctors’ data. An admin can also search a doctor by his name.

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Doctors management dashboard

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Creating a new doctor

A screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a medical form

AI-generated content may be incorrect.

Figure: Edit and Delete Pages

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Doctor Detail Page

### Patients Accounts Management Page

Similar to doctors account management page, an admin can do CRUD processes and find patient accounts by using search bar.

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Patient Account Management

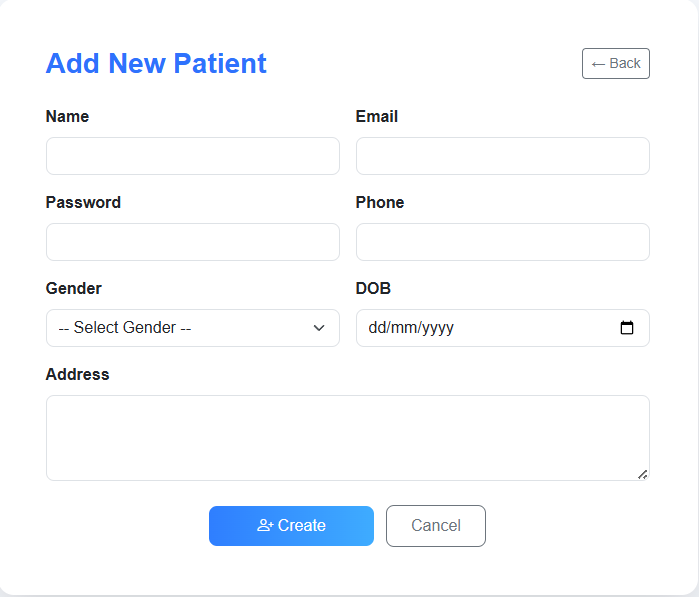


Figure: Create Patient Form

A screenshot of a computer

AI-generated content may be incorrect.

Figure: Patient Edit Form

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a medical survey

AI-generated content may be incorrect.

Figure: Patient Edit and Delete Pages