Course Submission Cover Sheet

Module: CC6012 Data and Web Application

Deadline:

Module Leader: Rattapol Kasemrat Student ID: E256406

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.

(i) **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.**
(ii) **Falsifying data in experimental results.**
(iii) 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.
(iv) **Bribery or attempted bribery of a person thought to have some influence on the candidate's assessment.**
(v) Collusion to present joint work as the work solely of one individual.
(vi) Plagiarisms, where the work or ideas of another are presented as the candidate's own.
(vii) 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. " $e = mc^2$ (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.

Contents

Introduction....4
Database Generation....5
Connecting Microsoft SQL Sever Management studio and SELECT Displays in the database....10
Insert data....11
Display Data....13
Workflow Diagram....14
Architecture & Tech Stack....15
Controllers....17
HomeContoller....17
AppointmentController....18
DoctorController....20
PatientController....21
FeedBackController....22
Models....24
Appointment Model....24
Doctor Model....25
Patient Model....26
FeedBack Model....27
Payment Model....28
User Manual....29
Patient User Guide....29
Patient Dashboard....29
Patient Appointment Dashboard....30
Patient Feedback Dashboard....33

Admin User Guide 34
Doctor Account Management Page 34
Patients Accounts Management Page 37
Payment Management Page 39
Report Dashboard 40
Admin Appointment Dashboard 40
Specializations Dashboard 41
Doctor User Guide 41
Weaknesses and Future Development 43
References 44

Introduction

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.

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.

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.

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.

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.

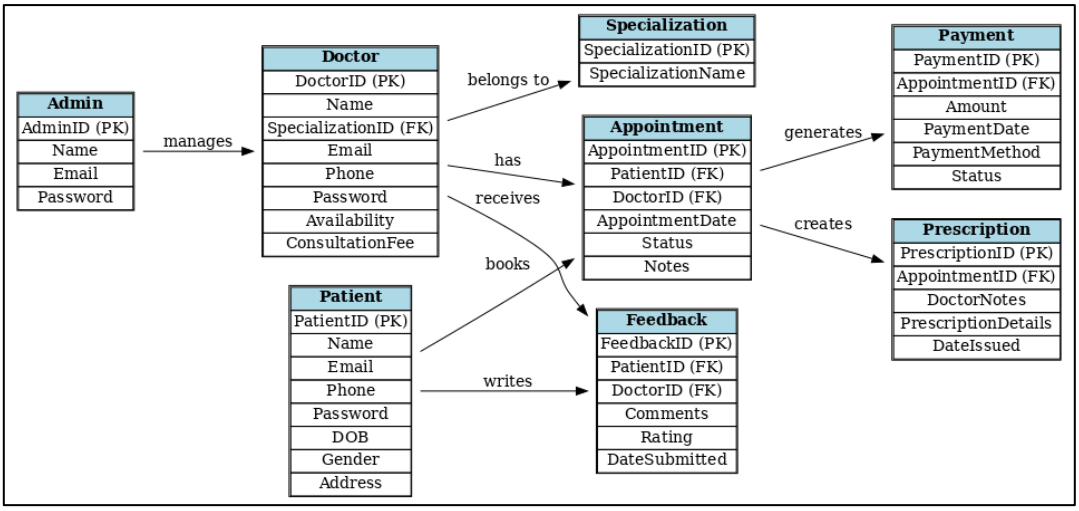


Figure: ER Diagram

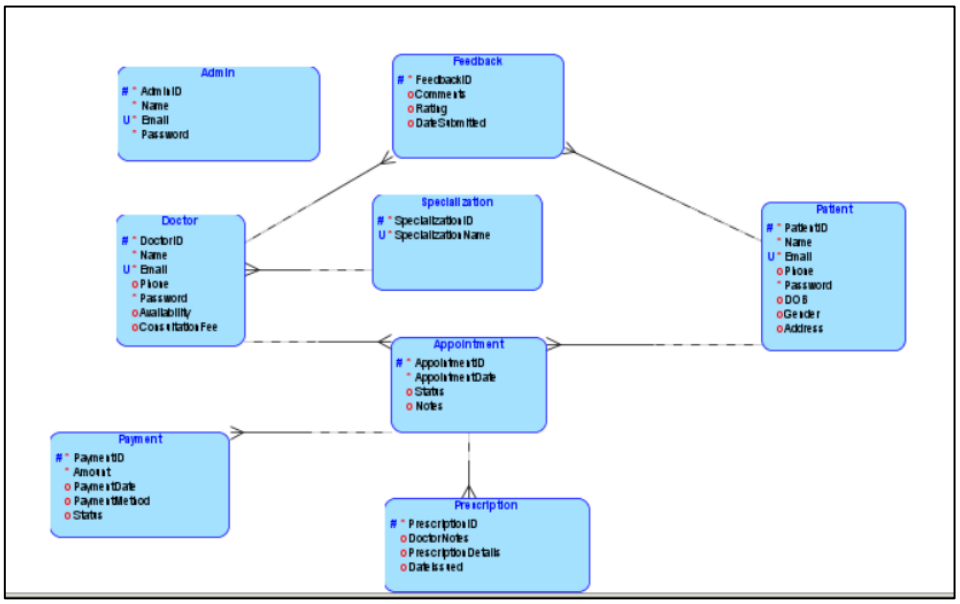


Figure: ER Diagram

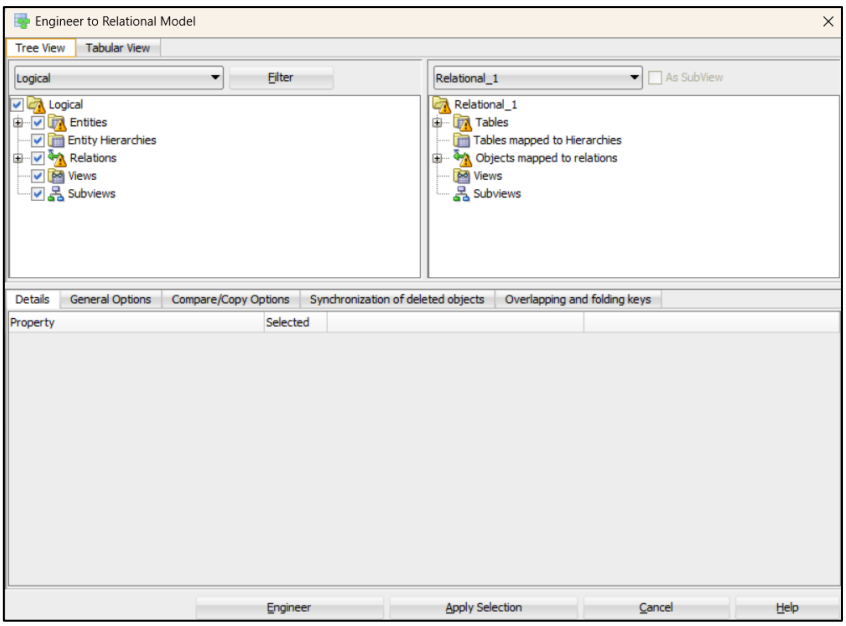


Figure: Engineer to Relational Model

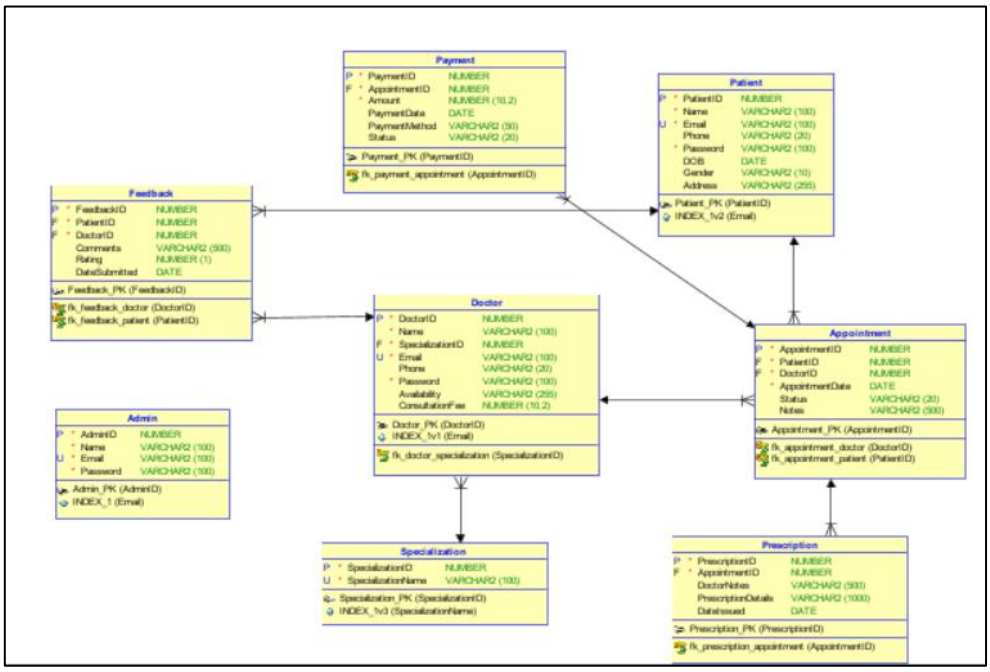


Figure: Converting ERD into DB Tables using SQL Developer Data Modeler

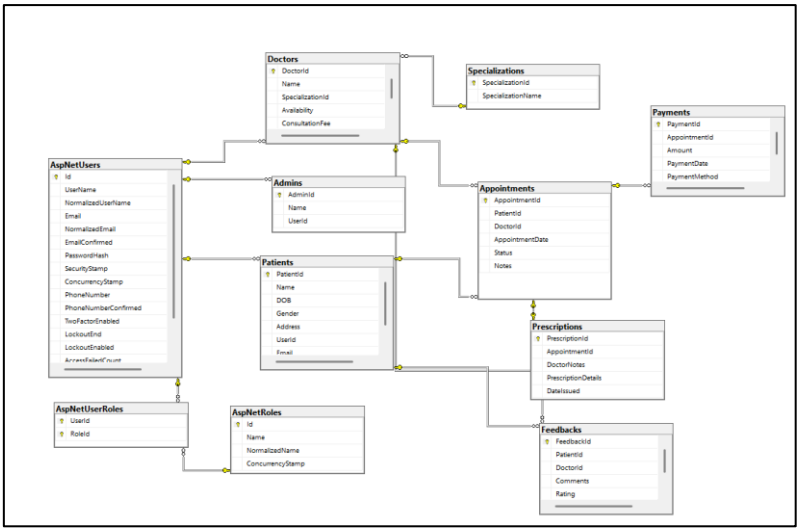


Figure: Database Diagram in Microsoft SQL Server Management Studio

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

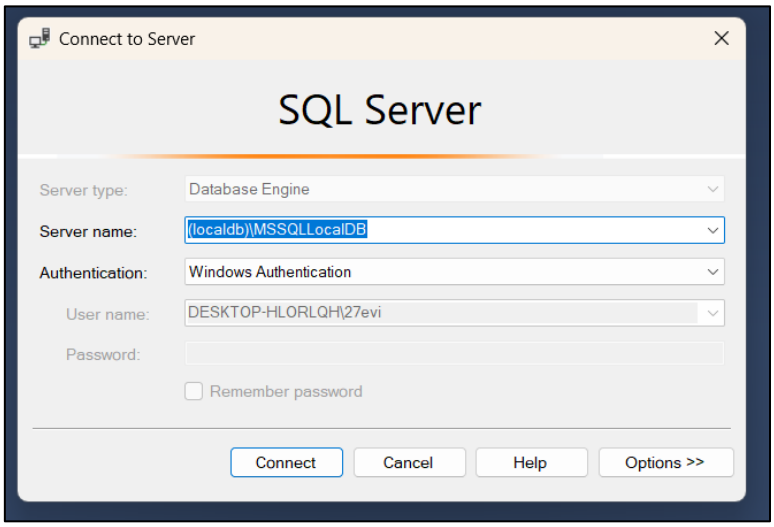


Figure: Connecting to the database

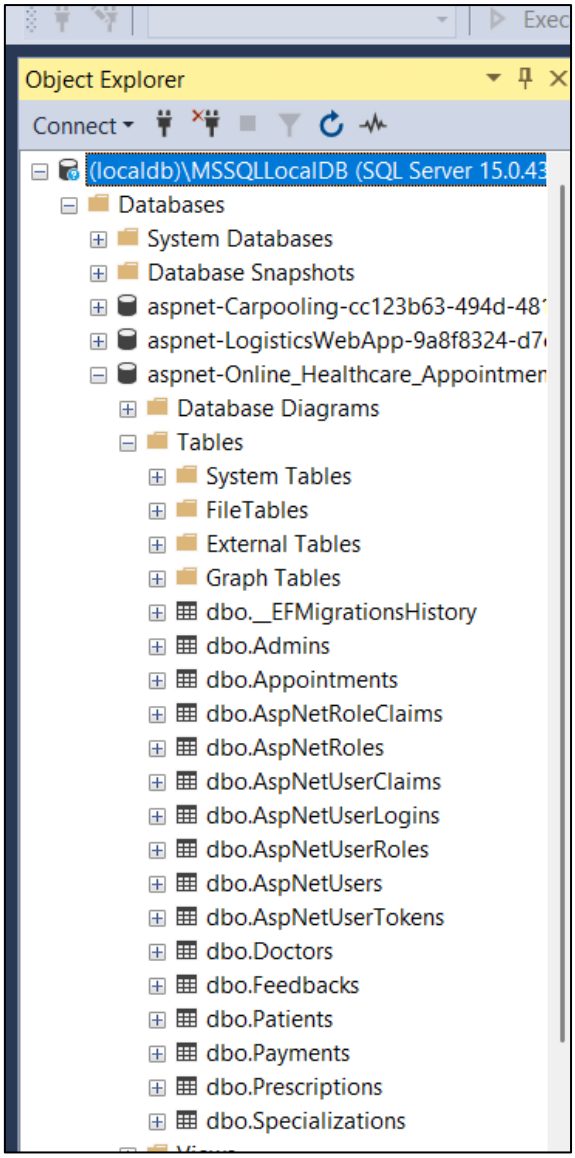


Figure: Database Creating Successful

Insert data

Inserting sample data to the **Feedbacks** table by sql command

“ INSERT INTO Feedbacks (PatientId, DoctorId, Comments, Rating, DateSubmitted)

VALUES (12, 6, 'Excellent service and kind doctor.', 5, GETDATE());”

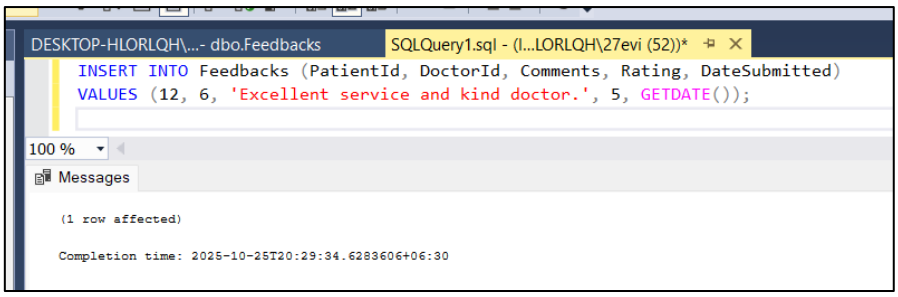
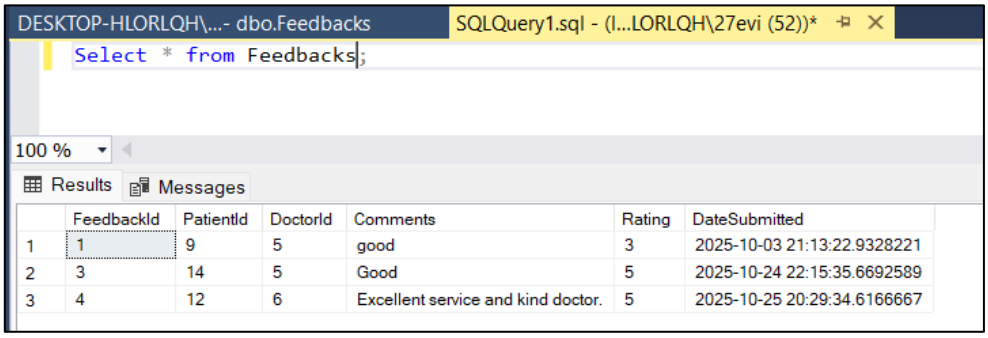


Figure: Inserting Data



	FeedbackId	PatientId	DoctorId	Comments	Rating	DateSubmitted
1	1	9	5	good	3	2025-10-03 21:13:22.9328221
2	3	14	5	Good	5	2025-10-24 22:15:35.6692589
3	4	12	6	Excellent service and kind doctor.	5	2025-10-25 20:29:34.6166667

Figure: Data inserting successful

Inserting sample data into the **"Specializations"** table.

" INSERT INTO Specializations (SpecializationName)

VALUES

('Dermatology'),

('Neurology'),

('Pediatrics');"

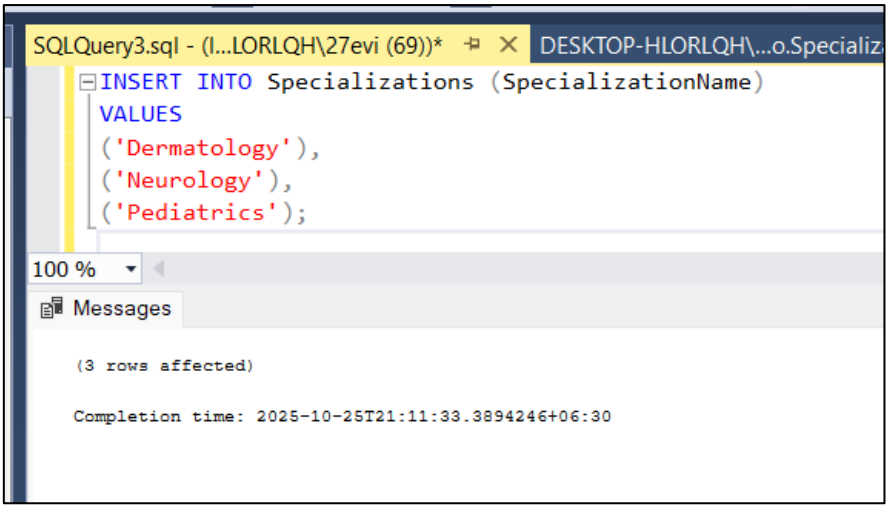


Figure: Data inserting

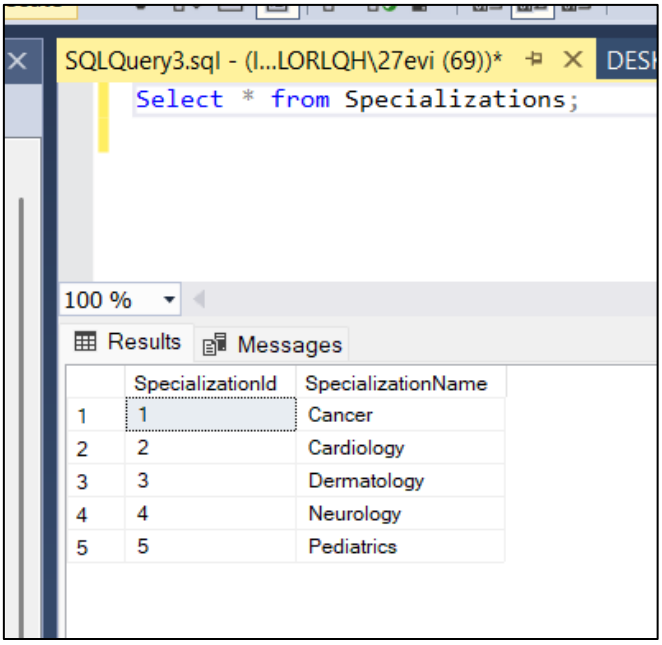


Figure: Inserting Data Successful

Display Data

Display all the data of the **Doctors** table by the following sql command:

“ Select * from Doctors:”

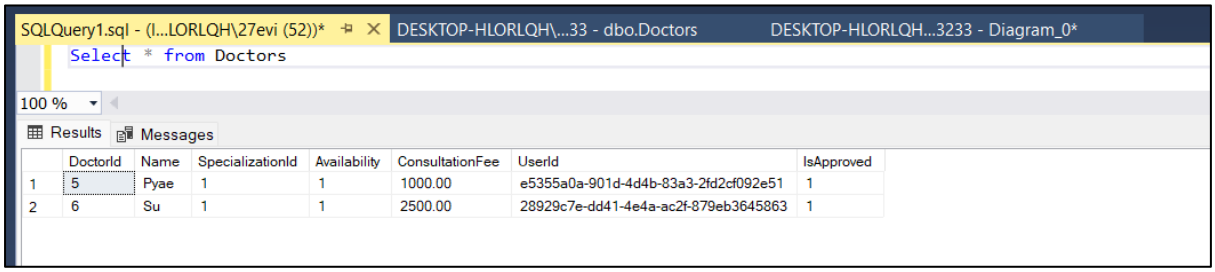


Figure: Display all data in doctor table

Display all the data of the **Appointments** table by the following sql command:

“ Select * from Appointments:”

SQLQuery2.sql - (l...LORLQH\27evi (52))*

```
select * from Appointments
```

100 %

Results | Messages

	AppointmentId	PatientId	DoctorId	AppointmentDate	Status	Notes
1	1	9	5	2025-09-29 14:52:00.0000000	Completed	Fever
2	2	9	5	2025-09-30 15:13:00.0000000	Completed	NULL
3	3	9	5	2025-10-10 21:11:00.0000000	Completed	NULL
4	4	9	6	2025-10-16 19:40:00.0000000	Completed	NULL
5	5	10	6	2025-10-20 15:22:00.0000000	Completed	NULL
6	6	9	5	2025-10-21 22:21:00.0000000	Completed	ill

Figure: Display all data in appointment table

SQLQuery2.sql - (l...LORLQH\27evi (52))*

```
select * from Patients
```

100 %

Results | Messages

	PatientId	Name	DOB	Gender	Address	UserId	Email	Phone
1	8	Moses Lin Thant	0001-01-01 00:00:00.0000000	Other	Rangsit-Pathum Thani Road , Progress Mansion	cfc41866-c1a5-4b5d-9783-b28354846491	Moses@gmail.com	+66636154094
2	9	kyaw	0001-01-01 00:00:00.0000000	Male	Rangsit-Pathum Thani Road , Progress Mansion	c4e66d13-365b-4468-9872-b1eac0190149	kyaw@gmail.com	+66636154094
3	10	Khaing	0001-01-01 00:00:00.0000000	Male	PathumThani,Pruksk	95b9c30f-de7f-4ef9-80ec-8092ac57d7af	Khaing@gmail.com	0981726124

Figure: Display all data in Patient table

Workflow Diagram

The following diagram is created by using draw.io software

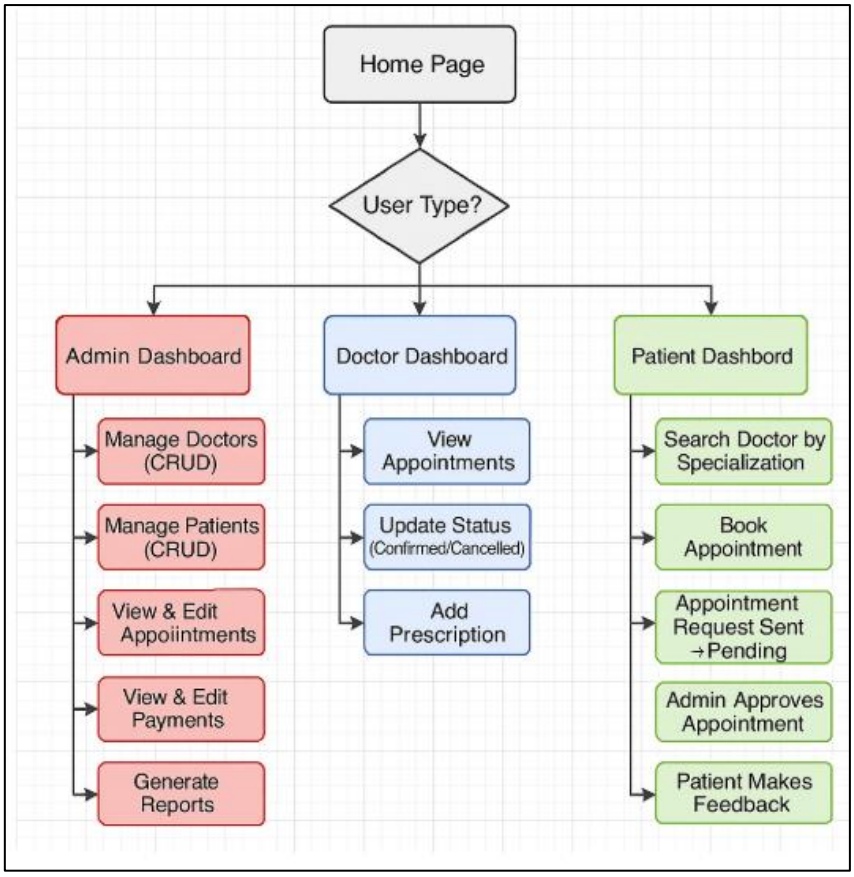
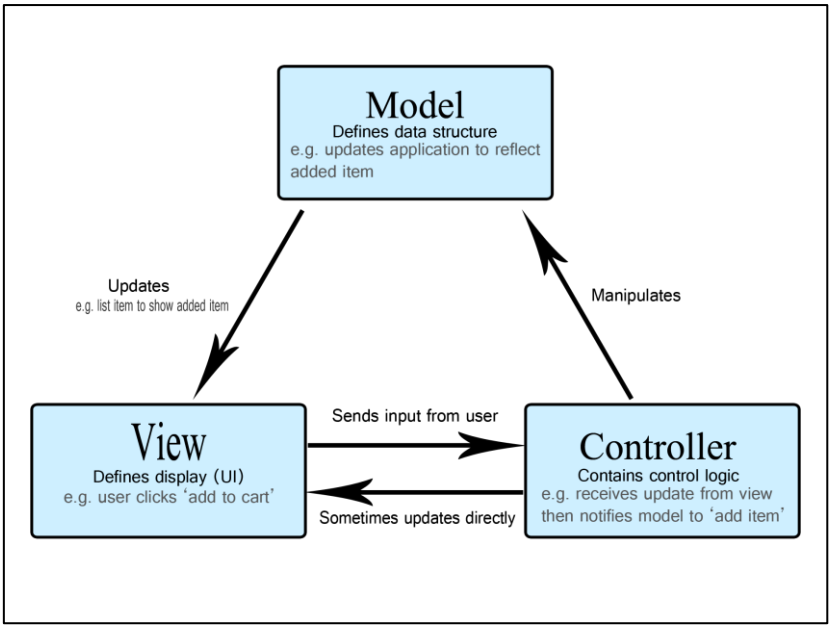


Figure: Workflow Diagram for HealthCare Application System

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.

```
HomeController.cs   Details.cshtml   Edit.cshtml   Create.cshtml   Index.cshtml
Online Healthcare Appointment System   Online_Healthcare_Appointment_System.Controllers.Home   _lo
1   using System.Diagnostics;
2   using Microsoft.AspNetCore.Mvc;
3   using Online_Healthcare_Appointment_System.Models;
4
5   namespace Online_Healthcare_Appointment_System.Controllers
6   {
        3 references
7       public class HomeController : Controller
8       {
9           private readonly ILogger<HomeController> _logger;
10
            0 references
11          public HomeController(ILogger<HomeController> logger)
12          {
13              _logger = logger;
14          }
15
            0 references
16          public IActionResult Index()
17          {
18              return View();
19          }
20
            0 references
21          public IActionResult Privacy()
22          {
23              return View();
24          }
25
26          [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
            0 references
27          public IActionResult Error()
28          {
29              return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
30          }
31      }
32  }
33
```

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.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Online_Healthcare_Appointment_System.Data;
using Online_Healthcare_Appointment_System.Models;
using Microsoft.AspNetCore.Authorization;

namespace Online_Healthcare_Appointment_System.Controllers
{
    [Authorize]

    public class AppointmentsController : Controller
    {
        private readonly ApplicationDbContext _context;


        public AppointmentsController(ApplicationDbContext context)
        {
            _context = context;
        }


        //  INDEX

        2 references
        public async Task<IActionResult> Index(string searchString)
        {
            IQueryable<Appointment> query = _context.Appointments
                .Include(a => a.Doctor)
                    .ThenInclude(d => d.Specialization)
                .Include(a => a.Patient);

            //  Admin can see all appointments
            if (User.IsInRole("Admin"))
            {
                if (!string.IsNullOrEmpty(searchString))
                {
                    string lowerSearch = searchString.ToLower();
                    query = query.Where(a =>
                        a.Doctor.Name.ToLower().Contains(lowerSearch) ||
                        a.Patient.Name.ToLower().Contains(lowerSearch));
                }

                return View(await query.OrderByDescending(a => a.AppointmentDate).ToListAsync());
```

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.

```
DoctorsController.cs
Online Healthcare Appointment System        Online_Healthcare_Appointment_Syste
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Online_Healthcare_Appointment_System.Data;
using Online_Healthcare_Appointment_System.Models;

namespace Online_Healthcare_Appointment_System.Controllers
{
    1 reference
    public class DoctorsController : Controller
    {
        private readonly ApplicationDbContext _context;

        0 references
        public DoctorsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Doctors
        5 references
        public async Task<IActionResult> Index(string searchString)
        {
            var doctors = _context.Doctors
                .Include(d => d.Specialization)
                .Include(d => d.User)
                .AsQueryable();

            if (!string.IsNullOrEmpty(searchString))
            {
                doctors = doctors.Where(d => d.Name.Contains(searchString));
            }

            return View(await doctors.ToListAsync());
        }


        // GET: Doctors/Details/5
        0 references
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
```

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.

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Online_Healthcare_Appointment_System.Data;
using Online_Healthcare_Appointment_System.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;

namespace Online_Healthcare_Appointment_System.Controllers
{
    [Authorize]

    public class PatientsController : Controller
    {
        private readonly ApplicationDbContext _context;


        public PatientsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Patients
        [Authorize(Roles = "Admin,Doctor")]

        public async Task<IActionResult> Index()
        {
            var applicationDbContext = _context.Patients.Include(p => p.User
            return View(await applicationDbContext.ToListAsync());
        }

        // Redirect to Identity Manage Page

        public IActionResult ManageProfile()
        {
            return Redirect("/Identity/Account/Manage");
        }

        // GET: Patients/Details/5
        [Authorize(Roles = "Admin,Doctor")]

        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
```

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.

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Online_Healthcare_Appointment_System.Data;
using Online_Healthcare_Appointment_System.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Online_Healthcare_Appointment_System.Controllers
{

    public class FeedbacksController : Controller
    {
        private readonly ApplicationDbContext _context;

        0 references
        public FeedbacksController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Feedbacks
        [Authorize]
        3 references
        public async Task<IActionResult> Index()
        {
            IQueryable<Feedback> feedbacks = _context.Feedbacks
                .Include(f => f.Doctor)
                .Include(f => f.Patient);

            if (User.IsInRole("Admin"))
            {
                //Admin can see all feedbacks
                return View(await feedbacks
                    .OrderByDescending(f => f.DateSubmitted)
                    .ToListAsync());
            }
            else if (User.IsInRole("Patient"))
            {
                // Patient sees only their own feedbacks
                var userEmail = User.Identity.Name;
                var patient = await _context.Patients
                    .Include(p => p.User)
                    .FirstOrDefaultAsync(p => p.User.Email == userEmail);
```

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

```
using System;

namespace Online_Healthcare_Appointment_System.Models
{
    18 references
    public class Appointment
    {
        23 references
        public int AppointmentId { get; set; }  // PK
        21 references
        public int PatientId { get; set; }      // FK
        33 references
        public int DoctorId { get; set; }       // FK
        40 references
        public DateTime AppointmentDate { get; set; }
        31 references
        public string Status { get; set; }
        9 references
        public string ? Notes { get; set; }

        // Navigation
        24 references
        public Patient Patient { get; set; }
        16 references
        public Doctor Doctor { get; set; }

        3 references
        public ICollection<Payment> Payments { get; set; } = new List<Payment>();


        public Prescription Prescription { get; set; } // one-to-one back reference
    }
}
```

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.

```
namespace Online_Healthcare_Appointment_System.Models
{

    public class Doctor
    {

        public int DoctorId { get; set; }
        public string Name { get; set; }
        15 references
        public int SpecializationId { get; set; }
        17 references
        public bool Availability { get; set; }
        23 references
        public decimal ConsultationFee { get; set; }

        // Identity Link
        14 references
        public string UserId { get; set; }  // FK to AspNetUsers
        24 references
        public ApplicationUser User { get; set; }

        // Navigation
        21 references
        public Specialization Specialization { get; set; }

        12 references
        public bool IsApproved { get; set; } = false;

    }
}
```

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.

```
using System;

namespace Online_Healthcare_Appointment_System.Models
{

    public class Patient
    {

        public int PatientId { get; set; }

        public string Name { get; set; }

        public string Email { get; set; }

        public string Phone { get; set; }

        public DateTime DOB { get; set; }

        public string Gender { get; set; }

        public string Address { get; set; }

        // Identity Link

        public string UserId { get; set; }

        public ApplicationUser User { get; set; }
    }
}
```

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.

```
namespace Online_Healthcare_Appointment_System.Models
{

    public class Feedback
    {

        public int FeedbackId { get; set; }  // PK

        public int PatientId { get; set; }   // FK

        public int DoctorId { get; set; }    // FK

        public string Comments { get; set; }

        public int Rating { get; set; }

        public DateTime DateSubmitted { get; set; }

        // Navigation

        public Patient Patient { get; set; }

        public Doctor Doctor { get; set; }



    }
}
```

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.

```
namespace Online_Healthcare_Appointment_System.Models
{

    public class Payment
    {

        public int PaymentId { get; set; }  // PK

        public int AppointmentId { get; set; }  // FK
        10 references
        public decimal Amount { get; set; }
        6 references
        public DateTime PaymentDate { get; set; }
        8 references
        public string PaymentMethod { get; set; }
        6 references
        public string Status { get; set; }

        // Navigation
        5 references
        public Appointment Appointment { get; set; }
    }
}
```

Figure: Payment.cs

User Manual

Patient User Guide

Patient Dashboard

In patient dashboard, there are five categories. They are appointments, prescription, payments, my profile and feedback.

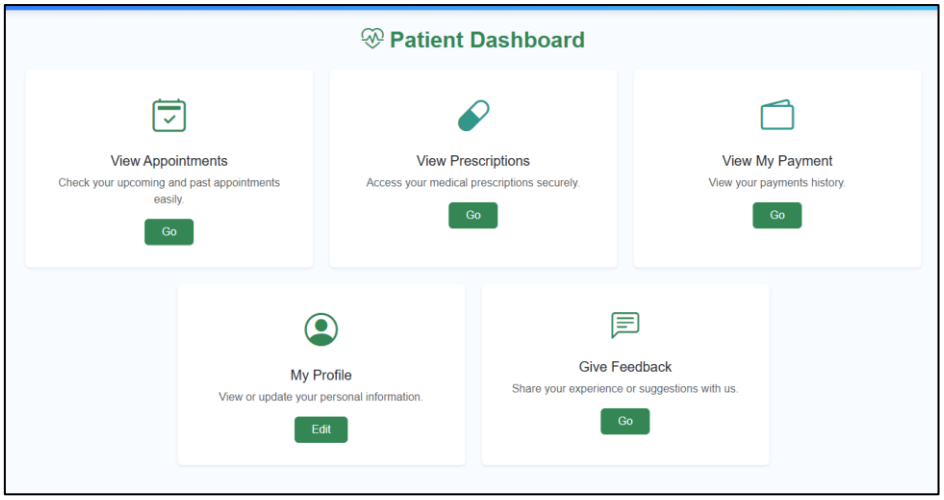
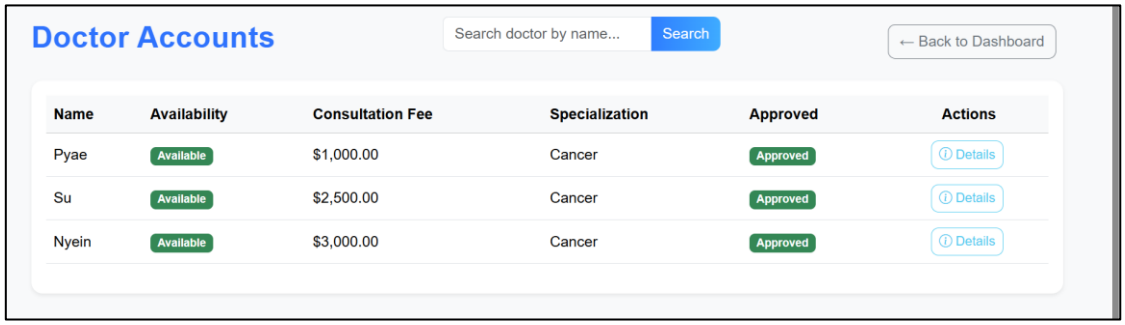


Figure: Patient Dashboard

Patient Appointment Dashboard

When a patient click "Go" in the view appointments box, the browser will lead the patient to the appointment dashboard. He can create new appointment. A patient can choose a doctor and his or her desired date time. A patient can also check all doctors information in doctors page. A patient cannot edit or delete doctors' information. In prescription page, a patient can check all his prescription history. Also the same in payment page.



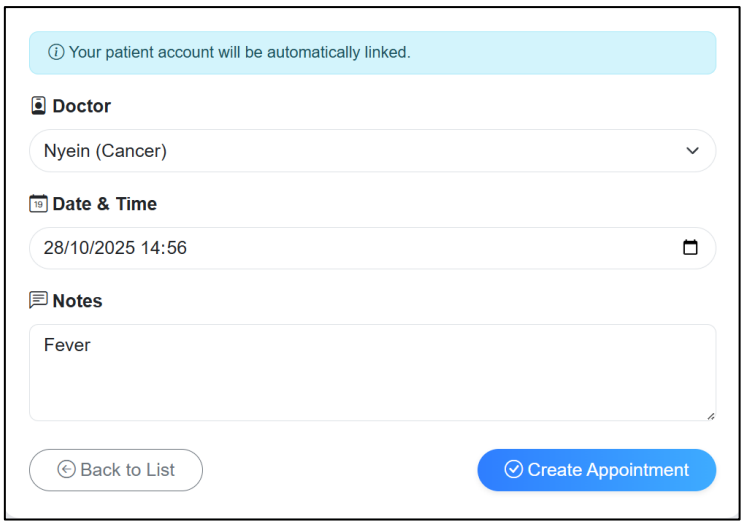


Figure: Creating appointment

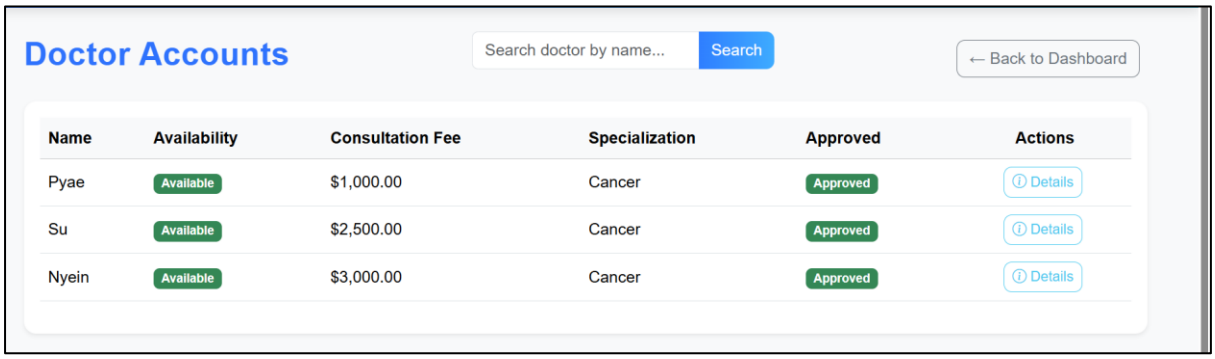


Figure Doctors information page

After a patient created an appointment, an admin have to approve his appointment. After an admin approve the appointment, a doctor can give prescription and the customer can pay. After the customer complete payment, the appointment status change to completed.

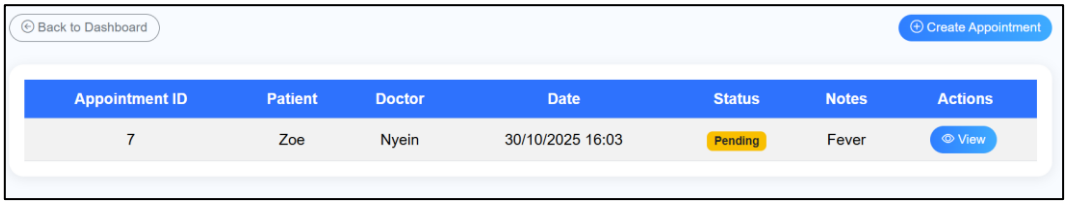


Figure: Create appointment

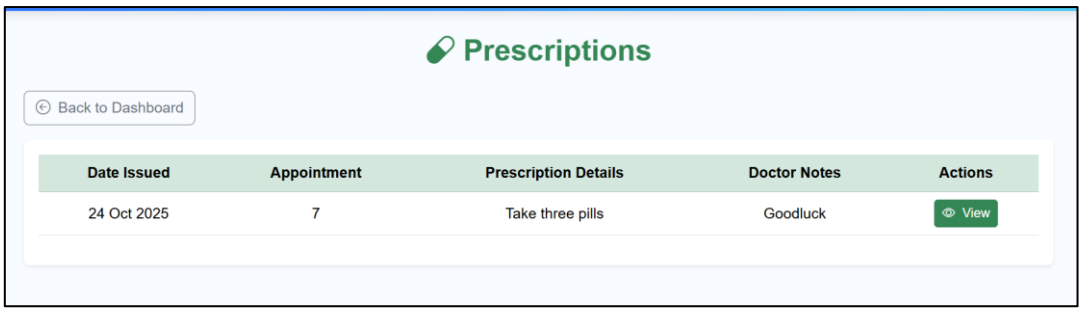


Figure: Patient Prescription

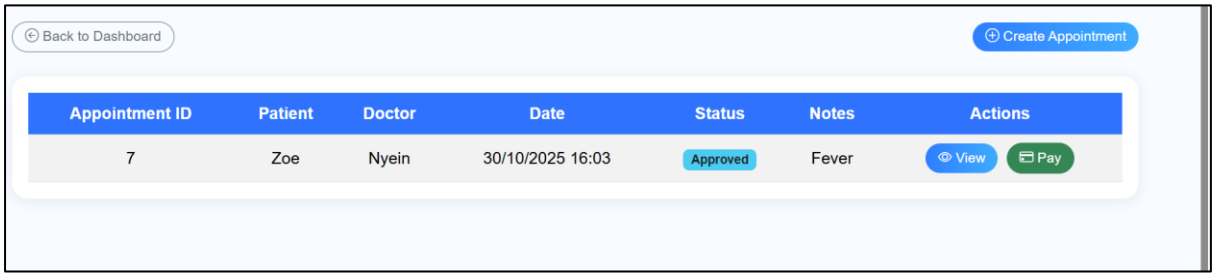


Figure: After a doctor gave prescription

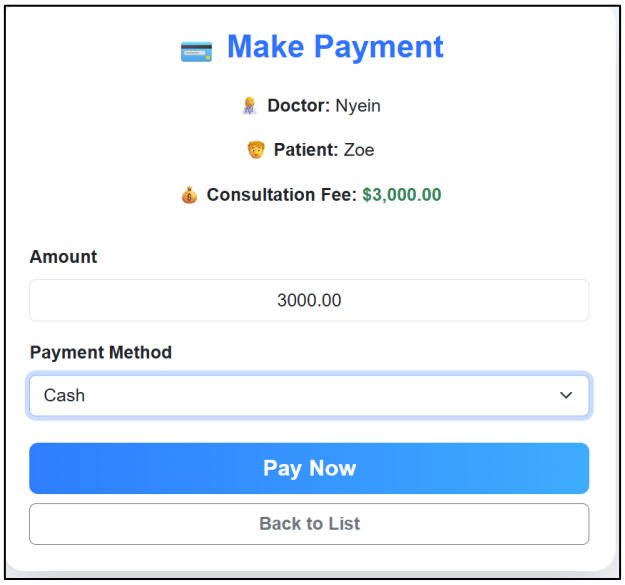
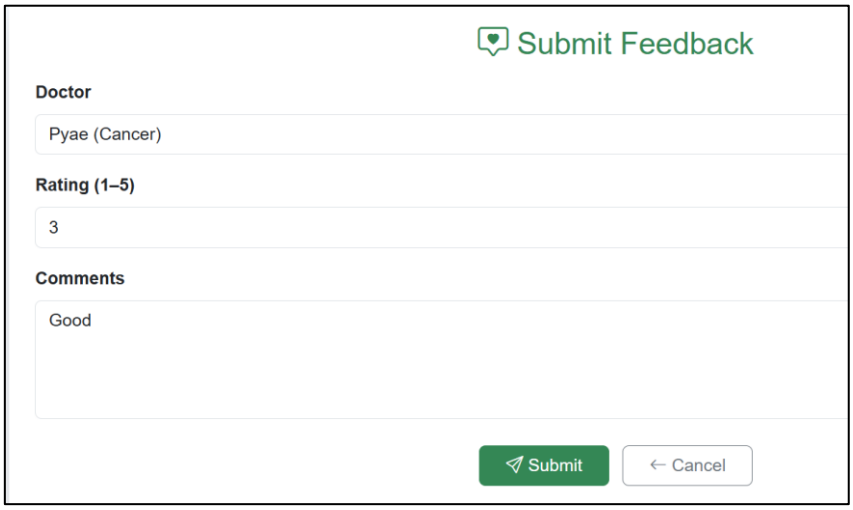


Figure: Make Payment

Figure: Payment Successful

Patient Feedback Dashboard

A patient can pay feedback rate the doctors with starts from 1 to five. A patient can view only his feedback but an admin can view all feedback.



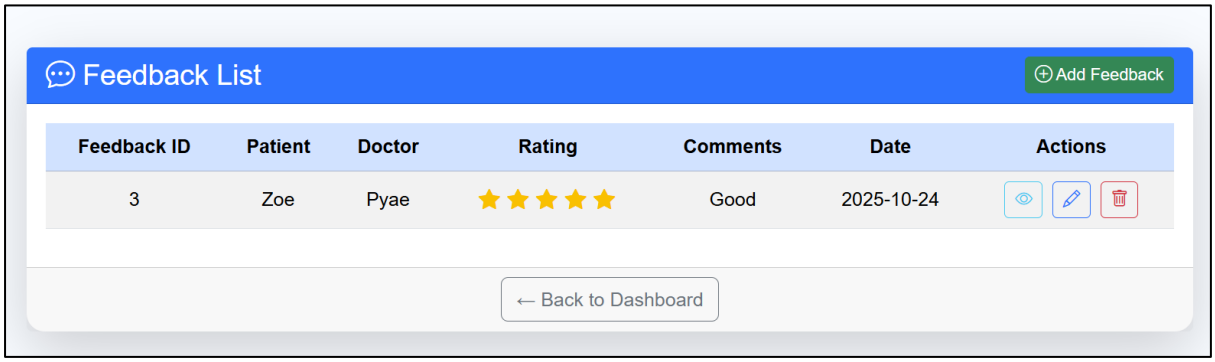


Figure: Feedback Dashboard

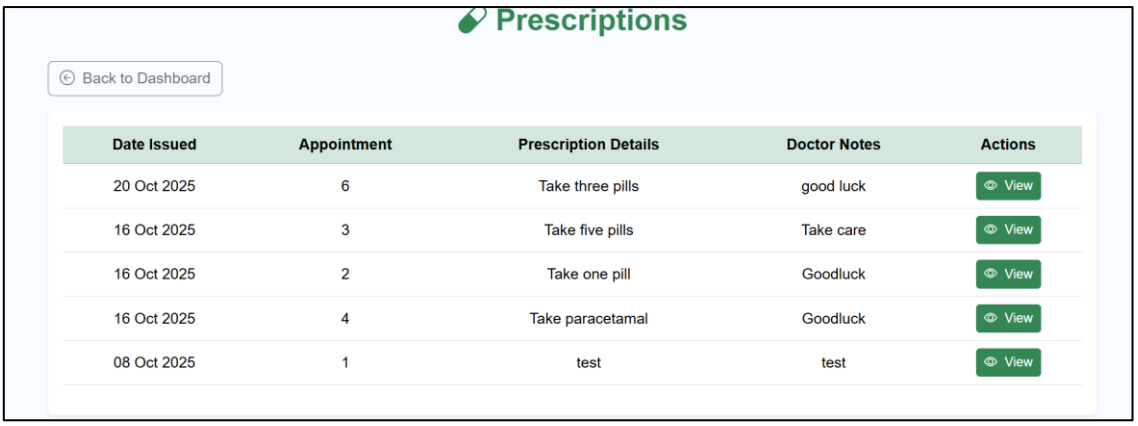
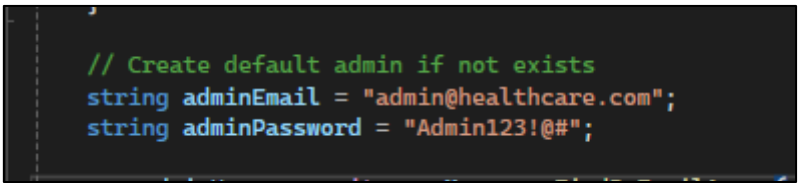


Figure: Patient Prescription Page

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!@#".



```
// Create default admin if not exists
string adminEmail = "admin@healthcare.com";
string adminPassword = "Admin123!@#";
```

Figure: Creating default admin Email and Pw

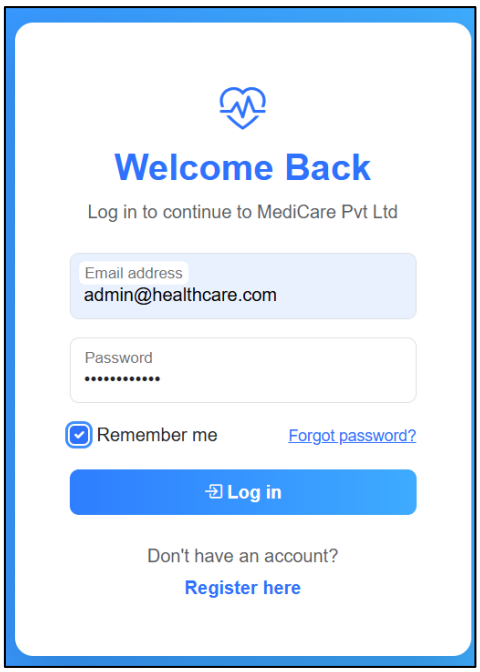


Figure: Login Page

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.

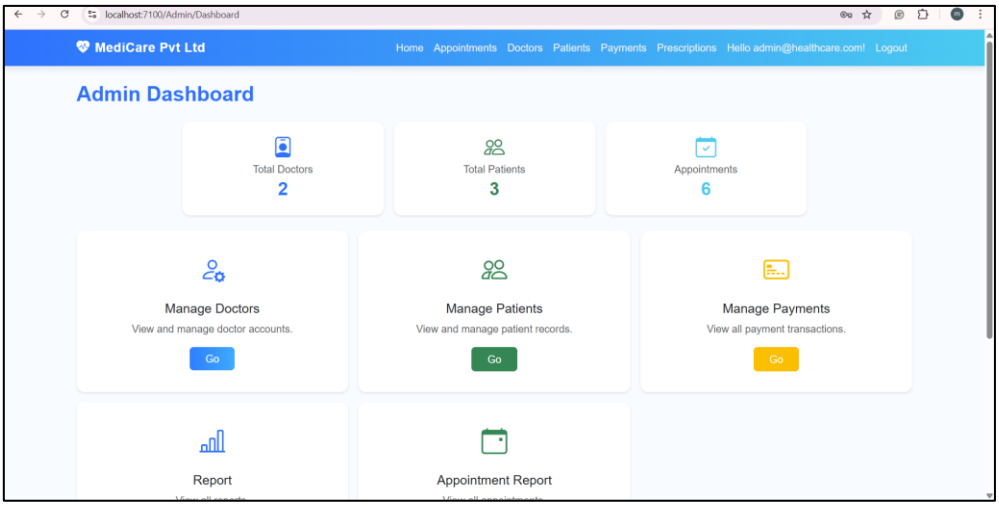


Figure: Admin Dashboard

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.

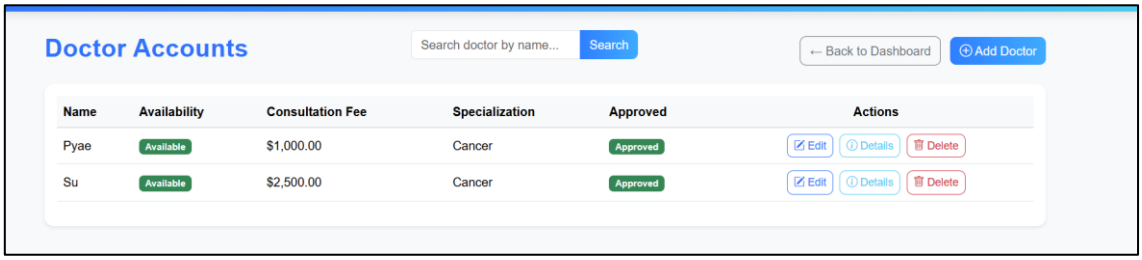


Figure: Doctors management dashboard

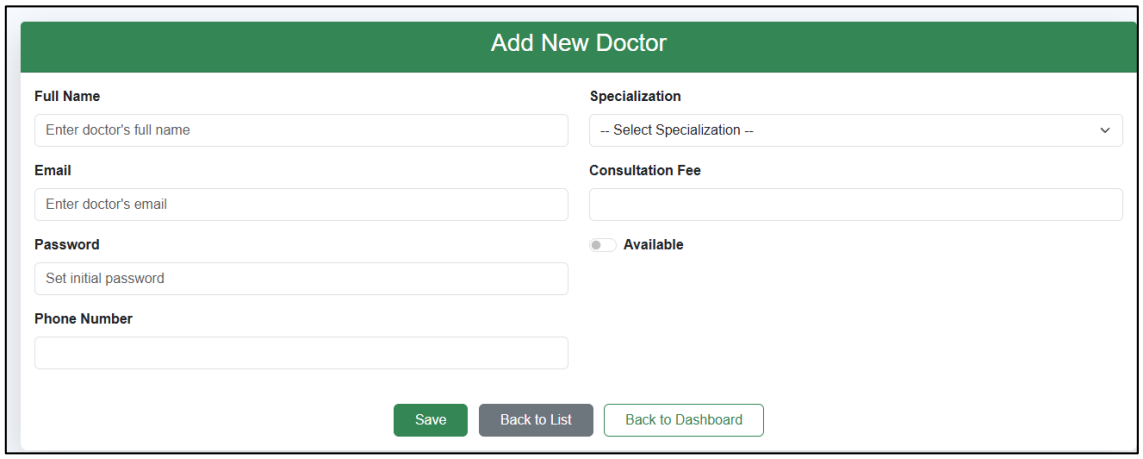


Figure: Creating a new doctor

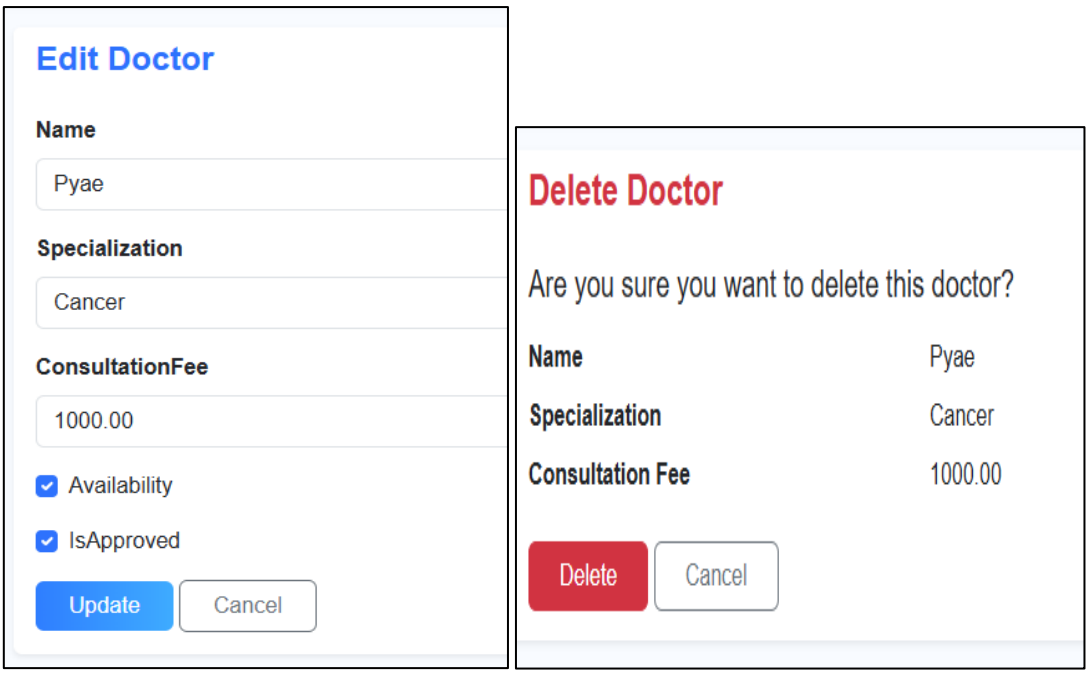


Figure: Edit and Delete Pages

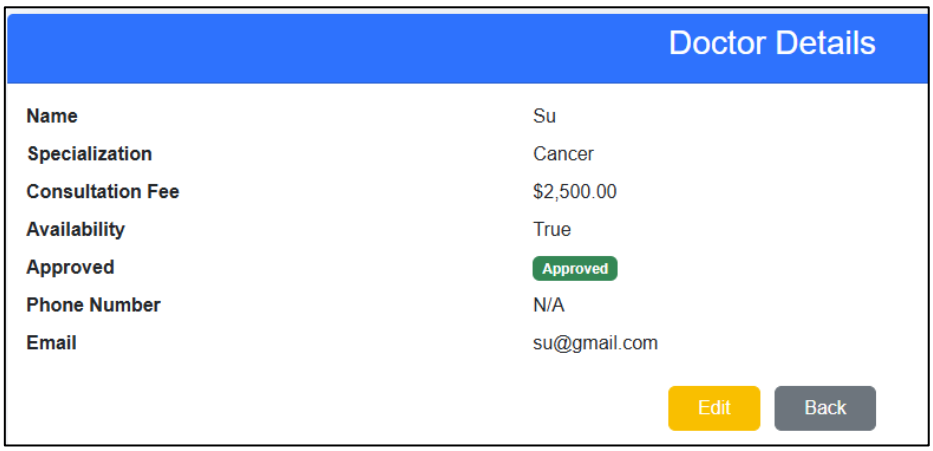


Figure: Doctor Detail Page

Patients Accounts Management Page

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

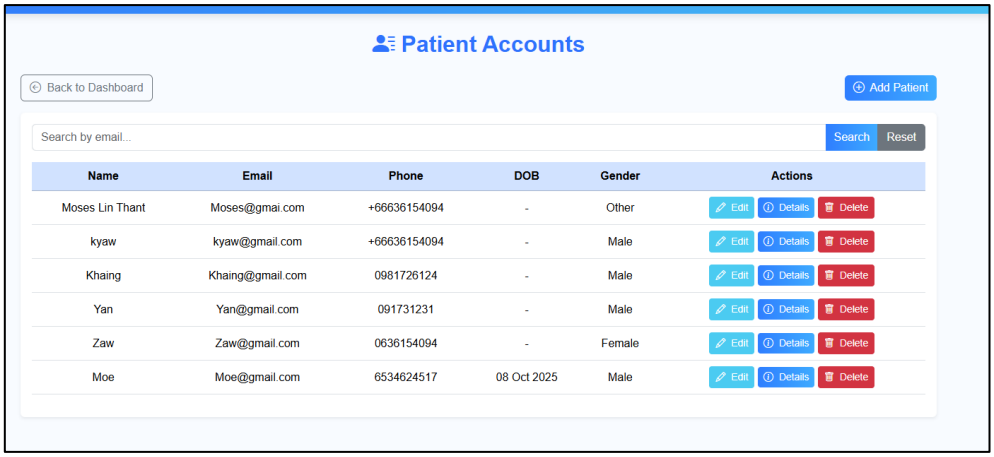


Figure: Patient Account Management

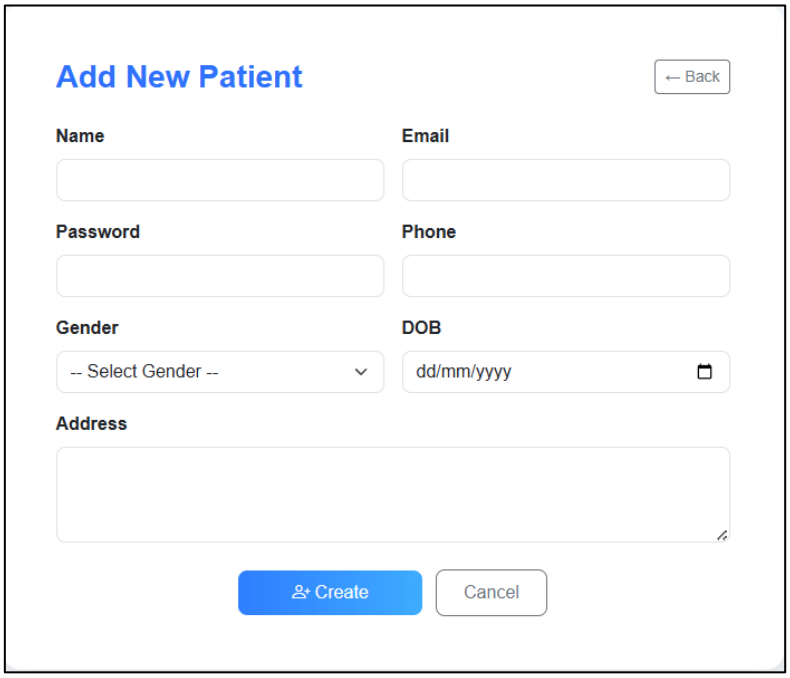


Figure: Create Patient Form

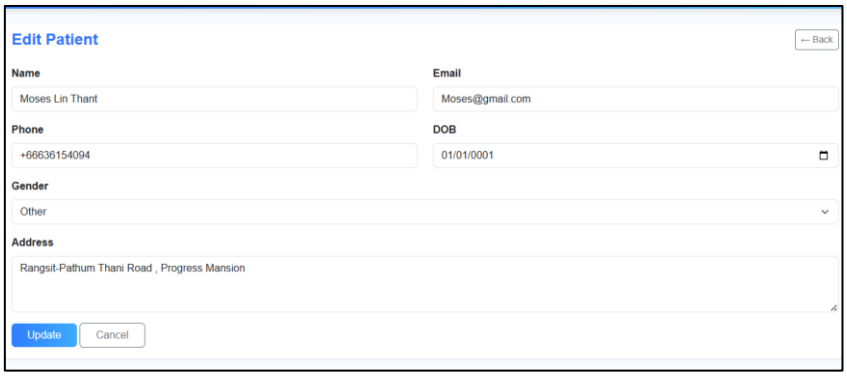


Figure: Patient Edit Form

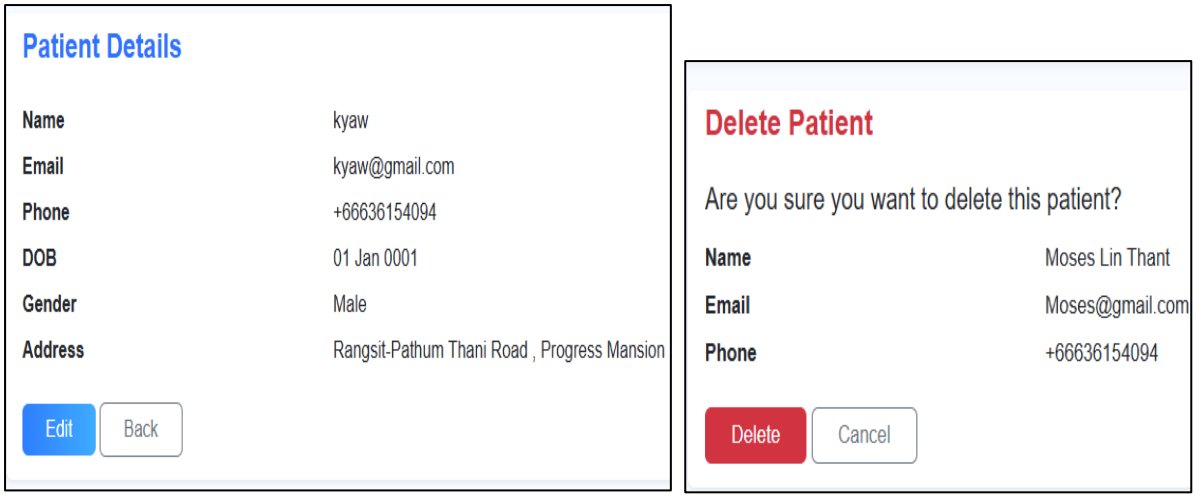


Figure: Patient Edit and Delete Pages

Payment Management Page

In the payment management page, admin can view and edit payment information but he cannot create new payments.

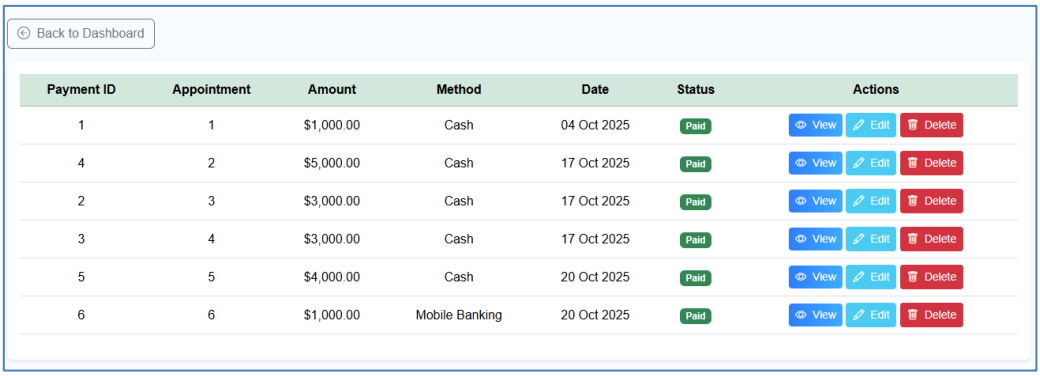


Figure: Payment Management Dashboard

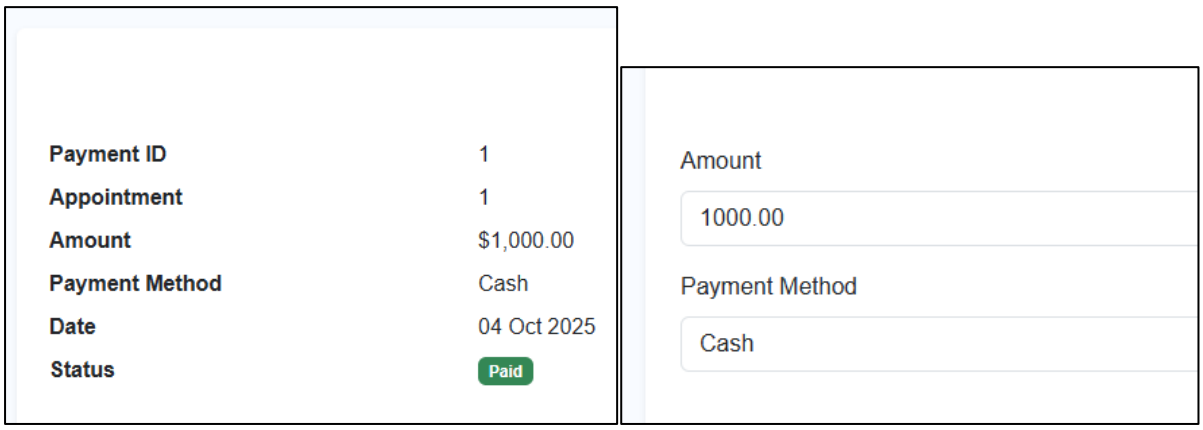


Figure: Detail and Edit payments page

Report Dashboard

In report dashboard, an can see the overall situation of the business. For example, how much income by month, patient age group, patient group by gender. An admin can download these report as a pdf file.

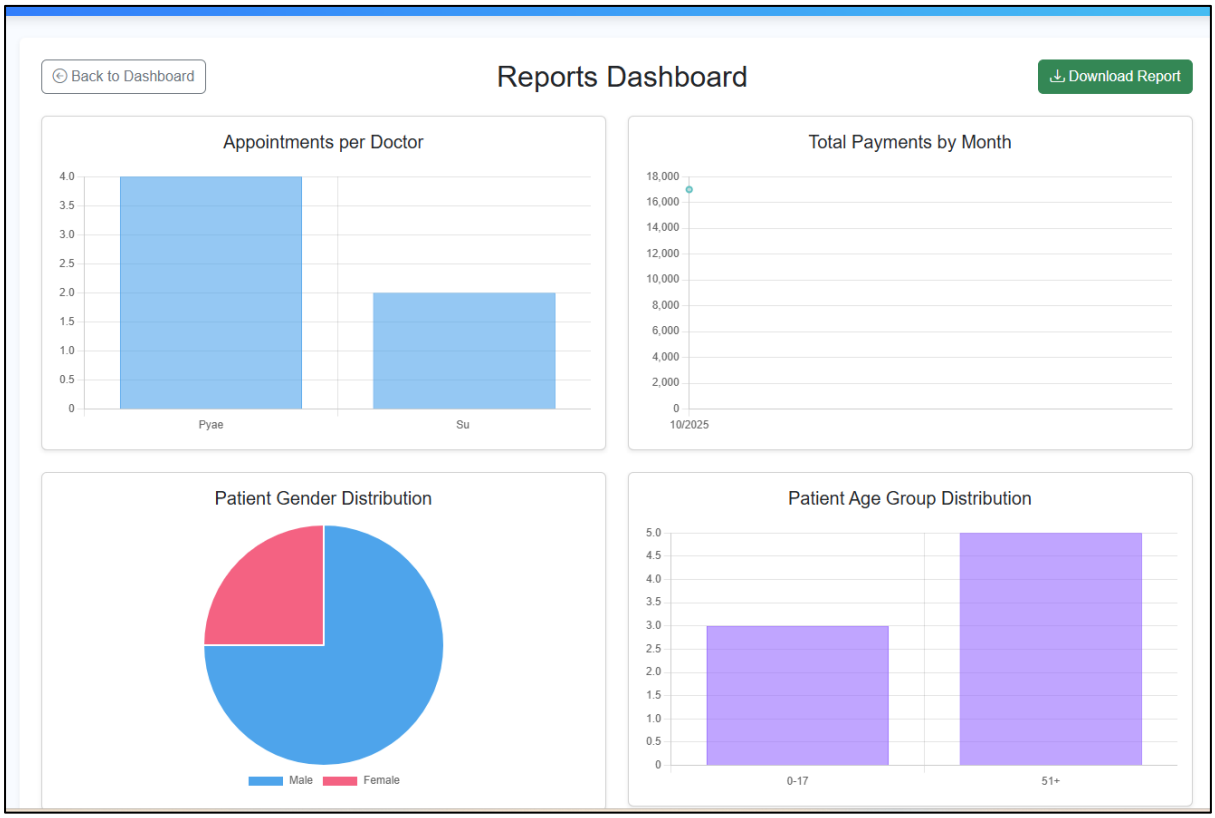


Figure: Report Dashboard

Admin Appointment Dashboard

In appoint dashboard, an admin's duty is to check the appointment details and decide to approve or not. After an admin decided to give an approvement, the status of appointment became "Approved". Only after that a doctor can see this appointment.

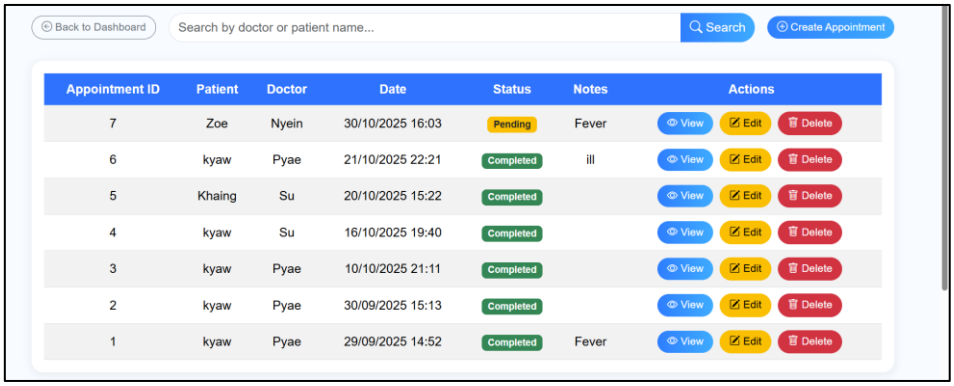


Figure: Before approve

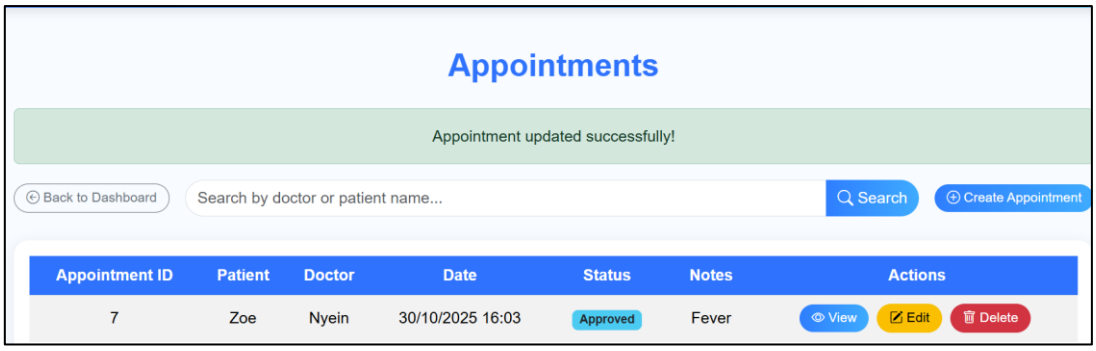


Figure: After Approve

Specializations Dashboard

This dashboard is nothing special but only an admin can view, create and edit.

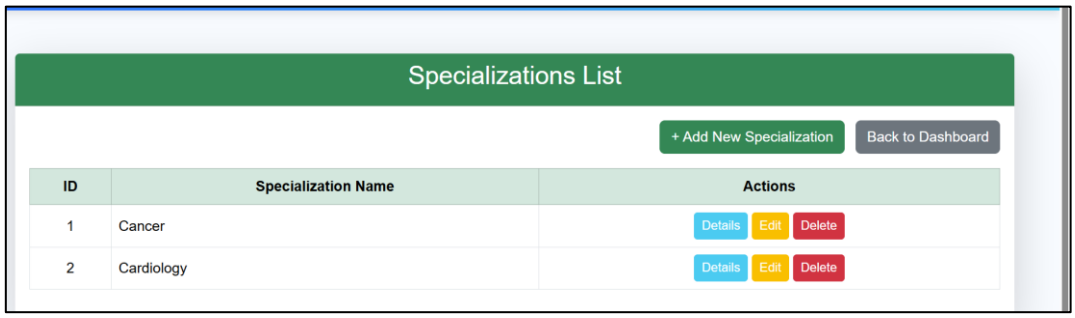


Figure: Specialization Dashboard

Doctor User Guide

In doctor dashboard, a doctor can see today appointment and this week's appointment. A doctor can view his appointments, patients and give prescriptions to the patients. Doctors are restricted to view only their patients and appointments. Only the appointments which are approved by admin can be used to give prescription by a doctor.

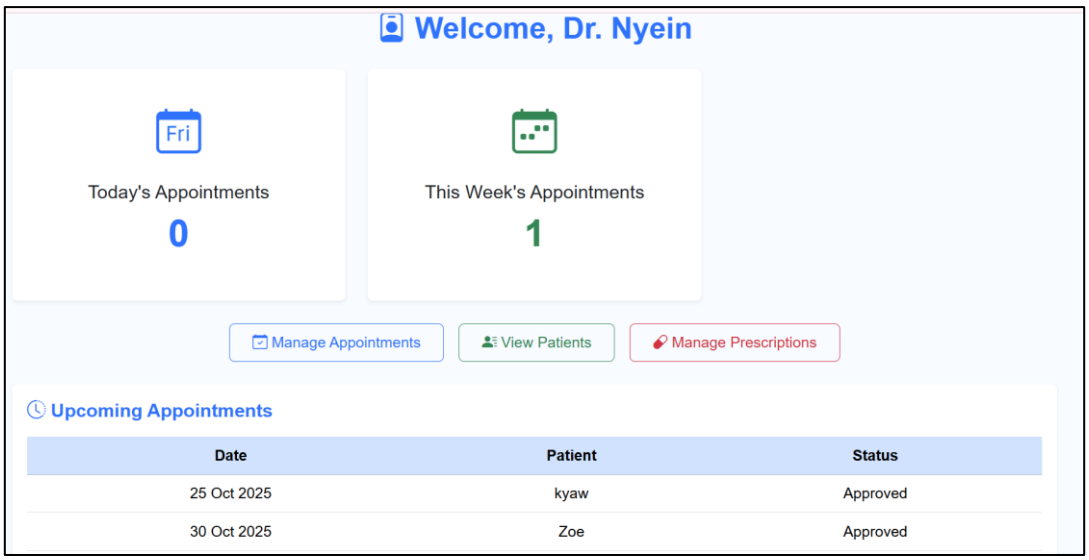


Figure: Doctor Dashboard

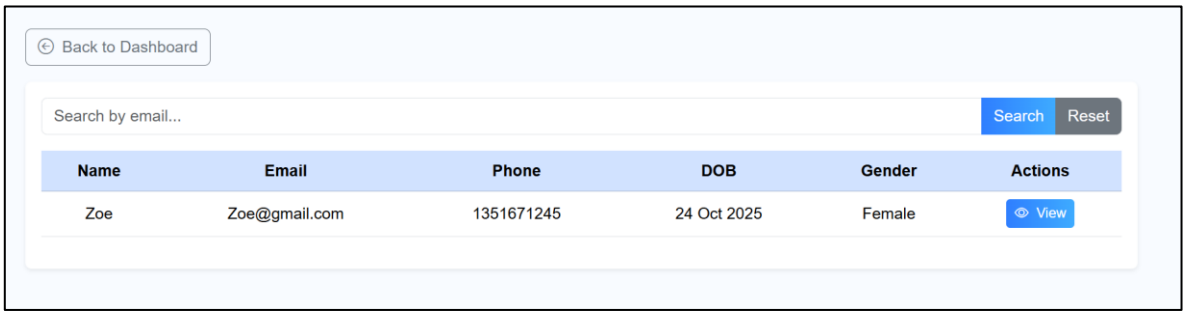


Figure: Current appointment

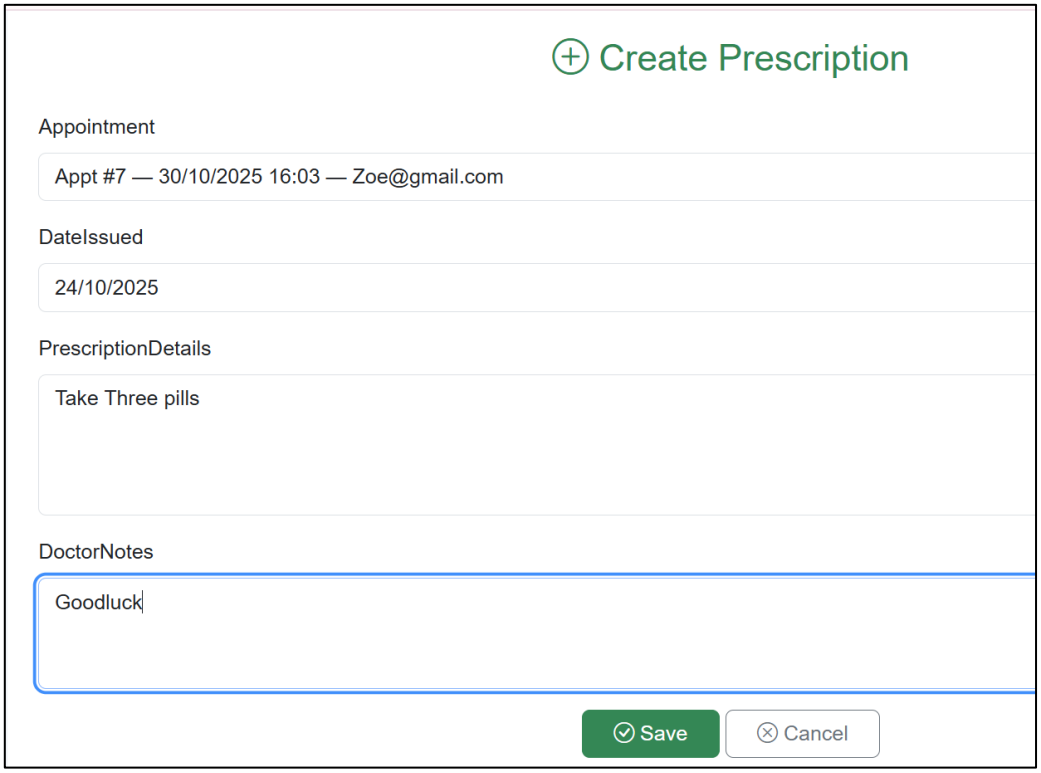


Figure: Create Prescription for the patient

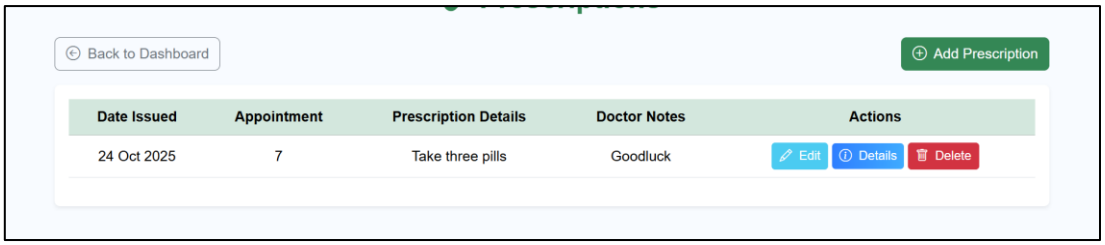


Figure: Creating Prescription successful

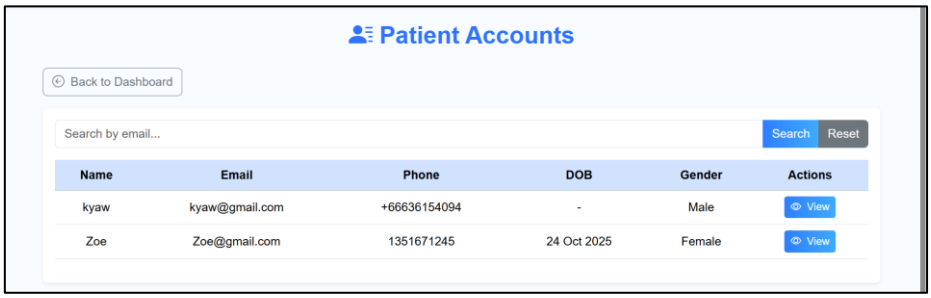


Figure: A doctor can only view their patients

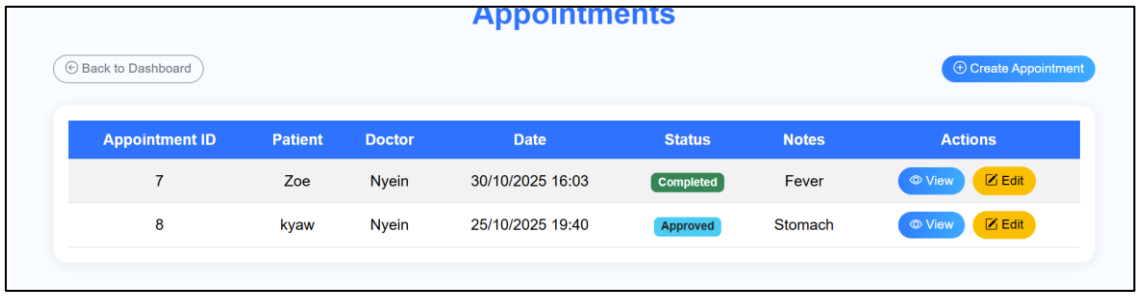


Figure: A doctor can only view his appointments

Weaknesses and Future Development

The Online Healthcare Appointment System works well for small healthcare businesses but there are some issues with big businesses. There are some weak points that can be improved in the future.

One weakness is the lack of real-time notifications. When an appointment is created or cancelled, the system does not send a message or alert to the doctor or patient. Adding automatic notifications through email, SMS or pop-up alerts would make communication faster

and reduce missed appointments. The system also does not have automatic schedule checking and the system does not stop double bookings. A future version should include a smart scheduling future that checks for time conflicts and only allows available slots to be booked.

In terms of security , the system uses ASP.NET identity for login and authentication. But sensitive data like patient records and payment information are not encrypted. Adding advanced data encryption algorithms, HTTPS and following privacy standards such as GDPR or HIPAA will make the system more secure and trustworthy. And also the user interface (UI) can also be improved. Some pages, like the form pages, are too simple and not well arranged. The layout is not very friendly for mobile users.

Lastly, the system is not yet deployed online. In the future, it can be hosted on a cloud platform like Microsoft Azure or AWS so it can be used by real users and can grow easily as more people use it.

References

ASP.NET MVC Pattern | .NET. (n.d.). Microsoft. https://dotnet.microsoft.com/en-us/apps/aspnet/mvc

Contributors, M. O. J. T. a. B. (n.d.). *Get started with Bootstrap*. https://getbootstrap.com/docs/5.3/getting-started/introduction/

rwestMSFT. (n.d.). *SQL Server Management Studio*. Microsoft Learn. https://learn.microsoft.com/en-us/ssms/

Jcjiang. (n.d.). *Entity Framework documentation hub*. Microsoft Learn. https://learn.microsoft.com/en-us/ef/

OpenJS Foundation - openjsf.org. (n.d.). *JQuery*. https://jquery.com/

studio96 & HEALTHCARE ENTERPRISES. (2024, November 4). *Home - Healthcare enterprises*. Healthcare Enterprises. https://healthcareent.co.th/