ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ЗА ДИГИТАЛНИ НАУКИ

„СОФТУНИ БУДИТЕЛ“, гр. София

ДИПЛОМЕН ПРОЕКТ

на..........................Алек Димитров Цветанов.........................

ученик/ученичка от XII Б клас

професия- код: 481030, “Приложен програмист”

специалност- код: 4810301, “Приложно програмиране”

Тема:..........Проследяване на посещения при лекар и издаване на рецепта......................................

Ръководител-

консултант:..............................................................................

Сесия: май-юни 2025г.

Дата:.........................

Съдържание

Увод……………………………………………………………………………..3

Глава 1…………………………………………………………………………..6

Глава 2…………………………………………………………………………..9

Глава 3…………………………………………………………………………..11

Глава 4…………………………………………………………………………..20

Увод

Ефективното управление на информацията на пациентите е много важна модерната практика при медицината. Проследяването на посещения, издадени рецепти и лекарства и медицинска история осигурява по-голяма ефективност и продължителност на услугите. Обаче, традиционната система за запазване на данни като папки и хартия или разпръснати документи се достига до объркване, забавяне на достъп или грешна комуникация сред доставчици на здравни услуги.

Нужда от това приложение

В днешната забързана медицинска среда здравните специалисти и пациентите се нуждаят от интуитивна система за рационализиране на административните процеси. Централизирана платформа, насочена към проследяване на посещенията на пациенти, управление на рецепти и наблюдение на лекарства, може значително да подобри качеството и точността на предоставянето на здравни грижи. Такова решение не само минимизира грешките, но също така позволява по-добро взаимодействие между лекар и пациент, като предоставя ясна представа за медицинската история на пациента.

Цел

Целта на това уеб приложение е да създаде цялостна и удобна за потребителя система, която опростява управлението на посещенията на пациентите, предписанията и лекарствата. Приложението има за цел да предостави интуитивен интерфейс за лекарите, за да регистрират посещения и предписания, за аптеките, за да проследяват издадените лекарства, и за пациентите, за да имат безпроблемен достъп до медицинската си история.

Потребители

Това уеб приложение е предназначено да обслужва:

* + Лекари: За записване на консултации с пациенти и ефективно управление на рецептите.
  + Фармацевти: За проследяване и управление на отпускането на лекарства.
  + Пациенти: Да имат лесен достъп до техните медицински досиета, рецепти и история на лекарствата, осигурявайки по-добро управление на личното здравеопазване.
  + Здравни администратори: За наблюдение и управление на потока на пациентите и точността на записите.

Подробен анализ на функционалностите

1. Проследяване на посещения при лекар

• График: Добавяне на бъдещи посещения с лесно персонализиране (час, дата, лекар).

• История на посещенията: Хронология на всички минали консултации, бележки и препоръки от специалисти.

• Уведомления: Изпращане на напомняния преди насрочени прегледи, интеграция с календара на устройството.

• Цифрови документи: Прикачване на документи, свързани с прегледа (рентгенови снимки, рецепти).

2. Управление на рецепти

• Съхранение на рецепти: Сканиране или ръчно въвеждане на рецепти. Информацията включва лекарство, дозировка и продължителност.

• Информация за лекарства: Интеграция с база данни за лекарства (например Drugs.com), за предоставяне на взаимодействия, странични ефекти или препоръки.

• Срок на валидност: Напомняне за изтичащи рецепти или необходимост от повторно издаване.

3. Напомняния за лекарства

• Автоматични известия: Време за прием, със звук и визуално напомняне.

• Персонализация: Въвеждане на тип лекарство, дозировка, честота и снимка.

• Синхронизация: Поддръжка на множество устройства за семейно използване.

4. Проследяване на здравни показатели

• Данни за здравето: Регистрация на кръвно налягане, пулс, тегло, кръвна захар и др.

• Графики и анализи: Автоматично генерирани отчети за здравни тенденции.

• Доклади за лекари: Споделяне на информация директно с медицински специалист.

Глава 1

Какво служи проследяваща система?

Системата за проследяване на пациенти наистина е точно както звучи. Това е система за наблюдение на движенията на пациентите през цялото им време в болницата. През годините е имало много различни начини, по които клиницистите проследяват своите пациенти, включително писалка и хартия, електронни таблици, а сега и с използването на RFID технология и Интернет на нещата (IoT).

Процесът на проследяване на пациенти се е развил изключително много и има защо. Електронните таблици, които съдържат големи обеми информация, могат бързо да се объркат, когато се споделят с множество служители и оставят твърде много място за грешки.

С огромното търсене на системите за здравеопазване става все по-важно да се внедри система за проследяване на пациенти, която може да намали натоварването на лекарите и да осигури по-безопасна болнична среда за пациентите. Големите болници се нуждаят от допълнителни ресурси, които могат да помогнат за правилната грижа за пациентите от приема до изписването.

Най-често етикетите за радиочестотна идентификация (RFID) и IoT технологията се използват в интелигентна болнична среда. RFID етикетите се носят от пациентите и се наблюдават от доставчици на здравни услуги, които могат да проследяват в реално време състоянието и местоположението на всеки пациент, както и да имат достъп до цифрова директория на техните здравни досиета. Тези етикети често са в болнични гривни или се носят около врата на ремък.

Как се документират посещенията?

При документирането на посещенията на пациенти здравните системи използват предимно структурирани методи като електронни здравни досиета (EHR) или традиционни писмени бележки, с ясни протоколи за събиране, съхранение и достъп на данни. Ето ключови точки относно начина, по който се обработва документацията за посещение:

Записване на данни

Бележки за SOAP: Обикновено се използва стандартна рамка, наречена SOAP (Subjective, Objective, Assessment, Plan).

Subjective: Patient-reported information, including symptoms and concerns.

Objective: Measurable data from physical exams, lab results, or imaging.

Assessment: Diagnosis or clinical impression based on the gathered information.

Plan: The treatment strategy, including prescriptions, procedures, or follow-ups​

Характеристики на EHR: Тези цифрови системи позволяват последователно въвеждане на данни за пациента, включително демографски данни, клинични находки, история на лекарствата и планове за лечение. Те предоставят стандартизирани шаблони за намаляване на грешките и осигуряване на пълнота.

Съхранение на данни

Сигурност и интегритет: ЕЗД използват криптиране, регистрационни файлове за одит и контрол на достъпа, за да защитят информацията за пациента. Регистрационните файлове проследяват кой е осъществил достъп или променил записите и предотвратява неоторизирани промени

Политики за съхранение: Клиничните данни често се съхраняват за дълги периоди (напр. десетилетия), тъй като може да са необходими за текущи грижи, законови изисквания или изследователски цели.

Достъпни механизми

Достъп, базиран на роли: Само упълномощен персонал (напр. лекари, медицински сестри или административен персонал) може да преглежда или променя конкретни данни, като гарантира спазване на правилата за поверителност.

Портали за пациенти: Много системи включват портали за пациенти, където хората могат да видят своята история на посещения, резултати от тестове и предписани лечения, насърчавайки прозрачността

Глава 2

Изисквания към приложението (основна част)

Технологии, използвани за изграждане на приложението:

* **ASP.NET Core MVC** (for Web Application)
* **Entity Framework Core** (for Database Operations)
* **Microsoft SQL Server** (for Data Storage)
* **Identity Framework** (for Authentication & Authorization)
* **Selenium with NUnit** (for Automated Testing)
* **Bootstrap & Razor Views** (for UI)

**2.1. Цели и задачи на проекта**

Основната цел на проекта е да се създаде уеб приложение, което да подпомогне процеса на управление на медицинските посещения. За постигане на тази цел трябва да бъдат изпълнени следните задачи:

* Разработка на база данни за съхранение на информация за пациенти, лекари, посещения и предписани лекарства.
* Създаване на ролево-базирана система за достъп.
* Осигуряване на възможност за лекарите да управляват пациентските досиета.
* Интеграция на механизъм за уведомяване на пациентите за предстоящи прегледи.
* Тестване и валидация на приложението.

**2.2. Анализ на съществуващи решения**

Съществуват множество приложения за управление на медицински посещения, като някои от тях са част от по-големи медицински информационни системи. Недостатъците на повечето налични решения включват сложност на интерфейса, липса на персонализирани функции и високи разходи за внедряване. Нашето приложение цели да предостави по-интуитивна и достъпна алтернатива.

2.3 Функционални изисквания

- Удостоверяване на потребител: Сигурно влизане и базиран на роли достъп.

- Резервиране на час: Пациентите могат да резервират час при налични лекари.

- Лекарско табло: Лекарите могат да управляват срещи и записи на пациенти.

- Табло за пациенти: Пациентите могат да преглеждат своята медицинска история и предстоящи срещи.

- Известия: Известия по имейл за потвърждения на срещи и напомняния.

3.2 Нефункционални изисквания

Сигурност: Използване на ASP.NET идентичност за удостоверяване.

Мащабируемост: Проектиран да управлява ефективно множество потребители.

Производителност: Оптимизирани заявки към база данни за бързо време за отговор.

Потребителско изживяване: Интуитивен потребителски интерфейс както за пациенти, така и за лекари.

Глава 3

**2.3. Архитектура и технология на приложението**

Приложението е разделено на четири основни проекта:

* **Core** – съдържа модели и услуги.
* **Infrastructure** – отговаря за управлението на базата данни.
* **Unit Tests** – съдържа тестовете на приложението.
* **Web App** – основният интерфейс на приложението.

Технологичният стек включва ASP.NET Core за бекенд, Entity Framework Core за работа с базата данни и Angular за фронтенд частта.

**2.4. Описание на модулите**

* **Управление на пациенти:** записване на нови пациенти, редакция и преглед на досиетата им.
* **Управление на лекари:** възможност за добавяне на нови лекари и преглед на техните пациенти.
* **Управление на медицински посещения:** създаване, редактиране и преглед на записани посещения.
* **Управление на предписани лекарства:** добавяне и проследяване на предписани медикаменти.
* **Ролево-базирана автентикация:** разграничение между роли на пациенти и лекари.

**2.5. Описание на базата данни**

Базата данни съдържа следните основни таблици:

* **Patients** (Пациенти)
* **Appointments(Предварителни срещи)**
* **Doctors** (Лекари)
* **Visits** (Посещения)
* **Prescriptions** (Предписани лекарства)
* **Users** (Потребители)
* **PatientDoctor** (Много-към-много връзка между пациенти и лекари)

**2.6. Ролева автентикация и функционалност**

Приложението използва IdentityUser за управление на автентикацията. Пациентите имат достъп до своите медицински досиета и известия, докато лекарите могат да редактират и добавят нови записи.

**2.7. Тестване и валидация**

За гарантиране на надеждността на приложението се използват тестове с NUnit и Selenium за автоматизация на потребителските взаимодействия.

**2.8. Roles & Permissions**

**3.1 Patients**

**✅ Can:**

* **Register and log in.**
* **View their medical records.**
* **Book an appointment with a doctor.**
* **View scheduled appointments.**

**❌ Cannot:**

* **Edit or delete other users' appointments.**
* **Modify doctor schedules.**

**3.2 Doctors**

**✅ Can:**

* **Register and log in.**
* **Manage patient records.**
* **Schedule and edit appointments.**
* **View their assigned patients.**

**❌ Cannot:**

* **Access other doctors' patients.**
* **Delete appointments without authorization.**

**3.3 Administrators**

**✅ Can:**

* **Manage both patient and doctor accounts.**
* **Assign roles.**
* **Oversee appointment management.**

**4. Project Structure**

**The project consists of the following core components:**

**4.1 Controllers**

**Controllers manage HTTP requests and interact with services to return appropriate responses to the views.**

**4.1.1 AppointmentController**

**Handles appointment-related actions, including:**

* **GET Create(): Displays the appointment creation page.**
* **POST Create(CreateAppointmentViewModel model): Handles form submissions for creating an appointment.**
* **GET Manage(): Displays appointments for doctors and patients.**

**4.1.2 AccountController**

**Manages authentication and authorization:**

* **Register(): Handles user registration.**
* **Login(): Authenticates users.**
* **Logout(): Ends user sessions.**

**4.1.3 HomeController**

**Handles general pages like the homepage and dashboard.**

**4.1.4 DoctorController**

**Manages appointments, medical records and adding users as doctors:**

* **Become(): An user can become a doctor after registration**
* **ViewMyPatients(): The doctor can see all of his patients**
* **Dashboard(): Can see the dashboard of his patients**
* **Add(): Creates a patient medical record for the assigned patients**
* **ViewPatientRecord: Detailed view of his patient’s record**
* **Edit: Editing a patient record**
* **Delete: Deleting a patient record**

**4.1.5 PatientController**

**Can view their medical record and assign doctors by their specialization and needs**

* **Become(): Handles user registration.**
* **ViewMyRecords(): Authenticates users.**
* **AssignDoctor(): Ends user sessions.**

**4.2 Services**

**Services act as an intermediary between controllers and database repositories.**

**4.2.1 AppointmentService**

* **GetAppointmentsByPatientAsync(patientId): Fetches appointments for a specific patient.**
* **GetAppointmentsByDoctorAsync(doctorId): Retrieves doctor appointments.**
* **CreateAppointmentAsync(model): Saves a new appointment to the database.**

**4.2.2 UserService**

* **GetUserById(userId): Retrieves user details.**
* **AssignRole(user, role): Assigns roles to users.**

**4.2.3 DoctorService**

* **Task<string> GetDoctorIdAsync(string userId);**

 **Purpose:** Retrieves the Doctor entity's unique ID (as a string) using the associated ASP.NET Identity userId.

 **Use case:** When you need to link the currently logged-in user to their doctor profile.

* **Task<bool> ExistsByIdAsync(string userId);**

 **Purpose:** Checks if a doctor profile exists for the given userId.

 **Use case:** Prevent duplicate doctor profiles or ensure the user is a doctor before allowing access to certain features.

* **Task<Guid> AddDoctorAsync(string userId, BecomeDoctorModel model);**

 **Purpose:** Creates a new doctor profile linked to the provided user ID using the submitted form data (BecomeDoctorModel).

 **Use case:** When a user signs up as a doctor and submits their details (name, specialization, etc.).

* **Task<List<PatientViewModel>> GetPatientsByDoctorIdAsync(Guid doctorId);**
* **Purpose:** Returns a list of patients assigned to the specified doctor.
* **Use case:** To display all patients under a specific doctor's care, e.g., in the doctor's dashboard.

**4.2.4 PatientService**

* **Task<string> GetPatientIdAsync(string userId);**

 **Purpose:** Retrieves the Patient entity's ID (as a string) associated with the given userId.

 **Use case:** Linking logged-in users to their patient profile.

* **Task<bool> ExistsByIdAsync(string userId);**

 **Purpose:** Checks whether a patient profile exists for the specified user.

 **Use case:** Ensures users don't register as patients multiple times.

* **Task<Guid> AddPatientAsync(string userId, BecomePatientViewModel model);**

 **Purpose:** Creates a new patient profile for the user with data from BecomePatientViewModel.

 **Use case:** When a user registers as a patient.

* **Task AssignDoctorToPatientAsync(Guid patientId, Guid doctorId);**

 **Purpose:** Links a patient with a doctor.

 **Use case:** Used after registration or admin panel to assign patients to doctors.

* **Task<Patient?> GetPatientByUserIdAsync(string userId);**

 **Purpose:** Retrieves a Patient entity by their linked userId.

 **Use case:** Useful when loading the patient profile or records for a logged-in user.

**4.2.5 MedicalRecordService**

* **Task<int> AddPatientRecordAsync(PatientRecordViewModel model, string doctorUserId);**

 **Purpose:** Adds a new medical record created by the doctor identified by doctorUserId.

 **Use case:** When a doctor submits a new diagnosis or prescription after visit with patient.

* **Task<List<SelectListItem>> GetPatientsForDoctorAsync(Guid doctorUserId);**

 **Purpose:** Returns a list of patients assigned to a doctor, formatted for dropdown selection.

 **Use case:** Used in views when a doctor is selecting a patient to manage.

* **Task<PatientRecordViewModel?> GetPatientRecordByIdAsync(int id);**

 **Purpose:** Fetches a single patient medical record by its unique ID.

 **Use case:** Viewing or editing a specific medical record.

* **Task<List<PatientRecordViewModel>> GetPatientRecordsByPatientIdAsync(Guid patientId);**

 **Purpose:** Returns all records associated with a specific patient.

 **Use case:** For patients reviewing their own medical history.

* **Task<List<PatientRecordViewModel>> GetPatientRecordsByDoctorIdAsync(string doctorId);**

 **Purpose:** Returns all medical records created by a specific doctor.

 **Use case:** A doctor reviewing or managing past patient interactions.

* **Task<IEnumerable<PatientRecordViewModel>> GetAllPatientRecordsAsync();**

 **Purpose:** Retrieves all patient records in the system.

 **Use case:** Useful for admin-level access or generating reports.

* **Task<bool> UpdatePatientRecordAsync(PatientRecordViewModel model);**

 **Purpose:** Updates an existing medical record using data from the form model.

 **Use case:** When a doctor edits a previously created record.

* **Task DeletePatientRecordAsync(int id);**
* **Purpose:** Deletes a patient record by its ID.
* **Use case:** Admin or doctor removes an outdated or incorrect record.

**4.3 Models**

**Defines the structure of the data being used in the system.**

**4.3.1 Entities (Database Models)**

**4.3.1.1 Appointment**

**Represents an appointment record:**

**public class Appointment {**

**[Key]**

**public int Id { get; set; }**

**public Guid PatientId { get; set; }**

**public virtual Patient Patient { get; set; }**

**public Guid DoctorId { get; set; }**

**public virtual Doctor Doctor { get; set; }**

**public DateTime AppointmentDate { get; set; }**

**public string Reason { get; set; }**

**public string Status { get; set; } = "Scheduled";**

**}**

**4.3.1.2 Patient**

**Signs and represents the user as a patient**

**public class Patient {**

**[Key]**

**public Guid Id { get; set; }**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string UserId { get; set; }**

**public virtual List<Appointment> Appointments { get; set; }**

**}**

**4.3.1.3 Doctor**

**Signs and represents the user as a doctor**

**public class Doctor {**

**[Key]**

**public Guid Id { get; set; }**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string UserId { get; set; }**

**public virtual List<Appointment> Appointments { get; set; }**

**}**

**4.3.1.4 Billing**

**The bill for the visit**

**public class Billing**

**{**

**[Key]**

**public int Id { get; set; }**

**public int VisitId { get; set; }**

**[Column(TypeName = "decimal(18,2)")]**

**public decimal Amount { get; set; }**

**public string InsuranceDetails { get; set; }**

**public string PaymentStatus { get; set; } // e.g., Paid, Pending, Denied**

**public Visit Visit { get; set; }**

**}**

**4.3.1.5 PatientDoctor**

**The many-to-many connection between patient and doctor tablas**

**public class PatientDoctor**

**{**

**public Guid PatientId { get; set; }**

**public Patient Patient { get; set; }**

**public Guid DoctorId { get; set; }**

**public Doctor Doctor { get; set; }**

**}**

**4.3.1.6 Visit**

**Stores the visit and the medical record of the patient**

**public class Visit**

**{**

**[Key]**

**public int Id { get; set; }**

**public Guid PatientId { get; set; }**

**public Guid DoctorId { get; set; }**

**public DateTime VisitDate { get; set; }**

**//public string Status { get; set; } // e.g., Scheduled, Completed, Canceled**

**public string ReasonForVisit { get; set; }**

**public string Diagnosis { get; set; }**

**public string Prescriptions { get; set; }**

**public string Notes { get; set; }**

**public Patient Patient { get; set; }**

**public Doctor Doctor { get; set; }**

**}**

**4.3.2 View Models**

**4.3.2.1Appointment view models**

**4.3.2.1.1 CreateAppointmentViewModel**

**Used to capture appointment creation details.**

**public class CreateAppointmentViewModel {**

**public Guid PatientId { get; set; }**

**public Guid DoctorId { get; set; }**

**public DateTime AppointmentDate { get; set; }**

**public string Reason { get; set; }**

**public List<SelectListItem> AvailableDoctors { get; set; }**

**}**

**4.3.2.1.2 ManageAppointmentsViewModel**

**Used to display appointments for patients and doctors.**

**public class ManageAppointmentsViewModel {**

**public List<AppointmentViewModel> Appointments { get; set; } = new List<AppointmentViewModel>();**

**public CreateAppointmentViewModel NewAppointment { get; set; } = new CreateAppointmentViewModel();**

**public List<SelectListItem> AvailableDoctors { get; set; } = new List<SelectListItem>();**

**public List<SelectListItem> AvailablePatients { get; set; } = new List<SelectListItem>();**

**}**

**4.3.2.1.3 AppointmentViewModel**

**The model of a booked appointment between patient and doctor**

**public class AppointmentViewModel**

**{**

**public int Id { get; set; }**

**public Guid PatientId { get; set; } // For doctor selection**

**public Guid DoctorId { get; set; } // For patient selection**

**public DateTime AppointmentDate { get; set; }**

**public string Reason { get; set; }**

**public string Status { get; set; }**

**public string? PatientName { get; set; } // For doctor view**

**public string? DoctorName { get; set; } // For patient view**

**public List<SelectListItem> AvailableDoctors { get; set; } = new List<SelectListItem>();**

**public List<SelectListItem> AvailablePatients { get; set; } = new List<SelectListItem>();**

**}**

**4.3.2.2 Doctor view models**

**4.3.2.2.1 AddPatientRecordModel**

**public class AddPatientRecordModel**

**{**

**public Guid PatientId { get; set; } // Select from existing patients**

**public string Diagnosis { get; set; }**

**public string Treatment { get; set; }**

**[DataType(DataType.Date)]**

**public DateTime VisitDate { get; set; } = DateTime.Now;**

**public string ReasonForVisit { get; set; }**

**public string Prescriptions { get; set; }**

**public string Notes { get; set; }**

**public List<SelectListItem> Patients { get; set; } = new();**

**}**

**4.3.2.2.2 BecomeDoctorModel**

**The requirements to become a doctor**

**public class BecomeDoctorModel**

**{**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string Specialization { get; set; }**

**public string ContactInformation { get; set; }**

**public string OfficeLocation { get; set; }**

**}**

**4.3.2.2.3 DoctorDashboardViewModel**

**Shows the list of medical records of patients**

**public class DoctorDashboardViewModel**

**{**

**public string DoctorId { get; set; }**

**public List<PatientRecordViewModel> PatientRecords { get; set; } = new List<PatientRecordViewModel>();**

**}**

**4.3.2.2.4 DoctorViewModel**

**Shows details of the doctor when assigning**

**public class DoctorViewModel**

**{**

**public Guid Id { get; set; }**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string Specialization { get; set; }**

**}**

**4.3.2.3 Patient view models**

**4.3.2.3.1 AssignDoctorViewModel**

**Shows the list of all doctors**

**public class AssignDoctorViewModel**

**{**

**public List<DoctorViewModel> Doctors { get; set; } = new List<DoctorViewModel>();**

**}**

**4.3.2.3.1 BecomePatientViewModel**

**The requirements of becoming a patient**

**public class BecomePatientViewModel**

**{**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public DateTime DateOfBirth { get; set; }**

**public string Gender { get; set; }**

**public string ContactInformation { get; set; }**

**public string Address { get; set; }**

**public string EmergencyContact { get; set; }**

**}**

**4.4 Views**

**Views render the UI elements.**

**4.4.1 Appointent views**

**4.4.1.1 Create.cshtml**

**Allows users to book an appointment.**

**<form asp-action="Create" method="post">**

**<label>Select Doctor</label>**

**<select asp-for="DoctorId" class="form-control" asp-items="Model.AvailableDoctors"></select>**

**<label>Date</label>**

**<input asp-for="AppointmentDate" type="datetime-local" class="form-control"/>**

**<label>Reason</label>**

**<textarea asp-for="Reason" class="form-control"></textarea>**

**<button type="submit">Book</button>**

**</form>**

**4.4.2 Manage.cshtml**

**Lists existing appointments.**

**@foreach (var appointment in Model.Appointments) {**

**<tr>**

**<td>@appointment.DoctorName</td>**

**<td>@appointment.AppointmentDate</td>**

**<td>@appointment.Reason</td>**

**<td>@appointment.Status</td>**

**</tr>**

**}**

**4.4.2 Doctor views**

**4.4.2.1 Add.cshtml**

**<div class="form-group">**

**<label for="PatientId">Patient</label>**

**<select asp-for="PatientId" asp-items="Model.Patients" class="form-control">**

**<option value="">-- Select Patient --</option>**

**</select>**

**<span asp-validation-for="PatientId" class="text-danger"></span>**

**</div>**

**<div class="form-group">**

**<label for="Diagnosis">Diagnosis</label>**

**<input asp-for="Diagnosis" class="form-control" required />**

**</div>**

**<div class="form-group">**

**<label for="VisitDate">Visit Date</label>**

**<input asp-for="VisitDate" type="date" class="form-control" required />**

**</div>**

**<div class="form-group">**

**<label for="ReasonForVisit">Reason for Visit</label>**

**<textarea asp-for="ReasonForVisit" class="form-control" required></textarea>**

**</div>**

**<div class="form-group">**

**<label for="Prescriptions">Prescriptions</label>**

**<textarea asp-for="Prescriptions" class="form-control"></textarea>**

**</div>**

**<div class="form-group">**

**<label for="Notes">Notes</label>**

**<textarea asp-for="Notes" class="form-control"></textarea>**

**</div>**

**<button type="submit" class="btn btn-success">Save</button>**

**<a asp-action="Dashboard" class="btn btn-secondary">Cancel</a>**

**</div>**

**4.4.2.2 Dashboard.cshtml**

**4.4.2.3 Edit.cshtml**

**4.4.2.4 Become.cshtml**

**4.4.2.5 MyPatients.cshtml**

**4.4.3 Patient views**

**4.4.3.1 AssignDoctor.cshtml**

**4.4.3.2 Become.cshtml**

**4.4.3.3 ViewMyRecords.cshtml**

**5. How Components Work Together**

1. **Users register/login via AccountController.**
2. **Patients book appointments using AppointmentController, selecting from available doctors.**
3. **Doctors manage appointments through the Manage view.**
4. **Data is stored in SQL Server using Entity Framework Core.**
5. **Authorization ensures role-based access control using Identity Framework.**
6. **Automated tests run with Selenium & NUnit.**

Глава 4

**Ръководство на потребителя**

**4.1. Вход в системата**

Потребителите трябва да се впишат чрез своя акаунт. В зависимост от ролята си, те получават достъп до различни модули.

**4.2. Достъп до пациентските досиета**

* Пациентите виждат своята медицинска информация.
* Лекарите могат да добавят нови записи и предписания.

**4.3. Управление на посещения**

* Записване на ново посещение.
* Преглед на предишни посещения.
* Актуализация на медицинските досиета.

Глава 5

The Doctor Appointment Management System simplifies scheduling and medical record management. The system ensures security, role-based access, and efficient database interactions.

**6.1 Future Improvements**

* **Implement Video Consultation Feature**
* **Add AI-based Recommendations for Appointment Scheduling**
* **Integrate Payment Gateway for Online Consultations**