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 или за посещения при доктор. 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)

Настоящият проект е изграден с ясна цел – да предостави ефективно уеб-базирано решение за управление на медицински посещения, пациентски досиета и финансови взаимодействия между пациенти, лекари и администратори. За постигането на тази цел, системата следва да отговаря на редица функционални и нефункционални изисквания.

Проектът е проектиран така, че да предоставя максимално добро потребителско изживяване, съобразено с ролевата динамика. Всеки тип потребител получава персонализиран достъп до функциите, които му принадлежат. За реализиране на тази концепция, в системата са дефинирани няколко основни задачи:

* Създаване на сигурна база данни за съхранение на информация за пациенти, лекари, медицински прегледи и предписания.
* Реализация на ролево-базиран контрол на достъпа с помощта на ASP.NET Identity.
* Осигуряване на възможност лекарите да създават, редактират и преглеждат пациентски досиета.
* Създаване на система за известия и комуникация, включително имейл нотификации за предстоящи прегледи и потвърждения.
* Разработване на механизъм за създаване и преглед на медицински сметки от страна на лекарите и пациентите.
* Тестване и валидация на всички основни модули.

Проучване на съществуващи решения показва, че повечето налични платформи са или твърде сложни за крайния потребител, или прекалено скъпи за внедряване в малки здравни практики. Проектът цели да предложи лека, функционална и достъпна алтернатива, съчетаваща необходимите елементи на една модерна медицинска система.

След като тези цели и изисквания са ясно дефинирани, изборът на използвани технологии играе ключова роля за реализирането на системата. В следващите секции се описват подробно компонентите, използвани за създаване на платформата:, които са доказали своята ефективност при разработването на сигурни и интуитивни приложения. Всеки използван инструмент е подбран с конкретна цел – да подпомогне определена част от системата, като осигури стабилност, поддръжка и лесно надграждане.

Технологии, използвани за изграждане на приложението:

**2.1. ASP.NET Core**

ASP.NET Core представлява модерен, кросплатформен framework за създаване на уеб приложения. В проекта се използва неговата MVC (Model-View-Controller) структура, която разделя приложението на логически компоненти, улеснявайки както разработката, така и поддръжката. Благодарение на ASP.NET Core, системата работи с висока производителност, сигурност и добра модулност.

**2.2. Entity Framework Core (EF Core)**

EF Core е ORM (Object-Relational Mapping) библиотека, която улеснява достъпа до базата от данни чрез използване на .NET обекти. В проекта EF Core се използва за дефиниране на модели (Entities), създаване на миграции и реализиране на CRUD операции (Create, Read, Update, Delete) по структуриран и сигурен начин. Чрез Fluent API се дефинират релации като one-to-many и many-to-many между пациентите, лекарите и посещенията.

**2.3. Microsoft SQL Server**

SQL Server служи като основна релационна база от данни в системата. Той осигурява сигурно и ефективно съхранение на чувствителни медицински и финансови данни. Използването на SQL Server в комбинация с EF Core позволява бързи заявки, лесни миграции и поддържане на целостта на информацията.

**2.4. Razor Pages и Bootstrap**

Потребителският интерфейс на приложението е реализиран с помощта на Razor Pages и библиотеката Bootstrap. Това осигурява адаптивен дизайн, който работи добре както на настолни компютри, така и на мобилни устройства. Bootstrap предоставя компоненти като таблици, бутони, модали и формуляри, които допринасят за добрата визуална структура и интуитивното потребителско изживяване.

**2.5. ASP.NET Identity**

За удостоверяване и ролево базирана авторизация се използва ASP.NET Identity. Всеки потребител получава роля (Patient, Doctor, Admin), като системата проверява достъпа до всеки контролер и метод чрез атрибути като [Authorize(Roles = "Doctor")]. Това гарантира, че пациентите не могат да достъпват административни функции, а лекарите – данни, които не са им назначени.

**2.6. JavaScript, Chart.js и Fetch API**

Клиентската логика се допълва от JavaScript. Използван е Chart.js за визуализиране на данни като брой прегледи по дни. Чрез Fetch API се извличат динамично данни от сървъра без презареждане на страницата – например списък с посещения за избрана дата.

Всички използвани технологии работят съвместно, за да осигурят функционална, сигурна и мащабируема система, отговаряща на съвременните нужди на здравния сектор.

**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.

- Избиране на роля: Потребителите могат да избират роля спрямо желанието на услуги

- 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.

- Сметки за посещенията: Лекарите изпращат сметките към съответните пациенти

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

**3.1. 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.

**3.2. 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.

**3.3. Описание на базата данни**

Базата данни съдържа следните основни таблици:

* **Patients** (Пациенти)
* **Appointments(Предварителни срещи)**
* **Doctors** (Лекари)
* **Visits** (Посещения)
* **Billings(Сметки за посещенията)**
* **Users** (Потребители) - AspNetUsers
* **PatientDoctor** (Много-към-много връзка между пациенти и лекари)

**3.4. 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.

**3.5. Тестване и валидация**

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

**3.6. Роли и разрешения**

**3.6.1 Пациенти**

Може да:

* Регистрирайте се и влезте.
* Вижте техните медицински досиета.
* Запазете час при лекар.
* Преглед на планирани срещи.
* Изпращате плащания за всяко посещение

Не може:

* Редактиране или изтриване на срещи на други потребители.
* Променете графиците на лекарите.
* Да редактирате своите срещи или да ги изтривате
* Да управлявате плащанията на своите или на други посещения

**3.6.2 Лекари**

Може да:

* Регистрирайте се и влезте.
* Управление на досиетата на пациентите.
* Планирайте и редактирайте срещи.
* Вижте назначените им пациенти.
* Изпратете сметка на пациента и получете плащането

Не може:

* Достъп до пациенти на други лекари.
* Изтриване на срещи без разрешение.

**3.6.3 Администратори**

Може да:

* Управлява както на пациенти, така и на лекари.
* Разпределя роли.
* Наблюдава управлението на срещите.

**4. Структура на проекта**

Проектът се състои от следните основни компоненти:

**4.1 Контролери**

**Контролерите управляват HTTP заявки и взаимодействат с услугите, за да върнат подходящи отговори на изгледите.**

**4.1.1 AppointmentController**

**Обработва действия, свързани със срещи, включително:**

* **GET Create() - Показва страницата за създаване на среща.**
* **POST Create(CreateAppointmentViewModel model) - Обработва изпращанията на формуляри за създаване на среща.**
* **GET Manage() - Показва срещи за лекари и пациенти.**

**4.1.2 AccountController**

**Управлява удостоверяването и оторизацията:**

* **Register(): Обработва регистрацията на потребителя.**
* **Login(): Удостоверява потребителите.**
* **Logout(): Прекратява потребителските сесии.**

**4.1.3 HomeController**

**Обработва общи страници като началната страница и таблото за управление.**

**4.1.4 DoctorController**

**Управлява срещи, медицински досиета и добавяне на потребители като лекари:**

**• Become() - Потребителят може да стане лекар след регистрация**

**• ViewMyPatients() - Лекарят може да види всички свои пациенти, които са записани при него**

**• Dashboard() - Това е таблото с всички документирани срещи според пациента**

**• Add() - Създава медицинско досие на срещите, които докторът е изследвал според деня на изпълнението**

**• ViewPatientRecord() - Подробен преглед на досието на неговия пациент**

**• Edit() - Редактиране на запис на пациент**

**• Delete() - Изтриване на запис на пациент**

**4.1.5 PatientController**

**Може да преглежда медицинско досие, написано от съответния лекар и да назначава лекари според тяхната специализация и нужди. Също да получава сметката за всяка среща със своя лекар и да предаде това, което му трябва.**

**• Become(): Обработва регистрацията на потребителя. Потребителят става пациент след като се регистрира**

**• ViewMyRecords(): Може да преглежда досието на срещата със своя доктор**

**• AssignDoctor(): Може да назначава лекар според неговата специалност и града, в който работи**

**• EditBill(): Попълва необходимото поле, за да се изпълни неговата транзакция**

**4.1.6 BillingController**

**Всички пациенти, лекари и администратори използват системата за таксуване спрямо тяхното разрешение**

* **Create – лекарят създава и изпраща сметката на пациента**
* **View – администраторът преглежда необходимите данни, за да продължи плащането**

**4.1.6 AdminController**

**Този контролер се изпълнява само от администратора за управление на пациенти, лекари, досиета, сметки и посещения.**

* **AllBills() – управлява всички сметки**
* **PendingBills() – насочва се към страница, която показват всички сметки в очакване да бъдат удостоверени**
* **Review() – преглежда съответната сметка**
* **ManageDoctors() – управлява всички доктори**
* **ManagePatients() – управлява всички пациенти**
* **DeleteBill() – изтрива сметката**

**4.2 Services**

**Услугите действат като посредник между контролерите и хранилищата на бази данни. Опростяват действията на контролерите. Те могат да имат достъп до моделите и до базата данни**

**4.2.1 AppointmentService**

* **GetAppointmentsByPatientAsync(patientId): Извлича срещи за конкретен пациент.**
* **GetAppointmentsByDoctorAsync(doctorId): Извлича назначения за лекар.**
* **CreateAppointmentAsync(model): Записва нова среща в базата данни.**

**4.2.2 DoctorService**

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

* **Цел: Извлича уникалния идентификатор на обекта Doctor (като низ), използвайки асоциирания потребителски идентификатор на ASP.NET идентичност.**
* **Случай на използване: Когато трябва да свържете влезлия в момента потребител с неговия лекарски профил.**

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

* **Цел: Проверява дали съществува лекарски профил за дадения потребителски идентификатор.**
* **Случай на използване: Предотвратете дублиране на лекарски профили или се уверете, че потребителят е лекар, преди да разрешите достъп до определени функции.**

**Task<Guid> AddDoctorAsync(низ userId, модел BecomeDoctorModel);**

* **Предназначение: Създава нов лекарски профил, свързан с предоставения потребителски идентификатор, използвайки изпратените данни от формуляра (BecomeDoctorModel).**
* **Случай на използване: Когато потребител се регистрира като лекар и изпрати своите данни (име, специализация и т.н.).**

**Task<List<PatientViewModel>> GetPatientsByDoctorIdAsync(Guid doctorId);**

* **Цел: Връща списък с пациенти, назначени за посочения лекар.**
* **Случай на използване: За показване на всички пациенти под грижите на определен лекар, напр. в таблото на лекаря.4.2.4 PatientService**

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

* **Цел: Извлича ИД на обекта на пациента (като низ), свързан с дадения потребителски идентификатор.**
* **Случай на употреба: Свързване на влезли потребители към техния пациентски профил.**

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

* **Цел: Проверява дали съществува пациентски профил за посочения потребител.**
* **Случай на използване: Гарантира, че потребителите няма да се регистрират като пациенти няколко пъти.**

**Task<Guid> AddPatientAsync(низ userId, модел BecomePatientViewModel);**

* **Цел: Създава нов пациентски профил за потребителя с данни от BecomePatientViewModel.**
* **Случай на използване: Когато потребител се регистрира като пациент.**

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

* **Цел: Свързва пациент с лекар.**
* **Случай на използване: Използва се след регистрация или административен панел за назначаване на пациенти на лекари.**

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

* **Цел: Извлича обект на пациент чрез техния свързан потребителски идентификатор.**
* **Случай на използване: Полезно при зареждане на профила на пациента или записи за влязъл потребител.**

**4.2.3 MedicalRecordService**

**Task<int> AddPatientRecordAsync(PatientRecordViewModel модел, низ doctorUserId);**

* **Цел: Добавя ново медицинско досие, създадено от лекаря, идентифициран от doctorUserId.**
* **Случай на употреба: Когато лекар представи нова диагноза или рецепта след посещение при пациент.**

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

* **Цел: Връща списък с пациенти, назначени за лекар, форматиран за избор от падащо меню.**
* **Случай на употреба: Използва се в изгледи, когато лекар избира пациент за лечение.**

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

* **Цел: Извлича медицинско досие на един пациент по неговия уникален идентификатор.**
* **Случай на използване: Преглед или редактиране на конкретно медицинско досие.**

**Task<List<PatientRecordViewModel>>GetPatientRecordsByPatientIdAsync(GuidpatientId);**

* **Цел: Връща всички записи, свързани с конкретен пациент.**
* **Случай на употреба: За пациенти, преглеждащи собствената си медицинска история.**

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

* **Цел: Връща всички медицински досиета, създадени от определен лекар.**
* **Случай на употреба: лекар, който преглежда или управлява минали взаимодействия с пациенти.**

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

* **Цел: Извлича всички записи на пациенти в системата.**
* **Случай на използване: Полезно за достъп на ниво администратор или генериране на отчети.**

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

* **Цел: Актуализира съществуващо медицинско досие с помощта на данни от модела на формуляра.**
* **Случай на използване: Когато лекар редактира предварително създаден запис.**

**Task DeletePatientRecordAsync(int id);**

* **Цел: Изтрива запис на пациент по неговия ID.**
* **Случай на употреба: Администраторът или лекарят премахва остарял или неправилен запис.**

**4.2.4 PatientService**

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

* **Цел: Извлича пациент по неговото Id**

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

* **Цел: Проверява дали пациента съществува**

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

* **Цел: Добавя всеки нов потребител, който е избрал да стане пациент**

**Task<List<PatientViewModel>> GetAllPatientsAsync();**

* **Цел: Извлича всички регистрирани пациенти**

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

* **Цел: Пациентът назначва доктор и създава връзка в базата данни**

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

* **Цел: Взима вече създаден пациент от базата данни**

**4.2.5 BillingService**

**Task<BillViewModel?> GetBillViewByVisitIdAsync(int visitId**); извлича сметка, асоциирана с дадено медицинско посещение. Използва се при зареждане на формуляра за преглед или редакция.

**Task CreateBillAsync(BillViewModel model);** създава нова сметка в системата, на база въведени данни от лекаря. Методът валидира съществуването на посещението и пациентската връзка.

**Task<List<BillViewModel>> GetAllBillsAsync();** връща списък с всички създадени сметки в системата, използван от администраторския панел за наблюдение и филтриране.

**Task<IEnumerable<BillViewModel>> GetBillsPendingApprovalAsync();** извлича само тези сметки, които все още не са одобрени от администратор. Използва се за визуализиране в таблото на админа.

**Task ApproveBillAsync(int billId); о**тбелязва конкретна сметка като „Одобрена“, след преглед от администратор. Променя стойността на PaymentStatus.

**Task<BillViewModel?> GetBillByIdAsync(int billId);** връща конкретна сметка по нейното ID. Използва се в детайлен изглед или при редакция.

**Task UpdateBillAsync(BillViewModel model);** позволява на админ или пациент да актуализира съществуваща сметка – например да добави застрахователна информация или промени сума.

**Task<List<BillViewModel>> GetBillsByPatientUserIdAsync(string userId);** връща всички сметки, асоциирани с даден потребител (пациент), използвайки UserId от ASP.NET Identity.

**Task<IEnumerable<BillViewModel>> GetBillsForPatientAsync(Guid patientId);** извлича сметки според GUID идентификатор на пациента. Методът се използва както от администратори, така и от самия пациент.

**Task<bool> DeleteBillByIdAsync(int id); изтрива сметка от базата, ако администратор реши, че тя е невалидна.**

**Task MarkAsPaidAsync(int billId); отбелязва сметката като „платена“, което се използва при финализиране на процеса и евентуално известяване на лекуващия лекар.**

**4.3 Models**

**Това са моделите, които представят скелета на страниците като в тях се съдържа съответната информация. Те могат да се използват за**

**4.3.1 Entities (Database Models)**

**4.3.1.1 Appointment**

**Представя запис на среща**

**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**

**Записва и представя потребителя като пациент**

**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**

**Записва и представя потребителя като лекар**

**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**

**Сметката за посещенията**

**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**

**Връзката между лекар и пациент**

**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**

**Съхранява посещенията на пациента**

**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. Как компонентите работят заедно**

**1. Потребителите се регистрират/влизат чрез AccountController.**

**2. Пациентите резервират срещи с помощта на AppointmentController, като избират от наличните лекари.**

**3. Лекарите управляват срещи чрез изгледа Управление.**

**4. Данните се съхраняват в SQL Server с помощта на Entity Framework Core.**

**5. Упълномощаването гарантира базиран на роли контрол на достъпа с помощта на Identity Framework.**

**6. Автоматизирани тестове, изпълнявани със Selenium & NUnit.**

Глава 4

**Ръководство на потребителя**

**4.1. Достъп и роля на потребител**

**След вписване в системата, потребителят автоматично се насочва към интерфейс, съобразен с неговата роля. Това е реализирано чрез ASP.NET Identity и ролево базирана проверка на достъп. Всеки тип потребител получава различна начална точка и възможности в зависимост от това дали е пациент, лекар или администратор.**

* **Пациентите биват насочвани към своето персонализирано табло, където имат достъп до своята здравна информация и опции за търсене на лекари.**
* **Лекарите се пренасочват към табло с функционалности за управление на пациенти, добавяне на медицински досиета и създаване на сметки.**
* **Администраторите разполагат с административен панел, който им предоставя контрол върху потребителите, ролите и финансовите записи.**

**4.2. Поток за лекари**

**Лекарите са основни участници в процеса на създаване и поддържане на медицинската документация. След успешно вписване, лекарят преминава през следните основни функционални стъпки:**

1. **Преглед на таблото – Лекарят вижда прегледна информация за своите пациенти и брой посещения по дни, седмици и месеци. Представена е визуална статистика чрез Chart.js.**
2. **Управление на пациенти – Системата предоставя достъп до списък с пациенти, които са назначени към лекаря. Всеки пациент е достъпен чрез детайли за контакт, медицинска история и възможност за редакция.**
3. **Създаване на медицински досиета – Лекарят избира пациент от списък и попълва информация за дата на прегледа, диагноза, причини, предписания и бележки. След записване, досието става достъпно за пациента и администратора.**
4. **Генериране на сметки – След записване на медицинско досие, се появява бутон за създаване на сметка. Лекарят попълва сума, уточнява специализация и допълва застрахователна информация. Сметката се изпраща директно към пациента.**
5. **Преглед на историята и статистика – Чрез Dashboard лекарят има достъп до посещенията, филтрирани по дата. При натискане на конкретен ден се зарежда списък с досиета за този ден.**

**4.3. Поток за администратори**

**Администраторите играят ключова роля в поддръжката на системата. Те разполагат с контрол върху потребителите, финансовата информация и ролите. Потокът на работа за админ е следният:**

1. **Преглед на потребители – Админът има достъп до пълния списък от потребители в системата, независимо дали са пациенти, лекари или други администратори. Това улеснява мониторинга и координацията между ролите.**
2. **Управление на роли – При нова регистрация, администраторът може да присвои необходимата роля на даден потребител (лекар или пациент) чрез административния интерфейс.**
3. **Преглед и одобрение на сметки – Всички създадени сметки от лекари се съхраняват в базата данни със статус. Админът може да ги филтрира по статус, да редактира детайли и да ги одобри за изпращане към съответния пациент.**
4. **Контрол върху досиета – Освен сметките, администраторът има достъп до всички медицински записи. Това гарантира, че няма злоупотреби и всички взаимодействия са проверими.**

**4.4. Поток за пациенти**

**Пациентите имат достъп до ограничен, но достатъчен набор от функционалности, които гарантират пълноценна проследимост на тяхната медицинска история и комуникация с лекари. Стъпките са следните:**

1. **Достъп до таблото – След вход, пациентът вижда своя Dashboard, където са изобразени последни посещения, предстоящи прегледи и получени сметки.**
2. **Назначаване на лекар – Пациентът може да използва търсачка по специалност и град, за да избере подходящ лекар. Интерфейсът включва карти с данни за специализация, местоположение и контакт.**
3. **Запазване на час – След избора на лекар, пациентът може да заяви преглед чрез формуляр, в който посочва дата, причина и предпочитано време. Това създава „заявка“, видима за лекаря.**
4. **Преглед на медицински записи – Пациентът има достъп до всички свои записи, включително диагнози, предписания и бележки от прегледи. Всичко е организирано хронологично.**
5. **Работа със сметки – Пациентът може да преглежда сметките, изпратени от лекарите. Попълва информация за застраховка и потвърждава. Сметката се подава за проверка към администратора.**

**4.5. Сигурност и достъп**

**Системата използва ASP.NET Identity за удостоверяване и авторизация. При вход се използват уникални идентификатори и криптирани пароли. Ролевата проверка се извършва при всяко действие чрез атрибути като [Authorize(Roles = "Doctor")]. Всички контролери са защитени и няма възможност за неоторизиран достъп до чувствителни данни.**

**Данните се съхраняват в Microsoft SQL Server чрез Entity Framework Core. Използват се миграции за поддържане на синхронизация между базата данни и моделите. Всяка операция върху чувствителна информация е проследима.**

**4.6. Примерен сценарий на потребителски поток**

**Пациент влиза в системата → Назначава лекар по специалност → Заявява преглед → Лекарят провежда преглед → Създава се медицинско досие → Генерира се сметка → Пациентът я попълва и подава → Администраторът преглежда и одобрява сметката → Завършен цикъл на взаимодействие.**

**Този пример илюстрира пълната интеграция между потребителските роли и данните в системата и демонстрира как всяка роля взаимодейства в рамките на бизнес логиката.**

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**