**Министерство цифрового развития, связи и массовых коммуникаций РФ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Кафедра ПМиК

Зав. кафедрой

\_\_\_\_\_\_\_\_\_\_\_ / /

### **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

по направлению 09.03.01 «Информатика и вычислительная техника»

профиль Программное обеспечение средств вычислительной техники и автоматизированных систем

РАЗРАБОТКА МЕДИЦИНСКОГО ВЕБ-ПРИЛОЖЕНИЯ СОПРОВОЖДЕНИЯ ЛЕЧЕНИЯ ПАЦИЕНТА

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**/ /

**Институт информатики и вычислительной техники**

**Группа**

**Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**/ /

Содержание

[Введение 3](#_Toc170068102)

[1 Обзор предметной области 5](#_Toc170068103)

[2 Постановка задачи 7](#_Toc170068104)

[3 ВЫБОР СРЕДСТВ РАЗРАБОТКИ 9](#_Toc170068105)

[3.1 Языки программирования 9](#_Toc170068106)

[3.2 Платформа 9](#_Toc170068107)

[3.3 Перечень дополнительных инструментов и библиотек 11](#_Toc170068108)

[3.4 Визуализация базы данных 11](#_Toc170068109)

[3.5 Структура архитектуры приложения 13](#_Toc170068110)

[3.6 Инструменты для работы с запросами 16](#_Toc170068111)

[4 разработка приложения 19](#_Toc170068112)

[4.1 Модели 19](#_Toc170068113)

[4.2 Контроллеры 22](#_Toc170068114)

[5 Руководство пользователя 31](#_Toc170068115)

[Заключение 41](#_Toc170068116)

[список использованных источников и литературы 43](#_Toc170068117)

[Приложение А 44](#_Toc170068118)

Введение

Современное развитие технологий и цифровизация общества приводят к появлению новых сервисов и платформ, направленных на улучшение различных сфер жизни, включая здравоохранение. Одним из таких направлений является создание медицинских веб-приложений, которые позволяют пациентам самостоятельно отслеживать свое здоровье и получать актуальную информацию о состоянии своего организма. Такие приложения значительно повышают качество медицинского обслуживания и удобство для пользователей.

На сегодняшний день взаимодействие между пациентами и медицинскими учреждениями становится все более интенсивным и частым, что сопровождается увеличением спроса на качественные и оперативные медицинские сервисы. С развитием цифровых технологий и изменением способов взаимодействия между пациентами и медицинскими специалистами появляются новые возможности для оптимизации и упрощения этого процесса. Создание веб-приложения для самостоятельного мониторинга здоровья пациента — надежного и удобного сервиса — является актуальным и важным предложением на рынке медицинских услуг.

Данный сервис может способствовать улучшению качества медицинского обслуживания. Путем предоставления доступа к актуальной информации о состоянии здоровья, планам лечения и рекомендациям врачей, такие сервисы позволяют пациентам более эффективно управлять своим здоровьем и соблюдать предписания специалистов. Это особенно важно в условиях растущей нагрузки на медицинские учреждения и стремления к снижению расходов на здравоохранение. Использование таких приложений помогает снизить количество ненужных визитов к врачу, оптимизировать расписание приема пациентов и улучшить общую эффективность работы медицинских учреждений.

Разработка такого программного обеспечения требует внедрения инновационных технологий, включая системы управления медицинскими записями, алгоритмы анализа медицинских данных и онлайн-платформы для взаимодействия между пациентами и медицинскими специалистами. Это способствует развитию технологической сферы здравоохранения и повышению конкурентоспособности медицинских учреждений.

Для решения вышеописанных задач идеально подходит медицинское веб-приложение, которое может привлечь широкий спектр пользователей: от людей, следящих за своим здоровьем, до пациентов с хроническими заболеваниями и медицинских специалистов. Пользователи смогут легко получить доступ к своим медицинским данным, рекомендациям врачей и напоминаниям о важных событиях, что способствует более эффективному управлению своим здоровьем. Разработка веб-приложения для самостоятельного мониторинга здоровья пациента представляет собой не только бизнес-возможность, но и социально значимый шаг в направлении улучшения качества медицинского обслуживания.

Благодаря использованию современных информационных технологий, такие сервисы обеспечивают прозрачность и удобство взаимодействия между пациентами и врачами, а также повышают безопасность и надежность процесса лечения. Это делает данную тему не только актуальной, но и перспективной для дальнейшего развития и внедрения в практику.

В рамках данной работы будут рассмотрены основные технологии и методы, используемые для создания подобных сервисов, а также проведен анализ потребностей пользователей и особенностей организации мониторинга здоровья пациентов. Результаты работы могут быть полезны как для предпринимателей, заинтересованных в создании подобных сервисов, так и для конечных пользователей, ищущих эффективные решения для управления своим здоровьем.

# Обзор предметной области

Текущее состояние рынка медицинских веб-приложений характеризуется рядом основных тенденций и характеристик, которые важно учитывать при разработке программного обеспечения для создания соответствующего сервиса. С увеличением числа пациентов и возрастанием значимости самоконтроля за здоровьем, растет и спрос на более эффективные и удобные способы мониторинга состояния здоровья. Пользователи все чаще обращают внимание на возможность самостоятельного отслеживания своего здоровья с помощью цифровых инструментов, что помогает улучшить качество жизни и повысить уровень медицинского обслуживания.

На рынке присутствует ряд крупных компаний, предоставляющих услуги медицинских веб-приложений и платформ, как в международном, так и в локальном масштабе. Это могут быть как крупные технологические компании, так и специализированные медицинские сервисы. Перед разработкой программного обеспечения необходимо изучить и проанализировать их стратегии и условия предоставления услуг.

Примеры крупных игроков на рынке включают такие компании, как "Apple Health", "Google Fit" и "MyChart", которые предоставляют широкий спектр услуг по мониторингу здоровья, включая хранение медицинских данных, отслеживание физической активности и предоставление рекомендаций по улучшению здоровья. Эти компании обладают значительным техническим оснащением и обширной пользовательской базой.

Крупные компании ориентированы на предоставление комплексных решений для мониторинга здоровья, что включает интеграцию с различными медицинскими устройствами и платформами. Однако, несмотря на обширные возможности, такие сервисы могут быть сложными в использовании и требовать дополнительных настроек для полной интеграции.

На рынке также существует множество небольших компаний и стартапов, предоставляющих специализированные решения для мониторинга здоровья. Эти компании могут предлагать более гибкие и персонализированные услуги, что особенно важно для пользователей с особыми медицинскими потребностями. Однако такие решения могут быть ограничены в функциональности и географии действия.

Множество онлайн-рынков и платформ, таких как "App Store" и "Google Play", предоставляют пользователям доступ к широкому спектру медицинских приложений. Однако множество приложений может сделать поиск подходящего инструмента трудоемким и затруднительным. Также существуют специализированные платформы и приложения, созданные специально для организации и мониторинга здоровья пациентов. Примеры таких сервисов включают "CareClinic" и "MyFitnessPal". Однако и у таких платформ есть свои ограничения, такие как высокие комиссии за премиум-услуги и ограниченная функциональность в бесплатных версиях.

Для того чтобы конкурировать с уже существующими решениями, необходимо иметь отличительные особенности. Предоставление персонализированных рекомендаций по текущему состоянию здоровья, возможность быстрого поиска ближайших медицинских учреждений или аптек, а также удобный интерфейс для пользователей. Таким образом, наша задача заключается в реализации приложения, которое позволит пациентам бесплатно и самостоятельно отслеживать свое здоровье, получать рекомендации и хранить все медицинские данные в одном месте.

Исходя из анализа, можно сделать вывод, что существует потребность в создании узкоспециализированной платформы, которая будет удовлетворять всем требованиям как со стороны пациентов, желающих самостоятельно управлять своим здоровьем, так и со стороны медицинских специалистов, которые могут использовать данные для улучшения качества медицинского обслуживания.

# Постановка задачи

Цель: разработать медицинское веб-приложение для сопровождения лечения пациентов, которое позволит пациентам самостоятельно отслеживать свое здоровье.

Задачи:

1. Реализовать хранение необходимой информации с использованием БД:
   * Добавление в БД информации о регистрирующемся в нашем приложении пациенте, включая его ФИО, возраст, город проживания, медицинскую историю, текущие назначения и результаты анализов.
   * Добавление в БД информации о записях на прием к врачу, которые пациент может вносить самостоятельно, включая дату, время и причину визита.
   * Хранение данных лекарственных средствах, включая возможность поиска лекарственных средств при конкретной болезни
   * Запросы в БД для получения необходимой информации, такой как история болезни пациента, результаты анализов и расписание визитов к врачу.
2. Создание регистрации и авторизации на сайте:
   * Реализация функционала регистрации и авторизации пользователей.
3. Создание пользовательского интерфейса:
   * Разработка удобного и интуитивно понятного интерфейса для пользователей.
   * Внедрение интерфейсов для получения информации о лекарственных средствах, нахождении ближайших медицинских учреждений и аптек, оценки состояния здоровья.
   * Внедрение интерфейса для различных заметок (визит к врачу, приеме лекарственных средств и т.п).
4. Разработка функционала для отслеживания здоровья:
   * Создание функций для ввода и отслеживания симптомов, показателей здоровья (таких как вес, давление, уровень сахара в крови)
5. Тестирование и оптимизация:
   * Проведение тестирования функционала приложения на различных устройствах и в различных браузерах.
   * Оптимизация производительности приложения для быстрой загрузки и отзывчивости интерфейса.
   * Проверка полей на соответствие критериям верификации.

# ВЫБОР СРЕДСТВ РАЗРАБОТКИ

## Языки программирования

Язык программирования: Для разработки используется язык программирования C# вместе с фреймворком ASP.NET Core. C# — это мощный и многогранный язык, предоставляющий разработчикам обширный набор инструментов для создания веб-приложений. ASP.NET Core, в свою очередь, представляет собой современный и гибкий фреймворк, который обеспечивает высокую производительность и масштабируемость веб-приложений, что делает его идеальным для создания сложных и надежных систем.

Интегрированная среда разработки (IDE): Для разработки используется Visual Studio. Эта интегрированная среда разработки предоставляет полный набор инструментов, необходимых для работы с C# и ASP.NET Core, включая возможности отладки, автоматической сборки и развертывания приложений. Visual Studio поддерживает разработчиков на всех этапах создания программного обеспечения, обеспечивая удобную и эффективную работу.

База данных: Для хранения данных используется SQL Server, управляемый с помощью SQL Server Management Studio (SSMS). SQL Server известен своей высокой производительностью и масштабируемостью, а также широким набором функций для управления данными, таких как безопасность, восстановление данных и аналитика. SSMS обеспечивает удобное и эффективное администрирование баз данных, предоставляя графический интерфейс для выполнения запросов, управления серверами и базами данных.

## Платформа

Visual Studio

Для разработки программного обеспечения была выбрана платформа «Visual Studio».

«Visual Studio» представляет собой интегрированную среду разработки (IDE) от компании Microsoft, которая предоставляет обширный набор инструментов для создания, отладки и развертывания программного обеспечения на множестве языков программирования.

Выбор пал на эту платформу, так как она удовлетворяет всем необходимым требованиям для выполнения проекта.

Основные преимущества данной платформы включают:

* Обилие доступных обучающих материалов, руководств и видеоуроков, которые можно найти бесплатно.
* Широкая поддержка множества языков программирования, таких как C#, C++, Python, JavaScript и другие.
* Интеграция с платформой Azure для разработки облачных приложений.
* Встроенные инструменты для тестирования и отладки, включая профилировщик производительности и пошаговую отладку.
* Поддержка разработки для различных платформ, включая Windows, macOS, iOS и Android.
* «Visual Studio» позволяет разработчикам создавать мощные приложения с использованием облачных технологий и интеграцией различных сервисов, что делает процесс разработки более гибким и эффективным.

SQL Server Management Studio (SSMS)

Для работы с базами данных была выбрана платформа «SQL Server Management Studio» (SSMS).

«SQL Server Management Studio» представляет собой интегрированную среду от компании Microsoft, предназначенную для управления и администрирования Microsoft SQL Server. Она предоставляет полный набор инструментов для настройки, управления и администрирования баз данных.

Выбор пал на эту платформу, так как она соответствует всем необходимым требованиям для выполнения проекта.

Основные преимущества данной платформы включают:

* Обширная документация и бесплатные обучающие материалы.
* Мощные инструменты для управления базами данных, включая графический интерфейс для написания запросов и выполнения скриптов.
* Возможность подключения и управления множественными серверами и базами данных из одного интерфейса.
* Интеграция с инструментами аналитики и отчетности, такими как SQL Server Reporting Services (SSRS) и SQL Server Analysis Services (SSAS).
* Поддержка различных типов данных и работа с большими объемами данных.
* SSMS позволяет администраторам и разработчикам баз данных эффективно управлять данными, оптимизировать производительность запросов и обеспечивать безопасность и целостность данных в облачных и локальных средах.

## Перечень дополнительных инструментов и библиотек

* Entity Framework Core: Этот ORM (объектно-реляционное отображение) используется для упрощения работы с базами данных, позволяя разработчикам взаимодействовать с данными в виде объектов, что ускоряет разработку и уменьшает количество ошибок.
* ASP.NET Core Identity: Применяется для реализации механизмов аутентификации и авторизации пользователей, обеспечивая надежную безопасность и управление доступом.
* ASP.NET Core MVC: Используется для создания веб-интерфейсов и управления HTTP-запросами, обеспечивая четкое разделение логики и представления.
* Bootstrap: Набор инструментов для создания адаптивных и привлекательных пользовательских интерфейсов, ускоряя процесс разработки и улучшая пользовательский опыт.
* Яндекс карты API и Bing maps API: Предоставляют возможности для интеграции интерактивных карт в веб-приложения, что позволяет отображать географическую информацию и улучшать функциональность пользовательских интерфейсов.

## Визуализация базы данных

SQL Server Management Studio (SSMS) предоставляет мощный инструмент для визуализации структуры базы данных под названием "Диаграммы базы данных". Этот инструмент позволяет создавать графические представления, показывающие зависимости и взаимосвязи между таблицами в одной базе данных. Диаграмма описывающая созданную базу данных показана на рисунке 3.1.

### Основные возможности и преимущества:

* **Визуализация структуры базы данных**: Диаграммы предоставляют наглядное представление связей между таблицами, что особенно полезно для работы со сложными базами данных, где существует множество взаимосвязанных таблиц. Это помогает лучше понять архитектуру и организацию данных.
* **Автоматическое обнаружение связей**: SSMS автоматически отображает внешние ключи и связи между таблицами. Это позволяет пользователям быстро понять, как данные организованы и как таблицы взаимодействуют друг с другом, облегчая анализ структуры базы данных.
* **Удобное редактирование**: Пользователи могут легко добавлять и удалять таблицы с диаграммы, а также изменять связи между таблицами непосредственно в интерфейсе диаграммы. Все внесенные изменения автоматически применяются к структуре базы данных, что упрощает процесс управления базой данных.
* **Создание новых связей и изменений структуры**: Диаграммы не только отображают существующие связи, но и позволяют добавлять новые связи, такие как внешние ключи, а также изменять структуру таблиц. Это упрощает процесс модификации базы данных и улучшает гибкость управления данными.
* **Документирование базы данных**: Диаграммы можно сохранять для последующего использования в документации. Это облегчает передачу знаний между разработчиками и администраторами, предоставляя четкое и понятное представление структуры базы данных.
* **Печать и экспорт диаграмм**: Диаграммы можно распечатать или экспортировать в различные форматы, такие как изображения или PDF-документы. Это удобно для создания презентаций и отчетов, а также для документирования и обмена информацией о структуре базы данных.

Эти особенности делают "Диаграммы базы данных" в SSMS незаменимым инструментом для визуализации, редактирования и документирования структуры баз данных, улучшая понимание и управление данными для разработчиков и администраторов.

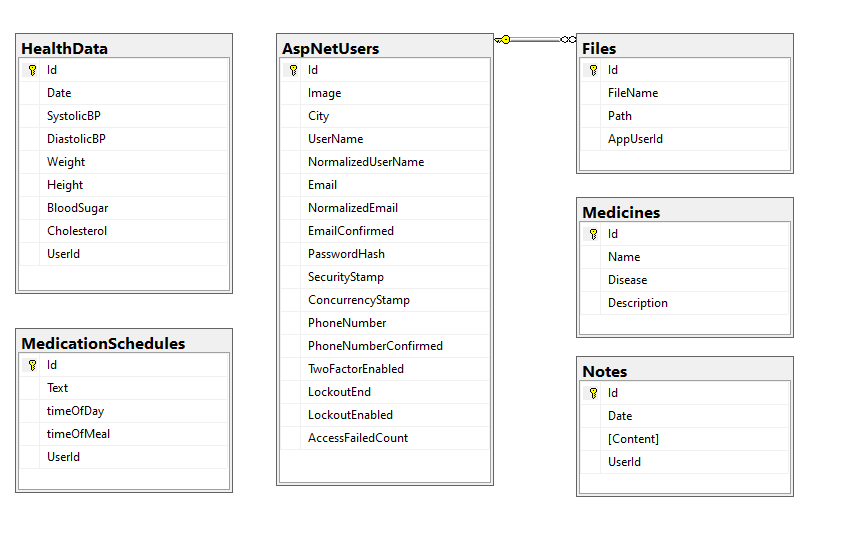


Рисунок 3.1 – Схема базы данных

## Структура архитектуры приложения

Для реализации структуры приложения был использован шаблон MVC (Model-View-Controller) в ASP.NET Core (Рисунок 3.2). Этот архитектурный паттерн разделяет приложение на три основных компонента: модели, представления и контроллеры, что способствует упрощению разработки, тестирования и сопровождения за счет четкого разделения ответственности.

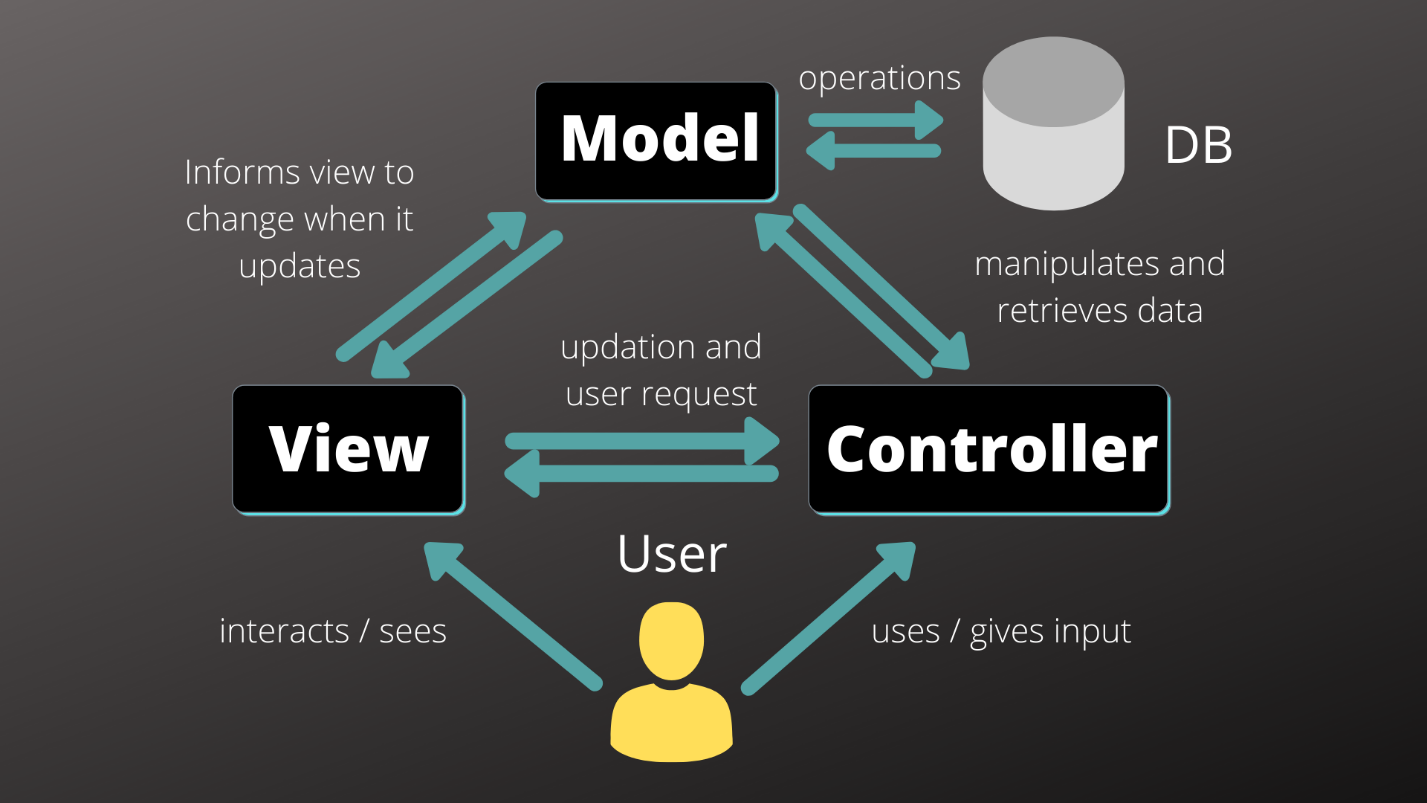


Рисунок 3.2 – Схема модели MVC

### **Основные компоненты MVC**

1. Model (Модель):
   * Описание данных и бизнес-логики: Модель отвечает за представление данных, реализацию бизнес-логики и правила валидации. Она определяет, как данные хранятся, изменяются и обрабатываются.
   * Взаимодействие с базой данных: В ASP.NET Core модели часто работают в связке с Entity Framework для взаимодействия с базой данных. Модели представляют таблицы базы данных и содержат свойства, соответствующие колонкам этих таблиц.
   * Примеры: Классы, представляющие сущности, такие как пользователь, продукт, заказ и т.д.
2. View (Представление):
   * Отображение данных: Представление отвечает за отображение данных пользователю. Оно получает данные от контроллера и рендерит их в HTML для отображения в браузере.
   * Шаблоны: Представления обычно создаются с использованием синтаксиса Razor, который позволяет смешивать HTML и C# код.
   * Отделение логики отображения: В представлениях не должна находиться бизнес-логика; их основная задача — отображение данных, полученных от контроллера.
   * Примеры: Страницы, показывающие списки продуктов, формы для ввода данных и т.д.
3. Controller (Контроллер):
   * Обработка пользовательских запросов: Контроллер принимает входящие HTTP-запросы, обрабатывает их и возвращает соответствующий ответ, будь то представление или данные в формате JSON.
   * Взаимодействие с моделью и представлением: Контроллер запрашивает данные у модели, обрабатывает их (если необходимо) и передает результат представлению для отображения.
   * Управление потоками данных: Контроллер управляет потоками данных между моделью и представлением, обеспечивая логическое разделение процессов обработки данных и их отображения.
   * Примеры: Контроллеры для управления пользователями, заказами, продуктами и т.д.

### **Преимущества использования шаблона MVC**

**Разделение обязанностей**: Четкое разделение обязанностей между моделью, представлением и контроллером облегчает разработку и поддержку кода. Это позволяет разработчикам параллельно работать над различными компонентами приложения, что повышает производительность команды.

**Тестируемость**: Отделение логики обработки данных (контроллеры и модели) от логики отображения (представления) упрощает тестирование приложения. Это способствует созданию модульных тестов для бизнес-логики и контроллеров, что повышает качество и надежность кода.

**Повторное использование кода**: Модели и контроллеры могут быть повторно использованы в различных представлениях. Это снижает количество дублирующегося кода и облегчает его поддержку, делая разработку более эффективной.

**Четкая структура**: Шаблон MVC способствует созданию организованной и четко структурированной архитектуры проекта. Это упрощает навигацию по проекту и делает код более понятным для новых разработчиков, что ускоряет их адаптацию.

### **Принцип работы в контексте ASP.NET Core**

1. **Запрос пользователя**: Пользователь отправляет HTTP-запрос, например, при переходе по URL или отправке формы.
2. **Роутинг**: ASP.NET Core использует систему маршрутизации для определения, какой контроллер и какое действие этого контроллера должны обработать данный запрос.
3. **Обработка контроллером**: Контроллер получает запрос, взаимодействует с моделью для получения или обработки данных и затем выбирает соответствующее представление для возврата пользователю.
4. **Рендеринг представления**: Представление получает данные от контроллера и рендерит их в HTML для отображения в браузере пользователя.
5. **Ответ пользователю**: Сформированный HTML-код возвращается пользователю в виде HTTP-ответа, который отображается в браузере.

Таким образом, использование шаблона MVC в ASP.NET Core обеспечивает структурированную и гибкую архитектуру для создания веб-приложений, что упрощает разработку, тестирование и поддержку.

## Инструменты для работы с запросами

Для удобной работы с SQL-запросами была выбрана технология Entity Framework Core (EF Core). EF Core — это объектно-реляционный диспетчер (ORM) для .NET, который упрощает взаимодействие приложений с базами данных, предоставляя разработчикам инструмент для работы с данными в виде объектов .NET и абстрагируя сложности взаимодействия с реляционными базами данных.

### Основные преимущества EF Core:

1. **ORM (Object-Relational Mapping)**:
   * EF Core позволяет разработчикам работать с базами данных через объекты .NET, устраняя необходимость в написании большинства SQL-запросов вручную.
2. **Кроссплатформенность**:
   * EF Core поддерживает работу на различных операционных системах, таких как Windows, macOS и Linux.
3. **Поддержка различных баз данных**:
   * EF Core совместим с множеством баз данных, включая Microsoft SQL Server, SQLite, PostgreSQL, MySQL и другие.
4. **Линейный запрос (LINQ)**:
   * EF Core интегрируется с LINQ, позволяя писать запросы к базе данных на C#, которые затем преобразуются в SQL и выполняются на сервере базы данных.
5. **Модели данных**:
   * Разработчики могут определять модели данных, представляющие структуру таблиц базы данных, используя классы .NET, которые можно настроить с помощью Fluent API или атрибутов.
6. **Миграции**:
   * EF Core поддерживает миграции, которые позволяют управлять изменениями схемы базы данных, обеспечивая создание, обновление и откат изменений.
7. **Трекер изменений**:
   * EF Core отслеживает изменения в объектах, загруженных из базы данных, и автоматически генерирует SQL-команды для их обновления.
8. **Загрузка связанных данных**:
   * EF Core поддерживает ленивую, явную и жадную загрузку связанных данных, что позволяет управлять процессом загрузки данных из базы.
9. **Работа с транзакциями**:
   * EF Core позволяет выполнять несколько операций с базой данных в рамках транзакции, обеспечивая целостность и согласованность данных.
10. **Конфигурация и расширяемость**:
    * EF Core предоставляет множество возможностей для настройки и расширения, включая настраиваемые соглашения и перехватчики запросов.
11. **Поддержка не-реляционных баз данных**:
    * EF Core также может работать с не-реляционными источниками данных, такими как Azure Cosmos DB.

Entity Framework Core использовался для работы с DBContext, который является центральной частью взаимодействия с базой данных в приложениях ASP.NET Core. DBContext обеспечивает удобный и мощный API для выполнения большинства задач, связанных с доступом к данным, таких как CRUD-операции (создание, чтение, обновление и удаление).

### **Основные функции DBContext:**

1. **Управление подключением к базе данных**:
   * DBContext инкапсулирует логику работы с подключением к базе данных, управляя открытием и закрытием соединений.
2. **Отслеживание изменений**:
   * DBContext отслеживает изменения в объектах, загруженных из базы данных, и синхронизирует эти изменения при сохранении.
3. **Выполнение запросов**:
   * С помощью DBContext можно выполнять LINQ-запросы для выборки данных из базы данных, преобразовывая их в SQL и выполняя на сервере.
4. **Сохранение изменений**:
   * DBContext предоставляет метод для сохранения изменений, сделанных в объектах, обратно в базу данных, генерируя соответствующие SQL-команды.
5. **Управление транзакциями**:
   * DBContext может управлять транзакциями, позволяя выполнять несколько операций как одну атомарную операцию.
6. **Конфигурация моделей**:
   * В DBContext можно настраивать модели данных с помощью Fluent API или атрибутов, задавая правила для таблиц, столбцов, связей и ограничений.
7. **Миграции**:
   * DBContext используется для применения миграций, обновляющих схему базы данных в соответствии с изменениями моделей данных.
8. **Работа с наборами данных (DbSet)**:
   * DBContext предоставляет коллекции объектов, представляющих таблицы базы данных, через свойства типа DbSet, что облегчает управление записями в таблицах.
9. **Инъекция зависимостей**:
   * DBContext легко интегрируется с системой внедрения зависимостей ASP.NET Core, что позволяет управлять его жизненным циклом и зависимостями.

Использование EF Core и DBContext в ASP.NET Core обеспечивает мощную и гибкую архитектуру для работы с данными, упрощая разработку, тестирование и поддержку приложений.

# разработка приложения

## Модели

В модели MVC компонент Models выполняет ключевую роль в управлении данными и бизнес-логикой приложения. Для нашего приложения были использованы модели, показанные на рисунке 4.1.

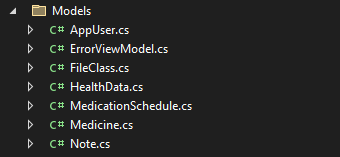


Рисунок 4.1 – Список моделей

#### **Модель AppUser**

**Назначение:** Модель AppUser представляет пользователя в системе и расширяет базовый класс IdentityUser, предоставляемый ASP.NET Identity для управления пользователями.

**Поля:**

* **Image (string?):** Путь к изображению пользователя. Может быть null, если изображение не загружено.
* **City (string):** Город проживания пользователя.
* **FileRecords (ICollection<FileClass>?):** Коллекция файлов, связанных с пользователем. Может быть null, если файлов нет.

**Зависимости:**

* Наследуется от IdentityUser, что обеспечивает стандартные функции управления пользователями (такие как логин, пароль, электронная почта и т.д.).
* Связь один-ко-многим с моделью FileClass.

#### **Модель FileClass**

**Назначение:** Модель FileClass представляет файлы, связанные с пользователем.

**Поля:**

* **Id (int):** Уникальный идентификатор файла.
* **FileName (string):** Имя файла.
* **Path (string):** Путь к файлу на сервере.
* **AppUserId (string):** Идентификатор пользователя, которому принадлежит файл.

**Зависимости:**

* Связь с моделью AppUser, где каждый файл принадлежит одному пользователю.

#### **Модель HealthData**

**Назначение:** Модель HealthData представляет данные о здоровье пользователя.

**Поля:**

* **Id (int):** Уникальный идентификатор записи.
* **Date (DateTime):** Дата записи данных.
* **SystolicBP (int):** Систолическое артериальное давление.
* **DiastolicBP (int):** Диастолическое артериальное давление.
* **Weight (double):** Вес пользователя.
* **Height (double):** Рост пользователя.
* **BloodSugar (double):** Уровень сахара в крови.
* **Cholesterol (double):** Уровень холестерина в крови.
* **UserId (string):** Идентификатор пользователя, которому принадлежат данные.

**Зависимости:**

* Связь с моделью AppUser, где каждая запись данных о здоровье принадлежит одному пользователю.

#### **Модель MedicationSchedule**

**Назначение:** Модель MedicationSchedule представляет расписание приема лекарств для пользователя.

**Поля:**

* **Id (int):** Уникальный идентификатор записи.
* **Text (string):** Описание расписания.
* **timeOfDay (TimeOfDay):** Время суток для приема лекарства (перечисление TimeOfDay).
* **timeOfMeal (TimeOfMeal):** Время приема пищи для приема лекарства (перечисление TimeOfMeal).
* **UserId (string):** Идентификатор пользователя, которому принадлежит расписание.

**Зависимости:**

* Связь с моделью AppUser, где каждое расписание приема лекарств принадлежит одному пользователю.

#### **Модель Medicine**

**Назначение:** Модель Medicine представляет лекарство.

**Поля:**

* **Id (int):** Уникальный идентификатор лекарства.
* **Name (string):** Название лекарства.
* **Disease (string):** Заболевание, для лечения которого предназначено лекарство.
* **Description (string):** Описание лекарства.

**Зависимости:**

* Не имеет прямых зависимостей от других моделей.

#### **Модель Note**

**Назначение:** Модель Note представляет заметку пользователя.

**Поля:**

* **Id (int):** Уникальный идентификатор заметки.
* **Date (DateTime):** Дата заметки. Обязательное поле.
* **Content (string):** Содержимое заметки. Обязательное поле с ограничением длины до 500 символов.
* **UserId (string):** Идентификатор пользователя, которому принадлежит заметка. Обязательное поле.

**Зависимости:**

* Связь с моделью AppUser, где каждая заметка принадлежит одному пользователю.

## Контроллеры

В модели MVC контроллеры играют важную роль в координации работы приложения, управлении пользовательскими запросами и взаимодействии между моделями и представлениями. Контроллеры которые использовались в приложении можно увидеть на рисунке 4.1.

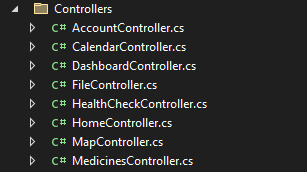


Рисунок 4.2 – Список контроллеров

### **Описание контроллера AccountController**

Контроллер AccountController отвечает за управление учетными записями пользователей, такими как регистрация, вход и выход из системы. Он использует сервисы UserManager и SignInManager для управления пользователями и их аутентификацией. Также в контроллере используется контекст базы данных ApplicationDbContext и среда хостинга IWebHostEnvironment для работы с изображениями пользователей.

#### Конструктор

* **AccountController(UserManager<AppUser> userManager, SignInManager<AppUser> signInManager, ApplicationDbContext context, IWebHostEnvironment webHostEnvironment):** Конструктор инициализирует зависимости, такие как UserManager, SignInManager, контекст базы данных и среду хостинга. Эти зависимости используются в различных методах контроллера для выполнения операций с пользователями.

#### Методы

1. **Login (GET)**
   * **Назначение:** Отображает форму для входа пользователя в систему.
   * **Описание:** Возвращает представление с моделью LoginViewModel, которая используется для ввода данных для входа (например, адрес электронной почты и пароль).
2. **Login (POST)**
   * **Назначение:** Обрабатывает данные, введенные пользователем в форму для входа.
   * **Описание:** Проверяет корректность данных. Если модель не валидна, возвращает ту же форму с ошибками. Иначе, находит пользователя по адресу электронной почты и проверяет его пароль. Если проверка успешна, аутентифицирует пользователя и перенаправляет его на главную страницу. В случае ошибки, показывает сообщение об ошибке и возвращает форму для входа.
3. **Register (GET)**
   * **Назначение:** Отображает форму для регистрации нового пользователя.
   * **Описание:** Возвращает представление с моделью RegisterViewModel, которая используется для ввода данных для регистрации (например, адрес электронной почты, имя пользователя, пароль, изображение и город).
4. **Register (POST)**
   * **Назначение:** Обрабатывает данные, введенные пользователем в форму для регистрации.
   * **Описание:** Проверяет корректность данных. Если модель не валидна, возвращает ту же форму с ошибками. Иначе, проверяет, существует ли пользователь с таким же адресом электронной почты. Если пользователь уже существует, показывает сообщение об ошибке. Если нет, сохраняет изображение пользователя (если оно было загружено), создает нового пользователя и сохраняет его в базе данных. После успешной регистрации, перенаправляет пользователя на страницу входа.
5. **Logout (POST)**
   * **Назначение:** Выполняет выход пользователя из системы.
   * **Описание:** Вызовет метод выхода у SignInManager для текущего пользователя и перенаправит его на страницу входа.

### **Описание контроллера CalendarController**

Контроллер CalendarController отвечает за управление записями (заметками) пользователей в календаре, такими как просмотр, создание, редактирование и удаление заметок. Он использует контекст базы данных ApplicationDbContext для выполнения операций с данными и IHttpContextAccessor для получения текущего пользователя.

#### Конструктор

* **CalendarController(ApplicationDbContext context, IHttpContextAccessor httpContextAccessor):** Конструктор инициализирует зависимости, такие как контекст базы данных и HTTP-контекст. Эти зависимости используются в различных методах контроллера для выполнения операций с заметками текущего пользователя.

#### Методы

1. **Index**
   * **Назначение:** Отображает список заметок текущего пользователя на указанную дату.
   * **Описание:** Получает заметки из базы данных и фильтрует их по дате и идентификатору пользователя. Если дата не указана, показывает все заметки пользователя. Возвращает представление с отфильтрованными заметками.
2. **Create (GET)**
   * **Назначение:** Отображает форму для создания новой заметки.
   * **Описание:** Устанавливает текущую дату и идентификатор пользователя во ViewBag для использования в представлении. Возвращает представление для создания новой заметки.
3. **Create (POST)**
   * **Назначение:** Обрабатывает данные, введенные пользователем в форму для создания новой заметки.
   * **Описание:** Проверяет корректность данных. Если модель валидна, добавляет новую заметку в базу данных и сохраняет изменения. Перенаправляет пользователя на страницу списка заметок. В случае ошибки возвращает форму для создания заметки с ошибками.
4. **Edit (GET)**
   * **Назначение:** Отображает форму для редактирования существующей заметки.
   * **Описание:** Получает идентификатор пользователя и проверяет наличие заметки с указанным идентификатором. Если заметка найдена, возвращает форму для редактирования. В противном случае возвращает страницу ошибки.
5. **Edit (POST)**
   * **Назначение:** Обрабатывает данные, введенные пользователем в форму для редактирования заметки.
   * **Описание:** Проверяет корректность данных и наличие заметки с указанным идентификатором. Если модель валидна, обновляет заметку в базе данных и сохраняет изменения. В случае ошибки возвращает форму для редактирования заметки с ошибками.
6. **Delete (GET)**
   * **Назначение:** Отображает подтверждение удаления заметки.
   * **Описание:** Получает идентификатор заметки и проверяет её наличие. Если заметка найдена, возвращает представление с подтверждением удаления. В противном случае возвращает страницу ошибки.
7. **Delete (POST)**
   * **Назначение:** Выполняет удаление заметки из базы данных.
   * **Описание:** Удаляет заметку с указанным идентификатором из базы данных и сохраняет изменения. Перенаправляет пользователя на страницу списка заметок.
8. **DownloadNotes (GET)**
   * **Назначение:** Скачивает все заметки текущего пользователя в виде документа Word.
   * **Описание:** Получает все заметки текущего пользователя из базы данных, создает документ Word с таблицей, содержащей даты и содержание заметок. Возвращает документ пользователю для скачивания.

### **Описание контроллера DashboardController**

Контроллер DashboardController управляет функционалом, связанным с пользовательским профилем и расписанием приема лекарств. Он использует различные зависимости, такие как репозиторий IDashboardRepository, контекст базы данных ApplicationDbContext, IHttpContextAccessor для доступа к текущему пользователю, IWebHostEnvironment для работы с файловой системой и IUserRepository для работы с пользователями.

#### Конструктор

* **DashboardController(IDashboardRepository dashboardRepository, IHttpContextAccessor httpContextAccessor, ApplicationDbContext context, IWebHostEnvironment webHostEnvironment, IUserRepository userRepository):** Конструктор инициализирует зависимости, необходимые для работы контроллера, такие как репозиторий для работы с данными, контекст базы данных для операций с хранилищем, доступ к текущему HTTP-контексту и окружение веб-хоста для работы с файлами и путями, а также репозиторий пользователей.

#### Методы

1. **Index**
   * **Назначение:** Отображает основную информацию о пользователе и его расписание приема лекарств.
   * **Описание:** Получает текущего пользователя через HTTP-контекст, затем получает расписание приема лекарств из репозитория. Формирует модель представления UserDetailViewModel, содержащую информацию о пользователе и его расписании. Возвращает представление с этой моделью.
2. **EditUserProfile (GET)**
   * **Назначение:** Отображает форму для редактирования профиля пользователя.
   * **Описание:** Получает идентификатор текущего пользователя через HTTP-контекст и использует его для получения данных о пользователе из репозитория. Если пользователь найден, инициализирует модель представления EditUserDashboardViewModel для редактирования профиля. Возвращает представление с этой моделью. Если пользователь не найден, возвращает страницу ошибки.
3. **EditUserProfile (POST)**
   * **Назначение:** Обрабатывает данные, введенные пользователем в форму для редактирования профиля.
   * **Описание:** Проверяет корректность данных в модели представления. Если данные валидны, загружает новое изображение пользователя (если оно было выбрано), обновляет данные пользователя и сохраняет изменения через репозиторий. Перенаправляет пользователя на главную страницу. В случае ошибки валидации возвращает представление с ошибками.
4. **Save (POST)**
   * **Назначение:** Сохраняет или обновляет расписание приема лекарств пользователя.
   * **Описание:** Получает список приемов лекарств и идентификатор пользователя из запроса. Удаляет существующие записи о приемах лекарств пользователя из базы данных и затем добавляет новые записи из списка приемов лекарств. Сохраняет изменения и перенаправляет пользователя на главную страницу.
5. **Delete (POST)**
   * **Назначение:** Удаляет запись о приеме лекарства из базы данных.
   * **Описание:** Получает идентификатор записи о приеме лекарства из запроса. Находит запись в базе данных и удаляет её. Сохраняет изменения и перенаправляет пользователя на главную страницу.
6. **DownloadMedicationSchedules (GET)**
   * **Назначение:** Скачивает расписание приема лекарств пользователя в виде документа Word.
   * **Описание:** Получает текущего пользователя через HTTP-контекст и использует его идентификатор для получения списка приемов лекарств из репозитория. Создает документ Word с таблицей, содержащей время дня, время приема и описание приема лекарства. Возвращает документ пользователю для скачивания.

### **Описание контроллера FileController**

Контроллер FileController обрабатывает операции с файлами пользователей, включая их загрузку, просмотр и удаление. Он использует различные зависимости, такие как репозиторий IDashboardRepository, IHttpContextAccessor для доступа к текущему пользователю, IHostingEnvironment для работы с файловой системой и ApplicationDbContext для операций с базой данных.

#### Конструктор

* **FileController(IDashboardRepository dashboardRepository, IHttpContextAccessor httpContextAccessor, IHostingEnvironment hostingEnvironment, ApplicationDbContext context):** Конструктор инициализирует зависимости, необходимые для работы контроллера, включая репозиторий для работы с данными, доступ к текущему HTTP-контексту, окружение хостинга для работы с файловой системой и контекст базы данных для операций с данными.

#### Методы

1. **Index (GET)**
   * **Назначение:** Отображает список файлов пользователя.
   * **Описание:** Получает идентификатор текущего пользователя через HTTP-контекст и использует его для получения списка файлов из базы данных. Формирует модель представления FileViewModel, содержащую список файлов и имя текущего файла (если указано). Возвращает представление с этой моделью.
2. **Index (POST)**
   * **Назначение:** Обрабатывает загрузку файла.
   * **Описание:** Принимает файл через параметр file типа IFormFile и сохраняет его на сервере в папку files в веб-корне проекта. Затем создает новую запись о файле в базе данных. Перенаправляет пользователя на метод Index(GET) для отображения обновленного списка файлов.
3. **DeleteFile (POST)**
   * **Назначение:** Удаляет файл.
   * **Описание:** Получает идентификатор файла из параметра запроса и находит соответствующий файл в базе данных. Удаляет файл с сервера и запись о нем из базы данных. Перенаправляет пользователя на метод Index(GET) для отображения обновленного списка файлов.
4. **PDFViewerNewTab (GET)**
   * **Назначение:** Открывает PDF-файл в новой вкладке браузера.
   * **Описание:** Получает имя файла из параметра запроса, формирует путь к файлу в папке files и отправляет содержимое файла пользователю в формате PDF для открытия в новой вкладке браузера.

### **Описание контроллера HealthCheckController**

Контроллер HealthCheckController предназначен для управления здоровьем пользователей, включая запись и отображение данных о здоровье, генерацию рекомендаций на основе этих данных.

#### Конструктор

* **HealthCheckController(ApplicationDbContext context, IHttpContextAccessor httpContextAccessor):** Конструктор инициализирует зависимости, необходимые для работы контроллера, включая контекст базы данных ApplicationDbContext и доступ к текущему HTTP-контексту IHttpContextAccessor.

#### Методы

1. **Index (GET)**
   * **Назначение:** Отображает страницу со списком предыдущих здоровых данных пользователя.
   * **Описание:** Получает идентификатор текущего пользователя через HTTP-контекст и использует его для извлечения списка предыдущих здоровых данных из базы данных, отсортированных по дате в обратном порядке. Затем передает данные в представление Index через ViewBag.
2. **CheckHealth (POST)**
   * **Назначение:** Обрабатывает данные о здоровье, сохраняет их в базу данных и генерирует рекомендации на основе введенных данных.
   * **Описание:** Проверяет валидность модели данных HealthDataViewModel. Если данные валидны, получает идентификатор текущего пользователя через HTTP-контекст, сохраняет новую запись HealthData в базу данных, используя данные из модели. Затем вызывает метод GenerateRecommendations для генерации рекомендаций на основе введенных данных. После сохранения данных в базу данных и генерации рекомендаций, обновляет список предыдущих здоровых данных и передает его в представление Result.
3. **GenerateRecommendations (private)**
   * **Назначение:** Генерирует рекомендации на основе введенных данных о здоровье.
   * **Описание:** Принимает модель HealthDataViewModel и проверяет различные показатели здоровья (артериальное давление, ИМТ, уровень сахара в крови, уровень холестерина). Для каждого показателя генерирует соответствующую рекомендацию и передает ее через ViewBag в представление Result.

### **Описание контроллера MapController**

Контроллер MapController предназначен для отображения информации о пользователе на карте.

#### Конструктор

* **MapController(IHttpContextAccessor httpContextAccessor, IUserRepository userRepository):** Конструктор инициализирует зависимости, необходимые для работы контроллера, включая доступ к текущему HTTP-контексту IHttpContextAccessor и репозиторий пользователей IUserRepository.

#### Методы

1. **Index (GET)**
   * **Назначение:** Отображает страницу с картой и информацией о городе пользователя.
   * **Описание:** Получает идентификатор текущего пользователя через HTTP-контекст и использует его для извлечения информации о пользователе из репозитория пользователей IUserRepository. Затем передает название города пользователя в представление через ViewBag.

### **Описание контроллера MedicinesController**

Контроллер MedicinesController предназначен для управления лекарствами в системе.

#### Конструктор

* **MedicinesController(ApplicationDbContext context):** Конструктор инициализирует контекст базы данных ApplicationDbContext, необходимый для доступа к данным о лекарствах.

#### Методы

1. **Index (GET)**
   * **Назначение:** Отображает список лекарств с возможностью фильтрации по имени и заболеванию.
   * **Описание:** Извлекает все лекарства из базы данных. Если задан параметр searchString, фильтрует лекарства по имени. Если задан параметр disease, фильтрует лекарства по заболеванию. Затем передает список лекарств в представление для отображения.
2. **Create (GET)**
   * **Назначение:** Отображает форму для создания нового лекарства.
   * **Описание:** Возвращает представление, которое содержит форму для ввода данных о новом лекарстве.
3. **Create (POST)**
   * **Назначение:** Обрабатывает данные формы для создания нового лекарства.
   * **Описание:** Проверяет валидность модели данных. Если данные введены корректно, добавляет новое лекарство в базу данных и перенаправляет пользователя на страницу со списком лекарств.
4. **Edit (GET)**
   * **Назначение:** Отображает форму для редактирования существующего лекарства.
   * **Описание:** Извлекает лекарство по заданному идентификатору id. Если лекарство найдено, отображает представление для редактирования существующих данных лекарства.
5. **Edit (POST)**
   * **Назначение:** Обрабатывает данные формы для редактирования существующего лекарства.
   * **Описание:** Проверяет валидность модели данных. Если данные введены корректно, обновляет информацию о лекарстве в базе данных и перенаправляет пользователя на страницу со списком лекарств.
6. **Delete (GET)**
   * **Назначение:** Отображает страницу подтверждения удаления лекарства.
   * **Описание:** Извлекает лекарство по заданному идентификатору id. Если лекарство найдено, отображает страницу подтверждения удаления.
7. **DeleteConfirmed (POST)**
   * **Назначение:** Удаляет выбранное лекарство из базы данных.
   * **Описание:** Извлекает лекарство по заданному идентификатору id. Если лекарство найдено, удаляет его из базы данных и перенаправляет пользователя на страницу со списком лекарств.
8. **MedicineExists**
   * **Назначение:** Проверяет существование лекарства по заданному идентификатору.
   * **Описание:** Выполняет запрос к базе данных для проверки наличия лекарства с указанным идентификатором.

# Руководство пользователя

При первом посещении сайта пользователь увидит главную страницу, где будет представлена краткая информация о ресурсе, это показано на рисунке 5.1.

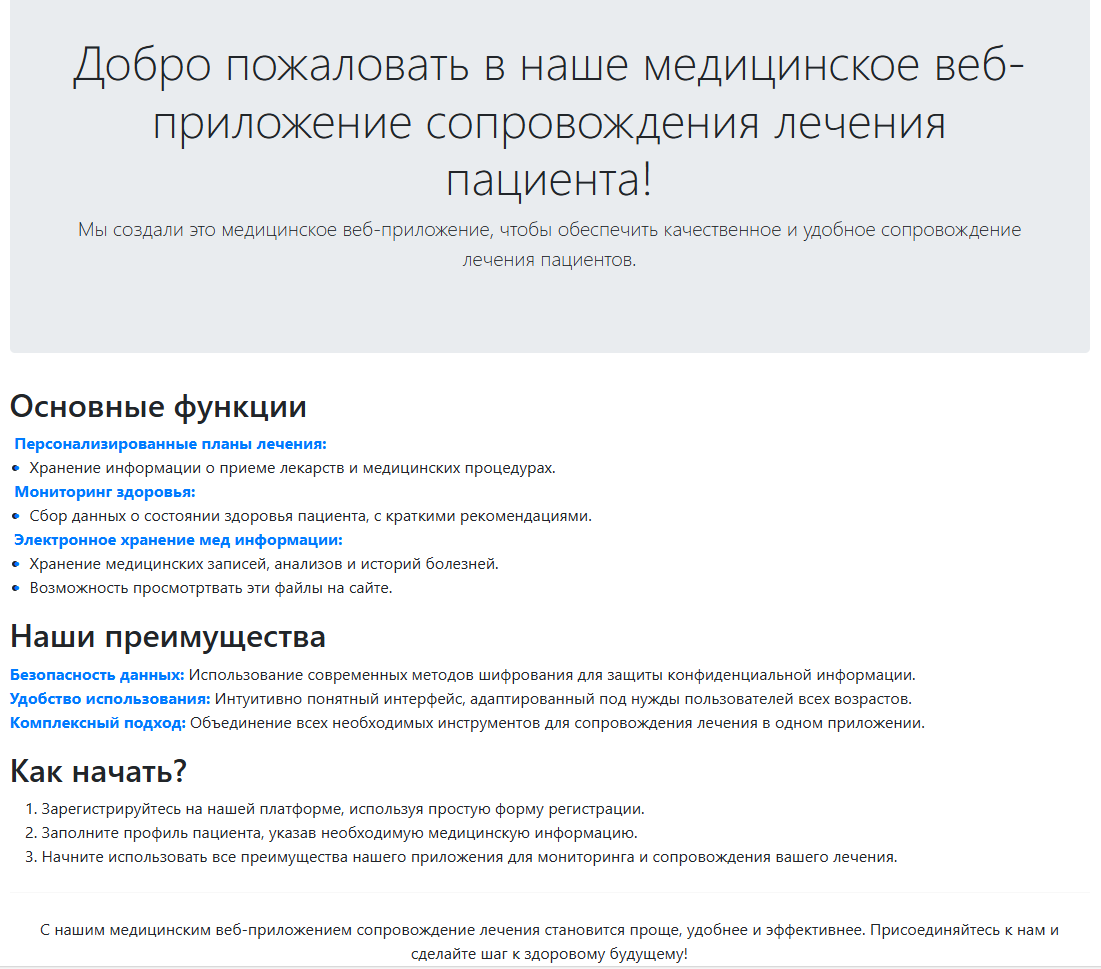


Рисунок 5.1 – Главная страница

После просмотра данной страницы, пользователю потребуется авторизоваться в своей учетной записи, как это выглядит на рисунке 5.2.

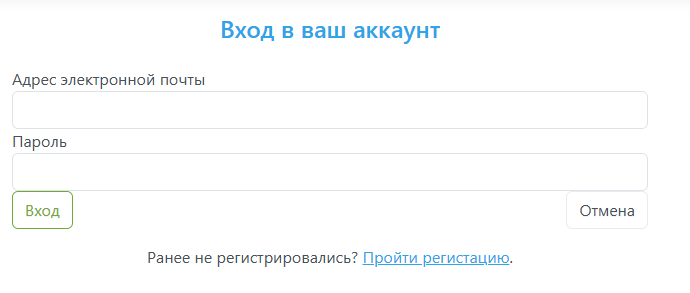


Рисунок 5.2 – Страница входа в аккаунт

Если пользователь впервые посещает сайт, это значит что он не имеет своего аккаунта на сайте и ему будет необходимо пройти регистрацию. Пример можно увидеть на рисунке 5.3.

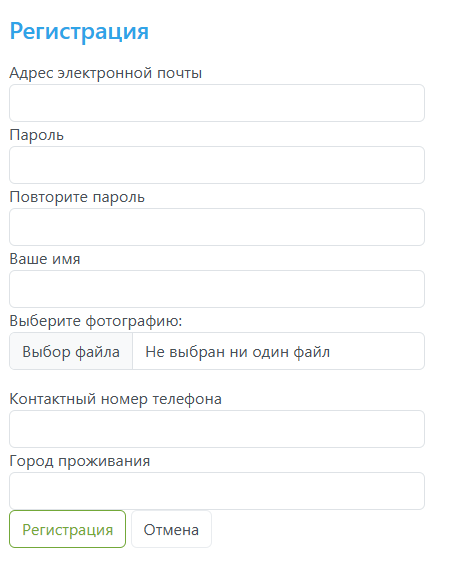


Рисунок 5.3 – Страница регистрации

После регистрации и входа в свой аккаунт пользователь сможет воспользоваться всем функционалом сайта. Он представлен на рисунке 5.4.

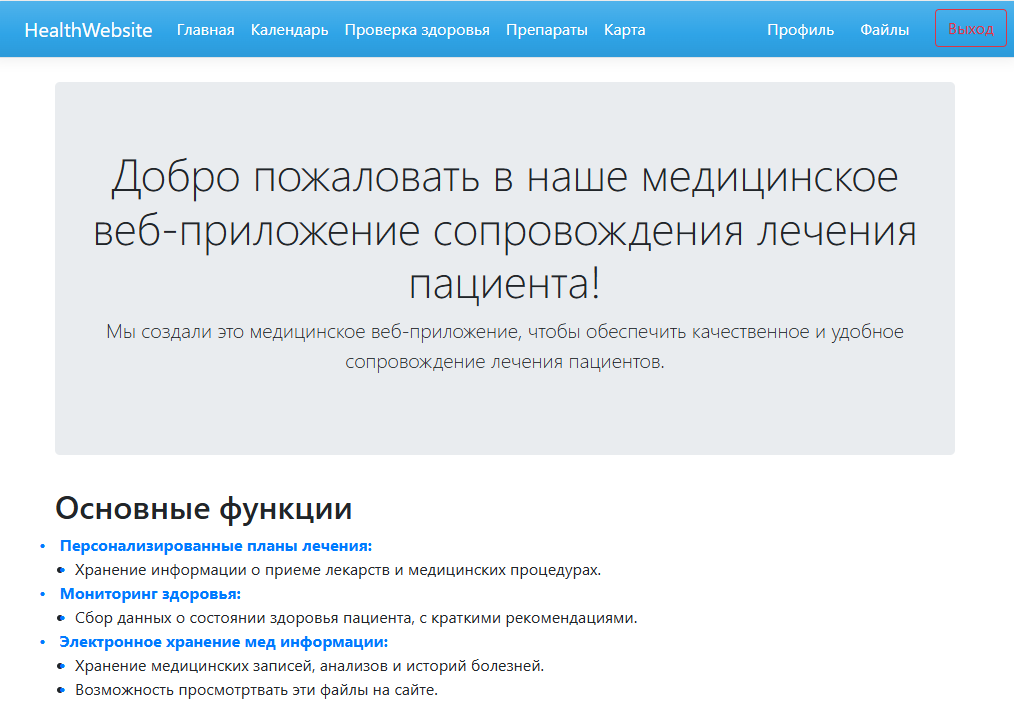


Рисунок 5.4 – Внешний вид сайта после входа в аккаунт

На странице «Профиль» пользователь может просмотреть свои данные, и у него будет возможность в любой момент их изменить.

Также на этой страничке будет находится раздел «Таблетница», в которой будет храниться информация о принимаемых на данный момент пользователем лекарственных средств. Он может как добавлять новые лекарства, так и удалять старые. Внешний вид профиля можно увидеть на рисунке 5.5.

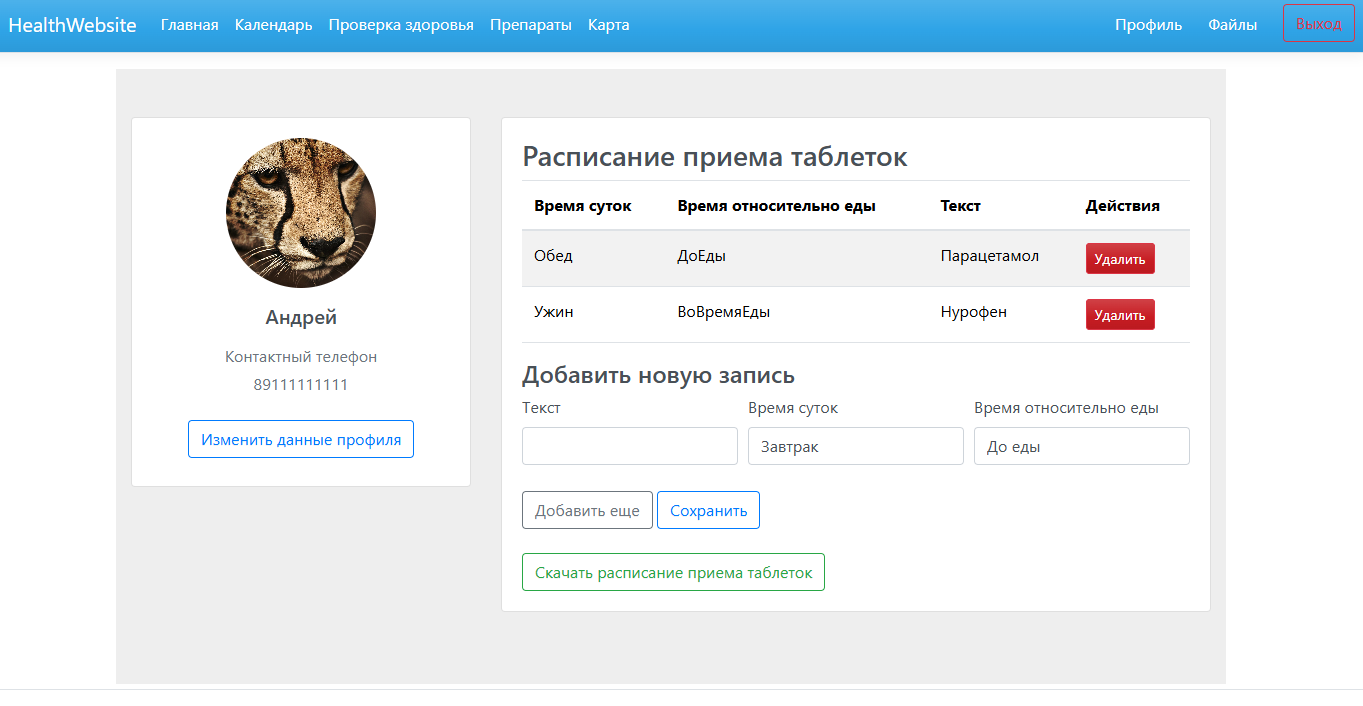


Рисунок 5.5 – Страница «Профиль»

Также у него есть возможность получить данные таблицы в виде word файла, как показано на рисунке 5.6.

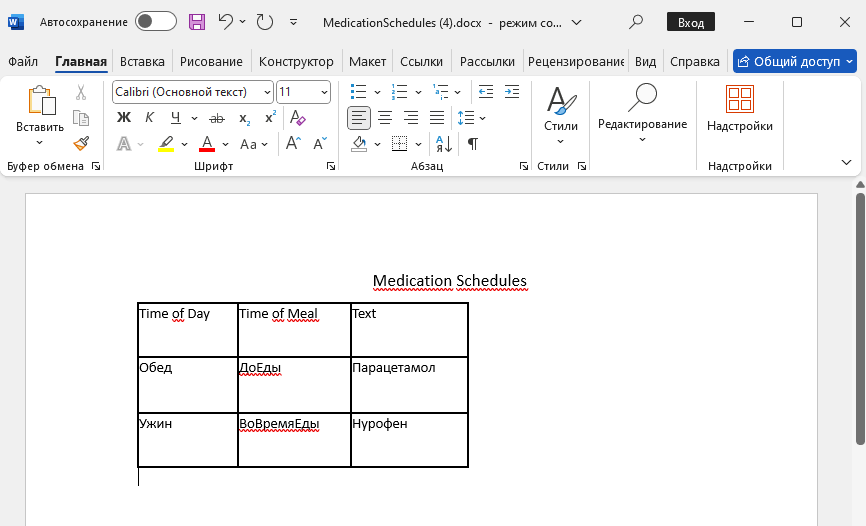


Рисунок 5.6 – Скаченный файл с информацией об принимаемых лекарствах

На странице «Файлы» пользователь может хранить и просматривать pdf файлы на сайте, это можно увидеть на рисунке 5.7.

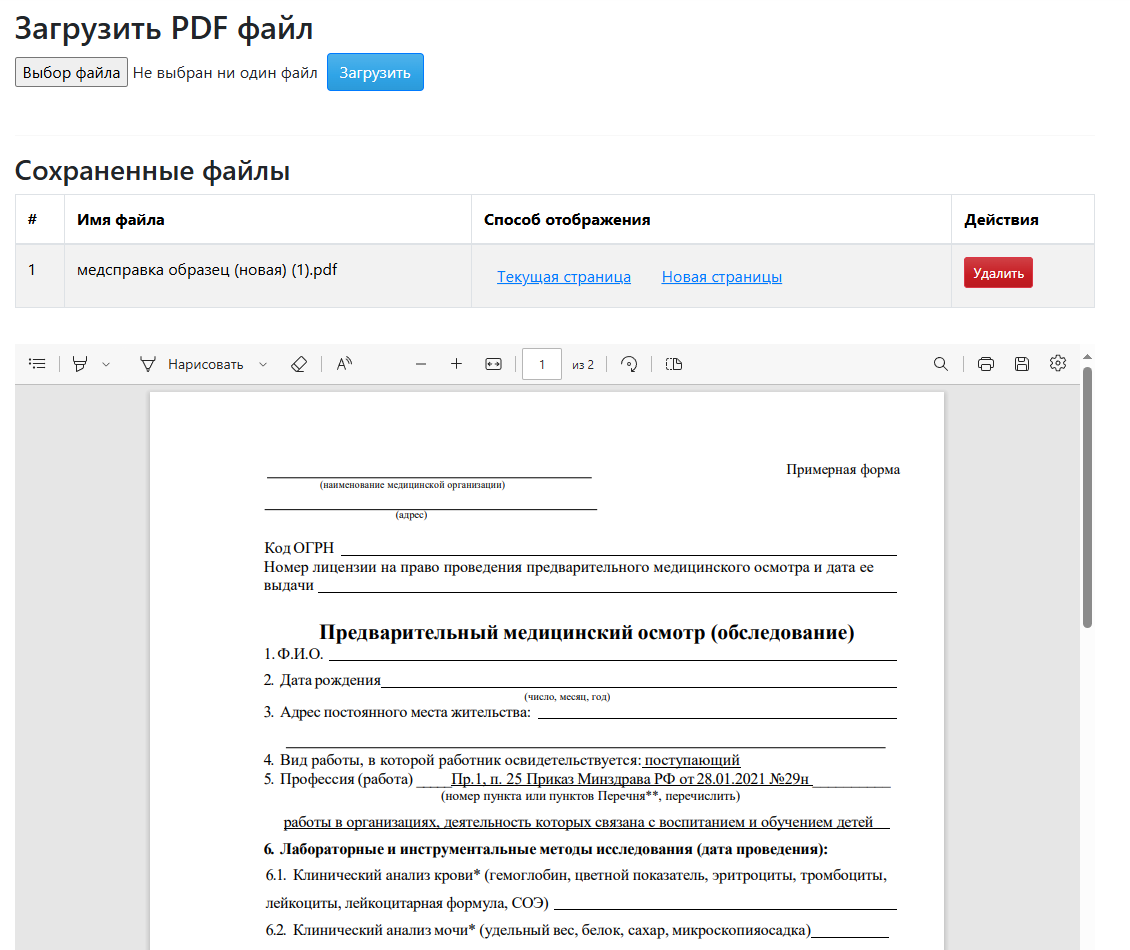


Рисунок 5.7 – Страница «Файлы»

На странице «Календарь» пользователь может хранить информацию об записи на прием к врачу и тп., в виде заметок. На рисунке 5.8 можно увидеть внешний вид данного раздела.

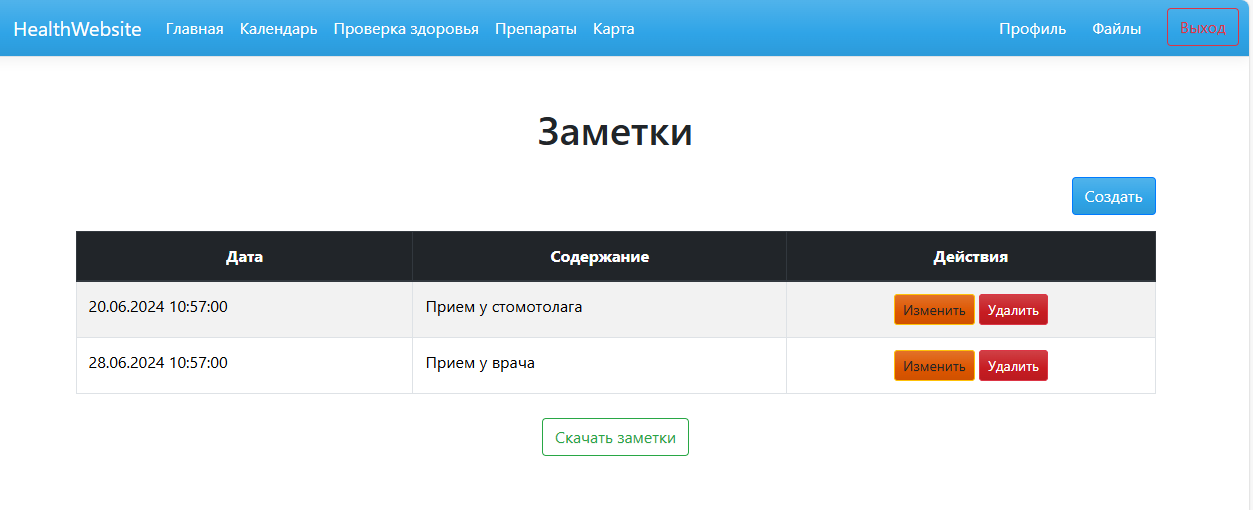


Рисунок 5.8 – Страница «Календарь»

Он может добавлять новые заметки, удалять и редактировать старые. А также он может скачать информацию о заметках в виде word файла. Рисунок 5.9 показывает, как это будет выглядеть.

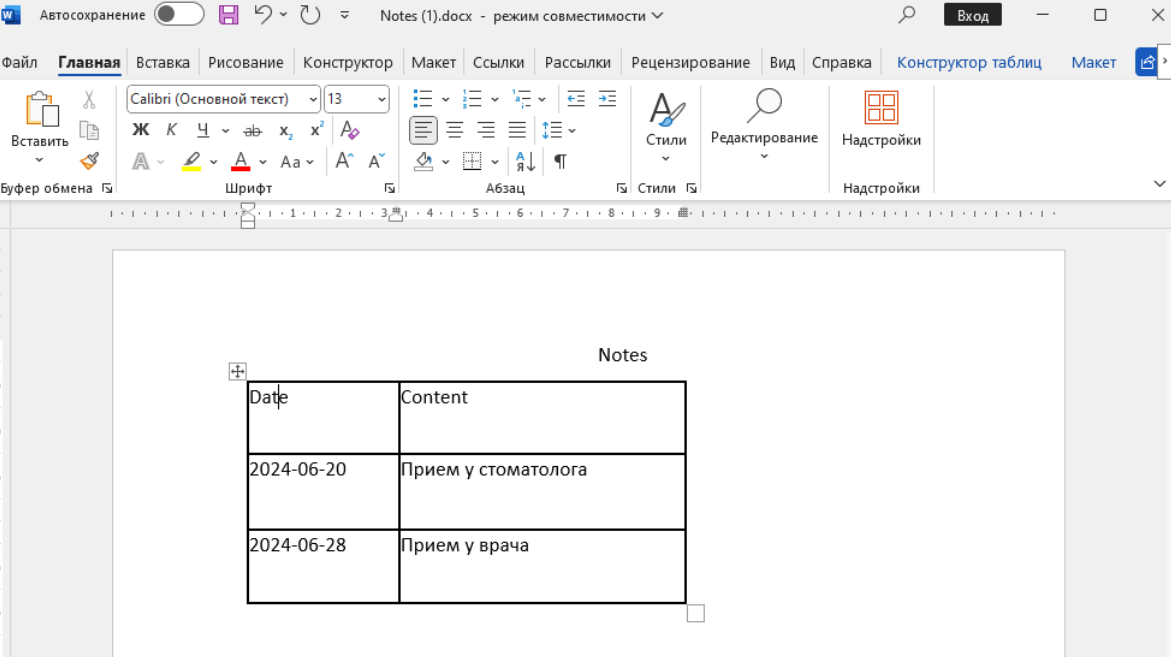


Рисунок 5.9 – Скаченный файл с заметками важных дат

На станице «Проверка здоровья» (Рисунок 5.10) пользователь может провести проверку своего здоровья, заполнив соответствующие поля.

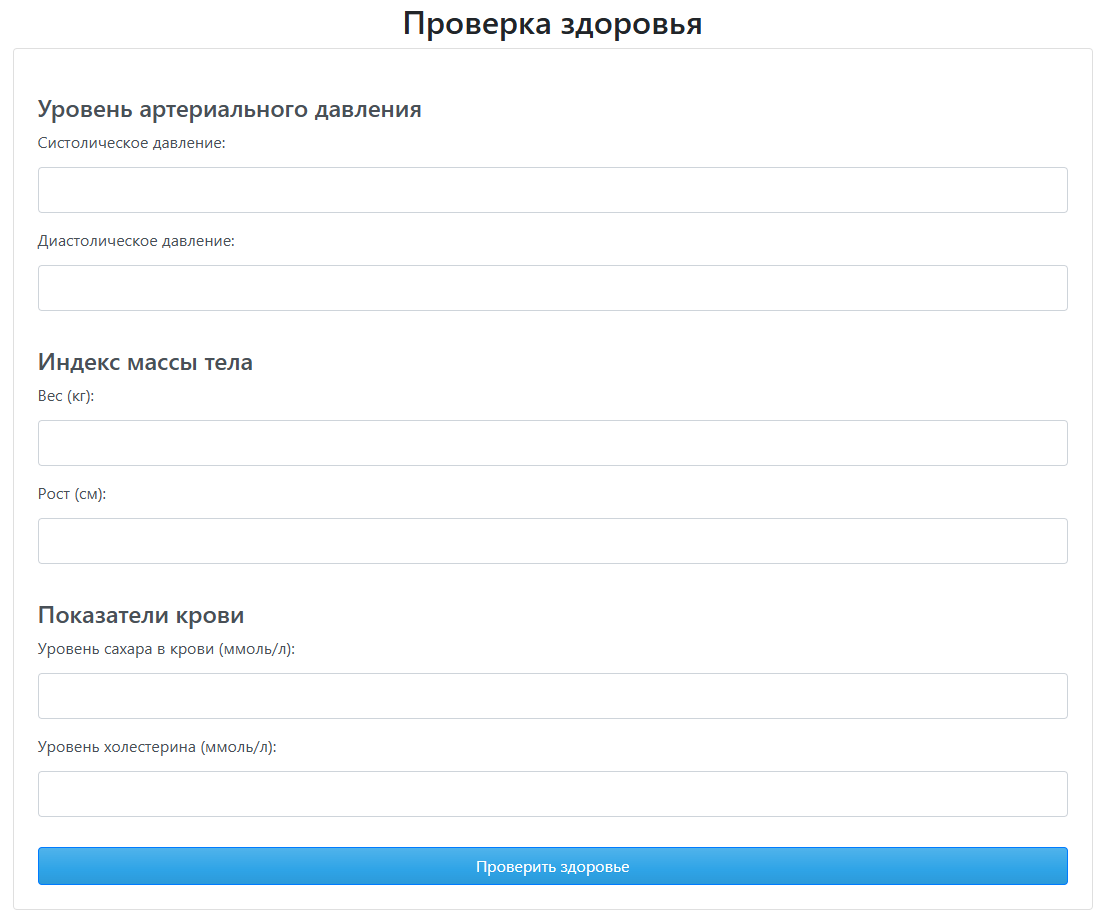
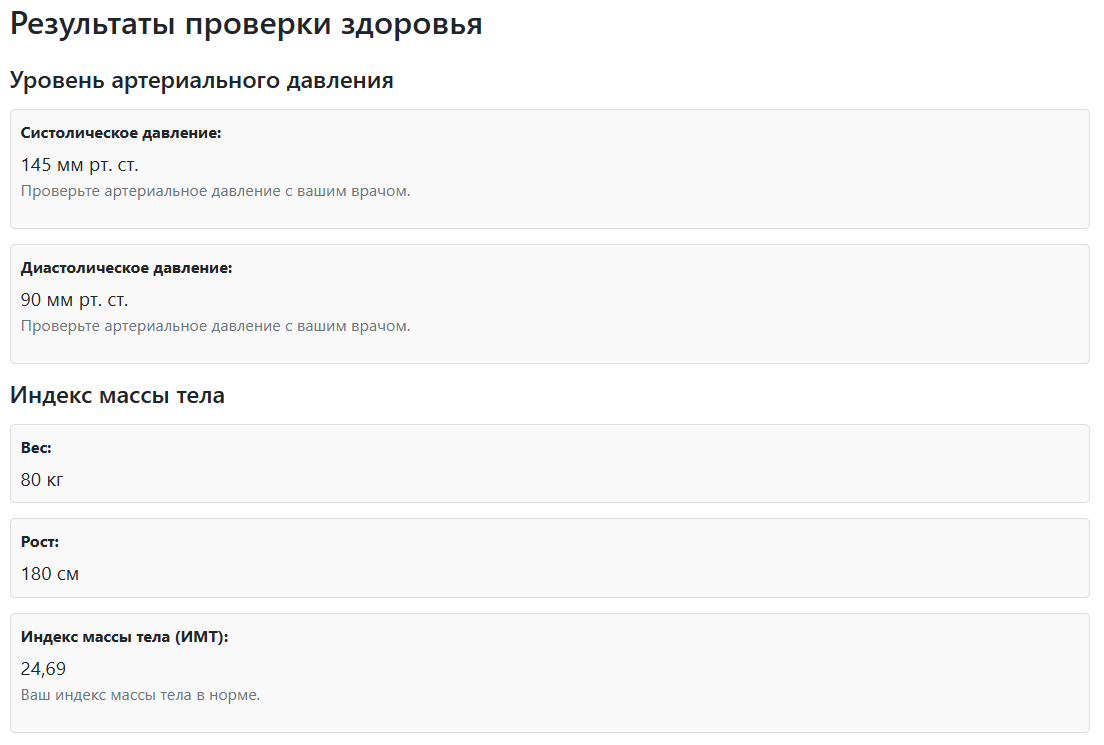


Рисунок 5.10 – Страница «Проверка здоровья»

После нажатия на кнопку «Проверить здоровье» пользователь получит краткие рекомендации по своим показателям, а также сможет посмотреть свои старые данные. Это отображено на рисунке 5.11.



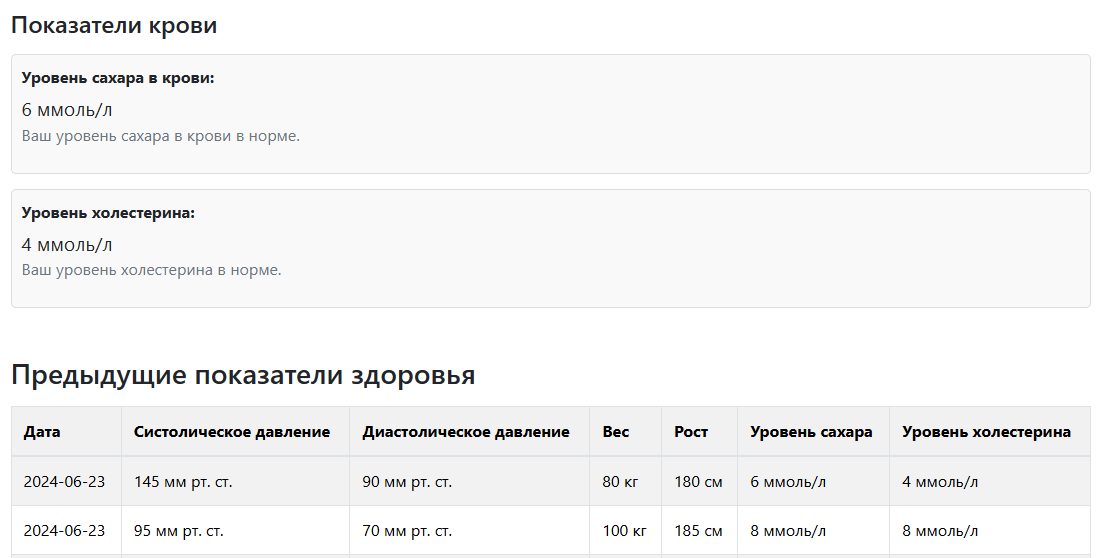


Рисунок 5.11 – Страница «Проверка здоровья» после ввода данных здоровья

На старнице «Карта» (Рисунок 5.12) пользователь сможет найти медицинские учреждения в своем городе.

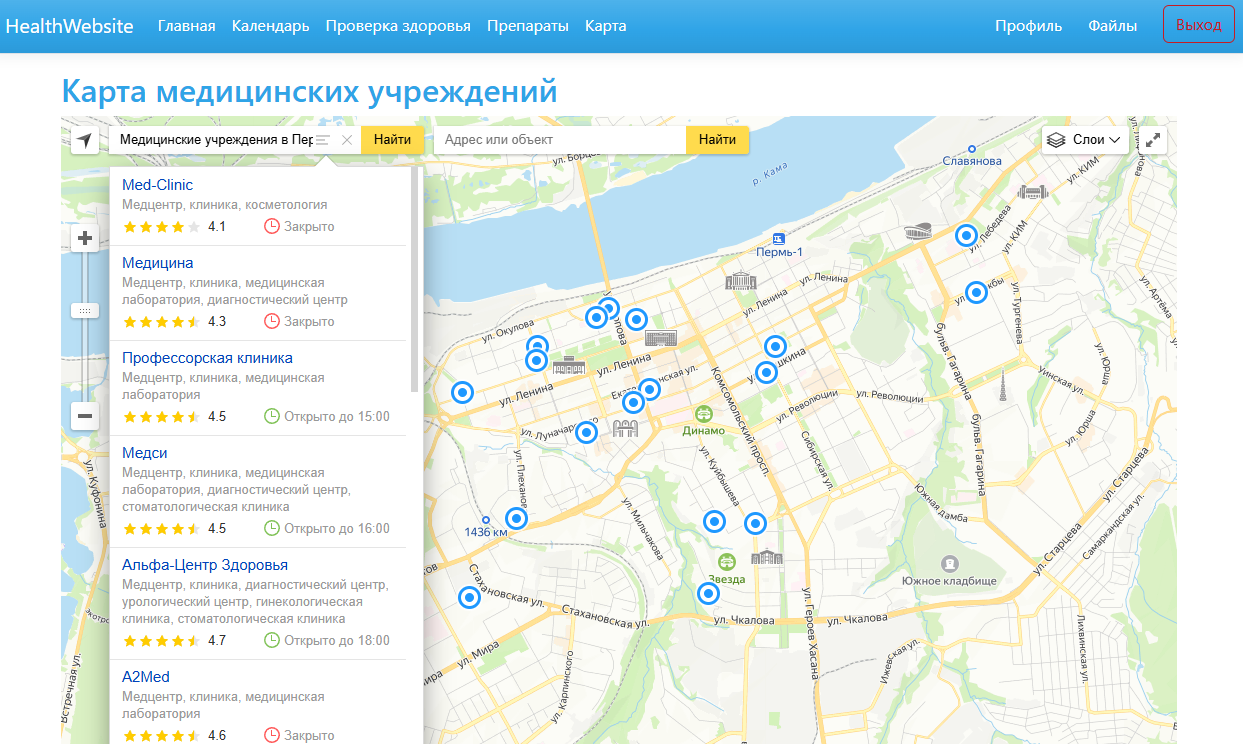


Рисунок 5.12 – Страница «Карта»

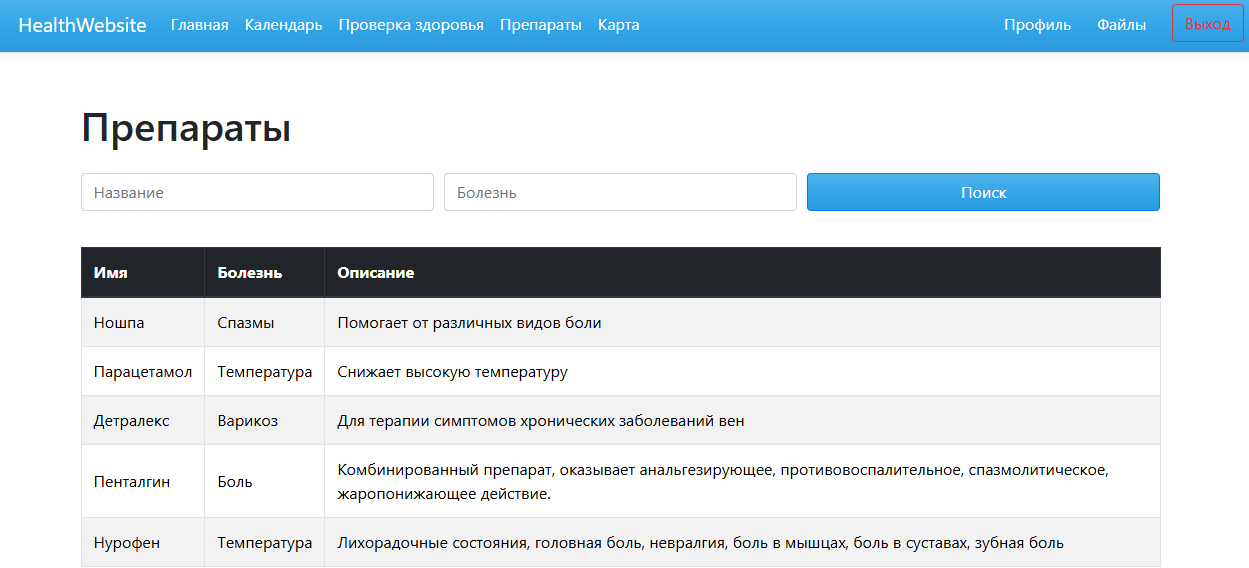
На станице «Препараты» (Рисунок 5.13) пользователь сможет осуществить поиск лекарственных средств для своих нужд. Он может воспользоваться поиском: найти либо сразу интересующее его лекарство, либо искать по конкретной болезни. 

Рисунок 5.13 – Страница «Препараты»

Добавлять новые лекарственные средства, а также редактировать и удалять уже добавленные, может только пользователь с ролью «Admin» (Рисунок 5.14).

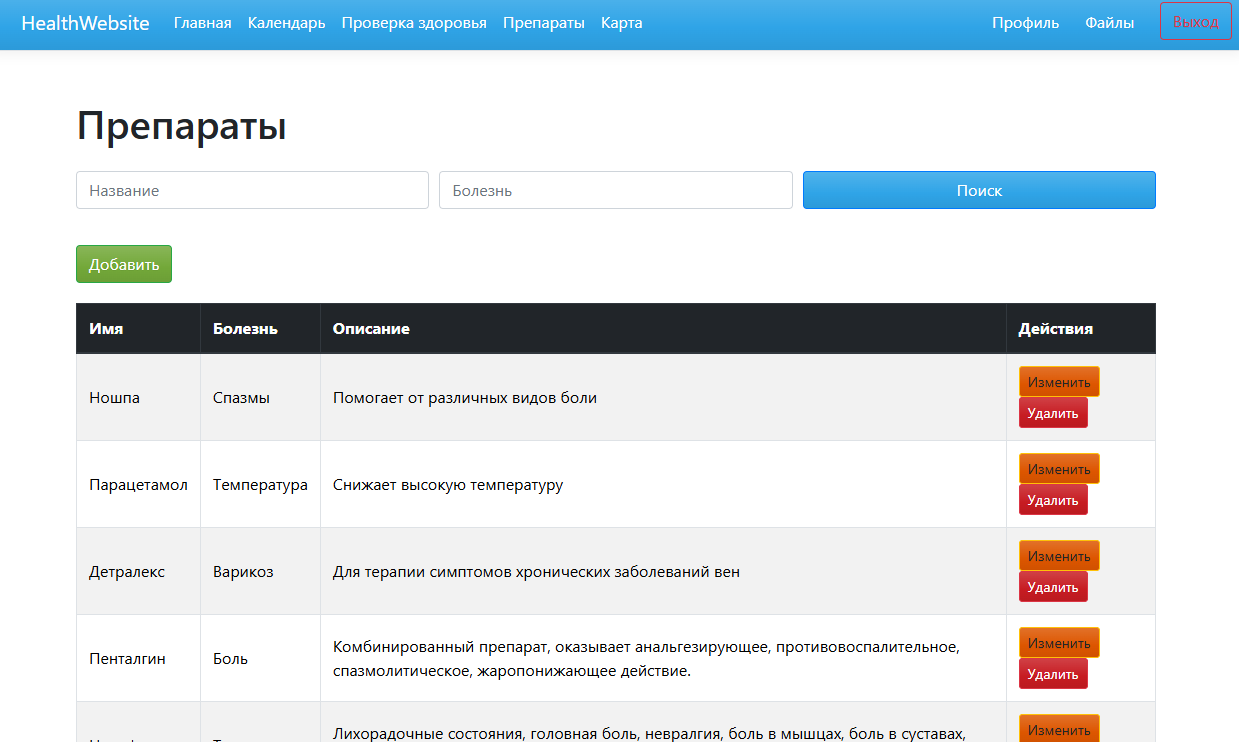


Рисунок 5.14 – Страница «Препараты» для пользователя с ролью «Admin»

# Заключение

В процессе разработки медицинского веб-приложения, предназначенного для сопровождения лечения пациентов и позволяющего им самостоятельно отслеживать своё здоровье, были применены передовые технологии и инструменты, обеспечивающие высокую эффективность, удобство использования и безопасность для пользователей. Использование языка программирования C# вместе с фреймворком ASP.NET Core послужило основой для нашего приложения. Эта комбинация позволяет быстро создавать веб-приложения, обеспечивает высокую производительность и возможность масштабирования, что критически важно для приложения, которое должно обрабатывать и хранить значительные объемы медицинских данных. Для надежного хранения и управления медицинскими данными была выбрана база данных SQL Server. Она гарантирует высокую производительность, надежность и безопасность данных, что особенно важно в медицинских приложениях, где конфиденциальность и целостность информации имеют первостепенное значение.

Решение использовать Visual Studio обусловлено её широкими функциональными возможностями и удобством работы с проектом. Visual Studio предоставляет мощные инструменты для написания, отладки и развертывания кода, что значительно упрощает процесс разработки. Entity Framework Core, этот объектно-реляционный диспетчер (ORM) для .NET, упростил работу с данными, предоставляя возможность взаимодействовать с базой данных через объекты .NET, что позволяет избежать написания большого количества SQL-кода вручную. ASP.NET Core Identity обеспечивает безопасное управление пользовательскими учетными записями, включая аутентификацию и авторизацию пользователей, что критично для защиты конфиденциальных медицинских данных.

Применение C# и ASP.NET Core в сочетании с Visual Studio позволяет разработчикам быстро и эффективно создавать новые функции и исправлять ошибки, что сокращает время разработки и улучшает качество кода. Использование SQL Server обеспечивает возможность масштабирования базы данных по мере роста числа пользователей и объема данных, а также поддерживает высокую производительность и надежность системы. ASP.NET Core Identity обеспечивает надежную защиту пользовательских данных, что особенно важно в медицинском приложении. Все операции по управлению учетными записями и доступом к данным выполняются с соблюдением высоких стандартов безопасности. Выбранные технологии и инструменты не только способствуют текущей разработке, но и облегчают будущую поддержку и расширение приложения. Это позволяет легко добавлять новые функции и улучшения в ответ на изменяющиеся потребности пользователей и медицинских учреждений. Использование указанных технологий и инструментов в разработке медицинского веб-приложения для сопровождения лечения пациентов является оптимальным решением, учитывающим требования проекта и потребности пользователей. Это обеспечивает пациентам удобный и надежный инструмент для мониторинга своего здоровья, что способствует улучшению качества их жизни и взаимодействия с медицинскими специалистами.