3 ПРОЕКТИРОВАНИЕ

3.1 Проектирование структур хранения данных

Для работы с данными была выбрана Microsoft SQL Server. Microsoft SQL Server - система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов - Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка. SQL Server - это основа платформы обработки данных Майкрософт, которая предоставляет надёжную и устойчивую производительность (в том числе благодаря технологиям обработки данных в памяти) и помогает быстрее извлечь ценную информацию из любых данных, расположенных как в локальной среде, так и в облаке [1].

Схема базы данных, отвечающей за хранение информации в проекте, представлена ниже (рисунок 3.1).

Для комфортной работы с данными в процессе разработки и тестирования используется среда SQL Server Management Studio.

Среда SQL Server Management Studio - это единая универсальная среда для доступа, настройки и администрирования всех компонентов MS SQL Server, а также для разработки компонентов системы, редактирования текстов запросов, создания скриптов и пр. Благодаря наличию большого количества визуальных средств управления, среда SQL Server Management Studio позволяет выполнять множество типовых операций по администрированию

MS SQL Server администраторам с любым уровнем знаний SQL Server. Удобная среда разработки, встроенный веб-браузер для быстрого обращения к библиотеке MSDN или получения справки в сети, подробный учебник, облегчающий освоение многих новых возможностей, встроенная справка от сообществ в Интернете и многое другое позволяют максимально облегчить процесс разработки в среде SQL Server, а также даёт богатые возможности для создания различных сценариев SQL Server [2].

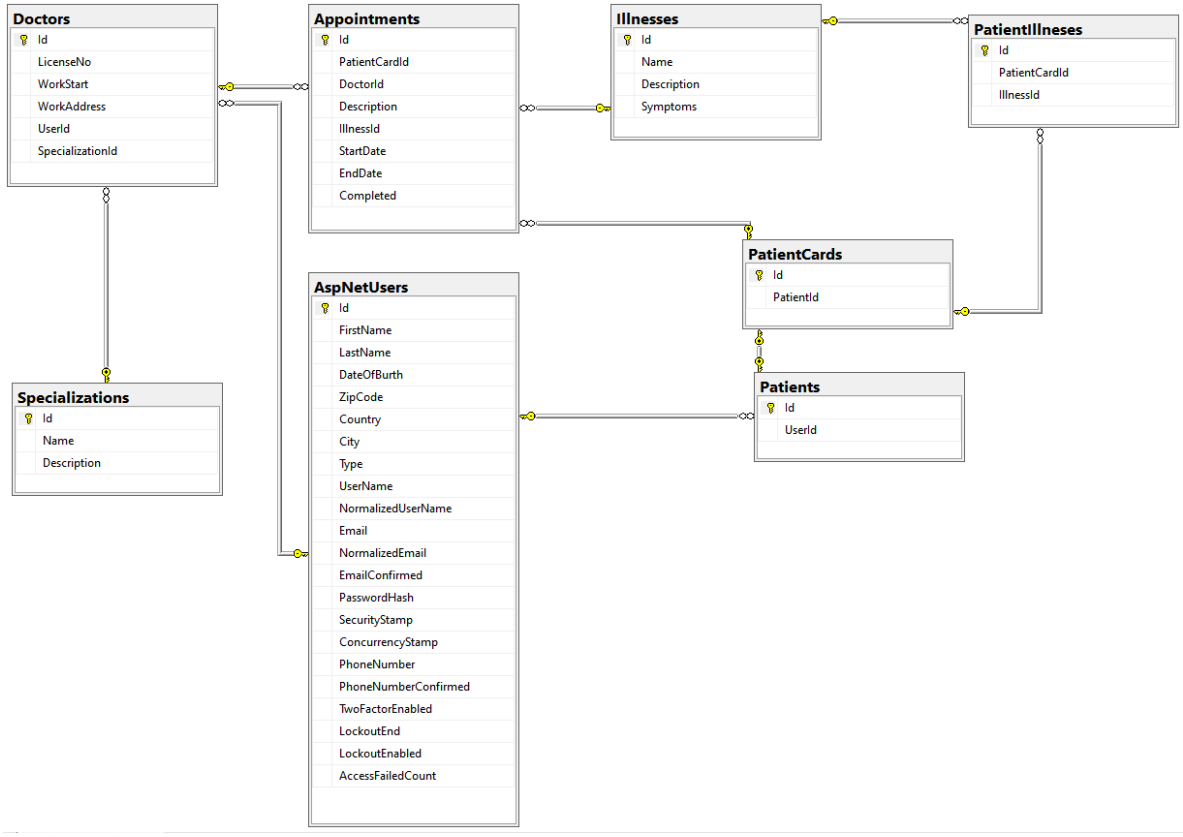

Рисунок 3.1 - Схема базы данных

Также, в рамках данного раздела, стоит упомянуть о способе хранения данных в фронтэнд части приложения. За это отвечает Redux.

Redux - это инструмент для управления состоянием данных и пользовательским интерфейсом в приложениях JavaScript с большим количеством сущностей. Представляет собой библиотеку JavaScript.

Название читается как «Редакс» и составлено из двух слов: reduce и flux. Reduce - это функция, которая приводит большую структуру данных к одному значению. Flux - архитектура приложения, при которой данные передаются в одну сторону. Инструмент основан на этих двух понятиях, поэтому они вынесены в название. Обычно Redux используется в связке с фреймворками для JavaScript: React, TypeScript, Vue, Angular и другими. Реже он бывает нужен для написания кода на чистом JS. Имеет открытый исходный код и доступен бесплатно. Со всеми зависимостями весит всего около 2 Кб.

Для чего нужен Redux:

- для управления состоянием приложения, работающего с большим количеством данных;
- для удобной замены встроенных средств работы с состоянием в React;
- для более лёгкого масштабирования приложения, его преобразования под разные задачи;
- для избавления от ошибок, связанных с беспорядком в объекте состояния;
- для предсказуемости и понятности работы приложения;
- для более простой отладки и доработки;
- для повышения производительности и работоспособности программы [3].

3.2 Разработка архитектуры программной системы

Приложение состоит из 2 частей: бекэнд и фронтэнд. Они общаются между собой посредством протоколов HTTP и HTTPS. Пользователь взаимодействует лишь с фронтэнд частью, которая предоставляет ему

интерфес приложения, взаимодействия с которым запускают запросы к бекэнд части приложения.

Схема развёртывания приложения представлена ниже (рисунок 3.2).

Для разработки бэкэнд части проекта была выбрана многоуровневая архитектура (N-layer).

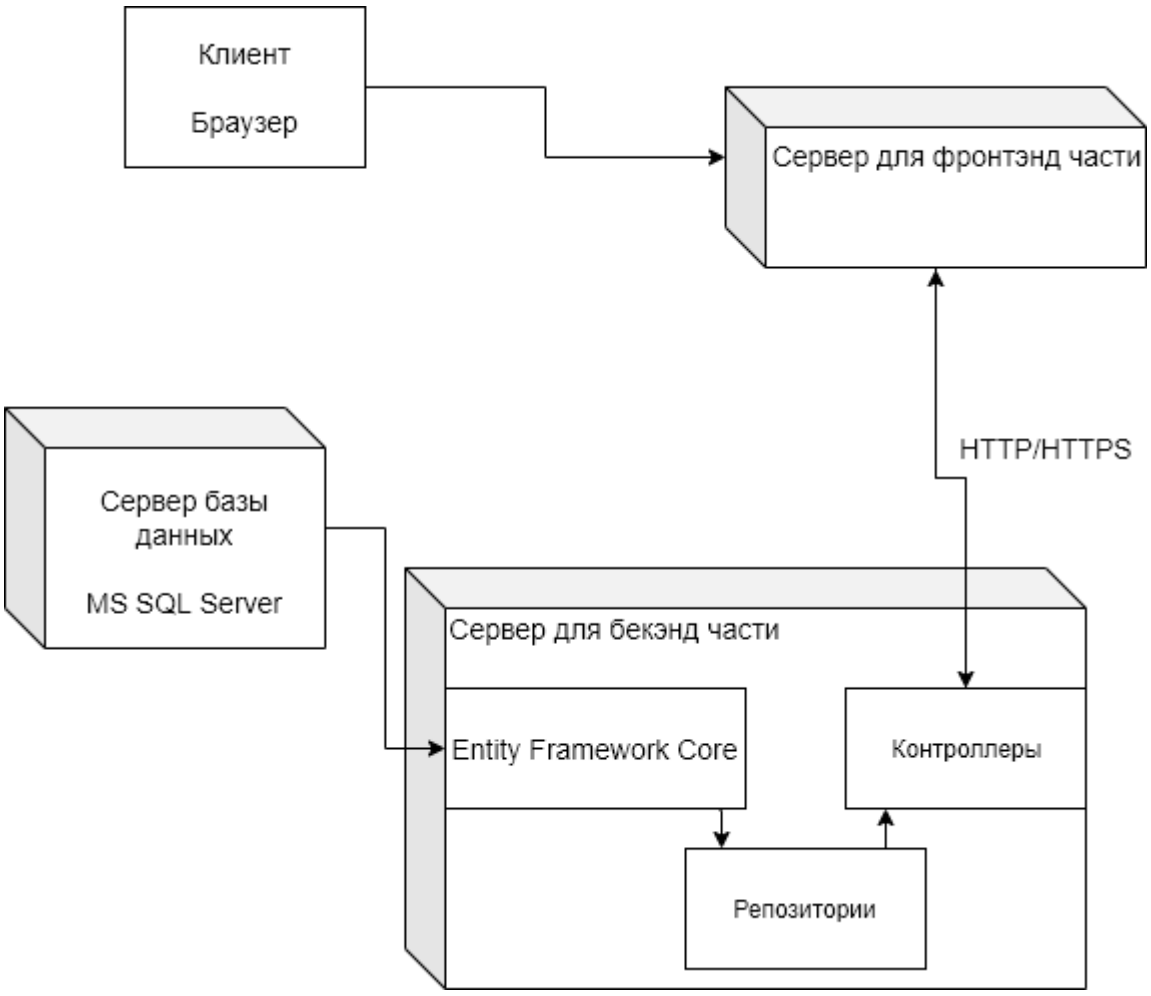

Рисунок 3.2 - Диаграмма развёртывания приложения

N-уровневая архитектура - это концепция клиент-серверной архитектуры в программной инженерии, в которой функции представления, обработки и управления данными логически и физически разделены. Каждая из этих функций работает на отдельном компьютере или в отдельных

кластерах, так что каждая из них может предоставлять услуги с максимальной пропускной способностью, поскольку отсутствует совместное использование ресурсов.

Такое разделение делает управление каждым отдельным ресурсом проще, так как выполнение работы над одним не влияет на другие, изолируя любые проблемы, которые могут возникнуть.

N-уровневая архитектура обычно делит приложение на три уровня: уровень представления, логический уровень и уровень данных. Это физическое разделение различных частей приложения в отличие от обычно концептуального или логического разделения элементов в структуре модель-представление-контроллер (MVC). Другое отличие от инфраструктуры MVC состоит в том, что n-уровневые уровни связаны линейно, то есть вся связь должна проходить через средний уровень, который является логическим уровнем. В MVC нет реального среднего слоя, потому что взаимодействие является треугольным; уровень управления имеет доступ как к слоям вида, так и к слою модели, а модель также обращается к виду; Контроллер также создаёт модель на основе требований и передаёт её в представление. Однако они не являются взаимоисключающими, поскольку инфраструктура MVC может использоваться в сочетании с n-уровневой архитектурой, причём n-уровень является общей используемой архитектурой, а MVC используется в качестве основы для уровня представления [4].

Данная архитектура была выбрана так как имеет следующие преимущества:

- более простая реализация по сравнению с другими подходами;

- предлагает абстракцию благодаря разделению ответственностей между уровнями;

- изолирование защищает одни слои от изменений других;

- повышает управляемость программного обеспечения за счёт слабой связанности [5].

Схема многоуровневой архитектуры представлена ниже (рисунок 3.3).

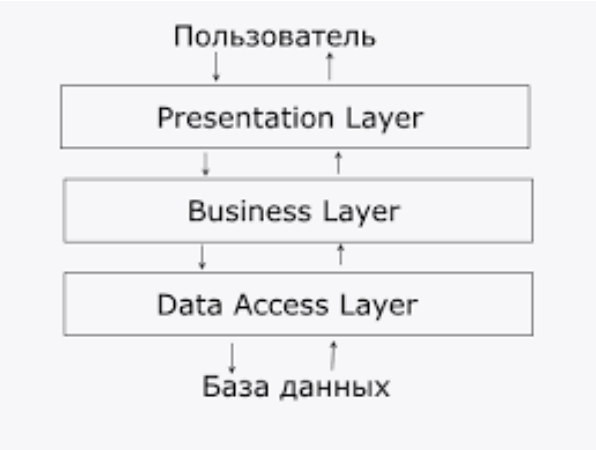

Рисунок 3.3 - Схема связей между слоями многоуровневой архитектуры

3.3 Разработка архитектуры компонентов программной системы

Приложение состоит из двух компонентов: бекэнд и фронтэнд.

В бэкэнд части приложения представлены 4 «слоя»:

- API - слой. Так как бэкэнд часть приложения представляет собой REST API, этот слой играет роль представления в классической N-layer архитектуре;

- слой бизнес-логики. Это библиотека классов, содержащая классы, необходимые для различных вычислений и преобразований, например, AutoMapper, который служит для автоматической трансляции объекта одного типа в объект другого типа;

- слой моделей. Это библиотека классов, содержащая классы, представляющие собой объекты и сущности системы;

- слой доступа к данным. Это библиотека классов, содержащая всё необходимое для работы с базой данных: контекст - позволяет работать с БД, репозитории - классы, работающие с определёнными частями контекста для

упрощённого доступа к данным, миграции - записи, диктующие как правильно транслировать код в базу данных и наоборот.

Как видно, в отличие от типичной N-уровневой архитектуры, которая содержит 3 слоя, приложение имеет 4 слоя. Это потому, что данная архитектура не устанавливает жёстких правил и позволяет вводить дополнительные слои. Можно сказать, что слой API и слой моделей вместе представляют собой слой представления, диктующий какие данные и как будут отображаться для конечного пользователя.

В обозревателе решений Visual Studio архитектура проекта выглядит следующим образом (рисунок 3.4).

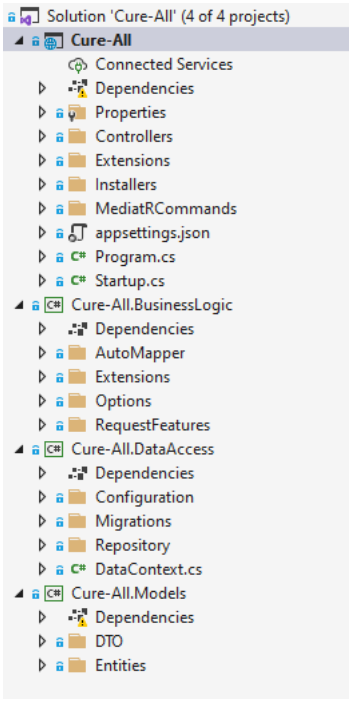

Рисунок 3.4 - Многоуровневая архитектура проекта в Visual Studio

Фронтэнд часть проекта представлена единым React-приложением, которое организовано в стиле многоуровневой архитектуры, слои представлены не отдельными библиотеками классов, а папками с файлами в основном проекте:

- Api - слой доступа к данным - папка содержит TypeSctipt файлы, осуществляющие запросы к бэкэнд части проекта;

- Components - слой представления - папка содержит компоненты представления интерфейса;

- Content - папка содержит файлы, необходимые для построения интерфейса, не являющиеся стилями, например изображения;

- Store - папка содержит все компоненты, необходимые для работы хранилища Redux;

- Styles - папка содержит файлы, содержащие стили для построения пользовательского интерфейса.

В обозревателе Visual Studio Code структура приложения выглядит следующим образом (рисунок 3.5).

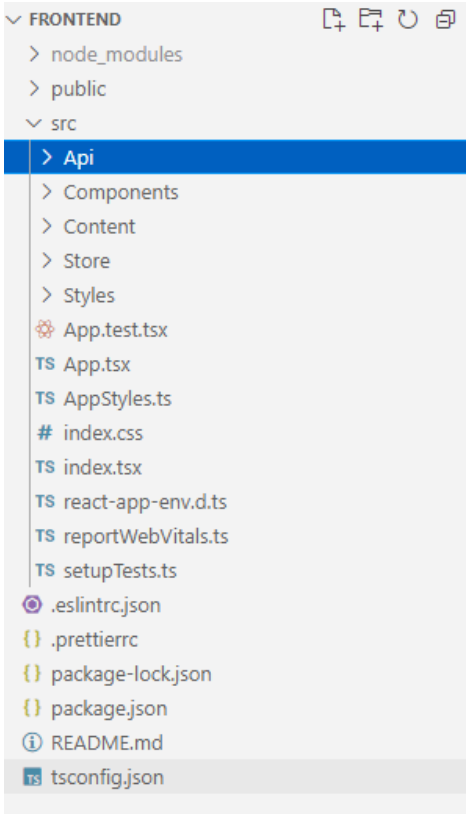

Рисунок 3.5 - Структура приложения в Visual Studio Code

Фронтэнд часть отвечает лишь за получение и отображение данных, поэтому не имеет привычной для многоуровневой архитектуры слой бизнес-логики.

3.4 Разработка интерфейса программного продукта

Фронтэнд часть приложения предоставляет графический пользовательский интерфейс(GUI). Как упоминалось в предыдущих разделах, фронтэнд часть создана с использованием библиотеки для JavaScript React с поддержкой TypeScript. Компоненты GUI представляют собой классовые компоненты React, маршрутизация между страницами реализована с использованием библиотеки React-Router. Для стилизации интерфейса использован стандартный JavaScript фреймворк Bootstrap.

Bootstrap - это открытый и бесплатный HTML, CSS и JS фреймворк, который используется веб-разработчиками для быстрого создания адаптивных дизайнов сайтов. Фреймворк Bootstrap используется не только независимыми разработчиками, но и целыми компаниями. Основная область его применения – это разработка фронтенд составляющих сайтов и интерфейсов админок. Среди аналогичных систем (Foundation, UIkit, Semantic UI, InK и др.) фреймворк Bootstrap является самым популярным. В сущности, Bootstrap - это просто набор файлов (CSS и JavaScript). После подключения этих файлов к странице вам станут доступны для вёрстки дизайна большое количество классов и готовых компонентов. Используя их можно очень быстро и качественно создать современный адаптивный дизайн сайта [6].

Bootstrap позволяет легко позиционировать элементы интерфейса, так как имеет, так называемую систему «сетки». Вся страница представляет собой одну большую матрицу, которая по умолчанию состоит из строк по 12 колонок. Регулируя размер компонентов интерфейса можно помещать до 12

элементов в одной строке, причём каждый элемент также представляет из себя строку из 12 колонок. Таким образом, используя вложенные друг в друга компоненты можно легко создать необходимый интерфейс.

За CSS стилизацию отвечают библиотеки Emotion и Styled.

Библиотека Emotion позволяет создавать CSS-классы так, что возможные повторяющиеся имена классов на разных уровнях интерфейса не будут препятствием. Имя CSS-класса формируется из названия самого класса, названия компонента, в котором он используется, а также особого, уникального, идентификатора. Таким образом можно выносить CSS-классы в отдельные файлы и не заботиться об уникальности их имён.

Библиотека Styled даёт возможность создавать собственные компоненты, опираясь на основные компоненты, например кнопка, button, присваивая новому компоненту имя, а также добавляя к нему CSS-стилизацию, можно затем использовать его как обычный компонент, например button.

Для валидации форм интерфейса, например, если неверно написан пароль, используется библиотека React-Hook-Form. Данная библиотека позволяет легко регистрировать определённые компоненты формы для определённых полей заполняемых форм, а также задавать правила валидации напрямую в этих компонентах, например, чтобы значение не превышало 10. Также позволяет установить определённые сообщения об ошибках для каждого правила валидации каждого поля и выводить эти сообщения если они появляются. С использованием TypeScript, использование React-Hook-Form становится ещё проще, так как можно заранее определить интерфейс для объекта, который будет заполнятся в форме и объявить React-Hook-Form что объект именно этого интерфейса заполняется в форме, что, например, заранее позволяет узнать тип данных определённого поля.

Интерфейс адаптивен под различные размеры экрана, так как Bootstrap предоставляет возможность указать размеры компонентов для различных размеров экрана устройства клиента.

Фронтэнд часть приложения содержит большое количество страниц, схема переходов между ними отличается для пациента и доктора из-за того, что они имеют доступ к разному функционалу приложения.

Схема переходов между страницами для пациента представлена ниже (рисунок 3.6).

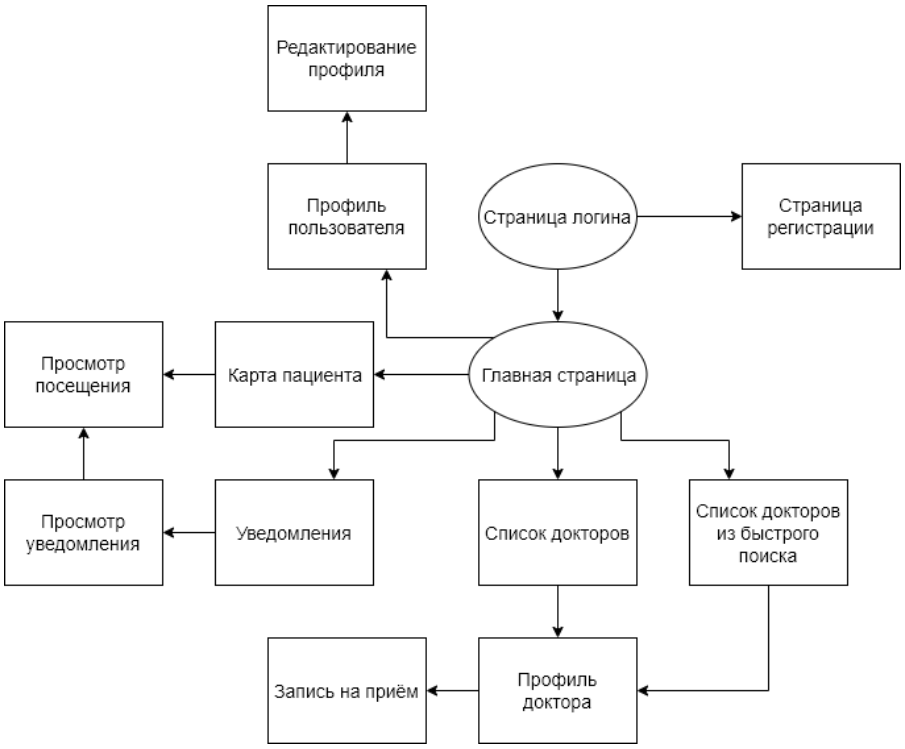

Рисунок 3.6 - Схема переходов между страницами для пациента

Благодаря навигационной панели пациент имеет доступ, например, к списку докторов, не только с главной страницы. Навигационная панель и заголовок сайта находятся на любой странице, таким образом, у пользователя всегда имеется доступ на любую из страниц, на которые есть переходы с главной страницы на схеме (рисунок 3.6).

Схема переходов между страницами для доктора представлена ниже (рисунок 3.7).

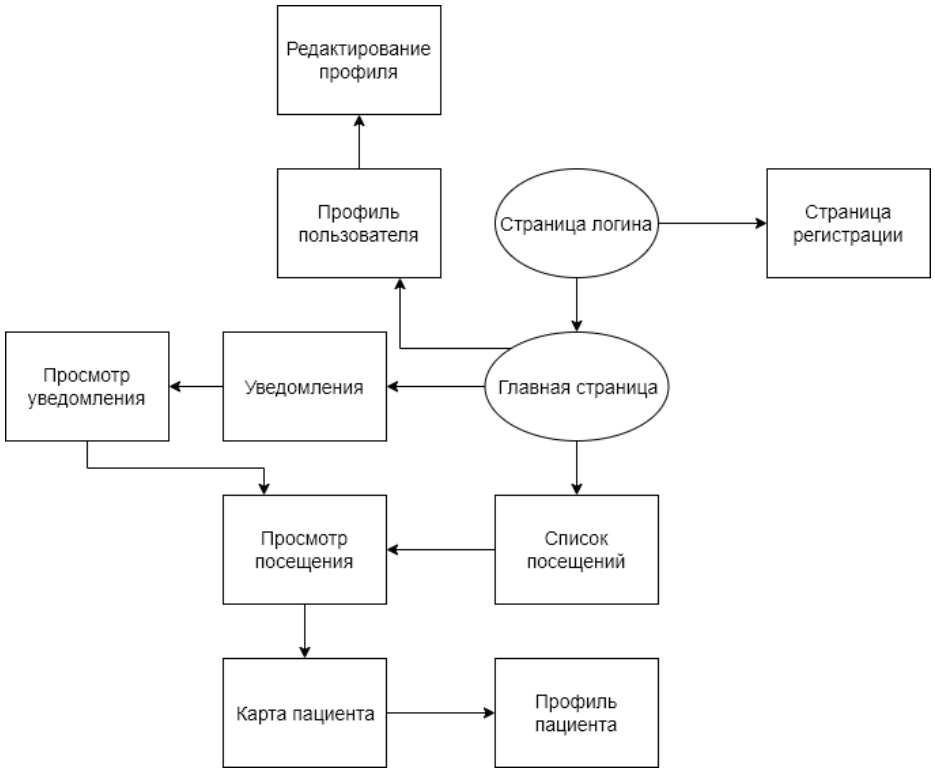

Рисунок 3.7 - Схема переходов между страницами для доктора

Доктор имеет немного отличный от пациента функционал, таким образом схема переходов между страницами отличается. Также, как и пациент, доктор имеет возможность переадресации при помощи навигационной панели и заголовка сайта.

Данные в приложении в большинстве случаев представлены в виде списка, например, список докторов, пример структуры списка в приложении представлен ниже (рисунок 3.8).

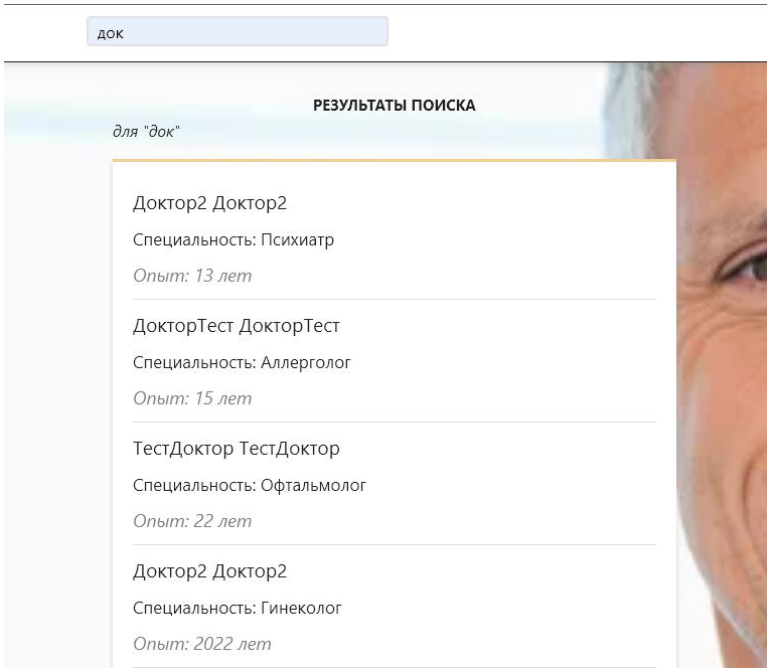

Рисунок 3.8 - Пример списка докторов в приложении

Также, любые данные из списка можно просмотреть индивидуально, например, профиль отдельного доктора из списка, пример представления отдельной сущности из списка представлен на рисунке 3.9.

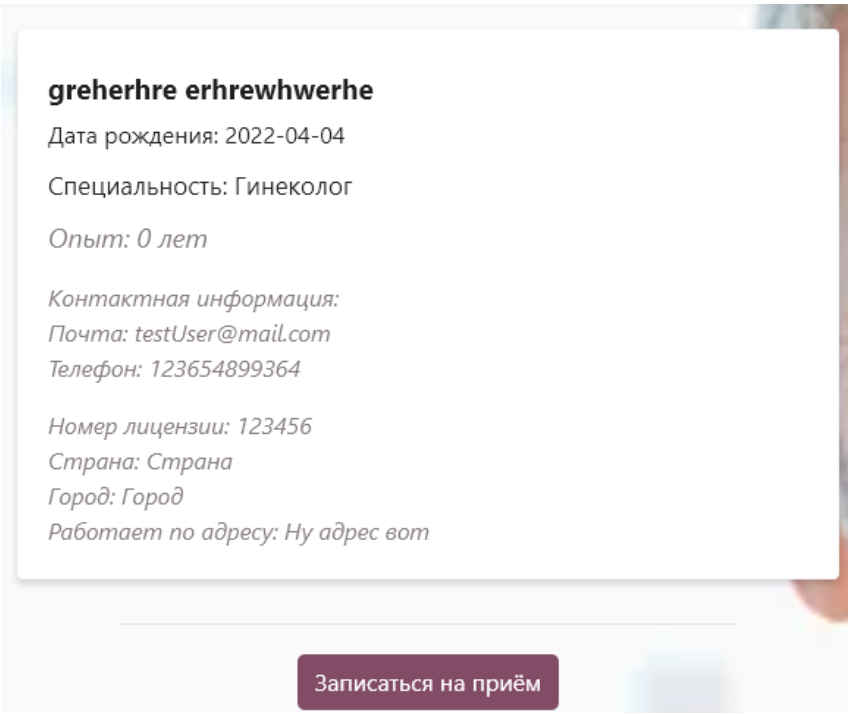

Рисунок 3.9 - Пример представления профиля доктора

Интерфейс приложения имеет функцию вывода окна подтверждения определённых действий, например, удаления аккаунта пользователя. Пример окна подтверждения представлен ниже (рисунок 3.10).

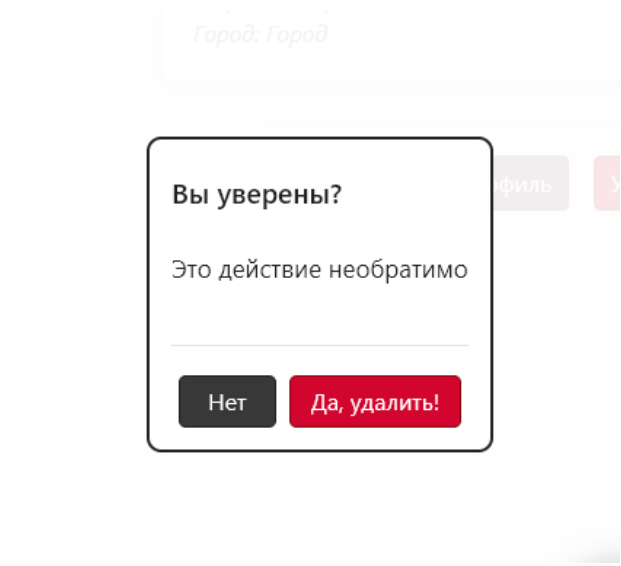

Рисунок 3.10 - Пример окна подтверждения