Министерство образования Республики Беларусь
ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра «ИСАП»

ОТЧЁТ
по преддипломной практике

Выполнил:
студент группы Ит-6
Лапко М. Л.
Руководитель от университета:
Дунина Е. Б.
Руководитель от предприятия ООО «Фабрика инноваций и решений»:
Медюха В. В.

Витебск 2022

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время каждый человек нуждается в периодическом посещении врачей различных специальностей, но сам процесс записи на приём зачастую занимает достаточно большой отрезок времени из-за того, что необходимо дойти до поликлиники/больницы, отстоять всю очередь, которая в большинстве случаев насчитывает от 5 человек, а затем добраться домой. Причём, в итоге может оказаться что требуемый доктор не принимает в удобное вам время или вообще в отпуске.

Конечно, практически во всех современный здравоохранительных учреждениях присутствует система записи посредством звонка, но, как показывает практика, даже на телефонной линии присутствует своеобразная «очередь», когда все телефонные операторы заняты обработкой заявок других пациентов.

При всех вышеописанных обстоятельствах невероятно удобной является система записи на приём посредством использования специализированного веб-сервиса. Бронирование времени для посещения посредством такого веб-сервиса занимает всего от 5 до 15 минут, в зависимости от специфики проблемы пациента и занятости врачей.

Сегодня пользование электронными услугами является простым для большинства современных людей и всё больше людей используют их каждый день. В этом и заключается актуальность разработки сервиса, описанного выше.

Целью данной практики является разработка программной системы «Веб-сервис для бронирования и управления посещениями врача».

Сформулируем основные задачи преддипломной практики:

- провести анализ предметной области;
- проанализировать и выбрать инструменты разработки программной системы;
- выполнить проектирование системы.

1 АНАЛИЗ ОБЪЕКТА

1.1 Описание предметной области

Посещение врача всегда являлось периодической необходимостью каждого отдельного человека, но процесс записи на приём зачастую вызывает определённые затруднения, особенно в последние несколько лет, из-за бушующей пандемии. Также дистанционное бронирование времени на приём рекомендуется из-за возможности передачи заболеваний от человека к человеку при личном общении с работником регистратуры. Телефонные линии поликлиник зачастую перегружены входящими звонками и поэтому довольно трудно дозвониться и выбрать время для посещения специалиста. В связи со всем вышеуказанным мной и была выбрана разработка информационного сервиса по организации именно этой области.

Для анализа избранной области рассмотрим диаграмму прецедентов (рисунок 1.1).

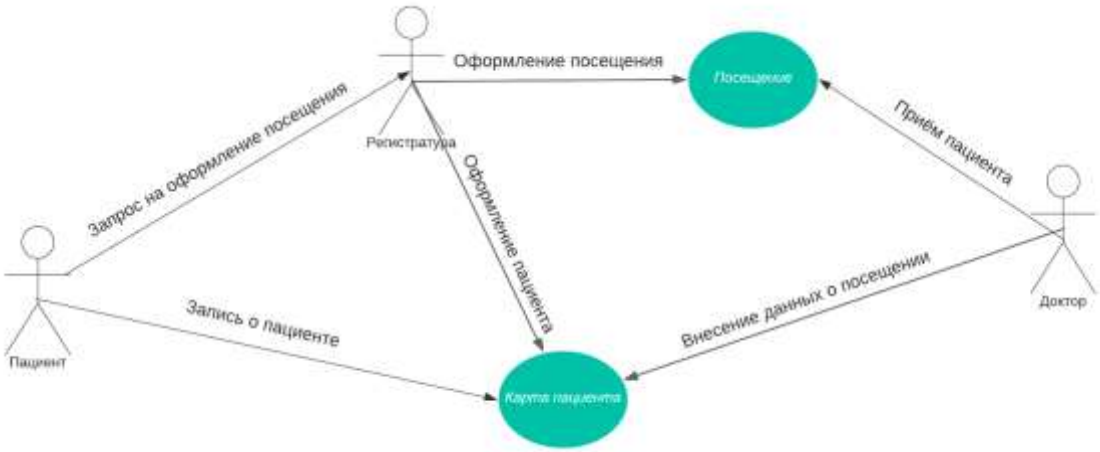

Рисунок 1.1 - Диаграмма прецедентов

Как видно на диаграмме, в рассматриваемой области присутствует 3 сущности: пациент, регистратура, доктор. На первый взгляд взаимодействия между сущностями довольно просты и интуитивно понятны. Единственное, что

делает пациент это подача заявления на посещение врача. У регистратуры же, наоборот, наибольшая роль: оформление и подтверждение посещения, а также оформление карты пациента, которая содержит всю необходимую информацию о пациенте, а также все посещения им специалистов. Доктор, непосредственно, принимает пациентов, то есть, обрабатывает посещения, в итоге он вносит данные о посещении в карту пациента.

Карты пациентов при всём этом обычно представлены лишь в физическом воплощении и не оцифровываются в электронный вариант, таким образом при утере карты невозможно восстановить данные. Учитывая это снова становится очевидным преимущество электронного варианта записи на приём, где карта пациента представлена строкой в базе данных.

1.2 Построение концептуальной модели предметной области

Концептуальная модель разрабатываемой системы представлена ниже (рисунок 1.2).

Можно увидеть, что изначально после входа на сайт пользователю необходимо авторизоваться, если пользователь не имеет аккаунта, то он может зарегистрироваться, причём при регистрации пользователь имеет выбор зарегистрировать доктора или пациента. Сама регистрация доктора отличается от регистрации пациента наличием полей для информации, специфической для специалиста, например, номер врачебной лицензии или специализация.

После авторизации/регистрации пользователь перенаправляется на главную страницу, которая отличается для доктора и пациента. Доктор может посетить свой профиль для редактирования информации, просмотреть уведомления, которые приходят при оформлении нового посещения к данному специалисту. Также доктор может просмотреть предстоящие для него посещения, каждое посещение содержит в себе ссылку на карту пациента, которую доктор также может просмотреть для ознакомления с пациентом и его заболеваниями. Доктор может редактировать информацию о посещении, например, указывать

заболевание, которое было найдено у пациента в результате приёма или добавлять определённое описание на своё усмотрение.

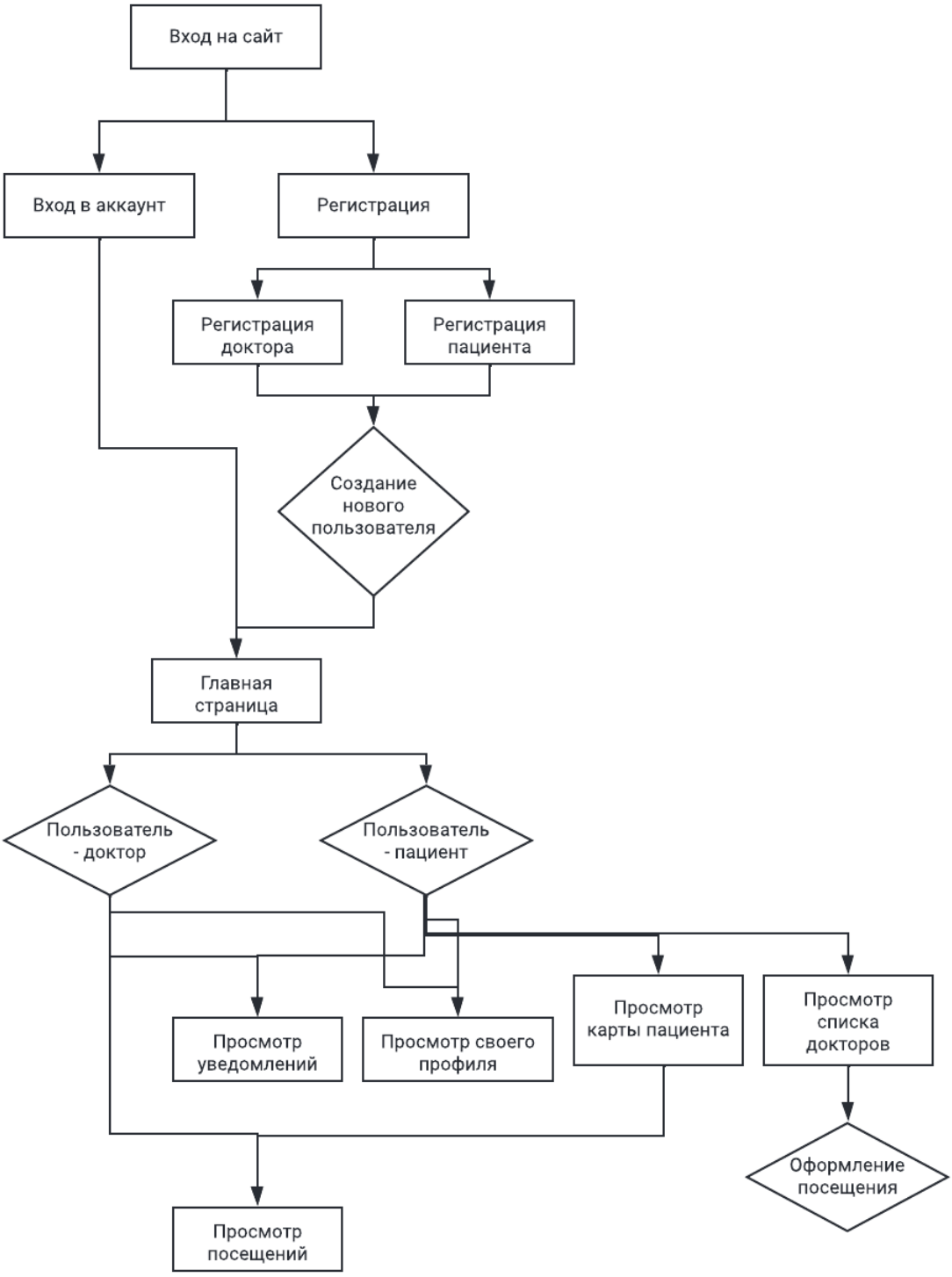

Рисунок 1.2 - Концептуальная модель

Пациент же имеет доступ к списку всех докторов, которых он может фильтровать различными способами: по специализации, опыту, фамилии, городу работы и так далее. Выбрав предпочтительного специалиста, пациент может оформить заявку на посещение.

Подобно доктору пациент может просмотреть свой профиль и свободно его редактировать, также пациент может напрямую просмотреть свою карту, которая содержит записи обо всех его посещениях. Также пациент может просмотреть свои уведомления, которые приходят при подтверждении его заявки на посещение и как напоминание о предстоящем посещении.

Можно заметить, что, в отличие от диаграммы прецедентов (рисунок 1.1) в концептуальной модели отсутствует регистратура. В отличие от реальной системы записи на приём, в веб-сервисе роль регистратуры частично исполняет сама система, сохраняя все данные и уведомляя пользователей, частично сам пациент, оформляя заявку на посещение и частично доктор, утверждая заявку.

2 ПОСТАНОВКА ЗАДАЧИ

2.1 Определение требований к программной системе

Требования определяются в зависимости от роли пользователя, так как пользователи разделяются на докторов и пациентов. Таким образом определим требования, разделяющие возможностей работы различных типов пользователей с информационной системой.

Возможности доктора:

- просмотр своего профиля;
- возможность редактирования профиля;
- возможность удаления профиля;
- просмотр уведомлений;
- просмотр предстоящих посещений;
- возможность редактирования самого посещения.

Возможности пациента:

- просмотр своего профиля;
- возможность редактирования профиля;
- возможность удаления профиля;
- просмотр уведомлений;
- просмотр списка докторов;
- возможность оформить запрос на посещение специалиста;
- просмотр своей карты;
- просмотр своих посещений.

Дополнительные требования:

- возможность новому пользователю зарегистрироваться как доктор или пациент;

- доктору приходит уведомление при записи нового пациента к нему на приём, пациенту приходит уведомление, когда доктор подтверждает посещение или как напоминание о предстоящем посещении.

2.2 Описание аналогов системы

В интернете существует множество примеров систем онлайн записи к доктору, практически у каждой поликлиники присутствует официальный сайт, где можно записаться на приём, также существуют сервисы, не относящиеся к какому-либо определённому медицинскому учреждению. Примеры таких систем:

- talon.by - самый популярный сервис по оформлению посещения онлайн в Беларуси;

- 2doc.by - сервис для записи на приём в большом количестве регионов Беларуси;

- omskzdrav.ru - региональный портал медицинских услуг, содержит возможность записи на приём к специалисту.

Представленные сервисы имеют возможность выбора определённого учреждения здравоохранения, чтобы впоследствии уметь выбор среди специалистов, работающих в этом учреждении. Некоторые сервисы дополнительно позволяют заранее выбрать предпочтительную область страны чтобы затем выбрать поликлинику.

Также данные интернет-ресурсы в большинстве своём имеют дополнительные возможности, зачастую относящиеся к определённой поликлинике, например: платные услуги, вызов врача на дом и так далее.

2.3 Обзор и обоснование выбора инструментальных средств

Приложение, разрабатывающееся в рамках дипломного проектирования, состоит из двух частей: бэкэнд и фронтэнд.

Для разработки бэкэнд части информационной системы был выбран язык C#. C# - это современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надёжных приложений, выполняющихся в .NET. C# относится к

широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

C# - объектно-ориентированный, ориентированный на компоненты язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО. В основном C# - объектно-ориентированный язык. Вы определяете типы и их поведение.

Вот лишь несколько функций языка C#, которые позволяют создавать надёжные и устойчивые приложения. Сборка мусора автоматически освобождает память, занятую недостижимыми неиспользуемыми объектами. Типы, допускающие значение null, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и восстановлению после них. Лямбда-выражения поддерживают приёмы функционального программирования. Синтаксис LINQ создаёт общий шаблон для работы с данными из любого источника. Поддержка языков для асинхронных операций предоставляет синтаксис для создания распределенных систем. В C# имеется Единая система типов. Все типы C#, включая типы-примитивы, такие, как int и double, наследуют от одного корневого типа object. Все типы используют общий набор операций, а значения любого типа можно хранить, передавать и обрабатывать схожим образом. Более того, C# поддерживает как определяемые пользователями ссылочные типы, так и типы значений. C# позволяет динамически выделять объекты и хранить упрощённые структуры в стеке. C# поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. C# предоставляет итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода [1].

Язык C# практически универсален. Можно использовать его для создания любого ПО: продвинутых бизнес-приложений, видеоигр, функциональных веб-приложений, приложений для Windows, macOS, мобильных программ для iOS и Android.

C# популярен за счет своей «простоты». Простоты для современных программистов и больших команд разработчиков, чтобы те могли в сжатые сроки создавать функциональные и производительные приложения. Этому способствуют нетипичные конструкции языка и специфичный синтаксис, помогающий максимально органично реализовать намеченные функции.

Популярность языка - ещё одно значимое преимущество. Большое количество поклонников C# способствуют его развитию. Также это благоприятно влияет на рост числа вакансий, связанных с разработкой на языке Microsoft. Программисты, хорошо знакомые с C#, востребованы в индустрии, несмотря на их большое и постоянно увеличивающееся количество [2].

Также в языке присутствует обилие синтаксического сахара, который делает тяжёлую жизнь программиста капельку слаще. Вместо того чтобы писать 100500 строк кода, присутствует возможность использовать готовую конструкцию, а компилятор сделает всю остальную работу. Но некоторые такие конструкции являются не самыми оптимальными с точки зрения производительности. Но все это перекрывается за счёт удобочитаемости кода и высокой скоростью разработки [3].

Сама разработка бэкэнд части выполнена при помощи платформы ASP.NET Core.

ASP.NET Core - это веб-инфраструктура с открытым исходным кодом, оптимизированная для облачных вычислений, для разработки современных веб-приложений, которые можно разрабатывать и запускать на Windows, Linux и Mac. Он включает в себя инфраструктуру MVC, которая теперь объединяет функции MVC и веб-API в единую среду веб-программирования. Он был переработан с нуля, чтобы быть быстрым, гибким, современным и работать на разных платформах. В дальнейшем ASP.NET Core - это фреймворк, который

можно использовать для веб-разработки с .NET. Также ASP.NET Core имеет схожие функции с MVC и веб-API [4].

Бэкэнд часть является веб-API. Платформа веб-API ASP.NET позволяет с легкостью создавать службы HTTP для широкого диапазона клиентов, включая браузеры и мобильные устройства. ASP.NET Web API - это идеальная платформа для сборки REST-приложений на базе .NET Framework [5].

Для написания приложения была выбрана среда программирования Microsoft Visual Studio.

Microsoft Visual Studio является средой программирования, разработанной компанией Microsoft. Эта среда позволяет создавать кроссплатформенные проекты на различных языках программирования, таких как Visual Basic, Visual C#, Visual C++, Visual F# и другие. Также она позволяет создавать программы, использующие в своей работе платформу .NET, которая позволяет использовать большой набор сервисов, реализующихся в виде промежуточного, независящего от базовой архитектуры, кода. Основной целью создания платформы .NET является возможность реализации разработчиками специальных сервисно-ориентированных программ, работающих на любых платформах.

MS Visual Studio позволяет разработчику иметь доступ к огромной коллекции различных функций, которые позволяют вести разработки для любой версии операционной системы семейства Windows, для интернет-приложений и мобильных приложений. Также среда программирования открывает широкие возможности в области облачных технологий. Эта среда открывает разработчику широкие возможности для реализации самых разных проектов, реализуя высокую производительность и независимость от особенностей оборудования.

Microsoft Visual Studio позволяет осуществлять проектирование программ, используя любые по размеру команды. Эта среда разработки предоставляет инструменты планирования для возможности внедрения методов последовательной разработки, а также для гибкого планирования. Используя

весь спектр возможностей, предоставляемых MS Visual Studio, можно реализовать максимально полную систему, наиболее удачно спроектировать любую архитектуру. Таким образом Microsoft Visual Studio представляет собой передовую среду разработки [6].

Фронтэед часть разработана при помощи JavaScript-библиотеки React при поддержке TypeScript.

React - библиотека, написанная на JavaScript, которая используется для работы с интерфейсами. В 2011 году её начали использовать для социальной сети Facebook, а уже в 2013 году библиотеку выложили в открытый доступ, и энтузиасты со всего мира начали создавать инструменты для расширения её возможностей. Разработчики используют React, чтобы создавать интерфейсы, которые способны менять контент без перезагрузки страницы. Благодаря этому сайты или нативные приложения быстро отзываются на действия пользователей. Можно добавлять товары в корзину без перезагрузки страницы или заполнять формы без переадресации.

React считается самой популярной библиотекой в мире, написанной на JS. Её популярность обеспечивается тем, что с помощью компонента можно решать разные задачи. Особенно, если использовать дополнительные инструменты, которые интегрируются в проекты и открывают доступ к нестандартным возможностям [7].

React предпочтительнее использовать по следующим причинам:

- технология SPA (single-page application, по-русски: “разработка одностраничных приложений”): React поможет вам создать одностраничное приложение. С помощью ReactJS вы сможете изменять (управлять/манипулировать) контент всей страницы с минимальным кодом;

- декларативный подход: React использует декларативный дизайн со всем синтаксическим сахаром, что помогает написать поддерживаемый код высокого уровня. Вам просто нужно определить цель, и React будет обрабатывать инструкции JavaScript DOM с учетом ситуаций, в которых они используются;

- компонент-управляемый пользовательский интерфейс: React основан на компонентной концепции. Компоненты являются многоразовыми строительными блоками в пользовательском интерфейсе. С помощью ReactJS вы можете создать инкапсулированный компонент, который управляет своими данными, и избежать сценария влияния на другие состояния и действия компонентов в DOM-дереве. Это только одна из характеристик компонент-управляемого интерфейса. Он также помогает повторно использовать код, разделять ответственность и избегать повторения [8].

TypeScript (TS, TScript или «тайпскрипт») - это язык программирования для веб-разработки, основанный на JavaScript. Делает код понятнее и надежнее, добавляет статическую типизацию (переменные привязаны к конкретным типам данных), а также может быть скомпилирован в JavaScript. TypeScript используют фронтенд- и бэкенд-разработчики.

TypeScript добавляет в язык строгую типизацию. Каждой переменной при создании присваивается определенный тип (type) — стандартный или созданный самим разработчиком. Создать тип можно в пределах возможностей языка: например, число от 1 до 31 для записи дня в месяц или массив из двух элементов для записи координат.

TypeScript помогает сократить время на выявление и устранение багов, которые иногда сложно найти в динамической среде JavaScript. С помощью TypeScript можно написать более понятный и читаемый код, который максимально описывает предметную область. Таким образом архитектура становится более выраженной.

Код, написанный на TypeScript, не выполнится напрямую в браузере. Поэтому TS - не самостоятельный язык, а именно языковая надстройка над JS. Для его работы нужен дополнительный этап - транспиляция, когда программное обеспечение преобразует написанный на TypeScript код в «чистый» JavaScript.

JS не требует установки в систему: его по умолчанию поддерживает любой браузер. А вот TypeScript понадобится установить, потому что для транспиляции необходим модуль tsc [9].

Код для фронтэнд части приложения написан при помощи редактора кода Visual Studio Code.

Visual Studio Code - это «бесплатный редактор, который помогает программисту писать код, помогает в отладке и исправлении кода с помощью метода IntelliSense». В обычных условиях это облегчает пользователю написание кода простым способом. Многие говорят, что это половина IDE и редактора, но решение остается за программистами. Visual Studio Code поддерживает несколько языков программирования, имеет кроссплатформенную поддержку, огромное количество расширений для упрощения создания кода, а также встроенную поддержку системы контроля версий Git.

Visual Studio Code поддерживает несколько языков программирования. Так что раньше программистам требовалась веб-поддержка: другой редактор для разных языков, но он имеет встроенную многоязычную поддержку. Это также означает, что он легко обнаруживает, если есть какая-либо ошибка или ссылка на другой язык, он сможет легко обнаружить её [10].

3 ПРОЕКТИРОВАНИЕ

3.1 Разработка архитектуры программного продукта

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

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 отдельно 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 [11].

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 [12].

Схема многоуровневой архитектуры представлена ниже (рисунок 3.1).

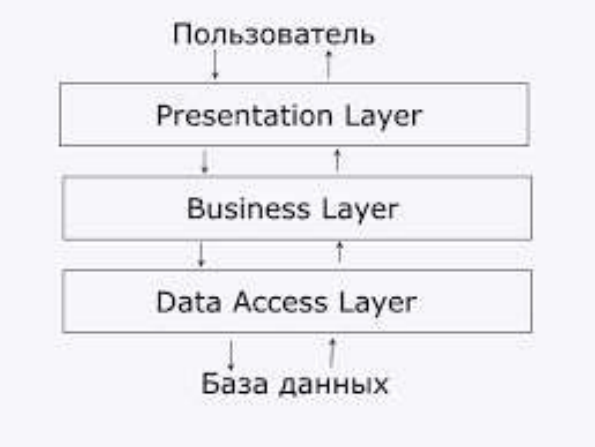

Рисунок 3.1 - Схема связей между слоями многоуровневой архитектуры

В приложении бэкэнд части 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 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 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 3.2).

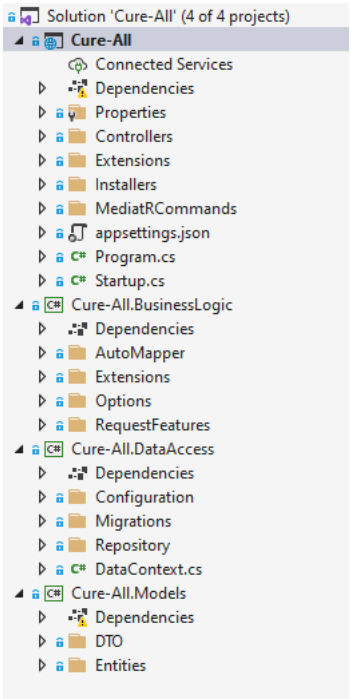

Рисунок 3.2 - Многоуровневая архитектура проекта в Visual Studio

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 3.3).

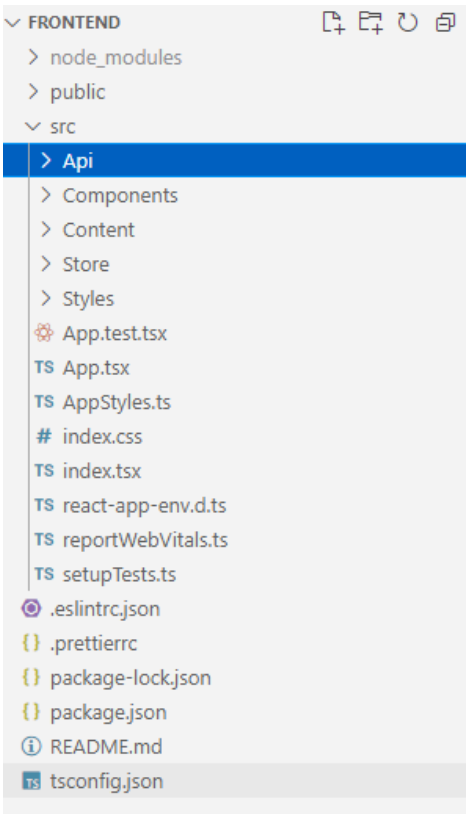

Рисунок 3.3 - Структура приложения в Visual Studio Code

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

3.2 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 [13].

Схема базы данных, отвечающей за хранение информации в проекте, представлена ниже (рисунок 3.4).

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

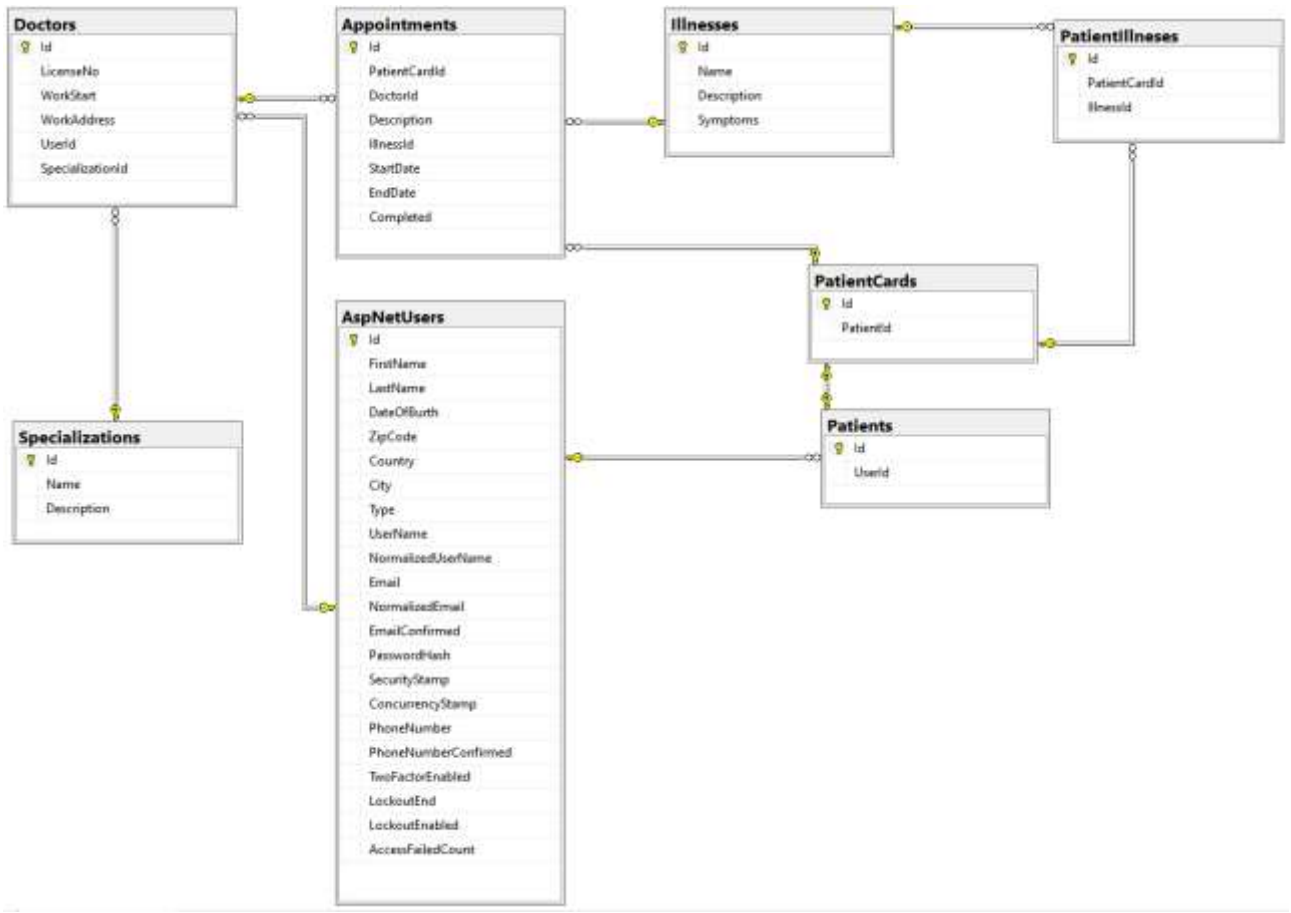

Рисунок 3.4 - Схема базы данных

разработки в среде SQL Server, а также даёт богатые возможности для создания различных сценариев SQL Server [14].

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 [15].

3.3 Описание реализации вариантов использования

Вариант использования - это связный блок функциональности, которую предоставляет классификатор (система, подсистема или класс). Этот блок описывает последовательность сообщений, которыми обменивается система и один или несколько внешних пользователей (актантов), а также действия, осуществляемые при этом системой.

Вариант использования служит для определения некой части поведения классификатора (которым можно также считать подсистему и даже всю систему целиком), без указания на его внутреннюю структуру. Каждый вариант использования описывает некую услугу, которую предоставляет своим пользователям классификатор. Иначе говоря, это некоторый способ использования классификатора, который виден со стороны. Вариант

использования описывает всю последовательность сообщений, которую начинает пользователь (и модели - актант), в терминах взаимодействия между пользователем и классификатором, включая ответы классификатора. К взаимодействию относятся только коммуникации между системой и актантами. Внутреннее поведение и реализация скрыты. Все множество вариантов использования какого-либо классификатора или системы разделяет и полностью описывает его поведение. Каждый вариант использования представляет собой некую разумную долю функциональности, которая доступна пользователям. Обратите внимание, что под термином пользователь следует понимать не только людей, но и компьютеры, а также прочие объекты. Актант представляет собой некую идеализацию намерений пользователя, а не самого этого пользователя. Один реальный пользователь может соответствовать нескольким актантам, а один актант может представлять одно и то же намерение сразу нескольких пользователей.

Вариант использования включает в себя описание основного поведения, осуществляемого в ответ на запрос пользователя, а также все возможные варианты этого поведения, например альтернативные последовательности, исключительное поведение и обработка ошибок. Для большего удобства варианты использования можно группировать в пакеты [16].

Диаграмма вариантов использования приложения пользователями представлена ниже (рисунок 3.5).

На схеме изображено 3 типа пользователей: анонимный посетитель, доктор, пациент. После авторизации или регистрации у анонимного пользователя появляется роль доктора или пациента. В зависимости от роли пользователям доступен различный функционал интерфейса, тем не менее, у всех типов пользователей имеются несколько одинаковых возможностей, например просмотр и редактирование своего профиля или просмотр уведомлений.

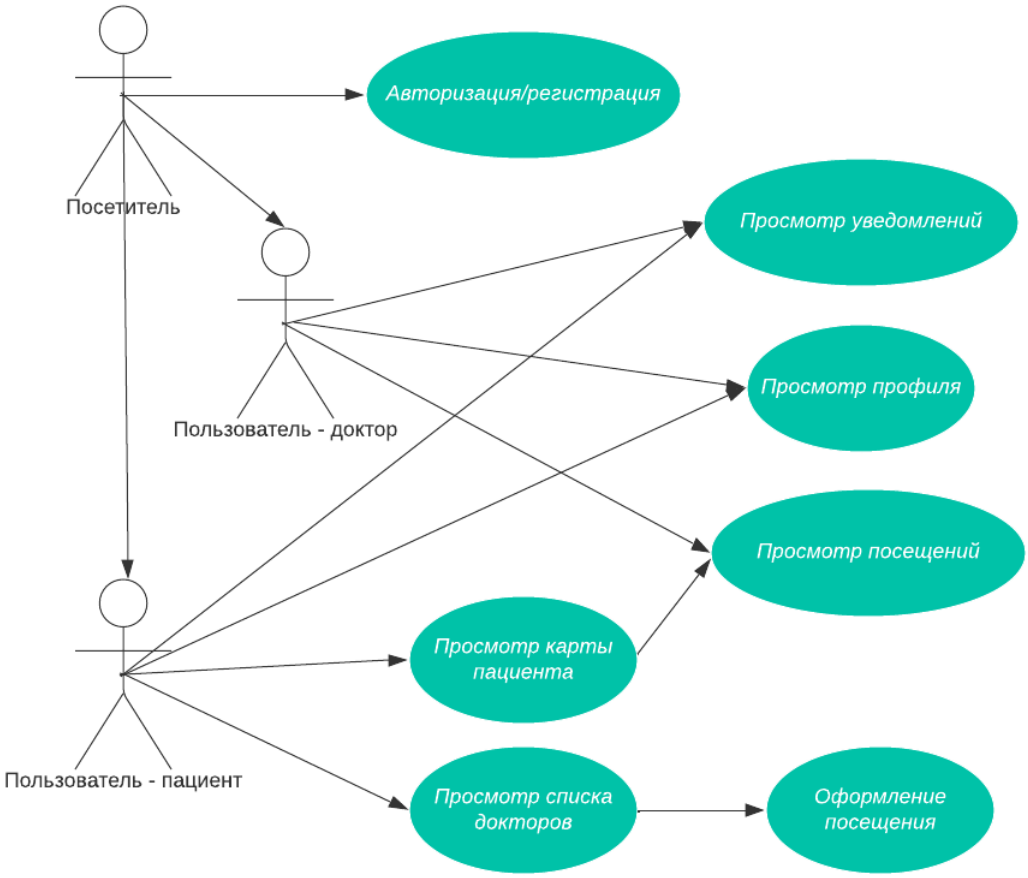

Рисунок 3.5 - Диаграмма вариантов использования

Из некоторых вариантов использования вытекают другие варианты, например из варианта просмотра докторов вытекает вариант оформления посещения для пациента.

ЗАКЛЮЧЕНИЕ

Результатами преддипломной практики являются: сформированные требования к программной системе, описаны аналоги системы, проведён анализ предметной области для разрабатываемого приложения, были проанализированы и выбраны инструменты разработки программной системы, а также было выполнено проектирование системы.

В первом разделе был выполнен анализ предметной области, построена диаграмма прецедентов, построена концептуальная модель.

Во втором разделе была проведена постановка задачи: определены требования к разрабатываемой системе, описаны аналоги системы, а также выбраны и проанализированы инструменты разработки.

В третьем разделе было проведено проектирование: разработана архитектура программного продукта, проведено проектирование структур хранения данных, а также описаны варианты использования программной системы.

В ходе преддипломной практики были выполнены все задачи, поставленные в начале.

Таким образом, можно сделать вывод, что цель преддипломной практики была полностью достигнута.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Краткий обзор языка C# [Электронный ресурс] / Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ - Дата доступа: 10.04.2022

2. Язык программирования C#: краткая история, возможности и перспективы [Электронный ресурс] / Режим доступа: https://timeweb.com/ru/community/articles/chto-takoe-csharp - Дата доступа: 10.04.2022

3. C# - Преимущества и недостатки [Электронный ресурс] / Режим доступа: https://shwanoff.ru/plus-minus-c-sharp/ - Дата доступа: 10.04.2022

4. ASP.NET Core - Обзор [Электронный ресурс] / Режим доступа: https://coderlessons.com/tutorials/microsoft-technologies/izuchite-asp-net-core/asp-net-core-obzor/ - Дата доступа: 11.04.2022

5. Обзор ASP.NET [Электронный ресурс] / Режим доступа: https://docs.microsoft.com/ru-ru/aspnet/overview - Дата доступа: 12.04.2022

6. Описание среды разработки MS Visual Studio [Электронный ресурс] / Режим доступа: https://prog.bobrodobro.ru/63233 - Дата доступа: 12.04.2022

7. Что такое React и почему он так популярен [Электронный ресурс] / Режим доступа: https://liquidhub.ru/blogs/blog/chto-takoe-react - Дата доступа: 13.04.2022

8. Знакомство с ReactJS на базовом уровне [Электронный ресурс] / Режим доступа: https://nuancesprog.ru/p/13297/ - Дата доступа: 13.04.2022

9. TypeScript [Электронный ресурс] / Режим доступа: https://blog.skillfactory.ru/glossary/typescript/ - Дата доступа: 14.04.2022

10. Что такое код Visual Studio? [Электронный ресурс] / Режим доступа: https://ru.education-wiki.com/3958588-what-is-visual-studio-code - Дата доступа: 14.04.2022

11. Что такое n-уровневая архитектура? [Электронный ресурс] / Режим доступа: https://ru.theastrologypage.com/n-tier-architecture - Дата доступа: 15.04.2022

12. 4 типа архитектуры программного обеспечения [Электронный ресурс] / Режим доступа: https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724 - Дата доступа: 15.04.2022

13. Microsoft SQL Server Краткое описание [Электронный ресурс] / Режим доступа: https://flexberry.github.io/ru/gbt_mssql.html - Дата доступа: 16.04.2022

14. SQL Server Management Studio — единое средство управления и среда разработки в MS SQL Server 2012 [Электронный ресурс] / Режим доступа: https://tavalik.ru/sql-server-management-studio/ - Дата доступа: 16.04.2022

15. Redux [Электронный ресурс] / Режим доступа: https://blog.skillfactory.ru/glossary/redux/ - Дата доступа: 17.04.2022

16. use case (вариант использования) [Электронный ресурс] / Режим доступа: https://openu.ru/Books/UML/Use_case.asp - Дата доступа: 17.04.2022