# 3 ПРОЕКТИРОВАНИЕ

## 3.1 Разработка архитектуры программного продукта

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

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

Разраб.

Лапко М. Л.

Провер.

Дунина Е.Б.

Реценз.

Н. Контр.

Соколова А.С.

Утверд.

Казаков В.Е.

ПОСТАНОКА ЗАДАЧИ

Лит.

Листов

УО «ВГТУ» каф. ИСАП гр. Ит-6

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.

1. 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].

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

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).

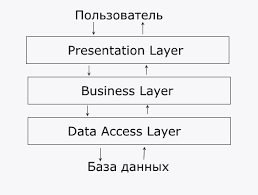


Рисунок 3.1 - Схема связей между слоями многоуровневой архитектуры

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.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

В обозревателе решений Visual Studio архитектура проекта выглядит следующим образом (рисунок 3.2).

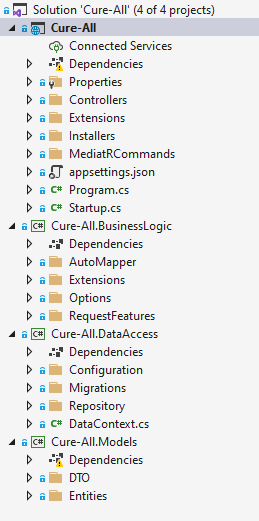


Рисунок 3.2 - Многоуровневая архитектура проекта в Visual Studio

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:

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

* 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).

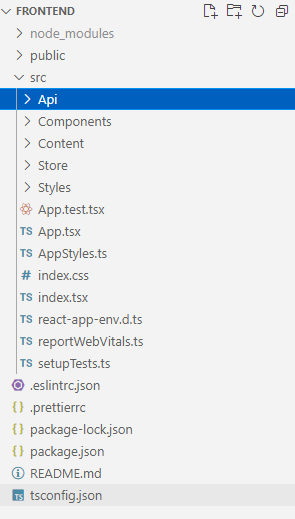


Рисунок 3.3 - Структура приложения в Visual Studio Code

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].

Изм.

Лист

№ докум.

Подпись

Дата

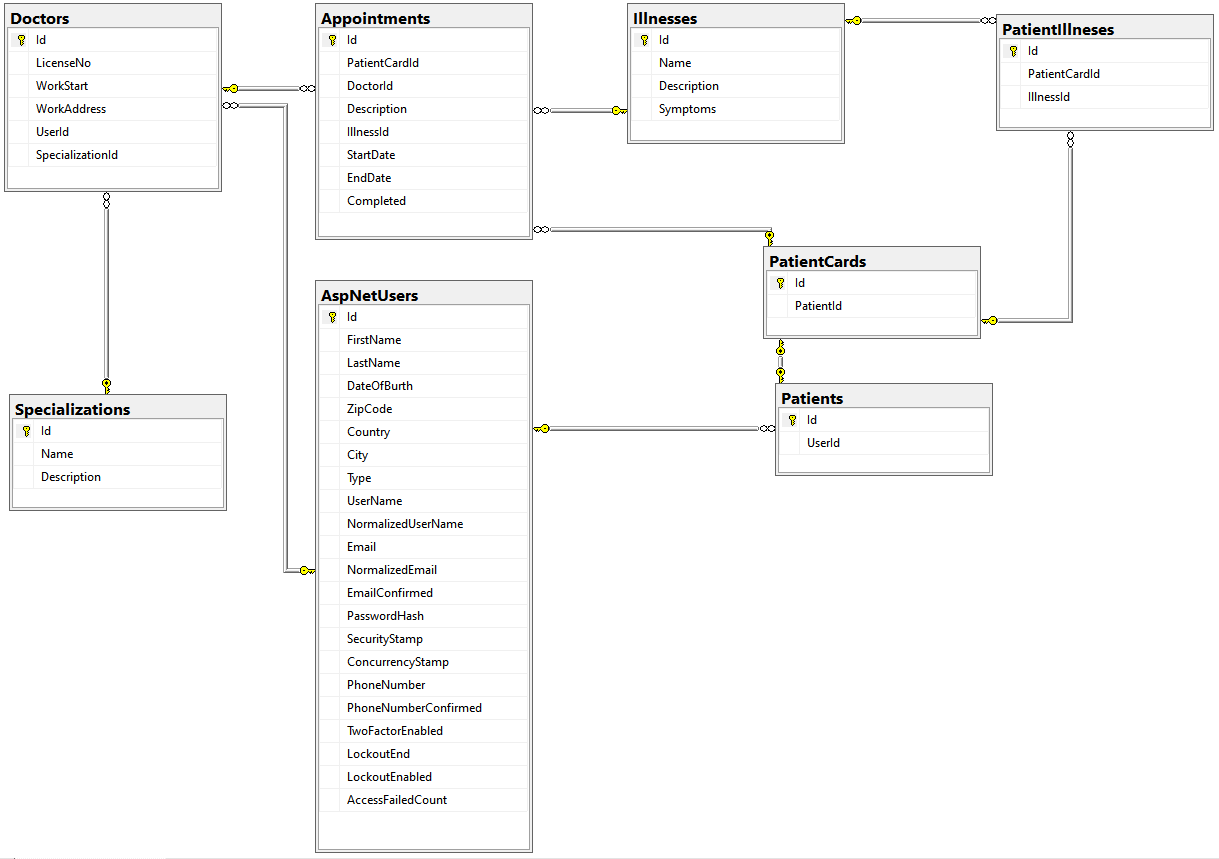
Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

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

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

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;

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

* 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.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ

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.

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

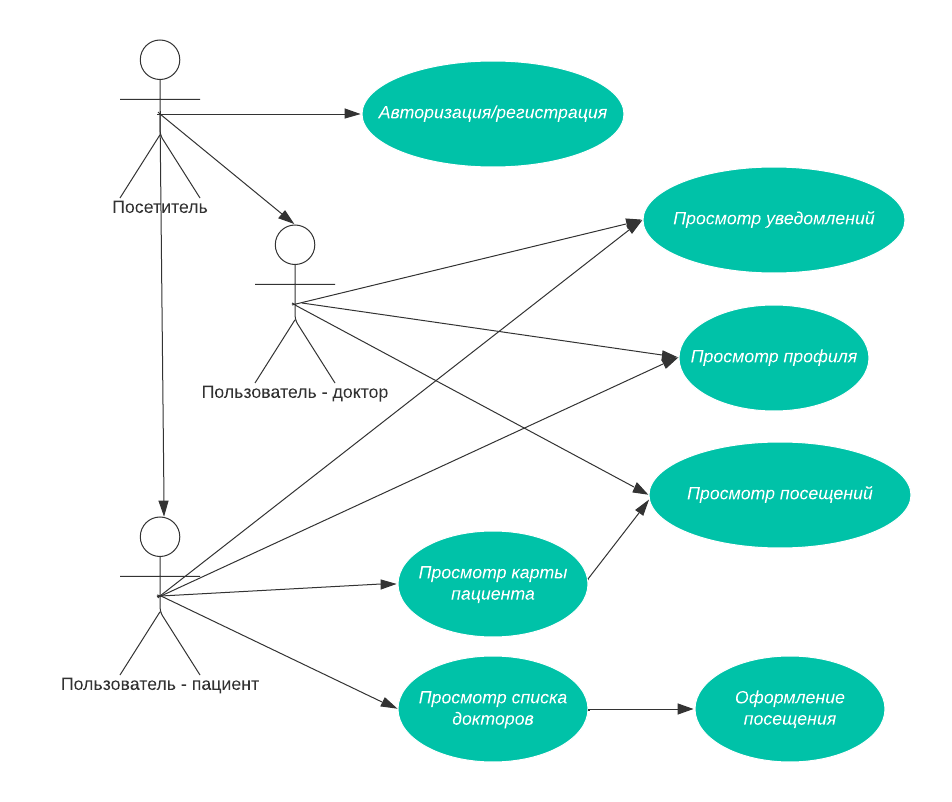


Рисунок 3.5 - Диаграмма вариантов использования

В приложении реализована навигационная панель, таким образом пользователи с любой страницы всегда могут перейти на любую другую.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ДП.006 1-40 05 01-01 ПЗ