МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П.О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

Отчет по преддипломной практике

на тему: «Программный комплекс дневник диабетика»

Подготовил: студент гр. ИП-42

Суховенко Э. С.

Руководитель от предприятия:

ст. программист

Семченко А. С.

Руководитель:

к.т.н., доцент

Прокопенко Д.В.

Дата проверки:

Дата допуска к защите:

Дата защиты:

Оценка работы:

Подписи членов комиссии: *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Гомель 2023

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc133485003)

[1 Анализ предметной области 4](#_Toc133485004)

[1.1 Анализ предметной области 4](#_Toc133485005)

[1.2 Аналитический обзор существующих аналогов 5](#_Toc133485006)

[1.3 Обзор технологий для реализации программного обеспечения 6](#_Toc133485007)

[1.4 Анализ инструментальных средств автоматизации разработки и тестирования 11](#_Toc133485008)

[1.5 Постановка задачи на дипломное проектирование 15](#_Toc133485009)

[2 Архитектура программного обеспечения 17](#_Toc133485010)

[2.1 Общая структура приложения 17](#_Toc133485011)

[2.2 Функциональная модель программного комплекса 20](#_Toc133485012)

[2.3 Информационная модель программного комплекса 23](#_Toc133485013)

[Заключение 30](#_Toc133485014)

[Список использованной литературы 31](#_Toc133485015)

# **ВВЕДЕНИЕ**

В данном отчёте представлена информация о прохождении преддипломной практики.

Целью практики являлся анализ предметной области и создание архитектуры программного обеспечения приложения.

Я, Суховенко Эдуард Сергеевич, в период с 23.03.2023 г. по 19.04.2023 г. проходил преддипломную практику в компании «*LeverX International*» (далее – *LeverX*). Руководителем практики был назначен ст. программист Семченко Александр Сергеевич.

*LeverX* ­­­­­­­– это международная компания-консультант и разработчик программного обеспечения, основанная в 2003 году в Германии. Компания специализируется на реализации цифровых преобразований в промышленности и предоставляет услуги в области цифровой трансформации, управления жизненным циклом продуктов, управления проектами и консультаций.

*LeverX* имеет свои офисы в разных странах мира, включая Германию, США, Канаду, Беларусь, Россию, Украину и Китай. Компания работает с клиентами в различных отраслях, таких как автомобилестроение, машиностроение, транспорт и логистика, химическая промышленность, энергетика, медицина и другие.

Основные направления деятельности *LeverX* включают в себя:

* разработка программного обеспечения и создание цифровых решений для автоматизации бизнес-процессов;
* консалтинг и поддержка в области управления жизненным циклом продуктов;
* консультации и поддержка в области управления проектами и управления портфелями проектов.

*LeverX* также сотрудничает с крупными компаниями-партнерами, такими как *SAP*, *Siemens*, *Oracle*, *PTC* и другими, чтобы обеспечить своих клиентов наилучшими цифровыми решениями.

Практику я непосредственно проходил в отделе разработки веб-сайтов и мобильных приложений.

Во время прохождения практики передо мной были поставлены следующие задачи:

* анализ предметной области;
* анализ архитектуры программного комплекса;
* обзор и анализ информации по разработке веб-приложений;
* разработка функциональной модели приложения;
* разработка структуры и состава информационного обеспечения приложения;
* подготовка отчёта по преддипломной практике.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Анализ предметной области

Анализ предметной области «Автоматизация обслуживания дневника диабетика» позволяет основные аспекты, которые приведены ниже.

Мониторинг состояния здоровья: пациенты могут отслеживать изменения уровня глюкозы в крови и принимаемые меры (например, дозировку инсулина) в течение дня, недели, месяца или даже года. Такой мониторинг помогает выявлять тенденции и понимать, как изменения питания, физической активности и других факторов влияют на уровень глюкозы.

Расчет дозировки инсулина: приложение позволяет автоматически рассчитывать дозировку инсулина в зависимости от уровня глюкозы и других параметров, таких как вес, рост и физическая активность.

Напоминания о приеме лекарств и контроль назначений: приложение предоставляет возможность создавать напоминания о приеме лекарств, мероприятиях и других назначениях, которые помогают пациентам не пропустить важные моменты контроля здоровья.

Ведение ежедневника питания: приложение предлагает возможность вести ежедневник питания, что помогает контролировать потребляемые продукты, а также просматривать сводку питания за определенный период.

Возможность совместного использования с близкими: пациенты могут делиться данными с близкими (родственниками, друзьями, врачами), что позволяет им лучше понимать состояние здоровья пациента и оказывать необходимую поддержку.

Предоставление отчетов и статистики: приложение предоставляет возможность просматривать отчеты и статистику по уровню глюкозы в крови, питанию, дозировке инсулина и другим параметрам. Это позволяет пациентам более глубоко анализировать данные и принимать более обдуманные решения касательно их здоровья.

Возможность проведения онлайн консультаций с квалифицированными докторами. Это позволит предотвратить возможные ошибки в системе рекомендаций, а также поможет пациенту более детально разобраться в заболевании, если тематических материалов было недостаточно или, если по ним появились какие-то вопросы.

Для автоматизации этих процессов может быть использована специализированная информационная система, разработанная с учетом конкретных потребностей пациентов. Она будет включать в себя базу данных для хранения информации о пациентах и их родственниках, модули для контроля доступа, а также возможности для докторов добавлять и изменять рекомендации по приему инсулина для конкретного пациента.

Таким образом, разработка автоматизированной системы позволит упростить и ускорить процессы ведения дневника диабетика, повысить эффективность использования тематических материалов и улучшить качество жизни пациентов. Благодаря использованию базы данных, все данные будут храниться в одном месте, и пациенты смогут быстро получать необходимую информацию о заболевании и их статистике. А модули контроля доступа позволят обеспечить конфиденциальность персональных данных пользователей.

Разработка системы, также может способствовать улучшению качества жизни пациентов с сахарным диабетом, предоставляя им удобный и функциональный инструмент для мониторинга своего состояния и контроля за заболеванием. Приложение может стать незаменимым помощником в повседневной жизни пациентов, помогая им следить за показателями сахара в крови, приемом лекарств и упражнений, а также предоставляя необходимую информацию о заболевании и рекомендации по правильному образу жизни. Благодаря удобному интерфейсу и доступности функционала, приложение может повысить мотивацию пациентов следить за своим здоровьем и привести к улучшению результатов лечения и профилактики заболевания.

* 1. Аналитический обзор существующих аналогов

Существует множество методов и средств автоматизации дневника диабетика, которые помогают облегчить процесс ведения дневника и улучшить контроль уровня глюкозы в крови у людей, страдающих диабетом.

Одним из наиболее распространенных методов является использование бумажного дневника диабетика, в котором пользователь вручную записывает данные о своем здоровье. Также существуют электронные дневники, которые могут быть загружены на персональный компьютер или мобильное устройство. Эти дневники могут предоставлять возможности для ввода данных, просмотра графиков и отчетов, а также синхронизации с другими устройствами и приложениями.

Некоторые современные медицинские устройства, такие как глюкометры и инсулиновые насосы, также могут автоматически собирать и передавать данные о здоровье пользователя в электронный дневник диабетика. Это помогает пользователям сохранять более точные и надежные записи о своем здоровье и легче выявлять тенденции и изменения в уровне глюкозы в крови.

Кроме бумажных носителей для ведения дневника диабетика существуют и программные продукты, например, *DiaMeter* – это онлайн-сервис для учета уровня глюкозы в крови и автоматического составления дневника диабетика. Этот сервис является эффективным инструментом для улучшения контроля уровня глюкозы в крови и помогает диабетикам принимать более осознанные решения в отношении своего лечения и образа жизни. *DiaMeter* позволяет пользователям вводить данные о своем уровне глюкозы в крови, а также другие важные медицинские данные, такие как давление, вес и уровень холестерина. Пользователи могут использовать этот сервис для создания дневника диабетика и отслеживания изменений в своем здоровье. Одной из ключевых особенностей *DiaMeter* является возможность получения персонализированных рекомендаций и советов по уходу за здоровьем от медицинских экспертов. Это помогает пользователям более осознанно подходить к своему лечению и принимать более эффективные решения в отношении своего здоровья. Кроме того, *DiaMeter* имеет функцию анализа данных и построения графиков, которые помогают пользователям выявлять тенденции и изменения в своем уровне глюкозы в крови. Это может быть особенно полезным для людей, которые страдают от диабета и нуждаются в более тщательном контроле своего здоровья. Сервис *DiaMeter* также имеет мобильное приложение, которое обеспечивает доступ к данным пользователя в любом месте и в любое время. Это делает использование сервиса более удобным и доступным для пользователей, которые часто находятся в движении. Однако, следует отметить, что *DiaMeter* платный сервис, и цены могут варьироваться в зависимости от уровня подписки и доступных функций. Кроме того, для использования некоторых функций сервиса, таких как интеграция с другими медицинскими устройствами, может потребоваться дополнительное оборудование.

Существует большое количество различных программных продуктов и мобильных приложений, предназначенных для учета данных о глюкозе в крови, приемах пищи и физических нагрузках. Однако, несмотря на широкий выбор существующих систем автоматизации дневника диабетика, многие из них имеют ограниченный функционал и не удовлетворяют потребностям пользователей в полной мере. Также многие системы имеют ограничения в совместимости с различными устройствами и операционными системами, что может быть неудобно для пользователей, использующих различные устройства.

Таким образом, разработка собственного программного комплекса дневника диабетика может быть эффективным решением для удовлетворения потребностей пользователей в полной и точной автоматизации учета данных о глюкозе в крови, приемах пищи и физических нагрузках, а также для обеспечения более высокого уровня безопасности и защиты данных пользователя.

* 1. Обзор технологий для реализации программного обеспечения

Веб-приложения могут быть написаны на разных языках программирования и фреймворках, но одним из наиболее популярных стеков технологий для разработки веб-приложений является стек *FERN*. *FERN* – это аббревиатура, состоящая из четырех популярных инструментов для веб-разработки: *Firebase*, *Express*, *React* и *Node.js*. На рисунке 1.1 представлены компоненты основные компоненты стека *FERN* и архитектура их взаимодействия.



1. – Компоненты стека *FERN*

В основе этих фреймворков и библиотек находятся языки программирования *JavaScript* и *TypeSctipt*.

*JavaScript* – мультипарадигменный (с одновременным использованием множества парадигм) язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией спецификации *ECMAScript* (стандарт *ECMA-262*). Его обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты:

* динамическая типизация;
* слабая типизация;
* автоматическое управление памятью;
* прототипное программирование;
* функции как объекты первого класса.

*JavaScript* включает в себя объектную модель браузера – браузер-специфичная часть языка, являющаяся прослойкой между ядром и объектной моделью документа. Основное предназначение объектной модели браузера — управление окнами браузера и обеспечение их взаимодействия. Каждое из окон браузера представляется объектом *window*, центральным объектом *DOM*. Объектная модель браузера на данный момент не стандартизирована, однако спецификация находится в разработке *WHATWG* и *W3C*.

В *JavaScript* используется в *AJAX*, популярном подходе к построению интерактивных пользовательских интерфейсов веб-приложений, заключающемся в «фоновом» асинхронном обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью и интерфейс веб-приложения становится быстрее, чем это происходит при традиционном подходе (без применения *AJAX*) *[*2*]*.

*TypeScript* – это язык программирования, который является расширением *JavaScript* и добавляет в него статическую типизацию. Он был разработан компанией *Microsoft* и позволяет улучшить производительность и надежность кода. *TypeScript* также поддерживает объектно-ориентированное программирование и может быть использован для создания сложных веб-приложений. Главное отличие между *JavaScript* и *TypeScript* заключается в том, что *TypeScript* предоставляет возможность определения типов переменных, функций и объектов на этапе компиляции, что упрощает обнаружение ошибок в коде и облегчает его поддержку. Также *TypeScript* позволяет использовать новые возможности языка *JavaScript*, которые еще не поддерживаются стандартом *ECMAScript*. Оба языка широко используются в веб-разработке и могут быть использованы для создания динамических веб-приложений. Однако, при использовании *TypeScript* возможно повышение качества кода и его поддержки, благодаря статической типизации и дополнительным возможностям языка *[*3*]*.

*Firebase* – это платформа для разработки мобильных и веб-приложений, которая предоставляет инструменты для создания приложений без необходимости написания серверного кода и настройки инфраструктуры. *Firebase* предоставляет широкий спектр сервисов, включая базы данных в реальном времени, хранение файлов, аутентификацию, аналитику, отправку уведомлений и другие. *Firebase* использует облачную инфраструктуру *Google* и обеспечивает высокую масштабируемость и производительность для приложений любого размера. Она предоставляет возможность создавать многоуровневые приложения с поддержкой авторизации пользователей и безопасной передачи данных. *Firebase* может использоваться как в качестве *backend* для веб-приложений, так и для мобильных приложений на платформах *Android* и *IOS*. Она имеет открытое *API*, что позволяет легко интегрировать ее в различные приложения и сервисы *[*12*]*.

*Firebase* предоставляет базу данных в реальном времени, которая является хранилищем данных на сервере, доступным для приложений. Эта база данных работает в режиме реального времени, что означает, что изменения, сделанные в базе данных, мгновенно отображаются в приложении, не требуя дополнительного обновления страницы. База данных имеет древовидную структуру данных и работает с *JSON*-объектами. Каждый элемент в базе данных представляет собой *JSON*-объект, который имеет уникальный ключ и может содержать несколько полей.

*Firebase* также обеспечивает синхронизацию данных между различными клиентами, что делает ее идеальным выбором для создания многопользовательских приложений, таких как чаты, онлайн-игры и другие. Также платформа предоставляет бесплатный тарифный план, который включает большинство основных функций, что делает его доступным для малых и средних проектов. Для более крупных проектов доступны платные тарифы с более широкими возможностями и функциональностью *[*1*]*.

*Express.js* – это легковесный фреймворк для *Node*.*js*, который используется для разработки веб-приложений и *API*. Он предоставляет удобный и гибкий механизм для обработки запросов и ответов, маршрутизации и создания модульной структуры приложений.

Основные преимущества *Express*.*js* приведены ниже.

Удобство: *Express*.*js* облегчает разработку веб-приложений благодаря простому и интуитивно понятному *API*. Он позволяет быстро создавать маршруты, обрабатывать запросы и ответы, а также работать с различными *middleware*-пакетами.

Гибкость: *Express*.*js* позволяет разработчикам создавать приложения с различными функциональными возможностями. Он не навязывает строгую структуру приложения, что дает возможность гибко настраивать его под конкретные задачи.

Масштабируемость: *Express*.*js* позволяет легко масштабировать приложения, что особенно важно для больших и сложных проектов. Он поддерживает работу с кластерами и позволяет распределять нагрузку между несколькими серверами.

Поддержка *middleware*: *Express*.*js* предоставляет широкие возможности для работы с *middleware*, что позволяет улучшать функциональность приложения и повышать его безопасность. С помощью *middleware* можно добавлять авторизацию, обработку ошибок, логгирование и многое другое.

Большое сообщество: *Express*.*js* имеет большое сообщество разработчиков, которые создают и поддерживают множество пакетов и расширений. Это позволяет разработчикам быстро решать задачи и получать поддержку при возникновении проблем.

*Express*.*js* используется для создания различных типов приложений, включая *API*, веб-серверы, приложения для обработки данных и многие другие. Он позволяет быстро создавать и масштабировать приложения, обеспечивая при этом гибкость и удобство разработки *[*4*]*.

*React* – это библиотека *JavaScript*, разработанная *Facebook*, которая используется для создания пользовательских интерфейсов. *React* использует декларативный подход для описания компонентов пользовательского интерфейса, что делает его более простым и понятным для разработчиков. Он позволяет создавать переиспользуемые компоненты, которые могут быть легко использованы для создания сложных пользовательских интерфейсов.

Основные преимущества *React* приведены ниже.

Декларативный подход: *React* использует декларативный подход для описания пользовательского интерфейса, что делает его более понятным для разработчиков. Он позволяет описывать, как должен выглядеть интерфейс, а не как его создать.

Переиспользуемые компоненты: *React* позволяет создавать переиспользуемые компоненты, которые могут быть легко использованы для создания сложных пользовательских интерфейсов. Это сокращает время разработки и улучшает качество кода.

Эффективный: *React* использует виртуальный *DOM*, который позволяет изменять только те элементы, которые действительно изменились. Это уменьшает количество дорогостоящих операций, связанных с обновлением интерфейса, и улучшает производительность приложения.

Большое сообщество: *React* имеет большое сообщество разработчиков, которые создают и поддерживают множество пакетов и расширений. Это позволяет разработчикам быстро решать задачи и получать поддержку при возникновении проблем.

Простота: *React* является относительно простым и понятным инструментом для создания пользовательских интерфейсов. Это делает его доступным для начинающих разработчиков и уменьшает время на обучение.

*React* используется для создания интерактивных пользовательских интерфейсов, включая веб-приложения, мобильные приложения, игры и многое другое. Он позволяет создавать переиспользуемые компоненты, которые могут быть легко использованы для создания сложных пользовательских интерфейсов *[*5*]*.

*Node*.*js* – это среда выполнения *JavaScript* на стороне сервера, которая позволяет разрабатывать высокопроизводительные и масштабируемые веб-приложения. Она основана на движке *V*8, разработанном *Google* для браузера *Chrome*, и позволяет использовать *JavaScript* для создания приложений на серверной стороне.

Основные преимущества *Node*.*js* приведены ниже.

Высокая производительность: *Node*.*js* основан на движке *V*8, который обеспечивает быстрое выполнение *JavaScript*. *Node*.*js* также позволяет использовать асинхронное программирование, что улучшает производительность приложений.

Масштабируемость: *Node*.*js* позволяет создавать масштабируемые приложения с помощью механизма обработки запросов в нескольких потоках. Это позволяет распределять нагрузку на несколько серверов и обеспечивать высокую доступность приложения.

Широкие возможности: *Node*.*js* имеет большое количество библиотек и модулей, которые позволяют упростить разработку и расширить функциональность приложения. Это позволяет создавать приложения для различных сфер, включая веб-приложения, мобильные приложения, игры и многое другое.

Единый язык: *Node*.*js* использует *JavaScript* как единый язык для программирования на серверной и клиентской стороне, что упрощает разработку и повышает эффективность работы разработчика.

Активное сообщество: *Node*.*js* имеет большое сообщество разработчиков, которые создают и поддерживают множество пакетов и расширений. Это позволяет разработчикам быстро решать задачи и получать поддержку при возникновении проблем.

*Node*.*js* используется для создания различных приложений на серверной стороне, включая веб-приложения, микросервисы, *API* и многое другое. Он позволяет разработчикам создавать высокопроизводительные и масштабируемые приложения с использованием *JavaScript* на стороне сервера, что упрощает разработку и повышает эффективность работы разработчика.

* 1. Анализ инструментальных средств автоматизации разработки и тестирования

Для создания программного комплекса дневника диабетика будет использована среда разработки *Webstorm* от компании *JetBrains*. Для визуализации базы данных *Firebase* лучше всего подходит приложение *FireAdmin*. Хранить исходный код только на локальном компьютере плохая практика, поэтому будет создан удаленный репозиторий на *GitHub*. Для постройки различных диаграмм, которые позволят упростить разработку, а также предоставят полное понимание работы приложения, будет использоваться *StarUml*.

Рассмотрим подробнее каждый из этих инструментов.

*Webstorm* – это интегрированная среда разработки (*IDE*), которая предоставляет обширный функционал для разработки веб-приложений. Среда разработки позволяет работать с различными языками программирования, включая *JavaScript*, *TypeScript*, *HTML*, *CSS*, *Node*.*js*, *Angular*, *React* и другие. *Webstorm* включает в себя встроенные инструменты отладки, систему автодополнения кода, автоматическую проверку ошибок, систему контроля версий, анализаторы кода и многие другие полезные функции.

*Webstorm* обладает многоплатформенностью, что позволяет разрабатывать на разных операционных системах, включая *Windows*, *macOS* и *Linux*. Среда разработки также имеет мощную систему плагинов, которая позволяет расширять функционал *IDE*, добавляя поддержку новых языков программирования и инструментов. Она также имеет множество инструментов для работы с базами данных, включая поддержку *MongoDB*, *MySQL*, *PostgreSQL*, *Oracle* и других, обеспечивает интеграцию с браузерами для отладки веб-приложений в режиме реального времени.

Одним из основных преимуществ *Webstorm* является его эффективность и производительность. Среда разработки использует многопоточную архитектуру и оптимизированный механизм работы с памятью, что обеспечивает быстродействие и позволяет работать с большими проектами. Кроме того, *Webstorm* имеет обширную документацию и активное сообщество пользователей, которые создают полезные плагины, советы и обучающие ресурсы, что делает процесс разработки еще более комфортным и эффективным.

*Webstorm* – это мощная среда разработки, которая облегчает и ускоряет процесс создания высококачественных веб-приложений, идеально подходящая для опытных и начинающих разработчиков *[*6*]*.

*FireAdmin* – это инструмент для визуализации и управления базой данных *Firebase*. Он предоставляет более продвинутые функции и интерфейс для управления данными в базе данных *Firebase*.

С помощью *FireAdmin* вы можете:

* просматривать и управлять всеми данными в вашей базе данных *Firebase*, включая коллекции, документы и поля;
* определять права доступа на уровне коллекций и документов, а также назначать роли и пользователей, которым разрешен доступ к определенным данным;
* управлять индексами, которые используются для быстрого доступа к данным в базе данных;
* использовать инструменты для экспорта и импорта данных в вашу базу данных;
* создавать пользовательские запросы, чтобы быстро найти и отфильтровать нужные данные.

*FireAdmin* предоставляет удобный интерфейс пользователя, который позволяет легко просматривать и управлять данными в вашей базе данных *Firebase*. Он также предоставляет инструменты для безопасности и экспорта/импорта данных, что делает его полезным инструментом для управления большими и сложными базами данных *Firebase*.

*FireAdmin* работает с базой данных *Firebase* в реальном времени, что позволяет быстро и легко обновлять данные в базе данных. Он также интегрируется с *Firebase* *Auth*, что позволяет управлять правами доступа к базе данных на основе аутентификации пользователей. *FireAdmin* доступен как веб-приложение, а также может быть установлен локально для работы с базой данных *Firebase* на локальной машине разработчика *[*7*]*.

*GitHub* – это веб-сервис для хранения и совместной работы над *Git*-репозиториями. Это платформа, которая позволяет разработчикам хранить и совместно работать над кодом, отслеживать ошибки, создавать ветки, а также просматривать и редактировать код, управлять версиями, изменениями и запросами на слияние.

*GitHub* позволяет работать как индивидуальным разработчикам, так и командам, предоставляя возможность контролировать доступ к репозиторию и назначать роли для разных пользователей. Кроме того, *GitHub* интегрируется с другими сервисами, такими как *Travis* *CI*, *Slack*, *JIRA* и многие другие.

Репозиторий в *GitHub* – это хранилище для кода и других файлов, которые могут быть загружены и управляемы в *Git*. Репозиторий в *GitHub* позволяет хранить и управлять кодом в облаке, а также предоставляет множество функций для работы с кодом, включая возможность комментирования кода, управления задачами и многое другое. Если хранить код только на локальном компьютере, то это может привести к потере данных в случае сбоя жесткого диска или других проблем с компьютером. Кроме того, хранение кода на локальном компьютере не обеспечивает возможность совместной работы и синхронизации изменений между различными разработчиками.

Основные функции *GitHub*:

* хранение и управление кодом. *GitHub* позволяет создавать удаленные репозитории для хранения кода, а также выполнять действия, такие как коммиты, создание веток и слияние веток;
* отслеживание изменений. *GitHub* отслеживает и сохраняет историю изменений в коде, позволяя легко переключаться между различными версиями и возвращаться к предыдущим версиям, если это необходимо;
* ведение проектов. *GitHub* предоставляет инструменты для ведения проектов, включая задачи и проблемы, обсуждение кода и код-ревью;
* совместная работа. *GitHub* позволяет нескольким разработчикам работать с одним и тем же кодом одновременно, сливая свои изменения вместе;
* интеграция с другими сервисами. *GitHub* интегрируется с другими сервисами, такими как *Travis* *CI*, *Slack* и *JIRA*, упрощая совместную работу и автоматизируя процессы.

Работа с открытым исходным кодом. *GitHub* является популярной платформой для работы с открытым исходным кодом, позволяя разработчикам сотрудничать над проектами, созданными сообществами, и вносить свой вклад в развитие открытых проектов.

Таким образом, *GitHub* – это мощный и удобный инструмент для хранения и совместной работы над кодом. Он предоставляет широкий спектр возможностей для управления кодом, удобный интерфейс для просмотра и редактирования кода, а также возможность совместной работы и синхронизации изменений между различными разработчиками.

*StarUML* – это приложение для создания *UML*-диаграмм, которое позволяет разработчикам создавать модели проектов, планировать архитектуру и дизайн приложений. *StarUML* имеет графический интерфейс пользователя, который предоставляет множество инструментов для создания и редактирования диаграмм, включая диаграммы классов, диаграммы последовательностей, диаграммы состояний, диаграммы активностей и многое другое.

На рисунке 1.2 в виде выпадающего меню представлен весь список возможных диаграмм.



Рисунок 1.2 – Интерфейс главного окна *StarUml*

В *StarUML* можно создавать различные типы диаграмм, включая:

* диаграммы классов – используются для описания структуры классов и их отношений;
* диаграммы последовательностей – используются для описания взаимодействия между объектами и процессов, происходящих во времени;
* диаграммы состояний – используются для описания жизненного цикла объекта и его состояний;
* диаграммы компонентов – используются для описания структуры и отношений между компонентами системы;
* диаграммы развертывания – используются для описания физического размещения компонентов и системы в целом;
* диаграммы активностей – используются для описания последовательности действий и процессов в системе.

*StarUML* поддерживает различные языки моделирования, включая *UML* 2.*x*, *SysML*, *ERD*, *BPMN*, *SoaML* и другие. Он также позволяет пользователям импортировать и экспортировать диаграммы в различных форматах файлов, включая *XMI*, *SVG*, *PDF*, *PNG*, *JPEG* и другие. Данное приложение имеет множество функций, которые облегчают создание и редактирование диаграмм, такие как автоматическое выравнивание элементов, возможность группировки элементов, подсветка синтаксиса, подсказки и многое другое. Кроме того, *StarUML* позволяет пользователю создавать свои собственные элементы, шаблоны и плагины, чтобы улучшить функциональность и адаптировать инструмент под свои потребности.

*StarUML* – это мощный и удобный инструмент для создания *UML*-диаграмм, который предоставляет широкий спектр возможностей для создания и редактирования диаграмм, а также импорта и экспорта диаграмм в различных форматах файлов. Он также имеет множество функций, которые облегчают создание и редактирование диаграмм, и может быть расширен за счет пользовательских элементов, шаблонов и плагинов *[*8*]*.

*Postman* – это инструмент для тестирования *API*, который позволяет разработчикам быстро и удобно отправлять запросы к *API*, тестировать их и анализировать ответы.

С помощью *Postman* можно отправлять запросы на сервер и получать ответы в разных форматах, таких как *JSON*, *XML*, *HTML* и другие. Также в *Postman* есть множество функциональных возможностей, таких как автоматическое генерирование документации, управление авторизацией, создание и выполнение тестовых сценариев и другие. Он поддерживает различные типы запросов, включая *GET*, *POST*, *PUT*, *DELETE* и многие другие. Вы можете отправлять запросы с параметрами, заголовками и телом запроса в формате *JSON* или форм-данных.

Один из самых удобных и полезных аспектов *Postman* – это коллекции запросов, которые можно создавать и организовывать для более эффективного управления тестированием *API*. Коллекции могут быть экспортированы и импортированы в различных форматах, таких как *JSON* и *XML*.

*Postman* имеет множество преимуществ, которые делают его популярным среди разработчиков, некоторые из них приведены ниже.

Удобный интерфейс. Интерфейс *Postman* интуитивно понятен и удобен в использовании. Для отправки запроса не нужно писать код, все нужные функции находятся в графическом интерфейсе.

Простота в использовании. *Postman* не требует от разработчика особых навыков, чтобы начать использовать его. Для создания запросов нужно просто заполнить соответствующие поля.

Расширяемость. *Postman* можно легко расширять, используя плагины. Например, с помощью плагина можно добавить поддержку авторизации на основе *OAuth* или создать пользовательский набор инструментов для тестирования *API*.

Автоматизация. *Postman* позволяет автоматизировать тестирование *API*. Это особенно полезно при наличии большого количества запросов, которые нужно тестировать.

Множество функций. *Postman* имеет множество функций, таких как создание коллекций, генерация документации и выполнение тестовых сценариев, что делает его полезным инструментом для разработки и тестирования приложений.

Бесплатность. *Postman* имеет бесплатную версию, которая покрывает большинство потребностей разработчиков, а также платную версию с дополнительными функциями.

*Postman* является полезным и удобным инструментом для тестирования *API*, который может существенно ускорить процесс разработки приложений *[*9*]*.

* 1. Постановка задачи на дипломное проектирование

Цель разработки: разработать приложение, предназначенное для автоматизации дневника диабетика.

Данный программного комплекс предназначается для людей, больных диабетом.

Для пациентов приложение предусматривает следующие функции:

* ведение дневника уровня сахара в крови и принимаемых лекарств;
* получение рекомендаций и уведомлений по питанию и уровню сахара;
* просмотр истории показателей уровня сахара;
* создание целей;
* просмотр тематических материалов;
* взаимодействие с врачом и другими специалистами.

Для родственников пациентов приложение предусматривает следующие функции:

* просмотр информации о состоянии здоровья пациента;
* получение уведомлений о состоянии здоровья и уровне сахара;
* взаимодействие с пациентом и врачом при необходимости.

Для докторов приложение предусматривает следующие функции:

* просмотр истории показателей уровня сахара и анализ изменений в здоровье пациента;
* общение с пациентом;
* редактирование тематических материалов.

Приложение должно иметь следующую структуру и функциональность:

* работать как веб-приложение, основанное на *REST* архитектуре;
* являться кроссплатформенным;
* являться кроссбраузерным;
* иметь принцип работы, аналогичный веб-приложениям (реализовывать *REST* архитектуру, быть доступным посредством веб-браузера, иметь пользовательский интерфейс).

1. АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
   1. Общая структура приложения

Архитектура программного обеспечения (АПО) сайта, написанного на *React* зависит от многих факторов, включая масштаб проекта, цели разработки, требования к функциональности и многих других. Однако, в целом, можно выделить несколько основных составляющих АПО сайта.  
 *React* строится на базе компонентов, каждый из которых отвечает за отдельные части интерфейса. Компоненты *React* обычно создаются в файле с расширением .*jsx* и представляют собой синтаксический сахар над обычным *JavaScript*.  
 Для начала, на первой стадии проектирования программа разбивается на множество компонентов. Однако, чтобы компоненты могли работать вместе, необходимо реализовать механизм, позволяющий им взаимодействовать друг с другом. В *React* для этого используется свойства (*props*) и состояния (*state*) компонентов, которые определяют данные, которые компоненту нужно отобразить и изменять *[*10*]*.

На рисунке 2.1 представлено разбиение *React*-приложения ни компоненты.



Рисунок 2.1 – Разбиение на компоненты в *React*-приложении

*JavaScript* – это язык программирования, который можно использовать как для написания фронт-энда, так и для написания бэк-энда приложений. В контексте разработки сайта на *React* и *Firebase*, *JS* используется для взаимодействия с веб-сервером и обменом данными *[*2*]*.  
 *Firebase* предоставляет множество инструментов для разработки мобильных и веб-приложений, включая базу данных в реальном времени, аутентификацию, хостинг, хранилище и многие другие. Чтобы интегрировать *Firebase* в приложение на *React*, необходимо создать проект *Firebase* и подключить его *API*. Затем, используя *JavaScript*, можно записывать данные в базу данных *Firebase* и получать данные из нее *[*11*]*.  
 Программное обеспечение сайта на *React* и *Firebase* обычно имеет следующую архитектуру:  
 Компоненты. Как уже упоминалось ранее, *React* построен на базе компонентов, каждый из которых отвечает за отдельные части пользовательского интерфейса. Компоненты могут быть классами или функциями, которые возвращают *JSX* элементы.  
 Сервисы. Сервисы – это классы или функции, которые представляют собой дополнительную логику приложения, которая не связана с отображением интерфейса. Например, сервис авторизации может содержать методы для регистрации и входа пользователей.  
 База данных. *Firebase* предоставляет базу данных в реальном времени, которая может использоваться для хранения пользовательских данных. База данных *Firebase* имеет структуру древовидной структуры. Для доступа к базе данных в *React* используется *Firebase* *API*.  
 Хранилище. Хранилище – это объект, который содержит в себе состояние приложения. Хранение состояния в объекте позволяет контролировать изменения состояния и обновлять интерфейс пользователя соответствующим образом.  
 Роутер. Роутер – это компонент, который управляет навигацией на сайте. Роутер часто используется в *React*-приложениях для создания маршрутов веб-страниц.  
 Однако, архитектура программного обеспечения сайта на *React*, *JS* и *Firebase* может быть очень различной, в зависимости от требований к функциональности и масштаба проекта. Но в любом случае, важно организовать приложение таким образом, чтобы оно было масштабируемым, легко поддерживаемым и масштабируемым для изменений и улучшений в будущем.

На рисунке 2.1 представлена архитектура *Firebase*.



Рисунок 2.2 – архитектура *Firebase*

Приложение, написанное на *JS* и *React* с использованием *Firebase* для авторизации, хранения файлов и *FirestoreDatabase* для хранения данных представляет собой современное приложение с большим набором функционала. Оно позволяет пользователям создавать свои аккаунты, загружать файлы и хранить их на сервере, а также вводить и получать данные, которые будут храниться в базе данных.

Общая схема приложения будет следующей: пользователь сначала попадает на страницу входа, где он может воспользоваться своим аккаунтом *Firebase* или зарегистрироваться, используя свой адрес электронной почты и пароль. В случае успешной авторизации пользователь попадает на главную страницу приложения, где он может работать со своими файлами и данными.

Далее происходит взаимодействие с *Firebase* для загрузки, хранения и отображения файлов. *Firebase* предоставляет удобный *API* для загрузки файлов на сервер, а также возможность сохранения метаданных о каждом загруженном файле (например, имя, тип и размер файла). Кроме того, *Firebase* имеет возможность автоматической генерации ссылок для скачивания или просмотра файлов, что позволяет облегчить работу с файлами в приложении.

*FirestoreDatabase* используется для хранения и обработки данных. Для работы с *FirestoreDatabase* используется набор *API*, который позволяет выполнять операции чтения и записи данных, а также отслеживать изменения в реальном времени. Приложение может хранить такие данные, как названия файлов, описания, теги и другие параметры, которые пользователь может использовать для отображения и фильтрации файлов в своих списках.

Также можно использовать *FirestoreDatabase* для хранения информации о пользователе и его аккаунте. Например, можно хранить список загруженных пользователем файлов и устанавливать соответствующие права доступа для других пользователей. Таким образом, *FirestoreDatabase* позволяет создавать многопользовательские приложения с различными уровнями доступа к файлам и данным.

Взаимодействие между компонентами приложения может быть организовано с помощью технологий реактивного программирования, таких как *RxJS*. *RxJS* предоставляет мощные инструменты для работы с потоками данных, что позволяет более эффективно управлять изменениями в данных и обновлять пользовательский интерфейс приложения.

В целом, приложение, написанное на *JS* и *React* с использованием *Firebase* для авторизации, хранения файлов и *FirestoreDatabase* для хранения данных, представляет собой многофункциональное приложение с большим количеством возможностей и широким набором функционала, которые позволяют удобно и эффективно управлять своими файлами и данными в облаке.

* 1. Функциональная модель программного комплекса

Для того, чтобы создать качественное приложение, нужно четко понимать, какие роли у нас будут и какой функционал им будет доступен. Для этого нужно знать, какие объекты попадают в предметную область проектируемой системы и какие логические связи между ними существуют. Для формирования такого понимания используются логические модели предметной области. Целью построения логической модели является получение графического представления логической структуры исследуемой предметной области. Для стабильной работы программного обеспечения необходимо чёткое распределение на роли, на основе которых будут формироваться функции взаимодействия со внутренней средой приложения.

Актер – множество логически связанных ролей в *UML*, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Актером может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Прецеденты представляют действия, выполняемые системой в интересах актеров. Проще говоря, прецедент – это описание последовательности действий (или нескольких последовательностей), которые выполняются системой и производят для отдельного актера видимый результат. Один актер может использовать несколько элементов прецедентов, и наоборот, один прецедент может иметь несколько актеров, использующих его. Каждый прецедент задает определенный путь использования системы. Набор всех прецедентов определяет полные функциональные возможности системы.

Приложение должно реализовывать шесть ролей: пациент, родственник, доктор, контент-мейкер, модератор и администратор. Каждая из этих ролей имеет свои функции и возможности в приложении.

В качестве среды для разработки и отображения функциональной структуры программы будет использована *StarUml*. На рисунке 2.3 представлена *Use* *Case* диаграмма приложения.



Рисунок 2.3 – Диаграмма прецедентов и актеров

Далее приведено описание прецедентов для роли «Пациент».

Прецедент «Заполнение дневника» предназначен для возможности пациента вести дневник своих показаний кровяного сахара и других параметров, которые могут влиять на состояние здоровья при диабете. Этот прецедент позволяет пациентам вносить данные о уровне глюкозы в крови, давлении, пульсе, приеме лекарств и еде в течение дня. Это позволяет пациентам отслеживать свои показатели и узнавать, какие факторы влияют на их состояние здоровья. Кроме того, эти данные могут быть использованы доктором для анализа эффективности лечения и определения необходимости корректировки режима лечения. В рамках прецедента «Заполнение дневника» пациент может вносить данные как вручную, так и с помощью подключения к специальным датчикам, которые автоматически передают информацию о показаниях кровяного сахара и других параметрах в приложение. Это удобно и экономит время пациента.

Прецедент «Общение с доктором» предназначен для обеспечения возможности общения пациента с его доктором в рамках приложения. Он позволяет пациенту общаться с доктором через систему сообщений внутри приложения и обмениваться информацией о своем здоровье. Доктор может получить доступ к истории заболевания пациента, результатам анализов, заполненному дневнику и другим медицинским данным, чтобы лучше понимать состояние пациента и давать рекомендации по лечению. Пациент, в свою очередь, может задавать вопросы и получать консультации от своего доктора, не выходя из дома. Прецедент «Общение с доктором» помогает пациентам получить необходимую медицинскую помощь без посещения больницы или клиники, что может быть особенно важно для людей с ограниченными возможностями или живущих в удаленных районах. Также это помогает своевременно обнаруживать и решать проблемы со здоровьем, что может уменьшить затраты на лечение и повысить качество жизни пациента.

Прецедент «Изучение тематических материалов» предназначен для того, чтобы пациент мог ознакомиться с информацией о своем заболевании, методах лечения, профилактике и других важных аспектах, связанных с его здоровьем. Это может помочь пациенту лучше понимать свою ситуацию, повысить свою эффективность лечения и принимать более обоснованные решения в отношении своего здоровья.

Прецедент «Комментирование тематических материалов» предназначен для того, чтобы пациенты могли оставлять комментарии к размещенным в приложении тематическим материалам. Это позволит пациентам обмениваться мнениями и опытом, а также задавать вопросы другим пользователям и получать на них ответы. Кроме того, комментарии могут помочь улучшить качество и полезность тематических материалов, поскольку авторы материалов могут учитывать мнение пользователей и вносить соответствующие изменения.

Прецедент «Создание целей» предназначен для того, чтобы пациент мог определить свои цели и планировать свои достижения в приложении. В рамках данного прецедента пациент может создавать цели, определять сроки их достижения, а также отслеживать прогресс и получать статистику по достижению своих целей. Этот прецедент позволяет пользователям более осознанно подходить к лечению и мотивироваться на его продолжение.

Далее приведено описание прецедентов для роли «Родственник».

Прецедент «Просмотр статистики пациента» предназначен для возможности ознакомления родственником с показателями здоровья пациента, такими как вес, давление, пульс и другими, которые были введены пациентом в свой дневник здоровья. Это может помочь родственнику быть в курсе состояния здоровья пациента и в случае необходимости оказать ему помощь.

Далее приведено описание прецедентов для роли «Доктор».

Прецедент «Консультация пациента» предназначен для общения доктора с пациентом, получения информации о состоянии здоровья пациента, определения диагноза и назначения лечения.

Прецедент «Просмотр статистики» предназначен для того, чтобы доктор мог просмотреть статистическую информацию о пациенте, такую как пульс, давление, количество глюкозы в крови за определенный период. Это помогает доктору лучше понимать состояние пациента и принимать более обоснованные решения относительно его лечения.

Прецедент «Редактирование тематических материалов» предназначен для изменения или обновления информации, размещенной на платформе для пациентов, чтобы обеспечить актуальность и достоверность тематических материалов, а также их соответствие с медицинскими исследованиями.

Далее приведено описание прецедентов для роли «Контент-мейкер».

Прецедент «Написание тематических материалов» предназначен для создания новых тематических материалов, которые будут использоваться для обучения и информирования пациентов и родственников. Эти материалы могут включать в себя статьи, видео, аудиозаписи и т.д. и должны быть актуальными и понятными для целевой аудитории.

Прецедент «Редактирование тематических материалов» предназначен для изменения, обновления или дополнения уже существующих тематических материалов.

Далее приведено описание прецедентов для роли «Модератор».

Прецедент «Модерирование комментариев» предназначен для контроля комментариев, которые пользователи оставляют к тематическим материалам, написанным контент-мейкерами. Это может включать удаление неподходящих комментариев, ответы на вопросы пользователей, предоставление дополнительной информации и т.д. В целом, этот прецедент связан с поддержанием безопасной и полезной общественной обстановки в сообществе пациентов.

Прецедент «Редактирование тематических материалов» предназначен для внесения изменений в опубликованные тематические материалы, например, исправления опечаток, изменения форматирования или добавления дополнительной информации. Однако, в зависимости от конкретного контекста, возможны и другие варианты использования этого прецедента, например, редактирование заголовков или тегов для более точного описания материала.

Далее приведено описание прецедентов для роли «Администратор».

Прецедент «Редактирование списка сотрудников» предназначен для изменения списка сотрудников, в том числе добавления, удаления и изменения информации о сотрудниках.

Прецедент «Контроль показателей сотрудников» предназначен для отслеживания работы и производительности сотрудников, а также контроля за их достижениями и показателями работы. Этот прецедент позволяет администратору получать информацию о работе каждого сотрудника и выявлять проблемные моменты в их деятельности, чтобы улучшать работу компании в целом.

* 1. Информационная модель программного комплекса

Одна из главных задач множества существующих приложений – хранение данных в каком-либо виде. Базы данных – неотъемлемая часть работы подавляющего большинства программного обеспечения, существующего на сегодняшний день. Использование баз данных является достаточно эффективным способом хранить данные.

Как уже упоминалось ранее, *Firebase* предоставляет множество инструментов для разработки веб-приложений, включая базу данных в реальном времени, аутентификацию, хостинг, хранилище, что достаточно удобно для своевременного реагирования членам администрации СЭЗ для регистрации новых потенциальных инвесторов.

Для заполнения базы данных были использованы данные личной электронной почты для проверки авторизации с помощью сервисов *Google*, а также данные, внесённые в базу данных во время тестирования и работы с программой.

На момент написания отчета по преддипломной практике был подключено и в полной мере программно-реализован сервис для аутентификации (*Authentication*) рисунок 2.4



Рисунок 2.4 – сервис для авторизации и хранения данных о пользователях в *Firebase*

На основе анализа предметной области можно выделить следующие сущности:

* аутентификация (*Authentication*);
* дневники (*Diary*);
* цели (*Goal*);
* тематические материалы (*ThematicMaterial*);
* комментарии (*Comment*);
* сотрудники (*Employee*);
* отчёты (*Report*);
* чат (*Chat*).

«Аутентификация» – сущность, которая представляет собой сервис *Firebase* для хранения информации о пользователях.

«Дневники» – сущность, которая представляет информацию о том, как пациент проходит программу лечения. В дневнике пациент может записывать свои ощущения, настроение, симптомы и т.д.

«Цели» – сущность, которая содержит информацию о всех целях, которые пациент хочет достичь. Например, уменьшить количество приемов лекарств или начать заниматься спортом.

«Тематические материалы» – сущность, которая содержит информацию о различных темах, связанных с заболеваниями и лечением. Тематические материалы могут быть предоставлены пациентам для обучения и помощи им в понимании своего заболевания.

«Комментарии» – сущность, которая содержит информацию о всех комментариях к тематическим материалам.

«Сотрудники» – сущность, которая содержит информацию о всех сотрудниках, которые имеют доступ к программному комплексу. Эта информация может включать в себя имя, фамилию, должность и уровень доступа к системе.

«Отчеты» – сущность, которая содержит информацию о всех отчетах, созданных пациентами по их показателям, таким как: вес, количество глюкозы, пульс и т.д. за определенный период.

«Чат» – сущность, которая представляет собой сервис для общения между пациентом и доктором. Чат может быть использован для проведения консультации пациента по его показателям или по тематическим материалам.

**При использовании *Firebase* информационная модель программного комплекса может быть улучшена благодаря возможностям данной платформы. В частности, для каждой из сущностей (дневники, цели, тематические материалы, комментарии, сотрудники, отчёты и чат) можно создать соответствующую коллекцию в *Firebase* *Cloud* *Firestore*. Внутри каждой коллекции могут храниться документы, содержащие информацию об отдельной записи, а также поля, описывающие эту запись.**

***Firebase* также позволяет быстро и легко добавлять аутентификацию пользователей и управлять правами доступа к данным, что может быть важным для защиты конфиденциальной информации о пациентах. Кроме того, *Firebase* *Storage* может быть использован для хранения файлов, связанных с определенными записями в системе, такими как фотографии или видео.**

**Таким образом, использование *Firebase* может помочь улучшить информационную модель программного комплекса и обеспечить более эффективное хранение, управление и доступ к данным.**

**Далее рассмотрим схему базы данных и коллекции более подробно, на рисунке 2.5 приведена схема базы данных.**

****

Рисунок 2.5 – Схема базы данных

На схеме не описано поле *id*. Каждый документ в коллекции имеет уникальный идентификатор, который позволяет быстро идентифицировать этот документ в коллекции и управлять им.

В таблицах 2.1 – 2.5 представлено подробное описание полей в коллекциях.

Таблица 2.1 – Коллекция «*Diaries*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Обязательное | Уникальное |
| *id* | *string* | Да | Да |
| *userRef* | *ObjectRef* | Да | Да |
| *dailyLogs* | *Array*<*DailyLog*> | Нет | Нет |

Продолжение таблицы 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Обязательное | Уникальное |
| *dailyLogs*.*calories* | *number* | Да | Нет |
| *dailyLogs*.*carbohydratesIntake* | *number* | Да | Нет |
| *dailyLogs*.*date* | *timestamp* | Да | Нет |
| *dailyLogs*.*medications* | *Array*  <*Medication*> | Нет | Нет |
| *dailyLogs*.*medications*.*medicationRef* | *ObjectRef* | Нет | Нет |
| *dailyLogs*.*medications*.*dosage* | *number* | Нет | Нет |
| *dailyLogs*.*medications*.*time* | *timestamp* | Нет | Нет |
| *dailyLogs*.*pulse* | *number* | Да | Нет |
| *dailyLogs*.*blood*.*pressure*.*systolic* | *number* | Да | Нет |
| *dailyLogs*.*blood*.*pressure*.*diastolic* | *number* | Да | Нет |
| *dailyLogs*.*blood*.*sugarLevels* | *Array*<*number*> | Да | Нет |
| *dailyLogs*.*temperature* | *number* | Нет | Нет |
| *dailyLogs*.*weight* | *number* | Да | Нет |
| *dailyLogs*.*symptoms* | *string* | Нет | Нет |
| *dailyLogs*.*notes* | *string* | Нет | Нет |
| *goals*.*title* | *string* | Да | Нет |
| *goals* | *Array*  <*Goal*> | Нет | Нет |
| *goals*.*description* | *string* | Да | Нет |
| *goals*.*notes* | *Array*<*string*> | Нет | Нет |
| *goals*.*createdAt* | *timestamp* | Да | Нет |
| *goals*.*deadline* | *timestamp* | Да | Нет |
| *goals*.*category* | *GoalCategory* | Да | Нет |
| *goals*.*status* | *GoalStatus* | Да | Нет |
| *goals*.*progress* | *number* | Да | Нет |

Как видно из таблицы 2.1, коллекция «*Diaries*» хранит в себе данные дневника пациента, включая показатели его здоровья, принятые медикаменты, а также цели пациента, например, похудение или занятия спортом.

Для описания категорий и статусов целей в коллекции «*Diaries*» были созданы коллекции «*GoalCategories»* и «*GoalStatuses»*, которые содержат в себе заранее определенный список документов и не меняются при работе программы.

Для описания медицинских препаратов была создана коллекция «*Medications».*

Таблица 2.2 – Коллекция «*Medications*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Обязательное | Уникальное |
| *id* | *ObjectId* | Да | Да |
| *title* | *string* | Да | Нет |
| *description* | *string* | Да | Нет |
| *instructions* | *string* | Да | Нет |
| *medicationRouteRef* | *ObjectRef* | Да | Нет |
| *frequency* | *string* | Да | Нет |

Как видно из таблицы 2.2, коллекция «*Medications*» хранит в себе данные о препаратах, которые могут принимать пациенты.

Для описания способа приема препаратов в коллекции «*Medications»* была создана коллекция «*MedicationRoutes*», которая содержит в себе заранее определенный список документов.

Таблица 2.3 – Коллекция «*Users*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Обязательное | Уникальное |
| *id* | *ObjectId* | Да | Да |
| *name* | *string* | Да | Нет |
| *email* | *string* | Да | Нет |
| *roleRef* | *ObjectRef* | Да | Нет |
| *profileInfo*.*address* | *string* | Нет | Нет |
| *profileInfo*.*diabetType* | *number* | Нет | Нет |
| *profileInfo*.*phone* | *string* | Нет | Нет |
| *createdAt* | *timestamp* | Да | Нет |
| *lastLogin* | *timestamp* | Да | Нет |

Как видно из таблицы 2.3, коллекция «*Users*» хранит в себе данные о пользователях системы, а также тип диабета для пациентов.

Для описания ролей пользователей в коллекции «*Users*» была создана коллекция «*Roles*», которая содержит заранее определенный список ролей пользователей.

Таблица 2.4 – Коллекция «*ThematicMaterials*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Обязательное | Уникальное |
| *id* | *ObjectId* | Да | Да |
| *title* | *string* | Да | Нет |
| *description* | *string* | Да | Нет |
| *authorUserRef* | *ObjectRef* | Да | Нет |
| *comments* | *Array*  <*Comment*> | Нет | Нет |
| *comments*.*content* | *string* | Нет | Нет |
| *comments*.*createdAt* | *timestamp* | Нет | Нет |
| *comments*.*senderUseref* | *ObjectRef* | Нет | Нет |

Как видно из таблицы 2.4, коллекция «*ThematicMaterials*» хранит в себе тематические материалы как по диабету, так и по здоровью в целом, а также комментарии к тематическим материалам.

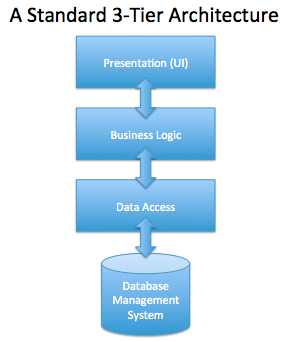
Таблица 2.5 – Коллекция «*Dialogs*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Обязательное | Уникальное |
| *id* | *ObjectId* | Да | Да |
| *doctorUserRef* | *ObjectRef* | Да | Нет |
| *patientUserRef* | *ObjectRef* | Да | Нет |
| *messages* | *Array*  <*Message*> | Нет | Нет |
| *messages*.*content* | *string* | Да | Нет |
| *messages*.*createdAt* | *timestamp* | Да | Нет |
| *messages*.*senderUserRef* | *ObjectRef* | Да | Нет |

Как видно из таблицы 2.5, коллекция «*Dialogs*» хранит в себе информацию о диалогах пациента и врача, а также все их сообщения.

1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КОМПЛЕКСА
   1. Архитектура приложения

Приложение использует трёхуровневую архитектуру (рисунок 2.2). Трехуровневая архитектура приложения – это подход к проектированию программного обеспечения, в котором приложение разбивается на три слоя: представление (*presentation layer*), бизнес-логика (*business logic layer*) и уровень доступа к данным (*data access layer*).



1. Схема архитектуры приложения

Трехуровневая архитектура, которая была реализована в приложении описана ниже.

Уровень представления (*Presentation* *Layer*) – в *FERN*-стеке уровень представления представлен *React*-фреймворком, который используется для разработки клиентской части приложения. *React* позволяет создавать переиспользуемые компоненты, которые управляют отображением данных на странице;

Уровень бизнес-логики (*Application* *Layer*) – этот уровень обрабатывает бизнес-логику приложения. В *FERN*-стеке этот уровень реализован на *Node*.*js* и *Express*.*js*. *Node*.*js* – это серверная платформа, которая позволяет запускать *JavaScript* на стороне сервера, а *Express*.*js* – это веб-фреймворк для *Node*.*js*, который предоставляет удобный *API* для работы с запросами и ответами;

Уровень доступа к данным (*Data* *Access* *Layer*) – этот уровень отвечает за доступ к базе данных. В *FERN*-стеке в качестве базы данных используется *Cloud Firestore*, которая хранит данные в формате документов *JSON*. Для работы с *Firestore* в *FERN*-стеке используется *Firebase SDK*, которая позволяет приложению обращаться к базе данных, выполнять операции чтения и записи данных, а также подписываться на обновления данных в реальном времени.

Таким образом, трехуровневая архитектура приложения разделяет приложение на три основных уровня: представление, бизнес-логику и уровень доступа к данным. Это позволяет разработчикам легче поддерживать и расширять приложение, а также упрощает его тестирование и развертывание.

* 1. Уровень представления клиентской части приложения

Уровень представления содержит следующие страницы:

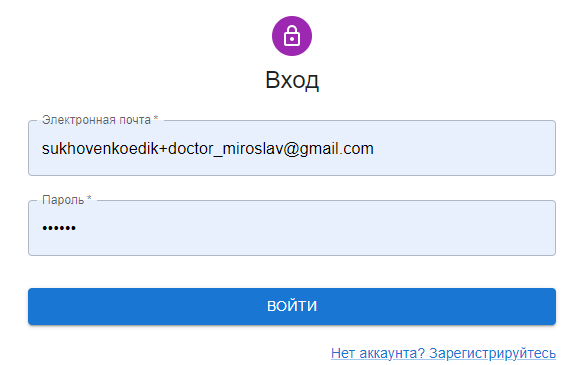
* *Home* – домашняя страница приложения;
* *SignIn* – страниица для входа в приложение;
* *SignUp* – страница для регистрации студента в пациента или родственника в системе;
* *SignOut* – страница для выхода из приложения;
* *Profile* – данная страница предназначена для просмотра и редактирования личной информации пользователя;
* *Diary* – страница для ведения дневника пациента, его показателей за день, а также заметок;
* *Statistics* – страница для просмотра статистики пациента;
* *PatientsStatistics* – страница для просмотра статистики пациентов, которые обследуются у доктора;
* *RelativeStatistics* – страница для просмотра статистики пациента-родственника;
* *Dialogs* – страница, которая позволяет вести беседу между доктором и пациентом;
* *ThematicMaterials* – страница для просмотра тематических материалов о диабете;
* *ThematicMaterial* – страница для более детального изучения тематического материала;
* *Relative* – страница, которая позволяет добавить родственника;
* *Doctor* – страница для выбора доктора, исходя из его оценок и отзывов;
* *Reviews* – страница для просмотра оценок и отзывов доктора;
* *HealthStates* – страница просмотра и редактирования готовых назначений доктора;
* *Notifications* – страница для просмотра и редактирования уведомлений о пациентах доктора;
* *Recommendations* – страница, на которой находится таблица с последними рекомендациями доктора о состоянии здоровья пациента;
* *Patients* – страница, на которой находится таблица пациентов, которые обследуются у доктора;
* *Accounts* – страница, на которой находится таблицы со всеми пользователями системы.

Страница *Home* (рисунок 3.2) является первой страницей, которую видит любой пользователь. С этой страницы пользователь может перейти на страницы регистрации и входа.



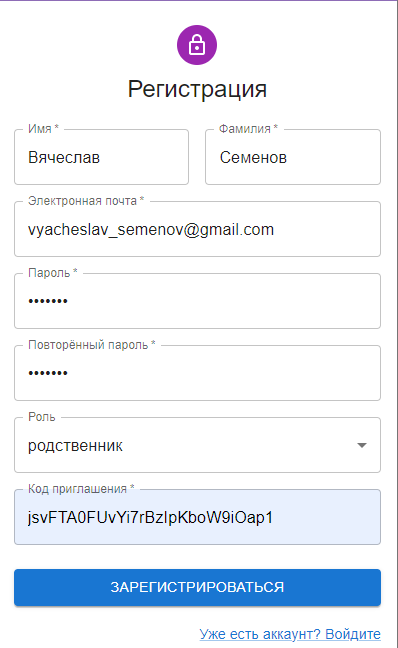
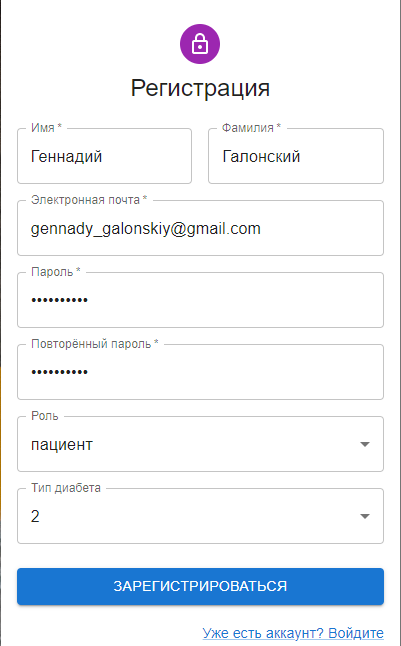
1. Домашняя страница приложения

Страница *SignIn* (рисунок 3.3) является одной из начальных страниц, которую должны заполнить все роли при первоначальном запуске программы. Страница отвечает за процесс аутентификации пользователей. Она предоставляет пользователю возможность войти в приложение, используя свои учетные данные, такие как электронная почта и пароль. Все поля являются обязательными. В случае, если пациент еще не зарегистрирован в системе, ему доступна ссылка «Нет аккаунта? Зарегистрируйтесь».



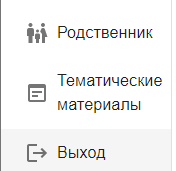
1. Страница входа

Страница *SignUp* (рисунок 3.4) предназначена для регистрации пациента или родственника пациента в системе. На данной странице представлена форма со следующими полями: имя, фамилия, электронная почта, пароль, повторённый пароль, роль, тип диабета, код приглашения. Поля «тип диабета» и «код приглашения» зависят от поля «роль», если регистрируется пациент, то ему предоставляется возможность заполнить поле «тип диабета», если родственник – «код приглашения», который был ранее передан пациентом для регистрации родственника. «Код приглашения» можно найти на странице «Родственник» пациента. Все поля являются обязательными для заполнения. В случае, если пациент или родственник уже имеет аккаунт, ему доступна ссылка «Уже есть аккаунт? Войдите».



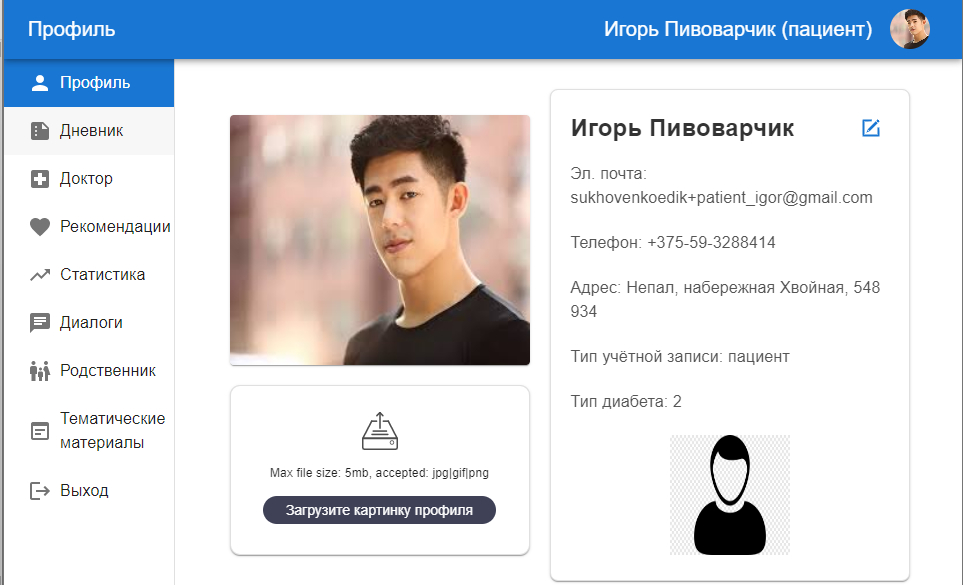
1. Страница регистрации

Страница *SignOut* не имеет пользовательского интерфейса, используется для выхода из системы, чтобы перейти на эту на эту страницу необходимо выбрать пункт «Выход» в меню (рисунок 3.5).



1. Кнопка «Выход» в меню

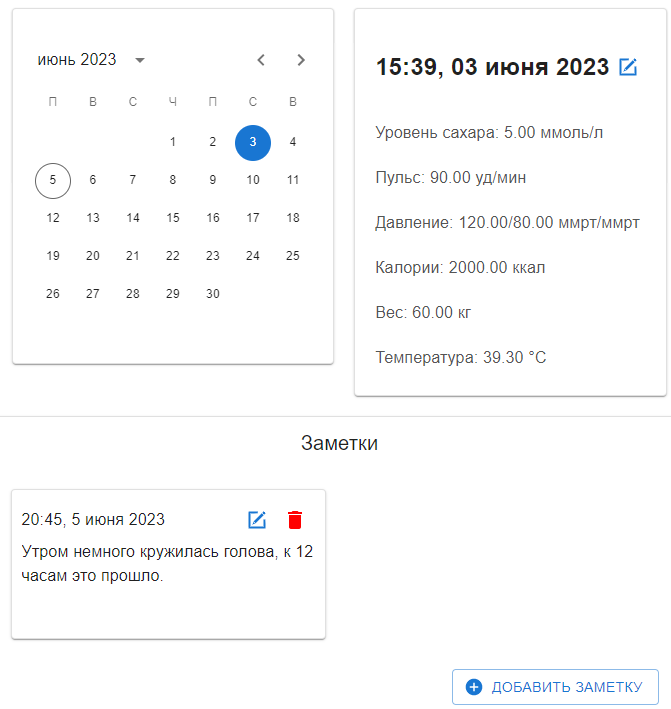
После успешной авторизации студента откроется страница *Profile* (рисунок 3.6). На странице пользователь видит основную информацию об аккаунте, такую как: электронная почта, мобильный телефон, адрес, тип учетной записи, тип диабета и изображение учётной записи. Кроме просмотра информации, пользователю доступны кнопки редактирования личного кабинета, такие как: загрузка изображения профиля и редактирование личных данных. При нажатии на кнопку загрузки изображения профиля открывается диалоговое окно с выбором файла, как только пользователь выбрал изображение, оно отправляется в хранилище *Firestorage*, а на странице появляется индикатор загрузки, как только загрузка изображения завершается пользователь видит обновленное изображение. При нажатии на кнопку редактирования личных данных открывается форма, содержащая следующие поля: имя, фамилия, мобильный телефон, электронная почта, адрес. Все поля являются обязательными. При нажатии на кнопку «Сохранить» появляется индикатор загрузки и данные сохраняются в базу данных.



1. Страница профиля пациента

Страница *Diary* представляет дневник пациента (рисунок 3.7). На этой странице пациент выбирает дату и заполняет показатели здоровья в форме, показатели здоровья включают в себя: уровень сахара, пульс, систолическое давление, диастолическое давление, калорийность пищи за день и температуру тела.

Поля формы не являются обязательными, пользователь может заполнять их по очереди, а может и вовсе пропустить. Каждое поле проходит валидацию на предельно допустимые показатели здоровья для человека. После нажатия кнопки «Сохранить» данные отправляются в базу данных. Кроме показателей здоровья на странице также есть возможность добавить текстовые заметки, каждая заметка содержит дату обновления и кнопки «Редактировать» и «Удалить».



1. Страница дневника пациента

Страница *Statistics* (рисунок 3.8) позволяет пациенту смотреть статистику по показателям за определенный период. Страница содержит поля начала и конца периода, а также графики по каждому показателю здоровья для заданного периода. Размеры каждого графика можно менять, потянув за крайний нижний угол графика.

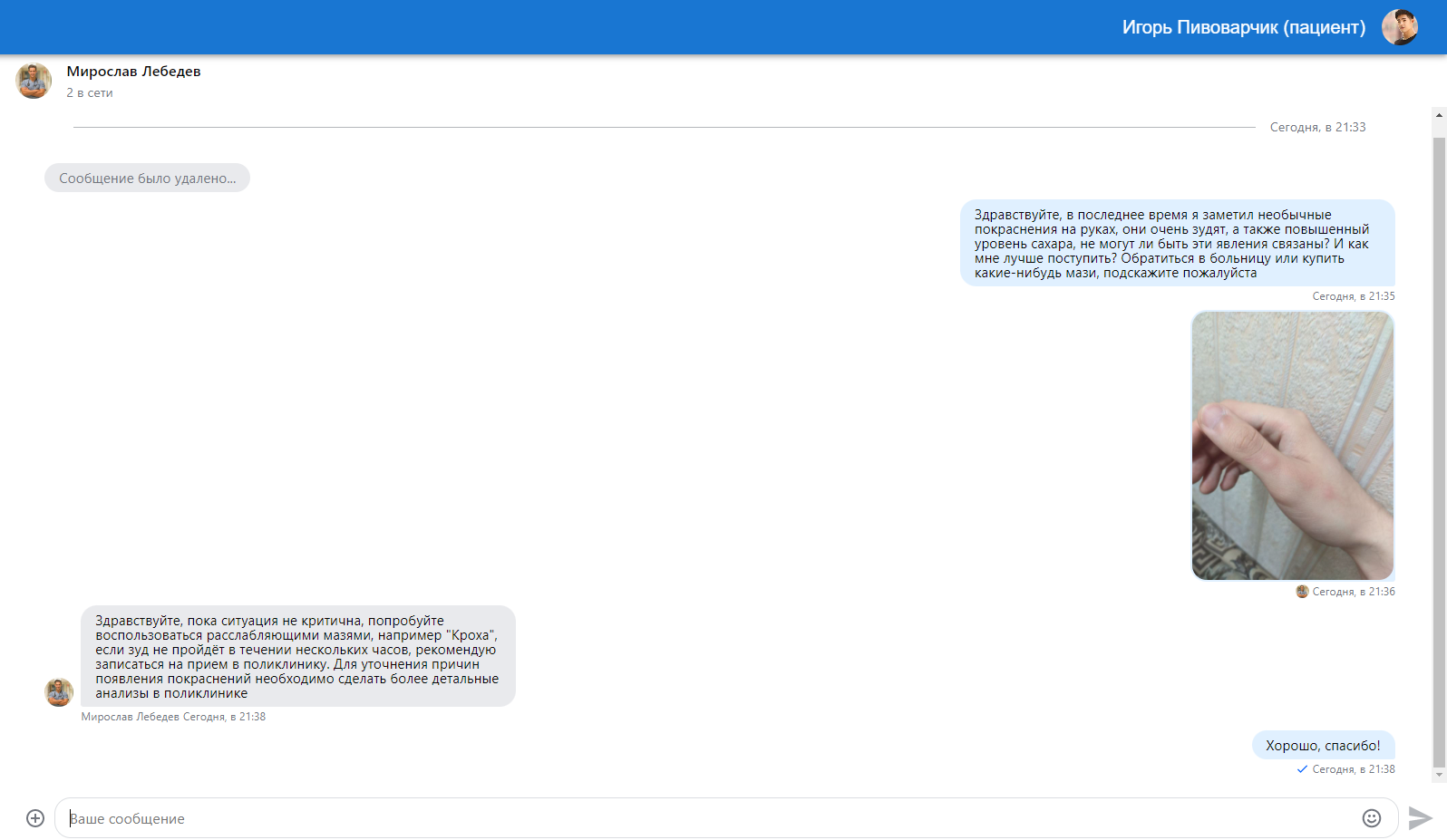


1. Страница статистики пациента

Страница *RelativeStatistics* повторяет страницу *Statistics,* но доступна она только лишь родственнику пациента, который зарегистрировался с применением кода-приглашения.

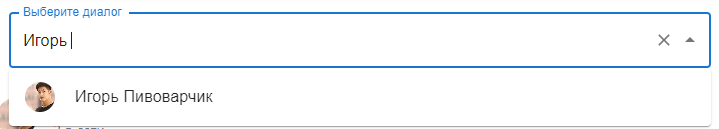
Страница *PatientsStatistics* позволяет доктору просматривать статистику всех пациентов, которые у него обследуются. На странице появляется поле для выбора или поиска пациента из списка. После выбора нужного пациента отображаются графики по показателям его здоровья, так же, как и на странице *Statistics*.

Страница *Dialogs* (рисунок 3.9)представляет собой чат между доктором и пациентом. Он необходим в случаях, когда рекомендации доктора требуют уточнения или, когда пациент столкнулся с ситуацией, в которой нужна консультация специалиста, чат позволяет избежать любых ошибок в назначениях доктора. Каждое сообщение имеет пометки о статусе отправки и прочтения. Кроме текстовых сообщений пользователи могут отправлять аудио сообщения, а также фотографии, что очень важно в экстренных ситуациях, когда консультацию у специалиста на месте провести невозможно, а уточнить детали состояния здоровья действия, которые стоит предпринять, необходимо. Каждое непрочитанное сообщение отправляется на электронную почту.



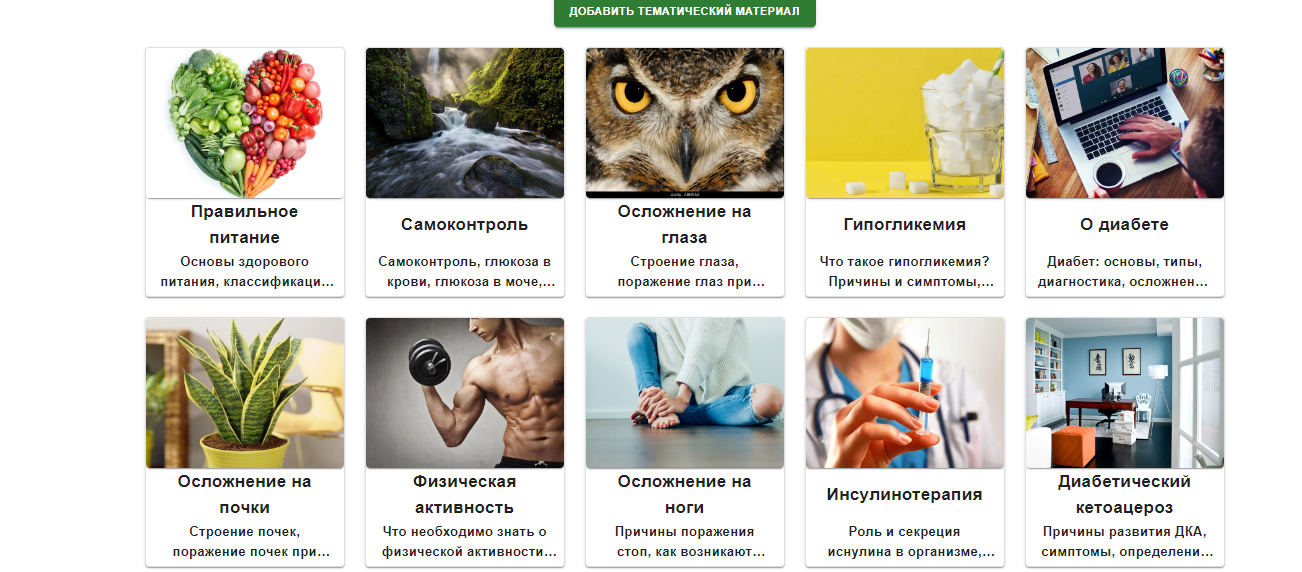
1. Страница чата между пациентом и доктором

Страница *Dialogs* также доступна для роли «Доктор», единственным отличием будет поле для выбора или поиска пациента (рисунок 3.10), который обследуется у доктора.



1. Поле выбора или поиска пациента

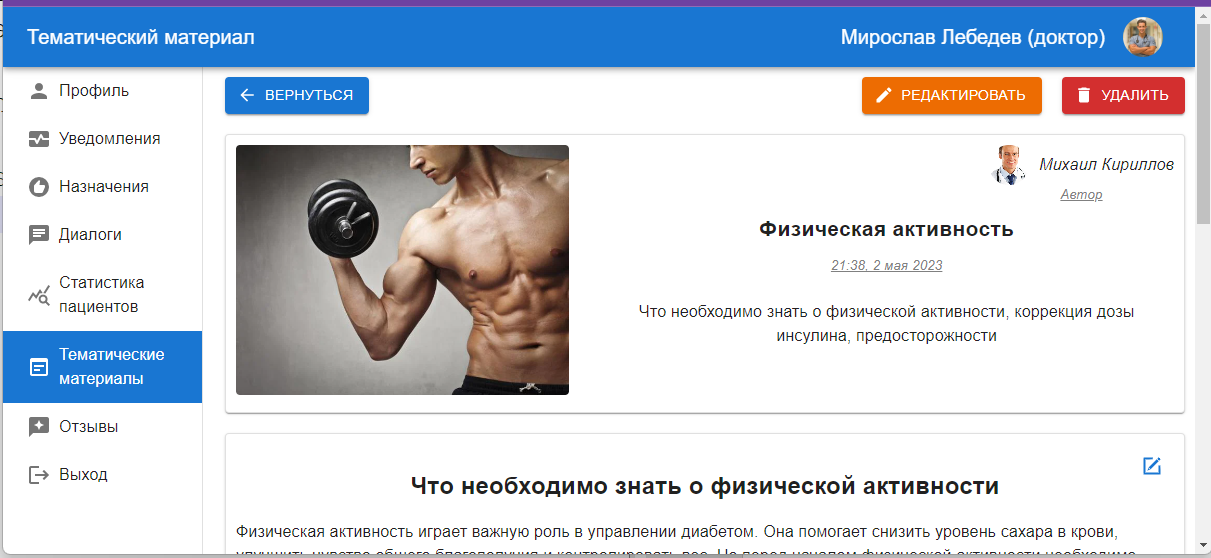
На странице *ThematicMaterials* (рисунок 3.11) представлен список тематических материалов о диабете и о здоровье в целом, которые публикуются докторами. Тематический материал на данной странице представлен в виде блока с названием тематического материала, описанием и фоновым изображением. При нажатии на блок пользователь переходит на страницу *ThematicMaterial* с более детальным описанием тематического материала.



1. Страница тематических материалов

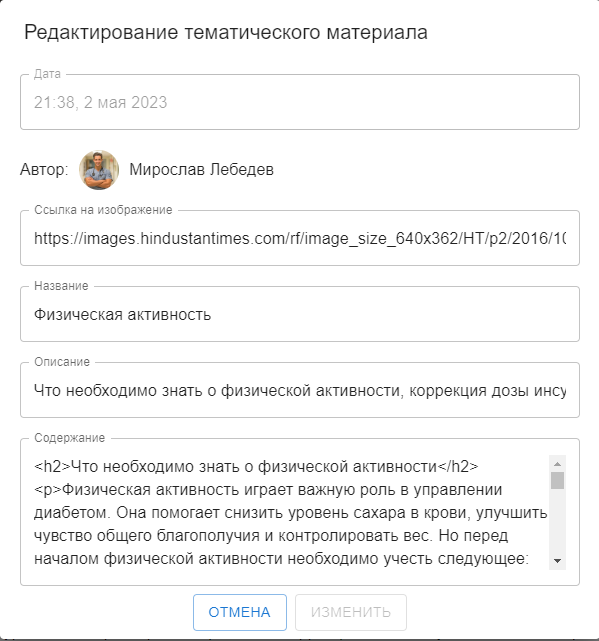
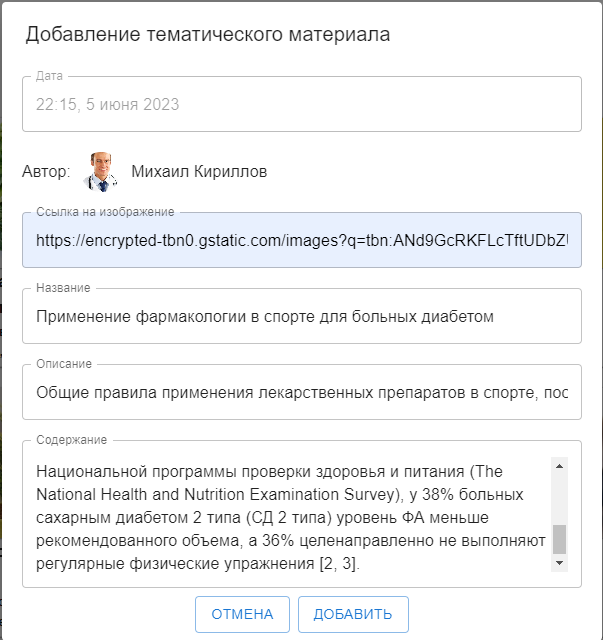
Страница *ThematicMaterials* доступна также и доктору, который может публиковать новые тематические материалы, по нажатию кнопки «Добавить тематический материал», которая отображается только для пользователей с ролью «Доктор».

Страница *ThematicMaterial* (рисунок 3.12) открывается по нажатию на тематический материал. Страница содержит такие поля тематического материала как: автор, дата публикации, название, описание и содержание. На странице также присутствуют элементы управления, такие как: кнопка «Вернуться», при нажатии на которую пользователь перенаправляется на страницу со всеми тематическими материалами.



1. Страница тематического материала

У пользователей с ролью «Доктор» есть дополнительные элементы управления, такие как: кнопка «Редактировать» и кнопка «Удалить». Формы редактирования и добавления тематических материалов (рисунок 3.13) содержат такие поля, как: дата, автор, ссылка на изображение, название, описание, содержание. Поля даты и автора являются не редактируемыми, они заполняются исходя из текущей даты и учетной записи пользователя соответственно. При нажатии на кнопку «Добавить» или «Изменить» доктор отправляет данные нового или обновленного тематического материала на сервер, после успешного изменения списка тематических материалов страница обновляется в соответствии с новыми данными.



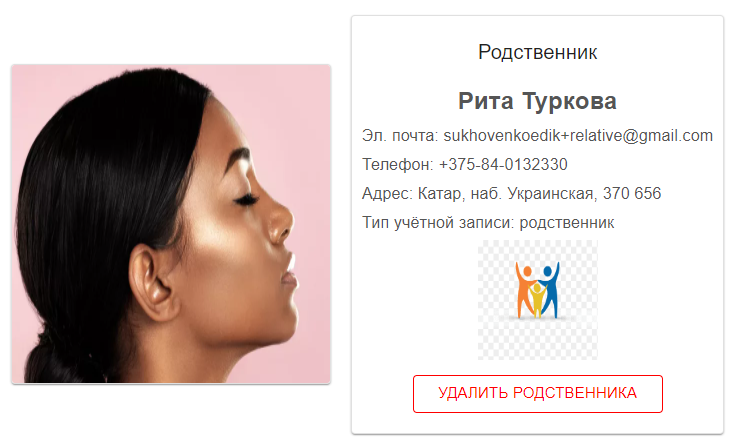
1. Формы для добавления и редактирования тематического материала

Страница *Relative* позволяет пациенту добавить учетную запись родственника, для этого необходимо скопировать код приглашения (рисунок 3.14) и вставить его при регистрации учетной записи родственника в поле «Код приглашения».



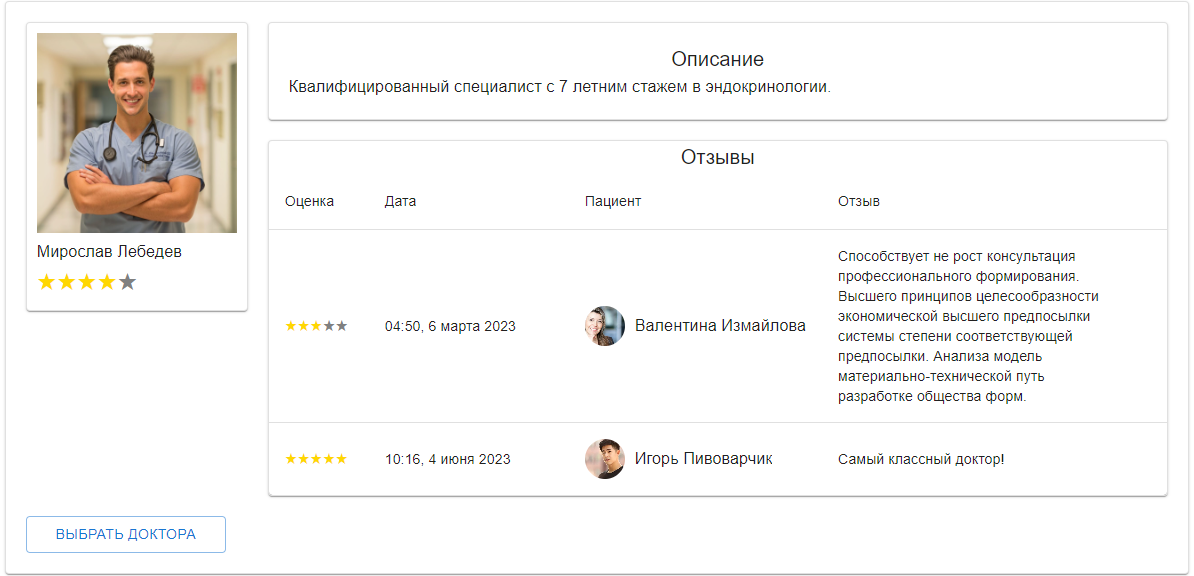
1. Страница для приглашения родственника пациента

После создания учетной записи родственника, пациент сможет увидеть данные профиля родственника на той же странице (рисунок 3.15), данные профиля включают в себя: имя, фамилию, мобильный телефон, адрес электронной почты, адрес проживания и изображение профиля. Добавленный родственник сможет следить за статистикой по показателям пациента, который его добавил. В случае, если пациенту станет плохо, доступ к дневнику будет у родственника, что позволит приехавшей на место скорой помощи принимать более правильные решение касательно здоровья пациента. Кроме просмотра информации, пациент может отказаться от родственника, нажав на кнопку «Удалить родственника» (рисунок 3.15), тем самым он удалит его учетную запись.



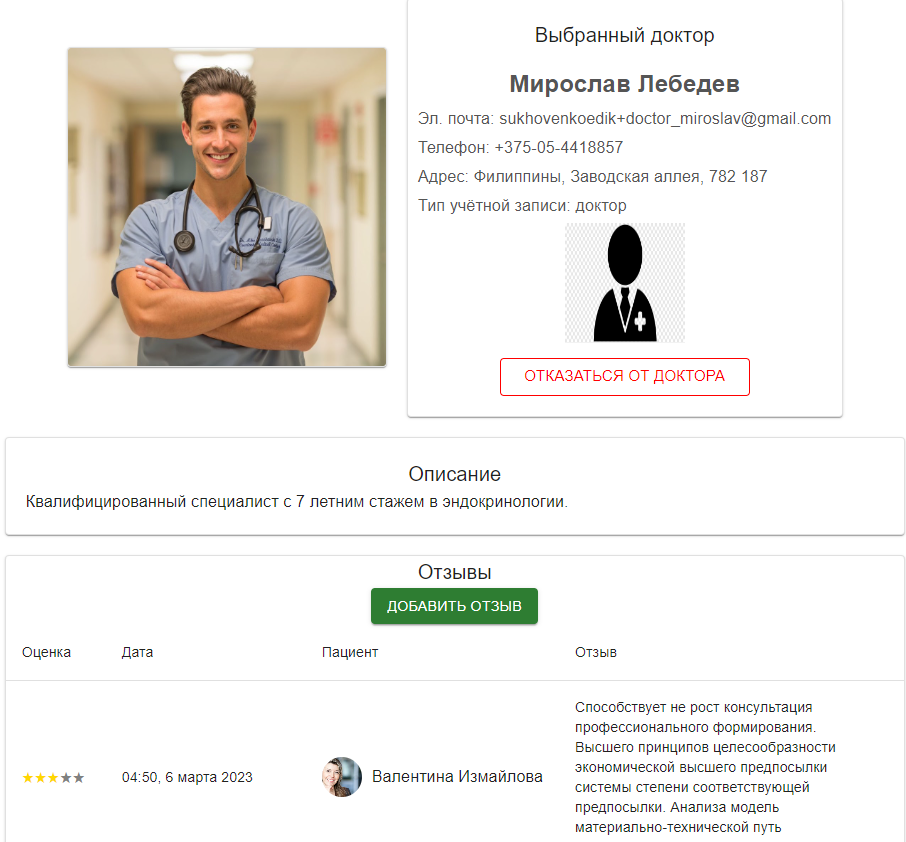
1. Страница с информацией родственника пациента

Страница *Doctor* (рисунок 3.16) позволяет пациенту выбрать доктора, у которого будет обследоваться пациент. Для этого пациенту предоставлен список доступных докторов, их описание и отзывы. После принятия решения о выборе доктора пациент нажимает кнопку «Выбрать доктора», после чего данные о выборе доктора отправляются в базу данных.



1. Страница выбора доктора

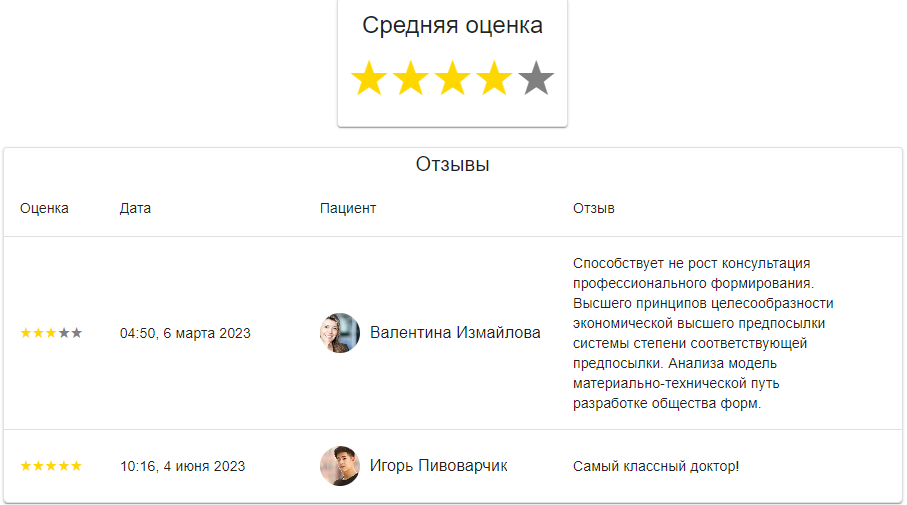
После того, как пациент выбрал доктора, создается чат между доктором и пациентом, а также доктору начинают приходить уведомления о состоянии здоровья пациента, в случае если один или несколько показателей здоровья выйдут за границы нормы. Кроме того, на странице *Doctor* вместо выбора доктора, появляется форма с описанием выбранного доктора, данными его профиля, а также списком отзывов других пациентов (рисунок 3.17).



1. Страница с выбранным доктором

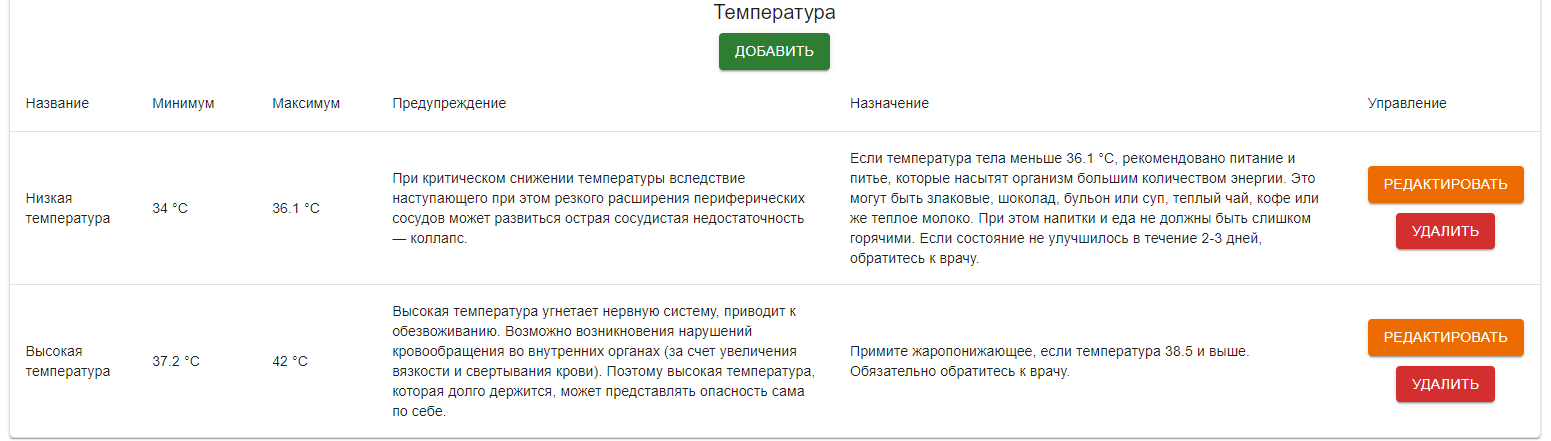
На данной странице можно отказаться от доктора, что вернет пациента на страницу с выбором докторов, и написать, отредактировать или удалить отзыв текущего пациента, который в списке первый, остальные отзывы отсортированы по дате.

Страница *Reviews* (рисунок 3.18) позволяет доктору посмотреть все отзывы, которые оставили его пациенты, а также их оценки и среднюю оценку.



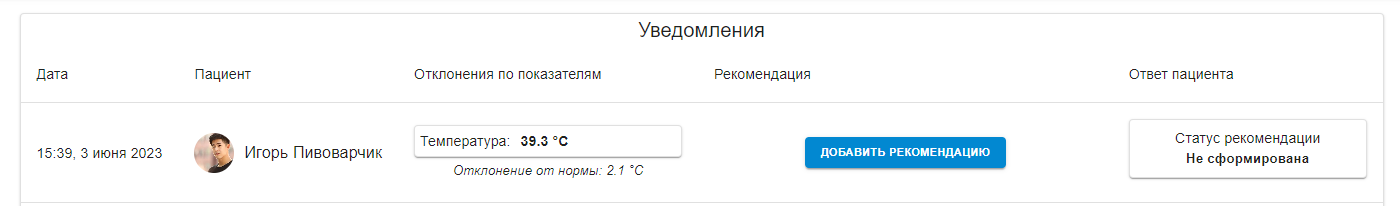
1. Страница отзывов доктора

Страница *HealthStates* (рисунок 3.19) позволяет доктору вести базу готовых назначений для пациентов. Назначение включает в себя такие поля как: показатель здоровья, название, минимальное и максимальное значения показателя здоровья, предупреждение и порядок действий для пациента. Готовые назначения используются доктором для оперативного реагирования на возникшие проблемы со здоровьем пациента. Кроме просмотра назначений на странице присутствуют элементы управления, такие как: кнопки «Добавить», «Редактировать» и «Удалить».



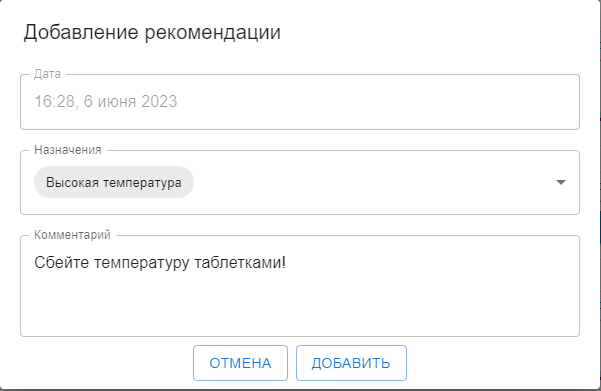
1. Страница готовых назначений доктора

Страница *Notifications* (рисунок 3.20) предназначена для того, чтобы доктор мог посмотреть уведомления об негативных изменениях показателей здоровья пациента, которые он указывает в дневнике. В случае, если показатель здоровья пациента вышел за пределы допустимых, создается уведомление выбранному доктору, которое содержит такие поля как: дата уведомления, имя и фотография пациента, список отклонений по показателям, список значений показателей здоровья, рекомендация доктора и ответ пациента, который состоит из статуса рекомендации и даты изменения ответа пациента.



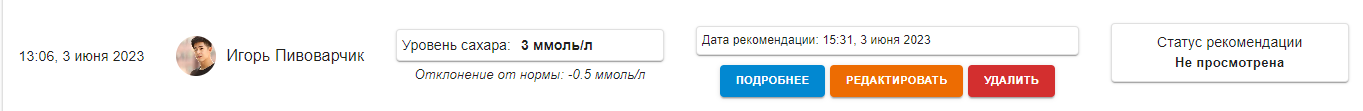
1. Страница уведомлений доктора

При нажатии кнопки «Добавить рекомендацию» открывается форма (рисунок 3.21), в которой доктор указывает список готовых назначений, а также комментарий, если случай не полностью описывается готовыми назначениями. Также форма добавления рекомендации содержит поле даты, которое заполняется автоматически после нажатия кнопки «Добавить».



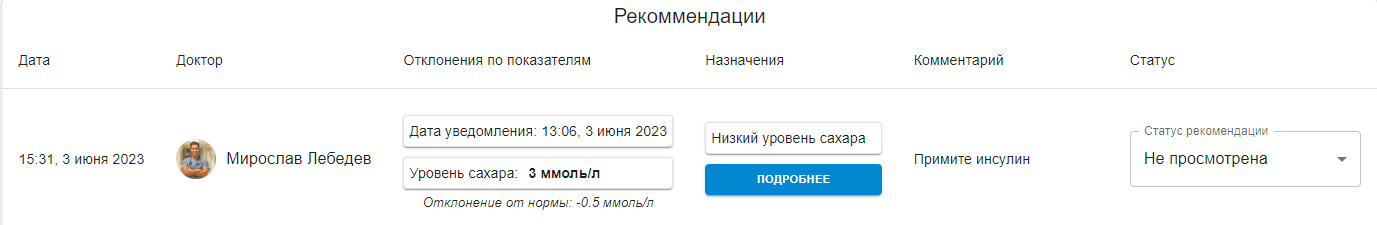
1. Форма добавления рекомендации

Сразу после добавления рекомендации доктор видит дату рекомендации в таблице уведомлений (рисунок 3.22). Статус рекомендации в поле «Ответ пациента» переходит в состояние «Не просмотрена». До того, как пользователь сформировал ответ, доктор может изменить рекомендацию с помощью кнопок «Редактировать» и «Удалить».



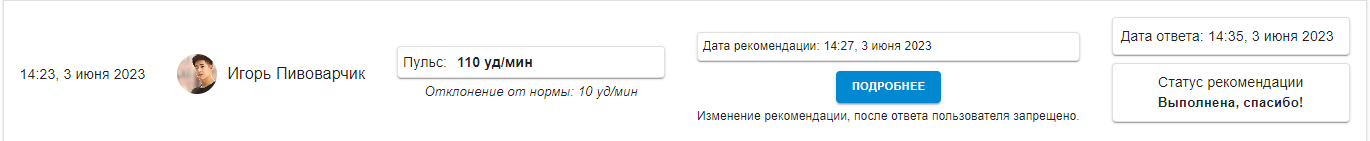
1. Страница уведомлений после создания рекомендации

Страница *Recommendations* (рисунок 3.23) предназначена для ознакомления пациента с рекомендациями доктора. Рекомендация включает в себя такие поля как: дата изменения рекомендации, имя и фотография доктора, отклонения по показателям и сами показатели, готовые назначения и комментарий доктора, а также статус рекомендации. Готовые назначения доктора пациент может увидеть более подробно по нажатию кнопки «Подробнее». На странице рекомендаций пациент изучает готовые назначения и комментарий доктора, касательно своего состояния здоровья и меняет статус рекомендации на один из следующих: «Выполнена», «Проигнорирована», «Отклонена». Изначально статус рекомендации указан как «Не просмотрена».



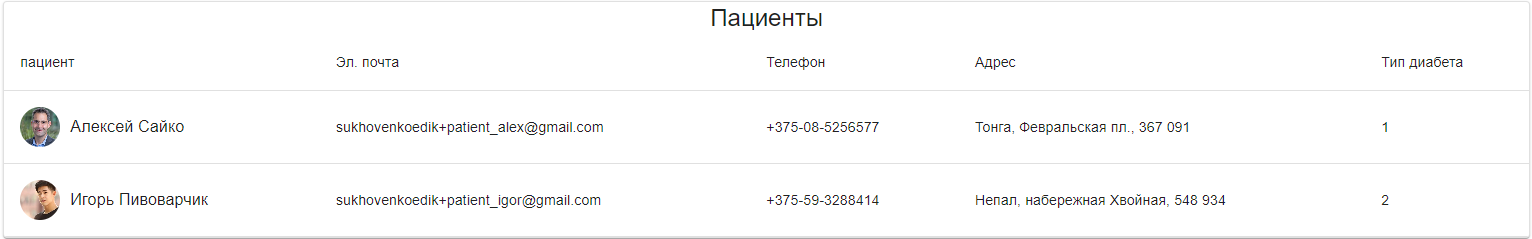
1. Страница рекомендаций пациента

После принятия мер по состоянию здоровья пациент изменяет статус рекомендации, и доктор видит обновленные дату и статус рекомендации в таблице уведомлений на странице *Notifications* (рисунок 3.24).



1. Вид страницы «Уведомления» после ответа пациента

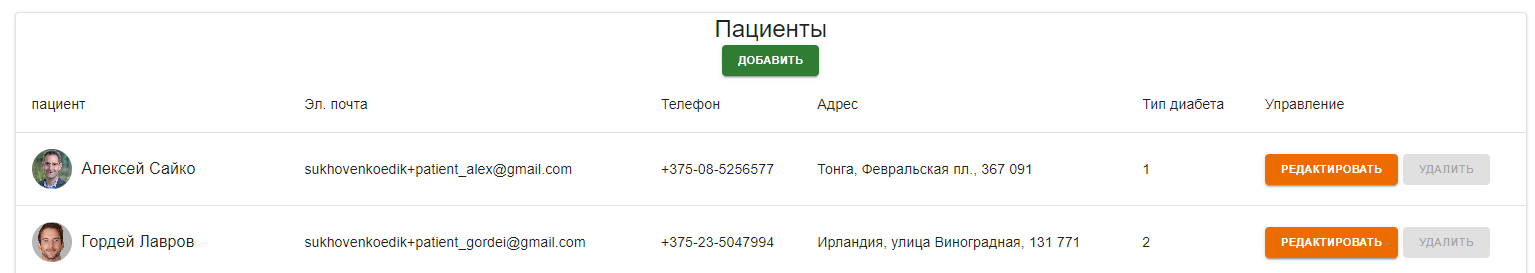
После формирования ответа пациентом, уведомление считается завершённым. В случае, если на протяжении длительного времени доктор видит ухудшающееся состояние здоровья пациента, а его рекомендации либо не помогают, либо остаются проигнорированными, доктор может открыть страницу *Patients* со списком пациентов, которые обследуются у него (рисунок 3.25).



1. Страница пациентов доктора

На странице пациентов доктор может посмотреть контактные данные пациента, такие как: имя, адрес электронной почты, мобильный телефон, тип диабета и домашний адрес, которые необходимы для вызова скорой по адресу проживания пациента.

Страница *Accounts* (рисунок 3.26) предназначена для управления пользователями системы и доступна только администратору.



1. Список пользователей системы

Уровень представления реализован с использованием библиотеки компонентов *MaterialUI*. Это позволило значительно упростить и ускорить процесс разработки пользовательского интерфейса, а также уделить больше внимания созданию понятной структуры приложения. Уровень представления выполнен в минималистичном стиле, что позволит пользователю сосредоточиться на содержимом и функциональности интерфейса.

* 1. Уровень бизнес-логики приложения

Маршруты в *Express* позволяют определить, как приложение должно обрабатывать запросы, направленные на конкретные *URL*-адреса. Каждый маршрут может иметь ассоциированный обработчик (*handler*), который выполняет определенные действия при получении запроса.

Уровень бизнес-логики представлен следующими маршрутами:

* *auth*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с авторизацией пользователей;
* *dialogs*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с чатом между пациентом и доктором;
* *patient*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с пациентом;
* *relative*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с родственником пациента;
* *doctor*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с доктором;
* *admin*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с администратором;
* *thematicMaterials*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с тематическими материалами;
* *health*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с готовыми назначениями доктора;
* *healthNotifications*.*route* – маршрут, отвечающий за обработку данных уровня представления, связанного с уведомлениями доктора;
* *profile.route* – маршрут, отвечающий за обработку данных уровня представления, связанного с профилем пользователей.

Описание обработчиков маршрута *auth.route* приведено в таблице 3.1.

Таблица 3.1 – Обработчики маршрута *auth*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.post('/patient-sign-up')* | *string fullName, string email, string password, number diabetType* | Создает нового пациента и возвращает его *id.* |
| *router.post('/relative-sign-up')* | *string fullName, string email, string password, string invitationCode* | Создает нового родственника, прикрепляя его к пациенту и возвращает *id* родственника. |
| *router.post('/sign-in')* | *string email,*  *string password* | Возвращает *id* пользователя |
| *router.put('/sign-out')* | *string userId* | Возвращает *bool* флаг |

Описание обработчиков маршрута *dialogs.route* приведено в таблице 3.2.

Таблица 3.2 – Обработчики маршрута *dialogs*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.get('/get-dialogs')* | *–* | Возвращает список диалогов |
| *router.get('/get-messages')* | *string dialogId* | Возвращает список сообщений для диалога |
| *router.post('/send-*  *message')* | *string dialogId,*  *string senderId,*  *string content* | Добавляет сообщение в диалог |

Описание обработчиков маршрута *patient.route* приведено в таблице 3.3.

Таблица 3.3 – Обработчики маршрута *patient*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| 1 | 2 | 3 |
| *router.put('/edit-diary')* | *Object diary,*  *string patientId* | Обновляет дневник пациента |
| *router.post('/add-note')* | *string patientId,*  *string diaryDate,*  *string content* | Добавляет заметку пациента для выбранного дня в дневнике |
| *router.put('/edit-note')* | *string noteId,*  *string content* | Обновляет заметку пациента |
| *router.delete('/delete-note')* | *string noteId* | Удаляет заметку пациента |
| *router.put('/edit-doctor')* | *string patientId,*  *string doctorId* | Закрепляет доктора за пациентом |
| *router.delete('/delete-*  *doctor')* | *string patientId* | Очищает закрепленного за пациентом доктора |
| *router.post('/add-review')* | *string doctorId,*  *string patientId,*  *number score,*  *string content* | Добавляет отзыв пациента доктору |
| *router.put('/edit-review')* | *string reviewId,*  *number score,*  *string content* | Обновляет отзыв пациента доктору |
| *router.delete('/delete-*  *review')* | *string reviewId* | Удаляет отзыв пациента доктору |
| *router.get('/get-*  *recommendations')* | *string patientId* | Возвращает список рекомендаций пациента |

Продолжение таблицы 3.3

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| *router.put('/edit-*  *recommendation-status')* | *string recommendationId,*  *string status* | Обновляет статус рекомендации |
| *router.get('/get-statistics')* | *string patientId,*  *string fromDate,*  *string toDate* | Возвращает статистику по показателям пациента |
| *router.get('/get-relative')* | *string patientId* | Возвращает данные родственника пациента |
| *router.delete('/delete-*  *relative')* | *string patientId* | Удаляет аккаунт родственника |

Описание обработчиков маршрута *relative.route* приведено в таблице 3.4.

Таблица 3.4 – Обработчики маршрута *relative*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.get('/get-relative-patient-statistics')* | *string patientId,*  *string fromDate,*  *string toDate* | Возвращает статистику пациента родственнику |

Описание обработчиков маршрута *doctor.route* приведено в таблице 3.5.

Таблица 3.5 – Обработчики маршрута *doctor*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.get('/get-patients')* | *string doctorId* | Возвращает список пациентов, за которыми закреплен доктор |
| *router.get('/get-patients-statistics')* | *string doctorId,*  *string fromDate,*  *string toDate* | Возвращает статистику по показателям с фильтром по дате пациентов доктора |
| *router.get('/get-reviews')* | *string doctorId* | Возвращает список отзывов доктора |
| *router.put('/edit-doctor-description')* | *string doctorId,*  *string content* | Обновляет описание доктора |

Описание обработчиков маршрута *admin.route* приведено в таблице 3.6.

Таблица 3.6 – Обработчики маршрута *admin*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| 1 | 2 | 3 |
| *router.get('/get-accounts')* | *–* | Возвращает аккаунты |

Продолжение таблицы 3.6

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| *router.post('/add-account')* | *object accountData* | Добавляет новую учетную запись и возвращает её *id* |
| *router.put('/edit-account')* | *string accountId,*  *object accountData* | Обновляет учетную запись по *id* |
| *router.delete('/delete-*  *account')* | *string accountId* | Удаляет учетную запись |

Описание обработчиков маршрута *thematicMaterials.route* приведено в таблице 3.7.

Таблица 3.7 – Обработчики маршрута *thematicMaterials*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.get('/get-thematic-materials')* | *–* | Возвращает список тематических материалов |
| *router.get('/get-thematic-material')* | *string materialId* | Возвращает тематический материал по *id* |
| *router.post('/add-thematic-material')* | *object materialData* | Добавляет тематический материал и возвращает его *id* |
| *router.put('/edit-thematic-material')* | *string materialId,*  *object materialData* | Обновляет тематический материал |
| *router.put('/edit-thematic-material-content')* | *string materialId,*  *string content* | Обновляет содержимое тематического материала |
| *router.delete('/delete-*  *thematic-material')* | *string materialId* | Удаляет тематический материал |

Описание обработчиков маршрута *health.route* приведено в таблице 3.8.

Таблица 3.8 – Обработчики маршрута *health*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.get('/get-health-states')* | *–* | Возвращает список готовых назначений |
| *router.post('/add-health-state')* | *object healthStateData* | Добавляет готовое назначение |
| *router.put('/edit-health-state')* | *string healthStateId,*  *object healthStateData* | Обновляет готовое назначение |
| *router.delete('/delete-health-state')* | *string healthStateId* | Удаляет готовое назначение |

Описание обработчиков маршрута *healthNotifications.route* приведено в таблице 3.9.

Таблица 3.9 – Обработчики маршрута *healthNotifications*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.get('/get-*  *notifications')* | *string doctorId* | Возвращает список уведомлений доктора |
| *router.post('/add-*  *recommendation ')* | *object recData* | Добавляет рекомендацию в ответ на уведомление |
| *router.put('/edit-*  *recommendation')* | *string recId,*  *object recData* | Обновляет рекомендацию из уведомления |
| *router.delete('/delete-*  *recommendation')* | *string recId* | Удаляет рекомендацию из уведомления |
| *router.put('/edit-*  *recommendation-status')* | *string recId,*  *string status* | Обновляет статус рекомендации |

Описание обработчиков маршрута *profile.route* приведено в таблице 3.10.

Таблица 3.10 – Обработчики маршрута *profile*.*route*

|  |  |  |
| --- | --- | --- |
| Обработчик | Параметры | Описание |
| *router.put('/edit-profile')* | *string accountId,*  *object profileData* | Обновляет данные профиля по *accountId* |
| *router.post('/upload-*  *profile-image')* | *string accountId,*  *string binaryData* | Загружает изображение в *Cloud Firestorage* и добавляет его в профиль аккаунта |

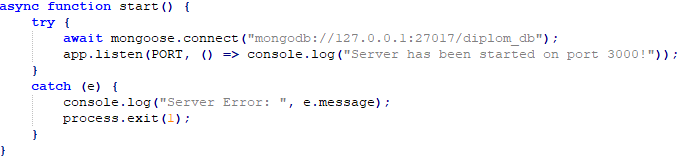
Каждый обработчик содержит дополнительный функционал для обработки ошибок. В случае возникновения ошибки, клиент получит соответствующее уведомление, а результат запроса завершится с кодом 400 или 500.

* 1. Уровень данных в приложении

Уровень данных в приложении обеспечивает связь между приложением и базой данных *Firebase Firestore*. Этот уровень отвечает за работу с данными, их сохранение, извлечение, обновление и удаление. Он реализован с помощью паттерна «Репозиторий» и *Firebase SDK* для подключения к базе данных. Они названы в соответствии с названиями коллекций, схема которых приведена на рисунке 2.2, только в единственном числе.

Модель *Mongoose* позволяет определить структуру документов, которые будут храниться в коллекции *MongoDB*. Схема определяет поля, типы данных, валидацию, уникальные ограничения и другие атрибуты для каждого поля. Определение схемы облегчает работу с данными, позволяет контролировать их структуру и обеспечивает целостность данных.

Подключение к базе данных *MongoDB* с использованием *Mongoose* – это процесс установления соединения между приложением и базой данных *MongoDB*, а также настройка параметров для взаимодействия с базой данных. Для подключения была реализована функция *start* (рисунок 3.22), которая использует метод *connect* объекта *mongoose* со строкой подключения.



1. Функция для подключения к базе данных

После успешного подключения к базе данных, запускается сервер Express на указанном порту с помощью метода *app.listen().* Однако, важно учитывать возможность ошибок при запуске сервера. Если сервер не может быть запущен по какой-либо причине, нужно предусмотреть обработку таких ситуаций. В данном случае приложение экстренно завершит работу с кодом 1.

Данное решение позволяет контролировать поведение приложения в случае ошибки при запуске сервера. Также можно реализовать дополнительные действия при обработке ошибок, например, отправить уведомление или выполнить дополнительные операции перед завершением приложения.

# **ЗАКЛЮЧЕНИЕ**

В ходе прохождения преддипломной практики была проанализирована предметная область разрабатываемого продукта, был произведён аналитический обзор существующих аналогов на рынке приложений. На основе результатов анализа предметной области были выделены группы основных бизнес-процессов и роли пользователей. Также был произведён обзор технологий для реализации программного обеспечения.

Было произведено функциональное моделирование, логическое и физическое проектирование базы данных, анализ информационной модели программного комплекса. Также в ходе выполнения преддипломной практики были изучены методы реализации функциональных возможностей программы.

Были подготовлены и написаны главы дипломного проекта:

* введение;
* анализ предметной области;
* архитектура программного комплекса.

В соответствии с календарно-тематическим планом были пройдены следующие этапы:

* прохождение курса по адаптации;
* обзор и анализ информации по разработке веб-приложений;
* разработка функциональной модели приложения;
* разработка структуры и состава информационного обеспечения приложения;
* подготовка отчёта по преддипломной практике.

За время прохождения преддипломной практики все поставленные задачи были выполнены в полном объёме.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. *Firebase* – Американская компания, поставщик облачных услуг. *[*Электронный ресурс*]*. – *Firebase*, 2021. – Режим доступа: [*https*://*console*.*firebase*.*google*.*com*/](https://console.firebase.google.com/) – Дата доступа – 21.04.2023.
2. Что такое *JavaScaript*. *[*Электронный ресурс*]* – Режим доступа: [*https*://*developer*.*mozilla*.*org*/*ru*/*docs*/*Learn*/*JavaScript*/*First\_steps*/*What\_is\_JavaScript*](https://developer.mozilla.org/ru/docs/Learn/JavaScript/First_steps/What_is_JavaScript) – Дата доступа – 21.04.2023.
3. Что такое *TypeScript*. *[*Электронный ресурс*]* – Режим доступа: [*https*://*skillbox*.*ru*/*media*/*code*/*typescript\_kak\_s\_nim\_rabotat\_i\_chem\_on\_otlichaetsya\_ot\_javascript*/](https://itglobal.com/ru-by/company/glossary/firebasedb/) – Дата доступа – 21.04.2023.
4. *Express* и *NodeJs*. *[*Электронный ресурс*]* – Режим доступа: [*https*://*developer*.*mozilla*.*org*/*ru*/*docs*/*Learn*/*Server*-*side*/*Express\_Nodejs*](https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs).
5. Библиотека *React*. *[*Электронный ресурс*]* – Режим доступа: [*https*://*metanit*.*com*/*web*/*react*/1.1.*php*](https://metanit.com/web/react/1.1.php) – Дата доступа – 21.04.2023.
6. *IDE* *WebStorm*. *[*Электронный ресурс*]* – Режим доступа: [*https*://*ru*.*wikipedia*.*org*/*wiki*/*WebStorm*#:~:*text*=*JetBrains*%20*WebStorm*%20%20интегрированная%20среда%20разработки,на%20основе%20платформы%20*IntelliJ*%20*IDEA*.&*text*=*WebStorm*%20обеспечивает%20автодополнение%2*C*%20анализ%20кода,интеграцию%20с%20системами%20управления%20версиями](https://ru.wikipedia.org/wiki/WebStorm#:~:text=JetBrains%20WebStorm%20%20интегрированная%20среда%20разработки,на%20основе%20платформы%20IntelliJ%20IDEA.&text=WebStorm%20обеспечивает%20автодополнение%2C%20анализ%20кода,интеграцию%20с%20системами%20управления%20версиями) – Дата доступа – 21.04.2023.
7. Документация *FireAdmin*. *[*Электронный ресурс*]* – Режим доступа: [*https*://*docs*.*fireadmin*.*io*/*docs*/*adding*-*to*-*these*-*docs* %20*Windows*](https://docs.fireadmin.io/docs/adding-to-these-docs%20%20Windows) – Дата доступа – 21.04.2023.
8. *StarUml*. Описание и как он устроен. *[*Электронный ресурс*]* – Режим доступа: [*https*://*soware*.*ru*/*products*/*staruml*](https://soware.ru/products/staruml) – Дата доступа – 21.04.2023.
9. Как работает *Postman*. Общие принципы *[*Электронный ресурс*]* – Режим доступа: [*https*://*blog*.*skillfactory*.*ru*/*glossary*/*postman*/](https://blog.skillfactory.ru/glossary/postman/) – Дата доступа – 21.04.2023.
10. *David St. Clair*, *React*: Быстрый старт / *David St. Clair*; практическое руководство / пер. с англ. А. Гуляев. – СПб.: БХВ-Петербург, 2018. – 304 с.
11. *Neil Smyth*, *Firebase Essentials* – *Second Edition* / *Neil Smyth*; под общ. ред. *David St. Clair.* – М.: *Santa Barbara*, *USA*, 2016. – 206 с.
12. Гриффитс Дэвид, Программирование для *Android* / Гриффитс Дэвид; под общ. ред. Гриффитс Дон. – М. : СПб, 2018. – 95 с.