

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»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

**к дипломной работе
на тему**

**«Программный комплекс для автоматизации ведения дневника
диабетика»**

Разработал студент гр. ИП-42	_____	<u>Суховенко Э.С.</u>
	(подпись)	(Ф.И.О.)
Руководитель работы	_____	<u>доцент, к.т.н. Прокопенко Д.В.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Консультант по экономической части	_____	<u>доцент, к.э.н. Соловьева Л.Л.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Консультант по охране труда и технике безопасности	_____	<u>профессор, д.т.н., Кудин В.П.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Нормоконтроль	_____	<u>Самовендюк Н.В.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)
Рецензент	_____	_____
	(подпись)	(ученое звание, ученая степень, должность, организация, Ф.И.О.)

Дипломная работа (_____ с.) допущена к защите
в Государственной экзаменационной комиссии.

Зав. кафедрой	_____	<u>доцент, к.т.н, Трохова Т.А.</u>
	(подпись)	(ученое звание, ученая степень, Ф.И.О.)

Гомель 2023

Лист задания

Реферат

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ АВТОМАТИЗАЦИИ ВЕДЕНИЯ ДНЕВНИКА ДИАБЕТИКА: дипломная работа / Э. С. Суховенко. – Гомель : ГГТУ им. П.О. Сухого, 2023. – Дипломная работа: 83 страницы, 45 рисунков, 26 таблиц, 14 источников, 6 приложений.

Ключевые слова: диабет, пациент, доктор, дневник, показатели здоровья, рекомендации доктора.

Объектом разработки является программный продукт, направленный на автоматизацию ведения дневника диабетика.

Целью дипломной работы является создание программного комплекса для автоматизации ведения дневника диабетика. Данный программный комплекс будет предназначен для упрощения и автоматизации процессов, связанных с ведением дневника пациентов, больных диабетом.

В процессе работы было сделано следующее: выполнен анализ существующего программного обеспечения по автоматизации дневника диабетика, создана база данных для хранения и обработки учетных записей и личных данных пользователей, а также показателей здоровья, уведомлений, рекомендаций, тематических материалов; произведен экономический анализ и обоснована рентабельность разработки; проведен анализ внедрения разработки с учетом аспектов энерго- и ресурсосбережения.

Областью практического использования является применение этого программного обеспечения в поликлиниках.

Студент-дипломник подтверждает, что дипломная работа выполнена самостоятельно, приведенный в дипломной работе материал объективно отражает состояние разрабатываемого объекта, пояснительная записка проверена в системе «Антиплагиат» «АО "Антиплагиат"» (режим доступа: <https://www.antiplagiat.ru/>). Процент оригинальности составляет 81.23%. Все заимствованные из литературных и других источников, теоретические и методологические положения и концепции сопровождаются ссылками на источники, указанные в «Списке использованных источников».

Резюме

Тема дипломной работы «Программный комплекс для автоматизации ведения дневника диабетика».

В процессе выполнения данной дипломной работы был разработан программный продукт, который предназначен для упрощения и автоматизации процессов, связанных ведением дневника диабетика, изучением тематических материалов о диабете и проведением консультаций с доктором.

Объектом разработки является программный продукт, направленный на автоматизацию ведения дневника диабетика.

Поставленная задача была выполнена в полном объеме.

Резюме

Тэма дыпломнай працы «Праграмны комплекс для аўтаматызацыі вядзення дзённіка дыябетыка».

У працэсе выканання дадзенай дыпломнай працы быў распрацаваны праграмны прадукт, які прызначаны для спрашчэння і аўтаматызацыі працэсаў, звязаных вядзеннем дзённіка дыябетыка, вывучэннем тэматычных матэрыялаў аб дыябеце і правядзеннем кансультацый з доктарам.

Аб'ектам распрацоўкі з'яўляецца праграмны прадукт, накіраваны на аўтаматызацыю вядзення дзённіка дыябетыка.

Пастаўленая задача была выканана ў поўным аб'ёме.

Summary

The theme of the thesis is «Software package for automating the keeping of a diabetic's diary».

In the process of completing this thesis, a software product was developed that is designed to simplify and automate the processes associated with keeping a diabetic diary, studying thematic materials about diabetes, and consulting with a doctor.

The object of development is a software product aimed at automating the keeping of a diabetic diary.

The task was fully completed.

СОДЕРЖАНИЕ

Введение.....	6
1 Аналитический обзор методов автоматизации дневника диабетика.....	7
1.1 Анализ предметной области	7
1.2 Аналитический обзор существующих аналогов.....	8
1.3 Обзор технологий для реализации программного обеспечения	9
1.4 Анализ инструментальных средств автоматизации разработки	14
1.5 Постановка задачи на дипломное проектирование	18
2 Архитектура программного обеспечения.....	19
2.1 Общая структура приложения	19
2.2 Функциональная модель программного комплекса	21
2.3 Информационная модель программного комплекса.....	25
3 Программная реализация комплекса.....	31
3.1 Архитектура приложения.....	31
3.2 Уровень представления клиентской части приложения	32
3.3 Уровень бизнес-логики приложения	45
3.4 Уровень данных в приложении	49
4 Тестирование, верификация и валидация программного комплекса....	Ошибка!
Закладка не определена.	
4.1 Тестирование пользовательских форм	51
4.2 Модульное тестирование бизнес-логики.....	54
5 Экономическое обоснование дипломной работы	62
5.1 Техничко-экономическое обоснование целесообразности разработки программного продукта.....	62
5.2 Расчет общей трудоемкости разработки программного обеспечения.	62
5.3 Расчет совокупных капитальных вложений в проект	66
5.5 Формирование цены при создании программного обеспечения.....	73
5.6 Статистическая оценка экономической эффективности проекта.....	74
6 Охрана труда и техника безопасности	76
6.1 Основные понятия охраны труда.....	76
6.2 Оздоровление воздушной среды на предприятии	76
6.3 Вентиляция и системы воздухообмена	77
6.4 Управление выбросами и технологии очистки	78
6.5 Контроль качества воздуха и мониторинг загрязнений.....	79
7 Ресурсо- и энергосбережение.....	80
7.1 Экономия ресурсов при использовании программного продукта	80
Заключение	82
Список использованной литературы.....	83
Приложение А Листинг программы.....	84
Приложение Б Каталог функций программного обеспечения	114
Приложение В Расчет общей трудоемкости разработки	115
Приложение Г Параметры для расчета производственных затрат	116
Приложение Д Расчет суммарных затрат на разработку	117
Приложение Е Техничко-экономические показатели проекта	118

ВВЕДЕНИЕ

В современном мире многие люди сталкиваются с проблемами, связанными с диабетом. Это хроническое заболевание требует постоянного контроля и учета различных показателей здоровья, таких как уровень глюкозы, питание, физическая активность и другие. Однако, ведение дневника диабетика может быть сложным и трудоемким процессом.

В связи с этим, разработка программного комплекса для автоматизации ведения дневника диабетика представляет собой актуальную задачу. Данный комплекс позволит докторам, пациентам и их родственникам эффективно вести учет и контроль показателей здоровья.

Целью данной дипломной работы является разработка и реализация программного комплекса, который предоставит удобный и надежный инструмент для автоматизации ведения дневника диабетика. Комплекс будет ориентирован на использование как докторами, так и пациентами, а также их родственниками, предоставляя им возможность удобно и эффективно отслеживать, и анализировать показатели здоровья.

В рамках дипломной работы будет разработана информационная модель программного комплекса, охватывающая основные сущности, такие как дневники, цели, медицинские данные, комментарии, сотрудники и другие. Также будет реализован функционал взаимодействия между пользователями через чат, обеспечивающий удобную и конфиденциальную коммуникацию.

Итогом данной работы будет готовый программный комплекс, который позволит докторам более эффективно контролировать состояние пациентов, пациентам удобно вести дневник и обмениваться информацией, а родственникам быть в курсе состояния здоровья своих близких. Результаты работы могут быть применены в медицинской сфере, с целью повышения качества жизни диабетиков и облегчения процесса учета и контроля их здоровья.

Задачами дипломной работы являются:

- изучение методик разработки клиент-серверных приложений;
- классификация ролей пользователей и их ролевые политики;
- изучение методов реализации серверной части для приложения;
- проектирование структуры приложения, базы данных для хранения информации, формирование пользовательских правил для доступа к ресурсам и функциям приложения;
- разработка программных модулей: обеспечивающих авторизацию и аутентификацию пользователей; работу с данными с помощью графического интерфейса;
- верификация и опытная эксплуатация разработанного программного обеспечения.

1 АНАЛИТИЧЕСКИЙ ОБЗОР МЕТОДОВ АВТОМАТИЗАЦИИ ДНЕВНИКА ДИАБЕТИКА

1.1 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 эффектив-

ность 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.

1.2 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.

1.3 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*. Архитектура стека *FERN* представляет собой взаимодействие между этими компонентами. *Firebase* предоставляет облачные сервисы и инструменты, которые могут быть использованы как клиентской, так и серверной стороной приложения. *Express* работает в качестве серверного фреймворка, обрабатывая *HTTP*-запросы и управляя бизнес-логикой приложения, а также выполняя валидацию входных данных, обеспечивая целостность базы данных. *React* используется для создания пользовательского интерфейса на клиентской стороне, обеспечивая отзывчивость и переиспользование компонентов. *Node.js* обеспечивает выполнение *JavaScript* кода на сервере и интеграцию с другими системами и сервисами. На рисунке 1.1 представлены компоненты основные компоненты стека *FERN* и архитектура их взаимодействия.

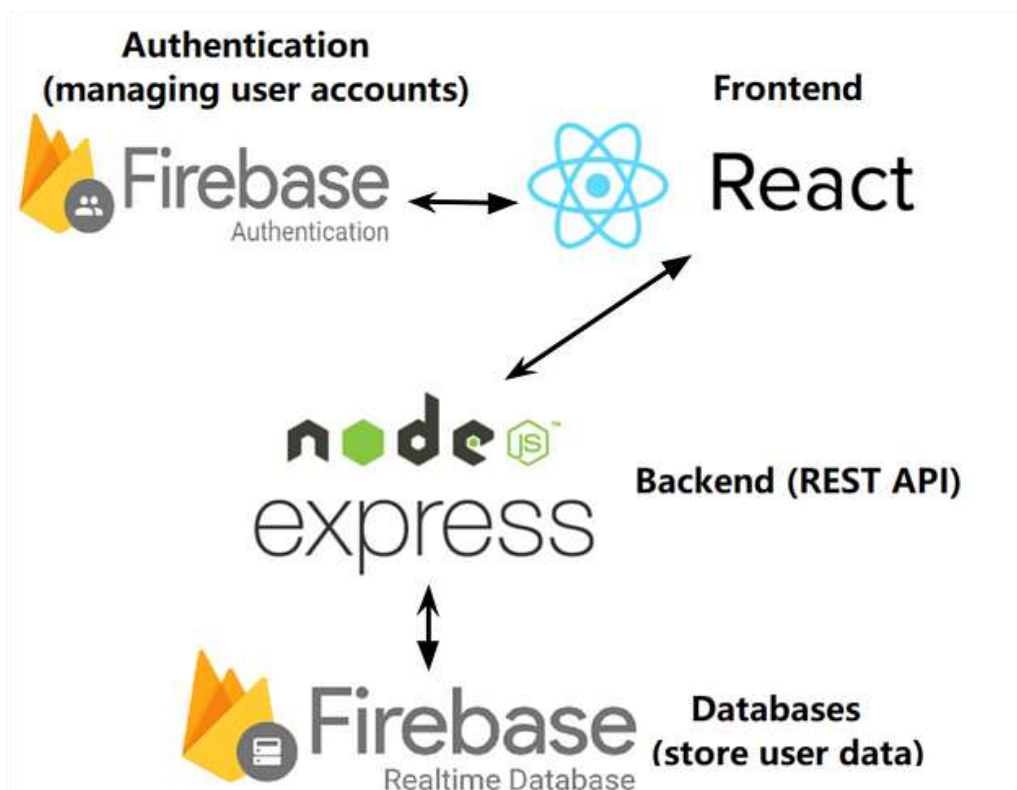


Рисунок 1.1 – Компоненты стека *FERN*

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

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. *JavaScript* имеет динамическую типизацию, что означает, что переменные могут менять тип данных в процессе выполнения.

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*) [1].

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 [2].

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 [3].

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 [4].

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 [5].

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 слож-

ных 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 [6].

Node.js – это среда выполнения *JavaScript* на стороне сервера, которая позволяет разрабатывать высокопроизводительные и масштабируемые веб-приложения. Она основана на движке *V8*, 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 *V8*, 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.

1.4 Анализ инструментальных средств автоматизации разработки

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 быстрое действие 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 [7].

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 [8].

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.

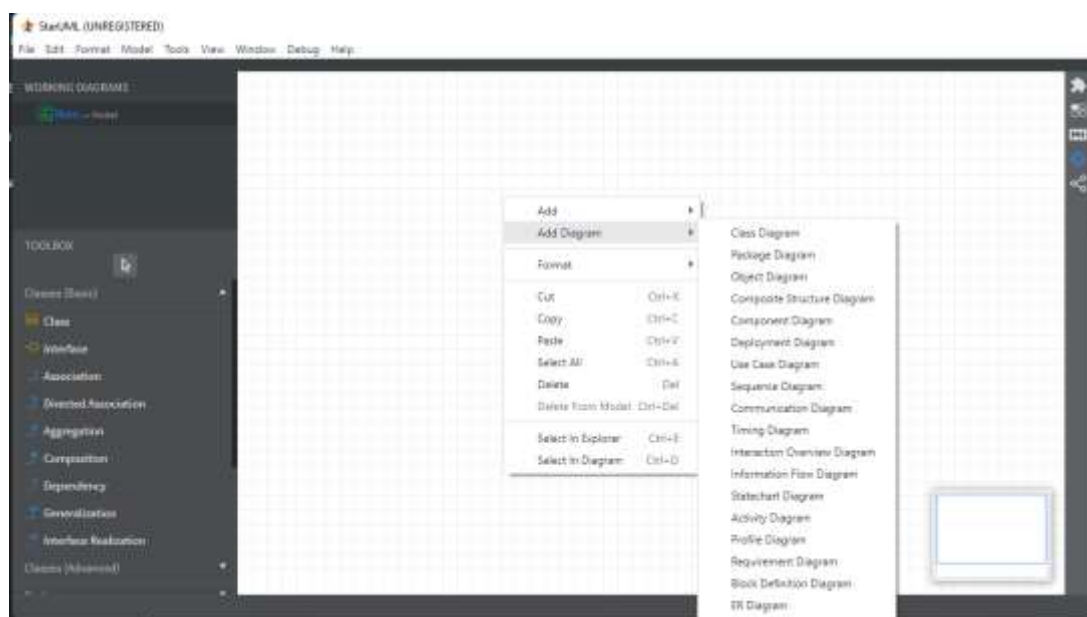


Рисунок 1.2 – Интерфейс главного окна *StarUml*

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 – 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 [9].

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. 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 [10].

1.5 Постановка задачи на дипломное проектирование

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

Данный программный комплекс предназначен 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).

2 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1 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 [11].

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

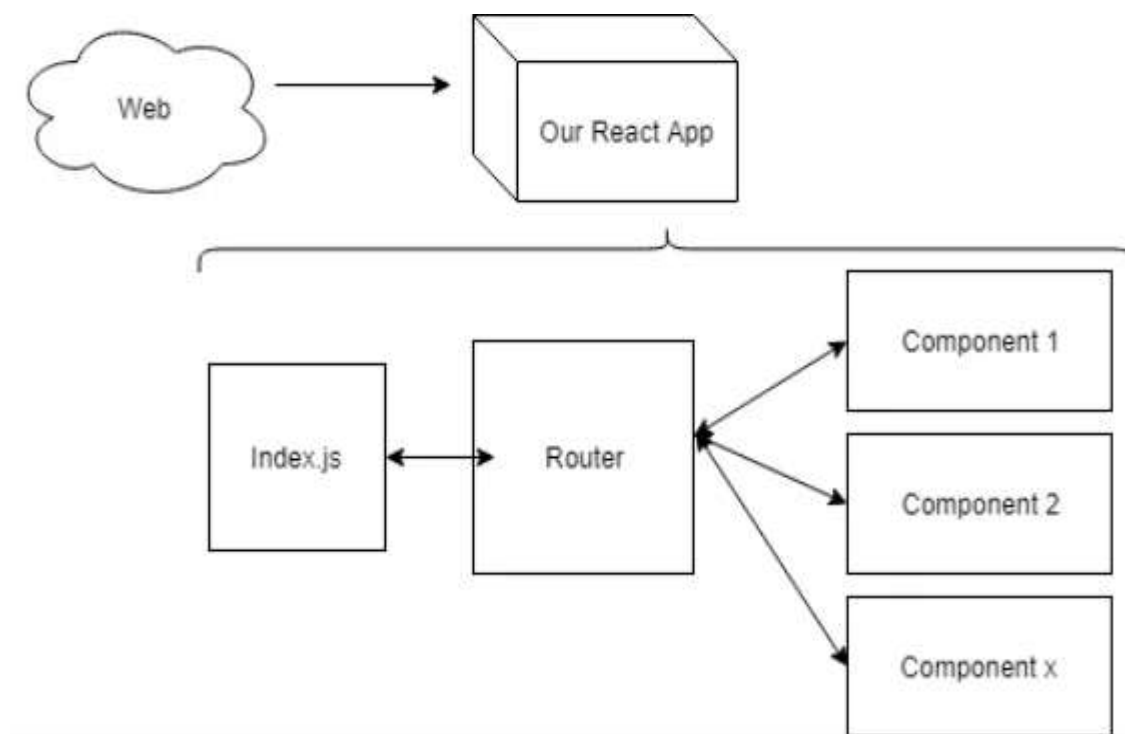


Рисунок 2.1 – Разбиение на компоненты в *React*-приложении

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.

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 [12].

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 2.2 представлена архитектура *Firebase*.

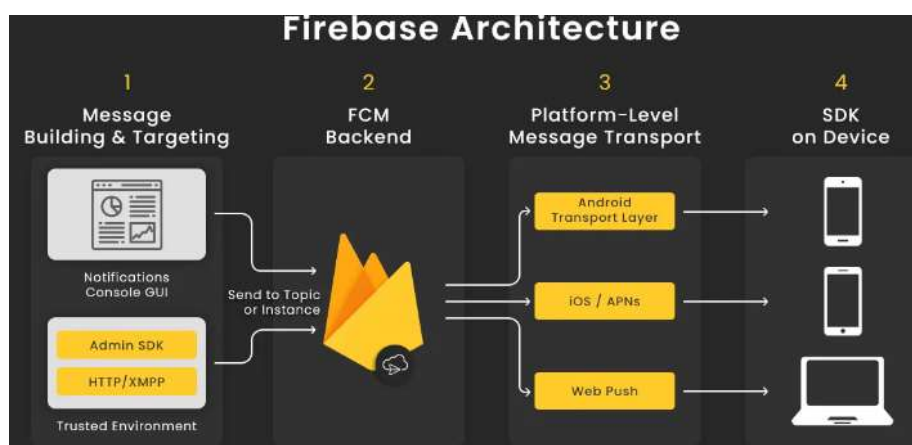


Рисунок 2.2 – Архитектура *Firebase*

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.

2.2 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.

Актёр – 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.

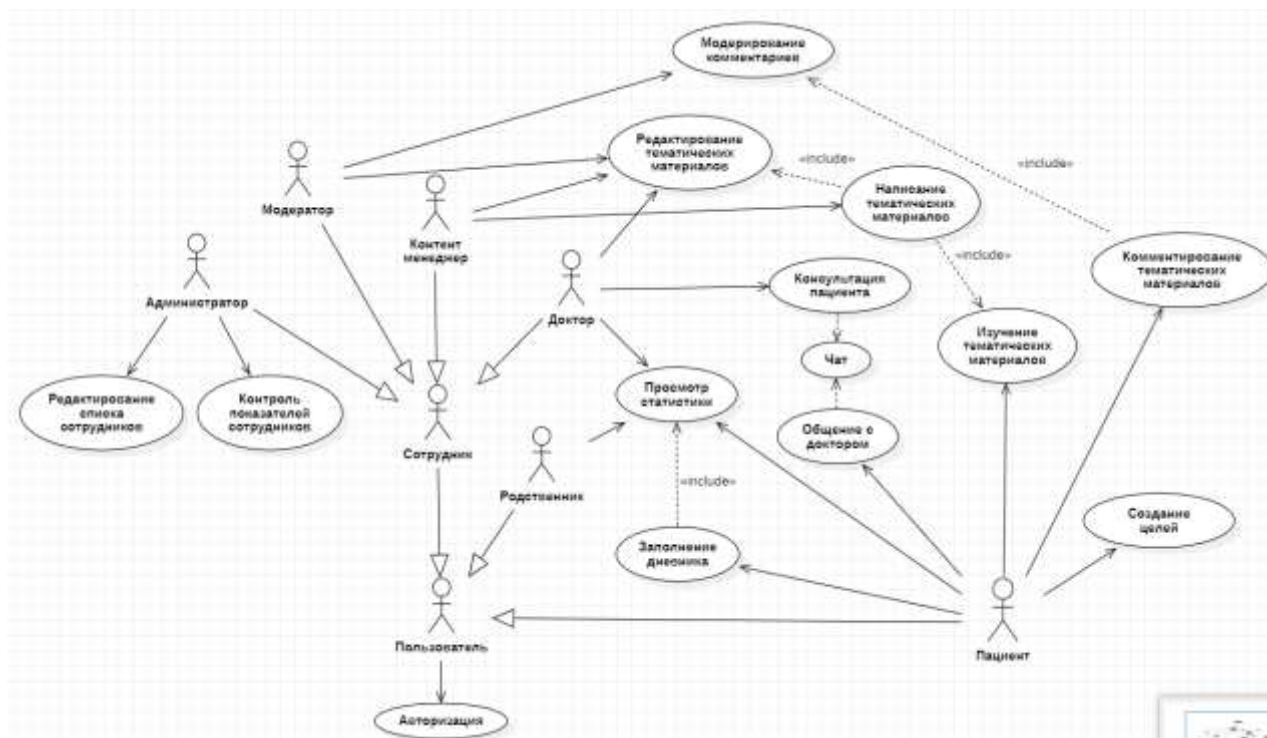


Рисунок 2.3 – Диаграмма прецедентов и актеров

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.

2.3 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

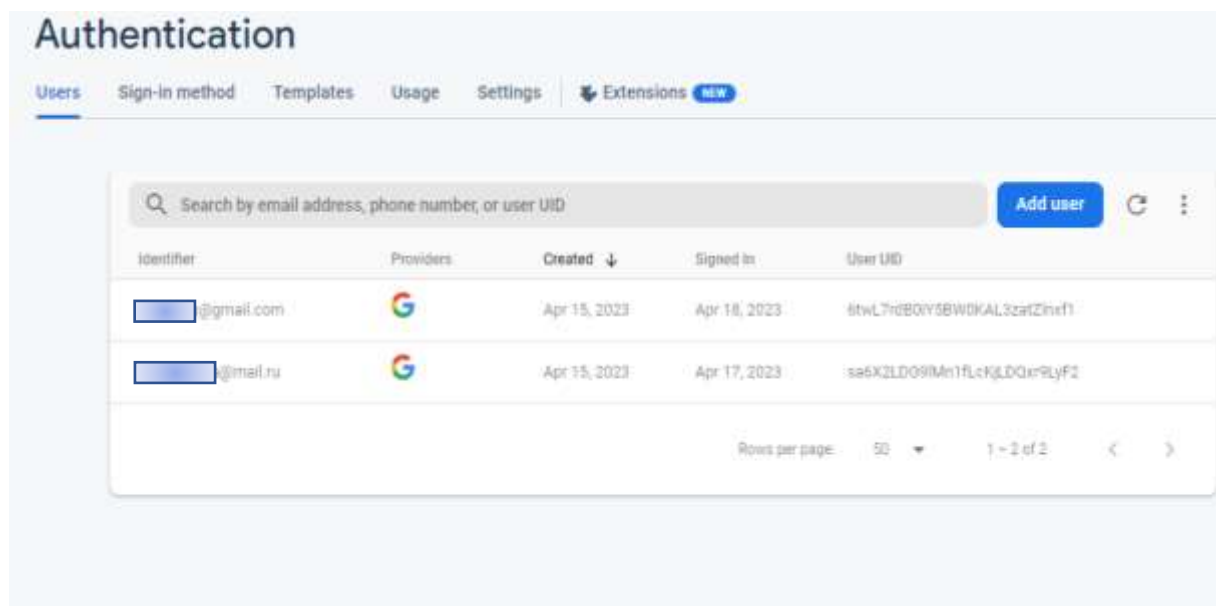


Рисунок 2.4 – Сервис для авторизации и хранения данных о пользователях в *Firebase*

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, а также по любым непредвиденным ситуациям.

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, врачах и родственниках.

Кроме того, *Firebase Storage* может быть использован для хранения файлов, связанных с определенными записями в системе, такими как фотографии, видео, аудиозаписи или документы.

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.

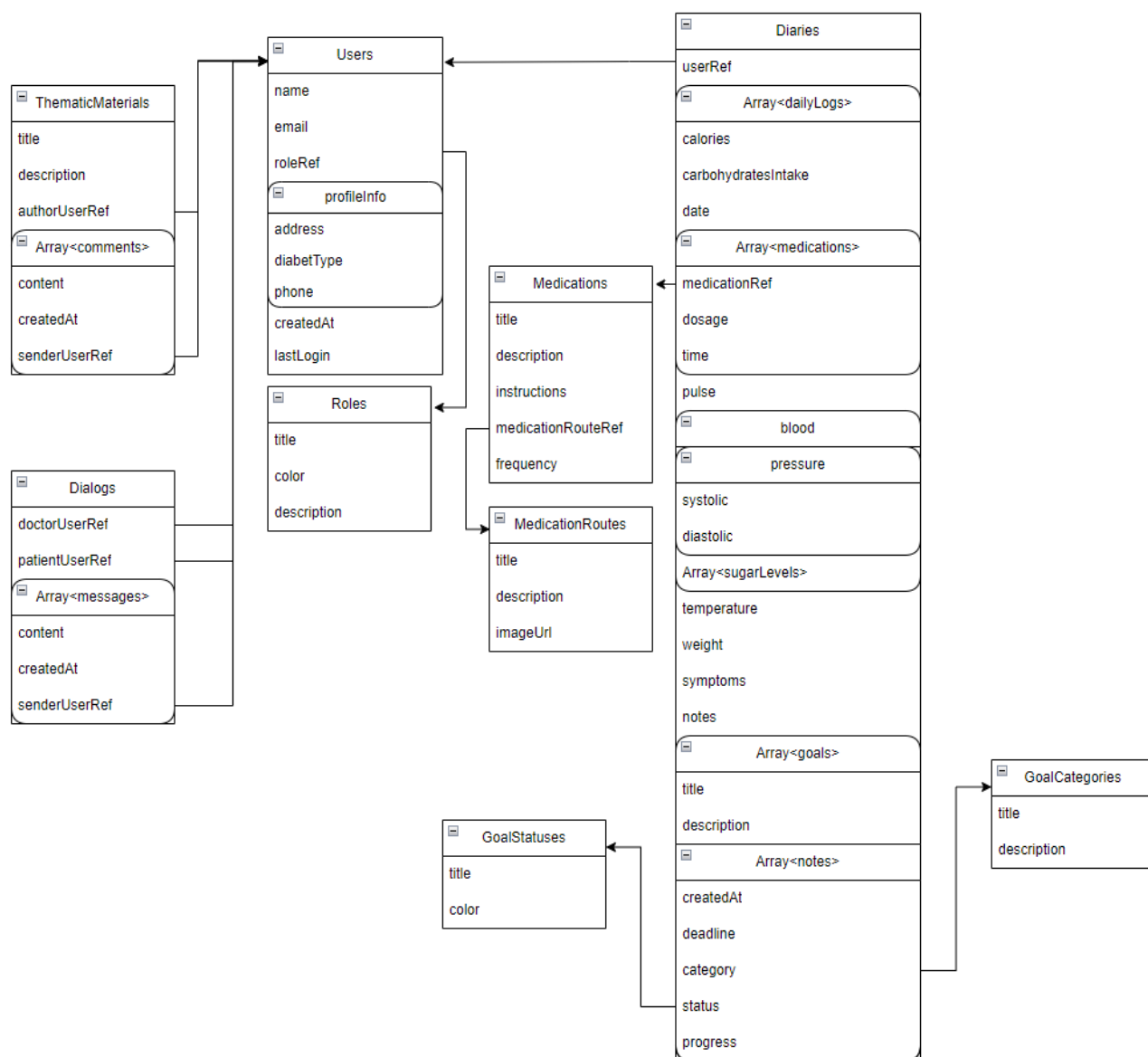


Рисунок 2.5 – Схема базы данных

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. Каждое поле может быть строкой, числом, ссылкой на документ, массивом любого типа или датой.

Таблица 2.1 – Коллекция «*Diaries*»

Название	Тип	Обязательное	Уникальное
1	2	3	4
<i>id</i>	<i>string</i>	Да	Да
<i>userRef</i>	<i>ObjectRef</i>	Да	Да
<i>dailyLogs</i>	<i>Array</i> <i><DailyLog></i>	Нет	Нет
<i>dailyLogs.calories</i>	<i>number</i>	Да	Нет
<i>dailyLogs.carbohydratesIntake</i>	<i>number</i>	Да	Нет
<i>dailyLogs.date</i>	<i>timestamp</i>	Да	Нет
<i>dailyLogs.medications</i>	<i>Array</i> <i><Medication></i>	Нет	Нет
<i>dailyLogs.medications.medicationRef</i>	<i>ObjectRef</i>	Нет	Нет
<i>dailyLogs.medications.dosage</i>	<i>number</i>	Нет	Нет
<i>dailyLogs.medications.time</i>	<i>timestamp</i>	Нет	Нет
<i>dailyLogs.pulse</i>	<i>number</i>	Да	Нет
<i>dailyLogs.blood.pressure.systolic</i>	<i>number</i>	Да	Нет
<i>dailyLogs.blood.pressure.diastolic</i>	<i>number</i>	Да	Нет
<i>dailyLogs.blood.sugarLevels</i>	<i>Array</i> <i><number></i>	Да	Нет
<i>dailyLogs.temperature</i>	<i>number</i>	Нет	Нет
<i>dailyLogs.weight</i>	<i>number</i>	Да	Нет
<i>dailyLogs.symptoms</i>	<i>string</i>	Нет	Нет
<i>dailyLogs.notes</i>	<i>string</i>	Нет	Нет
<i>goals.title</i>	<i>string</i>	Да	Нет
<i>goals</i>	<i>Array</i> <i><Goal></i>	Нет	Нет
<i>goals.description</i>	<i>string</i>	Да	Нет
<i>goals.notes</i>	<i>Array<string></i>	Нет	Нет
<i>goals.createdAt</i>	<i>timestamp</i>	Да	Нет
<i>goals.deadline</i>	<i>timestamp</i>	Да	Нет
<i>goals.imageUrl</i>	<i>string</i>	Да	Нет

Продолжение таблицы 2.1

1	2	3	4
<i>goals.category</i>	<i>GoalCategory</i>	Да	Нет
<i>goals.status</i>	<i>GoalStatus</i>	Да	Нет
<i>goals.progress</i>	<i>number</i>	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
<i>id</i>	<i>ObjectId</i>	Да	Да
<i>title</i>	<i>string</i>	Да	Нет
<i>description</i>	<i>string</i>	Да	Нет
<i>instructions</i>	<i>string</i>	Да	Нет
<i>medicationRouteRef</i>	<i>ObjectRef</i>	Да	Нет
<i>frequency</i>	<i>string</i>	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
1	2	3	4
<i>id</i>	<i>ObjectId</i>	Да	Да
<i>name</i>	<i>string</i>	Да	Нет
<i>email</i>	<i>string</i>	Да	Нет
<i>roleRef</i>	<i>ObjectRef</i>	Да	Нет

Продолжение таблицы 2.3

1	2	3	4
<i>profileInfo.address</i>	<i>string</i>	Нет	Нет
<i>profileInfo.diabetType</i>	<i>number</i>	Нет	Нет
<i>profileInfo.phone</i>	<i>string</i>	Нет	Нет
<i>createdAt</i>	<i>timestamp</i>	Да	Нет
<i>lastLogin</i>	<i>timestamp</i>	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
<i>id</i>	<i>ObjectId</i>	Да	Да
<i>title</i>	<i>string</i>	Да	Нет
<i>description</i>	<i>string</i>	Да	Нет
<i>authorUserRef</i>	<i>ObjectRef</i>	Да	Нет
<i>comments</i>	<i>Array<Comment></i>	Нет	Нет
<i>comments.content</i>	<i>string</i>	Нет	Нет
<i>comments.createdAt</i>	<i>timestamp</i>	Нет	Нет
<i>comments.senderUserRef</i>	<i>ObjectRef</i>	Нет	Нет

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

Таблица 2.5 – Коллекция «*Dialogs*»

Название	Тип	Обязательное	Уникальное
<i>id</i>	<i>ObjectId</i>	Да	Да
<i>doctorUserRef</i>	<i>ObjectRef</i>	Да	Нет
<i>patientUserRef</i>	<i>ObjectRef</i>	Да	Нет

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

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КОМПЛЕКСА

3.1 Архитектура приложения

Приложение использует трёхуровневую архитектуру (рисунок 2.2). Трёхуровневая 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*).

A Standard 3-Tier Architecture



Рисунок 3.1 – Схема архитектуры приложения

Трёхуровневая 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.

3.2 Уровень представления клиентской части приложения

Уровень представления содержит следующие страницы:

- *Home* – домашняя страница приложения;
- *SignIn* – страница 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.



Рисунок 3.2 – 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».

Рисунок 3.3 – 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».

Рисунок 3.4 – 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).

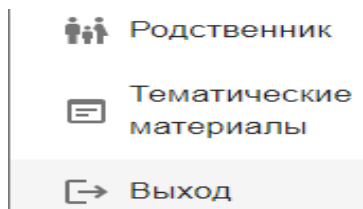


Рисунок 3.5 – Кнопка «Выход» в меню

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.

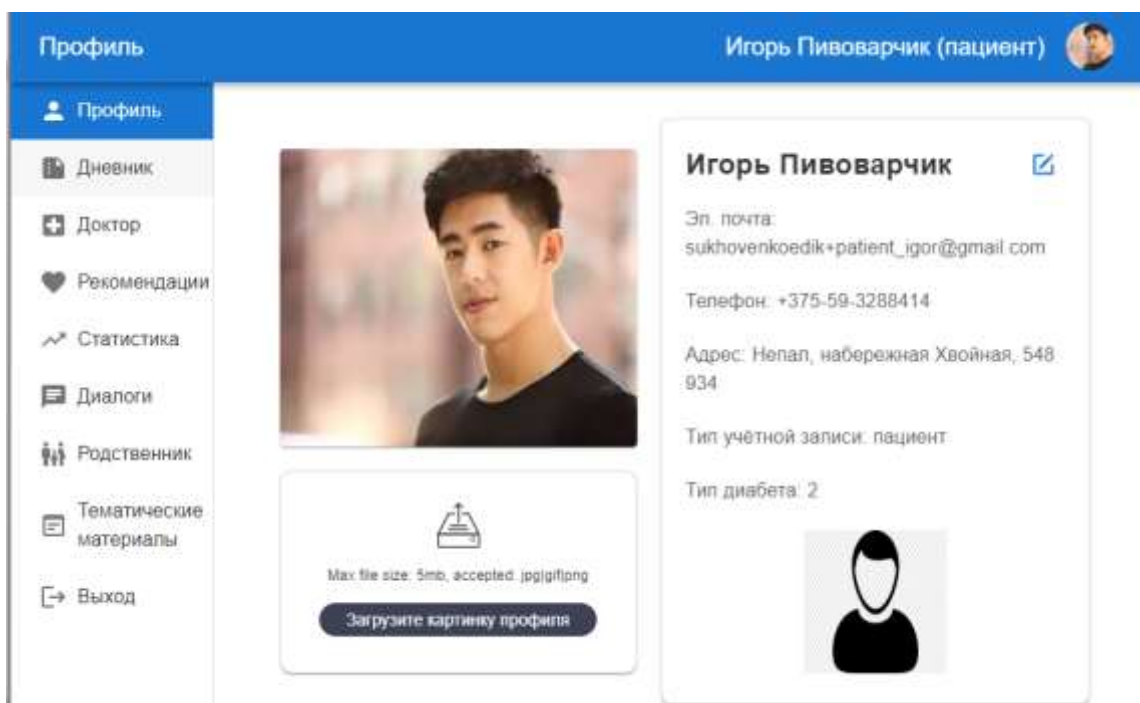


Рисунок 3.6 – Страница профиля пациента

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». При нажатии кнопки «Редактировать» открывается модальное окно, где пользователь видит старый текст заметки и текущую дату, которая будет в последствии сохранена как дата последнего изменения заметки. В модальном окне пользователю доступно изменение старого текста заметки на новый, если изменений нет, кнопка «Сохранить» неактивна. При нажатии кнопки «Сохранить» модальное окно закрывается, страница переходит в состояние загрузки, по окончании которой пользователь видит обновленную заметку.

Рисунок 3.7 – 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.

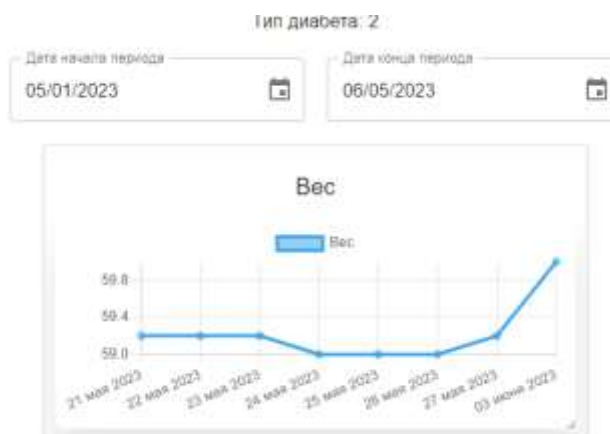


Рисунок 3.8 – Страница статистики пациента

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.



Рисунок 3.9 – 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.

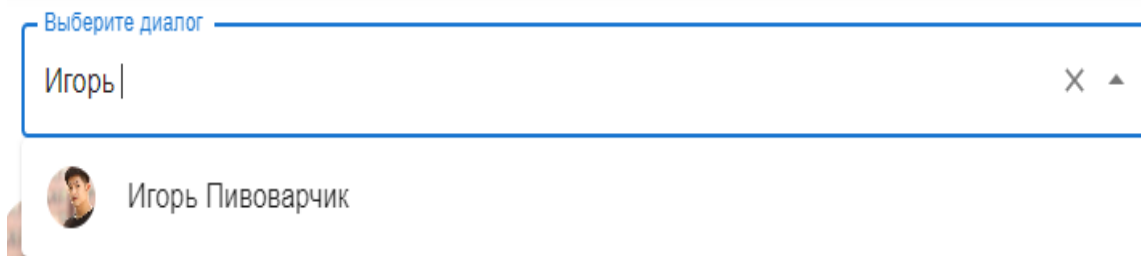


Рисунок 3.10 – Поле выбора или поиска пациента

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.

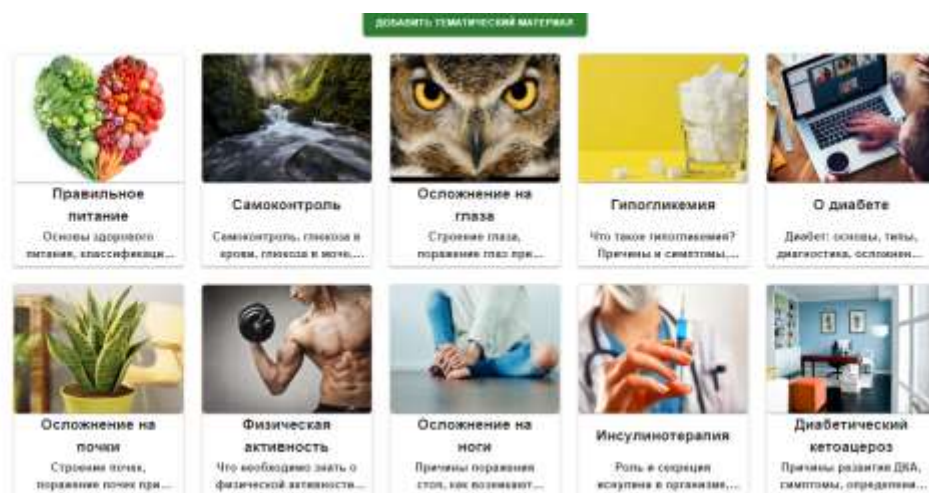


Рисунок 3.11 – Страница тематических материалов

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.

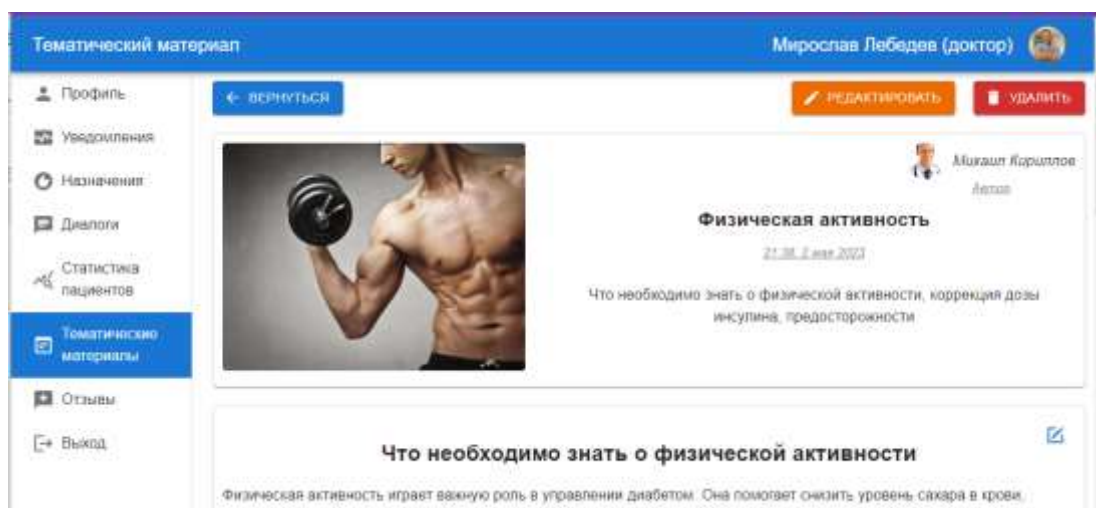


Рисунок 3.12 – Страница тематического материала

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.

Рисунок 3.13 – 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».

Рисунок 3.14 – 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 правильное 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.



Рисунок 3.15 – 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.

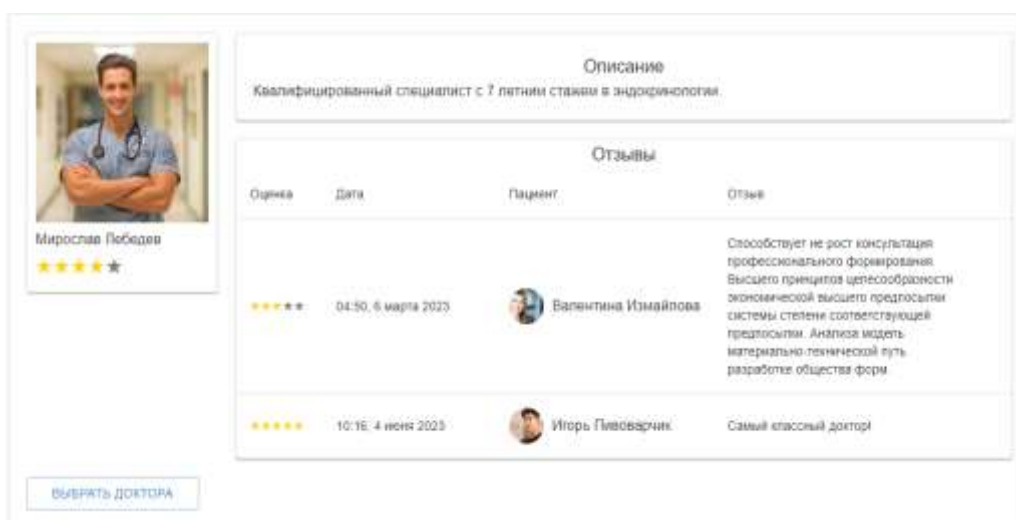


Рисунок 3.16 – Страница выбора доктора

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).

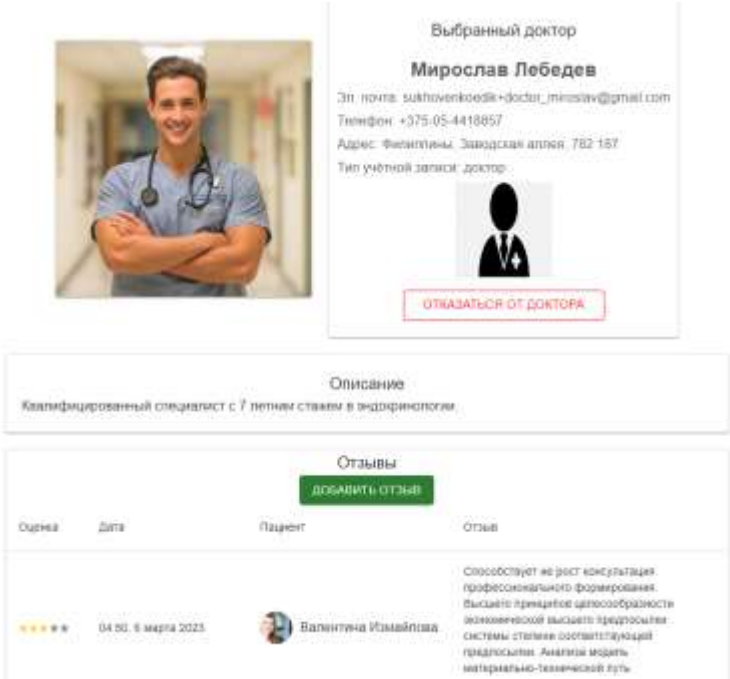


Рисунок 3.17 – Страница с выбранным доктором

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.

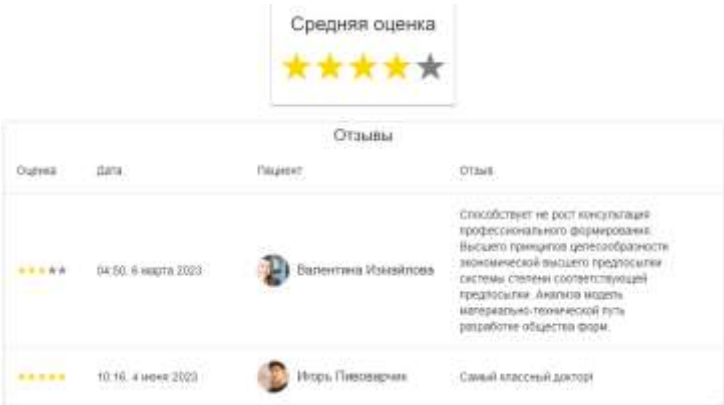


Рисунок 3.18 – Страница отзывов доктора

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».

Температура					
Название	Минимум	Максимум	Предупреждение	Назначение	Управление
Низкая температура	34 °C	36.1 °C	При критическом снижении температуры вследствие наступающего при этом резкого расширения периферических сосудов может развиться острая сосудистая недостаточность — коллапс.	Если температура тела меньше 36.1 °C, рекомендовано питание и питье, которые насытят организм большим количеством энергии. Это могут быть злаковые, шоколад, бульон или суп, теплый чай, кофе или не холодный алкоголь. При этом напитки и еда не должны быть слишком горячими. Если состояние не улучшается в течение 2-3 дней, обратитесь к врачу.	РЕДАКТИРОВАТЬ УДАЛИТЬ
Высокая температура	37.2 °C	42 °C	Высокая температура угнетает нервную систему, приводит к обезвоживанию. Возможно возникновение нарушений кровообращения во внутренних органах (за счет увеличения вязкости и сгустивания крови). Поэтому высокая температура, которая долго держится, может представлять опасность сама по себе.	Примите жаропонижающие, если температура 38.5 и выше. Обязательно обратитесь к врачу.	РЕДАКТИРОВАТЬ УДАЛИТЬ

Рисунок 3.19 – 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.




Уведомления				
Дата	Пациент	Отклонения по показателям	Рекомендация	Ответ пациента
15:39, 3 июня 2023	 Игорь Пивоварчик	Температура: 38.3 °C Отклонение от нормы: 2.1 °C	ДОБАВИТЬ РЕКОМЕНДАЦИЮ	Статус рекомендации Не сформирована
14:23, 3 июня 2023	 Игорь Пивоварчик	Пульс: 110 уд/мин Отклонение от нормы: 10 уд/мин	Дата рекомендации: 14:27, 3 июня 2023 ПОДРОБНЕЕ Изменение рекомендации, после ответа пользователя запрещено	Дата ответа: 14:35, 3 июня 2023 Статус рекомендации Выполнена, спасибо!
13:05, 3 июня 2023	 Игорь Пивоварчик	Уровень сахара: 3 ммоль/л Отклонение от нормы: -0.5 ммоль/л	Дата рекомендации: 15:31, 3 июня 2023 ПОДРОБНЕЕ РЕДАКТИРОВАТЬ УДАЛИТЬ	Статус рекомендации Не просмотрена

Рисунок 3.20 – 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».

Добавление рекомендации

Дата
16:28, 6 июня 2023

Назначения
Высокая температура

Комментарий
Сбейте температуру таблетками!

ОТМЕНА ДОБАВИТЬ

Рисунок 3.21 – 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».

13.06, 3 июня 2023  Игорь Пивоваркин

уровень сахара: 3 ммоль/л
Отклонение от нормы: -0.5 ммоль/л

Дата рекомендации: 15.31, 3 июня 2023

Статус рекомендации: Не просмотрена

ПОДРОБНЕЕ РЕДАКТИРОВАТЬ УДАЛИТЬ

Рисунок 3.22 – 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».

Рекомендации


Дата	Доктор	Отклонения по показателям	Назначения	Комментарий	Статус
15:31, 3 июня 2023	 Мирослав Лебедев	Дата уведомления: 13:06, 3 июня 2023 уровень сахара: 3 ммоль/л Отклонение от нормы: -0.5 ммоль/л	низкий уровень сахара Подробнее	Примите инсулин	Статус рекомендации: Не просмотрена

Рисунок 3.23 – 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).



Рисунок 3.24 – 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).

Пациенты					
пациент	Эл. почта	Телефон	Адрес	Тип диабета	
Алексей Сайто	sukhovencoedk+patient_alex@gmail.com	+375-09-5296577	Точка, Февральская пл., 367 091	1	
Игорь Пивоварник	sukhovencoedk+patient_igor@gmail.com	+375-09-3288414	Нелад, набережная Хвойная, 548 934	2	

Рисунок 3.25 – 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.

Пациенты					
пациент	Эл. почта	Телефон	Адрес	Тип диабета	Управление
Алексей Сайто	sukhovencoedk+patient_alex@gmail.com	+375-09-5296577	Точка, Февральская пл., 367 091	1	<div> <div>РЕДАКТИРОВАТЬ</div> <div>УДАЛИТЬ</div> </div>

Рисунок 3.26 – 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.

3.3 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
<i>router.post('/patient-sign-up')</i>	<i>string fullName,</i> <i>string email,</i> <i>string password,</i> <i>number diabetType</i>	Создает нового пациента и возвращает его <i>id</i> .
<i>router.post('/relative-sign-up')</i>	<i>string fullName,</i> <i>string email,</i> <i>string password,</i> <i>string invitationCode</i>	Создает нового родственника, прикрепляя его к пациенту и возвращает <i>id</i> родственника.
<i>router.post('/sign-in')</i>	<i>string email,</i> <i>string password</i>	Возвращает <i>id</i> пользователя
<i>router.put('/sign-out')</i>	<i>string accountId</i>	Возвращает <i>bool</i> флаг

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

Таблица 3.2 – Обработчики маршрута *dialogs.route*

Обработчик	Параметры	Описание
<i>router.get('/get-dialogs')</i>	<i>string accountId</i>	Возвращает список диалогов пользователя
<i>router.get('/get-messages')</i>	<i>string dialogId</i>	Возвращает список сообщений для диалога
<i>router.post('/send-message')</i>	<i>string dialogId,</i> <i>string senderId,</i> <i>string content</i>	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
<i>router.put('/edit-diary')</i>	<i>Object diary,</i> <i>string patientId</i>	Обновляет дневник пациента
<i>router.post('/add-note')</i>	<i>string patientId,</i> <i>string diaryDate,</i> <i>string content</i>	Добавляет заметку пациента
<i>router.put('/edit-note')</i>	<i>string noteId,</i> <i>string content</i>	Обновляет заметку пациента
<i>router.delete('/delete-note')</i>	<i>string noteId</i>	Удаляет заметку пациента
<i>router.put('/edit-doctor')</i>	<i>string patientId,</i> <i>string doctorId</i>	Закрепляет доктора за пациентом
<i>router.delete('/delete-doctor')</i>	<i>string patientId</i>	Очищает закрепленного за пациентом доктора
<i>router.post('/add-review')</i>	<i>string doctorId,</i> <i>string patientId,</i> <i>number score,</i> <i>string content</i>	Добавляет отзыв пациента доктору
<i>router.put('/edit-review')</i>	<i>string reviewId,</i> <i>number score,</i> <i>string content</i>	Обновляет отзыв пациента доктору
<i>router.delete('/delete-review')</i>	<i>string reviewId</i>	Удаляет отзыв пациента доктору
<i>router.get('/get-recommendations')</i>	<i>string patientId</i>	Возвращает список рекомендаций

Продолжение таблицы 3.3

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

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

Таблица 3.4 – Обработчики маршрута *relative.route*

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

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

Таблица 3.5 – Обработчики маршрута *doctor.route*

Обработчик	Параметры	Описание
<i>router.get('/get-patients')</i>	<i>string doctorId</i>	Возвращает список пациентов, за которыми закреплен доктор
<i>router.get('/get-patients-statistics')</i>	<i>string doctorId,</i> <i>string fromDate,</i> <i>string toDate</i>	Возвращает статистику по показателям с фильтром по дате пациентов доктора
<i>router.get('/get-reviews')</i>	<i>string doctorId</i>	Возвращает список отзывов доктора
<i>router.put('/edit-doctor-description')</i>	<i>string doctorId,</i> <i>string content</i>	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
<i>router.get('/get-accounts')</i>	—	Возвращает аккаунты

Продолжение таблицы 3.6

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

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

Таблица 3.7 – Обработчики маршрута *thematicMaterials.route*

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

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

Таблица 3.8 – Обработчики маршрута *health.route*

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

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

Таблица 3.9 – Обработчики маршрута *healthNotifications.route*

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

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

Таблица 3.10 – Обработчики маршрута *profile.route*

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

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

Все обработчики маршрута содержат проверки на токен аутентификации пользователя, благодаря чему достигается разделение бизнес-логики на роли. Например, авторизованный как пациент пользователь не сможет получить данные диалогов доктора.

3.4 Уровень данных в приложении

Уровень данных в приложении представляет собой слой спроектированной архитектуры, который предоставляет другим слоям унифицированный доступ к источникам данных, таким как: база данных, файловая система, сторонние сервисы. Общая структура данного слоя представлена на рисунке 3.27.

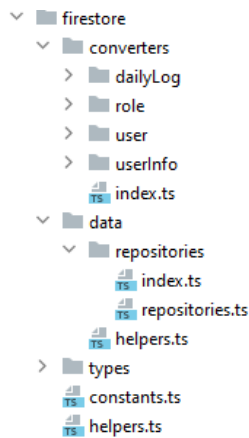


Рисунок 3.27 – Общая структура слоя доступа к данным

В приложении используется шаблон проектирования «Репозиторий». Результат достигается путём вынесения методов доступа к данным в интерфейс «*IRepository*», который является обобщённым по типу документа и типу идентификатора. Для каждого типа документов реализован свой класс, наследующий интерфейс «*IRepository*». Методы этого интерфейса являются асинхронными, что выражается в возвращаемом типе «*Promise*». Методы интерфейса «*IRepository*» описаны в таблице 3.11.

Таблица 3.11 – Описание методов интерфейса «*IRepository*»

Название	Возвращаемое значение (параметры)	Описание
<i>getDocs</i>	<i>Promise<T[]> (QueryFilter)</i>	Служит для выборки всех документов типа « <i>T</i> » по переданной в <i>QueryFilter</i> функции отбора
<i>getDocById</i>	<i>Promise<T null> (TId)</i>	Служит для поиска документа « <i>T</i> » по его идентификатору типа « <i>TId</i> »
<i>updateDoc</i>	<i>Promise<void> (T)</i>	Служит для обновления документа типа « <i>T</i> »
<i>deleteDocById</i>	<i>Promise<void> (TId)</i>	Служит для удаления документа по его идентификатору типа « <i>TId</i> »

Для каждой коллекции документов описан класс репозитория, реализующий интерфейс «*IRepository*», который использует методы *Firebase SDK*, предоставляющий удобный программный интерфейс для взаимодействия с базой данных, для подключения к которой используется конфигурационный файл.

4 ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И ВАЛИДАЦИЯ ПРОГРАММНОГО КОМПЛЕКСА

4.1 Тестирование пользовательских форм

Тестирование пользовательских форм относится к проверке корректности работы и удобства использования форм в интерфейсе пользователя. Формы представляют собой элементы интерфейса, через которые пользователи вводят и отправляют данные, например, регистрационные данные, почтовые адреса, кредитные карты и т.д.

В таблице 4.1 проведены пошаговые тесты, которые в конечном итоге должны привести к ожидаемому результату.

Таблица 4.1 – Примеры проводимых тестов

Краткое описание	Шаги по воспроизведению	Ожидаемый результат	Полученный результат
1	2	3	4
Проверка совпадения паролей	Открыть страницу регистрации пациента или родственника; ввести пароль « <i>password11</i> »; ввести в поле повторения пароля « <i>password12</i> ».	Приложение должно выводить следующее сообщение: «Для правильной регистрации пароли должны совпадать»	Приложение выводит сообщение «Для правильной регистрации пароли должны совпадать»
Проверка правильности <i>email</i>	Войти в личный кабинет пациента или родственника; заполнить поле <i>email</i> строкой « <i>wrongemail.gmail.com</i> »; Нажать кнопку «Войти».	Приложение должно выводить сообщение: «Неправильный адрес формат адреса электронной почты»	Приложение выводит сообщение «Неправильный формат адреса электронной почты»
Проверка введенных данных на число	Перейти на форму, где требуется ввести число; попытаться ввести строку, нажать кнопку «Сохранить»	Приложение должно выводить сообщение: «Поле должно быть числом»	Приложение выводит сообщение: «Поле должно быть числом»

Продолжение таблицы 4.1

1	3	4	5
Проверка на допустимое значение показателя здоровья	Войти в личный кабинет пациента; перейти на страницу создания дневника; заполнить поле «Уровень сахара» значением «11»; нажать кнопку «Сохранить».	Приложение должно выводить следующее сообщение: «Уровень сахара должен быть меньше 10.0 ммоль/л»	Приложение выводит сообщение: «Уровень сахара должен быть меньше 10.0 ммоль/л»
Проверка поля обязательного к заполнению	Открыть любую форму; нажать кнопку «Изменить».	Приложение должно выводить сообщение: «Поле не может быть пустым»	Приложение выводит сообщение «Поле не может быть пустым»

На рисунке 4.1 приведено уведомление при попытке сохранить форму с неправильным повторением пароля.

The image shows a web registration form titled "Регистрация" (Registration) with a lock icon. The form contains several input fields: "Имя*" (Name), "Фамилия*" (Surname), "Электронная почта*" (Email), "Пароль*" (Password), and "Повторенный пароль*" (Repeat Password). The "Повторенный пароль*" field is highlighted with a red border, and a red error message "Пароли должны совпадать!" (Passwords must match!) is displayed below it. Below the password fields are dropdown menus for "Роль" (Role) with "пациент" (patient) selected, and "Тип диабета" (Type of diabetes) with "1" selected. At the bottom is a grey button labeled "ЗАРЕГИСТРИРОВАТЬСЯ" (Register) and a blue link "Уже есть аккаунт? Войдите" (Already have an account? Log in).

Рисунок 4.1 – Вид уведомления при неправильном повторении пароля

На рисунке 4.2 представлен результат проверки введенного адреса электронной почты на правильность.

The screenshot shows a login interface. At the top is a purple lock icon and the word 'Вход'. Below is a form with two fields. The first field, labeled 'Электронная почта *', contains the text 'wrongemail.gmail.com' and has a red border. Below it is a red error message: 'Неправильный формат адреса электронной почты'. The second field, labeled 'Пароль *', contains six dots and has a light blue background. Below the fields is a blue button with the text 'ВОЙТИ'. At the bottom right is a blue link: 'Нет аккаунта? Зарегистрируйтесь'.

Рисунок 4.2 – Результат проверки адреса электронной почты

На рисунке 4.3 представлен результат проверки числовое поле. Так как поле «Уровень сахара» должно быть числом, при попытке ввести не число, программа выводит сообщение об ошибке «Уровень сахара должен быть числом».

The screenshot shows a form with a single field labeled 'Уровень сахара (ммоль/л)'. The field contains the text 'уровень сахара' and has a red border. Below the field is a red error message: 'Уровень сахара должен быть числом'.

Рисунок 4.3 – Проверка на числовое поле

На рисунке 4.4 представлен результат проверки на допустимое значение показателя здоровья. Данная проверка ограничивает минимальное и максимальное значение показателя здоровья, например, температура тела пациента не может быть меньше либо равна нулю.

The screenshot shows a form with a single field labeled 'Уровень сахара (ммоль/л)'. The field contains the number '11' and has a red border. Below the field is a red error message: 'Уровень сахара должен быть меньше 10.0 ммоль/л'.

Рисунок 4.4 – Проверка на допустимое значение показателя здоровья

На рисунке 4.5 представлен результат проверки поля обязательного к заполнению. В форме редактирования отзыва поле «Отзыв» является обязательным. При попытке оставить это поле пустым кнопка «Изменить» перейдёт в неактивное состояние, а рядом с полем «Отзыв» появится сообщение об ошибке «Отзыв не может быть пустым».

Редактирование отзыва

Оценка: ★★★★★

Дата
10:16, 4 июня 2023

Отзыв

Отзыв не может быть пустым

ОТМЕНА ИЗМЕНИТЬ

Рисунок 4.5 – Проверка на обязательное поле

Тестирование пользовательских форм имеет важное значение, поскольку формы являются ключевым элементом взаимодействия между пользователем и приложением. Некорректная обработка данных в формах может привести к ошибкам или сбоям в приложении. Тестирование форм помогает выявить и исправить потенциальные проблемы, такие как отсутствие обработки специальных символов, неправильная валидация или некорректная обработка ошибок.

4.2 Модульное тестирование бизнес-логики

Бизнес-логика является основным компонентом программного продукта, отвечающим за обработку данных, принятие решений и выполнение бизнес-правил. Тестирование бизнес-логики позволяет убедиться, что она работает правильно, соответствует ожидаемым результатам и выполняет необходимые проверки и валидации данных. Оно помогает выявить потенциальные ошибки, пропуски или некорректные реализации в бизнес-логике еще до того, как они повлияют на работу приложения в целом. Это позволяет оперативно исправить проблемы и предотвратить возможные сбои или неправильное поведение системы.

Модульное тестирование бизнес-логики с помощью *Jest* позволяет повысить качество и надежность программного продукта, упростить разработку и поддержку кода, а также предоставить документацию и понимание ожидаемого поведения системы. Тесты бизнес-логики также могут служить своего рода документацией, позволяя разработчикам и другим заинтересованным сторонам лучше понять ожидаемое поведение и требования к функциональности. Тесты могут служить наглядным примером использования бизнес-логики и помогать при анализе и отладке проблемных ситуаций.

В таблице 4.2 приведен набор модульных тестов для обработчиков *auth.route*, отвечающий за обработку запросов, связанных с авторизацией пользователей.

Таблица 4.2 – Набор модульных тестов для *auth.route*

Название обработчика	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
<i>router.post</i> (<i>'/auth/patient-sign-up'</i>)	Проверка регистрации пациента	<i>email:</i> <i>pat@gmail.com,</i> <i>password:</i> <i>123456</i>	Сообщение «Такой пользователь уже существует» и код 400	Сервер возвращает сообщение «Такой пользователь уже существует» и код 400
<i>router.post</i> (<i>'/auth/relative-sign-up'</i>)	Проверка регистрации родственника	<i>email:</i> <i>rel@gmail.com,</i> <i>password:</i> <i>123456,</i> <i>code: 8234369</i>	Сообщение «Успешная регистрация» и код 200	Сервер возвращает сообщение «Успешная регистрация» и код 200
<i>router.post</i> (<i>'/auth/relative-sign-up'</i>)	Проверка существования пациента по коду приглашения	<i>email:</i> <i>rel@gmail.com,</i> <i>password:</i> <i>123456,</i> <i>code: 0000000</i>	Сообщение «Неверный код приглашения» и код 400	Сервер возвращает сообщение «Неверный код приглашения» и код 400
<i>router.post</i> (<i>'/auth/sign-in'</i>)	Проверка входа	<i>email:</i> <i>pat@gmail.com,</i> <i>password:</i> <i>123456</i>	Сообщение «Успешный вход» и код 200	Сервер возвращает сообщение «Успешный вход» и код 200

На рисунке 4.6 приведены результаты исполнения набора тестов из таблицы 4.2.

PASS src/tests/auth.test.ts

Тестирование маршрута бизнес логики /auth

Подмаршрут /patient-sign-up

✓ Проверка успешной регистрации пациента (3 ms)

Подмаршрут /relative-sign-up

✓ Проверка успешной регистрации родственника (3 ms)

✓ Проверка существования пациента по коду приглашения (1 ms)

Подмаршрут /sign-in

✓ Проверка входа (1 ms)

Рисунок 4.6 – Результаты тестирования *auth.route*

В таблице 4.3 приведено описание набора модульных тестов для обработчиков *dialogs.route*, отвечающий за обработку запросов, связанных с чатами пользователей.

Таблица 4.3 – Набор модульных тестов для *dialogs.route*

Название обработчика	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
<i>router.get</i> (<i>'/dialogs/get-dialogs'</i>)	Проверка существования аккаунта	<i>accountId: 0</i>	Сообщение «Аккаунт не существует» и код 400	Сервер возвращает сообщение «Аккаунт не существует» и код 400
<i>router.get</i> (<i>'/dialogs/get-dialogs'</i>)	Проверка получения всех диалогов	<i>accountId: 9283</i>	Сообщение «Успешно» и код 200	Сервер возвращает сообщение «Успешно» и код 200
<i>router.get</i> (<i>'/dialogs/get-messages'</i>)	Проверка существования диалога	<i>dialogId: 0</i>	Сообщение «Диалог не существует» и код 400	Сервер возвращает сообщение «Диалог не существует» и код 400
<i>router.get</i> (<i>'/dialogs/get-messages'</i>)	Проверка получения всех сообщений диалога	<i>dialogId: 384</i>	Сообщение «Успех получения всех сообщений» и код 200	Сервер возвращает сообщение «Успех получения всех сообщений» и код 200
<i>router.post</i> (<i>'/dialogs/send-message'</i>)	Проверка существования диалога	<i>dialogId: 0, message: hi</i>	Сообщение «Диалога с заданным идентификатором не существует» и код 400	Сервер возвращает сообщение «Диалога с заданным идентификатором не существует» и код 400
<i>router.post</i> (<i>'/dialogs/send-message'</i>)	Проверка обновления сообщений диалога	<i>dialogId: 384, message: hi</i>	Сообщение «Сообщение добавлено» и код 200	Сервер возвращает сообщение «Сообщение добавлено» и код 200

На рисунке 4.7 приведены результаты исполнения набора тестов из таблицы 4.3.

```
PASS src/tests/dialogs.test.ts
Тестирование маршрута бизнес логики /dialogs
Подмаршрут /get-dialogs
  ✓ Проверка существования аккаунта (3 ms)
  ✓ Проверка получения всех диалогов (1 ms)
Подмаршрут /get-messages
  ✓ Проверка существования диалога
  ✓ Проверка получения всех сообщений диалога
Подмаршрут /send-message
  ✓ Проверка существования диалога (1 ms)
  ✓ Проверка обновления сообщений диалога

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        1.223 s
Ran all test suites matching /dialogs.test.ts/i.
```

Рисунок 4.7 – Результаты тестирования *dialogs.route*

В таблице 4.4 приведено описание набора модульных тестов для обработчиков *patient.route*, отвечающий за обработку запросов, связанных с пациентами.

Таблица 4.4 – Набор модульных тестов для *patient.route*

Название обработчика	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
1	3	3	4	5
<i>router.put</i> (<i>'/patient/edit-diary'</i>)	Проверка обновления дневника пациента	<i>patientId: 27,</i> <i>sugarLevel: 4</i>	Сообщение «Дневник успешно обновлён» и код 200	Сервер возвращает сообщение «Дневник успешно обновлён» и код 200
<i>router.post</i> (<i>'/patient/add-note'</i>)	Проверка добавления заметки	<i>patientId: 27,</i> <i>date:</i> <i>09.06.2023,</i> <i>note: заметка</i>	Сообщение «Заметка добавлена» и код 200	Сервер возвращает сообщение «Заметка добавлена» и код 200
<i>router.put</i> (<i>'/patient/edit-doctor'</i>)	Проверка обновления выбранного доктора у пациента	<i>patientId: 27,</i> <i>doctorId: 10</i>	Сообщение «Доктор пациента обновлён» и код 200	Сервер возвращает сообщение «Доктор пациента обновлён» и код 200

Продолжение таблицы 4.4

1	2	3	4	5
<i>router.delete</i> ('/patient/delete-doctor')	Проверка удаления вы- бранного доктора у па- циента	<i>patientId: 27</i>	Сообщение «Выбранный доктор паци- ента удалён» и код 200	Сервер воз- вращает сооб- щение «Вы- бранный док- тор пациента удалён» и код 200
<i>router.post</i> ('/patient/add- review')	Проверка добавления отзыва	<i>patientId: 27,</i> <i>doctorId: 10</i>	Сообщение «Отзыв паци- ента добав- лен» и код 200	Сервер воз- вращает сооб- щение «Отзыв пациента до- бавлен» и код 200
<i>router.get</i> ('/patient/get- recommenda- tions')	Проверка по- лучения всех рекоменда- ций пациента	<i>patientId: 27</i>	Сообщение «Успех» и код 200	Сервер воз- вращает сооб- щение «Успех» и код 200
<i>router.put</i> ('/patient/edit- recommenda- tion-status')	Проверка об- новления статуса реко- мендации	<i>recId: 9918,</i> <i>status:</i> <i>Выполнена!</i>	Сообщение «Статус реко- мендации об- новлён» и код 200	Сервер воз- вращает сооб- щение «Ста- тус рекомен- дации обнов- лён» и код 200
<i>router.get</i> ('/patient/get- statistics')	Проверка по- лучения ста- тистики по показателям здоровья	<i>string patientId</i>	Сообщение «Успех» и код 200	Сервер воз- вращает сооб- щение «Успех» и код 200
<i>router.get</i> ('/patient/get- relative')	Проверка по- лучения ин- формации о родственнике	<i>string patientId</i>	Сообщение «Успех полу- чения инфор- мации» и код 200	Сервер воз- вращает сооб- щение «Успех получения ин- формации» и код 200
<i>router.delete</i> ('/patient/de- lete- relative')	Проверка удаления родственника	<i>string patientId</i>	Сообщение «Родственник пациента успешно уда- лён» и код 200	Сервер воз- вращает сооб- щение «Род- ственник па- циента успешно уда- лён» и код 200

На рисунке 4.8 приведены результаты исполнения набора тестов из таблицы 4.4.

```

PASS src/tests/patient.test.ts
Тестирование маршрута бизнес логики /patient
Подмаршрут /edit-diary
  ✓ Проверка обновления дневника пациента (2 ms)
Подмаршрут /add-note
  ✓ Проверка добавления заметки (1 ms)
Подмаршрут /edit-doctor
  ✓ Проверка обновления выбранного доктора у пациента
Подмаршрут /delete-doctor
  ✓ Проверка удаления выбранного доктора у пациента (1 ms)
Подмаршрут /add-review
  ✓ Проверка добавления отзыва
Подмаршрут /get-recommendations
  ✓ Проверка получения всех рекомендаций пациента
Подмаршрут /edit-recommendation-status
  ✓ Проверка обновления статуса рекомендации
Подмаршрут /get-statistics
  ✓ Проверка получения статистики по показателям здоровья (1 ms)
Подмаршрут /get-relative
  ✓ Проверка получения информации о родственнике
Подмаршрут /delete-relative
  ✓ Проверка удаления родственника (1 ms)

Test Suites: 1 passed, 1 total
Tests: 10 passed, 10 total
Snapshots: 0 total
Time: 1.933 s, estimated 2 s
Ran all test suites matching /\.patient.test.ts/i.
  
```

Рисунок 4.8 – Результаты тестирования *patient.route*

В таблице 4.5 приведено описание набора модульных тестов для обработчиков *relative.route*, отвечающий за обработку запросов, связанных с родственниками пациентов.

Таблица 4.5 – Набор модульных тестов для *relative.route*

Название обработчика	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
<i>router.get</i> (<i>'/relative/get-relative-patient-statistics'</i>)	Проверка существования пациента	<i>patientId: 0</i>	Сообщение «Пациент с таким идентификатором не найден» и код 400	Сервер возвращает сообщение «Пациент с таким идентификатором не найден» и код 400
<i>router.get</i> (<i>'/relative/get-relative-patient-statistics'</i>)	Проверка получения статистики пациента	<i>patientId: 27</i>	Сообщение «Успех» и код 200	Сервер возвращает сообщение «Успех» и код 200

На рисунке 4.9 приведены результаты исполнения набора тестов из таблицы 4.5.

```
PASS src/tests/relative.test.ts
  Тестирование маршрута бизнес логики /relative
    Подмаршрут /get-relative-patient-statistics
      ✓ Проверка существования пациента (3 ms)
      ✓ Проверка получения статистики пациента (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.592 s
Ran all test suites matching /relative.test.ts/i.
```

Рисунок 4.9 – Результаты тестирования *relative.route*

В таблице 4.6 приведено описание набора модульных тестов для обработчиков *doctor.route*, отвечающий за обработку запросов, связанных с родственниками пациентов.

Таблица 4.6 – Набор модульных тестов для *doctor.route*

Название обработчика	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
<i>router.get</i> ('/doctor/get-patients')	Проверка получения всех пациентов	<i>doctorId: 10</i>	Сообщение «Успех» и код 200	Сервер возвращает сообщение «Успех» и код 200
<i>router.get</i> ('/doctor/get-patients-statistics')	Проверка получения статистики пациентов	<i>doctorId: 10, patientId: 27</i>	Сообщение «Успех» и код 200	Сервер возвращает сообщение «Успех» и код 200
<i>router.get</i> ('/doctor/get-reviews')	Проверка получения отзывов доктора	<i>doctorId: 10</i>	Сообщение «Успех» и код 200	Сервер возвращает сообщение «Успех» и код 200
<i>router.put</i> ('/doctor/edit-doctor-description')	Проверка обновления описания доктора	<i>doctorId: 10, description: текстовое описание</i>	Сообщение «Описание доктора успешно обновлено» и код 200	Сервер возвращает сообщение «Описание доктора успешно обновлено» и код 200

На рисунке 4.10 приведены результаты исполнения набора тестов из таблицы 4.6.

```
Тестирование маршрута бизнес логики /doctor
Подмаршрут /get-patients
  ✓ Проверка получения всех пациентов (2 ms)
Подмаршрут /get-patients-statistics
  ✓ Проверка получения статистики пациентов
Подмаршрут /get-reviews
  ✓ Проверка получения отзывов доктора
Подмаршрут /edit-doctor-description
```

Рисунок 4.10 – Результаты тестирования *doctor.route*

В таблице 4.7 приведено описание набора модульных тестов для обработчиков *admin.route*, отвечающий за обработку запросов, связанных с родственниками пациентов.

Таблица 4.7 – Набор модульных тестов для *admin.route*

Название обработчика	Описание теста	Исходные данные	Ожидаемый результат	Результат теста
<i>router.get</i> (<i>'/admin/get-accounts'</i>)	Проверка получения всех аккаунтов	—	Сообщение «Успех» и код 200	Сервер возвращает сообщение «Успех» и код 200
<i>router.put</i> (<i>'/admin/edit-account'</i>)	Проверка существования аккаунта	<i>accountId: 10</i>	Сообщение «Аккаунт не существует» и код 400	Сервер возвращает сообщение «Аккаунт не существует» и код 400

На рисунке 4.11 приведены результаты исполнения набора тестов из таблицы 4.7.

```
Тестирование маршрута бизнес логики /admin
Подмаршрут /get-accounts
  ✓ Проверка получения всех аккаунтов (7 ms)
Подмаршрут /edit-account
  ✓ Проверка существования аккаунта (2 ms)
```

Рисунок 4.11 – Результаты тестирования *admin.route*

Таким образом, модульное тестирование веб-приложений с использованием *Jest* помогает обеспечить стабильность и надежность приложения, позволяет выявить и исправить ошибки еще на ранних этапах разработки, а также упрощает процесс развертывания и поддержки приложения.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ

5.1 Техничко-экономическое обоснование целесообразности разработки программного продукта

Разработанный программный комплекс для автоматизации ведения дневника диабетика обладает преимуществами по сравнению с аналогами, которые перечислены ниже.

Программный комплекс предоставляет удобный и интуитивно понятный интерфейс, который позволяет пользователям легко записывать и отслеживать данные своего здоровья. Он предоставляет простые и понятные формы для ввода уровней глюкозы в крови, потребления пищи, физической активности и других показателей.

Программный комплекс автоматически рассчитывает различные параметры на основе введенных данных. Он может предоставлять средние значения уровня глюкозы, тренды изменения, анализировать факторы, влияющие на уровень глюкозы и давать рекомендации по регулировке режима и диеты.

Программный комплекс позволяет вести постоянный мониторинг уровня глюкозы и других показателей. Он может отправлять оповещения пользователю при превышении предельных значений или при необходимости выполнить дополнительные действия. Это помогает пользователю своевременно реагировать на изменения и поддерживать оптимальный уровень здоровья.

Программный комплекс предоставляет возможность анализировать данные и генерировать отчеты. Пользователи могут просматривать графики и диаграммы, которые позволяют им лучше понять и контролировать свое состояние. Отчеты также могут быть полезны для консультаций с врачами или специалистами в области диабета.

Программный комплекс обеспечивает высокий уровень защиты и конфиденциальности данных. Все личные данные пользователей хранятся в безопасной и зашифрованной форме. Также предусмотрены механизмы резервного копирования данных, чтобы избежать их потери.

Программный комплекс может предоставлять возможность общения с доктором. Это позволяет пользователям получать поддержку и советы.

5.2 Расчет общей трудоемкости разработки программного обеспечения

Общий объем трудоемкости разработки программного комплекса по автоматизации планирования и организации собеседований (V_0) определяется исходя из количества и объема функций, реализуемых программой, по каталогу функций ПО в соответствии с таблицей 1.1 приложения 1 источника по формуле (5.1):

$$V_0 = \sum_{i=1}^n V_i, \quad (5.1)$$

где V_i – объем отдельной функции ПО;

n – общее число функций.

$$V_o = 130 + 490 + 1040 + 280 + 1970 + 3500 + 1980 + 2370 + 7860 + 4720 + 2360 + 560 + 1050 + 2130 + 1540 + 1680 + 420 = 34080$$

Анализируя разработанную программу, уточненный объем ПО (V_y) определяем по формуле (5.2):

$$V_y = \sum_{i=1}^n V_{yi}, \quad (5.2)$$

где V_{yi} – уточнённый объем отдельной функции ПО в строках исходного кода (LOC).

$$V_y = 110 + 300 + 250 + 150 + 720 + 380 + 200 + 900 + 600 + 200 + 400 + 200 + 280 + 500 + 420 + 630 + 230 = 6460$$

Сравнение исходного и уточненных объемов строк исходного кода представлены в таблице Б.1 приложения Б.

Разработанное в ходе выполнения дипломной работы приложение относится к третьей категории сложности.

На основании принятого к расчету (уточненного) объема (V_y) и категории сложности ПО определяется нормативная трудоемкость ПО (T_n) выполняемых работ, которая приведена в таблице 5.1.

Таблица 5.1 – Нормативная трудоемкость на разработку ПО (T_n)

Уточненный объем, V_y	3-я категория сложности ПО	Номер нормы
6460	263	53

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО (K_c). K_c рассчитывается по формуле (5.3):

$$K_c = 1 + \sum_{i=1}^n K_i, \quad (5.3)$$

где K_i – коэффициент, соответствующий степени повышения сложности;

n – количество учитываемых характеристик.

Таким образом:

$$K_c = 1 + 0,07 = 1,07.$$

Новизна разработанного ПО определяется путем экспертной оценки данных, полученных при сравнении характеристик разрабатываемого ПО с имею-

щимися аналогами. Влияние фактора новизны на трудоемкость учитывается путем умножения нормативной трудоемкости на соответствующий коэффициент, учитывающий новизну ПО (K_H). Разработанная программа обладает категорией новизны В, а значение $K_H = 0,63$.

Современные технологии разработки компьютерных программ предусматривают широкое использование коробочных продуктов (пакетов, модулей, объектов). Степень использования в разрабатываемом ПО стандартных модулей определяется их удельным весом в общем объеме ПО [13].

В данном программном комплексе используется от 40% до 60% стандартных модулей, что соответствует значению коэффициента $K_T = 0,65$.

Приложение разработано на языке *Javascript*, что соответствует коэффициенту функционирования в *IBM-PC*, *Windows* сетях, учитывающему средства разработки ПО, $K_{yp} = 1,0$.

Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО определяются с учетом установленной категории новизны ПО согласно таблице 2.5.

При этом сумма значений коэффициентов удельных весов всех стадий в общей трудоемкости равна единице. Значения коэффициентов приведены в таблице 5.2.

Таблица 5.2 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО

Категория новизны ПО	Без применения CASE-технологии				
	Стадии разработки ПО				
	ТЗ	ЭП	ТП	РП	ВН
	Значения коэффициентов				
	$K_{ТЗ}$	$K_{ЭП}$	$K_{ТП}$	$K_{РП}$	$K_{ВН}$
В	0,08	0,19	0,28	0,34	0,11

Распределение нормативной трудоемкости ПО (V_0) по стадиям, чел.-ден. определяются по формулам:

– для стадии ТЗ по формуле (5.4):

$$V_{ТЗ} = T_H \cdot K_{ТЗ}; \quad (5.4)$$

– для стадии ЭП по формуле (5.5):

–

$$V_{ЭП} = T_H \cdot K_{ЭП}; \quad (5.5)$$

– для стадии ТП по формуле (5.6):

$$V_{ТП} = T_H \cdot K_{ТП}; \quad (5.6)$$

- для стадии РП по формуле (5.7):

$$V_{rp} = T_n \cdot K_{rp}; \quad (5.7)$$

- для стадии ВН по формуле (5.8):

$$V_{bn} = T_n \cdot K_{bn}. \quad (5.8)$$

Таким образом:

$$V_{tz} = 263 \cdot 0,08 \approx 21;$$

$$V_{эп} = 263 \cdot 0,19 \approx 50;$$

$$V_{тп} = 263 \cdot 0,28 \approx 74;$$

$$V_{rp} = 263 \cdot 0,34 \approx 89;$$

$$V_{bn} = 263 \cdot 0,11 \approx 29.$$

Нормативная трудоемкость ПО (T_n) выполняемых работ по стадиям разработки корректируется с учетом коэффициентов: повышения сложности ПО (T_c), учитывающих новизну ПО (K_n), учитывающих степень использования стандартных модулей (K_r), средства разработки ПО (K_{yp}) и определяются по формулам:

- для стадии ТЗ по формуле (5.9):

$$T_{y.tz} = T_n \cdot K_{tz} \cdot K_c \cdot K_n \cdot K_{yp}; \quad (5.9)$$

- для стадии ЭП по формуле (5.10):

$$T_{y.эп} = T_n \cdot K_{эп} \cdot K_c \cdot K_n \cdot K_{yp}; \quad (5.10)$$

- для стадии ТП по формуле (5.11):

$$T_{y.тп} = T_n \cdot K_{тп} \cdot K_c \cdot K_n \cdot K_{yp}; \quad (5.11)$$

- для стадии РП по формуле (5.12):

$$T_{y,rp} = T_n \cdot K_{rp} \cdot K_c \cdot K_n \cdot K_r; \quad (5.12)$$

- для стадии ВН по формуле (5.13):

$$T_{y.bn} = T_n \cdot K_{bn} \cdot K_c \cdot K_n \cdot K_{yp}. \quad (5.13)$$

Коэффициенты K_c , K_n , K_{yp} вводятся на всех стадиях разработки, а коэффициент K_t вводится только на стадии РП.

Таким образом:

$$T_{тз} = 263 \cdot 0,08 \cdot 1,07 \cdot 0,63 \cdot 1 \approx 14;$$

$$T_{эп} = 263 \cdot 0,19 \cdot 1,07 \cdot 0,63 \cdot 1 \approx 34;$$

$$T_{тп} = 263 \cdot 0,28 \cdot 1,07 \cdot 0,63 \cdot 1 \approx 50;$$

$$T_{рп} = 263 \cdot 0,34 \cdot 1,07 \cdot 0,63 \cdot 0,65 \approx 39;$$

$$T_{вн} = 263 \cdot 0,11 \cdot 1,07 \cdot 0,63 \cdot 1 \approx 20.$$

Общая трудоемкость разработки ПО (T_o) определяется суммированием нормативной (скорректированной) трудоемкости ПО по стадиям разработки формуле (5.14):

$$T_o = \sum_{i=1}^n T_{yi}, \quad (5.14)$$

где T_{yi} – нормативная (скорректированная) трудоемкость разработки ПО на i -й стадии (чел/дней);

n – количество стадий разработки.

Таким образом:

$$T_o = 14 + 34 + 50 + 39 + 20 = 157 \text{ чел. дн.}$$

Результаты расчетов по определению нормативной и скорректированной трудоемкости ПО по стадиям разработки и общую трудоемкость разработки ПО (T_o) представлены в таблице В.1 приложения В.

5.3 Расчет совокупных капитальных вложений в проект

В общем виде совокупность капитальных вложений в проект может быть рассчитан по формуле (5.16):

$$K = K_{об} + K_{на} - K_{л} + K_{пр}, \quad (5.15)$$

где $K_{об}$ – стоимость устанавливаемого оборудования, руб.;

$K_{на}$ – недоамортизированная часть стоимости демонтируемого оборудования, руб.;

$K_{л}$ – ликвидационная стоимость (выручка от продажи) демонтируемого оборудования, руб.;

$K_{пр}$ – стоимость приобретенных программных продуктов, руб.;

Таким образом:

$$K = 0 \text{ руб.}$$

Коэффициенты равны нулю, так как оборудование не демонтировалось и программные продукты не приобретались.

5.4 Расчёт затрат на разработку программного продукта

В состав затрат на разработку системы по автоматизации учёта лабораторных испытаний промышленного предприятия входят следующие статьи расходов:

- затраты труда на создание программного продукта (затраты по основной, дополнительной заработной плате и соответствующие отчисления) ($Z_{\text{тр}}$);
- затраты на изготовление эталонного экземпляра ($Z_{\text{эт}}$);
- затраты на технологию (затраты на приобретение и освоение программных средств, используемых при разработке программного продукта; затраты на ПО, используемое как эталон) ($Z_{\text{тех}}$);
- затраты на машинное время (расходы на содержание и эксплуатацию технических средств разработки, эксплуатации и сопровождения) ($Z_{\text{мв}}$);
- затраты на материалы (информационные носители) ($Z_{\text{мат}}$);
- затраты на энергию, на использование каналов связи (для отдельных видов);
- общепроизводственные расходы (затраты на управленческий персонал, на содержание помещений) ($Z_{\text{общ.пр}}$);
- непроизводственные (коммерческие) расходы (затраты, связанные с рекламой, поиском заказчиков, поставками конкретных экземпляров) ($Z_{\text{непр}}$).

В таблице В.1 приложения В приведены значения основных параметров, необходимых для расчёта затрат на разработку программного продукта.

Суммарные затраты на разработку ПО (Z_p) определяются по формуле (5.16):

$$Z_p = Z_{\text{тр}} + Z_{\text{эт}} + Z_{\text{тех}} + Z_{\text{мв}} + Z_{\text{мт}} + Z_{\text{общ.пр}} + Z_{\text{непр}} \quad (5.16)$$

Расходы на оплату труда разработчиков с отчислениями определяются по формуле (5.17):

$$Z_{\text{тр}} = 3П_{\text{осн}} + 3П_{\text{доп}} + ОТЧ_{\text{зп}}, \quad (5.17)$$

где $3П_{\text{осн}}$ – основная заработная плата разработчиков, руб.;

$3П_{\text{доп}}$ – дополнительная заработная плата разработчиков, руб.;

$ОТЧ_{\text{зп}}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

Основная заработная плата разработчиков приложения рассчитывается по

формуле (5.18):

$$ЗП_{\text{осн}} = C_{\text{ср_час}} \cdot T_o \cdot K_{\text{ув}}, \quad (5.18)$$

где $C_{\text{ср_час}}$ – средняя часовая тарифная ставка;

T_o – общая трудоемкость разработки, чел-час;

$K_{\text{ув}}$ – коэффициент, учитывающий доплаты стимулирующего характера.

Средняя часовая тарифная ставка определяется по формуле (5.19):

$$C_{\text{ср_час}} = \frac{\sum_i C_{\text{чи}} \cdot n_i}{\sum_i n_i}, \quad (5.19)$$

где $C_{\text{чи}}$ – часовая тарифная ставка разработчика 8 – го разряда;

n_i – количество разработчиков 8-го разряда.

Часовая тарифная ставка разработчика 8-го разряда определяется по формуле (5.20):

$$C_{\text{ч}} = T_{\text{ст}} \cdot k, \quad (5.20)$$

где $T_{\text{ст}}$ – базовая ставка;

k – тарифный коэффициент.

Таким образом:

$$C_{\text{ср_час}} = C_{\text{ч}} = \frac{228 \cdot 1,57 \cdot 2,08}{21 \cdot 8} = 4,43 \text{ руб.}$$

$$ЗП_{\text{осн}} = 4,43 \cdot 157 \cdot 8_{\text{ч}} \cdot 1,8 = 10015,34 \text{ руб.}$$

Дополнительная заработная плата определяется по формуле (5.21):

$$ЗП_{\text{доп}} = ЗП_{\text{осн}} \cdot \frac{Н_{\text{доп}}}{100\%}, \quad (5.21)$$

где $Н_{\text{доп}}$ – норматив отчислений на дополнительную заработную плату разработчиков.

Таким образом:

$$ЗП_{\text{доп}} = 10015,34 \cdot 0,15 = 1502,30 \text{ руб.}$$

$$ЗП = 10015,34 + 1502,30 = 11517,64 \text{ руб.}$$

Отчисления от основной и дополнительной заработной платы (отчисления на социальные нужды и обязательное страхование) рассчитываются по формуле (5.22):

$$ОТЧ_{\text{сн}} = (ЗП_{\text{осн}} + ЗП_{\text{доп}}) \cdot \frac{Н_{\text{зп}}}{100\%}, \quad (5.22)$$

где $H_{зп}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы ($H_{зп} = 34,6 \%$).

$$ОТЧ_{сн} = 11517,64 \cdot 0,346 = 3985,10 \text{ руб.}$$

$$З_{тр} = 10015,34 + 1502,30 + 3985,10 = 15502,74 \text{ руб.}$$

Затраты машинного времени определяются по формуле (5.23):

$$З_{мв} = C_{ч} \cdot K_{т} \cdot t_{эвм}, \quad (5.23)$$

где $C_{ч}$ – стоимость 1 часа машинного времени (руб./ч.);

$K_{т}$ – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ;

$K_{т}=1$;

$t_{эвм}$ – машинное время ЭВМ, необходимое для разработки и отладки проекта (ч.).

Стоимость машино-часа определяется по формуле (5.24):

$$C_{ч} = \frac{З_{Побсл} + З_{АР} + З_{АМ} + З_{ЭП} + З_{ВМ} + З_{ТР} + З_{ПР}}{F_{эвм}}, \quad (5.24)$$

где $З_{Побсл}$ – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, (руб. в год);

$З_{АР}$ – стоимость аренды помещения под размещение вычислительной техники, (руб. в год);

$З_{АМ}$ – амортизационные отчисления за год, (руб. в год);

$З_{ЭП}$ – затраты на электроэнергию, (руб. в год);

$З_{ВМ}$ – затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), (руб. в год);

$З_{ТР}$ – затраты на текущий и профилактический ремонт ЭВМ (руб. в год);

$З_{ПР}$ – прочие затраты, связанные с эксплуатацией ПЭВМ (руб. в год);

$F_{эвм}$ – действительный фонд времени работы ЭВМ (час/год).

Все статьи затрат формируются в расчете на единицу ПЭВМ.

Затраты на заработную плату обслуживающего персонала ($З_{Побсл}$) определяются по формуле (5.25):

$$З_{Побсл} = \frac{З_{Посн} + З_{Пдоп} + ОТЧ_{зп}}{Q_{эвм}}, \quad (5.25)$$

$$З_{Посн} = 12 \sum_{i=1}^n T_{ci},$$

$$З_{Пдоп} = З_{Посн} \cdot \frac{H_{доп}}{100\%},$$

$$\text{ОТЧ}_{\text{зп}} = (\text{ЗП}_{\text{осн}} + \text{ЗП}_{\text{доп}}) \cdot \frac{H_{\text{зп}}}{100\%},$$

где $\text{ЗП}_{\text{осн}}$ – основная заработная плата обслуживающего персонала, руб.;
 $\text{ЗП}_{\text{доп}}$ – дополнительная заработная плата обслуживающего персонала, руб.;
 $\text{ОТЧ}_{\text{зп}}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.;
 $Q_{\text{ЭВМ}}$ – количество обслуживаемых ПЭВМ, шт.;
 T_{ci} – месячная тарифная ставка i -го работника, руб.;
 n – численность обслуживающего персонала, чел.;
 $H_{\text{доп}}$ – процент дополнительной заработной платы обслуживающего персонала от основной;
 $H_{\text{зп}}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы.
Тарифная ставка 8-го разряда обслуживающего персонала:

$$T_{c8} = 228 \cdot 1,57 \cdot 2,08 = 744,56 \text{ руб.}$$

$$\text{ЗП}_{\text{осн}} = 12 \cdot 744,56 = 8934,72 \text{ руб.}$$

$$\text{ЗП}_{\text{доп}} = 8934,72 \cdot 0,15 = 1340,21 \text{ руб.}$$

$$\text{ЗП} = \text{ЗП}_{\text{осн}} + \text{ЗП}_{\text{доп}} = 8934,72 + 1340,21 = 10274,93 \text{ руб.}$$

$$\text{ОТЧ}_{\text{зп}} = 10274,93 \cdot 0,346 = 3555,13 \text{ руб.}$$

$$\text{ЗП}_{\text{обсл}} = 10274,93 + 3555,13 = 13830,06 \text{ руб.}$$

Годовые затраты на аренду помещения ($\text{З}_{\text{АР}}$) определяются по формуле (5.26):

$$\text{З}_{\text{АР}} = \frac{C_{\text{АР}} \cdot S}{Q_{\text{ЭВМ}}}, \quad (5.26)$$

где $C_{\text{АР}}$ – средняя годовая ставка арендных платежей, руб./м²;
 S – площадь помещения, м²;
 $Q_{\text{ЭВМ}}$ – количество ПЭВМ, шт.

$$\text{З}_{\text{АР}} = \frac{16,5 \cdot 12 \cdot 10}{1} = 1980 \text{ руб.}$$

Сумма годовых амортизационных отчислений ($\text{З}_{\text{АМ}}$) определяется по формуле (5.27):

$$З_{AM} = \frac{З_{приобр} \cdot (1 + K_{доп}) \cdot m \cdot H_{AM}}{Q_{ЭВМ}}, \quad (5.27)$$

где $З_{приобр}$ – стоимость уже существующей единицы ПЭВМ, руб.;

$K_{доп}$ – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{доп} = 12 - 13\%$ от $З_{приобр}$;

$З_{приобр} \cdot (1 + K_{доп})$ – балансовая стоимость ЭВМ, руб.;

H_{AM} – норма амортизации, %.

$$З_{AM} = \frac{2000 \cdot (1 + 0,12) \cdot 1 \cdot 0,2}{1} = 448 \text{ руб.}$$

Стоимость электроэнергии, потребляемой за год, ($З_{ЭП}$) определяется по формуле (5.28):

$$З_{ЭП} = M \cdot F_{ЭВМ} \cdot C_{эл} \cdot A, \quad (5.28)$$

где M – паспортная мощность ПЭВМ, (кВт), $M = 1$ кВт;

$C_{эл}$ – стоимость одного кВт-часа электроэнергии, руб.;

$F_{ЭВМ}$ – действительный годовой фонд времени работы ПЭВМ, $F_{ЭВМ} = 1713,6$ ч., согласно производственному календарю на 2023 год.

$$З_{ЭП} = 0,45 \cdot 1713,6 \cdot 0,3 \cdot 0,95 = 219,77 \text{ руб.}$$

Затраты на материалы ($З_{ВМ}$), необходимые для обеспечения нормальной работы ПЭВМ составляют около 1% от балансовой стоимости ЭВМ и определяются формулой (5.29):

$$З_{ВМ} = З_{приобр} \cdot (1 + K_{доп}) \cdot K_{МЗ}, \quad (5.29)$$

где $З_{приобр}$ – затраты на приобретение (стоимость) ЭВМ, руб.;

$K_{доп}$ – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{доп} = 12 - 13\%$ от $З_{приобр}$;

$K_{МЗ}$ – коэффициент, характеризующий затраты на вспомогательные материалы ($K_{МЗ} = 0,01$).

$$З_{ВМ} = 2000 \cdot (1 + 0,12) \cdot 0,01 = 22,4 \text{ руб.}$$

Затраты на текущий и профилактический ремонт ($З_{ТР}$) принимаются равными 5% от балансовой стоимости ЭВМ и рассчитываются по формуле (5.30):

$$З_{ТР} = З_{приобр} \cdot (1 + K_{доп}) \cdot K_{ТР}, \quad (5.30)$$

где $K_{\text{тр}}$ – коэффициент, характеризующий затраты на текущий и профилактический ремонт ($K_{\text{мз}} = 0,08$).

$$З_{\text{тр}} = 2000 \cdot (1 + 0,12) \cdot 0,08 = 179,2 \text{ руб.}$$

Прочие затраты, связанные с эксплуатацией ЭВМ ($З_{\text{пр}}$), состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций, составляют 5 % от балансовой стоимости и рассчитываются по формуле (5.31):

$$З_{\text{пр}} = З_{\text{приобр}} \cdot (1 + K_{\text{доп}}) \cdot K_{\text{пр}}, \quad (5.31)$$

где $K_{\text{пр}}$ – коэффициент, характеризующий размет прочих затрат, связанных с эксплуатацией ЭВМ ($K_{\text{пр}} = 0,05$).

$$З_{\text{пр}} = 2000 \cdot (1 + 0,12) \cdot 0,05 = 112 \text{ руб.}$$

Для расчета машинного времени ЭВМ ($t_{\text{эвм}}$ в часах), необходимого для разработки и отладки проекта, следует использовать формулу (5.32):

$$t_{\text{эвм}} = (t_{\text{рп}} + t_{\text{вн}}) \cdot F_{\text{см}} \cdot K_{\text{см}}, \quad (5.32)$$

где $t_{\text{рп}}$ – срок реализации стадии «Рабочий проект» (РП), 38 дней;
 $t_{\text{вн}}$ – срок реализации стадии «Ввод в действие» (ВП), 20 дней;
 $F_{\text{см}}$ – продолжительность рабочей смены, (ч.), $F_{\text{см}} = 8 \text{ ч.}$;
 $K_{\text{см}}$ – количество рабочих смен, $K_{\text{см}} = 1$.

$$t_{\text{эвм}} = (38 + 20) \cdot 8 \cdot 1 = 464 \text{ ч.}$$

$$C_{\text{ч}} = \frac{13830,06 + 1980 + 448 + 219,77 + 22,4 + 179,2 + 112}{1713,6} = 9,80 \text{ руб/ч}$$

$$З_{\text{мв}} = 9,80 \cdot 1 \cdot 368 = 3606,4 \text{ руб.}$$

При написании дипломной работы были использованы языки программирования *Javascript* и *Typescript*, поэтому затраты на технологию ($З_{\text{тех}}$) и изготовление эталонного экземпляра ($З_{\text{эт}}$) будут нулевыми.

Затраты на материалы (носители информации, распечатка и пр.), необходимые для обеспечения нормальной работы ПЭВМ примерно возьмем 75 руб.

Общепроизводственные затраты рассчитываются по формуле (5.33):

$$З_{\text{общ пр}} = З_{\text{посн}} \cdot \frac{H_{\text{доп}}}{100\%}, \quad (5.33)$$

где $N_{\text{доп}}$ – норматив общепроизводственных затрат.

$$З_{\text{общ пр}} = 8934,72 \cdot 0,05 = 446,74 \text{ руб.}$$

Непроизводственные затраты рассчитываются по формуле (5.34):

$$З_{\text{непр}} = З_{\text{Посн}} \cdot \frac{N_{\text{непр}}}{100\%}, \quad (5.34)$$

где $N_{\text{непр}}$ – норматив непроизводственных затрат.

$$З_{\text{непр}} = 8934,72 \cdot 0,03 = 268,04 \text{ руб.}$$

Итого получаем суммарные затраты на разработку:

$$\begin{aligned} З_p &= 13830,06 + 3606,4 + 0 + 0 + 75 + 446,74 + 268,04 = \\ &= 18226,24 \text{ руб.} \end{aligned}$$

Результаты расчетов приведены в таблице Е.1 приложения Е.

5.5 Формирование цены при создании программного обеспечения

Оптовая цена ПО ($Ц_{\text{опт}}$) определяется следующей формулой (5.35):

$$Ц_{\text{опт}} = З_p + П_p, \quad (5.35)$$

$$П_p = \frac{З_p \cdot Y_p}{100},$$

где $З_p$ – себестоимость ПО, руб.;

$П_p$ – прибыль от реализации ПО, руб.;

Y_p – уровень рентабельности ПО, % ($Y_p = 30\%$).

$$П_p = \frac{18226,24 \cdot 30}{100} = 5467,87 \text{ руб.}$$

$$Ц_{\text{опт}} = 18226,24 + 5467,87 = 23694,11 \text{ руб.}$$

Прогнозируемая отпускная цена ПО с НДС рассчитывается по формуле (5.36):

$$Ц_{\text{отп}} = З_p + П_p + P_{\text{ндс}}, \quad (5.36)$$

Налог на добавленную стоимость рассчитывается по формуле (5.37):

$$P_{\text{ндс}} = (З_p + П_p) \cdot \frac{H_{\text{ндс}}}{100}, \quad (5.37)$$

где $H_{\text{ндс}}$ – ставка налога на добавленную стоимость, %, $H_{\text{ндс}} = 20$ %.

$$P_{\text{ндс}} = (18226,24 + 5467,87) \cdot 0,2 = 4738,82 \text{ руб.}$$

$$Ц_{\text{отп}} = 18226,24 + 5467,87 + 4738,82 = 28432,93 \text{ руб.}$$

Розничную цену на программный продукт ($Ц_{\text{розн}}$) можно определить по формуле (5.38):

$$Ц_{\text{розн}} = Ц_{\text{отп}} + T_n \quad (5.38)$$

где T_n – торговая наценка при реализации программного обеспечения через специализированные магазины (торговых посредников), ее значение принимается равным 10-20% от отпускной цены с НДС.

$$Ц_{\text{розн}} = 28432,93 \cdot 1,1 = 31276,22 \text{ руб.}$$

Результаты расчетов формирования цены на разработку обучающей системы для формирования навыков приведены в таблице 5.3.

Таблица 5.3 – Расчет формирования цены на разработку программы

Наименование статьи расходов	Значение
Полная себестоимость	18226,24
Прибыль от реализации ПО	5467,87
Отпускная цена ПО без НДС	23694,11
Налог на добавленную стоимость	4738,82
Отпускная цена ПО с НДС	28432,93
Розничная цена	31276,22

5.6 Статистическая оценка экономической эффективности проекта

Данный продукт не является новым на рынке, но основывается на новых методах обучения, имеет удобный интерфейс и полезный функционал. Стоимость аналогичных обучающих систем ($З_{\text{баз}}$) равна 65000руб. Эффект может быть рассчитан по формуле (5.39):

$$\mathcal{E}(П) = З_{\text{баз}} - З_{\text{нов}}, \quad (5.39)$$

где $З_{\text{нов}}$ – текущие и инвестиционные затраты по новому проекту, руб.

$$\mathcal{E}(П) = 65000 - 31276,22 = 33723,78 \text{ руб.}$$

Рентабельность затрат (З) или инвестиций (И) на новую информационную технологию, программный продукт рассчитывается по формуле (5.40):

$$P = \frac{\text{Э(П)}}{\text{З(И)}} \cdot 100\%, \quad (5.40)$$

$$P = \frac{33723,78}{31276,22} \cdot 100\% = 107,83\%$$

Простой срок окупаемости затрат может быть рассчитан по следующей формуле (5.41):

$$T_{\text{пр}} = \frac{\text{З(И)}}{\text{Э(П)}}, \quad (5.41)$$

$$T_{\text{пр}} = \frac{31276,22}{33723,78} = 0,93 \text{ лет.}$$

Годовой экономический эффект может быть рассчитан по следующей формуле (5.42):

$$\text{ГЭЭ} = \text{Э(П)} - P_{\text{баз}} \cdot \text{З(И)}, \quad (5.42)$$

где $P_{\text{баз}}$ – рентабельность затрат (инвестиций) базового варианта, 25%.

Таким образом, годовой экономический эффект:

$$\text{ГЭЭ} = \text{Э(П)} = 65000 - 31276,22 = 33723,78 \text{ руб.}$$

На основании выполненных расчетов была сформирована таблица технико-экономических показателей проекта (таблица Д.1 приложения Д). После оценки технико-экономических показателей проектного программного обеспечения можно сделать вывод о том, что реализация проекта является обоснованной и экономически целесообразной, так как срок окупаемости проекта меньше года при размере годового экономического эффекта 33723,78 руб. с уровнем рентабельности 107,83%.

Экономическая составляющая программного комплекса является одним из важных аспектов, которые следует учитывать при оценке его эффективности и целесообразности внедрения. Одним из главных преимуществ является улучшение эффективности процесса учета: программный комплекс автоматизирует процесс учета показателей здоровья пациентов, что позволяет сократить время и усилия, затрачиваемые на ручной ввод и обработку данных. Это приводит к повышению эффективности работы врачей и снижению затрат на их рабочую силу.

6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ

6.1 Основные понятия охраны труда

Охрана труда – это система, которая обеспечивает безопасность и сохранение жизни и здоровья работников во время их профессиональной деятельности. В Республике Беларусь эта система основывается на комплексе действий и средств, которые включают правовые, социально-экономические, организационные, технические, психофизиологические, санитарно-гигиенические, лечебно-профилактические, реабилитационные и другие меры.

Согласно законодательству Республики Беларусь, система охраны труда распространяется на всех работников, включая граждан Республики Беларусь, иностранных граждан и лиц без гражданства, которые трудоустроены по трудовым и гражданско-правовым договорам. Она применяется независимо от организационно-правовых форм занятости. Кроме того, система охраны труда распространяется и на тех, кто приглашается юридическими лицами для выполнения работ или оказания услуг в соответствии с установленными законодательством порядком и условиями.

Цель системы охраны труда заключается в предотвращении производственных травматизмов, профессиональных заболеваний и создании безопасных и здоровых условий труда для работников. Это достигается через регулирование и нормирование рабочей среды, обучение и информирование работников, проведение проверок и анализов условий труда, внедрение соответствующих технических и организационных мероприятий, а также соблюдение соответствующего законодательства и нормативных требований.

Обеспечение здоровой рабочей среды и безопасности на рабочем месте имеет множество преимуществ для работников и организаций. Это включает снижение риска производственных несчастных случаев и заболеваний, повышение эффективности работы, снижение простоев и потерь рабочего времени, улучшение имиджа организации, повышение удовлетворенности и мотивации работников, а также соблюдение социальной ответственности и законодательных требований.

6.2 Оздоровление воздушной среды на предприятии

Оздоровление воздушной среды на предприятии представляет собой комплекс мер и действий, направленных на улучшение качества воздуха в рабочих и производственных помещениях. Это важный аспект охраны труда и здоровья работников, поскольку плохое качество воздуха может негативно сказываться на их здоровье, комфорте и производительности [14].

Для осуществления оздоровления воздушной среды предприятия применяются различные методы и технологии. Ниже представлены некоторые из них.

Вентиляция: обеспечение хорошей системы вентиляции является первоочередным шагом для улучшения качества воздуха. Это включает в себя правильную установку и обслуживание систем воздухообмена, таких как приточно-

вытяжная вентиляция, кондиционирование воздуха и фильтрация.

Управление выбросами: снижение выбросов опасных и вредных веществ в атмосферу может быть достигнуто с помощью применения современных технологий очистки газовых выбросов, установки фильтров и систем очистки дыма, а также использования экологически чистых и энергоэффективных процессов производства.

Контроль загрязнителей: регулярный мониторинг уровня загрязнителей в воздухе с помощью специального оборудования позволяет выявить проблемные зоны и источники загрязнений. На основе этих данных можно принять меры для устранения проблем и предотвращения дальнейшего загрязнения.

Использование экологически чистых материалов и технологий: замена опасных и вредных веществ, материалов и процессов на более безопасные и экологически дружелюбные варианты может существенно улучшить качество воздуха на предприятии. Это может включать переход к использованию низкотоксичных химических веществ, альтернативных источников энергии, а также применение технологий снижения шума и вибраций.

Проведение обучения и информирование работников: сознательность и знание о правилах и методах оздоровления воздушной среды важны для каждого работника. Обучение персонала по вопросам охраны труда, включая обучение по оздоровлению воздушной среды, помогает снизить риски заболеваний и повысить общую гигиеничность рабочей среды.

Соблюдение нормативных требований и законодательства: регулярное соблюдение всех применимых нормативных требований, правил и законодательных актов в области охраны труда и оздоровления воздушной среды является обязательным. Это включает проведение регулярных проверок, аудитов и испытаний, чтобы удостовериться, что предприятие соответствует всем необходимым стандартам и требованиям.

Оздоровление воздушной среды на предприятии является сложным и многогранным процессом, требующим комплексного подхода и сотрудничества между работниками, работодателями, специалистами по охране труда и органами государственного контроля. Это позволяет создать безопасную и здоровую рабочую среду, способствующую повышению эффективности производства и благополучию работников.

6.3 Вентиляция и системы воздухообмена

Вентиляция является ключевым аспектом обеспечения качественной воздушной среды на предприятии. Она отвечает за поступление свежего воздуха и удаление загрязненного воздуха из рабочих и производственных помещений. Хорошая система вентиляции способствует поддержанию оптимальных условий труда, предотвращает скопление вредных веществ и обеспечивает комфорт и безопасность для работников.

Важно установить и обслуживать правильные системы воздухообмена, которые отвечают требованиям конкретного предприятия. Они могут включать в себя компоненты, которые описаны ниже.

Приточная вентиляция – это система, которая обеспечивает поступление свежего воздуха в помещение. Приточный воздух фильтруется, подогревается или охлаждается до оптимальной температуры и распределяется по рабочей зоне. Это позволяет снизить концентрацию вредных веществ и поддерживать комфортные условия работы.

Вытяжная вентиляция – это система, которая удаляет загрязненный воздух из помещений. Она осуществляется с помощью вентиляционных вытяжек, расположенных вблизи источников загрязнения, таких как рабочие станки, печи или химические процессы. Вытяжные системы обеспечивают эффективное удаление вредных веществ и поддерживают оптимальную воздушную среду.

Фильтрация воздуха – использование фильтров в системе вентиляции помогает удалить пыль, аэрозоли, токсичные вещества и другие загрязнители из воздуха. Фильтры должны регулярно чиститься или заменяться, чтобы обеспечивать их эффективность.

Кондиционирование воздуха – это процесс поддержания оптимальной температуры и влажности в помещении. Кондиционирование воздуха может включать системы обогрева, охлаждения или увлажнения, которые помогают создать комфортные условия работы и предотвращают перегрев или переохлаждение работников.

Важно также регулярно проверять и обслуживать системы вентиляции, чтобы убедиться в их правильной работе. Это включает очистку и замену фильтров, проверку и ремонт вентиляционных каналов, а также периодическую проверку эффективности системы воздухообмена.

Хорошая вентиляционная система способна обеспечить свежий и чистый воздух на предприятии, что снижает риск заболеваний, повышает концентрацию работников и способствует общему благополучию на рабочем месте.

6.4 Управление выбросами и технологии очистки

Выбросы опасных веществ являются одним из основных источников загрязнения воздушной среды на предприятии. Управление выбросами и применение соответствующих технологий очистки имеют важное значение для снижения вредного воздействия на окружающую среду и здоровье работников. Ниже описаны аспекты, которые следует учесть.

Влияние выбросов на качество воздуха: выбросы опасных веществ, таких как токсичные газы, пары или пыль, могут негативно влиять на качество воздуха и вызывать серьезные проблемы для окружающей среды и здоровья людей. Понимание состава и количества выбросов помогает определить необходимость воздействия и выбор соответствующих технологий очистки.

Современные технологии очистки газовых выбросов: существует ряд эффективных технологий для очистки выбросов, которые позволяют улавливать и удалять опасные вещества перед их попаданием в атмосферу. Примеры таких технологий включают электростатические отделители, сорбционные системы, системы осаждения, каталитические фильтры и многое другое. Выбор определенной технологии зависит от типа загрязнителей и требований по очистке.

Применение систем фильтрации и очистки дыма: некоторые производственные процессы могут вызывать образование дыма, который содержит вредные частицы и газы. Применение систем фильтрации и очистки дыма, таких как механические фильтры или электрофильтры, позволяет улавливать и удалить опасные компоненты, прежде чем дым попадет в атмосферу.

Экологически чистые методы производства и снижение выбросов: разработка и внедрение экологически чистых методов производства является важным аспектом управления выбросами. Это может включать использование более эффективных технологий, переход на более экологически безопасные материалы, контроль и минимизацию потерь в процессах производства, а также поощрение энергоэффективности.

Управление выбросами и применение технологий очистки играют важную роль в снижении негативного воздействия предприятия на окружающую среду и обеспечении безопасных условий труда для работников. Это позволяет достичь соблюдения нормативов по качеству воздуха и поддерживать устойчивое и экологически ответственное производство.

6.5 Контроль качества воздуха и мониторинг загрязнений

Контроль качества воздуха и мониторинг загрязнений являются важными аспектами оздоровления воздушной среды на предприятии. Эти меры позволяют оценивать уровень загрязнения воздуха, идентифицировать и анализировать вредные вещества, а также принимать соответствующие меры для улучшения качества воздуха. Некоторые ключевые аспекты этой темы описаны ниже.

Мониторинг качества воздуха: проведение регулярного мониторинга качества воздуха позволяет измерять концентрацию вредных веществ и определять их соответствие нормативным требованиям. Для этого применяются различные методы и приборы, такие как анализаторы газов, пробоотборники, спектрометры и другие. Результаты мониторинга позволяют оценить эффективность применяемых мер по очистке и контролю загрязнений.

Идентификация вредных веществ: при контроле качества воздуха важно определить и идентифицировать вредные вещества, которые присутствуют в воздушной среде на предприятии. Это может быть осуществлено путем анализа проб воздуха на содержание токсичных газов, пыли, паров и других загрязнителей. Идентификация вредных веществ помогает принять целенаправленные меры по их устранению и контролю.

Установление нормативов и стандартов: регулирование качества воздуха осуществляется через установление нормативов и стандартов, которым должны соответствовать предприятия. Эти нормативы определяют предельно допустимые уровни концентрации вредных веществ и устанавливают требования к их контролю. Предприятия должны регулярно проверять свои показатели качества воздуха и соблюдать нормативы, чтобы предотвращать воздействие на окружающую среду и здоровье.

7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ

7.1 Экономия ресурсов при использовании программного продукта

В Республике Беларусь в области ресурсо- и энергосбережения активно применяются меры и стандарты, направленные на оптимизацию использования ресурсов и снижение энергопотребления. В этой сфере действует межгосударственный стандарт, разработанный Межгосударственным комитетом по стандартизации, метрологии и сертификации (МТК).

Межгосударственный стандарт, разработанный МТК, устанавливает требования и рекомендации по энергосбережению и энергоэффективности в различных секторах экономики. Он предоставляет руководство и регламентирует использование передовых технологий, методов и подходов, способствующих сокращению потребления энергии и оптимизации ресурсов.

МТК сотрудничает с органами государственного управления, предприятиями, общественными организациями и другими заинтересованными сторонами для проведения обучения, консультаций и информационной поддержки по внедрению стандарта и повышению энергоэффективности в стране. Разработанный стандарт служит основой для разработки и внедрения мер по сокращению потребления энергии, повышению эффективности использования ресурсов и снижению негативного влияния на окружающую среду.

Внедрение межгосударственного стандарта по ресурсо- и энергосбережению в Республике Беларусь имеет целью улучшение энергетической эффективности, сокращение затрат на энергию, снижение выбросов вредных веществ и обеспечение устойчивого развития экономики страны. Этот стандарт способствует повышению конкурентоспособности предприятий, снижению зависимости от импорта энергоносителей и общему сокращению негативного влияния на окружающую среду.

Стандарт «Энергетическая эффективность» в Республике Беларусь является важным нормативным документом, разработанным с целью регулирования и стимулирования повышения энергоэффективности в различных секторах экономики Беларуси. Он определяет требования, методы и процедуры, направленные на сокращение энергопотребления и оптимизацию энергетических процессов, а также устанавливает правила и нормы, которые должны соблюдаться организациями и предприятиями в целях повышения эффективности использования энергетических ресурсов. Он предусматривает различные меры, такие как проведение энергетического аудита, внедрение современных технологий энергосбережения, оптимизацию систем энергоснабжения и другие мероприятия, направленные на сокращение потребления энергии.

Применение ресурсо- и энергосберегающих практик при использовании программного обеспечения (ПО) имеет важное значение для экономии ресурсов, снижения негативного влияния на окружающую среду и повышения эффективности работы систем. Осознанное использование ресурсов и энергии является важным аспектом экологической ответственности. Время, энергия и материалы,

затрачиваемые на разработку, производство, эксплуатацию и утилизацию компьютерных систем и инфраструктуры, могут оказывать негативное воздействие на окружающую среду. Применение ресурсо- и энергосберегающих методов в ПО помогает уменьшить потребление энергии, сократить использование материалов и снизить выбросы вредных веществ.

Энергосбережение позволяет использовать ресурсы более эффективно и экономно. Оптимизированное использование вычислительной мощности, памяти, сетевых ресурсов и других компонентов системы позволяет сократить затраты на оборудование, улучшить производительность и уменьшить нагрузку на сетевую инфраструктуру. Это также может привести к снижению расходов на энергию, охлаждение и обслуживание. Также это может повысить эффективность работы систем и улучшить пользовательский опыт. Оптимизация алгоритмов, управление энергопотреблением, виртуализация и другие техники могут сократить время обработки задач, ускорить процессы и повысить отзывчивость системы. Это в свою очередь способствует повышению производительности бизнеса и улучшению пользовательского удовлетворения.

Применение ресурсо- и энергосберегающих стандартов в программном обеспечении играет важную роль в создании более устойчивых систем и инфраструктуры. В современных условиях, когда наблюдается рост нагрузки на вычислительные ресурсы и ограниченность природных ресурсов, эта проблематика становится особенно актуальной.

Сокращение потребления энергии, оптимизация использования ресурсов и эффективное управление инфраструктурой имеют несколько преимуществ. Во-первых, это позволяет снизить негативное воздействие на окружающую среду, уменьшить выбросы парниковых газов и других загрязнений. В результате, улучшается экологическая устойчивость системы и снижается ее негативный след. Во-вторых, применение ресурсо- и энергосберегающих стандартов помогает сократить затраты на энергию и ресурсы. Оптимизация работы программного обеспечения позволяет использовать вычислительные ресурсы более эффективно, что приводит к снижению расходов на их поддержку и увеличению экономической эффективности проектов.

Кроме того, использование ресурсо- и энергосберегающих стандартов способствует повышению устойчивости системы в условиях возможных энергетических и ресурсных рисков. Сокращение зависимости от энергии и ресурсов помогает снизить вероятность проблем, связанных с их нестабильностью, дефицитом или ростом цен. Это позволяет создавать более надежные и гибкие системы, способные адаптироваться к переменным условиям.

В итоге, межгосударственный стандарт, разработанный МТК в Республике Беларусь, является важным инструментом для содействия ресурсо- и энергосбережению. Он способствует повышению энергоэффективности в различных отраслях экономики и приводит к более устойчивому и экологически ответственному развитию страны. Применение ресурсо- и энергосбережения в ПО является важным аспектом современной разработки и использования технологий. Оно способствует экологической ответственности, экономии ресурсов, повышению эффективности и устойчивости систем, а также улучшению восприятия бренда.

ЗАКЛЮЧЕНИЕ

В ходе написания дипломной работы был разработан программный комплекс автоматизации дневника диабетика. Была подобрана и проанализирована литература о разработке приложений, используя стек *FERN*. В ходе анализа задачи было выявлено, что создание хорошего приложения требует сочетания продуманного интерфейса и корректной обработки различных ошибок на стороне сервера и клиента. Для достижения этой цели были выбраны наиболее подходящие технологии, которые обладают богатым набором инструментов и компонентов.

Результатом дипломной работы является *web*-приложение для ведения дневника диабетика. В ходе разработки программного продукта были решены задачи, которые описаны ниже.

Проведен анализ требований пользователей и определена основная функциональность приложения. На основе этой информации был разработан дизайн интерфейса, включающий в себя компоненты, макеты и взаимодействие элементов.

Выполнен процесс верстки, включающий создание *HTML*-структуры и стилей *CSS*. Корректное размещение и стилизация элементов интерфейса были реализованы с учетом современных стандартов и рекомендаций.

Разработана эффективная навигация и логика взаимодействия между различными компонентами интерфейса. Это включает реализацию кнопок, форм, меню, всплывающих окон и других элементов, необходимых для удобной работы с приложением;

Применены оптимизационные методы для улучшения производительности интерфейса: минимизация запросов к серверу, оптимизация загрузки изображений и ресурсов, а также кэширование данных для более быстрого доступа;

Проведены тесты интерфейса и бизнес-логики, чтобы убедиться в его корректной работе и соответствии требованиям.

Разработанный программный продукт предназначен для использования докторами, пациентами, и их родственниками, которые хотят быстро взаимодействовать друг с другом. Так как приложений такого плана в открытом доступе практически нет, разработанное приложение может стать очень ценным для людей, больных диабетом.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Язык высокого уровня *JavaScript*. [Электронный ресурс] – Режим доступа: https://developer.mozilla.org/ru/docs/JavaScript/First_steps/What_is_js/ – Дата доступа – 21.04.2023.
2. Язык высокого уровня *TypeScript*. [Электронный ресурс] – Режим доступа: https://skillbox.ru/media/code/typescript_chem_on_otlichaetsya_ot_js/ – Дата доступа – 21.04.2023.
3. Гриффитс Дэвид, Программирование для *Android* / Гриффитс Дэвид; под общ. ред. Гриффитс Дон. – М. : СПб, 2018. – 95 с.
4. *Firebase* – Американская компания, поставщик облачных услуг. [Электронный ресурс]. – *Firebase*, 2021. – Режим доступа: <https://console.firebase.google.com/> – Дата доступа – 21.04.2023.
5. *Express* и *NodeJs*. [Электронный ресурс] – Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs.
6. Библиотека *React*. [Электронный ресурс] – Режим доступа: <https://metanit.com/web/react/1.1.php> – Дата доступа – 21.04.2023.
7. *IDE WebStorm*. [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/WebStorm#:~:text=JetBrains%20WebStorm%20%20интегрированная%20среда%20разработки,на%20основе%20платформы%20IntelliJ%20IDEA.&text=WebStorm%20обеспечивает%20автодополнение%20C%20анализ%20кода,интеграцию%20с%20системами%20управления%20версиями> – Дата доступа – 21.04.2023.
8. Документация *FireAdmin*. [Электронный ресурс] – Режим доступа: [https://docs.fireadmin.io/docs/adding-to-these-docs %20Windows](https://docs.fireadmin.io/docs/adding-to-these-docs%20Windows) – Дата доступа – 21.04.2023.
9. *StarUml*. Описание и как он устроен. [Электронный ресурс] – Режим доступа: <https://soware.ru/products/staruml> – Дата доступа – 21.04.2023.
10. Как работает *Postman*. Общие принципы [Электронный ресурс] – Режим доступа: <https://blog.skillfactory.ru/glossary/postman/> – Дата доступа – 21.04.2023.
11. *David St. Clair, React: Быстрый старт* / *David St. Clair*; практическое руководство / пер. с англ. А. Гуляев. – СПб.: БХВ-Петербург, 2018. – 304 с.
12. *Neil Smyth, Firebase Essentials – Second Edition* / *Neil Smyth*; под общ. ред. *David St. Clair*. – М.: *Santa Barbara, USA*, 2016. – 206 с.
13. Кожевников, Е. А. Расчет экономической эффективности разработки программных продуктов: методические указания по подготовке организационно-экономического раздела дипломных работ для студентов специальности 1-40-01-02 «Информационные системы и технологии (по направлениям)» дневной формы обучения / Е. А. Кожевников, Н. В. Ермалинская. – Гомель: ГГТУ им. П. О. Сухого, 2012. – 68 с.
14. Б Ефремова, О.С. Охрана труда от А до Я / О.С. Ефремова. – Изд. 6-е, перераб. и доп. – М. : Альфа-Пресс, 2011. – 622 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

SignIn.tsx

```
import * as React from "react";
import Avatar from "@mui/material/Avatar";
import Button from "@mui/material/Button";
import TextField from "@mui/material/TextField";
import Link from "@mui/material/Link";
import Paper from "@mui/material/Paper";
import Box from "@mui/material/Box";
import Grid from "@mui/material/Grid";
import LockOutlinedIcon from "@mui/icons-material/LockOutlined";
import Typography from "@mui/material/Typography";
import { Route } from "components/routing";
import { useNavigate } from "react-router-dom";
import { NotificationManager } from "react-notifications";
import { auth } from "firebase_config";
import { useSignInWithEmailAndPassword } from "react-firebase-hooks/auth";
import { useEffect, useRef, useState } from "react";
import { LoadingSpinner } from "components/ui/LoadingSpinner";
import { successfulAuth } from "helpers/auth.helpers";
import { isMobile } from "react-device-detect";
export const SignIn = () => {
  const navigate = useNavigate();
  const formRef = useRef<HTMLFormElement>();
  const getFormField = (name: string): string => {
    const data = new FormData(formRef.current);
    return data.get(name)?.toString() || "";
  };
  const [signInWithEmailAndPassword, , loading, error] =
    useSignInWithEmailAndPassword(auth);
  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    const email = getFormField("email");
    const password = getFormField("password");
    const userCredential = await signInWithEmailAndPassword(email, password);
    if (userCredential) {
      await successfulAuth(userCredential, navigate);
    }
  };
  useEffect(() => {
    if (error) {
      NotificationManager.error(error?.name, error?.message);
    }
  }, [error]);
  const [isSignInDisabled, setIsSignInDisabled] = useState<boolean>(false);
  const updateIsSignInDisabled = () => {
    setIsSignInDisabled(!getFormField("email") || !getFormField("password"));
  };
  return (
    <Grid container component="main" sx={{ height: "100vh" }}>
      <Grid
        item
        xs={false}
        sm={4}
        md={7}

```

```

sx={ {
  backgroundImage: "url(/images/sign-in-background-min.jpg)",
  backgroundRepeat: "no-repeat",
  backgroundColor: (t) =>
    t.palette.mode === "light"
      ? t.palette.grey[50]
      : t.palette.grey[900],
  backgroundSize: "cover",
  backgroundPosition: "center",
}}
/>
<Grid
  item
  xs={ 12 }
  sm={ 8 }
  md={ 5 }
  component={ Paper }
  elevation={ 6 }
  square
  sx={ {
    display: "flex",
    alignItems: isMobile ? "center" : undefined,
  }}
>
  <Box
    sx={ {
      marginTop: "10px",
      marginLeft: "20px",
      marginRight: "20px",
      width: "100%",
      display: "flex",
      flexDirection: "column",
      alignItems: "center",
    }}
  >
    <Avatar sx={ { m: 1, bgcolor: "secondary.main" } }>
      <LockOutlinedIcon />
    </Avatar>
    <Typography component="h1" variant="h5">
      Вход
    </Typography>
    <Box
      component="form"
      noValidate
      onSubmit={ handleSubmit }
      sx={ { mt: 1 } }
      ref={ formRef }
    >
      <TextField
        margin="normal"
        required
        fullWidth
        id="email"
        label="Электронная почта"
        name="email"
        autoComplete="email"
        autoFocus
        onChange={ updateIsSignInDisabled }
      />
      <TextField
        margin="normal"
        required
        fullWidth

```

```

        name="password"
        label="Пароль"
        type="password"
        id="password"
        autoComplete="current-password"
        onChange={updateIsSignInDisabled}
      />
      {loading ? (
        <LoadingSpinner />
      ) : (
        <Button
          disabled={isSignInDisabled}
          type="submit"
          fullWidth
          variant="contained"
          sx={{ mt: 3, mb: 2 }}
        >
          Войти
        </Button>
      )}
      <Grid container justifyContent="flex-end">
        <Grid item>
          <Link href={Route.signUp} variant="body2">
            {"Нет аккаунта? Зарегистрируйтесь"}
          </Link>
        </Grid>
      </Grid>
    </Box>
  </Box>
</Grid>
</Grid>
);
};

```

SignOut.tsx

```

import React, { useCallback, useEffect } from "react";
import { auth } from "firebase_config";
import { ACTION_NAMES } from "store/reducers/user/constants";
import { NotificationManager } from "react-notifications";
import { Route } from "components/routing";
import { useAppDispatch } from "store";
import { Outlet, useNavigate } from "react-router-dom";
import { streamClient } from "containers/App/App";
export const SignOut = () => {
  const navigate = useNavigate();
  const dispatch = useAppDispatch();
  const onSignOut = useCallback(async () => {
    try {
      if (!auth.currentUser) {
        navigate(Route.home);
        return;
      }
      await auth.signOut();
      if (streamClient.user?.online) {
        streamClient.disconnectUser().catch((e) => {
          console.log(e);
          NotificationManager.error("Выход из сервиса чата", "Авторизация");
        });
      }
      dispatch({ type: ACTION_NAMES.userSignOut });
      NotificationManager.success("Успешный выход из системы!");
    }
  }, []);
};

```

```

    navigate(Route.home);
  } catch (e) {
    console.log(e);
    NotificationManager.error("Выход из системы", "Авторизация");
    navigate(-1);
  }
}, [dispatch, navigate]);
useEffect(() => {
  onSignOut();
}, [onSignOut]);
return <Outlet />;
};

```

SignUp.tsx

```

import * as React from "react";
import Avatar from "@mui/material/Avatar";
import Button from "@mui/material/Button";
import TextField from "@mui/material/TextField";
import Link from "@mui/material/Link";
import Grid from "@mui/material/Grid";
import Box from "@mui/material/Box";
import LockOutlinedIcon from "@mui/icons-material/LockOutlined";
import Typography from "@mui/material/Typography";
import Paper from "@mui/material/Paper";
import { Route } from "components/routing";
import { createUserWithEmailAndPassword } from "react-firebase-hooks/auth";
import { auth, firebaseConfig } from "firebase_config";
import { useEffect, useRef, useState } from "react";
import { useNavigate } from "react-router-dom";
import { NotificationManager } from "react-notifications";
import { LoadingSpinner } from "components/ui/LoadingSpinner";
import { successfulAuth, successfulSignUp } from "helpers/auth.helpers";
import { isMobile } from "react-device-detect";
import {
  FormControl,
  InputLabel,
  MenuItem,
  Select,
  SelectChangeEvent,
} from "@mui/material";
import { Role } from "firebase/types/collections.types";
import { convertRoleToRussian } from "firebase/converters";
import { firebaseRepositories } from "firebase/data/repositories";
import { getAuth, signInWithEmailAndPassword } from "firebase/auth";
import { initializeApp } from "firebase/app";
const firebaseApp2 = initializeApp(firebaseConfig, "user_creation");
const auth2 = getAuth(firebaseApp2);
export const SignUp = () => {
  const formRef = useRef<HTMLFormElement>();
  const getFormField = (name: string): string => {
    const data = new FormData(formRef.current);
    return data.get(name)?.toString() || "";
  };
  const [role, setRole] = useState<Role>(Role.PATIENT);
  const [isLoading, setIsLoading] = useState<boolean>(false);
  const [createUserWithEmailAndPassword, , loading, error] =
    createUserWithEmailAndPassword(auth2);
  const navigate = useNavigate();
  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    try {

```

```

setIsLoading(true);
const firstName = getFormField("firstName");
const lastName = getFormField("lastName");
const email = getFormField("email");
const password = getFormField("password");
const relativePatientId = getFormField("relativePatientId");
const diabetType = +getFormField("diabetType");
if (role === Role.RELATIVE) {
  if (!relativePatientId) {
    NotificationManager.error(
      "Код приглашения не может быть пустым",
      "Авторизация"
    );
    return;
  }
  const patient = await firebaseRepositories.users.getDocById(
    relativePatientId
  );
  if (!patient) {
    NotificationManager.error(
      "Неправильный код приглашения",
      "Авторизация"
    );
    return;
  }
}
const userCredential = await createUserWithEmailAndPassword(
  email,
  password
);
await auth2.signOut();
if (userCredential) {
  await successfulSignUp({
    firstName,
    lastName,
    email,
    password,
    role,
    relativePatientId,
    diabetType,
    userCredential,
  });
  await signInWithEmailAndPassword(auth, email, password);
  await successfulAuth(userCredential, navigate);
}
} catch (e) {
  console.log(e);
  NotificationManager.error(
    "Создание нового пользователя",
    "Ошибка при регистрации"
  );
} finally {
  setIsLoading(false);
}
};
useEffect(() => {
  if (error) {
    NotificationManager.error(error?.name, error?.message);
  }
}, [error]);
const [isSignUpDisabled, setIsSignUpDisabled] = useState<boolean>(true);
const [arePasswordsEqual, setArePasswordsEqual] = useState<boolean>(true);
const updateUpSignInDisabled = (

```



```

    currentRole: Role = getFormField("role") as Role
  ) => {
    const email = getFormField("email");
    const firstName = getFormField("firstName");
    const lastName = getFormField("lastName");
    const password = getFormField("password");
    const password2 = getFormField("password2");
    const relativePatientId = getFormField("relativePatientId");
    setArePasswordsEqual(password === password2);
    setIsSignUpDisabled(
      [
        email,
        firstName,
        lastName,
        password,
        password2,
        (!!relativePatientId && currentRole === Role.RELATIVE) ||
        currentRole === Role.PATIENT,
      ]
      .map(Boolean)
      .some((value) => !value) || password !== password2
    );
  };
  const handleRoleChange = (event: SelectChangeEvent) => {
    const newRole = event.target.value as Role;
    setRole(newRole);
    updateUpSignInDisabled(newRole);
  };
  return (
    <Grid container component="main" sx={{ height: "100vh" }}>
      <Grid
        item
        xs={false}
        sm={4}
        md={7}
        sx={{
          backgroundImage: "url(/images/sign-up-background-min.jpg)",
          backgroundRepeat: "no-repeat",
          backgroundColor: (t) =>
            t.palette.mode === "light"
              ? t.palette.grey[50]
              : t.palette.grey[900],
          backgroundSize: "cover",
          backgroundPosition: "center",
        }}
      />
      <Grid
        item
        xs={12}
        sm={8}
        md={5}
        component={Paper}
        elevation={6}
        square
        sx={{
          p: (_theme) => _theme.spacing(2),
          display: "flex",
          alignItems: isMobile ? "center" : undefined,
        }}
      />
      <Box
        sx={{
          display: "flex",

```

```

        flexDirection: "column",
        alignItems: "center",
    }}
>
<Avatar sx={{ m: 1, bgcolor: "secondary.main" }}>
  <LockOutlinedIcon />
</Avatar>
<Typography component="h1" variant="h5">
  Регистрация
</Typography>
<Box
  ref={formRef}
  component="form"
  noValidate
  onSubmit={handleSubmit}
  sx={{ mt: 3 }}
>
  <Grid container spacing={2}>
    <Grid item xs={12} sm={6}>
      <TextField
        autoComplete="given-name"
        name="firstName"
        required
        fullWidth
        id="firstName"
        disabled={loading || isLoading}
        label="Имя"
        autoFocus
        onChange={() => updateUpSignInDisabled()}
      />
    </Grid>
    <Grid item xs={12} sm={6}>
      <TextField
        required
        fullWidth
        id="lastName"
        label="Фамилия"
        name="lastName"
        disabled={loading || isLoading}
        autoComplete="family-name"
        onChange={() => updateUpSignInDisabled()}
      />
    </Grid>
    <Grid item xs={12}>
      <TextField
        required
        fullWidth
        id="email"
        disabled={loading || isLoading}
        label="Электронная почта"
        name="email"
        autoComplete="email"
        onChange={() => updateUpSignInDisabled()}
      />
    </Grid>
    <Grid item xs={12}>
      <TextField
        required
        fullWidth
        name="password"
        label="Пароль"
        type="password"
        id="password"

```

```

        disabled={loading || isLoading}
        autoComplete="new-password"
        onChange={() => updateUpSignInDisabled()}
      />
    </Grid>
    <Grid item xs={12}>
      <TextField
        required
        fullWidth
        error={!arePasswordsEqual}
        name="password2"
        label="Повторённый пароль"
        type="password"
        id="password2"
        disabled={loading || isLoading}
        autoComplete="repeat-password"
        onChange={() => updateUpSignInDisabled()}
        helperText={!arePasswordsEqual && "Пароли должны совпадать!"}
      />
    </Grid>
    <Grid item xs={12}>
      <FormControl fullWidth>
        <InputLabel id="role-select-label">Роль</InputLabel>
        <Select
          labelId="role-select-label"
          value={role}
          name="role"
          label="Роль"
          disabled={loading || isLoading}
          onChange={handleRoleChange}
        >
          {[Role.PATIENT, Role.RELATIVE].map((currentRole) => (
            <MenuItem key={currentRole} value={currentRole}>
              {convertRoleToRussian(currentRole)}
            </MenuItem>
          ))}
        </Select>
      </FormControl>
    </Grid>
    {role === Role.PATIENT && (
      <Grid item xs={12}>
        <FormControl fullWidth>
          <InputLabel id="diabet-type-select-label">
            Тип диабета
          </InputLabel>
          <Select
            labelId="diabet-type-select-label"
            name="diabetType"
            defaultValue={1}
            label="Тип диабета"
            disabled={loading || isLoading}
          >
            <MenuItem value={1}>1</MenuItem>
            <MenuItem value={2}>2</MenuItem>
          </Select>
        </FormControl>
      </Grid>
    )}
    {role === Role.RELATIVE && (
      <Grid item xs={12}>
        <TextField
          required
          fullWidth

```

```

        name="relativePatientId"
        label="Код приглашения"
        disabled={loading || isLoading}
        onChange={() => updateUpSignInDisabled()}
      />
    </Grid>
  )}
</Grid>
{loading || isLoading ? (
  <LoadingSpinner />
) : (
  <Button
    disabled={isSignUpDisabled}
    type="submit"
    fullWidth
    variant="contained"
    sx={{ mt: 3, mb: 2 }}
  >
    Зарегистрироваться
  </Button>
)}
<Grid container justifyContent="flex-end">
  <Grid item>
    <Link href={Route.signIn} variant="body2">
      Уже есть аккаунт? Войдите
    </Link>
  </Grid>
</Grid>
</Box>
</Box>
</Grid>
</Grid>
);
};

```

AppTitle.tsx

```

import * as React from "react";
import { styled } from "@mui/material/styles";
import MuiAppBar, { AppBarProps as MuiAppBarProps } from "@mui/material/AppBar";
import Toolbar from "@mui/material/Toolbar";
import Typography from "@mui/material/Typography";
import IconButton from "@mui/material/IconButton";
import MenuIcon from "@mui/icons-material/Menu";
import { isMobile } from "react-device-detect";
import { useSelector } from "react-redux";
import { getUserInfoSelector } from "store/selectors";
import { getUserFullName } from "firebase/helpers";
import Avatar from "@mui/material/Avatar";
import { useNavigate } from "react-router-dom";
import { Route } from "components/routing";
import { convertRoleToRussian } from "firebase/converters";
const AppBar = styled(MuiAppBar, {
  shouldForwardProp: (prop) => prop !== "open",
})<MuiAppBarProps>(({ theme }) => ({
  zIndex: theme.zIndex.drawer + 1,
  display: "flex",
  justifyContent: "center",
  transition: theme.transitions.create(["width", "margin"], {
    easing: theme.transitions.easing.sharp,
    duration: theme.transitions.duration.leavingScreen,
  }),

```

```

    ));
    export const AppTitle = ({
      title,
      isNavBarOpened,
      setIsNavBarOpened,
    }: {
      title: string;
      isNavBarOpened: boolean;
      setIsNavBarOpened: React.Dispatch<boolean>;
    }) => {
      const toggleDrawer = () => {
        setIsNavBarOpened(!isNavBarOpened);
      };
      const userInfo = useSelector(getUserInfoSelector);
      const navigate = useNavigate();
      const onProfileClick = () => {
        setIsNavBarOpened(false);
        navigate(Route.profile);
      };
      return (
        <AppBar sx={{ width: "100%", height: "60px !important" }}>
          <Toolbar>
            {isMobile && (
              <IconButton
                edge="start"
                color="inherit"
                aria-label="open drawer"
                onClick={toggleDrawer}
                sx={{
                  marginRight: "20px",
                }}
              >
                <MenuIcon />
              </IconButton>
            )}
            <Typography
              component="h1"
              variant="h6"
              color="inherit"
              noWrap
              sx={{ flexGrow: 1 }}
            >
              {title}
            </Typography>
            {!isMobile && (
              <Typography
                component="h1"
                variant="h6"
                color="inherit"
                noWrap
                onClick={onProfileClick}
                sx={{
                  marginRight: "10px",
                  userSelect: "none",
                  ":hover": { cursor: "pointer" },
                }}
              >
                {getUserFullName(userInfo)} ({convertRoleToRussian(userInfo.role)})
              </Typography>
            )}
            <IconButton onClick={onProfileClick}>
              <Avatar src={userInfo.imageUrl} alt={getUserFullName(userInfo)} />
            </IconButton>
          </Toolbar>
        </AppBar>
      );
    };
  };
}

```

```

    </Toolbar>
  </AppBar>
);
};

```

Layout.tsx

```

import React, { useState } from "react";
import { Grid } from "@mui/material";
import { Navbar } from "components/layout/Navbar";
import { matchPath, Outlet, useLocation } from "react-router-dom";
import { AppTitle } from "components/layout/AppTitle";
import { isMobile } from "react-device-detect";
import { Route, RouteTitles } from "components/routing/constants";
const findRoute = (path: string): Route =>
  Object.values(Route)
    .filter((route) => matchPath(route, path))
    .at(-1) as Route;
export const Layout = () => {
  const location = useLocation();
  const currentRoute = findRoute(location.pathname);
  const title = RouteTitles[currentRoute];
  const [isNavBarOpened, setIsNavBarOpened] = useState<boolean>(!isMobile);
  return (
    <Grid container columnSpacing={10}>
      <AppTitle
        title={title}
        isNavBarOpened={isNavBarOpened}
        setIsNavBarOpened={setIsNavBarOpened}
      />
      <Navbar
        isNavBarOpened={isNavBarOpened}
        setIsNavBarOpened={setIsNavBarOpened}
      />
      <Outlet />
    </Grid>
  );
};

```

constants.tsx

```

import React from "react";
import { Role } from "firestore/types/collections.types";
import { Route } from "components/routing/constants";
import PersonIcon from "@mui/icons-material/Person";
import SummarizeIcon from "@mui/icons-material/Summarize";
import AlarmIcon from "@mui/icons-material/Alarm";
import TrendingUpIcon from "@mui/icons-material/TrendingUp";
import ChatIcon from "@mui/icons-material/Chat";
import FamilyRestroomIcon from "@mui/icons-material/FamilyRestroom";
import InsightsIcon from "@mui/icons-material/Insights";
import QueryStatsIcon from "@mui/icons-material/QueryStats";
import WysiwygIcon from "@mui/icons-material/Wysiwyg";
import PeopleIcon from "@mui/icons-material/People";
import LogoutIcon from "@mui/icons-material/Logout";
import LocalHospitalIcon from "@mui/icons-material/LocalHospital";
import ReviewsIcon from "@mui/icons-material/Reviews";
import RecommendIcon from "@mui/icons-material/Recommend";
import MonitorHeartIcon from "@mui/icons-material/MonitorHeart";
import FavoriteIcon from "@mui/icons-material/Favorite";

```

```

export type NavBarItem = {
  id: number;
  icon: React.ReactNode;
  label: string;
  route: Route;
  roles: Role[];
};
export const topNavbarItems: NavBarItem[] = [
  {
    id: 0,
    icon: <PersonIcon />,
    label: "Профиль",
    route: Route.profile,
    roles: [...Object.values(Role)],
  },
];
export const navbarItemsData: Omit<NavBarItem, "id">[] = [
  {
    icon: <SummarizeIcon />,
    label: "Дневник",
    route: Route.diary,
    roles: [Role.PATIENT],
  },
  {
    icon: <LocalHospitalIcon />,
    label: "Доктор",
    route: Route.doctor,
    roles: [Role.PATIENT],
  },
  {
    icon: <FavoriteIcon />,
    label: "Рекомендации",
    route: Route.recommendations,
    roles: [Role.PATIENT],
  },
  {
    icon: <MonitorHeartIcon />,
    label: "Уведомления",
    route: Route.notifications,
    roles: [Role.DOCTOR],
  },
  {
    icon: <RecommendIcon />,
    label: "Назначения",
    route: Route.healthStates,
    roles: [Role.DOCTOR],
  },
  {
    icon: <PeopleIcon />,
    label: "Пациенты",
    route: Route.patients,
    roles: [Role.DOCTOR],
  },
  {
    icon: <AlarmIcon />,
    label: "Цели",
    route: Route.goals,
    roles: [],
  },
  {
    icon: <TrendingUpIcon />,
    label: "Статистика",
    route: Route.statistics,
  },
];

```

```

    roles: [Role.PATIENT],
  },
  {
    icon: <ChatIcon />,
    label: "Диалоги",
    route: Route.dialogs,
    roles: [Role.PATIENT, Role.DOCTOR],
  },
  {
    icon: <FamilyRestroomIcon />,
    label: "Родственник",
    route: Route.relative,
    roles: [Role.PATIENT, Role.RELATIVE],
  },
  {
    icon: <InsightsIcon />,
    label: "Статистика родственника-пациента",
    route: Route.relativeStatistics,
    roles: [Role.RELATIVE],
  },
  {
    icon: <QueryStatsIcon />,
    label: "Статистика пациентов",
    route: Route.patientsStatistics,
    roles: [Role.DOCTOR],
  },
  {
    icon: <WysiwygIcon />,
    label: "Тематические материалы",
    route: Route.thematicMaterials,
    roles: [
      Role.PATIENT,
      Role.RELATIVE,
      Role.DOCTOR,
      Role.CONTENT_MAKER,
      Role.MODERATOR,
    ],
  },
  {
    icon: <PeopleIcon />,
    label: "Пользователи",
    route: Route.accounts,
    roles: [Role.ADMIN],
  },
  {
    icon: <ReviewsIcon />,
    label: "Отзывы",
    route: Route.reviews,
    roles: [Role.DOCTOR],
  },
];
export const bottomNavBarItems: NavBarItem[] = [
  {
    id: 0,
    icon: <LogoutIcon />,
    label: "Выход",
    route: Route.signOut,
    roles: [...Object.values(Role)],
  },
];
export const navBarItems: NavBarItem[] = [
  ...topNavbarItems,
  ...navbarItemsData,

```



```

    ...bottomNavBarItems,
  ].map((data, index) => ({
    ...data,
    id: index,
  }));

```

Navbar.tsx

```

import React from "react";
import {
  Box,
  Grid,
  ListItemButton,
  ListItemIcon,
  ListItemText,
  SwipeableDrawer,
} from "@mui/material";
import { isMobile } from "react-device-detect";
import { useWindowSize } from "hooks/useWindowSize";
import { NavBarItem, navBarItems } from "components/layout/Navbar/constants";
import { useLocation, useNavigate } from "react-router-dom";
import { User } from "firebase/types/collections.types";
import { GlobalState } from "store";
import { useSelector } from "react-redux";
import "./navbar.scss";
export const Navbar = ({
  isNavBarOpened,
  setIsNavBarOpened,
}): {
  isNavBarOpened: boolean;
  setIsNavBarOpened: React.Dispatch<boolean>;
} => {
  const [width] = useWindowSize();
  const navigate = useNavigate();
  const user = useSelector<GlobalState, User>(
    (state) => state.currentUser.user as User
  );
  const location = useLocation();
  const navBarItem = (navBarItem: NavBarItem) => (
    <ListItemButton
      key={navBarItem.id}
      onClick={() => {
        if (isMobile) {
          setIsNavBarOpened(false);
        }
        navigate(navBarItem.route);
      }}
      className="item-container"
      selected={location.pathname.startsWith(navBarItem.route)}
    >
      <ListItemIcon className="item-icon">{navBarItem.icon}</ListItemIcon>
      <ListItemText primary={navBarItem.label} className="item-text" />
    </ListItemButton>
  );
  const mapNavBarItems = (items: NavBarItem[]) =>
    items
      .filter((navBarItem) => navBarItem.roles.includes(user.role))
      .map(navBarItem);
  const list = (
    <Box
      sx={{
        width: isMobile ? 250 : (width / 12) * 2,

```

```

        minWidth: 170,
        marginTop: "60px",
      }}
      role="presentation"
      className="navbar-items-container"
    >
      {mapNavBarItems(navBarItems)}
    </Box>
  );
  return (
    <Grid item xs={isMobile ? undefined : 2}>
      <SwipeableDrawer
        variant={isMobile ? "temporary" : "permanent"}
        anchor="left"
        open={isNavBarOpened}
        onClose={() => setIsNavBarOpened(false)}
        onOpen={() => setIsNavBarOpened(true)}
      >
        {list}
      </SwipeableDrawer>
    </Grid>
  );
};

```

PageContainer.tsx

```

import React, { CSSProperties } from "react";
import { Grid } from "@mui/material";
import { isMobile } from "react-device-detect";
export const PageContainer = ({
  children,
  className,
  style,
}): {
  children?: React.ReactNode;
  className?: string;
  style?: CSSProperties;
}) => {
  return (
    <Grid
      item
      xs={isMobile ? 12 : 10}
      sx={{ marginTop: "60px", maxWidth: "initial" }}
    >
      <Grid item xs={12} className={className} style={style}>
        {children}
      </Grid>
    </Grid>
  );
};

```

AccountsPage.jsx

```

import React, { useState } from "react";
import { PageContainer } from "../layout";
import { useGeneralDataHook } from "../hooks/useGeneralDataHook";
import { firebaseRepositories } from "../firebase/firestore/data/repositories";
import { LoadingSpinner } from "../ui/LoadingSpinner";
import { Role } from "../firebase/firestore/types/collections.types";
import { AccountModal, UsersView } from "../elements";

```

```

import { useGeneralModalHandlers } from "../../hooks/useGeneralModalHandlers";
import { initializeApp } from "firebase/app";
import { firebaseConfig } from "../../firebase_config";
import {
  getAuth,
  signInWithEmailAndPassword,
  createUserWithEmailAndPassword,
  updateEmail,
  updatePassword,
  deleteUser,
} from "firebase/auth";
const adminFirebaseApp = initializeApp(firebaseConfig, "admin-console");
const adminAuth = getAuth(adminFirebaseApp);
export const AccountsPage = () => {
  const [isAccountUpdateLoading, setIsAccountUpdateLoading] = useState(false);
  const [isUsersLoading, users, refreshUsers] = useGeneralDataHook(async () => {
    return firebaseRepositories.users.getDocs();
  }, []);
  const [selectedAccount, setSelectedAccount] = useState(null);
  const [selectedRole, setSelectedRole] = useState(null);
  const [isAccountModalOpened, openAccountModal, closeAccountModal] =
    useGeneralModalHandlers({
      onOpen: (role, user = null) => {
        setSelectedAccount(user);
        setSelectedRole(role);
      },
      onClose: () => {
        setSelectedAccount(null);
        setSelectedRole(null);
      },
    });
  const updateAccount = async (updatedAccount, existingAccount = null) => {
    try {
      setIsAccountUpdateLoading(true);
      const userCredential = Boolean(existingAccount)
        ? await signInWithEmailAndPassword(
            adminAuth,
            existingAccount.email,
            existingAccount.password
          )
        : await createUserWithEmailAndPassword(
            adminAuth,
            updatedAccount.email,
            updatedAccount.password
          );
      if (existingAccount) {
        await updateEmail(userCredential.user, updatedAccount.email);
        await updatePassword(userCredential.user, updatedAccount.password);
      }
      await firebaseRepositories.users.updateDoc({
        ...updatedAccount,
        docId: userCredential.user.uid,
      });
      await adminAuth.signOut();
      await refreshUsers();
    } catch (e) {
      console.log(e);
    } finally {
      setIsAccountUpdateLoading(false);
    }
  };
  const deleteAccount = async (account) => {
    try {

```

```

setIsAccountUpdateLoading(true);
const userCredential = await signInWithEmailAndPassword(
  adminAuth,
  selectedAccount.email,
  selectedAccount.password
);
// todo side effects, update connected users (reviews, relativePatient, relative, patient, doctor)
// await deleteUser(userCredential.user);
// await firebaseRepositories.users.deleteDocById(account.docId)
await adminAuth.signOut();
await refreshUsers();
} catch (e) {
  console.log(e);
} finally {
  setIsAccountUpdateLoading(false);
}
};
return (
  <PageContainer
    style={{
      padding: "30px",
      display: "flex",
      flexDirection: "column",
      alignItems: "center",
      justifyContent: "center",
      gap: "20px",
      minHeight: "calc(100vh - 60px)",
      width: "100%",
    }}
  >
    {isUsersLoading || isAccountUpdateLoading ? (
      <LoadingSpinner />
    ) : (
      users && (
        <div
          style={{
            display: "flex",
            flexDirection: "column",
            alignItems: "center",
            gap: "30px",
            width: "100%",
          }}
        >
          {[Role.PATIENT, Role.DOCTOR, Role.RELATIVE, Role.ADMIN].map(
            (role) => (
              <UsersView
                key={role}
                role={role}
                users={users.filter((user) => user.role === role)}
                editAccount={openAccountModal}
                deleteAccount={deleteAccount}
              />
            )
          )}
        </div>
      )
    )}
    {selectedRole && (
      <AccountModal
        isOpen={isAccountModalOpened}
        onClose={closeAccountModal}
        submitAccount={updateAccount}
        selectedAccount={selectedAccount}

```

```

        role={selectedRole}
      />
    )}
  </PageContainer>
);
};

```

AccountModal.helpers.ts

```

import dayjs from "dayjs";
import { Role, User } from "firestore/types/collections.types";
export const createUser = (role: Role): User => {
  const baseUser = {
    docId: "",
    email: "",
    password: "",
    imageUrl: "",
    name: {
      first: "",
      last: "",
    },
    address: "",
    phone: "",
    role,
  };
  const patientPart = {
    diary: {
      dailyLogs: [],
      goals: [],
      diabetType: 1,
    },
    doctor: "",
  };
  const relativePart = {
    relativePatient: "",
  };
  const employeePart = {
    employee: {
      hiredAt: dayjs().toDate().toString(),
      reviews: [],
      salary: 0,
      description: "",
    },
  };
  return {
    ...baseUser,
    ...(role === Role.PATIENT ? patientPart : {}),
    ...(role === Role.RELATIVE ? relativePart : {}),
    ...([Role.DOCTOR, Role.CONTENT_MAKER, Role.MODERATOR, Role.ADMIN].includes(
      role
    )
      ? employeePart
      : {}),
  };
};

```

AccountModal.jsx

```

import React, { useCallback, useEffect, useState } from "react";
import {

```

```

    Button,
    Dialog,
    DialogActions,
    DialogContent,
    DialogTitle,
    TextField,
  } from "@mui/material";
import { LoadingSpinner } from "components/ui/LoadingSpinner";
import { deepCopy } from "deep-copy-ts";
import { isMobile } from "react-device-detect";
import { convertRoleToRussian } from "firebase/converters";
import { createUser } from "../AccountModal.helpers";
import { Role } from "firebase/types/collections.types";
import { formatDate } from "../helpers/helpers";
import "../account-modal.scss";
export const AccountModal = ({
  isOpen,
  onClose,
  submitAccount,
  selectedAccount,
  role,
}) => {
  const getAccount = useCallback(
    () => (selectedAccount ? deepCopy(selectedAccount) : createUser(role)),
    [selectedAccount, role]
  );
  const [account, setAccount] = useState(getAccount());
  const [isLoading, setIsLoading] = useState(false);
  const resetAccount = useCallback(() => {
    setAccount(getAccount());
  }, [setAccount, getAccount]);
  const handleFieldChange = (e) => {
    const { name, value } = e.target;
    setAccount((prevData) => ({
      ...prevData,
      [name]: value,
    }));
  };
  const handleSubmitAccount = async (event) => {
    event.preventDefault();
    setIsLoading(true);
    await submitAccount(account, selectedAccount);
    setIsLoading(false);
    onClose();
  };
  useEffect(() => {
    resetAccount();
  }, [selectedAccount, isOpen, resetAccount]);
  return (
    <Dialog open={isOpen} onClose={onClose} className="account-modal-dialog">
      <DialogTitle>
        {(selectedAccount ? "Редактирование" : "Добавление") +
          " пользователя [" +
          convertRoleToRussian(role) +
          "]"}
      </DialogTitle>
      <DialogContent
        className="account-modal-content"
        sx={{
          width: isMobile ? "300px" : "600px",
        }}
      >
        <form className="account-form" onSubmit={handleSubmitAccount}>

```

```

<TextField
  label="Роль"
  name="role"
  value={account.role}
  disabled={true}
/>
{selectedAccount && (
  <TextField
    label="Идентификатор пользователя"
    name="docId"
    value={account.docId}
    disabled={true}
  />
)}
<TextField
  label="Эл. почта"
  name="email"
  value={account.email}
  onChange={handleFieldChange}
  disabled={isLoading}
/>
<TextField
  label="Пароль"
  name="password"
  value={account.password}
  onChange={handleFieldChange}
  disabled={isLoading}
/>
<TextField
  label="Ссылка на изображение профиля"
  name="imageUrl"
  value={account.imageUrl}
  onChange={handleFieldChange}
  disabled={isLoading}
/>
<TextField
  label="Имя"
  value={account.name.first}
  onChange={({ target: { value } }) =>
    setAccount((prevAccount) => ({
      ...prevAccount,
      name: {
        ...prevAccount.name,
        first: value,
      },
    }))
  }
  disabled={isLoading}
/>
<TextField
  label="Фамилия"
  value={account.name.last}
  onChange={({ target: { value } }) =>
    setAccount((prevAccount) => ({
      ...prevAccount,
      name: {
        ...prevAccount.name,
        last: value,
      },
    }))
  }
  disabled={isLoading}
/>

```

```

<TextField
  label="Адрес"
  name="address"
  value={account.address}
  onChange={handleFieldChange}
  disabled={isLoading}
/>
<TextField
  label="Телефон"
  name="phone"
  value={account.phone}
  onChange={handleFieldChange}
  disabled={isLoading}
/>
{/* patient */}
{role === Role.PATIENT && (
  <
    <TextField
      label="Тип диабета"
      value={account.diary.diabetType}
      onChange={({ target: { value } }) =>
        setAccount((prevAccount) => ({
          ...prevAccount,
          diary: {
            ...prevAccount.diary,
            diabetType: isNaN(+value) ? 0 : +value,
          },
        }))
      }
      disabled={isLoading}
    />
  </>
)}
{/* relative */}
{role === Role.RELATIVE && (
  <
    <TextField
      label="Код приглашения пациента родственника"
      name="relativePatient"
      value={account.relativePatient}
      onChange={handleFieldChange}
      disabled={isLoading}
    />
  </>
)}
{/* employee */}
{[
  Role.DOCTOR,
  Role.CONTENT_MAKER,
  Role.MODERATOR,
  Role.ADMIN,
].includes(role) && (
  <
    <TextField
      label="Дата найма"
      name="hiredAt"
      value={formatDate(account.employee.hiredAt)}
      disabled={true}
    />
    <TextField
      multiline
      minRows={3}
      maxRows={4}

```



```

        label="Описание"
        value={account.employee.description}
        onChange={({ target: { value } }) =>
          setAccount((prevAccount) => ({
            ...prevAccount,
            employee: {
              ...prevAccount.employee,
              description: value,
            },
          })))
      }
      disabled={isLoading}
      fullWidth
    />
    <TextField
      label="Зарплата"
      value={account.employee.salary}
      onChange={({ target: { value } }) =>
        setAccount((prevAccount) => ({
          ...prevAccount,
          employee: {
            ...prevAccount.employee,
            salary: isNaN(+value) ? 0 : +value,
          },
        })))
      }
      disabled={isLoading}
    />
  </>
)}
<DialogActions className="controls">
  <Button onClick={onClose} disabled={isLoading} variant="outlined">
    Отмена
  </Button>
  {isLoading ? (
    <LoadingSpinner style={{ margin: "0", width: "100px" }} />
  ) : (
    <Button
      type="submit"
      variant="outlined"
      disabled={false} // todo
      autoFocus
    >
      {selectedAccount ? "Изменить" : "Добавить"}
    </Button>
  )}
</DialogActions>
</form>
</DialogContent>
</Dialog>
);
};

```

UsersView.jsx

```

import React from "react";
import {
  Button,
  Table,
  TableBody,
  TableCell,
  TableContainer,

```

```

    TableHead,
    TableRow,
  } from "@mui/material";
import Typography from "@mui/material/Typography";
import {
  convertRoleToRussian,
  convertRoleToRussianPlural,
} from "firebase/firestore/converter/role/role.converter";
import { getUserFullName } from "../../../../../firebase/helpers";
import { CardContainer } from "../../../../../ui/CardContainer";
import { Role } from "../../../../../firebase/types/collections.types";
import Avatar from "@mui/material/Avatar";
export const UsersView = ({ role, users, editAccount, deleteAccount }) => {
  return (
    <div style={{ display: "flex", width: "100%" }}>
      <TableContainer component={CardContainer}>
        <Typography variant="h5" sx={{ textAlign: "center" }}>
          {convertRoleToRussianPlural(role)}
        </Typography>
        <div style={{ display: "flex", justifyContent: "center" }}>
          <Button
            color="success"
            variant="contained"
            sx={{ fontSize: "11px", fontWeight: "bold" }}
            onClick={() => editAccount(role)}
          >
            Добавить
          </Button>
        </div>
      </div>
      <Table>
        <TableHead>
          <TableRow>
            <TableCell>{convertRoleToRussian(role)}</TableCell>
            <TableCell>Эл. почта</TableCell>
            <TableCell>Телефон</TableCell>
            <TableCell>Адрес</TableCell>
            <TableCell>{role === Role.PATIENT && <TableCell>Тип диабета</TableCell>}
            {
              Role.DOCTOR,
              Role.CONTENT_MAKER,
              Role.MODERATOR,
              Role.ADMIN,
            ].includes(role) && <TableCell>Зарплата</TableCell>
            <TableCell>Управление</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {users.map((user) => (
            <TableRow key={user.docId}>
              <TableCell>
                <div
                  style={{
                    display: "flex",
                    gap: "10px",
                    alignItems: "center",
                  }}
                >
                  <Avatar src={user.imageUrl} alt={getUserFullName(user)} />
                  <Typography>{getUserFullName(user)}</Typography>
                </div>
              </TableCell>
              <TableCell>{user.email}</TableCell>
              <TableCell>{user.phone}</TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </div>
  );
};

```

```

<TableCell>{user.address}</TableCell>
{role === Role.PATIENT && (
  <TableCell>{user.diary.diabetType}</TableCell>
)}
{[
  Role.DOCTOR,
  Role.CONTENT_MAKER,
  Role.MODERATOR,
  Role.ADMIN,
].includes(role) && (
  <TableCell>{user.employee.salary} ${</TableCell>
)}
<TableCell>
  <div style={{ display: "flex", gap: "5px" }}>
    <Button
      color="warning"
      variant="contained"
      sx={{ fontSize: "11px", fontWeight: "bold" }}
      onClick={() => editAccount(role, user)}
    >
      Редактировать
    </Button>
    <Button
      color="error"
      variant="contained"
      sx={{ fontSize: "11px", fontWeight: "bold" }}
      onClick={() => deleteAccount(role, user)}
      disabled={true} // todo side effects
    >
      Удалить
    </Button>
  </div>
</TableCell>
</TableRow>
  )}
</TableBody>
</Table>
</TableContainer>
</div>
);
};

```

DialogsPage.tsx

```

import React, { SyntheticEvent, useCallback, useEffect, useState } from "react";
import { PageContainer } from "components/layout";
import { useDialogsData } from "components/pages/Dialogs/hooks/useDialogsData";
import { getUserSelector } from "store/selectors";
import { useSelector } from "react-redux";
import { LoadingSpinner } from "components/ui/LoadingSpinner";
import {
  Channel,
  ChannelHeader,
  Chat,
  MessageInput,
  MessageList,
  Thread,
  Window,
} from "stream-chat-react";
import "stream-chat-react/dist/css/v2/index.css";
import { i18nInstance, streamClient } from "containers/App/App";
import useAsyncEffect from "use-async-effect";

```

```

import { Role, UserInfo } from "firebase/types/collections.types";
import { Autocomplete, TextField } from "@mui/material";
import { getUserFullName } from "firebase/helpers";
import { firebaseRepositories } from "firebase/data/repositories";
import { where } from "firebase/firestore";
import Box from "@mui/material/Box";
import Avatar from "@mui/material/Avatar";
import * as emailjs from "emailjs/browser";
import "../dialogs.scss";
export const DialogsPage = () => {
  const currentUser = useSelector(getUserSelector);
  // dialogs
  const [dialogsUsers, setDialogsUsers] = useState<UserInfo[] | null>(null);
  const [selectedUser, setSelectedUser] = useState<UserInfo | null>(null);
  const { isLoading, channel } = useDialogsData({
    currentUser,
    dialogsUsers,
    selectedUser,
  });
  // effects
  // fetch dialogs on initial load
  useAsyncEffect(async () => {
    if (currentUser.role === Role.PATIENT && !currentUser.doctor) {
      setDialogsUsers([]);
      return;
    }
    const filter =
      currentUser.role === Role.PATIENT
        ? where("docId", "==", currentUser.doctor)
        : where("doctor", "==", currentUser.docId);
    const newDialogsUsers = await firebaseRepositories.users.getDocs(filter);
    setDialogsUsers(newDialogsUsers);
  }, [currentUser]);
  // set first user as selected if user is patient
  useEffect(() => {
    if (dialogsUsers && dialogsUsers.length > 0) {
      setSelectedUser(dialogsUsers[0]);
    }
  }, [dialogsUsers, currentUser]);
  // send emails when a new message occurs
  useEffect(() => {
    return streamClient.on("message.new", async (event) => {
      if (!selectedUser || !dialogsUsers || !dialogsUsers.length) {
        return;
      }
      if (event?.user?.id === currentUser.docId && event?.message?.text) {
        const message = event.message.text;
        const receiverId = selectedUser.docId;
        const receiverUserInfo = dialogsUsers.find(
          (user) => user.docId === receiverId
        ) as UserInfo;
        await emailjs.send("service_oxiflh8", "template_4mhfpd6", {
          to_email: receiverUserInfo.email,
          from_name: getUserFullName(currentUser),
          to_name: getUserFullName(receiverUserInfo),
          message,
        });
      }
    }).unsubscribe;
  }, [selectedUser, dialogsUsers, currentUser]);
  // handlers
  // handles dialog selection
  const handleSelectedDialogChange = (

```

```

event: SyntheticEvent<Element, Event>,
option: unknown
) => {
  if (!dialogsUsers) {
    return;
  }
  const castedOption = option as { value: string; label: string } | null;
  setSelectedUser(
    dialogsUsers.find((user) => user.docId === castedOption?.value) ?? null
  );
};
const getUserOption = useCallback((user: UserInfo) => {
  return {
    value: user.docId,
    label: getUserFullName(user),
    imageUrl: user.imageUrl,
  };
}, []);
return (
  <PageContainer className="dialogs-page-container">
    <div className="dialogs-container">
      {!dialogsUsers ? (
        <LoadingSpinner />
      ) : (
        <>
          {!dialogsUsers.length && (
            <div className="no-dialog-users">
              {currentUser.role === Role.DOCTOR && "Нет пациентов"}
              {currentUser.role === Role.PATIENT &&
                "Доктор не выбран, перейдите на вкладку [Доктор]"}
            </div>
          )}
          {dialogsUsers.length > 0 && currentUser.role === Role.DOCTOR && (
            <Autocomplete
              id="dialog-select"
              className="dialogs-page-select"
              value={selectedUser ? getUserOption(selectedUser) : null}
              onChange={handleSelectedDialogChange}
              options={dialogsUsers.map(getUserOption)}
              renderInput={(params) => (
                <TextField
                  {...params}
                  label="Выберите диалог"
                  inputProps={{
                    ...params.inputProps,
                  }}
                />
              )}
              blurOnSelect
              noOptionsText="Нет пациентов с таким именем."
              renderOption={(props, option) => (
                <Box
                  component="li"
                  sx={{ "& > img": { mr: 2, flexShrink: 0 } }}
                  {...props}
                >
                  <Avatar
                    src={option.imageUrl}
                    sx={{
                      width: "30px",
                      height: "30px",
                      marginRight: "20px",
                    }}

```

```

        />
        {option.label}
      </Box>
    )}
  />
)}
</>
)}
{selectedUser && dialogsUsers &&
dialogsUsers.length > 0 &&
// "!channel" check only because of typescript
(isLoading || !channel ? (
  <LoadingSpinner />
) : (
  <Chat
    client={streamClient}
    theme="str-chat__theme-light"
    i18nInstance={i18nInstance}
    customClasses={{
      chatContainer:
        currentUser.role === Role.DOCTOR
        ? "str-chat__container doctor-view"
        : "str-chat__container",
    }}
  >
    <Channel channel={channel}>
      <Window>
        <ChannelHeader />
        <MessageList />
        <MessageInput />
      </Window>
      <Thread />
    </Channel>
  </Chat>
)}
</div>
</PageContainer>
);
};

```

useDialogsData.ts

```

import { Role, User, UserInfo } from "firebase/types/collections.types";
import { Channel } from "stream-chat";
import useAsyncEffect from "use-async-effect";
import { useState } from "react";
import { getUserFullName } from "firebase/helpers";
import { NotificationManager } from "react-notifications";
import { streamClient } from "containers/App/App";
const getStreamUserData = (userInfo: UserInfo) => ({
  id: userInfo.docId,
  name: `${userInfo.name.first} ${userInfo.name.last}`,
  image: userInfo.imageUrl,
});
const getStreamUserToken = (uid: string) => streamClient.devToken(uid);
const connectStreamUser = async (userInfo: UserInfo) =>
  streamClient.connectUser(
    getStreamUserData(userInfo),
    getStreamUserToken(userInfo.docId)
  );
export const useDialogsData = ({
  currentUser,

```

```

dialogsUsers,
selectedUser,
}: {
  currentUser: User;
  dialogsUsers: UserInfo[] | null;
  selectedUser: UserInfo | null;
}) => {
  const [channel, setChannel] = useState<Channel | null>(null);
  const [isLoading, setIsLoading] = useState<boolean>(false);
  useAsyncEffect(async () => {
    if (!selectedUser || !dialogsUsers || !dialogsUsers.length) {
      return;
    }
    try {
      setIsLoading(true);
      const patient =
        currentUser.role === Role.PATIENT ? currentUser : selectedUser;
      const doctor =
        currentUser.role === Role.PATIENT ? selectedUser : currentUser;
      const channelId = `${doctor.docId}_${patient.docId}`;
      const channelName =
        currentUser.role === Role.PATIENT
          ? getUserFullName({ name: doctor.name })
          : getUserFullName({ name: patient.name });
      const channelImage = selectedUser.imageUrl;
      await connectStreamUser(currentUser);
      const newChannel = streamClient.channel("messaging", channelId);
      await newChannel.watch();
      await newChannel.update({
        name: channelName,
        image: channelImage,
        members: [currentUser.docId, selectedUser.docId],
      });
      setChannel(newChannel);
    } catch (e) {
      console.log(e);
      NotificationManager.error(
        "Загрузке данных для диалогов",
        "Ошибка данных диалогов"
      );
    } finally {
      setIsLoading(false);
    }
  }, [currentUser, dialogsUsers, selectedUser]);
  return {
    isLoading,
    channel,
  };
};

```

DailyLogDataView.tsx

```

import React from "react";
import dayjs from "dayjs";
import { EditButton } from "components/ui/EditButton";
import { CardContainer } from "components/ui/CardContainer";
import { DailyLogData } from "firestore/converters";
import "./dailylog-data-view.scss";
export const DailyLogDataView = ({
  dailyLogData,
  handleEditButtonClick,
  isEditable,

```

```

}: {
  dailyLogData: DailyLogData;
  handleEditButtonClick: () => void;
  isEditable: boolean;
}) => {
  const getDisplayValue = (propName: keyof DailyLogData): string | number =>
    (+dailyLogData[propName] ? (+dailyLogData[propName]).toFixed(2) : "-") as
    | string
    | number;
  return (
    <CardContainer className="diary-page-content__info-container">
      <div className="diary-page-content__info-container__title">
        <h2>`${dayjs(dailyLogData.createdAt).format(
          "HH:mm, DD MMMM YYYY"
        )}`</h2>
        {isEditable && <EditButton onClick={handleEditButtonClick} />}
      </div>
      <p>Уровень сахара: {getDisplayValue("sugarLevel")} ммоль/л</p>
      <p>Пульс: {getDisplayValue("pulse")} уд/мин</p>
      <p>
        Давление: {getDisplayValue("systolic")}/{getDisplayValue("diastolic")} {" "}
        ммрт/ммрт
      </p>
      <p>Калории: {getDisplayValue("total")} ккал</p>
      <p>Вес: {getDisplayValue("weight")} кг</p>
      <p>Температура: {getDisplayValue("temperature")} °C</p>
    </CardContainer>
  );
};

```

NoteModal.tsx

```

import React, { useCallback, useEffect, useState } from "react";
import {
  Button,
  Dialog,
  DialogActions,
  DialogContent,
  DialogTitle,
  TextField,
} from "@mui/material";
import { Note } from "firestore/types/collections.types";
import { LoadingSpinner } from "components/ui/LoadingSpinner";
import { deepCopy } from "deep-copy-ts";
import { formatDate } from "helpers/helpers";
import { isMobile } from "react-device-detect";
import dayjs from "dayjs";
import "../note-modal.scss";
export const NoteModal = ({
  isOpen,
  onClose,
  submitNote,
  selectedNote,
}): {
  isOpen: boolean;
  onClose: () => void;
  submitNote: (note: Note) => Promise<void>;
  selectedNote: Note | null;
} => {
  const [note, setNote] = useState<Note>(getNote());
  const [isLoading, setIsLoading] = useState<boolean>(false);
  const resetNote = useCallback(() => {

```



```

    setNote(getNote());
  }, [setNote, getNote]);
const handleSubmitNote = async (event: React.FormEvent<HTMLFormElement>) => {
  event.preventDefault();
  setIsLoading(true);
  await submitNote({
    ...note,
    createdAt: dayjs().toDate().toString(),
  });
  setIsLoading(false);
  onClose();
};
useEffect(() => {
  resetNote();
}, [selectedNote, isOpen, resetNote]);
return (
  <Dialog open={isOpen} onClose={onClose}>
    <DialogTitle>
      {(selectedNote ? "Редактирование" : "Добавление") + " заметки"}
    </DialogTitle>
    <DialogContent
      className="modal-content"
      sx={{
        width: isMobile ? "300px" : "600px",
      }}
    >
      <form className="note-form" onSubmit={handleSubmitNote}>
        <TextField
          label="Дата"
          name="createdAt"
          value={formatDate(note.createdAt)}
          disabled={true}
        />
        <TextField
          multiline
          minRows={3}
          maxRows={3}
          label="Текст"
          name="content"
          value={note.content}
          onChange={handleFieldChange}
          disabled={isLoading}
        />
        <DialogActions className="controls">
          <Button onClick={onClose} disabled={isLoading} variant="outlined">
            Отмена
          </Button>
          {isLoading ? ( <LoadingSpinner style={{ margin: "0", width: "100px" }} />) : (
            <Button
              type="submit"
              variant="outlined"
              disabled={selectedNote?.content === note.content}
              autoFocus
            >
              {selectedNote ? "Изменить" : "Добавить"}
            </Button>
          )}
        </DialogActions>
      </form>
    </DialogContent>
  </Dialog>
);
};

```

ПРИЛОЖЕНИЕ Б

(справочное)

Каталог функций программного обеспечения

Таблица Б.1 – Каталог функций программного обеспечения

Код функций	Наименование (содержание) функций	Объем функции строк исходного кода (LOC)	
		По каталогу (V_0)	уточненный (V_y)
Ввод, анализ входной информации			
101	Организация ввода информации	130	100
102	Контроль, предварительная обработка и ввод информации	490	300
104	Обработка входного языка и формирование таблиц	1040	250
107	Организация ввода-вывода информации в интерактивном режиме	280	150
109	Управление вводом-выводом	1970	720
Формирование, ведение и обслуживание базы данных			
201	Генерация структуры базы данных	3500	380
202	Формирование базы данных	1980	200
203	Обработка наборов и записей базы данных	2370	900
206	Манипулирование данными	7860	600
207	Организация поиска и поиск в базе данных	4720	200
209	Загрузки базы данных	2360	400
Формирование и обработка файлов			
301	Формирование последовательного файла	590	200
303	Обработка файлов	1050	280
305	Формирование файлов	2130	500
Управление ПО, компонентами ПО и внешними устройствами			
506	Обработка ошибочных сбойных ситуаций	1540	420
507	Обеспечение интерфейса между компонентами	1680	630
Расчетные задачи, формирование и вывод на внешние носители документов сложной формы и файлов			
707	Графический вывод результатов	420	230
	Итого:	34080	6460

ПРИЛОЖЕНИЕ В

(справочное)

Расчет общей трудоемкости разработки

Таблица В.1 – Расчет общей трудоемкости разработки программного обеспечения

Показатели	Стадии разработки					Итого
	ТЗ	ЭП	ТП	РП	ВН	
1	2	3	4	5	6	7
Общий объем ПО (V_0), кол-во строк LOC	-	-	-	-	-	34080
Общий уточненный объем ПО (V_y), кол-во строк LOC	-	-	-	-	-	6460
Категория сложности разрабатываемого ПО	-	-	-	-	-	3
Нормативная трудоемкость разработки ПО (T_n), чел./дн.	-	-	-	-	-	263
Коэффициент повышения сложности ПО (K_c)	1,07	1,07	1,07	1,07	1,07	-
Коэффициент, учитывающий новизну ПО (K_n)	0,63	0,63	0,63	0,63	0,63	-
Коэффициент, учитывающий степень использования стандартных модулей (K_t)	-	-	-	0,65	-	-
Коэффициент, учитывающий средства разработки ПО (K_{yp})	1,0	1,0	1,0	1,0	1,0	-
Коэффициенты удельных весов трудоемкости стадий разработки ПО ($K_{тз}$, $K_{эп}$, $K_{тп}$, $K_{рп}$, $K_{вн}$)	0,08	0,19	0,28	0,34	0,11	1,0
Распределить нормативной трудоемкости ПО по стадиям, чел.-дн. (V_0)	21	50	74	89	29	263
Распределение скорректированной (с учетом K_c , K_n , K_t , K_{yp}) трудоемкости ПО по стадиям, чел./дн.	14	34	50	39	20	-
Общая трудоемкость разработки ПО (T_0), чел./дн.	-	-	-	-	-	157

ПРИЛОЖЕНИЕ Г

(справочное)

Параметры для расчета производственных затрат

Таблица Г.1 – Параметры для расчета производственных затрат на разработку программного обеспечения

Параметр	Единица измерения	Значение
Базовая ставка	руб.	228
Разряд разработчика	—	8
Тарифный коэффициент 8-го разряда	—	1,57
Коэффициент $K_{ув}$	—	1,8
Норматив отчислений на доп. зарплату разработчиков ($H_{доп}$)	%	15
Численность обслуживающего персонала	чел.	1
Разряд обслуживающего персонала	—	8
Средняя годовая ставка арендных платежей ($C_{ар}$)	руб./м ²	198
Площадь помещения (S)	м ²	10
Количество ПЭВМ ($Q_{эвм}$)	шт.	1
Затраты на приобретение единицы ПЭВМ	руб.	2000
Стоимость одного кВт-часа электроэнергии ($C_{эл}$)	руб.	0,3
Затраты на технологию ($З_{тех}$)	руб.	—
Норматив общепроизводственных затрат ($H_{доп}$)	%	5
Норматив непроизводственных затрат ($H_{непр}$)	%	3

ПРИЛОЖЕНИЕ Д

(справочное)

Расчет суммарных затрат на разработку

Таблица Д.1 – Расчет суммарных затрат на разработку программного обеспечения

№	Статья затрат	Итого
1	Затраты на оплату труда разработчиков ($Z_{тр}$), руб.	15502,74
1.1	Основная заработная плата разработчиков	10015,34
1.2	Дополнительная заработная плата разработчиков	1502,30
1.3	Отчисления от основной и дополнительной ЗП	3985,10
2	Затраты машинного времени ($Z_{мв}$), руб.	3606,4
2.1	Стоимость машино-часа, руб/ч	9,80
	Затраты на заработную плату обслуживающего персонала	13830,06
	Годовые затраты на аренду помещения	1980
	Сумма годовых амортизационных отчислений, руб.	448
	Стоимость электроэнергии, потребляемой за год	219,77
	Действительный годовой фонд времени работы ПЭВМ, дн.	1713,6
	Затраты на материалы	22,4
	Затраты на текущий и профилактический ремонт, руб.	179,2
	Прочие затраты, связанные с эксплуатацией ЭВМ, руб.	112
2.2	Машинное время ЭВМ, ч.	464
3	Затраты на изготовление эталонного экземпляра ($Z_{эт}$), руб.	0
4	Затраты на технологию ($Z_{тех}$), руб.	0
5	Затраты на материалы ($Z_{мат}$), руб.	75
6	Общепроизводственные затраты ($Z_{общ.пр}$)	446,74
7	Непроизводственные (коммерческие) затраты ($Z_{непр}$)	268,04
8	Суммарные затраты на разработку ПО (Z_p)	18226,24

ПРИЛОЖЕНИЕ Е

(справочное)

Технико-экономические показатели проекта

Таблица Е.1 – Технико-экономические показатели проекта

№ п/п	Наименование показателя	Единица измерения	Базовый вариант	Проектный вариант
Показатели затрат на разработки				
1	Общая трудоемкость разработки ПО	чел.-дн.	×	157
2	Затраты на разработку ПО	руб.	×	18226,24
2.1	Затраты на оплату труда разработчиков	руб.	×	15502,74
2.2	Затраты машинного времени	руб.	×	3606,4
2.3	Затраты на материалы	руб.	×	75
2.4	Общепроизводственные затраты	руб.	×	446,74
2.5	Непроизводственные затраты	руб.	×	268,04
Показатели стоимости				
3	Отпускная цена ПП с НДС	руб.	×	28432,93
4	Розничная цена ПП	руб.	×	31276,22
Показатели экономической эффективности				
5	Рентабельность затрат	%	×	107,83
6	Простой срок окупаемости проекта	лет	×	0,93
7	Годовой экономический эффект	руб.	×	33723,78