МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка мобильного приложения для расчета стоимости коммунальных услуг»

Обучающийся Д.С. Меркулов _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.п.н., доцент, О.В. Оськина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.ф.н., доцент, М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Разработка мобильного приложения для расчета стоимости коммунальных услуг».

Актуальность проведенного исследования объясняется несколькими факторами: ежегодным повышением тарифов на коммунальные услуги, сложностью самостоятельных расчетов для рядовых потребителей, а также возрастающей потребностью в цифровых сервисах, которые упрощают взаимодействие граждан с жилищно-коммунальным хозяйством.

Целью работы является реализация мобильного приложения для расчета стоимости коммунальных услуг.

В процессе разработки проводился сравнительный анализ существующих аналогов, формулировались технические требования, был разработан пользовательский интерфейс и реализованы все необходимые функции с помощью оптимальных технологий.

Итогом исследования стало работоспособное мобильное приложение, прошедшее цикл тестирования. Программный продукт демонстрирует корректность расчетов, стабильную работу и обладает потенциалом для дальнейшего развития, включая возможность интеграции с базами данных поставщиков коммунальных услуг и интеграцию систем оплаты.

Объем работы составляет 64 страницы, включая 36 иллюстраций и 3 таблицы, с традиционной структурой, содержащей введение, три аналитических главы, заключение, библиографический список и приложения.

Abstract

The title of the graduation work is *Developing a mobile application for calculating the housing and utility cost*.

The graduation work consists of an introduction, 3 parts, 36 figures, 3 tables, a conclusion and list of 25 references.

The aim of this graduation work is to develop a mobile application that simplifies housing and utility cost calculations for end-users.

The object of the research is the process of calculating the housing and utility cost.

The subject of the study is the development of a digital solution to automate this process.

The key issue of the graduation work is the lack of user-friendly tools to help consumers navigate through the complicated utility charges assessment systems.

The graduation work may be divided into several logically connected parts which are: analysis, design, and implementation.

The first part conducts a comparative analysis of the existing applications, and considers their advantages and disadvantages, as well as the users' needs and requirements.

The second part describes the process of application design: prototyping, architecture development, and technology selection.

The third part presents the implementation process: development of the core functionality, testing, and optimization.

In conclusion, it should be noted that the developed application ensures accurate calculations, user-friendly interface, and has the potential for further development.

The work is of practical value for consumers, housing and utility services, and developers of digital services.

Оглавление

Введение

В современном мире, где темп жизни постоянно ускоряется, важность доступа к удобным и эффективным инструментам для управления финансами становится все более актуальной. С развитием информационных технологий появилась возможность существенно упростить процессы расчета и контроля коммунальных платежей. Одним из направлений такого улучшения является разработка мобильного приложения для расчета стоимости коммунальных услуг, что позволяет пользователям быстро и точно вычислять суммы платежей, избегая ошибок и сложностей, связанных с ручными расчетами.

Целью данной выпускной квалификационной работы является разработка мобильного приложения для расчета стоимости коммунальных услуг, которое будет способствовать оптимизации управления коммунальными платежами. В рамках работы будут рассмотрены существующие решения в данной области, проведен анализ их недостатков и преимуществ, а также разработано собственное программное обеспечение, учитывающее особенности тарифов и расчетов.

Объектом исследования является процесс расчета стоимости коммунальных услуг. Предметом исследования является разработка, тестирование и внедрение мобильного приложения для автоматизированного расчета коммунальных платежей.

Для достижения поставленной цели предусмотрено решение следующих задач:

- Исследование существующих решений для расчета коммунальных услуг и анализ их эффективности;

- определение требований к разрабатываемому приложению, учитывая потребности пользователей;

- разработка архитектуры мобильного приложения;

- изучить технологии для разработки системы и найти наиболее подходящее решение;

- программная реализация мобильного приложения для расчета стоимости коммунальных услуг;

- провести тестирование приложения.

Актуальность работы обусловлена растущей потребностью населения в цифровых инструментах для управления личными финансами и оптимизации коммунальных расчетов с помощью современных информационных технологий. Разработанное приложение позволит сделать процесс расчета коммунальных услуг более точным и удобным, что повысит комфорт пользователей и снизит вероятность ошибок.

Результатом работы будет разработанное мобильное приложение для расчета стоимости коммунальных услуг.

Данная работа состоит из введения, трех разделов, заключения и списка используемой литературы.

Первый раздел посвящен анализу предметной области и постановке целей и задач для разработки мобильного приложения, а также концептуальному моделированию процессов расчета коммунальных услуг.

Второй раздел посвящен проектированию архитектуры мобильного приложения.

Третий раздел посвящен разработке и тестированию готового программного продукта.

В заключении содержатся результаты разработки мобильного приложения.

Работа включает: 64 страницу, 36 рисунков, 3 таблицы и 25 источников.

Глава 1 Постановка задачи на разработку мобильного приложения для расчета стоимости коммунальных услуг

1.1 Описание организации.

ООО «Квартплата 24» представляет собой специализированного провайдера в сфере жилищно-коммунального хозяйства (ЖКХ), деятельность которого направлена на автоматизацию процессов управления платежами и переводами между поставщиками и потребителями коммунальных услуг. Основной задачей предприятия является разработка и внедрение цифровых решений, обеспечивающих удобство, прозрачность и надежность расчетных операций в ЖКХ.

Основной целью «Квартплата 24» является создание эффективной цифровой экосистемы для управления коммунальными платежами. Предприятие ориентировано на внедрение автоматизированных решений, минимизирующих влияние человеческого фактора, снижающих операционные затраты и повышающих прозрачность расчетов между всеми участниками рынка.

ООО «Квартплата 24» планирует дальнейшее расширение спектра цифровых услуг, включая:

- разработку аналитических инструментов для прогнозирования платежных потоков;

- расширение возможностей платформы для подключения новых категорий пользователей (например, малых и средних предприятий, управляющих компаний нового формата).

В основе деятельности «Квартплата 24» лежат принципы цифровизации и автоматизации финансовых процессов в ЖКХ. Ключевыми направлениями являются:

- внедрение автоматизированных расчетных систем;

- интеграция с банковскими и платежными платформами;

- повышение уровня информационной безопасности;

- создание интуитивно понятного интерфейса для пользователей.

Более подробное описание экосистемы «Квартплата 24» показано на рисунке схемы 1.

Рисунок 1 – Цифровая экосистема «Квартплата 24»

Предприятие функционирует на основе развитой технологической платформы, включающей серверные мощности, облачные технологии и

современные программные решения для обработки транзакций. Основными компонентами инфраструктуры являются:

- Центр обработки данных (ЦОД), обеспечивающий бесперебойную работу систем;
- личный кабинет плательщика;
- собственно-разработанная платежная платформа;
- сервисы для работы с клиентами;
- интеграционные API для взаимодействия с банковскими системами.

1.2 Значение мобильного приложения для расчета стоимости коммунальных платежей

Актуальность разработки мобильного приложения для расчета стоимости коммунальных услуг обусловлена возрастающими потребностями современных потребителей в удобных и прозрачных инструментах управления жилищно-коммунальными расходами. В условиях постоянного изменения тарифов, усложнения формул расчета и многообразия льготных программ, рядовые жильцы сталкиваются со значительными трудностями при самостоятельном расчете и проверке начислений за коммунальные услуги. Как указывает И. Соммервиль, «разработка пользовательских систем требует учета динамично меняющихся требований» [18]. Традиционные бумажные квитанции и стационарные сервисы расчетов уже не отвечают требованиям мобильности и оперативности предъявляемым современным пользователям.

В качестве примера компании «Квартплата 24» был выявлен ряд системных проблем, с которыми сталкиваются потребители. Основная сложность заключается в отсутствии доступного инструмента, который позволит быстро и точно рассчитать стоимость коммунальных услуг с учетом индивидуальных параметров. Большинство жильцов вынуждены либо слепо доверять расчетам управляющих компаний, либо тратить значительное время на

ручные вычисления по формулам, что часто приводит к ошибкам и недопониманию. Особенно актуальна эта проблема для владельцев нескольких объектов недвижимости, которым приходится вести учет и оплату по каждому объекту отдельно.

Существующие способы решения обладают существенными недостатками. Большое количество мобильных приложения управляющих компаний позволяют оплачивать только один вид услуг и не поддерживают функцию учета дифференцированных тарифов на электроэнергию. Отдельная проблема – отсутствие возможности объединения нескольких объектов недвижимости в одном аккаунте, что вынуждает пользователей регистрировать множество личных кабинетов или постоянно переключаться между разными учетными записями. Традиционные бумажные квитанции не предоставляют возможности быстрого сравнения потребления по периодам, а веб-порталы часто не поддерживают мобильные устройства.

На основании проведенного анализа построена схема причинно-следственных связей (рисунок 2).

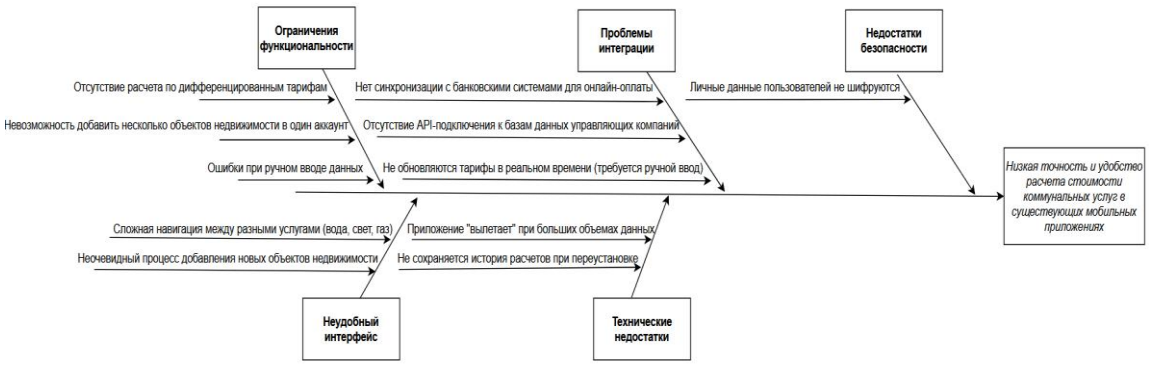


Рисунок 2 – Диаграмма причин и следствий низкой эффективности расчета стоимости коммунальных услуг

Диаграмма выявляет ключевые проблемы расчета коммунальных платежей: ошибки ручного ввода, отсутствие автоматизированных тарифов, сложности интеграции с УК и непрозрачность расчетов. Совокупность этих факторов обосновывает необходимость разработки мобильного приложения с автоматизированными расчетами и прозрачной системой учета.

1.3 Моделирование основных бизнес-процессов

Разработка мобильного приложения для расчета стоимости коммунальных услуг начинается с анализа и описания ключевых бизнес-процессов, подлежащих автоматизации. В контексте мобильного приложения для расчета коммунальных платежей основной задачей является создание удобного и надежного инструмента для управления расходами на ЖКУ – от ввода данных до формирования отчетов и осуществления платежей. Важную роль в этой процессе играет взаимодействие пользователя с системой, внутренняя логика расчетов и интеграция с внешними сервисами.

В приложении реализуются следующие ключевые процессы:

- интеграция с платежными системами. Реализована имитация передачи данных в расчетные центры и банковские сервисы для упрощения процедуры оплаты;

- расчет стоимости услуг. Приложение автоматически вычисляет сумму к оплате на основе введенных показаний, применяя актуальные тарифы;

- ввод и валидация данных о потреблении. Пользователь указывает текущие показания счетчиков, а система проверяет их корректность, сопоставляя с предыдущими значениями и тарифами;

- управление справочниками тарифов. Предусмотрены функции добавления, изменения и удаление данных тарифных планов;

- формирование платежных документов. Система генерирует квитанции в удобном формате (PDF, онлайн-просмотр) с возможностью скачивания.

Для наглядного отображения взаимодействия между пользователем, модулями приложения и внешними сервисами используется диаграмма потоков данных DFD уровня A0 (Рисунок 3). Диаграмма потоков данных, один из основных инструментов структурного анализа и проектирования информационных систем [23]. Она отражает ключевые процессы, их взаимосвязи и движение информации, что помогает обосновать архитектурные решения и структурировать описание системы. «Визуализация процессов в виде DFD помогает определить роли и зависимости в системе» [9], как подчеркивает К. Ларман.

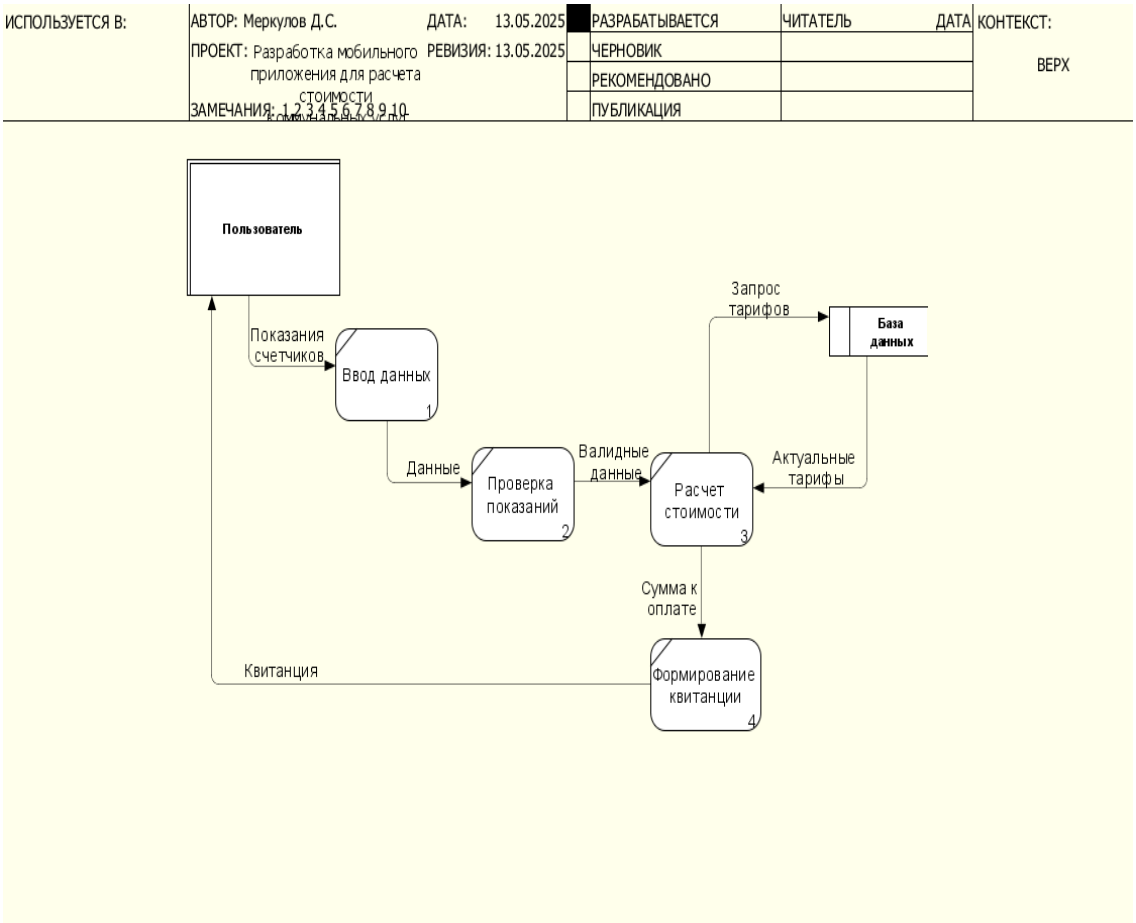


Рисунок 3 – Диаграмма потока данных A0 мобильного приложения для расчета стоимости коммунальных услуг

Диаграмма изображенная на рисунке 3 изображает внешнюю сущность – пользователя, который взаимодействует с системой. В системе реализованы четыре ключевых процесса: ввод данных (прием показаний от пользователя), проверка показаний (валидация введенных данных на корректность), расчет стоимости (определение суммы платежа на основе актуальных тарифов) и формирование квитанции (создание итогового платежного документа). Также диаграмма содержит одно хранилище данных – базу данных, в которой хранится вся необходимая информация для взаимодействия с системой.

1.4 Сравнительный анализ используемых аналогов

Перед тем, как начать проектирование нового ПО необходимо провести несколько исследований, одно из таких исследований – это сравнительный анализ аналогов. Мы будем оценивать несколько самых популярных мобильных приложений, основное внимание к сравниваемому ПО будет уделено сервису для расчета стоимости коммунальных услуг. Для проведения исследования были выбраны следующие платформы:

- Онлайн ЖКХ: Комплексный сервис для жителей многоквартирных домов, предоставляющий полный цикл услуг – от оплаты квитанций до подачи заявок в управляющие компании. Особенностью системы является интеграция с государственными информационными системами ЖКХ;

- Портал ЖКХ: Региональная платформа для жителей Иркутской области, которая сочетает в себе функции расчета коммунальных платежей с возможностью формирования детальных отчетов и получения уведомлений о будущих платежах;

- Квартплата Онлайн: Универсальное мобильное приложение для управления коммунальными платежами, оно позволяет контролировать несколько квартир на одном аккаунте. Отличается от остальных аналогов своим простым интерфейсом.

На таблице 1 будет проведено детальное сравнение функциональных возможностей рассмотренных сервисов, оценив их ключевые характеристики по шкале из 5 баллов, где 0 баллов – это минимальная оценка, а 5 – максимальная, анализ.

Таблица 1 – Сравнение аналогов по критериям

Наименование	Квартплата.Онлайн	Онлайн ЖКХ	Портал ЖКХ
Простота интерфейса	5	4	3
Отзывчивость приложения	5	5	3
Понятная навигация	4	2	3
Формирование отчетов	2	5	3
Скорость расчетов	3	4	5
Отображение стоимости услуг	2	3	4

В результате проведенного анализа было выявлено, что все приложения имеют ограничения:

- Сложная навигация, что приводит к перегрузке интерфейса вкладками;

- недостаточная функциональность, невозможность управлять несколькими объектами недвижимости в одном личном кабинете;

- отсутствие интуитивно понятного пользовательского интерфейса.

Выявленные недостатки существующих решений подтверждают необходимость разработки собственного мобильного приложения. Оно обеспечит простую в использовании навигацию без перегрузки вкладками, гибкость функционала, пользователь сможет добавлять и управлять несколькими объектами недвижимости, а также будет разработан максимально

простой интерфейс для пользователя. Несмотря на наличие аналогичных приложений, они не обладают полным функционалом, который удовлетворит все потребности пользователя. Подразумевается, что в будущем разработанное мобильное приложение будет внедрено в систему сервисов, что позволит ему взаимодействовать с другими системами и платформами.

1.5 Формирование требований для разрабатываемого приложения

При разработке практически любого приложения нужно составить список требований, которые необходимо учитывать в процессе разработки. Это позволяет точно и четко определить ключевые характеристики приложения и гарантировать, что готовый продукт будет соответствовать ожиданиям пользователей.

Как отмечается в методическом пособии, «диаграммы прецедентов описывают организацию поведения системы» [1]. В соответствии с этим подходом, для визуализации взаимодействий пользователя с разрабатываемым приложением была создана диаграмма прецедентов (рисунок 4)

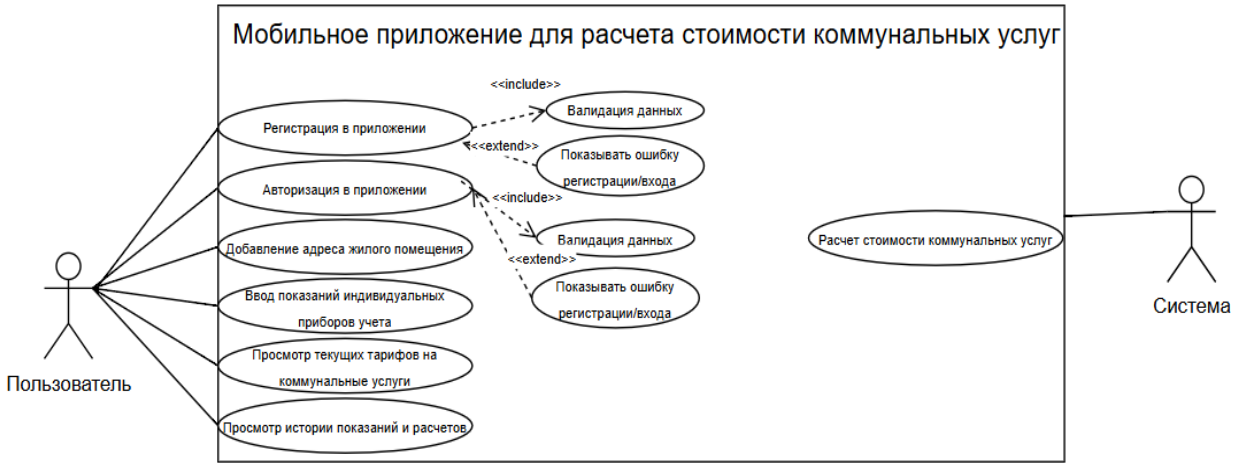


Рисунок 4 – Диаграмма прецедентов мобильного приложения

Диаграмма представленная на рисунке 3, где акторы, изображенные на ней – это пользователи или другие системы, взаимодействующие с данной системой, а прецеденты (варианты использования) – это сервисы, обеспечиваемые системой. Сплошными линиями на диаграмме использования являются отношения ассоциаций, олицетворяющие способность применения актором прецедента, пунктирные линии в свою очередь отображают вложенность операций (например, валидация данных).

Для структурирования требований к мобильному приложению была применена модель FURPS+, изображенная на таблице 2. Она охватывает основные аспекты разработки: функциональность, удобство интерфейса, надежность, производительность, а также дополнительные критерии, такие как безопасность и масштабируемость. Этот подход поможет четко структурировать требования, обеспечивая соответствие приложения потребностям пользователей и бизнес-логике.

Таблица 2 – FURPS+ категории требований к системе

Категория	Требования
Функциональность	- Ввод данных о квартире (адрес, ИНН УК, лицевой счет - Автоматический расчет стоимости услуг (вода, электричество) - Формирование квитанций с детализацией платежей
Удобство использования	- Простой и интуитивный интерфейс - Минимальное количество шагов для расчета - Адаптация под разные размеры экранов
Надежность	- Защита от некорректных значений - Использование JWT-токенов - Обработка ошибок и логирование

Продолжение таблицы 2

Производительность	- Быстрый расчет (≤ 1 секунды) - Эффективная работа на слабых устройствах - Оптимизация загрузки данных
Дополнительные требования	- Синхронизация данных между устройствами через аккаунт - Защита персональных данных - Экспорт квитанции в PDF

Таблица требований по модели FURPS+ охватывает ключевые аспекты приложения: от базового функционала до безопасности и производительности.

Вывод по первой главе

В результате проведенного исследования была обоснована необходимость разработки мобильного приложения для расчета стоимости коммунальных услуг, что обусловлено наличием существенных недостатков в существующих аналогах. Анализ предметной области позволил выявить ключевые проблемы пользователей: сложность в навигации в современных сервисах, отсутствие возможности управления несколькими объектами недвижимости в едином интерфейсе, а также недостаточную прозрачность расчетов. На основании проведенного сравнительного анализа существующих решений были определены конкурентные преимущества разрабатываемого приложения, включающие упрощенный пользовательский интерфейс, расширенный функционал для работы с множеством объектов недвижимости и автоматизированную систему расчетов. Моделирование бизнес-процессов позволило формализовать ключевые операции, такие как ввод данных, валидация показаний, автоматически расчет стоимости и формирование платежных документов. Сформулированные с использованием методологии FURPS+ требования к системе охватывают все критические аспекты

разработки, включая функциональность, удобство использования, надежность и производительность. Эти аспекты являются основополагающими для успешного создания конкурентоспособного продукта, который будет удовлетворять потребностям целевой аудитории. Важно также учитывать, что все эти требования взаимосвязаны и влияют друг на друга, что делает их комплексное рассмотрение особенно важным. Полученные результаты создают теоретическую основу для дальнейшего проектирования архитектуры и реализации мобильного приложения. Это включает не только технические спецификации, но и оценку пользовательского опыта, что в свою очередь формирует сильную конкурентную позицию на рынке. Внедрение четко определенных требований на раннем этапе разработки позволяет избежать значительных ошибок и перерасхода ресурсов в будущем.

Кроме того, применение методологии FURPS+ способствует более гибкому и адаптивному процессу разработки, позволяя команде учитывать изменения в запросах пользователей и быстро реагировать на них. Таким образом, хороший уровень качества на каждом этапе разработки напрямую влияет на общую успешность конечного продукта.

Глава 2 Проектирование мобильного приложения для расчета стоимости коммунальных услуг

2.1 Формализация структуры мобильного приложения

Проектирование мобильного приложения для автоматизированного расчета платежей за жилищно-коммунальные услуги представляет собой комплексный процесс, который требует системного подхода к разработке архитектурных решений и функциональных компонентов. Успешная реализация подобных систем напрямую зависит от тщательного анализа требований и выбора оптимальных технологических решений. В этой работе была выбрана архитектура Progressive Web Application (PWA). Как отмечает Google Developers, «PWA – это веб-приложения, созданные и усовершенствованные с помощью современных API-интерфейсов, обеспечивающие расширенные возможности, при этом охватывая любого веб-пользователя на любом устройстве с помощью единой базы кода» [24].

В рамках данного исследования рассматриваются ключевые аспекты проектирования, начиная от выбора оптимальной программной архитектуры до детализации диаграмм классов и баз данных.

2.1.1 Общая архитектура приложения

При проектировании архитектуры мобильного приложения особое внимание было уделено соблюдению современных подходов к построению программных систем. Это позволило обеспечить масштабируемость, надежность и высокую производительность системы. Архитектурное оформление системы предусматривает разделение на компоненты, которые взаимодействуют между собой строго заданным образом. На диаграмме компонентов (рисунок 5) визуализированы ключевые модули приложения и способы их взаимодействия, что наглядно демонстрирует структуру системы и упрощает ее дальнейшее сопровождение и развитие.

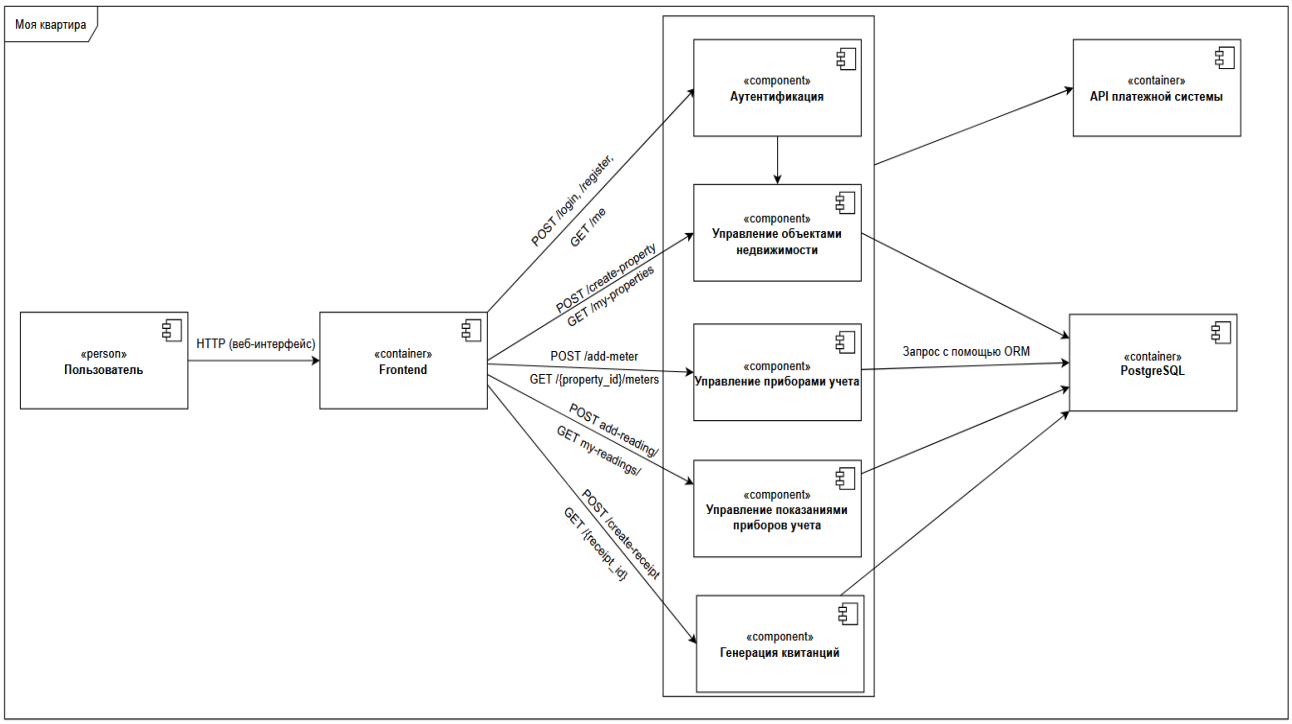


Рисунок 5 – Диаграмма компонентов приложения

На диаграмме представлена архитектура мобильного приложения, которая состоит из нескольких компонентов:

- клиентский модуль (отвечающий за взаимодействие с пользователем);

- серверный модуль (отвечающий за обработку запросов пользователя и логику работы приложения);

- модуль базы данных (отвечающий за хранения данных);

- модуль API платежной системы (отвечающий за тестовый платежный шлюз, который имитирует работу банковского API без реального списания средств)

После создания диаграммы компонентов следует построить диаграмму развертывания (рисунок 6), которая отобразит распределение логических компонентов системы между физическими узлами.

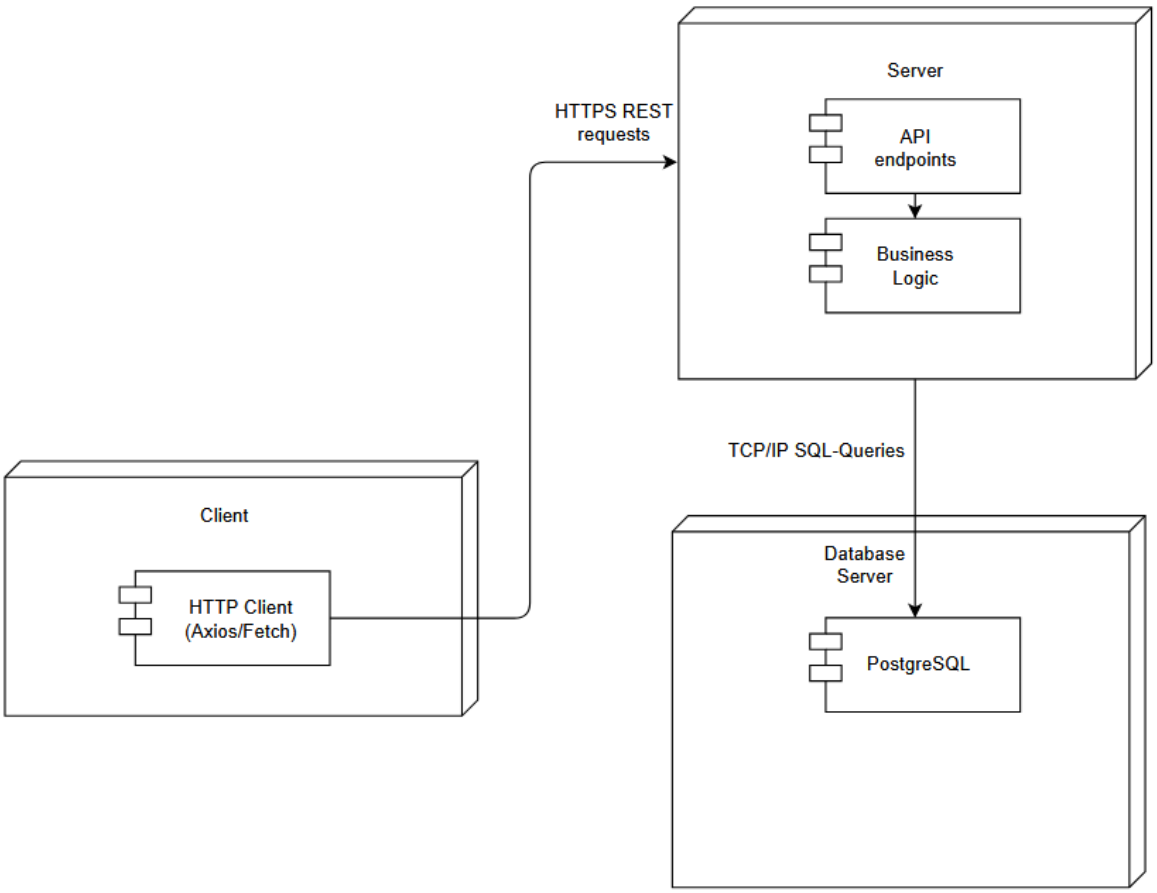


Рисунок 6 – диаграмма развертывания компонентов

Система реализована с помощью обычной трехуровневой архитектуры. Клиентский интерфейс, который разработан с помощью фреймворка React, работает в браузере пользователя. Серверный компонент разработанный с помощью библиотеки FastAPI развернут на хостинг-провайдере, служит для взаимодействия с базой данных и обработки запросов. База данных также развернута на хостинг-провайдере, но отдельном от серверного компонента. Взаимодействия между всеми компонентами реализовано с помощью протоколов, а именно: для серверного и клиентского взаимодействия используется HTTPS протокол, а для взаимодействия серверного компонента с базой данных используется TPC/IP, с использованием

SQL-запросов. Данная архитектура обеспечит безопасное и надежное функционирование системы расчета коммунальных платежей.

2.1.2 Моделирование поведения системы

Поведенческое проектирование системы было выполнено с помощью использования диаграмм активности и состояния. На рисунке 7 изображена диаграмма активности пользователя, которая отражает ключевые сценарии взаимодействия пользователя с мобильным приложением.

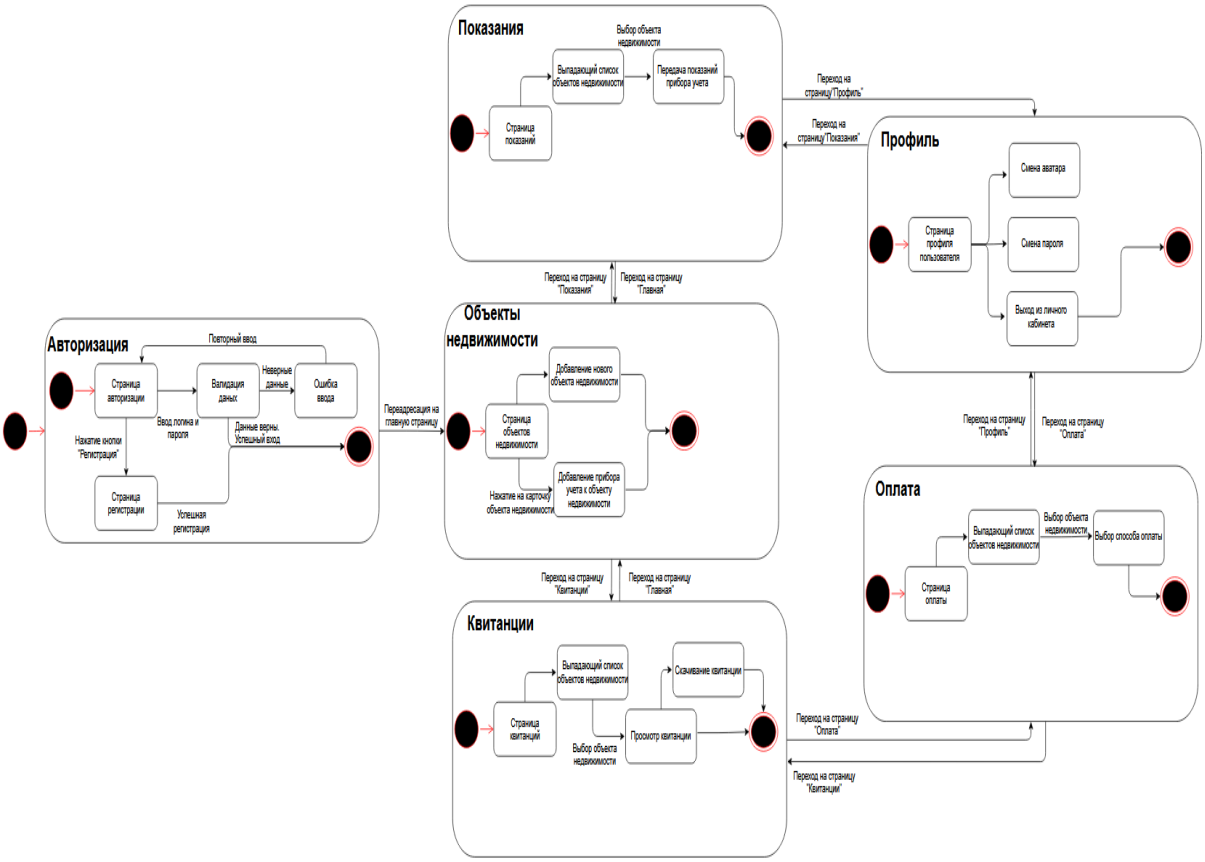


Рисунок 7 – Диаграмма активности пользователя

Эта диаграмма формализует полный цикл работы от авторизации до получения квитанций. Диаграмма активностей послужила основой для последующей детализации бизнес-процессов и позволила выявить критические точки взаимодействия пользователя с интерфейсом.

Для моделирования жизненного цикла сущностей системы была разработана диаграмма состояний объекта “Квитанция” (рисунок 8).

Рисунок 8 – Диаграмма состояний объекта “Квитанция”

Диаграмма отражает последовательность изменений статуса квитанции: первоначальное формирование происходит после ввода пользователем данных о коммунальных услугах и автоматического расчета суммы по установленным тарифам, что переводит квитанцию в состояние «Неоплаченная». После успешного проведения платежа система фиксирует факт оплаты, изменяя статус

на «Оплаченная». Особое внимание уделено альтернативным сценариям, таким как обработку случаев неудачных платежных операций. Данная модель наглядно демонстрирует полный цикл обработки квитанции в системе от момента создания до финального подтверждения оплаты и скачивания квитанции.

2.1.3 Проектирование диаграммы классов

Опираясь на диаграмму прецедентов, была спроектирована диаграмма классов (Рисунок 9), которая отображает все сущности системы, а также их атрибуты, методы и взаимосвязи.

На диаграмме представлены все основные классы, а именно: *User, Reading, Meter, Property, Receipt, Tariff.*

Класс *User* отвечает за хранение информации о пользователе, включая имя, email и хеш пароля. Пользователь может добавить объект недвижимости, фиксировать потребление услуг, просматривать счета и оплачивать их.

Класс *Reading* фиксирует объем потребления услуг пользователем. В нем хранятся данные о потребленных единицах, дате регистрации учета.

Класс Meter включает в себя тип счетчика (горячая вода, холодная вода, электричество) и уникальный идентификатор, который обычно изображен на приборе учета, также этот класс связан с классами объекта недвижимости (Property) и учета показаний (Reading).

Класс *Property* предназначен для хранения информации об объекте недвижимости и включает в себя данные о приборах учета с уникальным идентификатором.

Класс *Receipt* представляет собой объект квитанции выбранного пользователем объекта недвижимости. Она содержит в себе информацию о пользователе, общей сумме, номере транзакции и статусе оплаты (оплачен или не оплачен).

Класс *Tariff* содержит актуальный тариф за услуги ЖКХ, позволяет удобно производить расчеты стоимости коммунальных услуг.

На следующем рисунке изображена диаграмма классов мобильного приложения для расчета стоимости коммунальных услуг.

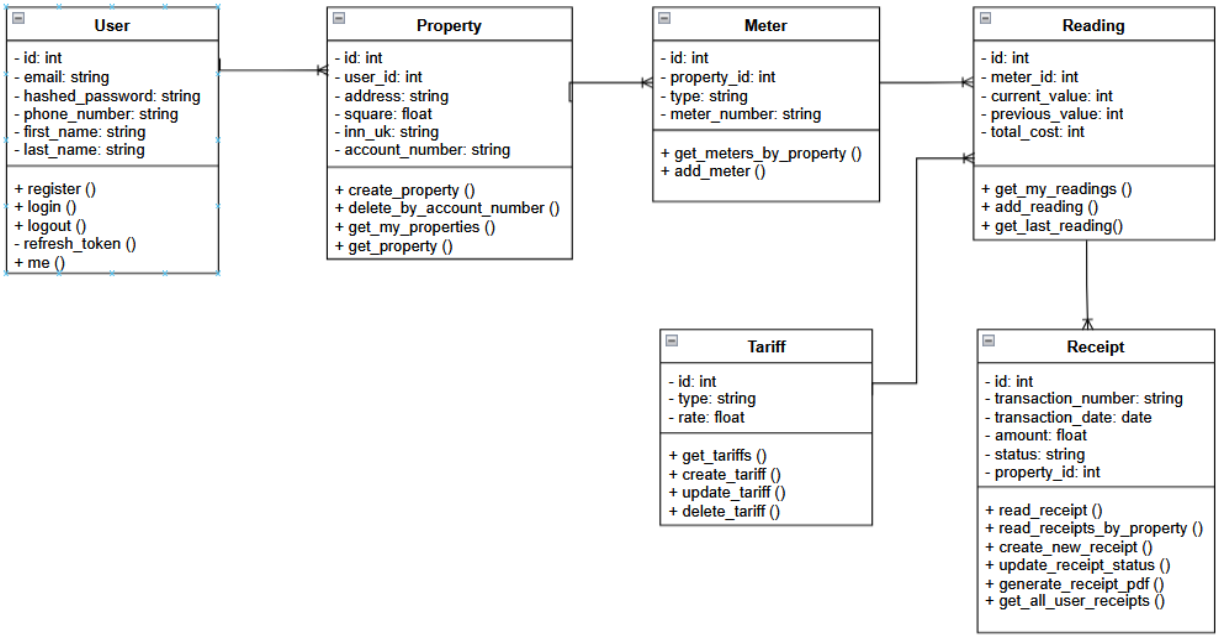


Рисунок 9 – Диаграмма классов

Таким образом, вследствие проектирования диаграммы классов, были разработаны основные классы и методы будущего мобильного приложения.

2.1.4 Логическая модель базы данных

Процесс логического проектирования базы данных предполагает объединение принципов объектно-ориентированного подхода с возможностями реляционных баз данных, что подмечает и М. Фаулер: «Такой подход позволяет сохранить гибкость объектно-ориентированного проектирования при работе с реляционными СУБД» [16]. В данном проекте для реализации этой связи используется PostgreSQL – современная реляционная СУБД, обеспечивающая полную поддержку ACID-транзакций, а именно четырьмя важными свойствами:

неразрывностью (atomicity), правильностью (correctness), изолированностью (isolation) и устойчивостью (durability) [4].

В текущих реалиях развития информационных технологий, системы управления базами данных занимают главное место в обеспечении эффективного хранения и обработки данных. Одним из устоявшихся подходов к проектированию реляционных баз данных является использование методологии IDEF1X. Логическая модель, которая простроена в рамках этой методологии, служит абстрактным представлением структуры данных, опирающимся на общую концептуальную модель предметной области.

На Рисунке 10 представлена логическая модель данных, разработанная на основе диаграммы классов.

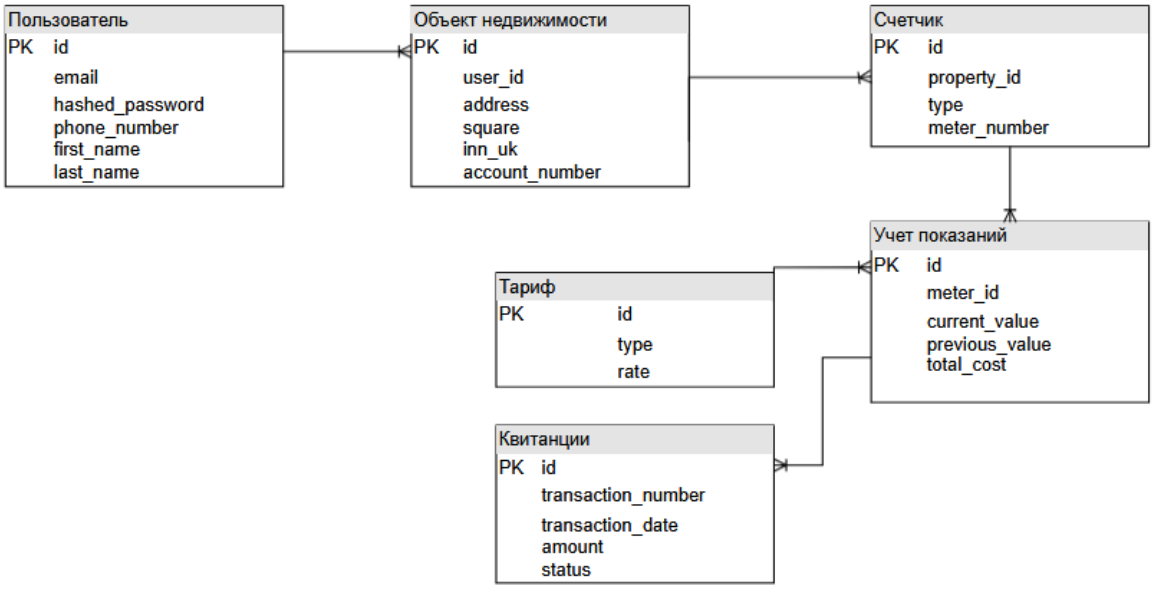


Рисунок 10 - Логическая модель базы данных.

А) User (Пользователь):

1) Хранит информацию о пользователях приложения.

2) Атрибуты:

- id (PK): универсальный идентификатор пользователя;

- email: адрес электронной почты пользователя;

- hashed_password: пароль пользователя;

- phone_number: номер телефона пользователя (опционно);

- first_name: имя пользователя;

- second_name: фамилия пользователя;

- created_at: дата регистрации пользователя.

Б) Property (Объект недвижимости):

1) Описывает объект недвижимости.

2) Атрибуты:

- id (PK): уникальный идентификатор услуги;

- user_id: связь пользователя и объекта недвижимости;

- address: адрес объекта недвижимости;

- square: площадь объекта недвижимости;

- inn_uk: ИНН управляющей компании объекта недвижимости;

- account_number: лицевой счет объекта недвижимости;

В) Reading (Показания прибора учета):

1) Хранит информацию о потребленных пользователем услугах.

2) Атрибуты:

- id (PK): уникальный идентификатор показаний прибора учета;

- meter_id: связь прибора учета и показаний;

- current_value: текущее значение прибора учета;

- previous_value: предыдущее значение прибора учета;

- total_cost: сумма учета показаний.

Г) Meter (Прибор учета):

1) Содержит в себе информацию о привязанных к объекту недвижимости приборах учета.

2) Атрибуты:

- id (PK): уникальный идентификатор счетчика;

- property_id: связь счетчика и объекта недвижимости;

- type: тип счетчика (горячая вода, холодная вода, электричество);

- meter_number: уникальный идентификатор прибора учета.

Д) Receipt (Квитанции):

1) Предоставляет пользователю общую информацию о его показаниях.

2) Атрибуты:

- id (PK): уникальный идентификатор квитанции;

- transaction_number: уникальный номер транзакции;

- transaction_date: дата транзакции

- amount: сумма платежа;

- status: статус платежа;

- property_id: связь квитанций и объекта недвижимости.

Связи между сущностями:

- User и Property: Один пользователь может добавлять множество объектов недвижимости (1:N);

- Property и Meter: К одному объекту недвижимости пользователь может добавить несколько приборов учета (1:N);

- Meter и Reading: Один счетчик может иметь множество показаний (1:N);

- Reading и Receipt: Показания могут иметь несколько квитанций (1:N).

Логическая модель данных отображает ключевые сущности системы и их взаимосвязи, обеспечивая структурированное хранение информации для корректной работы мобильного приложения.

Вывод по второй главе

При разработке архитектуры мобильного приложения для расчета стоимости ЖКХ была построена модель, которая предусматривает клиент-серверное взаимодействие как основную парадигму обмена данными между PWA-приложением на React и серверной частью на FastAPI с использованием PostgreSQL в качестве системы хранения данных. Применение методологии IDEF1X позволило создать оптимальную структуру, которая обеспечит целостность и надежность хранения информации. Ключевым аспектом стало формализованное описание поведенческих аспектов системы через диаграммы активностей и состояний, что важно для корректной обработки всей информации системы. Полученные результаты проектирования служат основой для последующей разработки приложения и позволят перейти к этапу непосредственной реализации системы.

Глава 3 Реализация и тестирование программного обеспечения

3.1 Выбор инструментов разработки

После проектирования мобильного приложения наступает этап реализации программного обеспечения. Главным аспектом на данной стадии является подбор оптимальных инструментов для разработки, которые обеспечат соответствие конечного решения предъявленным требованиям, а также обеспечит корректную реализацию заложенной архитектуры и логики взаимодействия модулей. Учитывая то, что мы разрабатываем мобильное приложение нам нужно определиться с целевой платформой.

В условиях разнообразия мобильных устройств от ведущих производителей (Apple, Honor, Redmi, Samsung, Xiaomi) и разделения рынка между двумя основными операционными системами – iOS (29% рынка) и Android (71% рынка по данным 2021 года) – особую популярность приобретают кроссплатформенные решения. Для данного проекта был выбран подход Progressive Web Application (PWA) обусловлен ключевыми преимуществами данной технологии, а именно: кроссплатформенность (работа на iOS, Andriod и ПК), возможность установки на домашний экран без магазинов приложений, оффлайн-работа через Service Workers и автоматическое обновление без участия пользователя.

Для реализации клиентской части приложения был выбран язык TypeScript с фреймворком React, а также фреймворк Bootstrap для CSS стилей. TypeScript является надмножеством JavaScript в синтаксическом смысле [2]. В свою очередь официальная документация React подчеркивает важность повторного использования компонентов, декларативный подход и использование виртуального DOM, что делает данный фреймворк особенно удобным для создания производительных веб-интерфейсов [22]. Компоненты – это именно то, что превращает React в мощный и гибкий фреймворк [13]. React

также обладает развитой экосистемой, включая множество готовых решений для создания PWA, таких как механизм service workers для оффлайн-работы и кэширование ресурсов. Серверная часть была реализована на языке Python – это интепретируемый объектно-ориентированный язык программирования высокого уровня, предназначенный для самого широкого круга задач [10]. Для реализации бэкенда использовался FastAPI – современный и асинхронный фреймворк, который сочетает простую разработку и высокую производительность. FastAPI также имеет автоматическую и удобную генерацию документации и встроенную валидацию данных через Pydantic. Это подтверждается в официальной документации: «FastAPI использует аннотации типов Python для автоматической генерации OpenAPI и валидации запросов» [21].

В качестве основной среды разработки используется Visual Studio Code – мощный и удобный редактор кода, который предоставляет полный набор необходимых инструментов для разработки на любом языке программирования. Он содержит встроенный отладчик и богатую коллекцию расширений для работы с библиотеками и фреймворками. Среда обеспечивает удобную работу и поддерживает современные практики разработки.

Выбранный мною технологический стек сочетает в себе производительность, кроссплатформенность и современные подходы к разработке веб-приложений. Использование React для фронтенда и FastAPI для бэкенда позволит создавать высоконагруженные приложения с богатой функциональностью, в то время как PWA-подход обеспечит нативоподобный пользовательский опыт на всех типах устройств без необходимости публикации в мобильных магазинах приложений. Такая архитектура сократит время разработки и упростит дальнейшую поддержку и масштабирование системы.

3.2 Разработка пользовательского интерфейса

После того, как мы определили технологический стек на основе React для клиентской части, FastAPI для серверной логики и Bootstrap для визуального оформления, можем переходить к этапу разработки пользовательского интерфейса [7]. Этот этап имеет важное значение, так как именно от правильно спроектированного пользовательского интерфейса зависит удобство взаимодействия пользователя с системой, так как качественный UX-дизайн способствует снижению количества ошибок пользователя и повышению лояльности к цифровому продукту.

Рассмотрим некоторые ключевые окна, а также опишем их функционал. На рисунке 11 и 12 изображены страницы авторизации и регистрации пользователя.

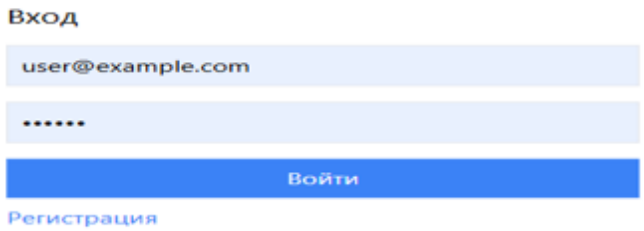


Рисунок 11 – Страница авторизации пользователя.

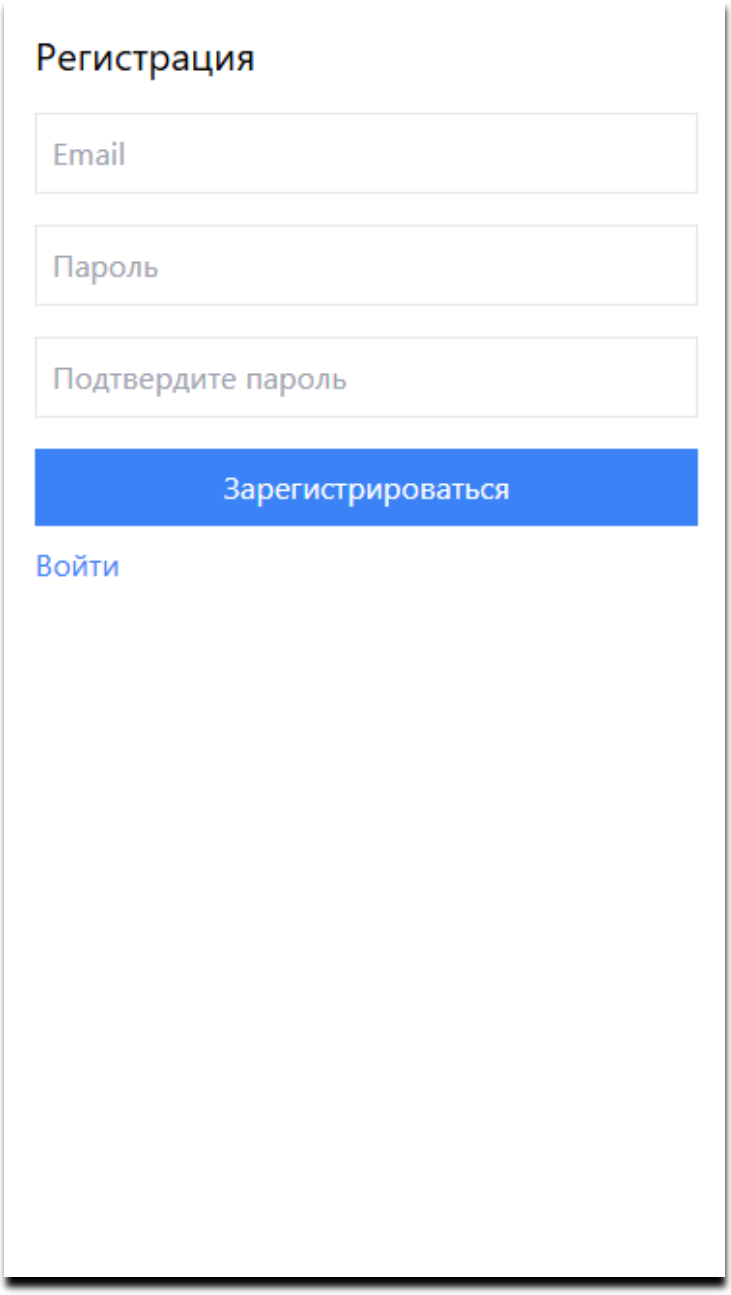


Рисунок 12 – Страница регистрации пользователя.

Эти страницы позволяют пользователю возможность регистрировать или авторизировать аккаунт для дальнейшего использования функционала мобильного приложения. Далее мы перейдем к разбору страниц отвечающих за добавление объектов недвижимости и добавление приборов учета, эти страницы изображены на рисунках 13, 14, 15.

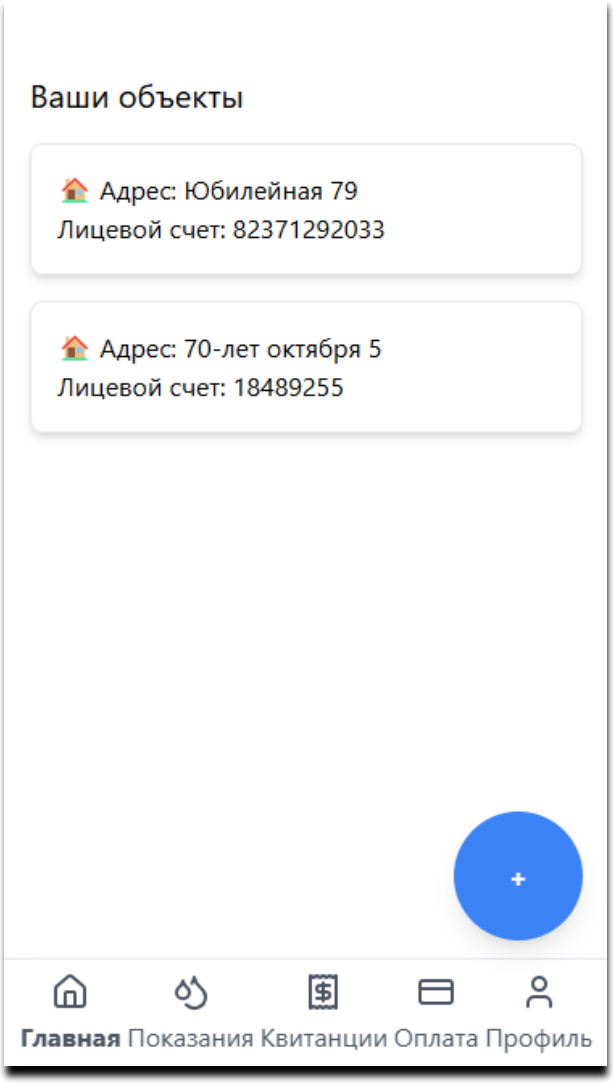


Рисунок 13 – Страница объектов недвижимости.

После авторизации или регистрации пользователь может просматривать свои объекты недвижимости и добавлять их с помощью кнопки, которая расположена в правом нижнем углу экрана. При нажатии на эту кнопку осуществляется переход на страницу создания объекта недвижимости. Выбрав конкретную карточку объекта недвижимости, осуществится переход на страницу с добавлением счетчика для выбранного объекта недвижимости. Страница добавления объекта недвижимости изображена на следующем изображении 14.

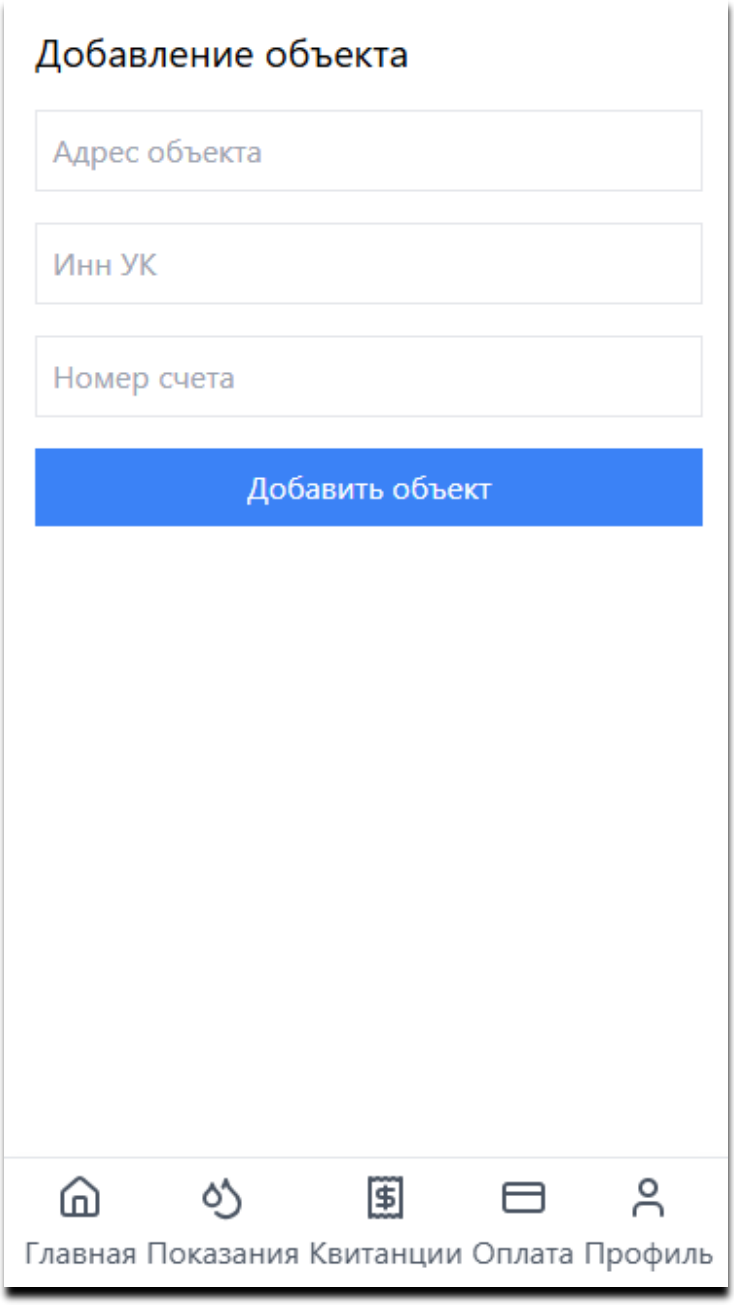


Рисунок 14 – Страница добавления объекта недвижимости.

На странице 14 изображены поля для ввода пользователем адреса недвижимости, ИНН управляющей компании и номера лицевого счета, а также кнопка добавления объекта. Страница подробного просмотра объекта недвижимости изображена на рисунке 15.

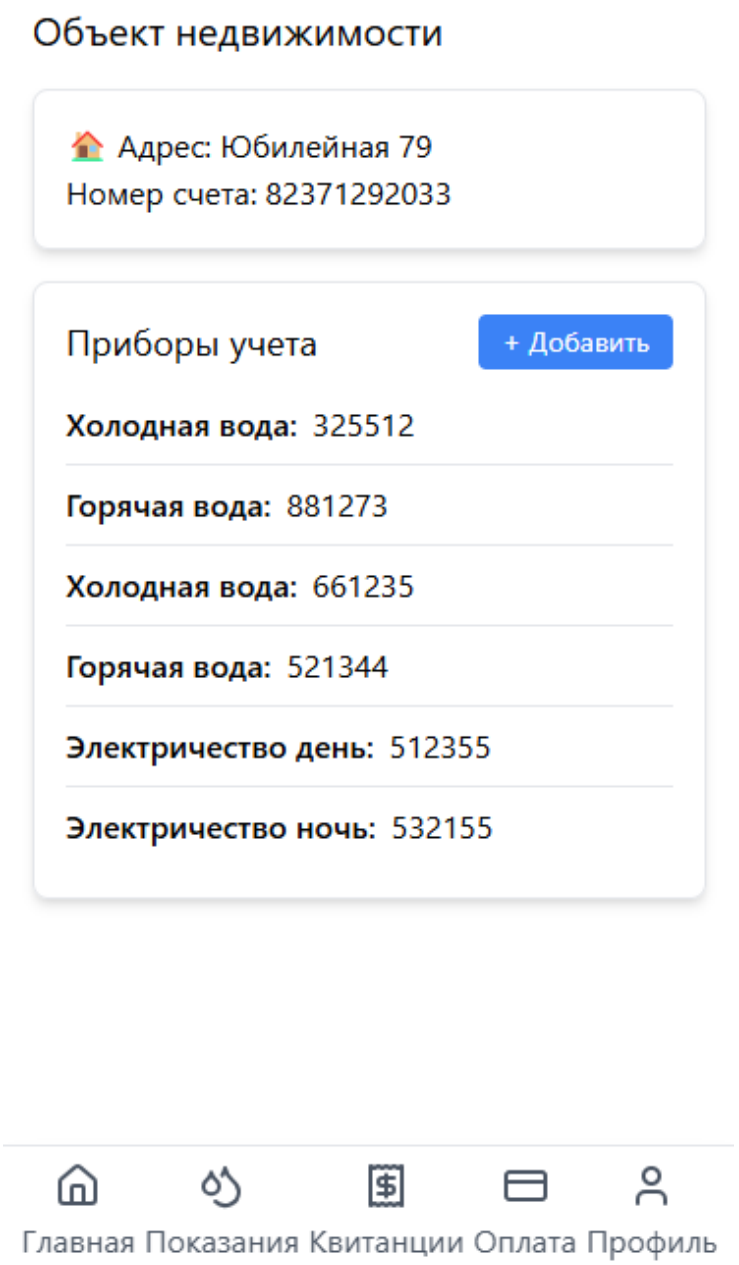


Рисунок 15 – страница подробного просмотра объекта недвижимости.

На странице подробного просмотра пользователь может увидеть адрес объекта, номер лицевого счета и добавленные пользователем приборы учета, а также кнопку для добавления приборов учета. На следующем рисунке 16 изображено модальное окно с добавлением счетчиков.

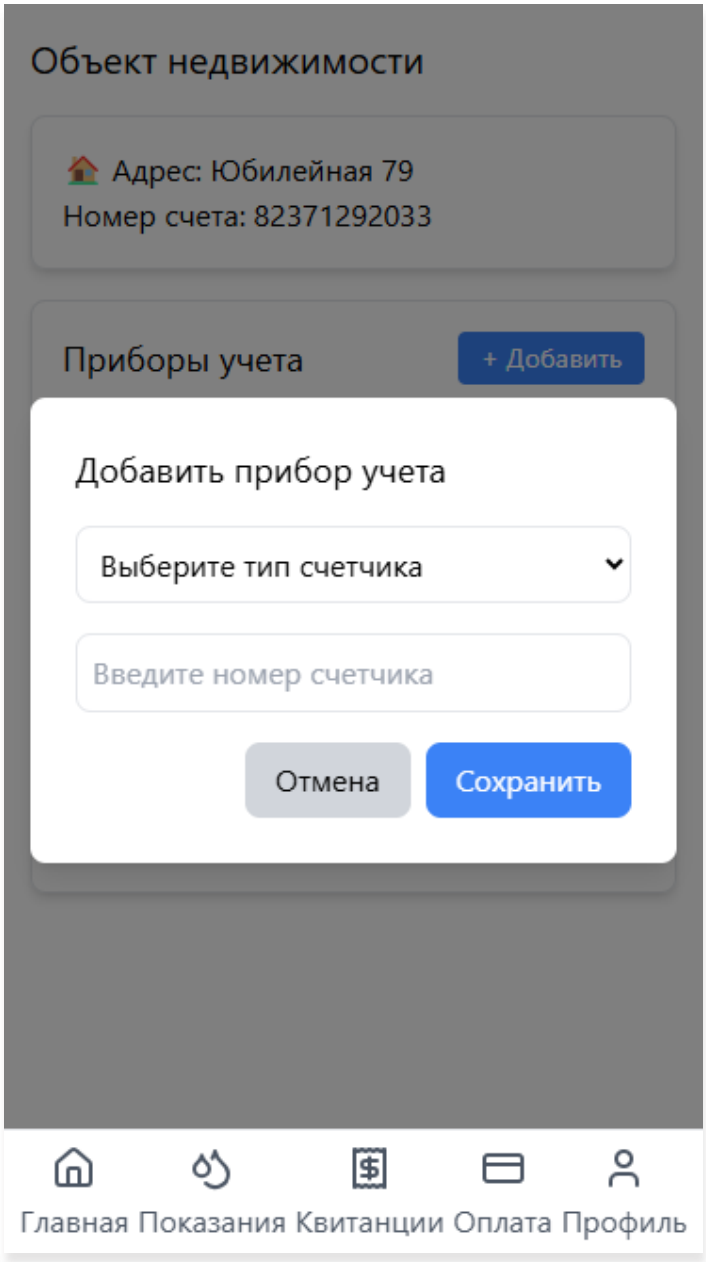


Рисунок 16 –Модальное окно создания прибора учета.

При нажатии на список типа счетчика, пользователю предоставится выбор нужного ему прибора учета, в поле номера счетчика пользователю нужно ввести его идентификационный номер. Следующий рисунок 17 изображает страницу управления приборами учета, где пользователь может просмотреть их или добавить показания.

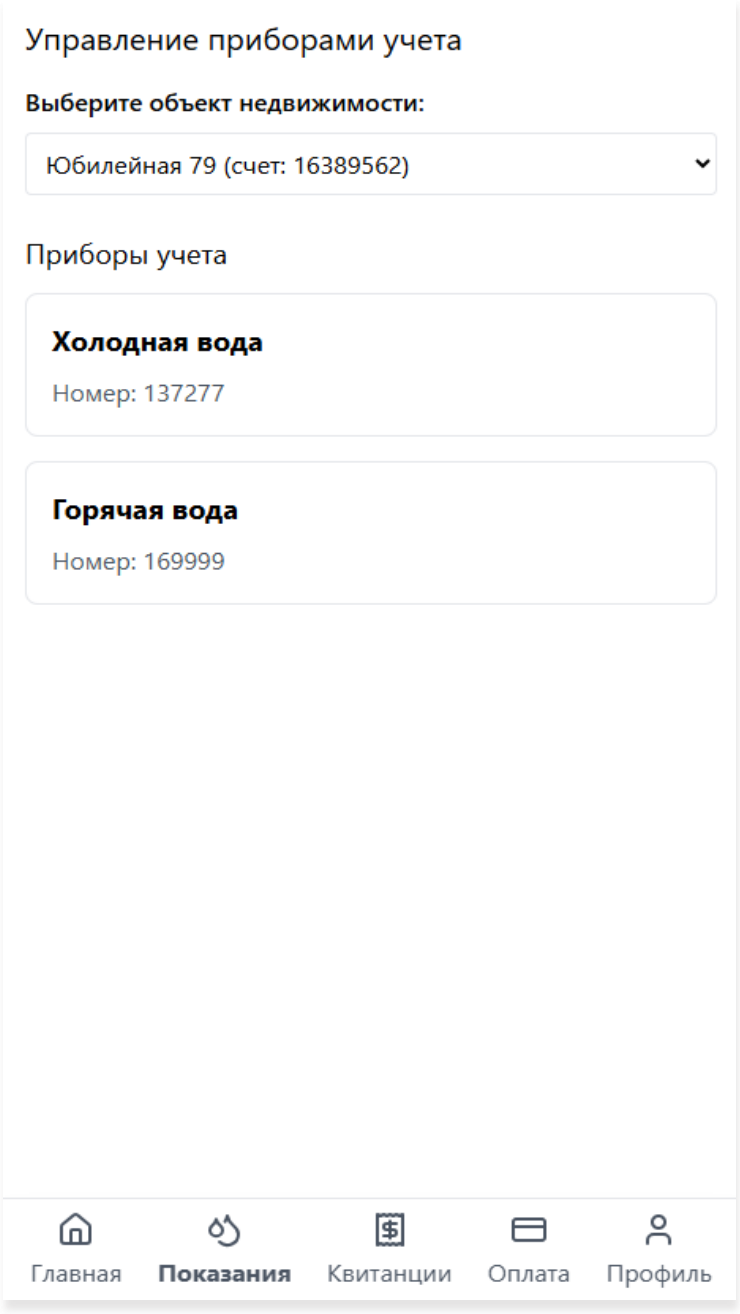


Рисунок 17 – Страница управления приборами учета.

На данной странице пользователь может выбрать нужный ему объект недвижимости и передать показания. Каждая карточка с информацией о приборе учета кликабельна, она перенаправит пользователя на страницу для передачи показаний эта страница будет изображена на рисунке 18.

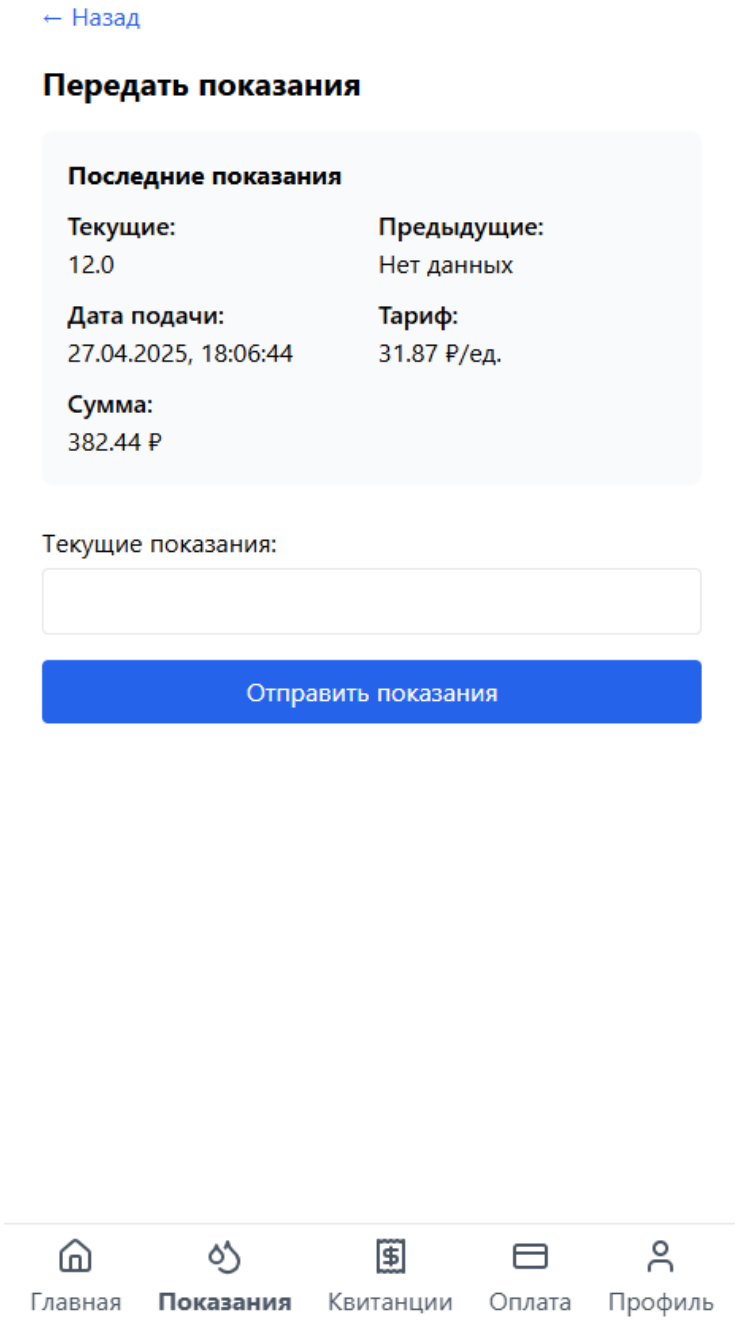


Рисунок 18 – Страница передачи показаний.

На этой странице показана дополнительная информация о предыдущих и текущих показаниях счетчика, дате, о тарифе и общей сумме. Также есть форма для ввода показаний и кнопка отправить. На рисунке 19 будет показана страница квитанций пользователя.

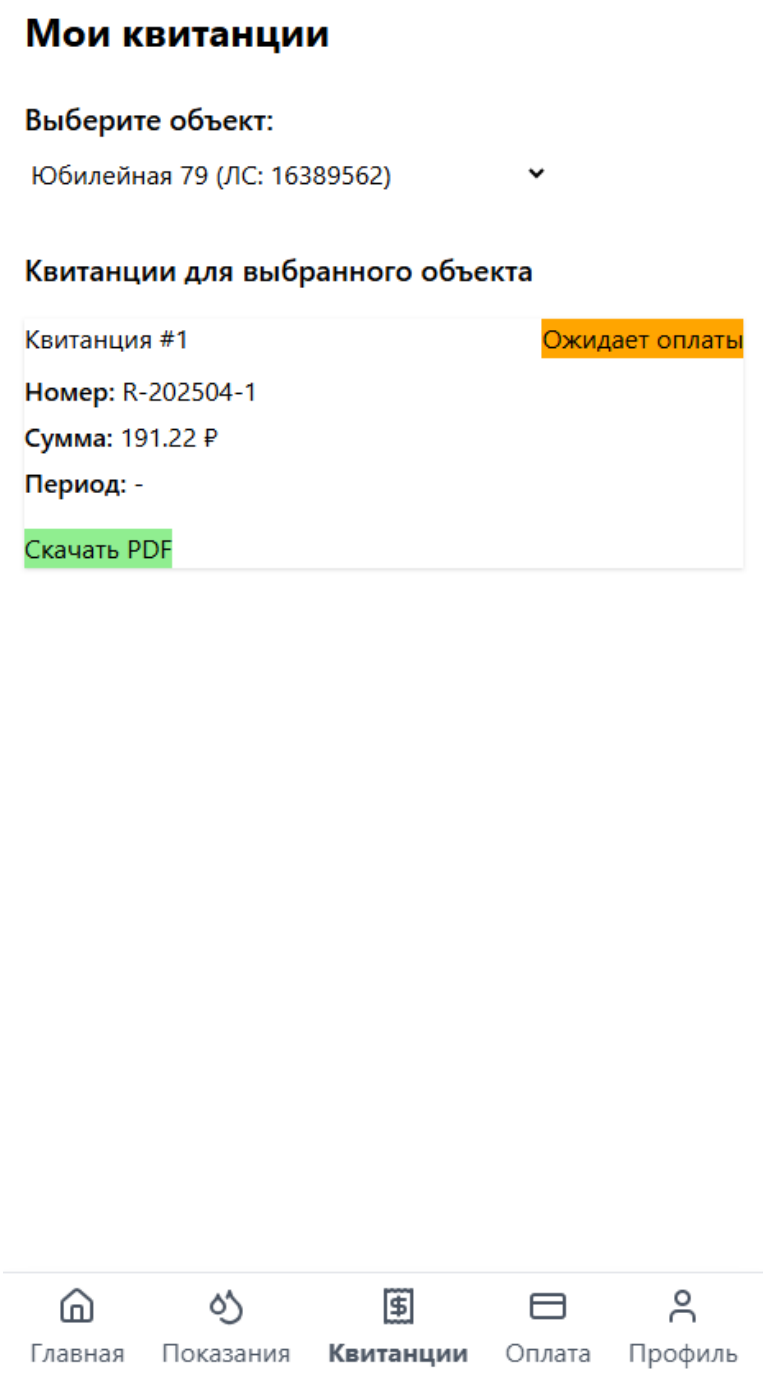


Рисунок 19 – Страница квитанций.

На этой странице пользователь может выбирать нужный ему объект недвижимости и отслеживать статус квитанции, общую сумму к оплате, а также скачать полный отчет в формате PDF. Следующий рисунок 20 будет изображать страницу оплаты.

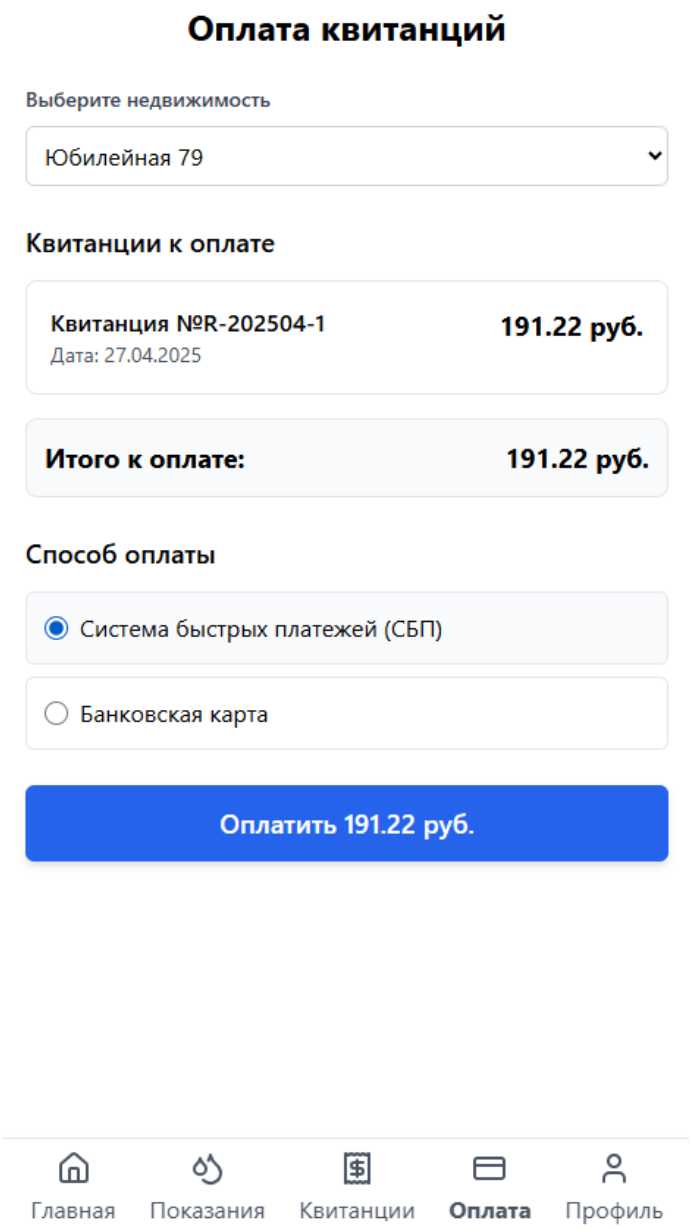


Рисунок 20 – Страница оплаты.

На этой странице пользователь может оплатить коммунальные услуги, система автоматически рассчитывает стоимость услуг по тарифу из базы данных и предоставляет пользователю квитанцию к оплате, после оплаты квитанция пометится как оплаченная и подсветится соответствующим цветом. На следующем рисунке 21 будет изображен профиль пользователя.

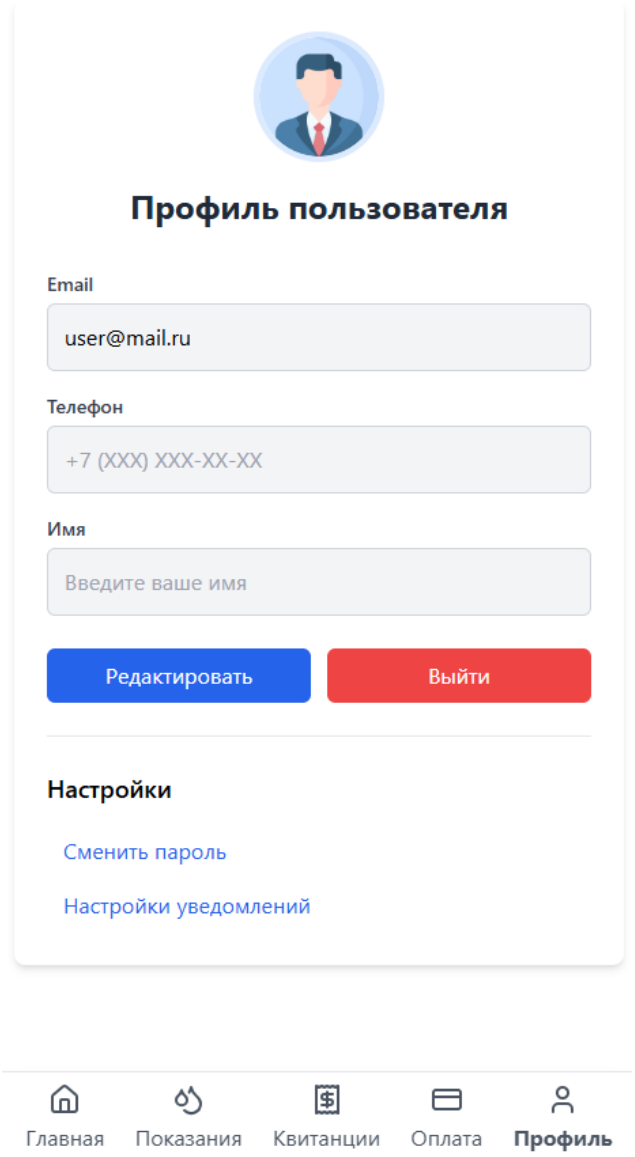


Рисунок 21 – Страница профиля пользователя.

Эта страница отвечает за настройки пользователя, а именно: его почту, пароль, имя, которое будет отображаться, аватар пользователя и телефон. Пользователь может менять все эти настройки, а также выходить из системы.

3.3 Реализация мобильного приложения

После создания UI&UX компонентов можно приступать к разработке мобильного приложения. Разработка начинается с инициализации проекта

FastAPI – создается виртуальная окружение Python для изоляции зависимостей проекта. Устанавливаются основные пакеты: сам фреймворк FastaAPI для создания API-интерфейсов, uvicron в качестве ASGI-сервера для запуска приложения, а также SQLAlchemy для работы с базой данных и Pydantic для валидации данных. Подробная файловая структура серверной части изображена на рисунке 22.

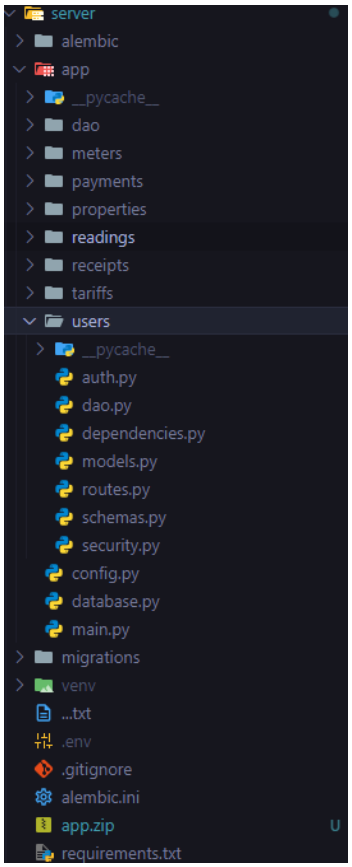


Рисунок 22 – Файловая структура серверной части приложения.

Файловая структура серверного приложения организуется по модульному принципу, где каждый функциональный блок-системы выделен в отдельный компонент. Модульная структура предполагает четкое разделение на логические компоненты – пользовательские операции (user), работа с объектами недвижимости (property) и другие сущности.

В каждом модуле реализован файл dao.py (Data Access Object), который отвечает за непосредственное взаимодействие с базой данных. Этот слой содержит классы с методами выполнения SQL-запросов через SQLAlchemy.

В файлах models.py определены ORM-модели, автоматически преобразуемые SQLAlchemy в таблицы PostgreSQL с сохранением всех связей и ограничений целостности. Это решение позволяет реализовать паттерн Data Mapper, который обеспечивает четкое разделение бизнес-логики и работы с БД. Для каждой сущности прописываются поля с типами данных, отношениями между таблицами и дополнительными валидаторами. Особое внимание уделяется индексам и ограничениям, обеспечивающим целостность данных при высоких нагрузках.

Файлы routes.py содержат определение API-эндпоинтов, сгруппированных по функциональному признаку. Каждый маршрут включает декораторы с указанием HTTP-метода, пути и возможных кодов ответов.

Схемы валидации request/response вынесены в отдельные файлы schemes.py, где с помощью Pydantic определяются строго типизированные модели для входящих и исходящих данных. Это позволяет автоматически проверять корректность поступающей информации и генерировать подробную документацию в Swagger UI.

Главный файл main.py выполняет интеграцию всех модулей: создает экземпляр FastAPI, подключает роутеры из модулей, настраивает middleware для обработки CORS и исключений.

Эта архитектура обеспечивает высокую степень поддерживаемости кода - каждый функциональный блок изолирован и может разрабатываться независимо. После настройки базовой структуры серверного приложения мы переходим к реализации ключевых функциональных модулей, а именно модулю аутентификации. Этот модуль обеспечивает безопасный доступ пользователей к ресурсам приложения. Для аутентификации пользователей в системе реализован механизм JWT (JSON Web Token), который обеспечивает

безопасную передачу данных между сторонами в виде JSON-объекта. Согласно спецификации RFC 7519: «JSON Web Token – это компактное, безопасное с точки зрения URL-адреса средство представления заявок, подлежащих передаче между двумя сторонами» [25]. На рисунке 23 будет изображена реализация создания access и refresh токенов авторизации.

```python
from passlib.context import CryptContext
from pydantic import EmailStr
from jose import jwt, JWTError
from datetime import datetime, timedelta, timezone
from app.config import get_auth_data
from app.users.dao import UsersDAO

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")


def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)


def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)


async def authenticate_user(email: EmailStr, password: str):
    user = await UsersDAO.find_one_or_none(email=email)
    if not user or verify_password(plain_password=password, hashed_password=user.hashed_password) is False:
        return None
    return user

def create_access_token(data: dict, expires_delta: timedelta | None = None):
    auth_data = get_auth_data()
    to_encode = data.copy()

    if expires_delta:
        expire = datetime.now(timezone.utc) + expires_delta
    else:
        expire = datetime.now(timezone.utc) + timedelta(minutes=30)

    to_encode.update({"exp": expire})

    encoded_jwt = jwt.encode(to_encode, auth_data['secret_key'], algorithm=auth_data['algorithm'])
    print(f'Время создания токена: {datetime.now(timezone.utc)}')
    print(f'Время истечения токена: {expire}')
    return encoded_jwt

def create_refresh_token(data: dict, expires_delta: timedelta | None = None):
    auth_data = get_auth_data()
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.now() + expires_delta
    else:
        expire = datetime.now() + timedelta(days=7)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, auth_data['secret_key'], algorithm=auth_data['algorithm'])
    return encoded_jwt
```

Рисунок 23 – Реализация JWT.

Такой подход обеспечит устойчивость к брут-форс атакам за счет адаптивного хеширования и использования “соли”. На следующем изображении 24 будет изображен процесс использования JWT-токенов, такой подход наглядно демонстрирует последовательность проверок и действий, выполняемых при

отправке запроса от пользователя (например, при отправке запроса на регистрацию).

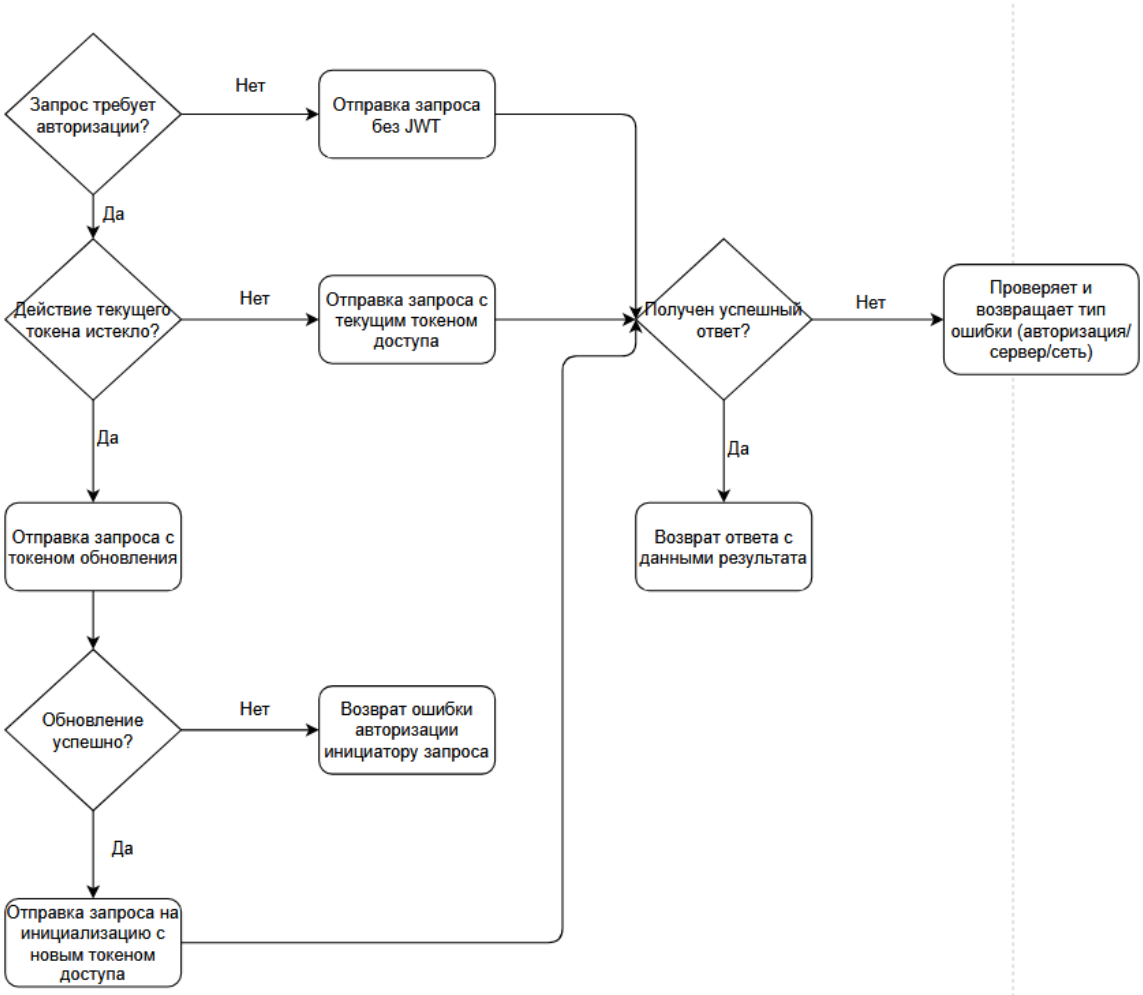
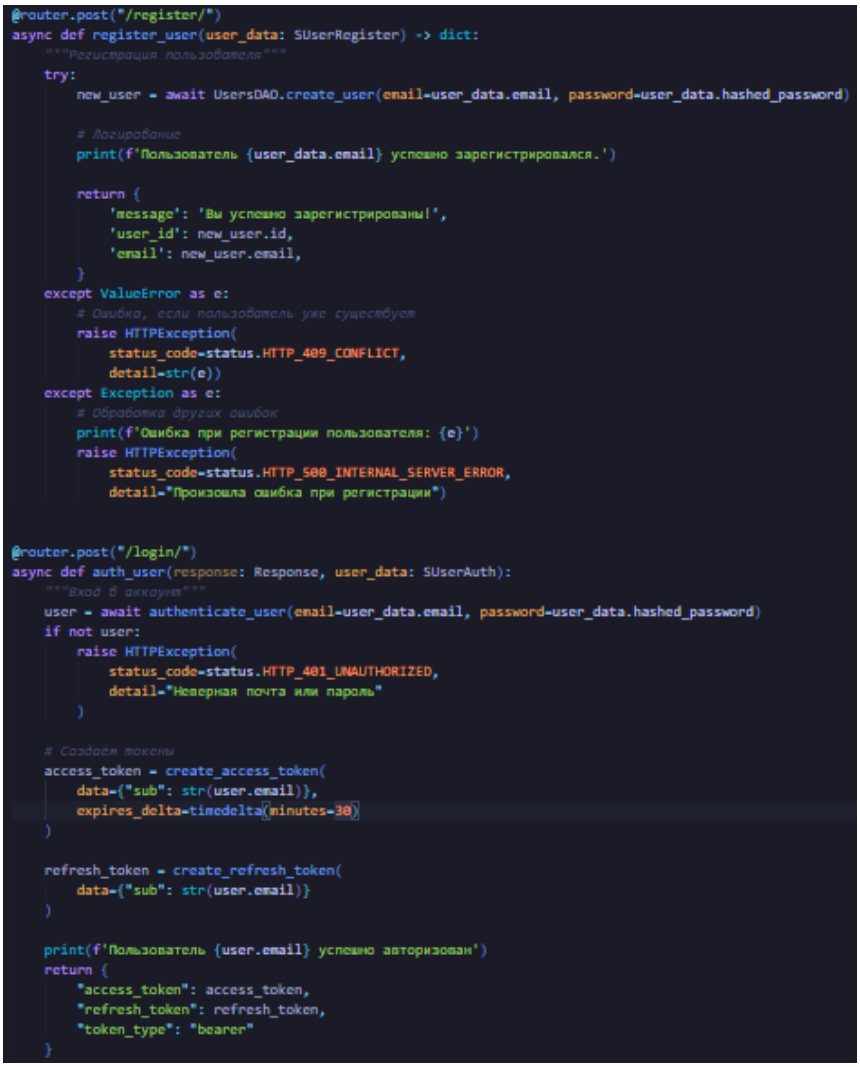


Рисунок 24 – Блок-схема запросов и ответов JWT.

На представленной блок-схеме (рис. 24) детализирован механизм взаимодействия клиентской и серверной частей системы при обработке JWT-аутентификации. Как видно из схемы, процесс начинается с инициирующего запроса от пользователя, после чего система последовательно выполняет ряд операций, включая валидацию данных, генерацию токена и его последующую верификацию.

Далее на рисунке 25 будет изображена реализация API-эндпоинтов регистрации и авторизации пользователя.



```python
@router.post("/register/")
async def register_user(user_data: SUserRegister) -> dict:
    """Регистрация пользователя"""
    try:
        new_user = await UsersDAO.create_user(email=user_data.email, password=user_data.hashed_password)

        # Логирование
        print(f'Пользователь {user_data.email} успешно зарегистрировался.')

        return {
            'message': 'Вы успешно зарегистрированы!',
            'user_id': new_user.id,
            'email': new_user.email,
        }
    except ValueError as e:
        # Ошибка, если пользователь уже существует
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail=str(e))
    except Exception as e:
        # Обработка других ошибок
        print(f'Ошибка при регистрации пользователя: {e}')
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail="Произошла ошибка при регистрации")


@router.post("/login/")
async def auth_user(response: Response, user_data: SUserAuth):
    """Вход в аккаунт"""
    user = await authenticate_user(email=user_data.email, password=user_data.hashed_password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверная почта или пароль"
        )

    # Создаем токены
    access_token = create_access_token(
        data={"sub": str(user.email)},
        expires_delta=timedelta(minutes=30)
    )

    refresh_token = create_refresh_token(
        data={"sub": str(user.email)}
    )

    print(f'Пользователь {user.email} успешно авторизован')
    return {
        "access_token": access_token,
        "refresh_token": refresh_token,
        "token_type": "bearer"
    }
```

Рисунок 25 – Реализация API-эндпоинтов регистрации и авторизации.

Эндпоинт регистрации “/register” принимает данные пользователя, включая электронную почту и пароль. Перед сохранением в базу данных система проверяет, что указанный email не занят другим пользователем, а пароль соответствует требованиям сложности. Если валидация пройдена, тогда пароль хешируется с помощью bcrypt, после чего создается новая запись в таблице пользователей. В ответ клиенту возвращаются сгенерированные JWT-

токены (access и refresh), которые будут использоваться для последующей аутентификации. В эндпоинте входа “/login” обрабатывается запрос на авторизацию. Сначала проверяется, что оба поля (email и пароль) заполнены. Затем система ищет пользователя по email в базе данных. Если запись найдена, тогда происходит проверка соответствия пароля путем сравнения хешей. В случае успеха генерируются новые токены доступа, а информация о пользователе (например, почта или идентификатор) включается в payload access-токена. Это позволяет в дальнейшем идентифицировать пользователя при запросах к защищенным эндпоинтам без необходимости повторной аутентификации. На рисунке 26 будет представлена ORM-модель объекта недвижимости, реализованная с помощью SQLAlchemy.

Рисунок 26 – Модель объекта недвижимости

На данном рисунке показана структура таблицы Property, которая хранит информацию об объектах недвижимости пользователей. Модель имеет следующие поля: идентификатор объекта, внешний ключ на таблицу пользователей, а также лицевой счет. В модели также реализованы индексы и ограничения, которые обеспечат целостность связей между пользователями и их недвижимостью. После определения модели следует описание схем

валидации данных, на рисунке 27 представлена реализация Pydantic-схем для объектов недвижимости.

```python
from pydantic import BaseModel, Field

class SPropertyCreate(BaseModel):
    address: str = Field(..., description="Адрес объекта недвижимости")
    inn_uk: str = Field(..., description="ИНН УК")
    account_number: str = Field(..., description="Номер счета")

    class Config:
        from_attributes = True
```

Рисунок 27 – Pydantic-схема валидации объектов недвижимости.

На этом рисунке показана схема “SpropertyCreate”, она используется для валидации данных при создании и получения объектов недвижимости через API. Использование схем позволяет автоматизировать проверку структуры входящих данных и генерировать подробную документацию через Swagger. Следующим этапом является создания слоя доступа к данным (DAO). На рисунке 28 показана реализация DAO-объекта для работы с объектами недвижимости.

```python
class PropertiesDAO(BaseDAO):
    model = Property

    @classmethod
    async def create_property(cls, user_id: int, address: str, inn_uk: str, account_number: str):
        async with async_session_maker() as session:
            print("Создали сессию бд")
            new_property = Property(user_id=user_id, address=address, inn_uk=inn_uk, account_number=account_number)
            print("Объект недвижимости создан")
            session.add(new_property)
            await session.commit()
            await session.refresh(new_property)
            return new_property

    @classmethod
    async def delete_by_account_number(cls, account_number: str) -> bool:
        async with async_session_maker() as session:
            print("session created")
            query = select(cls.model).where(cls.model.account_number == account_number)
            print(f"Выбрали модель, где аккаунт: {query}")
            result = await session.execute(query)
            property_to_delete = result.scalar_one_or_none()

            if not property_to_delete:
                return False

            await session.delete(property_to_delete)
            await session.commit()
            return True

    @classmethod
    async def get_properties_by_user(cls, user_id: int):
        async with async_session_maker() as session:
            # Получаем все объекты недвижимости для пользователя
            query = select(Property).where(Property.user_id == user_id)
            result = await session.execute(query)
            return result.scalars().all()

    @classmethod
    async def get_property_by_id(cls, property_id: int, user_id: int = None):
        async with async_session_maker() as session:
            query = select(Property).where(Property.id == property_id)

            if user_id:  # Если нужно проверить принадлежность пользователю
                query = query.where(Property.user_id == user_id)

            result = await session.execute(query)
            return result.scalar_one_or_none()
```

Рисунок 28 – DAO для работы с объектами недвижимости.

DAO (Data Access Object) – инкапсулирует операции с базой данных: создание, получение, обновление и удаление объектов недвижимости. Это позволяет централизовать логику работы с БД и упрощает тестирование и поддержку кода. Последним этапом является определение маршрутов API для взаимодействия с объектами недвижимости. На рисунке 29 изображена реализация эндпоинта создания объекта недвижимости и его получение.

```python
@router.post("/create-property")
async def create_property(property_data: SPropertyCreate, current_user: User = Depends(get_current_user)) -> dict:
    """Создать объект недвижимости"""
    try:
        new_property = await PropertiesDAO.create_property(
            user_id=current_user.id,
            address=property_data.address,
            inn_uk=property_data.inn_uk,
            account_number=property_data.account_number
            )

        return {
            'message': 'Объект зарегистрирован!',
            'user_id': new_property.user_id,
            'address': new_property.address,
            'account_number': new_property.account_number
        }
    except ValueError as e:
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail=str(e))
    except Exception as e:
        print(f'Ошибка при регистрации пользователя: {e}')
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail="Произошла ошибка при регистрации")

@router.get("/my-properties/")
async def get_my_properties(
    current_user: User = Depends(get_current_user)
) -> dict:
    """Получить объекты недвижимости для текущего пользователя"""
    try:
        properties = await PropertiesDAO.get_properties_by_user(current_user.id)
        return {
            "properties": [
                {
                    "id": property.id,
                    "address": property.address,
                    "user_id": property.user_id,
                    "account_number": property.account_number
                }
                for property in properties
            ]
        }
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail=f"Ошибка при получении объектов недвижимости: {e}"
        )
```

Рисунок 29 – API-эндпоинты создания и получения объекта недвижимости.

Реализованные эндпоинты позволяют пользователю создавать новые объекты недвижимости и получать их список. Доступ к маршрутам защищен JWT-аутентификацией, что гарантирует безопасность операций. Следующим этапом является описание разработки клиентской части приложения.

Клиентская часть была реализована с помощью библиотеки React и фреймворка Bootstrap для стилизации интерфейсов. После инициализации проекта через консольную команду “create-react-app” был произведен базовый рефакторинг структуры проекта: папки components, core, а также все сервисы, такие как users, properties, readings, receipts, payments, обеспечивают логическое разделение кода по функциональным областям. На этапе первичной настройки были установлены необходимые зависимости, а именно: axios – для упрощенной работы с HTTP-запросами к backend-серверу, react-router-dom – для организации маршрутизации внутри приложения, jwt-decode – для декодирования токенов и проверки их валидности на клиенте, а также react-query – для управления состоянием загрузки данных с сервера. На рисунке 30 изображена структура клиентской части.

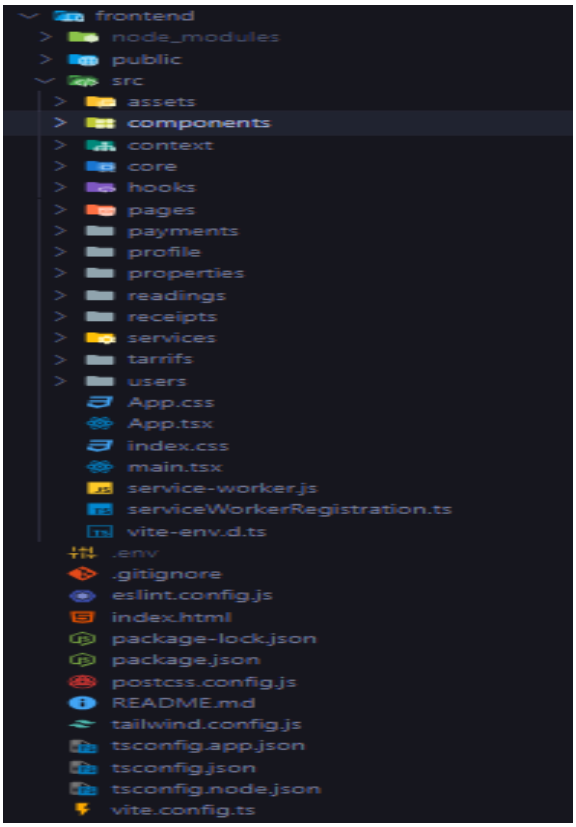


Рисунок 30 – Структура клиентской части приложения

Такая структура поможет легко масштабировать и отлаживать клиентскую сторону приложения. После базовой настройки структуры проекта были реализованы сервисы для работы с API. Например, для управления объектами недвижимости был создан модуль “PropertyAPI.ts” (рисунок 31).

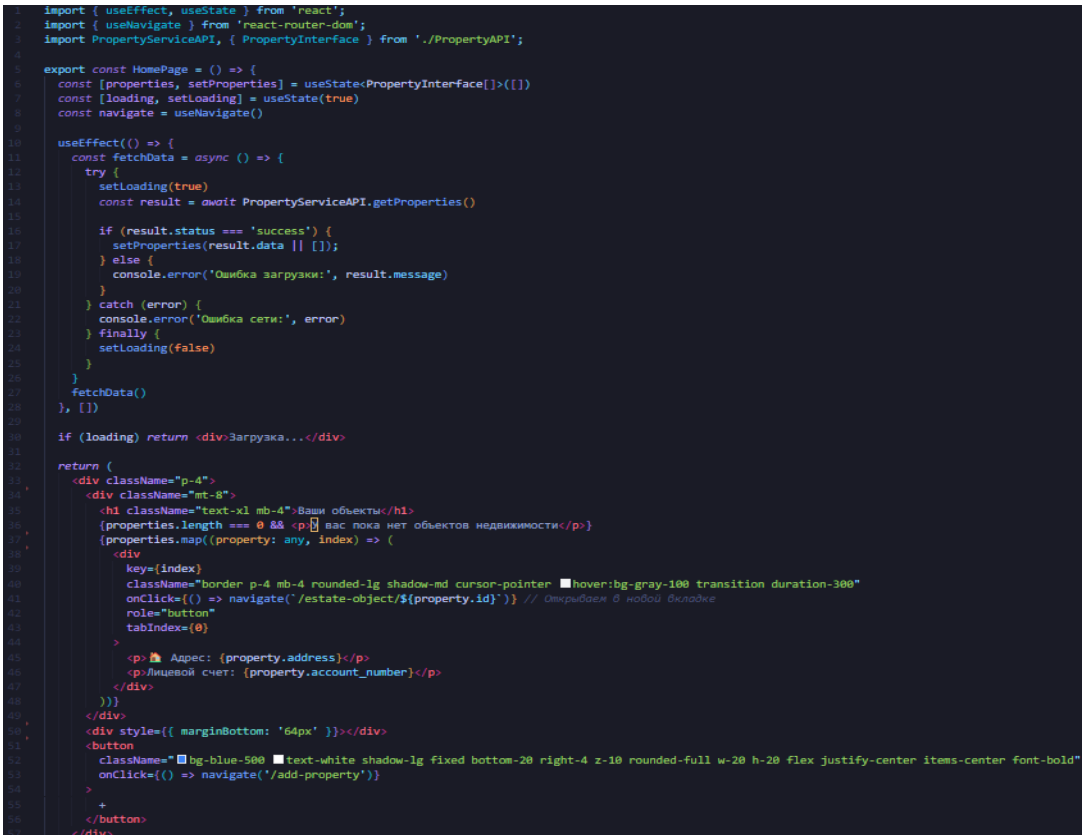
```
export interface PropertyInterface {
    id: number;
    address: string;
    inn_uk: string;
    account_number: string;
    unpaid_receipts_count?: number;
}

class PropertyServiceAPI {
    // Добавление объекта недвижимости
    static async addProperty(data: {
        address: string;
        inn_uk: string;
        account_number: string;
      }): Promise<ApiResponse<any>> {
        try {
          const response = await apiClient.post(
            '/property/create-property',
            data,
            { headers: getAuthHeader() }
          );
          return { status: 'success', data: response.data };
        } catch (error) {
          console.error('Add property error:', error);
          return { status: 'error', message: 'Failed to add property' };
        }
      }

    // Получение всех объектов недвижимости
    static async getProperties(): Promise<ApiResponse<PropertyInterface[]>> {
        try {
            const response = await apiClient.get(
                '/property/my-properties/',
                {
                    headers: getAuthHeader(),
                })
            const properties = response.data.properties || []

            return {
                status: 'success',
                data: properties
            }
        }
        catch (error: any) {
            console.error('Ошибка получения объектов:', error.response?.data)
            return {
                status: 'error',
                message: 'Ошибка получения объектов',
                data: []
            }
        }
    }
```

Рисунок 31 – Реализация сервиса для работы с объектами недвижимости.

В этом модуле определены интерфейсы типов данных для строгой типизации (PropertyInterface), а также методы взаимодействия с сервером через HTTP-запросы с использованием axios. Основные функции включают: добавление объектов недвижимости – отправка POST-запроса с данными нового объекта, а также получение списка объектов недвижимости

(getProperties) – отправка GET-запроса для получения всех объектов, которые привязаны к учетной записи пользователя. Каждый запрос сопровождается добавлением заголовка авторизации (Authorization), который получается через вспомогательную функцию getAuthHeader. Это обеспечивает безопасную работу клиента с защищенными эндпоинтами сервера. После реализации модуля взаимодействия с API для объектов недвижимости следующим этапом стало создание компонентов пользовательского интерфейса для управления этими объектами.

На клиентской стороне за функциональность работы с недвижимостью отвечают несколько компонентов: “AddProperty.tsx”, “EstateObject.tsx” и интеграция на главную страницу “Home.tsx”. На рисунке 32 будет изображена реализация главной страницы объектов недвижимости.



```tsx
import { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import PropertyServiceAPI, { PropertyInterface } from './PropertyAPI';

export const HomePage = () => {
  const [properties, setProperties] = useState<PropertyInterface[]>([])
  const [loading, setLoading] = useState(true)
  const navigate = useNavigate()

  useEffect(() => {
    const fetchData = async () => {
      try {
        setLoading(true)
        const result = await PropertyServiceAPI.getProperties()

        if (result.status === 'success') {
          setProperties(result.data || []);
        } else {
          console.error('Ошибка загрузки:', result.message)
        }
      } catch (error) {
        console.error('Ошибка сети:', error)
      } finally {
        setLoading(false)
      }
    }
    fetchData()
  }, [])

  if (loading) return <div>Загрузка...</div>

  return (
    <div className="p-4">
      <div className="mt-8">
        <h1 className="text-xl mb-4">Ваши объекты</h1>
        {properties.length === 0 && <p>У вас пока нет объектов недвижимости</p>}
        {properties.map((property: any, index) => (
          <div
            key={index}
            className="border p-4 mb-4 rounded-lg shadow-md cursor-pointer hover:bg-gray-100 transition duration-300"
            onClick={() => navigate(`/estate-object/${property.id}`)} // Открываем в новой вкладке
            role="button"
            tabIndex={0}
          >
            <p>🏠 Адрес: {property.address}</p>
            <p>Лицевой счет: {property.account_number}</p>
          </div>
        ))}
      </div>
      <div style={{ marginBottom: '64px' }}></div>
      <button
        className="bg-blue-500 text-white shadow-lg fixed bottom-20 right-4 z-10 rounded-full w-20 h-20 flex justify-center items-center font-bold"
        onClick={() => navigate('/add-property')}
      >
        +
      </button>
    </div>
```

Рисунок 32 – Главная страница объектов недвижимости.

На главной странице приложения происходит интеграция компонента списка объектов. При загрузке страницы автоматически вызывается метод “getProperties” из модуля “PropertyAPI.ts”, который получает все объекты недвижимости, привязанные к текущему пользователю. На основе полученных данных динамически рендерится список карточек “EstateObject.tsx”, реализация списка карточек будет изображена на следующем рисунке 33.

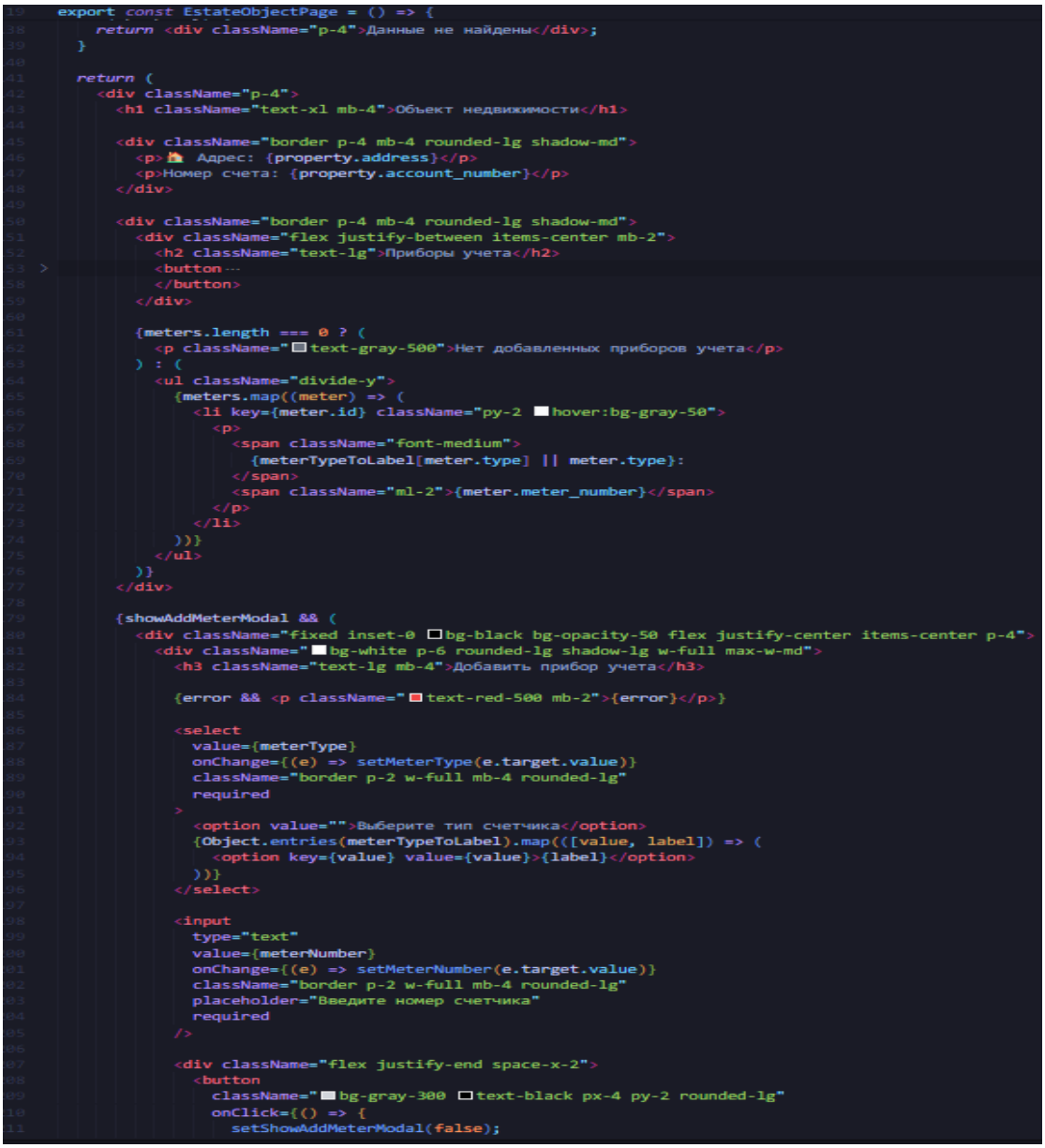


Рисунок 33 – Отображение карточки объекта недвижимости и модальное окно.

Компонент “EstateObject” отвечает за отображение кликабельной карточки отдельного объекта недвижимости в пользовательском интерфейсе. Каждая карточка выводит следующую информацию: адрес объекта, номер лицевого счета и управляющую компанию. Модальное окно в свою очередь открывается при нажатии на карточку объекта недвижимости, это модальное окно отображает список для выбора типа счетчика, форму для ввода номера счетчика, а также кнопку создания прибора учета. Далее, на следующем рисунке 34 будет показана реализация страница создания объекта недвижимости.

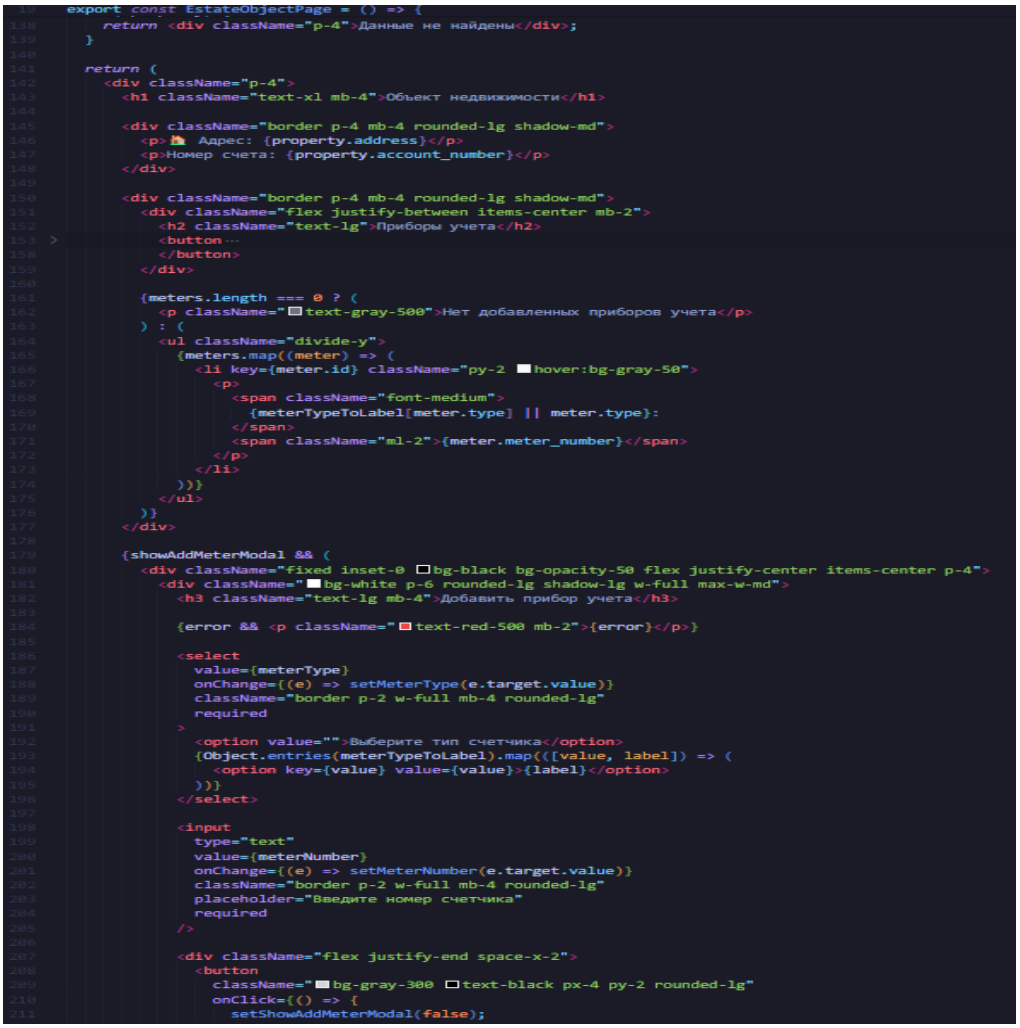


Рисунок 34 – Реализация формы добавления объекта недвижимости.

Компонент “AddProperty” представляет пользователю форму для ввода данных нового объекта недвижимости. В форме реализована валидация обязательных полей, таких как ИНН управляющей компании и номер лицевого счета. После заполнения формы при отправке данных вызывается функция “addProperty” из модуля “PropertyAPI.ts”, отправляющая POST-запрос на сервер. После успешной отправки формы происходит сброс полей ввода и отображение уведомления об успешном добавлении объекта. На следующем рисунке 33 изображена кликабельная карточка объекта недвижимости, а также модальное окно с добавлением прибора учета.

В этом разделе представлена реализация мобильного приложения для расчета стоимости коммунальных услуг. Разработан сервер на FastAPI с модульной структурой и аутентификацией через JWT, а также веб-клиент на React с интеграцией API. Архитектура приложения обеспечивает безопасность и удобство расширения. Следующим этапом будет тестирование работоспособности системы.

3.4 Тестирование программного обеспечения

Для проверки работоспособности мобильного приложения для расчета стоимости коммунальных услуг была выбрана методология тест-кейсов. Основная цель тестирования состоит в определении отклонений при реализации функциональных требований, степени их значимости, обнаружении ошибок в ходе выполнения программ и своевременном их исправлении [1]. Как отмечается в современных исследованиях, «жизненный цикл тестирования строится на итеративности: каждая фаза начинается с планирования и завершается отчетностью, которая влияет на следующие улучшения [24]».

В качестве примера, в таблице 3 представлен тест-кейс, ориентированный на проверку ключевого функционала, связанного с расчетом и оплатой коммунальных услуг. В нем рассмотрены основные этапы работы: выбор

объекта недвижимости ввод показаний счетчиков и автоматический расчет суммы к оплате. Особое внимание уделено валидации вводимых данных и точности расчетов.

Таблица 3 – Тест-кейс проверки функциональности

№	**Название теста**	**Действие**	**Ожидаемый результат**	**Фактический результат**
1	Регистрация нового пользователя	Ввести корректные данные при регистрации	Пользователь успешно зарегистрирован	Совпадает
2	Регистрация без заполнения обязательных полей	Попытаться зарегистрироваться оставив поле e-mail пустым	Система выдает ошибку: “Поле e-mail обязательно для заполнения”	Совпадает
3	Авторизация пользователя	Ввести корректные e-mail и пароль	Успешная авторизация и переход на главную страницу	Совпадает
4	Ошибочная авторизация	Ввести неверный пароль	Сообщение об ошибке	Совпадает
5	Открытие главной страницы	После авторизации открыть главную страницу приложения	Отображаются объекты недвижимости	Совпадает
6	Добавление нового объекта недвижимости	Нажать на кнопку «+», заполнить все поля	Новый объект успешно добавляется	Совпадает
7	Добавление объекта без заполнения ИНН	Оставить поле ИНН пустым и попытаться добавить объект	Система выдает ошибку: “Поле ИНН обязательно для заполнения”	Совпадает

Продолжение таблицы 3.

8	Просмотр показаний счетчиков	Перейти в раздел “Показания счетчиков”	Отображаются доступные счетчики по лицевым счетам	Совпадает
9	Ввод показаний для счетчика	Ввести новое значение для счетчика “Электричество день”	Показания успешно сохраняются в базе данных	Совпадает
10	Отправка пустого значения показаний	Оставить поле пустым и попытаться отправить	Система выдает ошибку “Это поле обязательно для заполнения”	Совпадает
11	Просмотр квитанций	Перейти в раздел “Квитанции”	Отображаются квитанции по объектам недвижимости	Совпадает
12	Оплата квитанций	Перейти в раздел “Оплата”, выбрать объект недвижимости и нажать оплатить	Квитанция помечается как «Оплачено»	Совпадает
13	Просмотр профиля пользователя	Открыть страницу профиля	Корректное отображение e-mail и кнопок управления профилем	Совпадает
14	Выход из учетной записи	Нажать кнопку “Выйти из системы” в профиле	Пользователь выходит из системы и попадает на экран выхода	Совпадает

Все тест-кейсы, представленные в таблице 3, были успешно пройдены. Фактические результаты полностью соответствуют ожидаемым, что свидетельствует о корректной работе приложения в стандартных условиях эксплуатации. В процессе тестирования особое внимание уделялось проверке устойчивости работы интерфейса при различных сценариях взаимодействия, а также точности расчетов сумм коммунальных платежей на основе введенных пользователями данных. Ошибок, приводящих к сбоям или нарушению логики работы приложения выявлено не было.

Проведенное тестирование подтверждает, что мобильное приложение для расчета стоимости коммунальных услуг соответствует всем предъявляемым требованиям, демонстрирует стабильную работу в стандартных условиях эксплуатации и готово к дальнейшему использованию. Для наглядной демонстрации успешного прохождения ключевых этапов тестирования были выполнены скриншоты интерфейса приложения. На рисунке 35 представлен процесс успешной регистрации нового пользователя. Пользователь вводит все необходимые данные, приложение корректно завершает регистрацию, что подтверждается отсутствием ошибок на экране.

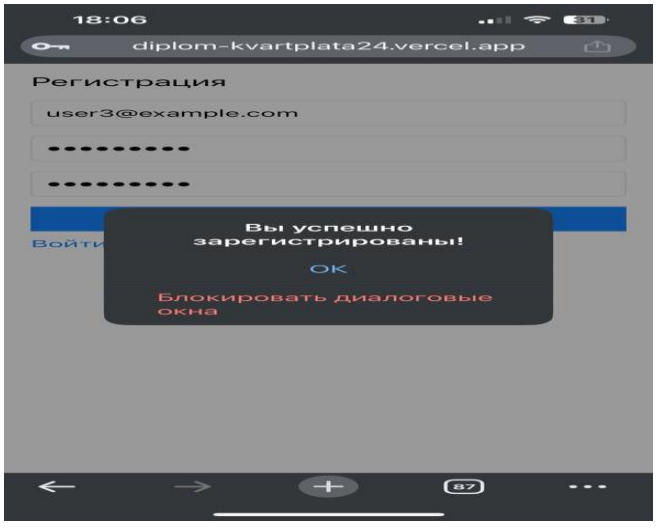


Рисунок 35 – Результат регистрации нового пользователя.

Как видно на изображении, все поля корректно заполняются, система безошибочно завершает процесс и выдает сообщение об успехе. Этот шаг подтверждает, что функциональность регистрации работает стабильно. Далее, на рисунке 36 показан процесс авторизации с неверными данными. В этом случае система должна вывести ошибку.

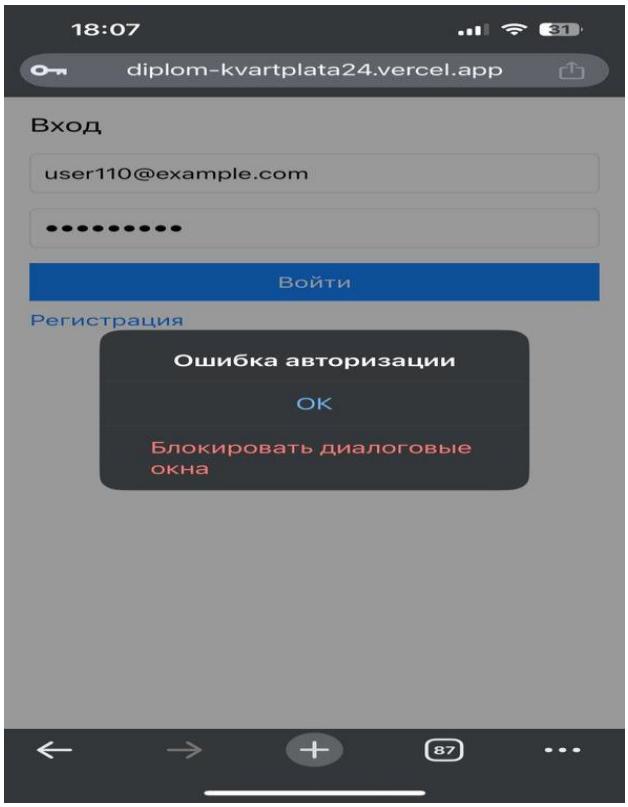


Рисунок 36 – Результат попытки входа в систему с неверными данными.

Данный скриншот демонстрирует, как приложение корректно обрабатывает ошибочные данные и информирует пользователя о проблемах с авторизацией. Это подтверждает, что система правильно распознает и реагирует на некорректный ввод данных.

Вывод по третьей главе

В данной главе был осуществлена реализация мобильного приложения для расчета стоимости коммунальных услуг. Проведен выбор, а также

обоснование используемого технологического стека: React и Bootstrap для клиентской части, FastAPI и PostgreSQL – для серверной логики и хранения данных. Разработан удобный пользовательский интерфейс, включающий все ключевые экраны – от авторизации и регистрации до расчета стоимости услуг и отображения квитанций. Также была реализована архитектура клиент-серверного взаимодействия с использованием JWT-аутентификации, Rest API и ORM-моделей.

Проведенные тесты доказывают, что приложение функционирует как ожидается, а все ключевые этапы работы – от регистрации до авторизации и работы с основным функционалом проходят без ошибок, что подтверждает готовность приложения к использованию в реальных условиях.

Заключение

Итогом данной бакалаврской работы является мобильное приложение, предназначенное для расчета стоимости коммунальных услуг. В приложении реализованы основные функции: расчет коммунальных платежей на основе введенных данных, отображение истории расчетов, а также удобный интерфейс для работы пользователей.

В ходе работы был проведен анализ предметной области, исследованы существующие решения, выявлены их преимущества и недостатки. На основе анализа были сформулированы основные требования к разрабатываемому приложению. Разработана контекстная диаграмма с декомпозицией основных бизнес-процессов, построена диаграмма потоков данных, что позволило определить ключевые информационные потоки системы. Были подробно описаны принципы работы мобильного приложения, реализована архитектура с использованием FastAPI для серверной части, PostgreSQL для надежного хранения данных и React для динамичной клиентской части. Осуществлено тестирование мобильного приложения на различных устройствах с операционной системой Android, что подтвердило корректность работы основных функций, таких как ввод данных, расчет коммунальных платежей и отображение истории транзакций. Разработанное мобильное приложение предоставит пользователям удобный инструмент для расчета коммунальных платежей, ведения учета расходов и анализа истории платежей за определенные периоды.

Оно будет интуитивно понятным и доступным для пользователей всех возрастов и уровней технической подготовки. В дальнейшем также возможна доработка функционала, включая автоматическую интеграцию с актуальными базами данных тарифов, поддержку push-уведомлений, а также возможность генерации отчетов в удобном формате. Это позволит пользователям еще более эффективно управлять своими финансами и контролировать свои расходы.

Список используемой литературы

1. Белик А.Г., Цыганенко В.Н. Проектирование и архитектура программных систем. – Омск: Изд-во ОмГТУ, 2016. – 96 с.
2. Вандеркам Д. Эффективный TypeScript: 62 способа улучшить код. – М.: Диалектика, 2021. – 272 с.
3. Дейтел П. Python для программистов. – М.: Питер, 2022. – 720 с.
4. Дейт К.Дж. Введение в системы баз данных. – М.: Вильямс, 2021. – 1328 с.
5. Златопольский Д.М. Основы программирования на Python. – М.: ДМК Пресс, 2023. – 390 с.
6. Куликов С.В. Тестирование ПО. Базовый курс. – М.: ДМК Пресс, 2022. – 298 с.
7. Куликов С.В., Куликова Л.А. Моделирование бизнес-процессов с использованием BPMN 2.0 // Информационные технологии. – 2021. – Т. 27, № 4. – С. 234-241.
8. Ларман К. Применение UML. – М.: Вильямс, 2019. – 736 с.
9. Орлов С.А. Технологии разработки ПО. – СПб.: Питер, 2021. – 416 с.
10. Прохоренок Н.А. Python 3 и PyQt 5. – СПб.: БХВ, 2022. – 704 с.
11. Скиена С. Алгоритмы. – М.: Вильямс, 2020. – 720 с.
12. Таненбаум Э. Архитектура компьютера. – СПб.: Питер, 2021. – 848 с.
13. Хортон А., Вайс Р. Разработка веб-приложений в ReactJS / пер. с англ. Р. Н. Рагимова. — 2-е изд., электронное. — М.: ДМК Пресс, 2023. — 255 с.
14. Черняк Л. Базы данных для разработчиков. – М.: БИНОМ, 2022. – 416 с.
15. Яценков В.С. Современные фреймворки. – М.: ДМК Пресс, 2021. – 274 с.

16. Fowler M. Patterns of Enterprise Architecture. – Boston: Addison-Wesley, 2002. – 560 p.

17. Newman S. Building Microservices. – Sebastopol: O'Reilly, 2015. – 288 p.

18. Sommerville I. Software Engineering. – Harlow: Pearson, 2021. – 864 p.

19. Martin R. Clean Code. – Boston: Prentice Hall, 2008. – 464 p.

20. Gamma E. Design Patterns. – Boston: Addison-Wesley, 1994. – 416 p.

21. Documentation FastAPI [Электронный ресурс]. – URL: https://fastapi.tiangolo.com/ru/ (дата обращения: 15.06.2024)

22. Documentation React [Электронный ресурс]. – URL: https://ru.reactjs.org/docs/ (дата обращения: 15.06.2024)

23. Data Flow Diagrams: A Practical Guide [Электронный ресурс] // Systems Education. – URL: https://systems.education/data-flow-diagrams (дата обращения: 15.06.2024)

24. Google Developers. PWA [Электронный ресурс]. – URL: https://developers.google.com/web/progressive-web-apps (дата обращения: 15.06.2024)

25. RFC 7519. JSON Web Token (JWT) [Электронный ресурс] // IETF. – Май. – URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата обращения: 15.06.2024)