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 управления процессом обслуживания клиентов»

Студент Р.Р. Каримов
(И.О. Фамилия) (личная подпись)

Руководитель к.э.н., доцент, Т.А. Раченко
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент С.А. Гудкова

Тольятти 2024

Аннотация

Тема бакалаврской работы – «Разработка мобильного приложения для управления процессом обслуживания клиентов»

Цель бакалаврской работы - разработка функционального мобильного приложения с возможностью ведения учета показателей счетчиков, автоматическим расчетом оплаты предоставляемых услуг и расчетом предположительных начислений за следующий месяц.

Объектом исследования является процесс учета вводимых энергоресурсов, автоматический расчет оплаты и расчет предположительного начисления за следующий месяц.

Предметом исследования является мобильное приложение, разрабатываемое для упрощения и улучшения процесса управления обслуживанием клиента.

Структура работы представлена введением, 3 главами, заключением, списком литературы и приложением.

В первой главе представлены описание предметной области, цель и задачи разработки мобильного приложения.

Во второй главе идет проектирование мобильного приложения.

В третьей главе идет реализация и тестирование мобильного приложения.

Разработано мобильное приложение и сервер, с возможностью взаимного взаимодействия для ведения учета энергоресурсов и расчета прогнозируемых начислений.

Работа состоит из 66 страниц текста, 42 рисунков, 3 таблицы, 1 приложения и списка из 25 источников.

Abstract

Bachelor Thesis Theme: "Development of a Mobile Application for Managing the Customer Service Process"

The aim of the bachelor thesis is to develop a functional mobile application with the capability to record meter readings, automatically calculate the payment for provided services, and estimate the charges for the following month.

The object of the research is the process of accounting for input energy resources, automatic payment calculation, and estimation of the charges for the next month.

The subject of the research is the mobile application, which is being developed to simplify and improve the process of managing customer service.

The structure of the work includes an introduction, three chapters, a conclusion, a list of references, and an appendix.

The first chapter presents a description of the subject area, the goal, and the objectives of developing the mobile application.

The second chapter covers the design of the mobile application.

The third chapter deals with the implementation and testing of the mobile application.

A mobile application and server have been developed, with the ability to interact mutually for accounting energy resources and calculating forecasted charges.

The work consists of 63 pages of text, 42 figures, 3 tables, 1 appendices, and a list of 25 references.

Оглавление

Введение

В современном мире, ключевым фактором успеха для любого предприятия является эффективное обслуживание клиентов. Для этого создаются приложения, которые обеспечивают клиента полным функционалом для работы с предоставляемым сервисом. Создание такого мобильного приложения для функционирования с клиентом, это одно из самых важных задач для предприятия в наши дни, так как смартфон имеется у каждого и это упрощает работу процесса обслуживания. Данная выпускная квалификационная работа как раз посвящена этой теме, ведь развитие мобильных технологий уже привело к появлению и внедрению их в сферы бизнеса и является актуальной задачей для решения. Каждый клиент должен быть удовлетворен разработанным сервисом, так как это напрямую влияет на репутацию предприятия, а также это увеличивает их конкурентоспособность на рынке.

Целью выпускной квалификационной работы является разработка функционального мобильного приложения для управления процессом обслуживания клиентов с возможностью ведения учета показателей счетчиков, автоматическим расчетом оплаты предоставляемых услуг и расчетом предположительных начислений за следующий месяц с использованием современных информационных технологий.

Объектом исследования является процесс учета вводимых энергоресурсов, автоматический расчет оплаты и расчет предположительного начисления за следующий месяц.

Предметом исследования является мобильное приложение, разрабатываемое для упрощения и улучшения процесса управления обслуживанием клиента.

Данная работа включает в себя следующие задачи, которые нужно решить для достижения цели:

– провести характеристику ООО «Кварплата 24»;
– обоснование необходимости управления процессом обслуживания клиентов;
– определение целей и задач для разработки мобильного приложения.
– определение функциональных и нефункциональных требований приложения;
– провести анализ и выбор технологии и подходов, которые используемых при разработке мобильного приложения.
– разработать схему базы данных;
– спроектировать архитектуру и пользовательский интерфейс разрабатываемого приложения;
– разработать и протестировать мобильное приложение.

Таким образом, для реализации поставленных задач, будут использованы методы анализа, проектирования, а также метод тестирования.

Структура работы включает в себя три главы. В первой главе приведен анализ предприятия, анализ предметной области и постановка задачи.

Во второй рассмотрены моделирование процессов приложения, проектирование основных функций, а также выбор главного средства, для реализации разработки.

Третья глава посвящена реализации функций приложения, разработка пользовательского интерфейса, тестирование приложения и разбор его результатов.

Глава 1. Описание предметной области. Цель и задачи разработки мобильного приложения

1.1 Характеристика ООО «Кварплата 24» и ее организационная структура”

ООО «Кварплата 24» – ИТ – компания, которая занимается разработкой сервисов, предоставляемых управляющим компаниям в сфере жилищно-коммунальных услуг. Компания была основана в 1996 году и на данный момент обслуживает более 1 миллиона лицевых счетов по всей России, сотрудничая с более чем 1000 ТСЖ (товариществ собственников жилья), управляющими организациями (УО) и ресурсоснабжающими организациями (РСО) в 70 субъектах РФ.

Компания предоставляет следующие решаемые задачи: “ задачи расчета и учета платы за жилищно-коммунальные услуги, приема и распределения платежей, востребования долгов в полном соответствии с законодательством для товариществ собственников жилья, управляющих и ресурсоснабжающих организаций, информационно-расчетных центров на всей территории РФ” [1].

Таким образом главной миссией организации «ООО Кварплата 24» является создание удобной и надежной платформы, обеспечивающей удобство и эффективность в управлении жилищно-коммунальными услугами с помощью онлайн-сервисов.

Далее рассмотрим организационную структуру предприятия представленную на рисунке 1.

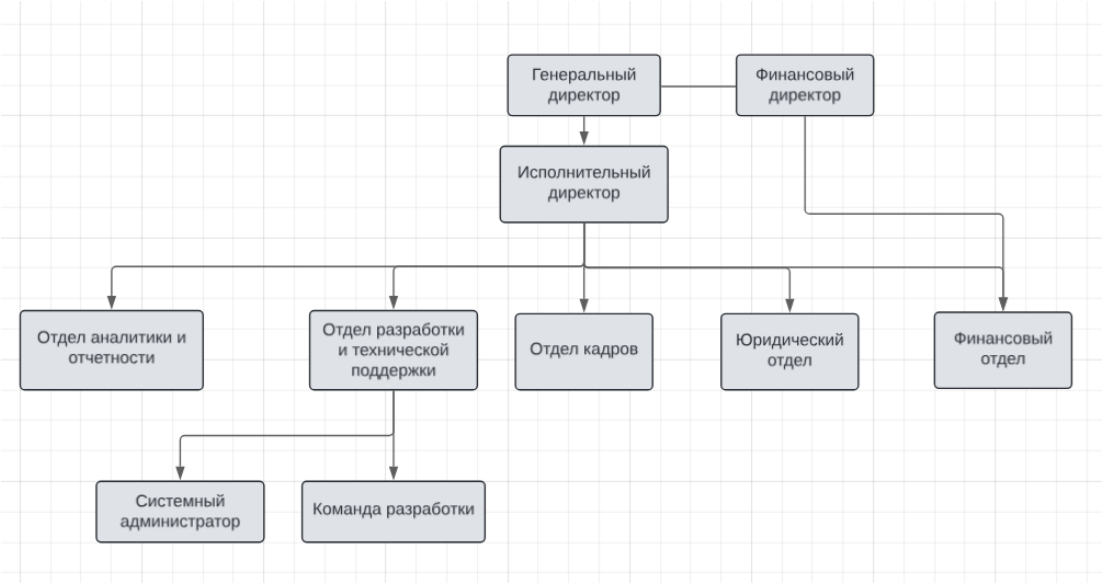


Рисунок 1 - Организационная структура компании «Кварплата 24»

Структура состоит из четырех уровней управления, с помощью которого обеспечивается высокий уровень предоставляемых услуг. Первый уровень состоит из генерального директора, который ответственен за стратегическое руководство, принимает ключевые решения и определяет общую стратегию развития компании. «Директор ООО «Кварплата24» несет ответственность за руководство и управление ее деятельностью» [1].

На следующем уровне находится исполнительный директор, который координирует повседневные операции компании и следит за выполнением стратегических целей и задач, поставленных генеральным директором.

На третьем уровне находятся основные подразделения: отдел разработки и технической поддержки, отдел аналитики и отчетности, юридический отдел и отдел кадров. Отдел разработки и технической поддержки отвечает за разработку и поддержку программного обеспечения, обеспечивает техническую поддержку клиентов. Отдел аналитики и отчетности проводит анализ данных, подготавливает отчеты для руководства и клиентов, а также выявляет тенденции и предлагает улучшения.

На четвертом уровне находятся специалисты и сотрудники, которые в зависимости от конкретного отдела выполняют соответствующие функции.

Офис компании оснащен всем необходимым для обеспечения эффективной работы и предоставления высококачественных услуг:

- компьютеры с установленным и настроенным программным обеспечением для работы с платежными системами, базами данных и расчетами,
- сетевое оборудование,
- принтеры,
- рабочие телефоны.

Таким образом эффективная работа компании обеспечивается благодаря современному оборудованию, четкой и слаженной организационной структуре и работе опытных и квалифицированных сотрудников. «Компания постоянно совершенствует свои сервисы и внимательно следит за рыночными изменениями, чтобы предоставлять своим клиентам наиболее актуальные и выгодные условия» [1]. Все это позволяет компании успешно выполнять свои функции и предоставлять клиентам качественные услуги в сфере жилищно-коммунальных хозяйства.

1.2 Обоснование необходимости управления процессом обслуживания клиентов

В данной главе начинается исследование темы управления процессом обслуживания клиентов в сфере жилищно-коммунального хозяйства с использованием мобильных технологий. Главной задачей подраздела это анализ процесса обслуживания клиентов.

В наши дни технологический прогресс не стоит на месте. Каждое предприятие внедряет в свою структуру процессы для решения проблем, связанных с улучшением процесса обслуживания клиентов. В связи с

развитием информационных технологий и увеличении числа клиентов, компании принимают решения о необходимости менять устаревшие и малоэффективные шаблоны, которыми они пользовались в недалеком прошлом.

Такие перемены, предпринимаемые в рамках улучшения процессов обслуживания, называется «Цифровизация». Так как выпускная квалификационная работа направлена на разработку мобильного приложения, то и речь пойдет именно о ней. «Цифровизация – это процесс превращения аналоговых данных и рабочих процессов в цифровой формат. Она включает в себя использование цифровых технологий для автоматизации бизнес-процессов, улучшения уровня качества услуг, оптимизации производства и повышения эффективности работы организаций и предприятий в целом» [2].

Цифровизация в сфере ЖКХ предполагает внедрение прикладных технологий для автоматизации различных процессов, таких как:

- прогнозирование,
- учет потребления ресурсов,
- начисление и прием платежей,
- документооборот в компании,
- прием заявок и распределение услуг на капитальный ремонт,
- обратная связь,
- предоставление дополнительных услуг.

Внедрение таких цифровых технологий в сферу жилищно-коммунальных услуг позволяет создать единое информационное поле, объединяющее как жителей одного дома, так и весь город, в частности. Так как количество процессов в сфере обширное и каждым процессом занимается разные подразделения.

Мобильное приложение существенно уменьшает необходимость в ручной обработке данных и использовании бумажных документов. Автоматизация процессов ввода данных, их обработки и хранения позволяет

снизить расходы на канцелярские товары, печать и архивирование документов. Электронный документооборот также ускоряет обмен информацией, что сокращает временные затраты сотрудников и позволяет направить их усилия на выполнение более стратегически важных задач. Таким образом, компания может существенно сократить операционные затраты и повысить общую эффективность работы.

Внедрение мобильного приложения не только приносит экономические выгоды для управляющей компании, но и оказывает положительное социальное влияние, улучшая взаимодействие с клиентами и повышая качество их жизни. Оно позволяет жителям удобнее и быстрее управлять своими коммунальными услугами используя возможность легко вводить показания счетчиков, оплачивать счета и получать актуальную информацию о расходах, что делает управление коммунальными услугами менее стрессовым и более предсказуемым.

Прозрачность и точность информации, которая предоставляется мобильным приложением, способствуют укреплению доверия между клиентами и управляющей компанией. Когда пользователи видят, что их данные обрабатываются быстро и без ошибок, и что они всегда могут получить подробную информацию о своих платежах и потреблении ресурсов, они начинают больше доверять компании. Это доверие важно для долгосрочных отношений и стабильности клиентской базы.

Мобильное приложение становится инструментом для активного вовлечения жителей в управление их домами и районами. Это повышает чувство ответственности у жителей, способствует более активному участию в общественной жизни.

Таким образом для дальнейшего повышения эффективности обслуживания клиентов в сфере ЖКХ необходимо разработать мобильное приложение, которое будет интегрировано с существующей системой управления и учета, что позволит обеспечить эффективность работы управляющих компаний, улучшить качество обслуживания клиентов.

1.3 Анализ модели бизнес-процесса «Управление платежами»

В данном подразделе проведем анализ бизнес-модели «Управление платежами», так как этот процесс является основным и будет использован при разработке приложения для управления процессом обслуживания клиентов.

Здесь будет представлена бизнес-модель, которая существует на сегодняшний день. Модель «КАК ЕСТЬ» в бизнес-процессах представляет текущее состояние или существующий способ выполнения процессов в организации. Она описывает, как процессы функционируют на данный момент без изменений или оптимизации. А также модель бизнес-процесса «КАК ДОЛЖНО БЫТЬ», которая является одной из ключевых этапом в создании автоматизированной системы, направленной на оптимизацию работы и повышение эффективности организации.

Для анализа модели и выявления сильных и слабых сторон процесса будет использована методология IDEF0.

«IDEF0 — методология функционального моделирования (англ. function modeling) и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является ее акцент на соподчиненность объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность (поток работ)» [3].

Модель «КАК ЕСТЬ» бизнес-процесса «Управление платежами» в IDEF0 представлена на рисунках 2 и 3.

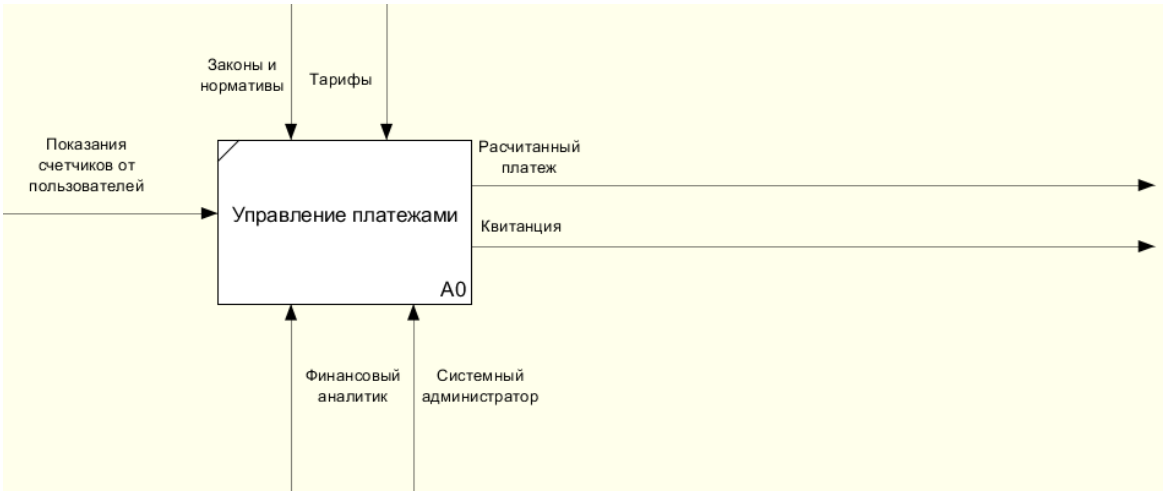


Рисунок 2 - Модель «КАК ЕСТЬ» бизнес-процесса «Управление платежами» в IDEF0

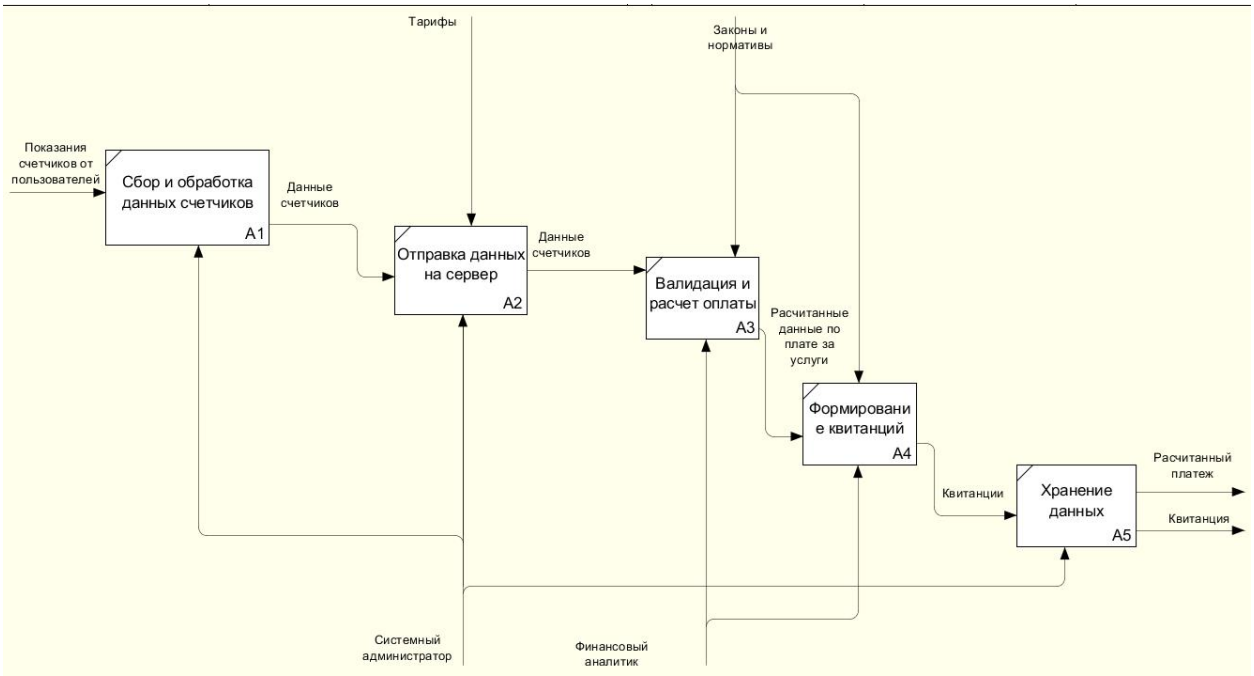


Рисунок 3 - Декомпозиция блока A0 функциональной модели «Управление платежами»

Она представляет собой последовательность шагов, включающих в себя сбор и обработку данных счетчиков, отправку данных на сервер, расчет оплаты, формирование квитанций и хранение данных. Каждый этап

выполняется с помощью специальных сотрудников, основным из которых является системный администратор, с помощью которого ведется управление серверным программным обеспечением, а также финансовый аналитик, следящий за соблюдением расчета оплаты и формировании квитанций. Механизмы обеспечивают качественное обслуживание с соблюдением законодательства и производят защиту данных. Входными данными модели являются показания счетчиков от пользователя, а выходными рассчитанный платеж и сформированные квитанции, которые в дальнейшем будут отправлены пользователю.

Исходя из анализа модели «КАК ЕСТЬ», можно выявить следующие слабые стороны:

- ограниченные возможности обратной связи: на этапе сохранения данных не учтены механизмы учета энергоресурсов и обработки обратной связи от пользователей для улучшения качества услуг;
- не производится расчет предположительной платы за следующий месяц: не учтены функции расчета предположительной платы за следующий месяц, что позволило бы пользователям получать прогнозируемую информацию о предстоящих расходах на коммунальные услуги. Это помогло бы клиентам грамотно распоряжаться своими расходами либо экономить на потреблении энергоресурсов.

Решение этих проблем будет осуществляться через разработку приложения, которое позволит повысить эффективность процесса обслуживания клиентов.

Измененная модель «КАК ДОЛЖНО БЫТЬ» бизнес-процесса «Управление платежами» в IDEF0 представлены на рисунке 4 и 5.

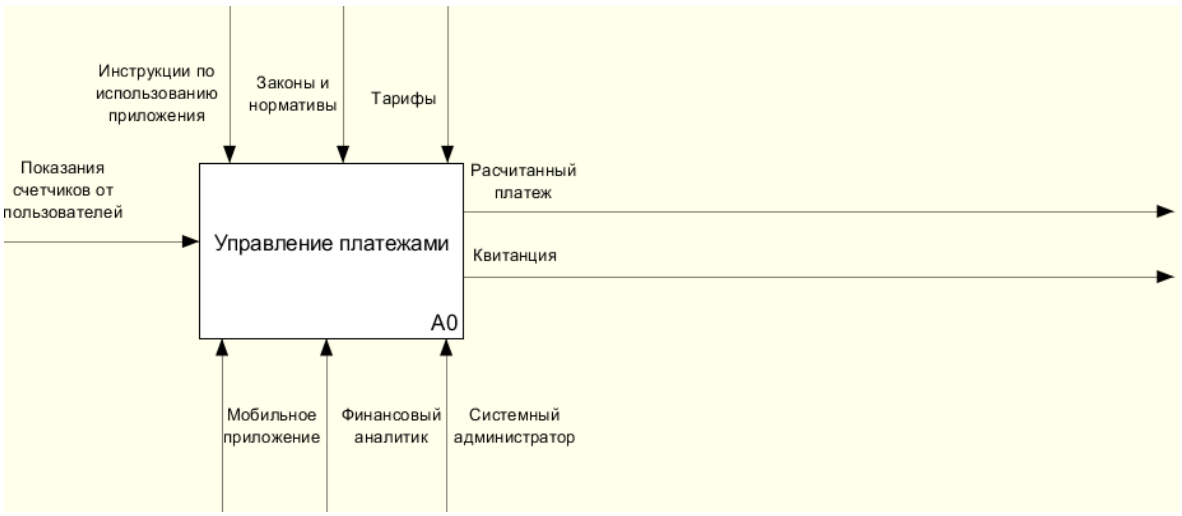


Рисунок 4 - Модель «КАК ДОЛЖНО БЫТЬ» бизнес-процесса «Управление платежами» в IDEF0.

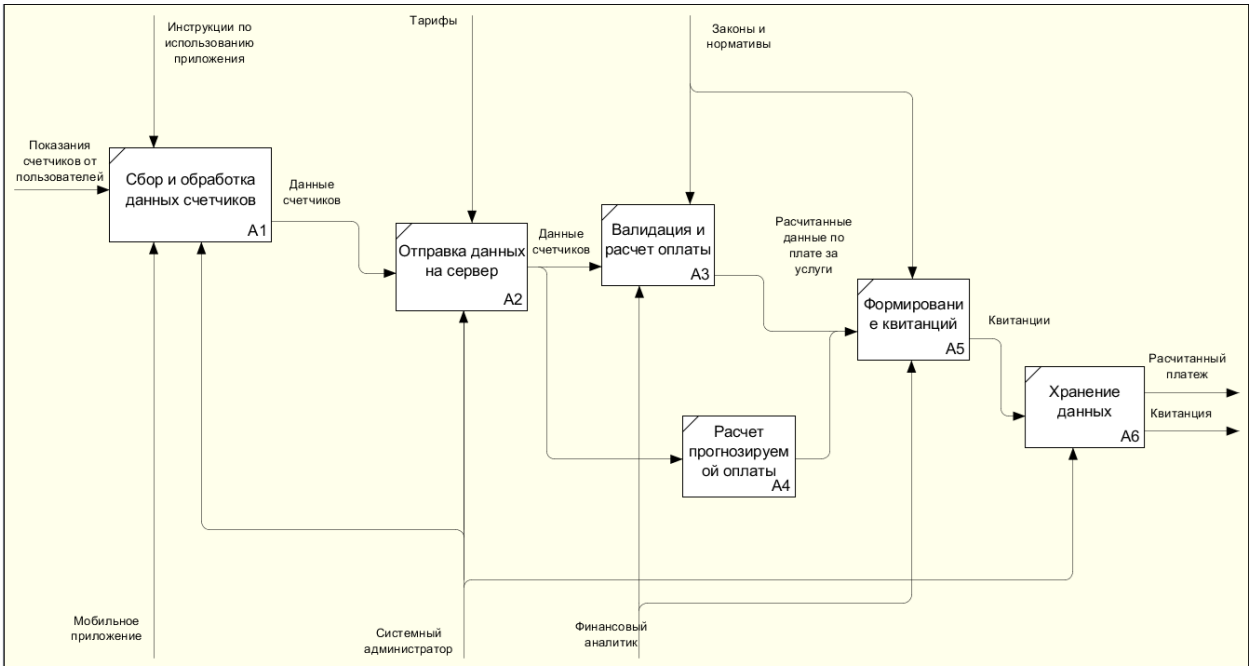


Рисунок 5 - Декомпозиция блока A0 функциональной модели «Управление платежами»

Таким образом, в модель добавлено приложение с помощью, которого будут собираться данные, а также расчет прогнозируемой оплаты.

В данном подразделе был проведен анализ бизнес-процесса «Управление платежами», в котором были выявлены слабые стороны, устранение которых улучшит взаимодействие с клиентами.

1.4 Анализ существующих решений

В данном подразделе будут представлены существующие решения, анализ которых поможет создать новое. Рассмотрим самые популярные из них

«Энергомера» представляет собой систему измерений, предназначенную для мониторинга и учета потребляемой электроэнергии и мощности. Эта система собирает, сохраняет, обрабатывает и визуализирует данные о потреблении энергии, предоставляя информацию о текущем использовании электроэнергии. «ИИС «Энергомера» применяется на энергетических объектах розничного рынка электроэнергии, промышленных предприятиях, коммунально-бытовом хозяйстве и мелкомоторном секторе» [4].

«Энфорс Лайт» — это программное обеспечение, разработанное для коммерческого учета электроэнергии. Оно автоматически собирает данные с приборов учета и сохраняет их в базе данных, используя специальные каналы связи. Основная функция программы — объединение данных с различных приборов учета в единую базу, а также анализ этих данных и создание отчетов. «Продукт разработан для минимизации затрат при создании систем учета электроэнергии для исполнения требований Постановления Правительства РФ № 442 о новых правилах розничного рынка» [5].

«ЭНЕРГОАУДИТКОНТРОЛЬ» — это автоматизированные системы для контроля и учета электроэнергии. Эти системы выполняют функции автоматического сбора данных с приборов учета и их передачи на сервер, анализа информации о потреблении энергии на предприятии для оптимизации, а также удаленного подключения и отключения конечных потребителей от

сети. «Специализируется на создании автоматизированных систем коммерческого учета, необходимых для обеспечения прозрачной системы взаиморасчетов между поставщиком и потребителем ресурсов, а также систем технического учета (АСТУЭ), которые позволяют решать проблемы неэффективного использования энергоресурсов» [6].

«ПУМА» — это автоматизированная система для коммерческого учета электроэнергии. Она формирует единый информационный ресурс, обеспечивающий удобный доступ к данным о расходах электроэнергии. Система автоматически собирает информацию со всех типов счетчиков и доступна как Web-сервис через обычный браузер. В дополнение к этому система включает сервис для сбора и хранения данных, а также построения графиков и отчетов. «Компания предлагает полный комплекс работ по внедрению автоматического учёта энергоресурсов на ваших объектах. Он поможет оптимизировать потребление ресурсов и сократить эксплуатационные расходы» [7].

Для более точного анализа данные о существующих решениях добавлены в таблицу 1, используя методику FURPS. «FURPS — классификация требований к программным системам:

- функциональные требования: свойства, возможности, безопасность, являются основными, по этим требованиям строятся диаграммы вариантов использования;
- требования к удобству использования (UX): человеческий фактор, эстетика, последовательность, документация;
- требования к надежности: частота возможных сбоев, отказоустойчивость, восстанавливаемость, предсказуемость устойчивости;
- требования к производительности: время отклика, использование ресурсов, эффективность, мощность, масштабируемость;

– требования к поддержке: возможность поддержки, ремонтопригодность, гибкость, модифицируемость, модульность, расширяемость, возможность локализации» [8];

Таблица 1 – Сравнение существующих решений

Критерий	«Энергомера»	«Энфорс Лайт»	«ЭНЕРГО-АУДИТКОНТРОЛЬ»
Функциональность	Контроль и учет электроэнергии, автоматизированный сбор, хранение и обработка данных	Сбор данных с приборов учета, занесение в базу, анализ и формирование отчетов	Автоматический сбор данных с приборов учета, анализ энергопотребления, удаленное подключение/отключение
Удобство использования	Средний уровень удобства, требуется специальное обучение	Достаточно удобна, но требуется специализированные каналы связи	Средний уровень удобства, интерфейс требует обучения
Надежность и безопасность	Умеренно надежна, хорошая защита данных, но только для электроэнергии	Надежность высокая, хорошая защита данных	Высокая надежность, требуется специализированное оборудование для обеспечения безопасности
Производительность	Высокая производительность в части сбора и обработки данных, ограниченная многозадачность	Высокая производительность при сборе данных, требует специальных каналов связи	Высокая производительность, эффективный сбор и анализ данных
Поддерживаемость	Ограниченная масштабируемость, требуется специальное оборудование	Ограниченная масштабируемость, подходит только для специализированных устройств	Хорошая масштабируемость, но требует специального оборудования

Благодаря обзору существующих решений можно выявить несколько недостатков:

– в большинстве решений не ведется полный учет энергоресурсов, таких как теплоэнергия и водоснабжение;
– крупные системы по контролю учета обладают ненужным перечнем функциональных возможностей, которые не являются полезными в рамках работы;
– для использования некоторых систем, нужны специализированные приборы учета, а для взаимодействия с сервером, нужна соответствующая инфраструктура.

В данном подразделе был проведен полный анализ существующих приложений, которые взаимодействуют с процессом обслуживания клиентов. В следующем подразделе будут выявлены функциональные и нефункциональные требования к разрабатываемому приложению, в котором будут учтены все недостатки, которые были выявлены в обзоре существующих решений.

1.5 Требования к мобильному приложению для управления процессом обслуживания клиентов

В подразделе проводится описание требований к разрабатываемому мобильному приложению для управления процессом обслуживания клиентов. Требования определяют какие-либо функции, возможности или задачи мобильного приложения, которые поясняют, как приложение должно вести себя в различных ситуациях. Требования подразделяются на функциональные и нефункциональные.

Функциональные требования – это требования, которые описывают конкретное поведение системы. «Эти требования обычно определяются через сценарии использования, пользовательские истории или спецификации» [9]. Главной задачей функциональных требований предоставить все функции, которые удовлетворяют всем потребностям клиентам.

Нефункциональные требования – это требования, которые как раз таки накладываются на разрабатываемую систему и описывают то, как система должна выполнять свои функции. «Нефункциональные требования касаются таких вопросов, как масштабируемость, ремонтопригодность, производительность, переносимость, безопасность, надежность и многие другие» [10]. Требования имеют как свои преимущества, в виде того, что помогают разработчикам убедиться, что система отвечает всем потребностям клиента, но также и недостатки, в виде того, что при неправильной постановке их бывает трудно понять и реалтзовать.

Основываясь на предыдущем анализе бизнес-процесса в подразделе 1.3, разделим классификацию требований FURPS на функциональные и нефункциональные.

а) Функциональные требования:

1) Функциональность:

- клиент должен иметь возможность авторизации;
- клиент должен иметь возможность вводить данные счетчиков;
- клиент должен видеть свои предыдущие показания;
- данные, вводимые клиентом, должны отправляться на сервер;
- система должна рассчитывать текущую плату на основе введенных данных счетчиков;
- система должна рассчитывать предположительную плату за следующий месяц на основе текущих данных и исторической информации;
- приложение должно иметь возможность получать данные с сервера;
- пользователь должен получать уведомления о результатах действия;

б) Нефункциональные требования:

1) Удобство использования:

- приложение должно иметь интуитивно понятный интерфейс для ввода данных;
- навигация должна быть простой и логичной;

2) Надежность:
   - приложение должно быть доступно 24/7;
   - должна сохраняться точность при расчетах в зависимости с установленными тарифами и правилами;
3) Производительность:
   - сервер должен быть способен обрабатывать большое количество данных;
   - приложение должно бысро обрабатывать запросы пользователя:
4) Поддерживаемость:
   - приложение должно поддерживаться на различных мобильных устройствах, минимально используя память и процессор;
   - возможность обработки ошибок.

Таким образом, на основе разработанных требований можно построить диаграмму прецендентов (диаграмма вариантов использования). «Диаграмма вариантов использования в UML — диаграмма, отражающая отношения между акторами (пользователями) и прецендентами, являющаяся составной частью модели прецендентов, позволяющей описать систему на концептуальном уровне» [11]. Эта диаграмма представлена на рисунке 6.

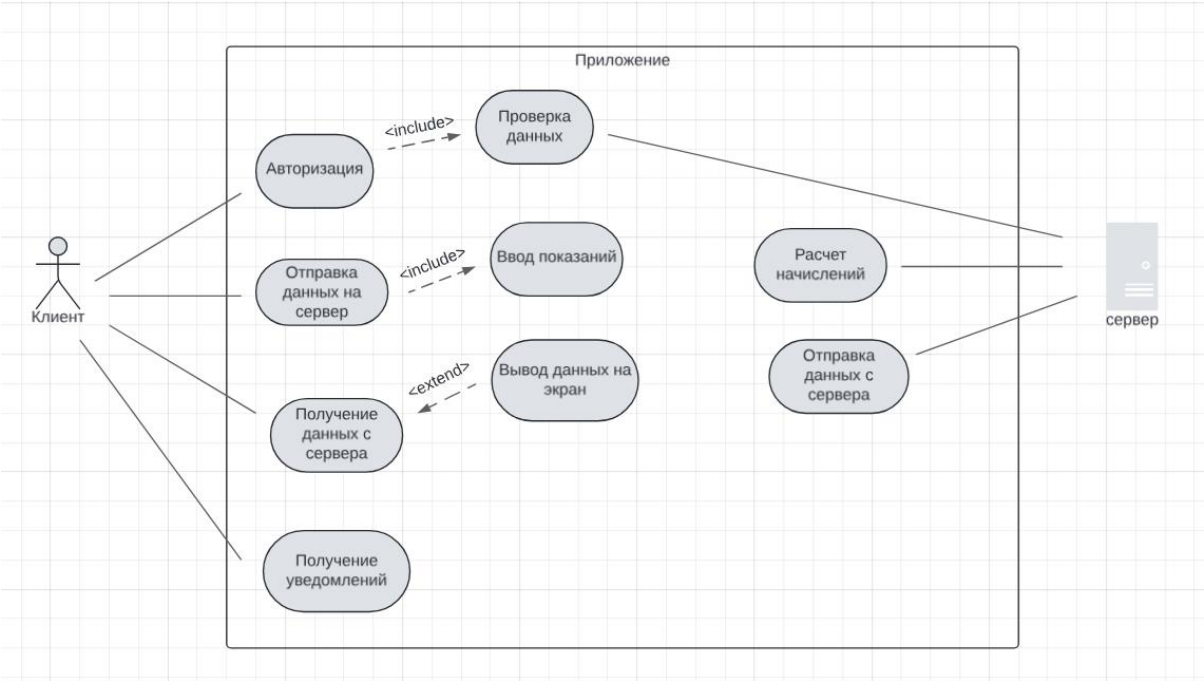


Рисунок 6 – Диаграмма прецендентов

«Диаграмма складывается из следующих компонентов:

- актер (Actor),
- собственно use-кейсы,
- ассоциации между ними,
- границы системы (Boundaries),
- отношения (Relationships или связи)» [19].

На диаграмме представлены два актера и их действия в приложении. Клиент взаимодействует с приложением через авторизацию, и вводе данных, которые отправляются на сервер. Сервер в свою очередь обрабатывает полученные данные и отправляет результаты и уведомления клиенту. Диаграмма иллюстрирует полной взаимодействие клиента с мобильным приложением и ее серверной частью.

1.6 Постановка задачи на разработку мобильного приложения

Целью разработки мобильного приложения является помощь клиентам, которые смогут позволить себе вводить и отправлять данные счетчиков на сервер, где эти данные будут храниться. Также рассчитываться текущая плата на основе введенных данных, а также предположительная плата за следующий месяц, что позволит клиентам экономить на потреблении энергоресурсов и бережнее относиться к своим расходам.

Задачи, которые предстоит выполнить при разработке мобильного приложения:

- проектирование удобного интерфейса для пользователя, которое позволит легко вводить данные и просматривать результаты расчетов,
- разработка серверной части, которая будет обрабатывать, хранить и анализировать данные пользователя,
- интеграция взаимодействия между сервером и приложением для стабильного взаимодействия между пользователем и приложением,
- функциональное тестирование приложения,
- обеспечение безопасности и конфиденциальности данных клиента.

Эти задачи помогут полностью удовлетворить потребности клиента и создать приложение, которое будет являться удобным и надежным инструментом для пользователей. Успешное выполнение этих задач обеспечит повышение эффективности управления коммунальными услугами и удовлетворенность пользователей.

Выводы по главе 1.

Таким образом, в результате первой главы был проведен большой анализ предметной области, в котором была описана характеристика предприятия и ее организационная структура, которая помогла понять, как работает организация. Также проведен анализ бизнес-процесса “Управление

платежами”, в котором были выявлены слабые стороны процесса, что в дальнейшем поможет создать более качественную систему и устранить выявленные недостатки.

Важной частью работы является проведение анализа уже готовых решений, в которых также были выявлены недостатки с помощью сравнительного анализа по методу FURPS, что позволило понять какие аспекты требуется добавить, либо улучшить существующие. В итоге были разработаны задачи и требования к будущему приложению, которые позволят обеспечить клиента удобным функционалом. В дополнение, анализ конкурентов и существующих решений дал возможность увидеть лучшие практики и внедрить их в разрабатываемое приложение, что обеспечит его конкурентоспособность на рынке.

Глава 2 Проектирование мобильного приложения

2.1 Описание математическая модели

Начинается вторая глава с описания математической модели, в которой будет рассмотрены формулы, по которым рассчитывается плата за услуги, а также прогнозируемая плата за энергоресурсы. «Математическая модель, в частности, предназначена для прогнозирования поведения реального объекта, но всегда представляет собой ту или иную степень его инициализации» [12].

Расчет оплаты потребляемых энергоресурсов:

Объем потребляемой горячей воды Q:

$$Q = P\text{тг} - P\text{пг}, \quad (1)$$

где Pтг – показание счетчика горячей воды на текущий момент.

Pпг – предыдущее показание горячей воды.

Объем потребляемой холодной воды V:

$$V = P\text{тх} - P\text{пх}. \quad (2)$$

где Pтх – показание счетчика холодной воды на текущий момент.

Pпх – предыдущее показание холодной воды.

Объем потребляемого электричества E:

$$E = P\text{тэ} - P\text{пэ}, \quad (3)$$

где Pтэ – показание счетчика горячей воды на текущий момент.

Pпэ – предыдущее показание горячей воды.

Расчет оплаты горячей вода по условиям тарифа Pг:

$$Рг = Q * Тг, \quad (4)$$

где Q - объём потребляемого горячего водоснабжения.

$Тг$ – тариф на горячую воду на текущий момент.

Расчет оплаты холодной воды по условиям тарифа $Рх$:

$$Рх = V * Тх, \quad (5)$$

где V - объём потребляемого холодного водоснабжения.

$Тх$ – тариф на холодную воду.

Расчет оплаты электроэнергии по условиям тарифа $Рэ$:

$$Рэ = E * Тэ, \quad (6)$$

где E - объём потребляемой электроэнергии.

$Тэ$ – тариф на электроэнергию.

Сумма оплаты для платежа A:

$$A = Рг + Рх + Рэ + Ро + Рд, \quad (7)$$

где $Рг$ - объём потребляемой электроэнергии.

$Рх$ – тариф на электроэнергию.

$Рэ$ - объём потребляемой электроэнергии.

$Ро$ – ежемесячный платеж за обслуживание и содержание общедомовых помещений.

$Рд$ – дополнительные расходы на ремонт общедомовых помещений.

Эта математическая модель показывает, как происходит расчет оплаты энергоресурсов. Так, из нынешних показаний счетчиков вычитаются

предыдущие и умножаются на тарифы для конкретного энергоресурса. Общая сумма рассчитывается путем суммирования полученных показаний с дополнительными услугами, такими как ежемесячный платеж за обслуживание и содержание общедомовых помещений и дополнительных расходов на ремонт.

Теперь рассмотрим расчет прогнозируемой оплаты следующего месяца, который будет собирать данные за несколько предыдущих месяцев, анализировать тенденции и прогнозировать потребление.

Анализ тенденций:

$$\Delta Q_t = Q_t - Q_{t-1}, \quad (8)$$

$$\Delta E_t = E_t - E_{t-1}, \quad (9)$$

$$\Delta V_t = V_t - V_{t-1}, \quad (10)$$

где ΔQ_t, ΔE_t, ΔV_t – изменения данных потребления за предыдущие месяцы.

Прогнозирование потребления:

$$Q_{t+1} = Q_t + \Delta Q_t, \quad (11)$$

$$E_{t+1} = E_t + \Delta E_t, \quad (12)$$

$$V_{t+1} = V_t + \Delta V_t, \quad (13)$$

где Q_{t+1}, E_{t+1}, V_{t+1} – прогнозируемые данные потребления на следующий месяц.

Расчет прогнозируемой оплаты:

$$P_Q = (Q_{t+1} - Q_t) \times T_{\text{г}}, \quad (14)$$

$$P_E = (E_{t+1} - E_t) \times T_{\text{Э}}, \quad (15)$$

$$P_V = (V_{t+1} - V_t) \times T_X, \quad (16)$$

где P_Q, P_E, P_V – Прогнозируемая оплата энергоресурсов, рассчитанная с условием тарифов.

Полная сумма прогнозируемого расчета:

$$P_t = P_Q + P_E + P_V + P_O + P_Д, \quad (17)$$

где P_t – Сумма прогнозируемого расчета с учетом дополнительных услуг.

Таким образом, данные математические модели помогут при дальнейшей реализации серверной части мобильного приложения.

2.2 Обоснование выбора средств разработки

В этом подразделе рассмотрим средства разработки мобильного приложения и обоснуем выбор конкретной из них. «Сегодня средства разработки информационных систем представлены в широком разнообразии. Их выбор отражает мнение команды разработчиков в рамках конкретного проекта, а поскольку и информационные системы разнообразны, и задачи у них разняться очень широко, ставка делается на оптимальное решение» [14].

Разработка программы будет выполнена в среде разработки Android Studio. Эта среда от компании Google, специально созданная для разработки приложений для операционной системы Android. «В ней присутствуют макеты для создания UI, с чего обычно начинается работа над приложением. В Android Studio содержатся инструменты для разработки решений для смартфонов и планшетов, а также новые технологические решения для Android TV, Android Wear, Android Auto, Glass и дополнительные контекстуальные модули» [13].

Убедимся в том, что Android Studio имеет ряд преимуществ проанализировав ее с другими средами разработки. Одними из самых

популярных являются IntelliJ IDEA и VisualStudio, с установленными расширениями. Далее рассмотрим каждую из них отдельно.

IntelliJ IDEA – это интегрированная среда разработки, разработанная российской компанией JetBrains. Среда имеет редактор кода с подсветкой, а также систему подсказок с умным автозаполнением кода. Среда является кросс-платформерной, что означает «вы можете пользоваться ей на разных операционных системах: Windows, macOS и системах семейства Linux» [14]. Интерфейс IntelliJ IDEA представлен на рисунке 7.

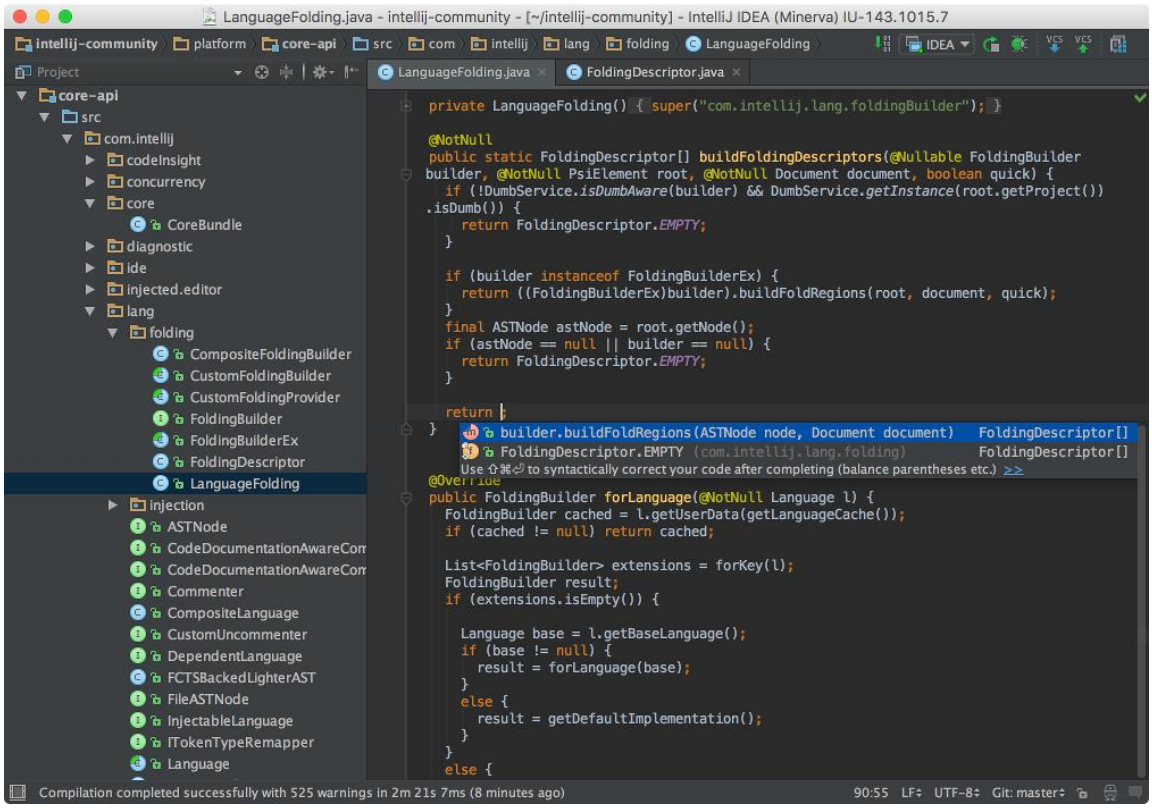


Рисунок 7 – Интерфейс IntelliJ IDEA

Visual Studio Code – это редактор кода от компании Microsoft. Несмотря на то, что это обычный редактор, он имеет большие возможности в виде плагинов и расширений, которые можно применить для разработки мобильного приложения. «В теории пользоваться VS Code может разработчик практически на любом из современных языков. Но на практике его применяют

там, где не нужны мощности полноценной IDE» [15]. Интерфейс Visual Studio Code представлен на рисунке 8.

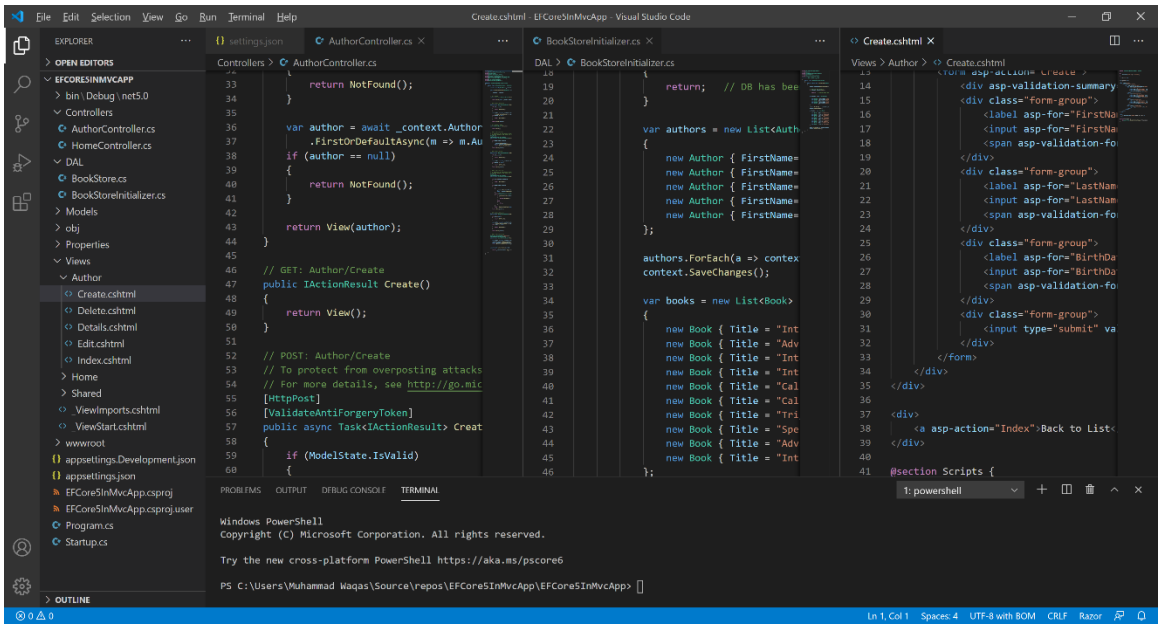


Рисунок 8 – Интерфейс Visual Studio Code

Применим метод FURPS, который был использован в подразделе 1.4 для оценки и анализа качества средств разработки, добавив всю информацию в таблицу 2.

Таблица 2 – Анализ средств разработки

Критерий	Android Studio	IntelliJ IDEA	Visual Studio
Удобство использования	Интуитивный интерфейс, обилие шаблонов и мастеров, отличная документация	Удобный и настраиваемый интерфейс, мощные рефакторинговые возможности	Сложный интерфейс, высокая настраиваемость, интеграция с Windows

Продолжение таблицы 2

Критерий	Android Studio	IntelliJ IDEA	Visual Studio
Надежность и безопасность	Высокая стабильность, регулярные обновления и исправления, поддержка Google	Высокая надежность, регулярные обновления от JetBrains	Высокая стабильность, надежная поддержка от Microsoft
Производительность	Высокая производительность, особенно на мощных машинах, оптимизированная сборка и отладка	Высокая производительность, адаптивность к различным проектам	Высокая производительность, хорошая поддержка многозадачности
Поддерживаемость	Отличная документация, большое сообщество, интеграция с Google Services, поддержка через форумы и ресурсы Google	Широкая документация, активное сообщество, поддержка JetBrains	Обширная документация, поддержка от Microsoft, интеграция с Azure
Функциональность	Полная поддержка Android, встроенный эмулятор, интеграция с Google Services, инструменты для тестирования	Поддержка множества языков, расширяемость через плагины, поддержка Android	Поддержка множества языков, интеграция с Azure, мощные инструменты отладки

Анализ показывает, что Android Studio является лучшим выбором, так как имеет следующие возможности:

- понятный интерфейс (рисунок 9);
- мощный редактор кода, который обладает всеми нужными функциями для повышения производительности при разработке (автозаполнение, подсветка синтаксиса, быстрая и понятная навигация);
- поддержка языка программирования Java,
- богатые возможности отладки;

– интеграция с другими инструментами Google;

– инструменты для тестирования;

– обширная документация.

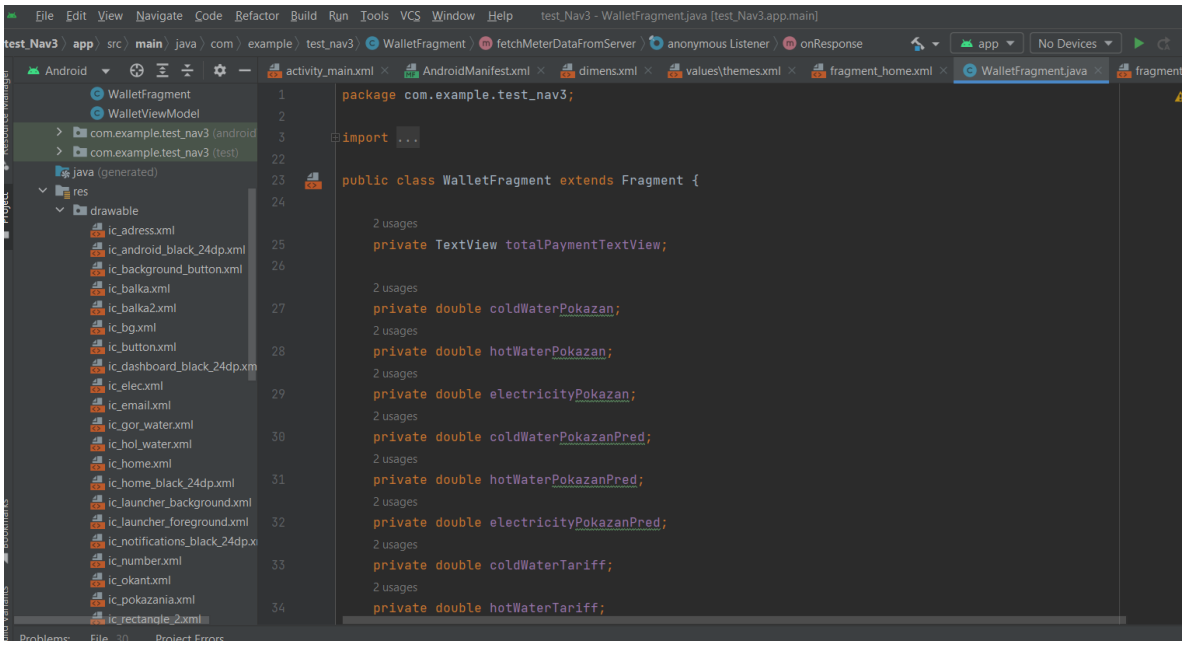


Рисунок 9 – Интерфейс Android Studio

В качестве хранения, отправки и получении данных будет использован созданный, локальный JSON Server. Это удобный инструмент, который позволяет быстро создавать REST API на основе JSON-файлов. «JSON Server — это библиотека, позволяющая "получить полный фейковый REST API без предварительной настройки менее чем за 30 секунд". Также имеется возможность создания полноценного сервера. Данная библиотека реализована с помощью lowdb и express. Наиболее известным примером ее использования является JSON Paseholder» [24]. Он идеально подходит для создания фиктивного сервера для тестирования работы приложения. В итоге, после того как приложение будет готово к развертыванию, можно будет легко заменить JSON Server на реальный сервер.

Для создания и запуска сервера, прибегнем к еще одной среде разработки под названием Visual Studio Code, в который встроено множество инструментов и расширений для работы с различными языками программированиями и инструментами для разработки. С помощью нужных расширений у меня есть возможность создать файл «server.js», который будет содержать код для запуска JSON Server, а также хранить скрипты для проведения нужных расчетов.

Таким образом, Android Studio является оптимальным выбором для разработки приложения на базе Android, а Visual Studio Code для разработки серверной части.

2.3 Логическая структура программного продукта

В данном подразделе будет разработана полная логичная структура мобильного приложения, которое включает в себя:

- логическую структуру приложения;
- логическую структуру взаимодействия между клиентом и приложением;
- логическую модель данных.

Далее рассмотрим каждую из них по отдельности.

Диаграмма логической структуры приложения, представленная на рисунке 10 — это подробный набор модулей с соответствующими функциями и связями между ними.

Структура разделена на кластеры, каждый из которых выполняет свою функцию. В кластере интерфейса приложения находятся сервисы, которые предоставляются клиенту в процессе взаимодействия.

В кластере возможностей представлена бизнес-логика приложения, в которую входят:

- учет потребления энергоресурсов,

– расчет платежа,

– расчет прогнозируемого платежа.

Также в этом кластере имеются функции доступа к серверу, с помощью которого происходит обмен данными между приложением и сервером. Также добавлен администратор, в обязанности которого входит:

– мониторинг работоспособности системы,

– безопасность данных.

Важно подметить, что в кластере интерфейса находится мобильный клиент, с помощью которого пользователь взаимодействует со всеми функциями приложения.

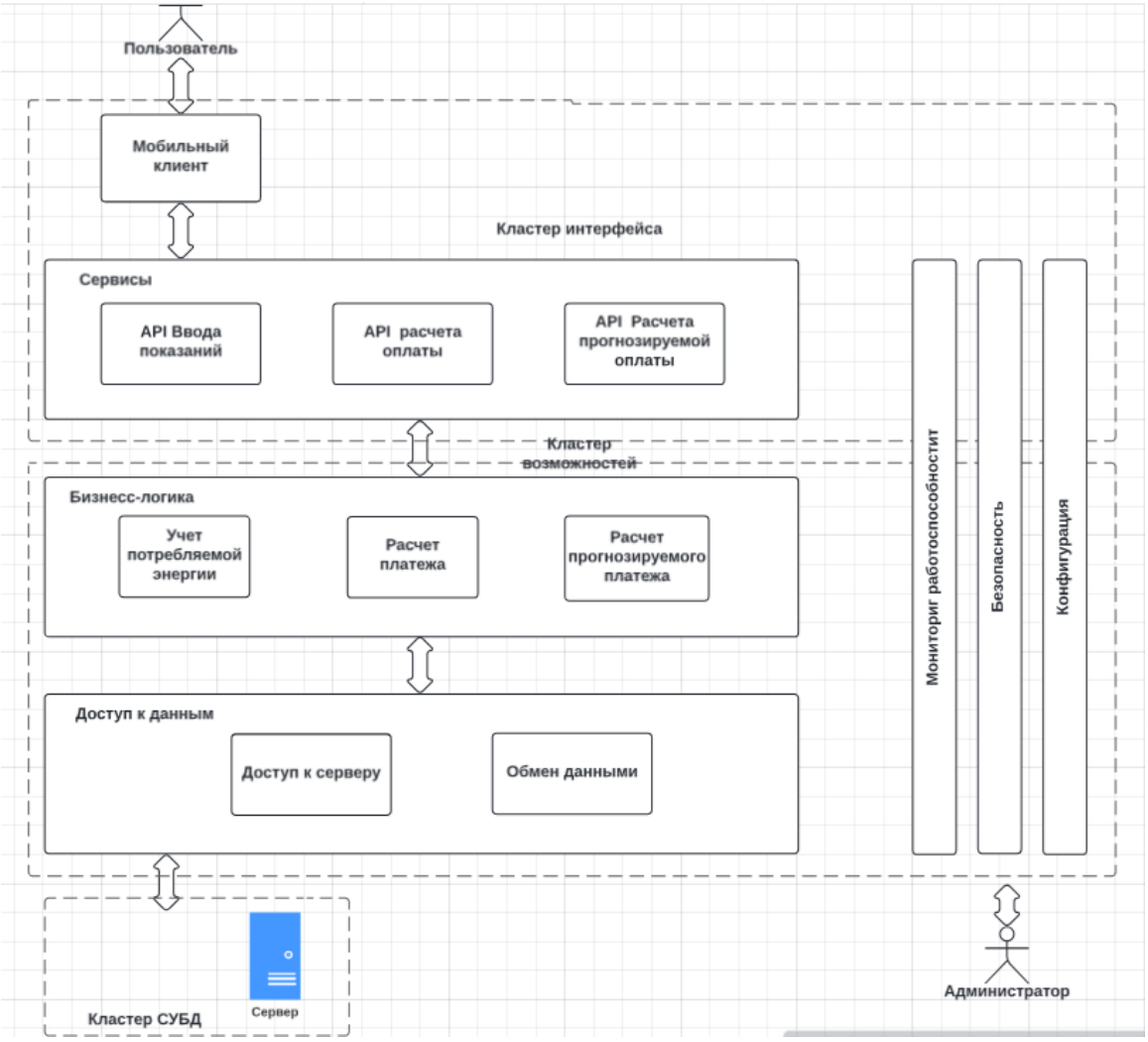


Рисунок 10 – Логическая структура

Далее подробно рассмотрим логическую структуру взаимодействия между приложением и сервером, которая представлена на рисунке 11.

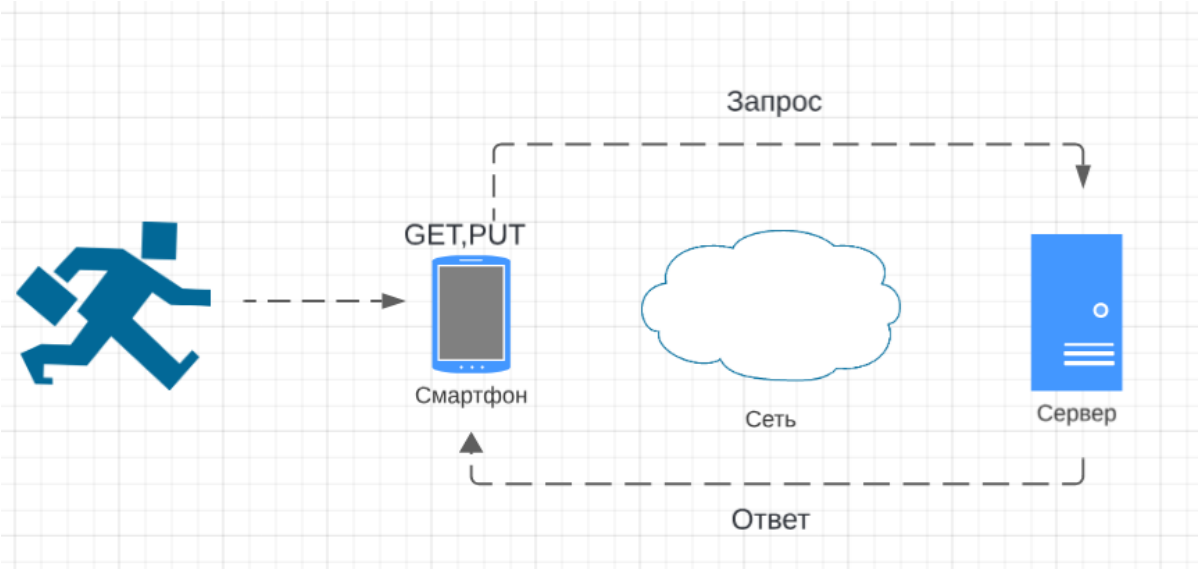


Рисунок 11 – Логическая структура между приложением и сервером.

Диаграмма показывает, что обмен данными между сервером и приложением происходит с помощью HTTP-запросов, каждый из которых выполняет свою функцию. Так, GET-запрос используется для получения данных с сервера «Также GET позволяет передавать первым параметром не адрес, а набор опций, из которого будет составлен адрес» [23]. POST-запрос отпраляет запрос на сервер для создания новых записей. «Так, используя метод fetch() и указав необходимые параметры мы можем отправить или получить данные по сети» [23]. PUT-запрос обновляет существующие данные на сервере. «Существует метод PUT для работы с сетевыми запросами и относящийся к стандартам HTTP протокола передачи данных. В javascript данный метод указывается при отправке запроса на сервер для создания нового ресурса» [23]. DELETE – запрос используется для удаления данных на сервере. PATCH-запрос отвечает за частичное обновление данных на сервере.

«Запрос PATCH является набором инструкций о том, как изменить ресурс. В отличие от PUT, который полностью заменяет ресурс» [23].Эти типы запросов обеспечивают полное взаимодействие приложения и сервера, позволяя передавать, получать, а также обновлять и удалять их по мере необходимости. Выбор конкретного типа запроса зависит от задачи, которую необходимо выполнить, и структуры API сервера. Для разработки приложения будут использоваться GET, POST – запросы.

Рассмотрим метод отправки и получения данных на сервер с помощью блок схемы, которая представлена на рисунке 12.

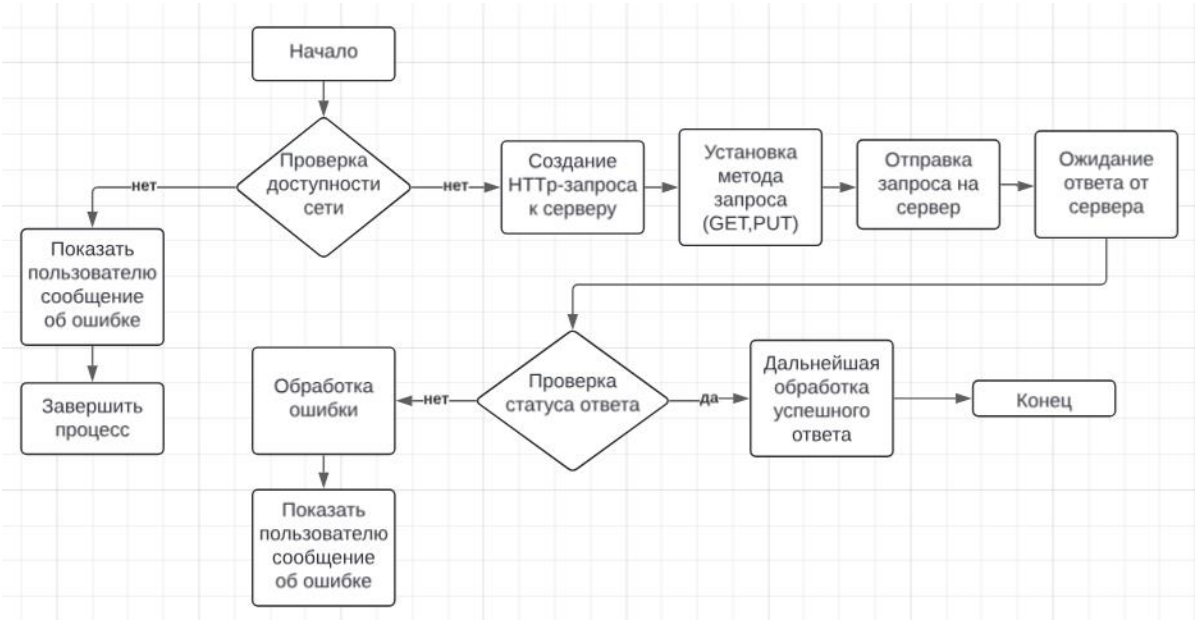


Рисунок 12 – Метод отправки и получения данных на сервер

Приложение проверяет доступность сети, если сеть недоступна, то клиенту будет показываться сообщение о плохом соединении. Если телефон имеет доступ к интернету, то создается HTTP-запрос к серверу, в котором устанавливается нужный метод (GET/PUT). Далее запрос отправляется на сервер и ожидает от него ответа. Если ответ не получен, пользователю также будет показано сообщение об ошибке. В противном случае дальнейший ответ обрабатывается.

Теперь можно перейти к разработке логической модели базы данных, которая будет включать в себя классы, которые создают между собой структуру классов. «Логическая модель данных — это расширение концептуальной модели данных. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи, которые представляют бизнес-информацию и определяют бизнес-правила» [18]. Эта структура представлена на рисунке 13.

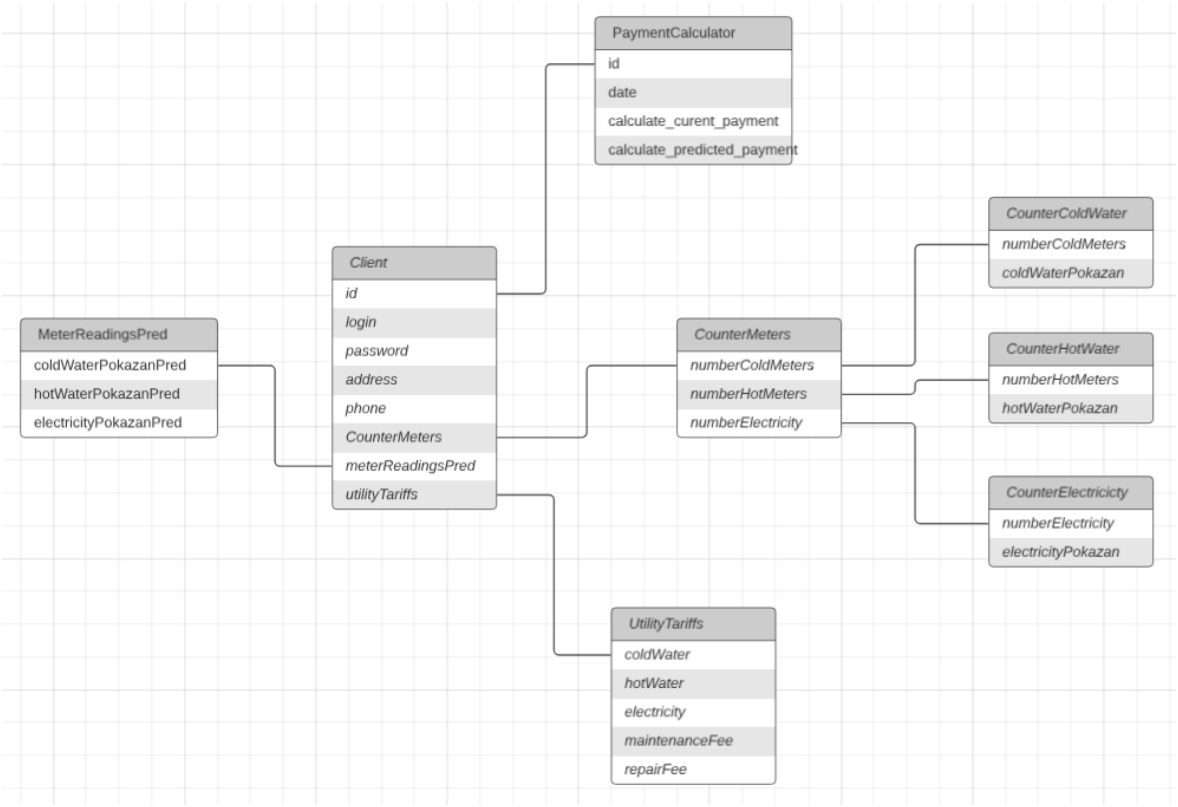


Рисунок 13 – Логическая модель базы данных

Основной класс Client несет в себе информацию об уникальном идентификаторе id, логине, пароле, адресе проживания и показаниях счетчиков клиента. Данные о тарифах и предыдущих показаниях хранятся в соответствующих классах, «MeterReadingsPred» и «UtilityTariffs».

Данные о текущих показаниях хранятся в классах «CounterColdWater», «CounterHotWater», «CounterElectricity». Также помимо самих показаний в

классах хранятся уникальные номера счетчиков, для корректной передачи показаний.

«PaymentCalculator» является местом хранения рассчитанных платежей. Он связывается с уникальным идентификатором «id» клиента, что позволяет хранить информацию о платеже исключительно для конкретного клиента.

Выводы по главе 2.

В данной главе были рассмотрены и проанализированы:

- математическая модель расчета оплаты на основе введенных показаний счетчика, а также прогнозируемой оплаты следующего месяца;
- три средства разработки, анализ которых помог выбрать оптимальное;
- полностью рассмотрена логическая структура приложения, которая включала в себя структуру взаимодействия между клиентом и приложением, структуру архитектуры и создание базы данных.

Опираясь на полученные результаты, можно смело перейти к реализации мобильного приложения, в котором будут описаны основные методы создания интуитивно понятного интерфейса, а также разработка функций для реализации авторизации пользователя, отправки данных на сервер и расчета нужных оплат.

Глава 3 Реализация мобильного приложения

3.1 Описание разработанного программного решения

В процессе работы было разработано мобильное приложение для управления процессом обслуживания клиентов в сфере жилищно-коммунальных услуг.

Созданное приложение позволяет клиентам отправлять показания счетчиков на сервер, где рассчитывается оплата потребляемых энергоресурсов, а также прогнозируемая оплата следующего месяца, что позволит некоторым клиентам сэкономить их средства.

В качестве языка программирования был использован язык java. Для взаимодействия между сервером и приложением была использована библиотека Retrofit и OkHttp, а для формирования данных для отправки библиотека Gson, так как данные отправляются на сервер в этом формате. «Retrofit (согласно официальному сайту) — типобезопасный HTTP-клиент для Android и Java. Он является незаменимым инструментом для работы с API в клиент-серверных приложениях» [15].

«Retrofit — это библиотека, которая упрощает работу с сетевыми запросами в приложениях на Android. Например, с этим инструментом можно настроить работу приложения с погодой. Разработчик задаёт параметры запросов, которые должны поступать на сервер и забирать данные по температуре в разных городах. Всё остальное происходит автоматически. В итоге пользователь видит актуальные данные каждый раз, когда открывает приложение» [22].

Также приложение позволяет видеть уведомления, которые показываются пользователю при отправке данных на сервер или при успешной или не успешной авторизации.

3.2 Реализация авторизации пользователя

При входе в приложении пользователя встречает первый экран авторизации интерфейс которого представлен на рисунке 14, на котором ему придется пройти авторизацию. Чтобы ее пройти ему нужно ввести данные в соответствующие поля.

Рисунок 14 – Интерфейс экрана авторизации.

Процесс авторизации осуществляется путем отправки и получения запросов на сервер. Для этого в среде разработки Android Studio создается класс «AuthRequest», представленный на рисунке 15, который хранит в себе логин и пароль пользователя. В листинге 1, в конструкторе инициализируются поля «login», «password», а геттеры и сеттеры обеспечивают доступ к полям с возможностью их изменения.

```
public class AuthRequest {
    3 usages
    private String login;
    3 usages
    private String password;

    1 usage
    public AuthRequest(String login, String password) {
        this.login = login;
        this.password = password;
    }
```

Рисунок 15 – Класс «AuthRequest»

Также создается класс «AuthResponse», представленный на рисунке 16, который предоставляет ответ от сервера на запрос авторизации. Класс имеет поле «success», которое указывает на успешность авторизации. Поле “message”, содержит сообщение об ошибке при попытке авторизации. И самым важным является поле «token», который хранит в себе токен, необходимый для продолжения авторизации.

```
public class AuthResponse {
    2 usages
    private boolean success;
    2 usages
    private String message;
    2 usages
    private String token;

    // Getters and setters
```

Рисунок 16 - Класс «AuthResponse»

Для определения метода выполнения HTTP - запроса настраивается интерфейс «ApiServise», представленный на рисунке 17. С помощью аннотации POST(«login») указывает, что метод выполняет POST запрос на конечную точку «login».

```
@POST("login")
Call<AuthResponse> loginUser(@Body AuthRequest authRequest);
```

Рисунок 17 – Интерфейс «ApiServise»

В классе «RetrofitClient» создается и настраивается экземпляр Retrofit. Метод «getClient» добавляется конвертер Gson для обработки данных в JSON формате. Метод представлен на рисунке 18.

```
public static Retrofit getClient(String baseUrl) {
    if (retrofit == null) {
        retrofit = new Retrofit.Builder()
                .baseUrl(baseUrl)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
    }
```

Рисунок 18 – Метод «getClient» в классе «RetrofitClient»

В MainActivity, прописан основной код авторизации, который представлен на рисунках 19, 20, 21. Сначала инициализируются поля «editTextLogin», «editTextPassword», «buttonLogin». Они в свлю очередь являются id элементами из “activity_login.xml”. Затем устанавливает метод

«loginUser» как слушатель нажатий для кнопки «buttonLogin». Метод «loginUser» получает введенные пользователем данные, и, если поля пусты, выводит сообщение об ошибке. Затем создается экземпляр «ApiService» с помощью «RetrofitClient». В итоге с помощью «loginUser» выполняется сетевой запрос и в «onResponse» обрабатывает ответ. Также, если авторизация не успешна, то с помощью метода «onFailure» отобразится сообщение об успешной авторизации.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    editTextLogin = findViewById(R.id.editTextLogin);
    editTextPassword = findViewById(R.id.editTextPassword);
    buttonLogin = findViewById(R.id.buttonLogin);

    buttonLogin.setOnClickListener(new View.OnClickListener() {
        @Override
```

Рисунок 19 – Инициализация полей

```
ApiService apiService = RetrofitClient.getClient( baseUrl: "http://192.168.230.62:3000").create(ApiService.class);
AuthRequest authRequest = new AuthRequest(login, password);
```

Рисунок 20 – Создание экземпляра «ApiService»

```
public void onResponse(Call<AuthResponse> call, Response<AuthResponse> response) {
    if (response.isSuccessful()) {
        AuthResponse authResponse = response.body();
        if (authResponse != null && authResponse.isSuccess()) {
            String token = authResponse.getToken();
            saveToken(token);  // Ensure saveToken is defined within this class
            Toast.makeText( context: LoginActivity.this,  text: "Login Successful", Toast.LENGTH_SHORT).show();
            // Переход к следующей активности
        } else {
            Toast.makeText( context: LoginActivity.this,  text: "Login Failed: " + (authResponse != null ? authResponse.getMe
        }
    } else {
        Toast.makeText( context: LoginActivity.this,  text: "Login Failed", Toast.LENGTH_SHORT).show();
    }
}

@Override
public void onFailure(Call<AuthResponse> call, Throwable t) {
    Toast.makeText( context: LoginActivity.this,  text: "Network Error: " + t.getMessage(), Toast.LENGTH_SHORT).show();
}
```

Рисунок 21 – Выполнение сетевого запрос

Также реализован метод «saveToken», который сохраняет токен пользователя в «SharedPrefences» для того, чтобы клиент всегда оставался авторизованным.

3.3 Реализация функционала отправки данных на сервер

После авторизации пользователь попадает в навигационное меню, которое состоит из четырех фрагментов. Первый фрагмент носит название “Показания”, представленный на рисунке 22, в котором проходит процесс отправки вводимых показаний счетчиков на сервер, а также вывод показаний за прошлый месяц на экран.

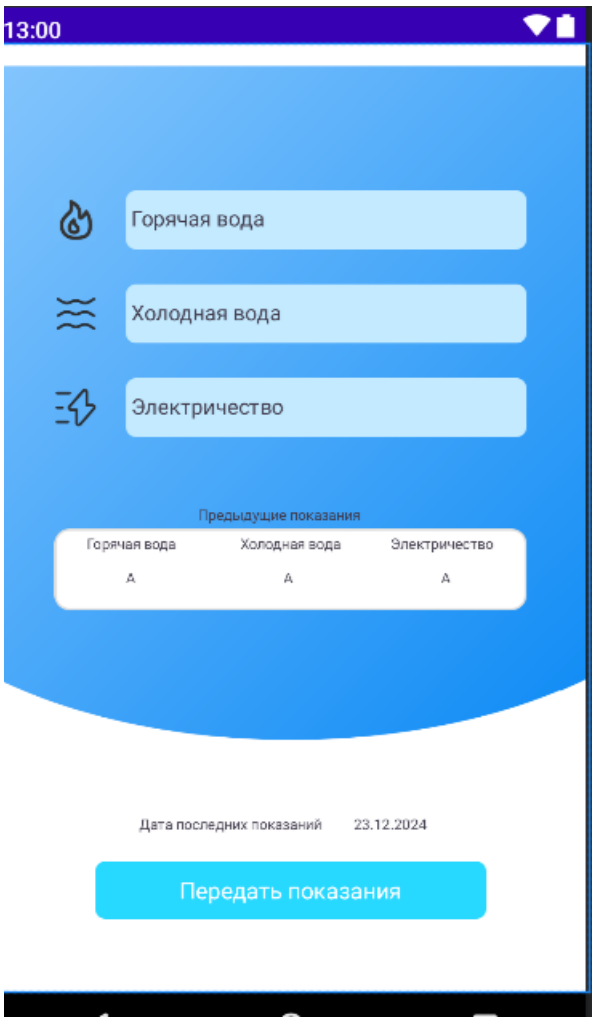


Рисунок 22 – Экран «Показания»

Для реализации этого процесса сначала создадим модель данных для отправки показаний счетчиков «MeterReadingsRequest» представленная на рисунке 23, которая содержит поля показания счетчиков (холодная вода, горячая вода, электричество), а также поле для даты.

```
1 usage
public MeterReadingsRequest(String date, int coldWaterReading, int hotWaterReading, int electricityReading) {
    this.date = date;
    this.coldWaterReading = coldWaterReading;
    this.hotWaterReading = hotWaterReading;
    this.electricityReading = electricityReading;
}
```

Рисунок 23 – Модель данных «MeterReadingsRequest»

Далее создадим интерфейс для API. Метод «sendMeterReadings» аннотирован POST для отправки данных на сервер. Метод представлен на рисунке 24.

```
@POST("/meterReadings")
Call<Void> sendMeterReadings(@Body MeterReadingsRequest meterReadingsRequest);
```

Рисунок 24 – Интерфейс для API

Настраиваем клиент Retrofit так, чтобы он использовал токен, который был использован при аутентификации. Это делается для того, чтобы токен добавлялся в заголовки запросов, что позволит отправлять данные конкретному пользователю из базы данных сервера. Настройка клиента представлена на рисунке 25.

```
public class RetrofitClient {
    3 usages
    private static Retrofit retrofit = null;

    1 usage
    public static Retrofit getClient(String baseUrl, String authToken) {
        OkHttpClient client = new OkHttpClient.Builder()
                .addInterceptor(new AuthInterceptor(authToken))
                .build();

        if (retrofit == null) {
            retrofit = new Retrofit.Builder()
                    .baseUrl(baseUrl)
                    .client(client)
                    .addConverterFactory(GsonConverterFactory.create())
                    .build();
        }
        return retrofit;
```

Рисунок 25 – Клиент Retrofit

В DashboardActivity, добавляем метод «sendMeterReadings», представленный на рисунке 26 для сборки данных из полей ввода. Затем формируем объект «MeterReadingsRequest» и извлекаем токен из «SharedPrefences». В итоге данные в формате JSON отправляются на сервер.

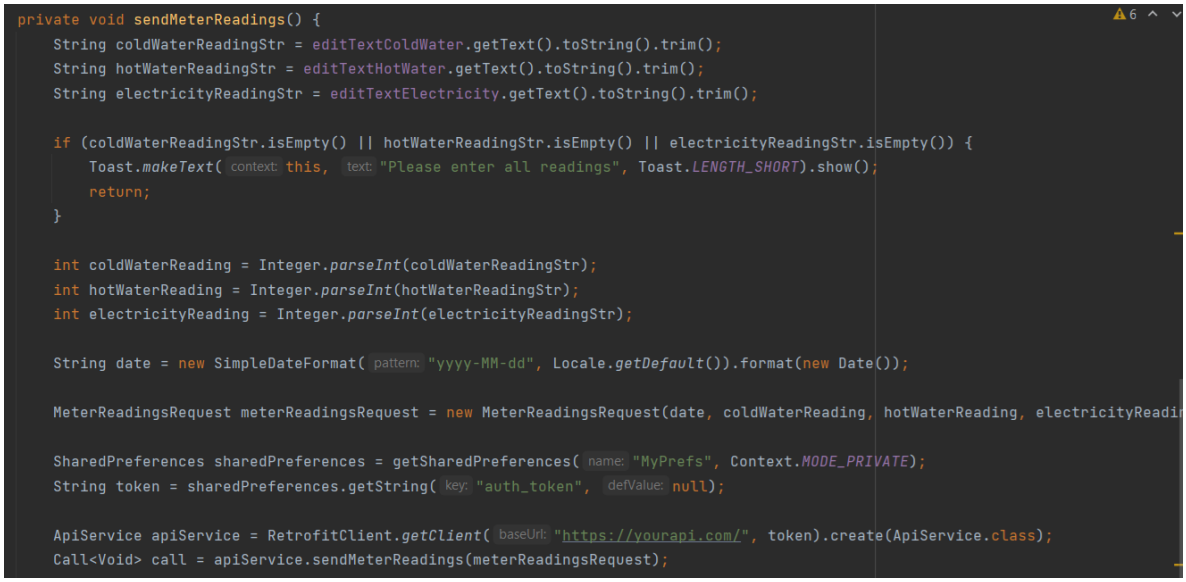


Рисунок 26 – Метод «sendMeterReadings»

В коде серверной части «Node.js» запросы на обновление данных счетчиков обрабатываются маршрутом «/updateMeterReadings. Также создается файл «Clients2», в котором хранятся данные о клиентах. Затем настраивается маршрут для обработки POST-запросов по которому обновляться данные клиента в файле Clients2.json. Запросы представлены на рисунках 27 и 28.

Для начала создадим сервер со своим портом 3000. «Компьютерный сетевой порт – это число, которое идентифицирует назначение сетевых потоков данных в пределах одного компьютера. Все хосты (компьютеры) обмениваются друг с другом информацией при помощи уникальных цифровых IP-адресов, представленных двоичной системой» [23].

```
const express = require('express');
const bodyParser = require('body-parser');
const fs = require('fs');
const path = require('path');

const app = express();
const port = 3000;
const dataFilePath = path.join(__dirname, 'Clients2.json');

app.use(bodyParser.json());
```

Рисунок 27 – Фрагмент кода настройки сервера

```
app.post('/updateMeterReadings', (req, res) => {
    const { id, meter_readings } = req.body;

    if (!id || !meter_readings) {
        return res.status(400).send('Missing required data');
    }

    fs.readFile(dataFilePath, 'utf8', (err, data) => {
        if (err) {
            return res.status(500).send('Error reading data file');
        }

        let clients = JSON.parse(data);
        const clientIndex = clients.findIndex(client => client.id === id);

        if (clientIndex === -1) {
            return res.status(404).send('Client not found');
        }

        // Обновление данных счетчика
        clients[clientIndex].meter_readings = meter_readings;

        fs.writeFile(dataFilePath, JSON.stringify(clients, null, 2), (err) => {
            if (err) {
                return res.status(500).send('Error writing data file');
            }

            const payments = calculatePayments(clients[clientIndex]);
            res.json(payments);
        });
    });
});
```

Рисунок 28 – Фрагмент кода для обновления данных счетчика по маршруту «/updateMeterReadings»

Таким образом, этот код обрабатывает POST - запросы, получаемые от приложения. Сначала данные получаются из тела запроса, а затем читаются данные из файла «Clients2.json». Полученные данные парсятся в объект JavaScript, и исходя из «id» клиента обновляются в этом файле.

3.4 Реализация функционала расчета оплат

За реализацию функционала расчета оплаты потребляемых энергоресурсов отвечает фрагмент «Оплата», представленный на рисунке 29. Он содержит в себе поля для вывода расчета сумма оплаты, а также прогнозируемая плата за следующий месяц.

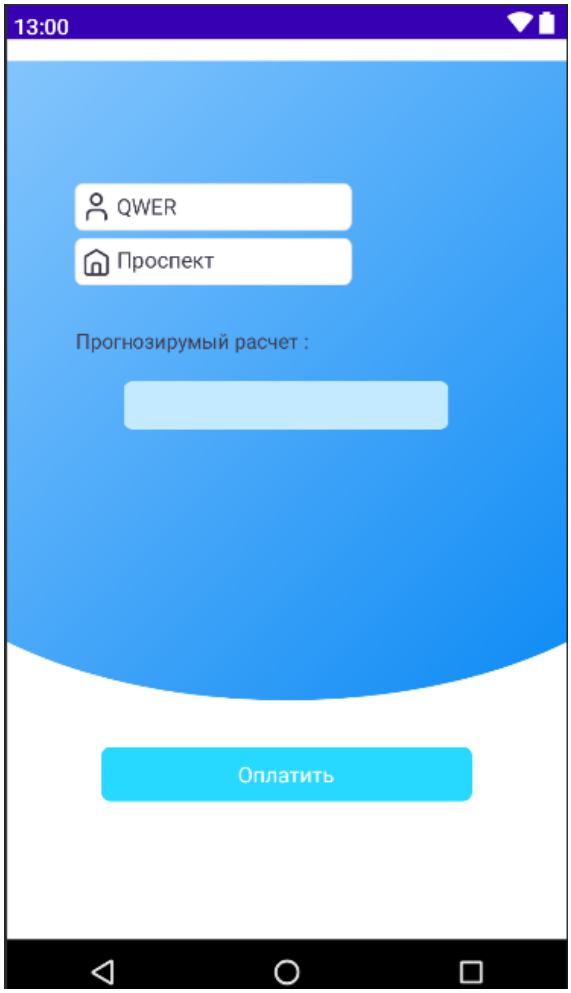
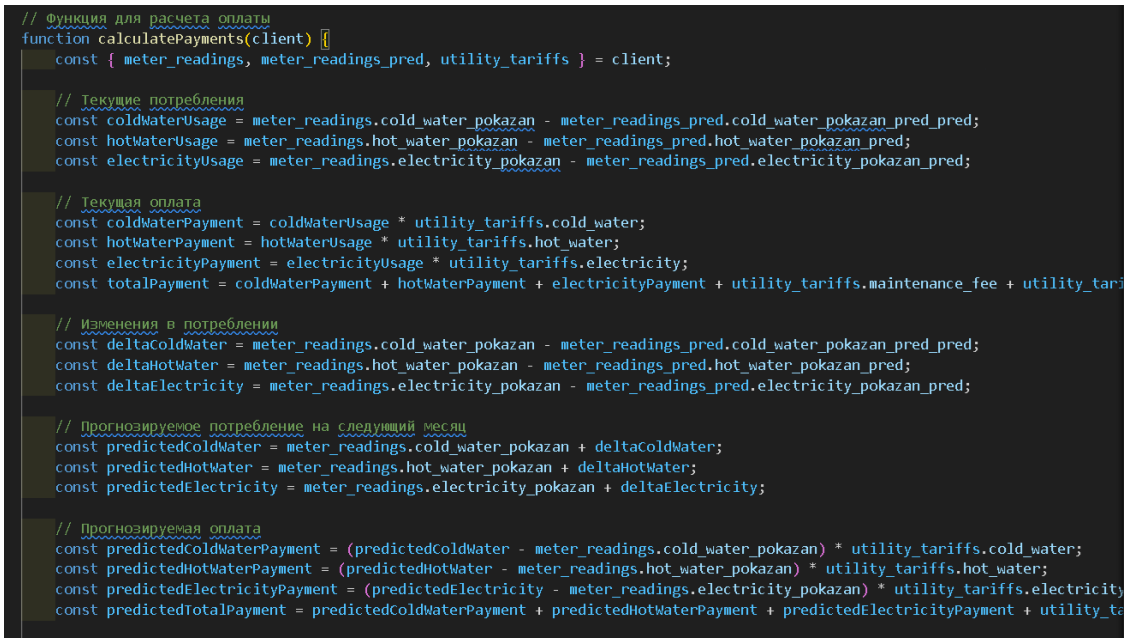


Рисунок 29 – Фрагмент «Оплата»

Для начала разработаем функции для серверной части, представленные на рисунке 30, которые проводят расчеты на основе полученных данных. Сервер будет получать обновленные данные из файла «Clients2», где хранятся данные счетчиков пользователя и на их основе проводить расчеты исходя из математической модели, разработанной в подразделе 2.1.



```
// Функция для расчета оплаты
function calculatePayments(client) {
    const { meter_readings, meter_readings_pred, utility_tariffs } = client;

    // Текущие потребления
    const coldWaterUsage = meter_readings.cold_water_pokazan - meter_readings_pred.cold_water_pokazan_pred_pred;
    const hotWaterUsage = meter_readings.hot_water_pokazan - meter_readings_pred.hot_water_pokazan_pred;
    const electricityUsage = meter_readings.electricity_pokazan - meter_readings_pred.electricity_pokazan_pred;

    // Текущая оплата
    const coldWaterPayment = coldWaterUsage * utility_tariffs.cold_water;
    const hotWaterPayment = hotWaterUsage * utility_tariffs.hot_water;
    const electricityPayment = electricityUsage * utility_tariffs.electricity;
    const totalPayment = coldWaterPayment + hotWaterPayment + electricityPayment + utility_tariffs.maintenance_fee + utility_tari

    // Изменения в потреблении
    const deltaColdWater = meter_readings.cold_water_pokazan - meter_readings_pred.cold_water_pokazan_pred_pred;
    const deltaHotWater = meter_readings.hot_water_pokazan - meter_readings_pred.hot_water_pokazan_pred;
    const deltaElectricity = meter_readings.electricity_pokazan - meter_readings_pred.electricity_pokazan_pred;

    // Прогнозируемое потребление на следующий месяц
    const predictedColdWater = meter_readings.cold_water_pokazan + deltaColdWater;
    const predictedHotWater = meter_readings.hot_water_pokazan + deltaHotWater;
    const predictedElectricity = meter_readings.electricity_pokazan + deltaElectricity;

    // Прогнозируемая оплата
    const predictedColdWaterPayment = (predictedColdWater - meter_readings.cold_water_pokazan) * utility_tariffs.cold_water;
    const predictedHotWaterPayment = (predictedHotWater - meter_readings.hot_water_pokazan) * utility_tariffs.hot_water;
    const predictedElectricityPayment = (predictedElectricity - meter_readings.electricity_pokazan) * utility_tariffs.electricity
    const predictedTotalPayment = predictedColdWaterPayment + predictedHotWaterPayment + predictedElectricityPayment + utility_ta
```

Рисунок 30 – Фрагмент кода для функций расчета оплат

Далее построим маршрут «/getPayments», представленный на рисунке 31, который будет обрабатывать GET - запросы для получения расчетной и прогнозируемой оплаты.

```
app.get('/getPayments', (req, res) => {
    const { id } = req.query;

    if (!id) {
        return res.status(400).send('Missing required data');
    }

    fs.readFile(dataFilePath, 'utf8', (err, data) => {
        if (err) {
            return res.status(500).send('Error reading data file');
        }

        let clients = JSON.parse(data);
        const client = clients.find(client => client.id === id);

        if (!client) {
            return res.status(404).send('Client not found');
        }

        const payments = calculatePayments(client);
        res.json(payments);
    });
});
```

Рисунок 31 - Фрагмент кода для расчета начислений по маршруту «/getPayments»

Маршрут получает параметр «id» и проверяет его наличие, если «id» не передан в запросе, сервер возвращает статус 400, что означает, что запрос не выполнен. Преобразует JSON – строку в объект JavaScript. Затем читает данные с файла «Clients2» и ищет клиента по «id». Вызывает функцию «calculatePayments», которая проводит расчет и отправляет платежи в формате JSON в ответ на запрос.

Теперь перейдем к клиентской части и разработаем модель данных для рассчитываемых платежей. Создадим класс «PaymentResponse», в котором будут поля «totalPayment» и «predictedPayment». Класс представлен на рисунке 32.

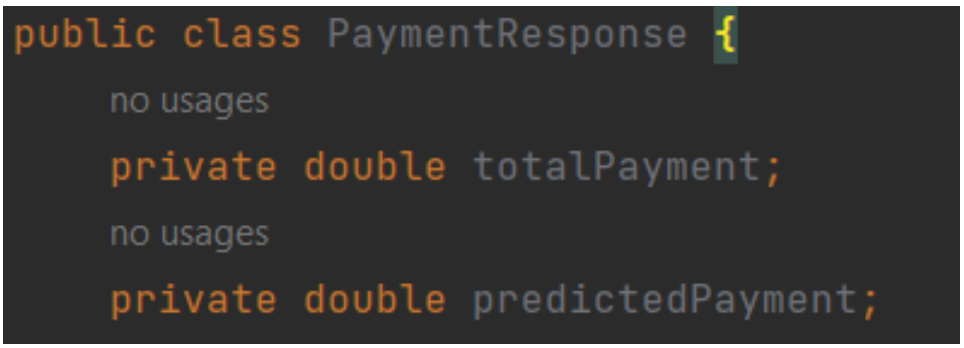


Рисунок 32 – Класс «PaymentResponse»

Далее создадим интерфейс для API. Метод «getPayments», представленный на рисунке 33, аннотирован GET для получения данных с сервера.

```
@GET("/getPayments")
Call<PaymentResponse> getPayments(@Query("id") String id);
```

Рисунок 33 – Интерфейс для API

Создадим активность, которая будет отправлять запрос на получение расчетов.

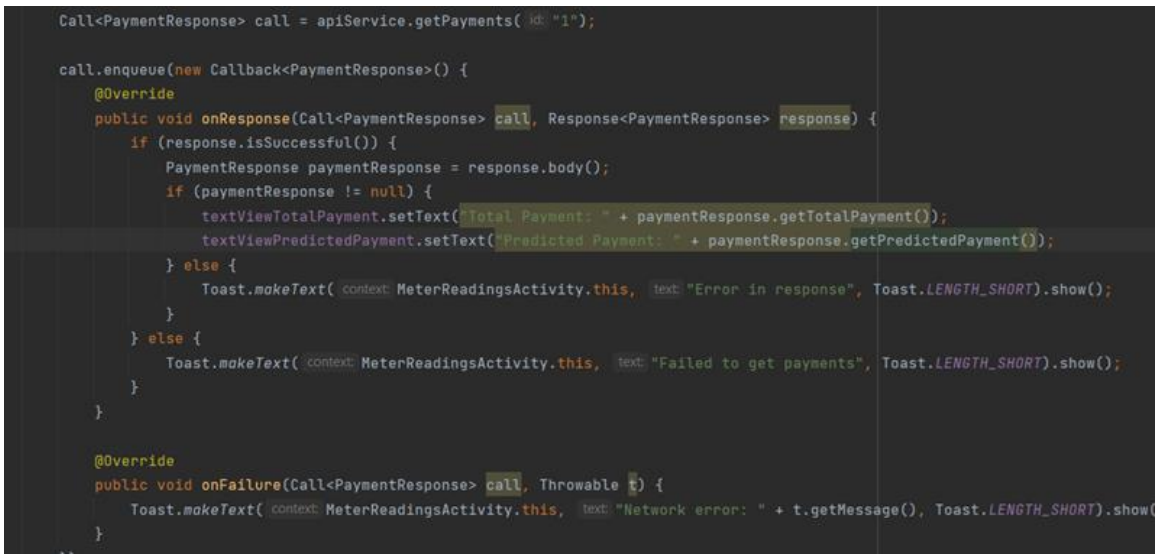


Рисунок 34 – Фрагмент кода функции «getPayment»

Таким образом, в методе «getPaymend», который представлен на рисунке 34, создается экземпляр интерфейса «ApiService», который определяет методы для выполнения запросов к серверу. Выполняется запрос на сервер для получения данных о платежах для клиента с определенным идентификатором «id». Ответ от сервера обрабатывается асинхронно. В случае успешного ответа данные из ответа извлекаются и отображаются на экране. Если сервер вернул ошибку, выводится сообщение об ошибке.

3.5 Тестирование приложения

Заключительным этапом в разработке мобильного приложения является тестирование. Тестирование помогает оценить качество созданного решения и убедится, что оно удовлетворяет всем требованиям, которые были выявлены в подразделе 1.5. «Основная задача тестирования — убедиться, что приложение выполняет свои функции: ожидаемый результат соответствует фактическому, требования соблюдаются и нет критических ошибок» [14].

Целью тестирования является корректность работы всех процессов, которые предоставляет приложение, проверка производительности и самое главное обнаружение ошибок и недочетов в работе приложения.

Объектами тестирования являются:

- вход в систему,
- получение данных с сервера,
- отправка данных на сервер,
- расчет оплаты,
- прогнозирование оплаты.

Также при тестировании приложения будет использован метод динамического тестирования (в режиме реального времени), а для проверки работоспособности приложения будет использоваться тестирование с использованием тестовых данных.

Приступим к тестированию окна авторизации. Этапы тестирования представлены на рисунках 35, 36, 37.

Рассмотрим разные варианты использования приложения:

- попытка входа без ввода данных,
- попытка входа при вводе неправильного пароля,
- успешная авторизация.

Рисунок 35 – Попытка входа без ввода данных

При попытке входа без ввода данных выводится сообщение «Введите логин и пароль».

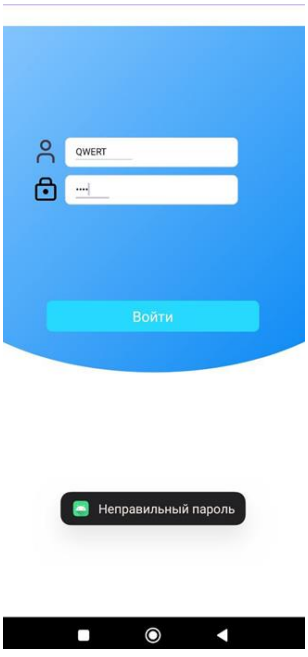


Рисунок 36 – Попытка входа при неправильном вводе пароля

При попытке авторизации с использованием неверного пароля, на экран выводится сообщение «Неправильный пароль»

Ну и наконец, авторизуемся с использованием правильных данных.

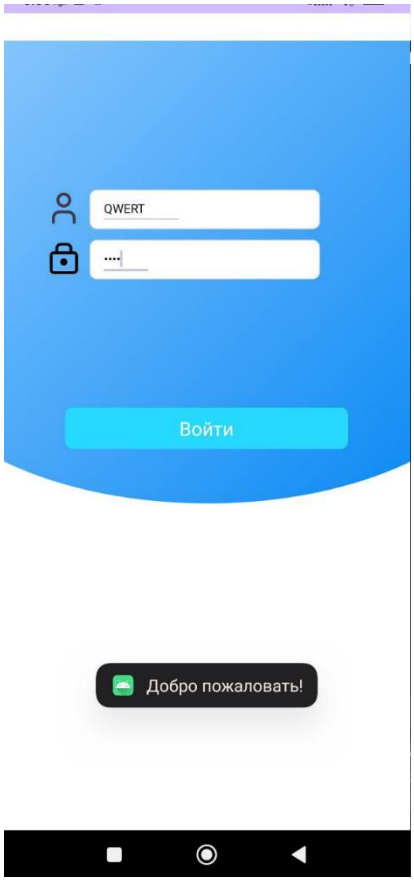


Рисунок 37 – Успешная авторизация

При правильном вводе логина и пароля на экран выводится сообщение «Добро пожаловать» и происходит переход на фрагмент «Показания».

Таким образом мы проверили несколько сценариев использования процесса авторизации. Анализ показал, что приложение на этом этапе работает корректно.

Далее перейдем к процессу отправки показаний счетчиков на сервер. Процесс отправки данных на сервер представлен на рисунке 38. Для тестовых

данных выберем фиктивные показания счетчиков, которые будут отправляться на сервер:

- показание счетчика горячей воды – 740,
- показание счетчика холодной воды – 750,
- показание счетчика электроэнергии – 760.

Рисунок 38 – Процесс отправки данных на сервер

При нажатии на кнопку «Передать показания» на экран выводится сообщение «Данные успешно отправлены на сервер», что говорит о том, что приложение справилось с поставленной задачей.

Убедимся в том, что данные были успешно отправлены, просмотрев обновленные данные сервера, представленные на рисунке 39.

```
"meter_readings": {
  "cold_water_pokazan": 750,
  "hot_water_pokazan": 740,
  "electricity_pokazan": 760
},
"meter_readings_pred": {
  "cold_water_pokazan_pred_pred": 720,
  "hot_water_pokazan_pred": 710,
  "electricity_pokazan_pred": 730
}
```


Рисунок 39 – Обновленные данные сервера

На рисунке 39 видно, что данные были успешно отправлены на сервер.

Далее проведем анализ расчета оплаты потребляемых энергоресурсов и прогнозируемой оплаты следующего месяца. Для тестовых данных будут использованы те же что и при тестировании метода отправки данных на сервер. Также помимо энергоресурсов будут использованы тарифы на потребляемые энергоресурсы и дополнительные услуги. Все тестовые данные необходимые для тестирования и результат успешной отправки данных на сервер представлены на рисунке 40 и 41.

```
"meter_readings": {
  "cold_water_pokazan": 750,
  "hot_water_pokazan": 740,
  "electricity_pokazan": 760
},
"meter_readings_pred": {
  "cold_water_pokazan_pred_pred": 720,
  "hot_water_pokazan_pred": 710,
  "electricity_pokazan_pred": 730
},
"utility_tariffs": {
  "cold_water": 41.88,
  "hot_water": 51.37,
  "electricity": 5.48,
  "maintenance_fee": 24.02,
  "repair_fee": 1.85
```


Рисунке 40 – Тестовые серверные данные для расчета оплат

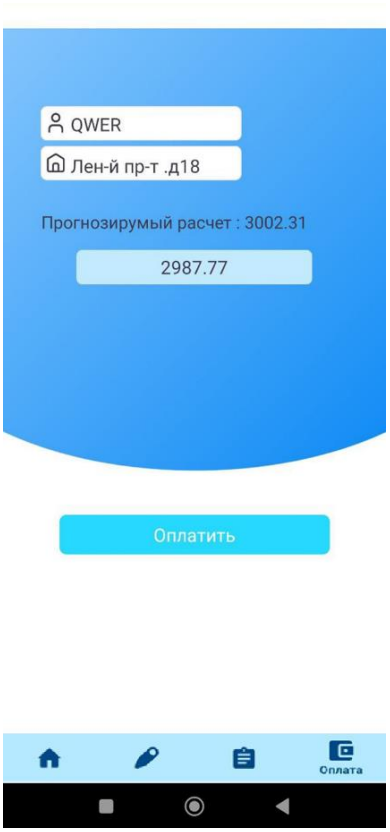


Рисунок 41 – Рассчитанные начисления выводимые на экран

На рисунке 41 видно, что рассчитанная оплата и прогнозируемая оплата следующего месяцы вывелись на экран. Чтобы удостовериться в том, что при расчетах начислений не было допущено никаких ошибок. Для этошо воспользуемся программой для работы с таблицами Excel и вычислим результат вручную используя серверные данные.

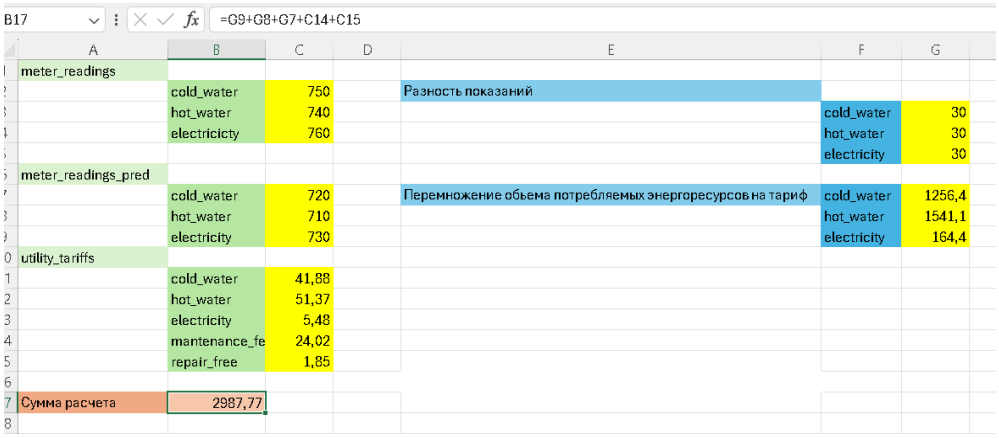


Рисунок 42 – Ручной расчет начислений

На рисунке 42 видно, что начисления рассчитались верно, согласно математическим моделям, представленным в подразделе 2.1. Это говорит о том, что процесс расчета начислений работает корректно. Полный код реализации методов в MainActivity представлен в Приложении А.

Для анализа проведенного тестирования полученные результаты занесем в таблицу 3.

Таблица 3 – Результаты тестирования приложения

Номер теста	Описание	Ожидаемый результат	Результат	Статус
1	Тестирование вариантов использования.	Вывод соответствующих сообщений, при неверном использовании функционала приложения.	В случае неверных попыток авторизации, на экран выводятся соответствующие сообщения.	Пройден
2	Тестирование отправки данных на сервер.	Тестируемые данные должны отправляться на сервер.	Данные успешно отправлены на сервер.	Пройден.
3	Тестирование расчета начислений на сервере.	Расчет начислений на основе тестируемых данных.	Расчет проводится корректно.	Пройден.

Таким образом, протестировав весь функционал приложения, можно сделать вывод, что разработанное мобильное приложение с удобным и понятным интерфейсом, а также серверная часть успешно решают задачу учета и расчета оплаты коммунальных услуг, предоставляя пользователю удобный и эффективный инструмент для управления своими расходами.

Заключение

В целом, разработка мобильного приложения для управления процессом обслуживания клиентов значительно улучшает эффективность взаимодействия с пользователями. С учетом роста удовлетворенных клиентов также растет и репутация компании, что позволяет ей, выделятся на рынке среди других.

Выполненная работа по разработке мобильного приложения для управления процессом обслуживания клиентов позволила достичь всех поставленных во введении целей и решить конкретные задачи.

Был проведен полный анализ предметной области, который позволил выявить слабые стороны бизнес-процесса и готовых решений. Также были разработаны функциональные и нефункциональные требования к разработке мобильного приложения.

Разработана математическая модель прогнозируемых начислений на следующий месяц, на основе вводимых данных, а также данных, которые хранились на сервере до этого.

Реализована серверная часть на платформе Node.js, которая принимает данные от мобильного приложения, сохраняет их в файл, производит расчеты текущей и прогнозируемой оплаты с помощью функций, и возвращает результаты обратно в приложение выводя их на экран.

Спроектирована логическая структура на основе которого было разработано мобильное приложение для платформы Android, которое предоставляет пользователю удобный интерфейс для ввода данных счетчиков, получения рассчитанных данных об оплате и прогнозируемой оплате следующего месяца.

Проведены всесторонние тестирования, чтобы убедиться, что приложение удовлетворяет всем требованиям и решает поставленные задачи. Тестирование проводилось с учетом разных вариантов использования

приложения, чтобы показать, как оно ведет себя в различных ситуациях. Так, при авторизации пользователь может видеть подсказки при неправильном вводе данных. Для проверки корректности расчета начислений и прогнозируемой оплаты, проведен ручной счет, который подтвердил корректность работы функций для расчета начислений, а также то, что данные, введенные пользователем успешно отправлены на сервер.

Весь процесс разработки проводился с учетом потребностей клиентов, что позволило создать удобный функционал, который будет приятен в использовании для любого пользователя.

Таким образом, проведенная работа имеет значительное практическое, научное и экономическое значение, а полученные результаты могут быть использованы для улучшения процессов управления коммунальными расходами и повышения их эффективности.

Список используемых источников

1.Васильев, А. Java. Объектно-ориентированное программирование: Учебное пособие Стандарт третьего поколения / А. Васильев. – СПб.: Питер, 2013. 400 с.

2.Гаст, Х. Объектно-ориентированное проектирование: концепции и программый код / Х. Гаст. - М.: Диалектика, 2018. - 1040 с.

3.Липаев В. В. Тестирование компонентов и комплексов программ: учебник. Москва: СИНТЕГ, 2010. – 393 с.

4.Математическая модель [Электронный ресурс] URL: https://ru.wikipedia.org/wiki (дата обращения 10.03.2024)

5.Назначение АСКУЭ АО [Электронный ресурс] URL: http://www.energomera.ru/ru/products/askue/appointment (дата обращения 7.04.2024)

6.О компании “Кварплата 24” [Электронный ресурс] URL: https://www.kvp24.ru/about/ (дата обращения: 10.03.2024)

7.Описание бизнес-процесов [Электронный ресурс] URL: https://trinion.org/blog/opisanie-biznes-processov-kak-est-as-is-i-kak-dolzhno-byt-to-be (дата обращения: 26.03.2024)

8.Пользовательские сценарии [Электронный ресурс] URL: https://netology.ru/blog/users-scenarios (дата обращения 7.04.2024)

9.Проектирование и разработка мобильного приложения [Электронный ресурс] URL: https://elib.sfu-kras.ru/bitstream/handle/2311/34120/diplom_v1.1.pdf?sequence (дата обращения 10.03.2024)

10.Тарасов С. В. СУБД для программиста. Базы данных изнутри. [Электронный ресурс] URL: https://cyberleninka.ru/article/n/pattern-proektirovaniya-mvvm-kak-odin-iz-sposobov-napisaniya-chistogo-koda-v-android-prilozhenii-na-jetpack-compose (дата обращения 17.03.2024)

11. Что такое Android Studio [Электронный ресурс]. URL: http://web.spt42.ru/index.php/chto-takoe-android-studio (дата обращения: 20.02.2024).
12. Что такое Android Studio [Электронный ресурс]. URL: http://web.spt42.ru/index.php/chto-takoe-android-studio (дата обращения: 20.02.2024). р.
13. Что такое цифровизация и зачем она нужна [Электронный ресурс] URL: https://dis-group.ru/blogs/czifrovizacziya-chto-eto-takoe-prostymi-slovami/ (дата обращения: 26.03.2024)
14. Энфорс Лайт - програмное обеспечение [Электронный ресурс] URL: https://nforceit.ru/napravleniya/programmnoe-obespechenie-askue/enfors-lajt/ (дата обращения: 10.03.2024)
15. APScheduler library user guide [Электронный ресурс] URL: https://apscheduler.readthedocs.io/en/stable/userguide.html (дата обращения: 26.02.2024)
16. askye.ru [Электронный ресурс]. URL: http://www.ackye.ru/ (дата обращения: 15.03.2024).
17. Java 3.10.4 documentation [Электронный ресурс] URL: https://docs.java.org/3.10 (дата обращения: 20.02.2024)
18. OneSignal SDK. [Электронный ресурс] / Режим доступа: URL: https://documentation.onesignal.com/docs/android-sdk-setup. (дата обращения: 13.04.24)
19. PySimpleGUI library documentation [Электронный ресурс] URL: https://pysimplegui.readthedocs.io/en/latest/ (дата обращения: 26.03.2024)
20. TOML documentation [Электронный ресурс] URL: https://github.com/toml-lang/toml/blob/main/README.md (дата обращения: 24.03.2024)
21. FURPS [Электронный ресурс]. URL: https://tproger.ru/translations/python-ide/ (дата обращения: 15.03.2024).

22. Java + Visual Studio Code [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/visual-studio-code/ (дата обращения 12.04.2024)
23. JetBrains IntelliJ IDEA [Электронный ресурс] URL: https://www.jetbrains.com/ru-ru/idea/features/ (дата обращения: 02.04.2024)
24. OneSignal SDK3. [Электронный ресурс] / Режим доступа: URL: https://documentation.onesignal.com/docs/androids-sdk-setup. (дата обращения: 13.04.24)
25. PySimpleGUI library documentation [Электронный ресурс] URL: https://pysimplegui.readthedocs.io/en/latest/ (дата обращения: 26.03.2024)

Приложение А

Отправка и получение данных в MainActivity

```
public class MeterReadingsActivity extends AppCompatActivity {

    private EditText editTextColdWater;
    private EditText editTextHotWater;
    private EditText editTextElectricity;
    private Button buttonSubmit;
    private TextView textViewTotalPayment;
    private TextView textViewPredictedPayment;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_meter_readings);

        editTextColdWater = findViewById(R.id.editTextColdWater);
        editTextHotWater = findViewById(R.id.editTextHotWater);
        editTextElectricity = findViewById(R.id.editTextElectricity);
        buttonSubmit = findViewById(R.id.buttonSubmit);
        textViewTotalPayment = findViewById(R.id.textViewTotalPayment);
        textViewPredictedPayment =
findViewById(R.id.textViewPredictedPayment);

        buttonSubmit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sendMeterReadings();
            }
        });

        getPayments();
    }

    private void sendMeterReadings() {
        String coldWaterReadingStr = editTextColdWater.getText().toString().trim();
        String hotWaterReadingStr = editTextHotWater.getText().toString().trim();
        String electricityReadingStr = editTextElectricity.getText().toString().trim();
```

```
        if (coldWaterReadingStr.isEmpty() || hotWaterReadingStr.isEmpty() ||
electricityReadingStr.isEmpty()) {
            Toast.makeText(this, "Please enter all readings",
Toast.LENGTH_SHORT).show();
            return;
        }

        int coldWaterReading = Integer.parseInt(coldWaterReadingStr);
        int hotWaterReading = Integer.parseInt(hotWaterReadingStr);
        int electricityReading = Integer.parseInt(electricityReadingStr);

        MeterReadings meterReadings = new MeterReadings(coldWaterReading,
hotWaterReading, electricityReading);
        MeterReadingsRequest request = new MeterReadingsRequest("1",
meterReadings);

        ApiService apiService =
RetrofitClient.getClient("http://yourserver.com:3000").create(ApiService.class);
        Call<PaymentResponse> call = apiService.updateMeterReadings(request);

        call.enqueue(new Callback<PaymentResponse>() {
            @Override
            public void onResponse(Call<PaymentResponse> call,
Response<PaymentResponse> response) {
                if (response.isSuccessful()) {
                    PaymentResponse paymentResponse = response.body();
                    if (paymentResponse != null) {
                        textViewTotalPayment.setText("Total Payment: " +
paymentResponse.getTotalPayment());
                        textViewPredictedPayment.setText("Predicted Payment: " +
paymentResponse.getPredictedPayment());
                    } else {
                        Toast.makeText(MeterReadingsActivity.this, "Error in response",
Toast.LENGTH_SHORT).show();
                    }
                } else {
                    Toast.makeText(MeterReadingsActivity.this, "Submission failed",
Toast.LENGTH_SHORT).show();
                }
```

```
}

        @Override
        public void onFailure(Call<PaymentResponse> call, Throwable t) {
            Toast.makeText(MeterReadingsActivity.this, "Network error: " +
t.getMessage(), Toast.LENGTH_SHORT).show();
        }
    });
  }

  private void getPayments() {
    ApiService apiService =
RetrofitClient.getClient("http://192.168.230.62:3000").create(ApiService.class);
    Call<PaymentResponse> call = apiService.getPayments("1");

    call.enqueue(new Callback<PaymentResponse>() {
        @Override
        public void onResponse(Call<PaymentResponse> call,
Response<PaymentResponse> response) {
            if (response.isSuccessful()) {
                PaymentResponse paymentResponse = response.body();
                if (paymentResponse != null) {
                    textViewTotalPayment.setText("Total Payment: " +
paymentResponse.getTotalPayment());
                    textViewPredictedPayment.setText("Predicted Payment: " +
paymentResponse.getPredictedPayment());
                } else {
                    Toast.makeText(MeterReadingsActivity.this, "Error in response",
Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(MeterReadingsActivity.this, "Failed to get
payments", Toast.LENGTH_SHORT).show();
            }
        }

        @Override
```

```
        public void onFailure(Call<PaymentResponse> call, Throwable t) {


   Toast.makeText(MeterReadingsActivity.this, "Network error: " + t.getMessage(),
Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```