МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки)

Разработка социальных и экономических информационных систем
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка мобильного приложения для управления арендой кофе-машин»

Обучающийся Г.А. Климов _____
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. пед. наук, доцент А.В. Егорова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Тема бакалаврской работы: «Разработка мобильного приложения для управления арендой кофе-машин».

Бакалаврская работа посвящена разработке кроссплатформенного приложения для управления арендой кофемашин.

В ходе выполнения исследований по бакалаврской работе было проведено моделирование приложения, разработан дизайн, выбраны технологии, произведена разработка и проведено тестирование приложения.

Во введении обоснована актуальность темы, сформулированы цели и задачи.

В первой главе в рамках концептуального моделирования рассматриваются принципиальные пользовательские сценарии, на основе которых формируются основные концепции, используемые при разработке.

Во второй главе описывается логическое моделирование приложения, в рамках которого детализированы пользовательские сценарии, разработан дизайн, определена архитектура приложения.

В третьей главе описывается выбор технологий разработки, среды разработки, системы управления базой данных, разбирается процесс разработки приложения, описывается процесс тестирования приложения.

В заключении подведены итоги работы и описывается полученный результат.

Бакалаврской работа состоит из введения, пяти глав, заключения и списка использованной литературы и используемых источников.

Бакалаврская работа состоит из 85 страниц текста, 63 рисунков, 12 таблиц, 3 приложений, содержит информацию из 38 источников.

Abstract

This graduation work is about developing a mobile application for managing coffee machine rentals.

The graduation work consists of an explanatory note on 84 pages, an introduction, five chapters, a conclusion, including 63 figures, 11 tables, a list of 38 references including 34 foreign sources and 3 appendices, and the graphic part on 22 A1 sheets.

The object of the graduation work is the coffee machine rental process.

The work aims to give some information about all stages of mobile application development. We fully cover the conceptual, logical, and physical modeling stages, user interface design, program code writing process, and testing. We first discuss a subject area. We then analyze the coffee machine rental business process and determine how to improve it. After that, we describe the functionality of the future application. Then we model the application using various UML diagrams and develop the design of the application. Finally, we create the application and test it.

In conclusion, we'd like to stress the research covers all stages of software development, so the work can be useful for a wide range of professionals related to information technology.

Оглавление

Введение

Мобильные приложения являются важной частью современного мира. Около 70% населения мира пользуются смартфонами [10], и большая часть из этих пользователей так или иначе использует приложения. Важным фактором выбора того или иного сервиса, в некоторых случаях даже главным, может служить именно наличие удобного приложения. Имея смартфон, приложение и доступ в интернет, пользователи могут воспользоваться сервисом и, например, заказать авиабилеты буквально за несколько кликов.

Это один из примеров того, как приложения помогают бизнесу, но этот пример далеко не единственный. Так, у компании «Бизнес-Апологетика», предоставляющей услуги по аренде кофемашин, появилась потребность своевременно взымать оплату у своих клиентов, предварительно их оповещая о задолженности: практика показала, что клиенты зачастую просто забывают оплачивать аренду. В качестве решения данной проблемы было решено разработать приложение, которое будет выполнять все эти и некоторые другие полезные функции. Это не только должно повысить лояльность пользователей к сервису, но и упростить его использование.

Таким образом, актуальность выбранной темы заключается в том, чтобы обеспечить сервис по аренде кофемашин приложением, которое удовлетворило бы возникшие потребности сервиса и упростило бы его использование.

Объект исследования – процесс аренды кофемашин.

Предметом исследования является мобильное приложение для сервиса по аренде кофемашин.

Целью бакалаврской работы является разработка мобильного приложения для сервиса по аренде кофемашин.

Для достижения цели работы следует выполнить следующие задачи:

- проанализировать бизнес-процессы компании и сформулировать требования для разработки;
- провести концептуальное моделирование ПО;
- произвести логическое моделирование ПО;
- произвести физическое моделирование ПО;
- разработать приложение;
- протестировать приложение.

Практическая значимость работы и приложения заключается в том, что приложение имеет ценность в первую очередь для бизнеса: оно будет автоматизировать ряд задач, связанных с оплатой, оповещениями и отключением кофемашин в случае неуплаты.

Исследования охватывают все этапы разработки программного обеспечения, включая создание концепций, описание пользовательских сценариев, разработку дизайна, разработку архитектуры, написание программного кода и тестирование. Поэтому работа может быть полезна для широкого круга специалистов, связанных с информационными технологиями.

Глава 1 Функциональное моделирование предметной области

1.1 Технико-экономическая характеристика компании

Компания «Бизнес-Апологетика» является небольшой компанией с общей численностью сотрудников не более 100 на момент 5 мая 2024 года. Организационная диаграмма компании показана на рисунке 1.

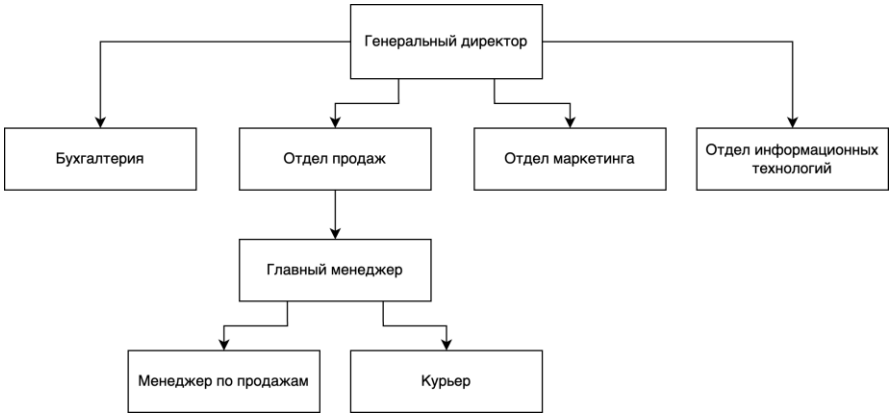

Рисунок 1 – Организационная структура ООО «Бизнес-Апологетика»

Во главе компании стоит генеральный директор, который управляет четырьмя отделами. Отдел продаж имеет особенность, которая заключается в том, что в отделе есть курьер, отвечающий за доставку арендованного оборудования. За координацию действий менеджера по продажам и курьера ответственен главный менеджер.

Офис компании находится в Санкт-Петербурге, там же находится и большинство компании. Компания занимается различными видами деятельности, одним из которых и является аренда кофе-машин. Другим примером может послужить продажа кофейных зерен и сопутствующих товаров. Основные конкуренты – крупные компании, предоставляющие услуги аренды кофемашин и сопутствующих товаров, а также продавцы кофейного оборудования.

1.2 Концептуальное моделирование предметной области

1.2.1 Выбор технологии концептуального моделирования предметной области

В качестве технологий концептуального моделирования предметной области рассматривались нотации ARIS, IDEF0 и UML.

ARIS (Architecture of Integrated Information Systems) – методология, предназначенная для анализа процессов и планирования информационных систем. Она включает в себя различные типы моделей, в том числе модели данных, процессов и организации.

IDEF0 – методология, используемая для моделирования решений, действий и активности организации или системы. IDEF0 является частью семейства языков моделирования IDEF в области системной и программной инженерии.

UML (Unified Modeling Language) – стандартизированный язык моделирования, используемый в основном для спецификации, визуализации, построения и документирования артефактов программных систем. UML включает в себя набор графических нотаций для создания визуальных моделей объектно-ориентированных программных систем.

Есть и другие языки и методологии моделирования, подобные вышеперечисленным, но именно эти три являются основными рассматриваемыми методологиями. Сравнение этих нотаций представлено в таблице 1.

Таблица 1– Сравнение технологий моделирования

Характеристика	ARIS	IDEF0	UML
Направленность	Бизнес-процессы и информационные системы	Функциональное моделирование процессов и систем	Проектирование и разработка программного обеспечения

Продолжение Таблицы 1

Типы моделей	Данные, процессы, организация, IT-инфраструктура	Диаграммы функциональной декомпозиции	Класс, объект, деятельность, сценарий использования
Основное применение	Анализ бизнес-процессов и архитектура ИС	Системный анализ, совершенствование процессов	Разработка программного обеспечения
Визуальное представление	Несколько типов диаграмм для различных аспектов	Блок-схемы со специфическими символами и обозначениями	Разнообразные диаграммы со стандартизированными обозначениями
Сложность	Высокая из-за всесторонней детализации и изменчивости	Средняя, фокусируется на процессах и их иерархиях	Высокая, охватывает широкий спектр аспектов разработки программного обеспечения

В результате была выбрана нотация IDEF0. Выбор был сделан именно этот, так как основной целью является понимание и анализ процессов в ясной и понятной форме без дополнительных сложностей, связанных с организационным моделированием или моделированием данных.

1.2.2 Моделирование бизнес-процессов предметной области для постановки задачи автоматизированного варианта решения

Аренда кофе-машин, отображенная на контекстной диаграмме [31] на рисунке 2**,** состоит из нескольких процессов, которые включают в себя заключение договора на аренду, а также процесс продления аренды. Компания «Бизнес-Апологетика» предоставляет сервис по аренде устройств для кофе. Кофемашины предоставляются в аренду другим бизнесам в основном на территории Санкт-Петербурга. Это происходит так: клиент берет кофемашину в аренду и пользуется ей, оплачивая при этом счета, которые выставляются ежемесячно. Для работы с клиентами на стороне сервиса используется 1С, при этом часть информации заносится в электронную таблицу.

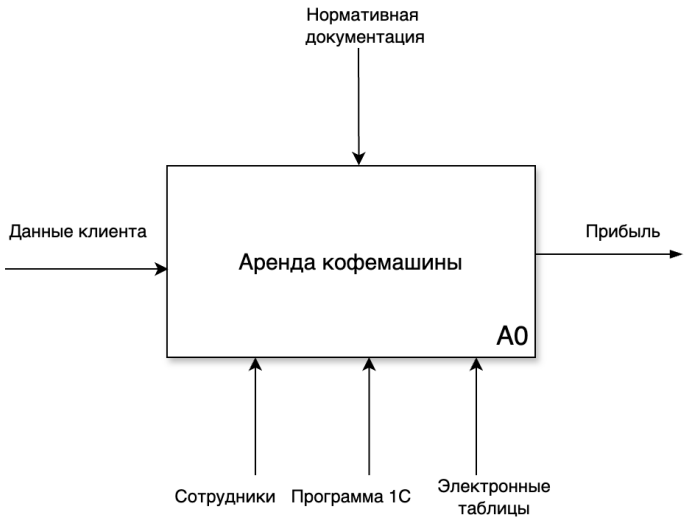

Рисунок 2 – Контекстная диаграмма

Из рисунка 2 видно, что данные расположены не только в 1С, но и в электронных таблицах. Но главным образом сервис функционирует с помощью работы сотрудников, которые коммуницируют с клиентами и, например, вносят все необходимые данные в 1С и электронные таблицы

Декомпозиция системы А0, отображенной на этой диаграмме, представлена на рисунке 3.

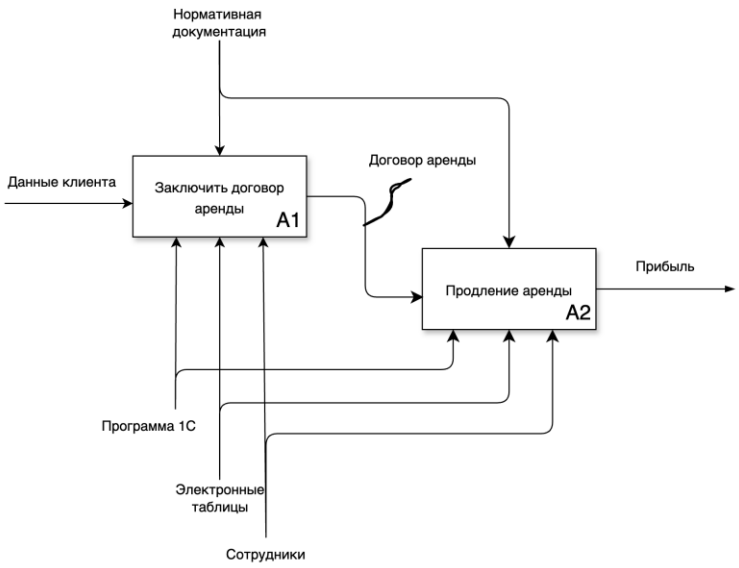

Рисунок 3 – Декомпозиция процесса аренды кофемашин

Таким образом, организация получает прибыль после того, как клиент оплачивает счет при продлении аренды.

1.2.3 Разработка и анализ модели бизнес-процесса «как есть» и формулировка требований

Из всего вышесказанного следует, что основной процесс, который следует рассмотреть – это процесс продления аренды. На рисунке 4 представлена декомпозиция этого процесса «как есть».

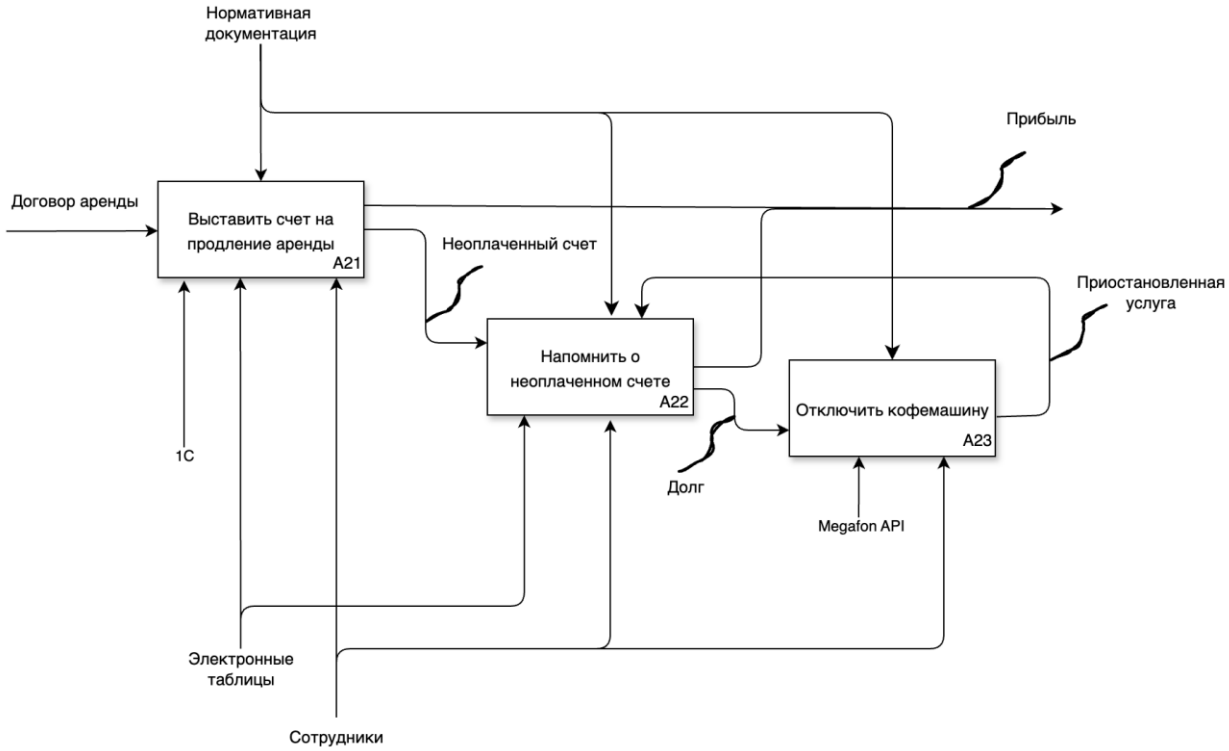

Рисунок 4 – Диаграмма бизнес-процесса продления аренды «как есть»

Из данной диаграммы видно, что практически на всех этапах, связанных с арендой кофемашин, необходимо привлекать сотрудников компании. Сотрудники делают все – выставляют счета клиентам, напоминают им об оплате, отключают кофемашины при необходимости – имеется возможность отключить кофе-машину путем отправки специального сообщения на телефонный номер.

Бизнес начинает сталкиваться с проблемами, когда клиент по тем или иным причинам не оплачивает счет. Так как клиентов много, нужны ресурсы для того, чтобы отследить, кто не оплатил счет, и напомнить им об этом. В некоторых случаях долг может быть незамеченным и накапливаться, это происходит из-за человеческого фактора. Таким образом, бизнес столкнулся с проблемой масштабируемости: чем больше клиентов, тем больше человеческих ресурсов требуется на их обслуживание.

Чтобы решить проблему, на всех кофемашинах были установлены специальные устройства, позволяющие удаленно их выключать, если клиент не оплатил аренду. Это может мотивировать клиента вовремя оплачивать аренду, однако большая часть проблем осталась. Так, например, остался человеческий фактор. Клиента могут забыть предупредить о том, что пора оплатить счет, либо клиент может по каким-то причинам забыть его оплатить. Также клиенты склонны откладывать оплату аренды, так как это сопряжено с некоторыми трудностями: для того, чтобы ее оплатить, нужно заполнить множество реквизитов.

Для решения всех проблем было предложено создать приложение, которое будет взаимодействовать с 1С, выставлять счета на оплату, выключать кофемашины и оповещать пользователя о том, что счет был выставлен.

На основе имеющихся данных определим требования к мобильному приложению, воспользовавшись моделью классификации FURPS+ [3]. Данная модель включает в себя Functionality (функциональность), Usability (удобство использования), Reliability (надежность), Performance (производительность), Supportability (поддерживаемость) [11]. Также модель включает определение возможных ограничений (символ «+» в названии модели) [18].

Функциональность:

− просмотр арендованных кофемашин и их состояний (включена/отключена);

- оплата аренды;
- оповещения;
- отключение кофемашин в случае неуплаты и включение в случае оплаты.

Удобство использования:

- возможность оплаты по QR-коду;
- возможность копировать платежные реквизиты в буфер обмена;
- поддержка светлой и темной цветовых схем.

Надежность:

- использование протокола SSL;
- использование безопасной авторизации.

Производительность:

- время отклика сервера до 5 секунд.

Поддерживаемость:

- iOS 17 и выше;
- Android 6 и выше;
- язык приложения – русский;
- соответствие приложения правилам магазинов App Store и Google Play.

Ограничением является законодательное: данные пользователей должны храниться на территории Российской Федерации.

1.3 Сравнительная характеристика существующих разработок и постановка задачи

На рынке приложений уже есть похожие разработки, позволяющие арендовать кофемашины. Сравнительная характеристика представлена в таблице 2.

Таблица 2 – Сравнение существующих разработок

Параметр	Brew Coffee	Coffee Rent	Coffee Machine Pro	Easy Coffee Rental
Предназначен для	Дом и офис	Бизнес	Кафе и рестораны	Дом и малый бизнес
Удобство использования	Интуитивное	Онлайн-консультант	Профессиональный	Легкая навигация
Стоимость	Гибкие тарифы	Пакеты с доп. услугами	Высокий сегмент	Конкурентные цены
Ассортимент оборудования	Широкий выбор	Профессиональные	Премиальные модели	Разные варианты
Условия аренды	Гибкие сроки, доставка	Гибкие, обслуживание	Долгосрочные, выкуп	Краткосрочные и долгосрочные

После сравнения и анализа аналогов было решено начать разработку нового приложения.

Выше были сформулированы требования на разработку приложения, которое покроет нужды бизнеса и освободит ресурсы. Из всего вышеперечисленного можно сформулировать задачу на разработку проекта. Задача состоит в следующем: разработать мобильное приложение в соответствии с вышеперечисленными требованиями, включая не только клиентскую, но и серверную часть. Серверная часть должна быть интегрирована с 1С таким образом, чтобы данные синхронизировались и были всегда актуальными.

1.4 Разработка модели бизнес-процесса «как должно быть»

Выше была представлена диаграмма бизнес-процесса «как есть». Так как все требования уже сформированы, есть возможность построить модель бизнес-процесса аренды кофемашин «как должно быть». На основе всех имеющихся данных и анализа бизнес-процессов предприятия и была построена эта модель. Она представлена на рисунке 5.

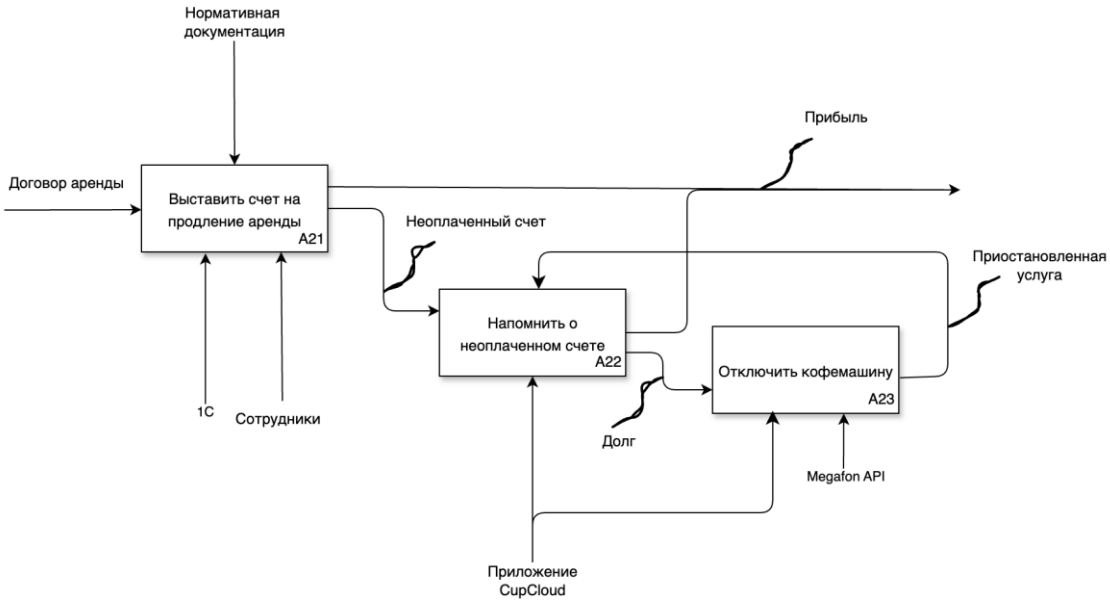

Рисунок 5 – Диаграмма «как должно быть»

Из диаграммы видно, что сотрудники не учувствуют в большом количестве процессов, вместо них большую часть работы берет на себя разрабатываемое приложение «CupCloud».

Выводы по главе 1

В главе 1 «Функциональное моделирование предметной области» рассматривается бизнес-сфера аренды кофемашин. Процесс аренды управляется с помощью программного обеспечения и ежемесячного выставления счетов. На контекстной диаграмме показано, как данные о клиентах и финансовые операции управляются с помощью программного обеспечения 1С и электронных таблиц, что предполагает значительное взаимодействие сотрудников при вводе данных и общении с клиентами.

В главе также рассматриваются различные методологии концептуального моделирования, включая ARIS, IDEF0 и UML, каждая из которых предназначена для различных аспектов моделирования систем и бизнес-процессов. В конечном счете, IDEF0 была выбрана за ее ясность и

простоту в понимании процессов без сложностей организационного моделирования или моделирования данных.

Далее в главе описывается текущий процесс аренды, подчеркивается зависимость от ручных процессов, таких как продление контрактов и контроль оплаты, которые подвержены человеческим ошибкам. В ней предлагается автоматизированное решение с помощью мобильного приложения, которое позволит оптимизировать операции за счет интеграции с 1С, обработки платежей, отключения машины за неуплату и уведомлений клиентов, повышая эффективность и масштабируемость.

Наконец, в главе подробно описаны требования к новой технологии, основанные на модели FURPS+, с акцентом на функциональность, удобство использования, надежность, производительность и поддерживаемость.

Глава 2 Логическое проектирование мобильного приложения

2.1 Выбор технологии логического моделирования мобильного приложения

В качестве основной технологии логического моделирования был выбран UML. Он является предпочтительным для логического моделирования благодаря своей стандартизации и универсальности в визуализации системных архитектур. Разработанный для унификации различных методов моделирования, UML имеет решающее значение как в образовательной, так и в профессиональной среде благодаря наглядному представлению системных проектов.

В отличие от других технологий, таких как ERD, которые фокусируются на отношениях в базе данных, или BPMN, которая нацелена на бизнес-процессы, UML охватывает как структурные, так и поведенческие аспекты систем. Это делает его адаптируемым для различных этапов разработки программного обеспечения и ценным в среде с различными заинтересованными сторонами. Его всеобъемлющий инструментарий способствует плавному переходу от проектирования к реализации, обеспечивая UML позицию предпочтительного инструмента в программной инженерии.

2.2 Логическая модель мобильного приложения и ее описание

2.2.1 Описание вариантов использования

Диаграмма вариантов использования для приложения представлена на рисунке 6.

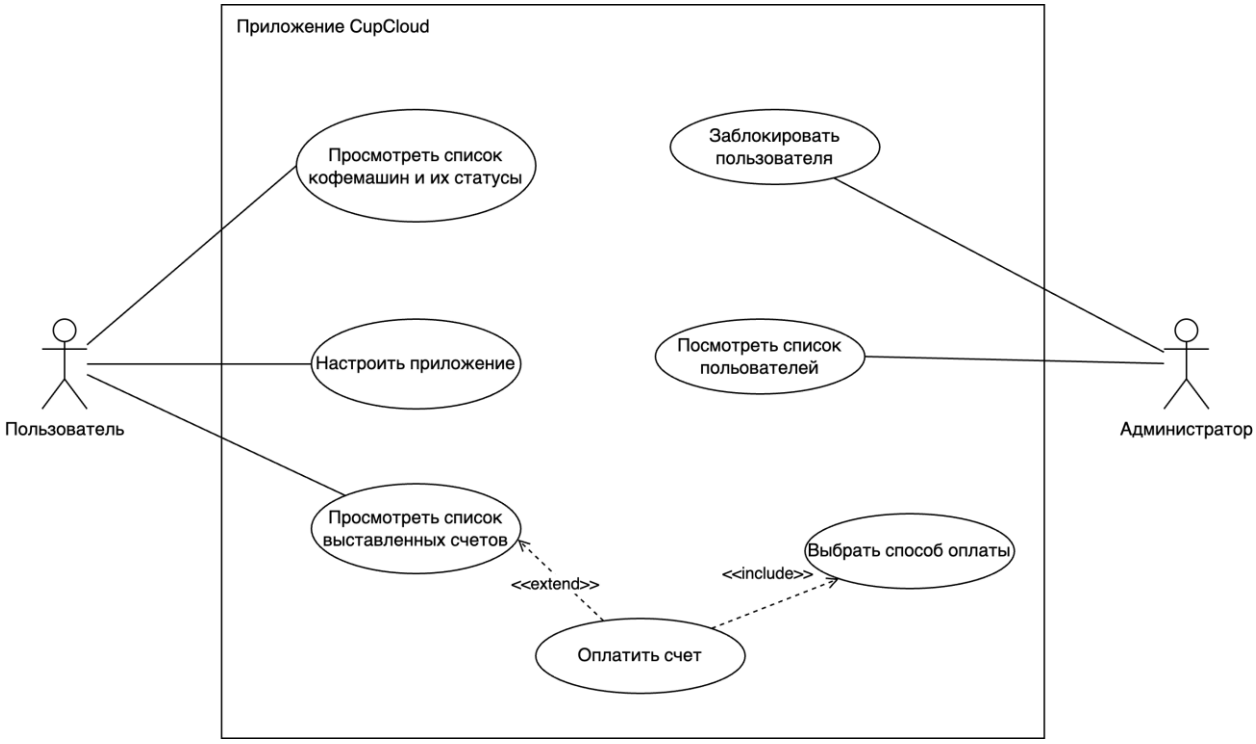

Рисунок 6 – Диаграмма вариантов использования

Как видно из этой диаграммы, любой пользователь приложения имеет возможность зарегистрироваться, войти в систему или восстановить доступ. В то же время, пользователь с обычными правами имеет возможность просматривать список кофемашин и их статусы, настраивать приложение и работать с выставленными счетами. Администратор имеет возможность управлять пользователями.

На основе данной диаграммы были разработаны и описаны сценарии использования (Use Cases) приложения детализовано.

В таблице 3 показан сценарий авторизации (входа в систему) для пользователя и администратора.

Таблица 3 – Сценарий авторизации

Описание	Войти в систему	
Акторы	Пользователь, Администратор	
Предусловия	Приложение открыто Актор зарегистрирован в системе	
Результат	Актор вошел в систему (авторизировался)	
Основной сценарий	Действующее лицо	Действие
	Пользователь, администратор	1. Ввод email и пароля
	Система	2. Проверка, что пользователь с этим email зарегистрирован В случае неуспеха переход на сценарий 2а
	Система	3. Проверка, что пользователь ввел верный пароль В случае неуспеха переход на сценарий 3а
	Система	4. Предоставление доступа к основному функционалу
Email не зарегистрирован (альтернативный сценарий)	Пользователь	2а. Ввел несуществующий email
	Система	2б. Вывод сообщения об ошибке
Пароль не верный (альтернативный сценарий)	Пользователь	3а. Ввел неверный пароль
	Система	3б. Вывод сообщения об ошибке

В случае ввода неправильной пары email-пароль пользователю будет показано сообщение об ошибке, в противном случае будет открыт основной интерфейс приложения. Интерфейс приложения различается в зависимости от роли пользователя: администраторам будет показан специальный интерфейс, позволяющий управлять пользователями. Сценарий управления пользователями будет описан ниже.

Если пользователь еще не зарегистрирован, у него имеется возможность пройти регистрацию. Сценарий показан в таблице 4.

Таблица 4 – Сценарий регистрации

Описание	Зарегистрироваться в системе	
Акторы	Пользователь, система	
Предусловия	Приложение открыто	
Результат	Пользователь зарегистрирован в системе и может использовать email и пароль для входа в нее	
Основной сценарий	Действующее лицо	Действие
	Пользователь	1. Ввод email, пароля и пароля повторно
	Система	2. Проверка, что введенные пароли совпадают В случае неуспеха переход на сценарий 2а
	Система	3. Проверка, что email еще не зарегистрирован В случае неуспеха переход на сценарий 3а
	Система	4. Отправка пользователю письма с проверочным кодом на указанный email
	Пользователь	5. Ввод проверочного кода
	Система	6. Проверка кода. С случае неверного кода переход на сценарий 6а
	Система	7. Регистрирует пользователя с email и введенным ранее паролем с правами обычного пользователя
	Система	8. Автоматически авторизирует пользователя
Введенные пароли не совпадают (альтернативный сценарий)	Пользователь	2а. Ввел разные пароли в поля для ввода пароля
	Система	2б. Вывод сообщения об ошибке
Email уже зарегистрирован (альтернативный сценарий)	Пользователь	3а. Ввел уже зарегистрированный email
	Система	3б. Вывод сообщения об ошибке
Пользователь ввел неверный проверочный код (альтернативный сценарий)	Пользователь	4а. Ввел неверный проверочный код
	Система	4б. Вывод сообщения об ошибке, Предложение отправить код повторно
Повторная отправка кода (Расширение 3б)	Пользователь	4б-1. Выбирает возможность повторной отправки кода
	Система	4б-2. Генерирует новый проверочный код и отправляет его на email
	Пользователь	4б-3. Переход к шагу 5 «Ввод проверочного кода»

После регистрации пользователь переходит в состояние «Авторизирован» сразу же, без повторного ввода учетных данных.

Последний сценарий, связанный с авторизацией – это восстановление доступа. Он представлен в таблице 5.

Таблица 5 – Сценарий восстановления доступа

Описание	Восстановить доступ	
Акторы	Пользователь, администратор, система	
Предусловия	Пользователь зарегистрирован в системе	
Результат	Пользователь зарегистрирован в системе и может использовать email и пароль для входа в нее	
Основной сценарий	Действующее лицо	Действие
	Пользователь	1. Ввод email
	Система	2. Проверка, что email зарегистрирован в системе. В случае неуспеха переход на сценарий 2а
	Система	3. Отправка пользователю письма с проверочным кодом на указанный email
	Пользователь	4. Ввод проверочного кода, нового пароля и нового пароля повторно
	Система	5. Проверка совпадения введенных паролей. В случае неверного кода переход на сценарий 5а
	Система	6. Проверка кода. С случае неверного кода переход на сценарий 6а
	Система	7. Устанавливает новый пароль для пользователя с данным email
	Система	8 Автоматически авторизирует пользователя
Email не зарегистрирован (альтернативный сценарий)	Пользователь	2а. Ввел незарегистрированной в системе email
	Система	2б. Вывод сообщения об ошибке
Email уже зарегистрирован (альтернативный сценарий)	Пользователь	3а. Ввел уже зарегистрированный email
	Система	3б. Вывод сообщения об ошибке
Введенные пароли не совпадают (альтернативный сценарий)	Пользователь	5а. Ввел неверный разные пароли в поля для ввода нового пароля
	Система	5б. Вывод сообщения об ошибке
Код введен неверно (альтернативный сценарий)	Пользователь	6а. Пользователь вводит ошибочный код
	Система	6б. Вывод сообщения об ошибке

Для восстановления доступа к системе пользователю с любой ролью необходимо указать email и затем ввести отправленный на него проверочный код. Таким образом, главная точка доступа – это email, имея доступ к которому можно будет всегда восстановить доступ к аккаунту.

Теперь рассмотрим сценарии использования приложения. При входе в приложение пользователь может просмотреть список арендованных устройств. Сценарий довольно прост и не имеет альтернативных сценариев и расширений. Он представлен в таблице 6.

Таблица 6 – Сценарий просмотра арендованных устройств

Описание	Просмотр арендованных устройств	
Акторы	Пользователь, система	
Предусловия	Пользователь авторизован в системе	
Результат	Пользователь может просмотреть список арендованных устройств	
Основной сценарий	Действующее лицо	Действие
	Пользователь	1. Пользователь открывает главную страницу
	Система	2. Система отображает список арендованных устройств

В таблице 7 описан сценарий просмотра выставленных системой счетов.

Таблица 7 – Сценарий просмотра выставленных системой счетов

Описание	Просмотреть список счетов	
Акторы	Пользователь, система	
Предусловия	Приложение открыто Пользователь авторизирован	
Результат	Пользователь может просмотреть список выставленных счетов за аренду	
Основной сценарий	Действующее лицо	Действие
	Пользователь	1. Пользователь открывает страницу «Счета»
	Система	2. Система отображает список неоплаченных счетов

Ключевым сценарием является возможность оплачивать счета. Этот сценарий представлен в таблице 8.

Таблица 8 – Сценарий оплаты счета

Описание	Оплатить счет	
Акторы	Пользователь, Система, Бухгалтер	
Предусловия	Пользователь авторизирован	
Результат	Счет оплачен	
Основной сценарий	Действующее лицо	Действие
	Пользователь	1. Открывает страницу «Счета», выбирает счет и нажимает «Оплатить»
	Система	2. Отображает все необходимые данные для оплаты счета: наименование организации, номер расчётного счета, наименование банка, корреспондентский счет, БИК и сумму к оплате, либо QR-код, содержащий все необходимые данные
	Пользователь	3. Оплачивает счет Если пользователь не оплачивает счет в течение трех дней после его выставления, переход на сценарий 3а
	Бухгалтер	4. Вносит в систему данные об оплате
	Система	5. Помечает счет оплаченным
	Система	6. Включает устройство, если оно было отключено
Пользователь не оплатил счет в течение трех дней (альтернативный сценарий)	Система	3а. Отключает устройство

Если пользователь не оплачивает счет в течение трех дней, система отключает устройство. На данный момент слабым местом системы является человеческий фактор, а именно – бухгалтер должен внести данные об оплате счета в систему. Недостаток может быть исправлен в будущем с помощью внедрения платежных систем в приложение, и приложение должно быть спроектировано таким образом, чтобы сложностей с будущей интеграцией не возникло. На текущем этапе внедрение платежной системы не предусмотрено, так как влечет за собой дополнительные расходы на эквайринг и не предусмотрено текущими бизнес-процессами.

Последний пользовательский сценарий – это сценарий, связанный с настройками приложения, он представлен в таблице 9.

Таблица 9 – Сценарий настройки приложения

Описание	Настроить приложение	
Акторы	Пользователь, система	
Предусловия	Приложение открыто Пользователь авторизирован	
Результат	Приложение настроено в зависимости от потребностей пользователя	
Основной сценарий	Действующее лицо	Действие
	Пользователь	1. Открывает страницу «Настройки»
	Система	2. Отображает текущие настройки аккаунта и уведомлений.
	Пользователь	3. Настраивает приложение посредством использования интерфейса

К вышеперечисленному следует также добавить, что настройки включают в себя возможность выйти из аккаунта и удалить аккаунт.

Все экраны клиентской части приложения показаны с помощью диаграммы состояний в приложении А на рисунке А.1.

Как видно из этой диаграммы, приложение будет состоять из 11 экранов. Основными из отображенных состояний являются экраны «Главная», «Счета» и «Настройки».

2.2.2 Архитектура приложения

Как было сказано в ходе концептуального проектирования, очевидным и практически безальтернативным выбором для разработки приложения является клиент-серверная архитектура, показанная на рисунке 7 [2].

.

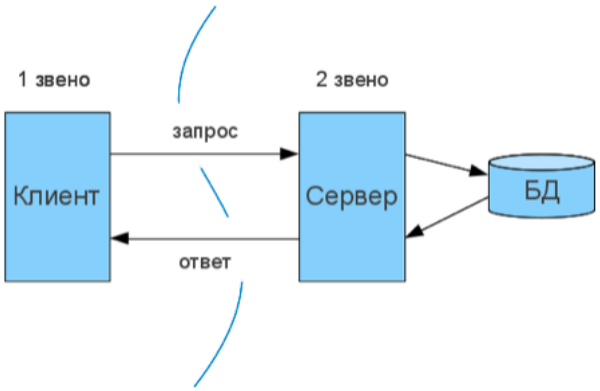

Рисунок 7 – Архитектура «Клиент-Сервер»

Однако, архитектуру как клиентской, так и серверной части следует рассмотреть отдельно, поскольку и клиент, и сервер сами по себе могут представлять из себя довольно сложные части приложения, которые могут состоять из различных модулей и иметь многослойную структуру.

Для клиентской и серверной части подойдет общий архитектурный концепт, называемый «Чистая архитектура». Чистая архитектура – это подход к проектированию, разработанный Робертом Мартином, который фокусируется на организации программного обеспечения таким образом, чтобы максимально способствовать масштабируемости и разделению задач. [17]. Главная идея заключается в том, что приложение разделяется на слои, взаимодействие между которыми осуществляется с использованием правила зависимости [1]. Это гарантирует, что основная функциональность приложения может развиваться независимо от внешних изменений, таких как технологии баз данных или веб-фреймворки.

На рисунке 8 показаны слои чистой архитектуры.

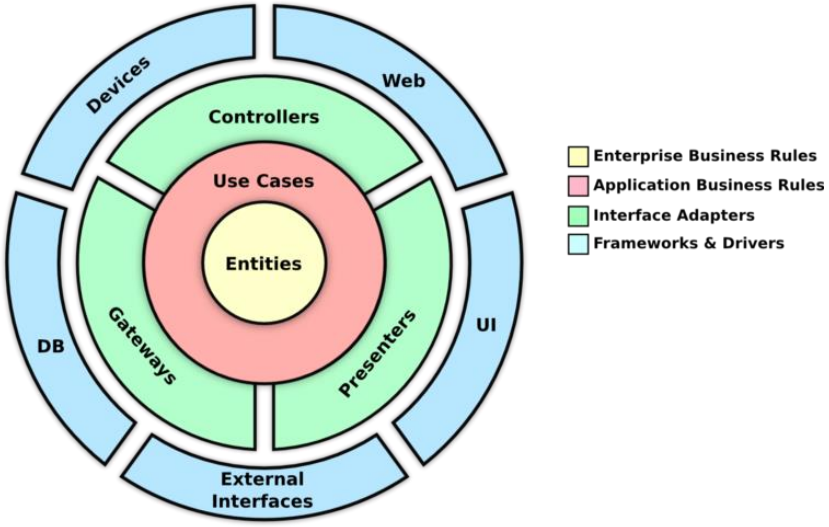

Рисунок 8 – Чистая архитектура

Правило зависимостей говорит о том, что внутренний слой не зависит от внешнего.

На внутреннем слое находятся сущности, связанные с бизнес-логикой. На слое Use Cases находится непосредственно бизнес-логика. Эти слои взаимодействуют с внешними слоями, к которым можно отнести базы данных, пользовательский интерфейс и другие внешние взаимодействия, через слой Interface Adapters.

Разделение задач в чистой архитектуре обеспечивает среду разработки, благоприятную для тестирования, сопровождения и долгосрочной устойчивости.

Для реализации этой архитектуры в проекте в первую очередь следует разделить по пакетам так, как показано на рисунке 9.

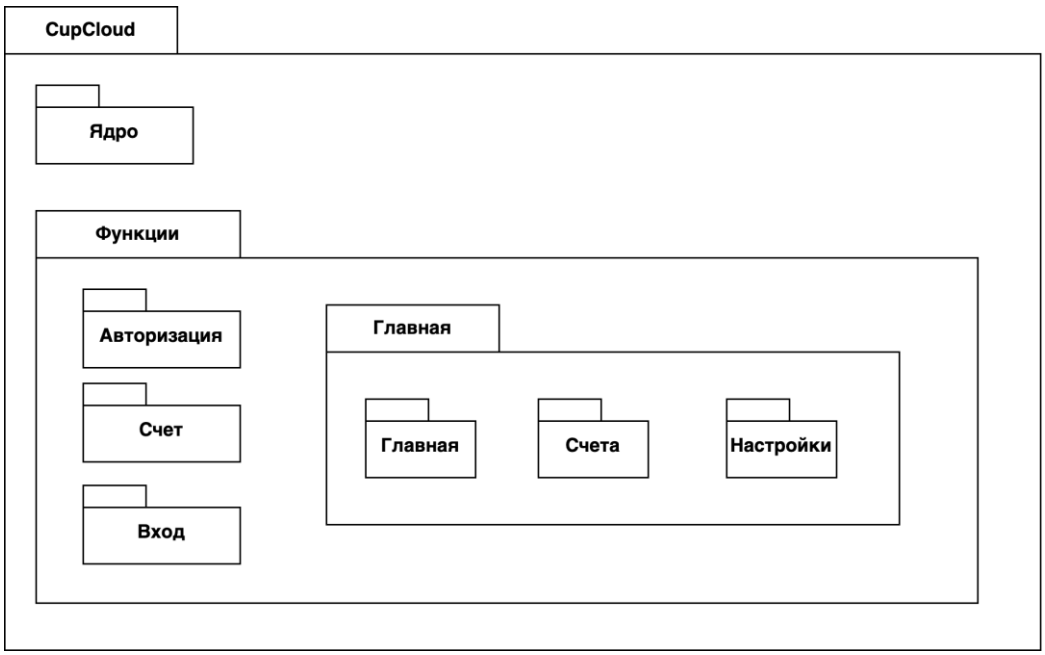

Рисунок 9 – Диаграмма пакетов приложения с использованием «Чистой архитектуры»

Приложение делится на основные функции – «Ядро», который включает различные классы, общие для всех функций, и пакет «Функции», который включает в себя пакеты, соответствующие различным функциям, которые в данном случае соответствуют различным экранам приложения.

Каждая функция (обычно содержит 3 пакета – data, domain и presentation, как показано в общем виде на рисунке 10.

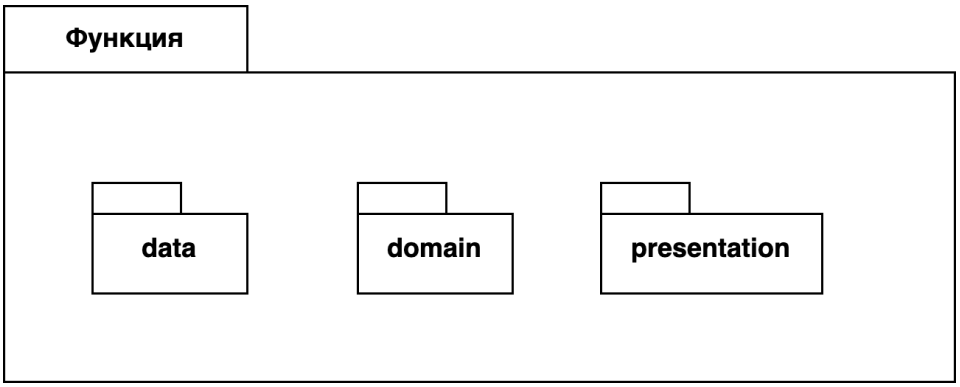

Рисунок 10 – Диаграмма пакетов функции в общем виде

Пакет domain является главным пакетом и содержит классы, связанные с бизнес-логикой приложения. Кроме того, он содержит интерфейс репозитория. Реализация репозитория находится в пакете data. Также в пакете data находятся различные источники данных – например, внешнее хранилище. В пакете presentation находятся классы, связанные с пользовательским интерфейсом. Диаграмма пакетов функции в общем виде подробно представлена на рисунке 11 .

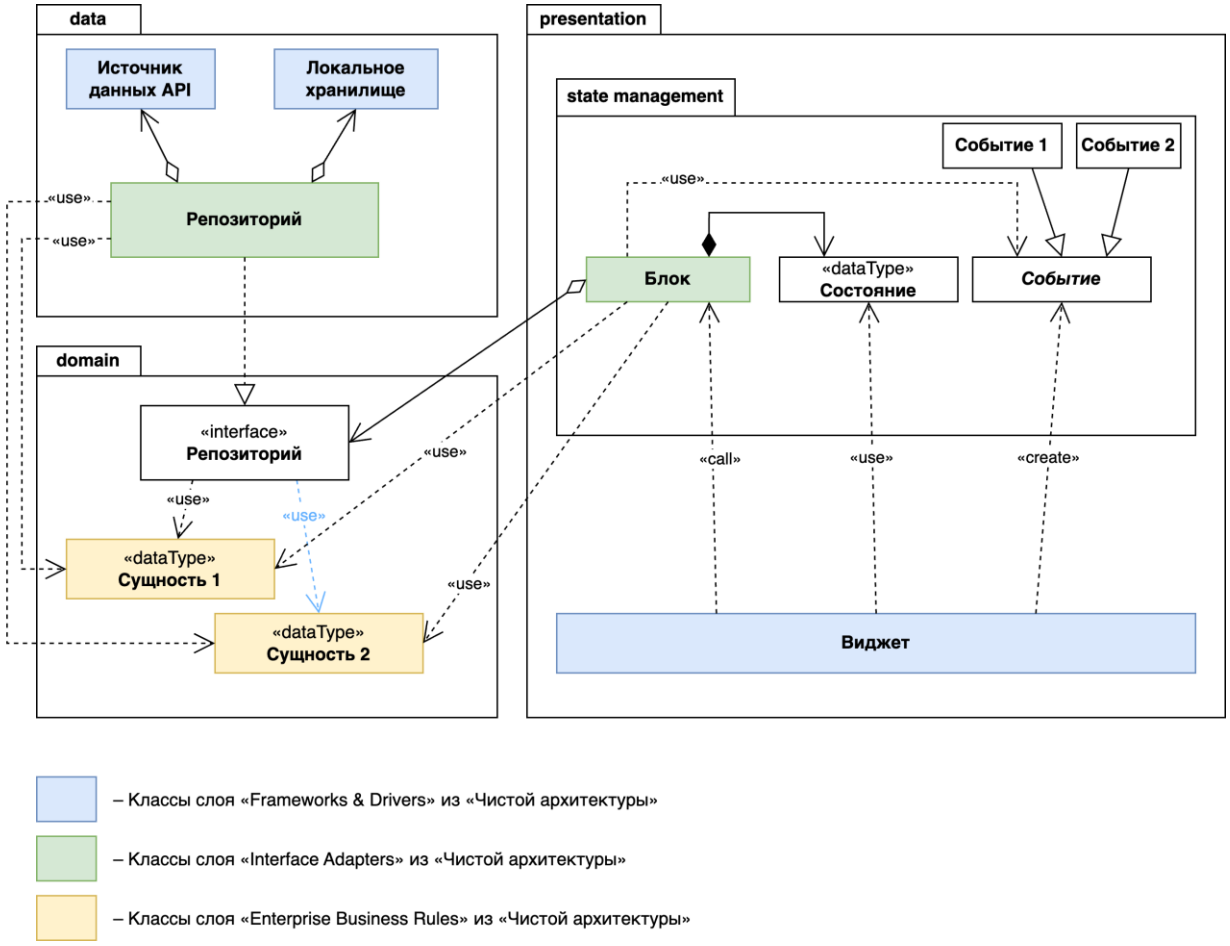

Рисунок 11 – Диаграмма пакетов функции

Из диаграммы видно, что пакет domain содержит основные сущности, которые могут быть использованы из любого места приложения, и интерфейс репозитория.

От репозитория зависит пакет presentation, в котором находятся пакет State Management, реализующий одноименный паттерн проектирования. Класс Виджет, который отвечает непосредственно за отображение пользовательского интерфейса, отправляет события в класс Блок. Блок реагирует на события, взаимодействует с репозиторием и изменяет свое состояние. Виджет перестраивается тогда, когда состояние изменяется.

Следует заметить, что в пакете domain во многих случаях отсутствует класс, отвечающий за бизнес-логику из-за его избыточности. Однако, могут возникнуть случаи, когда бизнес-логика может понадобиться. В этом случае реализуется класс Интерактор, с которым и взаимодействует класс Блок. Диаграмма пакетов, в которой показан Интерактор, представлена на рисунке 12.

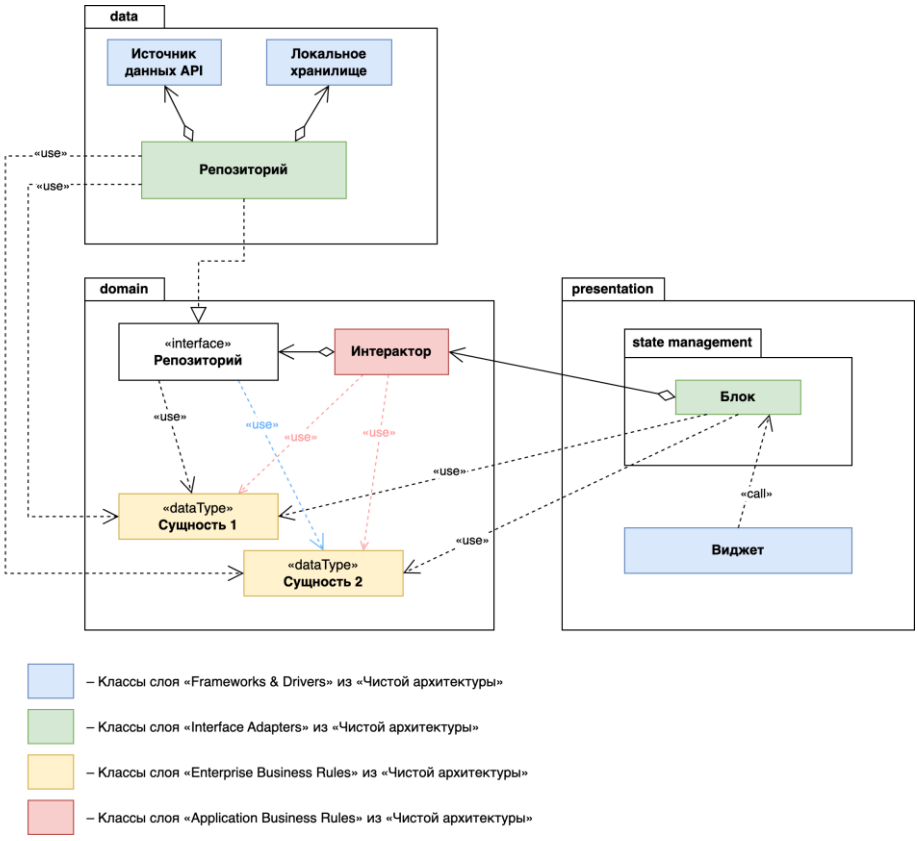

Рисунок 12 – Диаграмма пакетов функции с использованием класса Интерактор

Из данного рисунка видно, как Интерактор взаимодействует с другими классами.

На рисунках 11 и 12 также отображено, как слои, показанные на рисунке 8, отражены в пакетах. Например, видно, что классы Блок и Репозиторий являются «Interface Adapters», и именно через них слой бизнес-логики взаимодействует с другими слоями в архитектуре.

Рассмотрим реализацию авторизации на клиентском приложении в качестве примера. Диаграмма классов авторизации представлена на рисунке 13.

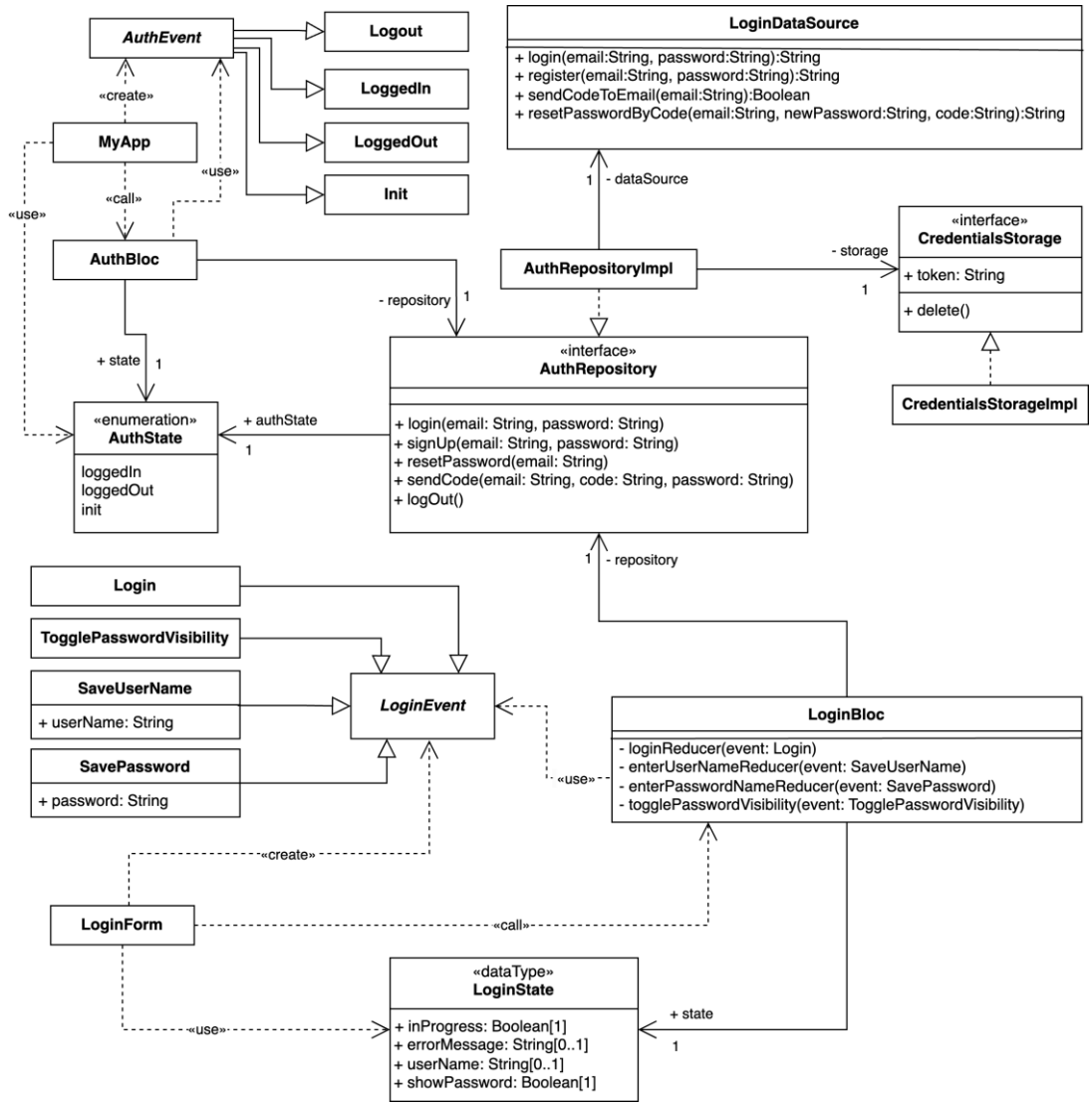

Рисунок 13 – Диаграмма классов авторизации

На диаграмме показано, что реализация интерфейса AuthRepository в данном случае взаимодействует с двумя источниками данных. Источник данных LoginDataSource взаимодействует с серверной частью приложения, получая оттуда данные: например, операция login(), показанная на диаграмме, возвращает строковое значение JWT-токена в случае успешной авторизации. Источник данных CredentialsStorage предоставляет сохраненный локально JWT-токен и содержит операцию delete() для его удаления. Интерфейс AuthRepository всегда содержит актуальное состояние AuthState, с помощью которого можно определить, авторизирован ли текущий пользователь или нет. Класс AuthBloc запрашивает это состояние у AuthRepository. Когда AuthState меняется, главный виджет приложения MyApp показывает либо экран логина, либо отображает интерфейс главной страницы приложения. События AuthEvent, ассоциированные с классом AuthBloc, создаются в MyApp и передаются в AuthBloc. Например, если пользователь в каком-либо месте приложения нажимает кнопку «Выйти из аккаунта», то в AuthBloc передается событие LoggedOut.

Виджет LoginForm является формой для входа пользователя в аккаунт. Например, при вводе имени пользователя LoginForm создает событие SaveUserName, и пароль из атрибута userName класса LoginForm попадает в одноименный атрибут состояния LoginState.

Если LoginBloc при вызове какого-либо метода у AuthRepository получает в ответ ошибку, LoginBloc изменит состояние LoginState, добавив сообщение об ошибке в атрибут errorMessage.

2.3 Информационное обеспечение мобильного приложения

2.3.1 Используемые классификаторы и системы кодирования

Используемые классификаторы и системы кодирования представлены в таблице 10.

Таблица 10 – Классификаторы

Наименование кодируемого множества объектов	Система кодирования	Вид классификатора
Электронный почтовый адрес пользователя	RFC 5322	Международный
Номер телефона кофемашины	ITU-T E.164	Международный
Идентификаторы объектов «Контрагент», «Пользователь», «Кофемашина», «Счет»	ObjectId [29]	Системный

Все идентификаторы соответствуют классификатору ObjectId, размер которого равен 12 байт [29]. Он содержит:

- 4-байтовую временную метку, отражающую создание ObjectId, измеряемую в секундах с момента эпохи Unix;
- 5-байтовое случайное значение, генерируемое один раз для каждого процесса. Это случайное значение уникально для машины и процесса;
- 3-байтовый инкрементирующий счетчик, инициализированный случайным значением.

Помимо этого, используются международные классификаторы для электронного почтового адреса и для номера телефона, ассоциированного с кофемашиной.

2.3.2 Характеристика нормативно-справочной, входной оперативной информации и выходной информации

За входную информацию для приложения отвечает 1С, с которым приложение взаимодействует через Rest API. Описание клиента в формате OpenAPI 3.0 представлено в приложении Б.

Rest-клиент, с которым взаимодействует серверная часть приложения, содержит метод «GET /contragents», который возвращает сразу всю необходимую информацию, которая впоследствии сохраняется в базе данных.

Вся входящая информация, описанная в предыдущем разделе, преобразовывается в информацию, необходимую для отображения в пользовательском интерфейсе. Она имеет схожие характеристики с входящей информацией. Особенностями исходящей информацией является разделение всех данных на данные, необходимые для отображения конкретным пользователям.

Еще одной особенностью исходящих данных являются уведомления, которые формируются на основе входящих данных. В приложении С на рисунке Б.1 с помощью диаграммы деятельности показано, как формируются уведомления и другие выходные данные.

2.4 Проектирование базы данных приложения

На рисунке 14 приведена UML-диаграмма классов, показывающая структуру базы данных.

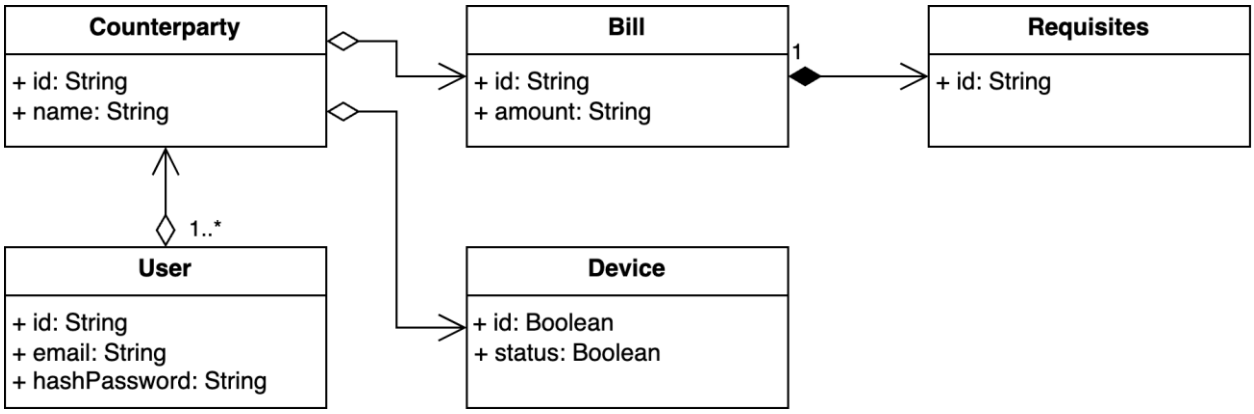

Рисунок 14 – Структура базы данных приложения

Из диаграммы видно, что были выявлены 4 базовые сущности приложения: User (Пользователь), Counterparty (Контрагент), Bill (Счет за аренду), Device (Кофемашина). Между User и Counterparty, между Counterparty и Bill, между Counterparty и Device связь «Один-ко многим».

Между Bill и Requisites связь «Один к одному», Requisites – вспомогательная сущность, которая может быть объединена с Bill.

2.5 Требования к программному обеспечению приложения

2.5.1 Формат обмена между клиентом и сервером

В качестве формата взаимодействия клиента и сервера были выбраны REST API и JSON, соответствующие современным практикам разработки. REST (Representational State Transfer) – это широко распространенный архитектурный стиль, который использует стандартные методы HTTP, такие как GET, POST, PUT и DELETE, что делает его интуитивно понятным и простым для разработки веб-сервисов [25].

JSON (JavaScript Object Notation) дополняет REST благодаря своему облегченному формату обмена данными, который легко читать и писать людям, а машинам – разбирать и генерировать [16]. Совместимость JSON с широким спектром языков программирования способствует его повсеместному распространению в качестве средства обмена данными. Его структурированный, но гибкий формат поддерживает иерархическое расположение, что особенно удобно для сложных структур данных, и менее многословен по сравнению с другими форматами, такими как XML, что приводит к ускорению разбора и снижению накладных расходов на передачу данных.

Вместе REST API и JSON способствуют быстрому циклу разработки и снижают сложность соединения клиентских и серверных операций [27]. Эта комбинация также поддерживает широкий спектр устройств и сервисов, что делает ее надежным выбором для приложений разного рода.

2.5.2 Авторизация

Для авторизации на стороне сервера была выбрана авторизация с помощью JWT-токенов [21]. Выбор JWT (JSON Web Tokens) для авторизации в приложении – это стратегическое решение, которое повышает

безопасность и масштабируемость, а также упрощает процессы транзакций между клиентами и серверами. JWT представляют собой компактный и самодостаточный способ безопасной передачи информации между сторонами в виде JSON-объекта, который можно проверить и доверить, поскольку он имеет цифровую подпись. Кроме того, JWT поддерживают децентрализованный подход к аутентификации [19]. Каждый токен является самодостаточным и несет в себе всю необходимую информацию о пользователе, что упрощает процесс проверки пользовательских сессий в архитектуре микросервисов без необходимости постоянно запрашивать центральный провайдер идентификации или базу данных.

2.5.3 Требования к аппаратным ресурсам

В данный момент у заказчика нет аппаратных ресурсов, необходимых для работы приложения, поэтому сформируем эти требования.

Как было определено выше, приложение состоит из клиентской части и серверной части, а также базы данных. Основываясь на этом, а также на проведенном моделировании, требования к аппаратным ресурсам следующие: необходимо арендовать 2 виртуальные машины с минимальной конфигурацией. Допускается арендовать одну виртуальную машину, а для СУБД использовать сторонний сервис.

2.6 Проектирование пользовательского интерфейса приложения

2.6.1 Выбор средств и технологий проектирования интерфейса

Для разработки дизайна пользовательского интерфейса приложения был выбран Material Design [26], разработанный Google. Это общепризнанный язык дизайна, который подчеркивает чистую, минималистичную эстетику, основанную на принципах бумаги и чернил [8]. Он использует эффекты глубины, такие как освещение и тени, чтобы создать ощущение иерархии и сосредоточенности, что делает его визуально интуитивным [20]. По сравнению с другими руководствами по дизайну,

такими как iOS Human Interface Guidelines от Apple, которые делают акцент на глубине, прозрачности и размытой эстетике, чтобы передать ощущение многослойности пользовательского интерфейса, Material Design использует больше геометрических форм и жестких краев, придавая приложениям более единообразный вид на Android и веб-платформах. В то время как дизайн Apple создан для глубокой интеграции с аппаратным обеспечением, Material Design нацелен на последовательный пользовательский опыт на разных платформах, включая устройства, не принадлежащие Google. Выбор Material Design может быть полезен для продуктов, нацеленных на широкую аудиторию на разных устройствах, то есть как раз для нашего случая [24]. Его принципы помогают создать бесшовный пользовательский опыт на Android и в веб-приложениях, что делает его идеальным для разработчиков, стремящихся к кроссплатформенной согласованности и удобству использования. Кроме того, Material Design опирается на надежный набор инструментов и компонентов, которые упрощают процесс разработки и обеспечивают доступность, что делает его практичным выбором для широкого круга приложений.

2.6.2 Описание принципов проектирования пользовательского интерфейса приложения

Дизайн приложения должен быть разработан в двух цветовых схемах – темной и светлой. Поддержка светлых и темных цветовых схем в современных приложениях очень важна в связи с растущим разнообразием предпочтений пользователей и ситуаций, в которых они используются [15]. Светлые темы обычно предпочитают использовать в ярком окружении, где они улучшают читаемость, а темные – в условиях недостаточной освещенности, чтобы уменьшить блики и нагрузку на глаза. Такая адаптация не только повышает комфорт пользователя, но и увеличивает время автономной работы устройства, поскольку темные темы могут потреблять меньше энергии на экранах OLED и AMOLED. Кроме того, обе темы отвечают требованиям доступности, помогая людям с нарушениями зрения

или чувствительностью, которым может быть легче работать с одной темой, чем с другой. Предоставляя возможность переключаться между светлой и темной темами, приложения могут предложить более персонализированный опыт, удовлетворяя индивидуальные эстетические предпочтения и повышая общую удовлетворенность пользователей. Такая гибкость становится стандартным требованием при разработке пользовательских интерфейсов, отражая приверженность принципам инклюзивности и дизайна, ориентированного на пользователя.

Кроме того, принципом дизайна является использование эффектов мерцания вместо традиционных загрузчиков. В современных приложениях это улучшает user experience, обеспечивая более интересную и менее раздражающую визуализацию во время загрузки контента. Эффект мерцания имитирует отражение света от поверхностей, анимируя места, где в конечном итоге появится контент. Такой подход создает у пользователей ощущение активности и прогресса без повторяемости или резкости, которые часто ассоциируются со стандартными индикаторами загрузки, такими как вращающиеся круги или индикаторы прогресса. Использование эффектов мерцания помогает сохранить эстетическую целостность приложения, поскольку они плавно вписываются в общий дизайн, уменьшая визуальное дрожание и делая пользовательский интерфейс элегантным и плавным. Кроме того, указывая на место загрузки контента, такие эффекты помогают управлять ожиданиями пользователей, делая ожидание более коротким и менее неопределенным.

2.6.3 Проект пользовательского интерфейса

На основе вышесказанного этого был разработан дизайн пользовательского интерфейса приложения. На рисунке 15 представлен дизайн главной страницы приложения. Помимо показа списка кофемашин и их статусов в дизайне предусмотрена возможность разрешить доступ к уведомлениям, что требуется на всех версиях iOs и на версиях Android, начиная с Android 6.0 (Marshmallow, API level 23) [33].

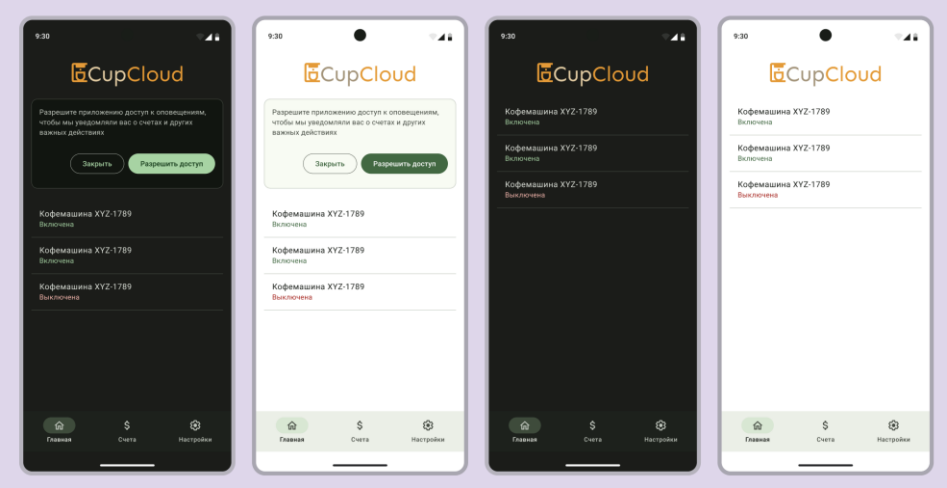

Рисунок 15 – Дизайн главной страницы приложения

На рисунке 16 представлен дизайн для показа неоплаченных счетов. Помимо различных счетов, существует возможность оплатить все счета одновременно.

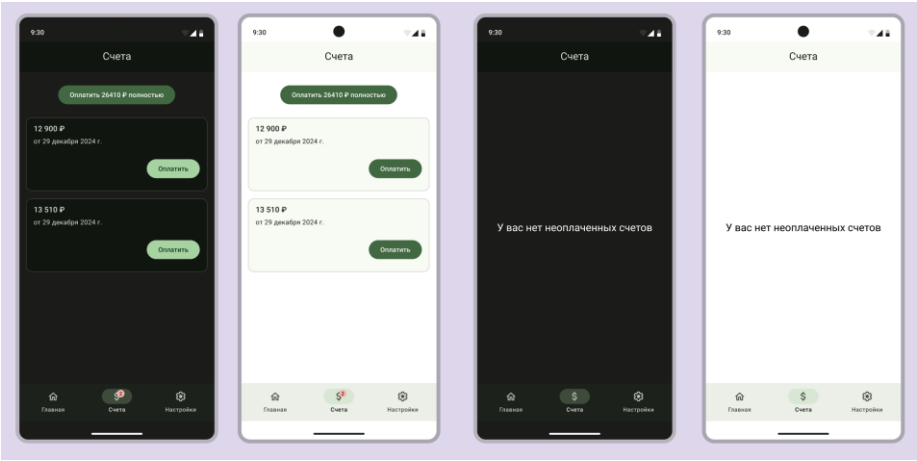

Рисунок 16 – Дизайн страницы «Счета»

Кроме того, в компоненте Bottom Navigation Bar отображается счетчик, позволяющий видеть количество неоплаченных счетов.

Реализация дизайна для сценария использования с оплатой счетов представлена на рисунке 17.

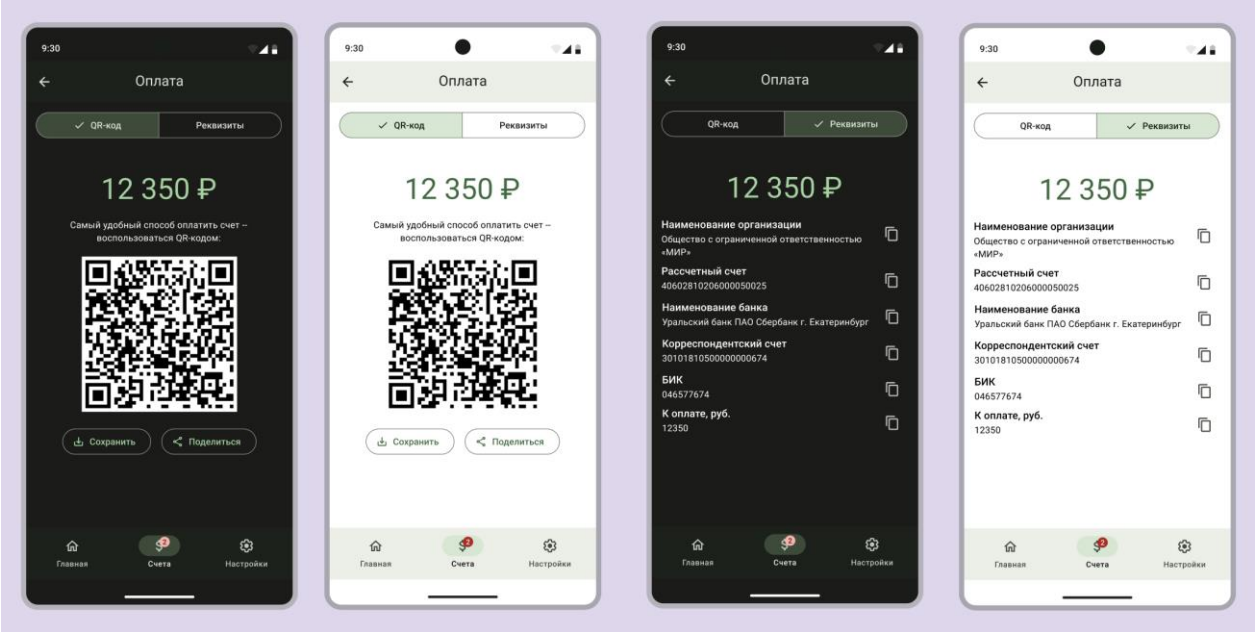

Рисунок 17 – Дизайн страницы оплаты счета

Для удобства использования, помимо прочего, представлена возможность сохранения QR-кода в галерею и кнопка «Поделиться», позволяющая открыть изображение с помощью других приложений, например, мессенджера, для передачи QR-кода. Все реквизиты можно скопировать в буфер обмена устройства, что так же необходимо для удобства использования приложения.

На рисунке 18 показан дизайн страницы «Настройки». На этом экране отображается основная информация о пользователе (контрагенте), предусмотрен выбор цветовой схемы и настройка оповещений. По требованию магазинов приложений нужно предусмотреть возможность удаления аккаунта, что тоже было предусмотрено в дизайне [7][35]. Настройка уведомлений привязана к устройству, в то время как настройка оповещений по электронной почте – к аккаунту.

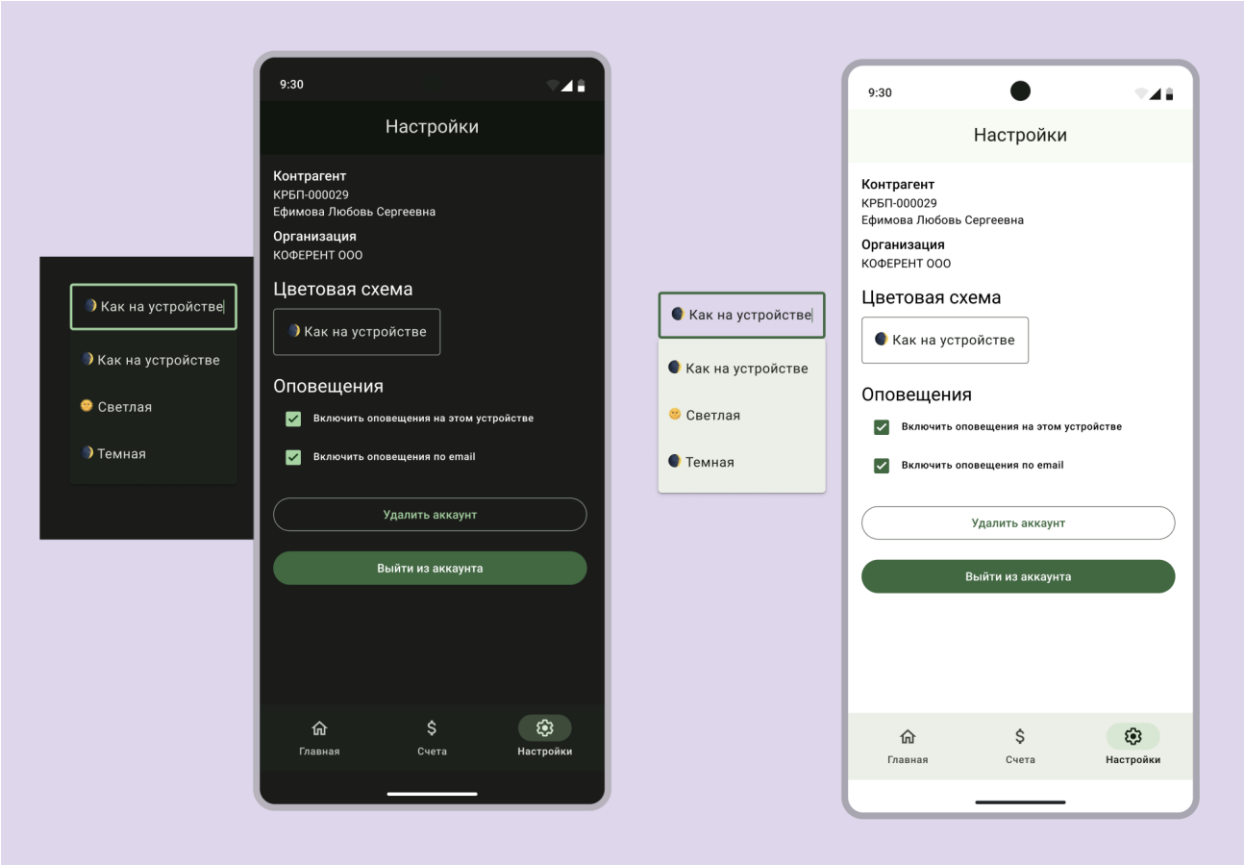

Рисунок 18 – Дизайн страницы «Настройки»

За основу дизайна интерфейса взят Material Design 3, который предусматривает задачу цветовой палитры. Основные цвета палитры представлены в таблице 11.

Таблица 11 – Цветовая палитра приложения

Название	Светлая палитра	Темная палитра
primary	#68548E	#D3BCFD
surfaceTint	#68548E	#D3BCFD
onPrimary	#FFFFFF	#38265C
primaryContainer	#EBDDFF	#4F3D74
onPrimaryContainer	#230F46	#EBDDFF
secondary	#635B70	#CDC2DB
onSecondary	#FFFFFF	#342D40

Продолжение Таблицы 11

secondaryContainer	#E9DEF8	#4B4358
onSecondaryContainer	#1F182B	#E9DEF8
tertiary	#7E525D	#F0B7C5
onTertiary	#FFFFFF	#4A2530
tertiaryContainer	#FFD9E1	#643B46
onTertiaryContainer	#31101B	#FFD9E1
error	#BA1A1A	#FFB4AB
onError	#BA1A1A	#690005
errorContainer	#FFDAD6	#93000A
onErrorContainer	#410002	#FFDAD6
background	#FEF7FF	#151218
onBackground	#1D1B20	#E7E0E8
surface	#FEF7FF	#151218
onSurface	#1D1B20	#E7E0E8
surfaceContainer	#F2ECF4	#211F24

При этом в Android 12 появились динамические цвета, позволяющие использовать для интерфейса цветовую палитру системы [13]. Это тоже следует реализовать для того, чтобы интерфейс приложения выглядел максимально естественно.

Выводы по главе 2

В главе 2 подробно описаны сценарии использования приложения, такие как вход пользователя в систему, администрирование, регистрация и восстановление доступа, с указанием шагов и решений в рамках каждого процесса.

В дизайне приложения использованы принципы Material Design для пользовательского интерфейса, предлагающие простую, но эффективную эстетику, которая хорошо реагирует на различные устройства и разрешения

экранов. Выбор светлой и темной тем позволяет учитывать предпочтения пользователей в различных условиях освещения и оптимизировать расход заряда батареи на устройствах с OLED-экранами.

С точки зрения архитектуры предлагается клиент-серверная модель, использующая «чистую архитектуру» для четкого разделения задач. Такая архитектура помогает создать обслуживаемую и масштабируемую систему, в которой бизнес-логика не зависит от пользовательского интерфейса. REST API и JSON выбраны за их простоту и эффективность при взаимодействии клиент-сервер, а токены JWT используются для безопасной и масштабируемой авторизации на стороне сервера. Общий подход обеспечивает баланс между дизайном пользовательского интерфейса и технической архитектурой.

Глава 3 Физическое моделирование приложения для аренды кофемашин

3.1 Выбор технологий для разработки клиентской части

Для разработки приложения был выбран Flutter – платформа для кроссплатформенной разработки приложений [14]. Если сравнивать Flutter с другими технологиями мобильной разработки, то Flutter отличается уникальным подходом к рендерингу пользовательского интерфейса и эффективностью разработки. В отличие от традиционной нативной разработки, требующей отдельных кодовых баз для iOS и Android, подход Flutter с единой кодовой базой позволяет значительно сократить время и ресурсы разработки.

По сравнению с такими кроссплатформенными фреймворками, как React Native или Xamarin, Flutter обеспечивает стабильную производительность 60 кадров в секунду благодаря прямой компиляции в нативный ARM-код и собственному движку рендеринга. Это означает, что производительность часто находится на одном уровне с нативными приложениями и не страдает от узких мест, которые могут возникать в других кроссплатформенных технологиях.

Еще одним отличительным преимуществом Flutter является функция горячей перезагрузки, которая способствует быстрому циклу разработки, позволяя разработчикам практически мгновенно видеть результаты своих изменений без потери состояния приложения [28]. Это менее заметно в нативных средах разработки, где время сборки может прервать процесс разработки.

Глубокая интеграция Flutter с Material Design – еще одна причина выбора Flutter. Flutter поддерживает Material Design и предлагает набор предварительно разработанных виджетов, которые соответствуют этим принципам «из коробки» [9].

Для разработки клиентской части приложения были выбраны следующие библиотеки:

- bloc – библиотека для стейт-менеджмента;
- dio – http-клиент с богатым функционалом;
- retrofit – библиотека, позволяющая создавать Rest-клиент без написания большого количества кода;
- getit – библиотека для Dependency Injection;
- freezed – библиотека, позволяющая генерировать методы для DTO-сущностей.

Следует заметить, что это основные библиотеки, используемые в разработке клиентской части, но так же используются и другие библиотеки.

3.2 Выбор технологий для разработки серверной части

В отличие от клиентской части, существует огромное количество языков программирования и технологий, позволяющих разработать Restful-сервис и без особых сложностей развернуть его на сервере или виртуальной машине. Такой сервис можно написать на многих языках, включая Java, php, Python, Dart, JavaScript (Node.js), C#, Ruby и многих других. Было решено использовать язык программирования Kotlin и фреймворк Ktor, позволяющий легко создавать полноценные Restful-серверы. Среди преимуществ выбора Kotlin можно выделить высокую производительность [22]. Также плюсами Kotlin являются современный и лаконичный синтаксис, а также наличие функционала, способствующего скорости и удобству разработки: null-safety, coroutines, data classes, extension methods и других.

Для создания сервиса был выбран фреймворк Ktor. Он разрабатывается и поддерживается компанией, создавшей Kotlin – JetBrains, имеет хорошую стабильность и высокую производительность. По сравнению в Java-фреймворками, такими, как Spring, которые тоже можно использовать в

Kotlin, Ktor написан непосредственно на Kotlin и имеет более удобную интеграцию в этот язык программирования [23].

Для асинхронных вызовов будут использоваться корутины, которые обладают рядом преимуществ по сравнению с обычными потоками Java – они очень производительны и легки в использовании.

3.3 Выбор СУБД

В качестве СУБД были рассмотрены различные варианты.

При выборе между документно-ориентированными и реляционными базами данных выбор был сделан в пользу первых, так как общая структура данных не имеет много реляционных связей, и, в то же время, может со временем меняться.

Среди документно-ориентированных баз данных выбор был остановлен на MongoDB. По сравнению с другими базами данных NoSQL, такими как Cassandra, которая предназначена для высокой пропускной способности и масштабируемости на нескольких узлах, MongoDB предлагает лучшую модель запросов и обширные вторичные индексы, что делает ее более подходящей для приложений, требующих сложных запросов и аналитики в реальном времени [30]. В отличие от хранилищ ключевых значений, таких как Redis, MongoDB позволяет хранить более сложные структуры данных и имеет более богатый набор операций запроса.

3.4 Выбор среды разработки

Разработка на Flutter предполагает использование одной из двух сред разработки: Android Studio и Visual Studio Code [36]. Сравнение сред разработки представлено в таблице 12.

Таблица 12 – Сравнение Android Studio и Visual Studio Code

Функция	Android Studio	Visual Studio Code
Поддержка Flutter	Полностью интегрирован	Поддерживается через расширения
Встроенный эмулятор	Встроенный эмулятор для Android	Нет встроенной поддержки, требуются внешние эмуляторы
Экосистема плагинов	Богатая экосистема плагинов, адаптированная для Android и Flutter	Обширная экосистема плагинов, управляемая сообществом, для различных технологий
Профилирование производительности	Передовые инструменты профилирования производительности для Flutter и Android	Основные сведения о производительности с помощью расширений
Пользовательский интерфейс	Сложная, многофункциональная IDE, предназначенная для больших проектов	Легкий и минималистичный дизайн
Интеграция с системой контроля версий	Сильная поддержка системы контроля версий, особенно Git	Хорошая поддержка системы контроля версий с акцентом на Git
Инструменты отладки	Обширный набор инструментов отладки, включая специфические виджеты Flutter для проверки	Надежные функции отладки с расширениями, менее обширные, чем в Android Studio

Исходя из этого сравнения, для разработки была выбрана среда разработки Android Studio.

Для серверной разработки на Kotlin была выбрана IntelliJ IDEA CE, на основе которой была разработана Android Studio. Помимо прочего, преимуществом этой IDE является наиболее полноценная поддержка языка Kotlin.

3.5 Развертывание серверной части приложения

Для того, чтобы серверная часть приложения работала, ее следует разместить и запустить на сервере или на виртуальной машине, и этот процесс называется «развертыванием». Концептуально, приложение может быть размещено на любом хостинге, что может быть достигнуто за счет

контейнеризации. Самым распространенным, универсальным и надежным решением для этого является Docker. Docker можно установить практически на любой сервер, после чего запустить в нем контейнер нашего приложения, написанного на Kotlin [12]. Также с помощью Docker будут решены и другие проблемы: во-первых, контейнер может быть запущен в режиме, позволяющем перезапускать контейнер в случае «вылета» приложения, что обеспечит стабильность работы. Во-вторых, с помощью Docker compose возможно запустить на том же сервере другие контейнеры, такие, как http-сервер nginx или MongoDb, что позволит не тратить много усилий на установку и настройку этих сервисов.

3.6 Система контроля версий и Continuous Integration

В ходе разработки будет использована система контроля версий Git, которая в современном мире повсеместно используется и практически не имеет альтернатив.

Для хостинга исходного кода были рассмотрены различные варианты, такие, как GitHub, GitLab, BitBucket или свой хостинг. В результате выбор был сделан в пользу GitHub. Частью этого сервиса является уже встроенная в сервис система CI/CD, называемая GitHub Actions, которая позволяет легко собирать и развертывать приложения. Это достигается за счет простого размещения файлов yaml в проекте, причем синтаксис легок для понимания и хорошо задокументирован [32]. Главным плюсом GitHub по сравнению с другими сервисами является то, что с GitHub Actions не нужно дополнительно настраивать.

3.7 Разработка клиентской части приложения

Клиентская часть приложения разработана на Flutter, а в качестве среды разработки используется Android Studio.

Разработка клиента начинается с добавления основных библиотек, которые используются при разработке, в файл pubspec.yaml [34]. В разделе dependencies, как показано на рисунке 19, указываются те зависимости, которые необходимы приложению во время выполнения. То есть, все библиотеки и пакеты, которые используются в вашем приложении напрямую и требуются для его работы, должны быть включены сюда.

```
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  dio: ^5.3.3
  retrofit: ^4.0.3
  freezed_annotation: ^2.4.1
  flutter_bloc: ^8.1.3
  get_it: ^7.6.4
  injectable: ^2.3.2
  logger: ^2.0.2+1
  shimmer: ^3.0.0
  routemaster: ^1.0.1
  rxdart: ^0.27.7
  json_annotation: ^4.8.1
  easy_localization: ^3.0.3
  awesome_dio_interceptor: ^1.0.0
  easy_localization_loader: ^2.0.0
  email_validator: ^2.1.17
  flutter_secure_storage: ^9.0.0
  checkbox_formfield: ^0.2.0
```

Рисунок 19 – Зависимости клиентской части приложения

- dio – мощный HTTP-клиент для Dart, поддерживающий перехватчики, глобальную конфигурацию, FormData, отмену запросов, загрузку файлов, таймаут и т.д.;
- retrofit – безопасный для типов HTTP-клиент для Dart и Flutter, вдохновленный Retrofit для Android. Он превращает ваш HTTP API в интерфейс Dart;
- freezed_annotation – предоставляет аннотации для пакета Freezed, который помогает генерировать неизменяемые классы и высокопроизводительные union-типы в Dart;
- flutter_bloc – библиотека управления предсказуемыми состояниями, которая помогает реализовать паттерн проектирования BLoC (Business Logic Component). Она построена на основе потоков и реактивного программирования;
- get_it – простой локатор сервисов для проектов Dart и Flutter. С его помощью можно получить доступ к объектам без отсоединения;
- injectable – автоматический генератор инъекций зависимостей для Dart и Flutter, основанный на генерации кода, который хорошо работает с GetIt;
- routemaster – гибкий и простой в использовании пакет маршрутизации для Flutter, поддерживающий глубокую перелинковку, вложенную навигацию и множество других возможностей навигации;
- json_annotation – предоставляет аннотации для использования с json_serializable для автоматической сериализации объектов из JSON/в JSON в Dart.

Инициализация зависимостей была помещена в файл dependencies.dart, фрагмент которого показан на рисунке 20. Здесь приведен пример зависимости dio.

Рисунок 20 – Инициализация зависимостей

При инициализации создается объект Dio, при этом ему задаются базовые опции и добавляются необходимые интерсепторы. Интерсептор ResponseInterceptor служит для правильной обработки ответов от сервера, например – преобразование ответов с http-кодом, отличным от 200, в объект ApiFailure. Объект ApiFailure содержит свойство, содержащее объект FailureCode, который исопльзуется для отображения ошибок в интерфейсе пользователя. Содержание класса показано на рисунке 21.

.

Рисунок 21 – Коды ошибок

При старте приложения, как показано на рисунке 22, инициализируются все необходимые зависимости, после чего запускается само приложение.

```
void main() async {
  usePathUrlStrategy();
  configureDependencies();
  WidgetsFlutterBinding.ensureInitialized();
  await EasyLocalization.ensureInitialized();
  runApp(const MyApp());
}
```

Рисунок 22 – Функция main

В файле router.dart содержатся данные, отвечающие за перемещение пользователя по различным экранам с помощью библиотеки routemaster. Фрагмент кода приведен на рисунке 23.

```dart
final routes = RouteMap(
  onUnknownRoute: (_) => const Redirect('/'),
  routes: {
    '/': (_) => const MaterialPage(child: HomeScreen()),
  },
);  // RouteMap

final unauthorisedRoutes = RouteMap(
  onUnknownRoute: (_) => const Redirect('/login'),
  routes: {
    '/login': (_) => const MaterialPage(child: LoginMobileScreen()),
    '/signup': (_) => const MaterialPage(child: SignUpMobileScreen()),
    '/resetPassword': (data) => MaterialPage(child: ResetPasswordMobileScreen(code: data.queryParameters['code'])),
  },
);  // RouteMap
```

Рисунок 23 – Маршруты в приложении

Есть два дерева маршрутов – один для авторизированного пользователя, второй – для неавторизированного. Они переключаются в зависимости от состояния с помощью библиотеки Bloc в main.dart.

Цветовые схемы находятся в файле themes.dart.

В корень проекта также был добавлен файл codegen.sh, в котором содержатся команды, отвечающие за кодогенерацию. Содержимое файла можно увидеть на рисунке 24.

```
flutter pub run build_runner build --delete-conflicting-outputs
flutter pub run easy_localization:generate --source-dir assets/translations -f keys -o locale_keys.g.dart
flutter pub run easy_localization:generate --source-dir assets/translations/
```

Рисунок 24 – Содержимое файла codegen.sh

Это сделано для удобства – теперь всю кодогенерацию можно запустить с помощью команды sh codegen.sh, а не вводить все команды в терминал последовательно каждый раз, когда нужна кодогенерация.

Проект разделен на папки, содержащие различный функционал. Например, на рисунке 25 показана структура папки login.

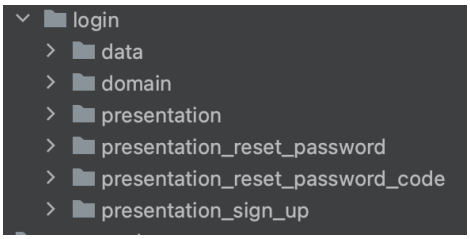

Рисунок 25 – Структура папки login

В соответствии с принципами чистой архитектуры, функционал разделен на пакеты data, domain и presentation [2]. Папки data и presentation являются внешними слоями в чистой архитектуре, а папка domain – внутренний слой, в котором находится бизнес-логика и сущности. Папок, начинающихся с presentation, несколько, но они используют схожую бизнес-логику.

В папке data, рисунок 26, находятся источники данных, соответствующие им сущности и репозиторий.

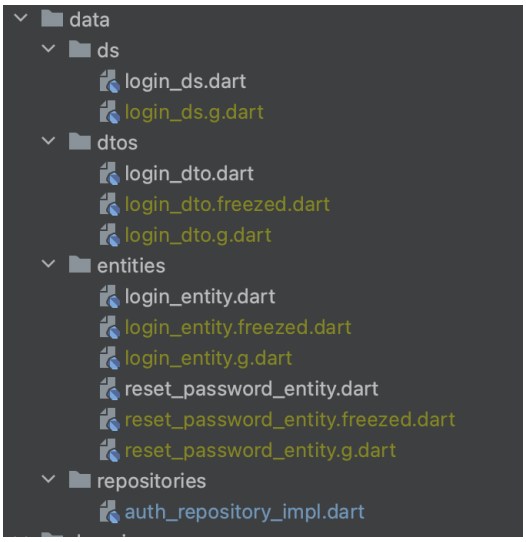

Рисунок 26 – Содержимое папки data

Источник данных (data source), в данном случае LoginDataSource, представляет из себя интерфейс, реализующий методы RestAPI, связанные с данным функционалом. Этот интерфейс показан на рисунке 27.

```dart
import 'package:dio/dio.dart';
import 'package:injectable/injectable.dart';
import 'package:retrofit/retrofit.dart';
import 'package:reviewmagic_flutter/features/login/data/dtos/login_dto.dart';
import 'package:reviewmagic_flutter/features/login/data/entities/login_entity.dart';
import 'package:reviewmagic_flutter/features/login/data/entities/reset_password_entity.dart';

part 'login_ds.g.dart';

@RestApi()
@LazySingleton()
abstract class LoginDataSource {
  @factoryMethod
  factory LoginDataSource(Dio dio) = _LoginDataSource;

  @POST('/auth/login')
  Future<LoginEntity> login(@Body() LoginDto loginDto);

  @POST('/auth/register')
  Future<LoginEntity> register(@Body() LoginDto loginDto);

  @POST('/auth/reset')
  Future<bool> sendCodeToEmail(@Body() String email);

  @PUT('/auth/reset')
  Future<LoginEntity> resetPasswordByCode(@Body() ResetPasswordEntity entity);
}
```

Рисунок 27 – Класс LoginDataSource

С помощью библиотеки retrofit и кодогенерации этот интерфейс реализуется.

В папке dtos содержатся dto-классы, которыми манипулирует данный слой. Например, LoginDto, как показано на рисунке 28, содержит класс, отвечающий за авторизацию и содержащий одно поле token, в котором будет находиться JWT-токен, полученный с сервера.

Рисунок 28 – Класс LoginDto

Аналогичные объекты содержатся и в папке entities, только они отвечают за входящие параметры запросов к API. Они были названы entities и выделены отдельно для различия.

В папке repositories находятся репозитории. В данном случае там находится AuthRepositoryImpl, который содержит методы для обращения к данным. Он реализует интерфейс AuthRepository, показанный на рисунке 29, который находится на уровне бизнес-логики. Это сделано в соответствии с правилом зависимостей из чистой архитектуры, а также в соответствии с SOLID-принципом Dependency Inversion.

Рисунок 29 – Интерфейс AuthRepository

В данном случае свойство authState – это стрим, отвечающий за состояние пользователя «авторизирован» и «не авторизирован». Подписавшись на него, мы можем получать это состояние и менять что-либо в зависимости от него.

В папке domain, как показано на рисунке 30, находится интерфейс репозитория, а также сущность, отвечающая за бизнес-логику.

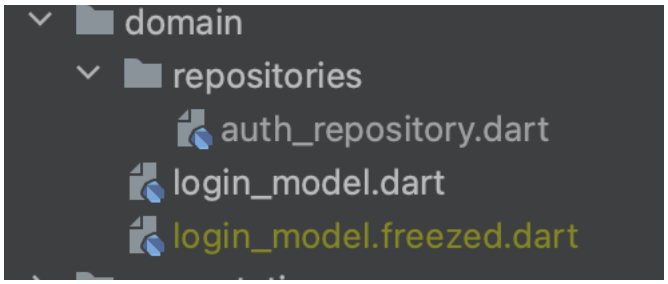

Рисунок 30 – Структура папки domain

Для упрощения в данном случае не был использован интерактор, который фактически бы просто делегировал свои вызовы репозиторию. Так как репозитории и сущности, отвечающие за состояние, практически находятся в одном слое, это соответствует чистой архитектуре.

В папке presentation находится все, что относится к представлению. Это показано на рисунке 31.

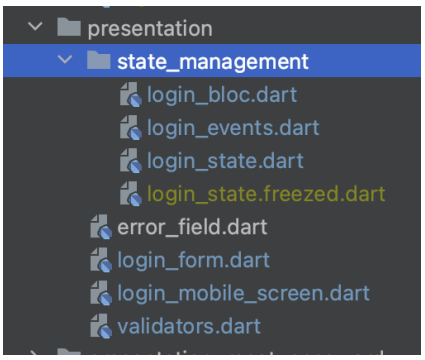

Рисунок 31 – Структура папки presentation

К представлению относится и State Management, для которого используется библиотека bloc [38].

Классы состояния, например, показанный на рисунке 32 класс LoginState, содержат непосредственно состояние.

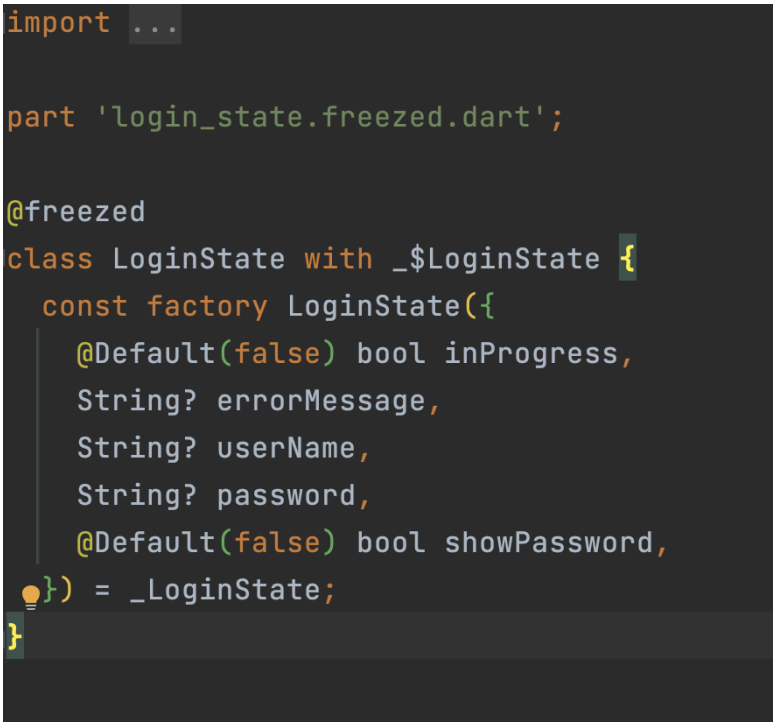

Рисунок 32 – Класс LoginState

Свойства этого класса типичны для большинства подобных классов. Свойство inProgress содержит значение, которое означает, загружаются ли в данный момент данные или нет. Если значение свойства true, в интерфейсе следует отображать, что данные загружаются, например, с помощью эффектов мерцания. Свойство errorMessage отвечает за отображение ошибки: если оно не равно null, то в интерфейсе должна отображаться соответствующая ошибка. Остальные поля – showPassword, userName и password содержат данные, которые пользователь ввел в интерфейсе.

В файле с суффиксом _events находятся события, которые передаются из интерфейса в блок. События LoginEvent находятся в файле login_events.dart, содержимое которого показано на рисунке 33.

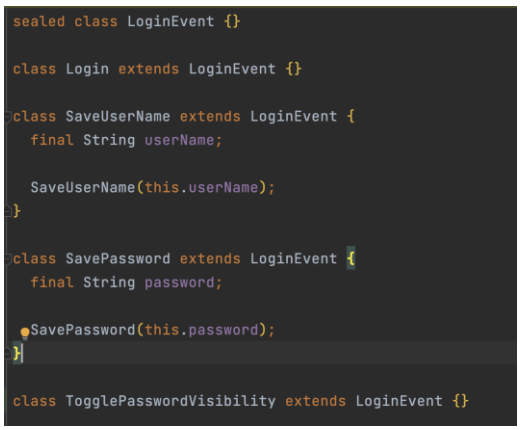

Рисунок 33 – Содержимое файла login_events.dart

В данном случае, например, событие SaveUserName вызывается, когда пользователь ввел имя пользователя, а событие Login – когда пользователь нажал на кнопку «Войти».

Управление состоянием осуществляется из класса Bloc, в данном случае – LoginBloc, показанный на рисунке 34.

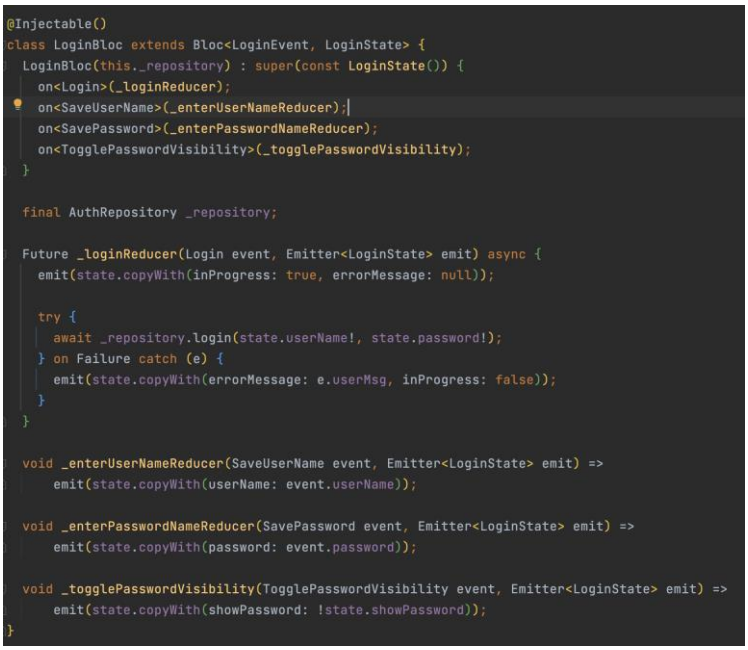

Рисунок 34 – Класс LoginBloc

При инициализации класса происходит подписка на события, и затем объект реагирует на эти события. В данном случае, при вводе имени пользователя или пароля, а также при переключении видимости пароля, просто происходит изменение свойств состояния. При событии Login происходит вызов метода login из репозитория, при этом состояние изменяется в зависимости от того, выполняется ли запрос и как он завершился.

Также в папке presentation расположены виджеты, которые меняются в зависимости от состояния. Рассмотрим класс LoginForm, представляющий из себя форму авторизации. Этот класс представляет из себя StatelessWidget, а за состояние отвечают виджеты BlocProvider и BlocBuilder, что можно увидеть на рисунке 35.

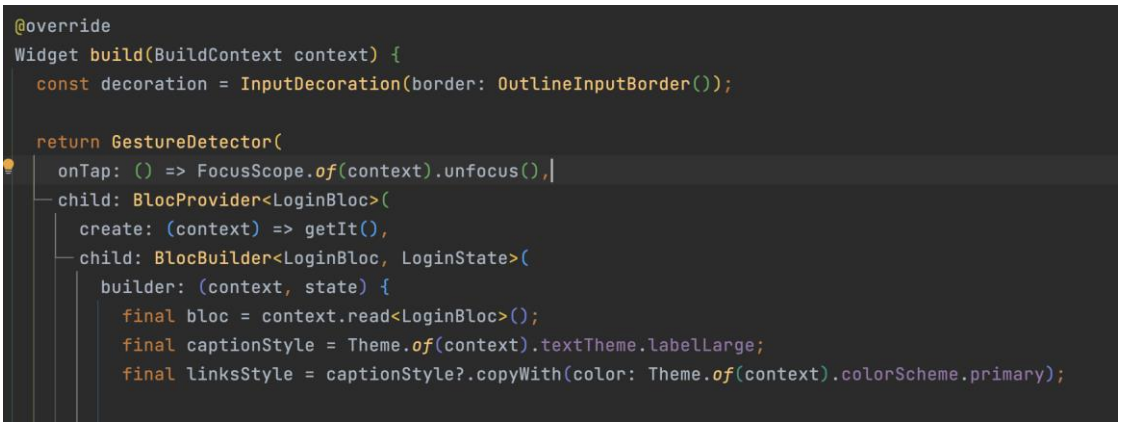

Рисунок 35 – Фрагмент виджета LoginForm

BlocProvider отвечает за создание блока, в данном случае для этого используется фунционал get_it и injectable: вызывается функция getIt(), при этом создается объект LoginBloc, класс которого помечен аннотацией @Injectable().

Рассмотрим одно из полей формы – например, поле для ввода пароля, рисунок 36.

```dart
TextFormField(
  initialValue: kDebugMode ? '123' : null,
  decoration: decoration.copyWith(
    labelText: LocaleKeys.login_passwordLabel.tr(),
    suffixIcon: IconButton(
      onPressed: () => bloc.add(TogglePasswordVisibility()),
      icon: Icon(state.showPassword ? Icons.visibility_off_outlined : Icons.visibility_outlined),
    ),  // IconButton
  ),
  obscureText: !state.showPassword,
  validator: passwordValidator(context),
  onSaved: (newValue) => bloc.add(SavePassword(newValue!)),
  textInputAction: TextInputAction.send,
  autofillHints: const [AutofillHints.password],
  keyboardType: TextInputType.visiblePassword,
  onFieldSubmitted: (value) => _submitForm(context),
),  // TextFormField
```

Рисунок 36 – Код поля для ввода пароля

При сохранении формы вызывается onSaved, и в bloc передается событие SavePassword, в котором содержится введенное значение. Также это текстовое поле содержит кнопку, позволяющую скрыть или показать введенный пароль. По нажатию на эту кнопку в bloc отправляется событие TogglePasswordVisibility(), а свойство obscuretext, отвечающее за скрытие текста, зависит от значения состояния showPassword. Так как это поле является последним в форме, свойству textInputAction присвоено значение TextInputAction.send, что влияет на отображение экранной клавиатуры на устройстве, что показано на рисунке 37.

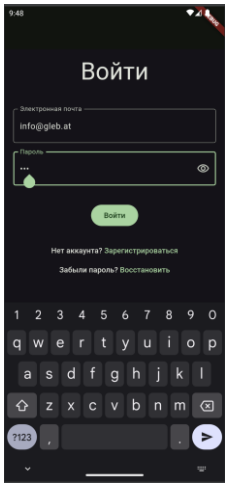

Рисунок 37 – Поле ввода пароля с открытой клавиатурой

Поэтому по нажатию на кнопку «Отправить» на клавиатуре происходит вызов onFieldSubmited и вызов метода _submitForm() – то же самое происходит и по нажатию на кнопку «Войти».

Метод _submitForm(), который можно посмотреть на рисунке 38, сначала вызывает метод, отвечающий за валидацию формы – то есть проверку всех введенных данных.

```
void _submitForm(BuildContext context) {
  final bloc = BlocProvider.of<LoginBloc>(context);
  if (_formKey.currentState!.validate()) {
    FocusScope.of(context).unfocus();
    _formKey.currentState!.save();
    bloc.add(Login());
  }
}
```

Рисунок 38 – Метод _submitForm()

В случае, если все данные введены пользователем корректно, вызывается метод формы save(), при вызове которого последовательно вызываются методы onSaved() у всех полей формы. Затем в bloc отправляется событие Login(), реакция блока на которое была рассмотрена выше.

Аналогично авторизации разработан и весь другой функционал приложения, включая разделение по папкам и управление состоянием.

3.8 Разработка серверной части приложения

Серверная часть приложения разработана на языке программирования Kotlin, за основу взят фреймворк Ktor, а в качестве среды разработки используется IntelliJ IDEA CE.

Настройки среды окружения задаются при в файле application.yaml [25], содержимот которого можно посмотреть на рисунке 39.

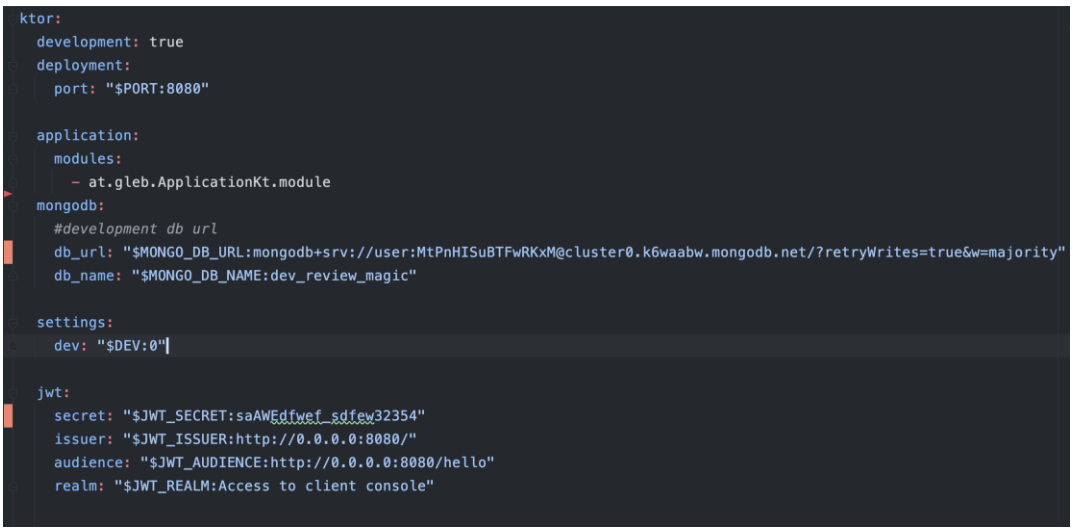

Рисунок 39 – Содержимое файла application.yaml

Здесь в первую очередь заданы настройки по умолчанию. В случае, если среда, в которой запущено приложение, содержит переменные, то эти значения будут заменены на них. Значения включают в себя используемый приложением порт, настройки доступа к MongoDB, настройки генерации JWT-токенов и некоторые другие настройки.

Точкой входа в приложение является функция main(), показанная на рисунке 40.

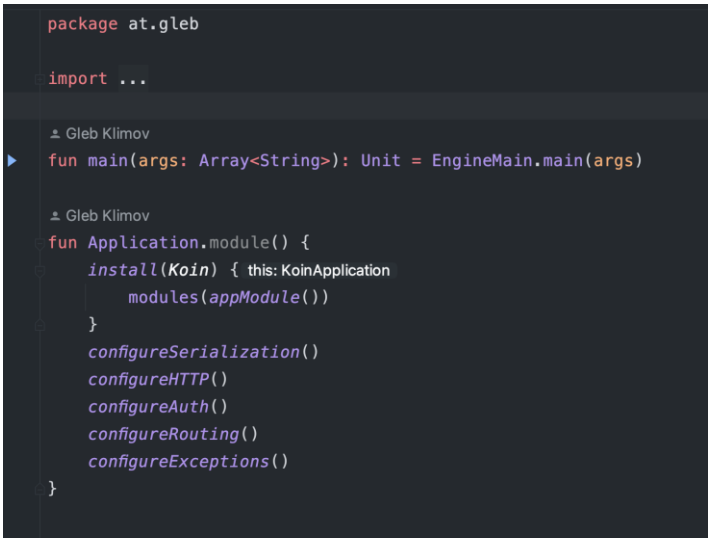

Рисунок 40 – Точка входа в приложение

При запуске приложения происходит ряд инициализаций.

Первая инициализация – это инициализация Koin. Koin – библиотека для инъекции зависимостей (Dependency Injection), позволяющая предоставлять зависимости разным модулям.

На рисунке 41 показан фрагмент модуля appModule, предоставляющего основные зависимости для приложения. В данном случае создаются объекты MongoClient, MongoDatabase, Cols – объект, содержащий коллекции MongoDb. и UserDataSource – источник данных, связанных с пользователями приложения. Аналогичным образом происходят и другие инициализации. В данном случае приложение использует только один модуль.

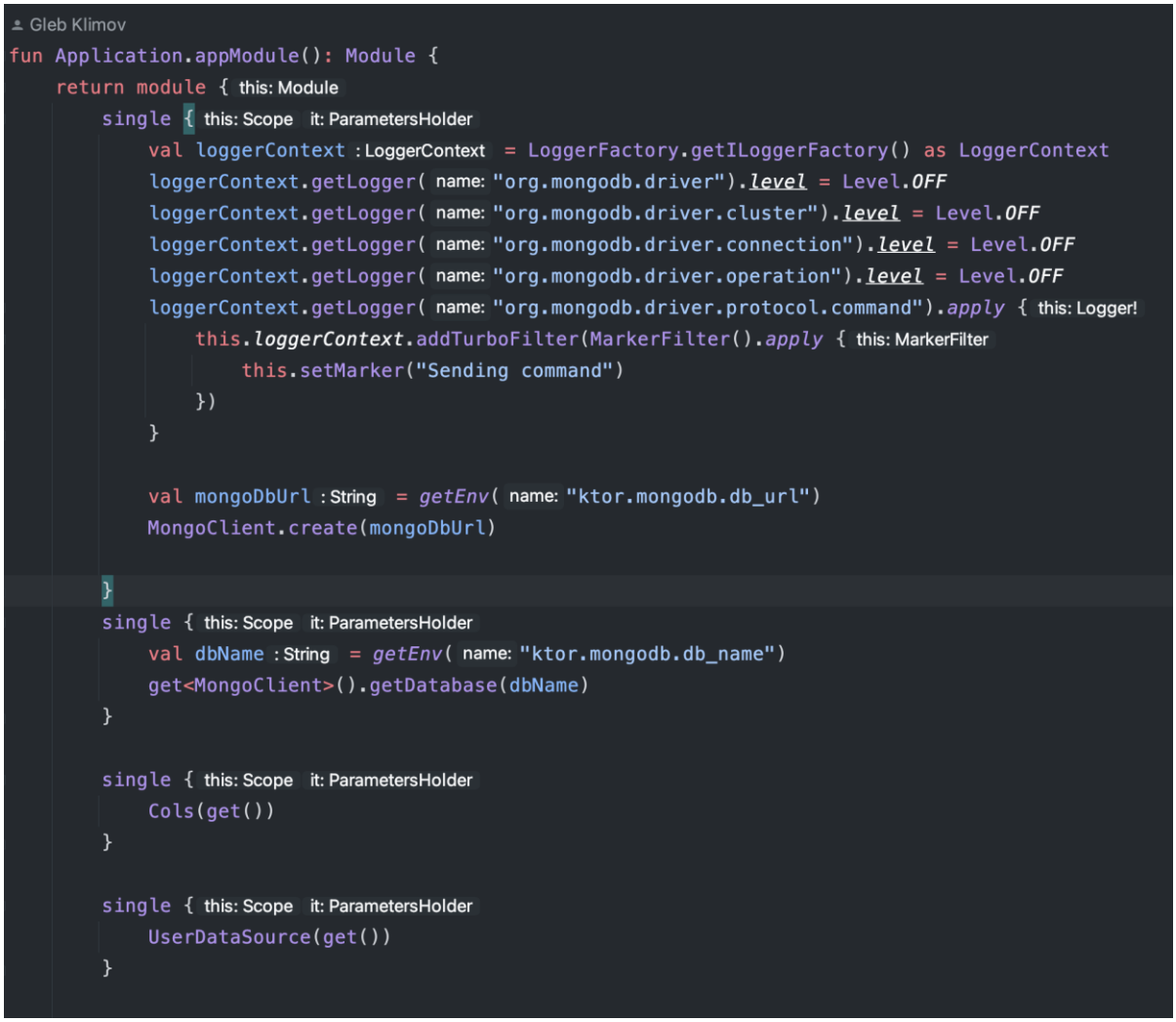

Рисунок 41 – Модуль зависимостей

Вызов configureSerialization() инициализирует библиотеку GSON, которая используется для преобразования объектов в Json и обратно, что показано на рисунке 42.

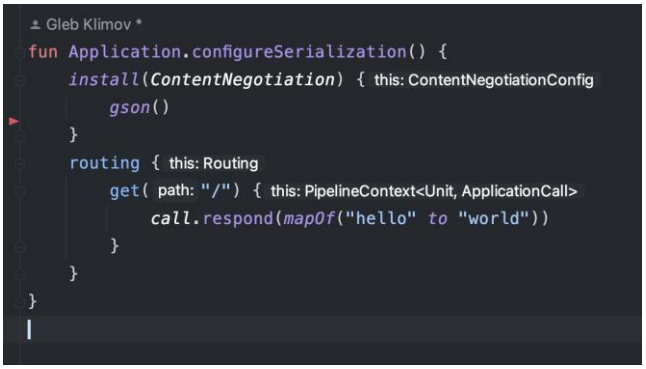

Рисунок 42 – Инициализация библиотеки GSON

Сериализация в Json будет происходить автоматически с помощью плагина ContentNegotiation. Так, например, при вызове запроса GET к корневому адресу будет возвращен ответ {"hello":"world"}.

Вызов configureHTTP() инициализирует плагины DefaultHeaders и CORS, что показано на рисунке 43.

.

Рисунок 43 – Инициализация плагинов DefaultHeaders и CORS

Они позволяют настроить заголовки, посылаемые сервером, и указать методы и заголовки, которые сервер может принимать.

Метод configureExceptions() инициализирует работу с исключениями, как показано на рисунке 44.

Рисунок 44 – Инициализация работы с исключениями

В некоторых ситуациях приложение может выкидывать исключения MyException, показанный на рисунке 45.

Рисунок 45 – Класс MyException

Эти исключения содержат информацию об ошибке, главным образом – в форме кода ошибки. Например, в случае неправильного ввода пароля будет проброшено исключение с кодом «wrong_password», и клиент в этом случае может показать пользователю соответствующее сообщение. Описываемый метод configureExceptions() конфигурирует Ktor таким образом, чтобы ошибка возвращалась в виде Json. В случае, если исключение не является экземпляром MyException, будет отдан ответ с кодом 500 и кодом «unknown».

Это основные методы, конфигурирующие KTor для работы приложения.

В методе main() также конфигурируется роутинг для функциональности приложения. В качестве примера рассмотрим вызов «configureAuth», который инициализирует функциональность, связанную с авторизацией пользователя. Пакет at.gleb.cupcloud.auth содержит всю эту функциональность, его содержимое показано на рисунке 46.

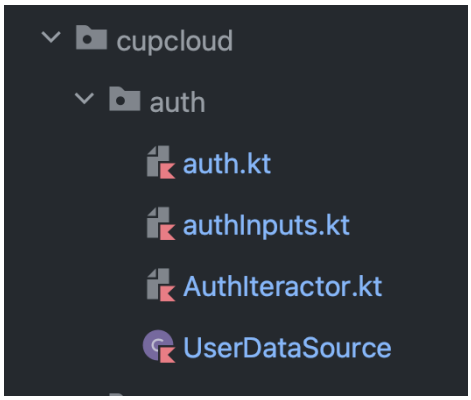

Рисунок 46 – Содержимое пакета auth

Файл auth.kt, находящийся внутри этого пакета, содержит функционал, связанный с созданием JWT-токенов, а также маршрутизацию, связанную с авторизацией. Файл authInputs.kt содержит объекты, в которые преобразуются входящие запросы. Класс AuthInteractor содержит бизнес-логику, связанную с авторизацией. Класс UserDataSource представляет из

себя источник данных, который взаимодействует с базой данных. Все эти классы в соответствии с чистой архитектурой должны находиться на разных слоях, однако в данном случае объединены в один пакет в целях упрощения. Практической необходимости в разделении их по разным пакетам нет, а сами классы довольно простые. При необходимости их легко можно будет перенести в разные пакеты, но в этой ситуации перенос их по разным пакетам усложнит читабельность и время разработки.

Рассмотрим в первую очередь содержание файла auth.kt, которое показано на рисунке 47.

Рисунок 47 – Фрагмент содержимого файла auth.kt

Во фрагменте на рисунке происходит инициализация верификации JWT-токена. Здесь можно увидеть, что в него кодируется id пользователя. В случае, если токен недействителен, сервером будет возвращаться ошибка 401, «Unauthorized». На рисунке 48 показано, каким образом можно защитить запрос токеном при указании путей.

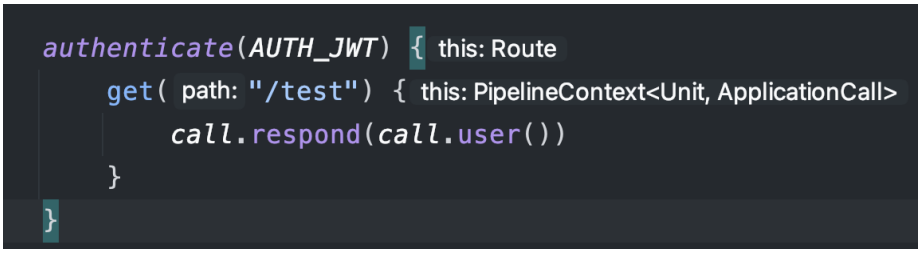

Рисунок 48 – Защита запроса токеном

В этом же файле auth.kt задается маршрутизация, связанная с авторизацией пользователей, что показано на рисунке 49

.

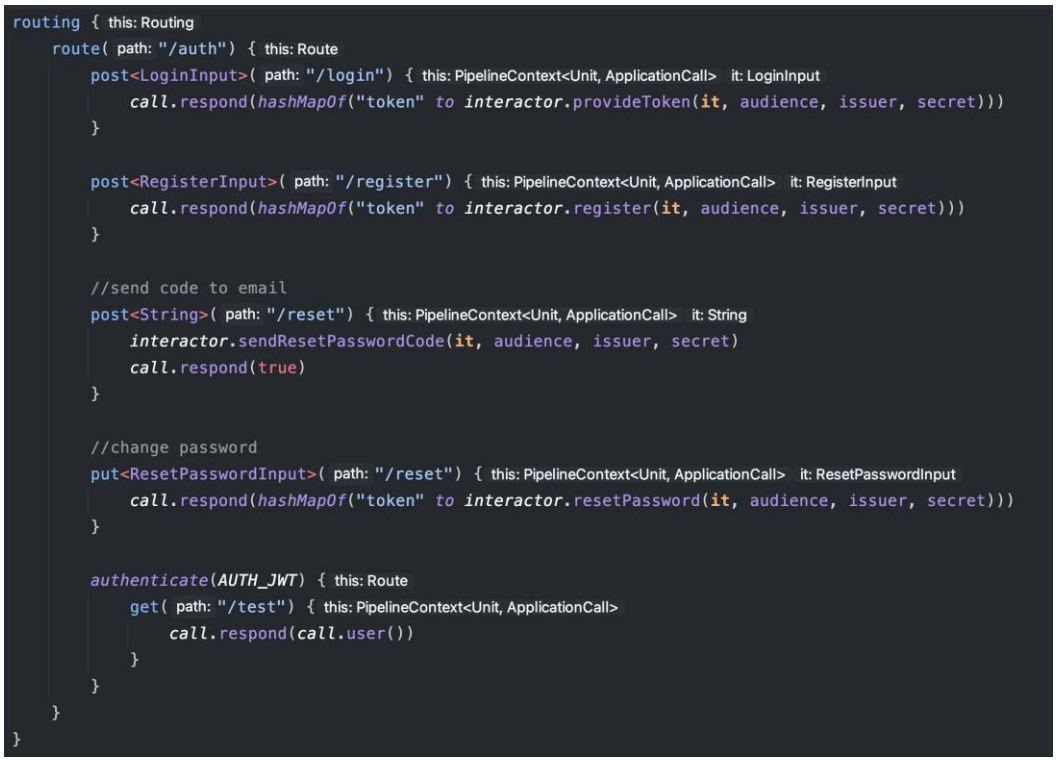

Рисунок 49 – Маршруты авторизации

Здесь задаются методы POST /auth/login, POST /auth/register, POST /auth/reset, PUT /auth/reset. При вызове этих методов происходит взаимодействие с интерактором, расположенным в файле AuthInteractor.kt. При взаимодействии передаются параметры из запроса, которые

преобразованы в один из объектов из файла authInput.kt. Сами объекты передаются с клиентской стороны в виде JSON в теле запроса, они находятся в поле «it». Они могут быть и просто строками, в этом случае нет необходимости преобразовывать их в объект. Такие параметры, как audience, issuer и secret, берутся их переменных среды. В данном случае эти три объекта нужны для создания JWT-токена.

Класс AuthInteractor зависит от UserDataSource и взаимодействует с ним, как показано на рисунке 50.

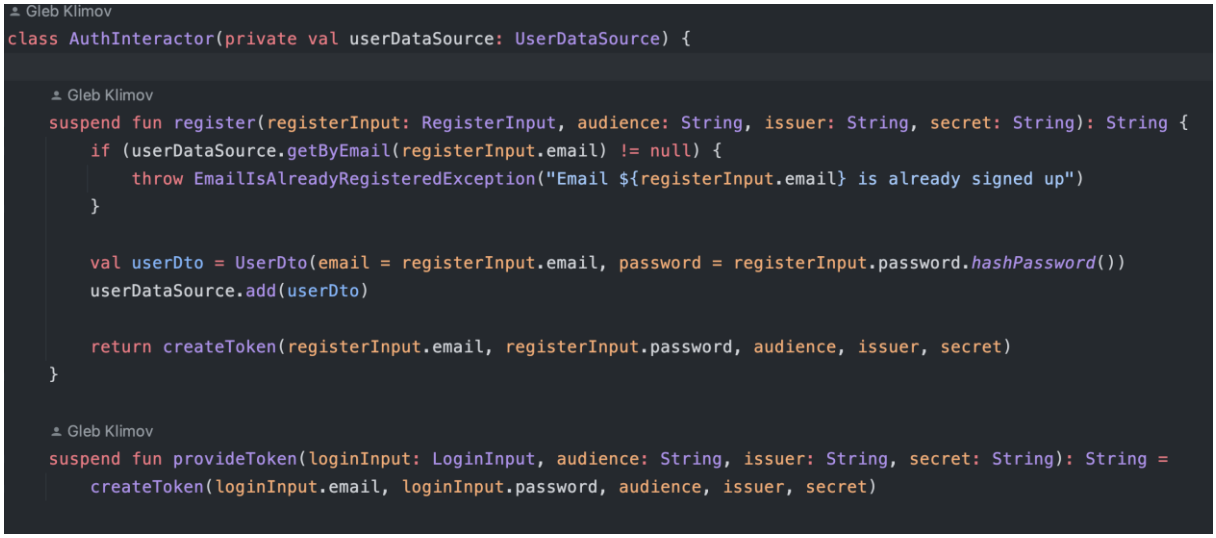

Рисунок 50 – Метод регистрации пользователя в классе AuthInteractor

Для примера рассмотрим, как работает метод регистрации пользователя.

В первую очередь происходит поиск пользователя в базе данных с помощью источника данных userDataSource. Если пользователь был найден, то выкидывается исключение EmailIsAlreadyRegisteredException, являющееся наследником от MyException, описанного выше, класс показан на рисунке 51.

```
class EmailIsAlreadyRegisteredException(message: String) :
    MyException(message, ErrorCodes.EMAIL_ALREADY_REGISTERED, HttpStatusCode.Unauthorized)
```

Рисунок 51 – Класс EmailIsAlreadyRegisteredException

Если же пользователь не найден, то происходит непосредственно регистрация – создается новый пользователь с полями, содержащими email и хеш от пароля, который добавляется затем в базу данных с помощью источника данных. Если все прошло успешно, то затем создается и возвращается JWT-токен для этого пользователя, как показано на рисунке 52.

```
private suspend fun createToken(
    email: String,
    password: String,
    audience: String,
    issuer: String,
    secret: String
): String {
    val userFromDb :UserDto  = userDataSource.getByEmail(email = email) ?: throw WrongCredentialsException()

    if (userFromDb.password != password.hashPassword()) throw WrongCredentialsException()

    return JWT.create()
        .withAudience(audience)
        .withIssuer(issuer)
        .withClaim( name: "id", userFromDb.id!!.toString())
        .withClaim( name: "email", userFromDb.email)
        .withExpiresAt(Date( date: System.currentTimeMillis() + ACCESS_TOKEN_EXPIRES_TIME_MILLS))
        .sign(Algorithm.HMAC256(secret))
}
```

Рисунок 52 – Создание JWT-токена

В JWT-токен закодированы id пользователя и email, а также задана дата истечения валидности токена. Закодированные надежно защищены и могут быть извлечены только на стороне сервера, процесс верификации токена был описан выше.

Источник данных UserDataSource, показанный на рисунке 53, зависит от Cols, который содержит коллекции, и отвечает за работу с базой данных MongoDB.

Рисунок 53 – Класс UserDataSource

Он содержит методы для поиска пользователя по id, для добавления пользователя, для поиска пользователя по email, а также содержит метод для установки нового пароля.

Мы рассмотрели в качестве примера реализацию API, связанного с авторизацией пользователя. Остальные методы Rest API реализованы аналогичным образом.

3.9 Покрытие основного функционала модульными тестами

Модульное тестирование представляет из себя написание тестов для различных функций, методов или классов. Этот вид тестов является самым производительным [4].

Основные классы приложения были покрыты модульными тестами. Для этого были использованы библиотеки mockito и bloc_test. Библиотека mockito позволяет создавать mock-объекты. Иными словами, она позволяет создавать зависимости, которые необходимы для тестирования того или

иного класса. Библиотека bloc_test упрощает написание модульных тестов для блоков [37].

Рассмотрим для примера покрытие Unit-тестами для класса AuthBloc, который расположен в файле test/auth_bloc_test.dart. Этот класс зависит от AuthRepository, поэтому был создан mock-объект, который реализует этот интерфейс, что показано на рисунке 54.

Рисунок 54 – Создание mock-объекта AuthRepository

С помощью фрагмента кода на рисунке генерируется файл auth_bloc_test.mocks.dart, содержащий реализацию интерфейса AuthRepository для целей тестирования.

Рассмотрим непосредственно сами тесты, покрывающие класс AuthBloc, показанные на рисунке 55.

Рисунок 55 – Модульные тесты для класса AuthBloc

На рисунке видно группу тестов, которые покрывают этот класс. Внутри метода setUp() создается mock-репозиторий, который используется в дальнейшем при создании объекта класса AuthBloc. Тесты покрывают различные случаи, тестируют реакцию блока на различные события. Так, например, последний тест в списке проверяет, что при отправке в блок события Logout() вызывается метод logOut() у репозитория.

Результаты запуска теста отображены на рисунке 56.

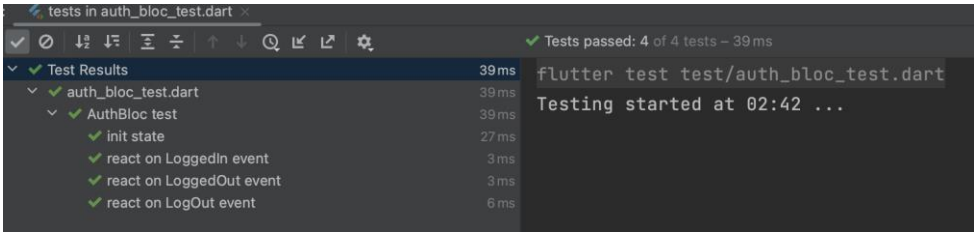

Рисунок 56 – Результаты запуска теста auth_bloc_test.dart

Как видно из рисунка, все тесты в файле auth_bloc.test были пройдены успешно.

Аналогичным образом модульными тестами были покрыты и другие классы. Эти тесты повышают стабильность кодовой базы. Например, в случае, если разработчик поменяет функциональность класса AuthBloc, дополнив ее, можно запустить эти тесты и убедиться, что уже написанный функционал не был затронут. Также для этих целей запуск тестов необходимо добавлять в средства непрерывной интеграции (Continious Integration).

3.10 Написание тестов для виджетов

Тестирование виджетов – вид тестов, которые проверяют работоспособность конкретных виджетов [5]. Такие тесты были написаны

для некоторых виджетов. Например, был создан тест для виджета BillCard, как показано на рисунке 57.

```dart
void main() {
  testWidgets('BillCard has correct title and subtitle', (tester) async {
    await initializeDateFormatting();
    final bill = BillModel(
      id: 'id',
      name: 'name',
      account: 'account',
      bankName: 'bankName',
      corBill: 'corBill',
      bik: 'bik',
      amount: 100000,
      created: DateTime.parse('2020-07-17T03:18:31.177769-04:00'),
    );
    await tester.pumpWidget(MaterialApp(home: BillCard(bill: bill)));

    expect(find.text('100 000 ₽'), findsOneWidget);
    expect(find.text('От 17 июл. 2020 г.'), findsOneWidget);
  });
}
```

Рисунок 57 – Тест виджета BillCard

Данный тест проверяет, что сумма к оплате форматируется корректно, как и отображаемая в виджете дата выставления счета. Результаты выполнения теста показаны на рисунке 58.

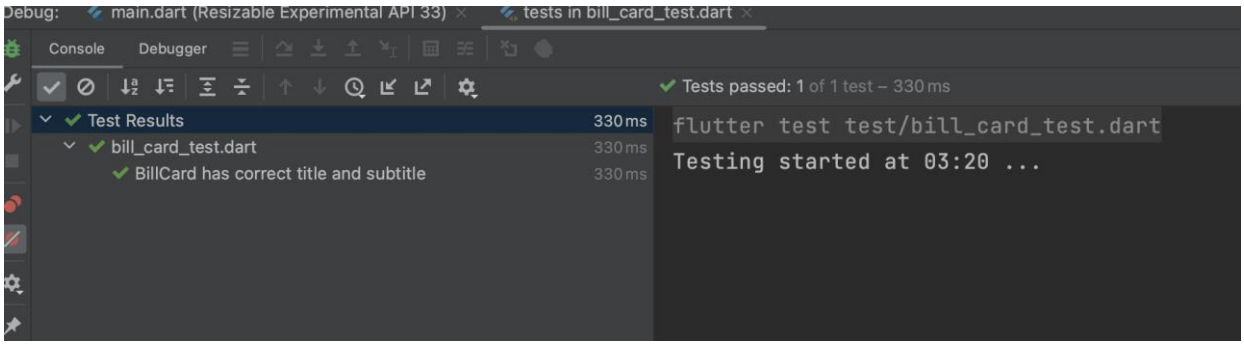

Рисунок 58 – Результаты запуска теста виджета BillCard

Как видно, тест был пройден успешно.

3.11 Интеграционное тестирование

Интеграционное тестирование позволяет протестировать все приложение или значительную его часть [6]. Этот вид тестов наиболее близок к ручному тестированию, практически он позволяет автоматизировать ручное тестирование. Интеграционное тестирование требует запуск приложения на устройстве или эмуляторе. Поэтому данный вид тестирования – наименее производительный из всех, предоставляемых Flutter.

Для автоматизации процесса ручного тестирования основные сценарии были покрыты интеграционными тестами. В качестве примера рассмотрим тест, который проводит регистрацию пользователя, переходит в раздел «Счета» и открывает экран оплаты счета. Фрагмент теста показан на рисунке 59.

```
await tester.pumpWidget(const MyApp());
await tester.pumpUntilFound(find.byKey(SignUpLabel.findKey));
await tester.tap(find.byKey(SignUpLabel.findKey));
await tester.pumpUntilFound(find.text('Регистрация'));

final email = '${getRandomString(10)}@test.at';
final password = getRandomString(5);

await tester.scrollTapAndEnter(find.byKey(SignUpForm.emailField), email);
await tester.scrollTapAndEnter(find.byKey(SignUpForm.passField), password);
await tester.scrollTapAndEnter(find.byKey(SignUpForm.pass2Field), password);

await tester.tap(find.byKey(SignUpForm.checkKey));
await tester.tap(find.byKey(SignUpForm.registerKey));

await tester.pumpUntilFound(find.byKey(HomeScreen.billsKey));
await tester.tap(find.byKey(HomeScreen.billsKey));

await tester.pumpUntilFound(find.text('Оплатить'));
await tester.pumpUntilButtonEnabled(find.byType(ElevatedButton).first);
await tester.tap(find.byType(ElevatedButton).first);
await tester.pumpUntilFound(find.text('Оплата'));
```

Рисунок 59 – Фрагмент интеграционного теста

В первую очередь после открытия приложения тест находит кнопку «Зарегистрироваться» и нажимает на нее. Затем происходит ввод тестовых

email и пароля, происходит принятие всех соглашений, а затем происходит нажатие на кнопку «Зарегистрироваться». При регистрации пользователей с email, содержащим домен test.at, на стороне сервера создается тестовый аккаунт с уже заполненными данными. Затем происходит нажатие на кнопку «Счета» внутри компонента BottomNavigationBar. После загрузки счетов происходит нажатие на первую кнопку «Оплатить», а затем ожидается открытие экрана «Оплата».

Результаты выполнения этого интеграционного теста представлены на рисунке 60.

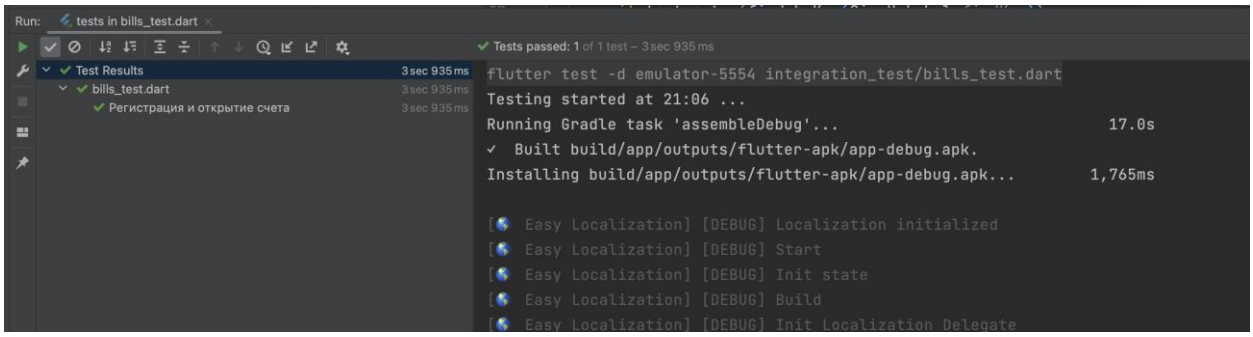

Рисунок 60 – Результат запуска интеграционного теста

Как видно, тест был пройден успешно. Его прохождение заняло около четырех секунд, что гораздо быстрее, чем прохождение аналогичного сценария человеком.

Аналогичными тестами покрыты основные сценарии приложения.

3.12 Тестирование серверной части приложения

Серверная часть приложения была также покрыта тестами. Для примера рассмотрим интеграционный тест процесса регистрации. На рисунке 61 представлен фрагмент этого теста.

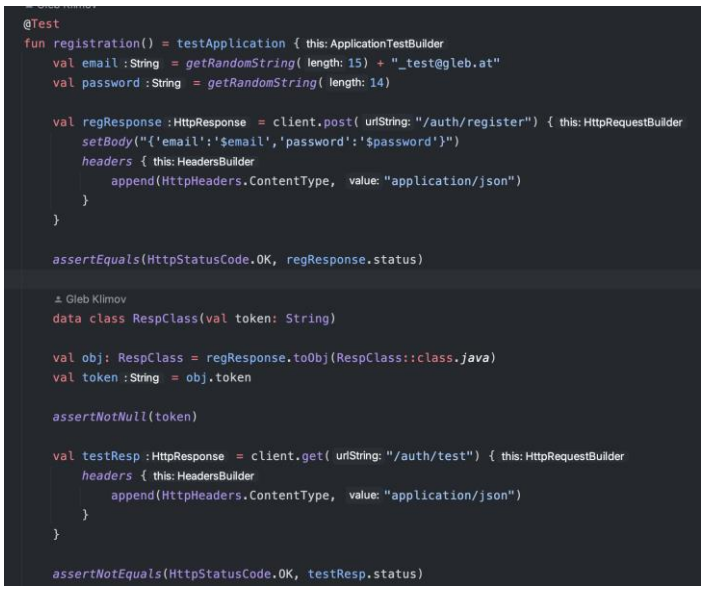

Рисунок 61 – Интеграционный тест процесса регистрации

В представленном фрагменте происходит вызов метода /auth/register, который регистрирует пользователя. Тест проверяет, что запрос прошел успешно, и в ответе был получен токен доступа. Затем делается запрос к тестовому методу /auth/test, после чего проверяется, что доступа к этому методу у пользователя нет. Это нужно для того, чтобы убедиться, что доступа к данным нет, если запрос происходит без токена доступа. После этого происходит аналогичный запрос, но уже с токеном доступа, что можно увидеть на фрагменте кода из рисунка 62.

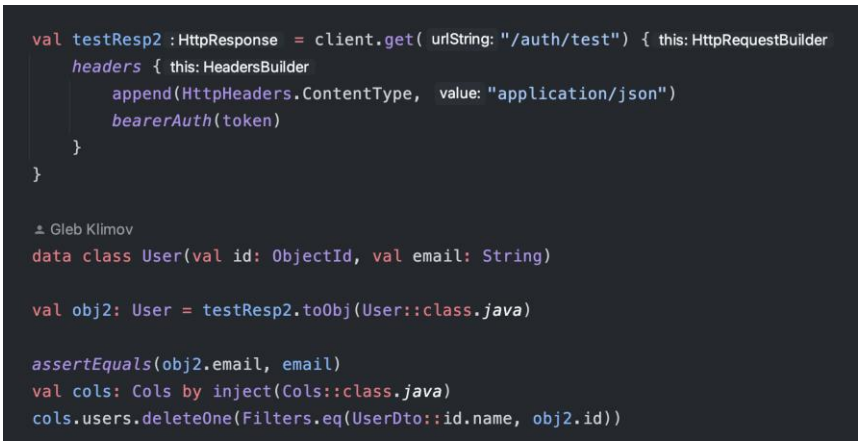

Рисунок 62 – Запрос с токеном доступа в рамках теста

В конце теста ненужный пользователь удаляется из базы данных. Результат успешного прохождения теста представлен на рисунке 63.

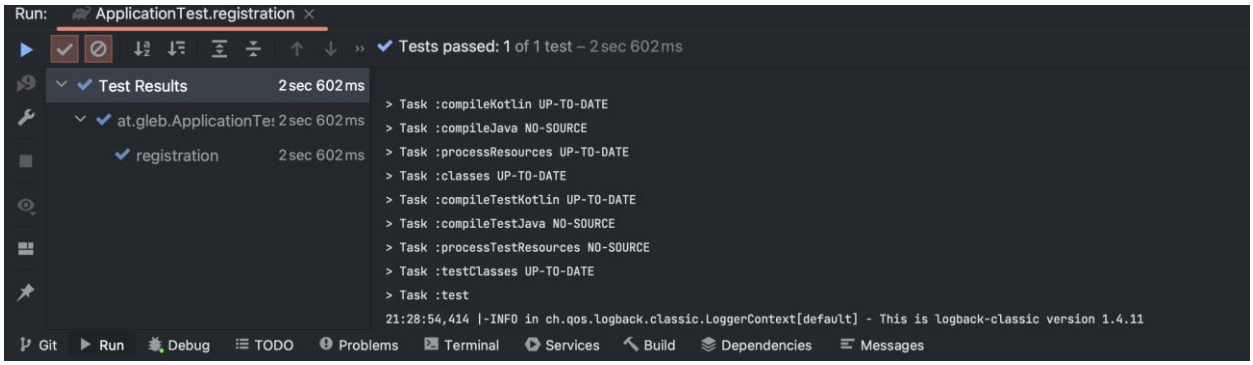

Рисунок 63 – Результаты запуска теста

Тест был пройден успешно чуть менее, чем за 3 секунды, что многократно меньше аналогичных проверок в случае, если бы они проводились человеком вручную.

Выводы по главе 3

В главе 3 было описано проведение физического моделирования как клиентской, так и серверной частей приложения для управления арендой кофе-машин. Для разработки клиентской части приложения был выбран Flutter, так как он является самым эффективным средством кроссплатформенной разработки мобильных приложений на данный момент. Для разработки серверной части приложения был выбран язык программирования Kotlin – высокопроизводительный и современный язык программирования, и фреймворк Ktor, позволяющий эффективно и быстро реализовать необходимый программный интерфейс для взаимодействия между клиентом и сервером. В качестве базы данных была выбрана база данных MongoDB.

В ходе разработки как на клиентской, так и на серверной стороне приложения была применена «чистая архитектура», позволяющая структурировать программный код эффективным образом, с учетом принципов SOLID. Благодаря этому код является легко тестируемым и изменяемым.

Для тестирования кода были применены различные виды тестов – модульное тестирование, тестирование виджетов и интеграционное тестирование. Модульное тестирование позволило покрыть код клиентской части высокопроизводительными тестами, улучшающими его стабильность. Тестирование виджетов позволило покрыть тестами некоторые ключевые виджеты. Интеграционные тесты позволили покрыть весь функционал приложения. Было показано, что они служат полноценной заменой ручному тестированию.

Заключение

В ходе проделанной работы было успешно спроектировано и разработано мобильное приложение для клиентов сервиса, предоставляющего возможность арендовать кофемашины. В рамках проектирования была проведена работа, включающая исследование бизнес-процесса, что показало необходимость его оптимизации. В рамках этого проектирования было решено разработать мобильное приложение для клиентов сервиса. Также были определены требования для разработки приложения с использованием нотации FURPS+. Помимо этого, был разработан современный дизайн интерфейса приложения, за основу которого был взят Material Design – руководство по дизайну, разработанное компанией Google.

Проектирование также включало в себя глубокую работу над архитектурой как клиентской, так и серверной части приложения. Так, помимо клиент-серверной архитектуры, серверная и клиентская части приложения были разработаны с использованием чистой архитектуры и принципов SOLID, что обеспечивает тестируемость приложения и позволяет легко масштабировать сложность кода, развивать приложение.

Разработанный программный код был покрыт различными тестами – модульными тестами, тестами виджетов, а также интеграционными тесты, которые автоматизируют ручное тестирование.

Разработанное приложение можно запускать на устройствах под управлением основных мобильных операционных систем – iOS и Android. Помимо этого, благодаря возможностям Flutter, имеется возможность расширить платформы для работы с приложением. Например, в будущем можно загрузить приложение под веб, а также под различные операционные системы, такие, как Windows.

Разработанное приложение удовлетворяет всем требованиям и успешно решает поставленные задачи.

Список используемой литературы и используемых источников

1. Заблуждения Clean Architecture [Электронный ресурс] URL: https://habr.com/ru/companies/mobileup/articles/335382/ (дата обращения: 04.05.2024)

2. Клиент — сервер [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80 (дата обращения: 04.05.2024)

3. Черемных С.В., Семенов И.О., Ручкин В.С. Моделирование и анализ систем. IDEF-технологии: практикум 2006. С. 39–45

4. An introduction to unit testing [Электронный ресурс] URL: https://docs.flutter.dev/cookbook/testing/unit/introduction (дата обращения: 04.05.2024)

5. An introduction to widget testing [Электронный ресурс] URL: https://docs.flutter.dev/cookbook/testing/widget/introduction (дата обращения: 04.05.2024)

6. An introduction to integration testing [Электронный ресурс] URL: https://docs.flutter.dev/cookbook/testing/integration/introduction (дата обращения: 04.05.2024)

7. App Review Guidelines [Электронный ресурс] URL: https://developer.apple.com/app-store/review/guidelines/ (дата обращения: 04.05.2024)

8. ChatGPT [Электронный ресурс] URL: https://chatgpt.com/ (дата обращения: 04.05.2024)

9. Carmine Zaccagnino Programming Flutter Native, Cross-Platform Apps the Easy Way 2020. 231 p.

10. DIGITAL 2024: GLOBAL OVERVIEW REPORT [Электронный ресурс] URL: https://datareportal.com/reports/digital-2024-global-overview-report (дата обращения: 04.05.2024)

11. Documenting non-functional requirements using FURPS+ [Электронный ресурс] URL: https://www.marcinziemek.com/blog/content/articles/8/article_en.html (дата обращения: 04.05.2024)

12. Docker [Электронный ресурс] URL: https://www.docker.com/ (дата обращения: 04.05.2024)

13. Enable users to personalize their color experience in your app [Электронный ресурс] URL: https://developer.android.com/develop/ui/views/theming/dynamic-colors (дата обращения: 04.05.2024)

14. Flutter Build for any screen [Электронный ресурс] URL: https://flutter.dev/ (дата обращения: 04.05.2024)

15. Human Interface Guidelines [Электронный ресурс] URL: https://developer.apple.com/design/human-interface-guidelines (дата обращения: 04.05.2024)

16. Introducing JSON [Электронный ресурс] URL: https://www.json.org/json-en.html (дата обращения: 04.05.2024)

17. Ivar Jacobson USE-CASE 2.0 The guide to succeeding with use cases 2011. 6 p.

18. Identifying Non-Functional Requirements [Электронный ресурс] URL: https://www.scribd.com/document/431815760/FURPS-docx (дата обращения: 04.05.2024)

19. Introduction to JSON Web Tokens [Электронный ресурс] URL: https://jwt.io/introduction (дата обращения: 04.05.2024)

20. Ian G. Clifton Android User Interface Design: Implementing Material Design for Developers (Usability) 2nd Edition 2015. 134 p.

21. JSON Web Token (JWT) [Электронный ресурс] URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата обращения: 04.05.2024)

22. Kotlin [Электронный ресурс] URL: https://kotlinlang.org/ (дата обращения: 04.05.2024)

23. Ktor [Электронный ресурс] URL: https://ktor.io/ (дата обращения: 04.05.2024)

24. Kyle Mew Learning Material Design 2015. 64 p.

25. Ktor Configuration in a file [Электронный ресурс] URL: https://ktor.io/docs/server-configuration-file.html (дата обращения: 04.05.2024)

26. Material Design [Электронный ресурс] URL: https://m3.material.io/ (дата обращения: 04.05.2024)

27. Mark Masse Rest API Design Rulebook 2012. 6 p.

28. Mouaz M. Al-Shahmeh Flutter Apps Development: Build Cross-Platform Flutter Apps with Trust 2023. 78 p.

29. MongoDB ObjectId [Электронный ресурс] URL: https://www.mongodb.com/docs/manual/reference/method/ObjectId/ (дата обращения: 03.06.2024)

30. Policy Center User Data [Электронный ресурс] URL: https://support.google.com/googleplay/android-developer/answer/10144311?hl=en&ref_topic=9877467&sjid=14412476242630736511-EU (дата обращения: 04.05.2024)

31. Peter Sugar Non-Functional Requirement Examples: with FURPS+ methodology 2016. 11 p.

32. Quickstart for GitHub Actions [Электронный ресурс] URL: https://docs.github.com/en/actions/quickstart (дата обращения: 04.05.2024)

33. Request runtime permissions [Электронный ресурс] URL: https://developer.android.com/training/permissions/requesting (дата обращения: 04.05.2024)

34. Rap Payne Beginning App Development with Flutter: Create Cross-Platform Mobile Apps 2019. 12 p.

35. Shannon Bradshaw, Eoin Brazil, Kristina Chodorow MongoDB: The Definitive Guide, 3rd Edition 2019. 28 p.

36. Set up an editor [Электронный ресурс] URL: https://docs.flutter.dev/get-started/editor (дата обращения: 04.05.2024)

37. Vladimir Khorikov Unit Testing Principles, Practices, and Patterns 2020. C. 5

38. Waleed Arshad Managing State in Flutter Pragmatically: Discover how to adopt the best state management approach for scaling your Flutter app 2021. 44 p.

Приложение А

Диаграмма состояний клиентской части приложения

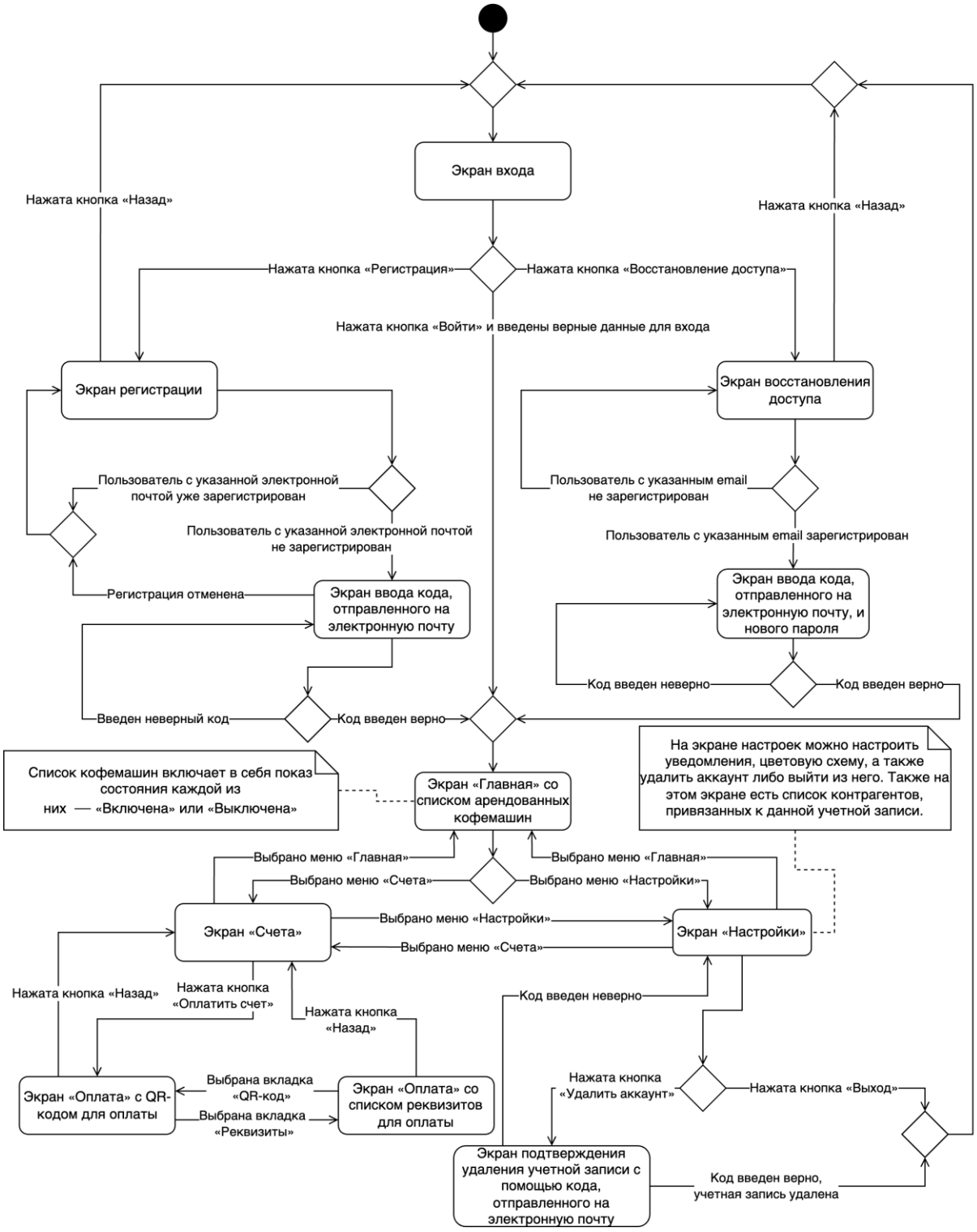

Рисунок А.1 – Диаграмма состояний клиентской части приложения

Приложение Б

Представление API-сервиса 1С в формате OpenApi 3.0

```
openapi: 3.0.0
info:
  title: Контрагенты API
  version: 1.0.0
paths:
  /contragents:
    get:
      summary: Получить всех контрагентов
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Contragent'

  /contragents/{code}:
    get:
      summary: Получить контрагента по коду
      parameters:
        - name: code
          in: path
          description: Код контрагента
          required: true
          schema:
            type: string
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Contragent'
        '400':
```

```
          description: Контрагент не найден

  /contragents/inn/{inn}:
    get:
      summary: Получить контрагента по ИНН
      parameters:
        - name: inn
          in: path
          description: ИНН контрагента
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Успешный запрос
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Contragent'
        '400':
          description: Контрагент не найден

components:
  schemas:
    Contragent:
      type: object
      required:
        - code
      properties:
        code:
          type: string
          description: Код контрагента (обязательное поле)
        inn:
          type: string
          description: ИНН контрагента
        email:
          type: string
          description: Адрес электронной почты контрагента
```

```
    name:
      type: string
      description: Имя контрагента
    bills:
      type: array
      items:
        $ref: '#/components/schemas/Bill'
      description: Выставленные счета
    devices:
      type: array
      items:
        $ref: '#/components/schemas/Device'
      description: Арендованные кофемашины
Bill:
  type: object
  required:
    - id
  properties:
    id:
      type: string
    name:
      type: string
    account:
      type: string
    bankName:
      type: string
    corBill:
      type: string
    amount:
      type: string
    created:
      type: integer
    paid:
      type: boolean
Device:
  type: object
  required:
    - id
```

```
  - phoneNumber
properties:
  id:
    type: integer
    description: Идентификатор кофемашины
  phoneNumber:
    type: string
    description: Номер телефона кофемашины (обязательное поле)
  machineName:
    type: string
    description: Имя кофемашины
```

Приложение С

Диаграмма деятельности приложения

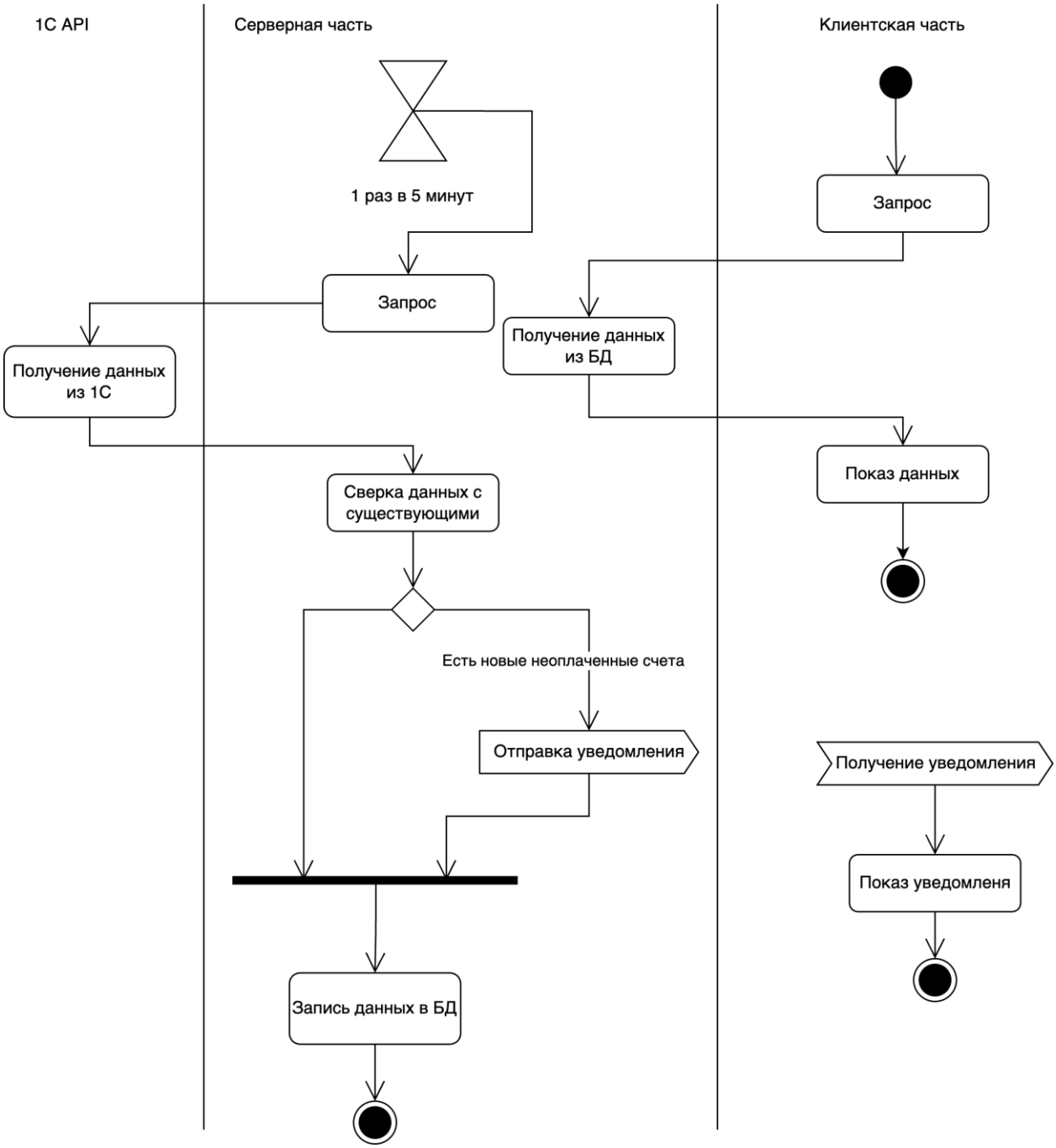

Рисунок Б.1 – Диаграмма деятельности приложения