Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Работа допущена к защите
Директор ВШПИ
_____ П.Д.Дробинцев
«___»_____2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

работа бакалавра

РАЗРАБОТКА ANDROID-ПРИЛОЖЕНИЯ ДЛЯ ПОКУПОК ЧЕРЕЗ АВТОМАТИЧЕСКИЕ КИОСКИ

по направлению подготовки (специальности)

09.03.04 Программная инженерия

Направленность (профиль)

09.03.04_01 Технология разработки и сопровождения качественного программного продукта

Выполнил студент гр. 5130904/10103 — А.А.Бокий

Руководитель
кандидат технических наук,
доцент ВШПИ ИКНК — О.Г.Малеев

Консультант
по нормоконтролю — Е.Г.Локшина

Санкт-Петербург
2025

Министерство науки и высшего образования Российской Федерации
Санкт-петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

УТВЕРЖДАЮ
Директор ВШПИ
_____ П.Д.Дробинцев
« » 20 г

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

студенту Бокий Алине Алексеевне, 5130904/10103
фамилия, имя, отчество (при наличии), номер группы

1. Тема работы: Разработка Android-приложения для покупок через автоматические киоски
2. Срок сдачи студентом законченной работы: 17.05.2025
3. Исходные данные по работе:
   - Документация языка программирования Kotlin
   - Документация Android SDK
4. Содержание работы (перечень подлежащих разработке вопросов):
   - Анализ решений в области автоматизированных киосков и постановка задач разработки
   - Выбор архитектурного подхода и средств разработки
   - Проектирование структуры и пользовательского интерфейса приложения
   - Реализация основных функциональных модулей системы
   - Реализация хранения данных и интеграция с внешними сервисами
   - Тестирование мобильного приложения
5. Перечень графического материала (с указанием обязательных чертежей): -

6. Перечень используемых информационных технологий, в том числе программное обеспечение, облачные сервисы, базы данных и прочие сквозные цифровые технологии (при наличии):

- Android SDK
- Kotlin
- Android Jetpack
- Dagger Hilt
- Firebase

7. Консультанты по работе: -

8. Дата выдачи задания: 14.04.2025

Руководитель ВКР _____ О.Г.Малеев
(подпись)

Задание принял к исполнению 14.04.2025

Студент _____ А.А.Бокий
(подпись)

РЕФЕРАТ

На 48 с., 2 рисунка.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, KOTLIN, ANDROID, XML, FIREBASE, АВТОМАТИЗИРОВАННЫЕ КИОСКИ, CLEAN ARCHITECTURE, MVVM, ROOM, HILT

Тема выпускной квалификационной работы: «Разработка Android-приложения для покупок через автоматические киоски».

Данная работа посвящена разработке Android-приложения для взаимодействия пользователя с автоматическим киоском самообслуживания. Приложение предоставляет базовый функционал: регистрацию и авторизацию, просмотр каталога и подробной информации о товарах, добавление товаров в корзину, оформление и оплату заказа. Также реализована возможность сканирования штрих-кодов товаров, привязки банковских карт и генерации QR-кода, с помощью которого пользователь входит в автокиоск.

Приложение реализовано с использованием архитектурного подхода Clean Architecture и паттерна MVVM на языке Kotlin. Для хранения данных используются облачная база Firestore и локальная база Room. Интерфейс создан с использованием XML и ViewBinding. Навигация построена с использованием Jetpack Navigation, внедрение зависимостей – на основе Hilt.

Проведено ручное и автоматизированное тестирование приложения. Результаты подтвердили корректность реализации и поведения интерфейса. Разработанное приложение подходит для использования в автоматических киосках и может развиваться в зависимости от потребностей бизнеса.

ABSTRACT

48 pages, 2 figures

MOBILE APPLICATION, KOTLIN, ANDROID, XML, FIREBASE, AUTOMATED KIOSKS, CLEAN ARCHITECTURE, MVVM, ROOM, HILT

The subject of the graduate qualification work is «Development of an Android application for shopping via automated kiosks».

This work focuses on the development of an Android application designed to interact with a self-service kiosk. The application provides core features including user registration and login, browsing a catalog with detailed product information, adding items to a cart, placing and paying for an order. It also supports barcode scanning, bank card linking, and QR code generation for kiosk access.

The system is built using the Clean Architecture approach and the MVVM design pattern in Kotlin. Data is stored using the Firestore cloud database and the local Room database. The user interface is implemented with XML and ViewBinding. Navigation is handled via Jetpack Navigation, and dependency injection is managed through Hilt.

Both manual and automated testing were performed. The results confirmed stable interface behavior and correct application logic. The developed solution is suitable for use in self-service kiosks and can be extended to meet additional business requirements.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Ритейлеры – компании или организации, осуществляющие розничную торговлю товарами напрямую конечному потребителю

СБП (Система быстрых платежей) – система, разработанная Банком России для мгновенных переводов между банками по номеру телефона или QR-коду

ADT (Android Development Tools) – набор плагинов для среды разработки Eclipse, предоставляющий инструменты для создания, отладки и сборки Android-приложений

Android – операционная система для мобильных устройств

Android SDK (Software Development Kit) – набор инструментов и библиотек для разработки Android-приложений

Clean Architecture – архитектурный подход, который разделяет приложение на слои для улучшения масштабируемости и тестируемости, упрощая поддержку и развитие

DAO (Data Access Object) – интерфейс для доступа к данным в базе данных, обеспечивающий абстракцию запросов к таблицам при использовании библиотеки Room

Firebase – платформа от Google, предоставляющая набор сервисов для разработки мобильных и веб-приложений, включая аутентификацию, базу данных в реальном времени, хранение файлов и аналитику

GitHub – онлайн-платформа для хранения, управления и совместной разработки исходного кода, использующая систему контроля версий Git

Gradle – система автоматической сборки, используемая в Android для компиляции, упаковки, тестирования и управления зависимостями проекта

LiveData – класс, который хранит объект данных и автоматически уведомляет подписчиков об изменениях, учитывает жизненный цикл компонентов

MVVM (Model-View-ViewModel) – архитектурный паттерн для разработки приложений

NoSQL-база – база данных, которая не использует привычную табличную структуру, а хранит данные в виде документов, пар «ключ–значение» или

других форматов

REST API (Representational State Transfer Application Programming Interface) – способ взаимодействия между клиентом и сервером через стандартные HTTP-запросы, используется для получения, добавления, изменения и удаления данных с помощью четко определенных URL и методов

Room – библиотека для работы с базами данных в Android-приложениях, использующая SQLite для хранения данных

StateFlow – поток данных из библиотеки Kotlin Coroutines, который применяется для хранения и отслеживания состояния с возможностью подписки на обновления

UID (Unique Identifier) – уникальный идентификатор, используемый для однозначного распознавания пользователя или объекта внутри системы

View – компонент пользовательского интерфейса в Android, отображающий элементы на экране и реагирующий на действия пользователя

ViewBinding – механизм Android, автоматически создающий классы привязки для XML-разметки, позволяющий безопасно и удобно работать с элементами интерфейса без использования findViewById

XML (eXtensible Markup Language) – расширяемый язык разметки

ВВЕДЕНИЕ

В условиях стремительного развития цифровых технологий все больше внимания уделяется автоматизации процессов в различных отраслях, в том числе в сфере розничной торговли. Одним из перспективных направлений является внедрение автоматических киосков, позволяющих осуществлять покупки без участия кассиров и продавцов. Такие решения направлены на снижение издержек для бизнеса, а также позволяют сократить очереди и ускорить процесс обслуживания.

Примеры внедрения киосков самообслуживания встречаются как за рубежом, так и в России. Среди известных зарубежных проектов можно отметить Amazon Go, а среди российских – микромаркеты Briskly. Однако массовое распространение таких решений пока ограничено, что определяет актуальность их дальнейшего развития и изучения.

Важную роль в работе автоматического киоска играет мобильное приложение, через которое пользователь может взаимодействовать с системой. Оно должно предоставлять доступ к каталогу товаров, содержать информацию о продуктах, позволять формировать корзину и осуществлять оплату. Таким образом, мобильное приложение становится основным интерфейсом взаимодействия с автоматическим киоском.

Практическая значимость настоящей работы заключается в разработке Android-приложения, предназначенного для взаимодействия пользователя с автоматическим киоском. Теоретическая значимость состоит в обосновании архитектурных и технологических решений, направленных на создание расширяемой и поддерживаемой архитектуры мобильного приложения.

Целью данной работы является разработка Android-приложения, обеспечивающего взаимодействие пользователя с автоматическим киоском. Приложение должно предоставлять доступ к каталогу товаров, отображать информацию о продуктах, позволять формировать покупки и производить

оплату без участия персонала. Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области и существующих решений.
2. Обосновать архитектурный подход и выбор технологий.
3. Разработать Android-приложение.
4. Протестировать приложение и проанализировать полученные результаты.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ

1.1. Обоснование актуальности

Идея автоматического киоска в последние годы вызывает все больший интерес и у покупателей, и у бизнеса. Такие магазины автоматизируют все процессы. Это предоставляет покупателям возможность совершать покупки без помощи персонала магазина. Эта модель пока не получила широкого распространения, но благодаря удобству и экономическим выгодам она будет востребована.

Мобильное приложение для автоматического киоска является связующим звеном между системой магазина и покупателями. Используя его, покупатели смогут войти в магазин, найти товары в каталоге, прочитать информацию о них, а так же оплатить их прямо в приложении. Следовательно, приложение выполняет основные этапы взаимодействия пользователя с магазином.

Бизнесам автоматические киоски дадут возможность значительно уменьшить операционные расходы. Одним из главных факторов экономии является отсутствие кассиров и продавцов, что позволяет сократить бюджет на зарплаты. Эти магазины могут работать круглосуточно, что делает их доступными для покупателей в любое время. Это добавит удобства для клиентов и позволит бизнесу увеличить объем продаж, а значит и прибыль.

Автоматический магазин соответствует тенденции на повышение скорости и комфорта покупки товаров. Покупатели смогут без участия персонала выбрать товары и расплатиться, не тратя время на очереди. Мобильное приложение в таких магазинах позволит автоматизировать процесс и предоставит простую и понятную схему взаимодействия с системой автокиоска.

Хотя такие магазины пока слабо распространены, они уже показали свою эффективность, например, магазины Amazon Go от компании Amazon и

микромаркеты от компании Briskly. По этим проектам можно понять, что такая концепция может быть успешно реализована в различных форматах торговли.

Таким образом, разработка мобильного приложения для интеграции с автоматическими киосками является актуальной задачей, которая отвечает современным потребностям бизнеса и покупателей.

1.2. Обзор существующих решений

Сейчас бесконтактные магазины без персонала представляют собой перспективное направление в ритейле. Такие решения обеспечивают автоматизацию процесса покупок и сокращают время на них. Далее описаны уже опробованные на практике и протестированные решения, которые используют такие технологии.

1.2.1. Amazon Go

Одним из первых магазинов такого типа стал магазин Amazon Go от компании Amazon. Его открыли в 2018 году.

Магазин использует технологию Just Walk Out. При входе в магазин покупатели должны отсканировать QR-код из своего аккаунта, затем они могут просто взять товары с полок и уйти. Система сама спишет сумму покупки с привязанной к аккаунту приложения банковской карты.

В магазинах установлены камеры, а также сенсоры и ИИ-алгоритмы. Эта технология задала высокую планку для своих конкурентов. На данный момент в США работает 16 магазинов Amazon Go.

1.2.2. Briskly

Briskly – российский стартап, который предлагает решение для создания автоматических киосков с использованием умных холодильников и микромаркетов. С 2018 года Briskly развивает собственные программные

продукты и оборудование автономной торговли в России.

Мобильное приложение B-Pay является кассой в смартфоне и использует функцию Scan&Go, клиенты сканируют штрих-коды и оплачивают товары без очередей в магазинах и микромаркетах [1]. Чтобы попасть в магазин без кассира, покупатель через приложение выбирает нужный супермаркет из списка, находит его и открывает электронный замок на двери.

Для предотвращения воровства в магазинах установлено много камер, через которые нейросеть оцифровывает каждого покупателя и отслеживает путь его перемещения и действия. Также при регистрации пользователь оставляет свои данные и заключает договор, который предусматривает серьезное наказание за кражу. Благодаря этому ритейлеры не боятся подключаться к новой технологии [2].

1.2.3. Магазин ВкуссВилл и Neurus

Компания Neurus совместно со ВкуссВилл открыла смарт-магазин без кассиров и продавцов в Москве в 2022 году. Принцип работы магазина основывается на системе компьютерного зрения и машинного обучения. На счету команды большое количество решений для ритейла, среди них контроль очередей и товарной выкладки, мониторинг кассовых операций на предмет выявления мошеннических действий со стороны персонала и покупателей. Одной из новейших разработок Neurus является собственная технология трекинга посетителей. В магазине используются практически все технологические наработки Neurus [3].

Для того чтобы пользоваться услугами магазина необходимо установить приложение “ВкуссВилл” и зарегистрироваться. На входе в магазин стоит турникет и терминал, который генерирует QR-коды. Этот код необходимо отсканировать через свой аккаунт в приложении.

Технологически магазин состоит из системы компьютерного зрения и полок с товарами, что изображено на рисунке 1. За действиями покупателей следит нейросеть с помощью камер. Если человек берет продукт, датчики на

полках это фиксируют, и выбранный товар добавляется в корзину в приложении, если возвращает на полку – удаляется из корзины.

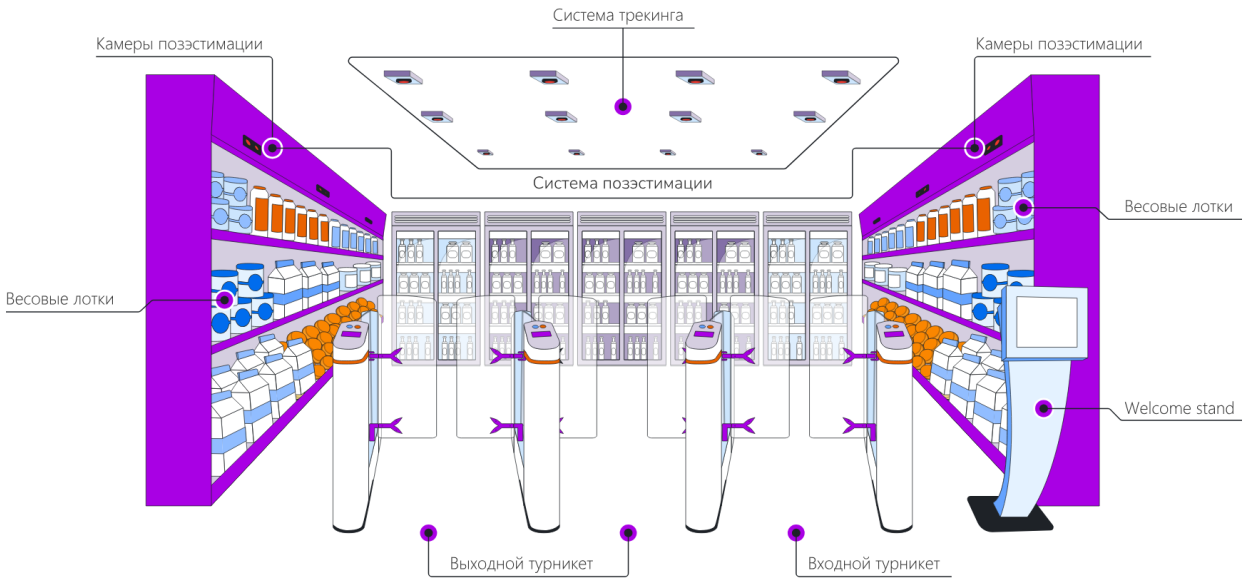

Рисунок 1. Принцип работы магазина

Магазин ВкуссВилл и Neurus является хорошим примеров внедрения технологий в розничную торговлю и демонстрирует потенциал таких решений в России.

1.2.4. Зоомагазин ZOOQI X

Технологическую платформу для зоомагазина ZOOQI X разработала минская IT-компания Kakadu Dev. Здесь используется система take&go, которая очень похожа на систему компании Amazon. Покупатель так же входит в магазин по QR-коду из приложения. Товары размечаются специальными метками, которые на выходе по тому же QR-коду удаленно считываются установленными антеннами, так система понимает, какой товар взял покупатель и сколько он стоит, и списывает с карты соответствующую сумму [4].

Если на карте недостаточно средств для оплаты покупки, в следующий раз приложение не выдаст покупателю QR-код и он не сможет войти в магазин. Кроме того, приложение будет пытаться списать оплату до тех пор, пока покупка не будет оплачена [5].

1.3. Сравнительный анализ и выявление недостатков

На основе рассмотренных решений можно выделить различные подходы к построению автоматизированных магазинов. Все системы стремятся упростить процесс покупок, однако реализуют это по-разному.

Amazon Go выделяется высокой степенью автоматизации: покупателю не нужно сканировать товары самостоятельно, что максимально экономит время. Дополнительным удобством является возможность настроить личную учетную запись – сохраняются предпочтения и история покупок. Среди недостатков – необходимость в большом количестве высокотехнологичного оборудования, что требует больших инвестиций. Алгоритмы искусственного интеллекта иногда допускают ошибки, что может приводить к неправильным начислениям. Кроме того, использование системы возможно только при наличии как приложения Amazon Go, так и учетной записи Amazon, что ограничивает и усложняет доступ.

Briskly отличается упрощенной технической реализацией: добавление товаров по штрих-коду делает систему менее затратной по сравнению с решениями, основанными на автоматическом распознавании. Также среди плюсов отмечается простой и понятный интерфейс приложения. Вместе с тем пользователи сталкиваются с рядом технических проблем – например, со сбоями при получении SMS-кодов для регистрации и с ошибками, мешающими завершить покупку. Кроме того, в приложении отсутствует поддержка оплаты через СБП, а также нет функции добавления микромаркетов и товаров в избранное, что снижает удобство использования.

Магазин ВкусВилл и Neurus использует одно и то же мобильное приложение как для автоматизированного формата, так и для обычных магазинов сети, что делает его более универсальным для пользователей. Еще одним преимуществом является использование весовых датчиков на полках: они позволяют точнее отслеживать действия покупателей – если товар берется или возвращается, это автоматически отображается в приложении. Главный

недостаток – высокая стоимость такой системы. Компьютерное зрение, интеллектуальные системы и датчики требуют больших вложени. Это затрудняет использование такого решения в большом масштабе.

Зоомагазин ZOOQI X выделяется дополнительными удобствами: в приложении можно быстро связаться с техподдержкой, если возникнут проблемы, а в магазине есть бесплатный Wi-Fi и зарядные устройства для смартфонов. Главный недостаток это использование электронных меток, каждая из которых стоит около 0,5 доллара. Это повышает себестоимость товаров и поэтому может отразиться на их цене для покупателей.

1.4. Выводы по главе

Автоматические киоски – одно из новых направлений в розничной торговле, связанное с автоматизацией процесса покупки. Они позволяют снизить издержки за счет отказа от кассиров и продавцов и упростить обслуживание покупателей. При этом, несмотря на преимущества, такие решения пока не получили широкого распространения, особенно в России. Основные причины – сложность реализации, высокая стоимость технологий и необходимость изменений в организации торговли.

Проведенный обзор показал, что на рынке уже существуют реализованные проекты с разным уровнем автоматизации – от полностью автономных магазинов, таких как Amazon Go, до решений с ручным сканированием, как в случае с Briskly. Некоторые системы демонстрируют высокий технологический уровень, но требуют значительных затрат на внедрение и обслуживание. Другие, наоборот, более просты в реализации, но уступают по удобству или функциональности.

Сравнение разных решений позволило определить их достоинства и недостатки. Наиболее частые проблемы связаны с большой стоимостью оборудования, зависимостью от сторонних экосистем, техническими сбоями, а также ограничениями пользовательского функционала. Эти выводы подтверждают актуальность задачи создания собственного мобильного

приложения, направленого на баланс между удобством, доступностью и возможностью адаптации под разные форматы автоматизированной торговли.

ГЛАВА 2. ОБОСНОВАНИЕ АРХИТЕКТУРНОГО ПОДХОДА И ВЫБОР ТЕХНОЛОГИЙ

2.1. Обоснование архитектурного подхода

2.1.1. Архитектурный подход и модель взаимодействия компонентов

Для построения архитектуры приложения была выбрана концепция Clean Architecture, которая предполагает четкое разделение ответственности между слоями. Такой подход помогает упростить разработку, сопровождение и масштабирование проекта, а также делает архитектуру более понятной и гибкой: каждый слой решает строго ограниченный круг задач и может изменяться независимо от остальных[6]. В рамках этой архитектуры приложение разделено на три уровня:

- Presentation – отвечает за отображение данных и взаимодействие с пользователем. В этом слое находятся View (фрагменты, адаптеры, основная активити), а также ViewModel, которые управляют состоянием интерфейса и обрабатывают пользовательские действия.
- Domain – содержит бизнес-логику приложения. В этом слое расположены сценарии использования (use-case), интерфейсы репозиториев и модели данных. ViewModel взаимодействуют с этим слоем для выполнения логических операций, не зависящих от конкретных способов хранения или получения данных.
- Data – отвечает за реализацию доступа к данным. Здесь находятся конкретные реализации репозиториев, DAO-интерфейсы для работы с базой данных Room, а также конфигурация базы. Этот слой не содержит бизнес-логики – он лишь предоставляет доступ к данным.

Для организации взаимодействия между пользовательским интерфейсом и логикой приложения в проекте был выбран архитектурный паттерн MVVM (Model-View-ViewModel)[7]. Одним из важных преимуществ MVVM является реактивная связь между View и ViewModel, то есть при изменении состояния данных интерфейс автоматически обновляется через

механизмы LiveData или StateFlow. Это снижает количество шаблонного кода, повышает надежность и упрощает поддержку проекта. Этот подход считается стандартом для разработки Android-приложений и рекомендован компанией Google.

Структура MVVM включает три основных компонента:

- View – отвечает за отображение данных и отправку действий пользователя. В проекте это реализовано через фрагменты, XML-разметку и активити.
- ViewModel – служит посредником между View и Model: она получает действия пользователя от View, запрашивает необходимые данные у Model и подготавливает результат для отображения. View подписывается на данные из ViewModel и автоматически обновляется при их изменении.
- Model – представляет собой источник данных и бизнес-логику. В контексте Clean Architecture она реализована в слоях domain и data. Именно здесь находятся use-case и репозитории, к которым обращается ViewModel для выполнения операций.

Совмещение MVVM с Clean Architecture помогло создать гибкую, масштабируемую и легко поддерживаемую архитектуру, в которой каждый компонент отвечает строго за свою часть логики.

2.1.2. Архитектура клиент-серверного взаимодействия

Клиент-серверное взаимодействие в приложении реализовано по модели «клиент – облачная база данных», без использования промежуточного REST API. Мобильное приложение напрямую взаимодействует с Firebase Firestore и Firebase Authentication, а для временного хранения данных используется локальная база Room. Общая структура этого взаимодействия представлена на рисунке 2.

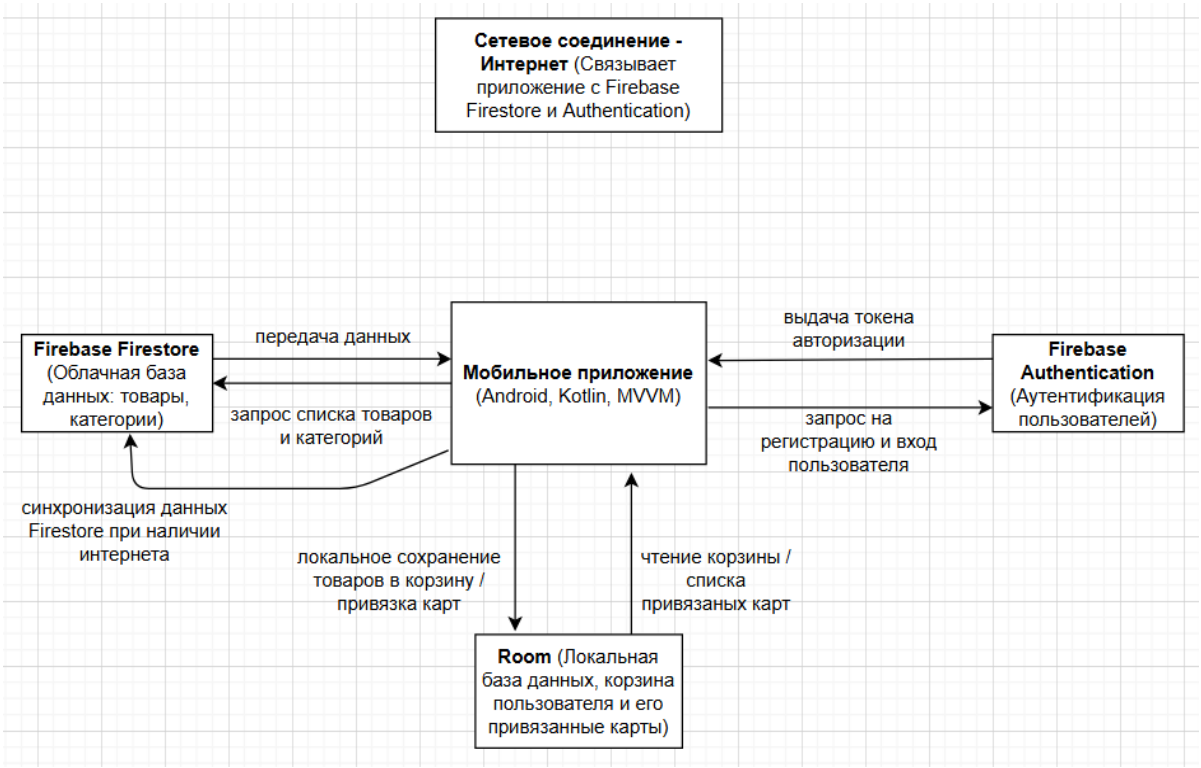

Рисунок 2. Архитектура клиент-серверного взаимодействия в приложении

Основные компоненты взаимодействия:

Мобильное приложение (клиент):

- отправляет запросы в Firebase Firestore для получения списка товаров и категорий;
- выполняет аутентификацию пользователей через Firebase Authentication;
- сохраняет корзину пользователя и его привязанные карты локально в Room, чтобы минимизировать сетевые запросы;
- обрабатывает полученные данные и отображает их в интерфейсе приложения.

Firebase Firestore (серверная часть):

- хранит информацию о товарах и категориях;
- обрабатывает запросы на получение данных, отправленные клиентским приложением;
- поддерживает автоматическую синхронизацию изменений.

Firebase Authentication (серверная часть)

- обеспечивает регистрацию и вход пользователей по email-адресу и

паролю;

- управляет хранением, безопасностью учетных данных на стороне сервера;
- предоставляет приложению информацию о текущем пользователе (например, его UID), необходимую для дальнейшего взаимодействия с базой данных.

Room (локальное хранилище)

- используется для хранения данных, которые не требуют постоянной синхронизации (корзины и банковских карт);
- снижает количество сетевых запросов, так как данные хранятся локально.

Сетевое соединение (интернет)

- используется для получения данных из Firestore;
- если интернет отсутствует данные временно хранятся локально и синхронизируются при восстановлении связи.

2.2. Выбор основных технологий разработки

2.2.1. Язык программирования и среда разработки

Так как разрабатываемое приложение будет использоваться на устройствах на платформе Android, в качестве основного языка программирования был выбран Kotlin. На сегодняшний день именно этот язык считается предпочитаемым для Android-разработки. Компания Google официально рекомендовала использовать Kotlin для создания новых проектов [8], это связано с его рядом преимуществ по сравнению с Java.

К достоинствам Kotlin относятся:

- лаконичность и «синтаксический сахар», упрощающий написание и чтение кода;
- полная совместимость с Java, это позволяет использовать множество существующих библиотек;

- поддержка современных возможностей, например, корутин [9], которые упрощают обработку асинхронных задач.

Android Studio является официальной интегрированной средой разработки (IDE), которая нужна для создания Android-приложений. Этот инструмент разработан Google и основан на платформе IntelliJ IDEA, созданной компанией JetBrains [10]. Впервые Android Studio была представлена в 2013 году, и уже тогда она стала заменой Eclipse с плагином ADT. С тех пор именно эта среда стала стандартом для разработки приложений под Android.

Одно из основных преимуществ Android Studio это тесная интеграция с Android SDK и Gradle. Благодаря этому разработка, тестирование и сборка мобильных приложений становится удобной. Также Android Studio предоставляет разработчику широкий набор инструментов для создания пользовательских интерфейсов, отладки и анализа производительности, это все упрощает этапы создания Android-приложения.

Основные преимущества Android Studio:

- интеграция с Android SDK и Gradle;
- встроенный редактор интерфейсов;
- эмулятор для тестирования приложений на различных версиях Android и на различных устройствах;
- поддержка Kotlin и Java;
- Gradle Build System – инструмент для автоматизации сборки приложений.

К недостаткам можно отнести:

- высокие системные требования;
- сравнительно медленный запуск IDE и эмулятора.

В целом в Android Studio есть все необходимое для разработки Android-приложений, и эта среда разработки остается на сегодняшний день наиболее подходящим инструментом в данной области.

2.2.2. База данных и работа с данными

В проекте используются два подхода к хранению данных: облачный и локальный. Облачное хранилище реализовано с помощью Firebase Firestore – масштабируемой NoSQL-базы данных от Google. Это облачная база, которая автоматически синхронизирует данные между сервером и клиентами. Приложение подключается к Firestore SDK и может выполнять операции чтения, записи и подписки на изменения в реальном времени. Firestore использует push-уведомления, чтобы мгновенно обновлять данные на клиенте при их изменении в базе[11]. Такой подход обеспечивает централизованный доступ к данным и удобную работу с ними в реальном времени.

Локальное хранилище реализовано с помощью библиотеки Room, которая работает поверх встроенной базы данных SQLite. Она позволяет использовать привычный объектный подход при работе с таблицами, а также автоматически компилирует SQL-запросы и проверяет их на наличие ошибок. Кроме того, Room поддерживает миграции, что упрощает изменение структуры базы данных при обновлениях приложения.[12]

В Firestore хранятся:

- список товаров с характеристиками;
- список категорий и подкатегорий товаров.

В локальной базе Room хранятся:

- товары, добавленные в корзину;
- информация о привязанных банковских картах.

Такое распределение данных делает приложение более гибким: локальное хранилище ускоряет работу с персональной информацией, а облачное обеспечивает доступ к общим данным.

2.2.3. Система контроля версий

Для управления исходным кодом приложения используется Git – распределенная система контроля версий. Она позволяет отслеживать любые

изменения в проекте, сохранять историю изменений, а также возвращаться к предыдущим версиям.

В рамках разработки приложения Git применяется для следующих задач:

- управление версиями и контроль изменений в проекте;
- хранение истории коммитов и возможность отката к предыдущим состояниям;
- интеграция с удаленным репозиторием на GitHub для резервного копирования и удобного доступа к коду с разных устройств.

Такая связка упрощает работу с кодом, позволяет быстро отслеживать изменения и надежно сохранять весь ход разработки.

2.2.4. Используемые библиотеки и инструменты

При создании приложения были выбраны библиотеки, которые решают конкретные задачи – от авторизации до навигации между экранами.

Firebase Authentication – предоставляет серверные службы, простые в использовании SDK и готовые библиотеки пользовательского интерфейса для аутентификации пользователей в приложении. Он поддерживает аутентификацию с использованием паролей, номеров телефонов, а также с помощью популярных поставщиков федеративных удостоверений (например, Google, Facebook и Twitter).[13]

ML Kit от Google – используется для сканирования штрих-кодов с помощью камеры устройства. Библиотека предоставляет инструменты, которые позволяют быстро и точно распознавать различные форматы штрих-кодов в реальном времени. В проекте используется модуль Barcode Scanning.[14]

Hilt – библиотека для внедрения зависимостей в Android-приложениях. Упрощает создание и подключение компонентов, снижает связанность между частями приложения [15].

Jetpack Navigation Component – используется для организации навигации между фрагментами. Navigation Component упрощает управление

навигационными переходами, поддерживает аргументы, граф переходов и позволяет централизованно управлять навигацией в приложении [16].

Kotlin Coroutines – используется для асинхронной обработки задач. Позволяют выполнять длительные операции, не блокируя основной поток.

2.3. Выводы по главе

Выбор архитектуры и технологий был основан на требованиях к читаемости кода и удобству масштабирования. Комбинация Clean Architecture и MVVM позволила четко разделить ответственность между слоями, упростить логику взаимодействия компонентов и сделать структуру проекта гибкой для доработок.

Клиент-серверное взаимодействие построено напрямую через Firebase Firestore и Firebase Authentication. Это позволило упростить архитектуру и обеспечить синхронную работу с облаком. Для временного хранения данных на устройстве используется Room, что снижает нагрузку на сеть и ускоряет доступ к персональной информации.

В качестве языка разработки выбран Kotlin – современный и лаконичный, официально поддерживаемый Google. Работа ведется в Android Studio, которая обеспечивает полную интеграцию с инструментами Android SDK и Gradle.

Таким образом, принятые архитектурные и технологические решения обеспечивают четкую организацию кода и возможность дальнейшего расширения функциональности без переработки существующих модулей.

ГЛАВА 3. РАЗРАБОТКА ANDROID-ПРИЛОЖЕНИЯ

3.1. Структура проекта и организация пакетов

Приложение разработано в одном модуле app с использованием среды Android Studio. Код проекта разделен на три основных логических слоя: presentation, domain и data.

Пакет presentation включает в себя код, отвечающий за отображение данных и взаимодействие с пользователем. Здесь расположены фрагменты, адаптеры для списков, классы ViewModel и главная Activity. Для упрощения навигации внутри слоя presentation структура дополнительно разделена на функциональные пакеты:

- auth – экраны регистрации и авторизации;
- barcode – экран сканирования штрих-кодов;
- card – экраны работы с банковскими картами;
- cart – экран корзины;
- catalog – экран каталога товаров;
- payment – экран оплаты;
- product – экран отображения информации о товаре;
- profile – экран профиля пользователя.

Пакет domain содержит бизнес-логику приложения. В этом слое находятся модели данных, интерфейсы репозиториев и сценарии использования (use-cases), описывающие основные операции над данными.

Пакет data реализует доступ к данным. Здесь расположены реализации репозиториев для взаимодействия с локальной базой данных Room и облачными сервисами Firebase, а также DAO-интерфейсы для работы с Room.

Пакет di предназначен для внедрения зависимостей с использованием библиотеки Hilt. В нем содержатся модули, обеспечивающие инициализацию основных компонентов приложения.

Такая структура проекта обеспечивает логическое разделение ответственности между компонентами, способствует упрощению навигации

по коду и облегчает процесс разработки и расширения функциональности приложения.

3.2. Проектирование и реализация пользовательских экранов

Пользовательские экраны приложения разработаны с использованием XML-разметки и технологии ViewBinding для упрощения работы с элементами пользовательского интерфейса. Переходы между экранами организованы с применением Navigation Component для упрощения управления навигацией в приложении.

Основные разделы приложения объединены в нижнюю панель навигации, которая предоставляет быстрый доступ к каталогу товаров, сканеру штрих-кодов, корзине и профилю пользователя. Отдельные экраны также предусмотрены для регистрации и авторизации пользователей, просмотра информации о товарах, работы с банковскими картами и оплаты.

3.2.1. Экран авторизации и регистрации

Экран авторизации нужен для входа пользователя в систему с помощью email-адреса и пароля. В интерфейсе есть два текстовых поля для ввода этих данных, и две кнопки: «Войти» для авторизации, и «Регистрация» для перехода к созданию нового профиля. При успешном входе приложение автоматически переключает пользователя на главный экран.

Экран регистрации позволяет создать новый аккаунт. Он многим похож на экран авторизации, здесь также есть поля для ввода email-адреса и пароля, но кроме этого тут предусмотрено поле для подтверждения пароля. Пользователю надо нажать кнопку «Регистрация», чтобы отправить данные, или кнопку перехода к авторизации чтобы вернуться к экрану авторизации. После успешного создания новой учётной записи система перенаправляет пользователя обратно на экран входа.

Для предотвращения ошибок при вводе реализована валидация данных. Приложение проверяет корректность email-адреса и минимальную длину

пароля. Оба данных процесса реализованы с использованием сервиса Firebase Authentication.

3.2.2. Главный экран и навигация

После успешной авторизации пользователь автоматически попадает на главный экран приложения. Этот экран служит контейнером для основных разделов, между которыми можно перемещаться с помощью нижней панели навигации. Сразу после входа пользователь видит каталог товаров.

В нижней части экрана находится панель, которая обеспечивает доступ к четырём разделам:

- каталог товаров;
- сканирование штрих-кодов;
- корзина покупок;
- профиль пользователя.

Каждый из пунктов панели связан со своим отдельным фрагментом. При выборе раздела отображается нужный экран. Для переходов между фрагментами используется навигационный граф приложения, это позволило реализовать навигацию без лишних сложностей.

3.2.3. Экран каталога товаров

Экран каталога товаров отвечает за отображение всех доступных товаров. Для реализации списка был выбран RecyclerView с адаптером, это решение довольно стандартное при работе с большим количеством элементов. Каждый товар в каталоге представлен отдельным элементом, который включает изображение, название, цену и вес. Вся информация для каталога загружается из облачной базы данных Firebase Firestore.

Пользователь может использовать встроенную фильтрацию товаров по категориям, а также быстро найти нужный продукт с помощью строки поиска. При нажатии на любой товар из списка открывается отдельный экран, на котором отображается более подробная информация о выбранном товаре.

Для удобства в каждом элементе списка реализована кнопка «В корзину». После добавления товара этот элемент преобразуется, то есть вместо кнопки появляется блок для управления количеством, это позволяет менять количество выбранных товаров прямо на экране каталога, не переходя отдельно в корзину.

3.2.4. Экран просмотра информации о товаре

Экран просмотра информации о товаре открывается при выборе товара из каталога. Он предназначен для отображения подробных характеристик выбранного продукта.

Интерфейс экрана включает:

- фотографии товара;
- название товара;
- вес товара;
- цену товара;
- кнопку «В корзину» (после добавления товара кнопка заменяется на блок управления количеством товаров).

Ниже основной информации расположен блок с подробной информацией о товаре, он включает состав продукта, пищевую ценность (КБЖУ), срок годности, условия хранения и бренд товара.

Данные для экрана передаются через аргументы навигации на основе выбранного товара.

3.2.5. Экран сканирования штрих-кодов

Экран сканирования штрих-кодов предназначен для быстрого поиска товаров с использованием камеры устройства. При открытии экрана камера активируется автоматически, и начинается процесс сканирования в реальном времени. Для распознавания штрих-кодов используется библиотека ML Kit от Google.

Если товар найден, в нижней части экрана появляется панель с названием товара и кнопкой «В корзину». Если товар не найден, пользователю показывается всплывающее диалоговое окно с сообщением «Товар не найден» и возможностью повторить попытку сканирования, нажав на кнопку «Попробовать еще раз». Также предусмотрена кнопка ручного ввода штрих-кода.

3.2.6. Экран корзины

Экран корзины предназначен для отображения товаров, добавленных пользователем в заказ. Для реализации списка товаров используется RecyclerView с адаптером.

Каждый элемент списка включает:

- изображение товара;
- название товара;
- текущее количество товара;
- общую стоимость выбранного количества товара.

Пользователь может изменять количество товаров с помощью кнопок «+» и «–». Товар можно удалить либо через уменьшение количества до нуля, либо с помощью отдельной кнопки удаления.

В нижней части экрана отображается итоговая сумма заказа, которая пересчитывается автоматически при каждом изменении корзины. Также здесь находится кнопка «Оформить заказ», при нажатии на которую пользователь переходит на экран оплаты.

3.2.7. Экран профиля пользователя

Экран профиля пользователя предназначен для просмотра информации об учетной записи и доступа к дополнительным функциям.

На экране отображается email-адрес пользователя, полученный через сервис Firebase Authentication. При наличии привязанных банковских карт

пользователю также доступен QR-код, который используется для входа в автоматизированный киоск. Если у пользователя нет привязанных карт, QR-код отображается в неактивном состоянии и нажатие на него не приводит к действию.

С экрана профиля доступен переход к списку банковских карт, а также возможность выхода из учетной записи. После подтверждения выхода пользователь автоматически перенаправляется на экран авторизации.

3.2.8. Экраны работы с банковскими картами и оплатой

Работа с банковскими картами реализована с помощью двух основных экранов: списка привязанных карт и экрана добавления новой карты.

На экране списка карт отображаются все карты, привязанные пользователем. Каждая карта представлена в формате маскированного номера (например, **** 1234), рядом с которым расположена кнопка удаления карты. Также на этом экране доступна кнопка «Добавить карту», при нажатии на которую пользователь переходит к экрану привязки карты.

Экран привязки карты позволяет пользователю добавить новую банковскую карту. Для этого необходимо ввести номер карты, срок ее действия и код безопасности карты. После ввода данных карта сохраняется и становится доступной для выбора при оплате.

Процесс оплаты осуществляется через отдельный экран. Пользователь выбирает одну из привязанных карт и нажимает кнопку «Оплатить». В случае отсутствия привязанных карт при попытке оплаты отображается уведомление с просьбой сначала выбрать карту.

В данной работе реализована имитация привязки банковских карт и проведения оплаты без фактической обработки платежных данных.

3.3. Разработка ключевых функций приложения

3.3.1. Регистрация и авторизация пользователей

Функциональность регистрации и авторизации пользователей реализована с использованием сервиса Firebase Authentication. Пользователь может создать новый аккаунт или войти в существующую учетную запись с помощью email-адреса и пароля.

Перед отправкой данных на сервер осуществляется локальная проверка корректности ввода, проверяется формат email-адреса и минимальная длина пароля. Логика работы с аутентификацией инкапсулирована в репозитории авторизации, который взаимодействует с Firebase. ViewModel авторизации обрабатывает результаты операций и обновляет состояние пользовательского интерфейса.

В случае успешной регистрации пользователь перенаправляется на экран авторизации. При успешной авторизации выполняется переход на главный экран приложения.

3.3.2. Работа с каталогом товаров

Каталог товаров загружается из облачной базы данных Firebase Firestore через репозиторий товаров. Доступ к данным реализован через отдельные use-case классы, которые предоставляют данные о товарах и категориях для дальнейшей работы во ViewModel.

Для отображения каталога используется RecyclerView с адаптером, который формирует карточки товаров на основе полученных данных. Реализована возможность поиска товаров по названию с помощью текстового поля: пользователь вводит ключевые слова, и ViewModel динамически обновляет отображаемый список товаров. Также предусмотрена фильтрация товаров по категориям, что позволяет упростить навигацию по большому количеству позиций.

3.3.3. Сканирование штрих-кодов и определение товаров

Функция сканирования штрих-кодов реализована с использованием библиотеки ML Kit от Google. После распознавания штрих-кода при наведении камеры или после ручного ввода приложение выполняет обращение к облачной базе данных Firebase Firestore для поиска соответствующего товара по его уникальному идентификатору. Поиск инкапсулирован в use-case и осуществляется через репозиторий товаров. Найденный товар или информация об отсутствии товара передаются во ViewModel для последующего отображения пользователю.

3.3.4. Работа с корзиной товаров

Функциональность корзины реализована с использованием локальной базы данных Room. Все добавленные пользователем товары сохраняются локально, что обеспечивает доступ к содержимому корзины без необходимости постоянного обращения к сети.

ViewModel корзины управляет состоянием списка товаров, обеспечивает возможность увеличения и уменьшения количества позиций, а также удаления товаров при необходимости. Каждое изменение количества автоматически отражается в расчете общей стоимости заказа.

Операции с корзиной инкапсулированы в соответствующих use-case классах, что позволяет отделить бизнес-логику от пользовательского интерфейса и упростить поддержку кода.

3.3.5. Привязка и удаление банковских карт

Функциональность работы с банковскими картами реализована с использованием локальной базы данных Room. Доступ к данным осуществляется через репозиторий карт, который взаимодействует с DAO-интерфейсом для выполнения операций добавления, получения списка и удаления карт.

Карты сохраняются только локально и не отправляются на сервер. В приложении привязка карт и работа с ними имитируется без реальной обработки платежных данных.

Операции с картами инкапсулированы в use-case классы, обеспечивающие отделение бизнес-логики от пользовательского интерфейса. ViewModel обеспечивает загрузку списка карт, их обновление при изменениях и удаление по запросу пользователя.

3.3.6. Оформление заказа и оплата

Процесс оформления заказа в приложении реализован через отдельный экран оплаты. После нажатия кнопки «Оформить заказ» в корзине пользователь переходит на экран выбора карты.

После выбора карты и нажатия кнопки «Оплатить» запускается имитация процесса обработки платежа: пользователю отображается сообщение о ходе оплаты. Успешность оплаты определяется случайным образом.

В случае успешной оплаты пользователю предлагается вернуться в каталог товаров, а содержимое корзины автоматически очищается. При неудачной попытке появляется возможность повторить оплату.

Если при попытке оплаты не выбрана ни одна карта, выводится уведомление о необходимости сначала привязать карту.

3.4. Выводы по главе

В ходе разработки мобильного приложения была реализована архитектура с разделением кода на слои presentation, domain и data, что обеспечило четкую организацию проекта и упростило поддержку.

Созданы пользовательские экраны для регистрации и авторизации, просмотра каталога товаров, сканирования штрих-кодов, управления корзиной, профилем пользователя, привязки банковских карт и оплаты. Ключевые функции приложения успешно реализованы с использованием

выбранных инструментов и технологий.

Использование паттернов MVVM и Clean Architecture позволило обеспечить устойчивость проекта к расширению и внесению изменений без переработки всех модулей.

ГЛАВА 4. ТЕСТИРОВАНИЕ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1. Общий подход к тестированию

Тестирование является важной частью процесса разработки и помогает убедиться в корректности работы реализованных функций и надежности приложения. В ходе работы проверялась как логика отдельных функций, так и поведение пользовательского интерфейса.

Основное внимание уделялось ручному функциональному тестированию, при котором проверялись ключевые пользовательские сценарии. Среди них регистрация и авторизация, навигация между экранами, взаимодействие с каталогом, добавление и удаление товаров из корзины, сканирование штрих-кодов, работа с банковскими картами и оформление заказа. Тестирование охватывало не только обычные действия, но и возможные ошибки, например, неправильный ввод пользователя или отсутствие нужных данных.

В дополнение к ручному тестированию в проекте были реализованы как модульные, так и инструментальные тесты. Модульные тесты размещены в директории test и охватывают бизнес-логику приложения: use-case классы, реализации репозиториев и отдельные ViewModel. Эти тесты не зависят от Android API и позволяют проверять логику в изоляции. Инструментальные тесты находятся в директории androidTest и используются для проверки компонентов, взаимодействующих с Android-средой, включая DAO-интерфейсы базы данных Room, миграции и функции, зависящие от Android API. Тестирование проводилось с использованием библиотек JUnit и MockK.

Тесты запускались как на эмуляторе Android, так и на реальном мобильном устройстве для проверки корректности работы приложения в различных условиях. Это позволило убедиться в стабильности поведения интерфейса и логики приложения на разных типах устройств и при различных сценариях использования.

Сочетание ручного и автоматизированного тестирования позволило

проверить корректность работы приложения и устранить ошибки еще на раннем этапе.

4.2. Ручное тестирование пользовательских сценариев

4.2.1. Регистрация и авторизация

Функции регистрации и авторизации служат начальной точкой в приложении. Эти этапы являются обязательными этапами начала работы с системой. Во время тестирования проверялось то, насколько корректно обрабатываются пользовательские данные, проверялась работа валидации, а также стабильность переходов между экранами.

Проверялись различные варианты взаимодействия, и стандартные сценарии с правильным вводом данных, и граничные ситуации. Например, отдельно анализировались попытки входа с ошибочными данными или случаи повторной регистрации на один и тот же email-адрес. Тестирование проводилось не только в среде эмулятора, но и на мобильном устройстве. Это позволило выявить возможные отличия и убедиться, что интерфейс работает стабильно и предсказуемо в разных условиях.

Тестируемые сценарии:

- Регистрация нового пользователя с корректными данными.
  После заполнения всех полей, а именно email-адреса, пароля и его подтверждения, а также после нажатия кнопки «Регистрация» успешно создавался новый аккаунт с помощью сервиса Firebase Authentication. Появлялось уведомление о завершении процесса регистрации: «Регистрация прошла успешно!». Затем происходил автоматический переход к экрану входа. Это говорит о том, что регистрация выполнялась корректно, логика навигации реализована правильно и не вызывает ошибок.
- Ввод некорректного email-адреса.
  При попытке ввода email-адреса в некорректном виде, например, без

символа «@», с пробелами или запрещенными символами, приложение не позволяет зарегистрироваться и отображает соответствующее предупреждение. Это указывает на корректную работу валидации данных.

- Регистрация с некорректным паролем.

  В случае задания пароля длиной менее 6 символов регистрация не происходит и пользователю отображается соответствующее предупреждение. Данное поведение соответствует спецификации Firebase Authentication.

- Попытка регистрации с уже существующим email-адресом.

  При регистрации с email-адресом, уже использовавшимся в системе, отображается сообщение об ошибке и регистрация не выполняется. Таким образом, данная ситуация обрабатывается корректно и дублирование учетных записей не происходит.

- Авторизация с корректными данными.

  Ввод ранее зарегистрированного email-адреса в сочетании с правильным паролем приводит к успешной аутентификации и переходу к основному экрану приложения.

- Авторизация с некорректными данными.

  При вводе неправильного пароля или несуществующего email-адреса авторизация не срабатывает. Интерфейс приложения отображает сообщение «Ошибка входа!», что соответствует ожидаемому поведению.

- Переход между экранами регистрации и входа.

  При нажатии на кнопку «Регистрация» на экране входа происходит переход к экрану создания аккаунта. Кнопка «Уже есть аккаунт? Войти» на экране регистрации возвращает пользователя к экрану авторизации. Навигация между экранами работает стабильно и без сбоев.

4.2.2. Каталог и карточка товара

Раздел каталога является одним из основных экранов приложения, предоставляя пользователю доступ к списку всех товаров и информации о них. При ручном тестировании проверялись корректность отображения данных, реакция интерфейса на действия пользователя, а также переходы к экрану карточки товара.

Тестируемые сценарии:

- Отображение списка товаров в каталоге.

  После входа в приложение отображается список товаров, полученных из базы данных Firebase Firestore. Для каждого товара корректно выводятся основные характеристики: изображение, название, вес, цена. Список загружается корректно и без задержек.

- Добавление товара в корзину из каталога.

  Под каждым товаром в каталоге находится кнопка «В корзину». После нажатия на нее товар успешно добавляется в корзину, и появляется блок управления количеством, состоящий из кнопок «−», текущего количества товара и «+». Повторные нажатия корректно изменяют количество товара. При уменьшении количества до нуля блок исчезает, и снова отображается кнопка «В корзину». Поведение интерфейса стабильное, все изменения происходят без задержек.

- Фильтрация по категориям и поиск товара.

  При выборе определенной категории или подкатегории список товаров корректно обновляется, отображаются только соответствующие позиции. Динамический поиск по названию товара работает корректно, при каждом вводе или удалении символа список автоматически обновляется и показывает только подходящие товары.

- Переход к карточке товара.

  При нажатии на карточку в каталоге происходит переход на экран с подробной информацией о выбранном товаре, навигация работает корректно.

- Отображение содержимого карточки товара.

  На экране карточки товара отображаются галерея фотографий, полное название, вес, цена, состав и другие характеристики. Все элементы корректно загружаются и заполняются на основе данных из Firebase Firestore.
- Добавление товара в корзину из карточки.

  На экране карточки товара также доступна кнопка «В корзину» с таким же поведением, как и в каталоге. После добавления отображается блок управления количеством. Добавление, увеличение и уменьшение количества работают корректно. Количество товара в подробной карточке товара соотвествует количеству товара в каталоге.

4.2.3. Сканирование штрих-кодов

Функция сканирования штрих-кодов позволяет пользователю быстро добавить товар в корзину, просто наведя камеру на товар. Во время ручного тестирования проверялись корректность распознавания кодов, а также реакция интерфейса на успешное распознавание и отсутствие товара в базе данных.

Тестируемые сценарии:

- Запуск сканера.

  При переходе на экран сканирования автоматически запускается камера устройства. Интерфейс отображает графическую подсказку для наведения. Камера активируется без ошибок и задержек.
- Распознавание корректного штрих-кода.

  При наведении камеры на штрих-код, зарегистрированный в базе данных, код распознается, и на экране отображается всплывающее окно с названием найденного товара и кнопкой «В корзину». Распознавание происходит быстро и не требует дополнительных действий.

- Попытка сканирования несуществующего товара.

  Если отсканированный штрих-код не соответствует ни одному из товаров в базе, пользователю отображается сообщение «Товар не найден» и кнопка «Попробовать еще раз», при нажатии на которую происходит возврат к сканеру.
- Повторное сканирование.

  После неудачного сканирования повторный запуск камеры работает корректно. Сканер не зависает, приложение стабильно обрабатывает множественные попытки.
- Ручной ввод штрих-кода.

  При нажатии на кнопку «Ввести код вручную» открывается поле для ввода цифрового значения штрих-кода. После введения кода и нажатия кнопки «Найти товар» приложение выполняет его поиск. При успешном совпадении отображается окно с названием товара и кнопкой «В корзину», аналогично сценарию со сканированием камерой. В случае, если товар не найден, выводится сообщение «Товар не найден». Интерфейс реагирует корректно, поведение соответствует логике обработки при сканировании.

4.2.4. Корзина и оформление заказа

Экран корзины дает пользователю возможность просматривать добавленные товары, управлять их количеством и переходить к оформлению заказа. При ручном тестировании проверялись корректность отображения товаров, изменение количества, удаление, отображение итоговой суммы, а также навигация к экрану оплаты.

Тестируемые сценарии:

- Отображение содержимого корзины.

  На экране корзины отображаются все добавленные товары с указанием названия, количества и итоговой стоимости по каждому товару. Изображения товаров загружаются корректно.

- Изменение количества товаров.

  Кнопки «+» и «−» у каждого товара работают корректно, при изменении количества оно сразу обновляется в интерфейсе, пересчитывается итоговая стоимость. При уменьшении до нуля товар удаляется из корзины.
- Удаление товара из корзины.

  Удаление происходит при нажатии на кнопку «×». После удаления товар исчезает из списка, итоговая сумма пересчитывается корректно.
- Итоговая сумма заказа.

  Корректная итоговая сумма отображается в нижней части экрана и автоматически обновляется при каждом изменении содержимого корзины. Расчет суммы осуществляется с учетом количества и стоимости всех позиций.
- Переход к оформлению заказа.

  При нажатии на кнопку «Оформить заказ» происходит переход на экран оплаты, навигация работает корректно.

4.2.5. Профиль и банковские карты

Экран профиля включает функции управления банковскими картами, выход из аккаунта и QR-код для входа в автокиоск. В процессе ручного тестирования проверялись работа соответствующих элементов интерфейса и корректность их поведения.

Тестируемые сценарии:

- Отображение данных профиля.

  На экране профиля отображается верный email-адрес пользователя, полученный из Firebase Authentication.
- Выход из аккаунта.

  При нажатии на кнопку «Выйти» отображается диалоговое окно с запросом подтверждения выхода. После нажатия на кнопку

подтверждения выполняется выход из аккаунта и происходит переход на экран входа. Поведение соответствует ожидаемому сценарию, интерфейс работает корректно.

- Отображение QR-кода.

  В профиле отображается QR-код, который нужен для входа в автокиоск. Если привязана хотя бы одна банковская карта, то при нажатии на QR-код открывается его увеличенная версия. При отсутствии привязанных карт QR-код выглядит неактивным и недоступен для открытия, при нажатии появляется сообщение «Сначала добавьте способ оплаты». В ходе тестирования проверялась логика отображения QR-кода в зависимости от наличия привязанных карт и поведение интерфейса соответствует ожидаемому.

- Список банковских карт.

  При нажатии на кнопку «Способы оплаты» отображается список ранее привязанных карт. Для каждой карты отображаются четыре последние цифры и иконка удаления. Все данные извлекаются из локального хранилища и отображаются корректно. При нажатии кнопки «×» карта сразу удаляется из списка.

- Добавление банковской карты.

  При нажатии на кнопку «Добавить карту» открывается форма ввода данных карты. При корректном заполнении всех полей и нажатии на кнопку «Привязать карту» карта добавляется в общий список. При некорректном вводе система отображает сообщение «Некорректные данные карты». Поведение проверено с различными комбинациями допустимого и недопустимого ввода, интерфейс реагирует корректно.

4.3. Выводы по результатам тестирования

Проведенное тестирование подтвердило корректность работы всех ключевых пользовательских сценариев и стабильность интерфейса при

типовых и граничных условиях. Проверялись как стандартные действия, так и ситуации с некорректным вводом, отсутствием данных и переходами между экранами. Во всех случаях приложение реагировало предсказуемо и без сбоев.

Были протестированы все основные модули: регистрация и авторизация, работа с каталогом и подробными карточками товаров, работа сканера штрих-кодов, добавление и удаление товаров из корзины, оформление заказа, управление способами оплаты и отображение QR-кода для входа в автокиоск.

Из результатов тестирования можно сделать вывод о готовности приложения к использованию. Выявленные ошибки устранены, а функционирование системы соответствует ожиданиям.

ЗАКЛЮЧЕНИЕ

В ходе данной работы было разработано Android-приложение для покупателей, которые пользуются автоматическими киосками самообслуживания. В приложение добавлены все основные функции, необходимые для удобного взаимодействия: регистрация нового пользователя, авторизация, просмотр каталога товаров, возможность добавления выбранных товаров в корзину как вручную, так и с помощью сканирования штрих-кода, оформление заказа и оплата привязанными банковскими картами. Реализованы все основные пользовательские экраны и сценарии, которые затем были протестированы.

Перед тем как приступить к разработке, был проведён анализ предметной области и изучены уже существующие решения в данной области. Этот подход помог определить, какие функции действительно важны для пользователей автоматизированных магазинов, а также понять, какие из них нужно реализовать в первую очередь.

Для построения структуры приложения использовалось сочетание принципов Clean Architecture и архитектурного паттерна MVVM. Такой подход позволил разделить код на отдельные слои, благодаря чему организация проекта стала более понятной, а сопровождение и доработка стали удобнее. В ходе разработки применялись такие инструменты, как Kotlin, Room, Firebase и другие библиотеки, нужные для реализации функционала.

Приложение прошло как ручное, так и автоматизированное тестирование. Проверялись все пользовательские сценарии от регистрации до оформления заказа. Интерфейс вел себя стабильно, ошибки обрабатывались корректно, навигация работала правильно.

В результате были выполнены все задачи, поставленные во введении:

1. Проведен анализ предметной области и существующих решений.
2. Обоснован архитектурный подход и выбран стек технологий.
3. Разработано Android-приложение.
4. Проведено тестирование и проанализированы результаты.

В дальнейшем приложение можно расширить, например добавить интеграцию с настоящей платежной системой и реализовать историю покупок. Также можно улучшить интерфейс и добавить дополнительные элементы управления.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Briskly: технологии для торговли без персонала [Электронный ресурс]. Режим доступа: https://briskly.online/
2. Как российский стартап Briskly запускает магазины без персонала [Электронный ресурс]. Режим доступа: https://www.retail.ru/articles/kak-rossiyskiy-startap-briskly-zapuskaet-magaziny-bez-personala/
3. Открытие смарт-магазина компанией Neurus совместно со ВкуссВилл [Электронный ресурс]. Режим доступа: https://www.retail.ru/rbc/pressreleases/budushchee-rossiyskoy-torgovli-uzhe-v-moskva-siti-neurus-sovmestno-so-vkusvill-otkryl-smart-magazin-/
4. Открытие зоомагазина ZOOQI X без касс и продавцов [Электронный ресурс]. Режим доступа: https://tech.onliner.by/2018/10/18/zooqi
5. Зоомагазин ZOOQI X без касс и продавцов в Минске [Электронный ресурс]. Режим доступа: https://web.archive.org/web/20200926022029/https://finance.tut.by/news612223.html
6. Статья «The Clean Architecture», Robert C. Martin [Электронный ресурс]. Режим доступа: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
7. Архитектурный паттерн MVVM в Android [Электронный ресурс]. Режим доступа: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/
8. Kotlin vs Java: which is better for android development [Электронный ресурс]. Режим доступа: https://www.octalsoftware.com/blog/kotlin-vs-java-for-android-development
9. Coroutines [Электронный ресурс]. Режим доступа: https://kotlinlang.org/docs/coroutines-overview.html
10. Android Studio [Электронный ресурс]. Режим доступа: https://developer.android.com/studio/intro

11. Документация Firebase Firestore [Электронный ресурс]. Режим доступа: https://firebase.google.com/docs/firestore
12. База данных Room, официальная документация [Электронный ресурс]. Режим доступа: https://developer.android.com/jetpack/androidx/releases/room
13. Документация Firebase Authentication [Электронный ресурс]. Режим доступа: https://firebase.google.com/docs/auth
14. Библиотека для сканирования штрих-кодов ML Kit, официальная документация [Электронный ресурс]. Режим доступа: https://developers.google.com/ml-kit/vision/barcode-scanning
15. Внедрение зависимостей с помощью Hilt, официальная документация [Электронный ресурс]. Режим доступа: https://developer.android.com/training/dependency-injection/hilt-android
16. Компонент навигации Android (Navigation Component), официальная документация [Электронный ресурс]. Режим доступа: https://developer.android.com/guide/navigation