# ВВЕДЕНИЕ

В современном мире технологический прогресс непрерывно изменяет наши повседневные привычки и потребности. Одним из динамично развивающихся секторов является сфера услуг доставки. В условиях активного обмена товарами и услугами как международного, так и локального характера, важным моментом является развитие эффективных и удобных инструментов управления курьерской доставкой.

В этом контексте мобильные приложения становятся одним из ключевых элементов современного бизнеса в сфере доставки. Их функциональность позволяет как заказчикам, так и поставщикам услуг управлять и отслеживать доставку, обеспечивая высокий уровень сервиса и комфорта.

Данный дипломный проект посвящен разработке мобильного приложения, направленного на управление процесса доставки и управления курьерскими услугами. Целью данного проекта является создание удобного решения, способного удовлетворить потребности как заказчиков, так и поставщиков услуг доставки, обеспечивая максимальную эффективность и удобство взаимодействия.

В работе будут рассмотрены основные аспекты разработки мобильного приложения, начиная от анализа потребностей пользователей и функциональных требований, заканчивая проектированием интерфейса и реализацией программного продукта.

Результаты данного проекта могут быть востребованы как в коммерческой сфере, так и в сфере общественного обслуживания, способствуя качеству услуг доставки в целом. Данное приложение имеет потенциал стать неотъемлемым инструментом для широкого круга пользователей, сокращая время и усилия, затрачиваемые на организацию и контроль процесса доставки.

# СИСТЕМНЫЙ АНАЛИЗ И ПОСТАНОВКА ЗАДАЧИ

## 1.1 Анализ предментной области

Рынок доставки еды демонстрирует стремительный рост в последние годы. Например, в 2024 году рынок доставки еды продолжает демонстрировать заметный рост, подтверждаемый увеличением его объема на 20% по сравнению с прошлым годом, и, по оценкам экспертов, может достигнуть 1 трлн рублей. Этот рост является результатом не только последствий COVID-19 и изменений в потребительских привычках, но и увеличения доли заведений с доставкой за последние два года выросла и составила 61% от общего числа заведений в России. Это подчеркивает важность понимания того, что доставка - это полноценный канал продаж и инструмент для повышения лояльности клиентов.

Среди ключевых трендов, которые заслуживают внимания, стоит отметить, что около трети заведений, у которых нет доставки в данный момент, планируют подключить доставку в горизонте 1-2 лет. При этом 46% из них планируют развивать собственную службы приема и исполнения заказов[1].

Для ресторанов, ценящих контроль над качеством и клиентским опытом, собственный сервис доставки оказывается более выгодным выбором. Начальные инвестиции окупаются за счет большей прибыльности и гибкости в управлении, обеспечивая прямое взаимодействие с клиентами и лучшее понимание их предпочтений.

Рост спроса на услуги доставки продовольственных товаров становится заметным явлением в современном мире, особенно в густонаселенных городах и урбанизированных областях. Этот рост спроса связан с изменением образа жизни современного общества. Городские жители все более загружены работой, учебой и другими обязанностями, что уменьшает время на покупки и приготовление еды. Популярность онлайн-шопинга и удобство использования мобильных приложений для заказа товаров также способствуют росту спроса на услуги доставки продуктов питания. Этот тренд в основном поддерживается молодыми семьями и активными рабочими людьми, которые предпочитают экономить время и силы, обращаясь к услугам доставки продуктов.

Рынок доставки продовольственных товаров представляет собой конкурентную среду, где действуют различные участники, такие как крупные сети супермаркетов, специализированные службы доставки, рестораны, кафе и небольшие местные бизнесы. Каждый из них стремится привлечь и удержать клиентов, предлагая различные услуги, ассортимент продуктов, ценовую политику и уровень сервиса. Крупные сети супермаркетов обычно имеют значительные ресурсы и брендовую известность, что делает их сильными участниками на рынке. Они могут предложить широкий ассортимент продуктов, конкурентные цены и интеграцию услуг доставки с уже существующей инфраструктурой магазинов.

Специализированные службы доставки, такие как такси-доставки еды или сервисы подписки на готовые блюда, обычно специализируются на определенных типах продуктов или услугах. Они могут предложить более гибкие условия доставки и персонализированный сервис, привлекая клиентов своей уникальной концепцией.

Стартапы и небольшие местные бизнесы часто ориентируются на узкую целевую аудиторию или на определенный сегмент рынка. Они могут выделяться инновационными подходами к услугам доставки, использованием местных продуктов или экологически устойчивыми практиками, чтобы привлечь внимание клиентов.

Конкурентный рынок доставки продовольственных товаров также характеризуется постоянным появлением новых участников и изменением стратегий существующих игроков. Технологические инновации, изменения в потребительских предпочтениях и внешние факторы, такие как экономические кризисы, могут значительно повлиять на конкуренцию на рынке. В такой среде важно проводить регулярный анализ конкурентов и адаптировать свои стратегии под изменяющиеся условия, чтобы оставаться конкурентоспособным.

Логистика и инфраструктура играют важную роль в сфере доставки продовольственных товаров. Эффективное управление маршрутами доставки, оптимизация складских операций и выбор подходящих транспортных средств являются ключевыми аспектами обеспечения быстрой и надежной доставки. Кроме того, современные информационные системы помогают отслеживать грузы, координировать доставки и повышать прозрачность процессов. Анализ инфраструктуры города также важен для определения оптимальных маршрутов и улучшения эффективности доставки. Все это содействует повышению конкурентоспособности компаний на рынке доставки продовольственных товаров.

## 1.2. Постановка задачи

***Цель проекта:*** Создание мобильного приложения, которое обеспечит удобную и эффективную доставку продовольственных товаров для пользователей, а также инструменты управления заказами для администраторов и курьеров.

***Функциональные требования для пользователя:***

1. ***Регистрация и аутентификация:*** Пользователи могут создать учетную запись, войти в систему с помощью электронной почты или номера телефона.
2. ***Просмотр ассортимента*:** Пользователи могут просматривать доступные продукты с описанием, изображениями и ценами.
3. ***Оформление заказа:*** Пользователи могут добавлять продукты в корзину, выбирать параметры доставки (время и адрес), редактировать и подтверждать заказ.
4. ***Корзина:* Пользователь может добавлять свои заказы в корзину, там выбирать нужное количество порций и удалять то, что он передумал покупать.**
5. ***Изменение данных пользователя:*** Важно обеспечить данную функцию для удобного пользования при возможном изменении адреса и т.д.

***Функциональные требования для администратора:***

1. ***Управление ассортиментом:*** Администраторы могут добавлять, редактировать и удалять продукты, устанавливать цены и параметры.
2. ***Управление заказами:*** Администраторы могут просматривать и управлять заказами, принимать заказы и отслеживать статусы доставок.
3. ***Управление пользователями:*** Администраторы могут просматривать данные пользователей, управлять учетными записями, решать вопросы поддержки.
4. ***Просмотр маршрута:*** Администраторы смогут отслеживать свой путь от своей точки расположения до заказчика.

## 1.3. Анализ существующих решений

Для анализа сервисов со схожим функционалом было выбрано два успешных существующих решения – Yandex.Food и Shef.

***Яндекс.Еда (Yandex.Food):*** представляет собой популярный сервис доставки еды, предоставляемый компанией Яндекс. Он предлагает широкий выбор ресторанов и кафе для заказа блюд различных кухонь прямо через мобильное приложение. Этот сервис отличается следующими ключевыми особенностями:

* *Удобное приложение.* Мобильное приложение Yandex.Food имеет простой и интуитивно понятный интерфейс, богатый разнообразием и детализацией (рис. \_). Пользователи могут искать рестораны по различным категориям, таким как тип кухни, местоположение и популярность. Каждое заведение имеет свое меню в приложении, где пользователи могут просматривать блюда и добавлять их в корзину.
* *Широкий выбор.* Yandex.Food предлагает клиентам доступ к многочисленным ресторанам и кафе, представляющим различные кухни и культуры. Это обеспечивает разнообразие в выборе блюд и удовлетворяет различные вкусы клиентов.
* *Отслеживание.*  Пользователи предпочитают систему, которая позволяет им следить за своими заказами после их размещения. Поэтому включение этого - разумный шаг со стороны Yandex.Food.
* *Дополнительные возможности:* Яндекс.Еда использует разные приложения и сервисы для различных участников процесса доставки еды: Яндекс.Еда для пользователей, Яндекс.Про для курьеров и веб-портал Яндекса для ресторанов.

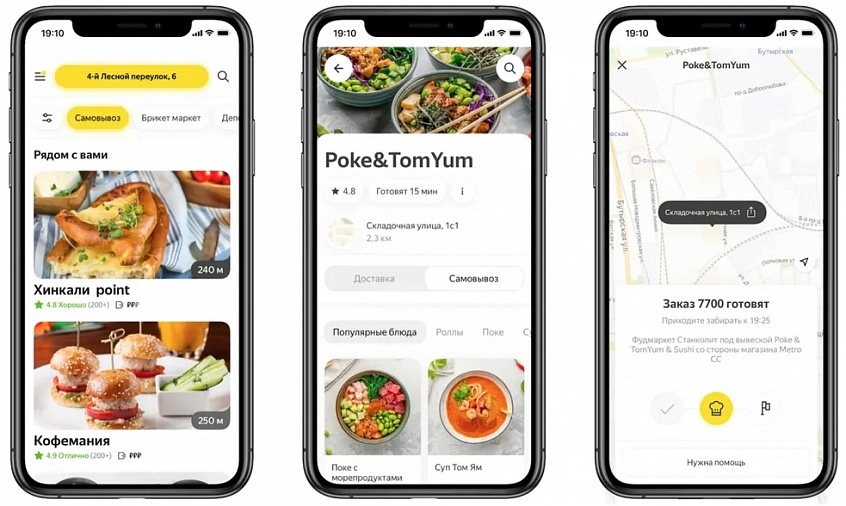


Рисунок 1.1 – Экраны приложения Yandex Food: главный экран, экран выбранного ресторана, экран отслеживания доставки

*Shef* : уникальный сервис доставки еды, который соединяет клиентов с местными поварами, готовящими домашние блюда. В отличие от многих других сервисов доставки еды, он делает акцент на свежеприготовленных, аутентичных блюдах от местных шефов.

Shef позволяет талантливым домашним поварам делиться своими кулинарными навыками с сообществом. Эти повара, называемые «шефами», готовят еду небольшими партиями, используя свежие и качественные ингредиенты. Такой подход гарантирует, что пища не производится массово, не транспортируется на большие расстояния и не замораживается, что позволяет сохранять высокое качество и вкус еды​. Платформа предлагает широкий ассортимент кухонь, отражающий культурное разнообразие поваров.

Веб-сайт и приложение Shef разработаны для обеспечения удобного пользовательского опыта. Клиенты могут просматривать меню, знакомиться с поварами и легко размещать заказы (рис. \_). Сервис также акцентирует внимание на прозрачности, предоставляя подробные описания ингредиентов и методов приготовления для каждого блюда​​.

Одним из ключевых преимуществ Shef является поддержка местных предпринимателей. Заказывая через Shef, клиенты напрямую поддерживают домашних поваров и владельцев малого бизнеса, помогая им зарабатывать на жизнь через их кулинарное увлечение. Эта модель приносит пользу не только поварам, но и способствует укреплению сообщества​.

Блюда, заказанные через Shef, доставляются свежими и готовыми к употреблению, что обеспечивает удобную альтернативу домашней готовке без компромиссов в качестве. Сервис особенно привлекателен для занятых людей и семей, которые хотят наслаждаться домашними блюдами без необходимости тратить время и усилия на их приготовление​.

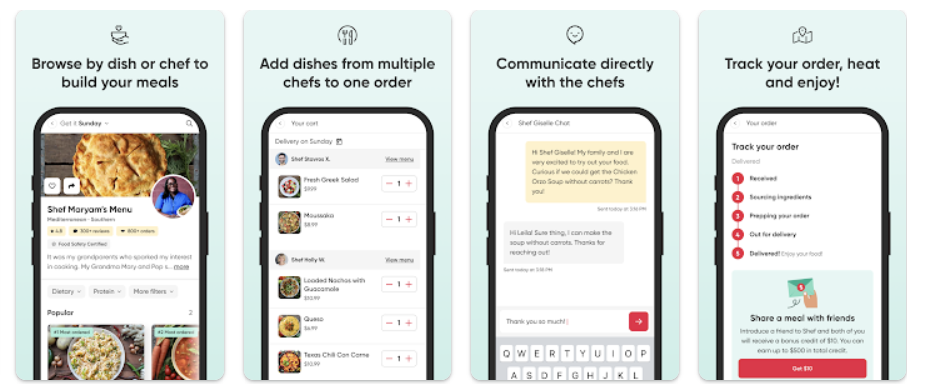


Рисунок 1.2 - Экраны мобильного приложения Shef слева направо: поиск блюд или поваров, добавление блюд в корзину, чат с поваром, отслеживание заказа

В ходе анализа сервисов доставки и ее управлением были выделены основные достаточные функции для успешного пользования мобильным приложением:

* Удобный и понятный интерфейс в приложении и для покупателя, и для продавца;
* Отдельные приложения для покупателей и продавцов, которые взаимодействуют между собой;
* Способность отслеживать маршрут заказа;
* Возможность менять свои данные аккаунта;

# Проектирование СТРУКТУРЫ ПРИЛОЖЕНИЯ

## 2.1 Выбор средств реализации

Перед началом разработки мобильного приложения для доставки еды было проведено исследование и анализ различных технологий, чтобы выбрать подходящие инструменты и платформы. В итоге было принято решение использовать Kotlin в среде разработки Android Studio с интеграцией Firebase. Этот выбор основан на следующих обоснованных причинах.

***Kotlin:* современный и безопасный язык программирования**, разработанный компанией JetBrains, был выбран в качестве основного языка программирования для разработки Android-приложения. Это язык, поддерживающий как объектно-ориентированное, так и функциональное программирование, обеспечивающий повышенную безопасность благодаря встроенной системе обработки null-значений, что значительно снижает вероятность возникновения ошибок, связанных с null-pointer exceptions (NPE).

Одним из ключевых факторов выбора Kotlin стала его полная совместимость с Java. Данное его свойство позволяет использовать существующие библиотеки и фреймворки на Java без необходимости полного переписывания кода, что ускоряет процесс разработки и облегчает интеграцию.

Kotlin является официально поддерживаемым языком программирования для разработки приложений под Android. Это гарантирует его долгосрочную поддержку и регулярные обновления от Google. Широкое сообщество разработчиков также предоставляет доступ к обширной документации, форумам и учебным материалам.

Android Studio, официальная IDE для разработки Android-приложений, была выбрана из-за своей высокой функциональности и интеграции с инструментами разработки. Среда предлагает все необходимые инструменты, такие как визуальный редактор макетов, анализатор APK и менеджер виртуальных устройств, значительно облегчающие процесс разработки и тестирования. Более того, Android Studio предоставляет полную поддержку Kotlin, включая автодополнение, рефакторинг кода и инструменты анализа.

*Firebase* была выбрана в качестве бэкенд-платформы из-за своего широкого набора функций и инструментов, которые идеально подходят для разработки и поддержки мобильных приложений. Firebase предлагает огромный спектр бесплатных сервисов, например аутентификация пользователей, база данных в реальном времени, хранение файлов и аналитика. Данные готовые решения позволяют сосредоточиться на разработке функционала приложения, не беспокоясь о инфраструктуре. Некоторые преимущества, предоставляемые в в Firebase:

***Аутентификация и безопасность***: Firebase Authentication позволяет легко интегрировать различные методы аутентификации, включая email, Google и социальные сети. Это обеспечивает высокий уровень безопасности и удобства для пользователей, что особенно важно для приложений, работающих с персональными данными.

***База данных в реальном времени и облачные функции****:* Firebase Realtime Database и Firestore предоставляют возможности для хранения и синхронизации данных в реальном времени, что критически важно для приложения доставки еды, где необходимо быстро обновлять статусы заказов и данные пользователей. Firebase Cloud Functions позволяет выполнять серверные операции, такие как обработка платежей и отправка уведомлений, без необходимости управления собственным сервером, что упрощает масштабирование и администрирование приложения.

***Аналитика и мониторинг:*** Firebase Analytics предоставляет детальную информацию о поведении пользователей, что помогает оптимизировать приложение и улучшать пользовательский опыт. Firebase Crashlytics помогает отслеживать и исправлять ошибки, повышая стабильность приложения.

Выбор Kotlin в качестве языка программирования, Android Studio как среды разработки и Firebase как облачной платформы вполне обоснован. Он был обусловлен их современными возможностями, высокой функциональностью, изобилием документации и общественной поддержкой. Эти факторы обеспечивают надежную, безопасную и эффективную разработку мобильного приложения предполагаемых объемов для сервиса доставки еды.

## 2.2 Анализ и проектирование архитектуры приложения

Для обеспечения модульности, повторного использования компонентов и легкости тестирования, было решено использовать архитектурный стиль MVVM (Model-View-ViewModel) при разработке мобильного приложения для доставки еды.

В современном мире требования к функциональности и возможностям программного обеспечения постоянно возрастают. В связи с этим увеличивается сложность разработки программного обеспечения и объемам данных, ужесточаются

требования и к архитектуре ПО. Паттерн MVVM позволяет улучшить архитектуру и качество программного обеспечения.

Он позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным.

Название шаблона отражает основные его компоненты: модель (Model), представление (View) и модель представления (ViewModel).

Данный паттерн был представлен Джоном Госсманом в 2005 году как модификация шаблона Presentation Model и был первоначально нацелен на разработку приложений в WPF. На данный момент паттерн вышел за пределы WPF и активно применяется в самых различных технологиях, в том числе при разработке под Android, iOS.

Модель – это объектное представление имеющихся данных. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления .

Нередко модель реализует интерфейсы INotifyPropertyChanged или INotifyCollectionChanged, которые позволяют уведомлять систему об изменениях свойств модели. Благодаря этому облегчается привязка к представлению, хотя опять же прямое взаимодействие между моделью и представлением отсутствует.

Представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Интерфейс необходимо проектировать так, чтобы он был чрезвычайно легковесным. Интерфейс должен отображать только необходимую информацию и механизмы взаимодействия. Представление MVVM должны разрабатываться с учетом целей, которые оно отображает. Любые интеллектуальные возможности должны встраиваться в другие места приложения.

Модель представления связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

Модель представления служит двум целям:

1) модель представления обеспечивает единственное местоположение для всех данных, необходимых представлению. Это вовсе не означает, что модель представления отвечает за получение действительных данных; взамен она обращается к соответствующему код, чтобы получить все данные вместе там, где они легко доступны. В результате между окнами и моделями представлений обычно сохраняется однозначное отношение, но архитектурные отличия существуют, и в каждом конкретном случае они могут варьироваться;

2) вторая задача модели представления касается ее действия в качестве контроллера для представления. Модель представления принимает указания от пользователя и вызывает код для выполнения подходящих действий .

Применение паттерна MVVM при разработке программного обеспечения дает следующие преимущества:

1) тестируемость приложений. Приложения, разработанные с использованием MVVM, обладают очень хорошим основанием для проведения модульного тестирования с целью проверки работы отдельных классов и методов;

2) меньшее количество кода. Объем кода, необходимого для управления представлением немного снижается при использовании MVVM, а это означает, что снижается риск допустить ошибки и уменьшается код для написания модульных тестов;

3) улучшенное проектирование приложений. Разработчики и дизайнеры могут самостоятельно работать над разными частями приложения. Есть возможность создать модель представления, которое предоставляет необходимые точки входа для связывания с представлением, которые в конечном представлении можно будет легко привязать. Это позволяет дизайнерам работать над внешним видом приложения, а программистам над бизнес-логикой приложения;

4) легкость понимания логики представления. MVVM предусматривает хорошо организованную и легкую для понимания конструкцию построения графического интерфейса за счет использования механизмов привязок, команд и шаблонов данных.

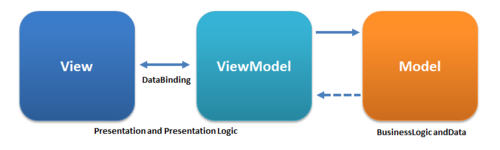


Рисунок 1.3 Схема паттерна MVVM

Для обеспечения удобства пользования и оптимизации рабочих процессов было принято решение разработать два отдельных мобильных приложения: одно для покупателей, другое для администраторов.

*Приложение для покупателей* предназначено для пользователей, которые хотят заказывать еду на дом или в офис. Оно должно включать в себя функции регистрации и аутентификации, позволяя пользователям создавать учетные записи и входить в систему. Покупатели могут просматривать доступное меню, искать нужное блюдо, добавлять их в корзину и оформлять заказы с указанием адреса доставки и выбора способа оплаты.

*Приложение для администраторов* предназначено для использования администраторами, поварами и курьерами, которые управляют заказами и доставкой. Оно включает в себя функции управления меню, позволяя администраторам и поварам добавлять новые блюда и удалять существующие. Администраторы и курьеры могут просматривать новые заказы, обновлять их статус и управлять процессом доставки. В приложении также реализованы функции аналитики и отчетности, предоставляющие возможность просмотра статистики заказов. Интерфейс приложения для администраторов адаптирован для профессионального использования, с возможностью быстрого выполнения необходимых действий и доступа к важной информации.

Разделение функционала на два отдельных приложения позволяет создать специализированные решения, которые лучше удовлетворяют потребности различных категорий пользователей, улучшает удобство и эффективность использования сервиса, обеспечивая высокое качество обслуживания клиентов и оптимизацию рабочих процессов сотрудников.

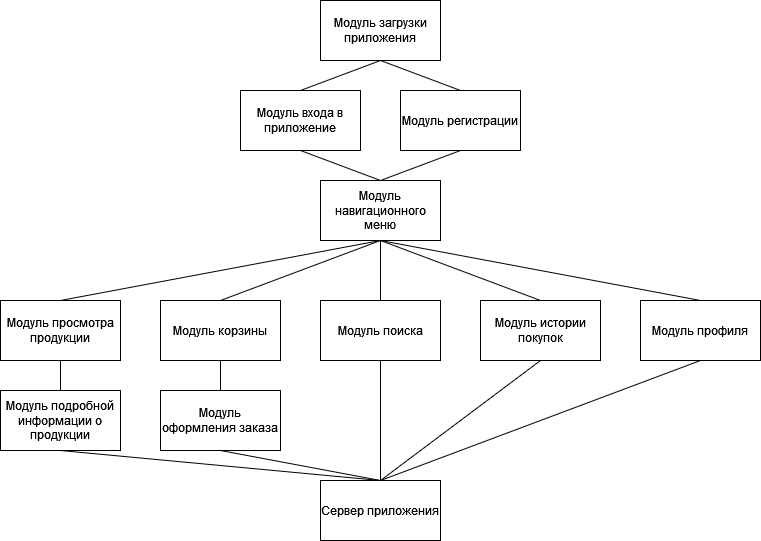


Рисунок 2.1 – Схема взаимодействия модулей в приложении для покупателей

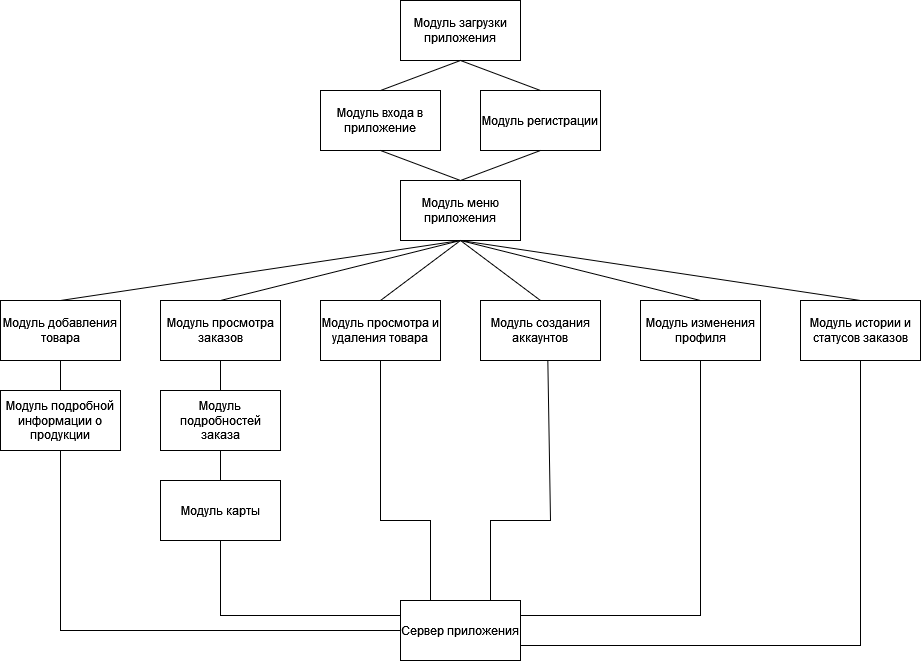


Рисунок 2.2 – Схема взаимодействия модулей в приложении для администраторов

## 2.3 Проектирование базы данных

Firebase Realtime Database — это облачная база данных, которая позволяет разработчикам хранить и синхронизировать данные между пользователями в режиме реального времени. Эта база данных управляется и поддерживается Google, что обеспечивает высокую надежность и масштабируемость.

Основной тип базы данных, используемой в Firebase Realtime Database — NoSQL. В отличие от реляционных баз данных, которые организованы в таблицы, NoSQL базы данных хранят данные в виде JSON-структур. Данные в Firebase Realtime Database представлены в виде огромного дерева JSON-объектов. Каждый узел этого дерева может содержать простой тип данных (например, строку, число) или более сложные вложенные объекты.

Структура базы данных в Firebase Realtime Database позволяет организовывать данные в виде иерархии, где каждое значение хранится по уникальному ключу. Эта структура идеально подходит для приложений, требующих частых и быстрых обновлений данных, таких как чаты, игровые состояния, и, конечно, системы заказа и доставки еды. Например, данные о пользователях, заказах, меню и статусах доставки могут быть организованы в виде различных веток JSON-дерева, что позволяет легко и эффективно извлекать необходимую информацию.

Одним из ключевых преимуществ Firebase Realtime Database является поддержка синхронизации данных в реальном времени. Это означает, что любое изменение в базе данных автоматически передается всем подключенным клиентам: если пользователь обновляет статус заказа или добавляет новый элемент в меню, это изменение моментально отражается у всех других пользователей приложения. Такая синхронизация достигается благодаря использованию технологий WebSocket, которые обеспечивают постоянное соединение между клиентами и сервером.

Безопасность данных в Firebase Realtime Database обеспечивается с помощью правил безопасности Firebase (Firebase Security Rules). Эти правила позволяют разработчикам точно контролировать доступ к данным на основе различных условий, таких как аутентификация пользователя и структура данных. Например, можно настроить правила так, чтобы только аутентифицированные пользователи могли просматривать или изменять свои собственные заказы, или чтобы только администраторы могли редактировать меню.

Еще одной важной функцией Firebase Realtime Database является поддержка оффлайн-режима. Клиенты могут продолжать работать с данными даже при отсутствии сетевого соединения. Все изменения, сделанные в оффлайн-режиме, будут автоматически синхронизированы с сервером, как только соединение восстановится. Это особенно полезно для мобильных приложений, где подключение к интернету может быть нестабильным.

Firebase Realtime Database также предоставляет удобные средства для масштабирования. С увеличением объема данных и количества пользователей, база данных автоматически масштабируется, обеспечивая стабильную работу приложения. Интеграция с другими сервисами Firebase, такими как Firebase Authentication и Firebase Cloud Messaging, позволяет создать комплексное решение для управления пользователями, аутентификацией и отправкой уведомлений.

Таким образом, Firebase Realtime Database является мощным и гибким инструментом для создания приложений, требующих надежного хранения и синхронизации данных в реальном времени. Использование NoSQL структуры, поддержка оффлайн-режима, гибкие правила безопасности и автоматическое масштабирование делают эту базу данных идеальным выбором для мобильных и веб-приложений, таких как системы доставки еды.

В Firebase Realtime Database узлы, такие как "users", "orders" и "menu", можно называть коллекциями. Хотя термин "коллекции" формально относится к Firestore, он также уместен и понятен при описании структур данных в Realtime Database. Эти узлы представляют собой логические группы данных, аналогичные таблицам в реляционных базах данных или коллекциям в других NoSQL базах данных. Firebase Realtime Database сохраняет уникальность объектов с помощью уникальных ключей, которые идентифицируют каждый узел в дереве JSON. Эти ключи могут быть либо сгенерированы автоматически Firebase, либо назначены разработчиком.

Описание документов коллекций:

Коллекция «Admin» содержит информацию о пользователях приложения для администраторов.

Содержимое коллекции «Admin»:

* **adminid** – уникальный идентификатор пользователя. Этот идентификатор автоматически сгенерирован Firebase для обеспечения уникальности каждого пользователя в базе данных;
* **address** – Содержит строку с адресом пользователя. Эта информация может использоваться для доставки или в других контекстах, где требуется знание адреса пользователя;
* **email** – содержит строку с адресом электронной почты пользователя. Электронная почта обычно используется для аутентификации и связи с пользователем.
* **name** – содержит строку с именем пользователя. Имя пользователя может отображаться в профиле, заказах и других частях приложения;
* **password** – Содержит строку с паролем пользователя.

Коллекция «Product» содержит информацию о блюдах:

* **productid** — это уникальный идентификатор продукта, который обеспечивает его уникальность в коллекции;
* **foodDescription** – поле, содержащее описание продукта. Описание предоставляет информацию о продукте, например, его особенности или происхождение;
* **foodImage** – поле, содержащее URL изображения продукта. Этот URL указывает на местоположение изображения в облачном хранилище Firebase, что позволяет отображать изображение продукта в приложении;
* **foodIngredient** – поле, содержащее список ингредиентов продукта. Это может быть полезно для пользователей, которые хотят знать состав продукта.
* **foodName** – поле, содержащее название продукта. Название используется для идентификации продукта и отображается пользователям;
* **foodPrice** – поле, содержащее цену продукта. Это числовое значение, представляющее стоимость продукта, и используется при оформлении заказов.

Коллекция «customer» содержит информацию о пользователях приложения для администраторов:

* **customerid** – уникальный идентификатор пользователя. Этот идентификатор автоматически сгенерирован Firebase для обеспечения уникальности каждого пользователя в базе данных;
* **address** – Содержит строку с адресом пользователя. Эта информация может использоваться для доставки или в других контекстах, где требуется знание адреса пользователя;
* **email** – содержит строку с адресом электронной почты пользователя. Электронная почта обычно используется для аутентификации и связи с пользователем.
* **name** – содержит строку с именем пользователя. Имя пользователя может отображаться в профиле, заказах и других частях приложения;
* **number** –поле, содержащее номер телефона пользователя;
* **password** – cодержит строку с паролем пользователя.

Узел «order» представляет собой коллекцию заказов:

* **orderid** – это уникальный идентификатор заказа, который обеспечивает его уникальность в коллекции. Под этим идентификатором хранится вся информация о конкретном заказе.
* **address** – поле, содержащее адрес доставки заказа.
* **currentTime** – поле, содержащее метку времени, представляющую момент создания заказа.
* **foodImages** – поле, предназначенное для хранения URL-адресов изображений продуктов, включенных в заказ.
* **foodNames** – поле, предназначенное для хранения названий продуктов, включенных в заказ.
* **foodPrices** – поле, предназначенное для хранения цен продуктов, включенных в заказ.
* **foodQuantities** – поле, предназначенное для хранения количества каждого продукта, включенного в заказ.
* **itemPushKey** – поле, содержащее уникальный ключ заказа.
* **orderAccepted** – поле, указывающее, был ли заказ принят.
* **paymentReceived** – поле, указывающее, была ли получена оплата за заказ;
* **phoneNumber** – Поле, содержащее номер телефона пользователя, связанного с заказом
* **totalPrice** – поле, содержащее общую стоимость заказа. Это значение представляет сумму, которую пользователь должен оплатить за заказ.
* **userName** – поле, содержащее имя пользователя, связанного с заказом, например, "user22". Имя пользователя может отображаться в интерфейсе администратора для идентификации заказа.
* **userUid** – поле, содержащее уникальный идентификатор пользователя в системе. Этот идентификатор используется для связи заказа с конкретным пользователем в базе данных.

Узел «BuyHistory» представляет собой коллекцию историй покупок.

* **historyid** – уникальный идентификатор записи в истории покупок, который обеспечивает её уникальность в коллекции.
* **foodPrices** – поле, предназначенное для хранения цен продуктов, включенных в заказ. На данный момент это поле пустое, но может содержать данные в виде JSON-объекта.
* **foodQuantities** – поле, предназначенное для хранения количества каждого продукта, включенного в заказ. На данный момент это поле пустое, но может содержать данные в виде JSON-объекта.
* **itemPushKey** – поле, содержащее уникальный ключ заказа. Оно совпадает с ключом узла и может использоваться для ссылок и операций с данным заказом.
* **orderAccepted** – поле, указывающее, был ли заказ принят.
* **paymentReceived** – поле, указывающее, была ли получена оплата за заказ.
* **userId** – id пользователя, сделавшего заказ

## 2.4 Проектирование интерфейса приложения

При первом запуске приложения пользователь видит экран с логотипом приложения (см. рисунок)



Рисунок – Пример экрана с логотипом

После логотипа появляется экран логина, где ему предлагается ввести свои учетные данные для входа в систему. Если у пользователя еще нет учетной записи, он может перейти на экран регистрации, используя соответствующую кнопку. На экране логина пользователь вводит свой email и пароль в предусмотренные для этого поля и нажимает кнопку входа для проверки введенных данных и авторизации в системе. Логотип приложения и текстовые инструкции помогают пользователю идентифицировать приложение и понять, что от него требуется. Кнопка для перехода на экран регистрации облегчает создание новой учетной записи, если у пользователя еще нет существующего аккаунта.

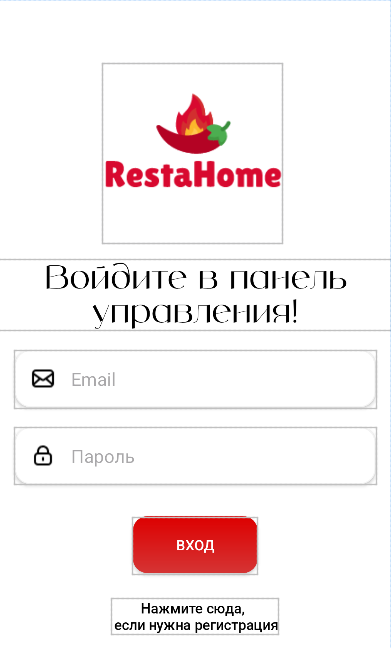


Рисунок – Пример экрана входа

На рисунке показан экран регистрации, который является ключевым для новых пользователей приложения. Экран регистрации позволяет пользователю создать новую учетную запись, вводя свои данные, такие как имя пользователя, адрес, email и пароль. Основной заголовок на экране сообщает пользователю о необходимости регистрации. После заполнения всех полей пользователь нажимает кнопку регистрации для завершения процесса создания аккаунта. В случае, если у пользователя уже есть учетная запись, он может перейти к экрану логина, используя специальную ссылку в нижней части экрана. Поле выбора местоположения предоставляет пользователю возможность указать свой текущий адрес с помощью выпадающего списка, что облегчает ввод данных и минимизирует ошибки. Поля для ввода адреса, электронной почты, имени пользователя и пароля снабжены подсказками и иконками, что улучшает удобство использования и делает интерфейс интуитивно понятным.

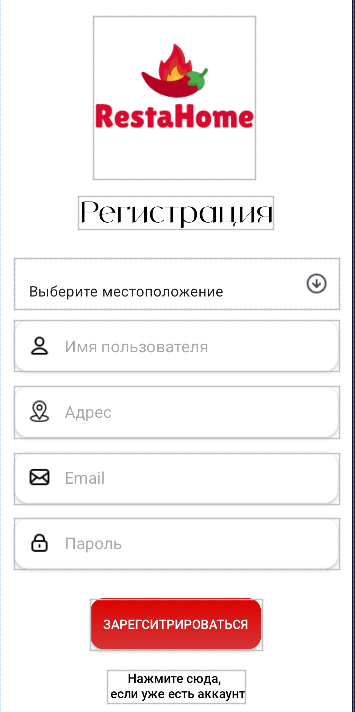


Рисунок – Пример экрана регистрации

На рисунке показан экран панели администратора. После входа в приложение пользователь видит главный экран панели администратора, где отображается основная информация и доступ к функциям управления.

В верхней части экрана расположено текстовое поле с надписью "Панель администратора". Под ним находится первая карточка с основными показателями: количеством заказов, завершёнными заказами и выручкой. В каждой секции отображается иконка, описательный текст и числовое значение.

Ниже первой карточки расположены шесть кнопок:

* **Добавить товар** — для перехода на экран добавления нового товара.
* **Все товары** — для просмотра всех товаров.
* **Профиль** — для управления профилем пользователя.
* **Добавить пользователя** — для создания нового аккаунта пользователя.
* **Отправка** — для перехода к управлению отправками заказов.
* **Выход** — для выхода из приложения.

Этот экран предоставляет пользователю все необходимые инструменты для управления магазином и заказами.

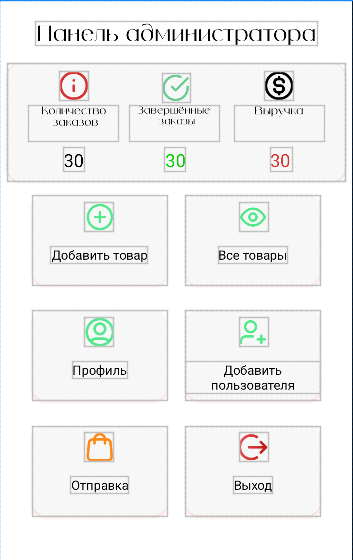


Рисунок – Пример панели управления

На рисунке показан экран добавления нового товара. Этот экран позволяет пользователю ввести необходимую информацию для добавления нового товара в систему.

Верхняя часть экрана содержит кнопку "Назад" в виде стрелки, расположенную в левом верхнем углу, что позволяет пользователю вернуться на предыдущий экран. Под этой кнопкой находится заголовок "Добавить товар", выполненный крупным шрифтом для обозначения текущего действия. Ниже заголовка расположены два текстовых поля для ввода названия и цены товара. Первое текстовое поле, подписанное "Введите название", позволяет пользователю ввести название нового товара. Второе текстовое поле, подписанное "Введите цену", предназначено для ввода стоимости товара. Под текстовыми полями находится кнопка "Выберите изображение" с иконкой плюса. Эта кнопка позволяет пользователю загрузить изображение товара. После выбора изображения, оно отображается в карточке ниже кнопки. Далее на экране есть текстовые поля для ввода краткого описания и состава товара. Поле для краткого описания подписано "Краткое описание", а поле для состава — "Состав". В нижней части экрана находится кнопка "Добавить", выполненная в красном цвете. Эта кнопка позволяет пользователю завершить процесс добавления нового товара, сохранив введённые данные.

Экран добавления товара обеспечивает удобный интерфейс для ввода и загрузки необходимой информации, что позволяет пользователю легко и быстро добавить новый товар в систему.

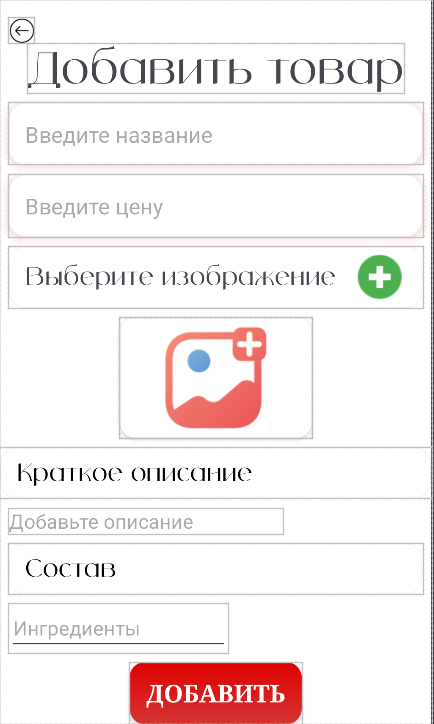


Рисунок – Пример экрана добавления товара

На рисунке показан экран профиля администратора. Этот экран позволяет администратору просматривать и изменять свою личную информацию.

В верхней части экрана находится кнопка "Назад" в виде стрелки, которая позволяет вернуться на предыдущий экран. Под этой кнопкой располагается заголовок "Профиль администратора", выполненный крупным шрифтом для обозначения текущего действия. Под заголовком размещен текст "Измените профиль", который служит напоминанием о возможности редактирования данных.

Ниже располагаются текстовые поля, организованные в виде линейных макетов ввода различных данных: имя, адрес, email, пароль. В нижней части экрана располагается кнопка "Сохранить", которая позволяет сохранить внесенные изменения. Кнопка выполнена в черном цвете с белым текстом и имеет достаточно крупный размер для удобного взаимодействия.

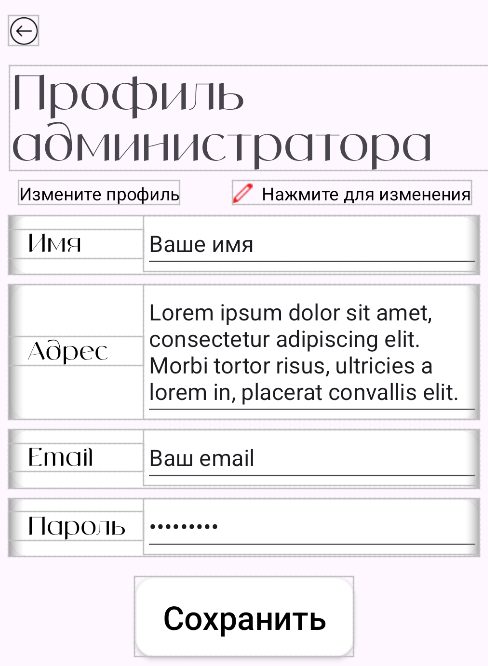


Рисунок – Пример экрана профиля

На рисунке показан экран с перечнем всех товаров. Этот экран предоставляет пользователю возможность просматривать список всех доступных товаров в приложении. В верхней части экрана находится кнопка "Назад" в виде стрелки, которая позволяет вернуться на предыдущий экран. Под этой кнопкой располагается заголовок "Все товары", выполненный крупным шрифтом для обозначения текущего действия. Основная часть экрана занята списком товаров, представленных в виде элементов RecyclerView. Каждый элемент списка включает в себя информацию о товаре, такую как название, изображение и цену. Список товаров позволяет пользователю просматривать, удалять, по необходимости, доступные товары, пролистывая их вниз.

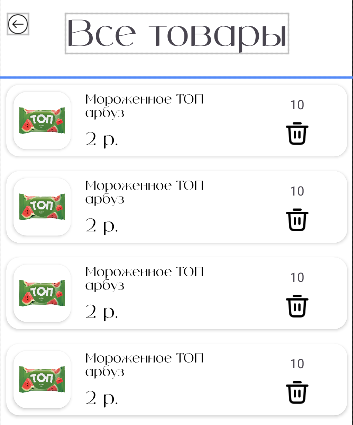


Рисунок – Пример экрана просмотра товаров

На рисунке показан экран деталей заказа. При открытии экрана пользователь видит информацию о конкретном заказе, такую как имя клиента, адрес доставки, телефон и сумма заказа. Также отображается список товаров, входящих в этот заказ.

Пользователь может вернуться на предыдущий экран, нажав на кнопку "Назад". Информация о заказе представлена в виде нескольких текстовых полей, сгруппированных в соответствующие секции.

Кнопка "Назад" расположена в верхнем левом углу и позволяет вернуться на предыдущий экран. Секция с информацией о клиенте содержит текстовые поля для отображения имени, адреса и телефона клиента, оформленные в виде горизонтального LinearLayout с текстовым лейблом и значением. Секция с информацией о заказе отображает общую сумму заказа, выделенную цветом для акцента.

Под секцией с информацией о клиенте расположена кнопка "Маршрут", которая позволяет построить маршрут до адреса доставки. Список товаров отображается с помощью RecyclerView и содержит все товары, включенные в заказ.



Рисунок – Пример экрана подробностей заказа

На рисунке показан экран ожидающих заказов. При открытии экрана пользователь видит список всех заказов, которые ожидают обработки. Здесь можно принять и закрыть заказ.

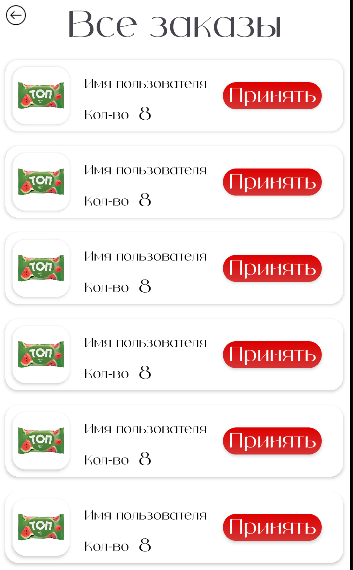


Рисунок – Пример экрана всех заказов

На рисунке показан экран статуса доставки. При открытии экрана пользователь видит список всех заказов, находящихся в процессе доставки. Основная часть экрана занята списком заказов, находящихся в процессе доставки или уже доставлены. Список занимает всю доступную высоту экрана под заголовком.

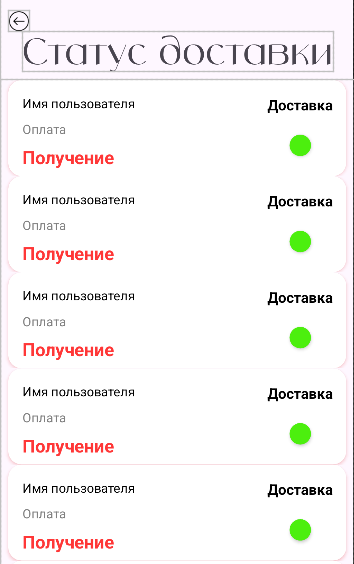


Рисунок – Пример экрана статусов доставки заказов

Далее перейдём к приложению для покупателей. На рисунке показан пример начального экрана приложения для пользователей.



Рисунок – Пример стартового экрана покупательского приложения

На рисунке показан экран входа пользователя. При открытии экрана пользователь видит логотип приложения. Ниже размещена форма для ввода учетных данных, включающая поля для ввода электронной почты или номера телефона и пароля. Пользователь может ввести свои учетные данные в соответствующие поля и нажать кнопку "Вход" для аутентификации. В случае, если у пользователя нет аккаунта, он может перейти на экран регистрации, нажав на соответствующую ссылку под кнопкой "Вход". Этот экран похож на аналогичный экран в приложении для администраторов, где также предоставляется возможность входа и регистрации.

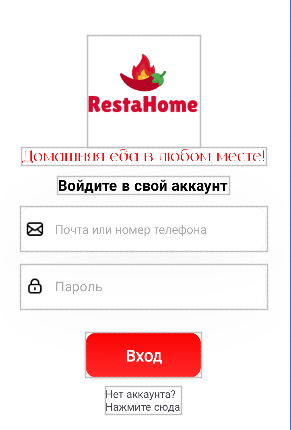


Рисунок – Пример экрана входа в приложение для покупателей

На рисунке показан экран регистрации пользователя, где размещена форма для ввода данных, включающая поля для имени пользователя, электронной почты или номера телефона и пароля. Пользователь может заполнить эти поля и нажать кнопку "Зарегистрироваться" для создания аккаунта. Если у пользователя уже есть аккаунт, он может перейти на экран входа, нажав на соответствующую ссылку под кнопкой "Зарегистрироваться". Этот экран похож на аналогичный экран в приложении для администраторов, где также предоставляется возможность ввода данных для регистрации и перехода на экран входа.

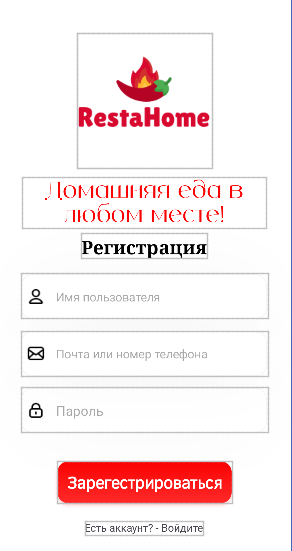


Рисунок – Пример экрана регистрации покупателей

На рисунке показан главный экран приложения. В верхней части экрана располагается карусель изображений, которая автоматически сменяет рекламные изображения через определенные промежутки времени. Ниже карусели находится раздел с заголовком "Популярное", который отображает список популярных блюд. Пользователь может нажать на кнопку "Полное меню", чтобы перейти к просмотру полного списка блюд. Этот экран позволяет пользователю ознакомиться с наиболее популярными предложениями и быстро получить доступ к основному меню.

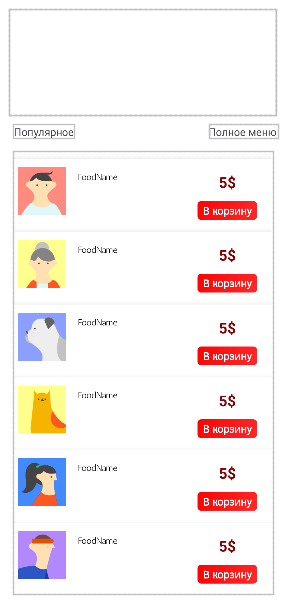


Рисунок – Пример главного экрана покупательского приложения

На рисунке показан экран корзины приложения. В верхней части экрана располагается заголовок "Корзина", под которым находится список добавленных в корзину товаров. Этот список представлен в виде RecyclerView. Под списком товаров находится кнопка "Заказать", при нажатии на которую пользователь может перейти к оформлению заказа. Этот экран позволяет пользователю просмотреть и управлять товарами в корзине перед оформлением заказа.

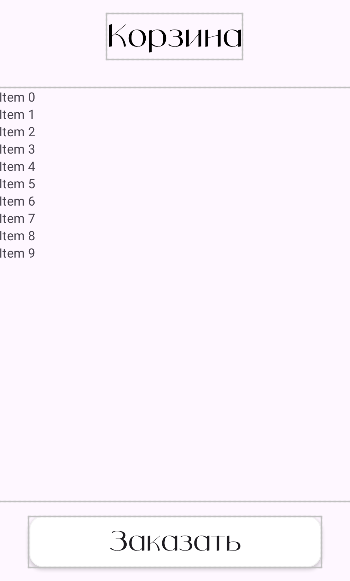


Рисунок – Пример экрана корзины

На рисунке показан экран истории покупок. Этот экран отображает список недавних покупок пользователя в приложении. В верхней части экрана находится заголовок "Недавние покупки". Каждая карточка содержит изображение продукта, его название, цену и статус заказа. Экран позволяет пользователю видеть и управлять своими недавними покупками, предоставляя возможность быстро оценить информацию о каждой покупке, включая название, цену и статус.

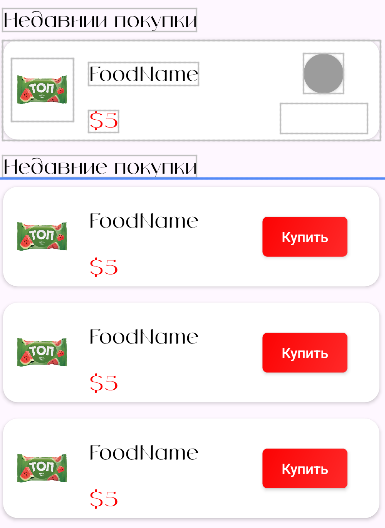


Рисунок – Пример экрана истории покупок

На рисунке показан экран поиска товаров. В верхней части экрана расположена строка поиска с заголовком "Поиск", где пользователь может ввести название товара, который он хочет найти. Ниже строки поиска находится список товаров, который отображается в виде RecyclerView.

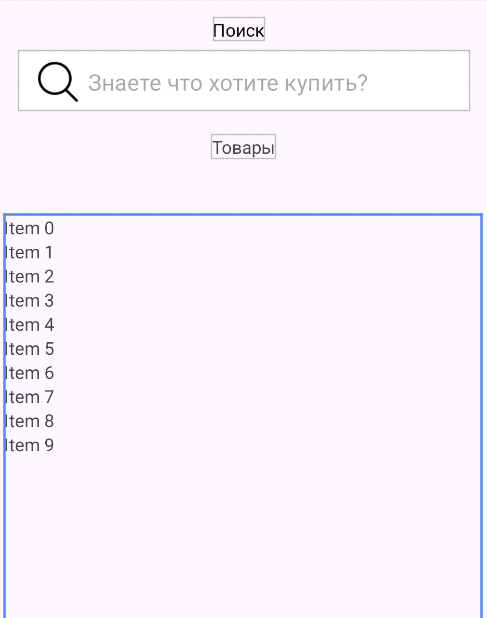


Рисунок – Пример экрана поиска

На рисунке показан экран профиля пользователя. Под ней расположены поля для ввода данных пользователя: имя, адрес, email и телефон. Эти поля оформлены в виде текстовых полей с фиксированной информацией, которую можно изменить после нажатия на кнопку "Изменить". В нижней части экрана находится кнопка "Сохранить", позволяющая сохранить изменения, внесенные в профиль.

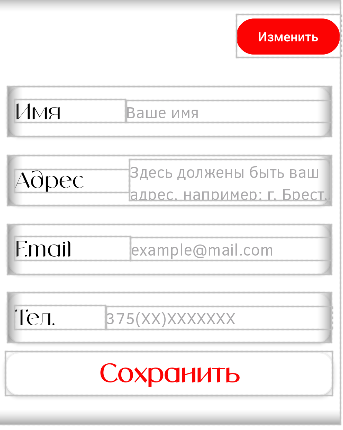


Рисунок – Пример экрана профиля

На рисунке показан экран деталей товара. В верхней части экрана находится кнопка для возврата, представленная в виде изображения стрелки. Ниже располагается название товара, оформленное в виде текста. Далее следует карточка с изображением товара, после которой идет краткое описание товара, представленное текстом. Под описанием находится раздел "Состав", где перечислены ингредиенты товара. В нижней части экрана размещена кнопка "Добавить в корзину", позволяющая пользователю добавить товар в корзину для последующей покупки.

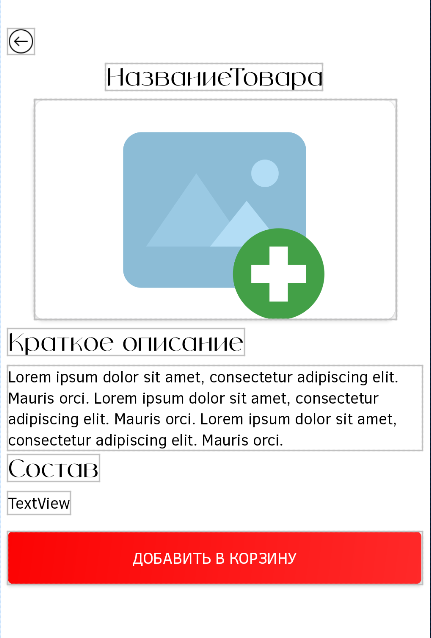


Рисунок – Пример экрана подробностей о товаре

Экран на рисунке представляет собой интерфейс для оформления заказа. Он состоит из нескольких основных элементов. Здесь расположены поля для ввода личных данных пользователя: имени, адреса и номера телефона. Эти поля оформлены в виде горизонтальных секций с метками, объясняющими, какую информацию необходимо ввести. Они заполняются автоматически, если пользователь уже их заполнил на экране профиля. Раздел "Оплата" представлен иконкой, указывающей на способ оплаты, а ниже находится поле для отображения суммы заказа, которое не позволяет вводить данные. В нижней части экрана расположена большая кнопка "Оформить заказ". В целом, этот экран обеспечивает удобный интерфейс для ввода необходимой информации и завершения процесса оформления заказа.

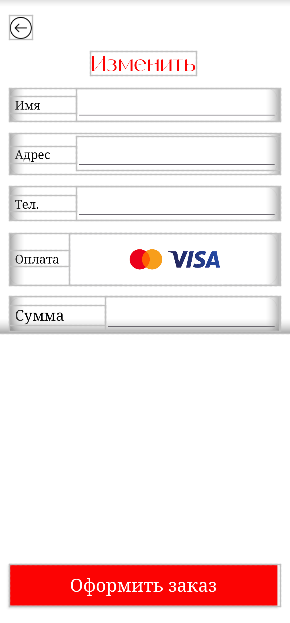


Рисунок – Пример экрана оформления заказа

На рисунке изображен экран с полной историей покупок пользователя.

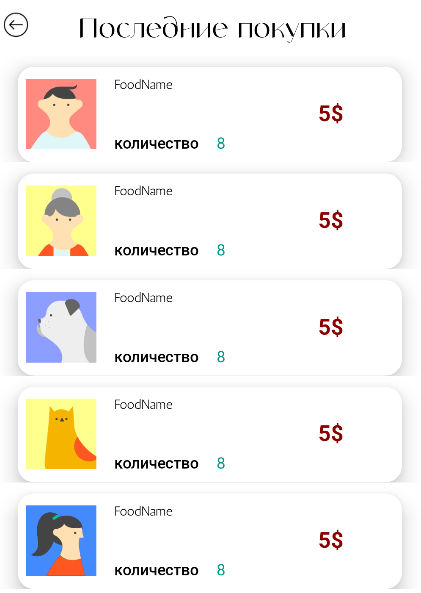
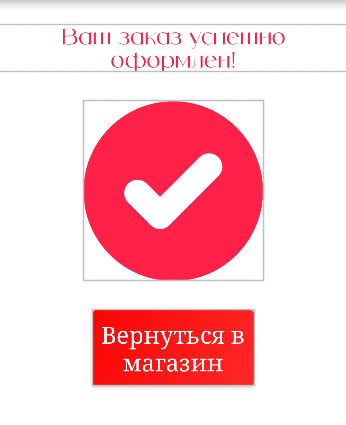


Рисунок – Пример экрана полной истории покупок

Экран на рисунке представляет собой интерфейс поздравления с успешным оформлением заказа. Он включает несколько основных элементов. В верхней части экрана располагается текстовое сообщение "Ваш заказ успешно оформлен!", выделенное крупным красным шрифтом и центрированное, что привлекает внимание пользователя и подтверждает успешное завершение процесса заказа. Под текстовым сообщением находится изображение, представляющее успешное завершение действия. Ниже изображения располагается кнопка "Вернуться в магазин". В целом, этот экран служит для уведомления пользователя об успешном завершении заказа и предоставляет ему возможность вернуться в магазин для продолжения покупок.

  
Рисунок – Пример экрана успешного оформления заказа

1. РЕАЛИЗАЦИЯ СИСТЕМЫ

## 3.1 Выбор средств реализаций

Для реализации дипломного проекта использовался ноутбук на базе процессора AMD Ryzen 5 2500U с 8 гигабайтами оперативной памяти. В качестве среды разработки была выбрана Android Studio Jellyfish 2023. Для создания фронтэнда в Android Studio называется **XML Layout Editor**. Этот редактор позволяет разработчикам визуально и программно создавать и настраивать пользовательские интерфейсы (UI) для приложений Android. В Android Studio также доступен **Design Editor**, который предоставляет возможность работать в режиме WYSIWYG (What You See Is What You Get), позволяя перетаскивать UI-компоненты на экран и изменять их свойства. Эти инструменты в Android Studio упрощают процесс создания и настройки пользовательских интерфейсов, делая его более интуитивно понятным и визуально наглядным.

Для обеспечения функциональности приложения доставки были использованы чистый Kotlin и несколько библиотек для взаимодействия с Firebase. Эти библиотеки включают в себя:

* **Firebase** предоставляет множество инструментов и сервисов для разработки мобильных и веб-приложений, включая аутентификацию пользователей, базы данных в реальном времени, облачное хранение и многое другое. Использование этих библиотек позволило легко интегрировать данные функции в приложение доставки, обеспечив надежную и эффективную работу с данными и пользователями.
* **FirebaseAuth** используется для управления процессом регистрации и входа пользователей в приложение. Эта библиотека поддерживает различные методы аутентификации, включая электронную почту и пароль, а также сторонние провайдеры, такие как Google, Facebook и другие.
* **FirebaseDatabase** предоставляет возможность работы с базой данных в реальном времени. Это позволяет мгновенно синхронизировать данные между клиентами, обеспечивая актуальность информации и улучшая взаимодействие пользователей с приложением.

В результате, использование чистого Kotlin и библиотек Firebase обеспечило создание надежного, масштабируемого и функционального приложения для доставки, которое может эффективно управлять пользователями и данными в реальном времени.

## 3.2 Реализация проекта

Разделение приложения для покупателей и продавцов еды на два отдельных приложения имеет несколько причин, каждая из которых связана с улучшением функциональности, удобства использования и управляемости:

**1. Удобство и простота для пользователей**

* Покупатели и продавцы имеют разные потребности и задачи. Разделение позволяет создать интерфейсы, специализированные для каждой группы пользователей, что делает использование приложения более интуитивным и удобным.
* Покупатели сосредотачиваются на поиске, заказе и оплате еды, в то время как продавцы занимаются управлением меню, заказами и логистикой. Отдельные приложения могут сократить количество ненужных элементов интерфейса для каждой группы пользователей.

**2. Улучшенная производительность**

* Разделение функциональности позволяет приложениям работать более эффективно, так как каждое приложение будет содержать только те функции, которые необходимы конкретной группе пользователей.
* Разделение трафика между двумя приложениями может уменьшить нагрузку на сервер и повысить общую производительность системы.

**3. Безопасность и управление данными**

* Данные продавцов и данные покупателей могут быть лучше защищены и управляемы в отдельных системах.
* Разделение позволяет легче контролировать доступ к различным уровням данных и функциональности, снижая риск несанкционированного доступа.

**4. Развитие и масштабируемость**

* Команды разработчиков могут независимо работать над обновлениями и улучшениями для каждого приложения. Это ускоряет процесс разработки и внедрения новых функций.
* Каждое приложение может масштабироваться в зависимости от своих уникальных требований и роста пользователей. Например, если количество продавцов растет быстрее, чем количество покупателей, можно сосредоточить ресурсы на расширении функциональности приложения для продавцов.

Структура приложений покупателя и продавца изображена на рисунке 3.1.

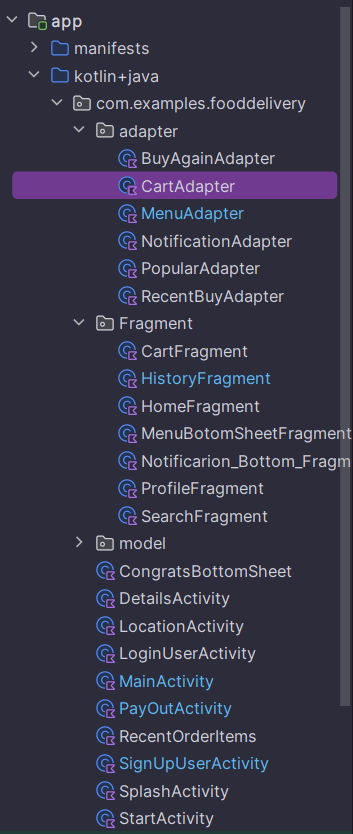
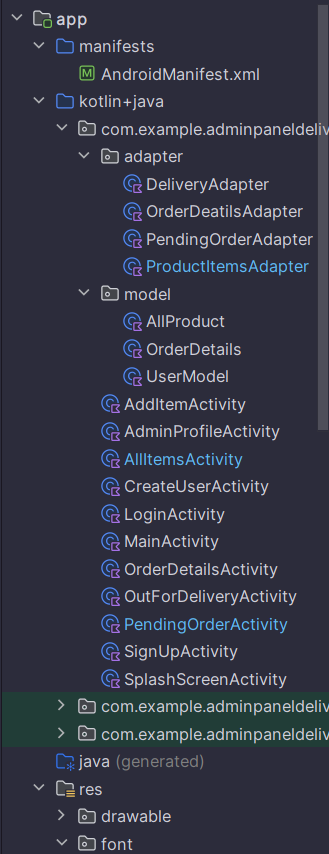
 

Рисунок 3.1 – Структура приложений покупателя и продавца

Приложения имеют некоторые сходства по структуре. Оба приложения имеют следующие похожие папки:

В папке «adapter» находятся файлы адаптеров. **Адаптеры** в Kotlin используются для подключения данных к компонентам пользовательского интерфейса (UI), такими как списки или сетки. Они играют ключевую роль в отображении данных в RecyclerView, ListView и других подобных компонентах. Основные функции адаптеров включают: с вязывание данных с представлением, переиспользование представлений, обработка событий.

В папке «models» находятся файлы моделей. **Модели** в Kotlin представляют данные приложения и логику управления этими данными. Они являются частью архитектурного паттерна Model-View-ViewModel. Они обеспечивают такие функции как: п**редставление данных, б**изнес-логику и взаимодействие с источниками данных.

И эксклюзивно для пользовательского приложения папка «Fragment» содержит файлы фрагментов для удобного интерфейса покупателей. **Фрагменты** — это модульные и переиспользуемые части интерфейса, которые можно включать в разные активности. Они позволяют создавать более гибкие и динамические пользовательские интерфейсы. В данном случае они используются для удобного и плавного использования меню.

**Для управляющего приложения:**

AddItemsActivity – класс, отвечающий за добавление товаров;

AdminProfileActivity – класс отвечающий за отображение изменение и просмотр данных профиля. На рисунке 3.2 изображен пример функции обновления данных.

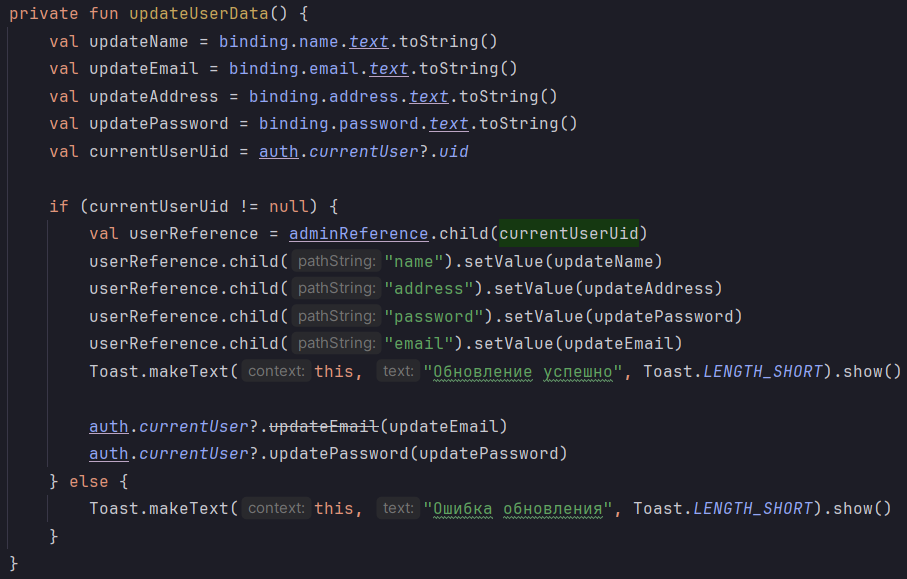


Рисунок 3.2 – Пример функции обновления данных профиля

AllItemActivity – класс, отвечающий за отображение всех блюд, принадлежавших данному администратору. На рисунке 3.3 изображен пример функции получения данных о блюдах из базы данных.



Рисунок 3.3 – Пример функции получения данных о блюдах из базы данных

LoginActivity – класс входа в приложение, используя email и пароль.

OrderDetailsActivity – класс, размещающий данные о заказе и отображающий маршрут отправки заказа.

OutForDeliveryActivity – класс, который отвечает за отображение статуса заверешённых заказов. Пример функции, размещающей данные, изображён на рисунке 3.4.

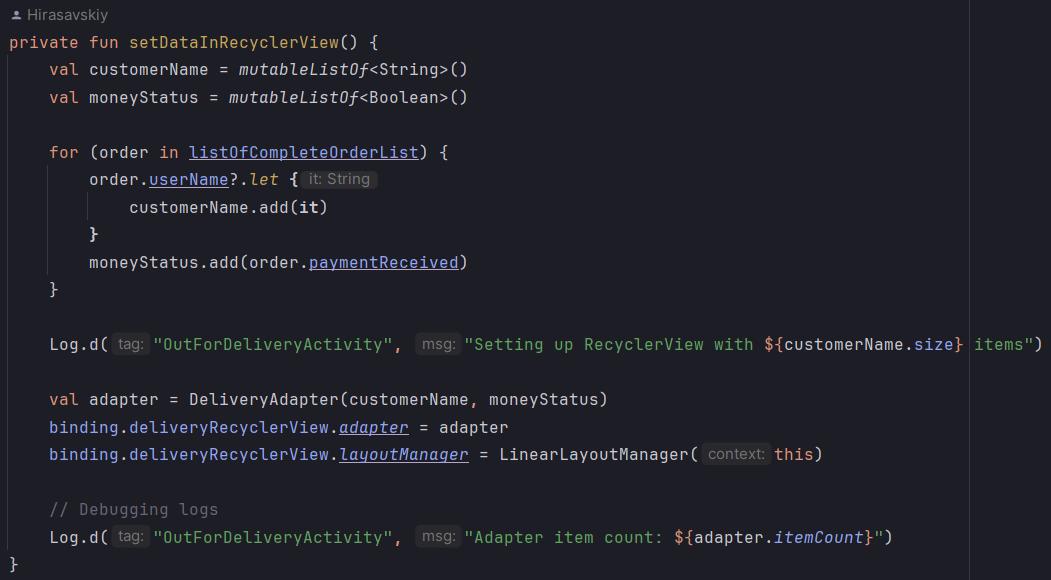


Рисунок 3.4 – Пример функции отображения завершённых заказов

PendingOrderActivity – класс, отображающий заказы в активности в порядке их получения.

SignUpActivity – класс, отвечающий за регистрацию в приложения для продавцов.

SplashScreenActivity – класс, отвечающий за активность стартового экрана.

**Для покупательского приложения:**

HomeFragment – класс активности главного экрана.

CartFragment – класс активности корзины.

SearchFragment – класс активности поиска. Пример функции поиска изображён на рисунке 3.5.

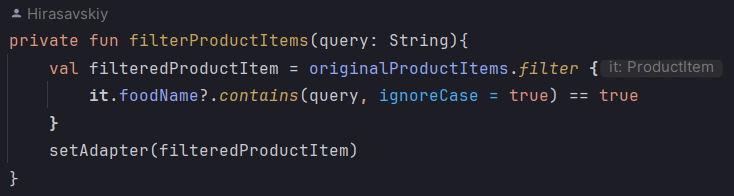


Рисунок 3.5– Пример функции поиска товара

MenuBottomSheetFragment – класс, отвечающий за отображение в актиности всех товаров в меню.

HistoryFragment – класс, отображающий историю покупок пользователя и последний заказ. Пример функции, устанавливающей данные о последнем купленном заказе показана на рисунке



Рисунок 3.6. – Пример функции, устанавливающей данные о последнем купленном заказе

CongratsBottomSheet – класс, отображающий уведомление об успешном оформлении заказа.

DetailsActivity – класс, отвечающий за отображение всех подробностей о продукте и за возможность добавить продукт в корзину. Пример функции, добавляющей заказ в корзину пользователя показа на рисунке 3.7.

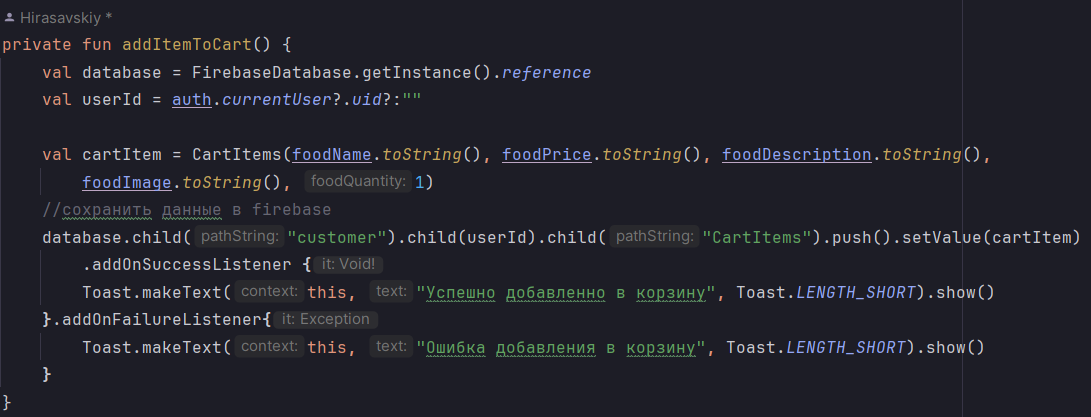


Рисунок 3.7 – Пример функции, добавляющей заказ в корзину пользователя

LocationActivity – класс, позволяющий выбрать свой город на этапе регистрации. LoginUserActivity – класс, отвечающий за активность входа в аккаунт.

PayOutActivity – класс, отвечающий за активность оформления заказа и заполнения всех данных о заказе вроде адреса и номера телефона. Пример функции, сохраняющей данные о заказе изображен на рисунке 3.8.

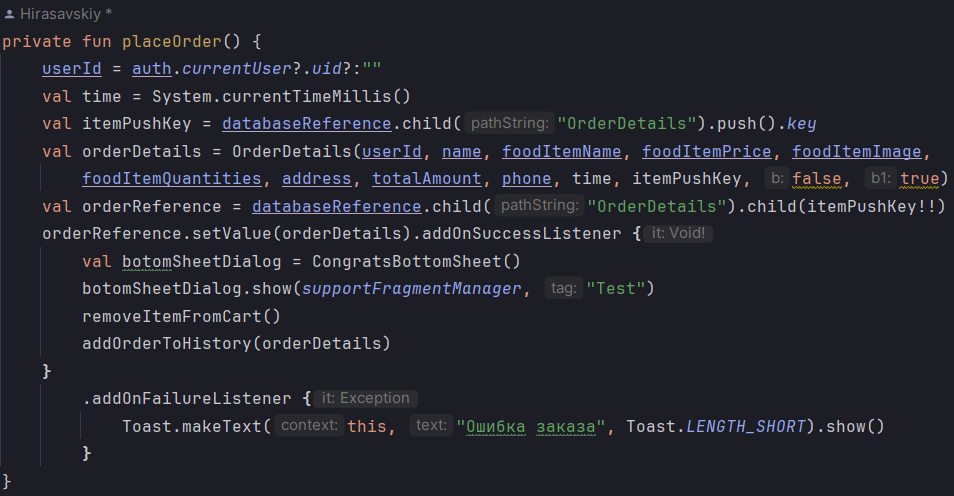


Рисунок 3.8 – Пример функции, сохраняющей данные о заказе

RecentOrderItems – класс, отвечающий за активность отображения подробных данных о последнем заказе.

SignUpUserActivity – класс, отвечающий за активность регистрации покупателя. Пример функции создания аккаунта покупателя показа на рисунке 3.9.

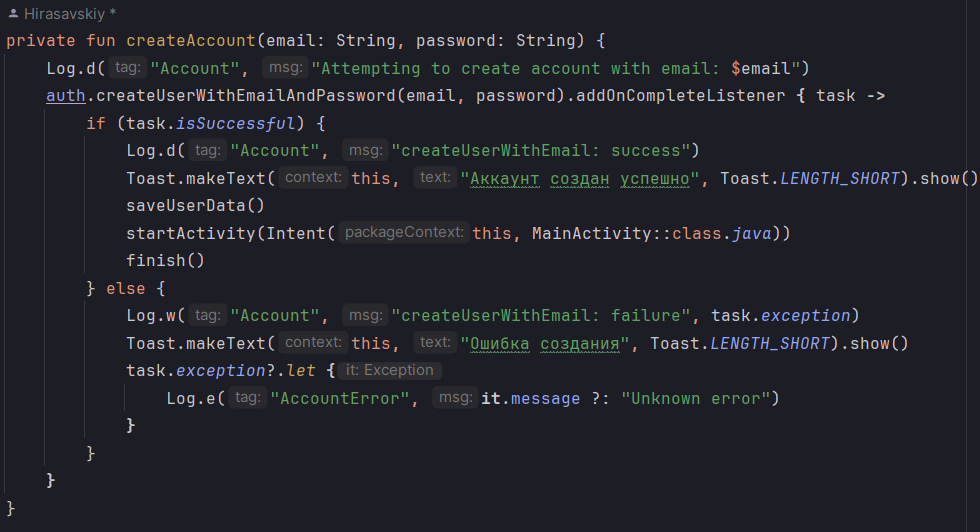


Рисунок 3.9 – Пример функции создания аккаунта покупателя

SplashActivity – класс, отвечающий за активность загрузочного экрана.

StartActivity – класс, отвечающий за активность стартового экрана.

# 4. Тестирование системы

# Заключение

Начало формы

# Список исользованных источников

1. ГОСТ 19.005-85. ЕСПД. Р-схемы алгоритмов и программ. Обозначения условные графические и правила выполнения.

2. ГОСТ 19.101-77. ЕСПД. Виды программ и программных документов.

3. ГОСТ 19.102-77. ЕСПД. Стадии разработки.

4. ГОСТ 19.103-77. ЕСПД. Обозначения программ и программных документов.

5. ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению.

6. ГОСТ 19.402-78. ЕСПД. Описание программы.

7. ГОСТ 19.504-79. Единая система программной документации ЕСПД. Руководство программиста. Требования к содержанию и оформлению.

8. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.

9. ГОСТ 2.105-95. Единая система конструкторской документации (ЕСКД). Общие требования к текстовым документам.

10. ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

1. https://restospace.com/blog/articles/article/organizaciia-dostavka-edy-v-2024-godu-effektivnye-reseniia-dlia-restoranov#articleHeader\_\_1