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

Кафедра Прикладная математика и информатика
(наименование)

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

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

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка мобильного приложения управления заказами малого предприятия»

Обучающийся К. И. Шутько _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.т.н., доцент, Н.В. Хрипунов
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.п.н., доцент, С.А. Гудкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Тема выпускной работы «Разработка мобильного приложения управления заказами малого предприятия».

Работа выполнена в объеме 52 страниц, включая 41 иллюстрацию и 1 таблицу. Структура работы включает введение, три главы, заключение, список используемой литературы и приложение.

Тема работы связана с разработкой мобильного приложения, которое помогает эффективно реализовать полный производственный цикл изделия и упорядочено хранит всю необходимую информацию по текущим и выполненным заказам. Работа обосновывает актуальность данной темы в контексте увеличения эффективности производства малого предприятия.

Целью работы является разработка мобильного приложения, позволяющего исключить проблему потери важной информации о заказе и неопределенности в этапе производственного цикла изделия.

В результате работы было разработано и успешно протестировано приложение, которое позволяет работникам всегда иметь доступ к необходимой информации о заказах, а также без препятствий отслеживать этап работы каждого изделия.

В конце работы представлено функциональное мобильное приложение и проведено тестирование, которое показывает полную работоспособность всех основных функций приложения.

Abstract

The topic of the graduation work is “The development of a mobile application for managing orders for a small enterprise.”

The work is completed in a volume of 52 pages, including 41 illustrations and 1 table. The structure of the work includes an introduction, three chapters, a conclusion, a list of references and a diploma supplements.

The topic of the work is related to the development of a mobile application that helps to effectively implement the full production cycle of a product and stores all the necessary information on current and completed orders in an orderly manner. The work substantiates the relevance of this topic in the context of increasing the production efficiency of a small enterprise.

The goal of the work is to develop a mobile application that eliminates the problem of losing important information about the orders and uncertainty at the stage of the production cycle.

As a result of the work, an application was developed and successfully tested, which allows employees to always have access to the necessary information about orders, as well as to easily track the stage of operation of each product.

At the end of the work, a functional mobile application is presented and testing is carried out, which shows the full functionality of all the main functions of the application.

Оглавление

Введение

С развитием технологий и ростом количества компаний, которые занимаются внедрением цифровых технологий в свою деятельность, малые предприятия все чаще сталкиваются с необходимостью использования эффективных инструментов для управления своим бизнесом. Это касается также и сферы управления заказами, где существует проблема утери чертежей, адреса и другой важной информации. Здесь важны четкий контроль над этапами выполнения заказа и удобный доступ ко всей имеющейся информации по заказу. Разработка мобильного приложения для управления заказами поможет исправить проблему и повысить эффективность работы малого предприятия.

Актуальность данной дипломной работы заключается в разработке мобильного приложения управления заказами, которое будет способствовать повышению эффективности работы малого предприятия. Приложение предназначено для использования на мобильных устройствах и позволит сотрудникам предприятия легко взаимодействовать с заказами и иметь всю необходимую информацию без необходимости использования бумажных записей или других приложений, как например блокнот, в котором нет необходимого инструментария и систем удобной записи.

Объектом исследования является процесс производства малого предприятия.

Предметом исследования является мобильное приложение управления заказами малого предприятия.

Целью дипломной работы является разработка мобильного приложения управления заказами малого предприятия.

Для достижения цели дипломной работы необходимо решить следующие задачи:

- разработка архитектуры и функционала приложения;
- создание функционала для авторизации, регистрации пользователями;

– создание интерфейса для работников предприятия с возможностью управления заказами;
– интеграция базы данных для хранения информации о работниках и заказах;
– запуск разработанного веб-приложения для проверки его функциональности и работоспособности.

Практическое значение данной работы заключается в разработке мобильного приложения для упрощения работы предприятия и управления заказами.

Дипломная работа состоит из введения, трех глав, заключения и списка используемой литературы.

Глава 1. Сущность объекта исследования

1.1 Краткая характеристика объекта исследования

ИП “ToffStone” является предприятием, специализирующимся на изготовлении изделий из искусственного камня. Предприятие ориентированно на предоставление услуг по изготовлению и установке различных кухонных изделий по самостоятельному или предоставленному чертежу. Предприятие широко известно на рынке благодаря высокому качеству предоставляемых услуг и профессиональности мастеров.

Структура предприятия представляет собой следующие должности:

- начальник цеха: организовывает работу предприятия и руководит производством;
- замерщики: выполняют работу по замеру заказанных изделий на адресах заказчика;
- мастера по камню: выполняют основную работу по изготовлению изделий по указанным чертежам;
- мастера по установке: осуществляют установку готовых изделий на адресе заказчика.

Структура данного предприятия приведена на рисунке 1.

Рисунок 1 – Организационная структура предприятия

В целях оптимизации бизнес-процессов малого предприятия ИП “ToffStone”, было принято решение по разработке приложения, которое позволит упорядочить, улучшить и эффективно отслеживать полный производственный цикл изделия.

1.2 Постановка технического задания на разработку мобильного приложения

В рамках работы необходимо разработать мобильное приложение для малого предприятия, которое будет предназначено для хранения данных заказов клиентов. Для роли “Начальник цеха” будет включена вкладка создания заказов и заполнения данных по заказу, для ролей работников реализовать возможность изменения статуса заказа.

Функциональные требования:

- реализация вкладок для просмотра списка заказов на различных стадиях, редактирование и изменение статуса заказа;
- регистрация и авторизация работников;
- адаптивный интерфейс для развертывания приложения на различных мобильных устройствах.

Нефункциональные требования:

- безопасность данных работников предприятия;
- производительность приложения при создании заказов и загрузке информации;
- интерфейс, соответствующий принципам UI/UX дизайна.

Выводы по главе 1

В первой главе был осуществлён краткий обзор состояния предприятия, в ходе которого были определены ключевые проблемы, требующие решения через реализацию мобильного приложения. В конце главы были определены цели для разработки данного мобильного приложения.

Глава 2. Проектирование архитектуры и разработка мобильного приложения

2.1 Выбор технологий и инструментов для разработки

Технология разработки программных приложений включает в себя различные процессы и методы, которые позволяют разработчикам создавать качественное программное обеспечение, удовлетворяющее потребности пользователей [8]. Для разработки мобильного приложения существует множество технологий, и каждая из них имеет свои преимущества и недостатки. Ниже приведен обзор основных средств и технологий для разработки.

Нативная разработка.

Такой способ разработки позволяет создавать приложения, оптимизированные под операционные системы Android и iOS, используя языки программирования, такие как Java и Kotlin для платформы Android, Swift и Objective-C для операционной системы iOS. Этот способ позволяет достичь максимальной производительности и улучшения взаимодействия с устройством, что существенно улучшает скорость и стабильность самого приложения.

Кроссплатформенные решения

Данный способ предоставляет возможность разработки приложений для разных платформ с использованием одной кодовой базы. Такой способ применяется для экономии времени и ресурсов при разработке, упрощения процесса поддержки и обновления приложения. Этот способ также имеет свои недостатки, такие как более низкая производительность по сравнению с нативными приложениями и ограничения в доступе к некоторым функциям устройства.

Веб-приложения

При веб-разработке создаются приложения, которые работают через веб-браузер и предоставляют UI/UX, схожий с нативными приложениями. Такие приложения подходят для обеспечения доступности приложений без необходимости их установки. При таком способе ограничен доступ к функциям самого устройства, а также их работоспособность зависит от интернет-соединения.

Исходя из сопоставления преимуществ и недостатков каждого подхода, было решено выбрать нативный способ разработки, так как в таком случае мы имеем полный доступ к функциям операционной системы, функционалу устройства, интеграцию с операционной системой и оптимизацию производительности, прирост скорости работы. Разработка будет выполнятся для платформы Android, поэтому в качестве языка программирования был выбран Kotlin, так как он является основным языком для разработки приложений.

Kotlin — это статически типизированный, объектно-ориентированный язык программирования, разработанный компанией JetBrains.Он был создан с целью улучшения языка Java и решения некоторых его проблем [3].

Kotlin требует от команды разработки большой компетенции и навыков, что в разы удорожает процесс разработки приложения, однако пропорционально увеличивает его качество [14].

Решение о выборе было сделано в пользу операционной системы Android в качестве платформы для разработки по следующим причинам:

- android является гибкой системой, которая поддерживает различные устройства. Это позволяет адаптировать приложение под различные экраны и устройства;
- на платформе Android существует обширный набор инструментов для разработки, такие как Android SDK, куда входят средства для управления навигацией, реализации локальной базы данных и многое другое. Эти средства позволяют упростить процесс разработки приложений;

– android предлагает открытую альтернативу разработке мобильных приложений. Без искусственных барьеров разработчики Android могут свободно писать приложения, которые в полной мере используют преимущества все более мощного мобильного оборудования, и распространять их на открытом рынке [20].

В качестве бэкенда и сервера приложения, была выбрана платформа Appwrite - открытая платформа бэкенда как сервиса BaaS[1], предназначенная для создания мобильных и web-приложений. Appwrite обладает необходимым функционалом которые нужны для реализации мобильного приложения: регистрация и аутентификация пользователей, создание базы данных, хранения файлов и многое другое.

2.2 Проектирование архитектуры приложения

При проектировании приложения основным этапом является выбор архитектуры. Выбор архитектуры зависит от того, какие конкретные задачи должно выполнять приложение.

После нескольких лет использования объектно-ориентированного программирования в 1990-х годах были разработаны паттерны проектирования. Эти шаблоны систематизировали и упорядочили общие подходы к решению конкретных общих задач программирования [17].

Паттерны — это повторяющиеся подходы и решения для различных задач, которые возникают при проектировании и разработке мобильных приложений. Паттерны помогают упростить процесс разработки, повысить качество и облегчить поддержку приложения в будущем [12].

Архитектура приложения — является структурой, которая определяет организацию системы. Она включает в себя компоненты, свойства и их взаимосвязи с другими компонентами программы. При разработке приложения, основными архитектурами являются MVC, MVVM и MVP.

[1] Backend-as-a-Service

MVC (Model-View-Controller):

– слой Model содержит данные и бизнес-логику приложения [11];
– слой View отображает данные модели пользователю и отправляет команды контроллеру [11];
– слой Controller обрабатывает ввод пользователя, изменяет модель и обновляет представление [11].

Этот паттерн упрощает разделение логики приложения и интерфейса, что облегчает тестирование и поддержку [11].

MVVM (Model-View-ViewModel):

– слой Model также содержит данные и бизнес-логику;
– слой View отображает визуальные элементы и пользовательский интерфейс;
– слой ViewModel действует как посредник между Model и View, содержит логику представления и обрабатывает команды для обновления Model.

MVVM использует двустороннее связывание данных для упрощения GUI-логики.

MVP (Model-View-Presenter):

– слой Model определяется так же, как и в MVC и MVVM;
– слой View отвечает за отображение данных и делегирует обработку пользовательских действий Presenter'у;
– слой Presenter содержит логику представления и работает напрямую с View для обновления пользовательского интерфейса.

MVP часто используется в приложениях, где необходим строгий контроль над пользовательским интерфейсом и его поведением.

Данные шаблоны проектирования помогают разработчикам организовать код модульным образом, для облегчения поддержки, тестирования и расширения проекта.

MVVM (Model-View-ViewModel) — это архитектурный паттерн, используемый в разработке программного обеспечения для создания

графических пользовательских интерфейсов. Он был разработан Microsoft для использования с технологиями WPF и Silverlight, но с тех пор стал популярным в различных средах разработки мобильных приложений [12]. Структура архитектуры MVVM представлен на рисунке 2.

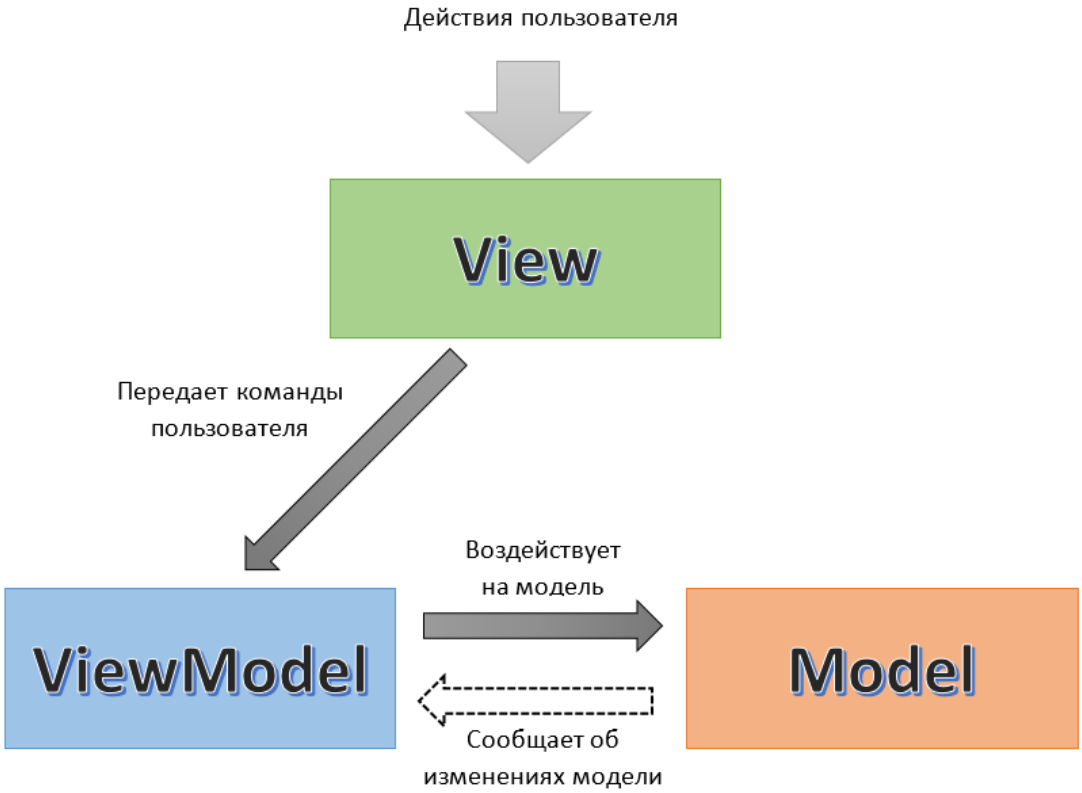

Рисунок 2 – Схема архитектуры MVVM

Вот как работает MVVM [12]:

– Пользователь выполняет действие на пользовательском интерфейсе [12];
– Представление обрабатывает это действие и отправляет запрос на ViewModel [12];
– ViewModel обрабатывает запрос, используя данные из модели, и возвращает результаты представлению [12];
– Представление обновляет себя на основе результатов ViewModel [12];

– Если изменения произошли в ViewModel, он обновляет модель [12];
– Если изменения произошли в модели, ViewModel получает уведомление об изменении и обновляет представление [12].

В данной работе была выбрана архитектура MVVM исходя из ряда преимуществ по сравнению с другими архитектурными шаблонами, такие как привязка данных, разделения проекта на модули и масштабируемость.

2.3 Проектирование модели данных

Инфологическая (семантическая) модель предметной области — информационная структура, которая отражает требования пользователей предметной области к представлению данных. При этом требования пользователей определяются в наиболее естественных для специалиста способах описания ввода, накопления, обработки информации, формирования выходных данных [10].

К семантическим моделям относится метод «сущность-связь». Этот метод был введен в практику в 1960-е годы в работах Брауна (A. P. G. Brown), Чена (P. Chen) и ряда других авторов.

Существуют три основные модели данных: сетевая модель, реляционная модель и модель набора сущностей. Модель набора сущностей, основанная на теории множеств, обеспечивает высокую степень независимости данных, она принимает более естественную точку зрения, согласно которой реальный мир состоит из сущностей и отношений между ними [18].

В данной работе будет использован метод разработки внешних модулей, а именно моделирование сущности-связи.

В метод «сущность-связь» входят следующие основные компоненты:

– сущность: это объект предметной области, информация о сущности записана и хранится в базе данных;
– атрибут: свойства сущности. Атрибут уникален для каждого объекта и имеет своё имя;

– связь: представляет собой ассоциации двух и более объектов предметной области, отражает зависимость между объектами;
– ключ: определяет уникальные идентификаторы для сущностей.

Каждая сущность в такой модели представляет множество однотипных экземпляров некоторого объекта предметной области и наделяется «атрибутами», описывающими свойства объектов, существенные в рамках решаемой задачи. Множество значений описательных атрибутов экземпляров сущностей является основным результатом выполнения пользовательских запросов к базе данных, при этом некоторые атрибуты могут выступать и в роли идентификаторов, с помощью которых производится выборка соответствующих экземпляров [4].

Для проектирования эффективной структуры базы данных можно выделить основные этапы [5]:

– анализ требований и бизнес-правил [5];
– анализ связей [5];
– организацию данных в таблицы [5];
– указание основных ограничений таблицы [5];
– нормализацию таблиц [5].

Концептуальная модель данных является важным аспектом построения модели базы данных. В ней продемонстрирована структура базы данных и взаимосвязи сущностей.

Концептуальная модель данных представлена на рисунке 3.

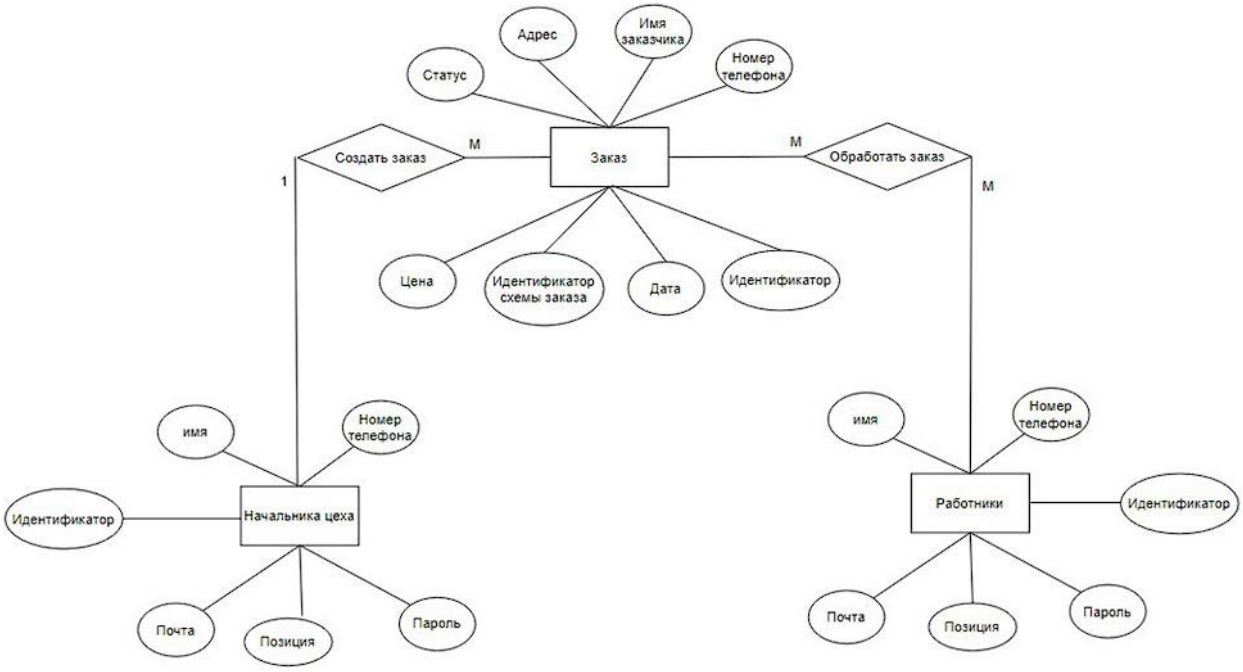

Рисунок 3 – Концептуальная модель данных

Логическая модель данных является более глубоким слоем модели данных, которая детально рассматривает сущности концептуальной модели, атрибуты и взаимосвязи между сущностями.

В логической модели данных описывается некоторый набор родовых понятий и признаков, которыми должны обладать все СУБД, входящие в определенный класс (множество), и управляемые ими базы данных, если они основываются на этой модели [2, с. 27].

Логическая модель данных представлена на рисунке 4.

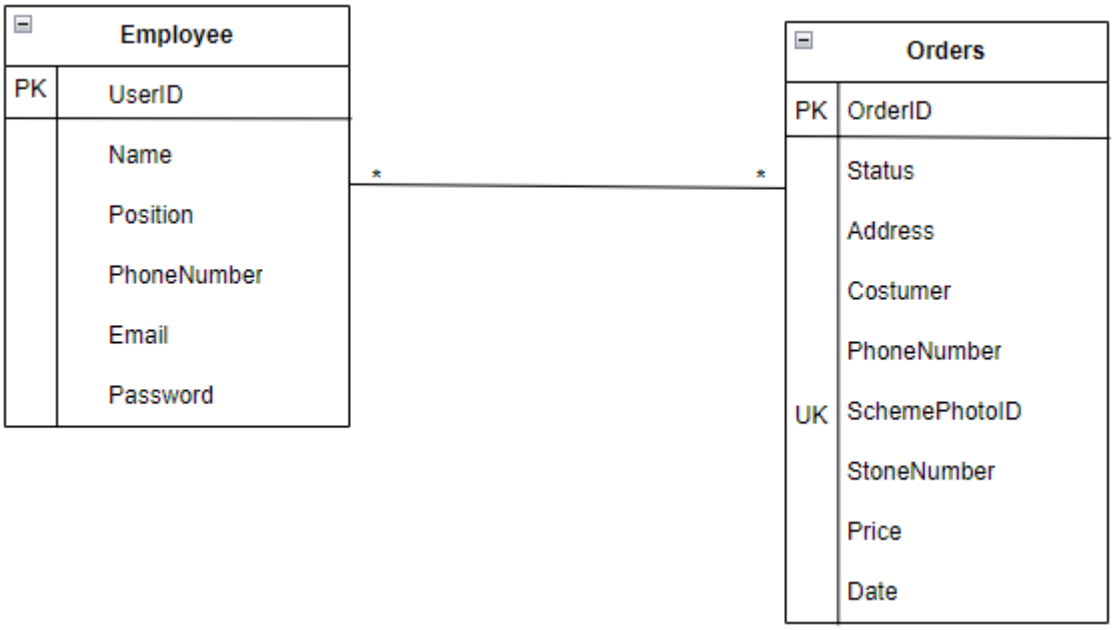

Рисунок 4 – Логическая модель данных

При написании физической модели данных за основу берется логическая модель данных. Физическая модель данных зависит от конкретной СУБД и описывает реализацию объектов модели. Она вводит дополнительные атрибуты, такие как хранимые процедуры и типы данных.

Основной целью физического проектирования базы данных является описание способа физической реализации логического проекта базы данных в конкретной СУБД [2, с. 81].

Физическая модель данных представлена на рисунке 5.

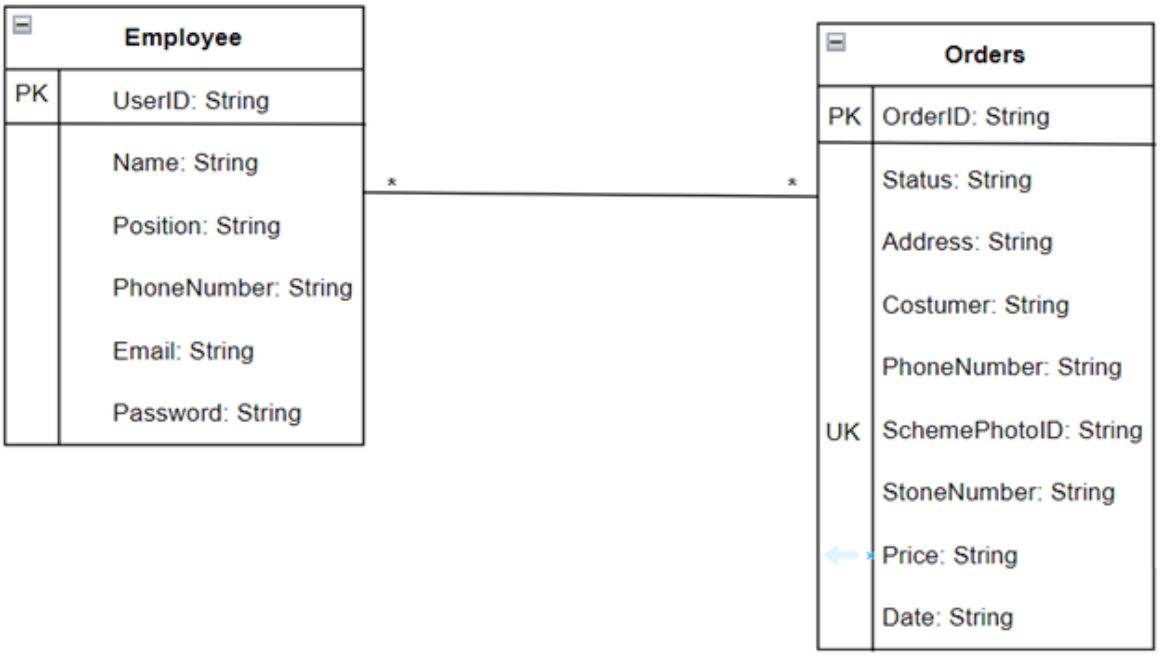

Рисунок 5 – Физическая модель данных

Таблица 1 – Структура таблиц

Название поля	Тип поля	Описание
Employee (Работник)		
UserID	String	ID пользователя (строка, содержащая уникальный идентификатор пользователя)
Name	String	Имя (строка, содержащая имя пользователя)
Position	String	Должность (строка, содержащая роль пользователя)
PhoneNumber	String	Номер телефона (строка, содержащая номер телефона пользователя)
Email	String	Почта (строка, содержащая адрес электронной почты пользователя)
Password	String	Пароль (строка, содержащая пароль пользователя)
Orders (Заказы)		
OrderID	String	ID заказа (строка, содержащая уникальный идентификатор заказа)
Status	String	Статус (строка, содержащая статус заказа)
Address	String	Адрес (строка, содержащая адрес клиента)
Costumer	String	Клиент (строка, содержащая имя клиента)
PhoneNumber	String	Номер телефона (строка, содержащая номер телефона клиента)
SchemePhotoID	String	ID фотографии схемы (строка, содержащая уникальный идентификатор фотографии)
StoneNumber	String	Номер камня (строка, содержащая номер цвета камня)

Price	String	Цена (строка, содержащая цену за заказ)
Date	String	Дата (строка, содержащая дату создания заказа)

2.4 Проектирование пользовательских сценариев интерфейса

Качество пользовательского интерфейса можно отнести к отдельным свойствам программного обеспечения. Если интерфейс не будет удовлетворять требованиям стандартов и руководящих документов по проектированию, то пользоваться им будет как минимум непросто [15]. Эффективный и удобный дизайн может повысить пользовательскую удовлетворенность и уровень вовлеченности [3].

Для этого составим требования, необходимые для проектирования мобильного приложения:

- простота использования. Интерфейс пользователя должен быть понятен, не перегружен лишними элементами дизайна, всем элементам необходимо следовать общему стилю. Кнопки и другие элементы управления обязаны выделяться и быть легко доступны для пользователя;
- отзывчивость. Интерфейс должен мгновенно и предсказуемо реагировать на действия пользователя;
- эффективность. Меню интерфейса должно быть спроектировано таким образом, чтобы пользователь мог быстро добраться до нужной ему страницы.

Диаграмма вариантов использования представлена на рисунке 6.

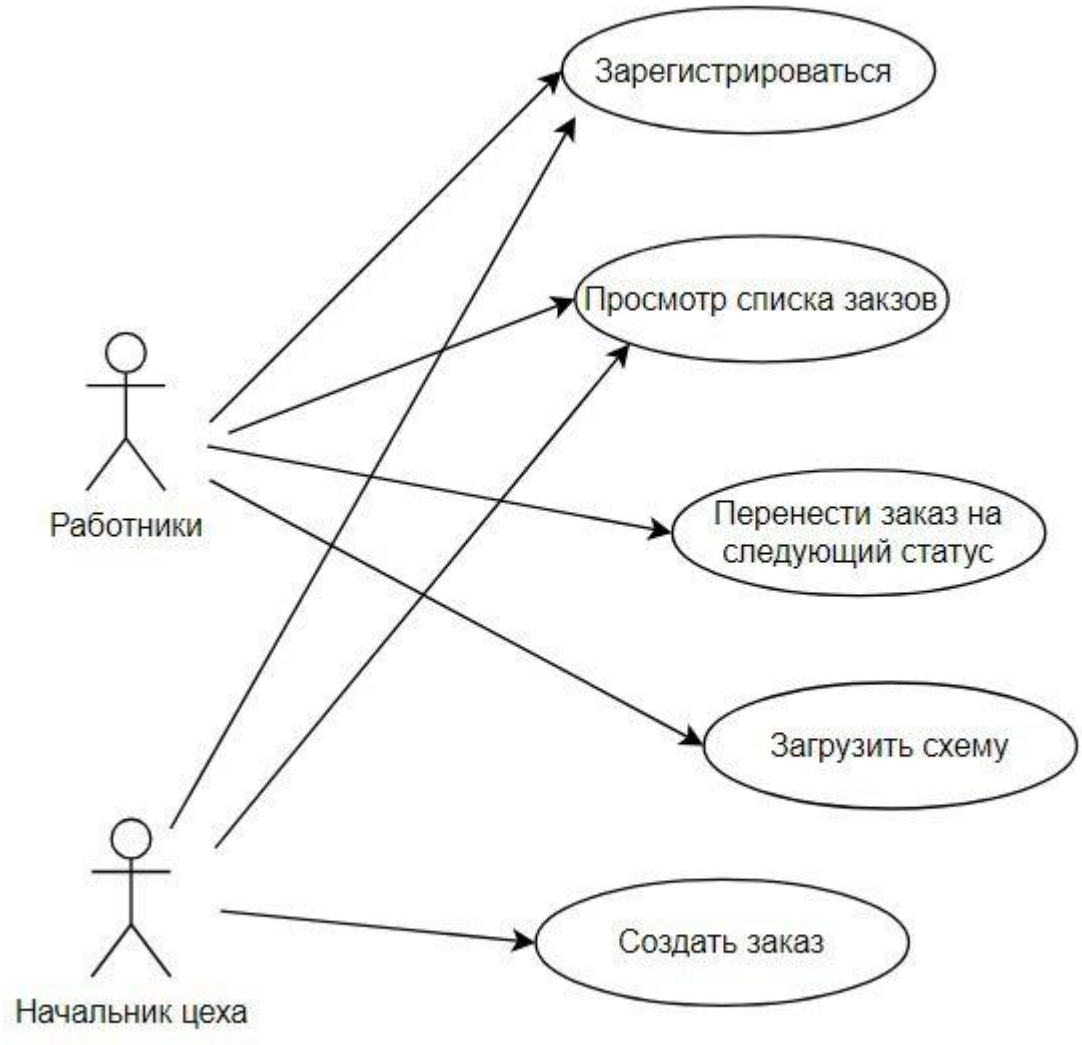

Рисунок 6 – Диаграмма вариантов использования.

Представление вариантов использования показывает, как система должна выглядеть «извне», то есть оно отражает функции системы и ее взаимодействие с внешним окружением. Основное внимание здесь уделяется представлению высокого уровня, отображающему, что система должна делать, а не как она будет делать это. Представление вариантов использования является исходным концептуальным представлением системы для последующей детализации [9].

Роли:

– работник: пользователь приложения;

– начальник цеха: пользователь с правами администратора.

Функция регистрирования:

– работник: вход в приложение с использованием учетной записи;
– начальник цеха: вход в приложение с использованием административной учетной записью.

Функция просмотра списка заказов:

– работник: просмотр списка заказов;
– начальник цеха: просмотр списка заказов, возможность редактирования заказа.

Функция переноса заказа на следующий статус:

– работник: перенос заказа на следующий статус;
– начальник цеха: недоступно.

Функция загрузки схемы:

– работник: загрузка фотографии схемы изделия;
– начальник цеха: недоступно.

Функция создания заказа:

– работник: недоступно;
– начальник цеха: создание нового заказа, ввод необходимой информации о заказе.

Выводы по главе 2

В рамках второй главы была выбрана основная технология разработки мобильного приложения, инструмент разработки, язык программирования и платформа для приложения. Спроектирована архитектура приложения и модели данных. Также были разработаны пользовательские сценарии и требования к интерфейсу.

Глава 3 Реализация, тестирование и запуск мобильного приложения для малого предприятия

3.1 Описание разработанного решения

Разработанное приложение представляет собой законченное, функциональное приложения для упорядочивания заказов малого предприятия. Оно позволяет работникам предприятия эффективно отслеживать, управлять и редактировать заказы, что существенно повышает удобство и ускорения бизнес-процессов предприятия. Для разработки приложения были использованы данные технологии:

- для реализации интерфейса использовались встроенные компоненты Android и язык разметки XML для верстки макетов [6];
- в качестве бэкенда, был выбран сервис Appwrite и его Appwrite SDK, который предоставляет библиотеку классов для доступа к удаленному серверу;
- вся разработка велась в среде Android Studio, которое является официальной интегрированной средой для разработки мобильных приложений [6] и предоставляет удобный интерфейс и широкий функционал.

Функционал мобильного приложения включает в себя:

- экраны авторизации и регистрации пользователя: предоставляют поля для ввода данных для авторизации;
- вкладки просмотра заказов: предоставляют списки заказов на различных стадиях производства;
- экран добавления заказа: позволяет добавлять заказы в базу данных и просматривать содержимое заказа.

Данное решение предоставляет интерфейс, соответствующий принципам UI/UX для удобного использования и управления заказами предприятия.

3.2 Создание проекта и добавление необходимых зависимостей

Для того чтобы создать проект необходимо запустить Android Studio и выбрать конфигурацию нашего проекта. В окне создания проекта необходимо указать имя проекта, имя пакета, язык на котором будет разрабатываться приложение и минимальную версию SDK (рисунок 7). Minimum SDK – это обозначение минимального уровня API на котором приложение может работать. В данном случае, при выборе API 24, приложение будет запускаться на 97% устройств. Выбор именно данного уровня API был обусловлен тем, что этот уровень предоставляет для использования большее количество функции библиотек.

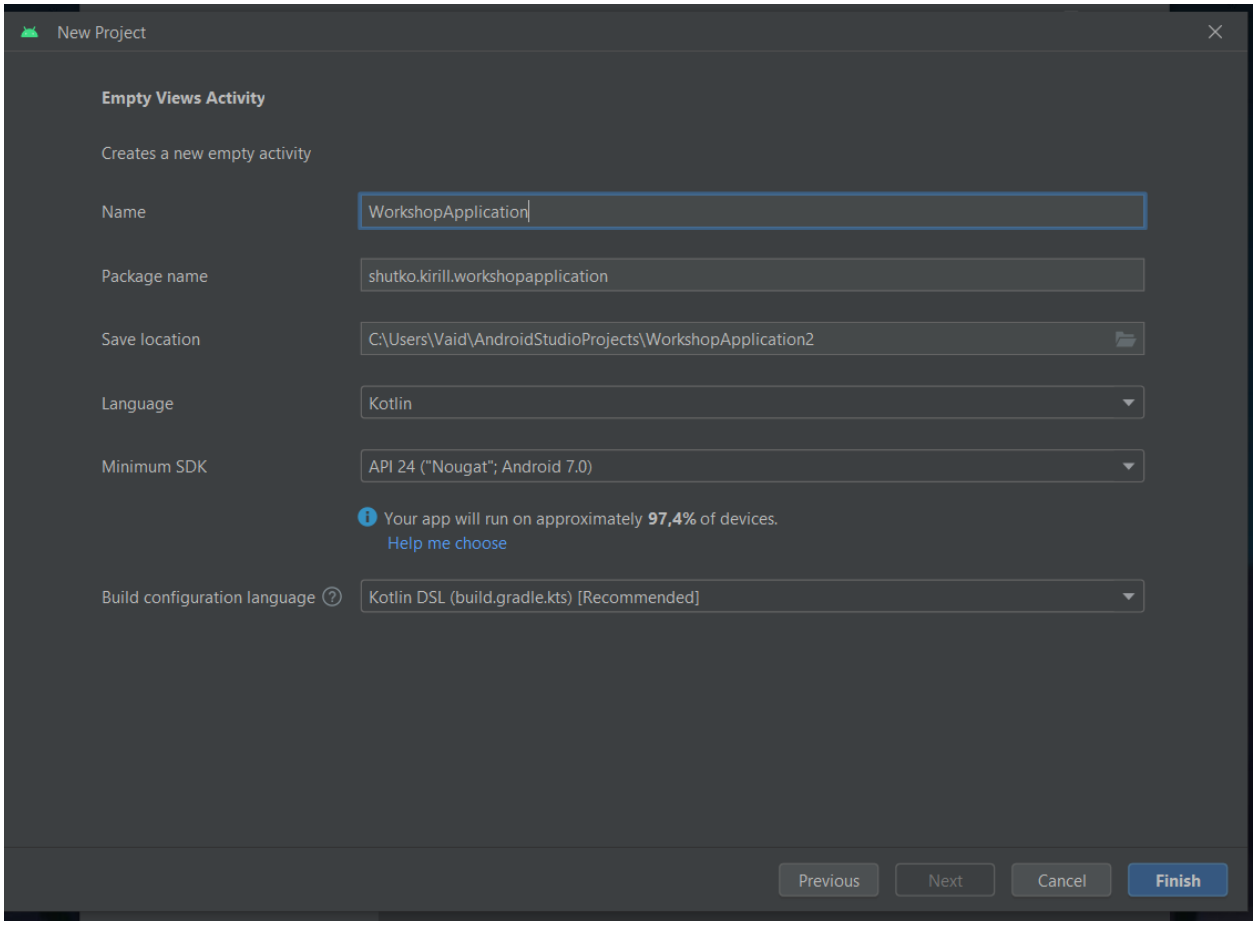

Рисунок 7 – Окно создания проекта в android studio

После создания, открывается редактор, где мы можем видеть всю структуру проекта в левой вкладке среды разработки (рисунок 8).

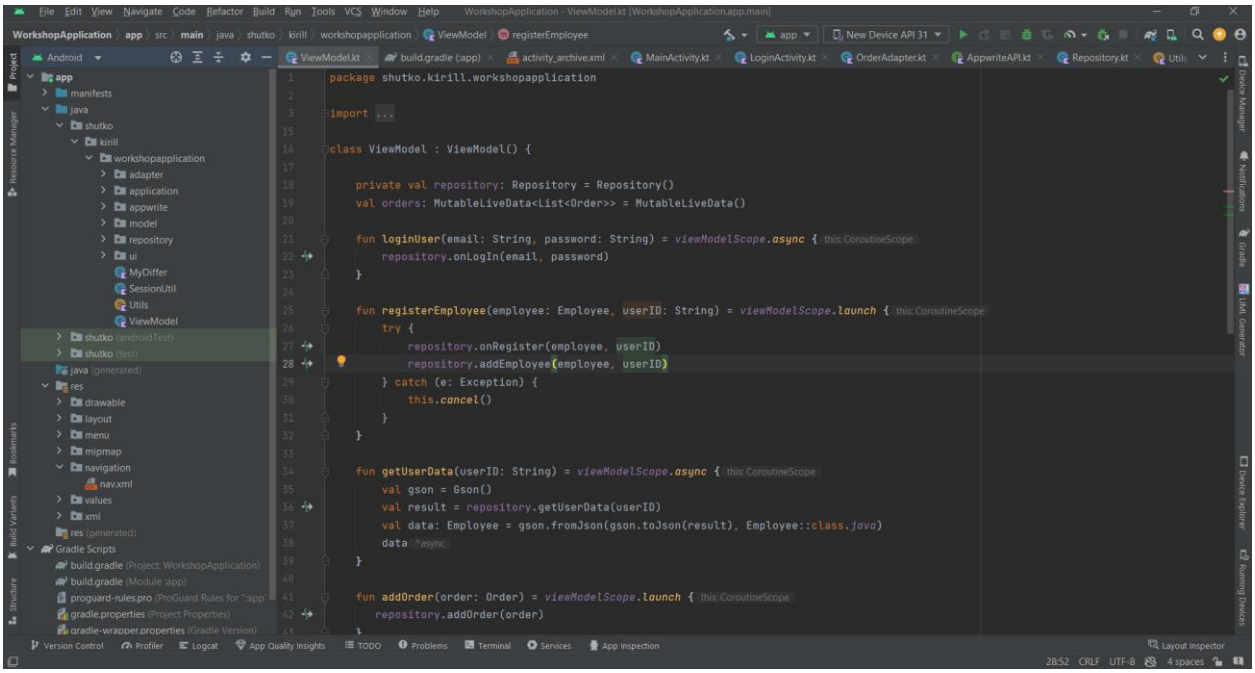

Рисунок 8 – Главное окно редактора среды android studio

Основные директории в структуре проекта являются следующими:

- app – является директорией где хранятся все классы проекта, здесь имеется файл MainActivity, который является основным классом при загрузке приложения;
- manifest – в этой папке хранится файл AndroidManifest. Файл манифеста является важной частью нашего приложения, поскольку он определяет структуру и метаданные приложения [1]. В частности, в манифесте приложения описываются компоненты приложения: службы, широковещательные приемники, контент-провайдеры; объявляются разрешения для доступа к функциям устройства, перечисляются библиотеки, с которыми приложение должно быть связано [1];

– res – эта папка представляет с собой директорию ресурсов для нашего приложения. Здесь хранятся файлы макетов интерфейса, константы цветов, стилей, иконок и графов навигации;
– gradle script – в этой директории находятся файлы для автоматической сборки и управления зависимостями Gradle, определяемом на предметно-ориентированном языке Groove. Файл конфигурации определяет глобальные настройки проекта [13].

Для описания необходимых зависимостей, переходим в файл build.gradle. Здесь имеются зависимости с библиотеками Android компонентов, таких как библиотек поддержки (appcompat), дизайна (material), верстки (constraintlayout) и тестирования (junit, espresso). Данные зависимости приведены на рисунке 9.

```
dependencies {

    implementation 'androidx.core:core-ktx:1.13.1'
    implementation 'androidx.appcompat:appcompat:1.7.0'
    implementation 'com.google.android.material:material:1.12.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.preference:preference-ktx:1.2.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
```

Рисунок 9 – Внедренные зависимости проекта

Дополнительно, добавим зависимость к библиотекам Appwrite (рисунок 10) и архитектурному компоненту ViewModel (рисунок 11).

```
//appwrite
implementation("io.appwrite:sdk-for-android:4.0.0")
```

Рисунок 10 – Добавление зависимости Appwrite

```
// Architectural Components
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.8.1"
```

Рисунок 11 – Добавление зависимости для ViewModel

После синхронизации проекта с фалами Gradle, проект будет перестроен с учетом добавленных зависимостей и классы данных библиотек будут доступны.

3.3 Реализация регистрации и аутентификации пользователя

Для добавления возможности регистрации и аутентификации, необходимо зарегистрироваться в сервисе Appwrite и выбрать бесплатный план “Starter”. После создания организации и проекта у нас появится консоль проекта с общей информацией по количеству таблиц, пользователей и запросов к базе данных. Консоль приведена на рисунке 12.

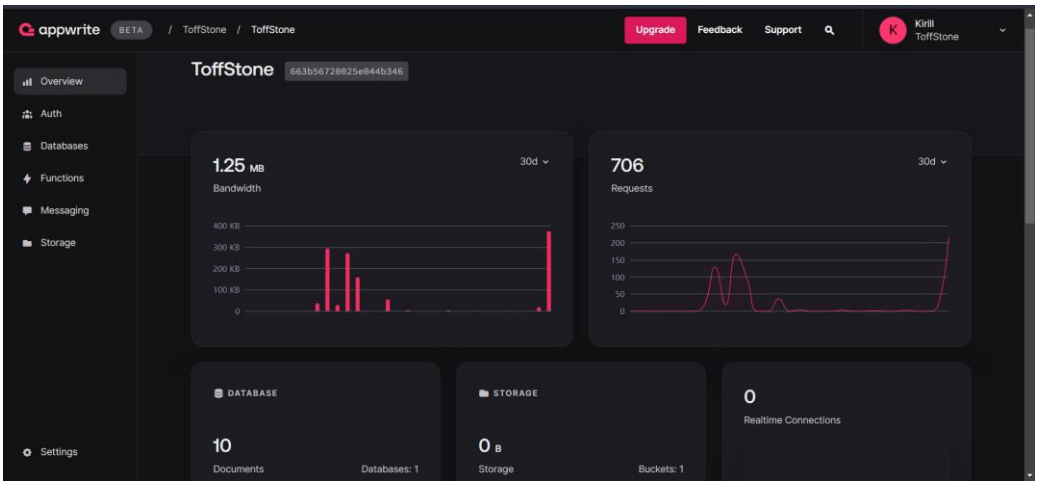

Рисунок 12 – Консоль проекта в сервисе Appwrite

В структуре проекта, создаем каталог appwrite в котором будет класс инициализации сервиса Appwrite и интерфейс AppwriteAPI. В классе Appwrite.kt инициализируем переменные для доступа к классам библиотек Client, Database, Account и Storage (рисунок 13).

Рисунок 13 – Инициализация классов библиотеки Appwrite

В интерфейс AppwriteAPI добавляем необходимы методы для регистрации и авторизации пользователя (рисунок 14).

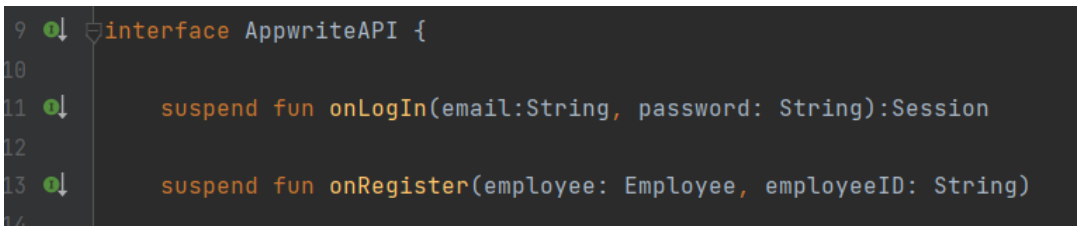

Рисунок 14 – Методы интерфейса для регистрации

Далее эти методы будут реализованы в классе Repository которая наследуется от интерфейса AppwriteAPI. Реализация этих методов приведена на рисунке 15.

```
class Repository : AppwriteAPI {

    private val appwriteDatabase = Appwrite().getDatabaseInstance()
    private val appwriteAccount = Appwrite().getAccountInstance()
    private val appwriteStorage = Appwrite().getStorageInstance()

    override suspend fun onLogIn(email: String, password: String): Session =
            appwriteAccount.createEmailSession(email, password)


    override suspend fun onRegister(employee: Employee, employeeID: String) {
        appwriteAccount.create(employeeID, employee.email, employee.password, employee.name)
    }
```

Рисунок 15 – Реализация методов интерфейса

В этом фрагменте в качестве параметров для метода onLogIn передаются поля email и password, внутри метода вызывает библиотечная функция createEmailSession которая создает сессию для пользователя по его почте и паролю [16]. В метод onRegister передается структура пользователя и его идентификатор, внутри метода вызывается функция create, которая предназначена для создания пользователя [16].

Чтобы мы могли вызывать методы, необходимо описать их в классе ViewModel. Данный класс является одним из слоев в архитектуре MVVM, в нем мы вызываем данные методы внутри сопрограмм для их асинхронного выполнения и заключаем их в блок try/catch для отлова ошибок (рисунок 16).

```
class ViewModel : ViewModel() {

    private val repository: Repository = Repository()
    val orders: MutableLiveData<List<Order>> = MutableLiveData()

    fun loginUser(email: String, password: String) = viewModelScope.async { this: CoroutineScope
        repository.onLogIn(email, password)
    }

    fun registerEmployee(employee: Employee, userID: String) = viewModelScope.launch { this: CoroutineScope
        try {
            repository.onRegister(employee, userID)
            repository.addEmployee(employee, userID)
        } catch (e: Exception) {
            this.cancel()
        }
    }
```

Рисунок 16 – Описание методов в классе ViewModel

Сопрограммы – это механизм для работы с асинхронными операциями в мобильных приложениях. Использование сопрограмм позволяет писать код, который выглядит как синхронный, но при этом выполняется асинхронно, что способствует повышению производительности [19].

Для работы с сопрограммами в Android приложениях используется библиотека Kotlin Coroutine, которая является частью языка Kotlin и предоставляет средства для создания и управления сопрограммами. С помощью сопрограмм можно выполнять операции в фоновом потоке, выполнять длительные задачи без блокирования основного потока выполнения, делать сетевые запросы и работать с базой данных [19].

В сопрограммах используются различные методы, позволяющие управлять и контролировать их выполнение.

launch() - метод, который запускает новую сопрограмму.

async() - метод, позволяющий запускать асинхронные операции и возвращать результат в виде объекта Deferred.

В сопрограммах также существует viewModelScope, которая предназначена для использования в ViewModel классах. ViewModelScope автоматически отменяет все запущенные сопрограммы при уничтожении ViewModel, что позволяет избежать утечки памяти и неправильного выполнения операций после завершения жизненного цикла ViewModel [19].

Методы работы с сопрограммами и использование viewModelScope позволяют совершать асинхронные операции в приложениях на платформе Android, обеспечивая управление ресурсами и жизненными циклами компонентов.

Теперь данные методы можно вызывать в классе LoginActivity и RegisterActivity, которые являются представлением пользовательского интерфейса для регистрации и аутентификации (рисунки 17 – 18).

Рисунок 17 – Вызов метода в классе авторизации

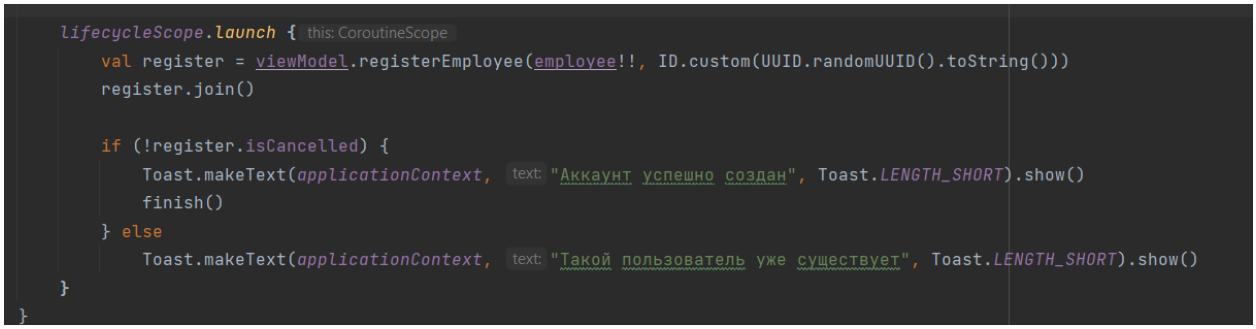

Рисунок 18 – Вызов метода в классе регистрации

Для создания интерфейса экранов регистрации и авторизации были применены компоненты верстки на языке xml, такие как TextView, EditText, LinearLayout и Button. Реализованные интерфейсы представлены на рисунках 19 – 20.

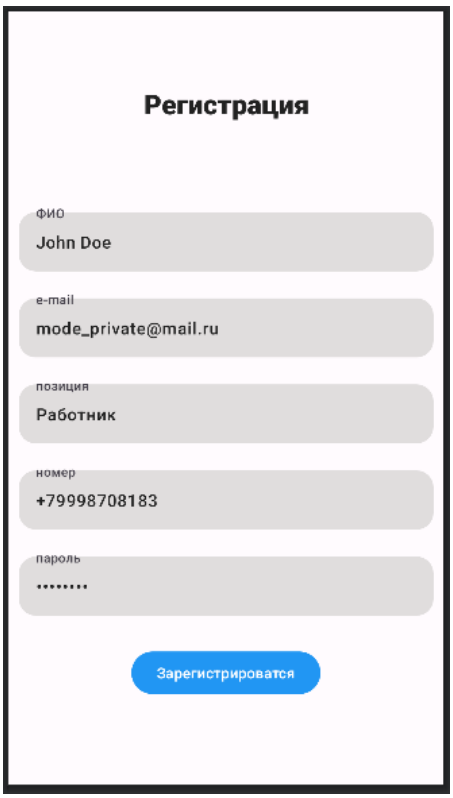

Рисунок 19 – Экран регистрации

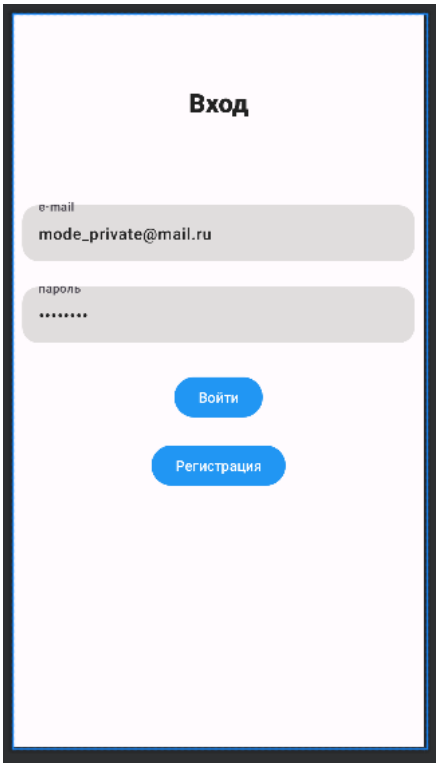

Рисунок 20 – Экран авторизации

Таким образом была создана функциональность для экранов авторизации и регистрации.

3.4 Реализация удаленной базы данных и взаимодействия

Для хранения данных заказов и работников, реализуем базу данных средствами Appwrite. Переходим во вкладку Databases и создаем базу данных WorkshopDatabase (рисунок 21).

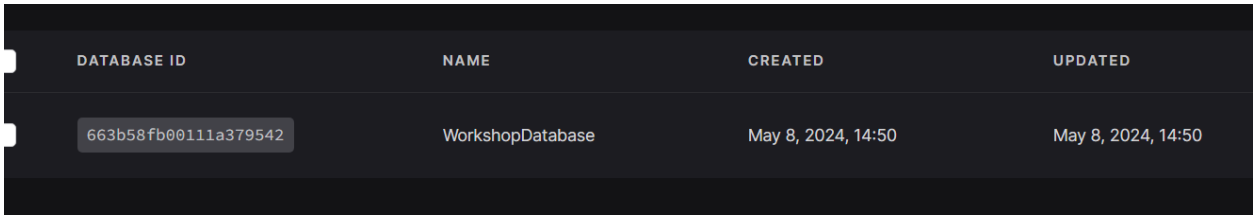

Рисунок 21 – База данных

Необходимо создать две таблицы – Employee и Orders, с атрибутами исходя из описанной модели (рисунки 22 – 23).

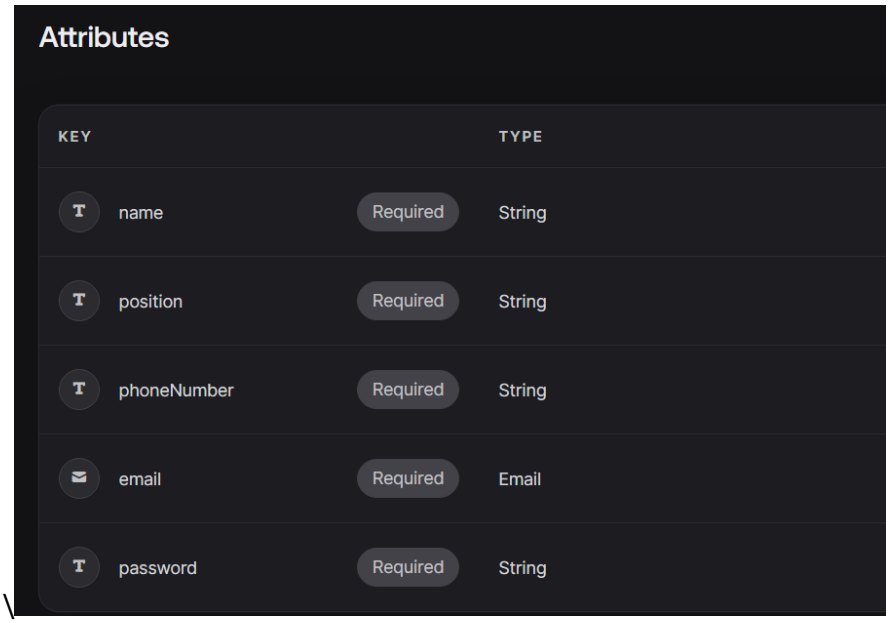

Рисунок 22 – атрибуты таблицы Employee

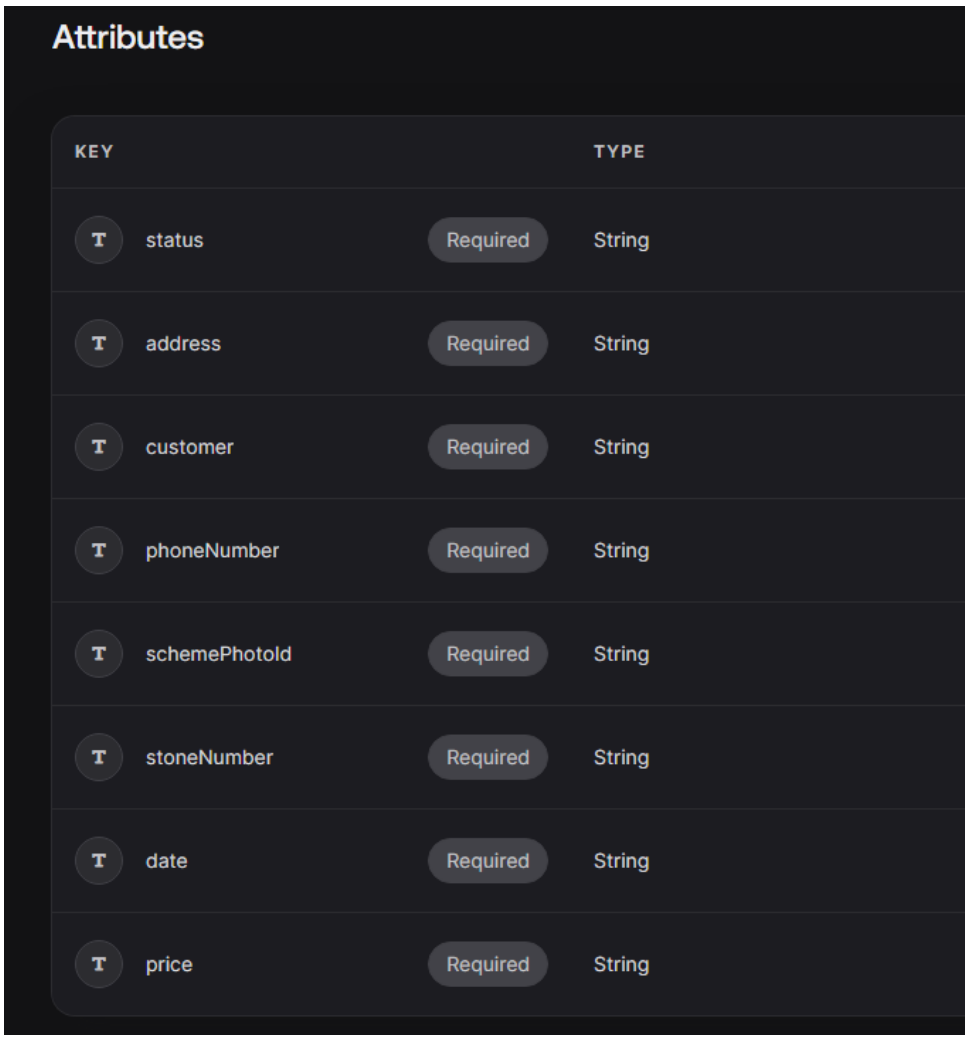

Рисунок 23 – атрибуты таблицы Orders

Эту структуру таблиц также описываем в слое модели приложения, создавая классы Employee и Order, которые являются классами данных (data class). Класс Order наследуется от интерфейса Serializable, которое позволяет сериализовать целый класс и передавать между экранами и фрагментами приложения.

Вся информация в базе данных хранится в таблицах. Каждая таблица определяется при помощи класса данных, который включает аннотации для имени таблицы и ее столбцов. Класс данных Kotlin позволяет создавать объекты, основным предназначением которых является хранение данных [6].

Классы Employee и Order представлены на рисунках 24 – 25.

```
data class Order(
    var id: String? = null,
    val status: String,
    val address: String,
    val customer: String,
    val phoneNumber: String,
    val schemePhotoId: String? = null,
    val stoneNumber: String,
    val stoneProduceCompany: String? = null,
    val date: String,
    val price: String
) : Serializable
```

Рисунок 24 – класс Order

```
data class Employee(
    val id: String? = null,
    val name: String,
    val position: String,
    val phoneNumber: String,
    val email: String,
    val password: String
)
```

Рисунок 25 – класс Employee

Теперь реализуем методы для записи в базу данных, получения списка заказов, удаления сессии, обновления заказа и выгрузку схемы в хранилище. Добавляем соответствующие методы в интерфейс AppwriteAPI (рисунок 26), реализуем их в классе Repository (рисунок 27), и вызываем в классе ViewModel как асинхронные операции (рисунок 28).

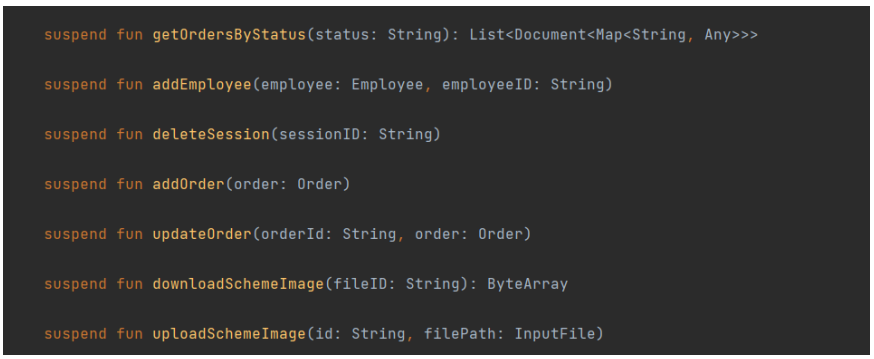

Рисунок 26 – Методы интерфейса для взаимодействия с базой данных

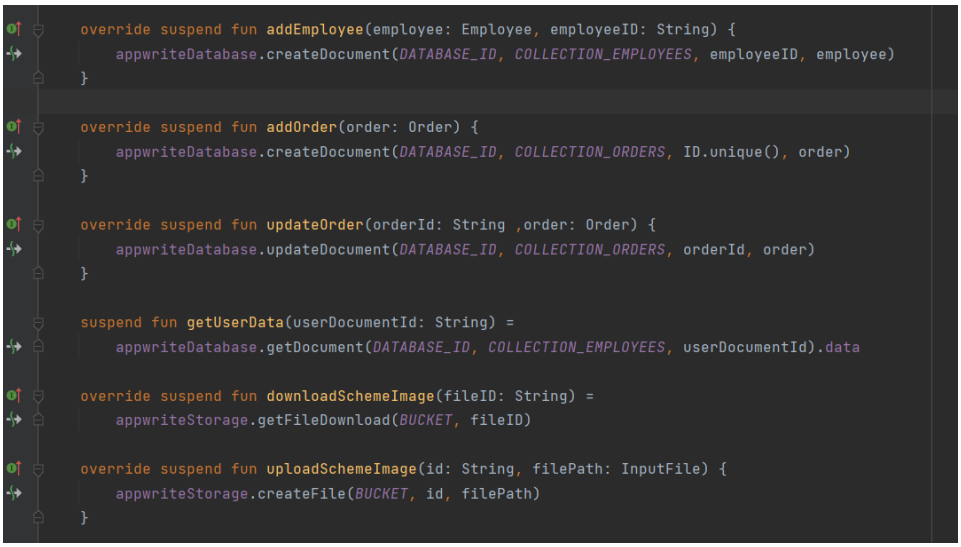

Рисунок 27 – Реализация методов интерфейса

Рисунок 28 – Вызов методов в виде асинхронных операций

Для отображения списка заказов был использован RecyclerView - инструмент для отображения списков элементов в приложениях Android. Он использует механизм переработки (view recycling), позволяющий переиспользовать элементы списка, что существенно снижает потребление памяти и повышает производительность при работе со списками большого объема (рисунок 29).

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.OrderFragment">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/orders_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

Рисунок 29 – Реализация списка на языке XML

Таким образом была создана функциональность для отображения списка и вкладок, также были реализованы методы для взаимодействия с удаленной базой данных.

3.5 Тестирование и запуск мобильного приложения

Тестирование функциональности приложения является неотъемлемой частью разработки, так как обеспечивает проверку соответствия его функций требованиям и ожиданиям.

Проведем тестирование всего функционала приложения. От аккаунта в роли «начальник цеха» создадим заказ и впишем всю необходимую информацию (рисунки 30-31).

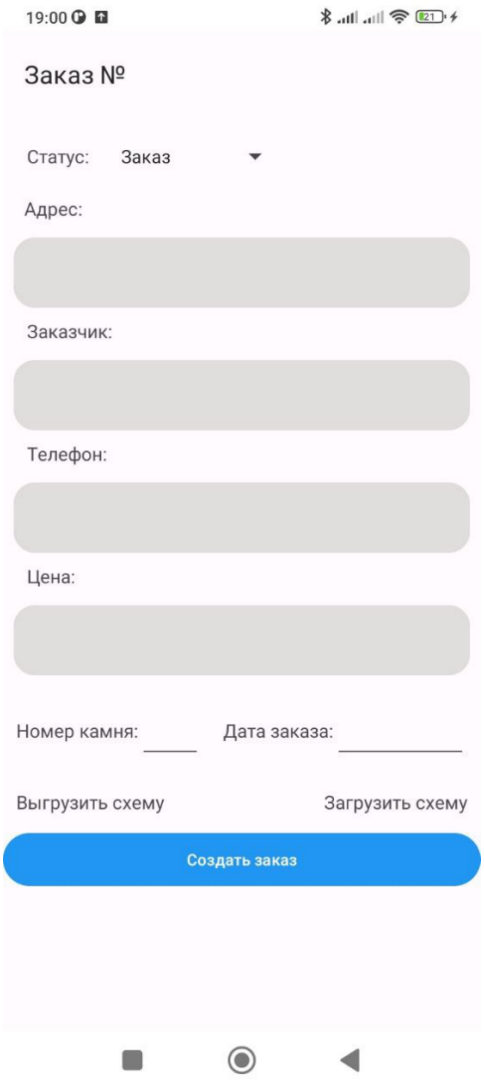

Рисунок 30 – Окно создания заказа

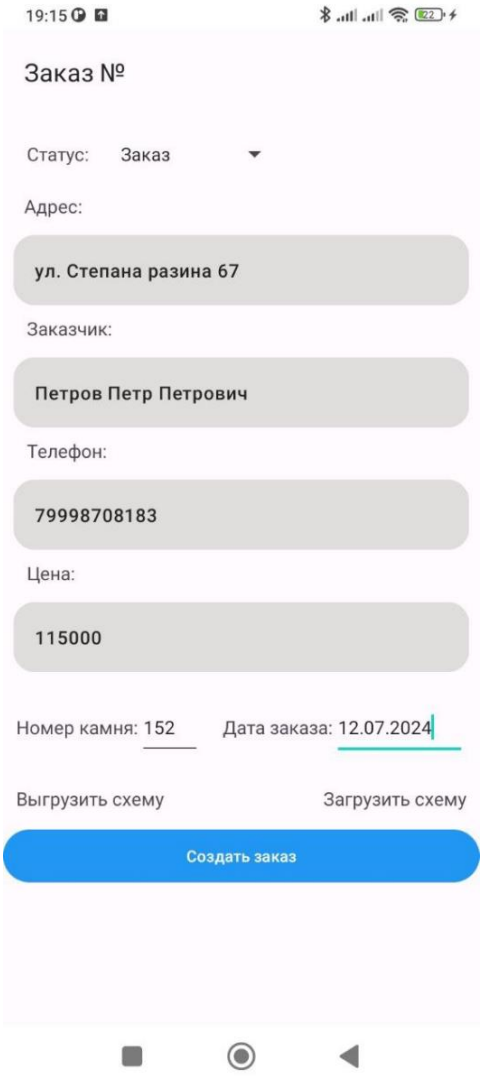

Рисунок 31 – Заполненное окно заказа

При создании заказа, мы можем видеть, что в удаленной базе данных, в таблице «Orders» добавились записи нашего заказа (рисунок 32).

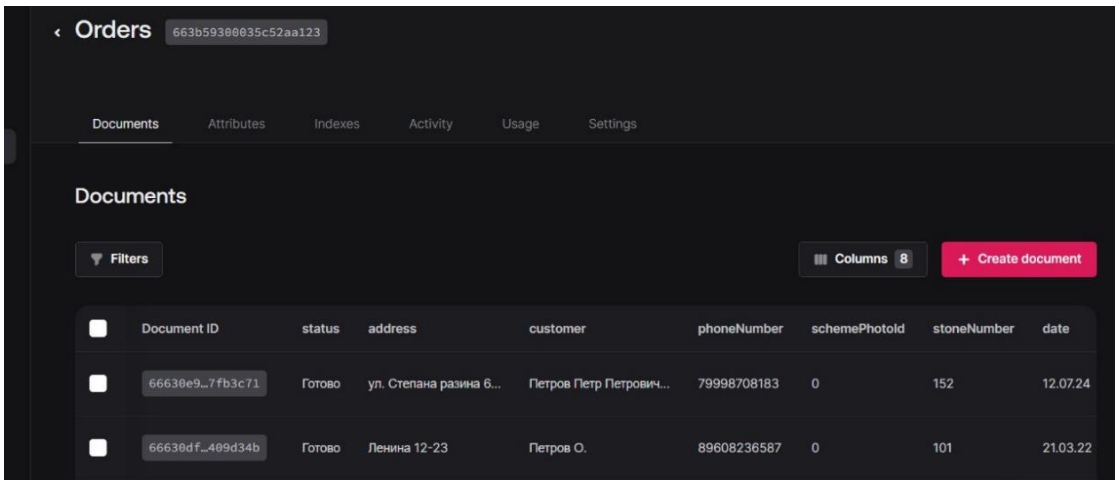

Рисунок 32 – Записи в базе данных

Далее из аккаунта в роли «работник» мы можем наблюдать созданный заказ и всю записанную информацию (рисунки 33-34).

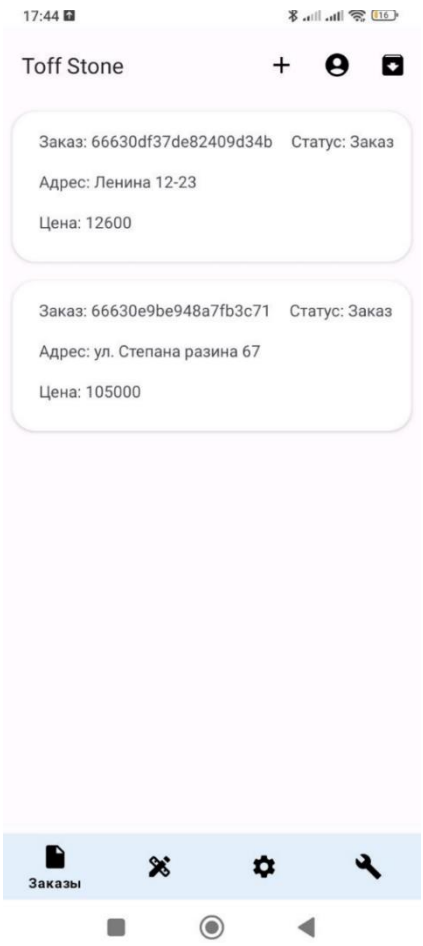

Рисунок 33 – Окно «Заказы»

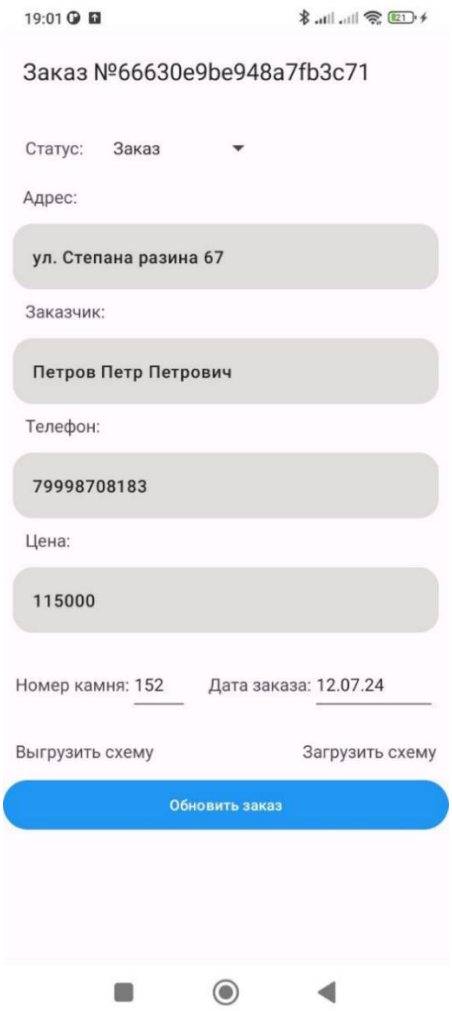

Рисунок 34 – Окно заказа

В статусе выбираем следующий этап и нажимаем «Обновить заказ». Таким образом, заказ переходит из статуса «Заказы» в статус «Замер», из статуса «Замер» в статус «В работе» и так далее, до завершения установки изделия и перехода заказа в архив (рисунки 35-38).

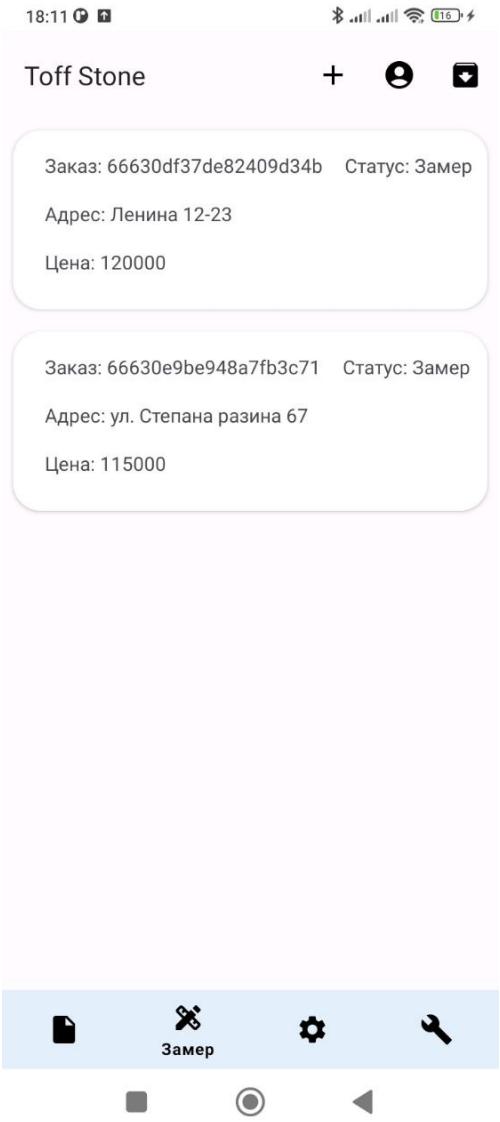

Рисунок 35 – Окно «Замер»

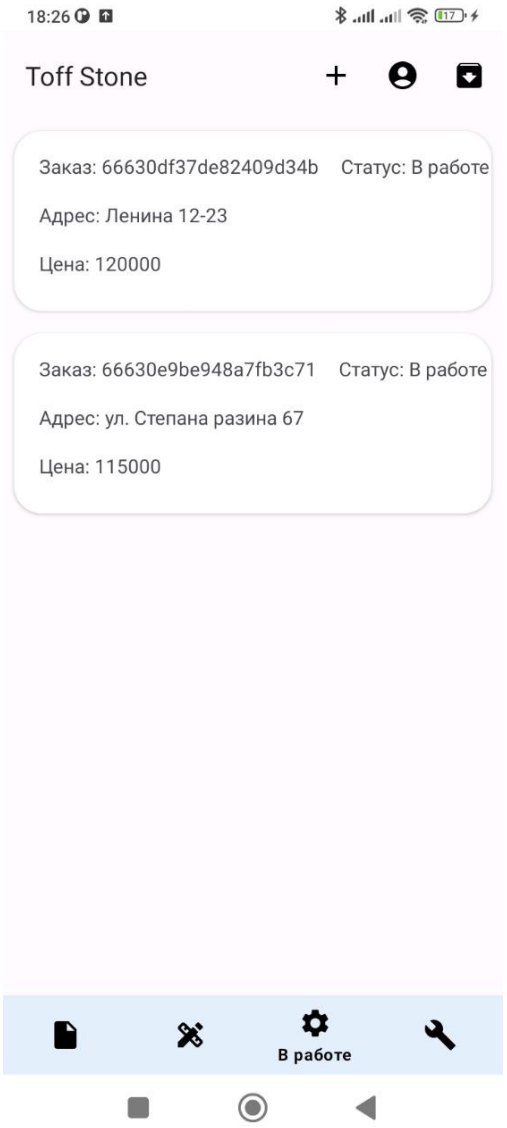

Рисунок 36 – Окно «В работе»

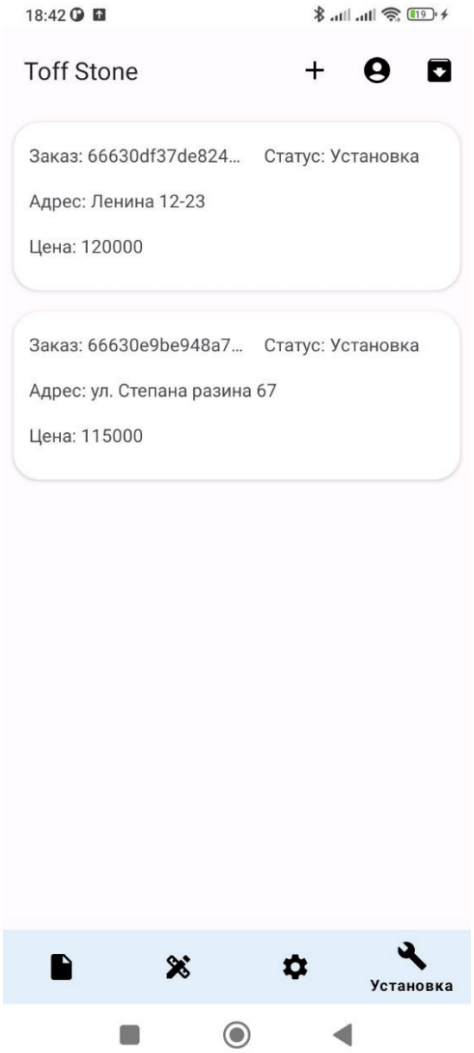

Рисунок 37 – Окно «Установка»

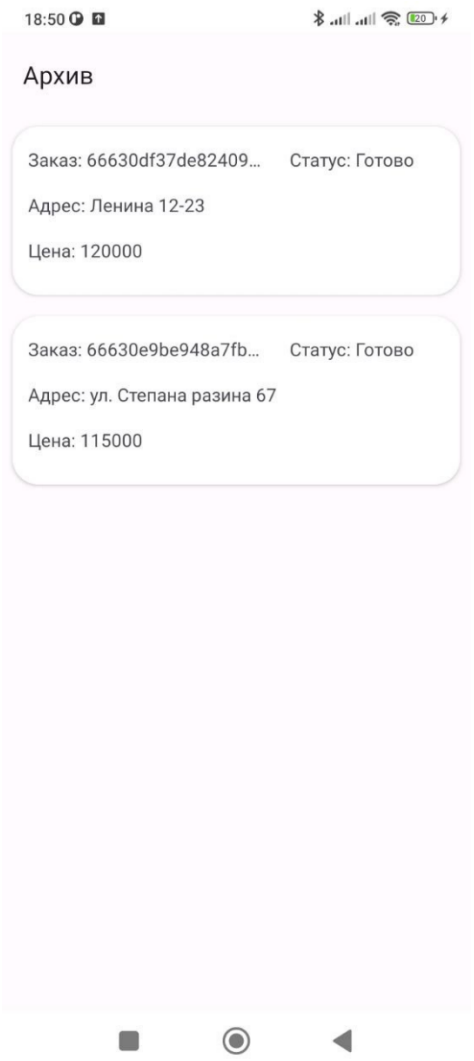

Рисунок 38 – Окно «Архив»

Было проведено тестирование функциональности и производительности мобильного приложения. Тестирование проводилось на разных android-устройствах с целью проверки работоспособности основных функций, а также оценки его адаптивности дизайна на различных android-устройствах.

В результате тестирования, приложение запускалось успешно и прошло все этапы проверки. Функции приложения, такие как регистрация, авторизация, создание, просмотр заказов, а также их редактирование, работают стабильно.

Тестирование показало высокий уровень адаптивности интерфейса приложения к различным устройствам и разрешениям экранов. Интерфейс приложения отображается корректно на всех мобильных устройствах.

Для распространения приложения и выгрузку его на площадках магазинов мобильных приложений, необходимо сгенерировать apk-файл.

APK (Android Package Kit) — формат архивных исполняемых файлов-приложений для Android и ряда других операционных систем, основанных на Android. Каждое приложение Android скомпилировано и упаковано в один файл, который включает в себя весь код приложения, ресурсы, активы, файл AndroidManifest.xml и библиотеки. Файл приложения может иметь любое имя, но расширение должно быть .APK [7].

Для генерации apk-файла, во вкладке Build нажимаем Generate signed Build / APK. В появившейся вкладке выбираем APK (рисунок 39) и создаем хранилище для ключа и пароль от хранилища (рисунок 40).

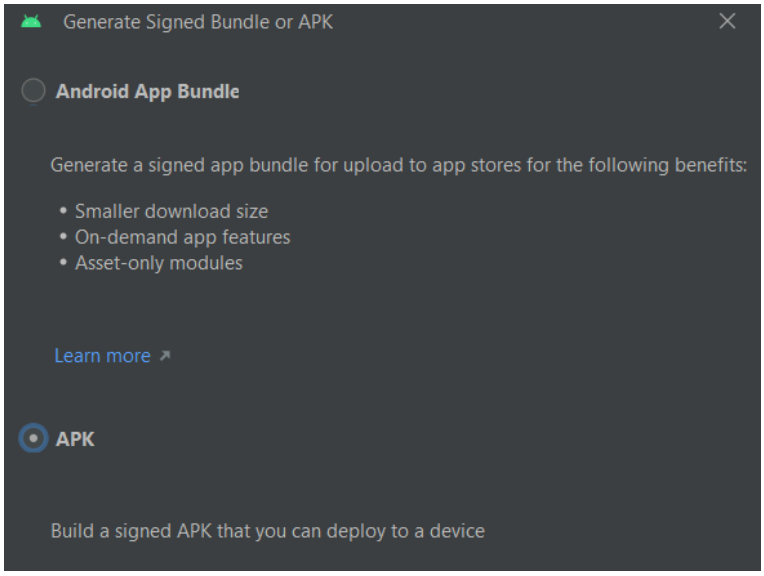

Рисунок 39 – Генерация apk-файла

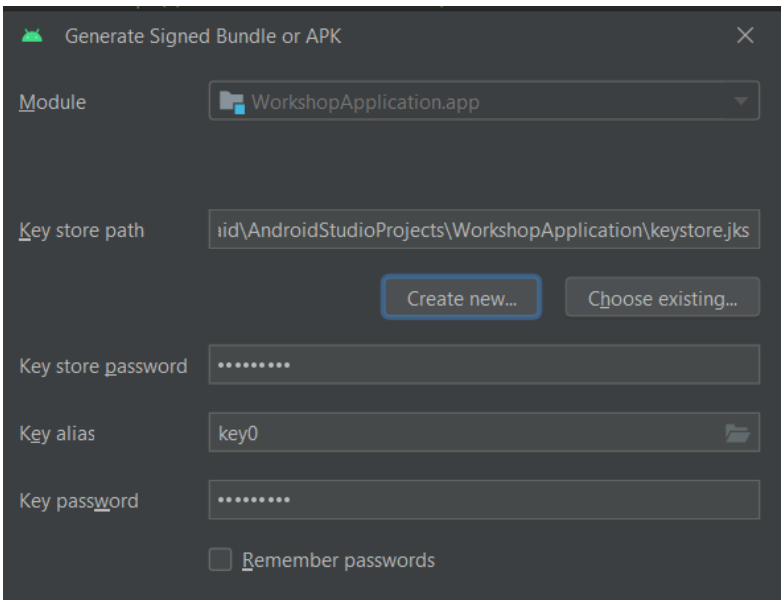

Рисунок 40 – Создание ключа

После этого в директории проекта появится сгенерированный apk-файл (рисунок 41).

Рисунок 41 – созданный apk-файл

Выводы по главе 3

В рамках 3 главы было кратко описана разработка приложения. Была описана разработка функционала приложения и последующее тестирование. В рамках тестирования была проведена работа над ошибками и исправленные недочеты. После исправления недочетов мобильное приложение показало полную функциональность и стабильность в рамках возложенных на него задач.

Заключение

В рамках данной дипломной работы была поставлена цель разработки мобильного приложения управления заказами малого предприятия с целью упрощения контроля над производственным циклом и решения проблем утери важной информации.

Был проведен анализ предприятия и бизнес-процессов для формулировки требований к мобильному приложению управления заказами малого предприятия.

Основываясь на сформулированных требованиях и проведенном анализе, была поставлена задача разработки мобильного приложения. Была выбрана платформа для реализации, архитектура и технология создания приложений, а также было произведено проектирование модели данных в трех слоях абстракции. Разработанное мобильное приложение базируется на стандартных компонентах операционной системы Android.

Мобильное приложение реализовано на выбранных средствах разработки. Тестирование приложения показало, что требуемые функциональность и стабильность достигнуты. Приложение работает исправно и может запускаться в работу на предприятии.

В результате проделанной работы была достигнута цель дипломной работы – разработано приложение управления заказами малого предприятия.

Разработанное мобильное приложение позволяет упростить и улучшить процесс изготовления изделия, повышая скорость и эффективность работы.

Список используемой литературы

1. Андроид Инструменты. Манифест приложения Android [Электронный ресурс]. URL: https://android-tools.ru/coding/manifest-prilozheniya/ (дата обращения: 15.05.2023).
2. Астапчук, В. А. Базы данных: проектирование и реализация : учебное пособие / В. А. Астапчук, Е. Н. Павенко, И. В. Эстрайх. — Новосибирск : НГТУ, 2023. — ISBN 978-5-7782-4917-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/404294 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 27.
3. Баланов, А. Н. Комплексное руководство по разработке: от мобильных приложений до веб-технологий : учебное пособие для вузов / А. Н. Баланов. — Санкт-Петербург : Лань, 2024. — ISBN 978-5-507-48841-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/394577 (дата обращения: 06.06.2024). — Режим доступа: для авториз. пользователей. — С. 97.
4. Волк, В. К. Базы данных. Проектирование, программирование, управление и администрирование / В. К. Волк. — 4-е изд., стер. — Санкт-Петербург : Лань, 2023. — ISBN 978-5-507-47243-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/346439 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 13.
5. Гринченко, Н. Н. Проектирование моделей данных : учебное пособие / Н. Н. Гринченко, Н. И. Хизриева, С. Н. Баранова. — Рязань : РГРТУ, 2022. — 48 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/380387 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 4.
6. Гриффитс Дэвид. Head First. Программирование для Android на Kotlin. 3-е изд. - Санкт-Петербург : Питер, 2023. - 912 с. - ISBN 978-5-4461-

2016-1. - URL: https://ibooks.ru/bookshelf/391722/reading (дата обращения: 04.06.2024). - Текст: электронный.

7. Гришанков, В. Что такое APK-файлы и зачем они нужны? // AndroiLime URL: https://androidlime.ru/apk-files

8. Гусев, К. В. Технология разработки программных приложений : учебное пособие / К. В. Гусев, М. Б. Туманова, Е. А. Чернов. — Москва : РТУ МИРЭА, 2023. — ISBN 978-5-7339-1938-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/382706 (дата обращения: 06.06.2024). — Режим доступа: для авториз. пользователей. — С. 4.

9. Дробот, П. Н. Автоматизация бизнес-процессов : учебно-методическое пособие / П. Н. Дробот, О. В. Штымова. — Москва : ТУСУР, 2012. — 49 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/11014 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 29.

10. Практикум по информатике / Н. М. Андреева, Н. Н. Василюк, Н. И. Пак, Е. К. Хеннер. — 3-е изд., стер. — Санкт-Петербург : Лань, 2024. — 248 с. — ISBN 978-5-507-47299-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/359810 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 106.

11. Рогачев С. Обобщенный Model-View-Controller // URL: https://rsdn.org/article/patterns/generic-mvc.xml

12. Синицын, И. В. Разработка мобильных приложений : учебное пособие / И. В. Синицын, Е. А. Чернов, Ю. А. Воронцов. — Москва : РТУ МИРЭА, 2023 — Часть 1 — 2023. — ISBN 978-5-7339-1799-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/368735 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 69-71.

13. Скин Джош, Гринхол Дэвид. Kotlin. Программирование для профессионалов. — СПб.: Питер, 2020. — 464 с.:ил.- ISBN 978-5-4461-1243-2.

14. Фешина, Е. В. С. А. Куштанок, Т. А. Крамаренко, Р. Я. Скорбатюк Анализ технологий разработки мобильных приложений и информационных систем на базе операционной системы Android // Вестник Адыгейского государственного университета. Серия 4: Естественно-математические и технические науки. — 2022. — № 1. — С. 85-91. — ISSN 2410-3225. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/journal/issue/331307 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 6.

15. Шибанов, С.В. Автоматизированное проектирование пользовательских интерфейсов / С.В. Шибанов, А.А. Пашкин // Вестник Пензенского государственного университета. — 2016. — № 4. — С. 67-73. — ISSN 2410-2083. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/journal/issue/301580 — С. 1.

16. Appwrite. Appwrite Cloud Client API Reference [Электронный ресурс]. URL: https://appwrite.io/docs/references/cloud/client-web/account (дата обращения: 15.05.2023).

17. Burns B., Grant B., Oppenheimer D. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Media, 2018. – 166 с.

18. Chen P.P. The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976. – 68 с.

19. Filip Babić, Luka Kordić, Nishant Srivastava Kotlin Coroutines by Tutorials: Best Practices to Create Safe & Performant Asynchronous Code With Coroutines. - 3 edition изд. - NY: Razeware LLC, 2022. - 301 с.

20. Meier R. Professional Android 4 Application Development. Wrox Press, 2012. – 864 с.

Приложение А

Фрагмент кода класса OrderAdapter

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.AsyncListDiffer
import androidx.recyclerview.widget.RecyclerView
import shutko.kirill.workshopapplication.MyDiffer
import shutko.kirill.workshopapplication.R
import shutko.kirill.workshopapplication.model.Order
class OrderAdapter() : RecyclerView.Adapter<OrderAdapter.OrderViewHolder>()
{
   val differ = AsyncListDiffer(this, MyDiffer.MyDiffUtil)
   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
OrderViewHolder {
      val view = LayoutInflater.from(parent.context).inflate(R.layout.item_list,
parent, false)
      return OrderViewHolder(view)
  }
   override fun onBindViewHolder(holder: OrderViewHolder, position: Int) {
      val order = differ.currentList[position]
      holder.apply {
         statusText.text = order.status
         orderNumber.text = order.id
         addressText.text = order.address
         price.text = order.price
      }
      holder.itemView.apply {
         setOnClickListener {
```

```
            onItemClickListener?.let{ it(order) }
          }
        }
    }
    override fun getItemCount(): Int {
        return differ.currentList.size
    }
    private var onItemClickListener: ((Order) -> Unit)? = null

    fun setOnItemClickListener(listener: (Order) -> Unit){
        onItemClickListener = listener
    }
    inner class OrderViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView){
        val orderNumber: TextView = itemView.findViewById(R.id.order_number)
        val statusText: TextView = itemView.findViewById(R.id.status)
        val addressText: TextView = itemView.findViewById(R.id.address)
        val price: TextView = itemView.findViewById(R.id.price)
    }
}
```