МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка Android-приложения для организации поставок автомобилей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-599.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент
_____ Т.А. Макаровских

Автор работы,
студент группы КЭ-401
_____ И.М. Чудов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-401
Чудову Ивану Максимовичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка Android-приложения для организации поставок автомобилей.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Гриффитс Д. Head First. Kotlin. – O'Reilly Media, Inc., 2019. – 452 с.

3.2. Официальный сайт проекта «itProger». [Электронный ресурс] URL: https://itproger.com (дата обращения: 20.01.2024 г.).

3.3. Скин Д., Гринхол Д. Kotlin. Программирование для профессионалов. – Издательство Питер, 2019. – 464 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Изучить методы создания приложений для операционной системы Android.

4.2. Привести описание требований к разрабатываемому продукту на основе диаграмм вариантов использования UML.

4.3. Спроектировать структуру приложения и разработать необходимые модули для ее функционирования.

4.4. Протестировать возможности системы на наличие программных ошибок.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент Т.А. Макаровских

Задание принял к исполнению И.М. Чудов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире увеличивается интерес к автомобилям, особенно к иностранным маркам, но ввиду текущих политических и экономических условий, возможны ограничения на импорт транспортных средств, что приводит к их периодической нехватке в автосалонах. Такая ситуация создаёт трудности как для покупателей, стремящихся приобрести конкретную модель автомобиля, так и для дилеров, которым важно удовлетворить возрастающий спрос. Разработка Android-приложения для организации поставок автомобилей является актуальной, поскольку это облегчит процессы покупки, доставки и управления заказами.

Постановка задачи

Целью данной выпускной квалификационной работы является создание Android-приложения для управления поставками автомобилей. Реализация этой цели включает в себя следующие задачи.

1. Изучение принципов разработки приложений для Android, включая изучение архитектуры системы, ключевых компонентов, жизненного цикла приложений и рекомендуемых методик разработки.

2. Анализ требований и функционала приложения, что включает определение потребностей пользователей, формулирование функциональных и нефункциональных требований, а также разработку пользовательских сценариев использования.

3. Создание диаграммы вариантов использования, которая будет демонстрировать различные сценарии взаимодействия с приложением, включая вход, просмотр объявлений, управление заявками и другие.

4. Разработка базы данных для хранения данных о пользователях, объявлениях и заявках клиентов, что подразумевает проектирование схемы базы данных, определение структуры таблиц и выбор подходящих технологий для реализации.

5. Реализация функционала для входа и регистрации пользователей, а также создание интерфейсов и логики для работы с пользовательскими и управленческими аккаунтами.

6. Разработка диаграммы классов, диаграммы деятельности, и диаграмм последовательности, чтобы организовать структуру приложения, обеспечить его модульность и оптимизировать процессы.

7. Создание функционала просмотра и редактирования объявлений, подачи и принятия заявок на покупку автомобилей, а также разработка пользовательских интерфейсов для этих функций.

8. Тестирование приложения на ошибки, что включает в себя проверку функций приложения на стабильность и корректность работы, а также исправление проблем. Этот этап тестирования поможет обеспечить надежность приложения перед его запуском.

Выполнение данных задач позволит создать функциональное и удобное приложение, которое будет соответствовать текущим требованиям рынка и удовлетворять как потребности автомобилистов, так и коммерческие интересы дилеров. В условиях, когда импорт иностранных автомобилей может быть ограничен из-за нестабильностей, наличие инструмента для организации поставок машин становится критически важным. Данное приложение является решением, которое не только упростит процесс покупки и доставки автомобилей, но и предоставляет дилерам мощный инструмент для управления бизнесом, обеспечивая более гладкое взаимодействие между покупателями и продавцами. Таким образом, разработка запланированного приложения по организации поставок автомобилей поможет смягчить негативные последствия ограничений на импорт, обеспечивая доступ к желаемым автомобилям.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения, списка литературы и пяти приложений. Объем работы составляет 73 страницы, объем списка литературы – 17 источников.

В первой главе «Анализ предметной области» описывается предметная область проекта и анализ существующих аналогов приложения. Также в ней были рассмотрены различные мобильные приложения, предназначенные для организации поставок автомобилей, в которых были выявлены их сильные и слабые стороны, а также определены ключевые функции, которые было необходимо реализовать в приложении [1-4]. Особое внимание было уделено анализу пользовательских интерфейсов, функциональности и пользовательского опыта [8,11].

Вторая глава «Требования к системе» посвящена выявлению функциональных и нефункциональных требований к разрабатываемой системе. Также была построена диаграмма вариантов использования и раскрыты возможные сценарии взаимодействия пользователя с приложением. Функциональные требования включают регистрацию и аутентификацию пользователей, просмотр автомобилей, заказ и покупку автомобилей. Нефункциональные требования включают надежность, производительность, безопасность, масштабируемость, совместимость и удобство использования [12,17].

В третьей главе «Архитектура системы» представлено описание системы, перечислены классы системы, представлены диаграммы последовательности для сценария использования приложения. Также в главе реализована диаграмма деятельности. Архитектура приложения основывается на модели MVVM (Model-View-ViewModel), что обеспечивает четкое разделение логики представления и бизнес-логики, улучшая тестируемость и масштабируемость системы. Описаны основные компоненты системы, их взаимодействие и роли в приложении.

В четвертой главе, посвященной процессу разработки, представлен исходный код программы, который является результатом реализации требований, описанных во второй главе. Эта глава также включает описание используемых технологий, архитектурных подходов и основных решений, принятых при разработке приложения. Также были рассмотрены ключевые модули и компоненты системы, такие как аутентификация пользователей, работа с базой данных Firebase, отображение списка автомобилей и обработка заявок пользователей.

В пятой главе проводится тестирование, включающее проверку созданного приложения на наличие ошибок и соответствие заявленным требованиям. Оно состоит из функционального тестирования, при котором проверяется правильность работы отдельных функций. Результаты тестирования подробно описаны, включая выявленные ошибки, их природу и способы их устранения, а также общую оценку качества приложения. Также были рассмотрены аспекты пользовательского тестирования и обратной связи от реальных пользователей для улучшения приложения.

Работа также включает приложения для дополнительного понимания разработки и функционирования.

В приложении А представлена спецификация вариантов использования и описание сценариев взаимодействия пользователей с системой.

В приложении Б содержатся скриншоты Android-приложения, показывающие визуальные примеры интерфейсов и экранов.

В приложении В приведены листинги Android-приложения с исходным кодом программы на языках Kotlin и XML.

Приложение Г включает примеры аналогичных проектов.

В приложении Д представлена диаграммы последовательности для сценария использования: «Просмотр объявления».

В приложении Е приведен листинг класса `AuthActivity`, отвечающего за аутентификацию в приложении.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

В рамках работы предстоит написать программу с помощью среды разработки Android Studio на языках программирования Java и Kotlin для Android устройств. Она включает в себя различные аспекты, связанные с процессом поставки автомобилей и управлением этим процессом с помощью мобильного приложения. Ниже приведены некоторые из ключевых аспектов, которые можно рассмотреть в проекте.

1. Учет и хранение информации об автомобилях. В приложении можно разработать систему для учета автомобилей, включая их марки, модели, характеристики, цены изображения и другие сопутствующие данные.

2. Заказы и поставки. Приложение может предоставлять возможность создания заказов на автомобили со стороны клиентов, а также управление процессом добавления объявлений, рассмотрения заявок и информирование клиентов о новых автомобилях со стороны администраторов.

3. Уведомления и оповещения. Мобильное приложение может использоваться для отправки уведомлений и оповещений клиентам о статусе и изменениях в их заказах, например, об обновлении статуса поставки, новых автомобилях и т.д.

4. Интеграция с другими системами: при необходимости можно рассмотреть возможность интеграции разрабатываемого приложения с другими системами, такими как системы управления складом, системы учета заказов, системы управления клиентскими данными или внешними платежными системами.

5. Взаимодействие с клиентами: приложение может предоставлять функции обратной связи и коммуникации с клиентами, такие как возможность задавать вопросы, оставлять отзывы, запросить консультацию и т.д.

1.2. Анализ аналогичных проектов

В ходе анализа аналогичных проектов были изучены популярные приложения, такие как Auto.ru и Drom.ru. Они предоставляют пользователям доступ к большому количеству объявлений о продаже автомобилей по всей России. В данных приложениях можно найти объявления о новых и подержанных автомобилях различных марок и моделей. Стартовые страницы данных проектов представлены в приложении Г на рисунке 4.

Auto.ru и Drom.ru имеют множество функций, включая фильтрацию по различным параметрам, просмотр подробных характеристик автомобиля, фотографий и контактной информации продавца. Однако, несмотря на широкие возможности, эти приложения не всегда удовлетворяют потребностям пользователей и дилеров, в связи с ограничениями, описанными ранее.

В списке ниже также представлены другие аналогичные проекты.

1. TrueCar: предлагает информацию о ценах на автомобили и позволяет сравнивать цены различных дилеров. Пользователи могут найти автомобиль, заполнить онлайн-заявку и связаться с дилером для покупки. TrueCar предоставляет удобный интерфейс для поиска автомобилей и анализа рыночных цен. Стартовая страница представлена в приложении Г на рисунке 5.
2. Cars.com: это еще один популярный иностранный сайт, предоставляющий пользователям доступ к различным инструментам для поиска, сравнения и покупки автомобилей. Пользователи могут искать автомобили по различным параметрам, читать отзывы и сравнивать цены. Стартовая страница сайта в приложении Г на рисунке 5.
3. Grand Auto: это российская компания, осуществляющая поставки автомобилей в Российскую Федерацию из множества регионов. Создаваемое приложение будет ориентировано на данную концепцию, поскольку у данной компании мобильное приложение отсутствует. Начальный экран сайта представлен в приложении Г на рисунке 5.

Создаваемое приложение, которое в дальнейшем тексте будет упоминаться как AutoSell, ориентировано на российский рынок и направлено на решение проблем ограниченного импорта автомобилей. Основные функции приложения должны включать выставление объявлений администратором и создание заявок на покупку автомобилей пользователями. Приложение должно облегчить процесс покупки и продажи автомобилей, создавая удобную платформу для взаимодействия между покупателями и продавцами.

AutoSell будет включать две основные функции.

1. Выставление объявлений администратором: администратор будет добавлять новые объявления о продаже автомобилей, управлять их состоянием и обновлять информацию по мере необходимости. Пользователи смогут откликнуться на объявление и приобрести выставленный автомобиль.

2. Создание заявок на покупку автомобилей пользователями: пользователи смогут просматривать объявления, выбирать интересующие автомобили и подавать заявки на их покупку. В заявке будет указана информация о пользователе и выбранном автомобиле. Администратор сможет просмотреть заявку и связаться с клиентом для обсуждения предложения.

Таким образом, приложение AutoSell будет способствовать созданию удобного и эффективного рынка автомобилей, удовлетворяя потребности как пользователей, так и дилеров в условиях ограниченного импорта автомобилей.

Этот анализ позволяет выделить основные требования и функциональные возможности, которые должны быть реализованы в разрабатываемом приложении для достижения поставленных целей.

В таблице 1 показано сравнение представленных сервисов.

Таблица 1 – Сравнительный анализ сервисов

Параметр	**Grand Auto**	**Auto.ru**	**Drom.ru**	**Cars.com**	**TrueCar**
Наличие мобильного приложения	Нет	Да	Да	Да	Нет
Доступность доставки из других стран	Да	Нет	Нет	Да	Да

Окончание таблицы 1

Параметр	Grand Auto	Auto.ru	Drom.ru	Cars.com	TrueCar
Удобство использования с помощью мобильного устройства	Нет	Да	Да	Нет	Нет
Доска объявлений	Нет	Да	Да	Да	Да
Заявки на покупку автомобиля	Да	Нет	Нет	Нет	Нет
Наличие аккаунтов	Да	Да	Да	Да	Да
Отслеживание доставки транспорта	Да	Нет	Нет	Нет	Нет

1.3. Инструменты разработки

В результате анализа предметной области и существующих работ по тематике дипломного проекта был выявлен набор инструментальных средств для решения поставленной задачи. Основным инструментом для создания запланированного приложения является среда разработки (IDE) Android Studio. Она включает в себя разработку на языках программирования Java, Kotlin и XML, специально созданных для решения задач, связанных с разработкой приложений для операционной системы Android.

Android Studio обеспечивает разработчиков всем необходимым для создания качественных приложений под Android. Она предлагает широкий набор инструментов, включая редакторы кода, средства отладки, дизайнеры пользовательского интерфейса, систему сборки и многое другое. Благодаря интеграции с Android SDK и официальной поддержке Google, Android Studio обеспечивает доступ к последним API и инструментам разработки, что позволит создать современное и технологичное приложение.

Вывод по первой главе

В данной главе рассмотрены предметная область проекта и существующие аналогичные приложения. Проведенный анализ выявил ключевые функции и инструменты, необходимые для разрабатываемого приложения.

2. ТРЕБОВАНИЯ К СИСТЕМЕ

2.1. Анализ требований к программной системе

На основе проведенного анализа предметной области и обзора аналогичных приложений, были определены следующие функциональные и нефункциональные требования к разрабатываемому приложению.

Функциональные требования к системе.

1. Регистрация и аутентификация:

– возможность регистрации новых пользователей с указанием логина, адреса электронной почты и пароля;

– вход в приложение с использованием учетных данных пользователя для обеспечения безопасности и персонализации.

2. Просмотр автомобилей:

– пользователи должны иметь возможность просматривать автомобили, выставленные администратором на доску объявлений;

– отображение подробной информации о каждом автомобиле, включая изображения, названия, технические характеристики, год выпуска, пробег и другие важные параметры.

3. Заказ автомобилей:

– возможность оформления заказа на приобретение выбранного автомобиля, включая заполнение необходимых данных;

– создание и принятие заказа на покупку автомобиля.

Нефункциональные требования к системе.

1. Надежность и стабильность:

– приложение должно быть стабильным и надежным, минимизируя возможные сбои или падения;

– должны быть предусмотрены механизмы для предотвращения потери данных и обеспечения их сохранности.

2. Производительность:

– приложение должно обеспечивать высокую отзывчивость и быструю загрузку данных, особенно при работе с большим объемом информации;

– время отклика системы должно быть минимальным, чтобы пользователи могли комфортно взаимодействовать с приложением.

3. Безопасность:

– все пользовательские данные должны быть защищены от несанкционированного доступа;

– реализация аутентификации и авторизации пользователей для защиты аккаунтов;

– критическая информация, такая как данные о платежах или личные данные, должна быть зашифрована.

4. Масштабируемость:

– приложение должно быть способным масштабироваться для работы с растущим объемом данных и увеличивающимся числом клиентов.

5. Совместимость:

– приложение должно быть совместимо с различными версиями операционной системы Android, обеспечивая доступность для широкого круга пользователей;

– учтены различные разрешения экрана и ориентации устройства для обеспечения оптимального отображения на всех устройствах.

6. Удобство использования:

– интерфейс приложения должен быть интуитивно понятным и легким для освоения пользователями;

– предусмотрены элементы навигации и подсказки, чтобы облегчить использование системы и улучшить пользовательский опыт.

2.2. Диаграмма вариантов использования

Для моделирования выделенных функциональных требований к приложению был использован язык объектного моделирования UML. С его помощью была создана диаграмма вариантов использования, отражающая отношения между актерами и прецедентами. В данной системе представлены четыре актера – клиент, администратор, новый пользователь и создатель.

Построенная диаграмма вариантов использования представлена на рисунке 1.

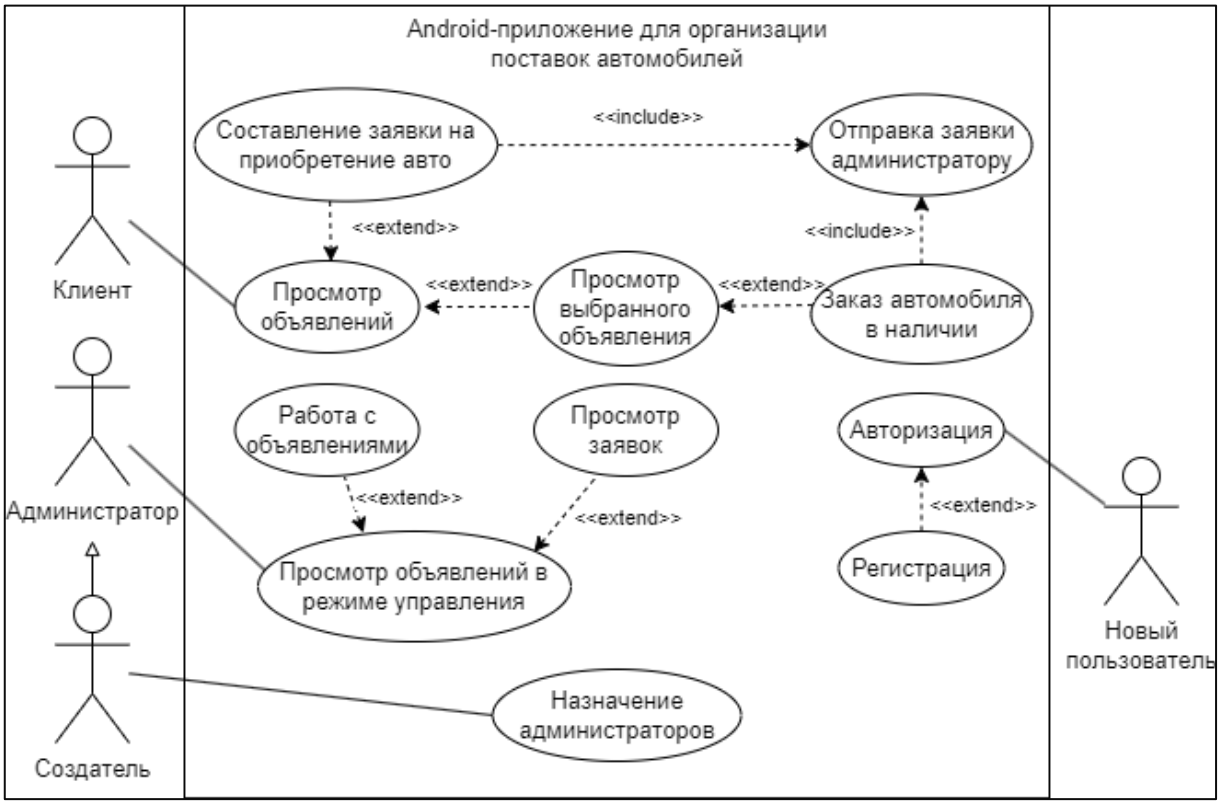

Рисунок 1 – Диаграмма вариантов использования

Ниже представлены актеры, взаимодействующие с системой.

1. Новый пользователь – это лицо, не имеющее учетной записи в системе. Этот актер может выполнить регистрацию для создания нового аккаунта в приложении.

2. Клиент – это пользователь, который планирует приобрести автомобиль. После авторизации в системе, пользователь может действовать в роли клиента.

3. Администратор – это пользователь, ответственный за управление системой и ее контентом. Администратор имеет расширенные права и может выполнять действия по управлению объявлениями и заявками.

4. Создатель – это разработчик или начальник компании, который обладает наивысшими правами в системе. Создатель может назначать других пользователей администраторами и управлять всеми аспектами системы, как и администратор.

Рассмотрим действия, которые могут совершать актеры.

1. Авторизация – пользователь вводит логин и пароль для входа в систему. В случае отсутствия учетной записи, пользователь проходит процедуру регистрации.

2. Регистрация – создание нового аккаунта в системе. Пользователь указывает логин и пароль, после чего получает возможность входа в систему.

3. Просмотр объявлений – клиент просматривает список автомобилей, доступных для приобретения. В каждом объявлении содержится подробная информация об автомобиле, включая изображения, технические характеристики, год выпуска, пробег и т.д.

4. Просмотр выбранного объявления – клиент выбирает и просматривает подробную информацию об интересующем его автомобиле. Далее он может заказать предложенное авто.

5. Заказ автомобиля в наличии – во время просмотра объявлений, клиент может выбрать интересующий его автомобиль и оформить заказ.

6. Составление заявки на приобретение авто – если клиент не нашел подходящий автомобиль в наличии, он может составить заявку на приобретение автомобиля, указав желаемую модель и марку.

7. Отправка заявки администратору – происходит в случаях, когда клиент оставил заявку на покупку автомобиля в наличии или составил заявку на приобретение отсутствующего автомобиля. Заявка отправляется администратору для обработки.

8. Просмотр объявлений в режиме управления – администратор и создатель могут просматривать и удалять объявления, размещенные в системе.

9. Работа с объявлениями – администратор или создатель могут добавлять новые объявления на доску объявлений, делая автомобили доступными для клиентов.

10. Просмотр заявок – администратор или создатель могут просматривать заявки клиентов на приобретение автомобилей, как из существующих объявлений, так и по индивидуальным заявкам.

11. Создатель может назначать клиентов администраторами, предоставляя им расширенные права для управления системой.

Вывод по второй главе

На основе анализа функциональных и нефункциональных требований к приложению была разработана диаграмма вариантов использования, которая структурированно отображает различные сценарии вариантов использования разрабатываемой системы.

Актеры, определенные в системе, включают клиентов, администраторов, новых пользователей и создателей. Каждый актер имеет свои уникальные роли и обязанности в системе, что отражено на диаграмме через соответствующие прецеденты.

Прецеденты представляют собой конкретные действия или функциональные возможности, которые могут быть выполнены в приложении, такие как регистрация нового пользователя, просмотр объявлений, добавление заявки на приобретение автомобиля и другие.

В результате диаграмма вариантов использования предоставляет понятное представление о том, как пользователи будут в будущем взаимодействовать с системой и какие возможности им будут доступны. Это может быть важно не только для процесса разработки, но и для заказчика, который бы мог убедиться в соответствии разрабатываемой системы его потребностям и ожиданиям.

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Диаграмма классов

Была разработана диаграмма классов, которая представляет собой структуру классов, используемых в приложении. Диаграмма классов помогает визуализировать отношения между классами и понять, как они взаимодействуют друг с другом в приложении, это полезный инструмент для разработчиков, позволяющий улучшить понимание кода, ускорить процесс разработки, и облегчить поддержку приложения в будущем. На рисунке 2 представлена диаграмма классов разработанного Android-приложения.

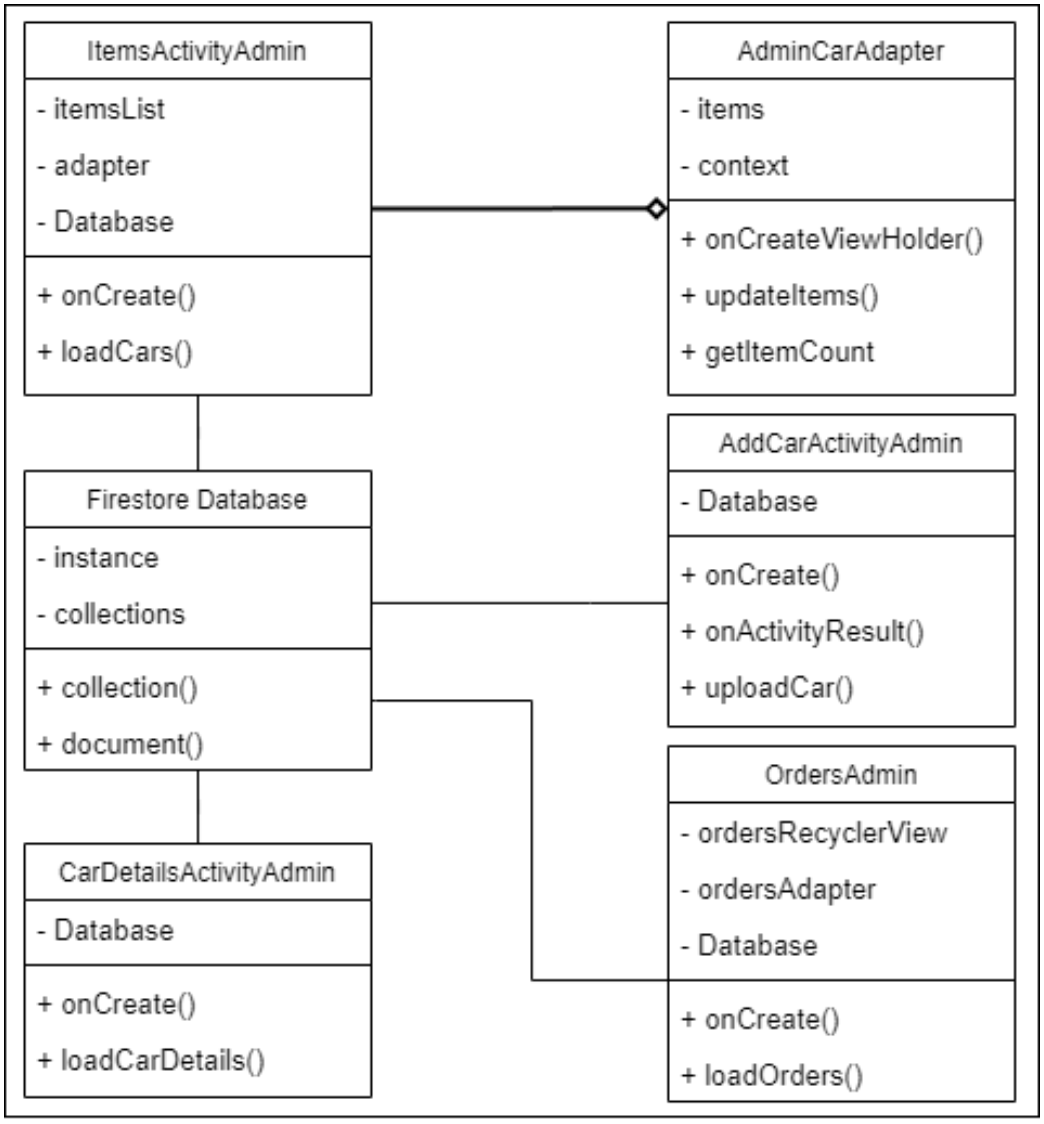

Рисунок 2 – Диаграмма классов административной части AutoSell

Рассмотрим классы более подробно.

1. Класс `ItemsActivityAdmin` отвечает за управление автомобилями. Он содержит `RecyclerView` для отображения списка автомобилей и методы `onCreate()` и `loadCars()` для инициализации активности и загрузки данных об автомобилях из базы данных Firebase. Также данный класс содержит атрибут itemsList, это элемент пользовательского интерфейса, используемый для отображения списка автомобилей в виде прокручиваемого списка. Атрибут `adapter`, связывающий данные автомобилей с элементами списка в `RecyclerView`, и атрибут `Database` –это экземпляр базы данных Firebase, используемый для взаимодействия с хранилищем данных.

2. Класс `CarDetailsActivityAdmin` отображает детали выбранного автомобиля для администратора. Он содержит элементы для отображения изображения, названия, модели, описания и цены автомобиля. В данном классе существует один отличительный от предыдущего класса метод, это `loadCarDetails()`, который используется для получения данных об автомобиле из базы данных Firebase. Атрибут `Database` и метод `onCreate()` имеют те же свойства, что и в предыдущем классе.

3. Класс `AddCarActivityAdmin` отвечает за добавление новых автомобилей. Он содержит поля для ввода названия, модели, описания, цены и загрузки изображения автомобиля, а также кнопку для добавления автомобиля. В нем присутствуют методы `onCreate()`, `onActivityResult()` и `uploadCar()`, которые используются для инициализации активности, обработки результата выбора изображения и загрузки данных об автомобиле в базу данных Firebase. Атрибут Database также используется для взаимодействия с хранилищем данных.

4. Класс `OrdersAdmin` отвечает за отображение списка заказов для администратора. Он содержит `RecyclerView` для отображения списка заказов и методы `onCreate()` и `loadOrders()` для инициализации активности и

загрузки данных о заказах из базы данных Firebase. Атрибуты данного класса: `ordersRecyclerView` – элемент пользовательского интерфейса, используемый для отображения списка заказов в виде прокручиваемого списка; `ordersAdapter` – адаптер, связывающий данные заказов с элементами списка в `RecyclerView`. `Database` – экземпляр базы данных Firebase, используемый для взаимодействия с хранилищем данных.

5. Класс `AdminCarAdapter` отвечает за отображение элементов списка автомобилей в `RecyclerView`. Содержит методы для создания, привязки и обновления элементов списка. `AdminCarAdapter` используется в `ItemsActivityAdmin` для отображения списка автомобилей. Атрибут `items` – это список автомобилей для отображения, атрибут `context` – это контекст активности, в которой используется адаптер. `OnCreateViewHolder()` – метод, создающий и возвращающий `ViewHolder` для элемента списка, `getItemCount()` – метод, возвращающий количество элементов в списке, `updateItems()` – метод для обновления данных в адаптере и уведомления о необходимости перерисовки списка.

6. Класс `FirestoreDatabase` предоставляет методы для работы с базой данных Firebase. Он используется в большинстве классов для управления данными пользователей, автомобилей и заказов. Атрибут `instance` – единственный экземпляр класса для взаимодействия с базой данных. Метод `collection()` – возвращает ссылку на коллекцию документов в базе данных; `document()` – метод, возвращающий ссылку на конкретный документ в коллекции базы данных.

Эти классы обеспечивают функциональность приложения для администраторов, позволяя им редактировать и добавлять объявления, просматривать список заявок, оставленных пользователями и просматривать автомобили, размещенные на доске объявлений.

3.2. Диаграмма последовательности

Диаграмма последовательности является наглядным представлением некоторого набора объектов на временной оси, на которой показан жизненный цикл объекта и взаимодействие актеров системы в рамках прецедента. Этот вид диаграммы помогает понять, как объекты и пользователи взаимодействуют друг с другом во времени, показывая последовательность сообщений, отправляемых и получаемых между собой.

В приложении Д представлена диаграмма последовательности, в которой отражены взаимодействия клиента, использующего приложение для заказа автомобиля. Взаимодействие в диаграмме осуществляется через окна приложения и базу данных, что позволяет наглядно представить весь процесс работы системы от начала до конца.

На диаграмме показано, как клиент просматривает доступные автомобили, выбирает интересующую его модель, просматривает ее детали и оформляет. Все эти процессы синхронизированы с базой данных Firebase, что обеспечивает надежное хранение и управление данными [13,15].

Модули и их функциональность представлены ниже.

1. `ItemsActivity`. Модуль позволяет пользователю просматривать список доступных автомобилей. При запуске активность загружает список автомобилей из Firebase Firestore с помощью метода `loadItems()`. Список отображается в виде `RecyclerView`, где каждый элемент представляет собой карточку автомобиля. Пользователь может выбрать автомобиль и нажать на кнопку «Посмотреть», чтобы увидеть подробности.

2. `CarDetailsActivity`. Модуль отображает подробную информацию об автомобиле. При запуске активность загружает данные конкретного автомобиля из Firebase Firestore по его идентификатору. Отображает изображение автомобиля, его название, модель, описание и цену. Включает кнопку «Заказать автомобиль», которая открывает `OrderActivity` для оформления заказа.

3. `OrderActivity`. Модуль позволяет пользователю оформить заказ на выбранный автомобиль. Пользователь заполняет форму с именем, электронной почтой и номером телефона. При нажатии на кнопку «Отправить» данные заказа вместе с названием и моделью автомобиля отправляются в Firebase Firestore. Пользователь получает уведомление о успешном оформлении заказа.

Таким образом, AutoSell является комплексным и многофункциональным приложением для продажи автомобилей, которое способно обеспечивать пользователям удобную аутентификацию, широкий выбор автомобилей, прозрачный процесс оформления заказов и эффективное управление ими. Приложение разработано с учетом современных требований и стандартов, что позволяет обеспечить высокое качество обслуживания и удовлетворить потребности клиентов сервиса.

Программа предоставляет удобные и интуитивно понятные интерфейсы, что делает процесс использования приложения приятным и простым для всех категорий пользователей. В частности, клиенты могут легко создавать учетные записи, входить в систему, просматривать доступные автомобили, получать детальную информацию о каждом автомобиле и оформлять заказы. Этот процесс сделан максимально простым, ведь это позволяет пользователям быстро и без затруднений выполнить все необходимые действия и с легкостью заказать автомобиль.

Благодаря использованию современных технологий и библиотек, таких как Firebase Firestore для хранения и управления данными, приложение AutoSell обеспечивает высокую надежность и производительность. Это позволяет пользователям получать доступ к актуальной информации и быстро выполнять все необходимые действия.

3.3. Диаграммы деятельности

Также были разработаны диаграммы деятельности, описывающие процессы работы приложения. Они помогают определить последовательность выполнения операций, а также выявить возможные узкие места или ошибки в логике работы программ. На диаграмме деятельности, которая представлена на рисунке 3, описана логика процедур, происходящих во время входа в приложение.

Она начинается с того, что пользователь находится на экране входа в приложение, после чего он вводит свои данные (логин и пароль), и нажимает кнопку для входа в приложение. Приложение начинает проверку введенных данных, после чего база данных получает запрос на предоставление данных об аккаунте и передает их. Система проверяет, являются ли введенные данные корректными, и, если данные правильны, то система загружает данные пользователя. Диаграмма заканчивается, когда пользователь попадает на главную страницу. Если данные некорректны, система показывает сообщение об ошибке. Диаграмма заканчивается, когда пользователь видит сообщение об ошибке.

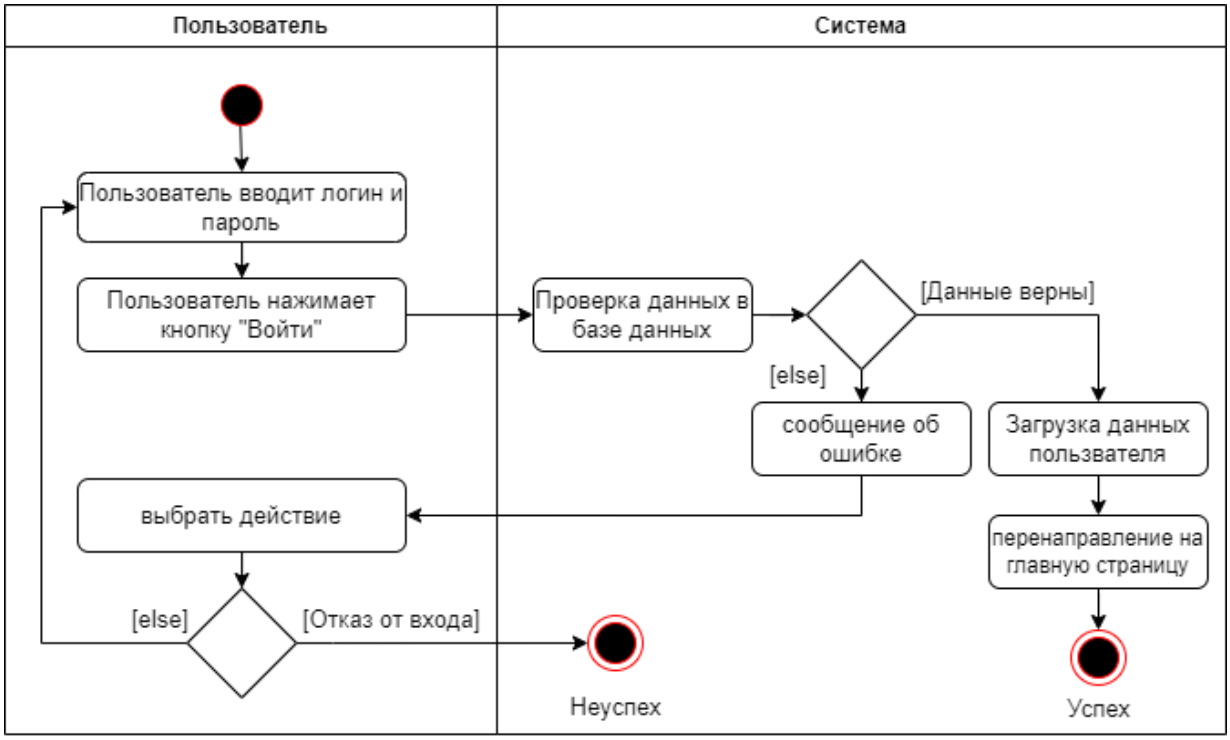

Рисунок 3 – Диаграмма деятельности «Вход в приложение»

Рисунок 4 – Диаграмма деятельности «Создание заявки»

Диаграмма «Создание заявки», представленная на рисунке 4, начинается с того, что пользователь находится на странице с объявлениями, после чего пользователь выбирает интересующий его автомобиль и нажимает на кнопку «Посмотреть» для просмотра информации об автомобиле. Далее приложение отображает страницу с деталями об автомобилях, пользователь просматривает интересующего его информацию и нажимает кнопку «Заказать». Далее приложение показывает экран для ввода данных заявки, пользователь заполняет окна и нажимает на кнопку «Отправить». Система сохраняет данные о заявке в базу данных, и пользователь перенаправляется обратно на страницу с автомобилями.

Вывод по третьей главе

В данной главе была представлена архитектура системы приложения AutoSell, включающая диаграмму классов, диаграмму последовательности и диаграмму деятельности.

Диаграмма классов отображает ключевые компоненты приложения и их взаимодействия. Ее компоненты обеспечивают различные аспекты функциональности приложения, такие как редактирование и просмотр объявлений [5-7].

Диаграмма последовательности иллюстрирует процесс взаимодействия клиентов с системой, показывая, как пользователь просматривает доступные автомобили, выбирает интересующую модель, просматривает ее детали и оформляет заказ.

Диаграмма деятельности описывает процессы входа в систему и создания заявок на покупку автомобиля, помогая выявить возможные узкие места и улучшить логику работы приложения.

Представленная архитектура системы играет ключевую роль в обеспечении структурированного и понятного представления взаимодействия компонентов. Это значительно облегчает процесс разработки, так как появляется возможность легко и доступно понимать, как различные части разрабатываемой системы взаимодействуют друг с другом. Такой подход упрощает поддержку приложения, поскольку четко определенные компоненты и их взаимодействия позволяют быстро идентифицировать и устранять проблемы. Кроме того, структурированная архитектура повышает качество и надежность приложения. Четкое разграничение ответственности между компонентами снижает вероятность возникновения ошибок и повышает устойчивость системы к изменениям.

4. РЕАЛИЗАЦИЯ

В данной главе будет подробно рассмотрена реализация приложения AutoSell, включая все этапы разработки и использованные ее в процессе технологии. Приложение будет предназначено для организации и управления процессами покупки и продажи автомобилей, обеспечивая удобный интерфейс для пользователей и администраторов.

В этой главе будут рассмотрены следующие аспекты.

1. Средства инструментов разработки, используемые для создания приложения.
2. Архитектура приложения, которая позволяет организовать и разделить логику работы.
3. Реализация базы данных, необходимой для хранения информации о пользователях, автомобилях и заказах.
4. Реализация слоя презентации, обеспечивающего взаимодействие пользователя с приложением.
5. Пользовательский интерфейс, который делает приложение интуитивно понятным и удобным для использования.

Эти аспекты будут подробно описаны с примерами кода, скриншотами и схемами для лучшего понимания процесса разработки и функциональности приложения.

4.1. Средства инструментов разработки

Для разработки приложения AutoSell была выбрана интегрированная среда разработки (IDE) Android Studio. Это современный и мощный инструмент, который предоставляет разработчикам все необходимые возможности для создания высококачественных приложений под платформу Android. Использование Android Studio обусловлено рядом причин ниже.

1. Эмуляция устройств. Возможность тестировать приложение на различных устройствах и версиях Android без необходимости физического доступа к ним.

2. Скорость сборки приложения. Обеспечивается быстрая и эффективная сборка проектов, что значительно ускоряет процесс разработки.

3. Удобный XML редактор. Предоставляются мощные инструменты для создания и редактирования пользовательского интерфейса.

4. Поддержка рендера средствами GPU. Улучшается производительность приложения за счет использования графического процессора.

5. Бесплатное использование. Android Studio является бесплатным инструментом, доступным для всех разработчиков.

6. Поддержка всех платформ и версий Android. Обеспечивается совместимость с различными устройствами и версиями операционной системы Android.

На рисунке 5 продемонстрирован скриншот среды Android Studio.

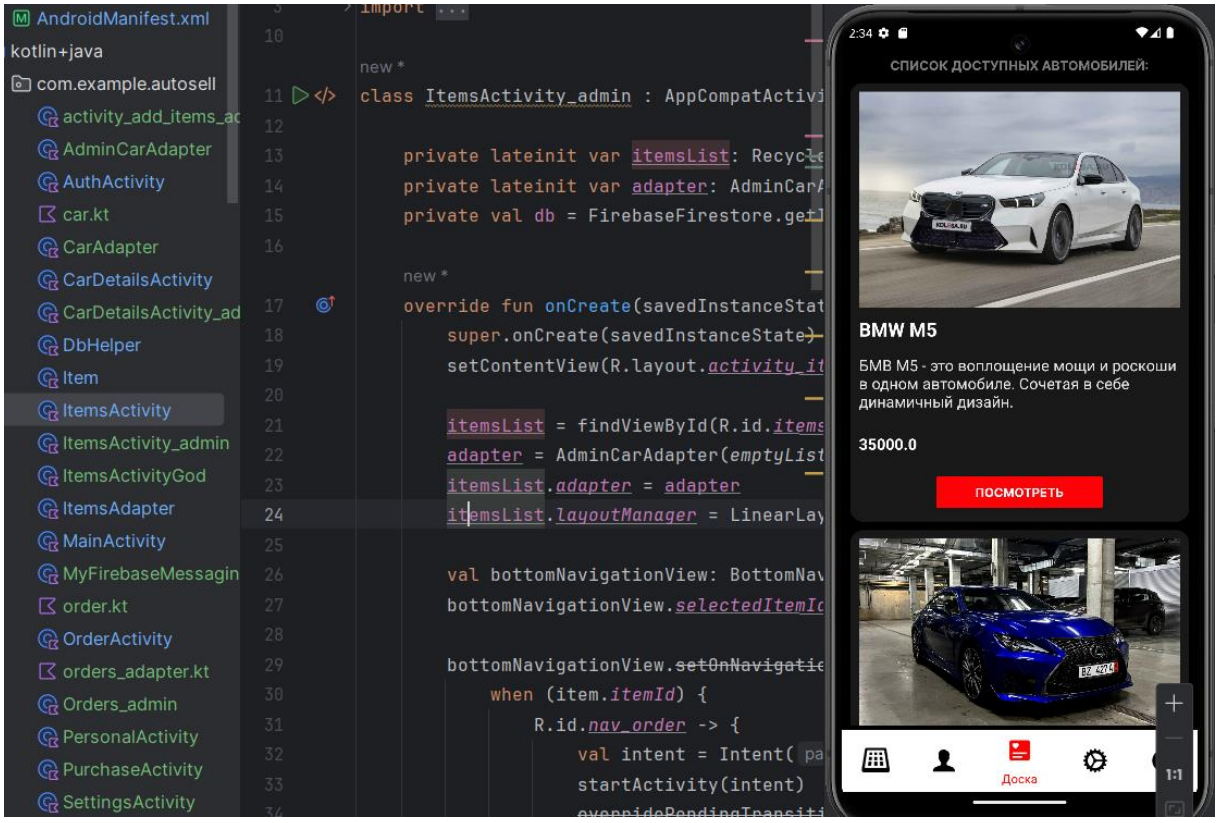

Рисунок 5 – Скриншот среды Android Studio

Для написания кода использовался язык программирования Kotlin. Kotlin – это современный язык, разработанный компанией JetBrains, который стал официальным языком для разработки Android-приложений. Он предоставляет множество преимуществ, таких как:

1) лаконичный и выразительный синтаксис;
2) безопасность типов;
3) поддержка функционального программирования;
4) высокая совместимость с языком Java, что позволяет использовать существующий код и библиотеки на платформе Android.

Также в процессе разработки были использованы различные библиотеки и инструменты, которые упрощают разработку и обеспечивают дополнительные функциональные возможности. Они перечислены ниже.

1. Android Jetpack: набор библиотек и инструментов от Google, включающий компоненты, такие как ViewModel, LiveData, Room и RecyclerView, использованные в приложении AutoSell.
2. Firebase: облачная платформа, предоставляющая функции для работы с базой данных, аутентификации пользователей и многого другого.
3. Retrofit: библиотека для выполнения HTTP-запросов и обработки ответов, использованная для взаимодействия с внешними API.
4. RecyclerView: компонент для эффективного отображения списка элементов с возможностью плавной прокрутки.

Эти инструменты и библиотеки значительно упрощают процесс разработки, обеспечивая высокую производительность и качество приложения. Они предоставляют разработчикам мощные возможности для создания удобных и функциональных приложений на платформе Android. Благодаря этим ресурсам разработчики могут быстрее и эффективнее решать задачи, связанные с проектированием интерфейсов, обработкой данных и обеспечением безопасности приложений.

4.2. Архитектура приложения

В приложении AutoSell была использована архитектура MVVM (Model-View-ViewModel), что позволило разделить логику приложения на отдельные компоненты, обеспечивая удобную организацию кода и улучшенную тестируемость [10,14].

Компоненты, входящие в архитектуру MVVM, описаны ниже.

1. Модель (Model): отвечает за предоставление данных и бизнес-логику приложения. В нашем случае модель представлена классами `Car`, `Order` и `User`, которые представляют соответствующие объекты данных.

2. Представление (View): отображает данные пользователю и отвечает за визуальное представление интерфейса. В нашем приложении представление реализовано через XML-разметку файлов, таких как `activity_main.xml`, `activity_order.xml` и другие.

3. Представитель (ViewModel): связывает модель и представление, предоставляет данные для отображения и обрабатывает пользовательские действия. В приложении AutoSell классы `ProfileViewModel` и `ItemsViewModel` выступают в качестве представителей.

Архитектура MVVM позволяет легко управлять данными и взаимодействием с пользователем, а также улучшает тестируемость кода за счет разделения ответственности между компонентами.

Архитектура MVVM представлена на рисунке 6.

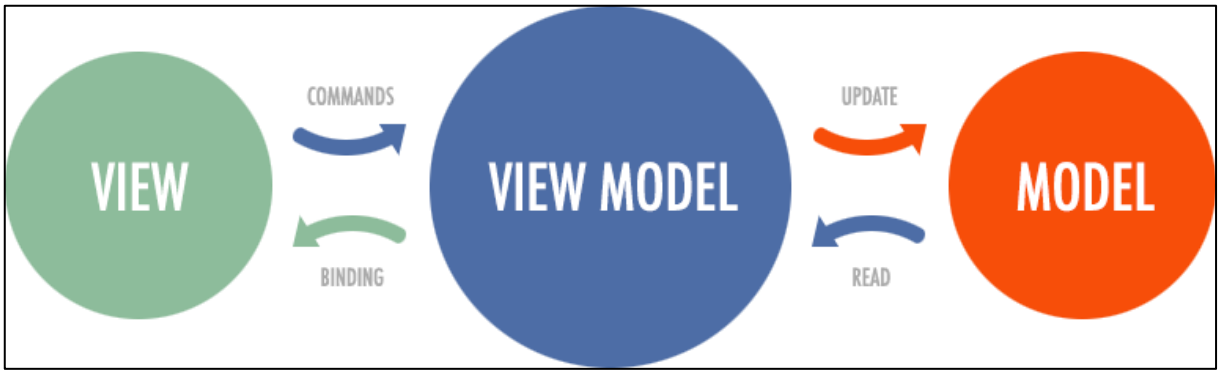

Рисунок 6 – Архитектура MVVM

4.3. Реализация базы данных

После тщательного изучения различных видов баз данных, было принято решение использовать базу данных Firebase для проекта [9]. Это решение основано на ряде факторов, которые делают Firebase оптимальным выбором для моего проекта. Эти факторы описаны ниже.

1. Реальное время. Firebase обеспечивает возможность работать с данными в реальном времени, что идеально подходит для приложений, где требуется мгновенная синхронизация данных между клиентами. Это особенно полезно для функций чата, обновления в реальном времени и коллаборативной работы.

2. Хорошая документация и поддержка. Firebase имеет обширную документацию и активное сообщество разработчиков, что облегчает процесс изучения и использования его функций. Также доступна официальная поддержка со стороны Google, что помогает решать любые возникающие проблемы.

3. Интеграция с другими сервисами Firebase. Firebase предоставляет широкий спектр инструментов и сервисов, таких как аутентификация пользователей, облачное хранение файлов, аналитика и многое другое. Использование Firebase для базы данных позволит легко интегрировать эти сервисы в приложение и сделать его более функциональным и удобным для пользователей.

Благодаря этим характеристикам Firebase обеспечит стабильную и эффективную работу моего приложения, удовлетворяя все текущие и будущие потребности проекта.

Для реализации базы данных была разработана схема, представленная на рисунке 7.

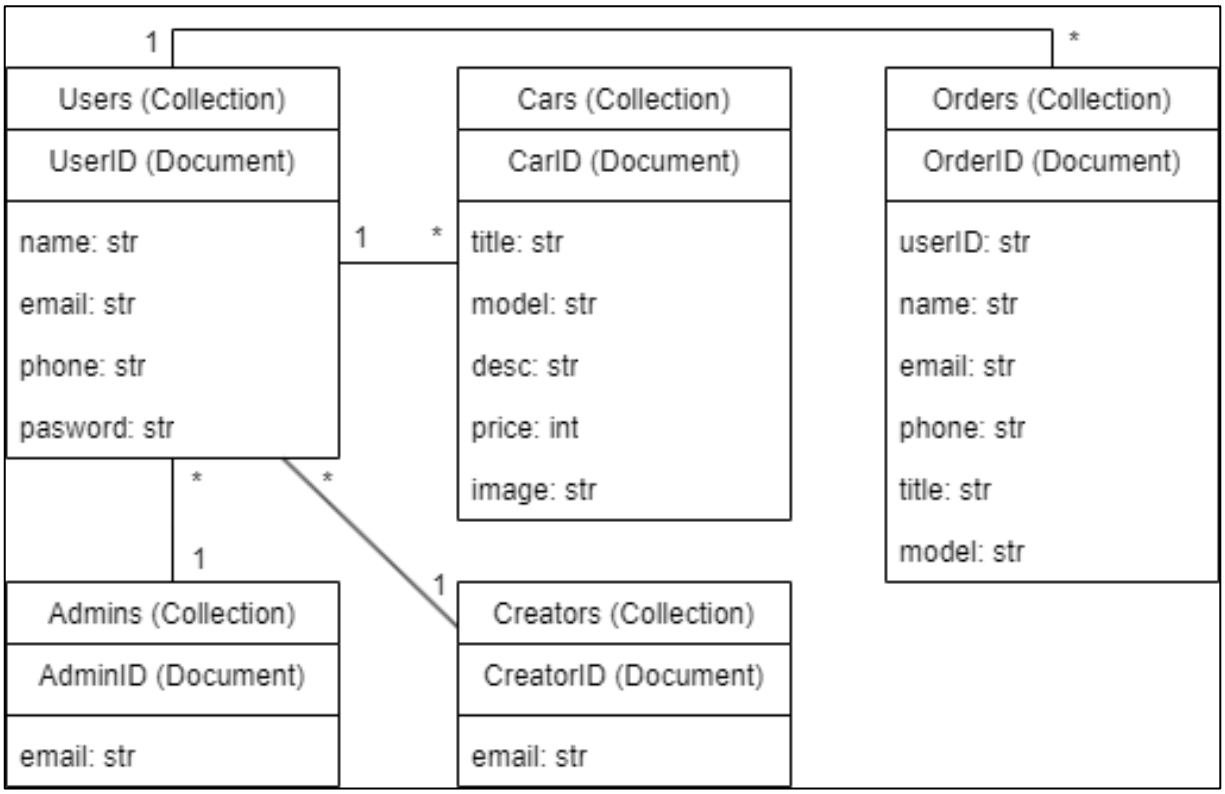

Рисунок 7 – Схема базы данных

Схема базы данных включает в себя несколько ключевых коллекций и таблиц, каждая из которых предназначена для хранения и управления определенным типом данных.

Ниже приведены коллекции и таблицы базы данных.

Коллекция «Пользователи (`Users`)» используется для хранения данных пользователей, их аутентификации и авторизации. Она включает следующие поля: `UserID` (уникальный идентификатор пользователя), `email` (электронная почта пользователя), `password` (пароль пользователя), `name` (имя пользователя) и `phone` (номер телефона пользователя).

Коллекция «Автомобили (`Cars`)» хранит информацию обо всех доступных для продажи автомобилях, включая такие характеристики, как `CarID` (уникальный идентификатор автомобиля), `title` (название автомобиля), `model` (модель автомобиля), `desc` (описание автомобиля), `price` (цена автомобиля), и `image` (URL изображения автомобиля).

Коллекция «Заказы (`Orders`)» используется для хранения информации о всех сделанных заказах. Она содержит поля: `OrderID` (уникальный номер заказа, `UserID` (идентификатор пользователя, сделавшего заказ), `name` (имя пользователя, сделавшего заказ), `email` (электронная почта пользователя, связанная с учетной записью), `phone` (номер телефона пользователя), `title` (название автомобиля, заказанного пользователем) и `model` (модель автомобиля, заказанного пользователем).

Коллекции `Admins` и `Creators` используются для аутентификации пользователей, путем проверки электронной почты входящих в приложение пользователей.

Преимущества предложенной схемы базы данных описаны ниже.

1. Структурированность данных. Разделение данных на коллекции, такие как пользователи, автомобили и заказы, обеспечивает четкую структуру и легкость управления данными. Это позволяет легко находить, изменять и удалять информацию, поддерживая целостность и согласованность данных.

2. Масштабируемость. База данных спроектирована для легкого масштабирования, что позволяет добавлять новые поля и коллекции без значительных изменений в структуре. Это поддерживает рост данных и функциональности приложения. Например, можно добавить новые характеристики для автомобилей или детали для заказов без переработки всей базы данных [17].

3. Безопасность. Разграничение данных пользователей и администраторов, а также надежные методы аутентификации и авторизации обеспечивают высокий уровень безопасности данных. Это включает шифрование, контроль доступа и регулярное обновление механизмов безопасности, защищая данные пользователей от несанкционированного доступа и обеспечивая доверие к приложению.

4.4. Реализация аутентификации через Firebase

Для добавления авторизации клиентов в базу данных Firebase были предприняты шаги, описанные ниже.

1. Настройка аутентификации в Firebase Console: в настройках проекта в консоли Firebase был выбран и настроен способ аутентификации по электронной почте и паролю. Список доступных способов регистрации представлен на рисунке 8.

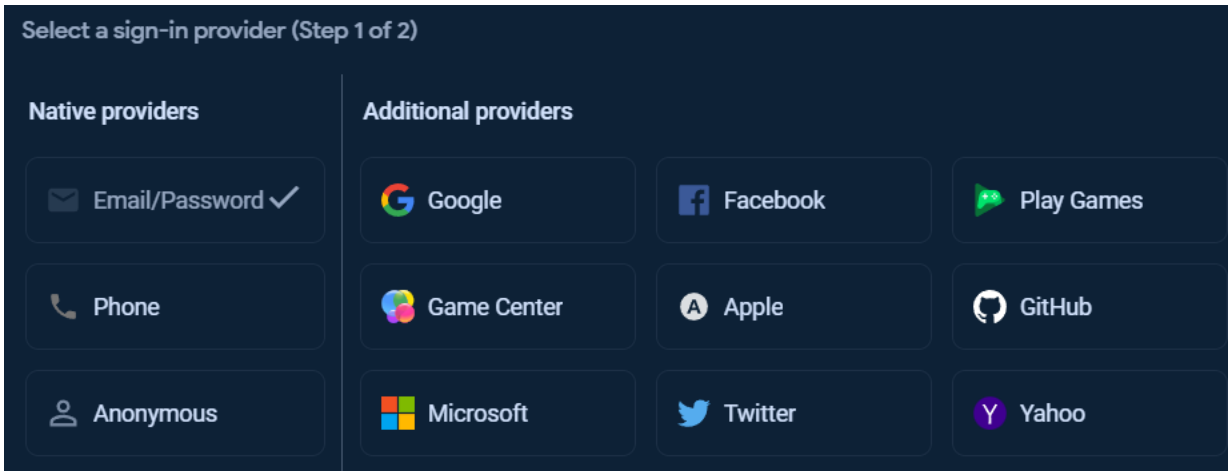

Рисунок 8 – Способы организации регистрации в Firebase

2. Интеграция SDK Firebase в приложение: для взаимодействия с базой данных Firebase из мобильного приложения была произведена интеграция SDK Firebase. Это включало добавление необходимых зависимостей в файл конфигурации проекта и настройку прав доступа.

3. Регистрация новых пользователей: после интеграции SDK Firebase в приложение была добавлена функциональность регистрации новых пользователей. Пользователи могли создавать учетные записи, используя свою электронную почту и устанавливая пароль для входа.

4. Хранение учетных данных в Firebase: после успешной аутентификации пользователей их учетные данные (электронная почта и пароль) были сохранены в базе данных Firebase. Firebase автоматически обеспечивает безопасное хранение и управление учетными данными пользователей. Добавленные в базу данных пользователи показаны на рисунке 9.

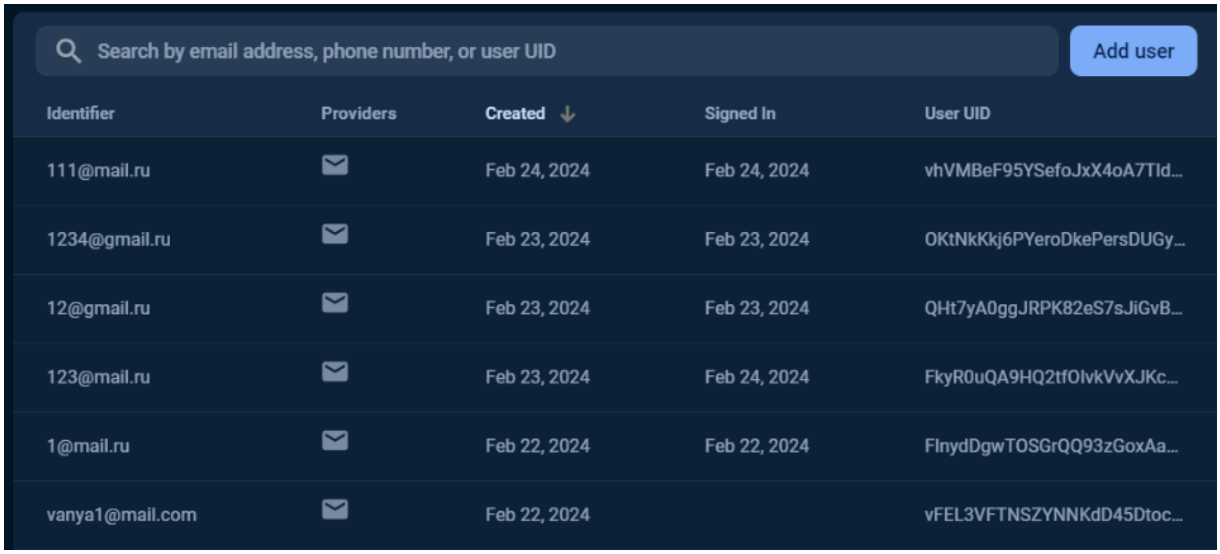

Рисунок 9 – Добавленные в Firebase пользователи

Таким образом, внедрение авторизации клиентов с использованием базы данных Firebase обеспечило высокий уровень безопасности учетных записей пользователей. Firebase Authentication предоставляет надежную систему аутентификации, позволяющую пользователям безопасно регистрироваться и входить в свои аккаунты с помощью электронной почты и пароля [16].

Firebase автоматически обрабатывает и защищает данные учетных записей, что исключает возможность несанкционированного доступа и гарантирует сохранность конфиденциальной информации пользователей. Интеграция Firebase в приложение не только упростила процесс управления учетными записями, но и значительно повысила общую безопасность системы.

Благодаря этим мерам пользователи могут быть уверены в надежности и безопасности своих данных, что является ключевым аспектом для приложений, связанных с покупкой и продажей автомобилей. Надежное хранение и защита данных предотвращают несанкционированный доступ и обеспечивают конфиденциальность личной информации.

4.5. Загрузка автомобилей в базу данных

Для добавления автомобилей клиентов в базу данных и хранения фотографий автомобилей в хранилище, необходимо было выполнить следующие шаги, представленные ниже.

1. Выбор структуры базы данных: на начальном этапе определялась структура базы данных для хранения информации об автомобилях. Были учтены различные параметры, такие как марка, модель, год выпуска, цена и другие важные характеристики, чтобы обеспечить полное и детальное описание каждого автомобиля.

2. Создание базы данных в Firebase: в Firebase Cloud Firestore была создана новая база данных для хранения информации об автомобилях. Для каждого автомобиля создавались соответствующие коллекции и документы, в которых хранились все необходимые данные. Поля базы данных представлены на рисунке 10.

Рисунок 10 – Поля данных об автомобилях

3. Хранение фотографий автомобилей в хранилище Firebase: для хранения фотографий автомобилей было использовано надежное хранилище Firebase. После загрузки фотографий они сохранялись в хранилище Firebase, а ссылки на них добавлялись в соответствующие документы базы данных. Это обеспечивало централизованное хранение изображений и их легкую доступность. Картинки, добавленные в облачное хранилище, показаны на рисунке 11.

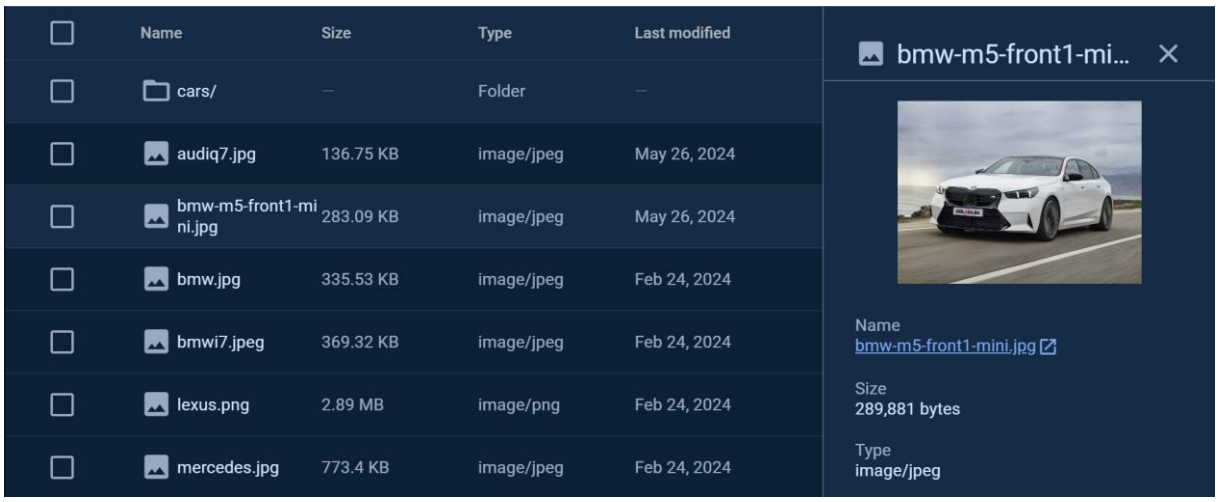

Рисунок 11 – Изображения, добавленные на облачное хранилище

4. Отображение информации об автомобилях в приложении: после успешного добавления данных об автомобилях в базу данных и загрузки фотографий в хранилище Firebase, информация об автомобилях была отображена в мобильном приложении. Пользователи получили возможность просматривать доступные автомобили, их характеристики и фотографии, полученные из базы данных Firebase. На рисунке 12 показано изображение, загруженное из облачной базы данных, а также информация об автомобиле.

Таким образом, благодаря использованию базы данных Firebase и хранилища для фотографий была создана высокоэффективная система хранения и отображения информации об автомобилях в мобильном приложении. Это решение обеспечило пользователям легкий доступ к обширному каталогу автомобилей, позволяя им быстро находить интересующие модели и получать подробную информацию о каждом автомобиле, включая характеристики, изображения и другие важные данные, прямо в приложении. Такая интеграция существенно улучшила удобство использования и удовлетворенность клиентов, делая процесс выбора и покупки автомобилей максимально простым и интуитивно понятным.

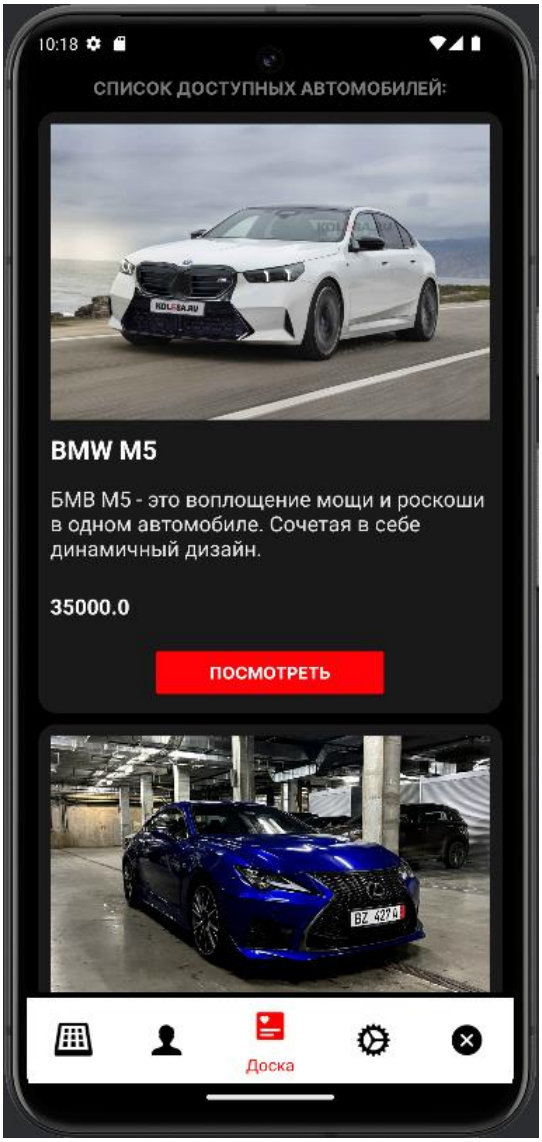

Рисунок 12 – Результат загрузки данных из Firebase

4.6. Реализация администратора

Для эффективного управления автомобилями в приложении была внедрена функция администратора, предоставляющая возможность добавлять, и удалять объявления о продаже автомобилей. Интерфейс администратора представлен на рисунке 13. Реализация этой функции была осуществлена с использованием Realtime Database, предоставляемой Firebase. Она показана на рисунке 14.

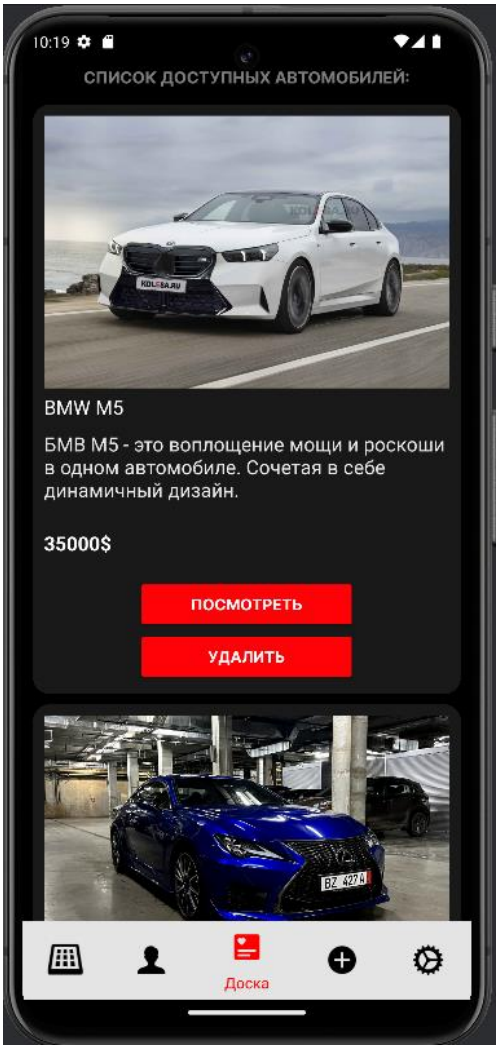

Рисунок 13 – Интерфейс администратора

Рисунок 14 – Realtime Database

Администратору доступен специальный интерфейс для добавления новых автомобилей. Он указан на рисунке 15.

После успешного добавления нового автомобиля администратором, объявление становится доступным для всех пользователей приложения. Объявления о продаже автомобилей отображаются в общем списке автомобилей, где пользователи могут просматривать подробную информацию о каждом автомобиле. Реализация функции `activity_add_items_admin`, которая отвечает за добавление объявления, показана в листинге 1.

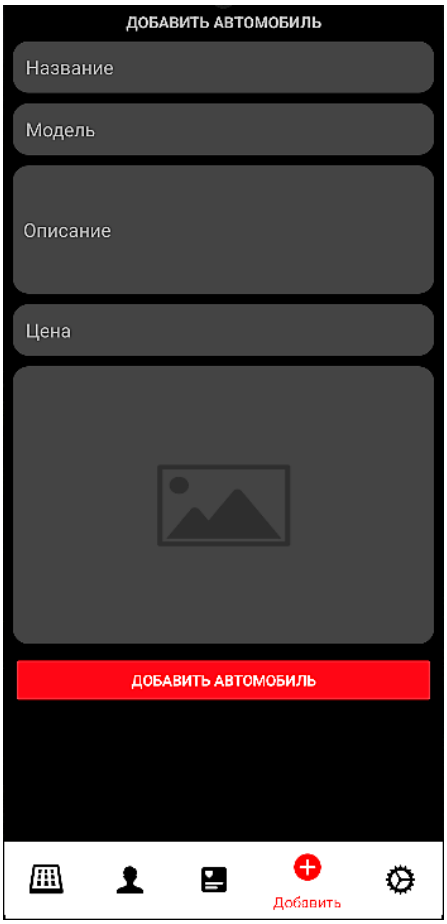

Рисунок 15 – Интерфейс добавления автомобиля администратором

Листинг 1 – Класс `activity_add_items_admin`

```
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_add_items_admin)
mAuth = FirebaseAuth.getInstance()
firestore = FirebaseFirestore.getInstance()
storage = FirebaseStorage.getInstance()
titleEditText = findViewById(R.id.edit_text_title)
modelEditText = findViewById(R.id.edit_text_model)
descriptionEditText = findViewById(R.id.edit_text_description)
priceEditText = findViewById(R.id.edit_text_price)
imageView = findViewById(R.id.image_view)
uploadButton = findViewById(R.id.button_upload)
imageView.setOnClickListener {
```

```
val intent = Intent(Intent.ACTION_PICK)
intent.type = "image/*"
startActivityForResult(intent, 1)}
uploadButton.setOnClickListener {
uploadCar()}}
override fun onActivityResult(requestCode: Int, resultCode: Int, data: In-
tent?) {
super.onActivityResult(requestCode, resultCode, data)
if (requestCode == 1 && resultCode == RESULT_OK) {
imageUri = data?.data
imageView.setImageURI(imageUri)}}
private fun uploadCar() {
val title = titleEditText.text.toString()
val model = modelEditText.text.toString()
val description = descriptionEditText.text.toString()
val price = priceEditText.text.toString().toDoubleOrNull()
if (title.isNotEmpty() && model.isNotEmpty() && description.isNotEmpty() &&
price != null && imageUri != null) {
val carId = firestore.collection("cars").document().id
val storageRef = storage.reference.child("cars/${carId}.jpg")
storageRef.putFile(imageUri!!)
.addOnSuccessListener {
storageRef.downloadUrl.addOnSuccessListener { uri ->
val car = Car(carId, title, model, description, price, uri.toString())
firestore.collection("cars").document(carId).set(car).addOnSuccessListener{
Toast.makeText(this, "Автомобиль добавлен", Toast.LENGTH_SHORT).show()}.ad-
dOnFailureListener {
Toast.makeText(this, "Ошибка добавления автомобиля",
Toast.LENGTH_SHORT).show()}}}} else {
Toast.makeText(this, "Пожалуйста, заполните все поля и добавьте изображе-
ние", Toast.LENGTH_SHORT).show()}}}
```

4.7 Реализация функции создателя

Функция создателя в приложении AutoSell предоставляет пользователю с высшими правами возможность добавлять новых администраторов, управлять объявлениями и следить за всеми аспектами системы. Создатель обладает правами выше администратора и может назначать других пользователей администраторами, что расширяет их возможности по управлению приложением. Функция добавления администраторов показана в листинге 2.

Листинг 2 – Функция добавления администратора

```
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_items_god)
database = FirebaseDatabase.getInstance().reference.child("ad-
mins").child("adminsDB")
newAdminEmailEditText = findViewById(R.id.new_admin_email)
addAdminButton = findViewById(R.id.item_list_add_admin_button)
addAdminButton.setOnClickListener {
val newAdminEmail = newAdminEmailEditText.text.toString().trim()
if (newAdminEmail.isNotEmpty()) {
addNewAdmin(newAdminEmail) } else {
```

```
Toast.makeText(this, "Введите email администратора",
Toast.LENGTH_LONG).show()}}}
private fun addNewAdmin(email: String) {
val newAdminRef = database.push()
newAdminRef.child("Email").setValue(email).addOnSuccessListener {
Toast.makeText(this, "Администратор $email добавлен",
Toast.LENGTH_LONG).show()
newAdminEmailEditText.text.clear()
}.addOnFailureListener {
Toast.makeText(this, "Ошибка добавления администратора",
Toast.LENGTH_LONG).show()}}}
```

Для реализации функции создателя была выполнена настройка базы данных Firebase, где в коллекции `Creator` хранится электронная почта создателя. Сначала было необходимо проверить, является ли пользователь создателем при входе в систему, а затем предоставить ему доступ к специальному интерфейсу управления.

При авторизации пользователя сначала проверяется, является ли он создателем. Если проверка подтверждает, что пользователь является создателем, он перенаправляется на соответствующий экран управления. Данная функция представлена в листинге 3.

Листинг 3 – Функция авторизации создателя

```
private fun authenticateUser(email: String, password: String) {
val db = DbHelper(this, null)
val isAuth = db.getUser(email, password)
if (isAuth) {
checkIfCreator(email) } else {
Toast.makeText(this, "Пользователь $email не авторизован",
Toast.LENGTH_LONG).show()}}
private fun checkIfCreator(email: String) {
val creatorRef = database.child("Creator").child("creator1")
creatorRef.addListenerForSingleValueEvent(object : ValueEventListener {
override fun onDataChange(dataSnapshot: DataSnapshot) {
val creatorEmail = dataSnapshot.child("Email").getValue(String::class.java)
if (creatorEmail == email) {
Toast.makeText(this@AuthActivity, "Пользователь $email авторизован как со-
здатель", Toast.LENGTH_LONG).show()
val intent = Intent(this@AuthActivity, ItemsActivityGod::class.java)
startActivity(intent) } else {
checkIfAdmin(email)}}
override fun onCancelled(databaseError: DatabaseError) {
Toast.makeText(this@AuthActivity, "Ошибка базы данных: ${databaseError.mes-
sage}", Toast.LENGTH_LONG).show()}})}
```

Таким образом, функция создателя в приложении AutoSell реализована с учетом всех необходимых требований, что обеспечивает удобное и безопасное управление системой на высшем уровне.

4.8. Реализация функции уведомлений

Для улучшения функциональности приложения AutoSell была внедрена функция отправки уведомлений пользователям при добавлении новых объявлений. Это позволяет пользователям своевременно получать информацию о новых автомобилях, доступных для покупки.

Настройка Firebase Cloud Messaging (FCM)

Для реализации данной функции использовалась платформа Firebase Cloud Messaging (FCM). Сначала были произведены настройки в консоли Firebase, перечисленные далее.

1. Добавление проекта. В консоли Firebase был создан проект для приложения AutoSell.

2. Настройка Cloud Messaging API. В разделе Cloud Messaging были включены необходимые API для отправки и получения сообщений.

3. Получение сервисного ключа. Для аутентификации и отправки сообщений FCM был получен JSON файл сервисного аккаунта, содержащий все необходимые ключи и идентификаторы.

Настройка приложения

Были выполнены следующие шаги для настройки приложения AutoSell для работы с FCM.

1. Добавление зависимостей. В файл `build.gradle` были добавлены зависимости для Firebase Cloud Messaging и других необходимых библиотек.

2. Настройка манифеста: В файле `AndroidManifest.xml` были добавлены разрешения и сервисы для обработки сообщений FCM.

Для реализации функции уведомлений в приложении были созданы два основных компонента в `MainActivity` происходит подписка пользователя на топик «`all`», чтобы он мог получать уведомления о новых объявлениях (подписка на уведомления показана в листинге 4), а в `MyFirebaseMessagingService` обрабатываются входящие уведомления и отображаются пользователю. Данный класс представлен в листинге 5.

Листинг 4 – Подписка на уведомления

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
FirebaseMessaging.getInstance().subscribeToTopic("all")
.addOnCompleteListener(task -> {
String msg = "Subscribed to topic";
if (!task.isSuccessful()) {
msg = "Subscription failed";}
Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show();});}
```

Листинг 5 – Обработка входящих уведомлений

```
class MyFirebaseMessagingService : FirebaseMessagingService() {
override fun onMessageReceived(remoteMessage: RemoteMessage) {
super.onMessageReceived(remoteMessage)
// Обработка входящих сообщений
Log.d("FCM", "Message received from: ${remoteMessage.from}")
remoteMessage.notification?.let {
sendNotification(it.title, it.body)}}
private fun sendNotification(title: String?, messageBody: String?) {
val intent = Intent(this, MainActivity::class.java)
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
val pendingIntent = PendingIntent.getActivity(this, 0, intent,
PendingIntent.FLAG_ONE_SHOT or PendingIntent.FLAG_IMMUTABLE)
val channelId = "DefaultChannel"
val defaultSoundUri = RingtoneManager.getDefaultUri(Ring-
toneManager.TYPE_NOTIFICATION)
val notificationBuilder = NotificationCompat.Builder(this, chan-
nelId).setSmallIcon(R.drawable.ic_launcher_foreground).setContentTitle(ti-
tle).setContentText(messageBody).setAutoCancel(true).setSound(de-
faultSoundUri).setContentIntent(pendingIntent)
val notificationManager = getSystemService(NOTIFICATION_SERVICE) as Notifi-
cationManager
notificationManager.notify(0, notificationBuilder.build())}
override fun onNewToken(token: String) {
super.onNewToken(token)
Log.d("FCM", "New token: $token")}}
```

Когда пользователь добавляет новое объявление, сервер отправляет уведомление всем подписанным пользователям. Это реализовано с помощью Python-скрипта, который использует FCM для отправки сообщений. Код скрипта показан в листинге 6.

Листинг 6 – Отправление сообщения на языке Python

```
import json
import requests
from google.oauth2 import service_account
from google.auth.transport.requests import Request
SERVICE_ACCOUNT_FILE = 'C:/Users/UpGrade/AndroidStudioProjects/AutoSell/au-
tosell-213a3-firebase-adminsdk-gm5gk-c80cb1f449.json'
SCOPES = ['https://www.googleapis.com/auth/firebase.messaging']
credentials = service_account.Credentials.from_service_account_file(
SERVICE_ACCOUNT_FILE, scopes=SCOPES)
request = Request()
```

```
credentials.refresh(request)
access_token = credentials.token
url = 'https://fcm.googleapis.com/v1/projects/autosell-213a3/messages:send'
headers = {
'Content-Type': 'application/json',
'Authorization': 'Bearer ' + access_token}
# Тело сообщения message = {"message": {"topic": "all","notification": {
"title": "Новое объявление",
"body": "Добавлен новый автомобиль. Проверьте его в приложении!"}}}
response = requests.post(url, headers=headers, data=json.dumps(message))
print(response.status_code, response.text)
```

Проверка работы функции уведомлений.

Для проверки работы функции уведомлений были выполнены следующие шаги.

1. Запуск приложения на устройстве. Приложение было установлено и запущено на физическом устройстве и эмуляторе. Пользователи успешно подписались на топик `all`.

2. Запуск скрипта отправки уведомлений. В терминале был запущен скрипт `send_notification.py`. После его выполнения в консоли было выведено сообщение «`200`», что свидетельствует об успешной отправке уведомления. Результат использования скрипта показан на рисунке 16.

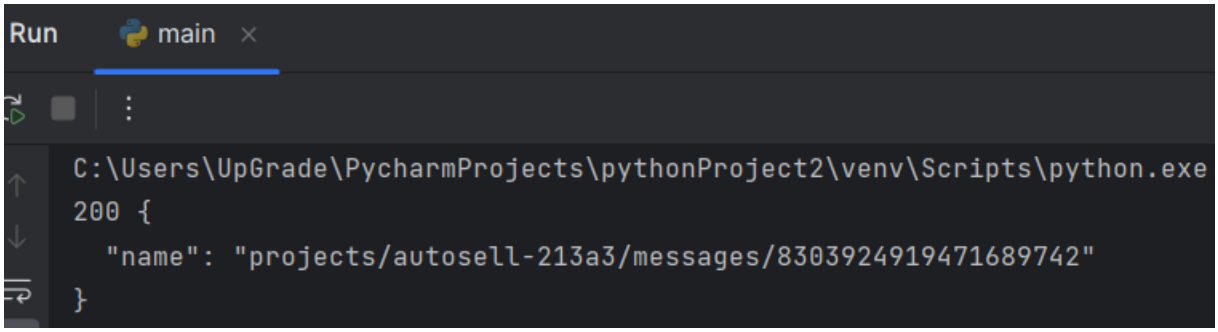

Рисунок 16 – Результат использования скрипта

3. Проверка получения уведомлений. На устройстве пользователя появилось уведомление с заголовком «Новое объявление» и текстом «Добавлен новый автомобиль. Проверьте его в приложении!».

Таким образом, функция уведомлений в приложении AutoSell была успешно реализована и протестирована.

4.9. Функция заказов

Функция заказов в приложении AutoSell предоставляет пользователям возможность оставлять заявки на покупку автомобилей, а администраторам - просматривать и обрабатывать эти заявки. Этот функционал является ключевым элементом приложения, обеспечивая удобный и эффективный процесс взаимодействия между покупателями и продавцами.

Для оформления заказа пользователь переходит на специальный экран «Оформить заказ». На этом экране представлена форма, включающая поля для ввода имени пользователя, адреса электронной почты и номера телефона. Эти поля обязательны для заполнения, так как предоставленная информация необходима для дальнейшей обработки заказа и связи с пользователем. Интерфейс создания заявки представлен на рисунке 17.

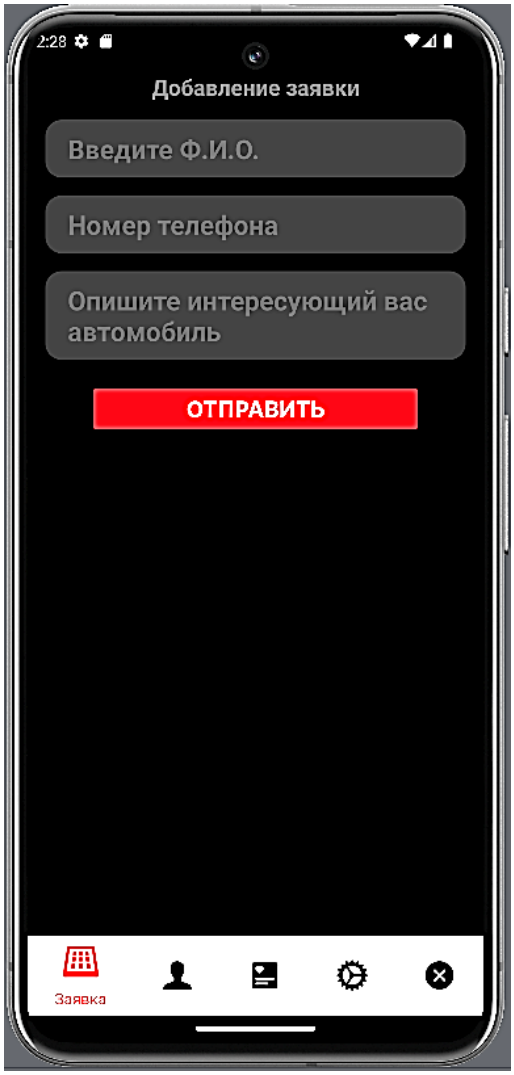

Рисунок 17 – Интерфейс создания заявки клиентом

После заполнения формы заказа и нажатия на кнопку «Отправить», данные о заказе отправляются в базу данных Firebase Cloud Database. Каждый заказ ассоциируется с уникальным идентификатором, который генерируется автоматически. Пример представлен на рисунке 18.

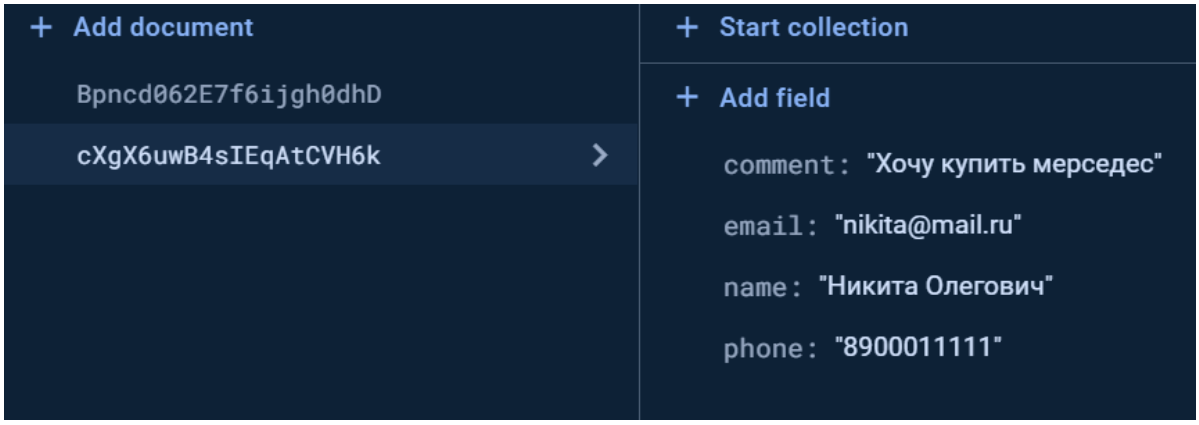

Рисунок 18 – Интерфейс добавления автомобиля администратором

Администраторы имеют доступ к разделу «Заказы», где они могут просматривать все оформленные заказы. Для этого используется специальный экран с RecyclerView, который загружает данные о заказах из базы данных Firebase.

Информация о каждом заказе в приложении AutoSell отображается в виде списка на специальном экране «Заказы», который продемонстрирован на рисунке 19. Этот экран предназначен для администраторов, которым необходимо управлять поступающими заявками на покупку автомобилей. Данный функционал упрощает обработку заказов и позволяет администраторам быстро получать доступ к необходимой информации.

Интерфейс этого экрана разработан таким образом, чтобы администраторы могли легко навигировать по списку заказов, видеть ключевые детали каждого заказа и быстро принимать решения. Такой подход к отображению информации способствует повышению эффективности работы и улучшению взаимодействия с пользователями.

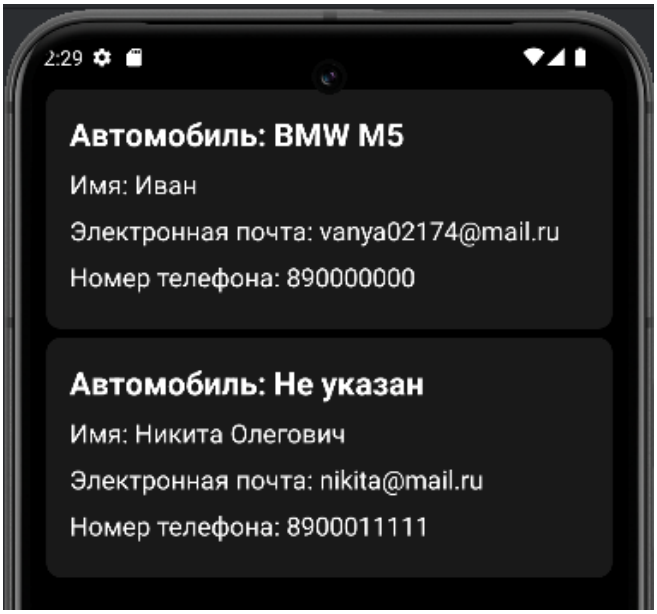

Рисунок 19 – Заказы, отображенные у администратора

4.10. Реализация слоя презентации

Слой презентации в приложении AutoSell реализован с использованием классов `OrderActivity`, `ItemsActivity`, `CarDetailsActivity`, `ItemsActivity_admin`, `Orders_admin`, `MainActivity`, `AuthActivity` и других. Эти классы отвечают за отображение данных пользователю и обработку его взаимодействий с интерфейсом.

Например, `OrderActivity` класс отвечает за представление экрана оформления заказа. В нем определены элементы пользовательского интерфейса, такие как поля ввода имени, электронной почты и телефона, а также кнопка «Отправить». При нажатии кнопки «Отправить» происходит проверка заполненности полей и отправка данных о заказе в базу данных Firebase. Класс `OrderActivity` представлен в листинге 7.

Листинг 7 – Класс `OrderActivity`

```
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_order)
mAuth = FirebaseAuth.getInstance()
database = FirebaseFirestore.getInstance()
nameEditText = findViewById(R.id.edit_text_name)
emailEditText = findViewById(R.id.edit_text_email)
phoneEditText = findViewById(R.id.edit_text_phone)
submitButton = findViewById(R.id.button_submit)
carTitle = intent.getStringExtra("carTitle")
```

```
carModel = intent.getStringExtra("carModel")
submitButton.setOnClickListener {
val name = nameEditText.text.toString()
val email = emailEditText.text.toString()
val phone = phoneEditText.text.toString()
if (name.isNotBlank() && email.isNotBlank() && phone.isNotBlank()) {
val user = mAuth.currentUser
user?.let { currentUser -> val userId = currentUser.uid
val userEmail = currentUser.email ?: ""
val orderRef = database.collection("orders").document()
val order = Order(userId, name, email, phone, userEmail, carTitle, car-
Model) orderRef.set(order).addOnSuccessListener {
Toast.makeText(this, "Заказ отправлен", Toast.LENGTH_SHORT).show()
val intent = Intent(this, ItemsActivity::class.java)
startActivity(intent) finish()} .addOnFailureListener {
Toast.makeText(this, "Ошибка отправки заказа", Toast.LENGTH_SHORT).show()}}
} else { Toast.makeText(this, "Пожалуйста, заполните все поля",
Toast.LENGTH_SHORT).show()}}}}
```

Класс `ItemsActivity`, который представлен в листинге 8, отображает список доступных автомобилей. В нем используется RecyclerView для эффективного отображения элементов списка. Метод `loadItems()` загружает данные об автомобилях из Firebase Firestore и обновляет список автомобилей.

Листинг 8 – Класс `ItemsActivity`

```
class ItemsActivity : AppCompatActivity() {
private lateinit var itemsList: RecyclerView
private lateinit var adapter: CarAdapter
private val db = FirebaseFirestore.getInstance()
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_items)
itemsList = findViewById(R.id.itemsList)
adapter = CarAdapter(emptyList(), this)
itemsList.adapter = adapter
itemsList.layoutManager = LinearLayoutManager(this)
val button: Button = findViewById(R.id.item_list_button_order)
button.setOnClickListener {
val intent = Intent(this, OrderActivity::class.java)
startActivity(intent)}
db.collection("cars").get().addOnSuccessListener { result ->
val items = mutableListOf<Car>()
for (document in result) {
val id = document.id
val title = document.getString("title") ?: ""
val model = document.getString("model") ?: ""
val desc = document.getString("desc") ?: ""
val price = document.getDouble("price") ?: 0.0
val image = document.getString("image") ?: ""
items.add(Car(id, title, model, desc, price, image))}
adapter.updateItems(items)}.addOnFailureListener {
exception ->}}}
```

Класс `CarDetailsActivity` предоставляет более подробную информацию об автомобиле, выбранном пользователем. Он отображает элементы интерфейса для показа изображения, названия, модели, описания и цены автомобиля. Также включает кнопку «Заказать автомобиль», которая перенаправляет пользователя на `OrderActivity` для оформления заказа.

Класс `ItemsActivity_admin` предназначен для администратора и отвечает за управление объявлениями. Он отображает список автомобилей и предоставляет функционал для добавления, редактирования и удаления объявлений. Методы `loadAdminItems()`, `addItem()`, `editItem()` и `deleteItem()` используются для управления данными об автомобилях в базе данных Firebase.

Класс `Orders_admin` отвечает за отображение списка заказов для администратора. Он содержит RecyclerView для отображения списка заказов и метод `loadOrders()` для получения данных о заказах из Firebase Firestore. Этот класс взаимодействует с `OrdersAdapter` для управления отображением данных.

Также была реализована система регистрации и авторизации, за которые отвечают классы `MainActivity` и `AuthActivity`. Класс `MainActivity` представлен в листинге 9, а класс AuthActivity представлен в приложении Е.

Листинг 9 – Класс `MainActivity`

```
class MainActivity : AppCompatActivity() {
override fun onCreate(savedInstanceState: Bundle?) {super.onCreate(savedIn-
stanceState)setContentView(R.layout.activity_main)
val userLogin: EditText = findViewById(R.id.user_login)
val userPass: EditText = findViewById(R.id.user_pass)
val button: Button = findViewById(R.id.button_reg)
val linkToAuth: TextView = findViewById(R.id.link_to_auth)
linkToAuth.setOnClickListener {
val intent = Intent(this, AuthActivity::class.java)
startActivity(intent)}button.setOnClickListener{
val login = userLogin.text.toString().trim()
val pass = userPass.text.toString().trim() if (login == "" || pass =="")
Toast.makeText(this, "Не все поля заполнены", Toast.LENGTH_LONG).show()
else {val user = User(login, pass)
val db = DbHelper(this, null)db.addUser(user)
Toast.makeText(this, "Пользователь $login добавлен",
Toast.LENGTH_LONG).show()userLogin.text.clear() userPass.text.clear()}}}
```

4.11. Реализация пользовательского интерфейса

Пользовательский интерфейс приложения AutoSell разработан с использованием XML-разметки файлов, что позволяет эффективно создавать интуитивно понятные и легко настраиваемые экраны. Этот подход способствует удобному взаимодействию пользователей с приложением, обеспечивая приятный и функциональный опыт использования. В интерфейсе используются разнообразные элементы, такие как текстовые поля, кнопки и списки, которые вместе формируют целостный и функциональный дизайн.

Настройка оформления пользовательского интерфейса осуществляется с помощью различных атрибутов, таких как цвета, размеры, отступы и другие параметры. Например, атрибут «`android:textColor="#fdfdfd"`» устанавливает цвет текста на белый, что способствует улучшению визуального восприятия и удобочитаемости.

Кроме того, взаимодействие с пользователем реализовано с помощью обработчиков событий, которые реагируют на действия пользователя, такие как нажатие кнопок или прокрутка списков. Эти обработчики позволяют обеспечить интерактивное и отзывчивое поведение приложения, мгновенно реагируя на действия пользователей и обеспечивая плавный и приятный пользовательский опыт. Нажатие кнопки «Отправить» может инициировать процесс отправки данных формы на сервер, а прокрутка списка автомобилей может динамически подгружать дополнительные элементы списка.

Пример XML-разметки «`activity_main.xml`», стартовой страницы регистрации, представлен в приложении В, в листинге 2. В этом листинге описывается структура интерфейса, включающая в себя различные элементы управления и их свойства.

Представление XML-файла «`activity_main.xml`» можно увидеть на рисунке 20, где наглядно показано, как эти элементы располагаются на экране и взаимодействуют друг с другом.

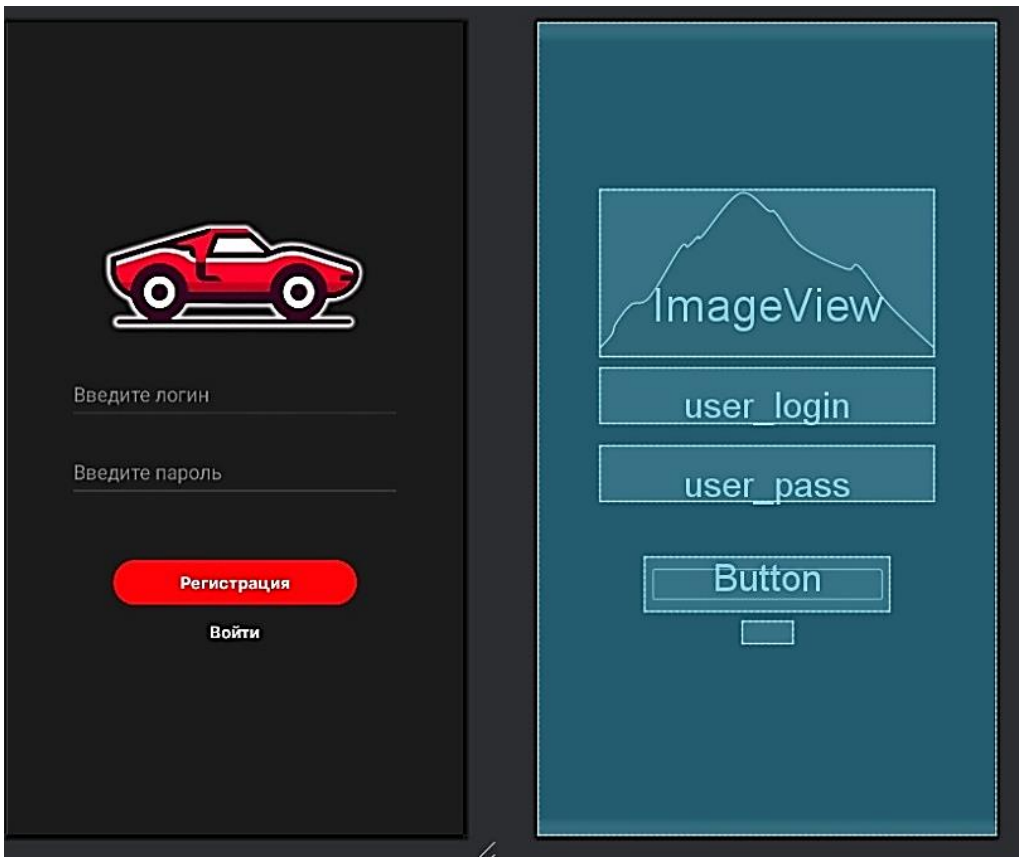

Рисунок 20 – Представление XML-файла «activity_main.xml»

Пример XML-файла под названием «activity_car_details.xml», с помощью которого реализован подробный просмотр информации об автомобиле, а также ее прокрутка, представлен в приложении В, на листинге 3. Этот XML-файл позволяет создавать интерфейс для детального отображения информации об автомобиле, включая все важные характеристики и описание.

На рисунке 21 показано представление XML-файла «activity_car_details.xml», где можно увидеть, как элементы интерфейса размещаются и взаимодействуют друг с другом для обеспечения удобного и информативного просмотра данных .

В этом файле используются различные элементы пользовательского интерфейса, такие как текстовые поля для отображения информации, изображения для визуального представления автомобиля и прокручиваемые области для удобного просмотра длинного списка характеристик и описаний.

Рисунок 21 – Представление XML-файла «`activity_car_details.xml`»

Вывод по четвертой главе

В результате разработки приложения AutoSell были использованы современные инструменты и технологии, включая Android Studio и архитектуру MVVM (Model-View-ViewModel). Для хранения информации о пользователях и товарах была использована база данных Firebase, обеспечивающая надежное и эффективное управление данными. Слой презентации и пользовательский интерфейс были реализованы с использованием соответствующих классов и XML-разметки файлов, что позволило создать интуитивно понятный и функциональный интерфейс. В результате была создана полноценная система, которая позволяет пользователям оформлять заказы, просматривать информацию о доступных автомобилях и взаимодействовать с администратором. Скриншоты приложения, демонстрирующие его функциональные возможности и интерфейс, представлены в приложении Б.

5. ТЕСТИРОВАНИЕ

В данной главе описан процесс тестирования приложения AutoSell на трех устройствах: двух виртуальных и одном физическом. Тестирование проводилось с целью выявления возможных проблем и обеспечения стабильной работы приложения на различных версиях операционной системы Android.

Тестирование проводилось на следующих устройствах.

1. Pixel 6 с API 30 (Android 11).
2. Pixel 8 с API 34 (Android 14).
3. Xiaomi 10C с Android 12.

Все три устройства показали положительные результаты. Приложение работало без сбоев, ошибок и задержек. Интерфейс был отзывчивым, все функции работали корректно. Тестирование охватывало взаимодействие с пользователем, работу с данными и сетевые операции. Результаты тестирования представлены в таблице 2, подтверждая готовность приложения AutoSell к использованию и выпуску.

Таблица 2 – Тестирование приложения

№	Описание теста	Шаги	Ожидаемый результат	Тест пройден?
1	Открытие приложения	Нажать на иконку приложения	Приложение успешно запускается	Да
2	Проверка системы валидации данных при регистрации аккаунта	Оставить окна регистрации пустыми	Приложение просит заполнить окна	Да
3	Создание аккаунта	Нажать на кнопку «зарегистрироваться»	Приложение уведомит об успешном создании аккаунта	Да
4	Намеренная ошибка входа	Ввести данные от несуществующего аккаунта	Приложение откажет в авторизации	Да
5	Вход в аккаунт	Ввести данный существующего аккаунта	Пользователь будет авторизован	Да
6	Исполнение функции просмотра объявлений	Просмотреть все представленные автомобили с помощью свайпов по экрану	Можно просматривать список автомобилей	Да

Окончание таблицы 2

№	Описание теста	Шаги	Ожидаемый результат	Тест пройден?
7	Просмотр подробной информации об автомобиле	Нажать на кнопку «Просмотреть» во время нахождения на доске объявлений	Приложение открывает подробную информацию о выбранном автомобиле	Да
8	Написание заявки администратору	Нажать на кнопку «Заказать автомобиль» во время нахождения на этапе просмотра доски объявлений	Приложение открывает окно, в котором необходимо заполнить поля об автомобиле и пользователе	Да
9	Проверка введенных данных	Отправить заявку с незаполненными текстовыми окнами	Приложение подсказывает, что необходимо ввести данные	Да
10	Работа кнопки «назад»	Нажать кнопку возврата	Пользователь возвращается в предыдущее состояние экрана	Да
11	Работа кнопки «назад», после написания заявки	Нажать кнопку возврата	Пользователь не может вернуться к отправленной заявке	Да
12	Сворачивание приложения	Находясь в приложении, нажать кнопку «Домой», а затем снова запустить приложение	Приложение сворачивается корректно, не перезапускается	Да
13	Загрузка данных из базы данных	Проверить, отображаются ли данные, загруженные из базы данных	Данные корректно загружаются и отображаются	Да
14	Загрузка изображений в базу данных	Проверить, загружаются ли изображения в базу данных и отображаются ли они корректно	Изображения успешно загружаются и корректно отображаются	Да
15	Уведомление о новом объявлении	Добавить новое объявление через панель администратора	Все пользователи получают уведомление о новом объявлении	Да
16	Авторизация создателя	Войти в аккаунт создателя	Пользователь авторизуется как создатель	Да
17	Добавление администратора создателем	Добавить новый аккаунт администратора через интерфейс создателя	Новый администратор успешно добавлен и может авторизоваться	Да

Вывод по пятой главе

После тщательного тестирования на виртуальных устройствах Pixel 6 API 30 и Pixel 8 API 34, а также физическом устройстве Xiaomi 10C с Android 12, были получены положительные результаты, и никаких проблем не было обнаружено. Тестирование охватывало основные функции приложения, включая регистрацию, авторизацию, просмотр списка автомобилей и оформление заявок. Все функции работали корректно и без сбоев.

Проверка валидности введенных данных и обработка ошибок при авторизации и регистрации прошли успешно. Приложение корректно обрабатывало неверные данные и предоставляло соответствующие уведомления пользователям.

Функция администратора, позволяющая добавлять и удалять объявления о продаже автомобилей, работала безупречно. Также успешно прошли тесты других функций, таких как авторизация создателя и добавление новых администраторов. Создатель мог добавлять новых администраторов через специальный интерфейс, а новые администраторы успешно авторизовались и выполняли свои обязанности.

Проверка загрузки данных из базы данных и отображения изображений показала, что приложение корректно загружает и отображает всю необходимую информацию. Изображения автомобилей загружаются и отображаются без задержек.

Дополнительно была проверена функция отправки уведомлений о новых объявлениях. Пользователи успешно получали уведомления на свои устройства, что подтверждает корректную работу системы уведомлений.

Таким образом, приложение AutoSell полностью соответствует заявленным функциональным требованиям. Оно демонстрирует стабильную работу на различных устройствах. Это обеспечивает положительный опыт для всех пользователей, делая процесс взаимодействия с приложением максимально комфортным и эффективным.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано мобильное приложение AutoSell для операционной системы Android, которое позволяет пользователям заказывать автомобили из-за границы. В процессе работы были выполнены следующие задачи.

1. Проведен анализ предметной области и аналогичных приложений.
2. Спроектирована архитектура приложения для заказа автомобилей из-за границы.
3. Реализовано приложение. Были разработаны элементы интерфейса, а также следующие функции, связанные с работой AutoSell:

- создание аккаунта пользователя;
- вход в аккаунт пользователя;
- просмотр списка доступных для приобретения автомобилей;
- просмотр подробной информации автомобиле;
- добавление новых объявлений администратором;
- создание заявки для заказа автомобиля;
- добавление администраторов управляющим приложения;
- просмотр заявок, отправленных клиентами.

4. Было проведено тестирование приложения, включая проверку корректности работы основных функций. Тестирование пройдено успешно.

Объем итогового приложения составил: 4100 строк.

В дальнейшем планируется расширить функционал приложения, будут добавлены следующие функции:

1) личный кабинет пользователя;
2) отслеживание поставки автомобиля;
3) настройки приложения.

ЛИТЕРАТУРА

1. Auto.ru. [Электронный ресурс] URL: https://auto.ru/ (дата обращения: 17.03.2024 г.).

2. Cars.com. [Электронный ресурс] URL: https://www.cars.com/ (дата обращения: 17.03.2024 г.).

3. Drom.ru. [Электронный ресурс] URL: https://www.drom.ru/ (дата обращения: 17.03.2024 г.).

4. Grand Auto. [Электронный ресурс] URL: https://grandeauto.ru/ (дата обращения: 17.03.2024 г.).

5. ItProger. [Электронный ресурс] URL: https://itproger.com/ (дата обращения: 20.04.2024 г.).

6. TrueCar. [Электронный ресурс] URL: https://www.truecar.com/ (дата обращения: 17.03.2024 г.).

7. Белл С. Android App Development For Dummies. // For Dummies, 2022. – 512 с.

8. Гайд Л. Mastering Android Development with Kotlin. // Packt Publishing, 2019. – 556 с.

9. Гриффитс Д. Head First. Kotlin. // O'Reilly Media, Inc., 2019. – 989 с.

10. Джексон А. Kotlin for Android Developers. // CreateSpace Independent Publishing Platform, 2018. – 344 с.

11. Ли С. Kotlin Programming: The Big Nerd Ranch Guide. // Big Nerd Ranch Guides, 2021. – 480 с.

12. Мерфи Дж. Android Programming for Beginners. // Independently published, 2020. – 700 с.

13. Смит Н. Android Studio 4.0 Development Essentials - Kotlin Edition. // Techotopia, 2020. – 720 с.

14. Соммерхоф П. Kotlin for Android App Development. // Packt Publishing, 2018. – 326 с.

15. Филлипс Б. Android Programming: The Big Nerd Ranch Guide. // Big Nerd Ranch Guides, 2017. – 624 с.

16. Хант Дж. Head First Android Development. // O'Reilly Media, Inc., 2020. – 930 с.

17. Хиллер Д. Kotlin in Action. // Manning Publications, 2017. – 384 с.

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–11.

Таблица 1 – Спецификация ВИ «Регистрация»

Прецедент: Регистрация.
ID: 1.
Краткое описание: Новый пользователь регистрирует свой аккаунт.
Главные актеры: Новый пользователь.
Второстепенные актеры: Нет.
Предусловия: Приложение запущено.
Основной поток: 1. Новый пользователь вводит логин, электронную почту и пароль. 2. Пользователь нажимает зарегистрироваться. 3. Система сохраняет данные пользователя. 4. Пользователя перенаправляют на страницу входа.
Постусловия*:* Создан новый аккаунт.
Альтернативные потоки: I. На шаге 1 новый пользователь не заполнил необходимые окна. 1. Показывается подсказка с допустимым форматом записи данных.

Таблица 2 – Спецификация ВИ «Авторизация»

Прецедент: Авторизация.
ID: 2.
Краткое описание: Пользователь входит в систему, используя логин и пароль.
Главные актеры: Пользователь.
Второстепенные актеры: Нет.
Предусловия*:* 1. Новый пользователь создал аккаунт. 2. Приложение запущено.
Основной поток: 1. Пользователь вводит логин и пароль. 2. Пользователь нажимает кнопку «Войти». 3. Система проверяет логин и пароль. 4. Если данные верны, пользователь перенаправляется на главную страницу.
Постусловия: Пользователь вошел в свой аккаунт.
Альтернативные потоки: I. На шаге 3 система определяет, что пользователь ввел неверные данные. 1. Показывается сообщение об ошибке. 2. Пользователю может попробовать ввести данные снова.

Таблица 3 – Спецификация ВИ «Просмотр списка автомобилей»

Прецедент: Просмотр списка автомобилей.
ID: 3.
Краткое описание: Пользователь просматривает список доступных автомобилей.
Главные актеры: Пользователь.
Второстепенные актеры: Нет.
Предусловия: Пользователь вошел в систему.
Основной поток: 1. Пользователь переходит на страницу списка автомобилей. 2. Система отображает список автомобилей.
Постусловия: Пользователь видит список доступных автомобилей.
Альтернативные потоки: Нет.

Таблица 4 – Спецификация ВИ «Просмотр деталей автомобиля»

Прецедент: Просмотр деталей автомобиля.
ID: 4.
Краткое описание: Пользователь просматривает информацию об автомобиле.
Главные актеры: Пользователь.
Второстепенные актеры: Нет.
Предусловия: Пользователь находится на странице списка автомобилей.
Основной поток: 1. Пользователь выбирает автомобиль из списка. 2. Система отображает детальную информацию об автомобиле.
Постусловия: Пользователь видит детальную информацию об автомобиле.
Альтернативные потоки: Нет.

Таблица 5 – Спецификация ВИ «Заказ автомобиля»

Прецедент: Заказ автомобиля.
ID: 5.
Краткое описание: Пользователь заказывает выбранный автомобиль.
Главные актеры: Пользователь.
Второстепенные актеры: Нет.
Предусловия: Пользователь просматривает детальную информацию об автомобиле.
Основной поток: 1. Пользователь нажимает кнопку «Заказать автомобиль». 2. Пользователь заполняет форму с данными. 3. Пользователь нажимает «Отправить». 4. Система сохраняет заказ в базе данных.
Постусловия: Заказ автомобиля успешно создан.
Альтернативные потоки: I. На шаге 2 пользователь не заполнил данные. 1. Показывается сообщение об ошибке с подсказками по заполнению.

Таблица 6 – Спецификация ВИ «Просмотр заявок»

Прецедент: Просмотр заявок.
ID: 6.
Краткое описание: Администратор просматривает список заявок.
Главные актеры: Администратор
Второстепенные актеры: Нет.
Предусловия: Администратор вошел в систему.
Основной поток: 1. Администратор переходит на страницу заявок. 2. Система отображает список всех заявок.
Постусловия: Администратор видит список всех заявок.
Альтернативные потоки: Нет.

Таблица 7 – Спецификация ВИ «Создание заявки на приобретение авто»

Прецедент: Создание заявки на приобретение автомобиля.
ID: 7.
Краткое описание: Пользователь создает заявку на приобретение отсутствующего автомобиля.
Главные актеры: Пользователь.
Второстепенные актеры: Администратор.
Предусловия: Пользователь авторизован в системе.
Основной поток: 1. Пользователь переходит на страницу создания заявки. 2. Пользователь заполняет форму заявки с указанием желаемой модели и марки авто. 3. Пользователь нажимает кнопку «Отправить заявку». 4. Система сохраняет заявку и отправляет ее администратору.
Постусловия: Заявка успешно создана и отправлена администратору.
Альтернативные потоки: I. На шаге 2 пользователь не заполнил все необходимые поля. 1. Показывается сообщение об ошибке с подсказками по заполнению.

Таблица 8 – Спецификация ВИ «Назначение администраторов»

Прецедент: Назначение администраторов.
ID: 8.
Краткое описание: Создатель назначает клиентов администраторами.
Главные актеры: Создатель.
Второстепенные актеры: Клиенты.
Предусловия: Создатель авторизован в системе.
Основной поток: 1. Создатель переходит на страницу управления пользователями. 2. Создатель выбирает клиента для назначения администратором. 3. Создатель нажимает кнопку «Назначить администратором». 4. Система обновляет уровень доступа клиента до администратора.
Постусловия*:* Клиент успешно назначен администратором.
Альтернативные потоки: Нет.

Таблица 9 – Спецификация ВИ «Отправка заявки администратору»

Прецедент: Отправка заявки администратору.
ID: 9.
Краткое описание: Заявка на приобретение авто отправляется адм. для обработки.
Главные актеры: Пользователь.
Второстепенные актеры: Администратор.
Предусловия: Пользователь создал или выбрал заявку.
Основной поток: 1. Пользователь завершает создание заявки и нажимает кнопку «Отправить». 2. Система отправляет заявку администратору. 3. Администратор получает уведомление о новой заявке.
Постусловия: Заявка успешно отправлена администратору для обработки.
Альтернативные потоки: Нет.

Таблица 10 – Спецификация ВИ «Просмотр объявлений в режиме управления»

Прецедент: Просмотр объявлений в режиме управления.
ID: 10.
Краткое описание: Адм. или созд. просматривает и удаляет объявления в режиме упр.
Главные актеры: Администратор, создатель.
Второстепенные актеры: Нет.
Предусловия: Администратор или создатель авторизованы в системе.
Основной поток: 1. Администратор или создатель переходит на страницу управления объявлениями. 2. Система отображает список всех объявлений с возможностью их удаления.
Постусловия: Объявления успешно управляются администратором или создателем.
Альтернативные потоки: Нет.

Таблица 11 – Спецификация ВИ «Работа с объявлениями»

Прецедент: Работа с объявлениями.
ID: 11.
Краткое описание: Адм. или созд. добавляет новые объявления на доску объявлений.
Главные актеры: Администратор, создатель.
Второстепенные актеры: Нет.
Предусловия: Администратор или создатель авторизованы в системе.
Основной поток: 1. Администратор или создатель переходит на страницу добавления объявления. 2. Администратор или создатель заполняет форму нового объявления с данными авто. 3. Администратор или создатель нажимает кнопку «Добавить». 4. Система сохраняет новое объявление и добавляет его на доску объявлений.
Постусловия: Новое объявление успешно добавлено на доску объявлений.
Альтернативные потоки: I. На шаге 2 администратор или создатель не заполнил все обязательные поля. 1. Показывается сообщение об ошибке с указанием обязательных полей.

Приложение Б. Скриншоты созданного Android-приложения

На рисунках представлены: регистрация и авторизация в приложении на рисунке 1, просмотр объявлений и выбранного объявления на рисунке 2, создание объявления администратором и заявки клиентом на рисунке 3.

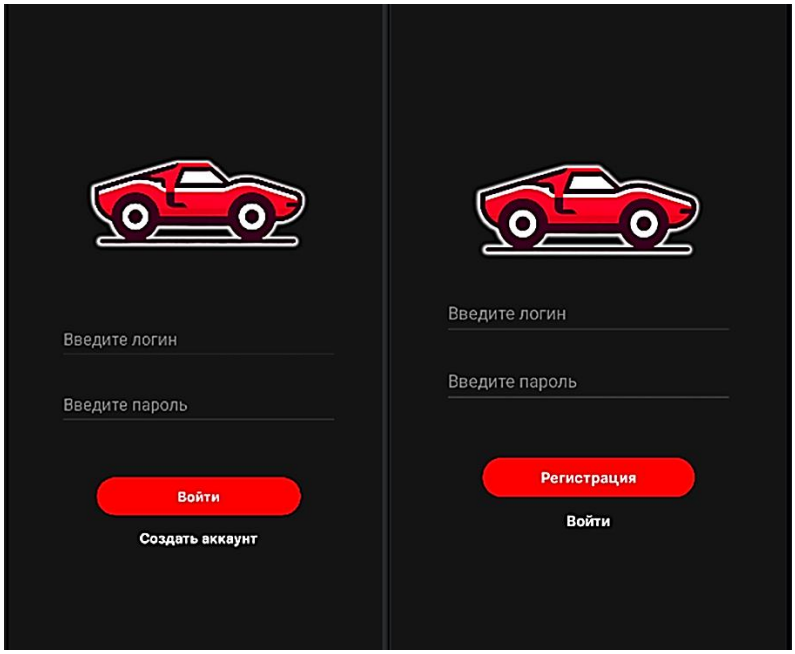

Рисунок 1 – Авторизация (слева), регистрация (справа)

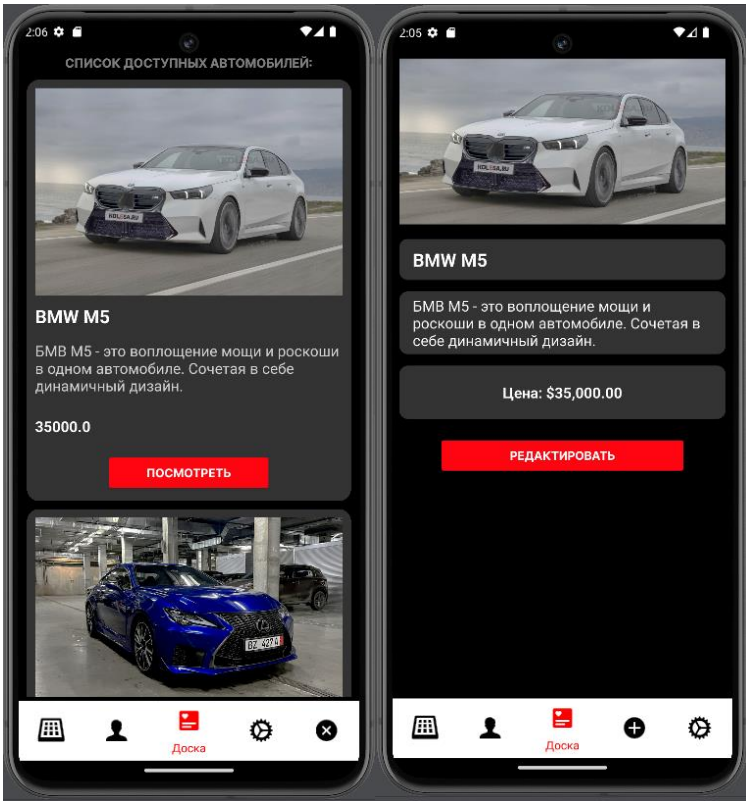

Рисунок 2 – Просмотр объявлений (слева), просмотр объявления (справа)

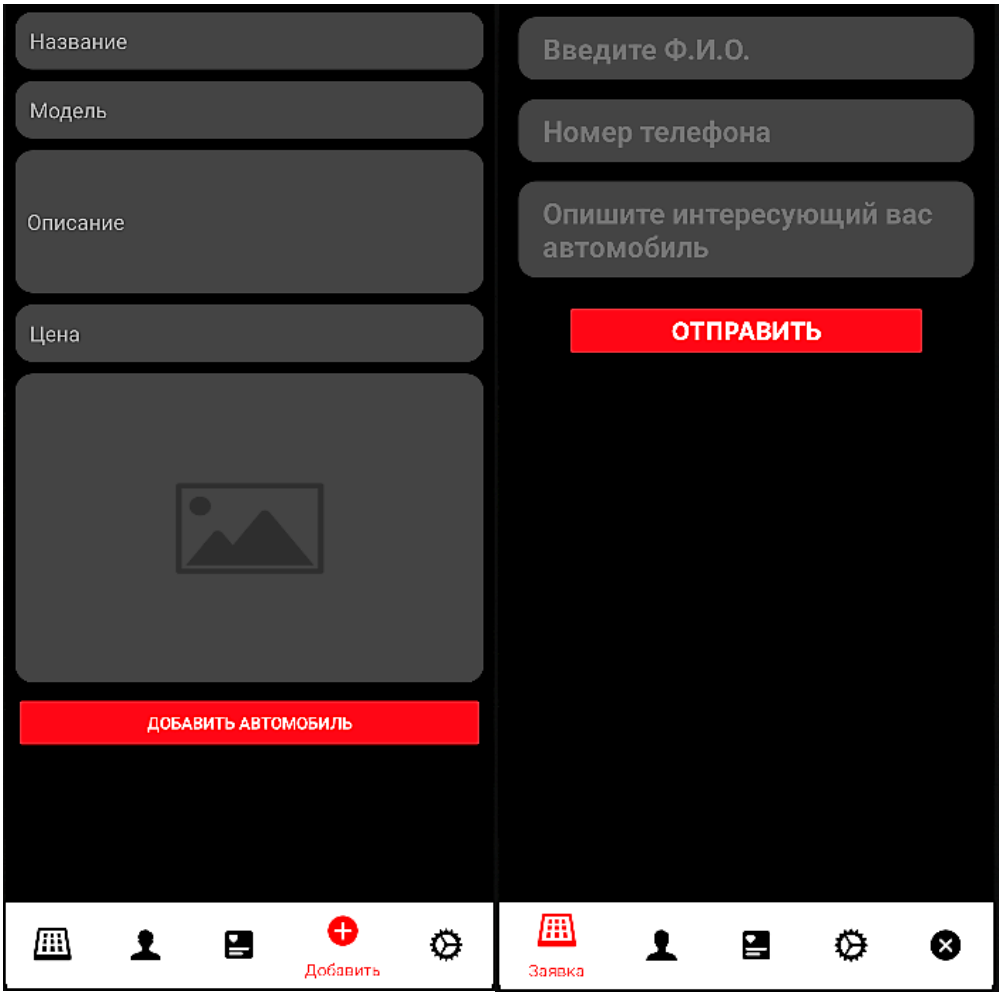

Рисунок 3 – Создание объявления (слева), создание заявки (справа)

Приложение В. Листинги Android-приложения

В листинге 1 представлен класс `CarDetailsActivity`, отвечающий за детали автомобиля в приложении. Он содержит методы для отображения информации о конкретном автомобиле, включая изображение, название, описание, цену и кнопку покупки. В листингах 2 и 3 представлены XML-файлы, определяющие макеты пользовательского интерфейса для основного экрана приложения и экрана с деталями автомобиля соответственно.

Листинг 1 – Класс `CarDetailsActivity`

```
class CarDetailsActivity : AppCompatActivity() {
private val db = FirebaseFirestore.getInstance()
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_car_details)
val carId = intent.getStringExtra("car_id")
val image: ImageView = findViewById(R.id.item_list_image)
val title: TextView = findViewById(R.id.item_list_title)
val model: TextView = findViewById(R.id.item_list_model)
val desc: TextView = findViewById(R.id.item_list_desc)
val price: TextView = findViewById(R.id.item_list_price)
val orderButton: Button = findViewById(R.id.item_list_button_order)
carId?.let { id -> db.collection("cars").document(id).get().addOnSuccess-
Listener { document -> if (document != null && document.exists()) {
val item = document.toObject(Car::class.java)item?.let {
title.text = it.title
model.text = it.model
desc.text = it.desc
val formattedPrice = NumberFormat.getCurrencyInstance(Locale.US).for-
mat(it.price)
price.text = "Цена: $formattedPrice"
Picasso.get().load(it.image).into(image)
val carTitle = it.title
val carModel = it.model
orderButton.setOnClickListener {
val intent = Intent(this, OrderActivity::class.java).apply {
putExtra("carTitle", carTitle)
putExtra("carModel", carModel)}
startActivity(intent)}}}}.addOnFailureListener { exception ->}}}}
```

Листинг 2 – XML-файл «`activity_main.xml`»

```
<ImageView
        android:id="@+id/logo8"
        android:layout_gravity="center"
        android:layout_marginTop="150dp"
        android:layout_width="300dp"
        android:layout_height="150dp"
        android:src="@drawable/logo8" />
    <EditText
        android:id="@+id/user_login"
        android:layout_width="300dp"
        android:layout_height="50dp"
        android:ems="10"
        android:inputType="text"
```

Окончание листинга 2 приложения В

```
        android:textColorHint="#6a6a6a"
        android:layout_marginTop="10dp"
        android:layout_gravity="center"
        android:textColor="#fdfdfd"
        android:hint="Введите логин" />
    <EditText
        android:id="@+id/user_pass"
        android:layout_width="300dp"
        android:layout_height="50dp"
        android:ems="10"
        android:inputType="textPassword"
        android:textColorHint="#6a6a6a"
        android:layout_marginTop="20dp"
        android:layout_gravity="center"
        android:textColor="#fdfdfd"
        android:hint="Введите пароль" />
    <Button
        android:id="@+id/button_reg"
        android:layout_width="220dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="50dp"
        android:backgroundTint="#ff0207"
        android:fontFamily="@font/inter_bold"
        android:text="Регистрация"
        android:textColor="#fdfdfd"
        android:textSize="15sp"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/link_to_auth"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:fontFamily="@font/inter_bold"
        android:text="Войти"
        android:textColor="#fdfdfd"
        android:textSize="15sp" />
</LinearLayout>
```

Листинг 3 – XML-файл XML-файл «activity_car_details.xml»

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000000">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fillViewport="true"
        android:paddingBottom="16dp"
        android:layout_above="@id/bottom_navigation">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
android:orientation="vertical"
android:padding="4dp">

<ImageView
    android:id="@+id/item_list_image"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:scaleType="centerCrop"
    android:layout_marginTop="16dp"
    android:layout_marginBottom="8dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp" />

<TextView
    android:id="@+id/item_list_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:textAppearanceLarge"
    android:text="TextView"
    android:paddingVertical="10dp"
    android:paddingHorizontal="16dp"
    android:background="@drawable/rounded_background"
    android:textColor="@android:color/white"
    android:textStyle="bold"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="4dp"
    android:layout_marginStart="4dp"
    android:layout_marginEnd="4dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp" />


<TextView
    android:id="@+id/item_list_desc"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:textAppearanceMedium"
    android:textColor="@android:color/white"
    android:paddingVertical="5dp"
    android:paddingHorizontal="16dp"
    android:background="@drawable/rounded_background"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="4dp"
    android:layout_marginStart="4dp"
    android:layout_marginEnd="4dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp" />

<TextView
    android:id="@+id/item_list_price"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:textAppearanceMedium"
    android:textColor="@android:color/white"
    android:paddingVertical="20dp"
    android:textStyle="bold"
    android:paddingHorizontal="16dp"
    android:background="@drawable/rounded_background"
    android:layout_marginTop="10dp"
    android:gravity="center"
```

```
                android:layout_marginBottom="4dp"
                android:layout_marginStart="4dp"
                android:layout_marginEnd="4dp"
                android:layout_marginLeft="4dp"
                android:layout_marginRight="4dp" />

            <Button
                android:id="@+id/item_list_button_order"
                android:layout_width="300dp"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:backgroundTint="#ff0207"
                android:fontFamily="@font/inter_bold"
                android:textColor="#fff"
                android:layout_marginTop="16dp"
                android:text="Заказать автомобиль" />

        </LinearLayout>
    </ScrollView>

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/bottom_navigation"
        style="@style/BottomNavigationViewStyle"
        android:layout_width="match_parent"
        android:layout_height="@dimen/bottom_navigation_height"
        android:layout_alignParentBottom="true"
        android:background="#FFFFFF"
        app:itemIconSize="@dimen/bottom_navigation_icon_size"
        app:itemIconTint="@color/nav_item_color"
        app:itemTextColor="@color/nav_item_color"
        app:menu="@menu/bottom_navigation_menu" />
</RelativeLayout>
```

Приложение Г. Примеры аналогичных проектов

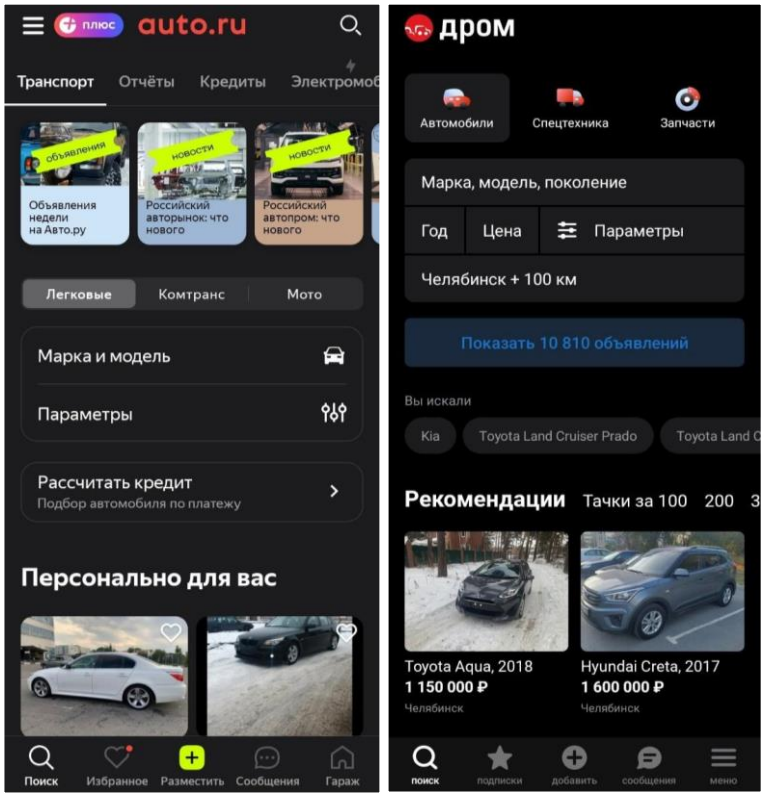

Рисунок 4 – Drom.ru (слева) и Auto.ru (справа)

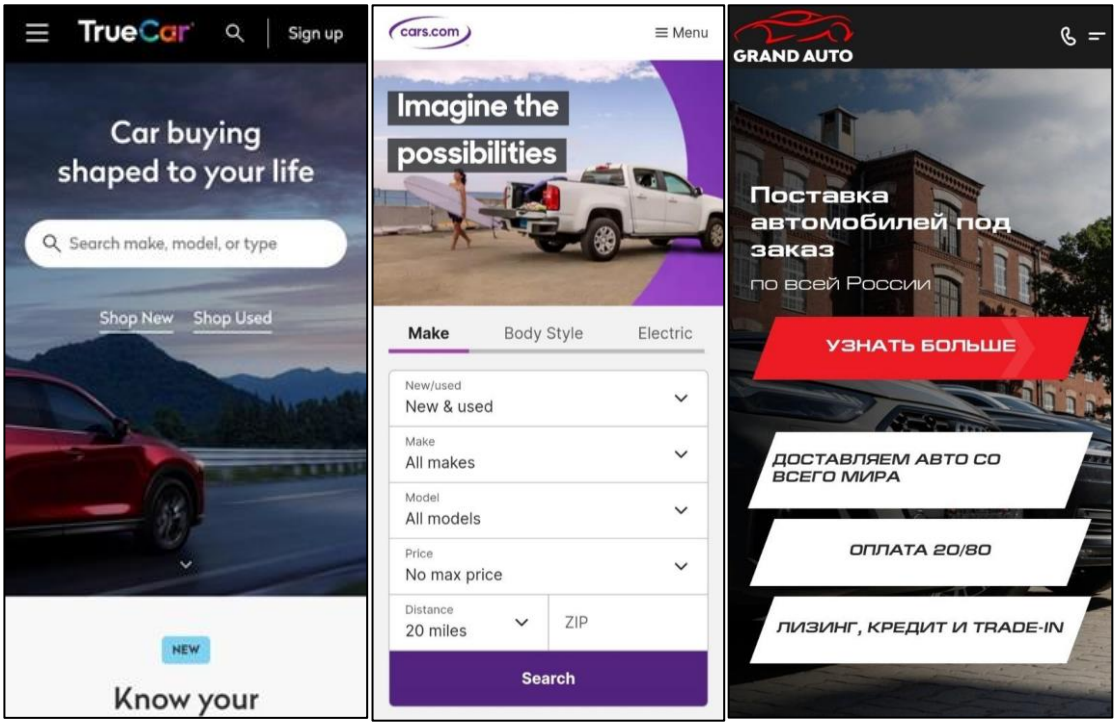

Рисунок 5 – TrueCar (слева), Cars.com (Посередине) и Grand Auto (справа)

Приложение Д. Диаграмма последовательности

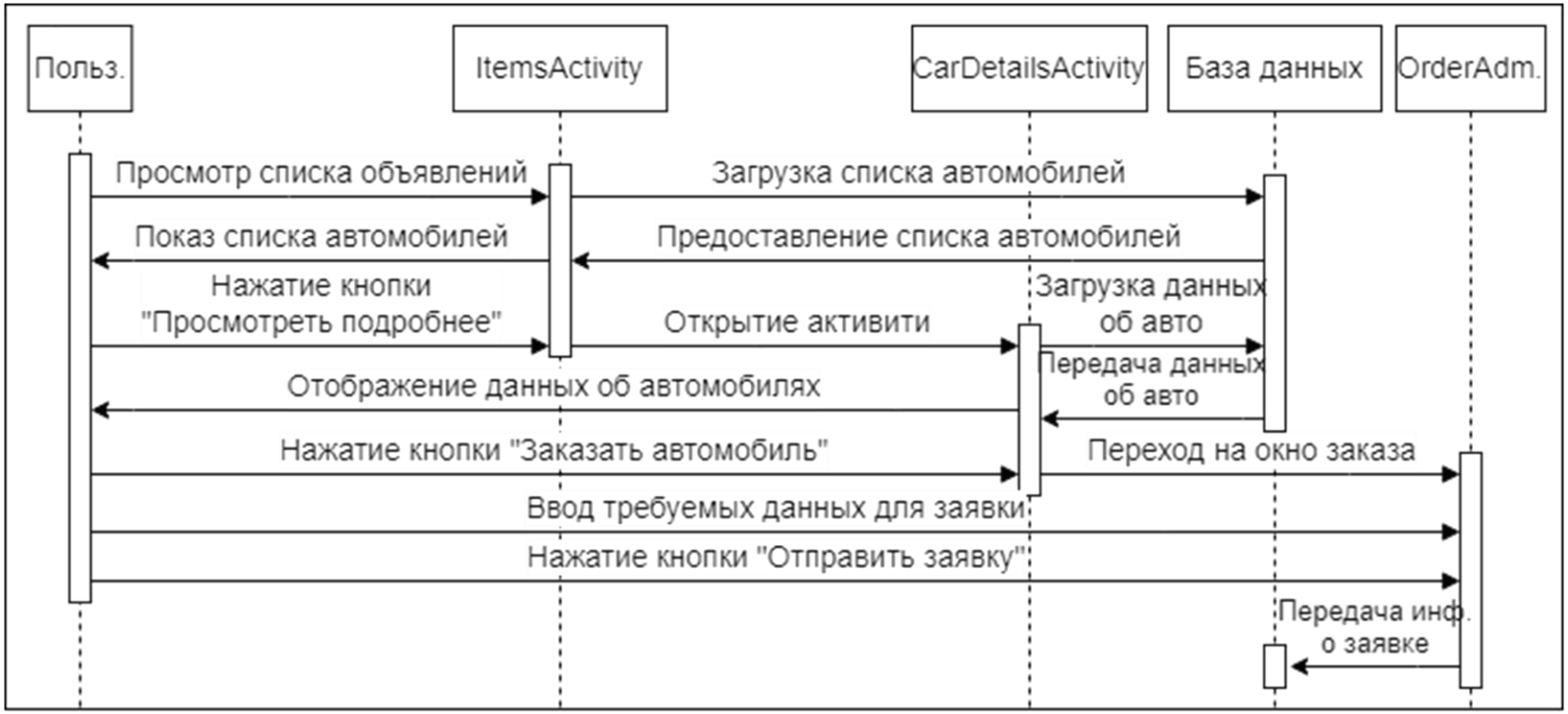

Рисунок 6 – Диаграмма последовательности

Приложение Е. Листинг AuthActivity

Листинг 4 – Класс `AuthActivity`

```
class AuthActivity : AppCompatActivity() {
private lateinit var database: DatabaseReference
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_auth)
val userLogin: EditText = findViewById(R.id.user_login_auth)
val userPass: EditText = findViewById(R.id.user_pass_auth)
val button: Button = findViewById(R.id.button_auth)
val linkToReg: TextView = findViewById(R.id.link_to_reg)
database = FirebaseDatabase.getInstance().reference
linkToReg.setOnClickListener {
val intent = Intent(this, MainActivity::class.java)
startActivity(intent)}button.setOnClickListener {
val login = userLogin.text.toString().trim()
val pass = userPass.text.toString().trim()
if (login.isEmpty() || pass.isEmpty())
Toast.makeText(this, "Не все поля заполнены",
Toast.LENGTH_LONG).show()else{ authenticateUser(login, pass)}}}
private fun authenticateUser(email: String, password: String) {
val db = DbHelper(this, null)
val isAuth = db.getUser(email, password)
Log.d("AuthActivity", "User authenticated: $isAuth")
if (isAuth) {checkIfCreator(email)} else {
Toast.makeText(this, "Пользователь $email не авторизо-
ван",Toast.LENGTH_LONG).show()}}
private fun checkIfCreator(email: String) {
val creatorRef = database.child("Creator")
creatorRef.addListenerForSingleValueEvent(object : ValueEventListener {
override fun onDataChange(dataSnapshot: DataSnapshot) { var isCreator =
false for (snapshot in dataSnapshot.children) {
val creatorEmail = snapshot.child("Email").getValue(String::class.java)
Log.d("AuthActivity", "Проверка email создателя: $creatorEmail с $email")
if (creatorEmail == email) {isCreator = true break}} if (isCreator) {
Toast.makeText(this@AuthActivity, "Пользователь $email авторизован как со-
здатель", Toast.LENGTH_LONG).show()
val intent = Intent(this@AuthActivity, ItemsActivityGod::class.java)
startActivity(intent)} else {checkIfAdmin(email)}}
override fun onCancelled(databaseError: DatabaseError) {
Toast.makeText(this@AuthActivity, "Ошибка базы данных: ${databaseError.mes-
sage}", Toast.LENGTH_LONG).show()}})}
private fun checkIfAdmin(email: String) {
val adminRef = database.child("admins").child("adminsDB")
adminRef.addListenerForSingleValueEvent(object : ValueEventListener {
override fun onDataChange(dataSnapshot: DataSnapshot) {var isAdmin = false
for (snapshot in dataSnapshot.children) {
val adminEmail = snapshot.child("Email").getValue(String::class.java)
Log.d("AuthActivity", "Проверка email администратора: $adminEmail с
$email") if (adminEmail == email) { isAdmin = true break}}
if (isAdmin) {Toast.makeText(this@AuthActivity, "Пользователь $email авто-
ризован как администратор", Toast.LENGTH_LONG).show()
val intent = Intent(this@AuthActivity, ItemsActivity_admin::class.java)
startActivity(intent)} else {
Toast.makeText(this@AuthActivity, "Пользователь $email авторизован",
Toast.LENGTH_LONG).show() val intent = Intent(this@AuthActivity, ItemsAc-
tivity::class.java) startActivity(intent)}} override fun onCancelled(data-
baseError: DatabaseError) {
Toast.makeText(this@AuthActivity, "Ошибка базы данных: ${databaseError.mes-
sage}", Toast.LENGTH_LONG).show()}})}}
```