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 решения задач логистики транспортной компании «Луч»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-347.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ Е.В. Иванова

Автор работы,
студент группы КЭ-403
_____ М.И. Озеров

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-403
Озерову Максиму Игоревичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12) Разработка Android-приложения для решения задач логистики транспортной компании «Луч».
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Kotlin Documentation. [Электронный ресурс] URL: https://kotlinlang.org/docs/home.html (дата обращения: 11.02.2024 г.).
3.2. Jetpack Compose UI App Development Toolkit – Android Developers. [Электронный ресурс] URL: https://developer.android.com/jetpack/compose (дата обращения: 11.02.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области и аналогов.
4.2. Спроектировать приложение и графический интерфейс.
4.3. Реализовать необходимые методы на серверной части.
4.4. Реализовать приложение.
4.5. Провести тестирование приложения.

5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. Е.В. Иванова

Задание принял к исполнению М.И. Озеров

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Транспортные компании играют важную роль в жизни людей, так как часто приходится отправлять те или иные вещи в другие города или даже страны. Особенно это касается частных предпринимателей, у которых нет собственной логистики по отправке товаров покупателям, из-за чего им приходится пользоваться услугами транспортных компаний.

Важным вопросом в работе транспортных компаний является решение задач логистики. В настоящее время решение этих задач упрощается из-за стремительного развития информационных технологий. Если раньше такие вещи, как маршруты перевозок и их расписания, данные о грузах и результаты ревизий складов хранились в бумажном виде и записывались вручную, то сейчас эти данные хранятся в базах данных, а их запись автоматизируется специальным программным обеспечением. Таким образом использование информационных технологий для решения задач логистики помогает в решении следующих задач.

1. Уменьшение затрат времени работников компании на различных этапах логистики за счет автоматизации и упрощения управления данными, тем самым увеличив количество обслуженных клиентов и оперативно выполненных перевозок.
2. Возможность более точно отслеживать статусы заявок и состояние грузов на каждом этапе логистики, а также хранить историю перемещения грузов и информацию о сотрудниках, которые с ними работали на том или ином этапе.
3. Уменьшение вероятности человеческих ошибок и количества потерянных данных.

Транспортная компания «Луч» [1] в данный момент активно развивается и открывает филиалы в различных городах как России, так и ближнего зарубежья, поэтому использование информационных технологий для решения задач логистики является актуальной задачей.

Постановка задачи

Целью выпускной квалификационной работы является разработка Android-приложения для решения задач логистики транспортной компании «Луч». Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области и аналогов;
2) спроектировать приложение и графический интерфейс;
3) реализовать необходимые методы на серверной части;
4) реализовать приложение;
5) провести тестирования приложения.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 55 страниц, список литературы содержит 27 источников.

В первой главе проводится анализ предметной области и существующих решений.

Во второй главе проводится анализ функциональных и нефункциональных требований к системе, определяется спецификация основных прецедентов и выстраивается диаграмма вариантов использования. Также в ней описывается архитектура системы.

В третьей главе описана реализация системы.

В четвертой главе приведены результаты тестирования.

В приложении А содержится схема базы данных.

В приложении Б содержится диаграмма компонентов архитектуры системы.

В приложении В содержатся листинги с исходным кодом приложения.

В приложении Г содержатся скриншоты интерфейса Android-приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

Предметной областью данной работы является логистика в транспортной компании. Логистика в компании «Луч» состоит из следующих этапов.

1. Клиент создает заявку на грузоперевозку через официальный сайт ТК «Луч» или на кассе любого склада. В заявке он указывает в какой город нужно доставить груз, а также какие дополнительные услуги и упаковки он хочет заказать.

2. Если клиент заказал услугу адресного забора груза, то к нему направляется водитель, который при заборе груза указывает количество полученных за услугу денег и время ожидания, если водитель приехал в назначенное время, но ему пришлось ждать клиента. После этого водитель везет забранный груз на склад. Если клиент не заказывал услугу адресного забора груза, то он сам везет груз на склад отправления.

3. Прием груза на складе представляет собой измерение количества, размеров и массы грузов, выбора упаковки для них и печать необходимых документов и QR-меток для маркировки груза. Пример такой QR-метки представлен на рисунке 1.

4. Для отправки груза в город получения, груз помещают в машину. Перед этим работник склада с помощью собственного смартфона или выданного ему на складе терминала сбора данных (спец. устройство со встроенным сканером штрихкодов и QR-кодов) сканирует QR-метки грузов, который он собирается положить в машину.

5. При выгрузке в городе получения работник склада с помощью телефона или выданного ему на складе терминала сбора данных сканирует QR-метки грузов, которые он достает из машины. Грузы после этого помещаются на склад.

6. Если клиент заказывал адресную доставку груза, то производится погрузка грузов в машину для доставки, также сканируя QR-метку каждого

груза перед погрузкой, после чего водитель доставляет грузы на нужный адрес и при выдаче указывает количество полученных за услугу денег и время ожидания, если он приехал в назначенное время, но ему пришлось ждать клиента. Если клиент не заказывал услугу адресной доставки груза, то он сам едет на пункт выдачи склада, чтобы забрать свой груз.

7. Между этапами приема груза, погрузки, выгрузки, погрузки на доставку и выдаче периодически на складах производят ревизии – сотрудник склада с помощью телефона или терминала сбора данных сканирует QR-метки всех находящихся на складе грузов. Это нужно чтобы указать в какой ячейке на складе лежит груз, а также обнаружить грузы, которые не должны были оказаться на складе ревизии.

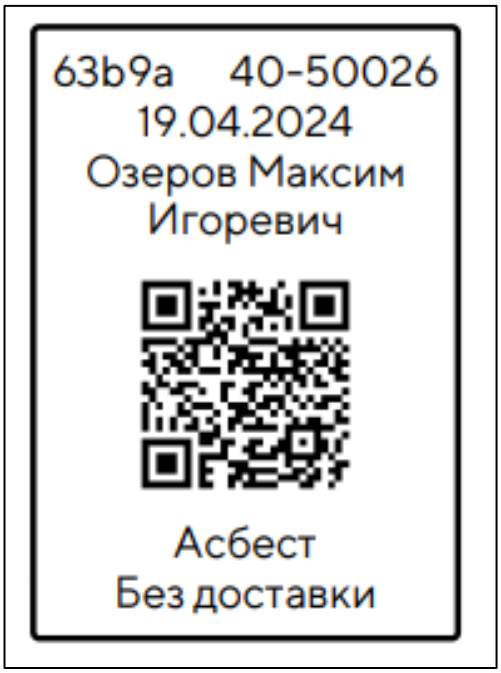

Рисунок 1 – Пример метки груза с QR-кодом

1.2. Обзор аналогов

Рассмотрим существующие решения для систем управления логистикой.

Компания «Wellsoft» [2] занимается разработкой различных приложений для транспортно-логистических компаний на заказ. Для наших целей у них есть несколько подходящих продуктов – приложение для организации

складской логистики, которое дает возможность находить груз на складе, взаимодействовать работникам склада и перевозчику, а также формировать необходимые документы, и Android-приложение для водителей.

Компания «АЙТОБ» [3] предлагает свое программное обеспечение для цифровизации логистики. Для наших целей у них есть приложение «ITOB: Мобильный клиент» – приложение для водителей, обменивающееся информацией с 1С конфигурациями. Таким образом приложение позволяет отслеживать местоположение водителей, контролировать их заказы и получать отчеты о выполнении заданий, в том числе с заполненными формами документов и фотоотчетами.

Выводы по первой главе

Так как каждая транспортная компания строит свои логистические операции по-своему, нет универсального ПО для этих целей. Существуют такие компании, как «Wellsoft» и «АЙТОБ», предлагающие собственные решения или разрабатывают системы для управления логистикой транспортных компаний на заказ с нуля, но у компании «Луч» уже есть налаженная система ПО для отслеживания грузоперевозок, на котором работают сотрудники, работающими за ПК. Поэтому стоит задача разработки Android-приложения, которое должно будет охватить все этапы существующей логистики компании от забора груза до его доставки и не конфликтовать с существующим ПО.

2. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

2.1. Требования к проектируемой системе

Функциональные требования – это требования, определяющие функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи. Для дальнейшего проектирования были составлены следующие функциональные требования:

1) приложение должно реализовать функциональные модули «Адресный забор груза», «Прием груза на складе», «Погрузка груза», «Выгрузка груза», «Адресная доставка груза» и «Ревизия»;

2) приложение должно поддерживать три типа пользователей с разными правами доступа к функциональным модулям: водитель, грузчик и работник склада;

3) пользователь должен иметь возможность авторизоваться и выйти из приложения;

4) пользователь должен иметь возможность обновить приложение, когда версии используемых им модулей станут не актуальными.

Нефункциональные требования – это требования к свойствам и характеристикам, которые система должна демонстрировать, или ограничениям, которые она должна соблюдать. Для дальнейшего проектирования были составлены следующие требования:

1) приложение должно быть реализовано для операционной системы Android, и запускаться на Android-смартфонах и терминалах сбора данных;

2) приложение должно корректно работать на платформе Android версии 10.0 и выше, так как эти версии поддерживаются на терминалах сбора данных;

3) авторизация в приложении должна быть реализована через JWT токены;

4) доступ пользователя к модулю приложения возможен только если он имеет последнюю версию;

5) приложение должно иметь возможность обрабатывать QR-коды как через камеру, так и через сканер терминалов сбора данных.

2.2. Диаграмма вариантов использования

Диаграмма вариантов использования позволяет получить визуальное представление о требованиях пользователей [4]. Были составлены диаграммы вариантов использования для логистических модулей. С системой приложения будут взаимодействовать 3 актера.

1. Водитель – пользователь с ролью «Водитель» и «Подотчетное лицо», который имеет доступ к модулям забора груза и доставки груза.
2. Работник склада – пользователь с ролью «Склад», который имеет доступ к модулям приема груза и ревизии.
3. Грузчик – пользователь с ролью «Грузчик», который имеет доступ к модулям погрузки, разгрузки и доставки груза.

На рисунке 2 представлена диаграмма вариантов использования для модуля «Адресный забор груза». На ней видны следующие варианты использования для актера «Водитель».

1. Посмотреть список заявок – водитель может посмотреть список заявок, назначенных на него для адресного забора груза.
2. Составить очередь из заявок – водитель может указать в какой последовательности он собирается приезжать на заявки, чтобы логисты видели это в системе и могли сказать клиентам о примерном времени прибытия.
3. Выбрать заявку – водитель может выбрать одну из заявок в списке, чтобы посмотреть информацию о ней.
4. Внести данные и завершить забор – водитель должен ввести время ожидания клиента, комментарии о заявке и завершить забор груза, тем самым исключая его из списка на адресный забор.

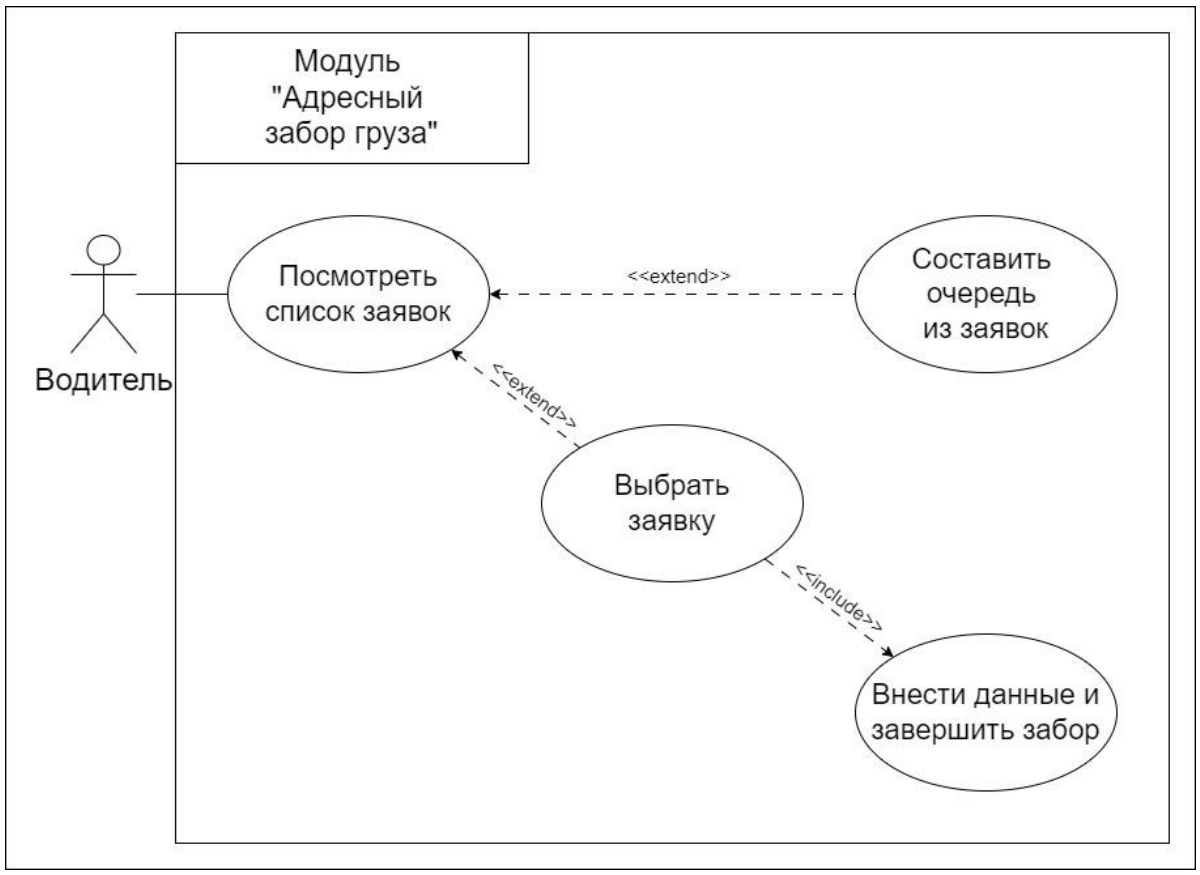

Рисунок 2 – Диаграмма для модуля «Адресный забор груза»

На рисунке 3 представлена диаграмма вариантов использования для модуля «Прием груза». На ней видны следующие варианты использования для актера «Работник склада».

1. Найти заявку по ее номеру – работник может ввести последние 5 цифр из накладной по заявке, чтобы начать принимать ее.

2. Сканировать метки грузов – если у грузов заявки уже присутствуют метки, то работник может сразу начать сканировать их и вводить данные об отсканированном грузе – отсканировав все метки, работник попадет на интерфейс для приема заявки.

3. Посмотреть список непринятых заявок – работник может выбрать одну из заявок в списке заявок, которые начали принимать сканированием меток, но не закончили – у них часть грузов уже числится на складе, а часть нет.

4. Открыть интерфейс для приема заявки – работник должен открыть окно с функциями для продолжения приема заявки.

5. Добавить груз – работник может добавить новый груз в заявку, введя его размеры, вес и количество.

6. Изменить груз – работник может изменить размеры груза и его вес, а также добавить к нему упаковку.

7. Удалить груз – работник может удалить груз, которого на самом деле не существует.

8. Завершить прием заявки – работник должен завершить прием заявки и распечатать необходимые документы.

9. Распечатать документы – работник должен распечатать недостающие документы к заявке.

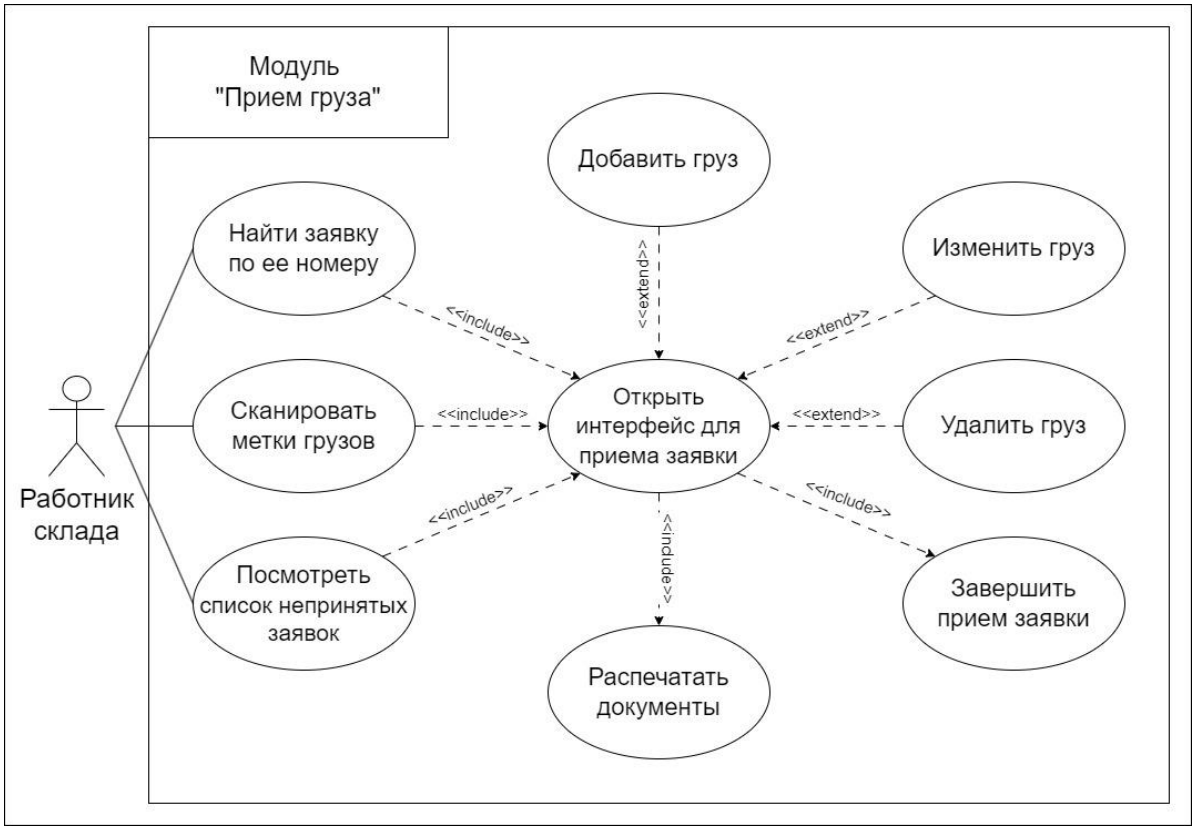

Рисунок 3 – Диаграмма для модуля «Прием груза»

На рисунке 4 представлена диаграмма вариантов использования для модулей «Погрузка», «Выгрузка» и «Ревизия». На ней видны следующие варианты использования для актера «Грузчик».

1. Посмотреть список машин – грузчик может посмотреть список машин, у которых назначена или идет погрузка или выгрузка грузов.

2. Выбрать машину – грузчик может выбрать какую машину хочет загрузить или разгрузить.

3. Сканировать метки грузов – грузчик должен сканировать метки всех грузов, с которыми он работает.

Также на ней видны следующие варианты использования для актера «Работник склада».

1. Начать ревизию – работник может начать ревизию.

2. Сканировать метки грузов – работник склада должен сканировать метки всех грузов, которые он находит на складе.

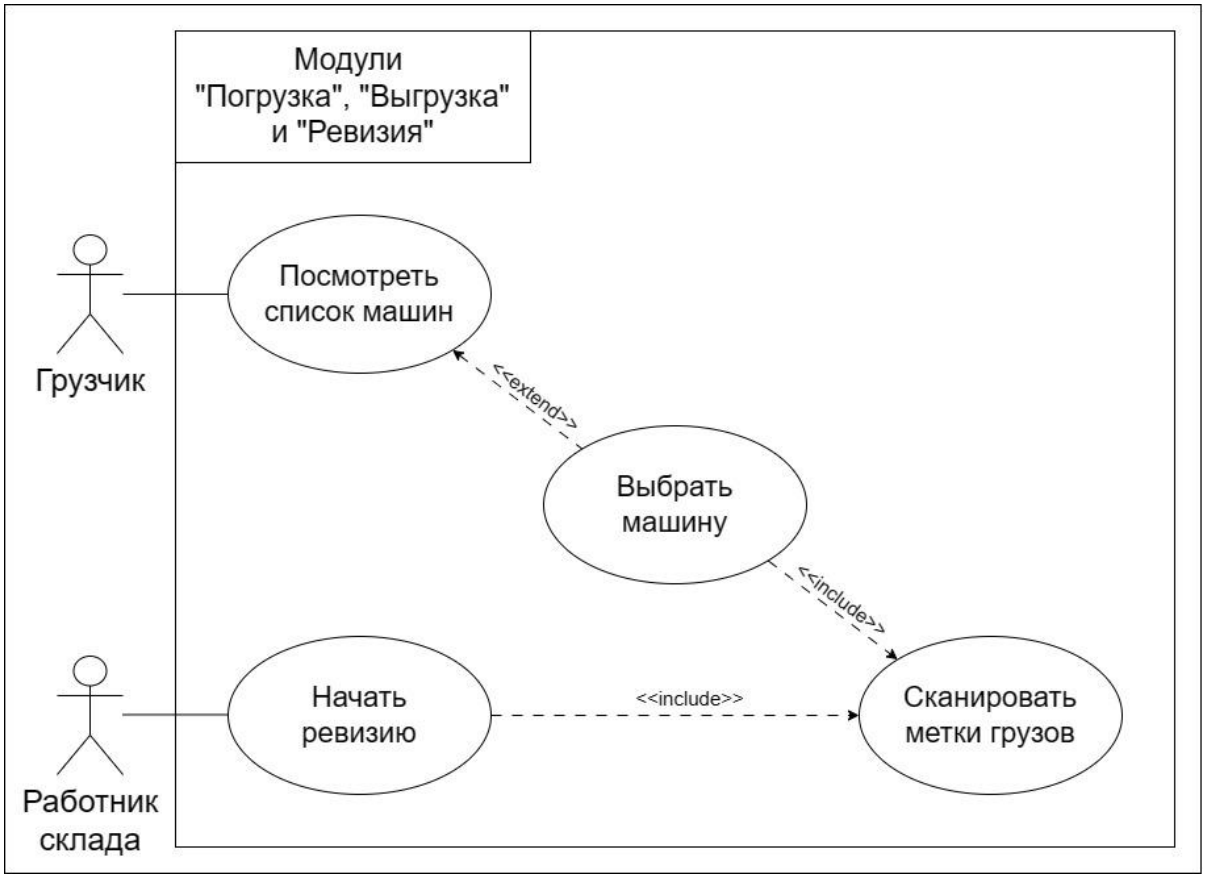

Рисунок 4 – Диаграмма для модулей «Погрузка», «Выгрузка» и «Ревизия»

На рисунке 5 представлена диаграмма вариантов использования для модуля «Адресная доставка груза». На ней видны следующие варианты использования для актера «Водитель».

1. Посмотреть список машин – водитель может посмотреть список машин, назначенных на него для адресной доставки груза.

2. Сканировать метки грузов на погрузку – водитель может отсканировать метки грузов в своей машине, чтобы они были отмечены как погруженные в машину.

3. Посмотреть список заявок в машине – водитель может выбрать одну из машин в списке, чтобы посмотреть список заявок в ней.

4. Выбрать заявку – водитель может выбрать одну из заявок в списке, чтобы посмотреть информацию о ней.

5. Идентифицировать клиента – перед тем как выдать заявку, водитель должен идентифицировать клиента, отправляя ему смс сообщение с кодом, которое нужно ввести в приложении или сверяя номер паспорта клиента.

6. Ввести данные и завершить доставку – водитель должен ввести длительность ожидания доставки и количество денег, взятых у клиента за каждую услугу, завершив после этого доставку, чтобы заявка исчезла из списка заявок в машине.

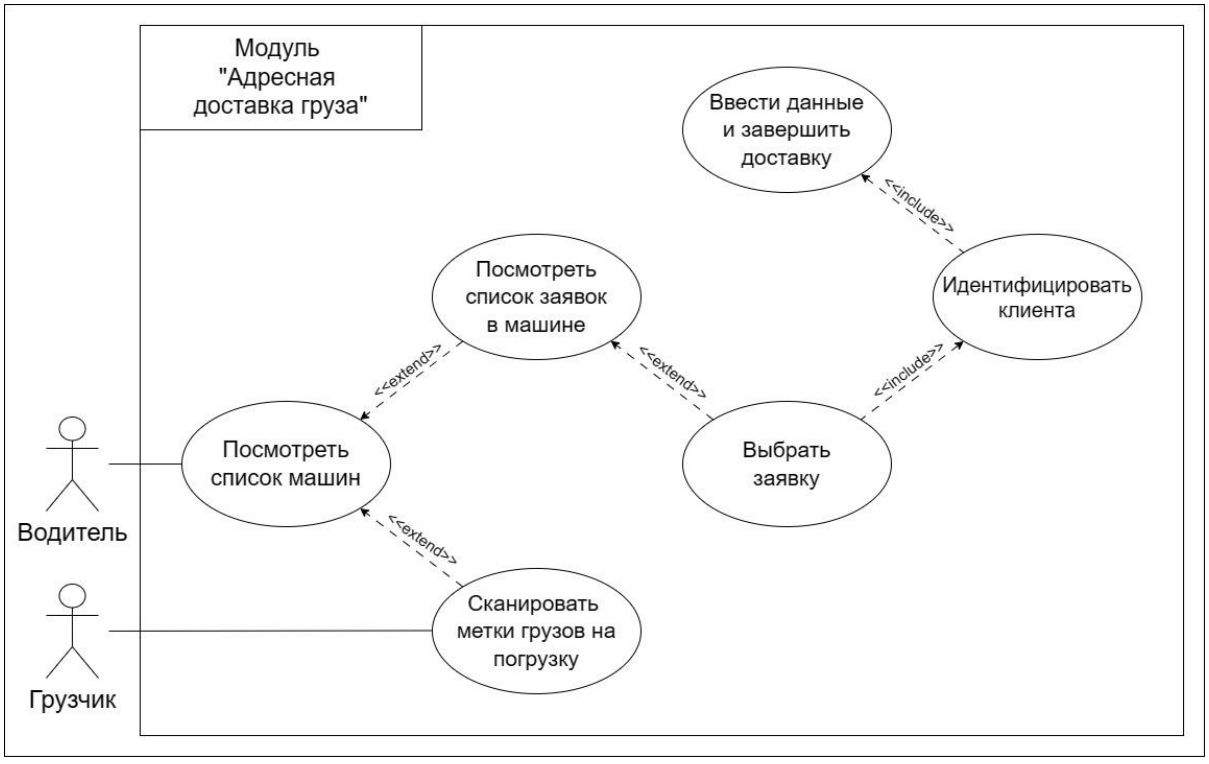

Рисунок 5 – Диаграмма для модуля «Адресная доставка груза»

Также на ней видны следующие варианты использования для актера «Грузчик».

1. Посмотреть список машин – грузчик может посмотреть список машин, назначенных на доставку на его складе.
2. Сканировать метки грузов – работник склада должен сканировать метки грузов из списка на погрузку в эту машину.

2.3. База данных

Для хранения своих данных транспортная компания «Луч» использует базу данных PostgreSQL, в которой находится уже более 100 таблиц. PostgreSQL – это реляционная (данные организованны в виде таблиц, а связи между ними устанавливаются с помощью первичных и внешних ключей) СУБД с открытым исходным кодом, поддерживающая сложные SQL-запросы и триггеры [5]. Для реализации Android-приложения будут использоваться лишь необходимая часть существующих таблицы из этой базы. В приложении А изображена неполная схема базы данных, на которой указаны только те таблицы, которые будут использоваться для реализации Android-приложения.

Для реализации Android-приложения необходимы следующие таблицы:

1) order – содержит информацию о заявке на грузоперевозку;
2) transportation – содержит информацию о том, откуда и куда направляется заявка;
3) relation_for_client – содержит информацию о связи между юридическим лицом и его представителем;
4) people_client – содержит информацию о физических лицах;
5) corporate_client – содержит информацию о юридических лицах;
6) pickup – содержит информацию об услуге адресного забора груза;
7) delivery – содержит информацию об услуге адресной доставки груза;

8) address – содержит уточняющую информацию об адресе для забора или доставки;

9) storage – содержит информацию о текущих складах транспортной компании «Луч»;

10) cargo – содержит информацию о грузах в заявках;

11) cargo_box_size – содержит информацию о размерах обычных грузов;

12) cargo_cylinder_size – содержит информацию о размерах цилиндрических грузов;

13) cargo_register – содержит информацию о регистрах грузов, для определения их местоположения (на складе/в машине);

14) document_type – содержит информацию о причине изменения регистра груза;

15) revision – содержит данные о проведенных на складах ревизиях;

16) revision_type – содержит данные о типах ревизий;

17) revision_table – содержит данные о грузах, отмеченных на ревизии.

2.4. Общее описание архитектуры системы

При проектировании архитектуры данного Android-приложения было решено логически разделить его на модули. Такой подход позволит создавать отдельные независимые компоненты, которые можно будет легко поддерживать и добавлять, не нарушая работу остальных модулей. Таким образом приложение будет состоять из следующих модулей.

1. Модуль «App» – входная точка в приложении, отвечает за авторизацию, настройки и разрешениями для перехода в нужный логистический модуль.

2. Модуль «Common» – вспомогательный модуль для общих компонентов приложения, таких как используемые во всех модулях модели, вспомогательные классы и функции, а также единые визуальные компоненты

приложения. Он необходим для того, чтобы не дублировать код в остальных модулях.

3. Модули логистики реализуют функциональные модули, указанные в п.2.1. В модулях логистики реализованы следующие логистические операции: забор груза, прием груза, погрузка, выгрузка, доставка и ревизия.

Таким образом, будет исключена прямая зависимость логистических модулей друг от друга, так как они будут зависеть только от модуля «Common». В свою очередь модуль «App» будет зависеть от всех модулей, так как он должен иметь доступ ко всем логистическим модулям для дальнейшей навигации. В приложении Б изображена диаграмма компонентов архитектуры системы. Кроме взаимодействия модулей самого Android-приложения, на диаграмме показано взаимодействие приложения со следующими компонентами:

1) с контроллерами серверной части при помощи протокола HTTPS, для получения и изменения данных;

2) с контроллером сервиса печати при помощи протокола HTTP, установленного на компьютер на складе для печати документов по запросу от Android-приложения;

3) с модулем Remote Config, предоставляемым магазином приложений RuStore [6] для удаленного управления конфигурацией Android-приложения, который будет использоваться для отслеживания актуальности версий модулей и самого приложения.

Для реализации каждого модуля было решено применить архитектурные паттерны «Clean Architecture» и MVVM (Model-View-ViewModel). Паттерн «Clean Architecture» необходим для изолирования различных компонентов приложения, чтобы улучшить его модульность и гибкость для тестирования [7]. При использовании данного паттерна приложение делится на следующие слои.

1. Слой доступа к данным – слой, отвечающий за доступ к данным приложения. Он состоит из Repository классов, которые реализуют обращения к серверу или к внутренним данным приложения.

2. Слой бизнес-логики – слой, отвечающий за правила и операции в бизнес-логике приложения. Он состоит из моделей и UseCase-классов, которые определяют варианты использования операций из классов Repository.

3. Слой представления – слой, отвечающий за представление данных и взаимодействие с пользователем.

Схема архитектуры модуля приведена на рисунке 6. На ней объясняется взаимодействие различных слоев друг с другом.

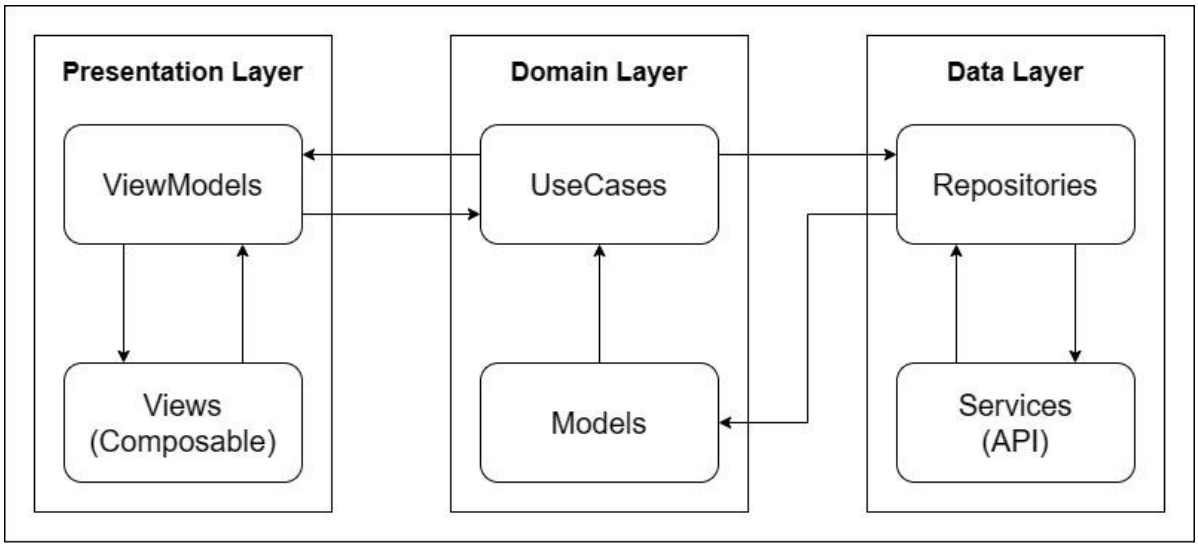

Рисунок 6 – Схема архитектуры модуля

На схеме представлены следующие элементы слоев.

1. Views (Composable) – компоненты, реализующие интерфейс, который видит пользователь. Взаимодействует с компонентами ViewModel, получая от них данные для отображения.

2. ViewModels – компоненты, которые принимают запрос на получение или отправку данных от View, и делают запрос к одному из вариантов использований (UseCase).

3. UseCases – компоненты, которые составляют запросы и преобразует ответы, которые потом отдаст ViewModel, в соответствии с бизнес-логикой приложения.

4. Repositories – компоненты, обрабатывающие ответы на запросы к внешнему API, которые делаются с помощью Service-классов, преобразуя их в модели.

5. Services – слой, формирующий и отправляющий запросы к внешнему API.

Паттерн MVVM расширяет слой представления, разделяя его на следующие три части.

1. Модель (Model) – используемые в приложении модели данных и бизнес-логика, связанная с ними.

2. Модель представления (ViewModel) – посредник между моделью и представлением. Эта часть отвечает за то, какие данные отображаются и как обрабатывать действия пользователя.

3. Представление (View) – пользовательский интерфейс. Эта часть отвечает за внешний вид приложения и отображает данные.

Выводы по второй главе

В данной главе были сформулированы функциональные и нефункциональные требования к приложению. Опираясь на логистику транспортной компании «Луч» были составлены диаграммы вариантов использования для каждого функционального модуля. Также были рассмотрены необходимые для реализации приложения таблицы в базе данных компании. Кроме этого, было составлено общее описание архитектуры приложения.

3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

3.1. Выбор технологий для реализации приложения

Для разработки Android-приложения можно использовать два подхода – мультиплатформенная разработка (для систем как на операционной системе Android, так и iOS) или нативная разработка (только для систем на операционной системе Android).

Плюсом мультиплатформенной разработки является экономия времени, так как разработчик пишет код, который будет работать сразу на нескольких операционных системах, и ему не обязательно иметь специализированные навыки, необходимые для разработки на той или иной платформе. Однако сильным минусом в нашем случае будет сложность в получении доступа к определенным функциям операционной системы, такой как сканер терминалов сбора данных. Примерами инструментов для мультиплатформенной разработки являются Xamarin [8], React Native [9], Flutter [10] и Kotlin Multiplatform [11].

К плюсам нативной разработки под Android можно отнести высокую производительность и полный доступ ко всем его функциям, так как разработчик имеет доступ к низкоуровневым API, таким как камера, геолокация и сенсоры. Минусом является то, что приложение будут работать только на операционной системе Android, что ограничивает потенциальных пользователей, но, так как мы ориентируемся на использование приложения в основном на терминалах сбора данных, нам это не так важно. Примерами инструментов для нативной разработки являются языки программирования Java [12] и Kotlin [13].

Так как мы рассматриваем разработку приложения только для операционной системы Android и для работы с терминалами сбора данных нам важен доступ к таким низкоуровневым функциям как Broadcast Receiver, была выбрана нативная разработка. В качестве языка программирования для реализации приложения был выбран Kotlin, так как компания Google, кото-

рая разработала и поддерживает операционную систему Android, на мероприятии Google I/O 2019 сделала его приоритетным языком для разработки под Android [14]. Он полностью совместим с языком программирования Java и поддерживает все ее библиотеки и фреймворки, но при этом имеет ряд преимуществ:

1) активная поддержка от Google и разработка документации и новых функций API Android именно с учетом использования Kotlin;

2) улучшенная система типов, предотвращающая ошибки еще во время компиляции;

3) поддержка библиотеки Jetpack Compose, позволяющей разрабатывать пользовательский интерфейс декларативным подходом, без использования XML файлов [15].

В качестве среды разработки была выбрана Android Studio, являющаяся официальной средой разработки под Android и предоставляющая полный набор инструментов для разработки с поддержкой Kotlin и Jetpack Compose [16].

3.2. Реализация Android-приложения

Основной функционал приложения состоит из обмена данными с сервером на основе протокола HTTP. Для реализации сетевых запросов было решен решено использовать библиотеку Retrofit [17], которая позволяет легко определять API-интерфейсы и настраивать HTTP-клиенты, используя методы, которые позволяют манипулировать запросами и ответами до отправки запроса и после получения ответа от сервера. Для работы с этой библиотекой необходимо сначала создать класс Retrofit и настроить его. Пример создания класса Retrofit представлен в листинге 1 приложения В. Для удобного использования этого экземпляра класса Retrofit в любом модуле приложения была использована библиотека Hilt [18], позволяющая реализовать паттерн «Dependency Injection». Суть данного паттерна заключается в

использовании контейнера внедрения зависимостей для предоставления зависимостей другим классам, вместо того чтобы давать классам самим создавать эти зависимости. Данный паттерн позволяет избежать повторение кода, удобно определять зависимости в классах, а также гибко настраивать сколько экземпляров класса может быть в программе. Так, в листинге 1 приложения Б, видны аннотации `@Singleton`, которые указывают, что на всю программу будет только один экземпляр данного класса Retrofit, тем самым предотвращая утечку ресурсов от создания и удаления множества экземпляров.

На сервере используется аутентификация при помощи JWT-токена. JWT-токен – это специальный формат токена, который содержит в себе необходимые данные и подпись для проверки токена [19]. При использовании такого вида аутентификации, при успешной авторизации пользователю возвращается JWT-токен, который сохраняется на устройстве и отправляется вместе с каждым запросом к серверу. Если пользователю нужна какая-то дополнительная информация из токена, то он может его легко расшифровать. Так как токен меняется после каждой авторизации, нельзя просто прописать его в коде создания HTTP-клиента. Для решения этой проблемы был использован класс-интерцептор, который срабатывает перед каждым запросом и записывает в заголовок запроса текущий токен. Кроме этого, необходимо учитывать возможность окончания времени жизни токена. В таком случае HTTP-клиент должен снова обратиться к серверу за обновлением токена. Такой функционал реализует специальный класс-интерцептор `TokenAuthenticatior`. Если с сервера придет сообщение с кодом 401 (не авторизован), то он приостановит поток, в котором выполнялся запрос и вызовет запрос на обновление токена. Если токен успешно обновится, то изначальный запрос повторится уже с новым токеном. Если токен уже не может обновиться, то сервер снова вернет ответ с кодом 401, который обрабатывается уже дальше в приложении и в итоге заставляет пользователя вернуться на экран авторизации. Исходный код класса представлен в листинге 1.

Листинг 1 – Класс `TokenAuthenticatior`

```
class TokenAuthentication @Inject constructor(
    private val authUseCases: AuthUseCases
) : Authenticator {

    override fun authenticate(route: Route?, response: Response): Request?
{
        val refreshTokenResponse = runCatching {
            runBlocking {
                authUseCases.refreshToken().first()
            }
        }

        return if (refreshTokenResponse.isSuccess) {
            val refreshedTokenFlow = refreshTokenResponse.getOrNull()

            if (refreshedTokenFlow is ApiResult.Success && refreshedToken-
Flow.data) {
                response.request().newBuilder()
                    .header("Authorization", "Bearer ${Globals.Token.ac-
cessToken}")
                    .build()
            } else {
                null
            }
        } else {
            null
        }
    }
}
```

Для того чтобы использовать JWT-токен, его необходимо хранить на устройстве пользователя и иметь возможность получить в любой момент. Для этого можно использовать библиотеки Room, SharedPreferences или PreferencesDataStore. Библиотека Room предоставляет ORM-интерфейс для работы с встроенной базой данных SQLite и позволяет хранить большие объемы структурированных данных [20]. Библиотека SharedPreferences является классическим способ хранения данных в Android в виде строковой пары ключ-значение [21]. Библиотека PreferencesDataStore предлагает такой же легковесный способ хранения данных как SharedPreferences, но позволяет сохранять данные разных типов, обеспечивая безопасность типов [22]. При этом операции сохранения и чтения данных являются асинхронными. Так как функционал библиотеки Room избыточен для хранения JWT-токена, была выбрана библиотека PreferencesDataStore. Исходный код класса для записи и чтения JWT-токена представлен в листинге 2.

Листинг 2 – Класс `TokenManager`

```
class TokenManager @Inject constructor(
    private val dataStore: DataStore<Preferences>
) {
    private val token = stringPreferencesKey("REFRESH_TOKEN")

    suspend fun saveToken(token: RefreshToken?) {
        dataStore.edit { preferences ->
            if (token == null && preferences.contains(DatastoreKey.To-
ken.key))
                preferences.remove(DatastoreKey.Token.key)
            else if (token != null)
                preferences[DatastoreKey.Token.key] = "${token.ac-
cessToken};${token.refreshToken}"
        }
    }

    val tokenFlow: Flow<RefreshToken?> = dataStore.data.map { preferences -
>
        preferences[DatastoreKey.Token.key]?.let {
            val tokenParts = it.split(";")
            RefreshToken(tokenParts[0], UUID.fromString(tokenParts[1]))
        }
    }
}
```

Для обработки ответа на запрос к серверу и отображения его результата используется класс `MatrixResult`, состоящего из следующих объектов.

1. Loading – состояние ожидания ответа от сервера.
2. Success – состояние успешного ответа от сервера, содержащее пришедшие данные.
3. NoAuth – состояние ответа с кодом 401, которое говорит о том, что у пользователя кончилось время жизни JWT-токена, и ему надо вернутся к авторизации.
4. Error – состояние ошибки с сервера, вместе с сообщением об этой ошибке.

Пример использования этого класса для отображения результата получения оплаты из модуля «Курьерский забор» представлен в листинге 3. В этом примере видно, как в зависимости от варианта класса `MatrixResult` после запроса пользователь увидит или анимацию ожидания ответа на запрос, или сообщение об результате запроса, или сообщение об ошибке при выполнении запроса, или будет возвращен на экран авторизации.

Листинг 3 – Зависимость интерфейса от состояния `MatrixResult`

```
when (val endingPickUp = viewModel.endingResult.value) {
        is MatrixResult.Loading -> LuchLoadingDialog(text = "Завершение
забора")
        is MatrixResult.Success -> {
            if (endingPickUp.data) {
                val many = viewModel.cargoesCount.value.toInt() > 1
                LuchAlertDialog(
                    title = if (many) "Заявки забраны" else "Заявка
забрана",
                    text = if (many) "Заявки успешно забраны, возвращайтесь
к оставшимся заявкам" else "Заявка успешно забрана, возвращайтесь к остав-
шимся заявкам",
                    onDismiss = {
                        viewModel.returnState()
                        navController.popBackStack()
                    }
                )
            }
        }
        is MatrixResult.NoAuth -> ErrorDialog(
            title = "Сессия истекла",
            text = "Чтобы продолжить забор грузов, авторизуйтесь в приложе-
нии заново"
        ) {
            viewModel.returnState()
            navController.navigate(AppScreens.LoginScreen.route) {
                popUpTo(AppScreens.MainScreen.route) {
                    inclusive = true
                }
            }
        }
        is MatrixResult.Error -> ErrorDialog(text = endingPickUp.message)
{ viewModel.returnState() }
    }
```

Так как для реализации приложения было решено использовать паттерн MVVM, модули состоят из классов View, ViewModel и Model.

Реализация классов View

Классы View используются для построения интерфейса приложения и состоят из Composable-функций (функций, помеченных аннотацией `@Composable`) из библиотеки Jetpack Compose. Основным преимуществом использования Composable-функций является их модульность – часто повторяющийся элемент интерфейса можно вынести в отдельную функцию и использовать в разных местах. Также это можно использовать для реализации единообразного интерфейса в приложении, создав свои собственные Composable-функции для реализации кнопок, текстовых полей и прочих элементов, использующихся во всех модулях. Для отображения данных на

интерфейсе в Composable-функциях используются специальные компоненты State, которые при изменении данных, сразу же обновляют часть интерфейса, в которой эти данные используются. Для примера View-класса в листинге 2 приложения В представлен исходный код Composable-функций, используемых для отображения данных о заявке, используемые в модуле «Pickup». Представленные в данном листинге Composable-функции состоят не только из стандартных компонентов, но и из следующих Composable-функций из модуля «Common»:

1) `ShimmerListItem` – элемент списка с переливающимся эффектом для отображения процесса загрузки данных;

2) `LuchButton` и `LuchOutlinedButton` – собственная реализация кнопок, стилизованных под общий стиль интерфейса;

3) `LuchExpandedCard` – сворачивающаяся/раскрывающаяся по нажатию карточка;

4) `LuchSmallListItem` и `LuchClickableListItem` – текстовые элемент списка, состоящий из заголовка и текста под ним, выделенные синим если на них можно нажать;

5) `LuchChoiceDialog` – собственная реализация диалогового окна с выбором из двух действий.

Все эти Composable-функции были вынесены в модуль «Common», чтобы не дублировать код для элементов интерфейса, часто использующихся в остальных модулях.

Реализация классов ViewModel

Классы ViewModel используются для привязки данных к классам View, и обработки взаимодействия пользователя с интерфейсом. Кроме этого, ViewModel имеет свой собственный жизненный цикл и позволяет выполнять асинхронные запросы к серверу, используя корутины и не замораживая интерфейс приложения. Для примера ViewModel-класса в листинге 4 представлен исходный код класса `OrderInfoViewModel`, который используется в показанном раннее View-классе.

Листинг 4 – Класс `OrderInfoViewModel`

```
@HiltViewModel
class OrderInfoViewModel @Inject constructor(
    savedStateHandle: SavedStateHandle,
    private val authUseCases: AuthUseCases,
    private val orderUseCases: OrderUseCases
) : ViewModel() {

    private val _orderId: UUID = UUID.fromString(check-
NotNull(savedStateHandle["orderId"]))

    private val _order = mutableStateOf<ApiResult<Order?>>(ApiResult.Load-
ing)
    val order: State<ApiResult<Order?>> = _order

    val refreshTokenState = mutableStateOf(true)

    fun checkRefreshToken() {
        viewModelScope.launch {
            val result = authUseCases.refreshToken().last()
            if (result is ApiResult.Success) {
                refreshTokenState.value = result.data
            } else {
                refreshTokenState.value = false
            }
        }
    }

    fun getOrderToDisplay() {
        viewModelScope.launch {
            orderUseCases.getOrderById(_orderId).collect {
                if (it is ApiResult.Success)
                    PickUpGlobals.PickUpOrders = mutableListOf(it.data)

                _order.value = it
            }
        }
    }
}
```

Реализация классов Model

Классы Model представляют собой модели данных, которые отображаются на интерфейсе. ViewModel-классы получают эти данные используя UseCases – специальные методы, обозначающие варианты взаимодействия с данными. Эти методы получают данные с сервера и передают их во ViewModel-классы в исходном варианте или учитывая бизнес-логику приложения. UseCases-методы обращаются к репозиториям – классам, содержащим методы для запросов к серверу и обработке ответов от них. Пример исходного кода модели заявки и UseCase-метода для ее получения представлены в листинге 5.

Листинг 5 – Модель заявки

```
data class OrderOrManifestInfo(
    val appointmentId: UUID,
    val id: UUID,
    val number: String,
    val isManifest: Boolean,
    val commentary: String?,

    val senderId: UUID,
    val senderLegacyId: UUID?,
    val senderName: String,
    val senderPhone: String,
    val cityFrom: String,

    val address: String?,
    val addressDetails: String?,
    val geoLongitude: Double,
    val geoLatitude: Double,

    val consigneeName: String,
    val consigneePhone: String,
    val cityTo: String,

    val ordersCount: Int,
    val cargoesCount: Int,
    val cargoesWeight: Double,
    val cargoesVolume: Double
)

class GetOrderOrManifestInfoInteractor @Inject constructor(
    private val pickUpRepository: PickUpRepository
) {
    suspend operator fun invoke(documentId: UUID) = pickUpRepository.getOr-
derOrManifestInfo(documentId)
}
```

Реализация сканирования

Для того, чтобы сканировать QR-метки грузов, необходимо реализовать два вида сканеров: для обычных смартфонов с помощью задней камеры и для терминалов сборов данных с помощью встроенного лазерного сканера.

Для реализации сканирования с камеры смартфона была использована библиотека Ml Kit от Google [23], которая позволяет использовать функции машинного обучения на смартфонах, в нашем случае это распознавание QR-кодов. В листинге 3 приложения В представлен класс, использующий функцию анализа QR-кода из этой библиотеки.

Для реализации сканирования с терминалов сбора данных используется встроенная в Android SDK возможность регистрации широковещатель-

ных приемников. Они позволяют обрабатывать широковещательные сообщений различных видов: подключение к WIFI, зарядке или наушникам. В нашем случае это обработка сообщений от лазерного сканнера. Пример исходного кода для регистрации приемника представлен в листинге 6.

Листинг 6 – Регистрация широковещательного приемника

```
val filter = IntentFilter("android.intent.action.SCANRESULT")
registerReceiver(context, receiver, filter, RECEIVER_EXPORTED)
```

Реализация проверки доступа к модулям

При авторизации пользователя, в ответе с сервера также приходят его роли, в зависимости от которых определяется доступен ему модуль или нет. Помимо этого, каждый раз, когда пользователь оказывается на главном экране, происходит проверка на актуальность версий – с помощью инструмента RuStore Remote Config. Он позволяет управлять конфигурацией мобильного приложения из удобного графического интерфейса [6]. С его помощью приложение получает список актуальных версий модулей, и если в приложении используются старая версия модуля, то он становится неактивным, а при нажатии на него пользователю предложит перейти на страницу приложения в RuStore для обновления приложения.

3.3. Реализация пользовательского интерфейса

Для реализации единого стиля пользовательского интерфейса в приложении, сначала были созданы часто используемые в интерфейсе компоненты, упомянутые в главе 3.2, такие как кнопки, текстовые поля, диалоги с пользователем и стили текста, а также элементы интерфейса, еще официально не реализованные в библиотеке Jetpack Compose, например кнопка со множественным выбором. Названия всех таких компонентов начинается со слова «Luch», чтобы их легко было отличить в коде. В качестве основного цвета интерфейса был выбран желтый цвет, так как это фирменный цвет транспортной компании «Луч». Для примера таких компонентов, в листинге 7 представлен исходный код стилизованной кнопки.

Листинг 7 – Composable-функция для обычной кнопки

```
@Composable
fun LuchButton(
    text: String,
    modifier: Modifier,
    onClick: () -> Unit,
) {
    Button(
        onClick = onClick,
        shape = RoundedCornerShape(16.dp)) {
        Text(
            text = text,
            style = MaterialTheme.typography.bodyMedium
        )
    }
}
```

При первом запуске приложения пользователь должен пройти авторизацию, которая состоит из двух экранов – для ввода телефона и пароля. После авторизации пользователь попадет на главный экран приложения со списком всех доступных ему модулей. Эти экраны представлены на рисунке 7.

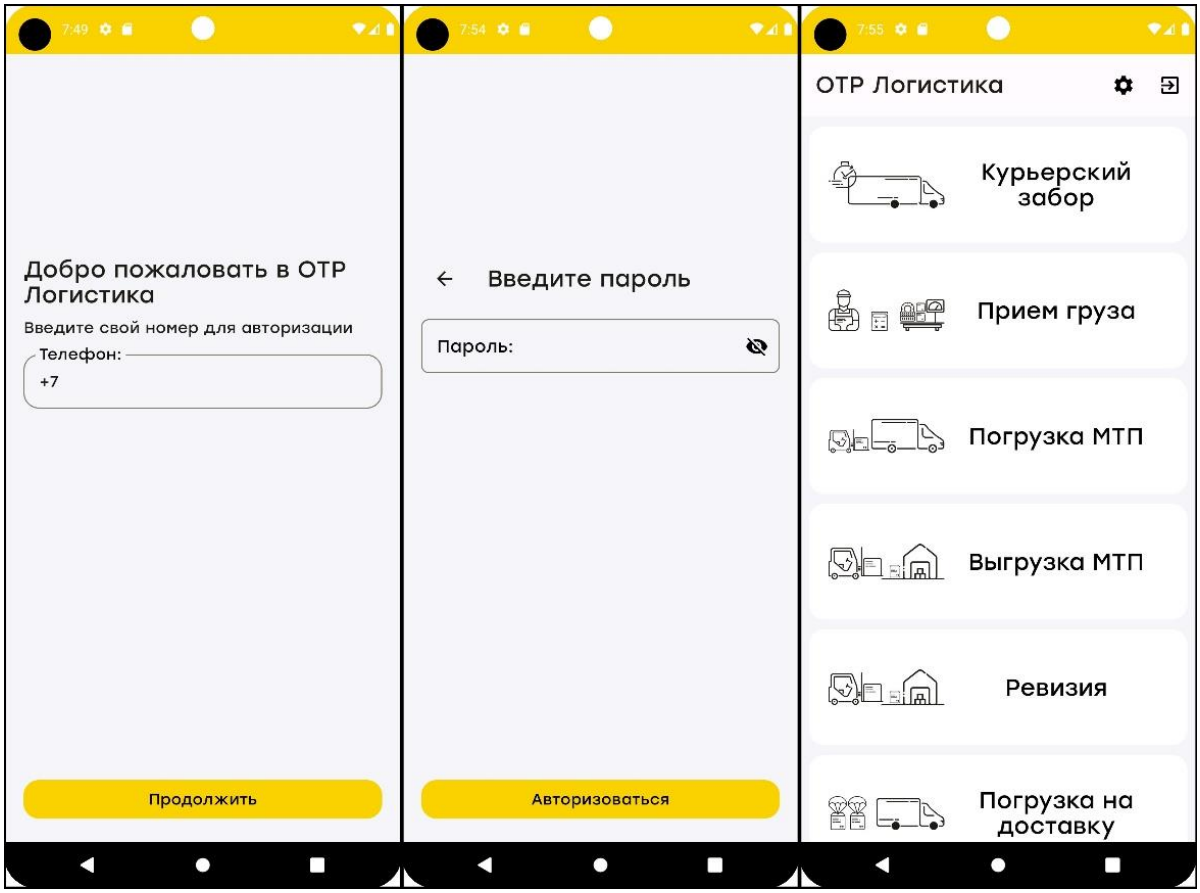

Рисунок 7 – Экраны авторизации и главный экран

В модуле «Курьерский забор» пользователь видит список заявок на забор груза. Если водителю нужно забрать манифест заявок, то он может выбрать, какие именно заявки он забирает. Пользователь также имеет возможность определить очередность, в которой он собирается ехать на эти заявки. На рисунке 8 представлен список заявок и интерфейс для определения очередности.

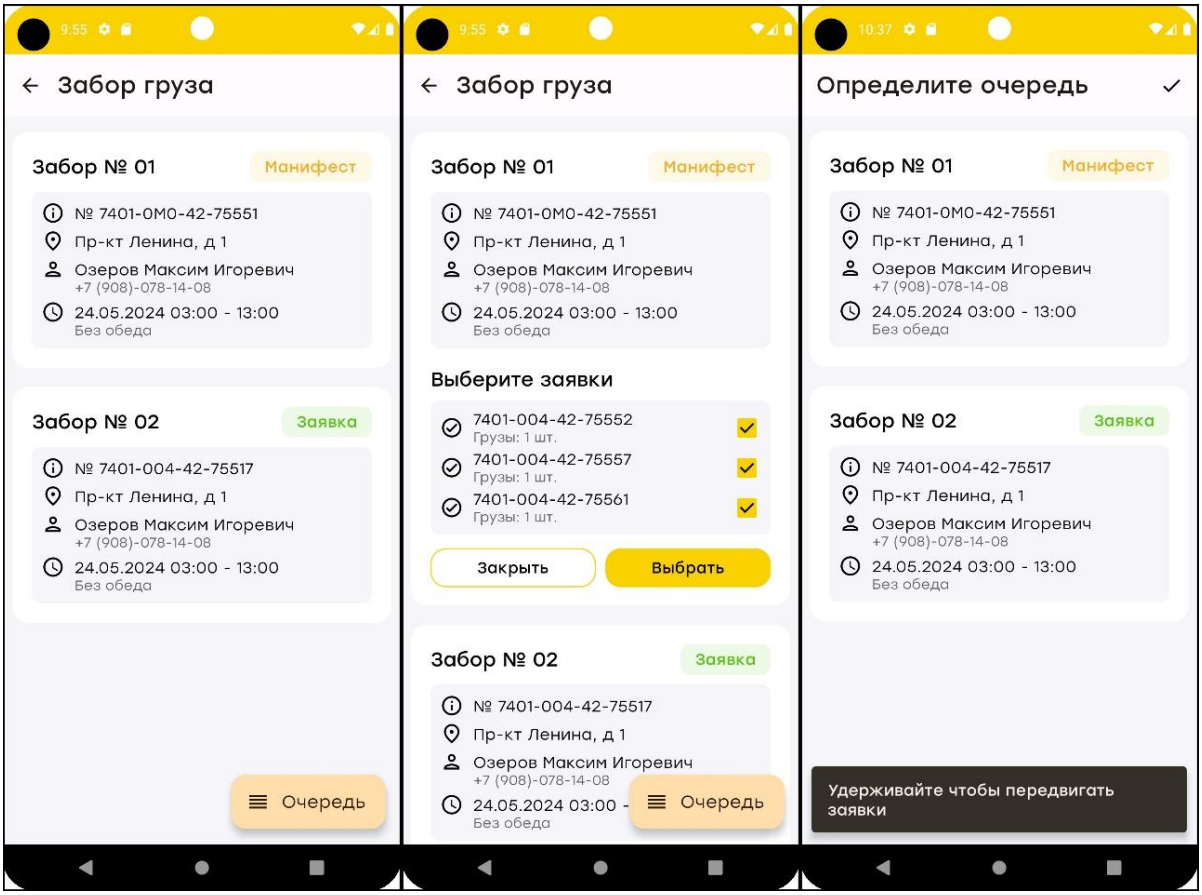

Рисунок 8 – Списки заявок на забор груза

После того как пользователь выбрал заявку, он может посмотреть информацию о ней – данные отправителя, получателя и о грузах. Для удобства пользователя он может сразу открыть карту при нажатии на адрес забора, а также позвонить клиентам, при нажатии на их телефоны. Для окончания забора заявки пользователь должен ввести время ожидания, если он ждал клиента, а также собственный комментарий к грузам и желаемую клиентом упаковку к нему. После этого, если за услугу забора груза платит отправитель,

то пользователь должен ввести полученную сумму, иначе он может завершить забор без оплаты. Экран с информацией о заявке и экран завершения курьерского забора груза представлены на рисунке 9.

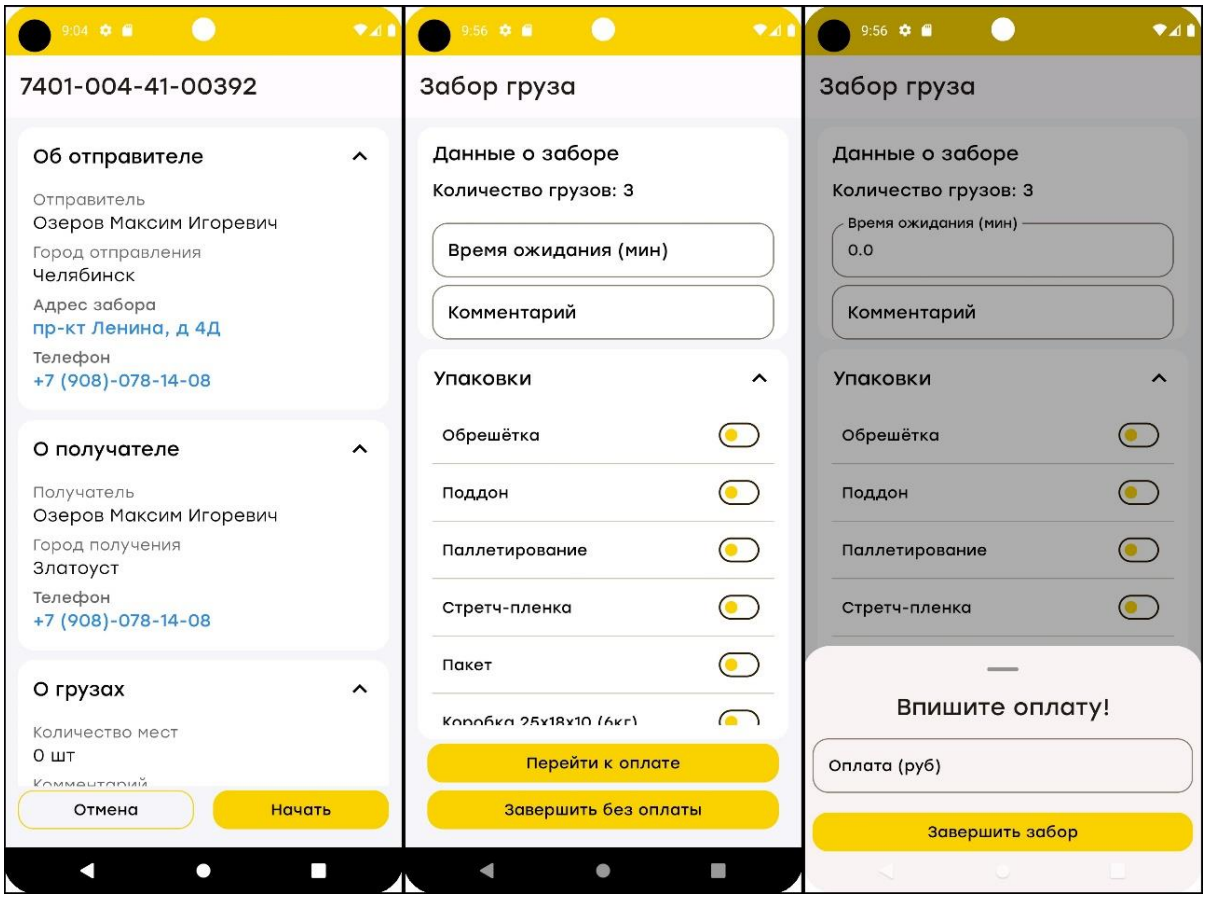

Рисунок 9 – Экраны для модуля «Курьерский забор»

В модуле «Прием груза» пользователь имеет возможность начать прием груза по номеру из накладной или начав сканирование меток на грузах. Также он видит список заявок с незавершенным приемом – кто-то начал сканировать груза этих заявок, но не закончил. Во всех случаях пользователь снова попадает на экран с информацией о заявке, откуда переходит на экран для приема груза. Этот экран содержит всю необходимую при приеме груза информацию – заказанные клиентом упаковки и информацию о грузах, которые уже записал пользователь. После того, как пользователь занес всю информацию о грузах, он сохраняет заявку и переходит к печати необходимых документов – QR-кодов и накладных. Экран поиска заявок и экран приема заявки представлены на рисунке 10. Кроме этого в модуле приемки есть

экраны для ввода данных о размерах и весе груза, а также дополнительных упаковках, в которые этот груз положат. Транспортная компания «Луч» разделяет грузы на 3 типа: стандартный, цилиндрический и шины. Экраны для заполнения данных всех трех типов груза представлены на рисунке 3 приложения Г.

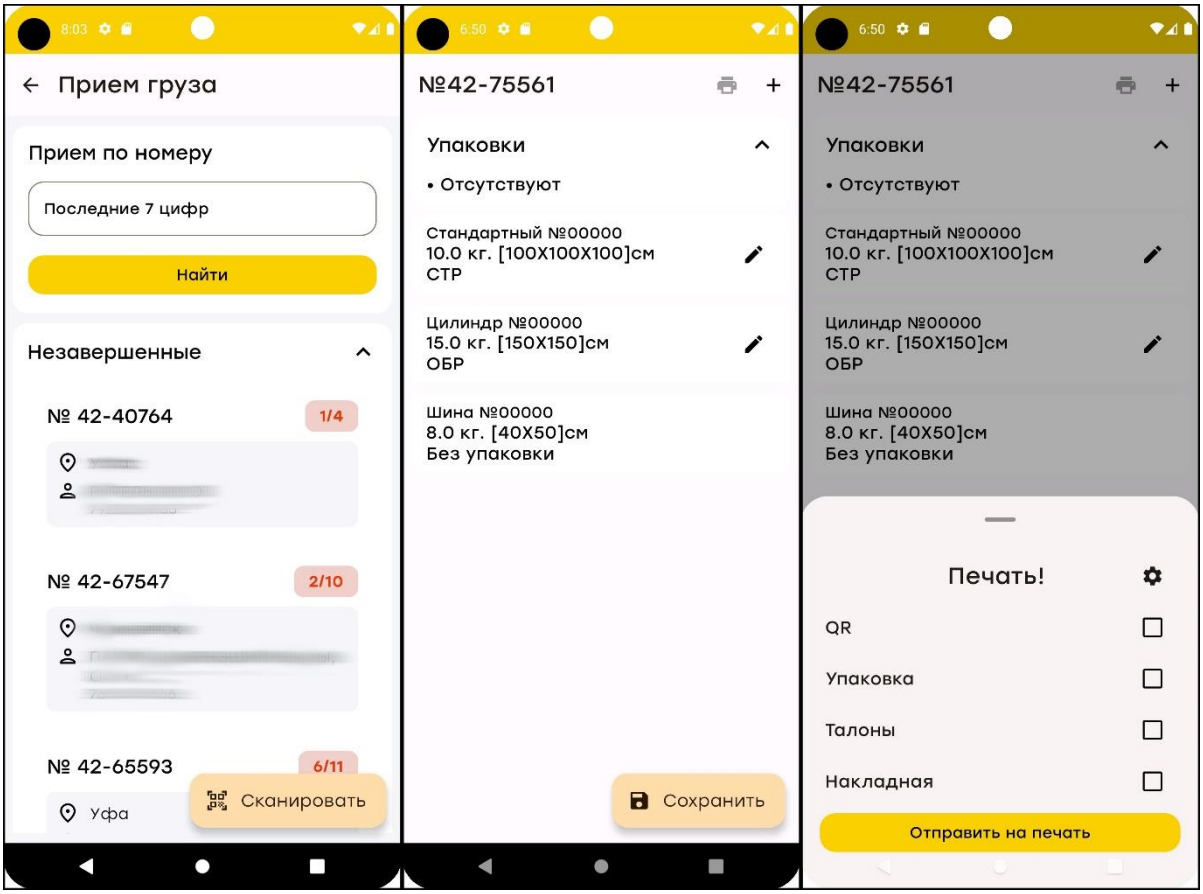

Рисунок 10 – Экраны для модуля «Прием груза»

В модулях «Погрузка», «Выгрузка» и «Погрузка на доставку» пользователь видит список машин готовых к погрузке/выгрузке. После выбора машины открывается камера для сканирования QR-кодов, если пользователь использует смартфон, или экран с сообщением о том, что можно начинать сканировать, если пользователь использует терминал сбора данных. В модуле «Погрузка на доставку», кроме этих двух экранов, есть экран со списком грузов, которые надо погрузить, который можно открыть при сканировании. Экраны из модуля «Погрузка на доставку» со списком машин и грузов и сканнер представлены на рисунке 11.

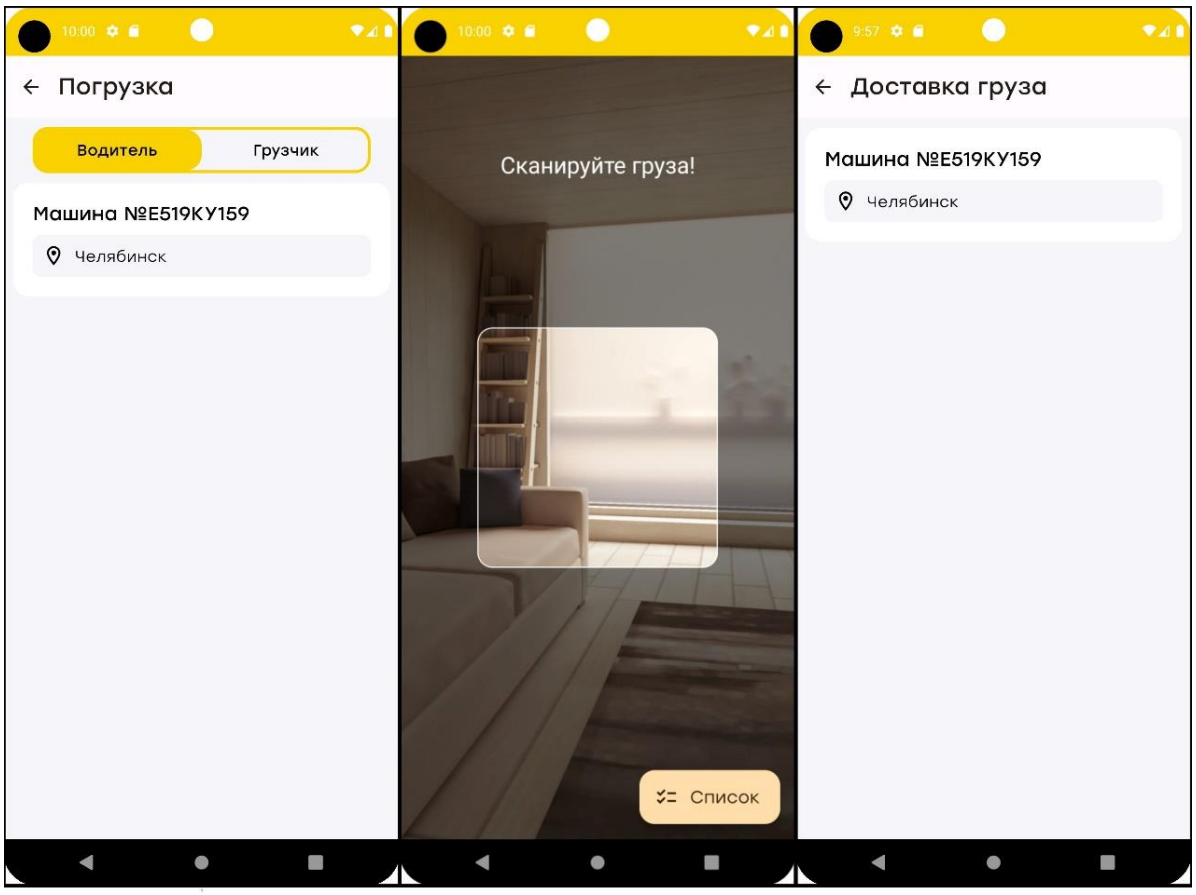

Рисунок 11 – Экраны для модуля «Погрузка на доставку»

В модуле «Курьерская доставка» пользователь видит список связанных с ним машин. При выборе машины он видит список заявок, погруженных в нее и информацию о том, оплачена ли эта заявка полностью. Если заявка не оплачена, то он не сможет ее выдать. При выборе оплаченной заявки пользователь увидит полную информацию о ней, как в модуле «Курьерский забор», после чего перейдет на экран для идентификации клиента. Клиента можно идентифицировать по номеру паспорта или отправив ему СМС с кодом для подтверждения, который он скажет пользователю. При успешной идентификации пользователь переходит к экрану завершения доставки, где он вводит время ожидания и количество полученных от клиента денег за различные услуги. Экраны из модуля «Курьерская доставка» представлены на рисунке 12.

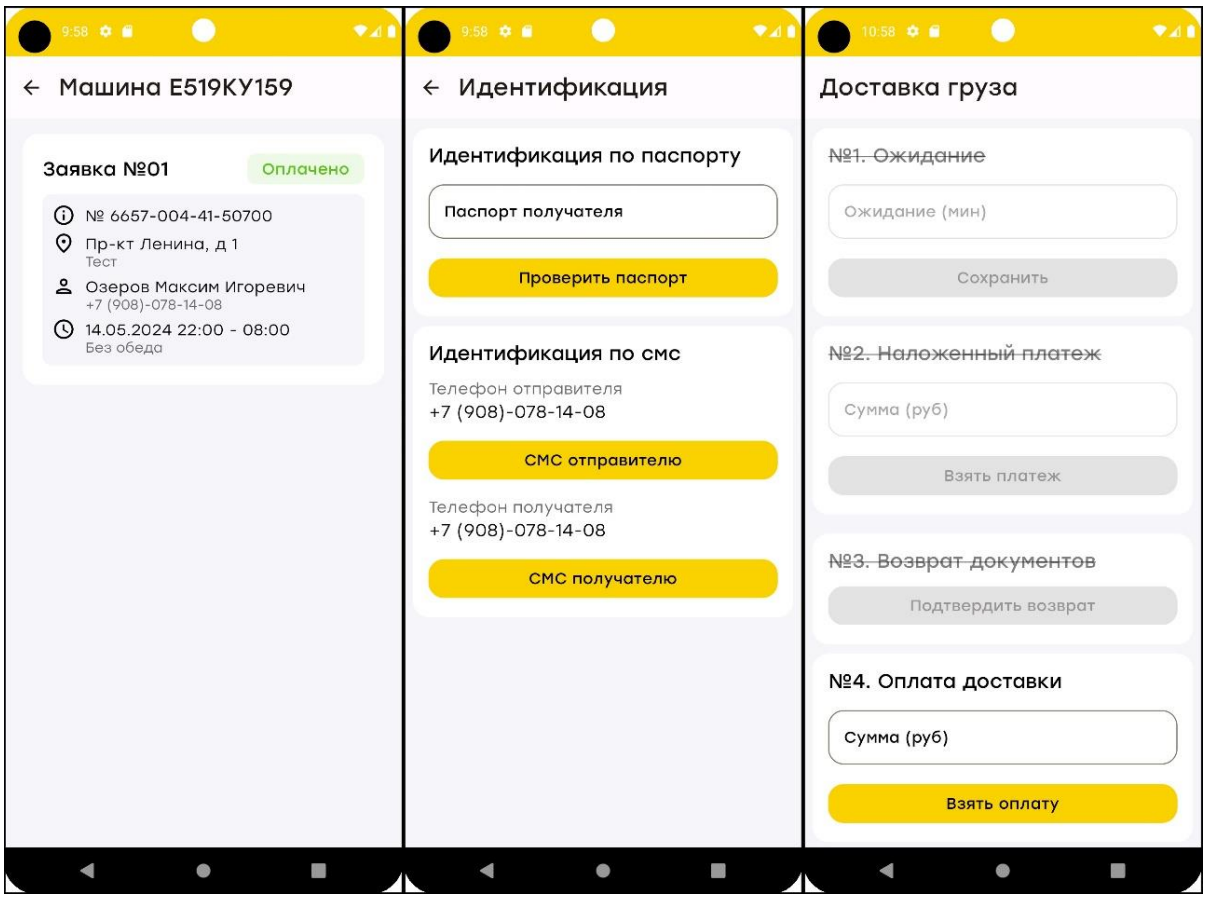

Рисунок 12 – Экраны модуля «Курьерская доставка»

Также в приложении есть настройки, в которых пользователь может увидеть свои личные данные и версию приложения, установить IP-адрес компьютера, к которому он будет подключаться для печати документов, включить режим для терминала сбора данных, чтобы сканировать лазерным сканнером, настроить видимость кнопки завершения погрузки и выгрузки, а также включить функцию «Автопогрузка» в модуле «Погрузка на доставку», чтобы отмечать груза, которые не назначены на машину. Экран настроек представлен на рисунке 4 приложения Г.

3.4. Реализация серверной части

Серверная часть приложения реализована в существующем REST API транспортной компании «Луч», который реализован на платформе ASP .NET Core [24], а в качестве базы данных используется СУБД Post-

greSQL. Для каждого модуля был создан отдельный контроллер – класс, который является входной точкой серверной части REST API и принимает запросы. Контроллеры после получения запроса обращаются к классам-сервисам, которые содержат основную бизнес-логику приложения, и репозиториям, предназначенным для работы с данными. Для выполнения запросов к базе данных используется библиотека Dapper, которая позволяет удобно трансформировать результаты SQL запросов в модели данных [25]. В листинге 8 представлен исходный код запроса для получения заявок из машины на доставку.

Листинг 8 – Запрос для получения заявок из машины на доставку

```
SELECT mo.pk1_fk_order AS {{nameof(OrderToDelivery.OrderId)}},
       mo.n AS {{nameof(OrderToDelivery.OrderNumber)}},
       COALESCE(mcc.title,spc.name) AS {{nameof(OrderToDelivery.Consign-
eeName)}},
       spc.phone AS {{nameof(OrderToDelivery.ConsigneePhone)}},
       st.fk_city_to AS {{nameof(OrderToDelivery.DeliveryCity)}},
       COALESCE(sa.street, 'Борт') AS {{nameof(OrderToDelivery.Deliver-
yStreet)}},
       sa.details AS {{nameof(OrderToDelivery.DeliveryDetails)}},
       sd.delivery_timestamp_from AS {{nameof(OrderToDelivery.DeliveryTime-
From)}},
       sd.delivery_timestamp_to AS {{nameof(OrderToDelivery.DeliveryTi-
meTo)}},
       sd.delivery_break_from AS {{nameof(OrderToDelivery.LunchTimeFrom)}},
       sd.delivery_break_to AS {{nameof(OrderToDelivery.LunchTimeTo)}}
FROM delivery_traffics dt
  INNER JOIN delivery_document dd ON dt.id = dd.delivery_traffics_id
  INNER JOIN mdt_cargo mc ON dd.cargo_id = mc.pk1_fk_cargo
  INNER JOIN mdt_order mo ON mc.fk_order = mo.pk1_fk_order
  INNER JOIN st_relation_for_client srfc ON mo.fk_consignee = srfc.st_rela-
tion_for_client
  INNER JOIN st_people_client spc ON srfc.fk_client = spc.pk_people_client
  LEFT JOIN mds_corporate_client mcc ON srfc.fk_corporate_client =
mcc.pk1_fk_client
  INNER JOIN st_transportation st ON mo.pk1_fk_order = st.pk1_fk_order
  LEFT JOIN st_delivery sd ON mo.pk1_fk_order = sd.pk1_fk_order
  LEFT JOIN st_address sa ON sd.fk_address = sa.pk_address
WHERE dt.id = :{{nameof(carId)}} AND
      NOT dt.is_archived AND
      NOT dd.is_archived AND
      mc.b_current AND
      mo.b_current AND
      st.b_current
GROUP BY mo.pk1_fk_order, mo.n, COALESCE(mcc.title,spc.name), spc.phone,
st.fk_city_to, COALESCE(sa.street, 'Борт'), sa.details,ьsd.deliv-
ery_timestamp_from, sd.delivery_timestamp_to, sd.delivery_break_from,
sd.delivery_break_to
HAVING SUM(CASE WHEN dd.is_loaded THEN 1 ELSE 0 END) = COUNT(dd) AND
       SUM(CASE WHEN dd.delivered THEN 1 ELSE 0 END) <> COUNT(dd);
```

3.5. Публикация приложения

Для возможности легко обновлять приложение у всех пользователей и иметь доступ к статистике этого приложения и ее аналитике, было решено опубликовать приложение в одном из официальных магазинов мобильных приложений. На данный момент приложение опубликовано только в официальном российском магазине мобильных приложений RuStore [26]. Был выбран именно этот магазин, так как с 1 января 2023 года он является обязательным к предустановке на всех продающихся смартфонах в России [27]. Также публикация приложения в RuStore позволяет использовать такие встроенные инструменты, как RuStore Remote Config, для динамического управления конфигурацией приложения, и RuStore Tracer, для отслеживания статистики по различным техническим проблемам приложения, в том числе вылеты из приложения, утечки памяти и зависания интерфейса. Эти инструменты позволяют быстро обнаруживать и решать проблемы, не дожидаясь обратной связи от пользователей.

Выводы по третьей главе

В данной главе на основе требований, определенных во второй главе, были разработаны Android-приложение и его серверная часть. Были выбраны инструменты разработки и реализованы основная архитектура приложения и все необходимые логистические модули, а также SQL-запросы и бизнес-логика на серверной части. После этого приложение было опубликовано в официальном российском магазине мобильных приложений для Android RuStore.

4. ТЕСТИРОВАНИЕ

4.1. Функциональное тестирование

Функциональное тестирование – это тестирование с целью проверить способность программы решать задачи, нужные пользователям. Тестирование проводилось на эмуляторах смартфона Pixel 8 с версиями Android от 10 до 13, а также на терминале сбора данных iData 70 с версией Android 10, которым пользуются работники склада. Набор тестов и их результаты представлены в таблице 1. Все тесты были успешно выполнены.

Таблица 1 – Протокол функционального тестирования

№	Название теста	Шаги	Ожидаемый результат
1	Авторизация пользователя с ролью «Водитель»	1. На экране логина ввести телефон. 2. На экране пароля ввести пароль. 3. Нажать на кнопку «Авторизоваться».	Переход на экран со списком модулей, где будут только модули «Курьерский забор», «Курьерская доставка» и «Погрузка на доставку». При следующем включении приложения, если у JWT-токена пользователя осталось время жизни, то пользователь сразу будет авторизован и перейдет на экран со списком модулей.
2	Выход из профиля	1. На экране списка модулей нажат на кнопку выхода из профиля. 2. Подтвердить выход.	Переход на страницу авторизации.
3	Настройка неправильного IP-адреса	1. Перейти на экран настроек. 2. Ввести IP-адрес в неправильном формате. 3. Сохранить IP-адрес	Сообщение о неправильном формате IP-адреса.
4	Забор груза с оплатой	1. Перейти в модуль «Курьерский забор». 2. Выбрать заявку из списка на забор. 3. Проверить данные о заявке и нажать на кнопку «Начать». 4. Ввести время ожидания, комментарий к заявке и выбрать упаковки к грузам. 5. Нажать на кнопку «Перейти к оплате». 6. Ввести сумму денег, которую отдал клиент и нажать «Завершить забор».	Сообщение об успешном завершении забора и возвращение на страницу с заявками на забор. Забранная заявка исчезает из списка. В базе данных у заявки меняется статус на «У курьера».

Продолжение таблицы 1

№	Название теста	Шаги	Ожидаемый результат
5	Прием груза по номеру заявки	1. Перейти в модуль «Прием груза». 2. Ввести последние 7 цифр номера заявки. 3. Проверить данные о заявке и нажать на кнопку «Начать». 4. Ввести объявленную стоимость заявки и перейти на следующий экран. 5. Добавить данные об измеренных грузах. 6. Сохранить заявку. 7. Отправить запрос на печать необходимых документов.	Переход обратно на экран поиска заявок. В базе данных у заявки появляются новые груза и статус меняется на «Грузоперевозка».
6	Прием груза по номеру несуществующей заявки	1. Перейти в модуль «Прием груза». 2. Ввести последние 7 цифр номера уже принятой заявки.	Сообщение о том, что заявка была принята более чем 15 минут назад и ее переприем запрещен.
7	Прием груза по номеру заявки, которую приняли час назад	1. Перейти в модуль «Прием груза». 2. Ввести последние 7 цифр номера уже принятой заявки.	Сообщение о том, что заявка была принята более чем 15 минут назад и ее переприем запрещен.
8	Прием груза по номеру заявки, склад отправления которой не совпадает со складом пользователя	1. Перейти в модуль «Прием груза». 2. Ввести последние 7 цифр номера заявки, склад отправления которой не совпадает со складом пользователя.	Сообщение о том, что склад пользователя не совпадает с городом отправления заявки.
9	Прием груза по QR-меткам	1. Перейти в модуль «Прием груза». 2. Нажать на кнопку «Сканировать». 3. Отсканировать все груза и перейти на экран с данными о заявке. 4. Ввести объявленную стоимость заявки и перейти на следующий экран. 5. Добавить упаковки. 6. Сохранить заявку. 7. Отправить запрос на печать необходимых документов.	Переход обратно на экран поиска заявок. В базе данных у грузов меняется склад нахождения на склад пользователя, а статус заявки меняется на «Грузоперевозка».
10	Погрузка груза при помощи смартфона	1. Перейти в модуль «Погрузка МТП». 2. Выбрать одну из машин в списке. 3. Отсканировать груз с помощью камеры.	Сообщение об успешной погрузке. В базе данных у груза меняется регистр и в ПО груз отображается как находящийся в машине.

Окончание таблицы 1

№	Название теста	Шаги	Ожидаемый результат
11	Выгрузка груза при помощи терминала сбора данных	1. Перейти в модуль «Выгрузка МТП». 2. Выбрать одну из машин в списке. 3. Отсканировать груз с помощью сканера на терминале сбора данных.	Сообщение об успешной выгрузке. В базе данных у груза меняется регистр и в ПО груз отображается как находящийся на складе.
12	Ревизия груза	1. Перейти в модуль «Ревизия». 2. Начать ревизию. 3. Отсканировать груз. 4. Ввести ячейку и сохранить.	Переход обратно на страницу сканнера. В базе данных у груза меняется поле ячейки, и если у груза был указан другой склад, то меняется на склад пользователя.
13	Ревизия груза, у которого не совпадает склад получения со складом пользователя	1. Перейти в модуль «Ревизия». 2. Начать ревизию. 3. Отсканировать груз.	Сообщение о несоответствии склада получения со складом пользователя и просьба отсканировать метку буфера склада, для подтверждения, что пользователь отнес туда груз.
14	Погрузка грузов в машину на доставку	1. Перейти в модуль «Погрузка на доставку». 2. Выбрать машину для погрузки. 3. Отсканировать все груза. 4. Перейти на экран со списком грузов и нажать на кнопку «Завершить».	Переход на страницу со списком машин. Погруженная машина исчезла из списка. В базе данных статус машины меняется на «Погружена».
15	Погрузка груза без услуги доставки в машину на доставку	1. Перейти в модуль «Погрузка на доставку». 2. Выбрать машину для погрузки. 3. Отсканировать груз, который не назначен на доставку.	Сообщение о том, что груз не назначен на доставки и его нужно отнести в буфер, после чего обратиться к ответственному логисту.
16	Доставка груза	1. Перейти в модуль «Курьерская доставка». 2. Выбрать машину из списка. 3. Выбрать оплаченную заявку из списка. 4. Проверить данные о заявке и нажать на кнопку «Начать». 5. Идентифицировать клиента через код из СМС или номер паспорта. 6. Ввести все полученные оплаты в необходимые поля.	Сообщение об успешной завершении доставки и возвращение на страницу с заявками на доставку. Доставленная заявка исчезает из списка. В базе данных у заявки меняется статус на «Выдана».

4.2. Юзабилити тестирование

Юзабилити тестирование – это тестирование пользовательского интерфейса и его взаимодействия с реальным пользователем. В данном тестировании принимали участие 5 наиболее ответственных сотрудников склада транспортной компании «Луч» в Челябинске – им предложили протестировать основной функционал приложения. По итогу тестирования были найдены следующие проблемы:

1) все текстовые поля, находящиеся в нижней половине экрана, при использовании оказывались перекрыты клавиатурой, и пользователи не видели, что они в них вводят;

2) на экране изменения очереди заявок на адресный забор груза практически невозможно прокручивать список, так как из-за слишком короткой задержки для захвата заявки палец цепляется за заявку и вместо скроллинга списка перемещает ее;

3) из-за специфичных настроек шрифта на смартфоне у пользователя часть текста в интерфейсе сливалась с фоном и была нечитаемая.

Первая проблема была решена переносом всех текстовых полей в верхнюю часть экрана. Вторая проблема была решена увеличением времени удержания пальца на заявке для начала ее движения. Чтобы долгая задержка не смутила пользователя, также было добавлено сообщение с инструкцией. Третья проблема была решена самостоятельной настройкой цвета шрифта в приложении, чтобы он не зависел от настроек конкретного смартфона.

Вывод по четвертой главе

В данной главе было проведено тестирование юзабилити, а также сформированы и пройдены тесты для функционального тестирования основных вариантов использования приложения на различных устройствах. Все функциональные тесты были пройдены успешно, а проблемы интерфейса были исправлены.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано Android-приложение для решения задач логистики транспортной компании «Луч», а также серверная часть для его работы. Были решены следующие задачи.

1. Проведен анализ предметной области и аналогов.
2. Спроектировано Android-приложение и графический интерфейс.
3. Реализованы необходимые методы на серверной части.
4. Реализовано приложение.
5. Проведено тестирование приложения.

В ходе выполнения данной работы мною были изучены основы разработки Android-приложения при помощи языка программирования Kotlin, с использованием библиотек Jetpack Compose для реализации интерфейса, Retrofit для реализации REST API запросов и DataStore для локального хранения данных на смартфоне. Кроме это я научился публиковать Android-приложение в официальных магазинах для Android-приложений и дорабатывать существующие решения в REST API сервере реализованного на платформе ASP .NET Core.

В будущем планируется добавлять в приложение новые модули, например, «Выдача груза» и «Помощь клиентам», а также добавить регистрацию для наемных пользователей-водителей. Кроме этого, планируется доработать дизайн приложения с помощью команды дизайнеров транспортной компании «Луч» и распространять приложение в других магазинах для Android-приложений.

В данный момент приложение внедрено в промышленную эксплуатацию на большинстве складов транспортной компании «Луч», в том числе в Челябинске и Златоусте.

ЛИТЕРАТУРА

1. ТК Луч. [Электронный ресурс] URL: https://тк-луч.рф/ (дата обращения: 11.02.2024 г.).

2. IT-компания Wellsoft. [Электронный ресурс] URL: https://well-soft.pro/ (дата обращения: 11.02.2024 г.).

3. АЙТОБ. [Электронный ресурс] URL: https://itob.ru/ (дата обращения: 11.02.2024 г.).

4. Вигерс Карл. В41. Разработка требований к программному обеспечению. // Пер, с англ. – М.: Издательско-торговый дом «Русская Редакция», 2004. – 576 с.

5. PostgreSQL: The World's Most Advanced Open Source Relational Database. [Электронный ресурс] URL: https://www.postgresql.org/ (дата обращения 11.02.2024 г.).

6. RuStore Remote Config: основы, быстрый старт и руководство по управлению конфигурацией. [Электронный ресурс] URL: https://www.rustore.ru/help/developers/tools/remote-config/general (дата обращения 11.02.2024 г.).

7. Роберт Мартин. Чистая архитектура. Искусство разработки программного обеспечения // Пер, с англ. – М.: Издательский дом «Питер», 2022. – 352с.

8. Xamarin. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/xamarin/ (дата обращения: 11.02.2024 г.).

9. React Native. [Электронный ресурс] URL: https://reactnative.dev/ (дата обращения: 11.02.2024 г.).

10. Flutter. [Электронный ресурс] URL: https://flutter.dev/ (дата обращения: 11.02.2024 г.).

11. Kotlin Multiplatform. [Электронный ресурс] URL: https://kotlinlang.org/docs/multiplatform.html (дата обращения: 11.02.2024 г.).

12. Java. [Электронный ресурс] URL: https://www.java.com/ru/ (дата обращения: 11.02.2024 г.).

13. Kotlin. [Электронный ресурс] URL: https://kotlinlang.org/ (дата обращения: 11.02.2024 г.).

14. Recap – Google I/O 2019. [Электронный ресурс] URL: https://events.google.com/io2019/recap (дата обращения: 11.02.2024 г.).

15. Jetpack Compose UI App Development Toolkit. [Электронный ресурс] URL: https://developer.android.com/jetpack/compose/ (дата обращения: 11.02.2024 г.).

16. Android Studio. [Электронный ресурс] URL: https://developer.android.com/studio/ (дата обращения: 11.02.2024 г.).

17. Retrofit. [Электронный ресурс] URL: https://square.github.io/retrofit/ (дата обращения: 11.02.2024 г.).

18. Hilt. [Электронный ресурс] URL: https://dagger.dev/hilt/ (дата обращения: 11.02.2024 г.).

19. Introduction to JSON Web Tokens. [Электронный ресурс] URL: https://jwt.io/introduction (дата обращения: 11.02.2024 г.).

20. Save data in a local database using Room. [Электронный ресурс] URL: https://developer.android.com/training/data-storage/room (дата обращения: 11.02.2024 г.).

21. Save simple data with SharedPreferences. [Электронный ресурс] URL: https://developer.android.com/training/data-storage/shared-preferences (дата обращения: 11.02.2024 г.).

22. App Architecture: Data Layer – DataStore. [Электронный ресурс] URL: https://developer.android.com/topic/libraries/architecture/datastore (дата обращения: 11.02.2024 г.).

23. ML Kit. [Электронный ресурс] URL: https://developers.google.com/ml-kit/guides (дата обращения: 11.02.2024 г.).

24. ASP .NET Core. [Электронный ресурс] URL: https://dotnet.microsoft.com/ru-ru/apps/aspnet (дата обращения: 11.02.2024 г.).

25. Learn Dapper. [Электронный ресурс] URL: https://www.learndapper.com/ (дата обращения: 11.02.2024 г.).

26. RuStore. Официальный магазин приложений для Android. [Электронный ресурс] URL: https://www.rustore.ru/ (дата обращения: 11.02.2024 г.).

27. Изменения, которые вносятся в перечень программ для электронных вычислительных машин, странами происхождения которых являются Российская Федерация или другие государства – члены Евразийского экономического союза, которые должны быть предварительно установлены на отдельные виды технически сложных товаров в 2023 году: Распоряжение Правительства Российской Федерации от 08.11.2022 № 3363-р.

ПРИЛОЖЕНИЯ

Приложение А. Схема базы данных

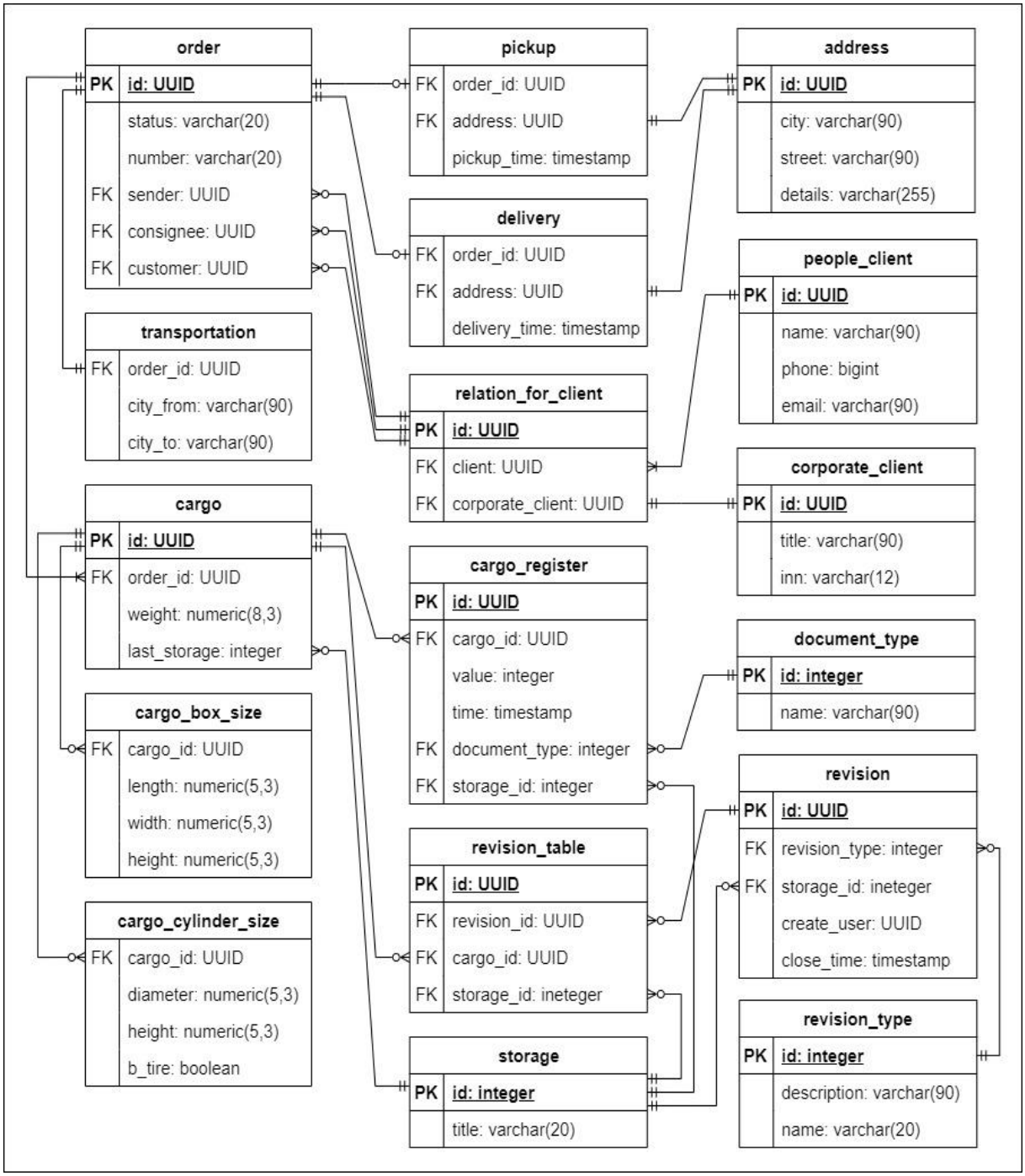

Рисунок 1 – Схема базы данных

Приложение Б. Диаграмма компонентов архитектуры системы

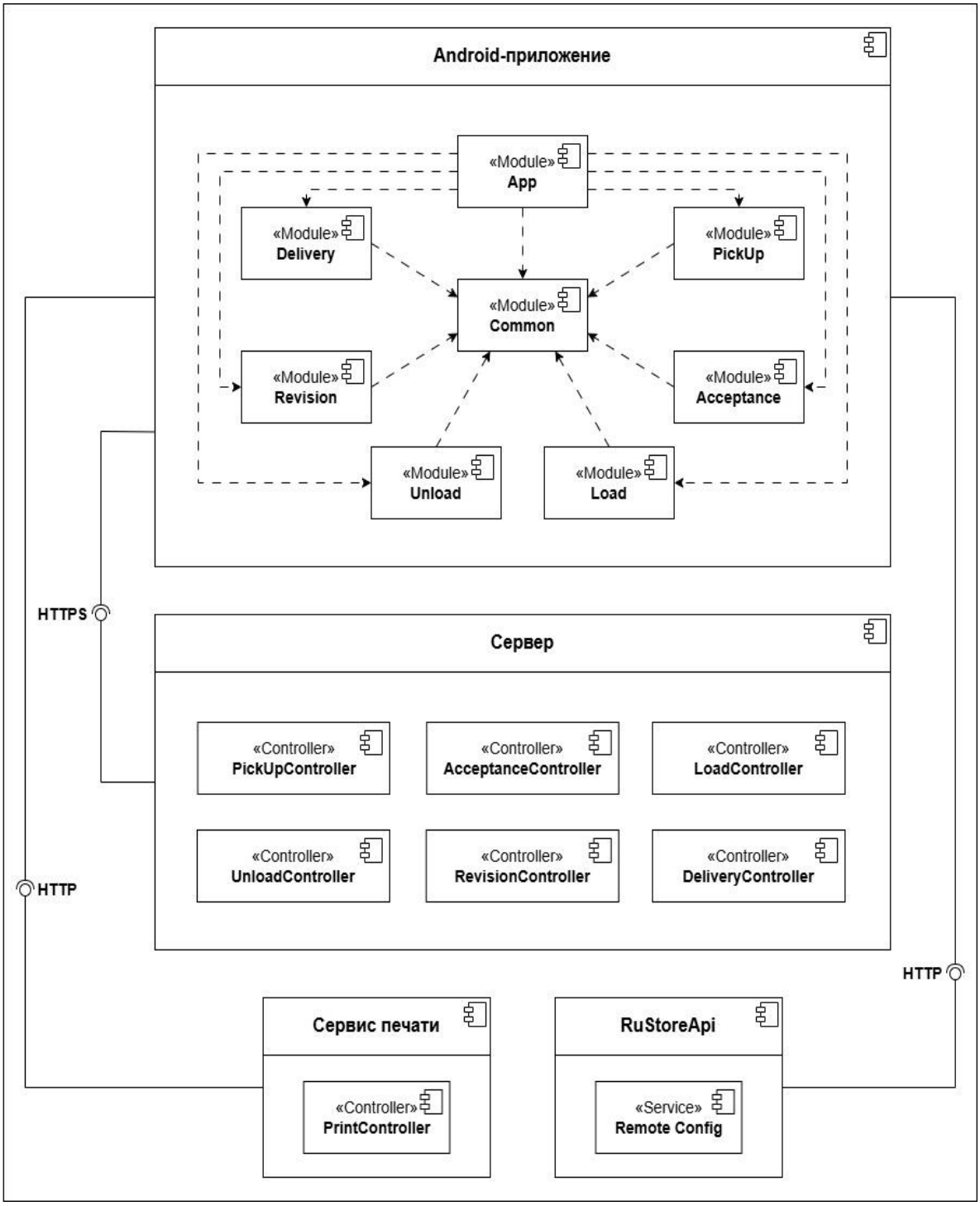

Рисунок 2 – Диаграмма компонентов архитектуры система

Приложение В. Листинги исходного кода Android-приложения

Листинг 1 – Создание класса `Retrofit`

```
@Module
@InstallIn(SingletonComponent::class)
object MatrixModule {

    @Singleton
    @Provides
    fun provideTokenInterceptor() : TokenInterceptor {
        return TokenInterceptor()
    }

    @Singleton
    @Provides
    fun provideTokenAuthentication(authV2UseCases: AuthUseCases) : TokenAu-
thentication {
        return TokenAuthentication(authV2UseCases)
    }

    @Singleton
    @Provides
    @Named("Matrix")
    fun provideMatrixOkHttpClient(tokenInterceptor: TokenInterceptor, to-
kenAuthentication: TokenAuthentication): OkHttpClient {
        return OkHttpClient.Builder()
            .addInterceptor { chain ->
                val request = chain.request().newBuilder()
                    .addHeader("X-Luch-App", "android")
                    .addHeader("X-Luch-App-Version", Constants.MIN_VER-
SION_DATE)
                    .addHeader("User-Agent", "android${Constants.MIN_VER-
SION_DATE}")
                    .build()
                chain.proceed(request)
            }.addInterceptor(HttpLoggingInterceptor().apply {
                level = HttpLoggingInterceptor.Level.BODY
            }).addInterceptor(tokenInterceptor)
            .authenticator(tokenAuthentication)
            .build()
    }

    @Singleton
    @Provides
    @Named("Matrix")
    fun provideMatrixRetrofit(@Named("Matrix")okHttpClient: OkHttpClient):
Retrofit {
        val dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
        val formatter = DateTimeFormatter.ofPattern(dateFormat)

        return Retrofit.Builder()
            .baseUrl(if (Globals.IsTest) Constants.MATRIX_URL_TEST else
Constants.MATRIX_URL)
            .client(okHttpClient)
            .addConverterFactory(
                GsonConverterFactory.create(
                    GsonBuilder()
                        .registerTypeAdapterFactory(TypeAdapters.UUID_FAC-
TORY)
                        .registerTypeAdapter(
                            LocalDateTime::class.java,
                            JsonDeserializer { json, _, _ ->
```

Окончание листинга 1 приложения В

```
                                ZonedDateTime.parse(json.asJsonPrimi-
tive.asString).toLocalDateTime()
                            } as JsonDeserializer<LocalDateTime?>
                        )
                        .registerTypeAdapter(
                            LocalDateTime::class.java,
                            JsonSerializer<LocalDateTime?> { localDate, _,
_ ->
                                JsonPrimitive(formatter.format(localDate))
                            }
                        )
                        .create()
                )
            )
            .build()
    }
}
```

Листинг 2 – Composable-функции для отображения информации о заявке

```
@Composable
fun OrderInfoScreen(navController: NavController) {
    val viewModel: OrderInfoViewModel = hiltViewModel()

    var orderNumber by remember { mutableStateOf("") }

    LaunchedEffect(Unit) {
        viewModel.checkRefreshToken()

        delay(750)

        viewModel.getOrderToDisplay()
    }

    Scaffold(
        topBar = {
            TopAppBar(title = { Text(text = orderNumber) })
        },
    ) {
        when (val order = viewModel.order.value) {
            is ApiResult.Loading -> OrderInfoShimmer(it, navController)

            is ApiResult.Success -> {
                if (order.data != null) {
                    orderNumber = order.data!!.number
                    OrderInfo(order.data!!, it, navController)
                }
            }

            is ApiResult.Error -> ErrorDialog(text = "Ошибка при загрузке
данных о заявке") { navController.popBackStack() }
        }
    }
}

@Composable
fun OrderInfoShimmer(
    padding: PaddingValues,
    navController: NavController
) {
```

```
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(
                top = padding.calculateTopPadding() + 10.dp,
                start = 15.dp,
                end = 15.dp,
                bottom = 15.dp
            ),
        verticalArrangement = Arrangement.Bottom
    ) {
        Column(
            verticalArrangement = Arrangement.spacedBy(10.dp),
            modifier = Modifier
                .verticalScroll(state = ScrollState(0), enabled = true)
                .weight(1f)
        ) {
            ShimmerListItem(itemHeight = 280)
            ShimmerListItem(itemHeight = 85)
            ShimmerListItem(itemHeight = 85)

            Spacer(modifier = Modifier.weight(1f))

            Row(horizontalArrangement = Arrangement.SpaceEvenly) {
                LuchOutlinedButton(
                    onClick = { navController.popBackStack() },
                    modifier = Modifier
                        .padding(end = 10.dp)
                        .weight(1f)
                        .fillMaxWidth(),
                    text = "Отмена"
                )
                LuchButton(
                    onClick = { },
                    isEnabled = false,
                    modifier = Modifier
                        .padding(start = 10.dp)
                        .weight(1f)
                        .fillMaxWidth(),
                    text = "Начать"
                )
            }
        }
    }
}


@Composable
fun OrderInfo(
    order: Order,
    padding: PaddingValues,
    navController: NavController
) {
    val launcher = rememberLauncherForActivityResult(ActivityResultCon-
tracts.StartActivityForResult()) { }
    val showNextPageDialog = remember { mutableStateOf(false) }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(
```

```
                top = padding.calculateTopPadding() + 10.dp,
                start = 15.dp,
                end = 15.dp,
                bottom = 15.dp
            ),
        verticalArrangement = Arrangement.Bottom
    ) {
        Column(
            verticalArrangement = Arrangement.spacedBy(10.dp),
            modifier = Modifier
                .verticalScroll(state = ScrollState(0), enabled = true)
                .weight(1f)
        ) {
            val senderExpanded = remember { mutableStateOf(true) }
            val consigneeExpanded = remember { mutableStateOf(false) }
            val cargoesInfoExpanded = remember { mutableStateOf(true) }
            LuchExpandedCard(
                expanded = senderExpanded,
                title = "Об отправителе",
                body = { SenderCard(order, launcher) }
            )
            LuchExpandedCard(
                expanded = consigneeExpanded,
                title = "О получателе",
                body = { ConsigneeCard(order, launcher) }
            )
            LuchExpandedCard(
                expanded = cargoesInfoExpanded,
                title = "О грузах",
                body = { CargoesInfoCard(order) }
            )
        }
        Row(horizontalArrangement = Arrangement.SpaceEvenly) {
            LuchOutlinedButton(
                onClick = { navController.popBackStack() },
                modifier = Modifier
                    .padding(end = 10.dp)
                    .weight(1f)
                    .fillMaxWidth(),
                text = "Отмена"
            )
            LuchButton(
                onClick = { showNextPageDialog.value = true },
                modifier = Modifier
                    .padding(start = 10.dp)
                    .weight(1f)
                    .fillMaxWidth(),
                text = "Начать"
            )
        }
    }

    if (showNextPageDialog.value)
        LuchChoiceDialog(
            title = "Забор груза",
            text = "У данной заявки есть грузы с QR-метками?",
            cancelText = "Да",
            okText = "Нет",
            onCancel = {
                PickUpGlobals.MarkCargoRelation = mutableMapOf()
                PickUpGlobals.PickUpOrders = mutableListOf(order)
```

```
                navController.navigate(AppScreens.PickUpScan-
nerScreen.route) {
                    showNextPageDialog.value = false
                    popUpTo(AppScreens.PickUpOrderInfoScreen.route) {
                        inclusive = true
                    }
                }
            },
            onOk = {
                PickUpGlobals.PickUpOrders = mutableListOf(order)
                navController.navigate(
                    AppScreens.PickUpFinalScreen.route
                        .replace("{isNew}", false.toString())
                        .replace("{cargoesCount}", order.quanti-
tyOfCargo.toString())
                ) {
                    showNextPageDialog.value = false
                    popUpTo(AppScreens.PickUpOrderInfoScreen.route) {
                        inclusive = true
                    }
                }
            },
            onDismiss = { showNextPageDialog.value = false }
        )
}

@Composable
fun SenderCard(order: Order, launcher: ManagedActivityResultLauncher<In-
tent, ActivityResult>) {
    Column {
        LuchSmallListItem(
            title = "Отправитель",
            text = if (order.sender.corpClient == null) order.sender.name
else  order.sender.corpClient!!.title
        )
        LuchSmallListItem(
            title = "Город отправления",
            text = order.transportation.cityFrom
        )
        if (order.pickUp!!.address.address != null) {
            LuchClickableListItem(
                title = "Адрес забора",
                text = order.pickUp!!.address.address!!,
                onClick = {
                    val address =
                        "${order.pickUp!!.address.address}, ${order.trans-
portation.cityFrom}"
                    tryToOpenMap(address, launcher)
                }
            )
        }
        if (order.pickUp!!.address.details != null) {
            LuchSmallListItem(
                title = "Уточнение адреса",
                text = order.pickUp!!.address.details!!
            )
        }
        LuchClickableListItem(
            title = "Телефон",
            text = HelpFunctions.formatPhoneNumber(or-
der.sender.phone.toString()),
```

```
            onClick = { tryToPhone(order.sender.phone.toString(),
launcher) }
        )
    }
}

@Composable
fun ConsigneeCard(order: Order, launcher: ManagedActivityResultLauncher<In-
tent, ActivityResult>) {
    Column {
        LuchSmallListItem(
            title = "Получатель",
            text = if (order.consignee.corpClient == null) order.con-
signee.name else order.consignee.corpClient!!.title
        )
        LuchSmallListItem(
            title = "Город доставки",
            text = order.transportation.cityTo
        )
        LuchClickableListItem(
            title = "Телефон",
            text = HelpFunctions.formatPhoneNumber(order.con-
signee.phone.toString()),
            onClick = { tryToPhone(order.consignee.phone.toString(),
launcher) }
        )
    }
}

@Composable
fun CargoesInfoCard(order: Order) {
    Column {
        LuchSmallListItem(
            title = "Количество мест",
            text = "${order.quantityOfCargo} шт"
        )
        LuchSmallListItem(
            title = "Общая масса",
            text = "${order.cargoList.sumOf { it.weight }} кг"
        )
        LuchSmallListItem(
            title = "Общий объем",
            text = "${(order.getVolume()*1000).roundToInt().toDou-
ble()/1000} м3"
        )
        LuchSmallListItem(
            title = "Комментарий",
            text = if (order.comment != null) order.comment!! else "нет"
        )
    }
}
```

Листинг 3 – Класс для анализа QR-кода

```
class QrCodeAnalyzer(
    previewView: PreviewView,
    private val onQrCodeDetected: (String) -> Unit,
) : ImageAnalysis.Analyzer {

    private val previewWidth = previewView.width.toFloat()
    private val previewHeight = previewView.height.toFloat()
```

```
    private var scaleFactor = 0f

    private val scannerOptions = BarcodeScannerOptions.Builder()
        .setBarcodeFormats(Barcode.FORMAT_QR_CODE)
        .build()
    private val scanner: BarcodeScanner = BarcodeScanning.getClient(scan-
nerOptions)

    private var lastScannedTimeStamp: LocalTime = LocalTime.now()
    private var lastScannedValue: String = ""

    @OptIn(ExperimentalGetImage::class)
    override fun analyze(image: ImageProxy) {
        if (image.image == null) {
            image.close()
            return
        }

        val frameWidth = image.width
        val frameHeight = image.height
        scaleFactor = max(
            previewHeight / frameHeight,
            previewWidth / frameWidth
        )
        val scaledFrameHeight = frameHeight * scaleFactor
        val scaledFrameWidth = frameWidth * scaleFactor
        val marginY = (scaledFrameHeight - previewHeight) / 2
        val marginX = (scaledFrameWidth - previewWidth) / 2

        val inputImage = InputImage.fromMediaImage(image.image!!, image.im-
ageInfo.rotationDegrees)
        scanner.process(inputImage)
            .addOnCompleteListener {
                if (it.isSuccessful) {
                    it.result.let { barcodes ->
                        barcodes.forEach { barcode ->
                            barcode.boundingBox?.transform { scaledBound ->
                                scaledBound.offset(-marginX, -marginY)
                                if (LocalTime.now().nano - last-
ScannedTimeStamp.nano <= 1000000000 && barcode.rawValue == last-
ScannedValue)
                                    return@transform
                                lastScannedTimeStamp = LocalTime.now()
                                lastScannedValue = barcode.rawValue!!
                                Log.e("TAG", "Scanner: scanned next ${bar-
code.rawValue}")
                                barcode.rawValue?.let(onQrCodeDetected)
                            }
                        }
                    }

                } else {
                    it.exception?.let { exception ->
                        Log.e("QR Analyzer", exception.message.orEmpty())
                    }
                }
                image.close()
            }
    }
```

Приложение Г. Пользовательский интерфейс Android-приложения

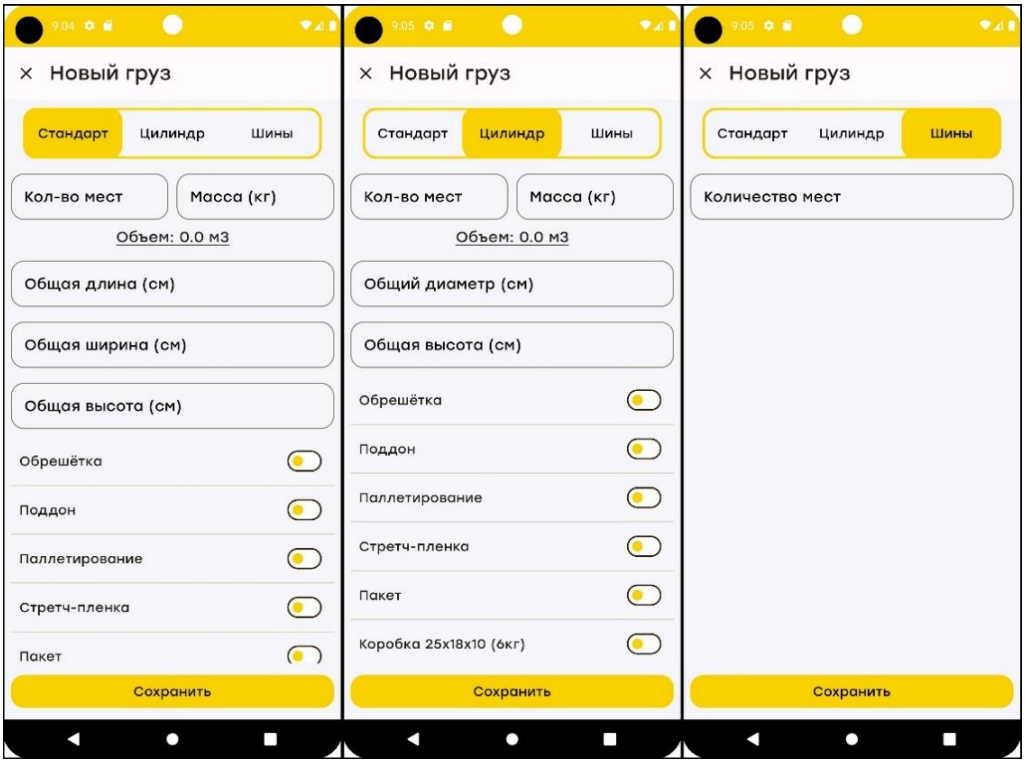

Рисунок 3 – Экраны ввода информации о грузе

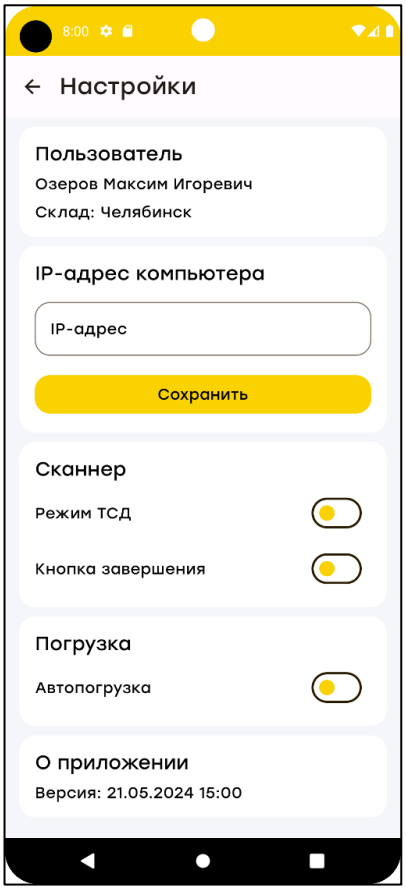

Рисунок 14 – Экран настроек