

## Реферат

Пояснительная записка дипломного проекта содержит 72 страницы пояснительной записки, 14 таблиц, 19 формул, 33 иллюстрации, 11 источников литературы, 7 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, KOTLIN, JETPACK COMPOSE, JAVA, SPRING FRAMEWORK, MS SQL SERVER, VUE.JS

Целью дипломного проекта является разработка программного средства, позволяющее упростить процесс доставки заказов для курьеров и заказчиков.

Пояснительная записка состоит из введения, семи разделов и заключения.

В первой главе проводится аналитический обзор аналогичных решений и постановка задач.

Вторая глава посвящена проектированию системы и содержит диаграмму вариантов использования, структурную схему базы данных, а также обзор инструментов разработки приложения.

В третьей главе описаны процесс разработки, принципы функционирования и назначение созданных компонентов проекта.

В четвертой главе описан процесс тестирования разработанного программного средства.

В пятой главе производится анализ информационной безопасности разрабатываемого программного средства.

В шестой главе приведено руководство пользователя, позволяющее подробно понять интерфейс приложения.

В седьмой главе производится расчет экономических параметров, срока окупаемости программного средства, разработанного в рамках дипломного проекта.

				БГТУ 00.00.ПЗ			
	Ф.И.О.	Подпись	Дата	Реферат			
Разраб.	Шичко В.С.						
Провер.	Нистюк О.А.						
Н. контр.	Нистюк О.А.						
Утв.	Смелов В.В.			74217084, 2022			
					Лит.	Лист	Листов
						1	1

## Abstract

The explanatory note of the thesis project contains 72 pages of an explanatory note, 14 tables, 19 formulas, 33 illustrations, 11 literature sources, 7 appendices.

MOBILE APPLICATION, ANDROID, KOTLIN, JETPACK COMPOSE, JAVA, SPRING FRAMEWORK, MS SQL SERVER, VUE.JS

The aim of the thesis project is to develop a software tool that simplifies the process of order delivery for couriers and customers.

The explanatory note consists of an introduction, seven sections and a conclusion.

The first chapter provides an analytical review of similar solutions and the settings of tasks.

The second chapter is devoted to system design and contains a use case diagram, a database block diagram, and an overview of application development tools.

The third chapter describes the development process, the principles of operation and the purpose of the created project components.

The fourth chapter describes the process of testing the developed software.

The fifth chapter analyzes the information security of the developed software tool.

The sixth chapter provides a user guide to help understand the application user interface in detail.

In the seventh chapter, the calculation of economic parameters, the payback period of the software tool developed as part of the thesis project is made.

				БГТУ 00.00.ПЗ			
	Ф.И.О.	Подпись	Дата	Abstract			
Разраб.	Шичко В.С.						
Провер.	Нистюк О.А.						
Н. контр.	Нистюк О.А.						
УТВ.	Смелов В.В.			74217084, 2022			
				Лит.	Лист	Листов	
					1	1	

## Содержание

Введение .....	6
1. Обзор аналогичных решений и постановка задачи.....	7
1.1 Постановка задачи.....	7
1.2 Обзор аналогичных решений.....	7
1.2.1 Приложение «MeaSoft App» .....	7
1.2.2 Приложение «Deliveries» .....	9
1.3 Выводы по разделу.....	11
2. Проектирование приложения .....	12
2.1 Функциональные требования к разработке.....	12
2.2 Диаграмма вариантов использования .....	12
2.3 База данных.....	13
2.3.1 Таблицы .....	14
2.3.2 Хранимые процедуры.....	16
2.3.3 Триггеры .....	17
2.4 Обзор инструментов разработки приложения .....	17
2.4.1 Интегрированная среда разработки Android Studio .....	17
2.4.2 Язык программирования Kotlin и библиотеки Android Jetpack .....	17
2.4.3 Интегрированная среда разработки Intelij IDEA.....	18
2.4.4 Spring Framework .....	18
2.5 Выводы по разделу.....	19
3. Разработка приложения.....	20
3.1 Разработка серверной части приложения.....	20
3.1.1 Файловая структура.....	20
3.1.2 Уровень модели приложения.....	21
3.1.2 Уровень контроллеров приложения .....	23
3.1.3 Конфигурация безопасности приложения .....	26
3.2 Разработка мобильного приложения.....	27
3.2.1 Разработка уровня бизнес-логики мобильного приложения .....	28
3.2.2 Разработка интерфейса мобильного приложения .....	32
3.3 Синхронизация с сервером.....	37
3.4 Разработка веб-клиента .....	38
3.5 Выводы по разделу.....	39
4. Тестирование приложения.....	40
4.1 Позитивное тестирование.....	40
4.2 Негативное тестирование .....	43
4.3 Выводы по разделу.....	44

				БГТУ 00.00.ПЗ			
	Ф.И.О.	Подпись	Дата	Содержание			
Разраб.	Шичко В.С.						
Провер.	Нистюк О.А.						
Н. контр.	Нистюк О.А.						
УТВ.	Смелов В.В.			74217084, 2022			

5. Анализ информационной безопасности приложения .....	45
5.1 Возможные угрозы безопасности приложения .....	45
5.2 Реализованные методы защиты приложения .....	45
5.2.1 Использование протокола HTTPS .....	45
5.2.2 Хеширование пароля .....	45
5.2.3 Механизм аутентификации .....	46
5.2.4 Методы защиты мобильного приложения .....	46
5.3 Выводы по разделу .....	47
6. Руководство пользователя .....	48
6.1 Мобильное приложение .....	48
6.2 Веб-приложение .....	58
6.3 Вывод по разделу .....	59
7. Техничко-экономическое обоснование проекта .....	60
7.1 Общая характеристика разрабатываемого программного средства .....	60
7.2 Исходные данные для проведения расчетов .....	60
7.3 Методика обоснования цены .....	61
7.4 Определение объема программного средства .....	62
7.5 Основная заработная плата .....	63
7.6 Дополнительная заработная плата .....	64
7.7 Отчисления в Фонд Социальной защиты населения .....	64
7.8 Расходы на материалы .....	65
7.9 Расходы на оплату машинного времени .....	65
7.10 Прочие прямые затраты .....	66
7.11 Накладные расходы .....	66
7.12 Сумма расходов на разработку программного средства .....	66
7.13 Расходы на сопровождение и адаптацию .....	67
7.14 Полная себестоимость .....	67
7.15 Определение цены, оценка эффективности .....	67
7.16 Выводы по разделу .....	70
Заключение .....	71
Список использованных источников .....	72
Приложение А Диаграмма вариантов использования .....	73
Приложение Б Логическая схема базы данных .....	74
Приложение В Диаграмма компонентов приложения .....	75
Приложение Г Блок-схема изменения состояния заказа .....	76
Приложение Д Пользовательский интерфейс приложения .....	77
Приложение Е Экономические показатели .....	78
Приложение Ж Листинг программного кода .....	79

## Введение

Курьерская доставка появилась достаточно давно. Основная ее суть состоит в доставке какой-либо вещи без ущерба ее состоянию и в срок. В настоящее время процесс доставки выглядит сложнее, начиная от оформления заказа заказчиком и заканчивая статусом доставки «Доставлено».

Вместе с усложнением процесса доставки, становится сложнее вести отчетность о доставке без внедрения программного обеспечения. Ведь в случае бумажного отчета будет тратиться гораздо больше времени на заполнение, отслеживание и хранение информации. Также будут возникать трудности, если заказчик захочет узнать о текущем состоянии доставки его заказа.

Функциональным назначением разрабатываемого приложения является создание запроса заказчиком на доставку заказа, последующее принятие заказа курьером, планирование маршрута и совершение доставки заказа.

Целью данного проекта является разработка программного средства для организации доставки заказов, позволяющее заказчикам создавать заказы и отслеживать состояние их доставки, а курьерам принимать доступные заказы, уведомлять пользователя о состоянии доставки, а также просматривать статистику своей работы.

Разрабатываемое приложение имеет следующие возможности для ролей:

Администратор:

- подтверждение зарегистрированных курьеров;
- подтверждение заказов;
- отмена заказов;
- просмотр статистики курьеров;
- вход в аккаунт.

Курьер:

- получение заказов;
- изменение состояния заказов;
- отказ от заказов;
- просмотр статистики;
- регистрация;
- вход в аккаунт.

Заказчик:

- создание заказа;
- регистрация;
- вход в аккаунт;
- отслеживание состояния активных заказов;
- просмотр истории заказов.

				БГТУ 00.00.ПЗ			
	Ф.И.О.	Подпись	Дата	Введение			
Разраб.	Шичко В.С.						
Провер.	Нистюк О.А.						
Н. контр.	Нистюк О.А.						
Утв.	Смелов В.В.			74217084, 2022			
				Лит.	Лист	Листов	
					1	1	

# 1 Обзор аналогичных решений и постановка задачи

## 1.1 Постановка задачи

Целью данного проекта является разработка программного средства для организации доставки заказов, позволяющее заказчикам создавать заказы и отслеживать состояние их доставки, а курьерам принимать доступные заказы, уведомлять пользователя о состоянии доставки, а также просматривать статистику своей работы.

Задачи дипломного проекта:

- изучить аналоги разрабатываемого приложения на рынке;
- спроектировать архитектуру базы данных и приложения;
- разработать базу данных и веб-приложение;
- разработать мобильное приложение;
- протестировать разработанное программное средство.

## 1.2 Обзор аналогичных решений

При создании мобильного приложения с привлекательным пользовательским интерфейсом и удобной функциональностью необходимо проанализировать аналоги. Анализ конкурентов позволяет выявить их достоинства и недостатки, а также помогает понять, к чему стоит стремиться при разработке своего приложения и выявить основные потребности пользователей.

В результате поиска аналогичных решений с похожими функциональными возможностями было отобрано два мобильных приложения. Анализ будет осуществляться путём оценки приложений по различным критериям: привлекательность интерфейса, реализованные возможности, наличие нестандартных решений и т. д.

### 1.2.1 Приложение «MeaSoft App»

Данное приложение является частью системы «MeaSoft» и предназначено только для курьеров – сотрудников курьерских служб, автоматизированных системой «MeaSoft». Оно предоставляет операторам информацию о местонахождении курьера, а также позволяет курьеру быть всегда на связи с диспетчерами [1].

Главный экран приложения представлен на рисунке 1.1. На главной странице располагается список заказов курьера с краткой информацией о каждом заказе. При сдвиге заказа влево, доступны быстрые действия, такие как: позвонить получателю, проложить маршрут, отменить заказ и т.д. Рамка вокруг заказа отображает количество оставшегося времени для доставки. Просроченные по времени заказы имеют красный фон.

				БГТУ 01.00.ПЗ						
	Ф.И.О.	Подпись	Дата							
Разраб.	Шичко В.С.			1 Обзор аналогичных решений и постановка задачи	Лит.		Лист		Листов	
Провер.	Нистюк О.А.							1	5	
					74217084, 2022					
Н. контр.	Нистюк О.А.									
Утв.	Смелов В.В.									

На верхней панели главного экрана представлена информация о количестве заказов, а также действия сортировки, поиска и обновления списка.

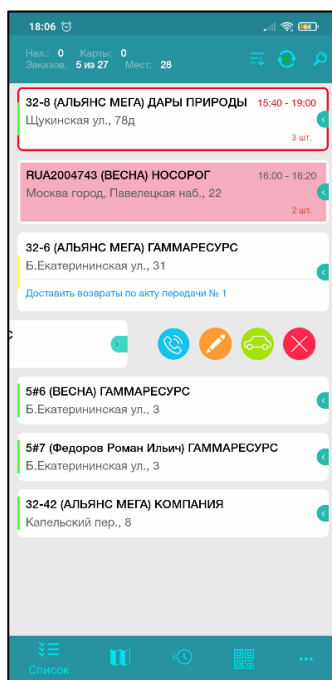


Рисунок 1.1 – Главная страница «MeaSoft App»

На экране заказа можно редактировать информацию о его массе и габаритах, изменять статус заказа, просмотреть дополнительную информацию, сделать фотографию доставленного заказа, включить таймер ожидания клиента.

Пример экрана заказа приведен на рисунке 1.2.

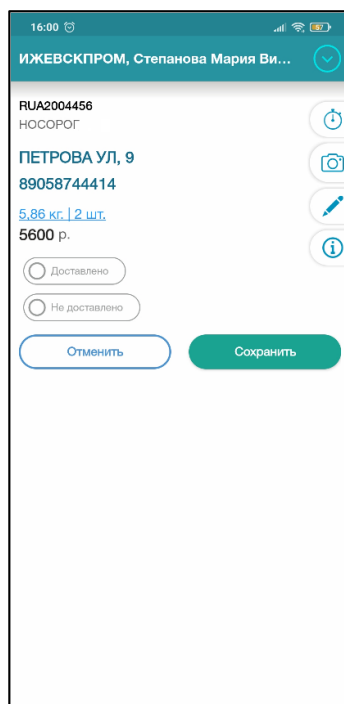


Рисунок 1.2 – Экран заказа приложения «MeaSoft App»

Также в приложении есть отдельный экран «Карта». На этом экране размещена интерактивная карта, на которой отмечены заказы и текущее положение устройства. Также есть возможность масштабировать карту, отображать информацию о пробках и фильтровать заказы. Экран «Карта» представлен на рисунке 1.3.

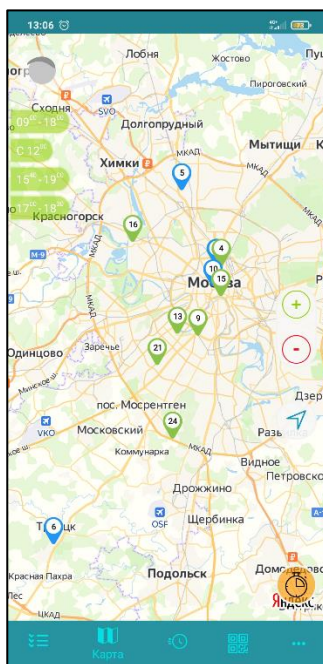


Рисунок 1.3 – Экран «Карта» приложения «MeaSoft App»

К достоинствам рассмотренного приложения можно отнести множество функций для курьера, связанных с процессом доставки, наличие интерактивной карты и отслеживания перемещений курьера.

Среди недостатков можно отметить отсутствие функциональных возможностей для заказчика, а также устаревший пользовательский интерфейс.

### 1.2.2 Приложение «Deliveries»

Приложение «Deliveries» предназначено для сотрудников курьерских компаний, позволяющее организовать рабочий день для максимальной производительности и эффективного управления временем [2]. Позволяет упорядочить список доставок по наиболее эффективному маршруту, отображает маршрут на текущий день и рассчитывает время прибытия.

Приложение имеет возможность оформить пользователю расширенную подписку для того, чтобы снять ограничение на максимальное количество создаваемых заказов в сутки.

Пользовательский интерфейс приложения включает в себя 3 основных экрана: экран доставок, экран управления и настройки.

На экране управления расположен список доставок с возможностями создавать новый заказ, изменять информацию о существующих, а также фильтровать список по состоянию доставки. Приложение не подразумевает роли заказчика, поэтому курьер должен создавать заказы в приложении сам. Пользовательский интерфейс экрана управления представлен на рисунке 1.4.



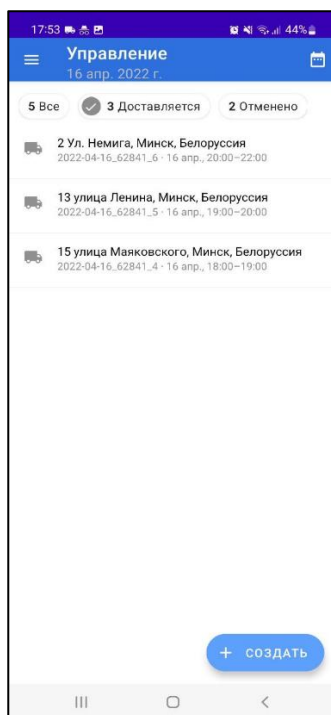


Рисунок 1.4 – Экран управления доставками приложения «Deliveries»

На экране доставок расположена карта с размещенными на ней точками маршрута, построенного с учетом временных окон доставок. Также внизу есть шкала расчетного времени доставки и информация о доставках. Экран доставок представлен на рисунке 1.5.

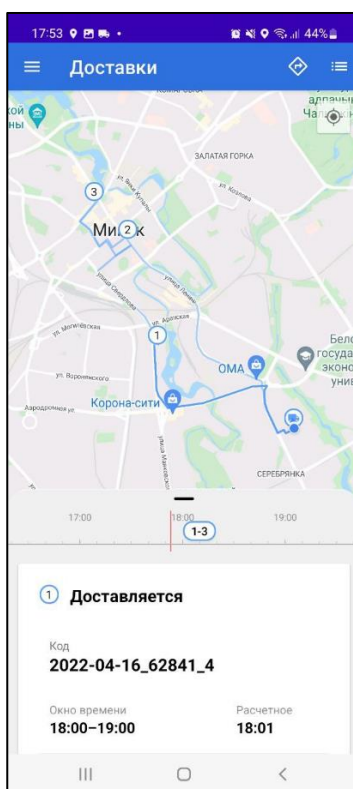


Рисунок 1.5 – Экран доставок приложения «Deliveries»

К преимуществам приложения «Deliveries» относится автоматическое планирование маршрута между доставками на сегодня с учетом времени. Среди недостатков можно выделить внесение информации о доставках курьером вручную, а также невозможность пользования приложением без подключения к сети.

### **1.3 Выводы по разделу**

В данном разделе проведен аналитический обзор существующих приложений, позволяющие организовать процесс доставки заказов, выявлены их достоинства и недостатки.

В ходе анализа существующих аналогов, была выявлена целесообразность разработки и выделен набор необходимых функциональных возможностей, которые необходимо реализовать при разработке данной программы. Также, на основании использования приложений-аналогов были выработаны основные критерии для построения дизайна приложения, позволяющего пользователю осуществлять наиболее простое и интуитивное использование.

## 2 Проектирование приложения

Основная задача дипломного проекта – создание клиент-серверного приложения, предоставляющего возможности создания заказа и отслеживание его состояния в процессе доставки.

### 2.1 Функциональные требования к разработке

Основные функции разрабатываемого приложения: вход в аккаунт и регистрация, создание, отслеживание заказов и просмотр истории, принятие заказов, изменение их состояния, а также просмотр статистики по доставленным заказам.

Для наглядного отображения функциональных возможностей разрабатываемого приложения необходимо составить диаграмму вариантов использования.

### 2.2 Диаграмма вариантов использования

Unified Modeling Language (UML) – это язык моделирования общего назначения в области разработки программного обеспечения, предназначенный для предоставления стандартного способа отображения структур системы.

Диаграмма вариантов использования – это поведенческая UML-диаграмма, описывающая набор возможностей системы (прецеденты), которые могут быть совершены одним или несколькими внешними пользователями (актерами).

В создаваемом приложении предусмотрено 4 актера:

- гость;
- заказчик;
- администратор;
- курьер.

Гостем является пользователь, не вошедший в свою учетную запись. Ему доступны регистрация и вход в приложение для доступа к основным возможностям.

Заказчик имеет следующие возможности: создание заказа, просмотр списка активных заказов и истории заказов, а также отслеживание состояния активных заказов.

Для роли курьера предусмотрены возможности просмотра списка доступных заказов, получения, отказа и изменения состояния заказа, а также просмотр своей статистики.

Администратору доступно подтверждение аккаунтов курьеров, удаление права курьера у пользователя, просмотр списка активных и доступных заказов, отмена заказов, просмотр статистики курьеров.

Диаграмма вариантов использования приведена на рисунке 2.1, а также в приложении А. В соответствии с прецедентами, изображенными на диаграмме, будет происходить дальнейшая разработка приложения.

				БГТУ 02.00.ПЗ			
	Ф.И.О.	Подпись	Дата				
Разраб.	Шичко В.С.			2 Проектирование приложения	Лит.	Лист	Листов
Провер.	Нистюк О.А.					1	8
Н. контр.	Нистюк О.А.						
Утв.	Смелов В.В.				74217084, 2022		

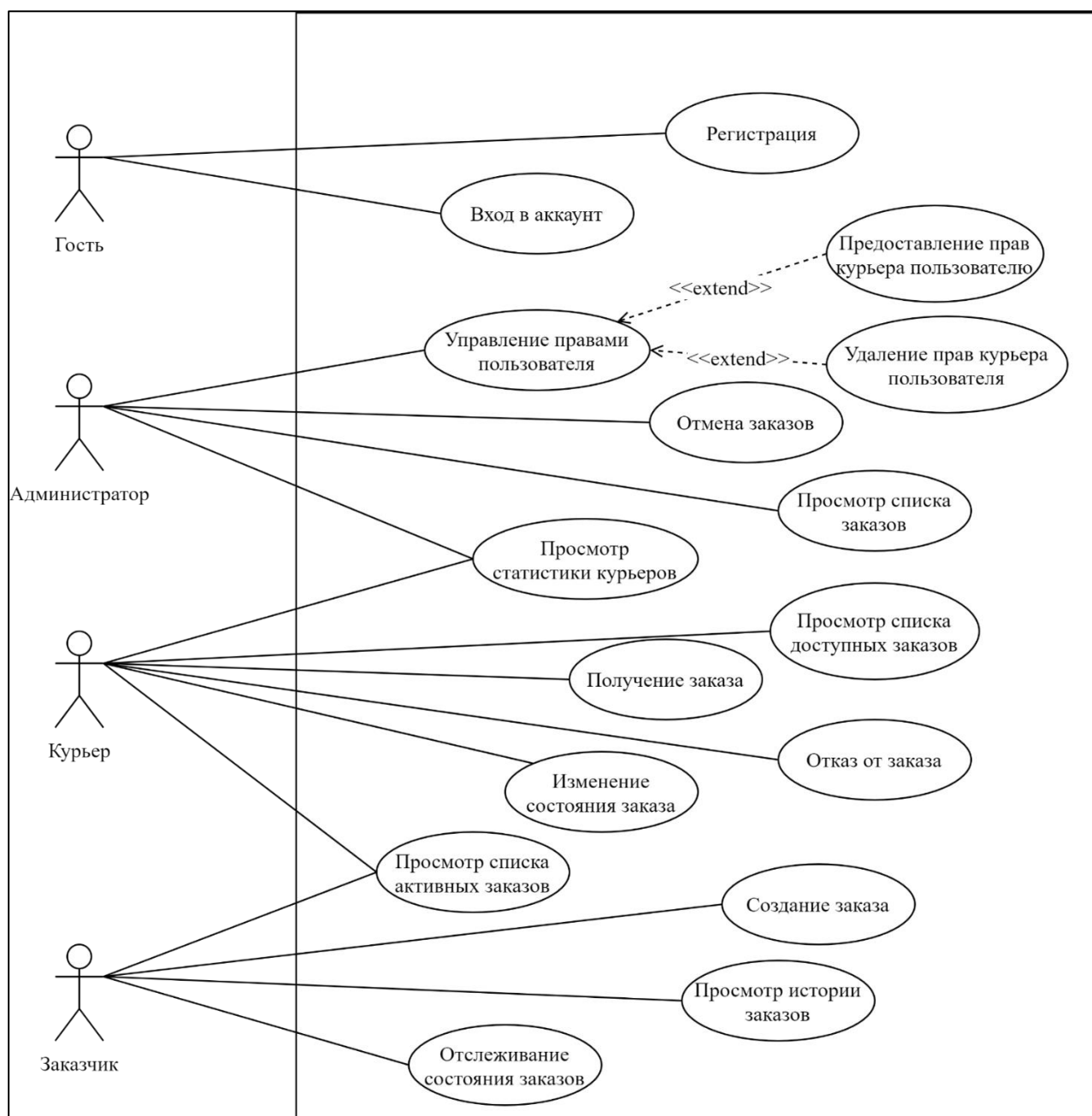


Рисунок 2.1 – Диаграмма вариантов использования

Как видно из спроектированной диаграммы вариантов использования, разрабатываемое программное средство подразумевает наличие нескольких актеров. Для администратора будет разработан веб-сайт, позволяющий реализовать соответствующие возможности для администратора. Помимо веб-сайта будет разработано мобильное приложение, которое реализует возможности курьера и заказчика. Актер гость сможет пользоваться обоими приложениями для входа.

## 2.3 База данных

В качестве системы управления базами данных была выбрана Microsoft SQL Server 2019. SQL Server – это система управления реляционными базами данных, разработанная корпорацией Microsoft. В СУБД используется язык запросов Transact-SQL, являющийся реализацией стандарта ANSI/ISO по структурированному языку

запросов SQL с расширениями. Microsoft SQL Server используется для работы как с персональными базами данных, так и с базами данных крупных предприятий.

Перед началом разработки приложения была спроектирована база данных для сервера с 7 таблицами. Список таблиц с их кратким описанием приведен в таблице 2.1. Логическая схема базы данных приведена в приложении Б.

Таблица 2.1 – Таблицы базы данных

Таблица	Описание
destination	Содержит данные о точках маршрута
orders	Содержит данные о заказах
product	Информация о товарах в заказе
ordered	Связывает таблицы Orders и Product
user	Содержит информацию о пользователях
role	Содержит роли
user_role	Содержит информацию о ролях пользователей

### 2.3.1 Таблицы

Таблица user содержит в себе всех пользователей, зарегистрированных в системе, состоит из полей, приведенных в таблице 2.2.

Таблица 2.2 – Поля таблицы user

Поле	Описание
id	Идентификатор пользователя
email	Электронная почта пользователя
password	Пароль пользователя
first_name	Имя пользователя
second_name	Фамилия пользователя
phone	Номер телефона пользователя
courier_type	Способ передвижения курьера

Поле courier\_type может содержать следующие значения:

- NotCourier – пользователь не является курьером;
- Walk – курьер передвигается пешком;
- Bicycle – курьер передвигается на велосипеде;
- Car – курьер передвигается на автомобиле.

Идентификатор пользователя генерируется автоматически при регистрации. Пароль хранится в виде хеша, основанном на алгоритме Blowfish, для повышения безопасности.

Таблица role необходима для хранения ролей пользователей и содержит поля, представленные в таблице 2.3.

Таблица 2.3 – Поля таблицы role

Поле	Описание
id	Идентификатор роли
name	Название роли

Поле name может содержать следующие значения:

- ROLE\_BASIC – роль, выдаваемая по умолчанию при регистрации, позволяет просматривать информацию об аккаунте;
- ROLE\_COURIER – роль курьера, позволяющая принимать и отказываться от заказов, а также изменять состояние заказа, и иметь доступ к статистике;
- ROLE\_ADMIN – роль администратора. Позволяет изменять данные всех основных таблиц. Принимать и отказываться от заказов администратор не может, так как не является курьером.

Таблица user\_role используется для реализации отношения многие-ко-многим между таблицами role и user и хранит информацию о ролях пользователей. Поля таблицы приведены в таблице 2.4.

Таблица 2.4 – Поля таблицы user\_role

Поле	Описание
user_id	Идентификатор пользователя
role_id	Идентификатор роли

Таблица orders необходима для хранения информации о заказах и состоит из следующих полей, представленных в таблице 2.5.

Таблица 2.5 – Поля таблицы orders

Поле	Описание
id	Идентификатор заказа
customer_id	Идентификатор заказчика
courier_id	Идентификатор курьера
info	Дополнительная информация
state	Состояние заказа
ordered_at	Время заказа
taken_at	Время получения заказа курьером
delivered_at	Время доставки
sender_destination_id	Идентификатор точки отправителя
recipient_destination_id	Идентификатор точки получателя

Поле state может принимать следующие значения:

- Ordered – состояние, принимаемое заказом при его создании;
- Delivering – состояние, устанавливаемое курьером при начале доставки заказа;
- Delivered – состояние, устанавливаемое курьером при доставке заказа;
- Canceled – состояние, устанавливаемое администратором при отмене.

Курьер также может отказаться от заказа, при этом значение поля courier\_id примет значение null, а состояние заказа изменится на Ordered.

Значение столбцов ordered\_at, taken\_at и delivered\_at устанавливаются автоматически при соответствующих событиях.

Таблица destination необходима для хранения информации о точках маршрута, которая указывается при создании заказа. Содержит поля, приведенные в таблице 2.6.

Таблица 2.6 – Поля таблицы destination

Поле	Описание
id	Идентификатор точки
name	Имя отправителя или получателя
phone	Номер телефона отправителя или получателя
address	Адрес точки маршрута
range_start	Начало интервала времени принятия/доставки
range_end	Конец интервала времени принятия/доставки

Таблица product используется для хранения информации о товарах и состоит из полей, приведенных в таблице 2.7.

Таблица 2.7 – Поля таблицы product

Поле	Описание
id	Идентификатор товара
name	Наименование товара
price	Стоимость товара
weight	Масса товара

Таблица ordered используется для связи заказа с заказанными товарами и содержит поля, приведенные в таблице 2.8

Таблица 2.8 – Поля таблицы ordered

Поле	Описание
id	Идентификатор записи
order_id	Идентификатор заказа
product_id	Идентификатор товара
amount	Количество товара

### 2.3.2 Хранимые процедуры

Хранимые процедуры используются в базе данных разрабатываемого приложения для получения статистики курьера. В приложении 4 хранимых процедуры:

- delivered\_orders\_amount\_by\_courier\_id – возвращает количество заказов доставленных курьером по его идентификатору;
- delivered\_orders\_in\_time\_by\_courier\_id – возвращает количество заказов доставленных курьером вовремя по его идентификатору;
- total\_delivered\_price\_by\_courier\_id – возвращает суммарную стоимость всех товаров, доставленных курьером;
- total\_delivered\_product\_amount\_by\_courier\_id – возвращает количество доставленных курьером товаров.

Перечисленные хранимые процедуры будут использованы для отображения статистики курьера администратору на веб-сайте, а также для отображения статистики в графическом виде курьеру в мобильном приложении.

### 2.3.3 Триггеры

В базе данных используются триггеры для таблицы orders:

- set\_timestamp\_on\_delivered – задает логику установки значения в столбец delivered\_at при изменении состояния заказа;
- set\_timestamp\_ordered\_at – задает логику установки значения в столбец ordered\_at при изменении состояния заказа.

## 2.4 Обзор инструментов разработки приложения

### 2.4.1 Интегрированная среда разработки Android Studio

Android Studio – это официальная интегрированная среда разработки, предназначенная для создания приложений под операционную систему Android. Хотя Android Studio основана на IntelliJ IDEA, но предлагает более широкий набор инструментов для создания Android-приложений, таких как:

- гибкая система сборки, основанная на Gradle;
- быстрый и многофункциональный эмулятор;
- единая среда для разработки приложений под все устройства на ОС Android;
- мощные инструменты тестирования и фреймворки;
- инструменты Lint для отслеживания проблем, связанных с производительностью, удобством использования, совместимостью версий и т.д.;
- поддержка C++ и Native Development Kit (NDK);
- встроенная поддержка Google Cloud Platform [3].

### 2.4.2 Язык программирования Kotlin и библиотеки Android Jetpack

Kotlin – это кроссплатформенный, статически-типизированный язык программирования общего назначения. Kotlin имеет интероперабельность с Java и может работать поверх Java Virtual Machine. Однако помимо Java, Kotlin может компилироваться в JavaScript и в нативный код ряда платформ (например, в Objective-C для исполнения в приложениях iOS).

Также в Kotlin есть библиотека coroutines для упрощения асинхронного программирования. В отличие от множества языков, где асинхронное выполнение основано на функциях обратного вызова, корутины позволяют выражать асинхронный код в последовательном виде, что упрощает его понимание [4].

С 2019 года Kotlin является рекомендуемым языком для разработки приложений под ОС Android. По сравнению с Java, код, написанный на языке Kotlin, является более лаконичным. Также в Kotlin есть null-безопасность, позволяющая свести к минимуму возможность возникновения исключения NullPointerException.

Помимо стандартных библиотек для разработки Android-приложений, существует Android Jetpack – набор библиотек и инструментов, созданный командой Google для упрощения разработки под Android. Библиотеки Jetpack разделены на четыре категории:

- Architecture – набор библиотек для упрощения реализации чистой архитектуры (ViewModel, Navigation, Room);



- UI – библиотеки, помогающие разрабатывать приложения с современными стилями (Animation & Transitions, Fragment, Layout, Palette);
- Foundation – библиотеки совместимости со старыми версиями Android;
- Behavior – набор библиотек, упрощающих реализацию уведомлений, загрузки, получения разрешений и т.д.

Летом 2021 года была выпущена стабильная версия библиотеки Jetpack Compose, позволяющая создавать пользовательский интерфейс в декларативном стиле. С использованием Compose отпадает необходимость в использовании xml-разметки, так как пользовательский интерфейс описывается на языке Kotlin с использованием функций, помеченных аннотацией @Composable. Эта библиотека значительно упрощает создание сложной логики отображения пользовательского интерфейса и в то же время полностью совместима со старой системой Android View: composable-разметку можно использовать внутри xml-разметки и наоборот [5].

### 2.4.3 Интегрированная среда разработки IntelliJ IDEA

IntelliJ IDEA – это интегрированная среда разработки программного обеспечения для многих языков программирования. Первая версия была разработана компанией JetBrains в 2001 году.

IntelliJ IDEA предназначена для разработки приложений на языках JVM, таких как Java, Kotlin, Scala и Groovy. Кроме того, IntelliJ IDEA Ultimate поможет в разработке веб-приложений: она предлагает эффективные встроенные инструменты, поддержку JavaScript и связанных с ним технологий, а также расширенную поддержку таких популярных фреймворков, как Spring, Spring Boot, Jakarta EE и др. Также можно расширить возможности IntelliJ IDEA при помощи плагинов и использовать ее для работы с другими языками программирования, в том числе Go, Python, SQL, Ruby и PHP.

Особенности IntelliJ IDEA:

- эргономичная среда, предоставляющая быстрый доступ ко всем функциям и инструментам, а также имеющая широкие возможности индивидуальной настройки;
- глубокий анализ кода позволяет совершать навигацию по проекту, предлагает варианты автодополнения кода, а также обнаруживает ошибки;
- наличие эффективного набора инструментов для конфигурации параметров запуска, сборки, отладки, а также применения и разработки тестов;
- интеграция с системами контроля версий, поддерживающая наиболее популярные из них, а также позволяющая выполнять соответствующие команды и разрешать конфликты при слиянии и просматривать изменения;
- наличие функции совместной разработки, позволяющей вместе работать над проектом в реальном времени [6].

### 2.4.4 Spring Framework

Spring – это Java фреймворк, предоставляющий комплексную модель программирования и конфигурации для современных Java EE приложений на любой платформе развертывания. Spring включает в множество компонентов, в том числе:

- Spring Boot необходим для упрощения создания Spring приложения, которое можно запустить;
- Spring Framework – компонент, предоставляющий поддержку внедрения зависимостей, управления транзакциями, веб-приложений, а также доступа к данным;
- Spring Data предоставляет модель доступа к данным, упрощающую доступ к реляционным и не реляционным базам данных и облачным хранилищам из приложения Spring;
- Spring Cloud предоставляет набор инструментов для реализации распределенных систем и применяется для сборки и запуска микросервисов;
- Spring Security позволяет защитить приложение при помощи поддержки аутентификации и авторизации [7].

## **2.5 Выводы по разделу**

В рамках данного раздела были описаны основные моменты проектирования приложения по осуществлению доставки заказов, схема компонентов которого представлена в приложении В. Построена диаграмма вариантов использования, спроектирована структура базы данных и выбраны технологии для разработки серверной и клиентской части приложения.

### 3 Разработка приложения

#### 3.1 Разработка серверной части приложения

Для реализации серверной части приложения будет использоваться Spring Framework. В данном разделе будет рассмотрена серверная часть приложения.

##### 3.1.1 Файловая структура

В процессе разработки классы проекта были распределены по пакетам в соответствии с их назначением, что упрощает навигацию в проекте, а также внесение изменений.

Файловая структура представлена на рисунке 3.1.

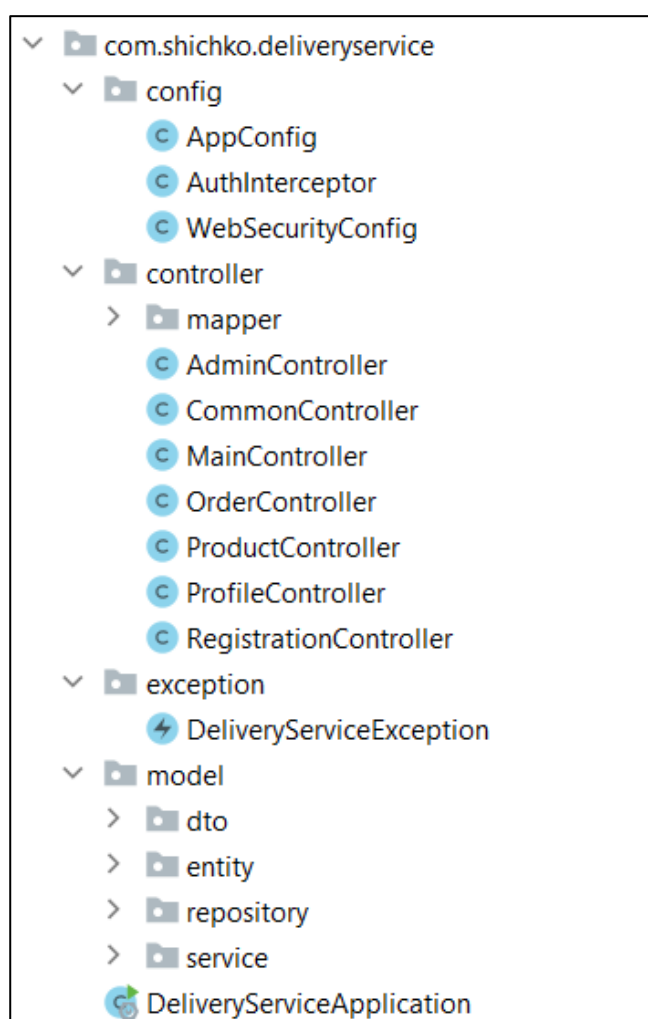


Рисунок 3.1 – Файловая структура проекта

				БГТУ 03.00.ПЗ			
	Ф.И.О.	Подпись	Дата				
Разраб.	Шичко В.С.			3 Разработка приложения	Лит.	Лист	Листов
Провер.	Нистюк О.А.					1	20
Н. контр.	Нистюк О.А.						
УТВ.	Смелов В.В.				74217084, 2022		

В пакете `config` расположены классы, предназначенные для настройки приложения. `WebSecurityConfig` содержит логику аутентификации и авторизации пользователей. В классе `AppConfig` регистрируется `AuthInterceptor` – класс-перехватчик запросов, позволяющий применять обновления ролей пользователя без необходимости повторной аутентификации.

Пакет `controller` содержит контроллеры сервера, к конечным точкам которых пользователь будет делать запросы для получения данных. Также в пакете `controller` размещен пакет `mapper`, содержащий интерфейсы с описанием логики преобразования объектов сущности в объекты передачи данных (`Data Transfer Object`) при помощи библиотеки `mapstruct`.

Пакет `model` содержит классы, относящиеся к уровню модели приложения. В пакете `dto` расположены классы передачи данных. Пакет `entity` содержит классы, представляющие данные, которые могут быть сохранены в базе данных. В пакете `service` находятся классы-сервисы, в которых содержится бизнес-логика приложения. Пакет `repository` содержит интерфейсы, описывающие запросы к базе данных.

Класс `DeliveryServiceApplication` является точкой входа приложения `Spring Boot` и содержит метод `main`.

### 3.1.2 Уровень модели приложения

В пакете `entity` расположены классы, которые соответствуют по структуре таблицам базы данных. Они представляют из себя `POJO`-классы, помеченные аннотациями библиотеки `Spring Data JPA`, которые позволяют указать информацию о том, какую сущность описывает данный класс, а также описать соответствие между полями класса и таблицы базы данных и связи между сущностями.

Аннотация `@Entity` помечает класс как сущность, а также позволяет задать имя таблицы, которой соответствует данный класс, при помощи аргумента `name`. Аннотация `@Basic` позволяет указать, что будет использоваться стандартное отображение типов базы данных в типы `Java`, а аннотация `@Column` позволяет в явном виде указать имя поля таблицы, которому соответствует данное поле класса. Аннотация `@Enumerated` позволяет указать тип, в который будет преобразовано перечисление `Java` для хранения в базе данных. Аннотации `@OneToMany`, `@ManyToOne`, `@ManyToMany` предназначены для описания отношений между сущностями, а аннотация `@JoinColumn` необходима для указания имени поля, на которое ссылается внешний ключ.

Помимо аннотаций `Spring Data JPA` в классах хранения данных также используются аннотации библиотеки `Lombok`, которые делают код класса более лаконичным при помощи автогенерации методов. Например, аннотация `@Data` позволяет сгенерировать шаблонный код геттеров, сеттеров, конструктора со всеми аргументами, конструктора без аргументов, а также методов `toString()`, `equals()` и `hashCode()`. Помимо данной аннотации, существует набор других аннотаций, позволяющих генерировать код для конкретных методов, а также задать некоторые параметры.

Код класса `Destination` приведен в листинге 3.1.

```

@EqualsAndHashCode(callSuper = true)
@Data
@Entity(name = "destination")
public class Destination extends AbstractEntity
    implements Serializable {
    @Basic
    @Column(name = "name", nullable = false, length = 50)

    private String name;
    @Basic@Column(name = "phone", nullable = false, length = 20)

    private String phone;
    @Basic@Column(name = "address", nullable = false)
    private String address;
    @Basic@Column(name = "range_start")

    private Timestamp rangeStart;
    @Basic@Column(name = "range_end")
    private Timestamp rangeEnd;
}

```

Листинг 3.1 – Класс Destination

В пакете `dto` находятся классы, предназначенные для передачи данных между клиентом и сервером. В них, в отличие от классов сущности, отсутствуют циклические зависимости, и содержат только необходимую информацию для получения или отправки клиенту. Логика трансформации объектов сущностей в объекты передачи данных и наоборот описана в интерфейсах пакета `mapper`.

Интерфейсы, расположенные в пакете `repository` описывают запросы к базе данных. Данные интерфейсы помечены аннотацией `@Repository`. Они позволяют абстрагироваться от деталей реализации доступа к базе данных. Аннотация методов `@Query` необходима для создания SQL и JPQL запросов. Пример метода для запроса к базе данных представлен в листинге 3.2.

```

@Query("select o from orders o where o.courier.id = :courierId " +
        "and not o.state = 'Delivered' and not o.state = 'Canceled'")

List<Order> getActiveOrdersByCourierId(
    @Param("courierId") long courierId
);

```

Листинг 3.2 – Метод получения списка активных заказов курьера

Пакет `service` содержит классы с основной бизнес-логикой сервисной части приложения. Они имеют ссылки на экземпляры необходимых репозиторий и прочих компонентов проекта, получаемых при помощи `Dependency Injection` и аннотации `@Autowired`. В листинге 3.3 представлен код метода сохранения пользователя при регистрации.

```

public void saveUser(UserDto dto) throws DeliveryServiceException {
    if (!dto.getPassword().equals(dto.getConfirmPassword())) {
        throw new DeliveryServiceException("Passwords not equals");
    }

    User user = mapper.dtoToEntity(dto);
    User userFromDB = userRepository.findByEmail(user.getUsername());

    if (userFromDB != null) {
        throw new DeliveryServiceException("User already exists");
    }

    user.setRoles(Collections.singleton(
        roleRepository.findFirstByName("ROLE_BASIC")
    ));

    user.setPassword(bCryptPasswordEncoder.encode(
        user.getPassword()
    ));

    userRepository.save(user);
}

```

Листинг 3.3 – Метод saveUser класса UserService

Метод принимает аргументом экземпляр класса UserDto, который передается при вызове метода в контроллере. Сначала происходит проверка на совпадение пароля и подтверждения пароля. В случае их несовпадения выбрасывается соответствующее исключение. Далее осуществляется преобразование объекта типа UserDto в объект User. Их основное отличие заключается в том, что в классе User отсутствует поле confirmPassword по причине того, что у нас нет необходимости хранить это поле в базе данных. После преобразования происходит поиск в базе данных пользователя с совпадающим email-адресом. Если таковой найден, выбрасывается исключение, так как поле email является уникальным и в приложении не может быть два пользователя с одинаковым email-адресом. Если пользователь уникален, то ему устанавливается стандартная роль, а также хешируется пароль по алгоритму BCrypt. После этого новый пользователь сохраняется в базе данных.

### 3.1.2 Уровень контроллеров приложения

В пакете controller.mapper расположены интерфейсы, помеченные аннотацией @Mapper из библиотеки mapstruct и предназначенные для преобразования объектов сущности в объекты передачи данных и обратно.

Рассмотрим интерфейс UserMapper, предназначенный для преобразования объектов типа UserDto и User. Интерфейс содержит 2 основных метода entityToDto и dtoToEntity. Во время компиляции будут сгенерированы классы, реализующие этот интерфейс, что позволяет уменьшить количество шаблонного кода, написанного вручную. При помощи аннотаций @Mapping можно задать особые правила преобразования для отдельных полей. Например, нет необходимости передавать поле password с сервера на клиент. Соответственно можно задать для этого поля особое

правило при преобразовании из entity в dto. С этой целью используется аргумент аннотации `qualifiedByName`, в который передается имя метода, осуществляющего данное преобразование. Имя указывается аргументом аннотации `@Named` соответствующего метода. Также в данном интерфейсе осуществляется преобразование коллекции заказов объекта `User` в коллекцию идентификаторов заказов объекта `UserDto` соответствующего курьера. Код интерфейса `UserMapper` представлен в листинге 3.4.

```
@Mapper(componentModel = "spring")
public interface UserMapper extends CommonMapper<User, UserDto> {

    @Mapping(source = "orders", target = "ordersId",
            qualifiedByName = "orderToOrderId")

    @Mapping(source = "password", target = "password",
            qualifiedByName = "passwordToDto")
    UserDto entityToDto(User entity);

    @Mapping(source = "ordersId", target = "orders",
            qualifiedByName = "orderIdToOrder")
    User dtoToEntity(UserDto dto);

    @Named("orderToOrderId")

    default Collection<Long> orderToOrderId(

        Collection<Order> orders
    ) {
        return orders.stream().map(AbstractEntity::getId)
            .collect(Collectors.toList());
    }

    @Named("orderIdToOrder")
    default Order orderIdToOrder(long id) {
        Order order = new Order();
        order.setId(id);
        return order;
    }

    @Named("passwordToDto")
    default String orderToOrderId(String password) {
        return null;
    }
}
```

Листинг 3.4 – Интерфейс `UserMapper`

В пакете `controller` расположены классы контроллеров, которые обрабатывают поступающие запросы к серверу.

Все контроллеры, за исключением `MainController`, являются REST-контроллерами и помечены соответствующими аннотациями. `MainController` поддерживает исключительно метод `GET` и имеет путь к корню приложения. Он предназначен для возврата `html`-страницы клиентской части веб-приложения.

Рассмотрим класс `AdminController`. Класс помечен двумя аннотациями. Аннотация `@RestController` используется для упрощения создания Restful-сервисов. `@RequestMapping` указывает базовый адрес для контроллера. Класс имеет два поля, значения которых внедряются при помощи `Dependency Injection`. Также в классе присутствуют 3 метода, помеченные аннотацией `@GetMapping`, которая указывает, что метод будет вызываться при получении запроса с http-методом GET, а также адрес. Если адрес не указывается, то используется базовый адрес контроллера. Также в строке могут содержаться шаблоны в фигурных скобках, значения которых могут быть получены в качестве аргумента метода, помеченного аннотацией `@PathVariable` с именем шаблона. В данном случае таким шаблоном является параметр `userId` для методов `confirmUser` и `revokeUser`, содержащий идентификатор пользователя. GET-запрос также может содержать дополнительные параметры, которые можно получить в качестве параметра, помеченного аннотацией `@Param`. Код класса приведен в листинге 3.5.

```
@RestController
@RequestMapping("admin")
public class AdminController {
    @Autowired
    private UserService userService;

    @Autowired
    private UserMapper mapper;

    @GetMapping
    public List<UserDto> userList() {
        return mapper.entitiesToDtos(userService.allUsers());
    }

    @GetMapping("/grant/{userId}")
    public void confirmUser(@PathVariable("userId") Long userId,
        @Param("role") String role) {
        userService.grantUserRole(userId, role);
    }

    @GetMapping("/revoke/{userId}")
    public void revokeUser(@PathVariable("userId") Long userId,
        @Param("role") String role) {
        userService.revokeUserRole(userId, role);
    }
}
```

Листинг 3.5 – Класс `AdminController`

Всего было разработано 4 REST-контроллера, каждый из которых содержит методы, обрабатывающие запросы. Контроллеры в сумме содержат 22 REST API метода, описание которых приведено в таблице 3.1.



Таблица 3.1 – Описание методов REST API

Адрес	Метод	Описание
/login	POST	Вход в аккаунт
/registration	POST	Регистрация нового пользователя
/profile	GET	Получить информацию о текущем пользователе
/profile/changetype	PUT	Изменить способ передвижения
/profile/stats	GET	Получить статистику текущего пользователя
/profile/stats/{id}	GET	Получить статистику по идентификатору пользователя
/admin	GET	Получить список всех пользователей
/admin/grant/{userId}	GET	Присвоить пользователю роль
/admin/revoke/{userId}	GET	Удалить роль пользователя
/order	GET	Получить список заказов
/order	POST	Создать заказ
/order	PUT	Изменить заказ
/order/{id}	GET	Получить заказ по идентификатору
/order/{id}	DELETE	Удалить заказ
/order/active	GET	Получить список активных заказов текущего курьера
/order/available	GET	Получить список доступных заказов
/order/created	GET	Получить список созданных заказов текущим заказчиком
/order/delivered	GET	Получить список доставленных заказов текущим курьером
/order/history	GET	Получить историю заказов текущего заказчика
/order/accept/{id}	GET	Принять заказ по идентификатору
/order/decline/{id}	GET	Отказаться от заказа по идентификатору
/order/updateState	PUT	Обновить состояние заказа

### 3.1.3 Конфигурация безопасности приложения

В пакете `config` расположены классы конфигурации приложения.

Класс `WebSecurityConfig` наследует класс `WebSecurityConfigurerAdapter` из библиотеки `Spring Security` и предназначен для настройки безопасности приложения.

В переопределенном методе `configure` при помощи паттерна строитель задаются настройки безопасности приложения: конфигурация `cors`, перечисление адресов и методов запроса, доступных для определенных ролей пользователя, а также настройка аутентификации [8]. Код метода `configure` приведен в приложении Ж.

Для определения паттернов адресов, доступных конкретной роли, используются методы `antMatchers` для указания паттерна адреса и `Http-метода` запроса, а также `hasRole` для указания соответствующей роли пользователя.

В конфигурации задается логика входа в приложения, обработчики успешного входа и входа с ошибкой, а также логика выхода из приложения.

При помощи методов `rememberMe()` и `key()` можно добавить возможность запоминания пользователя при помощи соответствующего `Cookie`. по умолчанию приложение запоминает пользователя на 2 недели.

Так как по умолчанию `Spring Security` не поддерживает изменение ролей пользователя без повторной аутентификации, а администратор может менять роли других пользователей, было принято решение реализовать перехватчик запросов `AuthInterceptor`, который позволит обновить роли пользователя. Данный перехватчик запросов добавляется в регистр перехватчиков в методе `addInterceptors` класса `AppConfig`.

## 3.2 Разработка мобильного приложения

Для реализации приложения на мобильной платформе `Android` был выбран язык программирования `Kotlin`, так как он является рекомендуемым языком для разработки `Android-приложений`. В качестве `minSdk` приложения была выбрана 26, что соответствует `Android 8.0`. Данная версия была выбрана по причине того, что это минимальная версия, необходимая для использования `API` работы с датами и временем. Согласно статистике в `Android Studio`, разрабатываемое приложение сможет запускаться на 82,7% устройств `Android`. Статистика приведена на рисунке 3.2.

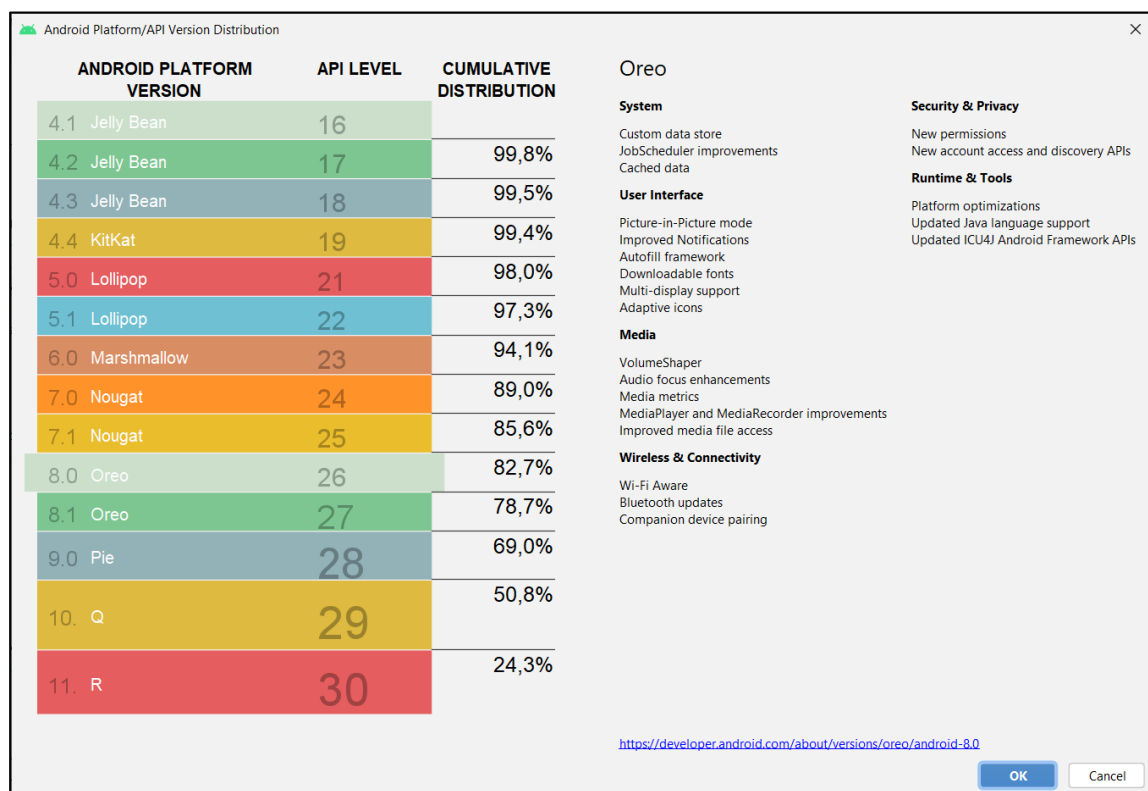


Рисунок 3.2 – Статистика количества поддерживаемых устройств по версиям `Android`

Файлы проекта Android-приложения разделены по пакетам в соответствии с их назначением. В корневом пакете расположены пакеты `model` и `ui`, в которых расположены файлы бизнес-логики и пользовательского интерфейса соответственно. Структура пакетов проекта приведена на рисунке 3.3.

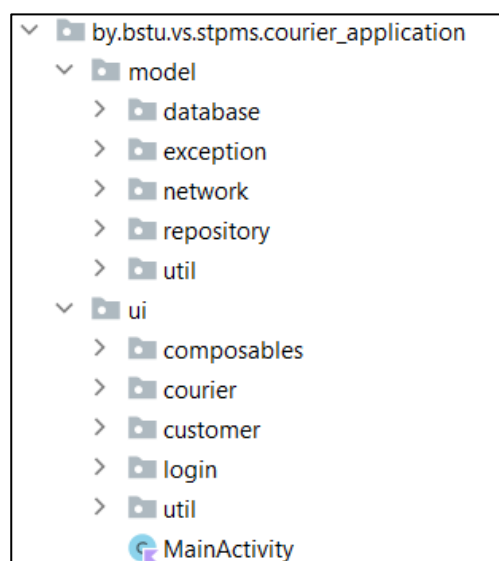


Рисунок 3.3 – Структура пакетов проекта мобильного приложения

Для разработки пользовательского интерфейса была выбрана библиотека Jetpack Compose. Данная библиотека позволяет создавать пользовательский интерфейс в декларативном стиле и без xml-разметки. Для доступа к локальной базе данных использовалась библиотека Room, а для взаимодействия с сервером – библиотека Retrofit. Room и Retrofit являются наиболее популярными решениями для доступа к данным в приложениях Android.

### 3.2.1 Разработка уровня бизнес-логики мобильного приложения

Пакет `database` содержит классы сущности, которые соответствуют таким же классам в веб-проекте, а также дополнительный класс `Change`, который соответствует одноименной таблице базы данных мобильного приложения, в которой хранятся изменения для последующей синхронизации с сервером.

Помимо классов сущностей, в этом пакете расположены интерфейсы DAO, описывающие методы доступа к базе данных и использующие аннотации библиотеки Room. Для реализации асинхронности в проекте используются корутины, поэтому методы интерфейсов DAO, помечены как `suspend`, так как запросы к базе данных являются относительно долгими операциями. Пример метода обновления заказа в базе данных приведен в листинге 3.6.

```
@Query("UPDATE orders SET state = :newState WHERE id = :id")
abstract suspend fun updateState(id: Long, newState: String)
```

Листинг 3.6 – Метод `updateState` интерфейса `OrderDao`

Генерация классов являющихся реализацией интерфейсов выполняется библиотекой Room на этапе сборки проекта.

Для работы с базой данных при помощи библиотеки Room также необходим класс, помеченный аннотацией @Database и наследующий класс RoomDatabase. В текущем проекте таким классом является класс CourierDatabase. В нем перечислены все использующиеся сущности для создания базы данных мобильного приложения, а также абстрактные поля использующихся Dao интерфейсов, для взаимодействия с базой данных.

В пакете network расположены файлы взаимодействия приложения с сетью. Для этого используется библиотека Retrofit. Логика работы с этим источником данных похожа на логику работы библиотеки Room с базой данных. Есть интерфейсы Api, в которых описаны suspend-методы запросов к серверу, а также синглтон-класс NetworkService, в котором происходит инициализация клиента, привязывание перехватчиков запросов и Api-интерфейсов для последующего использования.

Приложение имеет 3 интерфейса Api. OrderApi используется для получения информации о заказе, а также для создания новых заказов и обновления информации о существующих. UserApi содержит методы, регистрации, входа, выхода, получения информации о пользователе и в том числе его статистики. GoogleApi используется для построения маршрутов между точками. Код интерфейса GoogleApi приведен в листинге 3.7.

```
interface GoogleApi {
    @GET("maps/api/directions/json")
    suspend fun getRoute(
        @Query("origin") origin: String,
        @Query("key") key: String,
        @Query("mode") mode: String,
        @Query("destination") destination: String
    ): Response<RouteResponse>
}
```

Листинг 3.7 – Интерфейс GoogleApi

Метод getRoute возвращает параметризованный объект Response, содержащий информацию об ответе на запрос. Метод body() типа Response возвращает экземпляр типа-параметра, который получен при помощи десериализации тела ответа при помощи библиотеки Gson.

В пакете repository находятся интерфейсы преобразования объектов сущностей в объекты передачи данных по аналогии с серверным приложением. Для генерации классов, реализующих эти интерфейсы используется библиотека mapstruct.

Также в пакете repository расположены классы OrderRepository и UserRepository, содержащие основную бизнес-логику мобильного приложения и соответствующие компоненту Repository уровня Model архитектуры MVVM приложений Android.

В классе OrderRepository находится логика работы с заказами. Он имеет поля, хранящие ссылки на экземпляр CourierDatabase и OrderApi, а также набор методов работы с заказами. Исходный код класса приведен в приложении Ж.

Рассмотрим метод getOrderById, позволяющий получить заказ по его идентификатору. Метод имеет модификатор suspend, так как в нем вызываются другие

suspend-методы. Сначала происходит проверка, есть ли подключение к сети. Если подключение к сети присутствует, то выполняем запрос к серверу для получения заказа по его идентификатору. При получении объекта Response проверяем код статуса ответа. Если код соответствует ошибке, выбрасываем соответствующее исключение. Если ответ получен успешно, преобразуем объект передачи данных в объект сущность. После этого вставляем с заменой полученный заказ в базу данных и после этого возвращаем заказ, полученный по идентификатору из базы данных. В случае, если подключение к сети отсутствует, возвращаем заказ по идентификатору из локальной базы данных. Код метода приведен в листинге 3.8.

```
suspend fun getOrderById(id: Long): Result<Order> =
if (isOnline(context)) {
    try {
        val response = orderApi.getById(id)
        when (response.code()) {
            200 -> {
                val mapper = Mappers
                    .getMapper(OrderMapper::class.java)
                val order = mapper.dtoToEntity(response.body())
                Result.success(order)
                insertOrderWithReplaceToLocalDb(order)
                getOrderFromLocalDb(id)
            }
            403 -> {
                throw CourierNetworkException(
                    "Your account is not verified"
                )
            }
            else -> {
                throw CourierNetworkException("Network Troubles")
            }
        }
    } catch (e: Exception) {
        Result.error(e)
    }
} else {
    getOrderFromLocalDb(id)
}
```

Листинг 3.8 – Метод getOrderById класса OrderRepository

В классе UserRepository содержатся методы, реализующие логику взаимодействия пользователя с приложением. Рассмотрим метод tryAutoLogin. Метод вызывается параллельно с показом Splash-экрана пользователю и при успешном выполнении позволяет пропустить экран входа. Сначала происходит поиск пользователя в локальной базе данных по идентификатору, возвращенному методом getUserId. Метод getUserById возвращает идентификатор текущего пользователя, полученный из хранилища SharedPreferences. Идентификатор записывается в SharedPreferences при успешном входе пользователя в приложение и удаляется при выходе. Далее происходит проверка на наличие подключения устройства к сети Интернет. В случае, если подключение есть, и пользователь бы получен из базы данных по идентификатору,

происходит запрос к серверу на получение текущего пользователя сессии. Далее ответ на запрос передается в метод `userResponseHandler`, который вызывает лямбду, переданную вторым аргументом в случае успешного выполнения, и передает в нее объект сущности пользователя. Соответственно при успешном выполнении запроса, в `SharedPreferences` перезаписывается идентификатор пользователя, а также полученный пользователь записывается в локальную базу данных. Если же подключения к сети отсутствует, то текущий пользователь возвращается из локальной базы данных, а также ему присваиваются роли из таблицы ролей. Код метода `tryAutoLogin` приведен в листинге 3.9.

```
suspend fun tryAutoLogin(): Result<User> = try {
    val user = db.userDao.findById(getUserId())

    if (isOnline(context)) {
        if (user != null) {
            val result = userApi.currentUser()
            userResponseHandler(result) { userFromServer ->
                val id = userFromServer.id
                writeUserId(id)
                val dbUser = db.userDao.findById(id)
                if (dbUser == null) {
                    insertUserToDb(userFromServer)
                }
            }
        } else {
            Result.error(CourierNetworkException("User not found"))
        }
    } else {
        if (user != null) {
            user.roles = HashSet(
                db.userRoleDao.getUserRolesByUserId(user.id)
            )
            Result.success(user)
        } else {
            Result.error(CourierNetworkException("User not found"))
        }
    }
} catch (e: Exception) {
    Result.error(e)
}
```

Листинг 3.9 – Метод `tryAutoLogin` класса `UserRepository`

В пакете модели `util` расположены вспомогательные файлы. Здесь находятся два перехватчика ответов и запросов для получения и вставки `Cookie`. `Cookie` также хранятся в `SharedPreferences`. Помимо перехватчиков, в пакете находится класс `ConnectionListener`, который реагирует на появление и пропажу подключения устройства к сети. Когда подключение появляется, `ConnectionListener` посылает локальные изменения курьера на сервер для синхронизации. Режим оффлайн доступен исключительно для роли курьера и позволяет просматривать активные заказы, изменять их состояние и отклонять. Соответственно при синхронизации на сервер будут отправлены изменения, связанные с отказом от заказов и изменением их состояний.

### 3.2.2 Разработка интерфейса мобильного приложения

При разработке пользовательского интерфейса мобильного приложения использовалась библиотека Jetpack Compose. Данная библиотека является современным набором инструментов для создания нативного пользовательского интерфейса и позволяет упростить и ускорить разработку приложений для Android. Первая стабильная версия библиотеки Compose вышла в июле 2021 года. По состоянию на май 2022 года, актуальной версией является версия 1.1.1. Однако в проекте будет использоваться версия 1.0.5 по причине несовместимости сторонней библиотеки для построения графиков с более новыми версиями.

Пакет ui проекта мобильного приложения содержит класс MainActivity, являющийся точкой входа любого Android-приложения, пакеты login, courier, customer, которые содержат экраны и вьюмодели приложения, распределенные в соответствии с назначением. Пакет composables содержит файлы с общими composable-функциями, которые активно используются на различных экранах. В пакете util находятся файлы со вспомогательными классами и функциями, которые используются в коде пользовательского интерфейса.

Рассмотрим метод onCreate класса MainActivity. Код метода onCreate приведен в листинге 3.10.

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
        AppTheme {
            MainScreen()
        }
    }
    ConnectionListener.init(this)
}

```

Листинг 3.10 – Метод onCreate класса MainActivity

Здесь можно заметить отличие от содержимого метода onCreate классического Android-приложения, основанного на системе View: вместо вызова метода setContentView с передачей идентификатора ресурса xml-файла разметки здесь вызывается метод setContent. Одной из особенностей синтаксиса языка Kotlin является возможность выносить последний аргумент-лямбду функции за круглые скобки, а также опускать круглые скобки при вынесении единственного аргумента. Таким образом, вызов метода setContent { ... } соответствует вызову setContent({ ... }), но в первом случае код более лаконичен. Метод setContent имеет параметр типа @Composable () -> Unit, что обозначает функцию, не принимающую аргументов и не ничего возвращающую явно, а также помеченная аннотацией @Composable. Эта аннотация является основой библиотеки, ей помечаются все функции, в которых декларативно описывается пользовательский интерфейс. Такие функции называются composable-функциями и не могут быть вызваны внутри обычной функции. Метод setContent предназначен для передачи composable-функции активности в качестве разметки.

Внутри setContent вызывается composable-функция AppTheme, в которой определена цветовая тема приложения при помощи composable-функции

**MaterialTheme.** Данная функция реализует механизм **CompositionLocal**, позволяющий обращаться к теме из любого места дерева композиции под **MaterialTheme**. В нашем случае в лямбде **AppTheme** вызывается метод **MainScreen**, который содержит весь пользовательский интерфейс приложения. Соответственно тема, определенная в **AppTheme** применится ко всему приложению. Исходный код класса **MainActivity** приведен в приложении Ж.

Внутри метода **MainScreen** расположена **composable**-функция **AnimatedNavHost**, в которой описан основной граф навигации приложения. Помимо этого, данная функция позволяет описать анимации перехода между экранами. Функция **AnimatedNavHost** расположена в библиотеке **Accompanist**, которая является вспомогательной библиотекой для **Compose** и содержит некоторые **ui**-компоненты, которых нет в библиотеке **Compose**.

В листинге 3.11 представлен фрагмент кода **composable**-функции **SplashScreen**, которая описывает экран загрузки приложения.

```
@Composable
fun SplashScreen(
    authViewModel: AuthViewModel = viewModel(),
    navController: NavHostController
) {
    LaunchedEffect(true) {
        authViewModel.tryAutoLogin()
    }
    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Image(
            painter = painterResource(
                id = R.drawable.ic_express_delivery
            ),
            contentDescription = "Logo",
            modifier = Modifier.size(150.dp)
        )
    }
    LaunchedEffect(authViewModel.autoLoginUser?.status) {
        authViewModel.autoLoginUser?.let {
            when (it.status) {
                Status.SUCCESS -> {...}
                Status.ERROR -> {...}
                else -> { Log.d("SplashScreen", "Waiting...") }
            }
        }
    }
}
```

Листинг 3.11 – Функция **SplashScreen**

Функция **SplashScreen** принимает аргументом **AuthViewModel** со значением **viewModel()** по умолчанию. Функция **viewModel()** возвращает существующую или создает новую **viewModel** в соответствии с необходимым типом. Второй аргумент функции –



NavController, который необходим для осуществления навигации. В теле функции вызываются composable-функции Box, необходимая для выравнивания контента на экране, а также Image, выводящая изображение логотипа приложения на экран.

Помимо двух функций, отвечающих за построение пользовательского интерфейса, внутри SplashScreen дважды вызывается функция LaunchedEffect, которая является разновидностью side-effect функций. Эти функции необходимы из-за особенностей построения composable-интерфейса: код функций выполняется повторно лишь при обновлении состояния, к которому эти функции привязаны. Это позволяет значительно оптимизировать выполнение отрисовки пользовательского интерфейса: например, при изменении состояния кнопки не имеет смысла перерисовывать весь контент, находящийся на экране. Однако, кроме случая, когда содержимое composable-функции может не выполняться в тот момент, когда мы ожидаем выполнения, код может выполняться слишком часто из-за частых изменений состояния, к которому привязана composable-функция. Решением этих проблем является функция LaunchedEffect. Она принимает набор ключей, а также лямбду, которая будет выполняться исключительно при изменении состояния одного из ключей.

В первом вызове LaunchedEffect в качестве ключа передается значение true, а внутри лямбды расположен вызов метода вьюмодели tryAutoLogin. Из описанного выше следует, что этот метод вызовется лишь один раз при первой композиции экрана, так как значение ключа не изменяется.

Во втором вызове в качестве ключа используется состояние полученного пользователя в методе tryAutoLogin. При успешном получении пользователя, осуществляется навигация к основным экранам приложения. В противном случае осуществляется переход на экран входа в приложение.

Код класса AuthViewModel приведен в листинге 3.12.

```
class AuthViewModel: ViewModel() {

    var autoLoginUser: Result<User>? by mutableStateOf(null)
    var user: User? by mutableStateOf(null)
    var login by mutableStateOf("")
    var password by mutableStateOf("")

    private val userService = UserRepository

    fun tryAutoLogin() {
        viewModelScope.launch(Dispatchers.IO) {
            autoLoginUser = userService.tryAutoLogin()
        }
    }

    suspend fun login(): Result<User> = withContext(Dispatchers.IO) {
        userService.login(login, password)
    }
}
```

Листинг 3.12 – Класс AuthViewModel

Поле autoLoginUser вьюмодели представляет собой опциональный экземпляр типа Result<User>. Класс-обертка Result позволяет передавать статус получения значения: успешно или с ошибкой. Помимо этого, поле является делегированным от

класса `MutableState`, экземпляр которого возвращает функция `mutableStateOf`. Это означает, что у поля будут неявные геттер и сеттер, позволяющие уведомлять `Compose` об обновлении значения, но взаимодействовать с полем мы сможем как с обычной переменной указанного типа.

В функции `tryAutoLogin` осуществляется вызов одноименного метода типа `UserRepository` и присваивание его результата выполнения в поле `autoLoginUser`. Так как метод репозитория помечен как `suspend`, он может быть вызван исключительно внутри других `suspend`-функций или внутри `CoroutineScope`. В данном случае в качестве `CoroutineScope` используется `viewModelScope`. Это значит, что выполняемые в этом контексте `suspend`-функции будут отменены при разрушении вьюмодели, что позволяет избежать утечек памяти. Запуск корутины с аргументом `Dispatchers.IO` означает, что корутина будет выполняться на потоке для работы с данными, а не на основном `ui`-потоке.

Для обратной совместимости с системой `View` используется `composable`-функция `AndroidView`. Например, в библиотеке `Compose` нет готовых `ui`-компонентов для получения даты и времени. Поэтому, чтобы не реализовывать их с нуля, можно воспользоваться компонентами `Android View`.

В проекте есть `composable`-функция `DateTimePicker` для ввода пользователем даты и времени интервалов доставки при создании заказа. Код функции представлен в листинге 3.13.

```
@Composable
fun DateTimePicker(
    labelText: String,
    timestamp: Timestamp?,
    onTimeStampChange: (Timestamp?) -> Unit,
    isError: Boolean = false,
    errorMessage: String = ""
) {
    var showDateDialog by remember { mutableStateOf(false) }
    val context = LocalContext.current
    ValidatedTextField()
    if (showDateDialog) {
        DatePicker(onDateSelected = { date ->
            pickTime(context) { hour, minute ->
                val zone = TimeZone.getDefault().toZoneId()
                val zonedDateTime = ZonedDateTime.of(date,
                    LocalTime.of(hour, minute), zone)
                onTimeStampChange(Timestamp(Timestamp.from(
                    zonedDateTime.toInstant()).time))
            }
        }) { showDateDialog = false }
    }
}
```

Листинг 3.13 – Функция `DateTimePicker`

В качестве параметров функция принимает строку метки поля, поле `timestamp` состояния выбранного времени, callback `onTimeStampChange`, вызываемый при выборе времени, а также флаг `isError` и строку `errorMessage` для реализации валидации

поля. При работе со временем реализован подход Unidirectional Data Flow, который означает, что в composable-функцию передается состояние, а сама composable-функция возвращает события изменения состояния (в данном примере поля timestamp и onTimeStampChange соответственно).

В теле функции DatePicker находится переменная showDateTimeDialog, которая имеет делегированные геттер и сеттер для реализации поведения состояния переменной. Вызов функции mutableStateOf выполняется внутри лямбды-аргумента функции remember. Это означает, что состояние будет сохраняться внутри композиции. Так как данная переменная отвечает за ui-состояние: нужно ли показывать диалог, то нет необходимости выносить ее во view-модель.

Помимо переменной, в теле функции расположены вызовы функций ValidatedTextField, описывающее текстовое поле, в котором будет выводиться выбранное время и информация об ошибках валидации, а также DatePicker, которое представляет из себя диалог выбора времени и появляющийся в композиции в зависимости от значения переменной-состояния showDateTimeDialog. При выборе даты вызывается функция обратного вызова onDateSelected, в теле которой происходит запуск диалогового окна выбора времени.

В теле DatePicker присутствует вызов composable-функции CustomCalendarView, которая содержит внутри себя Android View выбора даты CalendarView. Код функции CustomCalendarView размещен в листинге 3.14.

```
@Composable
fun CustomCalendarView(onDateSelected: (LocalDate) -> Unit) {
    AndroidView(
        modifier = Modifier.wrapContentSize(),
        factory = { context ->
            CalendarView(ContextThemeWrapper(context,
                R.style.CalendarViewCustom))
        },
        update = { view ->
            val zone = TimeZone.getDefault().toZoneId()
            view.minDate = LocalDateTime.now(zone).atZone(zone)
                .toInstant().toEpochMilli()
            view.maxDate = LocalDateTime.now(zone).plusDays(6)
                .atZone(zone).toInstant().toEpochMilli()
            view.setOnDateChangeListener { _, year, month, dayOfMonth ->
                onDateSelected(
                    LocalDate
                        .now()
                        .withMonth(month + 1)
                        .withYear(year)
                        .withDayOfMonth(dayOfMonth)
                )
            }
        }
    )
}
```

Листинг 3.14 – Функция CustomCalendarView

Функция `AndroidView` принимает три аргумента: `Modifier` – модификатор `composable`-функции, `factory` – лямбда, которая должна вернуть экземпляр `View`, а также `update` – функция обратного вызова, которая выполняется после создания `View`. Соответственно в `factory` возвращаем экземпляр `CalendarView`, а в `update` настраиваем интервал выбора даты и слушатель выбора даты, который вызовет функцию обратного вызова `onDateSelected`.

Таким образом, в дерево `Compose` был вставлен стандартный `ui`-компонент `Android`.

### 3.3 Синхронизация с сервером

Так как мобильное приложение подразумевает возможность курьеру изменять состояния заказов без подключения к сети, необходимо реализовать механизм обновления данных на сервере при подключении устройства к сети. Алгоритм изменения состояния заказа с последующей синхронизацией обновленных данных с сервером представлен на рисунке 3.4, а также в приложении Г.

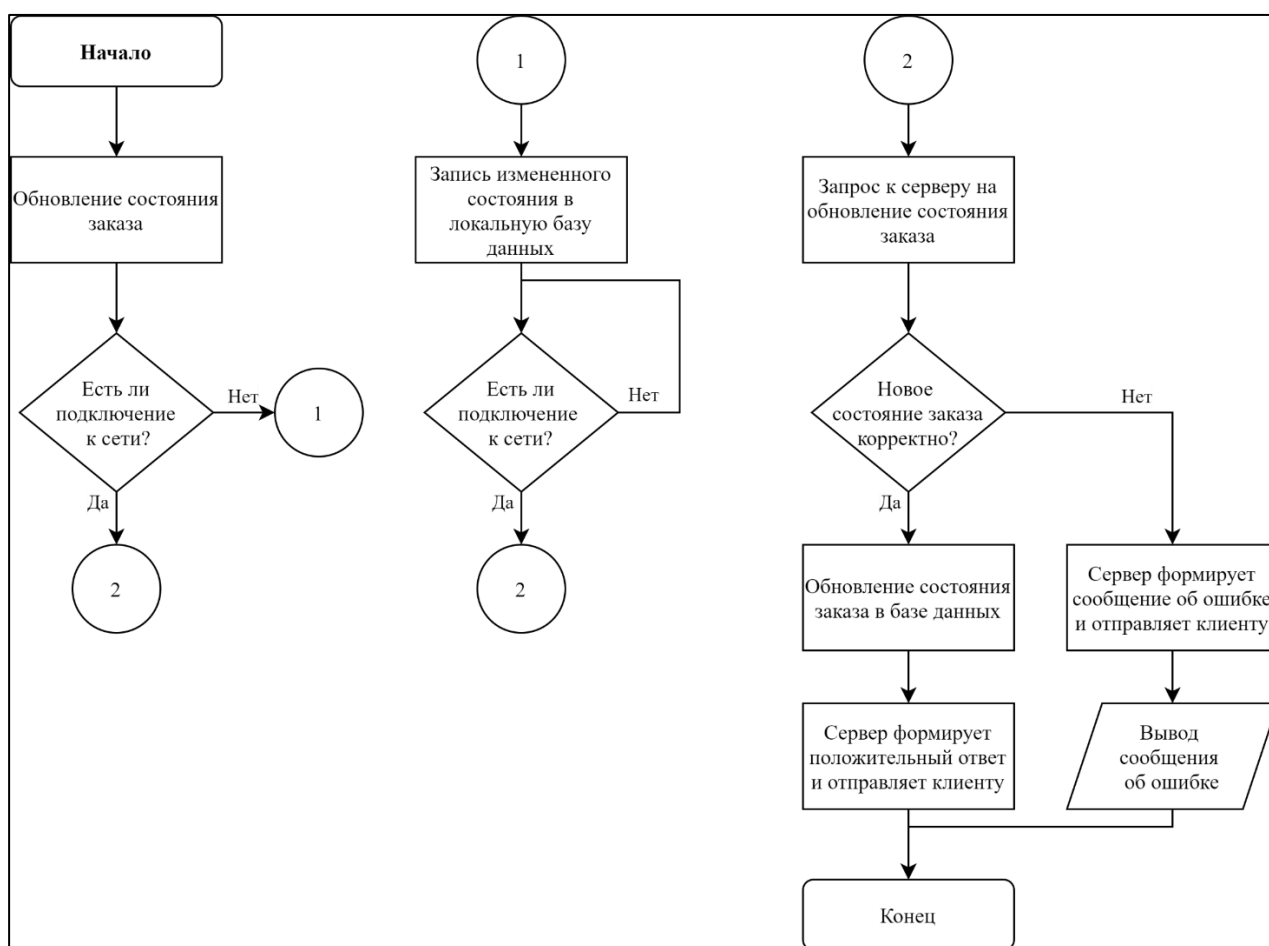


Рисунок 3.4 – Блок схема изменения состояния заказа

Для синхронизации с сервером в базе данных была создана дополнительная таблица `changes`, в которую при помощи триггеров записываются изменения состояний заказов. При подключении к сети интернет, модифицированные заказы, полученные из этой таблицы, отправляются на сервер для синхронизации. В случае, если заказ был отменен администратором, обновленные данные заказа не применяются.

### 3.4 Разработка веб-клиента

Для роли администратора приложения предусмотрено наличие веб-сайта. Веб-сайт создан с применением подхода Single Page Application и фреймворка Vue.js.

Веб-сайт состоит из 4 страниц: страница входа, регистрации, списка пользователей и списка заказов. Благодаря функциональным возможностям приложения, администратор может подтверждать аккаунты курьеров, отменять подтверждение, а также отменять заказы. Структура файлов приложения приведена на рисунке 3.5.

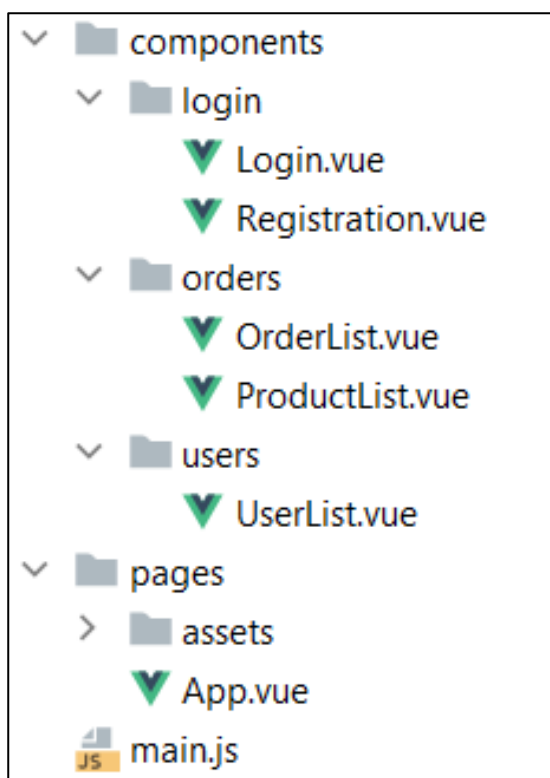


Рисунок 3.5 – Структура файлов веб-сайта

В файле main.js осуществляется основная настройка приложения: подключение библиотек Vue, настройка навигации между компонентами веб-сайта, задание темы приложения.

Файлы с расширением vue содержат разметку и логику компонентов приложения. Они имеют 3 основных тега: template, script и style. Внутри тега template описывается пользовательский интерфейс как обычными тегами html, так и специальными тегами библиотеки Vue, например, v-container, v-card, v-text-field и т.д. В разметке страницы можно обращаться к данным для отображения, а также вызывать методы в ответ на события, таких как нажатие на кнопку. Тег script предназначен для описания данных и методов JavaScript, которые будут использованы компонентом. Внутри тега script происходит экспорт объекта при помощи export default. Объект содержит поле data, представляющий из себя лямбда-функцию, возвращающую объект с полями, которые хранят необходимые данные, а также поле methods, хранящее в качестве значения объект, внутри которого расположены все необходимые методы. Внутри тега style описывается стиль компонента при помощи CSS [9].

В листинге 3.15 представлены методы файла Registration.vue.

```
methods: {
  signup() {
    this.$refs.form.validate()

    this.$http.post('/registration', {
      firstName: this.firstName,
      secondName: this.secondName,
      email: this.email,
      phone: this.phone,
      password: this.password,
      confirmPassword: this.confirmPassword
    }).then(response => {
      if (response.ok) {
        this.snackbarSuccess = true
      }
    }).catch(error => {
      this.failMessage = 'Failed To registrate. Invalid data'
      this.snackbarFail = true
    })
  },
  successClicked() {
    this.snackbarSuccess = false;
    this.$router.push('login')
  }
},
```

Листинг 3.15 – Методы файла Registration.vue

При нажатии на кнопку REGISTRATE на форме регистрации происходит вызов метода `signup`. Выполняется валидация формы, а затем POST-запрос на сервер с данными, необходимыми для регистрации. После получения ответа от сервера, на экране отображается `snackbar` с сообщением, зависящим от того, успешно ли выполнялся запрос. На `snackbar`'е успешной регистрации присутствует кнопка GO BACK, при нажатии на которую вызывается метод `successClicked` и осуществляется навигация на экран входа в приложение.

### 3.5 Выводы по разделу

В соответствии со спроектированной ранее архитектурой, разработаны:

- Серверная часть проекта, взаимодействующая с базой данных приложения;
- Клиентская часть (мобильное приложение для устройств, работающих под управлением операционной системы Android, а также веб-сайт).

Были подробно описаны используемые компоненты, необходимые для работы приложения, а также особенности их реализации.

Итогом выполнения разработки стал программный продукт для осуществления доставки заказов. Пользовательский интерфейс разработанного мобильного приложения приведен в приложении Д.

## 4 Тестирование приложения

Важной частью при разработке приложения является его тестирование с целью проверки его работоспособности при различных ситуациях. Будет рассмотрено два варианта развития событий: все пункты проходят успешно (позитивное тестирование), все пункты проваливаются (негативное тестирование).

В данном разделе будут протестированы несколько сценариев использования с применением как позитивного, так и негативного тестирования.

### 4.1 Позитивное тестирование

Позитивное тестирование – это один из видов тестирования, который позволяет проверить систему на корректное поведение. В ходе такого тестирования мы можем узнать, что система правильно функционирует.

Рассмотрим сценарий регистрации нового пользователя с использованием корректных данных (рисунок 4.1).

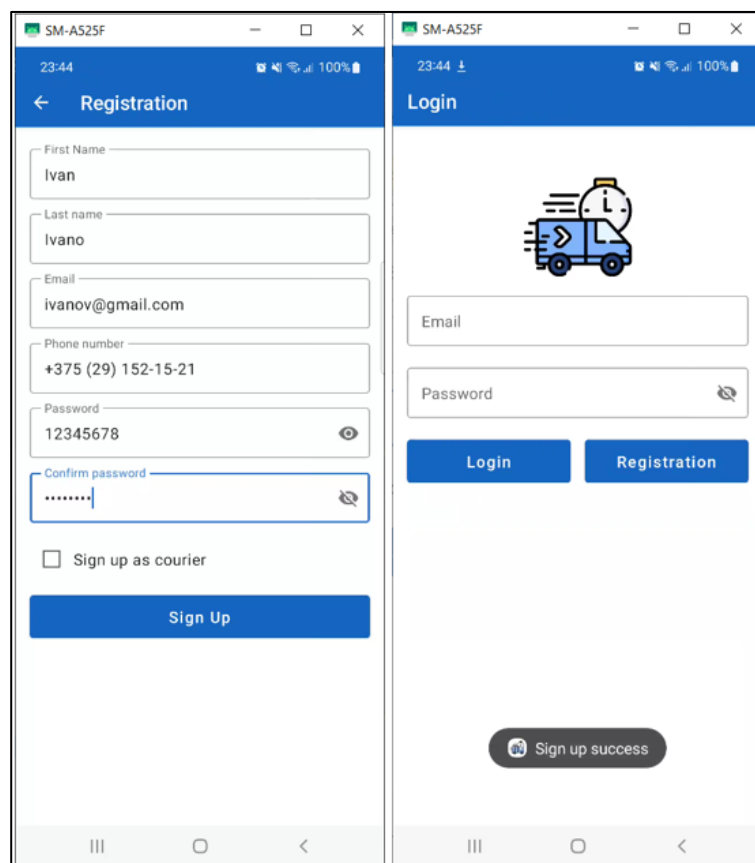


Рисунок 4.1 – Результат успешной регистрации нового пользователя

				БГТУ 04.00.ПЗ									
	Ф.И.О.	Подпись	Дата										
Разраб.	Шичко В.С.			4 Тестирование приложения			Лит.		Лист		Листов		
Провер.	Нистюк О.А.									1	5		
Н. контр.	Нистюк О.А.												
Утв.	Смелов В.В.						74217084, 2022						

При вводе корректных данных у формы не возникает ошибок, а при нажатии на кнопку SIGN UP происходит переход на форму логина с сообщением об успешной регистрации.

Также в веб-приложении можно увидеть только что зарегистрировавшегося пользователя с ролью по умолчанию ROLE\_BASIC (рисунок 4.2).

ID	First Name	Second Name	Phone	Email	Roles	GRANT COURIER	REVOKE COURIER	GRANT ADMIN	REVOKE ADMIN
1	Vladislav	Shichko	+375 (29) 772-91-44	vandl3511@gmail.com	ROLE_BASIC, ROLE_ADMIN				
2	courier	courier	+375 (29) 123-45-67	user@gmail.com	ROLE_BASIC				
4	New	Courier	+375 (29) 123-45-67	courier@gmail.com	ROLE_COURIER, ROLE_BASIC				
5	courier	new	+375 (29) 567-40-92	ewcorier@gmail.com	ROLE_BASIC				
7	Ivan	ivanc	+375 (29) 152-15-21	ivanov@gmail.com	ROLE_BASIC				

Рисунок 4.2 – Результат успешной регистрации нового пользователя

Рассмотрим еще один позитивный сценарий: курьер изменяет статус заказа без интернет-соединения, в это время администратор отменяет заказ, и при установке курьером соединения статус заказа в приложении администратора остается отмененным.

Пусть курьер №2 принял заказ №28 (рисунок 4.3).

ID	Amount	Name	Weight	Price	DETAILS	Status	SET CANCELLED	SET ORDERED
20	17	1	Вещь	0	0	Ordered		
21	18	1	коробка	10	0.5	Delivered		
27	24	1	Предмет заказа	15	20	Ordered		
25	2	Шляпа	1	2				
28	26	1	Вещь	10	10	Ordered		
29	27	1	Предмет	6	1	Delivered		
30	28	1	123	0	0	Ordered		

Рисунок 4.3 – Курьер №2 принял заказ №28



Далее он отключает интернет на устройстве, и изменяет состояние заказа №2 на «Доставляется» (рисунок 4.4).

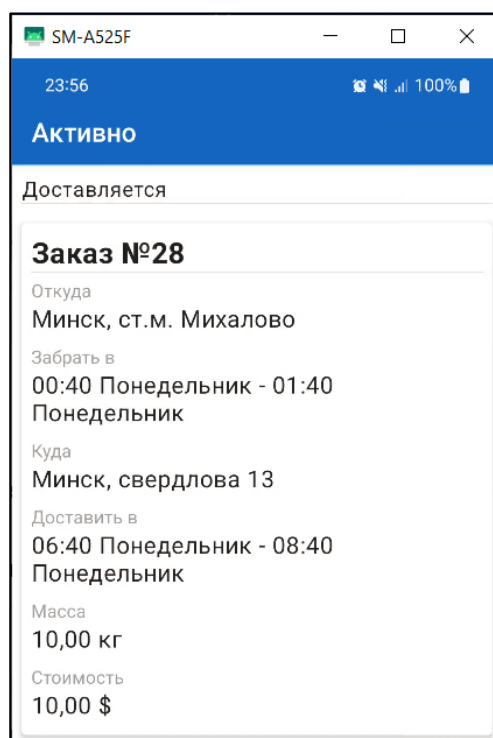


Рисунок 4.4 – Курьер обновил состояние заказа без подключения к сети

Далее администратор отменяет заказ, установив соответствующее состояние заказа. После этого курьер подключается к сети, в процессе чего происходит синхронизация обновленных данных с сервером, но состояние заказа остается отмененным (рисунок 4.5).

Delivery App

→ → https://boiling-scrubland-064.herokuapp.com/#/orders

🏠 Главная 📺 Видео (YO "БГТУ")... 📺 YouTube 📍 Maps 📧 Gmail

Delivery App

Users Orders Vladislav Shichko

20	<div></div>					DETAILS	+375 (29) 772-91-44	Vladislav Shichko	0	Ordered	SET CANCELLED	SET ORDERED	
	17	1	вещь	0	0								
21	<div></div>					DETAILS	+375 (29) 772-91-44	Vladislav Shichko	2	0.5	Delivered	SET CANCELLED	SET ORDERED
	ID	Amount	Name	Weight	Price								
	18	1	коробка	10	0.5								
27	<div></div>					DETAILS	+375 (29) 772-91-44	Vladislav Shichko	24	Ordered	SET CANCELLED	SET ORDERED	
	ID	Amount	Name	Weight	Price								
	24	1	Предмет заказа	15	20								
	25	2	Шляпа	1	2								
28	<div></div>					DETAILS	+375 (29) 123-45-67	New Courier	10	Canceled	SET CANCELLED	SET ORDERED	
	ID	Amount	Name	Weight	Price								
	26	1	Вещь	10	10								
29	<div></div>					DETAILS	+375 (29) 772-91-44	Vladislav Shichko	2	1	Delivered	SET CANCELLED	SET ORDERED
	ID	Amount	Name	Weight	Price								
	27	1	Предмет	6	1								
30	<div></div>					DETAILS	+375 (29) 772-91-44	Vladislav Shichko	0	Ordered	SET CANCELLED	SET ORDERED	
	ID	Amount	Name	Weight	Price								
	28	1	123	0	0								

Доставить аккуратно

Rows per page: 10 1-10 of 11

Рисунок 4.5 – Состояние заказа осталось отмененным

В ходе позитивного тестирования были проработаны 2 позитивных сценария. Проверка выполнения сценариев показала, что приложение выполняет функции корректно.

## 4.2 Негативное тестирование

Негативное тестирование – это один из видов тестирования, который позволяет проверить систему на некорректное поведение. В ходе такого тестирования мы можем узнать, что система справится с непредвиденными ситуациями.

Сценарий: уже зарегистрированный ранее пользователь пытается произвести процедуру регистрации.

При попытке регистрации пользователя с электронной почтой зарегистрированного пользователя, возникает ошибка (рисунок 4.6).

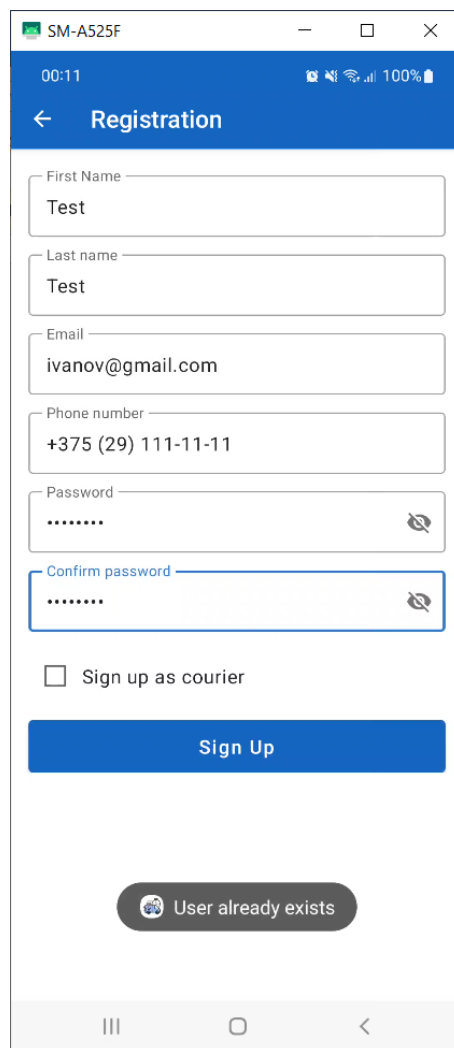


Рисунок 4.6 – Сообщение об ошибке при попытке регистрации второго аккаунта на электронную почту

Также рассмотрим еще один негативный сценарий: пользователь без роли курьера (не подтвержденный) заходит в мобильное приложение и пытается принять заказ.

При входе в мобильное приложение с использованием аккаунта без соответствующей роли список доступных заказов не отображается, а вместо него появляется соответствующее сообщение об ошибке (рисунок 4.7).

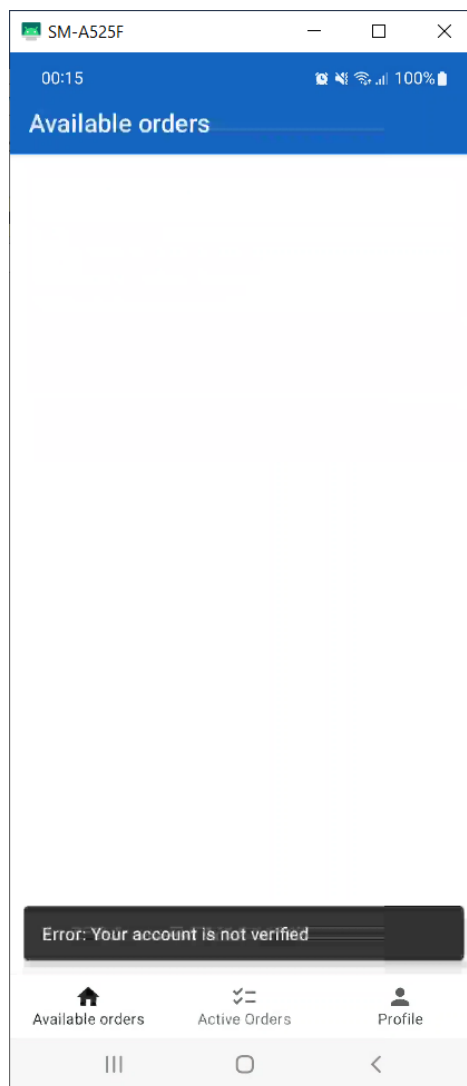


Рисунок 4.7 – Сообщение об ошибке при попытке просмотреть доступные заказы с неподтвержденного аккаунта курьера

В ходе негативного тестирования были проработаны 2 негативных сценария. Проверка выполнения сценариев показала, что приложение корректно обрабатывает нестандартные ситуации.

### 4.3 Выводы по разделу

В данном разделе предоставлены результаты тестирования приложения, разработанного для организации процесса доставки заказов.

Использовались позитивное и негативное тестирования, были осуществлены проверки на ввод некорректных и неверных данных.

Полученные результаты при ручном тестировании показали, что программное средство работает правильно и в соответствии с требованиями.

## 5 Анализ информационной безопасности приложения

Информационная безопасность системы – это свойство информационной системы или реализуемого в ней процесса, характеризующее способность обеспечить необходимый уровень своей защиты [10].

Разработанное приложение состоит из следующих компонентов: веб-клиент, мобильный клиент, мобильная база данных, сервер, база данных сервера. Каждый из компонентов приложения может иметь уязвимости, которые необходимо ликвидировать.

### 5.1 Возможные угрозы безопасности приложения

К уязвимостям серверной части можно отнести отсутствие какого-либо метода авторизации, у злоумышленников появляется доступ к данным, которые хранятся в базе данных через общедоступное API.

Потенциальной угрозой также может являться перехват трафика по незащищенному соединению.

Со стороны мобильного приложения существует угроза получения доступа к данным, хранящимся в базе данных, а также получения исходного кода путем декомпиляции байткода dex-файлов собранного apk приложения.

### 5.2 Реализованные методы защиты приложения

#### 5.2.1 Использование протокола HTTPS

Для защиты передаваемых данных в сети между клиентом и сервером, в приложении используется протокол HTTPS. Благодаря использованию протокола TLS исключается возможность получения передаваемого трафика третьими лицами. TLS (Transport Layer Security) – это протокол транспортного уровня, который обеспечивает взаимную аутентификацию сторон протокола, конфиденциальность и контроль целостности данных, передаваемых между сторонами на транспортном коммуникационном уровне. TLS использует асимметричное шифрование для аутентификации и симметричное шифрование для конфиденциальности [11].

#### 5.2.2 Хеширование пароля

Хеширование паролей учетных записей, хранящихся в базе данных используется для проверки подлинности пароля при аутентификации, а также для исключения возможности получения исходного пароля пользователя из базы данных и последующего взлома его учетной записи.

				БГТУ 05.00.ПЗ			
	Ф.И.О.	Подпись	Дата				
Разраб.	Шичко В.С.			5 Анализ информационной безопасности приложения	Лит.	Лист	Листов
Провер.	Нистюк О.А.					1	3
					74217084, 2022		
Н. контр.	Нистюк О.А.						
Утв.	Смелов В.В.						

Для хеширования пароля используется алгоритм BCrypt, основанный на криптографическом алгоритме Blowfish, который реализует блочное симметричное шифрование с переменной длиной ключа. Blowfish зарекомендовал себя как надежный алгоритм и используется для хеширования паролей, защиты электронной почты и файлов, в линиях связи, а также для обеспечения безопасности в протоколах сетевого и транспортного уровня.

### 5.2.3 Механизм аутентификации

Вход в приложение с использованием логина и пароля позволяет опознать пользователя. Пароль позволяет защитить данные от несанкционированного доступа к приложению.

### 5.2.4 Методы защиты мобильного приложения

Все данные приложения хранятся во внутреннем хранилище, что исключает возможность доступа к ним пользователя, а также других приложений. Доступ к внутреннему хранилищу доступен только при наличии ROOT-прав у пользователя смартфона, однако приложение хранит cookie, необходимые для входа в приложение, в файле SharedPreferences, а также мобильную базу данных, в которой находятся данные о заказах, доставляемых курьером и необходимы для работы приложения без подключения к сети интернет.

Также в мобильном приложении используется инструмент ProGuard, который необходим для оптимизации сборки релизного приложения, а также для обфускации исходного кода. Настройка работы ProGuard описывается в файле проекта proguard-rules.pro. Содержимое файла приведено в листинге 5.1.

```
-keep class by.bstu.vs.stpms.courier_application.model.network.dto.**
    { *; }
-keep class by.bstu.vs.stpms.courier_application.model
    .repository.mapper.** { *; }
-keep class * extends androidx.room.RoomDatabase
-keep @androidx.room.Entity class *

-keep class com.google.android.gms.maps.** { *; }
-keep interface com.google.android.gms.maps.** { *; }
```

Листинг 5.1 – Настройки в файле proguard-rules.pro

Данные настройки необходимы для сохранения исходного содержимого классов, используемых в библиотеках, связанных с кодогенерацией, таких как mapstruct, room и т.д. Для остальных классов ProGuard заменяет имена классов, методов, полей и переменных на набор букв и цифр. Помимо обфускации, ProGuard позволяет уменьшить размер исходных файлов, что в случае разработанного мобильного приложения снизило размер собранного арк-файла с 12 мегабайт до 3. Уменьшение размеров исходных файлов проекта производится путем удаления неиспользуемых конструкций, заменой вызова функции на ее код, в случае, если она вызывается из одного места, слияния родительского и дочернего класса, если родительский класс

имеет лишь одного наследника. Помимо оптимизаций файлов исходного кода проекта, Proguard также позволяет оптимизировать ресурсы путем удаления неиспользуемых ресурсов в соответствии с конфигурацией проекта.

Пример байт-кода одного из классов приведен на листинге 5.2.

```
.class public abstract Lc0/e;
.super Ljava/lang/Object;
.source ""
# annotations
.annotation system Ldalvik/annotation/Signature;
    value = {
        "<T:",
        "Lc0/e<",
        "TT;>;>",
        "Ljava/lang/Object;"
    }
.end annotation
# instance fields
.field public final a:Lr1/a;
.field public final b:J
.field public final c:Lr1/p;
.field public final d:Lw1/o;
.field public final e:Lc0/e0;
.field public f:J
.field public g:Lr1/a;
```

Листинг 5.2 – Обфусцированный байт-код класса проекта

Как можно видеть из приведенного выше листинга, имя класса, а также его полей были заменены на случайные символы, что значительно усложняет копирование бизнес-логики проекта при декомпиляции файлов из арк.

### 5.3 Выводы по разделу

В данном разделе были рассмотрены потенциальные угрозы информационной безопасности разработанного программного средства, а также описаны реализованные методы защиты безопасности. К ним можно отнести использование защищенного протокола HTTPS, хеширование паролей для хранения в базе данных, аутентификацию пользователей при использовании приложения, а также обфускацию исходного кода мобильного приложения.

## 6 Руководство пользователя

### 6.1 Мобильное приложение

Мобильное приложение поддерживает 2 языка локализации: английский и русский. Язык приложения выбирается в соответствии с системными настройками устройства. Английский язык используется по умолчанию, если язык системы не поддерживается.

При первом входе в приложение открывается экран входа в аккаунт. Здесь можно войти в приложение, введя логин и пароль, и нажав кнопку «Вход». Если логин и пароль верны, то осуществится вход в приложение на экран доступных заказов. Если нет, то появится соответствующее сообщение. Экран входа в аккаунт изображен на рисунке 6.1.

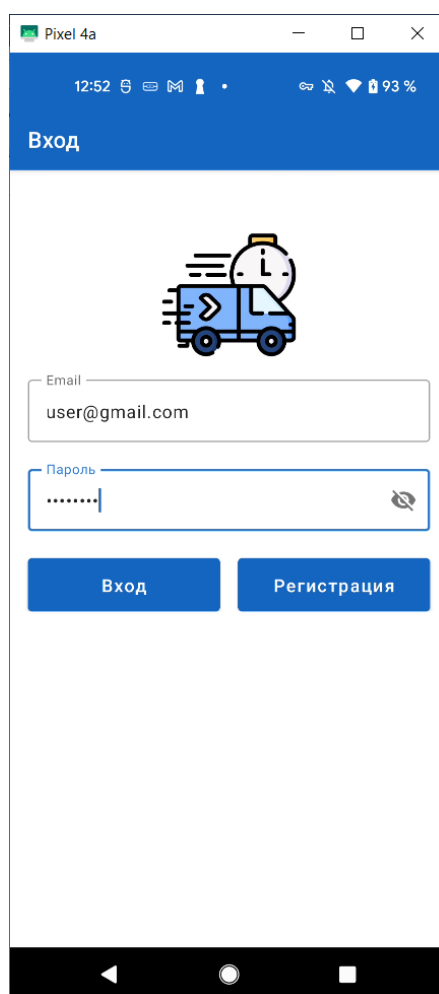


Рисунок 6.1 – Экран входа в аккаунт

				БГТУ 06.00.ПЗ									
	Ф.И.О.	Подпись	Дата										
Разраб.	Шичко В.С.			6 Руководство пользователя			Лит.		Лист		Листов		
Провер.	Нистюк О.А.									1	12		
Н. контр.	Нистюк О.А.						74217084, 2022						
Утв.	Смелов В.В.												

При нажатии на кнопку «Регистрация» произойдет переход на экран регистрации. На экране регистрации при вводе корректных данных и нажатии на кнопку «Зарегистрироваться» произойдет регистрация с выводом соответствующего сообщения и возвратом на экран входа в аккаунт. При вводе некорректных данных появится сообщение об ошибке. Экран регистрации позволяет зарегистрироваться с ролью курьера при отметке соответствующего флага. Однако зарегистрировавшийся пользователь будет неподтвержденным курьером, пока его аккаунт не подтвердит администратор. Экран регистрации изображен на рисунке 6.2.

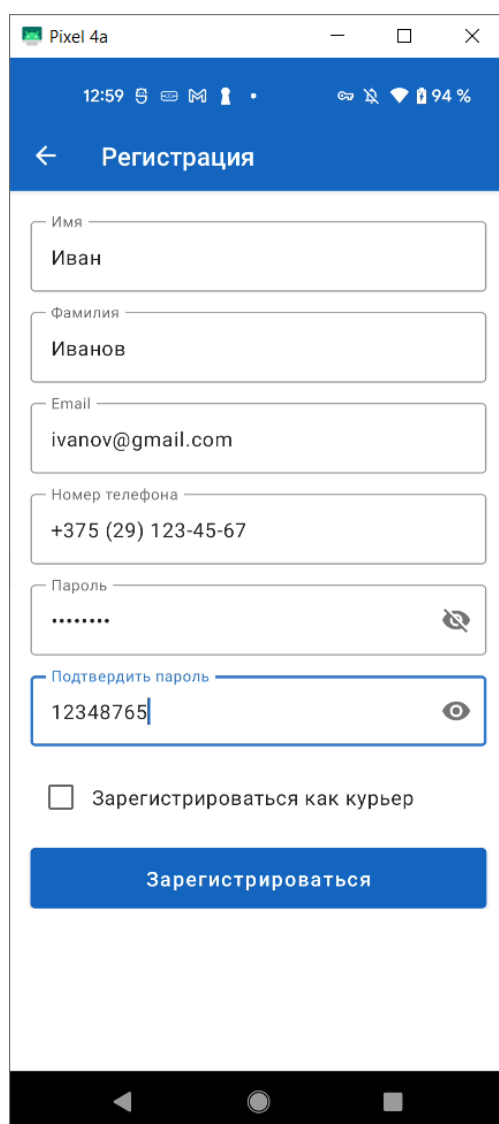


Рисунок 6.2 – Экран регистрации

При успешном входе в аккаунт или при запуске приложения (если до этого уже осуществлялся вход в аккаунт) без роли курьера откроется экран созданных заказов. На экране располагается список созданных заказов, которые не истекли по сроку, не были отменены администратором, а также не были успешно доставлены. Созданные заказы группируются в зависимости от их состояния. Созданные, но еще не подтвержденные заказы находятся в группе «Заказано». Заказы, которые курьер принял, но еще не начал доставку, находятся в группе «Подтвержден» и содержат информацию о курьере. Доставляемые заказы находятся в соответствующей группе заказов.



Остальные заказы (доставленные, отмененные, истекшие), находятся на экране истории заказов. В правом нижнем углу располагается кнопка с иконкой «+», позволяющая создать заказ. Пользовательский интерфейс экрана созданных заказов изображен на рисунке 6.3.

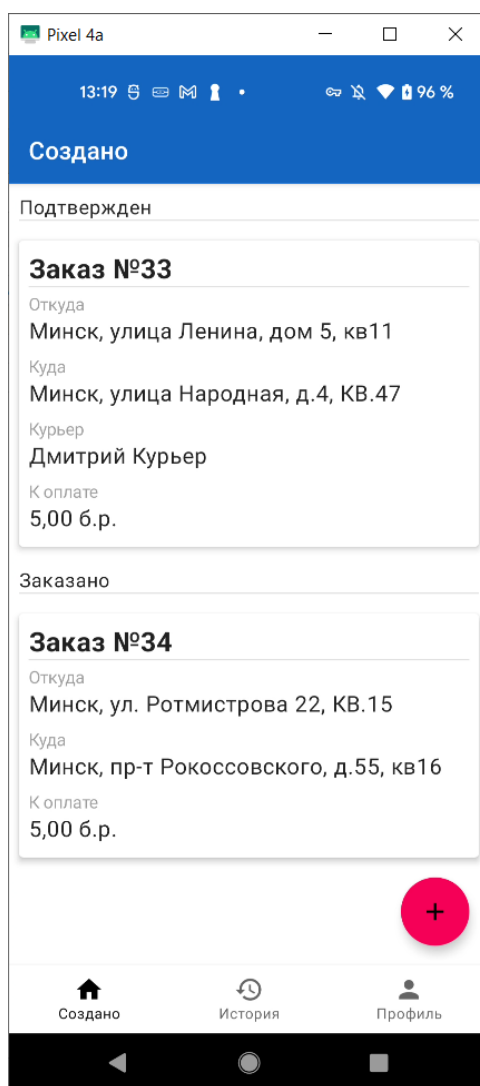


Рисунок 6.3 – Экран созданных заказов

При нажатии на кнопку «+» происходит переход на экран создания заказа. Экран создания заказа состоит из нескольких вкладок, что обусловлено большим количеством информации, которую необходимо ввести. На первой вкладке пользователь может добавить предметы в создаваемый заказ, указать их название, количество, массу и примерную стоимость. Поля формы добавления предмета в заказ имеют валидацию: поле «Кол-во» позволяет ввести исключительно целое число, большее нуля, поля «Цена» и «Масса» позволяют вводить валидные числа с двумя знаками после запятой. При нажатии кнопки «Добавить» новый предмет добавляется в список предметов заказа, расположенный над формой создания предмета. При нажатии на кнопку «Сброс» осуществляется сброс формы создания до значений по умолчанию. Вкладка добавления предметов изображена на рисунке 6.4.

Pixel 4a

13:14 95 %

← Создать заказ

**Предмет #1** ✕

Имя  
Монитор

Кол-во 1 Цена 100,00 Масса 5,00

**Новый предмет**

Имя

Кол-во 1 Цена 0,00 Масса 0,00

Сброс Добавить

Рисунок 6.4 – Вкладка добавления предметов в заказ

Далее располагаются вкладки информации отправителя, получателя, дополнительной информации, и итоговой информации. Навигация между вкладками может осуществляться как жестом свайпа влево-вправо, так и при помощи кнопок навигации внизу экрана.

На вкладке информации об отправителе вводится адрес, имя, номер телефона, а также интервал времени для доставки. Вкладка информации отправителя изображена на рисунке 6.5. Вкладка информации о получателе имеет аналогичный набор полей ввода. Все поля форм информации об отправителе и получателе имеют валидацию, которая не позволяет пользователю оставить обязательные поля пустыми, а также задать некорректный интервал времени доставки. В противном случае будут выведены сообщения об ошибках и создание заказа будет невозможным. На вкладке дополнительной информации заказчик может ввести дополнительную информацию, которая будет полезна курьеру при доставке, или оставить поле пустым. На вкладке итого отображается краткая информация о заказе: количество предметов, суммарная масса, откуда и куда будет доставлен заказ, а также сумма к оплате доставки, которая рассчитывается в зависимости от суммарной массы заказа.

Рисунок 6.5 – Вкладка информации отправителя

Если введенные данные были корректны, то при нажатии кнопки в верхнем меню на последней вкладке заказ успешно создается и произойдет переход на экран созданных заказов. При нажатии на заказ в списке созданных заказов осуществляется переход на экран полной информации о заказе. Здесь размещена таблица списка предметов в заказе, суммарная, масса, стоимость к оплате заказа, информация отправителя и получателя. Если заказ был принят курьером, то помимо информации о заказе, на этом экране также будет отображена информация о курьере: его имя и номер мобильного телефона. Также будет присутствовать кнопка «Позвонить», при нажатии на которую откроется приложение звонков с введенным номером телефона курьера, что позволит позвонить ему.

На экране истории заказов располагается список доставленных, отмененных и просроченных заказов с информацией о их состоянии. Заказ считается просроченным если ни один курьер не принял его до истечения интервала времени, в который необходимо было забрать заказ.

При нажатии на заказ в списке истории заказов, также откроется экран полной информации о заказе, содержащий аналогичную информацию, что и при открытии экрана со списка созданных заказов.

Экран истории заказов изображен на рисунке 6.6.

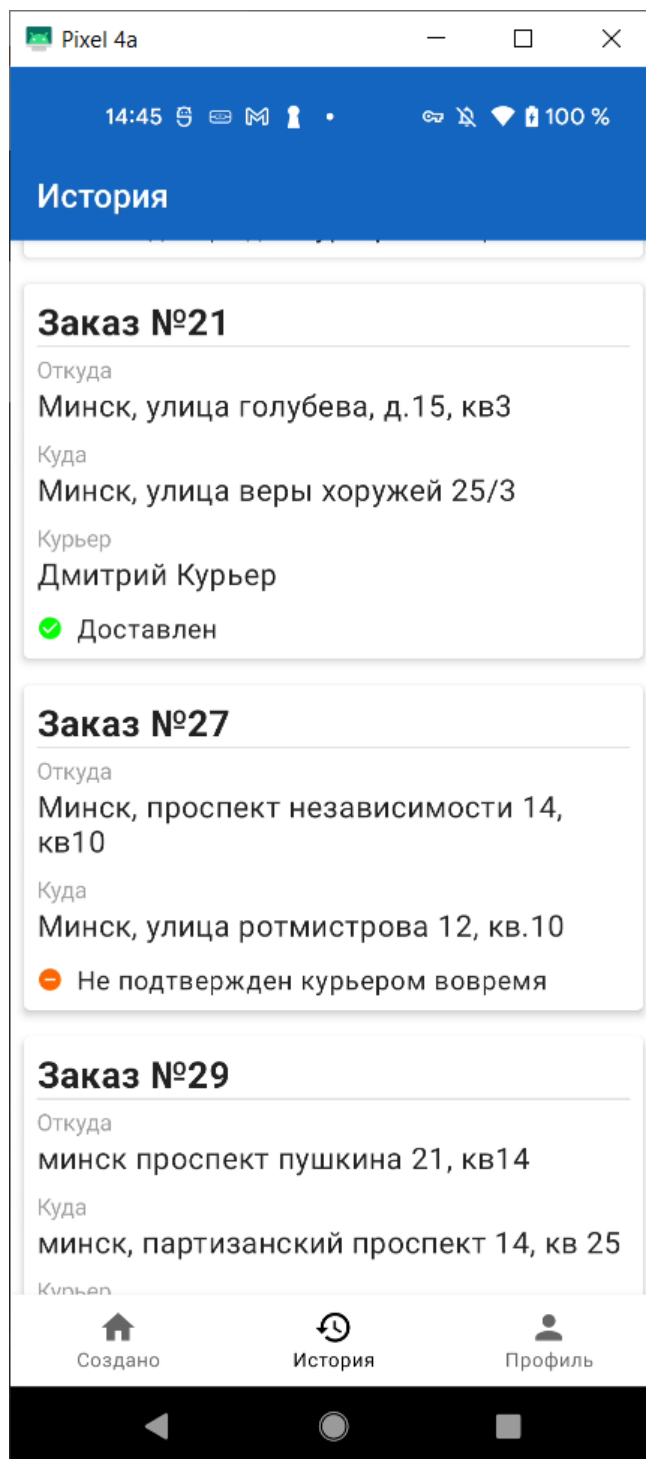


Рисунок 6.6 – Экран истории заказов

Последний экран, доступный пользователю без роли курьера – это экран профиля. Здесь размещена краткая информация о профиле, включающая имя пользователя, адрес электронной почты, номер телефона пользователя. Помимо информации о пользователе здесь расположена кнопка выхода, позволяющая выйти из аккаунта и вернуться на экран входа, а также кнопка, позволяющая получить роль курьера неподтвержденного администратором. После выбора способа передвижения кнопка

заменится на кнопку перехода к экранам курьера. Интерфейс экрана профиля заказчика представлен на рисунке 6.7.

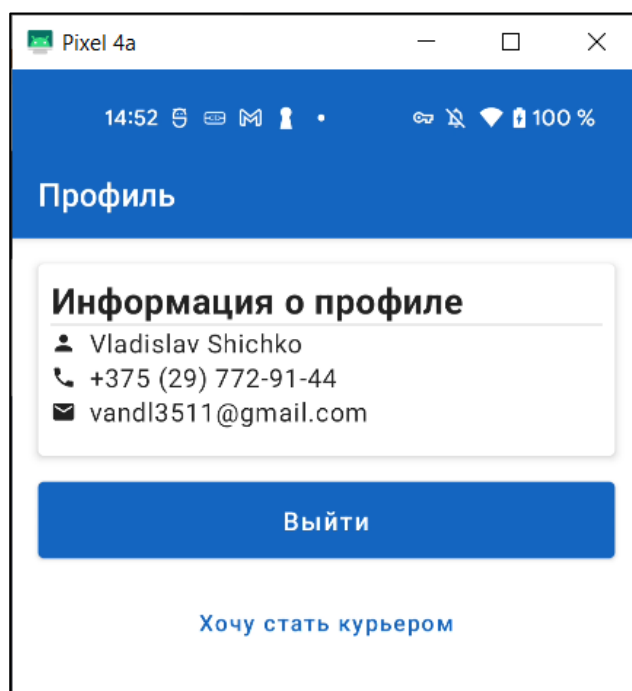


Рисунок 6.7 – Экран профиля заказчика

Далее будут рассмотрены экраны, доступные пользователю с ролью курьера. При входе в приложение под аккаунтом с ролью курьера, откроется экран доступных заказов, содержащий информацию о заказах, доступных к принятию. Интерфейс экрана доступных заказов представлен на рисунке 6.8.

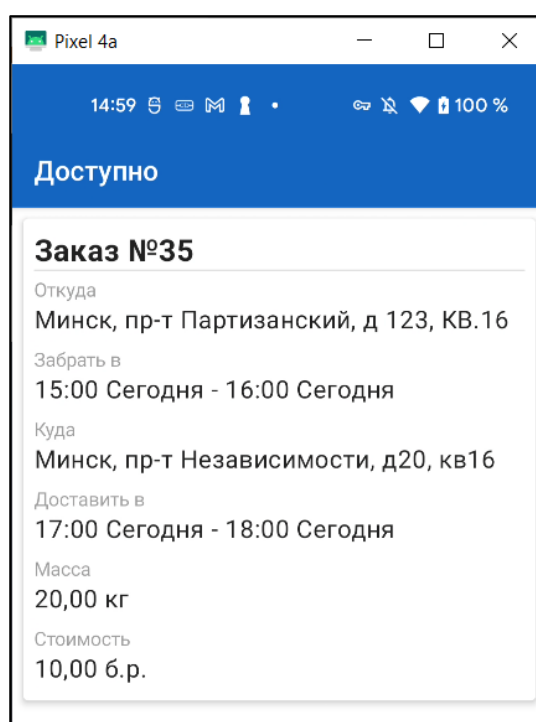


Рисунок 6.8 – Экран доступных заказов для курьера

Экран доступных заказов доступен только подтвержденным курьерам. В случае, если курьер не подтвержден, то вместо списка заказов будет получено сообщение об ошибке.

При нажатии на доступный заказ будет осуществлен переход на экран информации о заказе. На нем размещена информация о стоимости содержимого заказа, массе, а также о точках маршрута и интервалах времени. Помимо информации, здесь размещены две кнопки: снизу справа кнопка принятия заказа, а также кнопка перехода на экран карты в верхнем меню экрана. Если возможно построение маршрута между указанными точками, то при нажатии произойдет переход на экран карты. В противном случае кнопка будет неактивна.

На экране карты отображается карта Google Maps с наложенными на нее двумя маркерами и оптимальным маршрутом между ними. Маршрут строится в соответствии с приоритетным способом передвижения текущего курьера (пешком, на велосипеде, на машине). При нажатии на маркер отображается соответствующий маркеру адрес, а при нажатии на адрес открывается роуп-окно с информацией о точке. Экран карты изображен на рисунке 6.9.

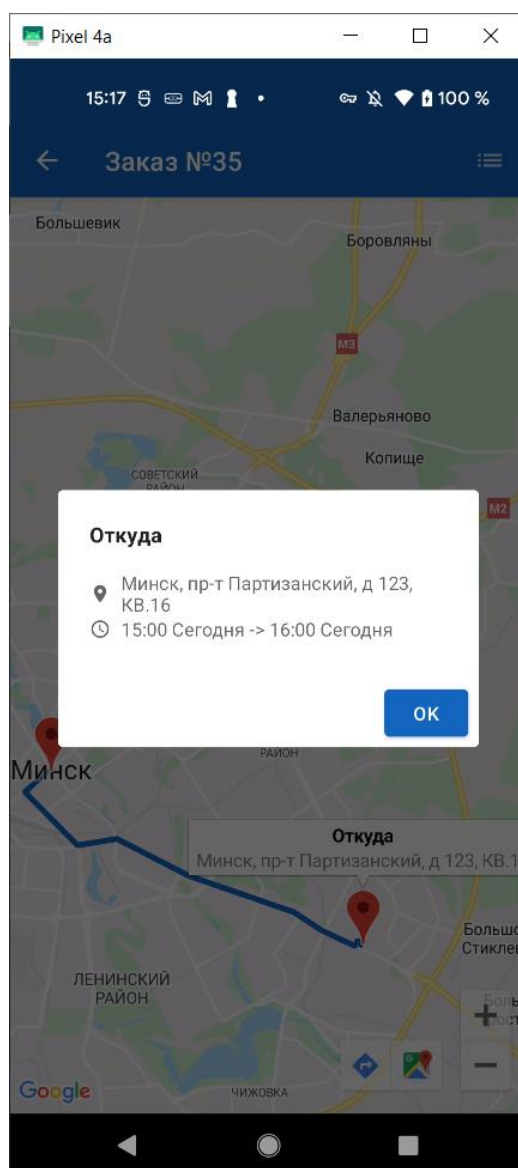


Рисунок 6.9 – Экран карты доступного заказа

При успешном принятии заказа он будет перемещен в список активных заказов на соответствующем экране. Экран активных заказов содержит список активных заказов и позволяет взаимодействовать с ними без подключения к сети интернет. При изменении состояния заказа или при отказе от заказа в режиме оффлайн, изменения будут сохранены в соответствующую таблицу локальной базы данных для последующей синхронизации с сервером при наличии интернет-соединения. Заказы в списке активных заказов группированы по их состоянию: заказы, которые были приняты курьером, но еще не доставляются, располагаются под меткой «Заказано», а доставляемые заказы находятся в списке под меткой «Доставляется».

При нажатии на заказ открывается экран с информацией об активном заказе. Интерфейс этого экрана изображен на рисунке 6.10.

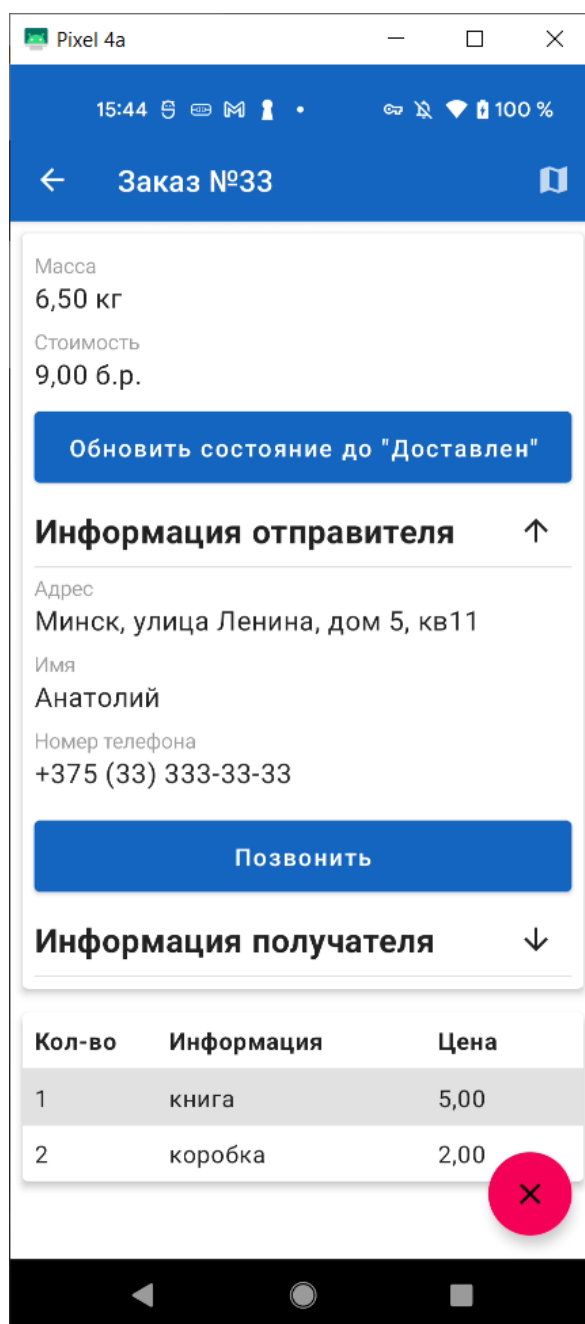


Рисунок 6.10 – Экран информации об активном заказе

На экране информации об активном заказе располагается информация об отправителе и получателе заказа, а также о его содержимом. Кнопка «Обновить состояние» позволяет переводить заказ последовательно из состояния «Заказан» в состояние «Доставляется» и «Доставлен». Кнопка снизу справа позволяет отказаться от принятого заказа, а кнопка в верхнем меню позволяет перейти на экран карты, аналогичный тому, который предназначен для доступных заказов.

Курьеру также доступен экран профиля, изображенный на рисунке 6.11.

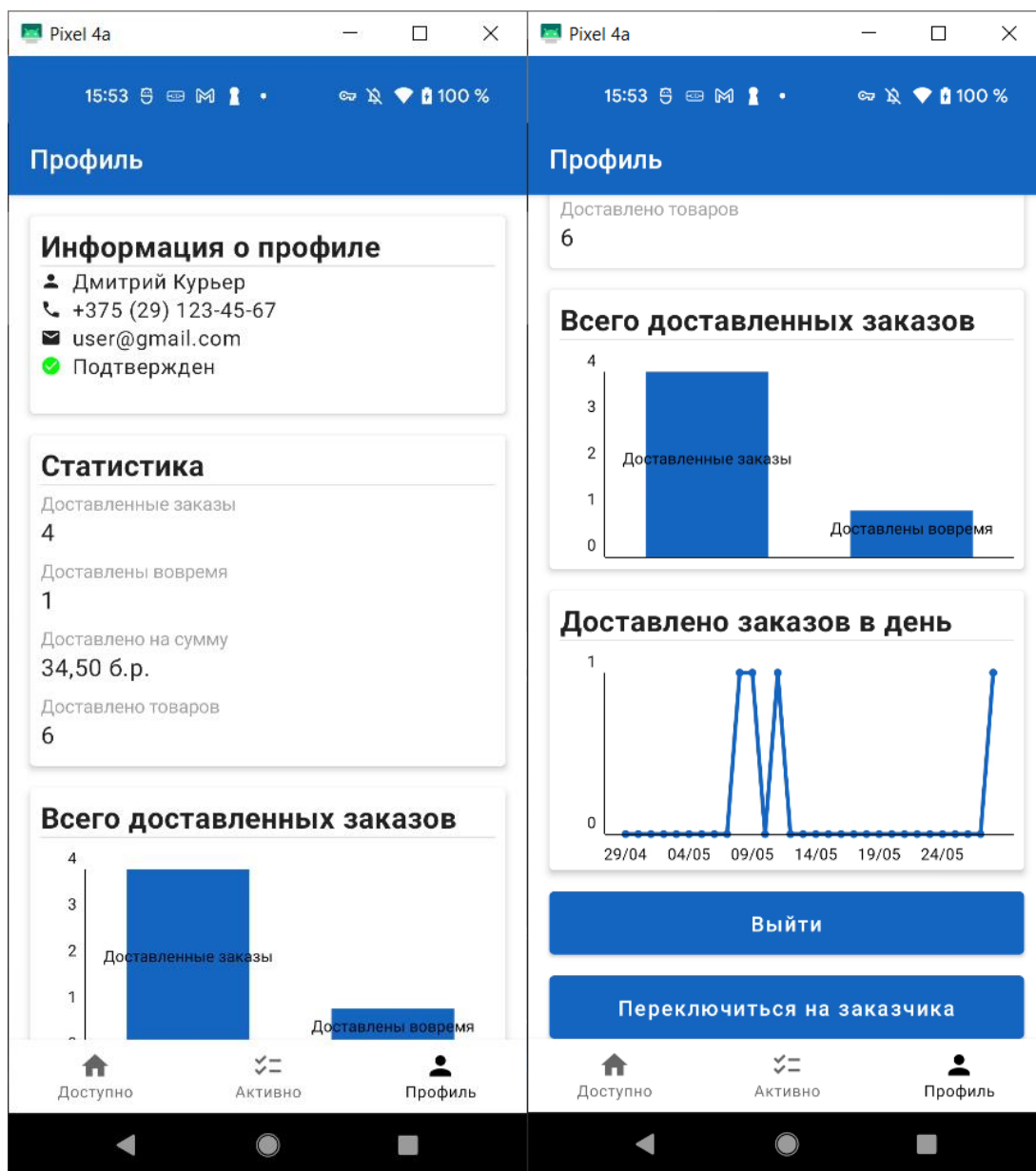


Рисунок 6.11 – Экран профиля курьера

На экране профиля курьера располагается информация об аккаунте: имя, адрес электронной почты, телефон, и состояние аккаунта. Помимо информации об аккаунте, на экране профиля курьера отображается статистика в виде текстовой информации и графиков. Также здесь расположены кнопки выхода из аккаунта и переключения на экраны заказчика.



## 6.2 Веб-приложение

При первом запуске приложения для страниц, требующих аутентификацию, на экране появится соответствующее сообщение (рисунок 6.12) с кнопкой «Login» для перехода на страницу входа. На страницу входа можно так же попасть, нажав на соответствующую иконку справа в шапке приложения.

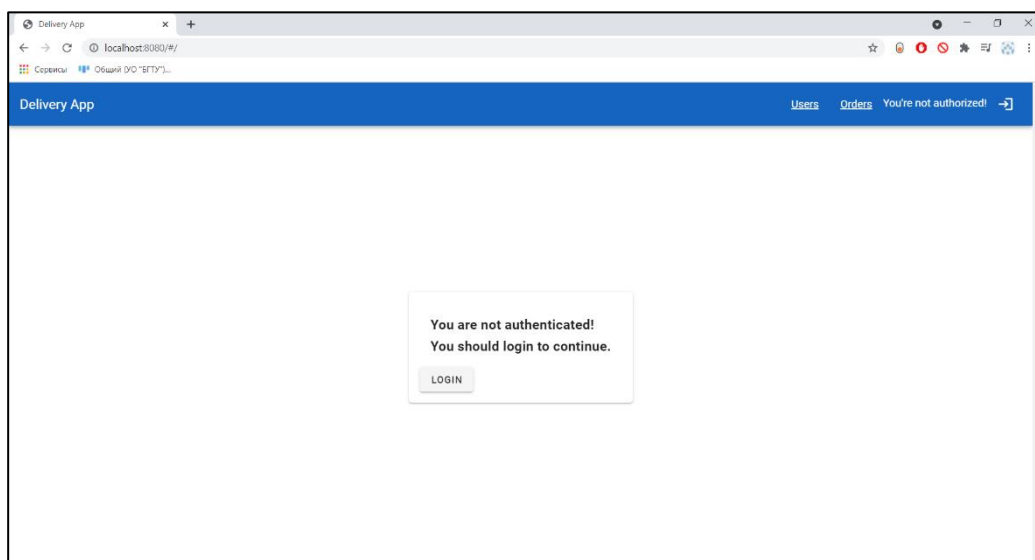


Рисунок 6.12 – Сообщение об ошибке

Страницы входа и регистрации аналогичны соответствующим экранам мобильного приложения за тем исключением, что это приложение предназначено только для администраторов и при попытке входа при помощи аккаунта не администратора возникнет ошибка.

При успешной аутентификации и авторизации пользователю доступны две страницы: страница пользователей и страница заказов. Переключаться между страницами можно по ссылками в шапке приложения.

На странице пользователей (рисунок 6.13) присутствует таблица пользователей приложения, позволяющая выдавать роли при нажатии на соответствующие кнопки.

 A screenshot of the 'Delivery App' web interface showing the 'Users' page. The browser's address bar shows 'localhost:8080/#/'. The app's header is blue with the text 'Delivery App' on the left and navigation links 'Users', 'Orders', and 'Ivan Adminov' on the right. The main content area is white and contains a table of users. The table has columns for ID, First Name, Second Name, Phone, Email, and Roles. There are four rows of user data. Each row has two buttons: 'GRANT COURIER' and 'REVOKE COURIER' for the first three rows, and 'GRANT ADMIN' and 'REVOKE ADMIN' for the last row. The table is paginated, showing '1-4 of 4' rows.
 

ID	First Name	Second Name	Phone	Email	Roles	Buttons
2	Ivan	Adminov	+375291234567	admin@gmail.com	ROLE_COURIER, ROLE_BASIC, ROLE_ADMIN	GRANT COURIER, REVOKE COURIER, GRANT ADMIN, REVOKE ADMIN
3	Ivan	Courierov	+375291234567	user@gmail.com	ROLE_COURIER, ROLE_BASIC	GRANT COURIER, REVOKE COURIER, GRANT ADMIN, REVOKE ADMIN
4	Ivan	Basic	+375291234567	basic@gmail.com	ROLE_BASIC	GRANT COURIER, REVOKE COURIER, GRANT ADMIN, REVOKE ADMIN
18	Test	User	+375 (33) 123-45-67	valid@gmail.com	ROLE_BASIC	GRANT COURIER, REVOKE COURIER, GRANT ADMIN, REVOKE ADMIN

Рисунок 6.13 – Страница пользователей

Также при нажатии на строку в таблице появится окно со статистикой соответствующего пользователя (рисунок 6.14).

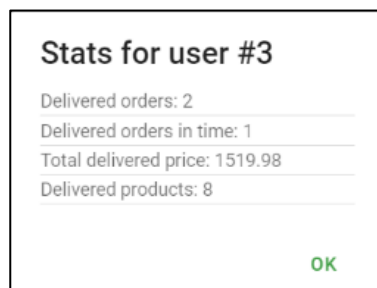


Рисунок 6.14 – Окно статистики пользователя

На странице заказов (рисунок 6.15) размещена таблица с информацией о заказах. Для каждой строки в таблице заданы две кнопки: «Set Canceled» и «Set Ordered». При нажатии на кнопку «Set Canceled» соответствующему заказу задается состояние «Отменен», а при нажатии на «Set Ordered» заказ принимает состояние по умолчанию (Заказан) с той целью, чтобы его можно было сделать доступным для курьеров.

Delivery App

localhost:8080/#/orders

СервисыОбщий (Ю "Сту")...

Delivery App

UsersOrdersIvan Adminov

ID	Products					Address	Info	Phone	Customer name	Courier ID	Total price	State	Cancel	Order
2	ID	Amount	Name	Weight	Price	Минск, ул. Плеханова д.25 кв.34		+375 (29) 772-94-93	Ivan Ivanov		314.97	Canceled	SET CANCELLED	SET ORDERED
	72	1	Smartphone Samsung A10	0.2	300									
	73	3	Cup with photo	0.55	4.99									
5	ID	Amount	Name	Weight	Price	Минск, ул. Ленина 1 кв12		+375 (44) 123-45-67	Maksim Maksimov	3	4502.48	Delivering	SET CANCELLED	SET ORDERED
	85	2	Sony Playstation 5	2.7	1000									
	86	1	Laptop MacBook Pro 15" 2021	1.2	2499.99									
	84	1	USB cable 1.5m	0.1	2.49									
9	ID	Amount	Name	Weight	Price	Минск, пр-т Рокоссовского 19-26	Код домофона 2211	+375 (29) 778-85-55	Anton Antonov	3	759.99	Delivered	SET CANCELLED	SET ORDERED
	78	1	Patch cord 15m Ethernet	1	9.99									
	77	3	Washing machine Indesit	50	250									
	ID	Amount	Name	Weight	Price									

Рисунок 6.15 – Страница заказов

Таким образом, страница заказов позволяет просматривать информацию о заказах, а также отменять заказы.

### 6.3 Выводы по разделу

Было разработано руководство для пользователей, предусмотренных в приложении для правильного использования приложения для доставки заказов.

Данный раздел предоставляет пояснения по работе пользователя с приложением. Так как интерфейс приложения прост и интуитивно понятен, то у пользователя не может возникнуть трудностей с его эксплуатацией.

## 7 Технико-экономическое обоснование проекта

### 7.1 Общая характеристика разрабатываемого программного средства

Основной целью экономического раздела является экономическое обоснование целесообразности разработки программного средства (ПС), представленного в дипломной работе. В этом разделе пояснительной записки проводится расчет затрат на всех стадиях разработки, а также анализ экономического эффекта в связи с использованием данного программного средства.

Во время разработки приложения использовались технологии Spring Framework, JPA, Compose, языки программирования Java, Kotlin, SQL, хостинг Heroku. Разработанное программное средство представляет собой мобильное приложение для создания заказов, а также для организации доставки заказов курьерами. Приложение предназначено для пользователей, которым необходимо доставить заказ, а также для курьеров, занимающихся доставкой. Помимо мобильного приложения разработано веб-приложение для администраторов, позволяющее управлять ролями пользователей и состоянием заказов.

Разработанное мобильное приложение для доставки заказов превосходит аналогичные решения, рассмотренные в дипломном проекте, по наличию функциональных возможностей для создания заказов заказчиком, а также отслеживания состояния заказа, и более современным дизайном, соответствующим стилю Material Design.

По результатам анализа применяемых продуктами-аналогами стратегий монетизации была выбрана стратегия расширенной подписки: курьеры могут доставлять ограниченное количество заказов в день при пользовании бесплатной версией приложения, а при оформлении платной подписки это ограничение снимается.

### 7.2 Исходные данные для проведения расчетов

Исходные данные для расчета приведены в таблице 7.1.

Таблица 7.1 – Исходные данные для расчета

Наименование показателя	Единица измерения	Условные обозначения	Норматив
1	2	3	4
Коэффициент изменения скорости обработки информации	ед.	$K_{ск}$	0,6
Численность разработчиков	чел.	$Ч_p$	1
Норматив дополнительной заработной платы	%	$H_{дз}$	15

				БГТУ 07.00.ПЗ							
	Ф.И.О.	Подпись	Дата								
Разраб.	Шичко В.С.			7 Технико-экономическое обоснование проекта	Лит.			Лист		Листов	
Провер.	Нистюк О.А.							1		12	
Консульт.	Соболевский А.С.				74217084, 2022						
Н. контр.	Нистюк О.А.										
Утв.	Смелов В.В.										

Продолжение таблицы 7.1

1	2	3	4
Ставка отчислений в Фонд социальной защиты населения	%	$H_{\text{фсзн}}$	34
Ставка отчислений в БРУСП «Белгосстрах»	%	$H_{\text{бгс}}$	0,6
Цена одного машино-часа	руб.	$C_{\text{мч}}$	0,06
Норматив прочих затрат	%	$H_{\text{пз}}$	18,5
Норматив накладных расходов	%	$H_{\text{обп, обх}}$	10
Норматив расходов на сопровождение и адаптацию	%	$H_{\text{рса}}$	17
Ставка НДС	%	$H_{\text{ндс}}$	20

Приведенные в таблице 7.1 данные будут использованы при дальнейших вычислениях.

### 7.3 Методика обоснования цены

Разработка проектов программных средств, требует затрат разнообразных ресурсов (трудовых, материальных и финансовых). В связи с этим, необходимость разработки и реализации каждого проекта обосновывается, как с технической точки зрения, так и с экономической. Для обоснования экономической целесообразности проекта вычисляются различные показатели.

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид ПС, реализуемые покупателям по рыночным отпускным ценам.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

- снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
- сокращения расходов на оплату машинного времени и других ресурсов на отладку программ;
- снижения расходов на материалы;
- ускорение ввода в эксплуатацию новых систем;
- улучшения показателей деятельности в результате использования ПС.

Стоимостная оценка ПС у разработчиков предполагает определение затрат, что включает следующие статьи:

- заработная плата исполнителей – основная и дополнительная;
- отчисления в фонд социальной защиты населения;

- отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
- расходы на материалы и комплектующие;
- расходы на спецоборудование;
- расходы на оплату машинного времени;
- прочие прямые затраты;
- накладные расходы.

На основании затрат рассчитывается себестоимость и отпускная цена ПС.

#### 7.4 Определение объема программного средства

В таблице 7.2 указаны в укрупнённом виде все работы, реально выполненные для создания, указанного в дипломной работе программного средства и количество рабочих дней, реально потраченных для выполнения этих работ.

Таблица 7.2 – Затраты рабочего времени на разработку ПС

Содержание работ	Затраты рабочего времени, дней
1. Построение диаграмм для проектирования дипломного проекта	4
2. Создание базы данных	3
3. Разработка серверной части	14
4. Разработка клиентской части	20
5. Тестирование внешнего вида	2
6. Тестирование бизнес-логики	3
7. Тестирование базы данных	2
8. Написание руководства пользователя	1
Всего	49

Результат по данной таблице будет использован далее для расчётов.

Для оценки объёма программного средства, все его функции классифицируются с использованием специального каталога функций, который определяет их объем. Общий объем программного средства  $V_o$ , вычисляется как сумма объёмов  $V_i$  каждой из  $n$  его функций по формуле 7.1.

$$V_o = \sum_{i=1}^n V_i, \quad (7.1)$$

где  $V_i$  – объем  $i$ -ой функции ПС, условных машинных команд;  
 $n$  – общее число функций.

В таблице 7.3 представлены функции, присутствующие в рассматриваемом программном средстве и соответствующий им объем в условных машино–командах.

Таблица 7.3 – Содержание и объем функций в программном средстве

№ функции	Содержание функции	Объем, условных машино-команд
101	Организация ввода информации	2100
102	Контроль, предварительная обработка	1500
111	Управление вводом/выводом	1200
202	Взаимодействие между компонентами системы	1600
401	Взаимодействие с базой данных	1000
402	Вспомогательные методы	500
506	Обработка ошибочных и сбойных ситуаций	300
707	Графический вывод результатов	1400
	Итого	8700

Опираясь на данные таблицы 7.3, можно определить объем программного средства, разработанного в ходе дипломного проектирования:

$$V_o = 2100 + 1500 + 1200 + 1600 + 1000 + 500 + 300 + 1400 = 8700 \text{ (условных машино-команд)}.$$

Скорректированный объем программного средства  $V'_o$  вычисляется по формуле 7.2.

$$V'_o = V_o \cdot K_{ск}, \quad (7.2)$$

где  $V_o$  – объем программного средства, условных машино-команд;  
 $K_{ск}$  – коэффициент изменения скорости обработки информации.

Исходя из вычисленного объёма программного средства, можно определить уточненный объем программного средства:

$$V'_o = 8700 \cdot 0,6 = 5220 \text{ (условных машино–команд)}.$$

## 7.5 Основная заработная плата

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов программирования на Kotlin Android. В итоге было установлено, что средняя месячная заработная плата на позиции junior составляет 1 670 рублей.

Согласно таблице 7.2, проект разрабатывался одним человеком на протяжении 49 рабочих дней, что соответствует 2,3 месяца. Таким образом, основная заработная плата будет рассчитываться по формуле 7.3:

$$C_{оз} = T_{раз} \cdot K_{раз} \cdot C_{зп}, \quad (7.3)$$

где  $C_{оз}$  – основная заработная плата, руб.;  
 $T_{раз}$  – время разработки, месяцев;  
 $K_{раз}$  – количество разработчиков, человек;  
 $C_{зп}$  – средняя месячная заработная плата.

$$C_{\text{оз}} = 2,3 \cdot 1 \cdot 1\,670 = 3\,841 \text{ руб.}$$

В дальнейшем для других расчётов используется основная заработная плата, рассчитанная по указанной выше методике.

### 7.6 Дополнительная заработная плата

Дополнительная заработная плата на конкретное программное средство включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах к основной заработной плате по формуле (7.4):

$$C_{\text{дз}} = \frac{C_{\text{оз}} \cdot H_{\text{дз}}}{100}, \quad (7.4)$$

где  $C_{\text{оз}}$  – основная заработная плата, руб.;

$H_{\text{дз}}$  – норматив дополнительной заработной платы, %.

$$C_{\text{дз}} = 3\,841 \cdot 15 / 100 = 576,15 \text{ руб.}$$

### 7.7 Отчисления в Фонд Социальной защиты населения

Отчисления в Фонд социальной защиты населения (ФСЗН) и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной заработной платы исполнителей.

Отчисления в Фонд социальной защиты населения вычисляются по формуле 7.5

$$C_{\text{фсзн}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot H_{\text{фсзн}}}{100}, \quad (7.5)$$

где  $C_{\text{оз}}$  – основная заработная плата, руб.;

$C_{\text{дз}}$  – дополнительная заработная плата на конкретное ПС, руб.;

$H_{\text{фсзн}}$  – норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 7.6

$$C_{\text{бгс}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot H_{\text{бгс}}}{100}, \quad (7.6)$$

$$C_{\text{фсзн}} = \frac{(3\,841 + 576,15) \cdot 34}{100} = 1\,501,83 \text{ руб.}$$

$$C_{\text{бгс}} = \frac{(3\,841 + 576,15) \cdot 0,6}{100} = 26,5 \text{ руб.}$$

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 26,5 руб., а в фонд социальной защиты населения – 1501,83 руб.

### 7.8 Расходы на материалы

Сумма расходов на материалы  $C_M$  определяется как произведение нормы расхода материалов в расчете на сто строк исходного кода  $H_M$  на уточненный объем программного средства  $V_o'$ , по формуле 7.7

$$C_M = H_M \cdot \frac{V_o'}{100}. \quad (7.7)$$

Учитывая, что норма расхода материалов в расчете на сто строк исходного кода равен 0,46 руб. (по данным, приведенным в приложении 2 таблице П 2.10 «Оценка значений среднего расхода материалов на разработку и отладку 100 строк кода применения ПС» методического пособия [22]), можно определить сумму расходов на материалы.

$$C_M = 0,46 \cdot 5\,220 / 100 = 24,01 \text{ руб.}$$

Сумма расходов на материалы была вычислена на основе данных, приведенных в таблице 7.1 данного дипломного проектирования.

### 7.9 Расходы на оплату машинного времени

Сумма расходов на оплату машинного времени  $C_{MB}$  определяется как произведение стоимости одного машино-часа  $C_{MЧ}$  на уточненный объем программного средства  $V_o'$  и на норматив расхода машинного времени на отладку ста строк исходного кода  $H_{MB}$ , по формуле 7.8

$$C_{MB} = C_{MЧ} \cdot \frac{V_o'}{100} \cdot H_{MB}. \quad (7.8)$$

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен 12 (по данным, приведенным в приложении 2 таблице П 2.11 «Оценка значений среднего машинного времени на отладку 100 строк исходного кода без применения ПС» методического пособия [22]), можно определить сумму расходов на оплату машинного времени.

$$C_{MB} = 0,06 \cdot 5\,220 \cdot 12 / 100 = 37,58 \text{ руб.}$$

Сумма расходов на оплату машинного времени была вычислена на основе данных, приведенных в таблице 7.1 данного дипломного проектирования.



### 7.10 Прочие прямые затраты

Сумма прочих затрат  $C_{пз}$  определяется как произведение основной заработной платы исполнителей на конкретное программное средство  $C_{оз}$  на норматив прочих затрат в целом по организации  $H_{пз}$ , и находится по формуле 7.9.

$$C_{пз} = \frac{C_{оз} \cdot H_{пз}}{100}. \quad (7.9)$$

Все данные, необходимые для вычисления, есть, поэтому можно определить сумму прочих затрат.

$$C_{пз} = 3\,841 \cdot 18,5 / 100 = 710,59 \text{ руб.}$$

### 7.11 Накладные расходы

Сумма накладных расходов  $C_{обп,обх}$  – произведение основной заработной платы исполнителей на конкретное программное средство  $C_{оз}$  на норматив накладных расходов в целом по организации  $H_{обп,обх}$ , по формуле 7.10.

$$C_{обп,обх} = \frac{C_{оз} \cdot H_{обп,обх}}{100}. \quad (7.10)$$

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов.

$$C_{обп,обх} = 3\,841 \cdot 10 / 100 = 384,1 \text{ руб.}$$

### 7.12 Сумма расходов на разработку программного средства

Сумма расходов на разработку программного средства  $C_p$  определяется как сумма основной и дополнительной заработной платы исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов, по формуле 7.11

$$C_p = C_{оз} + C_{дз} + C_{фсзн} + C_{бгс} + C_m + C_{мв} + C_{пз} + C_{обп,обх}. \quad (7.11)$$

Все данные необходимые для вычисления есть, поэтому можно определить сумму расходов на разработку программного средства.

$$C_p = 3\,841 + 576,15 + 1501,83 + 26,5 + 24,01 + 37,58 + 710,59 + 384,1 = 7\,101,76 \text{ руб.}$$

Сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 7 101,76 рублей.

### 7.13 Расходы на сопровождение и адаптацию

Сумма расходов на сопровождение и адаптацию программного средства  $C_{pca}$  определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию  $H_{pca}$ , и находится по формуле 7.12.

$$C_{pca} = \frac{C_p \cdot H_{pca}}{100}, \quad (7.12)$$

$$C_{pca} = 7\,101,76 \cdot 17 / 100 = 1\,207,3 \text{ руб.}$$

Сумма расходов на сопровождение и адаптацию была вычислена на основе данных, рассчитанных ранее в данном разделе.

Все проведенные выше расчеты необходимы для вычисления полной себестоимости проекта.

### 7.14 Полная себестоимость

Полная себестоимость  $C_{\Pi}$  определяется как сумма двух элементов: суммы расходов на разработку  $C_p$  и суммы расходов на сопровождение и адаптацию программного средства  $C_{pca}$ .

Полная себестоимость  $C_{\Pi}$  вычисляется по формуле 7.13

$$C_{\Pi} = C_p + C_{pca}, \quad (7.13)$$

$$C_{\Pi} = 7\,101,76 + 1\,207,3 = 8\,309,1 \text{ руб.}$$

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

### 7.15 Определение цены, оценка эффективности

Продукты-аналоги:

1. Measoft app – приложение для организации доставки заказов курьером. Позволяет автоматически планировать маршрут пешим курьерам и курьерам на автомобиле, производить взаиморасчеты с клиентами, вести складской учет. Имеет интеграцию с популярными CMS, СМС-уведомления, штрихкодирование, кассовое обслуживание. Распространяется по модели простой подписки.

2. Deliveries – приложение для сотрудников курьерских компаний. Позволяет организовать рабочий день для максимальной производительности и эффективного управления временем. Позволяет упорядочить список доставок по наиболее эффективному маршруту, отображает маршрут на текущий день и рассчитывает время прибытия. Распространяется по модели расширенной подписки.

Выбранная стратегия монетизации: расширенная подписка. Такой метод был выбран по той причине, что бесплатное использование даст возможность ознакомиться с продуктом большому количеству пользователей.

Перечень важнейших характеристик, определяющих качество рассматриваемого программного продукта:

1. Дизайн. Этот параметр характеризует привлекательность пользовательского интерфейса приложения.

2. Юзабилити (удобство использования). Этот параметр характеризует степень удобства ПО для пользователей, его наглядность, легкость эксплуатации и изучения.

3. Функциональность. ПО признается функциональным, если выполняет возложенные на него задачи, отвечает заданным потребностям пользователей. Данный аспект предполагает правильную и точную работу, совместимость всех входящих в состав компонентов.

4. Количество статистической информации. Параметр характеризует количество статистической информации, которая отображается в приложении.

Таблица 7.4 – Показатели качества рассматриваемых программных средств

Показатель качества	Весовой коэфф.	MeaSoft App	Deliveries	Рассматриваемый продукт
Дизайн	0,2	7	7	8
Юзабилити	0,3	8	6	7
Функциональность	0,4	10	8	8
Количество статистической информации	0,1	7	5	8
Всего	1	8,5	6,9	7,7

Интегральный показатель качества продукции конкурента ИК рассчитывается по формуле 7.14:

$$ИК = \Sigma (K_i \cdot ПК_i) / \Sigma K_i \text{ при } \Sigma K_i = 1, \quad (7.14)$$

где  $K_i$  – весовой коэффициент, отражающий значимость  $i$ -го показателя качества;

$ПК_i$  – число баллов, присвоенное  $i$ -му показателю качества продукта конкурента.

$$ИК_1 = 7 \cdot 0,2 + 8 \cdot 0,3 + 10 \cdot 0,4 + 7 \cdot 0,1 = 7,5$$

$$ИК_2 = 7 \cdot 0,2 + 6 \cdot 0,3 + 8 \cdot 0,4 + 5 \cdot 0,1 = 6,9$$

Показатель качества рассматриваемого продукта ИР рассчитывается по формуле 7.15:

$$ИР = \Sigma (K_i \cdot ПР_i) / \Sigma K_i \text{ при } \Sigma K_i = 1, \quad (7.15)$$

где  $ПР_i$  – число баллов, присвоенное  $i$ -му показателю качества рассматриваемого продукта.

$$ИР = 8 \cdot 0,2 + 7 \cdot 0,3 + 8 \cdot 0,4 + 8 \cdot 0,1 = 7,7$$

Определение отпускной цены нового продукта  $Ц_1$  осуществляется по формуле:

$$Ц_1 = (Ц_0 \cdot ИР) / ИК, \quad (7.16)$$

где  $C_0$  – цена программного продукта конкурента.

Цена за расширенную подписку на месяц составляет:

– MeaSoft App: 7\$ (18,2 рублей в месяц);

– Deliveries: 5\$ (13 рублей в месяц).

$$C_1 = 18,2 \cdot 7,7 / 8,5 = 16,5 \text{ (рублей в месяц),}$$

$$C_2 = 13 \cdot 7,7 / 6,9 = 14,5 \text{ (рублей в месяц),}$$

$$C = (16,5 + 14,5) / 2 = 15,5 \text{ (рублей в месяц).}$$

Расчёт прогнозного количества установок программного средства  $K_1$  при монетизации методом размещения рекламы, рассчитывается по формуле 7.17:

$$K_1 = (K_0 / T_0 \cdot \text{ИР}) / \text{ИК} \quad (7.17)$$

где  $K_0$  – количество установок ПС конкурента;

$T_0$  – количество лет существования приложения;

ИР – показатель рассматриваемого программного продукта;

ИК – показатель программного продукта конкурента.

$$K_1 = (10\,000 / 6 \cdot 7,7) / 8,5 = 1\,510 \text{ (установок в год),}$$

$$K_2 = (50\,000 / 4 \cdot 7,7) / 6,9 = 13\,949 \text{ (установок в год),}$$

$$K = (1\,510 + 13\,949) / 2 = 7\,730 \text{ (установок в год).}$$

Денежные поступления от расширенной подписки в приложении можно рассчитать по формуле 7.18:

$$P_{\text{ост.в год}} = K_{\text{п}} \cdot C \cdot 12 \quad (7.18)$$

где  $K_{\text{п}}$  – количество пользователей за год, оформивших расширенную подписку;

$C$  – цена расширенной подписки в месяц, руб.

Если 1% пользователей оформит расширенную подписку (т.е. 77 пользователей в год), то денежные поступления в год составят:

$$P_{\text{ост.в год}} = 77 \cdot 15,5 \cdot 12 = 14\,322 \text{ (рублей за год).}$$

Срок окупаемости приложения  $T_{\text{ок}}$  вычисляется по формуле 7.19

$$T_{\text{ок}} = C_{\text{п}} / P_{\text{ост.в год}}, \quad (7.19)$$

где  $C_{\text{п}}$  – полная себестоимость, руб.;

$P_{\text{ост.в год}}$  – денежные поступления от расширенной подписки в приложении за год, руб.

$$T_{\text{ок}} = 8309,1 / 14\,322 = 0,58 \text{ года.}$$

### 7.16 Выводы по разделу

Разработка программного средства, осуществляемая одним разработчиком в течение 2,3 месяцев, при заданных условиях обойдется в 8 309,1 руб. Реализация данного программного средства будет приносить годовые денежные поступления от расширенной подписки в размере 14 322,1 рублей и окупится через 0,58 года или 7 месяцев.

В таблице 7.5 и в приложении Д представлены результаты расчётов для основных показателей данного раздела в краткой форме.

Таблица 7.5 – Результаты расчетов

Наименование показателя	Значение
Время разработки, мес.	2,3
Количество программистов, чел.	1
Основная заработная плата, руб.	3 841
Дополнительная заработная плата, руб.	576,15
Отчисления в Фонд социальной защиты населения и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний, руб.	1 528,33
Расходы на материалы, оплату машинного времени, прочие, руб.	772,18
Накладные расходы, руб.	384,1
Себестоимость разработки программного средства, руб.	7 101,76
Расходы на сопровождение и адаптацию, руб.	1 207,3
Полная себестоимость, руб.	8 309,1
Годовые денежные поступления от расширенной подписки, руб.	14 322,1
Срок окупаемости, мес.	7

Необходимость разработки программного средства обусловлена ростом популярности сервисов доставки. Разработанное мобильное приложение для доставки заказов превосходит аналогичные решения, рассмотренные в дипломном проекте, по наличию функциональных возможностей для создания заказов заказчиком, а также отслеживания состояния заказа, и более современным дизайном.

## Заключение

В результате выполнения дипломного проектирования было разработано мобильное приложение для организации доставки заказов курьерами.

В рамках работы над проектом был проведен обзор 2 аналогичных приложений, были рассмотрены их преимущества и недостатки, изучены возможности рассматриваемых программных средств.

При разработке были спроектированы диаграмма вариантов использования, алгоритм изменения состояния заказа, структура базы данных, спроектирована архитектура приложения. Также было проведено тестирование, которое показало, что разработанное программное средство соответствует заданным требованиям.

Были рассмотрены возможные угрозы информационной безопасности, методы защиты сети передачи данных, а также клиентской и серверной частей приложения.

При составлении руководства пользователя была подробно описана работа с программным средством.

При рассмотрении технико-экономического обоснования разработанного проекта было рассчитано количество денежных затрат и трудозатрат на разработку программного средства, прибыль от реализации разработанного программного продукта с учетом способа монетизации в виде расширенной подписки.

В соответствии с поставленными задачами и достигнутым результатом, можно сделать вывод, что дипломный проект выполнен и поставленная цель достигнута. В рамках дипломного проекта было разработано программное средство, включающее в себя мобильное приложение, веб-клиент, сервер, и удаленную базу данных, а также соответствующее предъявленным требованиям.

				БГТУ 00.00.ПЗ			
	Ф.И.О.	Подпись	Дата	Заключение			
Разраб.	Шичко В.С.						
Провер.	Нистюк О.А.						
Н. контр.	Нистюк О.А.						
УТВ.	Смелов В.В.			74217084, 2022			
					Лит.	Лист	Листов
						1	1

## Список использованных источников

- 1 Приложение Measoft. [Электронный ресурс]. – Режим доступа: <https://courierexe.ru/>. – Дата доступа: 15.03.2022.
- 2 Приложение Deliveries. [Электронный ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=com.shakhaev.deliveries>. – Дата доступа: 16.03.2022.
- 3 Android Studio. [Электронный ресурс]. – Режим доступа: <https://developer.android.com/studio/intro>. – Дата доступа: 20.03.2022.
- 4 Kotlin Documentation. [Электронный ресурс]. – Режим доступа: <https://kotlinlang.org/docs/home.html>. – Дата доступа: 20.03.2022.
- 5 Документация Jetpack Compose. [Электронный ресурс]. – Режим доступа: <https://developer.android.com/jetpack/compose/documentation>. – Дата доступа: 28.03.2022.
- 6 IntelliJ IDEA. [Электронный ресурс]. – Режим доступа: <https://www.jetbrains.com/ru-ru/idea/>. – Дата доступа 25.03.2022.
- 7 Spring Projects. [Электронный ресурс]. – Режим доступа: <https://spring.io/projects>. – Дата доступа: 02.04.2022.
- 8 Spring Security [Электронный ресурс] – Режим доступа: <https://spring.io/projects/spring-security>. – Дата доступа: 03.05.2022.
- 9 Vue js [Электронный ресурс] – Режим доступа: <https://vuejs.org/>. – Дата доступа: 01.05.2022.
- 10 Урбанович, П. П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие / П.П. Урбанович. – Минск : БГТУ, 2016. – 220 с.
- 11 Информационные технологии и безопасность. Протокол защиты транспортного уровня (TLS): СТБ 34.101.65-2014. – Введ. 22.05.2014. – Минск: Госстандарт Республики Беларусь, 2004. – 56с.

				<i>БГТУ 00.00.ПЗ</i>			
	Ф.И.О.	Подпись	Дата	Список использованных источников			
Разраб.	Шичко В.С.						
Провер.	Нистюк О.А.						
Н. контр.	Нистюк О.А.						
УТВ.	Смелов В.В.			74217084, 2022			

## **Приложение А Диаграмма вариантов использования**



## **Приложение Б Логическая схема базы данных**

## **Приложение В Диаграмма компонентов приложения**

## **Приложение Г Блок-схема алгоритма изменения состояния заказа**

## **Приложение Д Пользовательский интерфейс приложения**

**Приложение Е Таблица экономических показателей**

## Приложение Ж Листинг программного кода

```

@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
        .csrf()
        .disable()
        .cors()
        .configurationSource(corsConfigurationSource())
        .and()
        .authorizeRequests()
        .antMatchers("/registration").not().fullyAuthenticated()
        .antMatchers("/admin/**").hasRole("ADMIN")
        .antMatchers(HttpMethod.GET, "/order").hasRole("ADMIN")
        .antMatchers("/profile/**").hasRole("BASIC")
        .antMatchers(HttpMethod.POST, "/order/**").hasRole("BASIC")
        .antMatchers(HttpMethod.GET, "/order/created",
            "/order/history").hasRole("BASIC")
        .antMatchers("/order/updateState").hasAnyRole("ADMIN",
            "COURIER")
        .antMatchers(HttpMethod.GET, "/order/{id:\\d+}",
            "/order/delivered").hasAnyRole("COURIER", "BASIC")
        .antMatchers("/order/**").hasRole("COURIER")
        .antMatchers("/", "/login", "/resources/**", "/js/**",
            "/ws/**", "/main.js").permitAll()
        .anyRequest().hasRole("ADMIN")
        .and()
        .formLogin()
        .loginPage("/login")
        .successHandler((request, response, exception) -> {
            response.setStatus(HttpStatus.OK.value());
            response.setHeader("Content-Type", "application/json");
            UserDetails details = (UserDetails)SecurityContextHolder
                .getContext().getAuthentication().getPrincipal();
            UserDto authorizedUser = userMapper.entityToDto(
                (User) userService
                    .loadUserByUsername(details.getUsername())
            );
            PrintWriter pw = response.getWriter();
            ObjectMapper om = new ObjectMapper();
            String json = om.writeValueAsString(authorizedUser);
            pw.print(json);
        })
        .failureHandler((request, response, exception) ->
            response.setStatus(HttpStatus.UNAUTHORIZED.value()))
        .and()
        .exceptionHandling().authenticationEntryPoint(
            new Http401UnauthorizedEntryPoint())
        .and().logout()
        .invalidateHttpSession(true).permitAll()
        .logoutSuccessUrl("/")
        .and()
        .rememberMe().key("D3liv3ryS3rvic3S3cr3tK3y");
}

```

```

object OrderRepository {
    private final val TAG = "OrderService"
    private val orderApi = NetworkService.orderApi.value
    private val db: CourierDatabase = CourierDatabase.getDatabase(context)

    //Получение списка доступных заказов (только онлайн)
    suspend fun getAvailableOrders(): Result<List<Order>> = try {
        val response = orderApi.getAvailableOrders()
        when (response.code()) {
            //Успешное получение данных
            200 -> {
                val mapper = Mappers.getMapper(
                    OrderMapper::class.java
                )
                val orders = mapper.dtosToEntities(response.body())
                Result.success(orders)
            }
            //Возвращается сервером, если пользователь не имеет роли
            //курьера
            403 -> {
                throw CourierNetworkException(
                    "Your account is not verified"
                )
            }
            else -> {
                throw CourierNetworkException("Network Troubles")
            }
        }
    } catch (e: Exception) {
        Result.error(e)
    }

    suspend fun getDeliveredOrders(): Result<List<Order>> = try {
        val response = orderApi.getDeliveredOrders()
        when (response.code()) {
            //Успешное получение данных
            200 -> {
                val mapper = Mappers.getMapper(
                    OrderMapper::class.java
                )
                val orders = mapper.dtosToEntities(response.body())
                Result.success(orders)
            }
            //Возвращается сервером, если пользователь не имеет роли
            //курьера
            403 -> {
                throw CourierNetworkException(
                    "Your account is not verified"
                )
            }
            else -> {

```

```

        throw CourierNetworkException("Network Troubles")
    }
}
} catch (e: Exception) {
    Result.error(e)
}

//Получение списка активных заказов (доступно без подключения к
//интернету)
suspend fun getActiveOrders(): Result<List<Order>> =
if (isOnline(context)) {
    try {
        val response = orderApi.getActiveOrders()
        when (response.code()) {
            //Успешное получение данных
            200 -> {
                val mapper = Mappers.getMapper(
                    OrderMapper::class.java
                )
                val orders = mapper.dtosToEntities(
                    response.body()
                )

                val orderIdList = db.orderDao.getAll()
                    .map { order -> order.id }

                //Обновление локальной базы данных
                //Очистка записей связанных с заказами
                db.orderedDao.truncate()
                db.orderDao.truncate()

                //Помечаем записи об удаленных заказах, как
                //актуальные
                orderIdList.forEach {
                    db.changeDao.setUpToDate("orders", it)
                }

                //Вставка актуальной информации о активных заказах
                //в локальную бд
                for (order in orders) {
                    insertOrderWithReplaceToLocalDb(order)
                }
                //Получение списка активных заказов из локальной
                //бд
                Result.success(getActiveOrdersFromLocalDb())
            }
        }
    }
    //Возвращается сервером, если пользователь не имеет
    //роли курьера
    403 -> {
        db.orderDao.getAll().forEach {
            fillOrderFromDb(it)
            deleteOrderAndOrderedFromLocalDb(it)
        }
    }
}

```



```

        throw CourierNetworkException(
            "Your account is not verified"
        )
    }
    else -> {
        throw CourierNetworkException("Network Troubles")
    }
}
} catch (e: Exception) {
    Result.error(e)
}
} else {
    Result.success(getActiveOrdersFromLocalDb())
}

suspend fun getOrderById(id: Long): Result<Order> =
if (isOnline(context)) {
    try {
        val response = orderApi.getById(id)
        when (response.code()) {
            200 -> {
                val mapper = Mappers.getMapper(
                    OrderMapper::class.java
                )
                val order = mapper.dtoToEntity(response.body())
                Result.success(order)
                insertOrderWithReplaceToLocalDb(order)
                getOrderFromLocalDb(id)
            }
            403 -> {
                throw CourierNetworkException(
                    "Your account is not verified"
                )
            }
            else -> {
                throw CourierNetworkException("Network Troubles")
            }
        }
    } catch (e: Exception) {
        Result.error(e)
    }
} else {
    getOrderFromLocalDb(id)
}

suspend fun getCreatedOrders(): Result<List<Order>> = try {
    val response = orderApi.getCreatedOrders()
    when (response.code()) {
        //Успешное получение данных
        200 -> {
            val mapper = Mappers.getMapper(
                OrderMapper::class.java
            )
            val orders = mapper.dtosToEntities(response.body())

```

```

        Result.success(orders)
    }
    403 -> {
        throw CourierNetworkException(
            "Your account is not verified"
        )
    }
    else -> {
        throw CourierNetworkException("Network Troubles")
    }
}
} catch (e: Exception) {
    Result.error(e)
}

suspend fun getHistory(): Result<List<Order>> = try {
    val response = orderApi.getHistory()
    when (response.code()) {
        //Успешное получение данных
        200 -> {
            val mapper = Mappers.getMapper(
                OrderMapper::class.java
            )
            val orders = mapper.dtosToEntities(response.body())
            Result.success(orders)
        }
        403 -> {
            throw CourierNetworkException(
                "Your account is not verified"
            )
        }
        else -> {
            throw CourierNetworkException("Network Troubles")
        }
    }
} catch (e: Exception) {
    Result.error(e)
}

//Принять заказ (только онлайн)
suspend fun accept(order: Order): Result<Unit> = try {
    val result = orderApi.acceptOrder(order.id)
    if (result.isSuccessful) {
        insertOrderWithReplaceToLocalDb(order)
        Result.success(Unit)
    } else {
        val gson = Gson()
        val jsonObject: JsonObject =
            gson.fromJson(result.errorBody()?.string(),
                JsonObject::class.java)
        Result.error(CourierNetworkException(jsonObject["mes-
sage"].asString))
    }
} catch (e: Exception) {

```

```

        Result.error(e)
    }

    //Отмена заказа (доступно без подключения к интернету)
    suspend fun decline(order: Order): Result<Unit> = try {
        //Удаляем запись о заказе из локальной бд
        //При удалении в таблице changes появится соответствующая
        //запись, при возобновлении соединения,
        //по этой записи можно будет отменить заказ на сервере для
        //синхронизации
        deleteOrderAndOrderedFromLocalDb(order)
        //Если есть интернет, то выполняем отмену заказа и на сервере
        if (isOnline(context)) {
            val result = orderApi.declineOrder(order.id)
            if (!result.isSuccessful) {
                val gson = Gson()
                val jsonObject: JsonObject =
                    gson.fromJson(result.errorBody()?.string(),
                        JsonObject::class.java)
                throw CourierNetworkException(
                    jsonObject["message"].asString
                )
            }
        }
        Result.success(Unit)
    } catch (e: Exception) {
        Result.error(e)
    }

    suspend fun updateState(order: Order): Result<Order> = try {
        val nextState = order.state.next

        if (isOnline(context)) {
            val result = orderApi.updateState(order.id,
                nextState.name)
            if (!result.isSuccessful) {
                val gson = Gson()
                val jsonObject: JsonObject =
                    gson.fromJson(result.errorBody()?.string(),
                        JsonObject::class.java)
                throw CourierNetworkException(
                    jsonObject["message"].asString
                )
            }
        }

        db.orderDao.updateState(order.id, nextState.name)
        val newOrderState = db.orderDao.findById(order.id)!!.state
        order.state = newOrderState
        Result.success(order)
    } catch (e: Exception) {

```

```

        Result.error(e)
    }

    //Метод вызывается при появлении соединения для отправки
    //информации о заказах, от которых курьер отказался
    suspend fun sendDecline() {
        //Получаем список ID заказов, которые были удалены из таблицы
        //заказов во время оффлайн использования
        val declinedOrdersIdList = db.changeDao
            .findAllByTableAndOperation("orders", "delete")
            .filter { change -> !change.isUpToDate }.map { it.itemId }
        for (id in declinedOrdersIdList) {
            //Отправляем соответствующий отмене заказа запрос
            try {
                val result = orderApi.declineOrder(id)
                //Удаляем запись в таблице изменений для
                //предотвращения последующего повторения
                db.changeDao.deleteByTableAndItemId("orders", id)
                Log.d(TAG, "sendDecline status: " + result.code())
            } catch (e: Exception) {
                Log.d(TAG, "sendDecline error: ${e.message}")
            }
        }
    }

    //Метод вызывается при появлении соединения для отправки
    //информации о заказах, состояние которых было обновлено курьером
    suspend fun sendUpdateState() {
        //Получаем список заказов, состояние которых было обновлено во
        //время оффлайн использования
        val updateStateOrdersIdList =
            db.changeDao.findAllByTableAndOperation("orders", "update")
                .map { it.itemId }
        for (id in updateStateOrdersIdList) {
            //Получаем состояние заказа из БД по id
            val state = db.orderDao.getStateById(id)
            //Отправляем соответствующий обновлению состояния заказа
            //запрос на сервер
            try {
                val result = orderApi.updateState(id, state)
                //Удаляем запись в таблице изменений для
                //предотвращения последующего повторения
                db.changeDao.deleteByTableAndItemId("orders", id)
                Log.d(TAG, "sendUpdate status: ${result.code()}")
            } catch (e: Exception) {
                Log.d(TAG, "sendUpdate error: ${e.message}")
            }
        }
    }

    suspend fun save(order: Order): Result<Unit> = try {
        val mapper = Mappers.getMapper(OrderMapper::class.java)
        val response = orderApi.save(mapper.entityToDto(order))
        if (response.isSuccessful)
    }

```

```

        Result.success(Unit)
    else {
        val gson = Gson()
        val jsonObject: JsonObject =
            gson.fromJson(response.errorBody()?.string(),
                JsonObject::class.java)
        throw CourierNetworkException(
            jsonObject["message"].asString
        )
    }
} catch (e: Exception) {
    Result.error(e)
}

private suspend fun insertOrderWithReplaceToLocalDb(order: Order)
{
    db.userDao.insertWithReplace(order.customer)
    if (order.courier != null) {
        db.userDao.insertWithReplace(order.courier)
    }
    db.destinationDao.insertWithReplace(order.sender)
    db.destinationDao.insertWithReplace(order.recipient)
    Log.d("OrderService", "customer")
    db.productDao.insertAll(order.ordered.map { it.product })
    Log.d("OrderService", "products $order")
    db.orderDao.insertWithReplace(order)
    Log.d("OrderService", "order")
    db.orderedDao.insertAll(order.ordered.toList())
    Log.d("OrderService", "ordered")
    db.changeDao.setUpToDate("orders", order.id)
    Log.d("OrderService", "change")
}

private suspend fun deleteOrderAndOrderedFromLocalDb(order: Order)
{
    db.orderedDao.deleteAll(order.ordered)
    db.productDao.deleteAll(order.ordered.map { it.product })
    db.orderDao.delete(order)
}

private suspend fun fillOrderFromDb(order: Order) {
    order.customer = db.userDao.findById(order.customerId)
    if (order.courierId != null) {
        order.courier = db.userDao.findById(order.courierId)
    }
    order.sender = db.destinationDao.findById(order.senderId)
    order.recipient = db.destinationDao
        .findById(order.recipientId)
    val orderedList = db.orderedDao.findByOrderId(order.id)
    for (ordered in orderedList) {
        val product = db.productDao.findById(ordered.productId)
        ordered.product = product
    }
}

```

```

        order.ordered = orderedList
    }

    private suspend fun getActiveOrdersFromLocalDb(): List<Order> {
        //Получаем активные заказы из локальной бд
        val orders = db.orderDao.getAll()
        for (order in orders) {
            fillOrderFromDb(order)
        }
        return orders
    }

    private suspend fun getOrderFromLocalDb(id: Long): Result<Order> {
        val order = db.orderDao.findById(id)
        return if (order != null) {
            fillOrderFromDb(order)
            Result.success(order)
        } else {
            Result.error(
                CourierNetworkException("No order with id $id")
            )
        }
    }
}

class MainActivity : AppCompatActivity() {

    @OptIn(ExperimentalAnimationApi::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView {
            AppTheme {
                val navController = rememberAnimatedNavController()
                AnimatedNavHost(
                    navController = navController,
                    startDestination = Route.SPLASH,
                    enterTransition = {
                        if (disableAnimation())
                            EnterTransition.None
                        else
                            slideIntoContainer(
                                AnimatedContentScope
                                    .SlideDirection.Left,
                                animationSpec = tween(300)
                            )
                    },
                    exitTransition = {
                        if (disableAnimation())
                            ExitTransition.None
                        else
                            slideOutOfContainer(

```

```

        AnimatedContentScope
            .SlideDirection.Left,
        animationSpec = tween(300)
    )
},
popEnterTransition = {
    if (disableAnimation())
        EnterTransition.None
    else
        slideIntoContainer(
            AnimatedContentScope
                .SlideDirection.Right,
            animationSpec = tween(300)
        )
},
popExitTransition = {
    if (disableAnimation())
        ExitTransition.None
    else
        slideOutOfContainer(
            AnimatedContentScope
                .SlideDirection.Right,
            animationSpec = tween(300)
        )
}
) {
    composable(Route.SPLASH) {
        SplashScreen(navController = navController)
    }
    composable(Route.LOGIN) {
        LoginScreen(navController = navController)
    }
    composable(Route.REGISTRATION) {
        RegistrationScreen(
            navController = navController
        )
    }
    composable(Route.CUSTOMER) {
        CustomerScreen(navController)
    }
    composable(Route.COURIER) {
        CourierScreen(navController)
    }

    composable(Route.CREATE_NEW) {
        CreateOrderScreen(navController)
    }
    composable(
        "${Route.CREATE_DETAILS}/{order}/{callNumber}",
        arguments = listOf(navArgument("order") {
            type = NavType.LongType },
            navArgument("callNumber") {
                type = NavType.BoolType }
        )
    )
}

```

```

    ) { entry ->
        val callNumber = entry.arguments
            ?.getBoolean("callNumber", false) ?: false

        CreatedOrderDetailsScreen(
            orderId = entry.getOrderId(),
            callNumber = callNumber,
            navHostController = navController
        )
    }
    composable(
        "${Route.AVAILABLE_DETAILS}/{order}",
        arguments = listOf(navArgument("order") {
            type = NavType.LongType })
    ) { entry ->
        AvailableDetailsScreen(entry.getOrderId(),
            navController)
    }
    composable(
        "${Route.ACTIVE_DETAILS}/{order}",
        arguments = listOf(navArgument("order") {
            type = NavType.LongType })
    ) { entry ->
        ActiveDetailsScreen(entry.getOrderId(),
            navController)
    }
}

}

}
ConnectionListener.init(this)
}

@OptIn(ExperimentalAnimationApi::class)
fun AnimatedContentScope<NavBackStackEntry>.disableAnimation() =
    ((initialState.destination.route == Route.CUSTOMER
        || initialState.destination.route == Route.COURIER)
        && (targetState.destination.route == Route.CUSTOMER
            || targetState.destination.route == Route.COURIER
            || targetState.destination.route == Route.LOGIN))

fun NavBackStackEntry.getOrderId() = arguments?.getLong("order")
?: -1
}

```