

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программное обеспечение автоматизированной
системы управления магазином электроники**

Студент _____ Барабанов Андрей Валерьевич _____
(подпись, дата) (ФИО полностью)

Руководитель _____ Князев В.Н. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В. _____
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю. _____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А. _____
(подпись) (фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«26» декабря 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Барабанов Андрей Валерьевич гр. 22ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программное обеспечение автоматизированной системы управления магазином электроники

Тема утверждена приказом ПГУ № _____ от “___” _____ 20__

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н. _____

4. Задание на работу (назначение разработки, исходные данные и т.п.)

- 1) *Разработать программное обеспечение автоматизированной системы управления магазином электроники в форме мобильного программного приложения.*
- 2) *Среда выполнения приложения: операционная система Android 7 и выше*
Среда выполнения сервера: операционная система Ubuntu Server 24.04 и выше
Языковые средства разработки: C#, Kotlin, UML, SQL, Linq, Bash
Инструментальные средства разработки: СУБД PostgreSQL, PgAdmin, Microsoft Code, Visual Paradigm, Microsoft Project
Технологии разработки: RUD, ООП, MVVM, Entity Framework
- 3) *Основные функции мобильного приложения:*
 - а) просмотр каталога электроники;*
 - б) оформление заказов электроники;*
 - в) добавление, удаление и редактирования информации о товарах в каталоге;*
 - г) обработка отзывов и вопросов к менеджеру;*
 - д) доставка сообщений через систему пуш-уведомлений*
 - е) фильтрация каталога;*
 - ж) просмотр отчета о продажах.*

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
	Анализ предметной области и требований к системе автоматизации	10%	07.04.2026 – 13.04.2026	
	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 145 листов, 55 рисунков, 37 таблиц, 38 использованных источников, 6 приложений.

ANDROID, МАГАЗИН ЭЛЕКТРОНИКИ, POSTGRESQL, C#, KOTLIN, .NET, RAD, КРОССПЛАТФОРМЕННОСТЬ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, МОДУЛЬНЫЙ МОНОЛИТ, UML, MVVM, MS OFFICE PROJECT

Объектом разработки является мобильное приложение для осуществления автоматизированного управления магазином электроники.

Цель работы – разработка мобильного приложения для автоматизации процессов функционирования магазина электроники, в котором клиенты могут искать нужный товар, оформлять заказы, оставлять отзывы и обращаться в поддержку, а персонал магазина может управлять информационной системой магазина.

Разработка серверной части проводилась на языке программирования C# 14.0. В качестве среды разработки использовался редактор кода Visual Code. Свободно распространяемое решение PostgreSQL 18 выбрано в качестве СУБД.

Разработка клиентской части осуществлялась на языке программирования Kotlin. В качестве интегрированной среды разработки использовалась IDE Android Studio Panda 3 (версия для Ubuntu Desktop 24.04). Мобильное приложение предназначено для использования под управлением ОС Android 7 и выше, серверное кроссплатформенное приложение работает под управлением дистрибутива Ubuntu Server 24.04 и выше. Основные термины, используемые в пояснительной записке приведены в приложении А.

					ПГУ 09.03.04 – 08БР221.03 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Программное обеспечение автоматизированной системы управления магазином электроники. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Барабанов А.В.					4	145
Пров.		Князев В.Н.						
Н. контр.		Такташкин Д.В.				Группа 22ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и требований.....	9
1.1 Анализ предметной области и постановка задачи	9
1.2 Анализ программных продуктов.....	10
1.2.1 Интернет-магазин радиодеталей и электроники «Диполь»	10
1.2.2 Интернет-магазин «Электрорадиодетали».....	11
1.2.3 Интернет-магазин «Вольтмастер».....	12
1.2.4 Сравнение аналогов	13
1.3 Анализ целевой аудитории	14
1.4 Анализ функциональных требований	17
1.5 Анализ нефункциональных требований	25
1.6 Планирование разработки и оценка бюджета.....	26
2 Проектирование программного обеспечения.....	41
2.1 Выбор технологий и инструментальных средств.....	41
2.2 Проектирование архитектуры программного обеспечения	43
2.3 Проектирование базы данных	44
2.3.1 Концептуальное проектирование базы данных	44
2.3.2 Логическое проектирование базы данных	46
2.4 Проектирование серверной части приложения	47
2.5 Проектирование интерфейса мобильного приложения.....	49
3 Реализация программного обеспечения	65
3.1 Разработка схем базы данных.....	65
3.2 Разработка серверной части приложения.....	73
3.3 Разработка клиентской части приложения.....	77
4 Тестирование программного продукта	82
4.1 Функциональное тестирование. Метод черного ящика.....	82
4.2 Модульное тестирование	84
4.3 Метрики кода.....	87

4.3.1 Метрики размера программы	88
4.3.2 Метрики Холстеда.....	89
Заключение.....	94
Список использованных источников.....	96
Приложение А (справочное) Глоссарий	100
Приложение Б (рекомендуемое) Диаграммы классов	103
Приложение В (обязательное) Исходный код серверной части приложения ...	106
Приложение Г (обязательное) Исходный код клиентской части приложения..	120
Приложение Д (рекомендуемое) Результаты тестирования.....	127
Приложение Е (рекомендуемое) Исходный код модульных тестов	133

Введение

Информационные системы и средства автоматизации все больше проникают в самые различные сферы жизни общества. Это является естественным процессом и составляет фундаментальный вектор развития современной цивилизации. В наши дни цифровизация охватывает финансовую сферу, производство, здравоохранение, социальную сферу и государственное управление. В том числе сегодня невозможно представить быт и повседневную деятельность отдельного человека без применения каких-либо средств автоматизации. В докладе о цифровизации России по результатам 2025 года одним из ключевых выводов является то, что главной конкуренцией XXI века является конкуренция не за ресурсы, а за доверие и скорость обработки информации.

Важность развития сферы информационных технологий неоднократно отмечал и Президент Российской Федерации. Например, в указе № 203 от 9 мая 2017 года «О стратегии развития информационного общества...» Президент ставит конкретные цели развития информационных технологий на 13 лет вперед [1].

Указ Президента Российской Федерации № 309 от 7 мая 2024 года «О национальных целях развития Российской Федерации на период до 2030 года...» устанавливает цель «цифровой трансформации ... экономики и социальной сферы» как одну из важнейших целей национального развития Российской Федерации [2].

Неотъемлемой частью любой информационной системы являются оконечные устройства, которые предоставляют удобный и полнофункциональный интерфейс пользователю. В современных условиях развития техники особое место в этой группе занимают мобильные устройства. Если раньше для доступа к информационным ресурсам необходим был громоздкий стационарный компьютер, то сейчас почти каждый может использовать ресурсы сети Интернет с миниатюрного устройства размером с

ладонь. По данным Бюро цифровых проектов по состоянию на февраль 2026 года около 97% россиян используют смартфоны как основное средство связи [3]. Очевидно, что при организации бизнеса в условиях цифровизации необходимо учитывать необходимость распространения своих услуг и продукции в том числе и посредством мобильного программного обеспечения.

Продажа отечественной электроники и ее компонентов пользуется спросом в России. Это обуславливается переориентацией на новые логистические цепочки как производителей техники, так и компаний сектора услуг. Использование мобильного приложения под управлением операционной системы Android призвано автоматизировать ведение бизнеса по продаже электроники. Это упрощает администрирование магазина, модерацию общедоступного контента, а также делает более удобным оформление и выполнение заказов клиентов.

Таким образом из вышесказанного следует, что данный проект по разработке программного обеспечения для автоматизированной системы управления магазином электроники является актуальным и перспективным.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Разрабатываемое современное мобильное приложение должно предоставлять удобный и полнофункциональный интерфейс для всех пользователей магазина электроники и обеспечивать стабильную работу на всех совместимых устройствах. Мобильное приложение должно позволять пользователям просматривать каталог в удобном структурированном виде, а также предлагать загрузку структурированного файла с перечнем электронных компонентов, который представляет собой список необходимых товаров для покупки. Также необходимо организовать систему обратной связи с менеджерами-консультантами через простой встроенный чат. Поскольку все операции в системе имеют реальную экономическую составляющую, то необходимо также обеспечить административную часть магазина надежными и удобными средствами управления автоматизированной системой.

Таким образом, в создаваемом приложении должен быть реализован следующий основной функционал:

- регистрация пользователя;
- вход пользователя в систему;
- просмотр каталога электроники;
- загрузка перечня необходимых элементов;
- оформление/отмена заказа;
- отправка вопросов менеджеру магазина;
- просмотр истории заказов;
- оставить отзыв;
- добавить товар и изменить информацию о нем;
- завершить сеанс (выйти из аккаунта).

Для осуществления безопасного управления системой кроме обычных пользователей должны присутствовать менеджер, модератор и администратор.

Менеджер отвечает на вопросы клиентов, обрабатывает заказы и собирает статистику по продажам.

Модератор контролирует отзывы и блокировку пользователей, а администратор управляет назначением ролей и наследует функции модератора и менеджера, а также управляет товарами (добавляет и редактирует информацию).

1.2 Анализ программных продуктов

На российском рынке существует множество интернет-магазинов для покупки электроники. Краткая характеристика наиболее популярных из них представлена ниже.

1.2.1 Интернет-магазин радиодеталей и электроники «Диполь»

«Диполь» – одна из самых известных компаний в Пензенской области по продаже радиодеталей и электроники. Была создана в 2002 году. На официальном сайте магазина сообщается, что компания осуществляет продажу электроники дистанционным способом [4].

Ниже представлена главная страница интернет-магазина радиодеталей «Диполь» (рисунок 1).

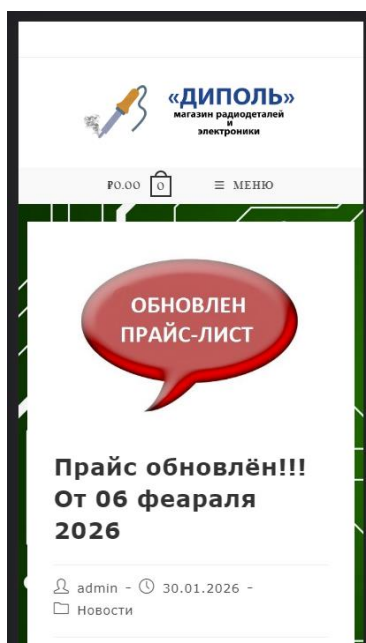


Рисунок 1 – Главная страница интернет-магазина «Диполь»

К плюсам необходимо отнести широкий ассортимент товаров электроники и длительное нахождение на рынке.

Из недостатков необходимо отметить отсутствие веб-каталога, удобных средств для взаимодействия с поддержкой, а также отсутствие мобильного приложения. Весь каталог товаров необходимо вручную скачивать в виде таблиц и искать нужные позиции самому.

1.2.3 Интернет-магазин «Вольтмастер»

«Вольтмастер» — это магазин профессионального оборудования и электронных компонентов, предоставляющий обширный каталог товаров. Магазин ориентирован на развитую систему поддержки клиентов и эффективную обратную связь. Компания работает как с физическими, так и с юридическими лицами [6].

Главная страница поиска товаров представлена на рисунке 3.

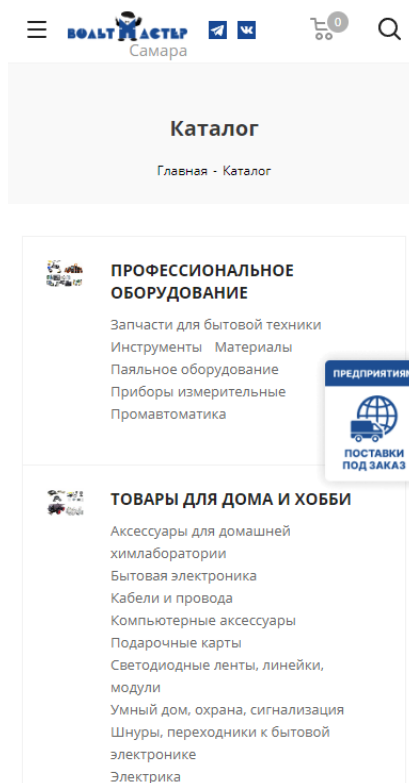


Рисунок 3 – Главная страница интернет-магазина «Вольтмастер»

К достоинствам приложения относится высокая адаптивность веб-интерфейса при отсутствии отдельной мобильной версии сайта, что является очень эффективной моделью разработки. Каталог электроники хорошо структурирован и является удобным для поиска товаров на всех устройствах, которые поддерживают веб-браузер.

Недостатком является отсутствие мобильного приложения и узкая направленность на торговлю профессиональной электроникой, что потенциально снижает торговые охваты и популярность магазина.

1.2.4 Сравнение аналогов

По приведенным выше характеристикам был проведен сравнительный анализ особенностей каждого интернет-магазина. Результаты приведены в таблице 1.

Таблица 1 – Сравнение аналогов

Интернет-магазин	Удобство интерфейса	Наличие мобильного приложения	Поддержка безопасного соединения	Наличие встроенного чата поддержки
Диполь	Ограниченный	Нет	Есть	Нет
Электрорадиодетали	Устаревший, неадаптивный	Нет	Нет	Нет
Вольтмастер	Интуитивно-понятный	Нет	Есть	Нет
Разрабатываемое приложение	Интуитивно-понятный	Есть	Есть	Есть

Анализ существующих программных продуктов выявил у них ряд ограничений и недостатков, в связи с чем было принято решение о разработке нового приложения, учитывающего выявленные проблемы и предлагающего способы их решения.

1.3 Анализ целевой аудитории

В настоящее время спрос на производство, транспортировку и продажу электроники остается высоким в условиях повсеместной автоматизации и цифровизации. Целевой аудиторией выступают как производители электроники и радиолюбители, так и рядовой потребитель.

Пресс-центр АО «Промтехкомплект» отмечает, что по состоянию на начало 2026 года доля электронных компонентов в структуре отечественного рынка занимает всего 20 – 30%. Однако государство активно стимулирует локализацию и развитие собственной электроники, так как сектор является стратегически важным. Санкции и ограничения на поставки западных технологий привели к увеличению спроса на отечественные продукты внутри страны. Тенденции развития: сокращение импортозависимости, локализация производства, государственное субсидирование сектора электроники и микроэлектроники [7].

«Интерфакс» со ссылкой на статистику предприятий предоставляет информацию об объемах рынка российской электроники и дает прогнозы на 2026 год. При позитивном сценарии обороты российских поставщиков будут расти на 33% в год. График объемов рынка представлен на рисунке 4 [8].



Рисунок 4 – Объем рынка российской электроники

Android – доминирующая платформа среди мобильных операционных систем как в России, так и во всем мире. Она остается ключевой в распространении приложений, особенно в секторе e-commerce. К 2026 году ожидаются сотни миллиардов скачиваний Android-приложений с трендом на увеличение вовлеченности пользователей.

Ниже представлена статистика распространенности мобильных операционных систем на российском рынке на январь 2026 года (рисунок 5) [9].

Базовые данные

AI Анализ

Изменение

поиск по наименованию

Скачать

Объект	↑↓	%	↓	Место	↑↓
Android		72,23		1	
iOS		27,39		2	
Samsung		0,22		3	
Не известно		0,11		4	
KaiOS		0,02		5	

Рисунок 5 – Распределение мобильных ОС в России на январь 2026 года

Также растет и доля людей старшего поколения, которые активно используют мобильные приложения (рисунок 6) [10].

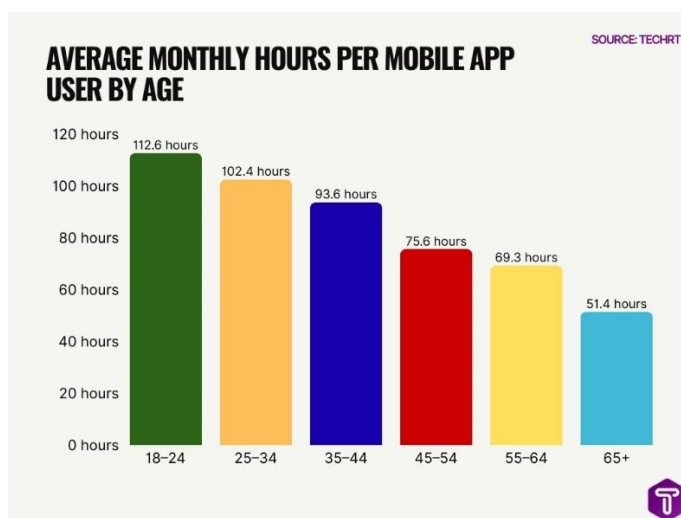


Рисунок 6 – Гистограмма экранного времени приложений

Также необходимо отметить, что в сложившейся экономической обстановке активно развивается импортозамещение в том числе и мобильных операционных систем [11].

Самые популярные из них:

- «Astra Linux Mobile» – один из самых распространенных аналогов Android, операционная система разработана на базе ядра Linux, выпускается в вариантах, сертифицированных ФСТЭК, ФСБ и МО РФ;
- «Ред ОС М» – одна из самых новых российских мобильных операционных систем, аналог Android, разработанная также на базе ядра Linux, на данный момент проходит тестирование;
- «Аврора» – семейство операционных систем, нацеленная на обеспечение безопасности и конфиденциальности, доступна только для юридических лиц.

Разработка под Android также подразумевает легкую поддержку отечественных мобильных операционных систем (рисунок 7).

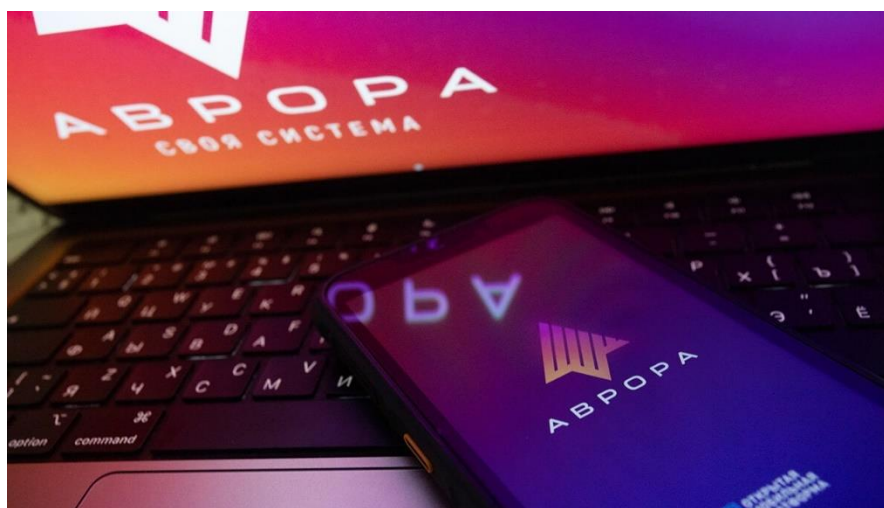


Рисунок 7 – Семейство ОС «Аврора»

Распределение конкретных версий операционной системы Android, установленных на мобильных устройствах, по состоянию на декабрь 2025 года представлено на рисунке 8 [12].



Рисунок 8 – Кумулятивное распределение версий Android

Приведенная выше статистика указывает на то, что разработка мобильного приложения для магазина электроники под платформу Android является обоснованной. Если исходить из графика распределения версий, то оптимальной является разработка с использованием следующих SDK: минимальная Android 7 (API 24), целевая Android 16 (API 36). Это позволит поддерживать 99,1% мобильных устройств с операционной системой Android, а также будут доступны к использованию все новейшие функции API 36.

1.4 Анализ функциональных требований

Пользователями разрабатываемого приложения являются: гость, зарегистрированный пользователь, менеджер, модератор и администратор. Гость – это неавторизованный пользователь, который имеет возможность войти в систему, зарегистрироваться и просматривать в мобильном приложении каталог электроники. Зарегистрированный пользователь наследует функционал гостя и дополнительно может управлять содержимым своей корзины, управлять заказами, отслеживать статус заказов, просматривать историю заказов. Также он имеет возможность задавать вопросы менеджеру магазина чате поддержки,

оставлять отзывы о купленной электронике и выходить из учетной записи с завершением сеанса в случае необходимости.

Менеджер и модератор наследуют функционал зарегистрированного пользователя (так как на самом деле являются зарегистрированными пользователями). Менеджер отвечает на вопросы клиентов, формирует отчет о продажах за определенный период и выполняет заказы пользователей. Модератор может блокировать недобросовестных пользователей, просматривать и фильтровать нецензурные отзывы о товарах.

Администратор является суперпользователем системы, поэтому он наследует весь функционал менеджера и модератора и дополнительно имеет права на управление ролями пользователей системы и редактирование (добавление) товаров в каталог, временное скрывание товаров из каталога при необходимости. Важно обеспечить строгое разделение прав пользователей согласно их роли в системе.

За корректное накопление, предоставление и упорядоченное хранение данных отвечает система управления базами данных (СУБД). Она должна обеспечивать целостность данных при добавлении, изменении данных, а также предоставлять средства механизма транзакций для выполнения критических операций с данными (оформление заказов, подсчет суммы заказа, запрос доступного количества товаров). Все варианты использования связаны с СУБД, так как она является полноценным участником системы.

Таким образом, на основании проведённого анализа функциональных требований, предъявляемых к разрабатываемому приложению, была определена совокупность основных функций и сценариев взаимодействия пользователей с системой. Для наглядного представления требований и моделирования поведения пользователей в среде Visual Paradigm Community Edition 18.0 была построена диаграмма вариантов использования. Созданная диаграмма представлена на рисунке 9 [13].

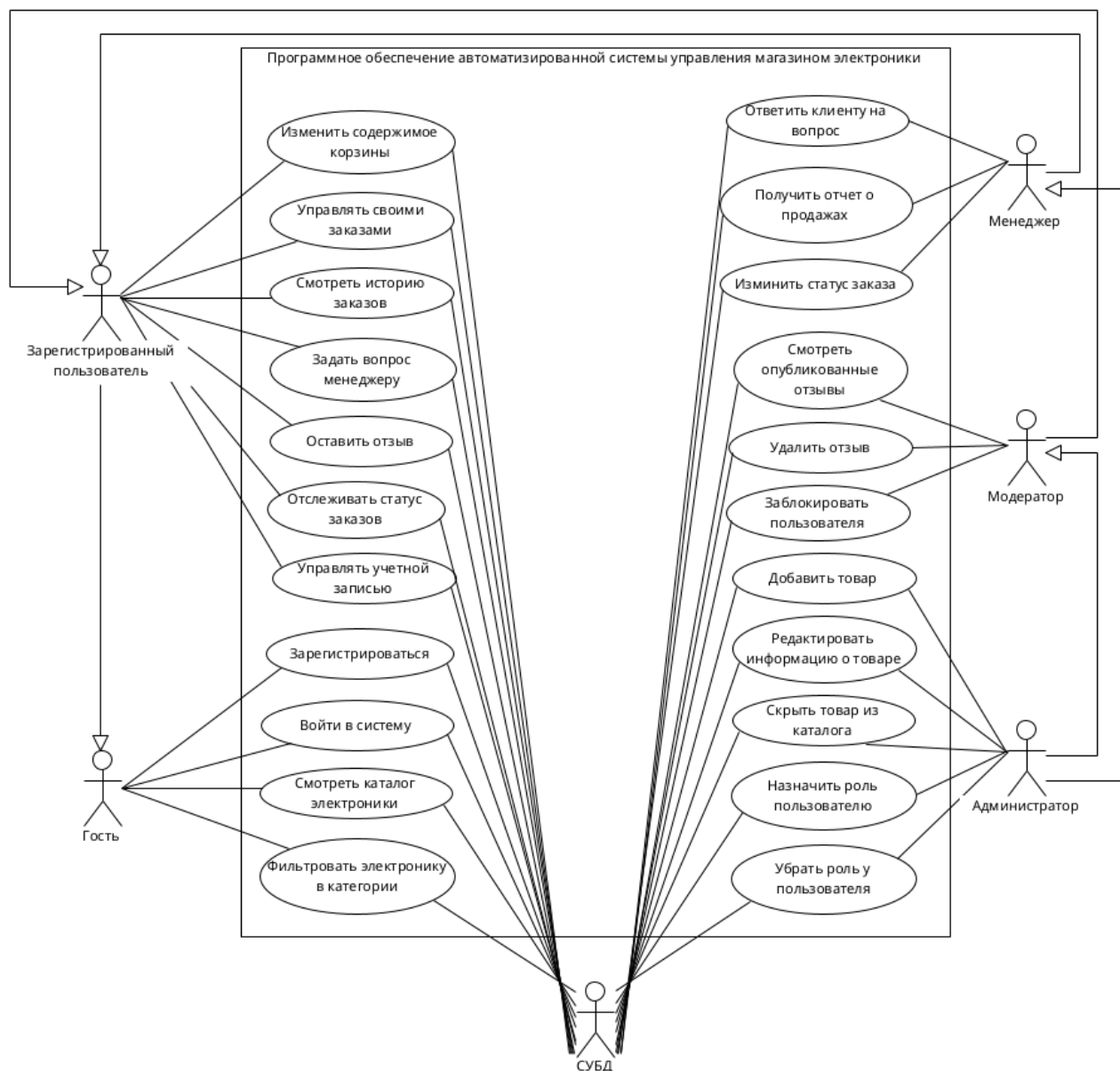


Рисунок 9 – Диаграмма вариантов использования мобильного приложения

Опишем спецификацию нескольких прецедентов для разных пользователей системы (гостя, зарегистрированного пользователя, менеджера, модератора, администратора). В качестве формы представления используем стандартную спецификацию прецедентов UML. В табличной форме описаны основные характеристики прецедентов.

Ниже представлены спецификации для прецедентов: «Зарегистрироваться» (рисунок 10), «Задать вопрос менеджеру» (рисунок 11).

Наименование: «Зарегистрироваться»
ID: 1
Краткое описание: гость регистрирует личную учетную запись в системе
Действующие лица: гость
Предусловие: 1. Гость не авторизован.
Основной поток: 1. Гость вводит необходимые данные в форме регистрации. 2. Отправляет запрос с введенными данными на регистрацию. 3. Если учетная запись может быть создана 3.1 СУБД создает новую запись пользователя в БД. 4. Система сообщает пользователю статус операции.
Постусловие: новая учетная запись создана в системе.

Рисунок 10 – Спецификация варианта использования «Зарегистрироваться»

Наименование: «Задать вопрос менеджеру»
ID: 8
Краткое описание: клиент запускает активность в виде встроенного мессенджера и задает вопрос менеджеру.
Действующие лица: зарегистрированный пользователь
Предусловие: 1. Зарегистрированный пользователь авторизован.
Основной поток: 1. Клиент открывает вкладку для общения с менеджером. 2. Вводит сообщение. 3. Подтверждает отправку сообщения. 4. СУБД сохраняет вопрос в БД в очередь на обработку менеджером. 5. Система отправляет подтверждение клиенту, что вопрос находится в обработке.
Постусловие: вопрос опубликован в очереди на обработку.

Рисунок 11 – Спецификация варианта использования «Задать вопрос менеджеру»

Также рассмотрим спецификации прецедентов «Добавить товар» (рисунок 12), «Изменить содержимое корзины» (рисунок 13).

Наименование: «Добавить товар»
ID: 11
Краткое описание: администратор добавляет полную информацию о новом товаре
Действующие лица: администратор
Предусловие: 1. Администратор авторизован.
Основной поток: 1. Администратор открывает раздел управления. 2. Вводит полную информацию о новом товаре. 3. Выбирает, будет ли опубликован товар в каталоге. 4. СУБД сохраняет новый товар в БД. 5. Система выводит сообщение о результате операции.
Постусловие: новый товар добавлен.

Рисунок 12 – Спецификация варианта использования «Добавить товар»

Наименование: «Изменить содержимое корзины»
ID: 4
Краткое описание: зарегистрированный пользователь редактирует количество товаров в каждой позиции корзины и добавляет/удаляет позиции в корзине с сохранением изменений на сервере.
Действующие лица: зарегистрированный пользователь
Предусловие: 1. Зарегистрированный пользователь авторизован.
Основной поток: 1. Зарегистрированный пользователь открывает корзину. 2. Добавляет, удаляет или изменяет позиции в заказе. 3. Подтверждает сохранение изменений. 4. Данные синхронизируются с сервером, СУБД сохраняет данные в БД.
Постусловие: измененное содержание корзины сохранено.

Рисунок 13 – Спецификация варианта использования «Изменить содержимое корзины»

Далее рассмотрим несколько спецификаций прецедентов для актера «Менеджер» (рисунки 14 – 15).

Наименование: «Изменить статус заказа»
ID: 14
Краткое описание: менеджер выбирает необходимый незавершенный заказ и изменяет его статус с сохранением изменений на сервере.
Действующие лица: менеджер
Предусловие: 1. Менеджер авторизован. 2. Заказ не отменен.
Основной поток: 1. Менеджер открывает вкладку с невыполненными заказами. 2. Менеджер выбирает нужный заказ. 3. Менеджер выбирает новый статус заказа. 3. Менеджер подтверждает сохранение изменений. 4. Данные синхронизируются с сервером, СУБД сохраняет данные в БД.
Постусловие: измененный статус заказа сохранен.

Рисунок 14 – Спецификация варианта использования «Изменить статус заказа»

Наименование: «Получить отчет о продажах»
ID: 13
Краткое описание: менеджер запрашивает с сервера структурированную информацию о статистике продаж за определенный период.
Действующие лица: менеджер
Предусловие: 1. Менеджер авторизован.
Основной поток: 1. Менеджер открывает вкладку с отчетом о продажах. 2. Менеджер выбирает нужный период отчетности. 3. Менеджер запрашивает структурированные данные о статистике продаж. 3. СУБД собирает необходимые данные. 4. Мобильное приложение строит график и выводит данные.
Постусловие: отчет о продажах предоставлен менеджеру в графической форме.

Рисунок 15 – Спецификация варианта использования «Получить отчет о продажах»

Дополнительно рассмотрим спецификацию для прецедента «Назначить роль пользователю» актера «Администратор» (рисунок 16).

Наименование: «Назначить роль пользователю»
ID: 17
Краткое описание: администратор назначает определенную роль пользователю
Действующие лица: администратор
Предусловие: 1. Администратор авторизован. 2. Целевой пользователь существует.
Основной поток: 1. Администратор открывает раздел управления системой. 2. Вводит идентификатор пользователя. 3. Выбирает роль, которую нужно назначить пользователю. 4. СУБД обновляет данные об учетной записи пользователя в БД.
Постусловие: роль назначена пользователю.

Рисунок 16 – Спецификация варианта использования «Назначить роль пользователю»

Для более подробного рассмотрения вариантов использования «Добавить товар» и «Изменить содержимое корзины» составлены диаграммы деятельности (рисунки 17 – 18).

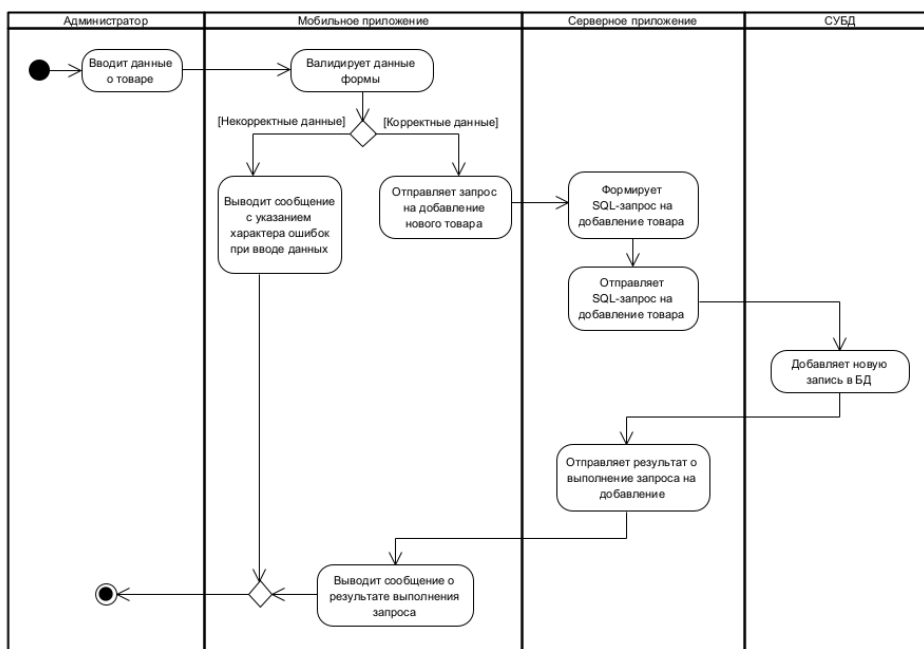


Рисунок 17 – Диаграмма деятельности для варианта использования «Добавить товар»

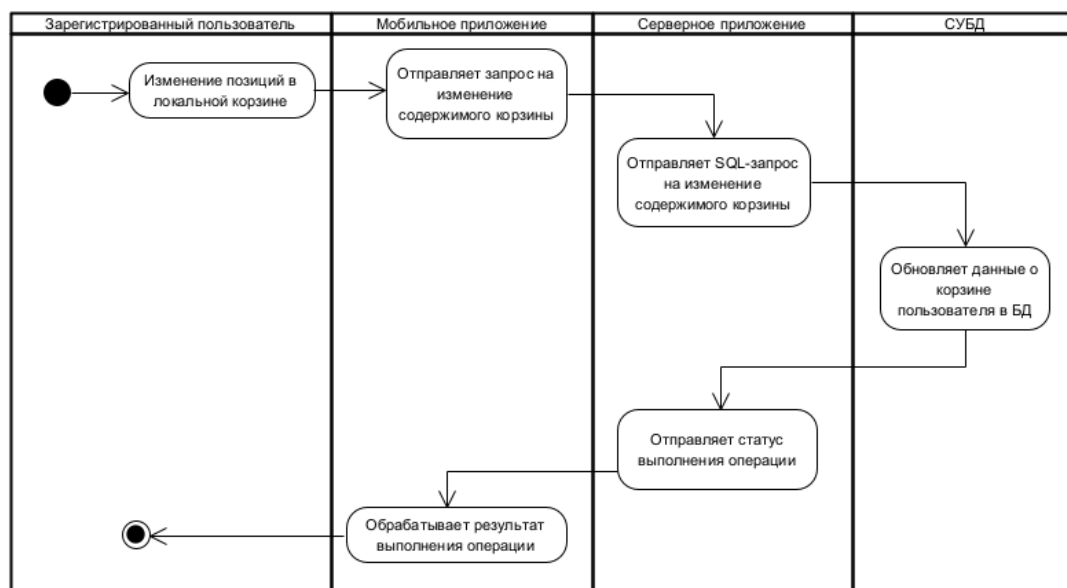


Рисунок 18 – Диаграмма деятельности для варианта использования «Изменить содержимое корзины»

Также для вариантов использования «Зарегистрироваться» и «Назначить роль пользователю» были созданы диаграммы последовательности (рисунки 19 – 20).

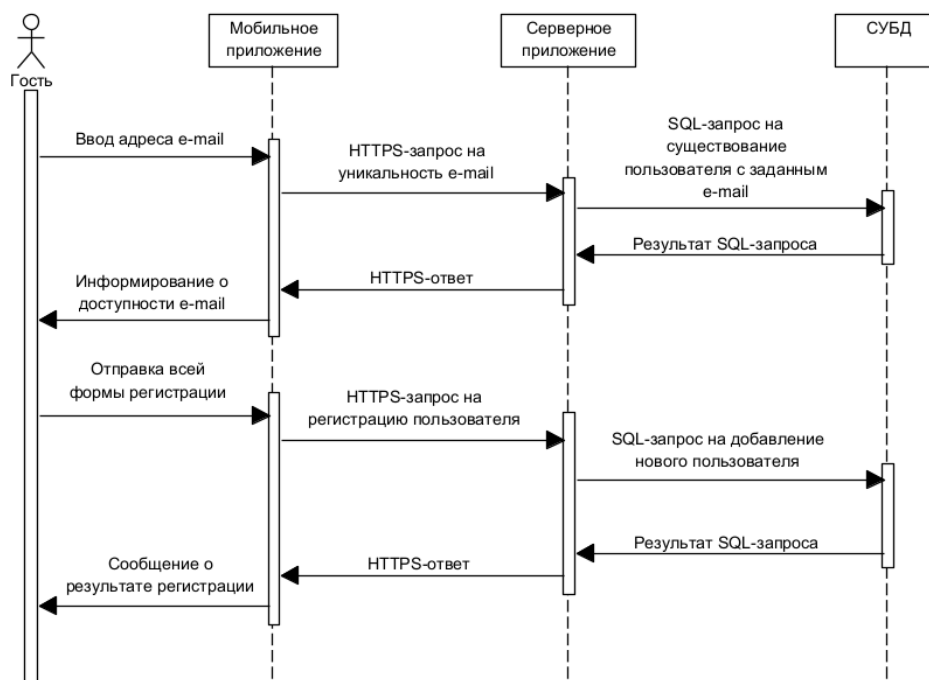


Рисунок 19 – Диаграмма последовательности для варианта использования «Зарегистрироваться»

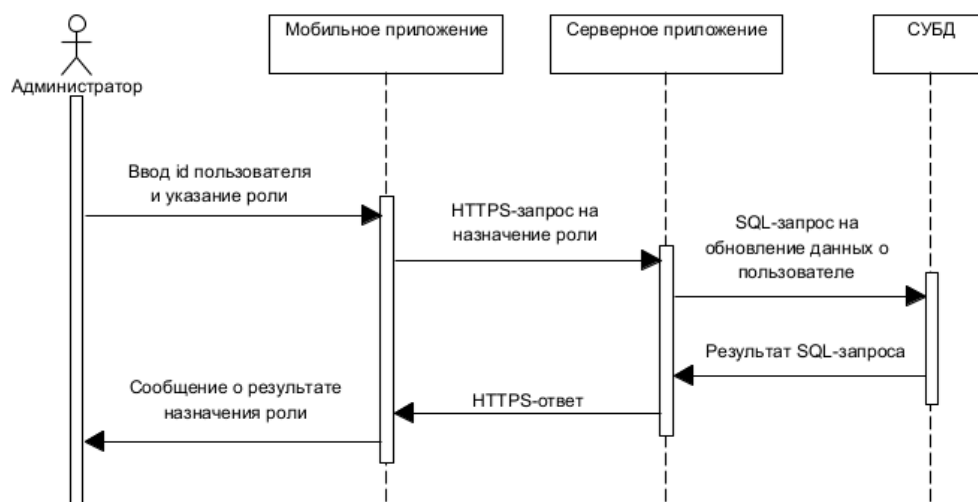


Рисунок 20 – Диаграмма последовательности для варианта использования «Назначить роль пользователю»

На данном этапе были определены основные функциональные возможности разрабатываемого мобильного приложения. Для формализации и наглядного представления требований к системе был разработан комплект UML-диаграмм, включающий диаграммы вариантов использования, деятельности и последовательности.

1.5 Анализ нефункциональных требований

При анализе требований были выделены следующие нефункциональные требования к разрабатываемому приложению: безопасность, масштабируемость, удобство пользования. В качестве шаблона проектирования был выбран шаблон MVVM. Методология разработки – RAD (Rapid Application Development).

Серверную часть планируется реализовать на языке C# 14.0 в редакторе кода Visual Code с использованием современной платформы ASP.NET Core. Кодовая база и серверная инфраструктура должны по необходимости обеспечивать возможность к быстрому и простому переходу на микросервисное развертывание, поэтому в качестве компромиссного решения была выбрана архитектура модульного монолита. Сервер должен обеспечивать связь по безопасному HTTPS соединению. Управление базой данных осуществляет СУБД

PostgreSQL, с которой серверная часть приложения взаимодействует посредством Entity Framework (входит в состав открытых пакетов NuGet). Серверная часть приложения должна быть отлажена и протестирована на дистрибутиве Ubuntu Server 24.04.

Клиентская часть приложения должна быть реализована на языке Kotlin с использованием инструментов Android SDK в IDE Android Studio. Мобильное приложение должно обладать неперегруженным графическим интерфейсом, понятным любому пользователю автоматизированной системы. Разрабатываемое приложение должно быть отлажено и протестировано для обеспечения стабильной работы на устройствах под управлением операционной системы Android 7 и выше.

1.6 Планирование разработки и оценка бюджета

Ключевым фактором успешной реализации любого проекта является тщательная и продуманная организация его этапов [14]. Планирование разработки программного обеспечения обеспечивает системное распределение задач и ресурсов на всех стадиях работы, что позволяет достигать поставленных целей в установленные сроки, оптимизируя затраты и соблюдая требования к качеству создаваемого продукта. Основные задачи проекта по разработке программного обеспечения приведены в таблице 2.

Таблица 2 – Задачи проекта по разработке ПО АСУ для магазина электроники

№	Название задачи	Длительность	Предшественники
1	Начало проекта по разработке ПО для магазина электроники	–	–
2	Анализ требований	9 дней	–
3	Анализ целевой аудитории	1 день	1
4	Анализ предметной области	6 дней	3
5	Анализ аналогичных интернет-магазинов	3 дня	3
6	Анализ функциональных требований	3 день	5
7	Анализ нефункциональных требований	2 дня	5

Продолжение таблицы 2

№	Название задачи	Длительность	Предшественники
8	Анализ требований к моб. интерфейсу	2 дня	5
9	Анализ методологий разработки ПО	1 дня	8;6;7
10	Выбор методологии разработки ПО	1 дня	9
11	Анализ требований завершен	-	10
12	Проектирование программных средств	8 дней	-
13	Обсуждение и выбор набор технологий и инструментов	2 дня	11
14	Проектирование базы данных	6 дней	13
15	Проектирование архитектуры серверной части приложения	5 дней	13
16	Проектирование архитектуры клиентской части приложения	6 дней	13
17	Моделирование дизайна приложения	5 дней	13
18	Проектирование приложения завершено	-	14;15;16;17
19	Реализация программных средств	20 дней	-
20	Реализация базы данных	4 дня	18
21	Реализация серверного приложения	16 дней	20
22	Реализация мобильного приложения	15 дней	20
23	Реализация завершена	-	21;22;
24	Тестирование программных средств	6 дней	-
25	Функциональное тестирование	2 дня	23
26	Модульное тестирование	2 дня	25
27	Нагрузочное тестирование	2 дня	26
28	Тестирование завершено	-	27
29	Оформление документации	5 дней	28
30	Окончание проекта по разработке	-	29

В процессе выполнения задач проекта используются трудовые, материальные ресурсы, а также затраты [15] на услуги Интернет-провайдера (таблица 3).

Таблица 3 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Backend-разработчик	Трудовой	600 руб./ч	1500 руб./ч
Android-разработчик	Трудовой	400 руб./ч	1200 руб./ч
Архитектор баз данных	Трудовой	800 руб./ч	1100 руб./ч
Аналитик	Трудовой	900 руб./ч	1500 руб./ч
Дизайнер	Трудовой	400 руб./ч	800 руб./ч
Тестировщик	Трудовой	300 руб./ч	550 руб./ч
Руководитель проекта	Трудовой	800 руб./ч	2100 руб./ч
Интернет	Затраты	2000 руб./месяц	–
Персональный компьютер	Материальный	4 шт.	–
Мобильное устройство	Материальный	1 шт.	–
Бумага	Материальный	2000 руб.	–

Распределение вышеперечисленных ресурсов по задачам проекта представлено в таблице 4.

Таблица 4 – Задачи проекта по разработки ПО для магазина электроники

№	Название задачи	Ресурс	Единицы
1	Начало проекта по разработке ПО для магазина электроники	–	–
2	Анализ требований	–	–
3	Анализ целевой аудитории	Руководитель проекта ПК	100% 1 шт.
4	Анализ предметной области	Руководитель проекта Аналитик ПК	100% 25% 2 шт.
5	Анализ аналогичных интернет-магазинов	Аналитик ПК	25% 1 шт.
6	Анализ функциональных требований	Аналитик ПК	25% 1 шт.

Продолжение таблицы 4

№	Название задачи	Ресурс	Единицы
7	Анализ нефункциональных требований	Аналитик ПК	25% 1 шт.
8	Анализ требований к моб. интерфейсу	Дизайнер ПК	100% 1 шт.
9	Анализ методологий разработки ПО	Руководитель проекта ПК	100% 1 шт.
10	Выбор методологии разработки ПО	Руководитель проекта Android-разработчик Backend-разработчик Аналитик	50% 50% 50% 50%
11	Анализ требований завершен	—	—
12	Проектирование программных средств	—	—
13	Обсуждение и выбор технологического стека	Руководитель проекта Android-разработчик Backend-разработчик Аналитик	50% 50% 50% 50%
14	Проектирование базы данных	Архитектор БД Руководитель проекта ПК	100% 25% 1 шт.
15	Проектирование архитектуры серверной части приложения	Backend-разработчик Руководитель проекта ПК	100% 25% 1 шт.
16	Проектирование архитектуры клиентской части приложения	Android-разработчик Руководитель проекта ПК	100% 25% 1 шт.
17	Моделирование дизайна приложения	Архитектор БД Руководитель проекта ПК	100% 25% 1 шт.
18	Проектирование приложения завершено	—	—
19	Реализация программных средств	—	—

Окончание таблицы 4

№	Название задачи	Ресурс	Единицы
20	Реализация базы данных	Архитектор БД Руководитель проекта ПК	100% 100% 1 шт.
21	Реализация серверного приложения	Backend-разработчик Руководитель проекта ПК	100% 50% 1 шт.
22	Реализация мобильного приложения	Android-разработчик Руководитель проекта ПК Мобильное устройство	100% 50% 1 шт. 1 шт.
23	Реализация завершена	—	—
24	Тестирование программных средств	—	—
25	Функциональное тестирование	Тестировщик ПК Мобильное устройство	100% 1 шт. 1 шт.
26	Модульное тестирование	Тестировщик ПК Мобильное устройство	100% 1 шт. 1 шт.
27	Нагрузочное тестирование	Тестировщик ПК Мобильное устройство	100% 1 шт. 1 шт.
28	Тестирование завершено	—	—
29	Оформление документации	Аналитик ПК	100% 1 шт.
30	Окончание проекта по разработке	—	—

Реализация проекта предполагает участие руководителя проекта, двух программистов, аналитика, дизайнера и тестировщика. Для обеспечения возможности одновременного выполнения работ требуется подготовить четыре персональных компьютера, одно мобильное устройство с доступом к

разрабатываемому приложению, подключение к сети Интернет, а также расходные материалы для оформления и печати проектной документации.

Для определения последовательности работ, оказывающей наибольшее влияние на сроки его завершения, был построен сетевой график. В качестве его узлов выступают задачи, сформированные на этапе планирования проекта. Построенный сетевой график представлен на рисунке 21.

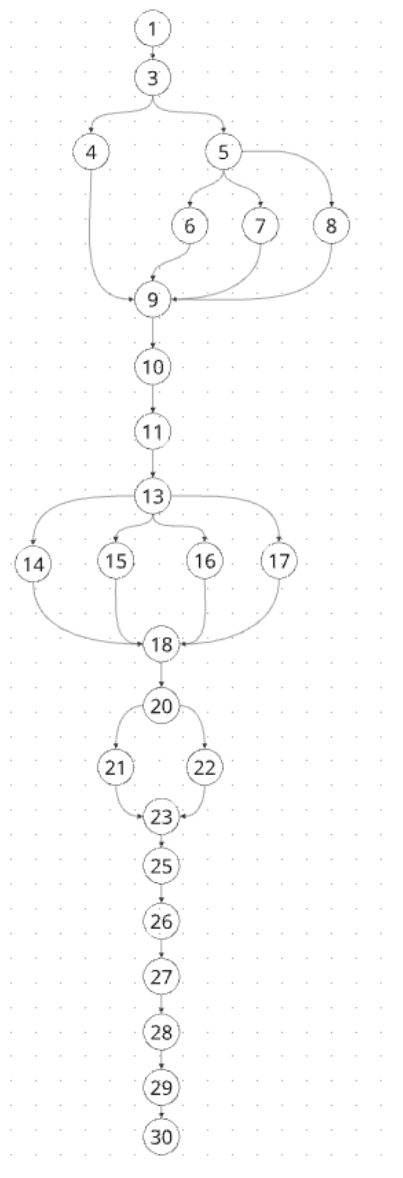


Рисунок 21 – Сетевой граф проекта

В табличном виде представлено нахождение необходимых параметров для нахождения критического пути проекта. Ниже приведен ручной расчет раннего времени начала работ для каждой задачи проекта (таблица 5).

Таблица 5 – Расчет раннего времени начала работ

Работа	Предшественники			Вычисление	Раннее время начала
	№	раннее время начала	длительность		
1	-	-	-	-	-
3	1	0	0	0+0	0
4	3	0	1	0+1	1
5	3	0	1	0+1	1
6	5	1	3	1+3	4
7	5	1	3	1+3	4
8	5	1	3	1+3	4
9	4 6 7 8	1 4 4 4	3 3 2 2	Max(1+3; 4+3; 4+2; 4+2)	7
10	9	7	1	7+1	8
11	10	8	1	8+1	9
13	11	9	0	9+0	9
14	13	9	2	9+2	11
15	13	9	2	9+2	11
16	13	9	2	9+2	11
17	13	9	2	9+2	11
18	14 15 16 17	11 11 11 11	6 5 6 5	Max(11+6; 11+5; 11+6; 11+5)	17
20	18	17	0	17+0	17
21	20	17	4	17+4	21
22	20	17	4	17+4	21
23	21 22	21 21	16 15	Max(21+16; 21+15)	37
25	23	37	0	37+0	37
26	25	37	2	37+2	39

Продолжение таблицы 5

Работа	Предшественники			Вычисление	Раннее время начала
	№	раннее время начала	длительность		
27	26	39	2	39+2	41
28	27	41	2	41+2	43
29	28	43	0	43+0	43
30	29	43	5	43+5	48

В таблице 6 приведен ручной расчет позднего времени начала работ для каждой задачи проекта.

Таблица 6 – Расчет позднего времени начала работ

Работа	Последователи			Вычисление	Позднее время начала
	№	позднее время начала	длительность		
30	-	-	-		48
29	30	48	0	48-0	48
28	29	48	5	48-5	43
27	28	43	0	43-0	43
26	27	43	2	43-2	41
25	26	41	2	41-2	39
23	25	39	2	39-2	37
21	23	37	0	37-0	37
22	23	37	0	37-0	37
20	21	37	16	Min(37-16;37-15)	21
	22	37	15		
18	20	21	4	21-4	17
14	18	17	0	17-0	17
15	18	17	0	17-0	17
16	18	17	0	17-0	17
17	18	17	0	17-0	17

Продолжение таблицы 6

Работа	Последователи			Вычисление	Позднее время начала
	№	позднее время начала	длительность		
13	14	17	6	Min(17-6;17-5;17-6;17-5)	11
	15	17	5		
	16	17	6		
	17	17	5		
11	13	11	2	11-2	9
10	11	9	0	9-0	9
9	10	9	1	9-1	8
4	9	8	1	8-1	7
6	9	8	1	8-1	7
7	9	8	1	8-1	7
8	9	8	1	8-1	7
5	6	7	3	Min(7-2;7-3)	4
	7	7	2		
	8	7	2		
3	4	7	6	Min(7-6;4-3)	1
	5	4	3		
1	3	1	1	1-1	0

Для вычисления времени раннего начала работ была использована формула:

$$T_P(i) = \max_{j \in G} (T_P(j) + t_j),$$

где $T_P(i)$ – раннее время начала работы i ;

$T_P(j)$ – раннее время начала работы j , предшествующей работе i ;

t_j – время выполнения работы j ;

G – множество работ, непосредственно предшествующих работе i .

Для вычисления времени позднего начала работ была использована формула:

$$T_{\Pi}(i) = \min_{j \in H} (T_{\Pi}(j) - t_i),$$

где $T_{\Pi}(i)$ – позднее время начала работы i ;

$T_{\Pi}(j)$ – позднее время начала работы j , следующих за работой i ;

t_i – время выполнения работы i ;

N – множество работ, непосредственно следующих за работой i .

Критический путь – это определенная последовательность задач проекта, которая определяет дату окончания реализации проекта согласно плана [17]. Задержка выполнения задач критического пути приведет к срыву сроков всего проекта. Расчет последовательности представлен в таблице 7.

Таблица 7 – Вычисление критического пути

Номер работы	Раннее время начала	Позднее время начала	Резерв времени
1	0	0	0
3	0	1	1
4	1	7	6
5	1	4	3
6	4	7	3
7	4	7	3
8	4	7	3
9	7	8	1
10	8	9	1
11	9	9	0
13	9	11	2
14	11	17	6
15	11	17	6
16	11	17	6
17	11	17	6
18	17	17	0
20	17	21	4
21	21	37	16
22	21	37	16
23	37	37	0
25	37	39	2

Продолжение таблицы 7

Номер работы	Раннее время начала	Позднее время начала	Резерв времени
26	39	41	3
27	41	43	2
28	43	43	0
29	43	48	5
30	48	48	0

По полученным расчетным данным построим критический путь проекта. Этот путь будет проходить через узлы графа в следующей последовательности номеров задач: 1, 3, 5, 6, 9, 10, 11, 13, 14, 18, 20, 21, 23, 25, 26, 27, 28, 29, 30. Критический путь определяет набор критических задач, выполнение которых не может быть отложено без нарушения сроков реализации проекта. Поэтому задачи, находящиеся на критическом пути, также являются критическими. Сетевой граф с изображением критического пути пунктирными линиями представлен на рисунке 22.

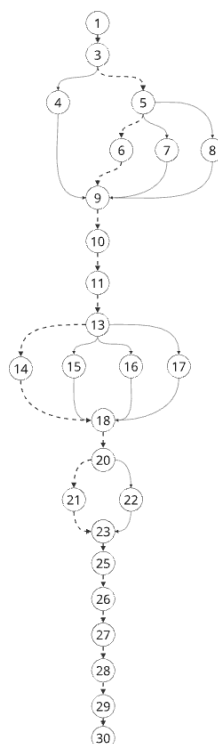


Рисунок 22 – Критический путь проекта

Для формирования календарного плана проекта, оценки требуемого бюджета и последующего документирования использовалась программа Microsoft Project 2007. Данное программное средство позволяет рассчитывать сроки выполнения работ и затраты на основе сведений о задачах проекта, задействованных ресурсах и их стоимости. На основании перечня работ, представленного в таблице 4, в Microsoft Project был сформирован проектный план. Результат переноса задач в программную среду представлен на рисунке 23.

	И	Название задачи	Длительность	Начало	Окончание	Предшест	Названия ресурсов
1		Начало проекта по разработке приложения	0 дней	Пт 20.02.26	Пт 20.02.26		
2		Анализ требований	9 дней	Пт 20.02.26	Ср 04.03.26		Интернет[700,00 Р]
3		Анализ целевой аудитории	1 день	Пт 20.02.26	Пт 20.02.26	1	Руководитель проекта
4		Анализ предметной области	6 дней	Пн 23.02.26	Пн 02.03.26	3	Руководитель проекта;Аналитик[25%]
5		Анализ аналогичных интернет-магазинов	3 дней	Пн 23.02.26	Ср 25.02.26	3	Аналитик[25%]
6		Анализ функциональных требований	3 дней	Чт 26.02.26	Пн 02.03.26	5	Аналитик[25%]
7		Анализ нефункциональных требований	2 дней	Чт 26.02.26	Пт 27.02.26	5	Аналитик[25%]
8		Анализ требований к мобильному интерфейсу	2 дней	Чт 26.02.26	Пт 27.02.26	5	Дизайнер
9		Анализ методологий разработки ПО	1 день	Вт 03.03.26	Вт 03.03.26	8,6,7,4	Руководитель проекта
10		Выбор методологии разработки	1 день	Ср 04.03.26	Ср 04.03.26	9	Android-разработчик[50%];Backend-раз
11		Анализ требований завершен	0 дней	Ср 04.03.26	Ср 04.03.26	10	
12		Проектирование программных средств	8 дней	Чт 05.03.26	Пн 16.03.26		Интернет[1 470,00 Р]
13		Обсуждение и выбор технологического стека	2 дней	Чт 05.03.26	Пт 06.03.26	11	Android-разработчик[50%];Backend-раз
14		Проектирование структуры базы данных	6 дней	Пн 09.03.26	Пн 16.03.26	13	Архитектор баз данных;Руководитель
15		Проектирование серверной архитектуры	5 дней	Пн 09.03.26	Пт 13.03.26	13	Backend-разработчик;Руководитель пр
16		Проектирование клиентской архитектуры	6 дней	Пн 09.03.26	Пн 16.03.26	13	Android-разработчик;Руководитель про
17		Моделирование дизайна приложения	5 дней	Пн 09.03.26	Пт 13.03.26	13	Дизайнер;Руководитель проекта[25%]
18		Проектирование ПО завершено	0 дней	Пн 16.03.26	Пн 16.03.26	17;14;15;1	
19		Реализация программных средств	20 дней	Вт 17.03.26	Пн 13.04.26		Интернет[490,00 Р]
20		Реализация базы данных	4 дней	Вт 17.03.26	Пт 20.03.26	18	Архитектор баз данных;Руководитель
21		Реализация серверного приложения	16 дней	Пн 23.03.26	Пн 13.04.26	20	Backend-разработчик;Руководитель пр
22		Реализация мобильного приложения	15 дней	Пн 23.03.26	Пт 10.04.26	20	Android-разработчик;Руководитель про
23		Релизация завершена	0 дней	Пн 13.04.26	Пн 13.04.26	21;22	
24		Тестирование программных средств	6 дней	Вт 14.04.26	Вт 21.04.26		Интернет[560,00 Р]
25		Функциональное тестирование	2 дней	Вт 14.04.26	Ср 15.04.26	23	Тестировщик
26		Модульное тестирование	2 дней	Чт 16.04.26	Пт 17.04.26	25	Тестировщик
27		Нагрузочное тестирование	2 дней	Пн 20.04.26	Вт 21.04.26	26	Тестировщик
28		Тестирование завершено	0 дней	Вт 21.04.26	Вт 21.04.26	27	
29		Оформление документации	5 дней	Ср 22.04.26	Вт 28.04.26	28	Бумага[1 шт.];Аналитик
30		Окончание проекта по разработке	0 дней	Вт 28.04.26	Вт 28.04.26	29	

Рисунок 23 – Список задач проекта в MS Office Project 2007

Таблица с задачами имеет иерархическое разделение задач на подзадачи и является удобным средством упорядочивания данных. Однако в среде управления проектами есть более удобная форма представления перечня задач в виде диаграммы Ганта, которая особенно необходима для отображения и анализа параллельно выполняющихся задач. Ниже представлена диаграмма Ганта [18], автоматически составленная по таблице задач в среде Microsoft Office Project (рисунок 24).

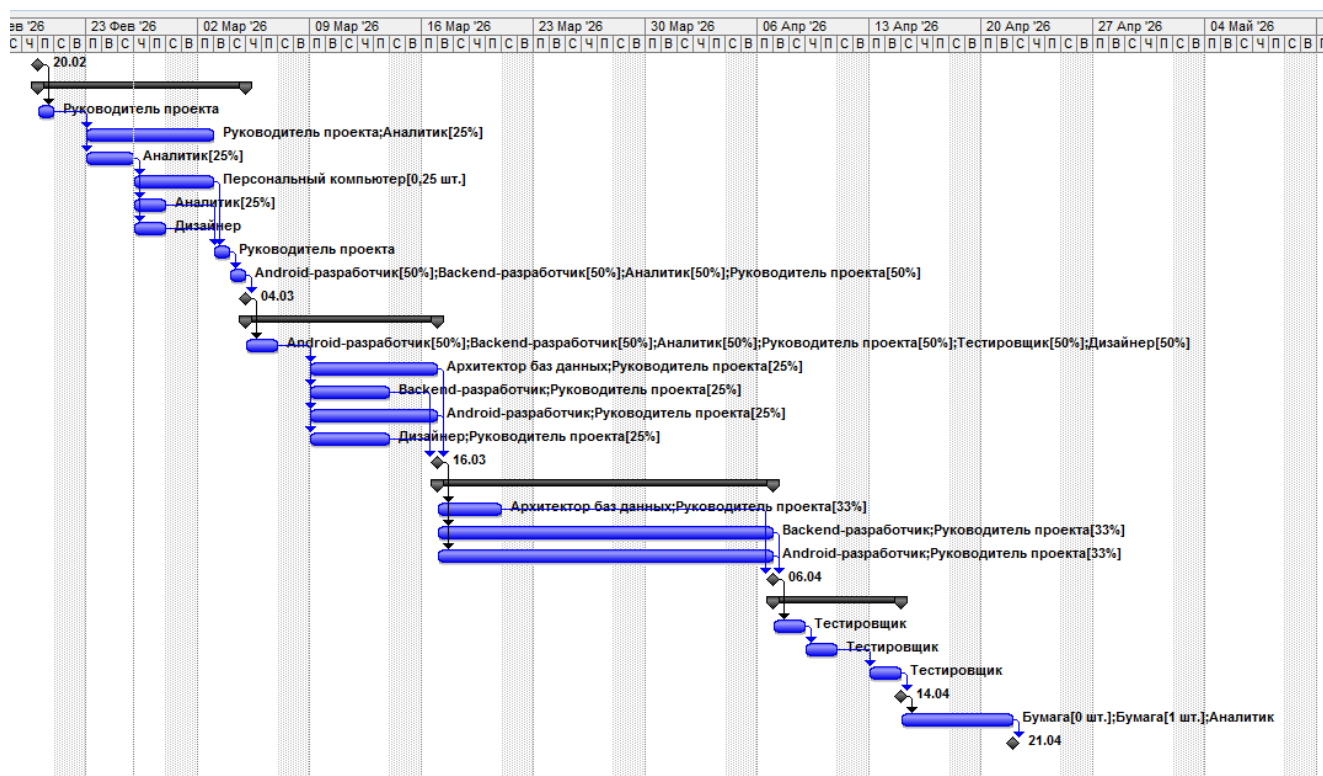


Рисунок 24 – Диаграмма Ганта

Ниже на рисунках 25 – 26 представлены графики использования трудовых ресурсов (backend-разработчика и android-разработчика).

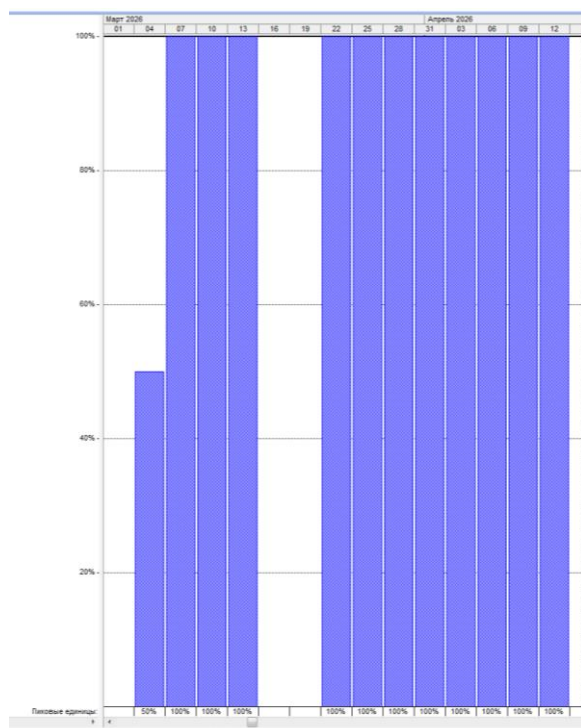


Рисунок 25 – График использования трудового ресурса «Backend-разработчик»

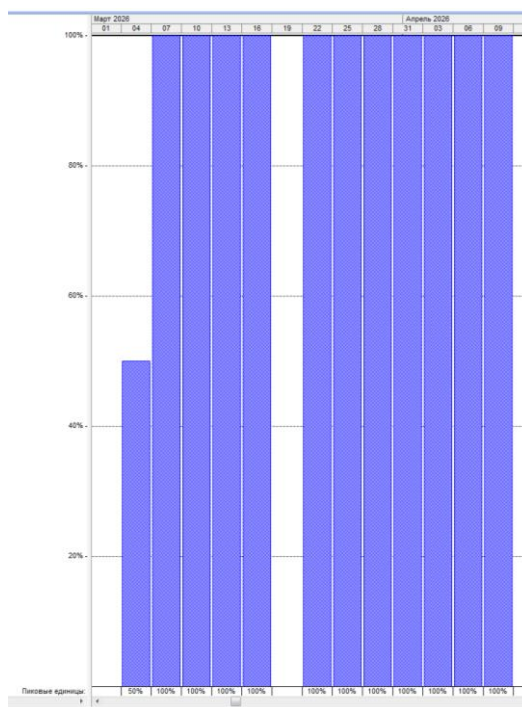


Рисунок 26 – График использования трудового ресурса «Android-разработчик»

Общая длительность проекта составляет 48 дней. За дату начала разработки выбрано 20 февраля 2026 года. Стоимость реализации – 567 353,33 руб. Сводка по проекту представлена на рисунке 27.

Даты			
Начало:	Пт 20.02.26	Окончание:	Вт 28.04.26
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	48 дней	Оставшаяся:	48 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	48 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	898,67 ч	Оставшиеся:	898,67 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	898,67 ч	Процент завершения:	0%
Затраты			
Текущий план:	567 353,33 Р	Оставшиеся:	567 353,33 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	567 353,33 Р		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	30	Трудовые ресурсы:	8
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дос:	0
Завершенные задачи:	0	Материальные ресурсы:	3
Итого задач:	30	Итого ресурсов:	11

Рисунок 27 – Сводка по проекту

Проведённое планирование показало, что разработка программного обеспечения для автоматизированной системы магазина электроники является экономически обоснованной и может быть реализована с имеющимися ресурсами в установленные сроки.

2 Проектирование программного обеспечения

2.1 Выбор технологий и инструментальных средств

Для создания мобильного приложения было принято решение использовать методологию разработки Rapid Application Development (RAD) – быстрая разработка приложений. Она является одной из самых популярных итеративных методологий разработки программного обеспечения. Ее особенностью является максимально выгодное сочетание скорости, качества и стоимости разработки программного продукта [19].

Методология подразумевает использование итераций для создания прототипов ПО. Причем прототип каждой последующей итерации включает в себя функционал прототипа итерации предыдущей. Поводами для очередной итерации могут выступать:

- замечания заказчика;
- детализация (программирование или прототипирование ранее нереализованного функционала);
- анализ результатов программирования (повышение эффективности, исправление ошибок).

Данная методология была выбрана благодаря ее преимуществам, которые безусловно важны для настоящего проекта. Это обусловлено сроками разработки (несколько месяцев), постоянное взаимодействие со стороной, принимающей проект, размером штата команды разработчиков (три-четыре специалиста), отсутствием особой вычислительной сложности и жестких требований к надежности и отказоустойчивости разрабатываемого программного обеспечения. Также, согласно методологии, необходимо использовать средства автоматизации сборки, разработки и тестирования для ускорения создания прототипов. К ним относятся: Docker, ассистент Copilot, пакет xUnit.

В процессе разработки по методологии RAD необходимо уделять внимание моделированию. Для этой задачи был выбран язык UML. Оптимальной и современной средой разработки UML-диаграмм является Visual Paradigm 18.0.

Для составления плана проекта, распределения трудовых и материальных ресурсов по задачам, составления сметы проекта решено использовать Microsoft Project. Данное программное обеспечение является многофункциональным и позволяет легко выполнять задачи по планированию проекта по разработке программного обеспечения. Также есть встроенные средства визуализации (диаграмма Ганта, графики использования ресурсов).

Разработка серверной части будет вестись на языке программирования C# с использованием платформы ASP.NET в редакторе кода Visual Code с установленными расширениями редактора для работы с исходным кодом C#. В качестве протокола взаимодействия с клиентской частью приложения предполагается использовать легковесный REST API, который отлично подходит для мобильных приложений, так как клиенту нужно передавать только необходимые данные, а не весь интерфейс целиком (как это происходит в веб-приложениях). Для организации хранения, обработки, накопления информации принято решение использовать реляционные базы данных. В качестве системы управления базами данных предполагается использовать систему с открытым исходным кодом – PostgreSQL 18. Данная СУБД распространяется по свободной лицензии PostgreSQL License. Проектирование базы данных удобно осуществить в среде Visual Paradigm.

Разработка клиентской части приложения будет осуществлена на языке Kotlin с использованием платформы Jetpack Compose. Подходящей интегрированной средой разработки для данной задачи является Android Studio, которую официально поддерживает Google. Выбранный паттерн проектирования клиентской части мобильного приложения – Model-View-ViewModel (MVVM). Паттерн MVVM призван организовать удобный, богатый и отзывчивый пользовательский интерфейс, который соответствует современным стандартам. Выбор инструментов для Android-разработки подразумевает легкий перенос мобильного приложения на все другие платформы.

2.2 Проектирование архитектуры программного обеспечения

Структурное разделение программного обеспечения автоматизированной системы магазина электроники предполагается произвести по классическому принципу трехуровневой архитектуры «клиент-сервер» (рисунок 28). Тогда клиент – это разрабатываемое приложение для Android, а сервер – приложение ASP.NET, сервер базы данных и сопутствующая серверная инфраструктура [20].

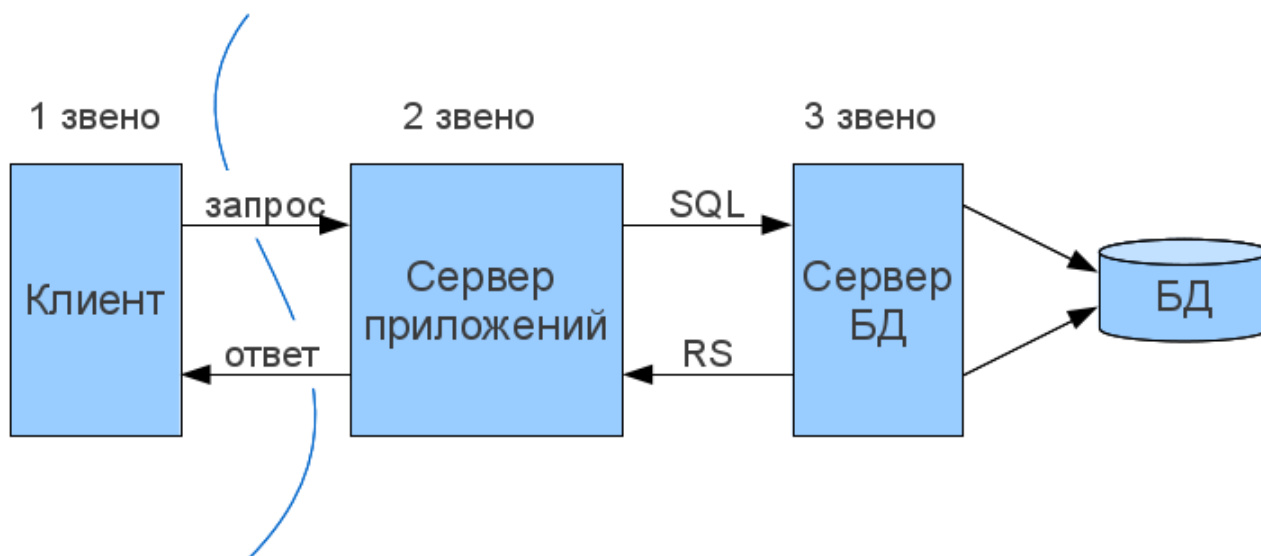


Рисунок 28 – Архитектура «клиент-сервер»

Было принято решение выделить следующие компоненты в выбранном типе архитектуры:

- клиент – программное обеспечение, создающее запросы к серверу (браузер, Android-приложение, специальное устройство);
- сеть (инфраструктура) – вторичная сеть, предназначенная для передачи данных (локальная сеть, сеть Интернет);
- сервер – совокупность аппаратного и программного обеспечения для обработки запросов от клиентов (ASP.NET приложение, СУБД PostgreSQL).

Выбор архитектуры «клиент-сервер» обоснован, так как она является относительно простотой (тривиальной) при разработке автоматизированных систем для сектора услуг (магазинов).

2.3 Проектирование базы данных

2.3.1 Концептуальное проектирование базы данных

С использованием средств Visual Paradigm 18.0 были созданы модели базы данных. На основании анализа предметной области была составлена концептуальная модель [21]. Она отражает информационные атрибуты сущностей и связи между ними. В графическом виде модель представлена на рисунке 29.

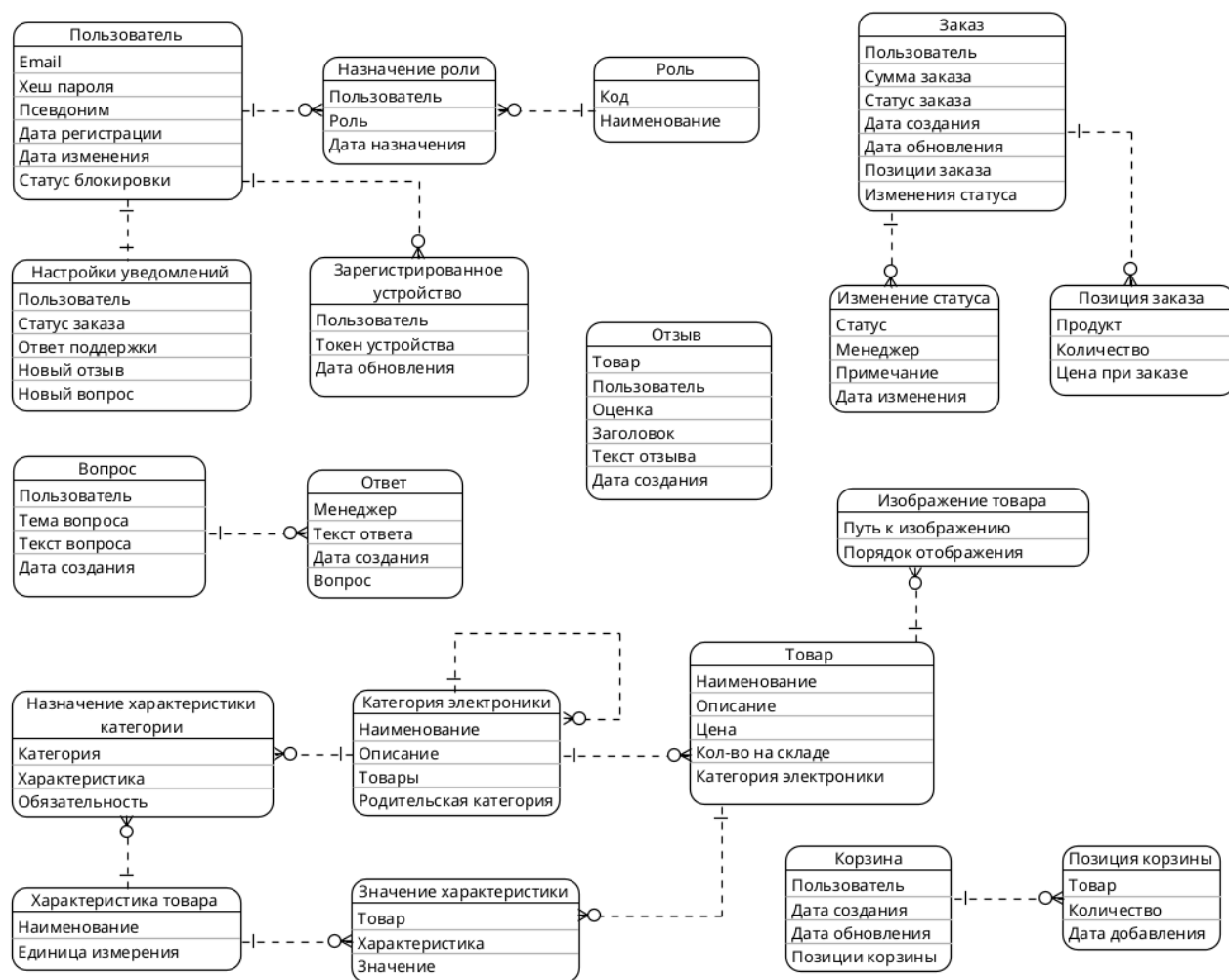


Рисунок 29 – Концептуальная модель базы данных

Основные характеристики и краткое описание сущностей описаны в таблице 8.

Таблица 8 – Сущности концептуальной модели базы данных

№	Название	Описание	Количество экземпляров	Режим обновления
1	Пользователь	Список покупателей, менеджеров по работе с клиентами и администраторов	15000	При регистрации нового пользователя в системе
2	Роль	Список ролей пользователей	5	Определяется администратором по необходимости
3	Назначение роли	Список назначений ролей	30000	При добавлении, удалении роли пользователя
4	Настройки уведомлений	Список настроек уведомлений пользователя	15000	При изменении пользователем настроек своих уведомлений
5	Зарегистрированное устройство	Список зарегистрированных устройств пользователей	20000	При входе пользователя с нового устройства
6	Вопрос	Список вопросов к менеджерам	20000	При создании клиентом вопроса в поддержку
7	Ответ	Список ответов от менеджеров	20000	При добавлении нового ответа
8	Отзыв	Список отзывов	200000	При добавлении нового отзыва, при одобрении модератором
9	Товар	Список товаров электроники	2000	При добавлении, изменении информации о товаре
10	Категория электроники	Список категорий электроники	50	При добавлении, удалении, редактировании каталога электроники
11	Характеристика	Список характеристик товаров	200	При добавлении, изменении списка характеристик
12	Назначение характеристики	Список сопоставления характеристик каждой категории электроники	500	При добавлении, изменении информации о категории электроники
13	Значение характеристики	Список значений каждой характеристики товара	1000	При добавлении, изменении информации о товаре
14	Изображение товара	Список изображений товаров	3000	При добавлении, изменении информации о товаре
15	Заказ	Список заказов	200000	При оформлении заказа
16	Позиция заказа	Список позиций заказа	200000	При оформлении заказа
17	Изменение статуса заказа	Список изменений статуса заказов	1000000	При изменении менеджером статуса заказа
18	Корзина	Список корзин пользователей	15000	При добавлении товара в корзину, при оформлении заказа
19	Позиция корзины	Список позиций в корзинах пользователей	100000	При добавлении товара в корзину, при оформлении заказа

Для каждой связи между сущностями на концептуальной модели базы данных была составлена характеристика. Краткое описание связей между сущностями концептуальной модели базы данных представлено в таблице 9.

Таблица 9 – Связи между сущностями концептуальной модели

№	Связь	Описание	Мощность	Обязательность
1	Пользователь – Назначение роли	Определяет пользователя в записи о назначении роли	1:M	Да
2	Назначение роли – Роль	Определяет роль в записи о назначении роли пользователю	1:M	Да
3	Пользователь – Настройки уведомлений	Определяет настройки пуш-уведомлений для пользователя	1:1	Да
4	Пользователь – Зарегистрированные устройства	Определяет устройства, с которых произведен вход под учетной записью пользователя	1:M	Да
5	Вопрос – Ответ	Определяет ответы на вопрос	1:M	Да
6	Категория – Категория	Определяет иерархию категорий электроники	1:M	Да
7	Категория – Товар	Определяет товары в категории электроники	1:M	Да
8	Товар – Значение характеристики	Определяет значение одной из характеристик товара	1:M	Да
9	Назначение характеристики – Характеристика	Определяет характеристику в записи о назначении характеристики категории электроники	1:M	Да
10	Категория – Назначение характеристики	Определяет категорию в записи о назначении характеристики категории электроники	1:M	Да
11	Товар – Изображение	Определяет изображения товара	1:M	Да
12	Корзина – Позиция корзины	Определяет позиции в корзине клиента	1:M	Да
13	Заказ – Изменение статуса	Определяет записи об изменениях статуса заказа	1:M	Да
14	Заказ – Позиция заказа	Определяет позиции в заказе	1:M	Да

Таким образом была смоделирована общая структура базы данных, включающая в себя описание сущностей и наличие определенных связей между ними [22].

2.3.2 Логическое проектирование базы данных

В ходе логического проектирования базы данных автоматизированной системы сформирована её структура, включающая первичные ключи сущностей, механизмы установления связей между ними с использованием внешних ключей и базовые типы данных для каждого атрибута сущностей (без привязки к конкретной СУБД).

Для реализации отношения «Пользователь – Роль» (многие-ко-многим) была создана связывающая сущность «Назначение роли», которая представляет

собой список записей о том, когда была назначена та или иная роль пользователю. Сущности «Назначение характеристики», «Значение характеристики товара» также используются для связи сущностей отношениями «многие-ко-многим» [24]. Построенная логическая модель представлена на рисунке 30.

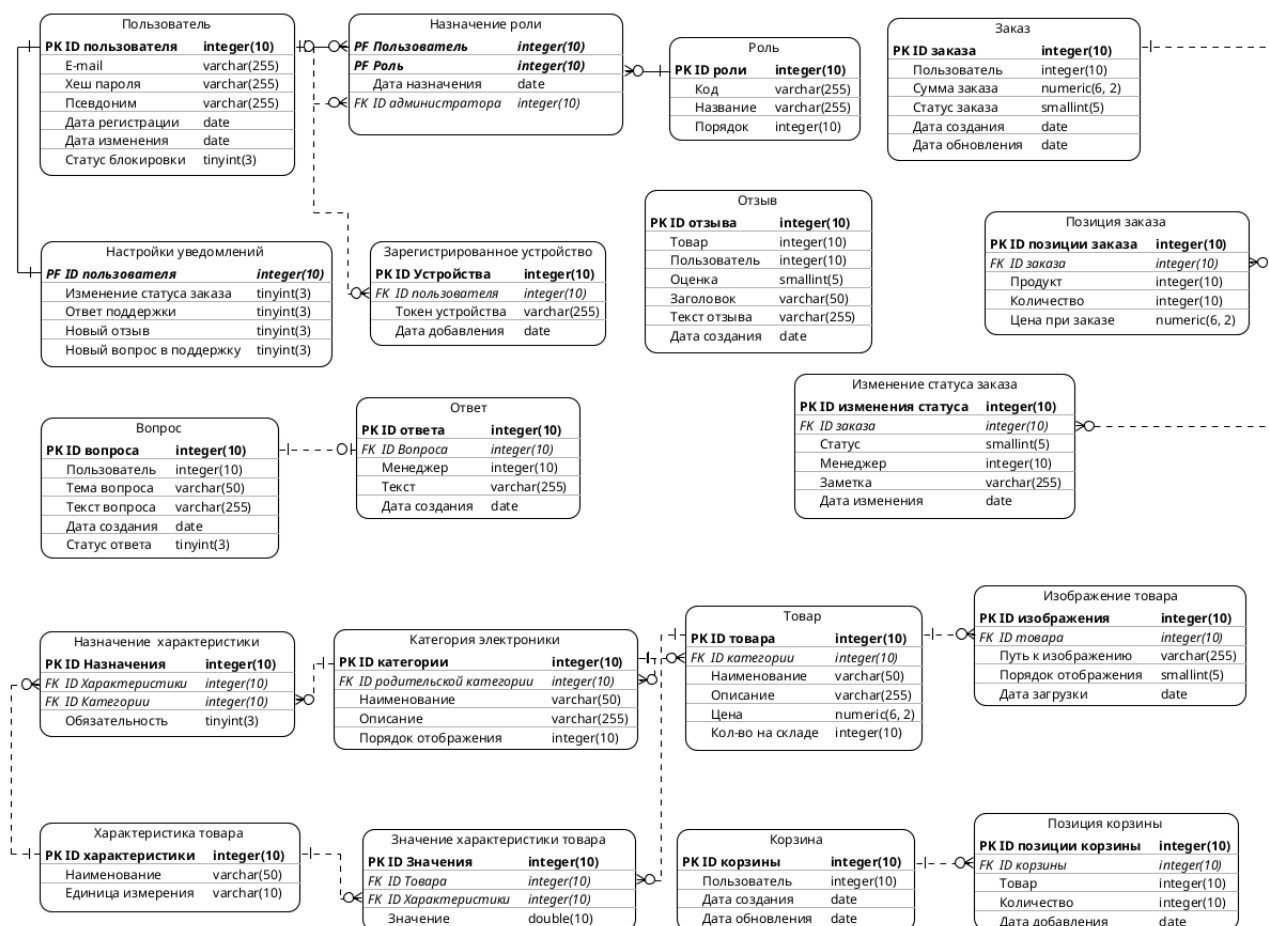


Рисунок 30 – Логическая модель базы данных

Логическая модель подробнее описывает структуру базы данных и не зависит от физической реализации [25].

2.4 Проектирование серверной части приложения

В качестве архитектуры серверной части выбран модульный монолит [26]. Это означает, что вся серверная бизнес-логика будет разворачиваться как один архитектурный квант (один исполняемый файл, один процесс в операционной системе). Однако структура исходного кода серверной части приложения будет состоять из независимых модулей, что позволяет при необходимости выделить

каждый модуль в отдельный архитектурный квант. Также необходимо учесть требование независимости к модулям и при разработке базы данных: сущности предметной области каждого модуля не должны быть связаны отношениями с сущностями предметных областей других модулей. Использование данной архитектуры позволяет избежать сложностей и затрат на разработку системы с полноценной масштабируемой микросервисной архитектурой, однако позволяет в случае необходимости быстро перейти от модульного монолита к микросервисам. Такой подход является наиболее предпочтительным для недорогой разработки программного обеспечения, которое впервые запускается в рамках определенного проекта, но подразумевает возможность дальнейшего развития при увеличении числа пользователей и повышении нагрузки на систему (рисунок 31).

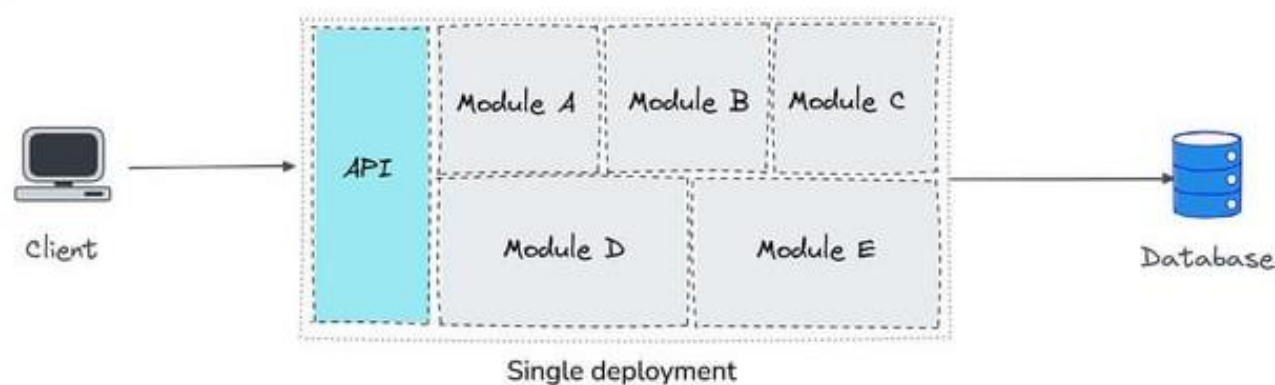


Рисунок 31 – Архитектура модульного монолита

Для обоснования выбора архитектуры программного обеспечения серверной части был проведен анализ и сравнение основных современных подходов к построению архитектуры. В качестве источника использовалась проверенная литература, которая включает в себя множество описаний и сравнений разных архитектурных подходов. Одними из них являются архитектуры микросервисов и модульный монолит. Сравнительная характеристика микросервисной архитектуры и архитектуры модульного монолита на основе архитектурных свойств представлена на рисунке 32 [27].

Архитектурное свойство	Рейтинг в звездах	Архитектурное свойство	Оценка в звездах
Тип разбиения	Предметный	Тип разбиения	Предметный
Количество квантов	1	Количество квантов	От одного и более
Развертываемость	**	Развертываемость	☆☆☆☆
Адаптируемость	***	Адаптируемость	☆☆☆☆☆
Эволюционность	***	Эволюционность	☆☆☆☆☆
Отказоустойчивость	*	Отказоустойчивость	☆☆☆☆
Модульность	****	Модульность	☆☆☆☆☆
Общие затраты	*****	Общие затраты	☆
Производительность	**	Производительность	☆☆
Надежность	***	Надежность	☆☆☆☆
Масштабируемость	*	Масштабируемость	☆☆☆☆☆
Простота	****	Простота	☆
Тестируемость	***	Тестируемость	☆☆☆☆

Рисунок 32 – Сравнение модульного монолита (слева) и микросервисов (справа)

На основании приведенного выше сравнения сделан вывод, что архитектура модульного монолита подходит для данного проекта лучше всего, так как с одной стороны она является перспективной для дальнейшего развития автоматизированной системы магазина электроники, а с другой не является избыточно сложной и дорогой для разработки.

В качестве протокола взаимодействия выбран REST API на основе HTTPS – гибкий вариант Web API. Передача данных между сервером и клиентской частью приложения с использованием REST API позволяет минимизировать использование сетевого трафика и унифицировать интерфейс взаимодействия. В проекте предполагается использовать HTTP-методы GET, POST, PUT, DELETE, PATCH. Шифрованный HTTPS обеспечивает безопасность и целостность данных при их передаче.

2.5 Проектирование интерфейса мобильного приложения

В разделе проектирования были учтены функциональные требования к приложению и на их основе составлены модели интерфейса мобильного приложения. Они включают в себя: страницы авторизации и регистрации, боковую навигационную панель, корневую страницу каталога, листовую

страницу категории, страницу товара, страницу настроек мобильного приложения, страницу профиля пользователя, страницу добавления и редактирования информации о товаре, страницу обращений пользователя в поддержку, страницу корзины пользователя, меню фильтрации товаров по характеристикам, страницу отчета по продажам, страницу для составления отзыва на товар, страницу обработки заказов менеджером.

Макет интерфейса страницы регистрации представлен на рисунке 33.

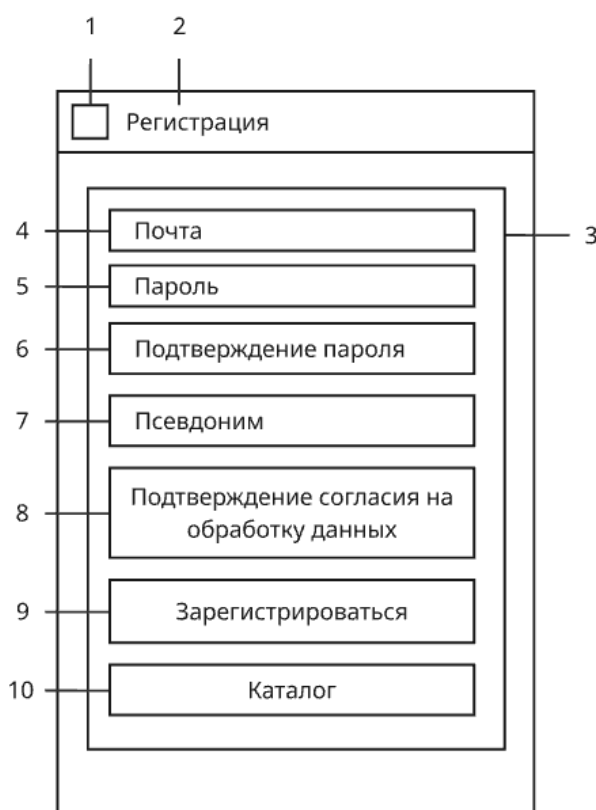


Рисунок 33 – Макет страницы регистрации

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка открытия боковой панели;
- 2 – наименование страницы;
- 3 – форма регистрации;
- 4 – текстовое поле для ввода почты;
- 5 – текстовое поле для ввода пароля (со скрыванием ввода);
- 6 – текстовое поле для повторного ввода пароля (со скрыванием ввода);
- 7 – текстовое поле для ввода псевдонима;

8 – флажок подтверждения согласия на обработку персональных данных;

9 – кнопка для отправки данных пользователя на регистрацию;

10 – кнопка для перехода на страницу каталога товаров электроники.

Макет интерфейса страницы авторизации представлен на рисунке 34.

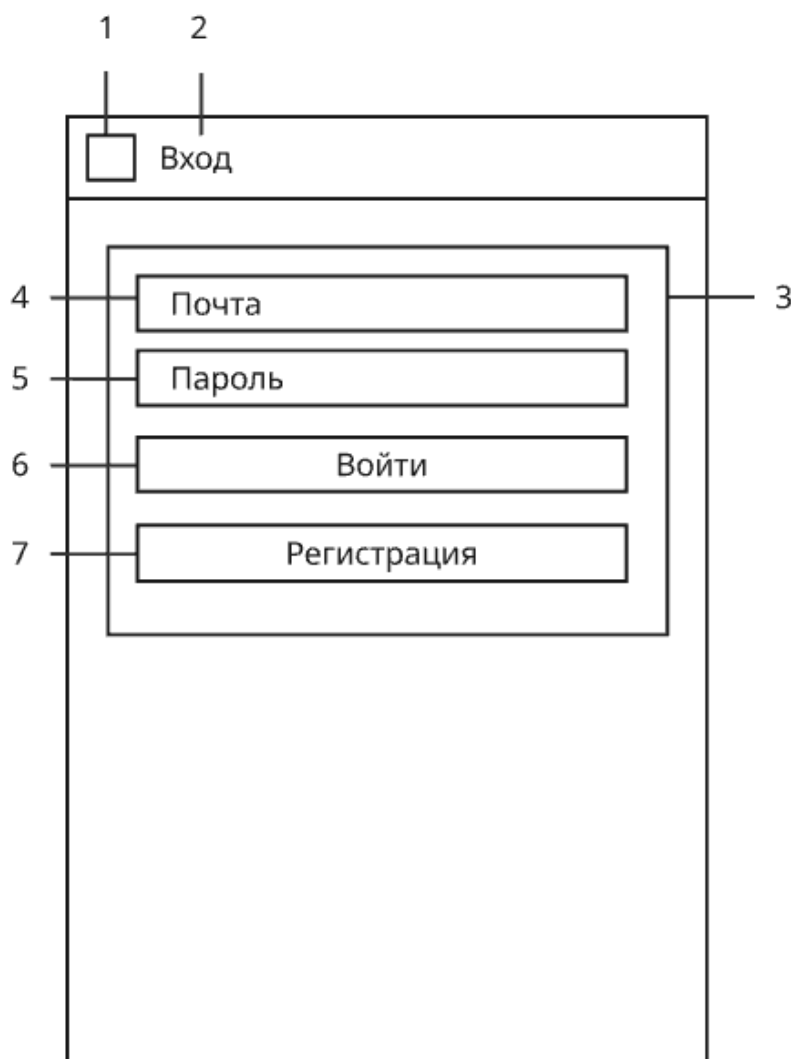


Рисунок 34 – Макет страницы авторизации

Перечень элементов, находящихся на макете страницы:

1 – кнопка открытия боковой панели;

2 – наименование страницы;

3 – форма авторизации;

4 – текстовое поле для ввода почты;

5 – текстовое поле для ввода пароля (со скрытием ввода);

6 – кнопка для отправки данных на авторизацию пользователя;

7 – кнопка для перехода на страницу регистрации.

Макет боковой навигационной панели (администратор) представлен на рисунке 35.

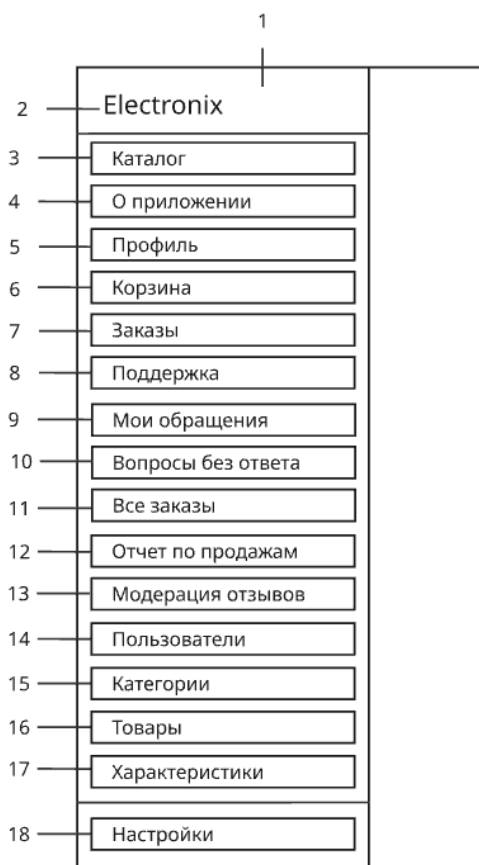


Рисунок 35 – Макет боковой навигационной панели (администратор)

Перечень элементов, находящихся на макете страницы:

1 – боковая навигационная панель;

2 – название магазина электроники (мобильного приложения);

3 – вкладка каталога электроники;

4 – вкладка с информацией о приложении;

5 – вкладка с информацией о текущем пользователе;

6 – вкладка для просмотра и изменения содержимого корзины;

7 – вкладка для просмотра завершенных и незавершенных заказов;

8 – вкладка для составления и отправки вопроса менеджером;

9 – вкладка для просмотра ранее отправленных менеджерам вопросов;
 10 – вкладка для просмотра вопросов пользователей и ответа на них;
 11 – вкладка для просмотра всех заказов, находящихся в работе;
 12 – вкладка для просмотра отчета по продажам за период;
 13 – вкладка для публикации и удаления отзывов пользователей;
 14 – вкладка для блокировки пользователей и назначения им ролей;
 15 – вкладка для редактирования категорий электроники;
 16 – вкладка для редактирования информации о товарах;
 17 – вкладка для редактирования информации о характеристиках товаров;
 18 – вкладка для настройки оформления и push-уведомлений мобильного приложения.

Макет корневой страницы каталога представлен на рисунке 36.



Рисунок 36 – Макет корневой страницы каталога

Перечень элементов, находящихся на макете страницы:

1 – кнопка открытия боковой навигационной панели;

2 – наименование страницы;

3 – верхняя панель;

4 – кнопка перехода на страницу корзины;

5 – кнопки перехода в указанную категорию электроники.

Макет страницы каталога листовой категории представлен на рисунке 37.

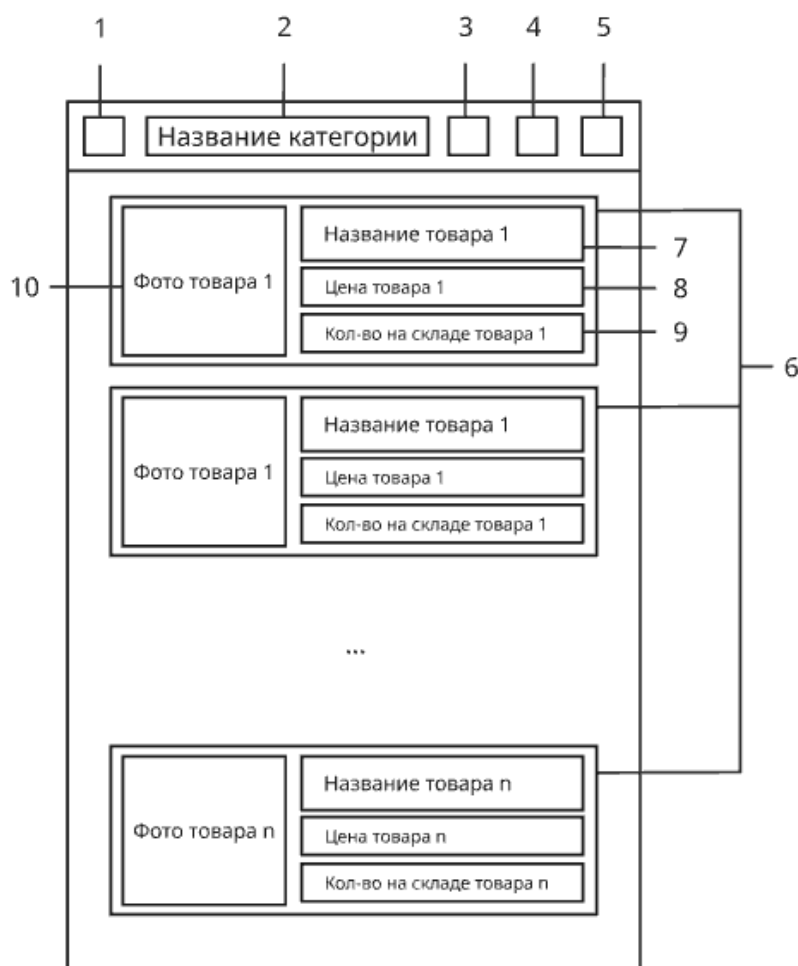


Рисунок 37 – Макет страницы каталога листовой категории

Перечень элементов, находящихся на макете страницы:

1 – кнопка возврата на предыдущую страницу;

2 – название текущей листовой категории;

3 – кнопка открытия меню фильтрации по характеристикам;

4 – кнопка открытия боковой навигационной панели;

5 – кнопка перехода на страницу корзины;

6 – карточки товаров с краткой информацией о них;

7 – название товара;

8 – цена товара;

9 – актуальное количество товара на складе;

10 – первая фотография товара.

Макет страницы товара представлен на рисунке 38.

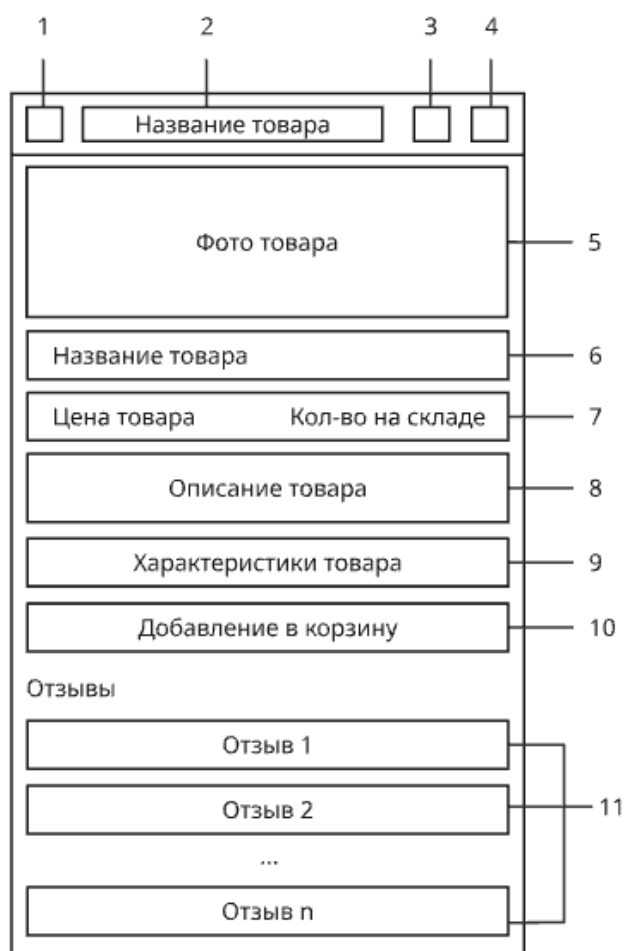


Рисунок 38 – Макет страницы товара

Перечень элементов, находящихся на макете страницы:

1 – кнопка возврата на предыдущую страницу;

2 – название просматриваемого товара;

3 – кнопка перехода на страницу корзины;

4 – кнопка открытия боковой навигационной панели;

5 – рамка для фотографий товара;

6 – название просматриваемого товара;

- 7 – информация о цене и актуальном количестве товара на складе;
- 8 – полное описание товара;
- 9 – список характеристик товара с числовыми значениями;
- 10 – добавление в корзину n экземпляров товара;
- 11 – отзывы пользователей о товаре.

Макет страницы настроек приложения рисунке 39.

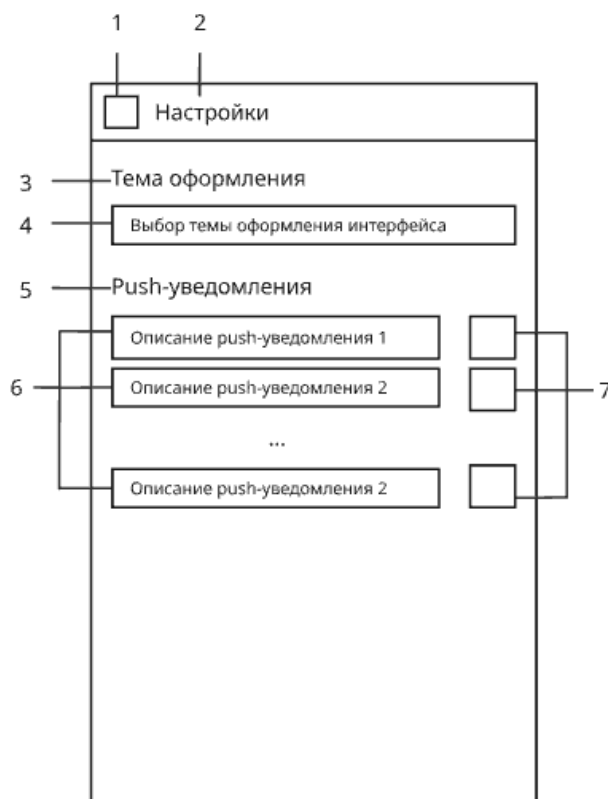


Рисунок 39 – Макет страницы настроек приложения

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка открытия боковой навигационной панели;
- 2 – наименование страницы;
- 3 – наименование раздела настроек оформления;
- 4 – селектор темы оформления интерфейса мобильного приложения;
- 5 – наименования раздела настроек push-уведомлений;
- 6 – подробные описания предназначений каждого push-уведомления;

7 – флаг включения и выключения push-уведомления, соответствующего описанию.

Макет страницы профиля пользователя представлен на рисунке 40.



Рисунок 40 – Макет страницы профиля пользователя

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка открытия боковой навигационной панели;
- 2 – наименование страницы;
- 3 – форма с информацией об учетной записи текущего пользователя;
- 4 – текстовое поле для вывода псевдонима;
- 5 – текстовое поле для вывода частичной скрытой почты;
- 6 – текстовое поле для вывода идентификатора пользователя (быстрое копирование);
- 7 – текстовое поле для вывода актуального состояния аккаунта;
- 8 – текстовое поле для вывода ролей пользователя;
- 9 – кнопка для открытия формы смены пароля.

Макет страницы добавления и редактирования информации о товаре представлен на рисунке 41.

The wireframe shows a vertical form layout. At the top, there is a header bar containing a small square button labeled '1' and the text 'Товары' labeled '2'. Below the header, the form contains several input fields: 'Название' (labeled '3'), 'Описание' (labeled '4'), 'Цена' (labeled '5'), and 'На складе' (labeled '6'). Below these is a category selector labeled '8' with the text 'Выбор категории электроники', and a small square button labeled '7' next to it. At the bottom of the form is a button labeled 'Создать' (labeled '9').

Рисунок 41 – Макет страницы добавления и редактирования информации о товаре

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка возврата на предыдущую страницу;
- 2 – наименование страницы;
- 3 – текстовое поле для ввода названия продукта;
- 4 – увеличенное текстовое поле для ввода описания продукта;
- 5 – числовое поле для ввода цены товара;
- 6 – числовое поле для ввода актуального количества товара на складе;
- 7 – кнопка для сброса селектора категории электроники;
- 8 – селектор категории электроники;
- 9 – кнопка для подтверждения обновления данных о товаре.

Макет страницы обращений пользователя в поддержку рисунке 42.

1 2

☐ Товары

Название 3

Описание 4

Цена 5

На складе 6

8 Выбор категории электроники ☐ 7

Создать 9

Рисунок 42 – Макет страницы обращений пользователя в поддержку

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка возврата на предыдущую страницу;
- 2 – наименование страницы;
- 3 – формы, отражающие информацию о каждом обращении;
- 4 – текстовое поле, содержащее тему обращения;
- 5 – текстовое поле, содержащее основной текст обращения;
- 6 – текстовое поле, содержащее актуальный статус ответа на обращение.

Макет страницы корзины пользователя представлен на рисунке 43.

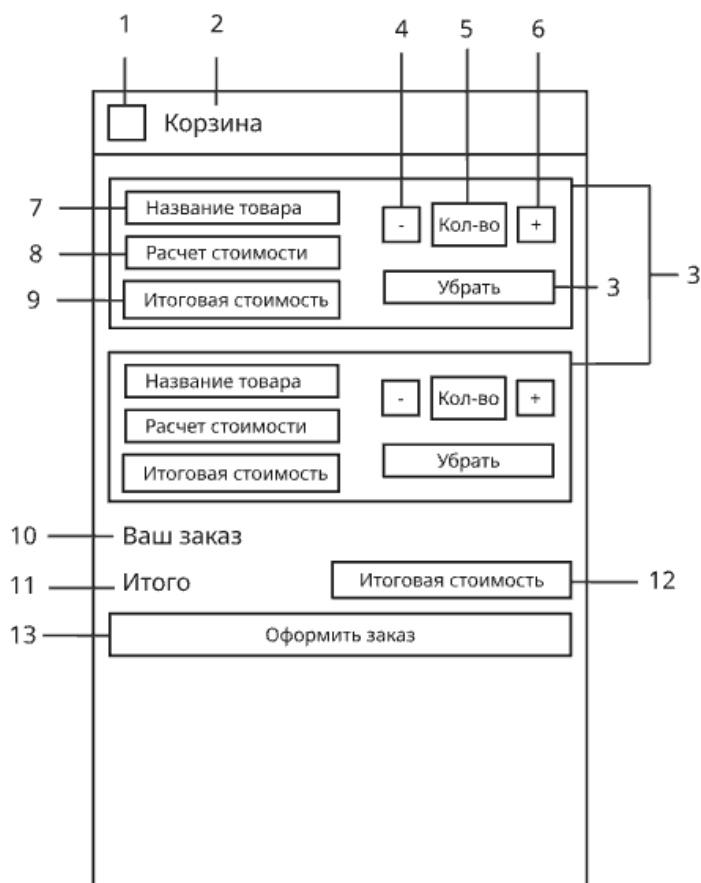


Рисунок 43 – Макет страницы корзины пользователя

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка возврата на предыдущую страницу;
- 2 – наименование страницы;
- 3 – форма, содержащая всю информацию об одной позиции заказа;
- 4 – кнопка для декремента количества товара в позиции;
- 5 – поле, отражающее текущее количество товара в позиции;
- 6 – кнопка для инкремента количества товара в позиции;
- 7 – название товара;
- 8 – расчет стоимости позиции через умножение (в виде выражения);
- 9 – текстовое поле с выводом итоговой стоимости данной позиции заказа;
- 10 – текстовая подпись для отделения позиций заказа от подведения итогов;
- 11 – текстовая подпись для определения положения итоговой стоимости заказа на странице;

12 – поле вывода общей стоимости всего заказа по сумме всех позиций заказа;

13 – кнопка для начала оформления заказа.

Макет меню фильтрации товаров по характеристикам представлен на рисунке 44.

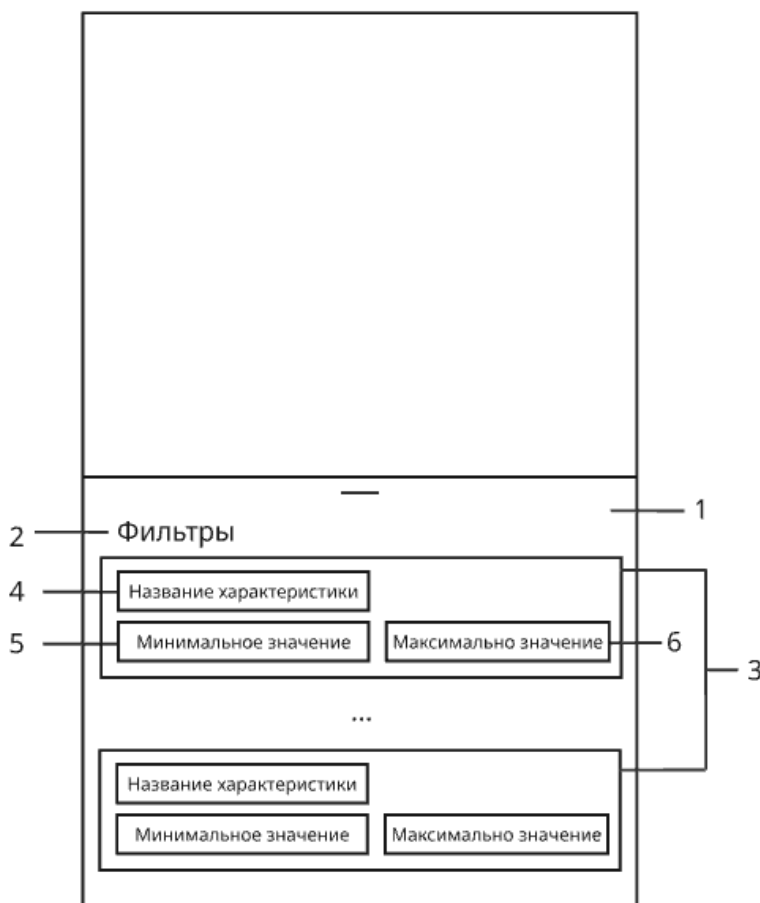


Рисунок 44 – Макет меню фильтрации товаров по характеристикам

Перечень элементов, находящихся на макете страницы:

- 1 – всплывающее меню фильтрации;
- 2 – наименование меню;
- 3 – формы, предоставляющие выбор граничных значений характеристик;
- 4 – текстовое поле для вывода названия характеристики;
- 5 – числовое поле для ввода минимально допустимого значения;
- 6 – числовое поле для ввода максимально допустимого значения.

Макет страницы отчета по продажам представлен на рисунке 45.

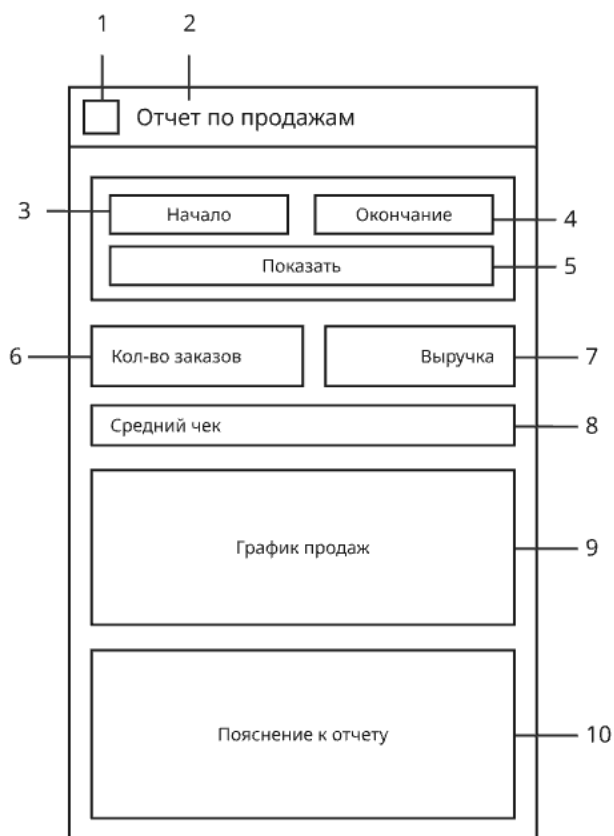


Рисунок 45 – Макет страницы отчета по продажам

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка открытия боковой навигационной панели;
- 2 – наименование страницы;
- 3 – поле даты для выбора начала периода отчетности;
- 4 – поле даты для выбора окончания периода отчетности;
- 5 – кнопка для запроса статистики продаж;
- 6 – числовое поле для вывода общего количества заказов за период;
- 7 – числовое поле для вывода общей выручки за период;
- 8 – числовое поле для вывода средней стоимости чека;
- 9 – динамически построенный график на основе данных статистики;
- 10 – текст с пояснением к содержанию страницы отчета по продажам.

Макет страницы для составления отзыва на товар представлен на рисунке 46.

The wireframe shows a vertical layout for a review page. At the top left is a small square button labeled '1'. To its right is a horizontal line labeled '2'. Below these is a box containing a small square icon and the text 'Написать отзыв'. Underneath is the word 'Оценка' followed by five stars, with a line labeled '3' pointing to the first star. Below the stars is a rectangular input field labeled 'Заголовок' with a line labeled '4' pointing to it. At the bottom is a large rectangular text area labeled 'Текст' with a line labeled '5' pointing to its left side.

Рисунок 46 – Макет страницы для составления отзыва на товар

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка возврата на предыдущую страницу;
- 2 – наименование страницы;
- 3 – строка из фигур «звезда» для установки оценки от 1 до 5;
- 4 – текстовое поле для ввода общего впечатления о товаре;
- 5 – текстовое поле для ввода основного текста отзыва.

Макет страницы обработки заказов менеджером представлен на рисунке 47.

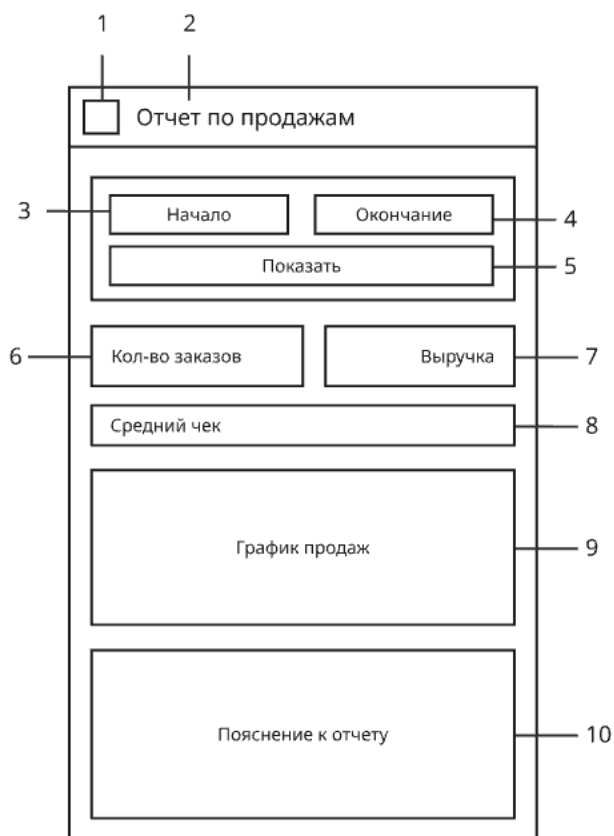


Рисунок 47 – Макет страницы обработки заказов менеджером

Перечень элементов, находящихся на макете страницы:

- 1 – кнопка открытия боковой навигационной панели;
- 2 – наименование страницы;
- 3 – текстовое поле для вывода идентификатора заказа (быстрое копирование);
- 4 – текстовое поле для вывода идентификатора покупателя (быстрое копирование);
- 5 – текстовое поле для вывода идентификатора менеджера (быстрое копирование);
- 6 – текстовое поле для вывода актуального статуса заказа;
- 7 – селектор нового статуса заказа;
- 8 – кнопка для подтверждения изменения статуса заказа.

Спроектированные макеты пользовательского интерфейса четко соответствуют функциональности разрабатываемого мобильного приложения.

3 Реализация программного обеспечения

3.1 Разработка схем базы данных

При построении физической модели базы данных были учтены особенности СУБД PostgreSQL 18. По содержанию логической модели (раздел 2) с помощью встроенных средств pgAdmin 4 была разработана физическая модель базы данных (рисунок 48).

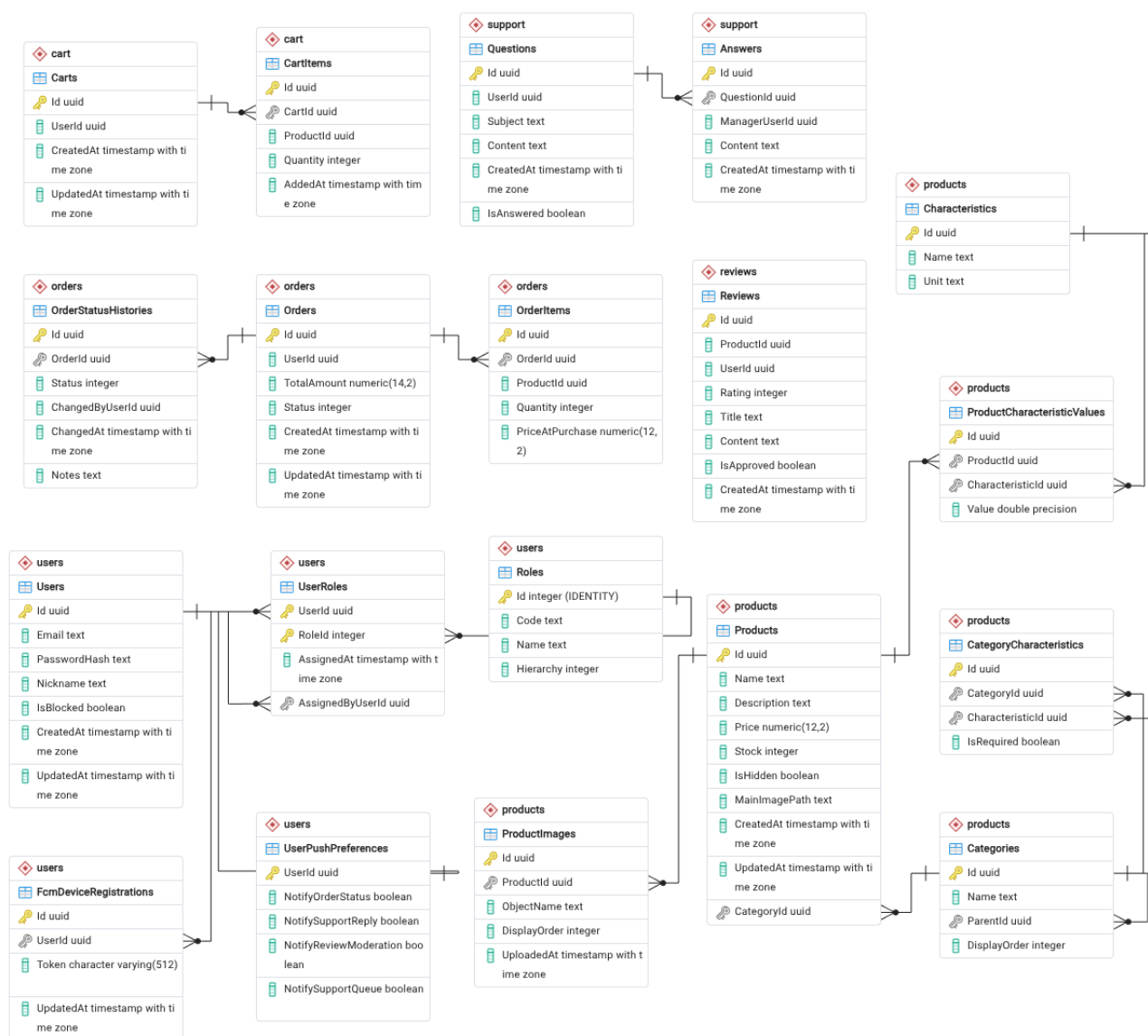


Рисунок 48 – Физическая модель базы данных

Далее приведены характеристики сущностей схемы «cart». Подробное описание атрибутов сущности «Корзина» представлено в таблице 10.

Таблица 10 – Описание атрибутов сущности «Корзина»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
UserId	Идентификатор владельца корзины	uuid	Нет	Нет	-
CreatedAt	Дата создания	timestamp	Нет	Нет	-
UpdatedAt	Дата изменения	timestamp	Нет	Нет	-

Подробное описание атрибутов сущности «Позиция корзины» представлено в таблице 11.

Таблица 11 – Описание атрибутов сущности «Позиция корзины»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
CartId	Идентификатор корзины	uuid	Нет	Нет	FK
ProductId	Идентификатор товара в позиции	uuid	Нет	Нет	-
Quantity	Количество	integer	Нет	Нет	-
AddedAt	Дата добавления	timestamp	Нет	Нет	-

Далее приведены характеристики сущностей схемы «orders». Подробное описание атрибутов сущности «Заказ» представлено в таблице 12.

Таблица 12 – Описание атрибутов сущности «Заказ»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
UserId	Идентификатор заказчика	uuid	Нет	Нет	-
TotalAmount	Сумма заказа	numeric(14,2)	Нет	Нет	-
Status	Статус заказа	integer	Нет	Нет	-
CreatedAt	Дата создания	timestamp	Нет	Нет	-
UpdatedAt	Дата изменения	timestamp	Нет	Нет	-

Подробное описание атрибутов сущности «Позиция заказа» представлено в таблице 13.

Таблица 13 – Описание атрибутов сущности «Позиция заказа»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
OrderId	Идентификатор заказа	uuid	Нет	Нет	FK
ProductId	Идентификатор товара	uuid	Нет	Нет	-
Quantity	Количество	integer	Нет	Нет	-
PriceAtPurchase	Цена на момент оформления заказа	numeric(12,2)	Нет	Нет	-

Подробное описание атрибутов сущности «История статуса заказа» представлено в таблице 14.

Таблица 14 – Описание атрибутов сущности «История статуса заказа»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
OrderId	Идентификатор заказа	uuid	Нет	Нет	FK
Status	Новый статус	integer	Нет	Нет	-
ChangedByUserId	Идентификатор сотрудника, изменившего статус заказа	uuid	Нет	Нет	-
ChangedAt	Дата изменения	timestamp	Нет	Нет	-
Notes	Примечание	text	Да	Нет	-

Далее приведены характеристики сущностей схемы «users». Подробное описание атрибутов сущности «Пользователь» представлено в таблице 15.

Таблица 15 – Описание атрибутов сущности «Пользователь»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
Email	Адрес электронной почты	text	Нет	Нет	-
PasswordHash	Хеш пароля	text	Нет	Нет	-
Nickname	Псевдоним	text	Нет	Нет	-
IsBlocked	Статус блокировки	boolean	Нет	Нет	-
CreatedAt	Дата создания	timestamp	Нет	Нет	-

Продолжение таблицы 15

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
UpdatedAt	Дата изменения	timestamp	Нет	Нет	-

Подробное описание атрибутов сущности «Настройки пуш-уведомлений» представлено в таблице 16.

Таблица 16 – Описание атрибутов сущности «Настройки пуш-уведомлений»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
UserId	Идентификатор пользователя	uuid	Нет	Да	PF
NotifyOrderStatus	Уведомление об изменении статуса заказа	boolean	Нет	Нет	-
NotifySupportReply	Уведомление о поступлении ответа на обращение	boolean	Нет	Нет	-
NotifyReviewModeration	Уведомление о поступлении нового отзыва	boolean	Нет	Нет	-
NotifySupportQueue	Уведомление о поступлении нового обращения в поддержку	boolean	Нет	Нет	-

Подробное описание атрибутов сущности «Регистрация устройства в FCM» представлено в таблице 17.

Таблица 17 – Описание атрибутов сущности «Регистрация устройства в FCM»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
UserId	Идентификатор пользователя	uuid	Нет	Нет	FK
Token	Токен устройства для FCM	varying(512)	Нет	Нет	-
UpdatedAt	Дата обновления	timestamp	Нет	Нет	-

Подробное описание атрибутов сущности «Роль» представлено в таблице 18.

Таблица 18 – Описание атрибутов сущности «Роль»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	integer (IDENTITY)	Нет	Да	PK
Code	Код роли	text	Нет	Нет	-
Name	Наименование роли	text	Нет	Нет	-
Hierarchy	Порядковый номер	integer	Нет	Нет	-

Подробное описание атрибутов сущности «Роли пользователя» представлено в таблице 19.

Таблица 19 – Описание атрибутов сущности «Роли пользователя»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
UserId	Идентификатор пользователя	uuid	Нет	Да	PF
RoleId	Идентификатор роли	integer	Нет	Да	PF
AssignedByUserID	Идентификатор администратора, назначившего роль пользователю	uuid	Да	Нет	FK
AssignedAt	Дата назначения роли	timestamp	Нет	Нет	-

Далее приведены характеристики сущностей схемы «support». Подробное описание атрибутов сущности «Вопрос» представлено в таблице 20.

Таблица 20 – Описание атрибутов сущности «Вопрос»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
UserId	Идентификатор автора вопроса	uuid	Нет	Нет	-
Subject	Тема вопроса	text	Нет	Нет	-
Content	Текст вопроса	text	Нет	Нет	-
CreatedAt	Дата создания	timestamp	Нет	Нет	-
IsAnswered	Статус ответа	boolean	Нет	Нет	-

Подробное описание атрибутов сущности «Ответ» представлено в таблице 21.

Таблица 21 – Описание атрибутов сущности «Ответ»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
QuestionId	Идентификатор вопроса	uuid	Нет	Нет	FK
ManagerUserId	Идентификатор автора ответа	uuid	Нет	Нет	-
Content	Текст ответа	text	Нет	Нет	-
CreatedAt	Дата создания	timestamp	Нет	Нет	-

Далее приведены характеристики сущностей схемы «reviews». Подробное описание атрибутов сущности «Отзыв» представлено в таблице 22.

Таблица 22 – Описание атрибутов сущности «Отзыв»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
ProductId	Идентификатор товара	uuid	Нет	Нет	-
UserId	Идентификатор автора отзыва	uuid	Нет	Нет	-
Rating	Оценка (1 – 5)	integer	Нет	Нет	-
Title	Заголовок отзыва	text	Нет	Нет	-
Content	Содержание отзыва	text	Нет	Нет	-
IsApproved	Статус публикации	boolean	Нет	Нет	-
CreatedAt	Дата создания	timestamp	Нет	Нет	-

Далее приведены характеристики сущностей схемы «products». Подробное описание атрибутов сущности «Товар» представлено в таблице 23.

Таблица 23 – Описание атрибутов сущности «Товар»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
Name	Наименование	text	Нет	Нет	-
Description	Название	text	Нет	Нет	-
Price	Цена	numeric(12,2)	Нет	Нет	-
Stock	Наличие на складе	integer	Нет	Нет	-

Продолжение таблицы 23

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
IsHidden	Год издания	boolean	Нет	Нет	-
MainImagePath	Жанр	text	Да	Нет	-
CreatedAt	Дата создания	timestamp	Нет	Нет	-
UpdatedAt	Дата изменения	timestamp	Нет	Нет	-
CategoryId	Идентификатор категории	uuid	Да	Нет	FK

Подробное описание атрибутов сущности «Изображение товара» представлено в таблице 24.

Таблица 24 – Описание атрибутов сущности «Изображение товара»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
ProductId	Идентификатор товара	uuid	Нет	Нет	FK
ObjectName	Имя объекта	text	Нет	Нет	-
DisplayOrder	Порядок отображения	integer	Нет	Нет	-
UploadedAt	Дата изменения	timestamp	Нет	Нет	-

Подробное описание атрибутов сущности «Характеристика» представлено в таблице 25.

Таблица 25 – Описание атрибутов сущности «Характеристика»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
Name	Наименование	text	Нет	Нет	-
Unit	Единица измерения	text	Нет	Нет	-

Подробное описание атрибутов сущности «Категория» представлено в таблице 26.

Таблица 26 – Описание атрибутов сущности «Категория»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
Name	Наименование	text	Нет	Нет	-
ParentId	Идентификатор родительской категории	uuid	Нет	Нет	FK
DisplayOrder	Порядок отображения	integer	Нет	Нет	-

Подробное описание атрибутов сущности «Характеристики категории» представлено в таблице 27.

Таблица 27 – Описание атрибутов сущности «Характеристики категории»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
CategoryId	Идентификатор категории	uuid	Нет	Нет	FK
CharacteristicId	Идентификатор характеристики	uuid	Нет	Нет	FK
IsRequired	Цена	boolean	Нет	Нет	-

Подробное описание атрибутов сущности «Значение характеристики товара» представлено в таблице 28.

Таблица 28 – Описание атрибутов сущности «Значение характеристики товара»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	uuid	Нет	Да	PK
ProductId	Идентификатор товара	uuid	Нет	Нет	FK
CharacteristicId	Идентификатор характеристики	uuid	Нет	Нет	FK
Value	Значение	double precision	Нет	Нет	-

Структура физической модели и типы данных полностью соответствуют спецификации типов СУБД PostgreSQL 18.

3.2 Разработка серверной части приложения

В среде Visual Code было разработано серверное кроссплатформенное приложение на языке C# с использованием технологии ASP.NET. В качестве программной основы разработки Web API были выбраны контроллеры модуля ASP.NET Core MVC [28].

Уровень представления отсутствует в серверной части (т. к. используется REST протокол для обмена данными с клиентской частью), поэтому разрабатываемое приложение ASP.NET нельзя отнести к классическому паттерну «Model-View-Controller». Разделение произведено на два уровня: уровень данных (модели, контексты, объекты перемещения данных) и уровень бизнес-логики (Web API контроллеры и службы). Создание кода серверной части велось в соответствии с этим разделением.

Для взаимодействия с СУБД PostgreSQL [29] был использован Entity Framework, позволяющий применить подход Domain-Driven Design (DDD) [30], ориентированный на первичное описание базы данных в виде классов на языке C#. Entity Framework позволяет не только формировать гибкую связь базы данных с кодом ASP.NET приложения, но предоставляет возможности для создания миграций баз данных, организуя тем самым контроль версий для структуры базы данных. Это очень важно для разработки итерационно развивающегося приложения. В качестве базового класса контекстов базы данных был выбран класс DbContext библиотеки Entity Framework.

В качестве базового класса для Web API контроллеров был выбран класс ControllerBase библиотеки ASP.NET Core MVC. В дополнение к функциональным службам были созданы: служба отправки пуш-уведомлений и служба защиты от атак методом перебора пароля. При этом служба отправки пуш-уведомлений реализована через сервис Firebase Cloud Messaging, но может быть легко заменена на любую другую реализацию без изменения существующего кода. Это возможно благодаря интерфейсу IPushNotificationSender.

Диаграммы классов серверной части для уровня данных и уровня бизнес-логики представлены на рисунках 49 – 50. Укрупненные копии диаграмм представлены в приложении Б.

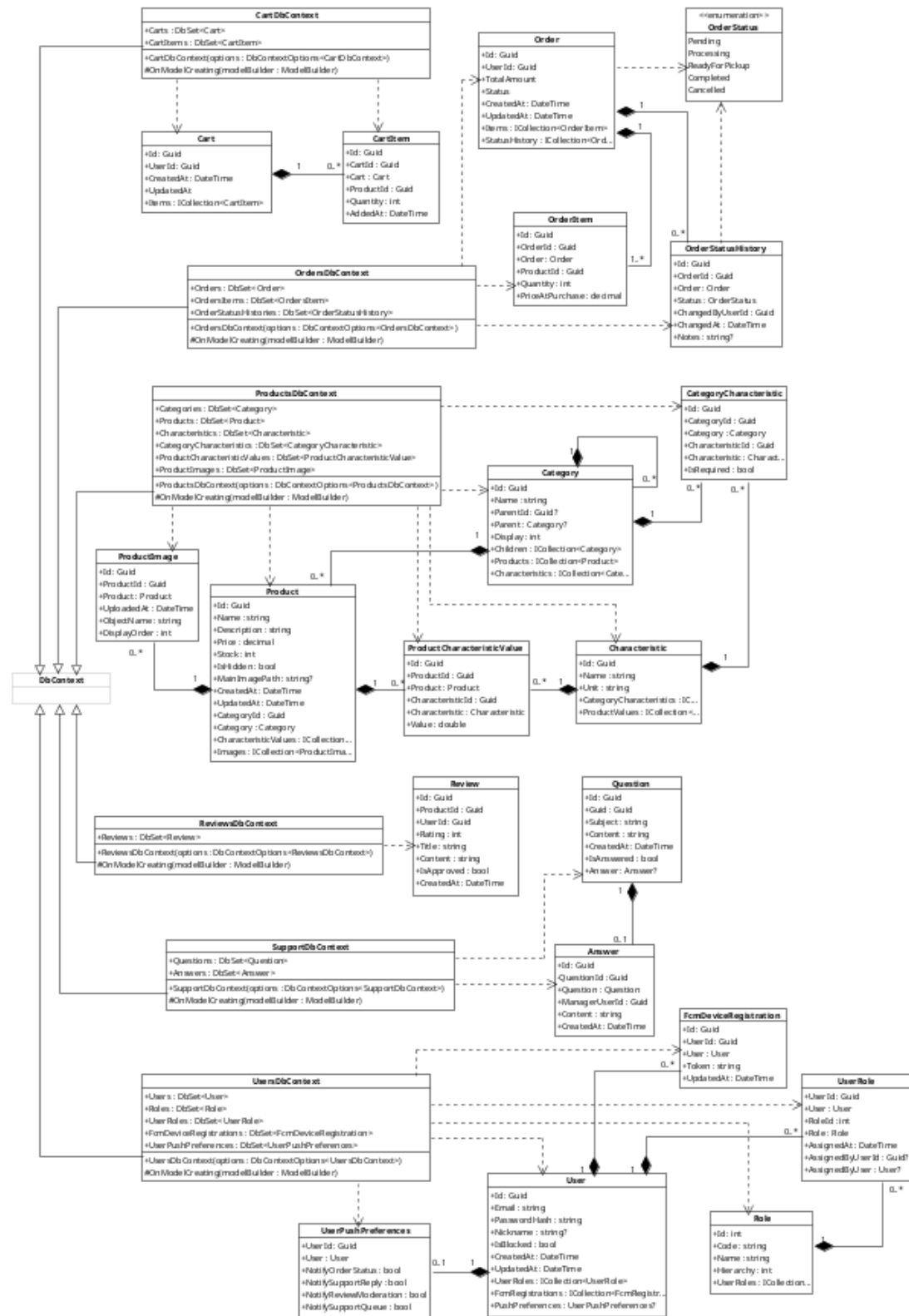


Рисунок 49 – Диаграмма классов уровня данных

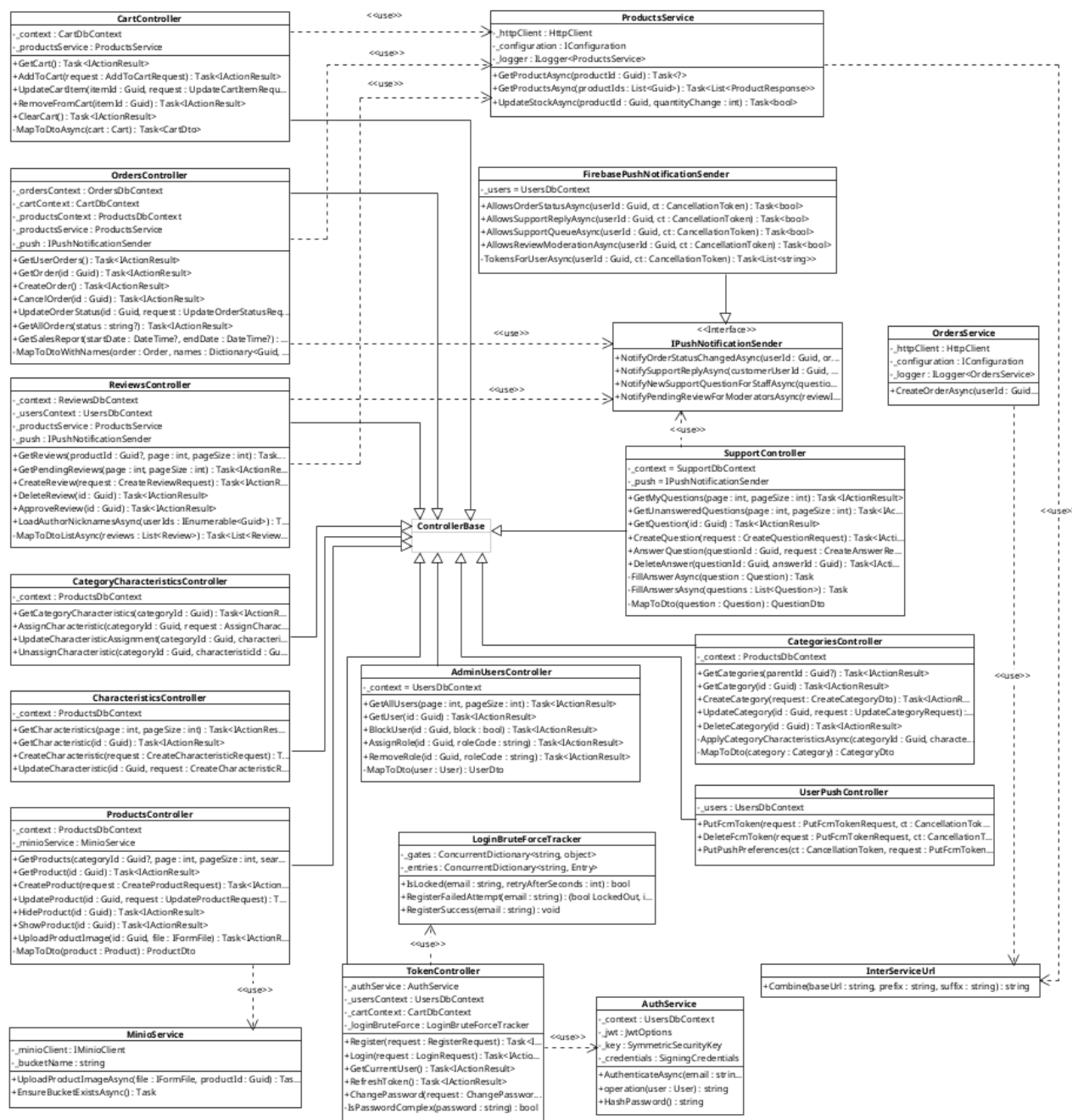


Рисунок 50 – Диаграмма классов уровня бизнес-логики

Описание и назначение классов уровня бизнес-логики представлено в таблице 29.

Таблица 29 – Описание классов уровня бизнес-логики

Наименование класса	Назначение
ControllerBase	Является базовым классом для всех контроллеров
CartController	Организует обработку запросов к конечным точкам группы /api/cart/

Продолжение таблицы 29

Наименование класса	Назначение
OrdersController	Организует обработку запросов к конечным точкам группы /api/orders/
ReviewsController	Организует обработку запросов к конечным точкам группы /api/reviews/
CategoryCharacteristicsController	Организует обработку запросов к конечным точкам группы /api/categories/{id}/characteristics/
CharacteristicsController	Организует обработку запросов к конечным точкам группы /api/characteristics/
ProductsController	Организует обработку запросов к конечным точкам группы /api/products/
SupportController	Организует обработку запросов к конечным точкам группы /api/support/
CategoriesController	Организует обработку запросов к конечным точкам группы /api/categories/
UserPushController	Организует обработку запросов к конечным точкам группы /api/users/me/fcm-token
TokenController	Организует обработку запросов к конечным точкам группы /api/users/{register login me}/
AdminUsersController	Организует обработку запросов к конечным точкам группы /api/users/
ProductService	Осуществляет выполнение сложных или составных операций при обработке запросов, связанных с товарами и категориями товаров электроники.
OrderService	Осуществляет выполнение сложных или составных операций при обработке запросов, связанных с заказами.
AuthService	Осуществляет выполнение сложных или составных операций при обработке запросов, связанных с регистрацией и входом в систему.
MinioService	Осуществляет загрузку изображений в хранилище MinIO и проверку наличия изображения в хранилище.
InterServiceUrl	Предоставляет актуальный URL для межмодульного взаимодействия.
IPushNotificationSender	Интерфейс, определяющий функционал службы отправки пуш-уведомлений.
FirebasePushNotificationSender	Реализация интерфейса IPushNotificationSender для сервиса Google FCM.
LoginBruteForceTracker	Служба, контролирующая количество попыток входа для одного аккаунта в режиме реального времени.

С целью автоматизации тестирования и развертывания серверной части была использована контейнеризация Docker с плагином Compose, который

позволяет запускать готовую среду выполнения, состоящую из нескольких контейнеров (служб). Диаграмма компонентов серверной части и их взаимодействие представлено на рисунке 51.

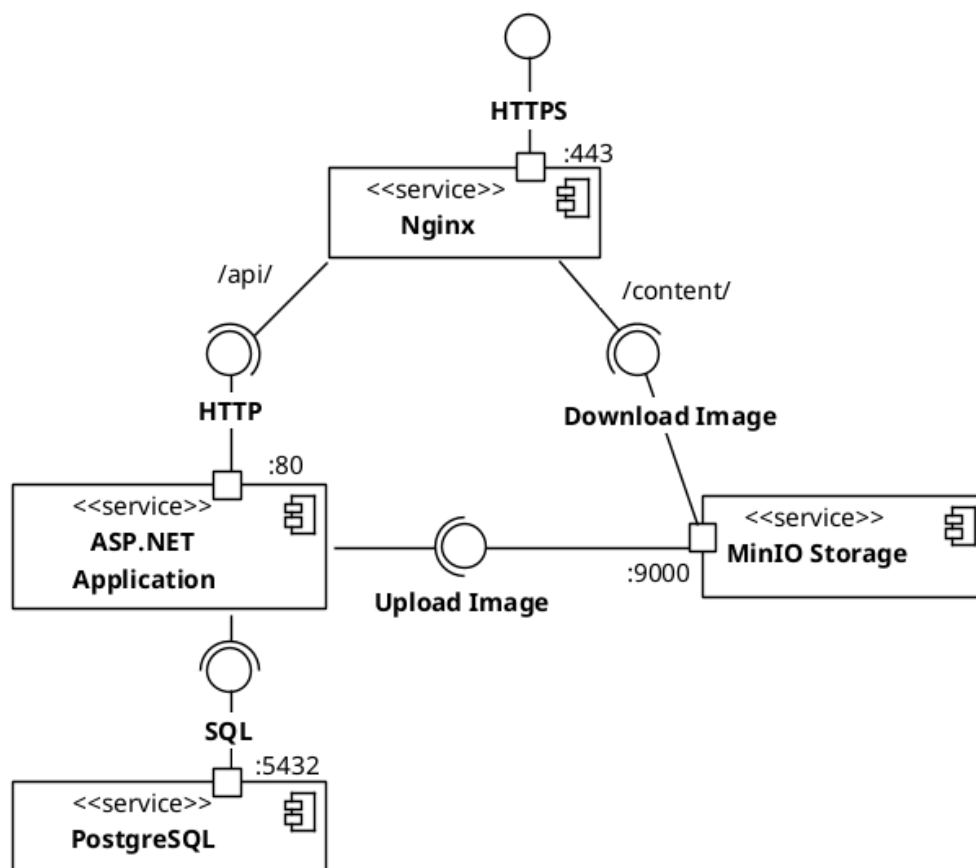


Рисунок 51 – Диаграмма компонентов серверной части

При разворачивании серверной части необходимы следующие контейнеры: Nginx в режиме прокси-сервера, разработанное ASP.NET приложение, СУБД PostgreSQL 18, хранилище изображений MinIO.

Листинг контроллера заказов представлен в приложении В. Полный код серверной части представлен в репозитории GitHub [37].

3.3 Разработка клиентской части приложения

В интегрированной среде разработки Android Studio Panda 3 была реализована и отлажена клиентская часть автоматизированной системы управления магазином электроники. Структура кода разработанного мобильного

приложения представлена на диаграмме компонентов клиентской части (рисунок 52).

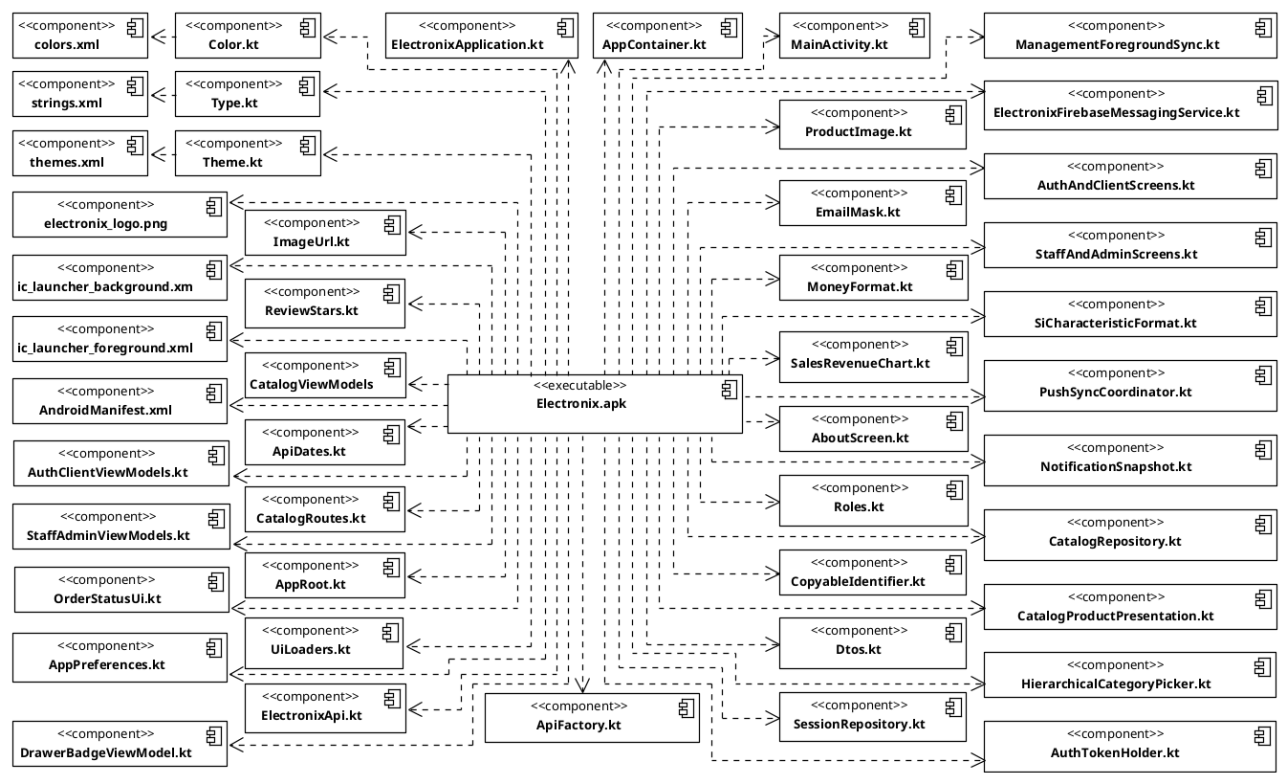


Рисунок 52 – Диаграмма компонентов клиентской части

Подробное описание компонентов клиентской части представлено в таблице 30.

Таблица 30 – Описание компонентов клиентской части

Наименование компонента	Назначение
Electronix.apk	Самодостаточный архив скомпилированного байт-кода и ресурсов мобильного приложения для установки и выполнения системой Android Runtime.
MainActivity.kt	Точка входа Activity: edge-to-edge, тема из настроек и корневой AppRoot с общим контейнером.
ElectronixApplication.kt	Application: создаёт AppContainer, канал push-уведомлений и глобальный ImageLoader для Coil.
AppContainer.kt	Корневой DI-контейнер: API, репозитории, сессия, push-синхронизация, кэш изображений и события обновления UI.
AppRoot.kt	Корневая навигация Compose: drawer, маршруты гостя, клиента и персонала, проверка ролей.

Продолжение таблицы 30

Наименование компонента	Назначение
ManagementForegroundSync.kt	Composable: периодическое обновление данных персонала, пока экран приложения на переднем плане.
OrderStatusUi.kt	Локализованные подписи статусов заказа (Pending, Processing и др.) для отображения в UI.
AboutScreen.kt	Информационный экран «О приложении» из бокового меню.
AuthAndClientScreens.kt	Compose-экраны клиента: логин, регистрация, профиль, корзина, заказы, support и создание отзыва.
CatalogRoutes.kt	Экраны каталога: дерево категорий, список товаров и карточка товара с переходами.
StaffAndAdminScreens.kt	Экраны менеджера, модератора и администратора: заказы, пользователи, каталог, отчёт, настройки.
CatalogProductPresentation.kt	Переиспользуемые блоки UI карточки и детали товара: галерея, цена, описание, характеристики.
UiLoaders.kt	Общие индикаторы загрузки и состояния «пусто» и «ошибка» для экранов.
SalesRevenueChart.kt	График выручки на экране отчёта продаж, построенный в Jetpack Compose.
ReviewStars.kt	Отображение рейтинга отзыва в виде звёзд.
CopyableIdentifier.kt	Строка с идентификатором (UUID) и копированием значения в буфер обмена.
ProductImage.kt	Загрузка и показ изображения товара через библиотеку Coil.
HierarchicalCategoryPicker.kt	Выбор категории в админке с учётом иерархии дерева категорий.
Theme.kt	Material 3 тема приложения с поддержкой светлой, тёмной и системной схемы.
Color.kt	Палитра цветов бренда Electronix для темы оформления.
Type.kt	Типографика: шрифты и стили текста для темы приложения.
AuthClientViewModels.kt	ViewModel клиента: авторизация, корзина, заказы, support, отзывы и смена пароля.
CatalogViewModels.kt	ViewModel каталога: навигация по категориям, список товаров и экран детали товара.
StaffAdminViewModels.kt	ViewModel админки и персонала: пользователи, товары, заказы, support и отчёт продаж.
ElectronixApi.kt	Retrofit-интерфейс со всеми конечными точками REST Electronix.

Окончание таблицы 30

Наименование компонента	Назначение
ApiFactory.kt	Сборка OkHttp, Retrofit и Kotlinx Serialization с базовым URL API из конфигурации сборки.
AuthInterceptor.kt	OkHttp-интерцептор: добавляет JWT из AuthTokenHolder в исходящие HTTP-запросы.
AuthTokenHolder.kt	Хранит текущий токен авторизации в памяти для интерцептора и сессии.
Dtos.kt	Serializable DTO запросов и ответов API в соответствии с контрактом бэкенда.
SessionRepository.kt	Вход, выход, загрузка профиля me и сохранение токена в локальных настройках.
CatalogRepository.kt	Загрузка категорий, товаров и характеристик каталога через API.
AppPreferences.kt	DataStore: токен, тема, FCM-токен, настройки push и метки последнего просмотра разделов.
NotificationSnapshot.kt	Модель локальных флагов уведомлений и timestamp для бейджей в drawer.
Roles.kt	Проверка ролей пользователя (MANAGER, MODERATOR, ADMINISTRATOR) для навигации и UI.
ElectronixFirebaseMessagingService.kt	FCM-сервис: приём push, показ уведомлений и отправка FCM-токена на сервер.
PushSyncCoordinator.kt	Синхронизация FCM-токена и настроек push с API при входе и изменении настроек.
SiCharacteristicFormat.kt	Форматирование числовых характеристик товара с SI-единицами для карточек каталога.
ApiDates.kt	Парсинг и форматирование дат и времени из API для отображения в интерфейсе.
EmailMask.kt	Маскирование адреса email при отображении в профиле и списках.
MoneyFormat.kt	Форматирование денежных сумм в рублях для цен, корзины и заказов.
ImageUrl.kt	Построение полного URL изображения товара (MinIO или API) для загрузки в Coil.

На время разработки, отладки и тестирования мобильного приложения в настройках безопасности было создано правило о доверии самоподписанному TLS-сертификату, сгенерированному для организации взаимодействия клиентской части с серверной по защищенному HTTPS-соединению [31]. В будущем предполагается внедрение доверенного TLS-сертификата.

Так как в базе данных системы предполагается хранение адреса электронной почты и других данных пользователей, в рамках разработки автоматизированной системы были учтены базовые требования ФЗ №152 «О персональных данных» [32]. Интерфейс мобильного приложения подразумевает получение согласия пользователя на обработку персональных данных при регистрации в системе.

Ниже представлена диаграмма разворачивания автоматизированной системы магазином электроники, отражающая взаимодействие клиентской и серверной частей (рисунок 53).

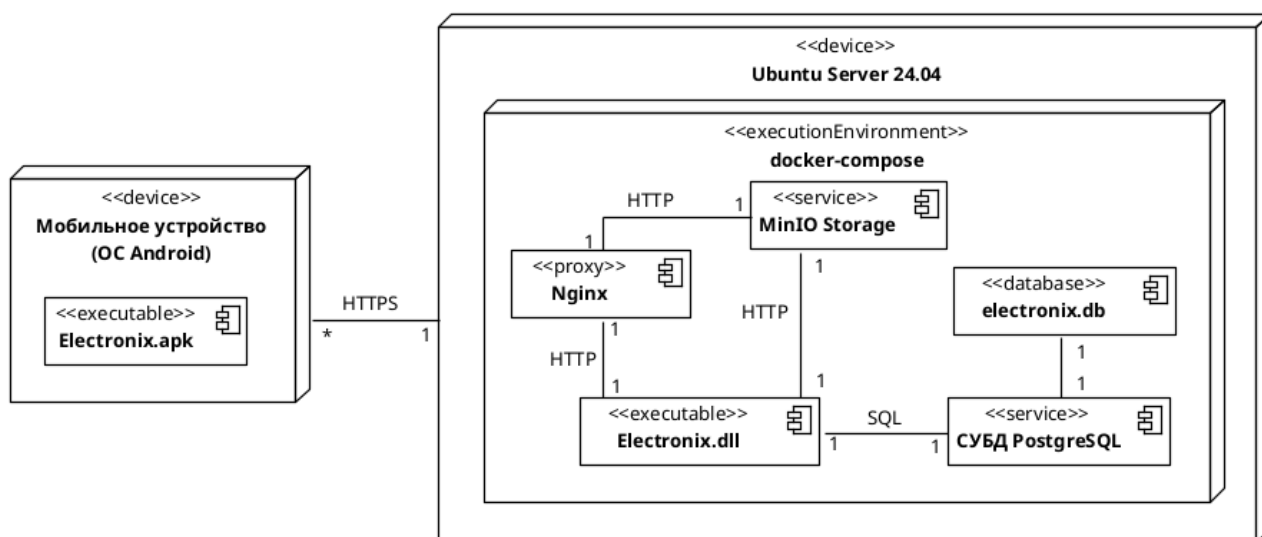


Рисунок 53 – Диаграмма разворачивания мобильного приложения

На приведенной выше диаграмме видно, что пользователь взаимодействует с серверной частью через клиентское приложение, установленное на устройстве под управлением ОС Android по защищенному HTTPS-соединению. Компоненты серверной части преимущественно взаимодействуют внутри системы по HTTP-протоколу. При этом к основной базе данных доступ имеет только ASP.NET приложение, которое взаимодействует с базой данных через СУБД PostgreSQL 18 с использованием SQL-запросов.

Листинг манифеста приложения и файл с описанием REST API представлен в приложении Г. Полный код клиентской части автоматизированной системы представлен в репозитории GitHub [38].

4 Тестирование программного продукта

4.1 Функциональное тестирование. Метод черного ящика

Функциональное тестирование – это вид проверки программного обеспечения, при котором оценивается, выполняет ли система заявленные функции в соответствии с требованиями, и корректно ли она реагирует на различные входные данные.

Тестирование методом черного ящика – это вид функционального тестирования, при котором тестируемой стороне не известна внутренняя структура программы [33]. Данный вид тестирования фокусируется только на оценке входных и выходных данных при работе программы.

Применим данный вид функционального тестирования к методу регистрации нового пользователя в автоматизированной системе интернет-магазина. В таблице 31 приведены классы эквивалентности (КЭ), разработанные для тестирования поля «Псевдоним» формы регистрации.

Таблица 31 – Классы эквивалентности для поля «Псевдоним»

Параметры	Правильные КЭ	Неправильные КЭ
Длина псевдонима	4 – 24 (1)	0 – 3 (2) 25 – ∞ (3)
Структура	Начинается с буквы (4)	Начинается не с буквы (5)
Содержание	Буквы русские и латинские (6) Цифры (7) Нижние подчеркивания (8)	Остальное (9)

Выше представлены все правильные и неправильные классы эквивалентности, ввод которых предполагается при работе приложения с формой регистрации.

В таблице 32 приведена часть разработанных тестов. Их фактические результаты приведены в приложении Д.

Таблица 32 – Классы эквивалентности для поля «Псевдоним»

№ п/п	КЭ	Входные данные	Выходные данные	Фактические результаты
1	1, 4, 6, 7, 8	Potato_1978	Отсутствуют	Пользователь успешно зарегистрирован и перенаправлен на страницу для входа в учетную запись (рисунок Д.1).
2	2, 4, 6, 7, 8	Ko	Ошибка: «Псевдоним слишком короткий»	Приложение корректно обработало ошибку (рисунок Д.2).
3	3, 4, 6, 7, 8	ZvzvzvZvzvzvZvzvzv ZvzvzvZvzvzv	Отсутствуют	Ограничение на длину ввода не позволяет ввести строку, длина которой превышает заданную (рисунок Д.3).
4	1, 5, 6, 7, 8	6Potato_1978	Ошибка: «сообщение с объяснением правил заполнения поля»	Приложение корректно обработало ошибку (рисунок Д.4).
5	1, 4, 9	Шмидт Старший	Ошибка: «Псевдоним не может содержать спецсимволы кроме нижнего подчеркивания».	Приложение корректно обработало ошибку (рисунок Д.5).

Были определены границы значений для поля «Псевдоним» (таблица 33).

Таблица 33 – Границы тестирования поля «Псевдоним»

Границы	Определение
Физические	Псевдоним содержит пустую строку
Логические	Псевдоним содержит недопустимые спецсимволы
Технологические	Символ, плохо отображаемый в приложении
Произвольные	Длина псевдонима менее 4-х символов

Результаты тестирования с применением граничных значений для поля «Псевдоним» приведены в таблице 34.

Таблица 34 – Результаты тестирования границ поля «Псевдоним»

№ п/п	Границы	Входные данные («»)	Выходные данные	Фактические результаты
6	Ф1	«»	Ошибка: псевдоним не должен быть пустым	Приложение корректно обработало ошибку (рисунок Д.6)
7	Т1	«Котик□»	Ошибка: псевдоним не должен содержать нестандартные символы UTF-8	Приложение корректно обработало ошибку (рисунок Д.7)
8	П1	«Fox»	Ошибка: псевдоним слишком короткий	Приложение корректно обработало ошибку (рисунок Д.8)
9	Л1	«@#!@#\$\$@#%»	Ошибка: псевдоним не может содержать спецсимволы кроме нижнего подчеркивания	Приложение корректно обработало ошибку (рисунок Д.9)

Аналогично было проведено функциональное тестирование всех модулей приложения. Все тесты были успешно пройдены, ошибки ввода выявлены, что подтверждает корректную работу функций при введении данных.

4.2 Модульное тестирование

Модульное тестирование – это подход к тестированию, который основан на независимой проверке функционирования отдельных модулей программы [34]. Тестирование проводится автоматизировано и в изолированной среде, подготовленной к тестам.

Так как архитектура модульного монолита (серверная часть) требует независимой проверки отдельных модулей, то проведение модульного тестирования необходимо.

Для реализации кода модульных тестов был создан отдельный проект с использованием NuGet пакета «xUnit». Для каждого модуля реализован отдельный класс, функции которого являются отдельными тестами.

Изолированная среда тестирования создана с помощью docker compose, в которой также выполняется заполнение базы данных тестовыми данными. Также в тестовую среду входит объектное хранилище MinIO. Код конфигурационного файла docker представлен на листинге 1.

Листинг 1 – Содержание конфигурационного файла docker-compose

```
services:
  application-test:
    container_name: application-test
    build:
      context: .
      dockerfile: Application/Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Testing
    ports:
      - "8080:80"
    depends_on:
      postgres-test:
        condition: service_healthy
      minio-test:
        condition: service_healthy
    networks:
      - test-network

  minio-test:
    image: minio/minio:latest
    container_name: minio-test
    command: server /data --console-address ":9001"
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    ports:
      - "9010:9000"
      - "9011:9001"
    volumes:
      - minio_test_data:/data
    networks:
      - test-network
    healthcheck:
      test: ["CMD", "curl", "-f",
"http://localhost:9000/minio/health/live"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  postgres_test_data:
  minio_test_data:
```

Ниже приведена часть кода тестов для модуля «Products» (листинг 2).

Листинг 2 – Код тестов для модуля «Products»

```
public class ProductDomainTests
{
    [Fact]
    public void ProductImage_ShouldStoreImageMetadata()
    {
        // Arrange
        var imageId = Guid.NewGuid();
        var productId = Guid.NewGuid();
        var createdAt = DateTime.UtcNow;

        // Act
        var image = new ProductImage
        {
            Id = imageId,
            ProductId = productId,
            ObjectName = "products/123/image-uuid.jpg",
            UploadedAt = createdAt
        };

        // Assert
        image.Id.Should().Be(imageId);
        image.ProductId.Should().Be(productId);
        image.ObjectName.Should().Be("products/123/image-
uuid.jpg");
        image.UploadedAt.Should().Be(createdAt);
    }
}
```

Результаты выполнения всех модульных тестов для всей системы представлены на рисунке 54.



```
Test accounts:
Admin: altmannvonw@icloud.com / 12345678
User: testuser@example.com / TestPassword123
altmann@home:~/Desktop/electronix/electronix-monolith$ ./run-tests.sh test
=== Running Tests ===
Restore complete (0.9s)
Application net10.0 succeeded (0.5s) → /home/altmann/Desktop/electronix/electronix-monolith/
Tests net10.0 succeeded (0.2s) → bin/Debug/net10.0/Tests.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v3.1.4+50e68bbb8b (64-bit .NET 10.0.4)
[xUnit.net 00:00:00.13] Discovering: Tests
[xUnit.net 00:00:00.22] Discovered: Tests
[xUnit.net 00:00:00.27] Starting: Tests
[xUnit.net 00:00:00.43] Finished: Tests
Tests test net10.0 succeeded (1.4s)

Test summary: total: 44, failed: 0, succeeded: 44, skipped: 0, duration: 1.4s
Build succeeded in 3.9s
```

Рисунок 54 – Результаты выполнения модульного тестирования

По результатам проведенного модульного тестирования был сделан вывод о корректной работе всех конечных точек каждого модуля системы. Полный код модульных тестов представлен в приложении Е.

4.3 Метрики кода

Метрики кода – это некоторые количественные характеристики кода, позволяющие определить его сложность, читаемость и качество комментирования [35].

Для анализа был выбран модуль товаров серверного приложения «ProductService.cs». Код модуля представлен на листинге 3.

Листинг 3 – Код модуля ProductService

```
namespace Application.Services;

/// <summary>
/// HTTP client service for Products module communication
/// </summary>
public class ProductsService
{
    private readonly HttpClient _httpClient;
    private readonly IConfiguration _configuration;

    public ProductsService(HttpClient httpClient, IConfiguration
configuration)
    {
        _httpClient = httpClient;
        _configuration = configuration;
    }

    private string GetProductsUrl() =>
_configuration["ServiceUrls:Products"] ?? "http://localhost:80";

    /// <summary>
    /// Get product by ID
    /// </summary>
    public async Task<ProductResponse?> GetProductAsync(Guid
productId)
    {
        try
        {
            var response = await
_httpClient.GetAsync($"{GetProductsUrl()}/api/products/{productId}
");
            if (response.IsSuccessStatusCode)
```

```

        {
            var json = await
response.Content.ReadAsStringAsync();
            return
System.Text.Json.JsonSerializer.Deserialize<ProductResponse>(json,
                new System.Text.Json.JsonSerializerOptions
{ PropertyNameCaseInsensitive = true });
        }
        return null;
    }
    catch
    {
        return null;
    }
}

```

Данный модуль отвечает за управление информацией о товарах магазина и электроники и их наличии на складах магазина. Результаты автоматического вычисления метрик сложности с помощью библиотеки «xlizard» для функций модуля «Products» представлены на рисунке 55.

NLOC	CCN	token	PARAM	length	location
5	1	18	2	5	ProductsService::ProductsService@11-15@ProductsService.cs
18	3	82	1	18	ProductsService::GetProductAsync@22-39@ProductsService.cs
6	1	64	1	6	ProductsService::GetProductsAsync@44-49@ProductsService.cs
18	2	76	2	19	ProductsService::UpdateStockAsync@54-72@ProductsService.cs
1 file analyzed.					
NLOC	Avg.NLOC	AvgCCN	Avg.token	function_cnt	file
62	11.8	1.8	60.0	4	ProductsService.cs

Рисунок 55 – Результаты автоматического вычисления метрик сложности

Основные показатели: CCN, NLOC. Значение CCN определяет цикломатическую сложность функции. Значение NLOC равно количеству непустых строк без комментариев. Цикломатическая сложность функций модуля «Products» низкая и количество строк кода находится в пределах нормы.

4.3.1 Метрики размера программы

Метрики размера программы являются количественными характеристиками исходного кода и отражают общее количество строк и комментариев к ним. Результат подсчета метрик отображен в таблице 35.

Таблица 35 – Результаты подсчета метрик размера программы

Название метрики	Значение
Общее количество строк кода	73
Количество пустых строк	9
Количество строк-комментариев	12
Среднее число строк для функций	$73/5 = 14,6$
Процент комментариев	$12/73 = 16,4 \%$

Показатель «Процент комментариев» в таблице х показывает насколько хорошо код закомментирован. По результатам вычисления метрик размера программы сделан вывод, что код модуля имеет небольшой размер, удобный для восприятия программистом благодаря наличию комментирования.

4.3.2 Метрики Холстеда

Метрики Холстеда позволяют оценить сложность кода программы и призваны обнаружить необходимость в переработке исходного при плохих числовых показателях [36]. Данные метрики позволяют выявлять потенциальные возможности для оптимизации кода.

Для вычисления метрик Холстеда необходимо найти следующие значения:

- количество уникальных операторов $n1$;
- количество уникальных операндов $n2$;
- общее число операторов $N1$;
- общее число операндов $N2$.

Для определения перечисленных выше значений и вычисления метрик составим словари операторов и операндов модуля. Словарь операторов представлен ниже (таблица 36).

Таблица 36 – Словарь операторов модуля

Оператор	Число вхождений
=	11

Продолжение таблицы 36

Оператор	Число вхождений
??	1
=>	1
return	4
await	4
if	1
try	2
catch	2
new	3
. (доступ/вызов)	28
() (вызов/группировка)	18
{ } (блоки/инициализация)	10
[]	1
,	8
GetProductsUrl	2
GetAsync	1
ReadAsStringAsync	1
Deserialize	1
Select	2
Where	1
ToList	2
WhenAll	1
Serialize	1
PatchAsync	1
	Всего: 107

Словарь операндов представлен в таблице 37.

Таблица 37 – Словарь операндов модуля

Оператор	Число вхождений
_httpClient	4
_configuration	3
httpClient	2
configuration	2
productId	3
productIds	2
quantityChange	2
response	3
json	2
tasks	2
results	2
request	2
content	2
Id	1
Name	1
Price	1
Stock	1
IsHidden	1
«application/json»	1
«http://localhost:80»	1
«ServiceUrls:Products»	1
	Всего: 45

Таким образом, после сложения всех операторов $n1 = 24$, после сложения всех уникальных операндов $n2 = 21$, после сложения всех вхождений операторов $N1 = 107$, после сложения всех вхождений операндов $N2 = 45$.

На основе метрик Холстеда можно вычислить целый ряд характеристик программы. Рассмотрим некоторые из них.

Расчет словаря программы производится по формуле:

$$n = n1 + n2 = 24 + 21 = 45,$$

где $n1$ – количество операторов в программе;

$n2$ – количество операндов программе.

Расчет длины программы производится по формуле:

$$N = N1 + N2 = 107 + 45 = 152,$$

где $N1$ – число вхождений всех операторов в программе;

$N2$ – число вхождений всех операндов программе.

Расчет объема программы производится по формуле:

$$V = N * \log_2 n = 152 * \log_2 45 = 834,$$

где N – длина программы;

n – словарь программы.

Расчет сложности программы производится по формуле:

$$D = (n1/2) * (N2/n2) = (24 / 2) * (45/ 21) = 25,7,$$

где $n1$ – количество операторов в программе;

$n2$ – количество операндов программе;

$N2$ – число вхождений всех операндов программе.

Расчет оценки интеллектуальных усилий производится по формуле:

$$E = D * V = 25,7 * 834 = 21445,$$

где D – сложность программы;

V – объем программы.

Рассмотрим значения вычисленных метрик.

Словарь программы $n = 45$, длина программы $N = 152$ были вычислены для нахождения ее объема. Объем программы $V = 834$, находится в пределах нормы и свидетельствует о том, что модуль программы не перегружен, деление на модули осуществлено оптимально.

Показатель сложности программы $D = 25.7$ указывает на низкую сложность при разработке и поддержке кода модуля.

Значение усилий $E = 21445$ указывает, что усилия, затраченные на разработку модуля, сопоставимы с разработкой бизнес-логики средней сложности.

На основе полученных характеристик можно сделать вывод, что протестированный код обладает небольшим объемом, легок для понимания, легко анализируется и требует меньше умственных затрат для поддержки и отладки. Созданный программный код представляет собой бизнес-логику средней сложности.

Заключение

В рамках выпускной квалификационной работы было разработано программное обеспечение для автоматизированной системы управления магазином электроники в форме мобильного приложения.

На этапе анализа требований были изучены особенности предметной области и требования к системе, проведён анализ целевой аудитории и рассмотрены существующие аналогичные решения. На основе полученных данных были определены функциональные и нефункциональные требования к разрабатываемому программному продукту.

В ходе планирования разработки и оценки бюджета был разработан план работ, составлена оценочная стоимость проекта. Срок разработки по плану составляет 48 дней. Общая стоимость разработки составляет 567353 рублей. Такие показатели сроков и стоимости выполнения работ указывают на экономическую целесообразность создания разрабатываемого программного продукта.

В процессе планирования разработки составлен график выполнения работ и выполнена предварительная оценка затрат. Согласно расчётам, продолжительность разработки составляет 48 дней, а ориентировочная стоимость – 567 353 рублей. Полученные значения позволяют сделать вывод о целесообразности реализации проекта с точки зрения сроков и бюджета.

На этапе проектирования были выбраны технологии и инструментальные средства разработки приложений. Также были спроектированы схемы базы данных, архитектура серверной и клиентской частей приложения.

В ходе разработки были созданы концептуальная, логическая и физическая модели базы данных, по которой позже была развернута реальная база данных под управлением СУБД PostgreSQL 18. Серверная часть представляет собой кроссплатформенное .NET-приложение, разработанное на языке C# 14.0 с использованием библиотек ASP.NET и редактора кода Visual Code. Клиентская часть является мобильным приложением, разработанным для устройств с

операционной системой Android. В качестве интегрированной среды разработки мобильного приложения использовалась IDE Android Studio Panda 3.

На этапе тестирования проверена работоспособность автоматизированной системы и стабильность ее работы. Были вычислены показатели метрик кода программ для анализа его качественных показателей. На этапе тестирования были устранены ошибки и устранены недочеты.

В качестве дальнейшего развития проекта могут быть созданы приложения для мобильных устройств с операционной системой iOS и клиентская часть в формате классического web-сайта. Также возможна публикация приложения в онлайн магазинах мобильных приложений «Google Play» и «RuStore».

Таким образом, в ходе выполнения выпускной квалификационной работы все поставленные задачи были успешно выполнены.

Список использованных источников

1. Указ Президента Российской Федерации от 9 мая 2017 г. № 203 «О Стратегии развития информационного общества в Российской Федерации на 2017 – 2030 годы». Официальный сайт [Электронный ресурс]. URL: <http://www.kremlin.ru/acts/bank/41919/> (дата обращения 06.04.2026).
2. Указ Президента Российской Федерации от 7 мая 2024 г. № 309 «О национальных целях развития Российской Федерации на период до 2030 года и на перспективу до 2036 года». Официальный сайт [Электронный ресурс]. URL: <http://www.kremlin.ru/acts/bank/50542/> (дата обращения 06.04.2026).
3. Бюро цифровых проектов. Новости. [Электронный ресурс]. URL: <https://www.comnews.ru/content/243899/2026-02-19/2026-w08/1010/68-rossiyan-gotovy-k-prosmotru-reklamy-obmen-besplatnye-cifrovyie-servisy> (дата обращения 06.04.2026).
4. Анализ магазина «Диполь». [Электронный ресурс]. URL: <https://dipolpenza.ru> (дата обращения 06.04.2026).
5. Анализ магазина «Электрорадиодетали». [Электронный ресурс]. URL: <http://elradio.ru/> (дата обращения 06.04.2026).
6. Анализ магазина «Вольтмастер». [Электронный ресурс]. URL: <https://www.voltmaster-samara.ru/?ysclid=mljm34tnik476344389> (дата обращения 06.04.2026).
7. Тенденции на сокращение импортозависимости в области электроники. [Электронный ресурс]. URL: <https://ptkgroup.ru/press-centre/stati/ryinok-elektronnyix-komponentov-v-rossii-tekushhie-trendyi-i-perspektivy/> (дата обращения 06.04.2026).
8. Объем рынка российской электроники. [Электронный ресурс]. URL: <https://www.interfax.ru/business/1032953/> (дата обращения 08.04.2026).
9. Распространенность мобильных ОС в России. [Электронный ресурс]. URL: <https://statbase.ru/datasets/internet-and-technology/operating-system-market-share-worldwide> (дата обращения 08.04.2026).

10. Статистика экранного времени приложений ОС Android. [Электронный ресурс]. URL: <https://techrt.com/android-app-usage-statistics/> (дата обращения 09.04.2026).
11. Российский рынок мобильных ОС. [Электронный ресурс]. URL: <https://сра.rip/articles/mobile-os-map/> (дата обращения 09.04.2026).
12. Распределение версий Android. [Электронный ресурс]. URL: <https://composables.com/android-distribution-chart> (дата обращения 11.04.2026).
13. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка, 2-е издание / Дж. Рамбо, М. Блаха // СПб.: Питер, 2007. – 544 с.
14. Беркун, С. Искусство управления IT-проектами / С. Беркун // СПб.: Питер, 2007. – 400 с.: ил.
15. Разу, М.Л. Управление проектом. Основы проектного управления: учебник, 4-е издание / М.Л. Разу, Т.М. Бронникова, А.М. Лялин // М.: КНОРУС, 2015. – 756 с.
16. Гвоздев, В.Е. Управление программными проектами: учебное пособие для вузов / В.Е. Гвоздев // Москва: Издательство Юрайт, 2023. – 167 с.
17. Ехлаков, Ю.П. Управление программными проектами. Стандарты, модели: Учебное пособие / Ю.П. Ехлаков // СПб.: Лань, 2019. – 244 с.
18. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов // Рн/Д: Феникс, 2018. – 102 с.
19. Общие сведения о методологии RAD. [Электронный ресурс]. URL: <https://www.informicus.ru/items-rad.html> (дата обращения 15.04.2026).
20. Общие сведения об архитектуре «клиент-сервер». [Электронный ресурс]. URL: <https://sky.pro/wiki/javascript/osnovnye-komponenty-klient-servernoj-arhitektury/> (дата обращения 15.04.2026).
21. Карпова, И. П. Базы данных : учебное пособие / И. П. Карпова. – Москва ; Санкт-Петербург : Питер, 2013. – 304 с.

22. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. – 3-е изд. – Москва ; Санкт-Петербург ; Киев : Вильямс, 2017. – 1440 с.
23. Кузнецов, С. Д. Базы данных : учебник для бакалавров / С. Д. Кузнецов. – Москва : Издательский центр «Академия», 2012. – 496 с.
24. Нестеров, С. А. Базы данных : учебник и практикум для вузов / С. А. Нестеров. – Москва : Издательство Юрайт, 2023. – 230 с. – Текст : электронный // Образовательная платформа Юрайт [сайт]. – URL: <https://urait.ru/bcode/511650> (дата обращения: 19.05.2026).
25. Окулов, С. М. Базы данных : учебное пособие / С. М. Окулов. – 2-е изд. – Москва : Лаборатория знаний, 2020. – 451 с.
26. Общие сведения об архитектуре «Модульный монолит». [Электронный ресурс]. URL: <https://newsletter.techworld-with-milan.com/p/what-is-a-modular-monolith> (дата обращения 15.04.2026).
27. Ричардс, Марк. Фундаментальный подход к архитектуре / Марк Ричардс, Нил Форд. ; пер. с англ. – Санкт-Петербург : Питер, 2023. – 445 с.
28. Документация по Microsoft ASP.NET Web API. [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/aspnet/web-api/> (дата обращения: 16.05.2025).
29. Документация по PostgreSQL 18. [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения: 16.05.2026).
30. Документация по Entity Framework Domain-Driven Design. [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/> (дата обращения: 18.05.2026).
31. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы : учебник для вузов / В. Г. Олифер, Н. А. Олифер. – 5-е изд. – Санкт-Петербург : Питер, 2020. – 992 с.

32. О персональных данных : Федеральный закон от 27.07.2006 № 152-ФЗ // Собрание законодательства РФ. – 2006. – № 31 (ч. 1), ст. 3451.
33. Метод тестирования «Черный ящик». [Электронный ресурс]. URL: <https://blog.infra-tech.ru/black-box-testing/> (дата обращения: 20.05.2025).
34. Модульное тестирование. [Электронный ресурс]. URL: https://logrocon.ru/news/unit_testing (дата обращения: 18.05.2026).
35. Метрики кода программы. [Электронный ресурс]. URL: <https://www.baeldung.com/cs/software-metrics-loc> (дата обращения: 20.05.2026).
36. Метрики Холстеда. [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/> (дата обращения: 20.05.2026).
37. Репозиторий GitHub для серверной части приложения. [Электронный ресурс]. URL: <https://github.com/altmann-von-wissenschaft/electronix-monolith/> (дата обращения: 25.05.2026).
38. Репозиторий GitHub для клиентской части приложения. [Электронный ресурс]. URL: <https://github.com/altmann-von-wissenschaft/electronix-android/> (дата обращения: 25.05.2026).

Приложение А
(справочное)
Глоссарий

Android – мобильная ОС для смартфонов, планшетов и встраиваемых устройств на базе ядра Linux.

Android Studio – интегрированная среда разработки (IDE – Integrated Development Environment) для создания Android-приложений.

API (Application Programming Interface) – программный интерфейс взаимодействия между программными компонентами или системами.

ASP.NET Core (Active Server Pages for .NET Core) – кроссплатформенный фреймворк компании Microsoft для разработки веб-приложений и API.

CCN (Cyclomatic Complexity Number) – показатель цикломатической сложности кода, характеризующий количество независимых путей выполнения программы.

CCN (Cyclomatic Complexity Number) – показатель цикломатической сложности кода, характеризующий количество независимых путей выполнения программы.

Docker – платформа контейнеризации приложений с изолированным окружением выполнения.

Entity Framework – ORM (Object-Relational Mapping) инфраструктура для работы с базами данных в среде .NET.

FCM (Firebase Cloud Messaging) – сервис доставки push-уведомлений и сообщений для мобильных и веб-приложений.

HTTP (HyperText Transfer Protocol) – прикладной протокол передачи гипертекста в сети Интернет.

HTTPS (HyperText Transfer Protocol Secure) – защищённая версия протокола HTTP с использованием TLS-шифрования.

iOS (iPhone Operating System) – мобильная операционная система компании Apple для устройств iPhone и iPad.

Linux – ядро для семейства Unix-подобных операционных систем с открытым исходным кодом.

MVVM (Model-View-ViewModel) – архитектурный шаблон проектирования, разделяющий модель данных, пользовательский интерфейс и логику представления.

NLOC (Non-Commenting Lines of Code) – метрика количества строк исходного кода без учёта пустых строк и комментариев.

NuGet (NuGet Package Manager) – система управления пакетами для платформы .NET.

Rapid Application Development (RAD) – итерационная методология ускоренной разработки программного обеспечения на основе создания прототипов.

REST API (Representational State Transfer Application Programming Interface) – интерфейс взаимодействия между системами, основанный на архитектурном стиле REST и HTTP-запросах.

SDK (Software Development Kit) – набор инструментов, библиотек и документации для разработки программного обеспечения под определённую платформу.

TLS (Transport Layer Security) – криптографический протокол защиты передаваемых по сети данных.

Ubuntu Server 24.04 – серверная версия операционной системы (дистрибутива) Ubuntu с долгосрочной поддержкой (LTS – Long Term Support).

UML (Unified Modeling Language) – унифицированный язык моделирования для визуального проектирования программных систем.

СУБД – программная система управления базами данных, обеспечивающая хранение, обработку и доступ к данным.

Приложение Б
(рекомендуемое)
Диаграммы классов

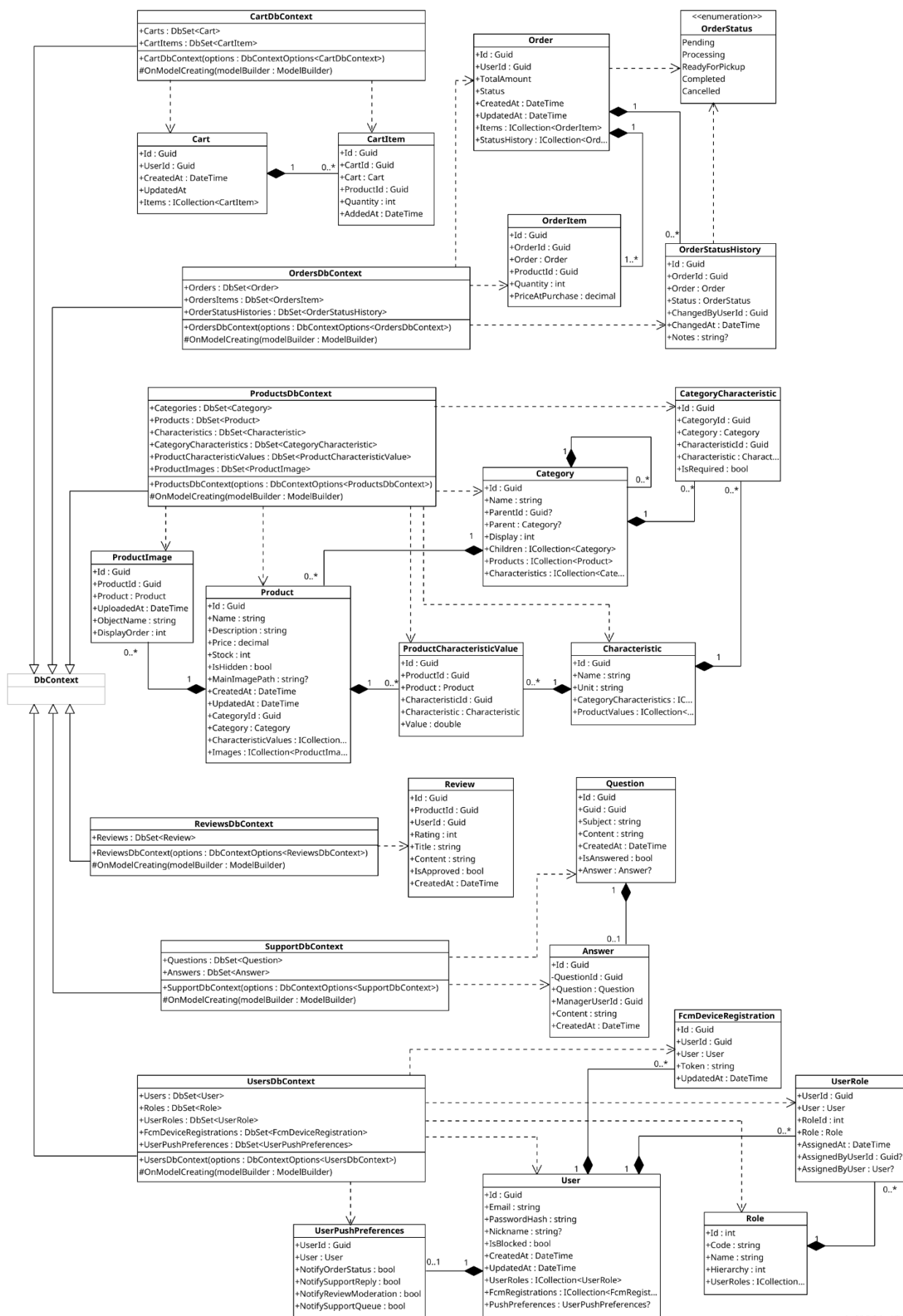


Рисунок Б.1 – Диаграмма классов для уровня данных

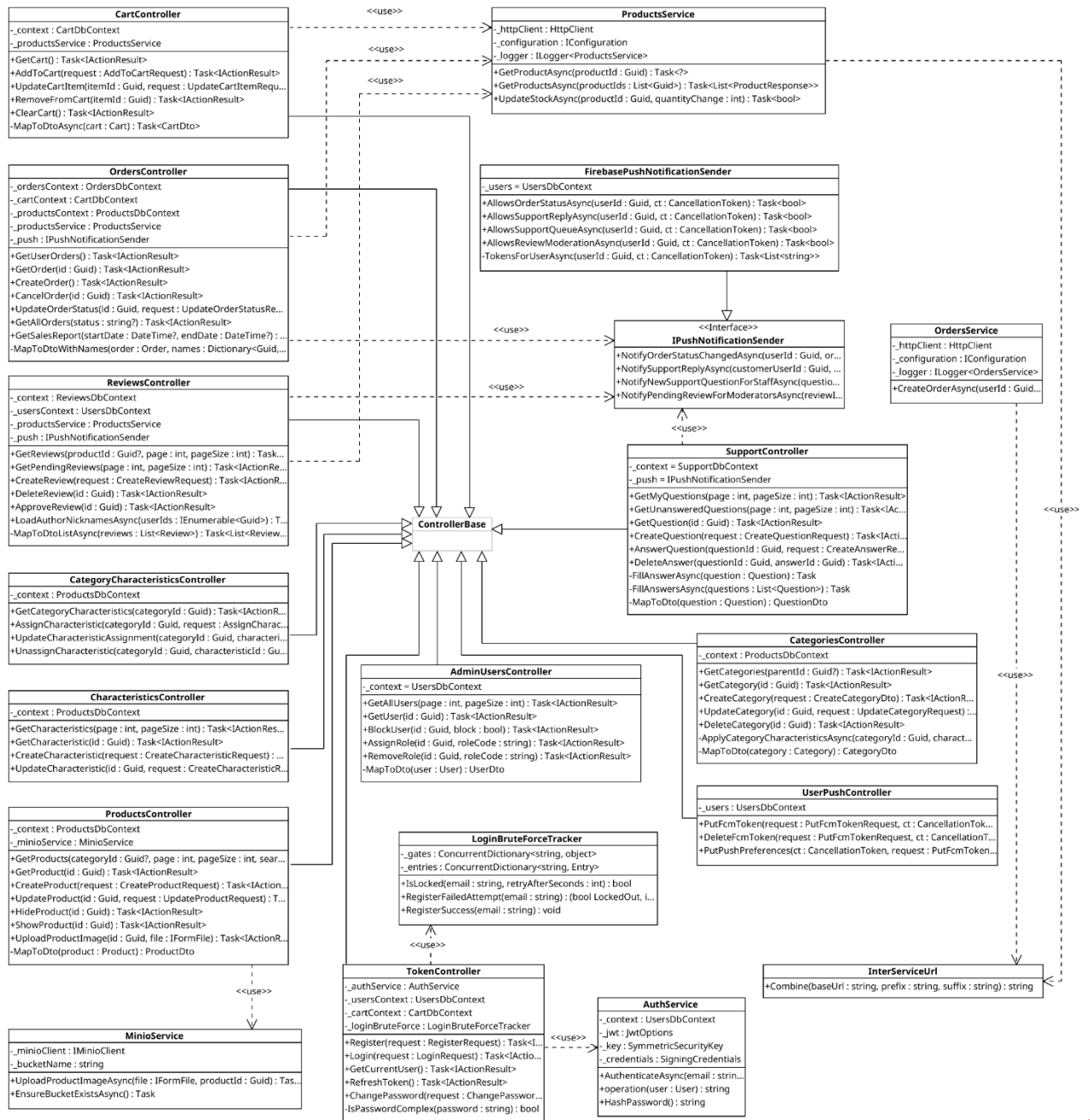


Рисунок Б.2 – Диаграмма классов для уровня бизнес-логики

Приложение В

(обязательное)

Исходный код серверной части приложения

Листинг В.1 – Файл OrdersController.cs

```

namespace Application.Controllers.Orders
{
    [ApiController]
    [Route("api/orders")]
    [Authorize]
    public class OrdersController : ControllerBase
    {
        private readonly OrdersDbContext _ordersContext;
        private readonly CartDbContext _cartContext;
        private readonly ProductsDbContext _productsContext;
        private readonly ProductsService _productsService;
        private readonly IPushNotificationSender _push;

        public OrdersController(
            OrdersDbContext ordersContext,
            CartDbContext cartContext,
            ProductsDbContext productsContext,
            ProductsService productsService,
            IPushNotificationSender push)
        {
            _ordersContext = ordersContext;
            _cartContext = cartContext;
            _productsContext = productsContext;
            _productsService = productsService;
            _push = push;
        }

        /// <summary>
        /// Get user's orders
        /// </summary>
        [HttpGet]
        public async Task<IActionResult> GetUserOrders()
        {
            var userId = AuthToken.GetID(User);
            if (!userId.HasValue)
                return Unauthorized();

            var orders = await _ordersContext.Orders
                .AsNoTracking()
                .Where(o => o.UserId == userId.Value)
                .Include(o => o.Items)
                .Include(o => o.StatusHistory)
                .AsSplitQuery()
                .OrderByDescending(o => o.CreatedAt)
                .ToListAsync();

            var dtos = await MapOrdersToDtosAsync(orders);
            return Ok(dtos);
        }
    }
}

```

```

    /// <summary>
    /// Get single order
    /// </summary>
    [HttpGet("{id}")]
    public async Task<IActionResult> GetOrder(Guid id)
    {
        var userId = AuthToken.GetID(User);
        if (!userId.HasValue)
            return Unauthorized();

        var order = await _ordersContext.Orders
            .AsNoTracking()
            .Include(o => o.Items)
            .Include(o => o.StatusHistory)
            .AsSplitQuery()
            .FirstOrDefaultAsync(o => o.Id == id);

        if (order == null)
            return NotFound();

        // Clients can only see their own orders
        var userRoles = AuthToken.GetRoles(User).ToList();
        if (!userRoles.Contains("MANAGER") &&
            !userRoles.Contains("ADMINISTRATOR") && order.UserId !=
            userId.Value)
            return Forbid();

        var names = await
        ResolveProductNames(order.Items.Select(i => i.ProductId));
        return Ok(MapToDtoWithNames(order, names));
    }

    /// <summary>
    /// Create new order from cart (client)
    /// </summary>
    [HttpPost]
    public async Task<IActionResult> CreateOrder()
    {
        var userId = AuthToken.GetID(User);
        if (!userId.HasValue)
            return Unauthorized();

        await using var transaction = await
        _ordersContext.Database.BeginTransactionAsync();

        _productsContext.Database.UseTransaction(transaction.GetDbTransact
        ion());

        _cartContext.Database.UseTransaction(transaction.GetDbTransaction(
        ));

        try

```

```

        {
            var cart = await _cartContext.Carts
                .Include(c => c.Items)
                .FirstOrDefaultAsync(c => c.UserId ==
userId.Value);

            if (cart == null || !cart.Items.Any())
            {
                await transaction.RollbackAsync();
                return BadRequest(new { message = "Cart is
empty" });
            }

            var order = new Order
            {
                Id = Guid.NewGuid(),
                UserId = userId.Value,
                Status = OrderStatus.Pending,
                CreatedAt = DateTime.UtcNow,
                UpdatedAt = DateTime.UtcNow,
                TotalAmount = 0
            };

            foreach (var cartItem in cart.Items)
            {
                var product = await _productsContext.Products
                    .AsNoTracking()
                    .FirstOrDefaultAsync(p => p.Id ==
cartItem.ProductId);

                if (product == null)
                {
                    await transaction.RollbackAsync();
                    return BadRequest(new { message = "Product
not found" });
                }

                var (decremented, stockError) = await
TryDecrementProductStockAsync(
                    cartItem.ProductId, cartItem.Quantity);
                if (!decremented)
                {
                    await transaction.RollbackAsync();
                    return BadRequest(new { message =
stockError });
                }

                var orderItem = new OrderItem
                {
                    Id = Guid.NewGuid(),
                    OrderId = order.Id,
                    ProductId = product.Id,

```

```

        Quantity = cartItem.Quantity,
        PriceAtPurchase = product.Price
    };

    order.Items.Add(orderItem);
    order.TotalAmount += product.Price *
cartItem.Quantity;
    }

    order.StatusHistory.Add(new OrderStatusHistory
    {
        Id = Guid.NewGuid(),
        OrderId = order.Id,
        Status = OrderStatus.Pending,
        ChangedAt = DateTime.UtcNow
    });

    _ordersContext.Orders.Add(order);
    await _ordersContext.SaveChangesAsync();

    _cartContext.CartItems.RemoveRange(cart.Items);
    cart.UpdatedAt = DateTime.UtcNow;
    await _cartContext.SaveChangesAsync();

    await transaction.CommitAsync();

    var names = await
ResolveProductNames(order.Items.Select(i => i.ProductId));
    return CreatedAtAction(nameof(GetOrder), new { id
= order.Id }, MapToDtoWithNames(order, names));
    }
    catch
    {
        await transaction.RollbackAsync();
        throw;
    }
}

/// <summary>
/// Cancel order (client can cancel pending orders only)
/// </summary>
[HttpPost("{id}/cancel")]
public async Task<IActionResult> CancelOrder(Guid id)
{
    var userId = AuthToken.GetID(User);
    if (!userId.HasValue)
        return Unauthorized();

    var order = await _ordersContext.Orders
        .Include(o => o.Items)
        .FirstOrDefaultAsync(o => o.Id == id);

```

```

        if (order == null)
            return NotFound();

        if (order.UserId != userId.Value)
            return Forbid();

        if (order.Status != OrderStatus.Pending)
            return BadRequest(new { message = "Can only cancel
pending orders" });

        await using var transaction = await
_ordersContext.Database.BeginTransactionAsync();

_productsContext.Database.UseTransaction(transaction.GetDbTransact
ion());

        try
        {
            order.Status = OrderStatus.Cancelled;
            order.UpdatedAt = DateTime.UtcNow;

            foreach (var item in order.Items)
                await RestoreProductStockAsync(item.ProductId,
item.Quantity);

            var statusHistory = new OrderStatusHistory
            {
                Id = Guid.NewGuid(),
                OrderId = order.Id,
                Status = OrderStatus.Cancelled,
                ChangedByUserId = userId.Value,
                ChangedAt = DateTime.UtcNow
            };

_ordersContext.OrderStatusHistories.Add(statusHistory);

            await _productsContext.SaveChangesAsync();
            await _ordersContext.SaveChangesAsync();

            await transaction.CommitAsync();
        }
        catch
        {
            await transaction.RollbackAsync();
            throw;
        }

        order = await _ordersContext.Orders
            .Include(o => o.Items)
            .Include(o => o.StatusHistory)
            .AsSplitQuery()
            .FirstAsync(o => o.Id == id);

```

```

        var namesCancel = await
ResolveProductNames(order.Items.Select(i => i.ProductId));
        return Ok(MapToDtoWithNames(order, namesCancel));
    }

    /// <summary>
    /// Update order status (manager/administrator only)
    /// </summary>
    [Authorize(Roles = "MANAGER,ADMINISTRATOR")]
    [HttpPut("{id}/status")]
    public async Task<IActionResult> UpdateOrderStatus(Guid
id, [FromBody] UpdateOrderStatusRequest request)
    {
        var managerId = AuthToken.GetID(User);
        if (!managerId.HasValue)
            return Unauthorized();

        var order = await _ordersContext.Orders
            .Include(o => o.StatusHistory)
            .Include(o => o.Items)
            .AsSplitQuery()
            .FirstOrDefaultAsync(o => o.Id == id);

        if (order == null)
            return NotFound();

        if (!Enum.TryParse<OrderStatus>(request.Status, true,
out var newStatus))
            return BadRequest(new { message = "Invalid order
status" });

        if (order.Status == OrderStatus.Cancelled)
            return BadRequest(new { message = "Cannot change a
cancelled order" });

        if (newStatus == OrderStatus.Cancelled)
        {
            var noteLen = request.Notes?.Trim().Length ?? 0;
            if (noteLen <= 20)
                return BadRequest(new { message =
"Cancellation requires a note longer than 20 characters" });
        }
        else
        {
            static int PipelineRank(OrderStatus s) => s switch
            {
                OrderStatus.Pending => 0,
                OrderStatus.Processing => 1,
                OrderStatus.ReadyForPickup => 2,
                OrderStatus.Completed => 3,
                OrderStatus.Cancelled => -1,
                _ => -1
            }

```

```

        };

        var currentRank = PipelineRank(order.Status);
        var nextRank = PipelineRank(newStatus);
        if (nextRank <= currentRank)
            return BadRequest(new { message = "Invalid
status transition (cannot revert to an earlier stage)" });
    }

    await using var transaction = await
_ordersContext.Database.BeginTransactionAsync();

_productsContext.Database.UseTransaction(transaction.GetDbTransact
ion());

var statusHistory = new OrderStatusHistory
{
    Id = Guid.NewGuid(),
    OrderId = order.Id,
    Status = newStatus,
    ChangedByUserId = managerId.Value,
    ChangedAt = DateTime.UtcNow,
    Notes = request.Notes
};

try
{
    order.Status = newStatus;
    order.UpdatedAt = DateTime.UtcNow;

    if (newStatus == OrderStatus.Cancelled)
    {
        foreach (var item in order.Items)
            await
RestoreProductStockAsync(item.ProductId, item.Quantity);
    }

_ordersContext.OrderStatusHistories.Add(statusHistory);
    await _ordersContext.SaveChangesAsync();
    await transaction.CommitAsync();
}
catch
{
    await transaction.RollbackAsync();
    throw;
}

await
_push.NotifyOrderStatusChangedAsync(order.UserId, order.Id,
newStatus.ToString(), HttpContext.RequestAborted);

```

```

        var namesStatus = await
ResolveProductNames(order.Items.Select(i => i.ProductId));
        return Ok(MapToDtoWithNames(order, namesStatus));
    }

    /// <summary>
    /// Get all orders (manager/administrator only)
    /// </summary>
    [Authorize(Roles = "MANAGER,ADMINISTRATOR")]
    [HttpGet("admin/all")]
    public async Task<IActionResult> GetAllOrders([FromQuery]
string? status = null)
    {
        var query = _ordersContext.Orders.AsQueryable();

        if (!string.IsNullOrEmpty(status))
        {
            if (!Enum.TryParse<OrderStatus>(status, true, out
var statusEnum))
                return BadRequest(new { message = "Invalid
status filter" });
            query = query.Where(o => o.Status == statusEnum);
        }

        var orders = await query
            .AsNoTracking()
            .Include(o => o.Items)
            .Include(o => o.StatusHistory)
            .AsSplitQuery()
            .OrderByDescending(o => o.CreatedAt)
            .ToListAsync();

        var dtos = await MapOrdersToDtosAsync(orders);
        return Ok(dtos);
    }

    /// <summary>
    /// Sales report: revenue from orders marked Completed in
the period (by status history timestamp). Manager/admin only.
    /// </summary>
    [Authorize(Roles = "MANAGER,ADMINISTRATOR")]
    [HttpGet("reports/sales")]
    public async Task<IActionResult>
GetSalesReport([FromQuery] DateTime? startDate = null, [FromQuery]
DateTime? endDate = null)
    {
        var endDayEnd = endDate.HasValue
            ? UtcEndOfDay(NormalizeToUtc(endDate.Value))
            : UtcEndOfDay(DateTime.UtcNow);

        var startDayStart = startDate.HasValue
            ? UtcStartOfDay(NormalizeToUtc(startDate.Value))

```



```

                                OrderCount: g.Select(x =>
x.OrderId).Distinct().Count()));

    var series = new List<SalesReportPointDto>();
    if (useDailyBuckets)
    {
        for (var d =.UtcStartOfDay(startDayStart); d <=
UtcStartOfDay(endDayEnd); d = d.AddDays(1))
        {
            bucketTotals.TryGetValue(d, out var agg);
            series.Add(new SalesReportPointDto
            {
                PeriodStart = d,
                Revenue = agg.Revenue,
                OrderCount = agg.OrderCount,
            });
        }
    }
    else
    {
        var cursor = MonthStartUtc(startDayStart);
        var last = MonthStartUtc(endDayEnd);
        while (cursor <= last)
        {
            bucketTotals.TryGetValue(cursor, out var agg);
            series.Add(new SalesReportPointDto
            {
                PeriodStart = cursor,
                Revenue = agg.Revenue,
                OrderCount = agg.OrderCount,
            });
            cursor = cursor.AddMonths(1);
        }
    }

    var report = new SalesReportDto
    {
        Period = new SalesReportPeriodDto
        {
            StartDate = startDayStart,
            EndDate = endDayEnd,
        },
        TotalOrders = totalOrders,
        TotalRevenue = totalRevenue,
        AverageOrderValue = averageOrderValue,
        Granularity = granularity,
        Series = series,
    };

    return Ok(report);
}

```

```

private static DateTime NormalizeToUtc(DateTime dt) =>
    dt.Kind switch
    {
        DateTimeKind.Utc => dt,
        DateTimeKind.Local => dt.ToUniversalTime(),
        _ => DateTime.SpecifyKind(dt, DateTimeKind.Utc),
    };

private static DateTime UtcStartOfDay(DateTime utcAny)
{
    var utc = NormalizeToUtc(utcAny);
    return new DateTime(utc.Year, utc.Month, utc.Day, 0,
0, 0, DateTimeKind.Utc);
}

private static DateTime UtcEndOfDay(DateTime utcAny)
{
    return UtcStartOfDay(utcAny).AddDays(1).AddTicks(-1);
}

private static DateTime MonthStartUtc(DateTime utcAny)
{
    var utc = NormalizeToUtc(utcAny);
    return new DateTime(utc.Year, utc.Month, 1, 0, 0, 0,
DateTimeKind.Utc);
}

private async Task<(bool Success, string? ErrorMessage)>
TryDecrementProductStockAsync(
    Guid productId,
    int quantity)
{
    if (quantity <= 0)
        return (false, "Invalid quantity");

    var updated = await _productsContext.Products
        .Where(p => p.Id == productId && !p.IsHidden &&
p.Stock >= quantity)
        .ExecuteUpdateAsync(setters => setters
            .SetProperty(p => p.Stock, p => p.Stock -
quantity)
            .SetProperty(p => p.UpdatedAt,
DateTime.UtcNow));

    if (updated == 1)
        return (true, null);

    var product = await _productsContext.Products
        .AsNoTracking()
        .Where(p => p.Id == productId)
        .Select(p => new { p.Name, p.Stock, p.IsHidden })
        .FirstOrDefaultAsync();

```

```

        if (product == null)
            return (false, "Product not found");
        if (product.IsHidden)
            return (false, $"Product \"{product.Name}\" is not
available");
        return (false,
            $"Product \"{product.Name}\" is not available in
requested quantity (in stock: {product.Stock})");
    }

    private Task RestoreProductStockAsync(Guid productId, int
quantity) =>
        _productsContext.Products
            .Where(p => p.Id == productId)
            .ExecuteUpdateAsync(setters => setters
                .SetProperty(p => p.Stock, p => p.Stock +
quantity)
                .SetProperty(p => p.UpdatedAt,
DateTime.UtcNow));

    private async Task<Dictionary<Guid, string>>
ResolveProductNames(IEnumerable<Guid> productIds)
    {
        var idList = productIds.Distinct().ToList();
        if (idList.Count == 0)
            return new Dictionary<Guid, string>();

        var products = await
_productsService.GetProductsAsync(idList);
        if (products == null || products.Count == 0)
            return new Dictionary<Guid, string>();

        return products.ToDictionary(p => p.Id, p => p.Name);
    }

    private async Task<List<OrderDto>>
MapOrdersToDtosAsync(List<Order> orders)
    {
        var allIds = orders.SelectMany(o => o.Items ??
Array.Empty<OrderItem>()).Select(i => i.ProductId);
        var names = await ResolveProductNames(allIds);
        return orders.Select(o => MapToDtoWithNames(o,
names)).ToList();
    }

    private static OrderDto MapToDtoWithNames(Order order,
Dictionary<Guid, string> names)
    {
        var items = order.Items ?? Array.Empty<OrderItem>();
        var latestHistory = order.StatusHistory?
            .OrderByDescending(h => h.ChangedAt)

```

```

        .FirstOrDefault();
return new OrderDto
{
    Id = order.Id,
    UserId = order.UserId,
    LastStatusChangedByUserId =
latestHistory?.ChangedByUserId,
    TotalAmount = order.TotalAmount,
    Status = order.Status.ToString(),
    CreatedAt = order.CreatedAt,
    UpdatedAt = order.UpdatedAt,
    Items = items.Select(oi => new OrderItemDto
    {
        Id = oi.Id,
        ProductId = oi.ProductId,
        ProductName =
names.GetValueOrDefault(oi.ProductId),
        Quantity = oi.Quantity,
        PriceAtPurchase = oi.PriceAtPurchase
    }).ToList()
};
    }
}
}

```

Приложение Г

(обязательное)

Исходный код клиентской части приложения

Листинг Г.1 – Файл AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.POST_NOTIFICATIONS" />

    <application
        android:name=".ElectronixApplication"
        android:allowBackup="true"

        android:networkSecurityConfig="@xml/network_security_config"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ElectronixAndroid"
        tools:targetApi="34">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:windowSoftInputMode="adjustResize"
            android:label="@string/app_name"
            android:theme="@style/Theme.ElectronixAndroid">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"

/>

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service

android:name=".push.ElectronixFirebaseMessagingService"
            android:exported="false">
            <intent-filter>
                <action
android:name="com.google.firebase.MESSAGING_EVENT" />
            </intent-filter>
        </service>
    </application>

</manifest>

```

Листинг Г.2 – Файл ElectronixApi.kt

```
package com.pnzgu.electronix.data.api

@Suppress("ComplexInterface")
interface ElectronixApi {
    @POST("api/users/login")
    suspend fun login(@Body body: LoginRequest): LoginResponse

    @POST("api/users/register")
    suspend fun register(@Body body: RegisterRequest):
    RegisterResponse

    @GET("api/users/me")
    suspend fun me(): UserDto

    @POST("api/users/refresh")
    suspend fun refresh(): RefreshResponse

    @POST("api/users/change-password")
    suspend fun changePassword(@Body body: ChangePasswordRequest):
    MessageResponse

    @PUT("api/users/me/fcm-token")
    suspend fun putFcmToken(@Body body: PutFcmTokenRequest)

    @HTTP(method = "DELETE", path = "api/users/me/fcm-token",
    hasBody = true)
    suspend fun deleteFcmToken(@Body body: PutFcmTokenRequest)

    @PUT("api/users/me/push-preferences")
    suspend fun putPushPreferences(@Body body:
    PutPushPreferencesRequest)

    @GET("api/admin/users")
    suspend fun adminUsers(
        @Query("page") page: Int,
        @Query("pageSize") pageSize: Int,
    ): PagedUsersResponse

    @GET("api/admin/users/{id}")
    suspend fun adminUser(@Path("id") id: String): UserDto

    @POST("api/admin/users/{id}/block")
    suspend fun adminBlockUser(@Path("id") id: String, @Body
    block: Boolean): MessageResponse

    @POST("api/admin/users/{id}/roles")
    suspend fun adminAssignRole(@Path("id") id: String, @Body
    body: RequestBody): MessageResponse

    @DELETE("api/admin/users/{id}/roles/{roleCode}")
```

```

suspend fun adminRemoveRole(
    @Path("id") id: String,
    @Path("roleCode") roleCode: String,
): MessageResponse

@GET("api/categories")
suspend fun categories(@Query("parentId") parentId: String?):
List<CategoryDto>

@GET("api/categories/{id}")
suspend fun category(@Path("id") id: String): CategoryDto

@POST("api/categories")
suspend fun createCategory(@Body body: CreateCategoryDto):
CategoryDto

@PUT("api/categories/{id}")
suspend fun updateCategory(@Path("id") id: String, @Body body:
UpdateCategoryRequest): CategoryDto

@DELETE("api/categories/{id}")
suspend fun deleteCategory(@Path("id") id: String):
MessageResponse

@GET("api/categories/{categoryId}/characteristics")
suspend fun categoryCharacteristics(@Path("categoryId")
categoryId: String): List<CategoryCharacteristicDto>

@POST("api/categories/{categoryId}/characteristics")
suspend fun assignCategoryCharacteristic(
    @Path("categoryId") categoryId: String,
    @Body body: AssignCharacteristicRequest,
): CategoryCharacteristicDto

@PUT("api/categories/{categoryId}/characteristics/{characteristicI
d}")
suspend fun updateCategoryCharacteristicAssignment(
    @Path("categoryId") categoryId: String,
    @Path("characteristicId") characteristicId: String,
    @Body body: UpdateCharacteristicAssignmentRequest,
): CategoryCharacteristicDto

@DELETE("api/categories/{categoryId}/characteristics/{characterist
icId}")
suspend fun unassignCategoryCharacteristic(
    @Path("categoryId") categoryId: String,
    @Path("characteristicId") characteristicId: String,
): MessageResponse

@GET("api/characteristics")

```

```

suspend fun characteristics(
    @Query("page") page: Int,
    @Query("pageSize") pageSize: Int,
): PagedCharacteristicsResponse

@GET("api/characteristics/{id}")
suspend fun characteristic(@Path("id") id: String):
CharacteristicDto

@POST("api/characteristics")
suspend fun createCharacteristic(@Body body:
CreateCharacteristicRequest): CharacteristicDto

@PUT("api/characteristics/{id}")
suspend fun updateCharacteristic(
    @Path("id") id: String,
    @Body body: CreateCharacteristicRequest,
): CharacteristicDto

@GET("api/products")
suspend fun products(
    @Query("categoryId") categoryId: String?,
    @Query("page") page: Int,
    @Query("pageSize") pageSize: Int,
    @Query("search") search: String? = null,
    @QueryMap(encoded = true) filters: Map<String, String> =
emptyMap(),
): PagedProductsResponse

@GET("api/products/{id}")
suspend fun product(@Path("id") id: String): ProductDto

@POST("api/products")
suspend fun createProduct(@Body body: CreateProductRequest):
ProductDto

@PUT("api/products/{id}")
suspend fun updateProduct(@Path("id") id: String, @Body body:
UpdateProductRequest): ProductDto

@POST("api/products/{id}/hide")
suspend fun hideProduct(@Path("id") id: String):
MessageResponse

@POST("api/products/{id}/show")
suspend fun showProduct(@Path("id") id: String):
MessageResponse

@Multipart
@POST("api/products/{id}/upload-image")
suspend fun uploadProductImage(
    @Path("id") id: String,

```

```

        @Part file: MultipartBody.Part,
    ): UploadImageResponse

    @GET("api/cart")
    suspend fun cart(): CartDto

    @POST("api/cart/items")
    suspend fun addCartItem(@Body body: AddToCartRequest): CartDto

    @PUT("api/cart/items/{itemId}")
    suspend fun updateCartItem(
        @Path("itemId") itemId: String,
        @Body body: UpdateCartItemRequest,
    ): CartDto

    @DELETE("api/cart/items/{itemId}")
    suspend fun removeCartItem(@Path("itemId") itemId: String):
    CartDto

    @DELETE("api/cart")
    suspend fun clearCart(): retrofit2.Response<Unit>

    @GET("api/orders")
    suspend fun myOrders(): List<OrderDto>

    @GET("api/orders/{id}")
    suspend fun order(@Path("id") id: String): OrderDto

    @POST("api/orders")
    suspend fun createOrder(): OrderDto

    @POST("api/orders/{id}/cancel")
    suspend fun cancelOrder(@Path("id") id: String): OrderDto

    @GET("api/orders/admin/all")
    suspend fun adminOrders(@Query("status") status: String? =
    null): List<OrderDto>

    @PUT("api/orders/{id}/status")
    suspend fun updateOrderStatus(
        @Path("id") id: String,
        @Body body: UpdateOrderStatusRequest,
    ): OrderDto

    @GET("api/orders/reports/sales")
    suspend fun salesReport(
        @Query("startDate") startDate: String? = null,
        @Query("endDate") endDate: String? = null,
    ): SalesReportDto

    @GET("api/reviews")
    suspend fun reviews(

```

```

        @Query("productId") productId: String? = null,
        @Query("page") page: Int,
        @Query("pageSize") pageSize: Int,
    ): PagedReviewsResponse

    @GET("api/reviews/pending")
    suspend fun reviewsPending(
        @Query("page") page: Int,
        @Query("pageSize") pageSize: Int,
    ): PagedReviewsResponse

    @POST("api/reviews")
    suspend fun createReview(@Body body: CreateReviewRequest):
ReviewDto

    @DELETE("api/reviews/{id}")
    suspend fun deleteReview(@Path("id") id: String):
MessageResponse

    @POST("api/reviews/{id}/approve")
    suspend fun approveReview(@Path("id") id: String): ReviewDto

    @GET("api/support/questions")
    suspend fun myQuestions(
        @Query("page") page: Int,
        @Query("pageSize") pageSize: Int,
    ): PagedQuestionsResponse

    @GET("api/support/questions/unanswered")
    suspend fun unansweredQuestions(
        @Query("page") page: Int,
        @Query("pageSize") pageSize: Int,
    ): PagedQuestionsResponse

    @GET("api/support/questions/{id}")
    suspend fun question(@Path("id") id: String): QuestionDto

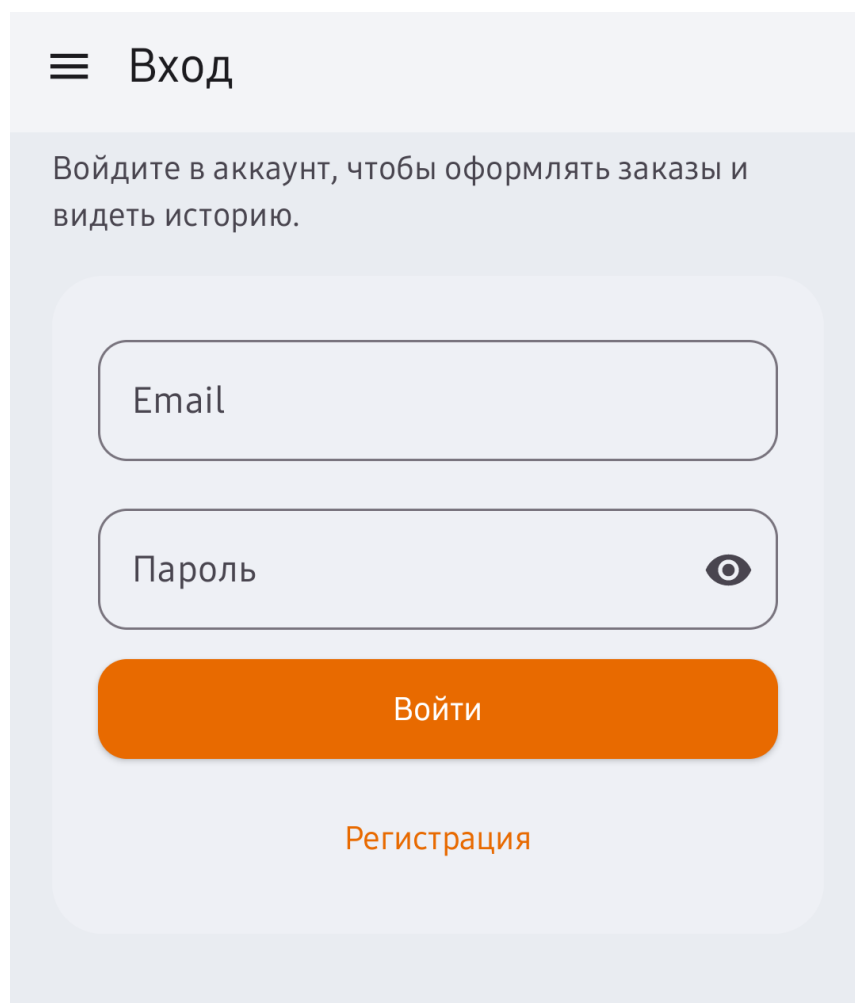
    @POST("api/support/questions")
    suspend fun createQuestion(@Body body: CreateQuestionRequest):
QuestionDto

    @POST("api/support/questions/{questionId}/answer")
    suspend fun answerQuestion(
        @Path("questionId") questionId: String,
        @Body body: CreateAnswerRequest,
    ): QuestionDto

    @DELETE("api/support/questions/{questionId}/answers/{answerId}")
    suspend fun deleteAnswer(
        @Path("questionId") questionId: String,
        @Path("answerId") answerId: String,

```

Приложение Д
(рекомендуемое)
Результаты тестирования



≡ Вход

Войдите в аккаунт, чтобы оформлять заказы и видеть историю.

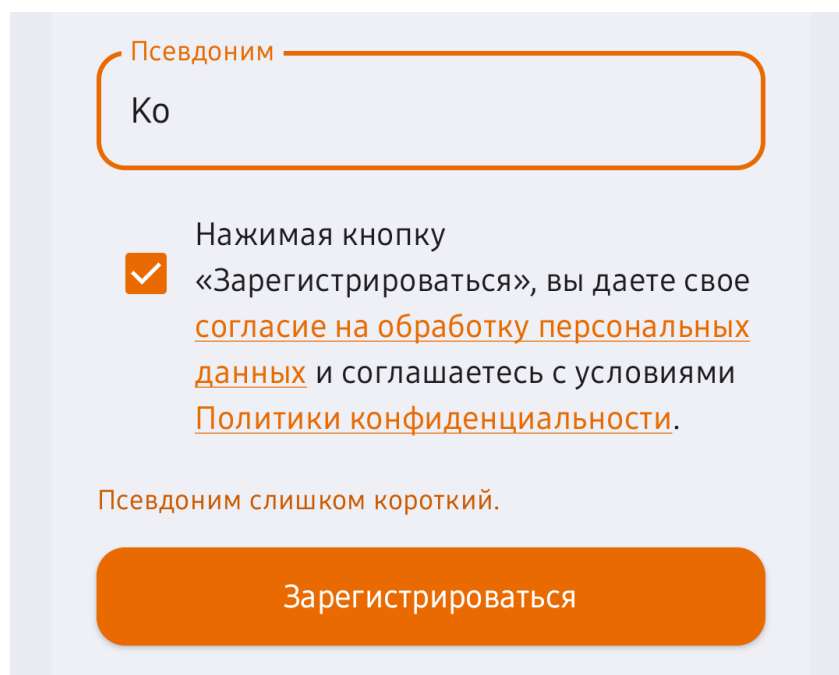
Email

Пароль 

Войти

Регистрация

Рисунок Д.1 – Результат выполнения теста №1



Псевдоним

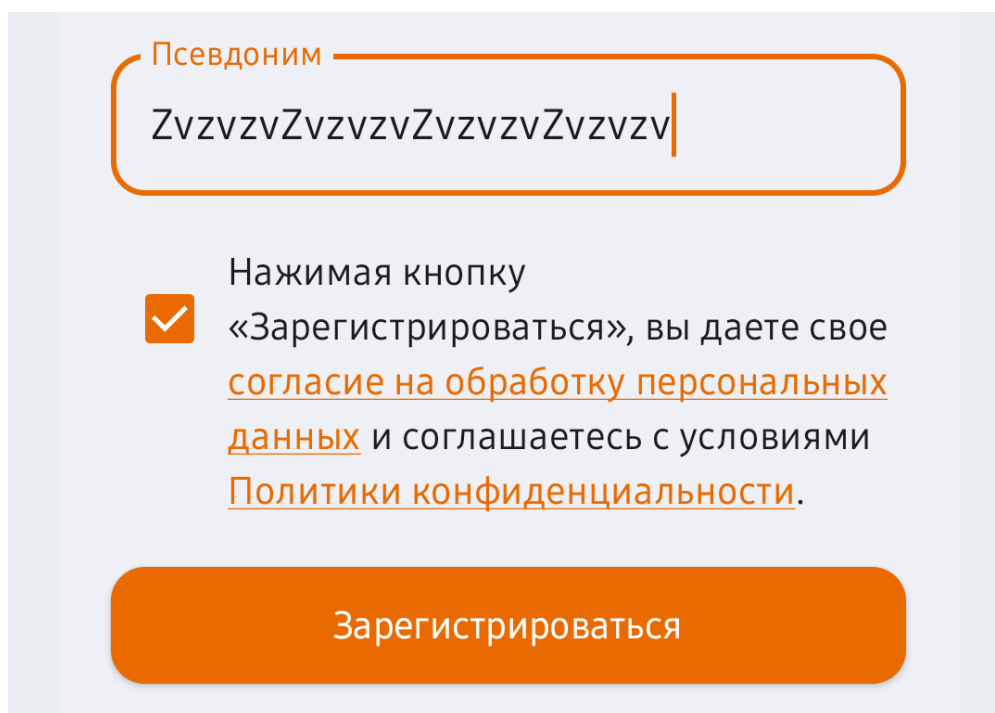
Ко

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Псевдоним слишком короткий.

Зарегистрироваться

Рисунок Д.2 – Результат выполнения теста №2



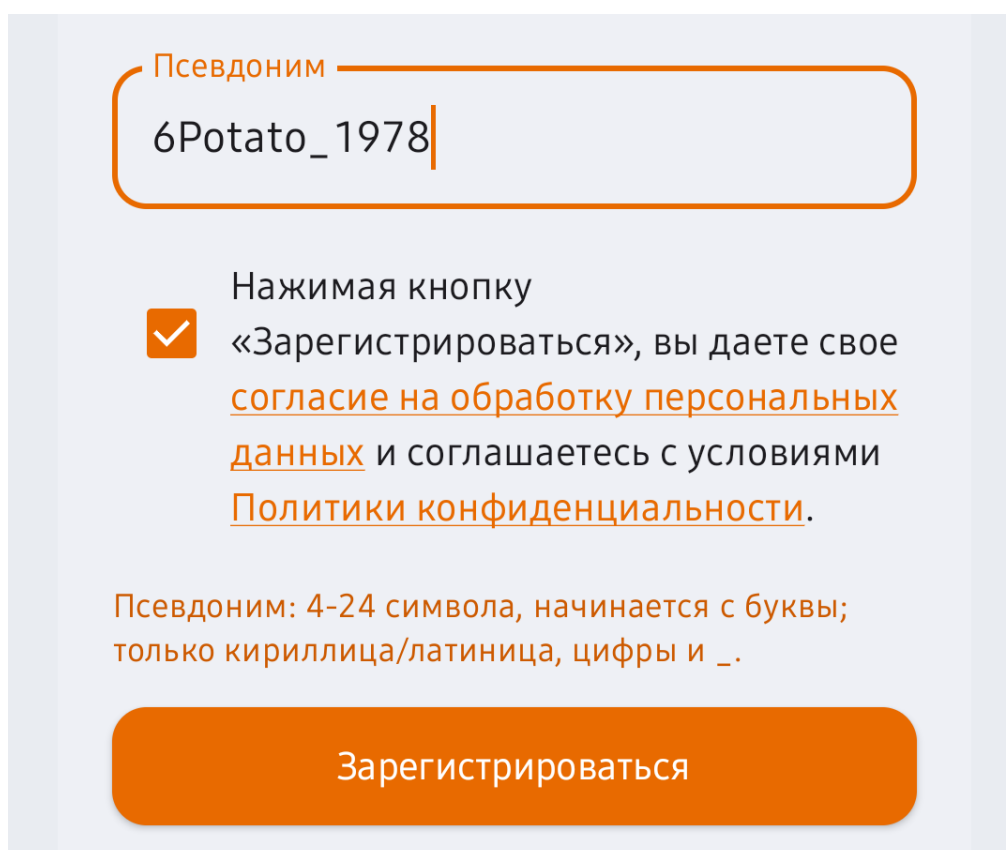
Псевдоним

ZvzvzvZvzvzvZvzvzvZvzvzv|

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Зарегистрироваться

Рисунок Д.3 – Результат выполнения теста №3



Псевдоним

6Potato_1978|

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Псевдоним: 4-24 символа, начинается с буквы; только кириллица/латиница, цифры и _.

Зарегистрироваться

Рисунок Д.4 – Результат выполнения теста №4

Псевдоним

Шмидт Старший

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Псевдоним не может содержать спецсимволы кроме нижнего подчеркивания.

Зарегистрироваться

Рисунок Д.5 – Результат выполнения теста №5

Псевдоним

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Псевдоним не должен быть пустым.

Зарегистрироваться

Рисунок Д.6 – Результат выполнения теста №6

Псевдоним

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Псевдоним не должен содержать нестандартные символы UTF-8.

Рисунок Д.7 – Результат выполнения теста №7

Псевдоним

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Псевдоним слишком короткий.

Рисунок Д.8 – Результат выполнения теста №8

Псевдоним

@#!@#\$\$\$@#%

☒ Нажимая кнопку «Зарегистрироваться», вы даете свое согласие на обработку персональных данных и соглашаетесь с условиями Политики конфиденциальности.

Псевдоним не может содержать спецсимволы кроме нижнего подчеркивания.

Зарегистрироваться

Рисунок Д.9 – Результат выполнения теста №9

Приложение Е
(рекомендуемое)
Исходный код модульных тестов

Листинг Е.1 – Файл DtoMappingTest.cs

```

using System.Collections.Generic;
using Application.DTOs.Orders;
using Domain.Orders;

namespace Tests.UnitTests;

/// <summary>
/// Unit tests for DTO mapping and validation
/// </summary>
public class OrderDtoMappingTests
{
    [Fact]
    public void SalesReportDto_ShouldInitializeWithValidValues()
    {
        // Arrange & Act
        var report = new SalesReportDto
        {
            Period = new SalesReportPeriodDto
            {
                StartDate = DateTime.UtcNow.AddMonths(-1),
                EndDate = DateTime.UtcNow
            },
            TotalOrders = 5,
            TotalRevenue = 500.00m,
            AverageOrderValue = 100.00m,
            Granularity = "day",
            Series = new List<SalesReportPointDto>(),
        };

        // Assert
        report.Period.Should().NotNull();
        report.TotalOrders.Should().Be(5);
        report.TotalRevenue.Should().Be(500.00m);
        report.AverageOrderValue.Should().Be(100.00m);
    }

    [Fact]
    public void SalesReportPeriodDto_ShouldHaveValidDateRange()
    {
        // Arrange
        var startDate = new DateTime(2025, 01, 01, 0, 0, 0,
            DateTimeKind.Utc);
        var endDate = new DateTime(2025, 12, 31, 23, 59, 59,
            DateTimeKind.Utc);

        // Act
        var period = new SalesReportPeriodDto
        {
            StartDate = startDate,
            EndDate = endDate
        }
    }
}

```

```

    };

    // Assert
    period.StartDate.Should().Be(startDate);
    period.EndDate.Should().Be(endDate);
    period.EndDate.Should().BeOnOrAfter(period.StartDate);
}

[Fact]
public void OrderDto_ShouldMapFromDomainOrder()
{
    // Arrange
    var order = new Order
    {
        Id = Guid.NewGuid(),
        UserId = Guid.NewGuid(),
        TotalAmount = 199.99m,
        Status = OrderStatus.Completed,
        CreatedAt = DateTime.UtcNow,
        UpdatedAt = DateTime.UtcNow
    };

    // Act
    var orderDto = new OrderDto
    {
        Id = order.Id,
        UserId = order.UserId,
        TotalAmount = order.TotalAmount,
        Status = order.Status.ToString(),
        CreatedAt = order.CreatedAt,
        UpdatedAt = order.UpdatedAt
    };

    // Assert
    orderDto.Id.Should().Be(order.Id);
    orderDto.UserId.Should().Be(order.UserId);
    orderDto.TotalAmount.Should().Be(order.TotalAmount);
    orderDto.Status.Should().Be("Completed");
}

[Fact]
public void UpdateOrderStatusRequest_ShouldValidateStatus()
{
    // Arrange
    var validStatuses = new[] { "Pending", "Processing",
    "ReadyForPickup", "Completed", "Cancelled" };

    // Act & Assert
    foreach (var status in validStatuses)
    {
        var request = new UpdateOrderStatusRequest { Status =
status };
    }
}

```

```

        request.Status.Should().NotBeNullOrEmpty();
        Enum.TryParse<OrderStatus>(status, true, out
_).Should().BeTrue();
    }
}

[Fact]
public void OrderItemDto_ShouldContainRequiredProperties()
{
    // Arrange & Act
    var itemDto = new OrderItemDto
    {
        Id = Guid.NewGuid(),
        ProductId = Guid.NewGuid(),
        Quantity = 5,
        PriceAtPurchase = 49.99m
    };

    // Assert
    itemDto.Id.Should().NotBe(Guid.Empty);
    itemDto.ProductId.Should().NotBe(Guid.Empty);
    itemDto.Quantity.Should().Be(5);
    itemDto.PriceAtPurchase.Should().Be(49.99m);
}

[Theory]
[InlineData(1, 99.99, 99.99)]
[InlineData(5, 20.00, 100.00)]
[InlineData(10, 15.50, 155.00)]
public void OrderItem_ShouldCalculateTotalPrice(int quantity,
decimal pricePerUnit, decimal expectedTotal)
{
    // Arrange & Act
    var totalPrice = quantity * pricePerUnit;

    // Assert
    totalPrice.Should().Be(expectedTotal);
}
}

```

Листинг E.2 – Файл OrderDomainTests.cs

```

using Domain.Orders;

namespace Tests.UnitTests;

/// <summary>
/// Unit tests for Domain models and enums
/// </summary>
public class OrderDomainTests
{
    [Fact]
    public void OrderStatus_ShouldHaveValidValues()

```

```

    {
        // Arrange & Act & Assert
        OrderStatus.Pending.Should().Be(OrderStatus.Pending);

        OrderStatus.Processing.Should().Be(OrderStatus.Processing);

        OrderStatus.ReadyForPickup.Should().Be(OrderStatus.ReadyForPickup)
    ;
        OrderStatus.Completed.Should().Be(OrderStatus.Completed);
        OrderStatus.Cancelled.Should().Be(OrderStatus.Cancelled);
    }

    [Fact]
    public void Order_ShouldInitializeWithDefaultValues()
    {
        // Arrange & Act
        var order = new Order();

        // Assert
        order.Id.Should().Be(Guid.Empty);
        order.UserId.Should().Be(Guid.Empty);
        order.TotalAmount.Should().Be(0);
        order.Status.Should().Be(OrderStatus.Pending);
        order.Items.Should().BeEmpty();
        order.StatusHistory.Should().BeEmpty();
    }

    [Fact]
    public void Order_ShouldAddItems()
    {
        // Arrange
        var order = new Order { Id = Guid.NewGuid(), UserId =
Guid.NewGuid() };
        var item = new OrderItem
        {
            Id = Guid.NewGuid(),
            OrderId = order.Id,
            ProductId = Guid.NewGuid(),
            Quantity = 2,
            PriceAtPurchase = 99.99m
        };

        // Act
        order.Items.Add(item);

        // Assert
        order.Items.Should().HaveCount(1);
        order.Items.First().Should().Be(item);
    }

    [Fact]
    public void OrderStatusHistory_ShouldTrackStatusChanges()

```

```

{
    // Arrange
    var order = new Order { Id = Guid.NewGuid() };
    var statusChange = new OrderStatusHistory
    {
        Id = Guid.NewGuid(),
        OrderId = order.Id,
        Status = OrderStatus.Processing,
        ChangedAt = DateTime.UtcNow
    };

    // Act
    order.StatusHistory.Add(statusChange);
    order.Status = OrderStatus.Processing;

    // Assert
    order.StatusHistory.Should().HaveCount(1);
    order.Status.Should().Be(OrderStatus.Processing);
}

[Theory]
[InlineData(0)]
[InlineData(1)]
[InlineData(2)]
[InlineData(3)]
[InlineData(4)]
public void OrderStatus_ShouldHaveValidEnumValues(int
statusValue)
{
    // Arrange & Act
    var status = (OrderStatus)statusValue;

    // Assert
    Enum.IsDefined(typeof(OrderStatus),
status).Should().BeTrue();
}
}

```

Листинг Е.3 – Файл ProductDomainTests.cs

```

using Domain.Products;

namespace Tests.UnitTests;

/// <summary>
/// Unit tests for Product domain models
/// </summary>
public class ProductDomainTests
{
    [Fact]
    public void Product_ShouldInitializeWithDefaultValues()
    {
        // Arrange & Act

```

```

var product = new Product();

// Assert
product.Id.Should().Be(Guid.Empty);
product.Name.Should().BeNullOrEmpty();
product.Price.Should().Be(0);
product.Stock.Should().Be(0);
product.CharacteristicValues.Should().BeEmpty();
product.Images.Should().BeEmpty();
}

[Fact]
public void Product_ShouldSetPropertiesCorrectly()
{
    // Arrange
    var productId = Guid.NewGuid();
    var categoryId = Guid.NewGuid();

    // Act
    var product = new Product
    {
        Id = productId,
        Name = "Test Product",
        Description = "A test product",
        Price = 99.99m,
        Stock = 50,
        CategoryId = categoryId,
        CreatedAt = DateTime.UtcNow
    };

    // Assert
    product.Id.Should().Be(productId);
    product.Name.Should().Be("Test Product");
    product.Description.Should().Be("A test product");
    product.Price.Should().Be(99.99m);
    product.Stock.Should().Be(50);
    product.CategoryId.Should().Be(categoryId);
}

[Fact]
public void Product_ShouldAddCharacteristicValues()
{
    // Arrange
    var product = new Product { Id = Guid.NewGuid(), Name =
"Test" };
    var characteristicId = Guid.NewGuid();
    var characteristicValue = new ProductCharacteristicValue
    {
        Id = Guid.NewGuid(),
        ProductId = product.Id,
        CharacteristicId = characteristicId,
        Value = 42.5,
    };
}

```

```

        Characteristic = new Characteristic
        {
            Id = characteristicId,
            Name = "Power",
            Unit = "W",
        },
    };

    // Act
    product.CharacteristicValues.Add(characteristicValue);

    // Assert
    product.CharacteristicValues.Should().HaveCount(1);

    product.CharacteristicValues.First().Characteristic.Name.Should().
    Be("Power");

    product.CharacteristicValues.First().Value.Should().Be(42.5);
}

[Fact]
public void Product_ShouldAddImages()
{
    // Arrange
    var product = new Product { Id = Guid.NewGuid(), Name =
    "Test" };
    var image = new ProductImage
    {
        Id = Guid.NewGuid(),
        ProductId = product.Id,
        ObjectName = "test/image.jpg",
        UploadedAt = DateTime.UtcNow
    };

    // Act
    product.Images.Add(image);

    // Assert
    product.Images.Should().HaveCount(1);

    product.Images.First().ObjectName.Should().Be("test/image.jpg");
}

[Theory]
[InlineData(0)]
[InlineData(-1)]
[InlineData(1000)]
public void Product_ShouldAcceptVariousStockLevels(int stock)
{
    // Arrange & Act
    var product = new Product { Stock = stock };

```

```

        // Assert
        product.Stock.Should().Be(stock);
    }

    [Theory]
    [InlineData(0.01)]
    [InlineData(99.99)]
    [InlineData(1000.00)]
    public void Product_ShouldAcceptVariousPrices(decimal price)
    {
        // Arrange & Act
        var product = new Product { Price = price };

        // Assert
        product.Price.Should().Be(price);
    }

    [Fact]
    public void
    ProductCharacteristicValue_ShouldStoreValueWithCharacteristic()
    {
        // Arrange & Act
        var characteristicId = Guid.NewGuid();
        var value = new ProductCharacteristicValue
        {
            Id = Guid.NewGuid(),
            CharacteristicId = characteristicId,
            Value = 15.7,
            Characteristic = new Characteristic
            {
                Id = characteristicId,
                Name = "Weight",
                Unit = "kg",
            },
        };

        // Assert
        value.Characteristic.Name.Should().Be("Weight");
        value.Characteristic.Unit.Should().Be("kg");
        value.Value.Should().Be(15.7);
    }

    [Fact]
    public void ProductImage_ShouldStoreImageMetadata()
    {
        // Arrange
        var imageId = Guid.NewGuid();
        var productId = Guid.NewGuid();
        var createdAt = DateTime.UtcNow;

        // Act
        var image = new ProductImage

```

```

        {
            Id = imageId,
            ProductId = productId,
            ObjectName = "products/123/image-uuid.jpg",
            UploadedAt = createdAt
        };

        // Assert
        image.Id.Should().Be(imageId);
        image.ProductId.Should().Be(productId);
        image.ObjectName.Should().Be("products/123/image-
uuid.jpg");
        image.UploadedAt.Should().Be(createdAt);
    }
}

```

Листинг Е.4 – Файл ValidationTests.cs

```

using Domain.Products;
using Domain.Users;

namespace Tests.UnitTests;

/// <summary>
/// Unit tests for validation and business logic
/// </summary>
public class ValidationTests
{
    [Theory]
    [InlineData(null)]
    [InlineData("")]
    [InlineData(" ")]
    public void
Product_WithNullOrEmptyName_ShouldHaveEmptyName(string? name)
    {
        // Act
        var product = new Product { Name = name ?? "" };

        // Assert
        string.IsNullOrEmpty(product.Name).Should().BeTrue();
    }

    [Fact]
    public void Product_WithNegativePrice_ShouldInitialize()
    {
        // Arrange & Act
        var product = new Product { Price = -10m };

        // Assert
        product.Price.Should().Be(-10m);
        // Note: Real validation should prevent this in business
        logic
    }
}

```

```

[Fact]
public void
Product_WithNegativeStock_ShouldAllowInitialization()
{
    // Arrange & Act
    var product = new Product { Stock = -5 };

    // Assert
    product.Stock.Should().Be(-5);
    // Note: Real validation should prevent overselling
}

[Fact]
public void Order_CannotHaveNullItems()
{
    // Arrange & Act
    var order = new Domain.Orders.Order();

    // Assert
    order.Items.Should().NotBeNull();
    order.Items.Should().BeEmpty();
}

[Fact]
public void OrderItem_ShouldHaveValidQuantity()
{
    // Arrange & Act
    var item = new Domain.Orders.OrderItem { Quantity = 5 };

    // Assert
    item.Quantity.Should().Be(5);
}

[Fact]
public void OrderItem_WithZeroQuantity_ShouldInitialize()
{
    // Arrange & Act
    var item = new Domain.Orders.OrderItem { Quantity = 0 };

    // Assert
    item.Quantity.Should().Be(0);
    // Note: Real validation should prevent zero quantity
orders
}

[Theory]
[InlineData("user@example.com")]
[InlineData("test.user@domain.co.uk")]
[InlineData("name+tag@example.com")]
public void User_ShouldAcceptVariousEmailFormats(string email)
{

```

```

        // Arrange & Act
        var user = new User { Email = email };

        // Assert
        user.Email.Should().Be(email);
    }

    [Fact]
    public void Category_ShouldInitializeWithEmptyProducts()
    {
        // Arrange & Act
        var category = new Category();

        // Assert
        category.Products.Should().BeEmpty();
        category.Products.Should().NotNull();
    }

    [Fact]
    public void ProductCharacteristicValue_ShouldHaveCharacteristicAndValue()
    {
        // Arrange & Act
        var characteristicId = Guid.NewGuid();
        var value = new ProductCharacteristicValue
        {
            CharacteristicId = characteristicId,
            Value = 100,
            Characteristic = new Characteristic
            {
                Id = characteristicId,
                Name = "Battery capacity",
                Unit = "mAh",
            },
        };

        // Assert
        value.Characteristic.Name.Should().NotNullOrEmpty();
        value.Characteristic.Unit.Should().NotNullOrEmpty();
        value.Value.Should().BeGreaterThan(0);
    }

    [Fact]
    public void Guid_ShouldNotBeEmptyForValidEntities()
    {
        // Arrange & Act
        var id = Guid.NewGuid();
        var product = new Product { Id = id };

        // Assert
        product.Id.Should().NotBe(Guid.Empty);
        product.Id.Should().Be(id);
    }

```

```
}

[Fact]
public void DateTime_ShouldBeUtcInDomainEntities()
{
    // Arrange
    var order = new Domain.Orders.Order
    {
        CreatedAt = DateTime.UtcNow,
        UpdatedAt = DateTime.UtcNow
    };

    // Assert
    order.CreatedAt.Kind.Should().Be(DateTimeKind.Utc);
    order.UpdatedAt.Kind.Should().Be(DateTimeKind.Utc);
}
}
```