

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 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Информационная система фирмы по продаже и
подбору автозапчастей**

Студент _____ Гордиенко Алексей Павлович
(подпись, дата) (ФИО полностью)

Руководитель _____ Кольчугина Е.А.
(подпись, дата) (фамилия, инициалы)

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

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

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 Гордиенко Алексей Павлович гр. 22ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Информационная система фирмы по продаже и подбору автозапчастей

Тема утверждена приказом ПГУ № 1583/о от "30" декабря 2025г.

3. Руководитель работы д.т.н., доцент кафедры «МО и ПЭВМ» Кольчугина Е.А.

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

1) Назначение: *информационная система фирмы по продаже и подбору автозапчастей*

2) Среда выполнения: *операционная система Windows*

Языковые средства разработки: Java, React, PostgreSQL

Инструментальные средства: IntelliJIdea, VSCode, DbBrowser

Технологии разработки: микросервисная архитектура, ООП.

3) Основные функции приложения:

а) регистрация и авторизация пользователей;

б) подбор комплектующих для транспортного средства;

в) добавление комплектующих при поступлении на склад;

г) оформление покупки после подбора комплектующих;

д) формирование статистики о продажах.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и требований к системе;

2) Проектирование программных средств;

3) Реализация программных средств;

4) Тестирование программных средств;

5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	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 89 листов, 26 рисунков, 20 таблиц, 17 использованных источников и 2 приложения.

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, REST, SPRING BOOT, POSTGRESQL, REACTJS, БАЗА ДАННЫХ, WEB.

Объект исследования – информационная система фирмы по продаже и подбору автозапчастей.

Цель выпускной квалификационной работы – разработать информационную систему в виде web – приложения для фирмы по продаже и подбору автозапчастей.

Технологии разработки – разработка клиентской части приложения проводилась на языке программирования JavaScript с использованием библиотеки ReactJS в среде разработки VSCode. Разработка серверной части приложения производилась на языке программирования Java17 в среде разработки IntelliJ idea.. Разработка базы данных проводилась в СУБД PostgreSQL.

Среда исполнения – Windows, web-приложение.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть информационной системы фирмы по подбору и продаже автозапчастей. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.02 – 08BP221.06 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система фирмы по продаже и подбору автозапчастей. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Гордиенко А.П.					4	89
Пров.		Кольчугина Е.А.				Группа 22ВИ1		
Реценз.								
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановки задачи на разработку информационной системы по продаже и подбору автозапчастей	9
1.1 Общая характеристика деятельности фирмы по продаже и подбору автозапчастей	9
1.1.1 Типовые процессы фирмы по продаже и подбору автозапчастей ...	11
1.1.2 Процессы, подлежащие автоматизации в фирме по продаже и подбору автозапчастей.....	12
1.2 Сравнительный анализ приложений аналогов.....	12
1.2.1 Цели и задачи сравнительного анализа	12
1.2.2 Обзор приложений аналогов	13
1.3 Постановка задачи на разработку информационной системы фирмы по продаже и подбору автозапчастей	17
1.3.1 Анализ предметной области.....	17
1.3.2 Основные функции системы	18
1.4 Планирование процесса разработки программного обеспечения	19
2 Анализ требований на разработку	21
2.1 Функциональные требования	21
2.2 Разработка архитектуры программного обеспечения.....	25
2.3 Обоснование выбора средств разработки.....	28
2.3.1 Обоснование выбора инструментальных средств разработки программного обеспечения	28
2.3.2 Обоснование выбора языковых средств разработки	30
3 Реализация приложения для работы информационной системы фирмы по продаже и подбору автозапчастей.....	34
3.1 Разработка клиентской части приложения.....	34
3.1.1 Разработка пользовательского интерфейса клиентского приложения	34

3.2 Разработка программного обеспечения клиентского приложения.....	38
3.3 Разработка серверной части приложения.....	40
3.4 Разработка базы данных фирмы по продаже и подбору автодеталей.....	46
4 Контроль качества программного обеспечения информационной системы фирмы по продаже и подбору автомобильных запчастей	52
4.1 Планирование процесса контроля качества программного обеспечения	52
4.1.1 Мероприятия по контролю качества программного обеспечения...	52
4.2 Обоснование выбора модели тестирования	53
4.3 Обоснование выбора метрик программного кода	55
4.4 Тестирование программных средств	55
4.4.1 Тестирование с помощью юнит-тестов.....	55
4.4.2 Автоматическое интеграционное тестирование	56
4.5 Оценка качества программного кода на основе метрик	57
4.5.1 Метрика SLOC	57
Заключение	59
Список использованных источников	60
Приложение А (обязательное) Листинг программы	62
Приложение Б (рекомендуемое) Скриншоты работы приложения	87

Введение

На сегодняшний день автомобильная отрасль является одной из самых динамичных и развивающихся в стране, поскольку довольно значимая часть населения интересуется приобретением персонального автомобиля для повышения своего бытового комфорта и удобства. С учетом современных тенденций, приобрести новый автомобиль иностранного производства становится недоступной опцией для многих людей, что повышает спрос на поддержанные автомобили, называемые в общеупотребительном смысле «Б/У». С учетом того, что данные автомобили уже побывали в эксплуатации, возникает необходимость их поддержания для комфортного использования, что повышает спрос на различные запчасти у автодилеров. Многообразие как самих запчастей, так и их аналогов от различных производителей создает проблему поддержки их учета и классификации. Определенное время назад для этого использовались различные бумажные носители, такие как бумажные тетради и таблицы, но на сегодняшний день с учетом развития программного обеспечения появляется возможность повысить эффективность данного процесса, используя различные соответствующие инструменты. Таким инструментом выступает ПО, являющееся темой данной выпускной квалификационной работы, а именно информационная система по продаже и подбору автомобильных запчастей.

Актуальность дипломной работы заключается в том, что используя данный сервис различные автодилеры, как федерального, так и регионального уровня смогут повысить скорость и точность подбора автозапчастей, упростить ведение складского и клиентского учета, а также временные и трудовые затраты сотрудников фирмы.

Объектом исследования является деятельность фирмы по продаже и подбору автозапчастей. Предметом исследования выступают процессы автоматизации учета товаров, подбора запчастей и обслуживания клиентов.

Целью разработки является создание программного обеспечения для автоматизации работы фирмы по продаже и подбору автозапчастей.

Для достижения поставленной цели нужно решить следующие задачи:

- проанализировать предметную область;
- осуществить сравнительный анализ приложений-аналогов;
- выполнить постановку задачи на разработку ПО для автоматизации работы фирмы по продаже и подбору автозапчастей;
- смоделировать функциональные возможности приложения с помощью языка UML;
- выбрать подходящие инструменты для разработки, развертывания, тестирования и интеграции приложения;
- выполнить проектирование архитектуры приложения, пользовательского интерфейса;
- разработать ПО для автоматизации продажи и подбора автомобильных запчастей;
- выполнить тестирование программного обеспечения.

В результате выполнения работы должно быть реализовано средство для автоматизации работы фирмы по продаже и подбору автомобильных запчастей.

1 Анализ предметной области и постановки задачи на разработку информационной системы по продаже и подбору автозапчастей

1.1 Общая характеристика деятельности фирмы по продаже и подбору автозапчастей

Торговля автомобильными запчастями в настоящее время является устойчивым и постоянно растущим сегментом российской экономики. Данный вид экономической деятельности представляет собой совокупность процессов по удовлетворению потребностей владельцев транспортных средств в приобретении качественных и исправных автомобильных запчастей для обслуживания и ремонта своих автомобилей [1].

Можно заметить, что на сегодняшний день данная сфера деятельности представлена довольно большим количеством компании различного уровня, дифференцированного по уровню количества предоставляемых запчастей, их качества, а также различного сегмента ценовой доступности запчастей, от самых простых и дешевых до оригинальных запчастей от производителя высокого ценового сегмента. Для конечного потребителя это создает возможность выбора комплектующих для транспортных средств любого уровня, от потребительских до используемых в производстве или иных сферах экономической деятельности.

Так как при производстве автомобильных запчастей производитель учитывают, что они разрабатываются для применения в конечном множестве транспортных средств, это позволяет выделить ключевые признаки для их классификации, по типу (оригинальные, неоригинальные и аналоги), по производителю, по применимости к конкретным автомобилям, а также по принадлежности к узлам и системам (двигатель, ходовая часть, тормозная система, электрооборудование, кузовные элементы и др.), это дает возможность их конечному продавцу довольно быстро подобрать необходимые комплектующие для покупателя. Для ускорения данного вида деятельности, с учетом роста количества номенклатуры товаров и объёма клиентов, а как

следствие, и роста объёма проводимых операций, фирмы остро нуждаются в автоматизированных средствах управления.

Внутри фирмы по продаже автозапчастей можно выделить несколько ключевых ролей, каждая из которых взаимодействует с информацией и участвует в её обработке:

- клиенты (покупатели): физические и юридические лица, приобретающие запасные части;
- менеджеры по продажам (продавцы-консультанты): сотрудники, отвечающие за взаимодействие с клиентами, консультирование и оформление заказов;
- специалисты по подбору запчастей: работники, осуществляющие поиск и проверку применимости деталей к конкретному автомобилю по VIN-коду или каталожным данным;
- работники склада: отвечают за приёмку, хранение, учёт остатков и выдачу товаров;
- отдел закупок: формирует заказы поставщикам, контролирует поступление и наличие товаров;
- бухгалтерия/финансовый отдел: обрабатывает платежи, выручку, начисления и расчёты с поставщиками;
- руководство и аналитики: используют отчётность для принятия управленческих решений;
- администраторы ИТ-систем: поддерживают работу автоматизированной системы, контролируют доступ, обеспечивают информационную безопасность.

Таким образом, предметная область продажи и подбора автомобильных запчастей охватывает широкий спектр процессов, требующих строгого учёта, точности и оперативности. Автоматизация данных процессов не только повышает эффективность работы фирмы, но и обеспечивает корректность подбора деталей, ускоряет обслуживание клиентов и минимизирует риски человеческих ошибок. Внедрение специализированной информационной

системы становится неотъемлемым условием для закрепления современной фирмы по продаже автозапчастей в конкурентной среде.

1.1.1 Типовые процессы фирмы по продаже и подбору автозапчастей

Современная фирма по продаже и подбору автомобильных запчастей ежедневно сталкивается с повторяющимися задачами, которые требуют высокой точности и скорости выполнения. Основные процессы, которые свойственны любой фирме, занимающейся предоставлением данной экономической услуги:

- регистрация и учёт клиентов;
- ведение базы данных клиентов;
- история заказов и обращений;
- учёт контактных и юридических данных, информации об автомобилях клиента (марка, модель, VIN-код);
- подбор автомобильных запчастей;
- поиск деталей по VIN-коду, каталожному номеру или параметрам автомобиля;
- подбор оригинальных деталей и их аналогов с проверкой применимости;
- сравнение цен, сроков поставки и наличия по разным позициям;
- ведение каталога товаров;
- учёт номенклатуры запчастей, их характеристик и применимости;
- управление ценами, скидками и наценками;
- учёт остатков на складе и резервирование товаров;
- оформление и обработка заказов;
- формирование заказа клиента по подобранным позициям;
- контроль статуса заказа (оформлен, оплачен, собран, выдан);
- складской учёт и закупки;
- приёмка и оприходование поступившего товара;
- формирование заказов поставщикам при нехватке позиций;

- инвентаризация и контроль складских остатков;
- отчётность и аналитика.

Для фирмы по продаже и подбору автомобильных запчастей выделено много процессов, которые могут быть автоматизированы и ускорены благодаря разрабатываемому ПО. Рассмотрим некоторые из них.

1.1.2 Процессы, подлежащие автоматизации в фирме по продаже и подбору автозапчастей

Рассматривая описанные выше процессы, актуальные для фирм по подбору и продаже автозапчастей, можно заметить, что ряд процессов, происходящих в данных фирмах, можно значительно ускорить, разработав необходимое программное обеспечение. Речь идет о процессе подбора автозапчастей: вручную сотрудник должен перебирать огромное количество номенклатуры, которая распределена по различным источникам, используя же автоматизированную систему и ее базу данных, можно по VIN – коду и характеристикам автомобиля за несколько минут подобрать все необходимые автозапчасти и отправить заявку на склад, что сэкономит большое количество времени.

Система также позволяет легко вести учет запчастей: смотреть остаток на складе, отслеживать наиболее популярные у клиентов детали, чтобы затем учитывать это при оптовой закупке, а также страховать компанию от проблем долгого поиска уже отсутствующих на складе деталей, если по ним не были внесены сведения в номенклатуру, и тем самым не допускать финансовых ошибок, когда уже полученные денежные средства приходится возвращать клиенту [2]. Это значительно повышает эффективность ведения деятельности любой подобной фирмы.

1.2 Сравнительный анализ приложений аналогов

1.2.1 Цели и задачи сравнительного анализа

Информационной системы для подбора автозапчастей не является новым

уникальным продуктом, еще не существующем на рынке. На сегодняшний день несколько подобных продуктов уже существуют и используются в деятельности коммерческих фирм, однако они далеки от идеала, и все содержать в себе как преимущества, так и недостатки.

Целью изучения подобных систем является их сравнительный анализ, это позволяет понять, какой функционал уже существует и может быть переработан для того, чтобы повысить удобность использования в ежедневных процессах, а какой может быть доработан для того, чтобы решать задача более эффективно.

1.2.2 Обзор приложений аналогов

Аналоги, найденные в ходе использования различных информационных систем, можно разделить на две категории: универсальные учётные системы, адаптированные под торговлю запчастями, и специализированные отраслевые решения.

Первым аналогом является «АвтоДилер». Система выполняет функции CRM и позволяет вести клиентскую базу, отслеживать историю покупок, создавать заказы поставщикам и хранить подробные карточки товаров с оригинальными номерами, аналогами и фотографиями. Среди достоинств – широкий охват складских операций и наличие инструментов аналитики. Вместе с тем система ориентирована преимущественно на десктопное использование и требует значительных усилий при первоначальной настройке и наполнении номенклатуры.

При оценке решения АвтоДилер важно учитывать не только набор функций, но и удобство внедрения в ежедневную работу небольшого магазина. Для малой организации особенно значимы простота интерфейса, минимальное количество обязательных настроек и возможность быстро обучить сотрудников. В этом отношении продукт выглядит функционально сильным, но достаточно тяжелым для начального использования: перед полноценной эксплуатацией требуется подготовить справочники, настроить права доступа и заполнить номенклатуру. Интерфейс программы представлен на рисунке 1.

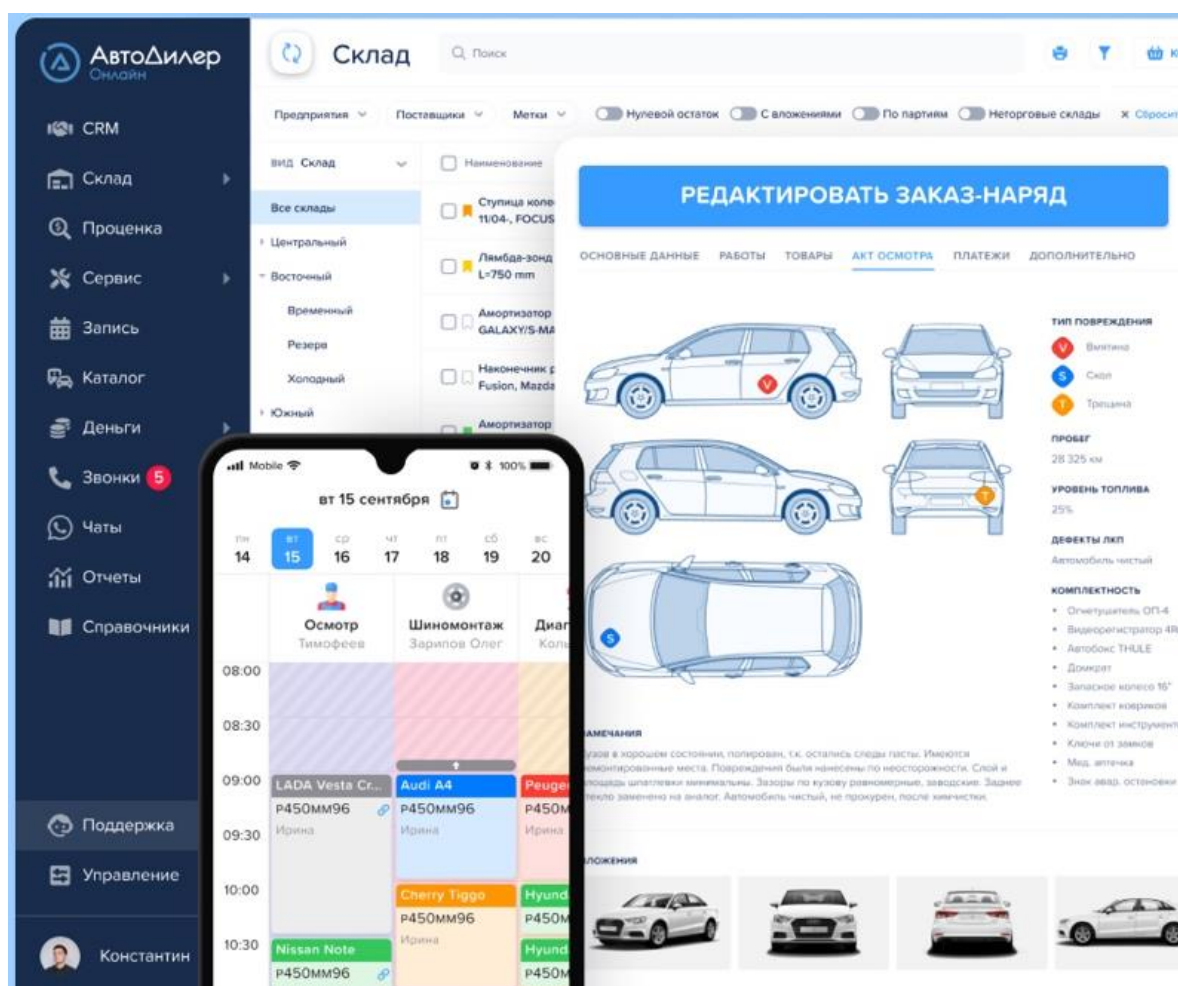


Рисунок 1 – Интерфейс программы «АвтоДилер»

Вторым аналогом выступает «Tezarius». Платформа поддерживает кроссировку складских остатков при поиске у поставщиков, гибкую настройку ценообразования, поиск по параметрам и работу с несколькими складами, а также включает собственный модуль интернет-магазина с синхронизацией остатков и цен. Однако, система требует основательного обучения, а предусмотренный функционал нередко используется лишь частично из-за сложности настройки.

С точки зрения рассматриваемого проекта Tezarius полезен как пример веб-ориентированного подхода и синхронизации данных между каталогом, складом и интернет-магазином. Однако избыточность настроек делает такое решение менее удобным для небольшого предприятия, где важнее быстрый поиск, понятная карточка товара и простой порядок оформления заявки. Поэтому при проектировании собственной системы целесообразно

использовать только необходимые функции, избегая перегруженности интерфейса и сложных сценариев работы. Такой подход позволит сохранить практическую ценность решения и одновременно сделать его доступным для сотрудников без длительного предварительного обучения. Интерфейс программы «Tezarius» представлен на рисунке 2.

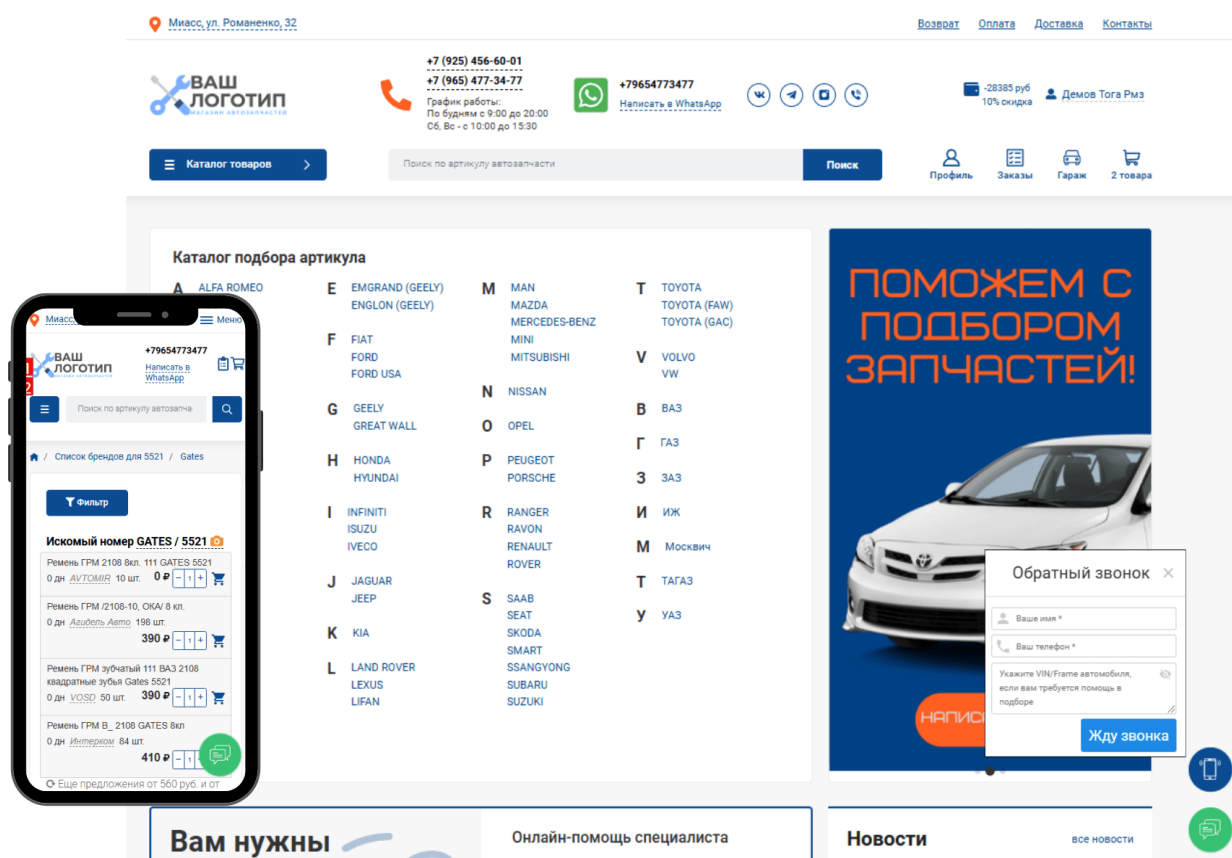


Рисунок 2 – Интерфейс программы «Tezarius»

Третьим аналогом является «1С:Розница. Магазин автозапчастей». Одна из ключевых возможностей – учёт аналогов, позволяющий не терять клиентов при отсутствии нужного товара, а также функция «Применяемость», дающая возможность привязать деталь к конкретным маркам автомобилей. Недостатком является то, что «1С» в базовой конфигурации представляет собой лишь учётный инструмент, и для полноценной работы с запчастями к нему необходимо отдельно докупать базу кросс-номеров, VIN-каталог, поисковик по поставщикам и CRM. Интерфейс программы «1С:Розница Магазин автозапчастей» представлен на рисунке 3.

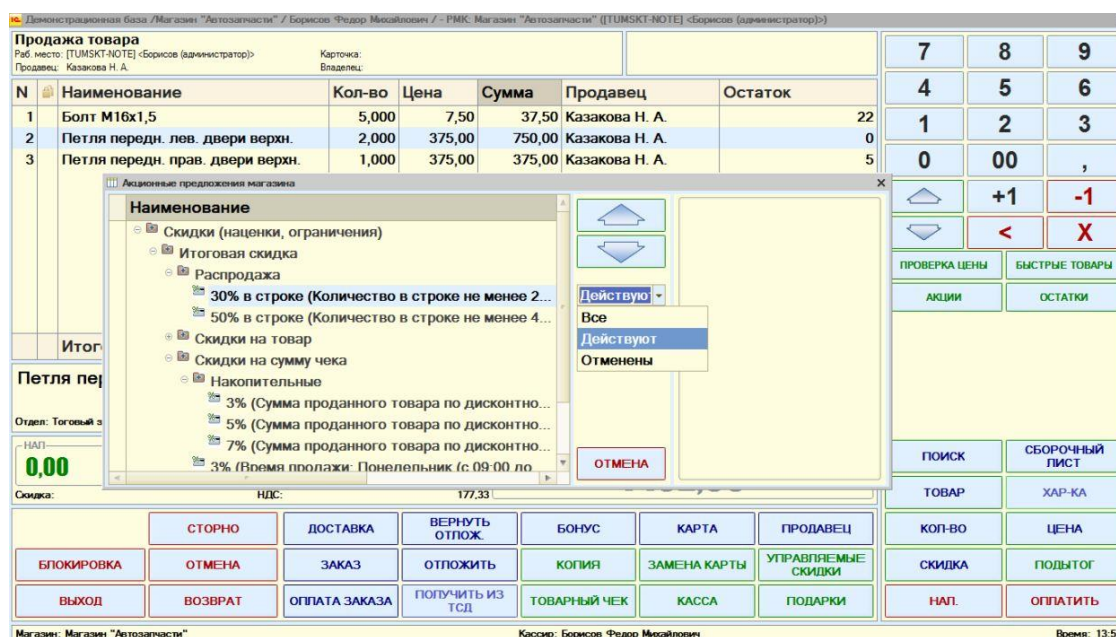


Рисунок 3 – Интерфейс программы «1С:Розница Магазин автозапчастей»

По итогам изучения аналогов была составлена сравнительная таблица, в которой отражено наличие или отсутствие ключевых возможностей в каждом из рассмотренных продуктов и в разрабатываемой системе.

Таблица 1 – Сравнительная таблица аналогов разрабатываемой системы

Критерий	АвтоДилер	Tezarius	1С: Розница	Разрабатываемая система
Каталог запчастей с артикулами	+	+	+	+
Подбор по марке /модели авто	+	+	+	+
Учёт аналогов и кросс-номеров	+	+	+	+
Складской учет	+	+	+	+
Управление заказами	+	+	+	+
Ведение клиентской базы (CRM)	+	-	-	+
Работа с поставщиками	+	+	-	+
Веб-интерфейс без установки ПО	-	+	-	+
Простота освоения без обучения	-	-	-	+
Адаптация под малый магазин	-	-	-	+

Картина, которую даёт этот анализ, довольно показательна. Все три рассмотренных продукта справляются с базовыми задачами – хранением номенклатуры, складским учётом и подбором деталей. Однако за этим функционалом скрывается общая проблема: системы создавались с расчётом на крупные или средние предприятия с выделенным IT-специалистом и временем на обучение персонала. Небольшой магазин, где работают два-три человека,

попросту не сможет использовать весь этот потенциал в полной мере.

Разрабатываемая информационная система нацелена на то, чтобы собрать в одном месте всё необходимое для повседневной работы небольшой фирмы по продаже запчастей – удобный поиск и подбор деталей, контроль склада, оформление заказов и клиентскую базу – без избыточной сложности и необходимости покупать дополнительные модули. Именно в этом состоит её принципиальное отличие от существующих решений.

1.3 Постановка задачи на разработку информационной системы фирмы по продаже и подбору автозапчастей

1.3.1 Анализ предметной области

В рамках дипломного проекта предполагается разработка автоматизированной информационной системы фирмы по продаже и подбору автомобильных запчастей с упором на подсистему подбора запчастей и оформления заказов. Основное назначение системы звучит следующим образом: «Автоматизация работы менеджеров по продажам и специалистов по подбору запчастей в части поиска, подбора и проверки применимости деталей к конкретному автомобилю, а также оформления и сопровождения заказов клиентов».

Система должна обеспечивать:

- ведение единой базы клиентов и их автомобилей;
- ведение каталога запчастей с учётом применимости и аналогов;
- подбор деталей по VIN-коду, артикулу или параметрам автомобиля;
- учёт складских остатков и резервирование товаров;
- формирование и сопровождение заказов клиентов.

Разрабатываемое программное обеспечение призвано повысить эффективность взаимодействия между подразделениями фирмы, минимизировать человеческий фактор при подборе деталей, обеспечить точность учёта и прозрачность всех процессов.

1.3.2 Основные функции системы

С точки зрения разработки систему целесообразно построить по модульному принципу: каждый модуль отвечает за свою область функциональности и взаимодействует с остальными через чётко определённые интерфейсы. Такая декомпозиция упрощает разработку, тестирование. В составе разрабатываемого ПО выделяются следующие модули.

Модуль аутентификации и авторизации отвечает за вход пользователей в систему и разграничение прав доступа между ролями «менеджер» и «клиент». Он выступает точкой входа для всех остальных модулей: любой запрос к функциям системы проходит проверку прав, и в зависимости от роли пользователю предоставляется соответствующий набор операций.

Модуль каталога запчастей хранит и обслуживает номенклатуру товаров: наименования, артикулы, характеристики, цены, а также данные о применимости деталей к маркам, моделям и годам выпуска автомобилей и сведения об аналогах. Этот модуль является справочным ядром системы – к нему обращаются модули подбора, заказов и складского учёта.

Модуль подбора деталей реализует поиск запчастей по артикулу либо по параметрам автомобиля. Он не хранит данные самостоятельно, а формирует запросы к модулю каталога (для проверки применимости и поиска аналогов) и к модулю складского учёта (для получения актуального наличия), возвращая пользователю готовый результат с ценой и остатком.

Модуль складского учёта ведёт информацию об остатках товаров, обрабатывает приёмку поступившей продукции и резервирование позиций. Он принимает запросы от модуля подбора (на чтение остатков) и от модуля заказов (на резервирование и списание товара), а при оформлении продажи автоматически корректирует количество на складе.

Модуль заказов обеспечивает формирование и сопровождение заказов. При оформлении он взаимодействует с модулем каталога (получение позиций), модулем складского учёта (резервирование и списание) и модулем клиентов (привязка заказа к покупателю), а также отслеживает статус заказа на всех

этапах его обработки.

Модуль управления клиентами хранит единую базу покупателей, их автомобили и историю обращений и заказов. К нему обращается модуль заказов при оформлении новой продажи, а клиент через личный кабинет получает доступ к своим данным и заявкам.

Таким образом, модули связаны между собой по принципу разделения ответственности: модуль каталога и модуль складского учёта выступают источниками справочных и оперативных данных, модули подбора и заказов реализуют основную бизнес-логику и обращаются к ним за информацией, модуль клиентов хранит сведения о покупателях, а модули аутентификации и отчётности обеспечивают соответственно безопасность и анализ работы системы. Подобное взаимодействие позволяет изменять или расширять отдельный модуль, не затрагивая остальные части системы.

1.4 Планирование процесса разработки программного обеспечения

Одним из ключевых этапов создания информационной системы фирмы по продаже и подбору автомобильных запчастей является планирование процесса разработки. Особенность планирования в проектной деятельности состоит в том, что план не остаётся неизменным: он постоянно уточняется и корректируется на протяжении всей работы – от постановки задачи до сдачи готового приложения.

Для планирования разработки применялась диаграмма Ганта. Она является одним из наиболее распространённых инструментов представления плана работ и позволяет наглядно отобразить состав задач, их длительность, последовательность выполнения и взаимосвязи – в данном случае применительно к разработке информационной системы.

В качестве средства построения диаграммы Ганта была выбрана программа Microsoft Project 2019 [3], поскольку она позволяет отдельно описывать ресурсы и задачи проекта, задавать их длительность, связи и исполнителей, а также автоматически рассчитывать сроки и стоимость работ.

Ресурсы проекта представлены на рисунке 4.

Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочн	Затраты на исполъз.	Начисление	Базовый календарь
Разработчик ПО	Трудовой		P		100%	150,00 Р/ч	300,00 Р/ч	0,00 Р	Пропорциональн	Стандартный
Аналитик	Трудовой		A		100%	110,00 Р/ч	200,00 Р/ч	0,00 Р	Пропорциональн	Стандартный
Тестировщик	Трудовой		T		100%	110,00 Р/ч	190,00 Р/ч	0,00 Р	Пропорциональн	Стандартный
DevOps - инженер	Трудовой		D		100%	100,00 Р/ч	130,00 Р/ч	0,00 Р	Пропорциональн	Стандартный
Архитектор ПО	Трудовой		A		100%	300,00 Р/ч	500,00 Р/ч	0,00 Р	Пропорциональн	Стандартный
Интернет	Затраты		И						Пропорциональн	

Рисунок 4 – Ресурсы проекта

Исходя из имеющейся информации о ресурсах проекта можно создать графическое представление календарного плана проекта.

На рисунке 5 представлена диаграмма Ганта.

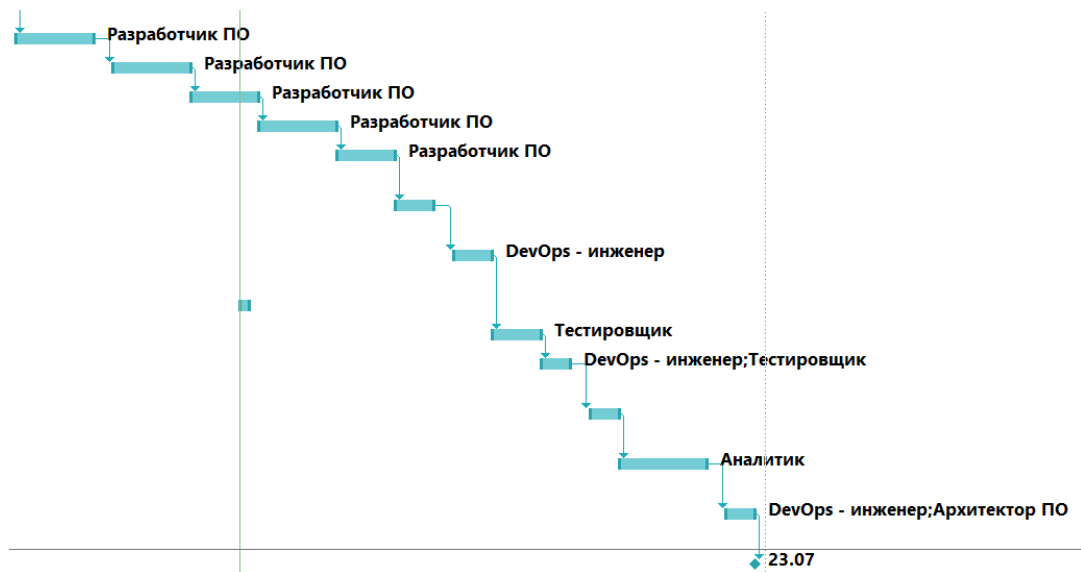


Рисунок 5 – Диаграмма Ганта

Помимо построения диаграммы Ганта, Microsoft Project располагает достаточным набором готовых отчётов и панелью мониторинга, которые позволяют отслеживать общее состояние проекта, контролировать ход выполнения работ и проверять их соответствие установленному графику.

Планирование разработки системы дало возможность рационально распределить ресурсы между этапами, определить последовательность выполнения задач и предусмотреть временные резервы на случай отклонений от графика. Применение диаграммы Ганта совместно с современными инструментами управления проектами обеспечивает прозрачность, гибкость и контроль над реализацией всех этапов создания программного продукта.

2 Анализ требований на разработку

2.1 Функциональные требования

Функциональные требования – это описание требований к разрабатываемому программному обеспечению, которые данная система должна реализовывать во время ее использования. Они определяют поведение системы в ответ на те или иные действия (например, «система должна выполнять подбор детали по VIN-коду», «система должна формировать заказ» и т. д.). В отличие от нефункциональных требований (производительность, безопасность, удобство), функциональные требования отвечают именно на вопрос «что система делает», а не «как хорошо она это делает».

Диаграмма вариантов использования (use case diagram) – это диаграмма языка UML [4], которая в наглядном виде показывает функциональные возможности системы с точки зрения её пользователей. Она отражает, кто работает с системой и какие действия каждому из них доступны, очерчивая при этом границы системы без погружения в детали внутренней реализации.

Диаграмма вариантов использования состоит из следующих основных элементов [5]:

- актёры (actors) – внешние сущности, взаимодействующие с системой (пользователи в определённых ролях, например менеджер и клиент, либо внешние системы); изображаются в виде фигурок «человечков»;
- варианты использования (use cases) – отдельные функции или сценарии, которые система выполняет для актёров (например, «оформить заказ», «подобрать деталь»); изображаются овалами;
- границы системы – прямоугольник, внутри которого размещаются варианты использования, отделяющий систему от внешних актёров;
- связи (отношения) – линии между актёрами и вариантами использования (ассоциации), а также отношения между самими вариантами использования: включение «include», расширение «extend» и обобщение (наследование).

На рисунке 6 показана диаграмма вариантов использования информационной системы для фирм по продаже и подбору автомобильных запчастей.

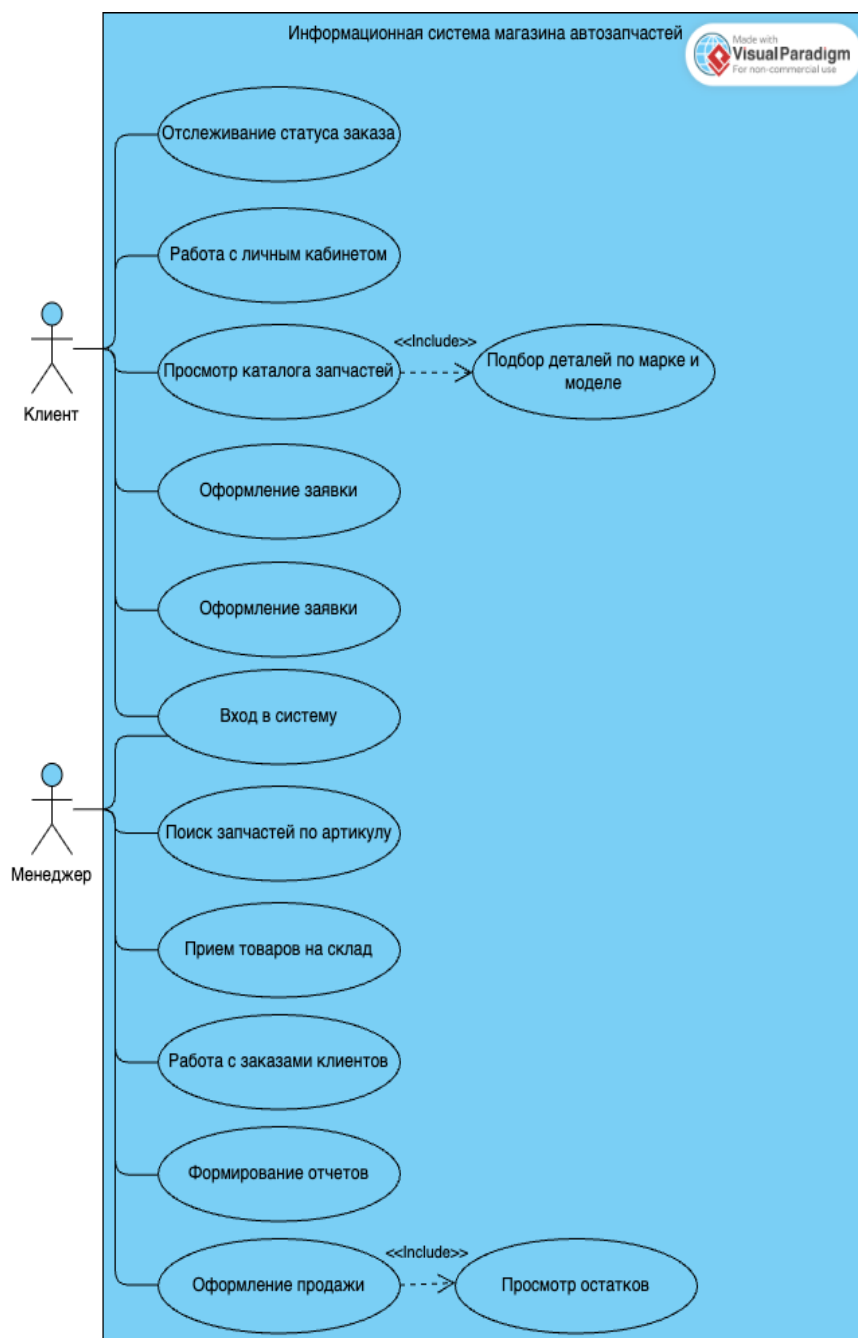


Рисунок 6 – Диаграмма вариантов использования

На диаграмме отражены две роли пользователей и закреплённые за каждой из них функции:

– менеджер: вход в систему, поиск подходящих запчастей по VIN - номеру, оформление продажи, приём товара на склад, просмотр складских

остатков, обработка заказов клиентов, формирование отчётов;

– клиент: вход в систему, просмотр каталога запчастей, подбор деталей по марке и модели автомобиля, оформление покупки, работа с личным кабинетом.

Ниже, в таблице 2, дано подробное описание вариантов использования приложения.

Таблица 2 – Описание вариантов использования

Цель	Описание
Вход в систему	Пользователь авторизуется в системе под одной из двух существующих ролей
Поиск подходящих запчастей по VIN - номеру	Менеджер может ввести в систему VIN – номер автомобиля, чтобы найти по нему подходящие для автомобиля детали
Оформление покупки	Продавец нажимает кнопку «купить» на странице детали и вводит в появившейся форме данные пользователя для оформления покупки; деталь исчезает из списка доступных для приобретения
Просмотр складских остатков	При переходе на страницу детали отображается ее остаток на складе
Обработка заказов клиентов	При вводе номера заказа можно попасть на страницу с информацией о статусе оформленного заказа
Подбор деталей по марке и модели	Пользователь может выставить фильтры по марке и модели автомобиля чтобы подобрать наиболее релевантные для себя детали
Работа с личным кабинетом	Пользователь может поменять пароль в личном кабинете
Формирование отчетов	Менеджер может нажать кнопку сформировать отчет, чтобы посмотреть количество по категориям приобретенных деталей за последний месяц

Рассмотрим более подробно несколько вариантов использования приложения. В таблице 3 описан сценарий «Поиск запчасти по VIN - номеру».

Таблица 3 – Сценарий и описание варианта использования «Поиск запчасти по VIN - номеру»

Название пункта	Описание
Наименование	Поиск подходящих запчастей по VIN - номеру
Краткое описание	Сценарий позволяет менеджеру по VIN – номеру автомобиля найти подходящие для него детали
Основные действующие лица	Менеджер
Второстепенные действующие лица	Нет
Предусловие	Менеджер нажал кнопку «Поиск по VIN – номеру»
Основной сценарий	<ol style="list-style-type: none"> 1. Менеджер переходит на страницу поиска детали по VIN – номеру автомобиля 2. Вводит в поисковую строку VIN -номер автомобиля 3. Нажимает кнопку найти «Найти» 4. Просматривает подходящие для автомобиля детали или ошибку при их отсутствии
Постусловие	Появляется возможность перейти на страницу каждой из доступных деталей

В таблице 4 показан сценарий, а также полное описание варианта использования «Оформить покупку».

Таблица 4 – Сценарий и полное описание варианта использования «Оформить покупку»

Название пункта	Описание
Наименование	Оформить покупку
Краткое описание	Сценарий описывает оформить покупку выбранной детали
Основные действующие лица	Менеджер, пользователь
Второстепенные действующие лица	Нет
Предусловие	Пользователь системы находится на странице выбранной детали
Основной сценарий	<ol style="list-style-type: none"> 1. Пользователь системы нажимает на странице детали кнопку «Купить» 2. В появившейся форме нужно ввести имя, фамилию и электронную почту, выбрать вариант оплаты (наличные) 3. Нажать кнопку завершить покупку
Постусловие	Появляется страница с надписью «покупка оформлена»

Таким образом, были смоделированы функциональные возможности приложения.

2.2 Разработка архитектуры программного обеспечения

Разработка архитектуры ПО является ключевым этапом разработки, предшествующему непосредственно самому написанию кода, и важность данного этапа нельзя недооценивать, поскольку неверный выбор архитектуры может привести в будущем к необходимости полной замены всех составных компонентов ПО, а именно, переработке клиентского интерфейса, серверной части, выбор других драйверов при необходимости смены базы данных и так далее, что выльется в значительные издержки при его разработке. Также, неверно выбранная архитектура не позволяет в дальнейшем модифицировать ПО, добавлять в него новый функционал, снижает его гибкость, устойчивость и возможность добавления новых модулей. Определить архитектуру можно как логическую организацию всех компонентов системы, связь между ними, а также те способы, с помощью которых они обмениваются данными.

Перед системой стоит задача автоматизировать работу фирмы, торгующей автозапчастями: подбирать детали под конкретный автомобиль, хранить каталог товаров, отслеживать складские остатки, оформлять заказы и готовить отчёты. Работать с приложением будут две группы пользователей – сотрудники фирмы и покупатели, - причём все они должны опираться на единые, актуальные данные. Это естественным образом приводит к клиент-серверному решению: данные и логика их обработки сосредоточены на сервере, а пользователи обращаются к ним через клиентскую часть. Внутренне приложение разбито на три самостоятельных слоя – отображения, обработки и хранения, – что отвечает широко применяемой трёхуровневой модели [6].

Слой отображения – это та часть, с которой непосредственно работает пользователь. Он выполняется в браузере и написан на JavaScript с использованием библиотеки React. Его задача – отобразить интерфейс, обработать действия пользователя (поиск детали, заполнение формы, отображение отчета) и передать их на сервер, а затем показать полученный ответ. Сам по себе этот слой решений не принимает и данные не хранит.

Слой обработки – ядро системы, где реализованы все правила её работы.

Он построен на языке Java с применением фреймворка Spring Boot и предоставляет набор методов REST API [7], через которые к нему обращается клиент. Получив запрос, серверная часть проверяет входные данные, выполняет требуемую операцию (подбирает совместимые детали, рассчитывает стоимость, создаёт заказ) и обращается к хранилищу данных за нужной информацией или для её изменения.

Слой хранения отвечает за сохранность данных и реализован на СУБД PostgreSQL. В базе данных размещаются все объекты предметной области – учётные записи пользователей, номенклатура запчастей с данными о применимости, складские остатки. Прямого доступа к базе у клиентской части нет: любое обращение к данным проходит через серверный слой.

Связь между слоями выстроена на основе REST API. Клиент и сервер обмениваются сообщениями по протоколу HTTP, передавая данные в формате JSON [8]: клиент формирует и отправляет запрос, сервер обрабатывает его и возвращает результат. Благодаря тому, что слои общаются только через заранее оговорённые интерфейсы и не зависят от внутреннего устройства друг друга, каждый из них можно разрабатывать, проверять и дорабатывать отдельно, не затрагивая остальные.

Чтобы наглядно показать, как компоненты взаимодействуют при выполнении конкретной операции, применяется диаграмма последовательности. Она изображает порядок обмена сообщениями между участниками в том порядке, в котором те происходят, и прослеживает путь от действия пользователя до итогового результата. Например, пользователь выполняет поиск или подбор запчасти, клиентское приложение передаёт запрос на сервер, сервер обрабатывает его и обращается к базе данных за необходимыми сведениями. После получения данных сервер формирует ответ, а клиентская часть отображает найденные позиции пользователю. Диаграмма последовательности для разрабатываемого приложения представлена на рисунке 7.



Рисунок 7 – Диаграмма последовательности взаимодействия компонентов приложения

Проследим этот путь на примере оформления заказа. Пользователь, находясь на странице детали, нажимает кнопку «Оплатить». Клиентская часть на React собирает введенные сведения в JSON-объект и отправляет его HTTP-запросом к соответствующему методу API. Сервер на Spring Boot принимает запрос и убеждается в корректности данных. В процессе он обращается к PostgreSQL, где создается запись о заказе, уменьшаются складские остатки и при необходимости обновляются данные клиента. Завершив обработку, сервер собирает ответ с номером и статусом заказа и отправляет его обратно клиенту, а тот преобразует полученные данные и показывает результат пользователю.

В итоге приложение представляет собой трёхуровневую клиент-серверную систему, в которой интерфейс, обработка данных и их хранение разделены и связаны между собой через строго определённые интерфейсы. Такое разделение упрощает разработку и сопровождение и позволяет развивать систему по частям.

2.3 Обоснование выбора средств разработки

2.3.1 Обоснование выбора инструментальных средств разработки программного обеспечения

На сегодняшний день инструменты для разработки ПО представлены обширным списком различных текстовых редакторов, интегрированных средств разработки, приложений для работы данных, приложений для визуализации архитектуры и т. д. С учетом стека для разработки, выбранного выше, было решено остановиться на нескольких конкретных инструментах, которые дают максимальную скорость разработки, удобство сопровождения кода, возможность быстро отлаживать программу и исправлять найденные при тестировании баги.

Клиентскую часть на React удобнее всего разрабатывать в Visual Studio Code. Это лёгкий редактор, который не нагружает систему, но при этом легко превращается в полноценную среду для веб-разработки за счёт расширений. Для работы с JavaScript и React он даёт всё необходимое: умное автодополнение, подсветку ошибок прямо во время набора кода, форматирование, встроенный терминал и работу с Git без переключения на другие программы.

Серверная часть пишется на Java со Spring Boot, и здесь логичнее использовать IntelliJ IDEA – среду, изначально заточенную именно под Java. В отличие от обычного редактора, она глубоко понимает структуру проекта: подсказывает по ходу написания кода, помогает с рефакторингом, берёт на себя сборку через Maven или Gradle и предоставляет полноценный отладчик. Для серверной логики средней и высокой сложности это заметно сокращает время разработки и снижает количество ошибок.

Отдельный плюс IntelliJ IDEA в том, что работать с базой данных можно прямо из неё – через встроенный инструмент Database. Он подключается к PostgreSQL, показывает структуру таблиц, позволяет писать и выполнять SQL-запросы и редактировать записи. Благодаря этому не нужно держать отдельную

программу для администрирования БД: разработка серверного кода и работа с данными ведутся в одном окне.

На рисунках 8 и 9 представлен интерфейс VSCode и IntelliJ idea.

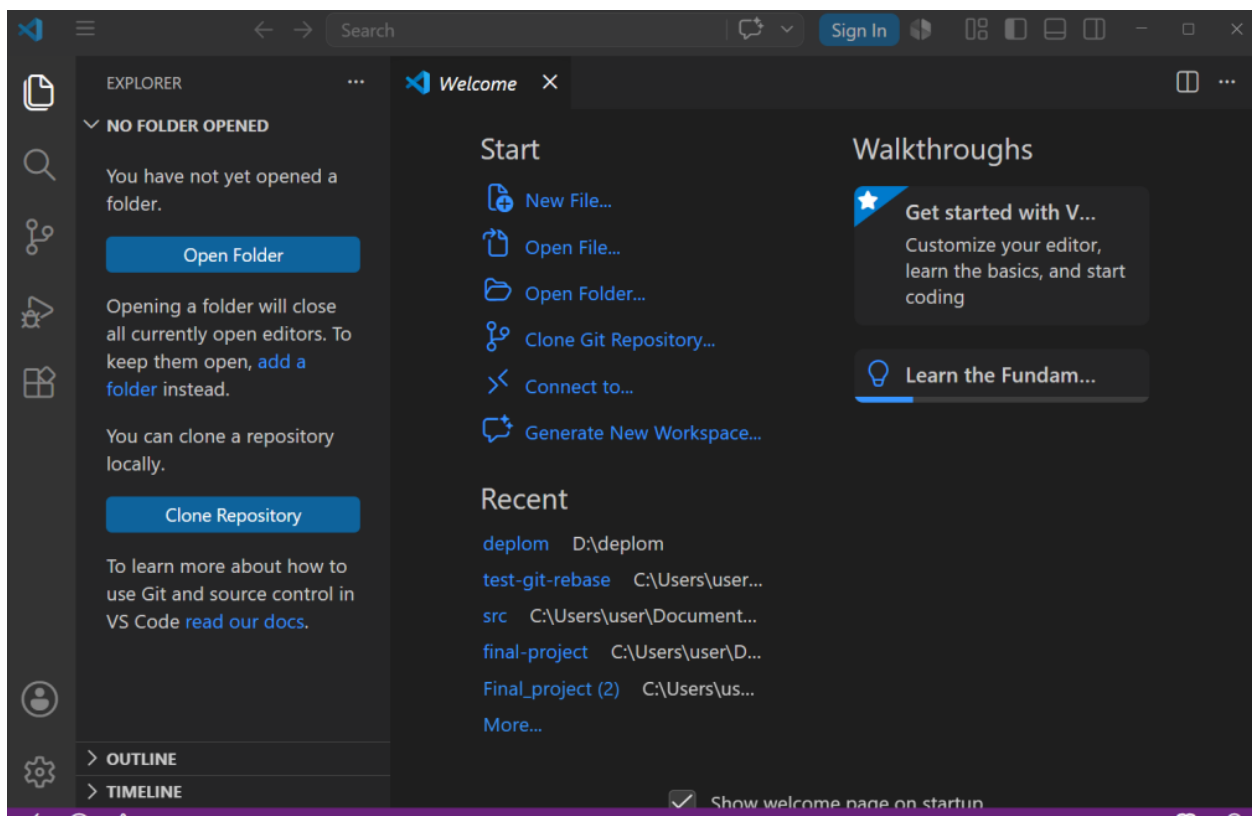


Рисунок 8 – Интерфейс VSCode

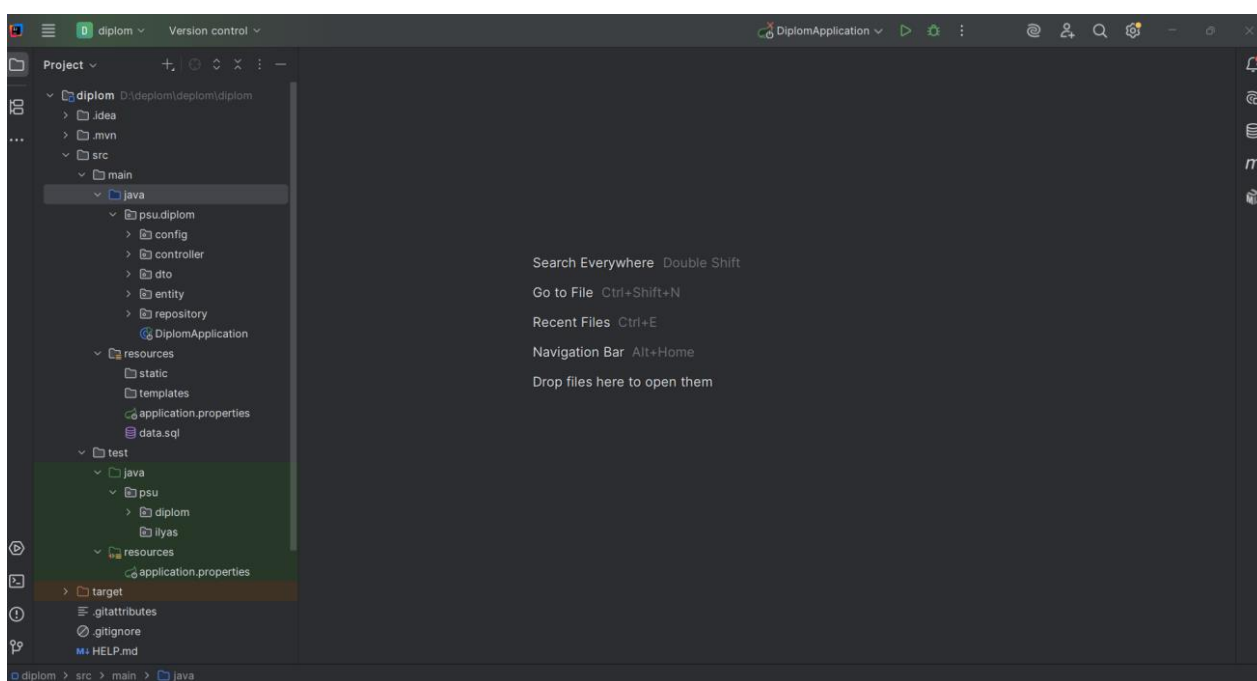


Рисунок 9 – Интерфейс IntelliJ Idea

На рисунке 10 представлен интерфейс DB browser.

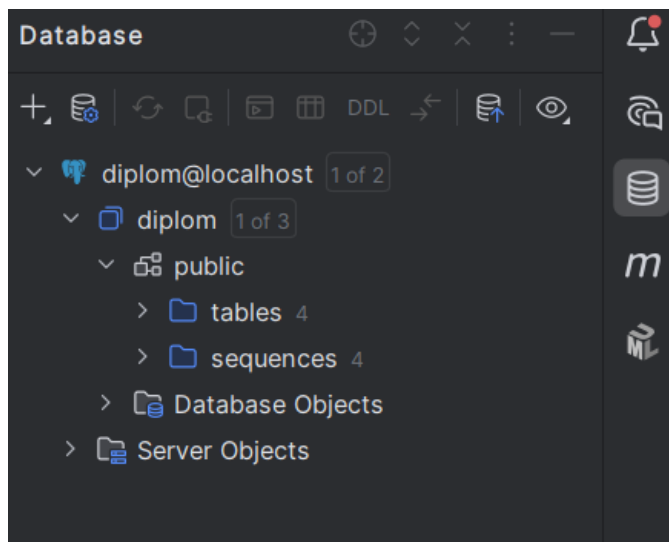


Рисунок 10 – Интерфейс DB Browser

Данные инструменты позволят решить задачу написания ПО наиболее эффективно и для решаемых задач подходят больше, чем их аналоги, такие как VIM, Eclipse, Atom, Visual Studio.

2.3.2 Обоснование выбора языковых средств разработки

Разработанная информационная система фирмы по продаже и подбору автомобильных запчастей состоит из двух частей – серверной и клиентской, каждая из которых написана на своих технологиях.

Серверная часть реализована на языке Java с применением фреймворка Spring Boot.

Java – это язык программирования общего назначения со строгой типизацией и автоматическим управлением памятью (сборкой мусора) [9]. Его главная особенность – независимость от платформы: программа на Java компилируется в промежуточный байт-код, который выполняется виртуальной машиной (JVM) на любой операционной системе. За счёт надёжности, развитой экосистемы и большого сообщества Java остаётся одним из основных языков для разработки серверной части корпоративных и информационных систем.

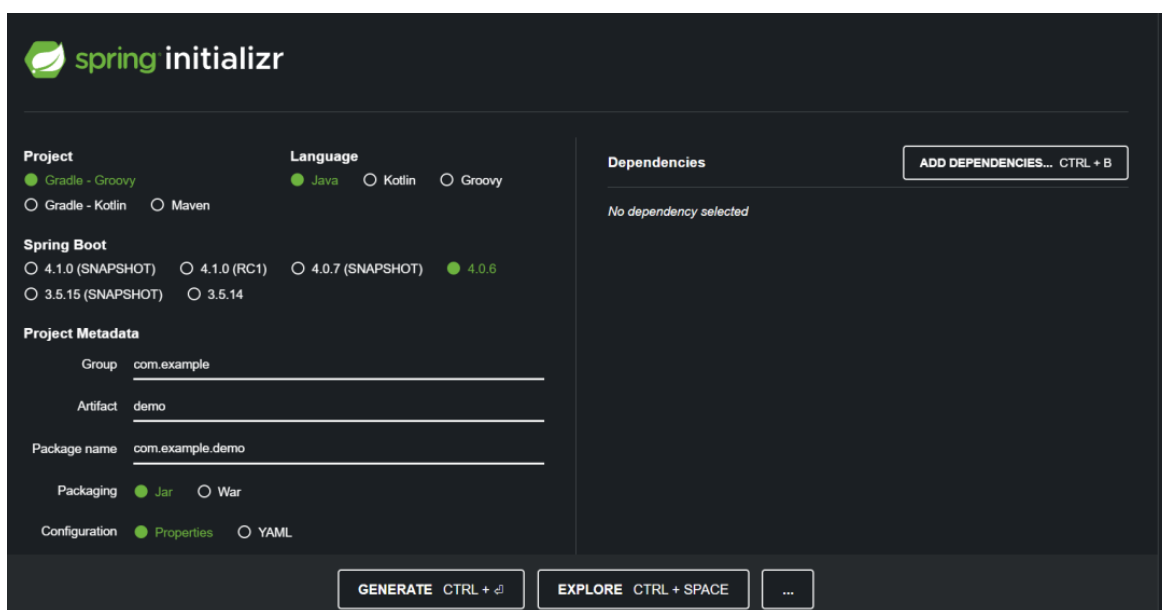
Spring Boot – это фреймворк, надстроенный над платформой Spring [10], который значительно упрощает создание серверных приложений на Java. Он

берёт на себя большую часть настройки проекта, предоставляет встроенный веб-сервер и набор готовых модулей для работы с базой данных, обработки HTTP-запросов, валидации данных и обеспечения безопасности. Благодаря этому разработчик может сосредоточиться на бизнес-логике, а не на рутинной конфигурации, что ускоряет создание REST API – основного способа взаимодействия сервера с клиентом.

Одним из удобных преимуществ Spring Boot является возможность сконфигурировать и сгенерировать заготовку проекта прямо в браузере – с помощью официального веб-сервиса Spring Initializr (start.spring.io) [11].

На этом сайте разработчику достаточно выбрать основные параметры будущего приложения: систему сборки (Maven или Gradle), язык (Java), версию Spring Boot и Java, а также указать сведения о проекте (название, группу, описание). Главная же возможность – это выбор зависимостей: из готового списка отмечаются нужные модули (например, Spring Web для создания REST API, Spring Data JPA для работы с базой данных, драйвер PostgreSQL, Spring Security и др.). После этого сервис формирует и предоставляет для скачивания готовый архив с настроенной структурой проекта и всеми подключёнными библиотеками.

На рисунке 11 представлен сервис Spring Initializr.



The screenshot shows the Spring Initializr web interface. The header features the Spring logo and the text "spring initializr". The main content area is divided into several sections:

- Project:** Includes radio buttons for "Gradle - Groovy" (selected), "Gradle - Kotlin", and "Maven".
- Language:** Includes radio buttons for "Java" (selected), "Kotlin", and "Groovy".
- Spring Boot:** Includes radio buttons for "4.1.0 (SNAPSHOT)", "4.1.0 (RC1)", "4.0.7 (SNAPSHOT)", "4.0.6" (selected), "3.5.15 (SNAPSHOT)", and "3.5.14".
- Project Metadata:** Includes input fields for "Group" (com.example), "Artifact" (demo), and "Package name" (com.example.demo). It also has radio buttons for "Packaging" (Jar selected, War) and "Configuration" (Properties selected, YAML).
- Dependencies:** A section with a button "ADD DEPENDENCIES... CTRL + B" and the text "No dependency selected".

At the bottom, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and a menu button "...".

Рисунок 11 – Spring Initializr

Такой подход избавляет разработчика от ручной настройки конфигурационных файлов и подбора совместимых версий библиотек: проект сразу готов к работе, его остаётся лишь открыть в среде разработки (в данном случае IntelliJ IDEA) и приступить к написанию кода. Это заметно ускоряет начало разработки и снижает вероятность ошибок при настройке окружения.

Клиентская часть написана на языке JavaScript с использованием библиотеки React.

JavaScript – основной язык веб-разработки, выполняющийся непосредственно в браузере пользователя. Он отвечает за интерактивность страниц: обработку действий пользователя, изменение содержимого без перезагрузки и обмен данными с сервером. Сегодня это один из самых распространённых языков программирования, вокруг которого сложилась обширная экосистема библиотек и инструментов.

React – это библиотека с открытым исходным кодом для построения пользовательских веб-интерфейсов, разработанная компанией Meta (Facebook) [12]. В основе её работы лежит компонентный подход: интерфейс собирается из отдельных переиспользуемых компонентов, каждый из которых отвечает за свою часть страницы и отображается в браузере как блок HTML. Такой подход упрощает разработку и поддержку сложных интерфейсов, а большое сообщество и множество готовых решений делают React одним из самых популярных инструментов наряду с Angular и Vue. На рисунке 12 представлен официальный сайт React.

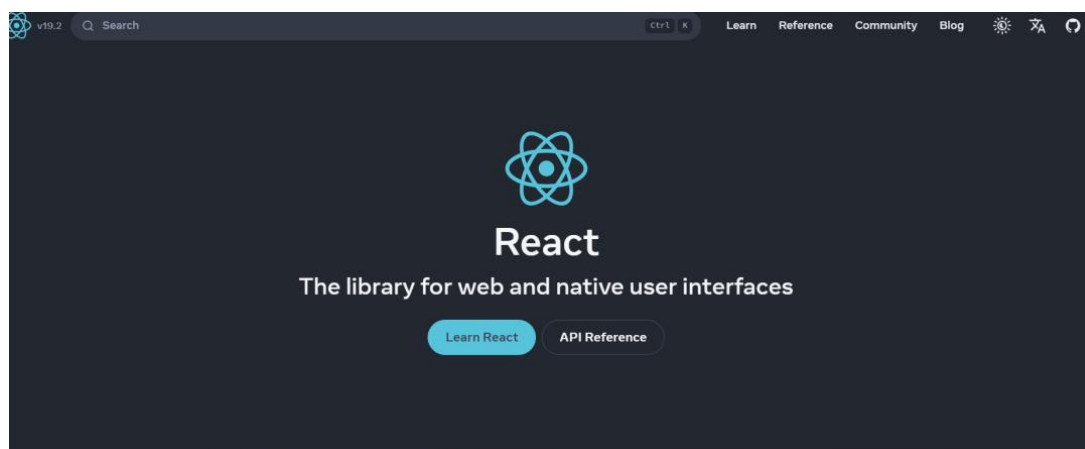


Рисунок 12 – Официальный сайт React JS

В первой версии информационная система фирмы по продаже и подбору автомобильных запчастей реализуется только в виде веб-приложения, работающего в браузере. Такой вариант не требует установки на устройства пользователей и доступен с любого компьютера или телефона при наличии доступа к сети, что позволяет быстрее запустить систему в работу и охватить основную аудиторию.

В дальнейшем планируется расширить систему и сделать её доступной на всех типах устройств: в виде мобильных приложений для Android и iOS, а также настольных приложений для Windows, Linux и macOS. Важным преимуществом выбранного стека является то, что значительную часть наработок при этом можно будет переиспользовать.

Для разработки мобильных приложений предполагается применить React Native – технологию, родственную React, которая позволяет создавать приложения под Android и iOS, опираясь на уже знакомые подходы и компоненты. Настольные версии под Windows, Linux и macOS планируется реализовать с помощью Electron – фреймворка, который даёт возможность собирать кроссплатформенные desktop-приложения на основе тех же веб-технологий. При этом серверная часть на Spring Boot и база данных PostgreSQL останутся общими для всех версий: новые клиенты будут обращаться к тому же REST API, что и веб-приложение, поэтому дорабатывать потребуется в основном клиентскую часть.

3 Реализация приложения для работы информационной системы фирмы по продаже и подбору запчастей

3.1 Разработка клиентской части приложения

3.1.1 Разработка пользовательского интерфейса клиентского приложения

Перед тем как приступить к написанию кода для клиентской части приложения, было выполнено его прототипирование. Рассмотрим прототипы основных страниц приложения.

На рисунке 13 изображен пользовательский интерфейс главной страницы приложения.

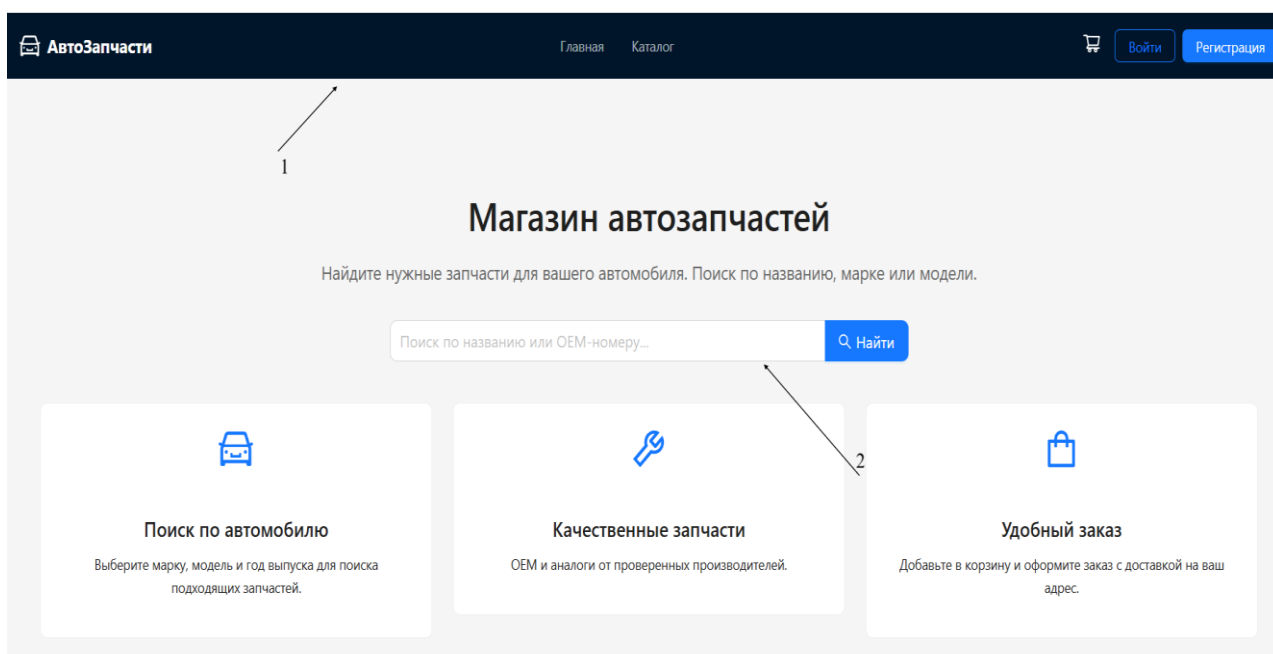


Рисунок 13 – Главная страница приложения

На рисунке расположены элементы:

- 1 – шапка сайта;
- 2 – окно поиска.

На рисунке 14 показана панель менеджера магазина.

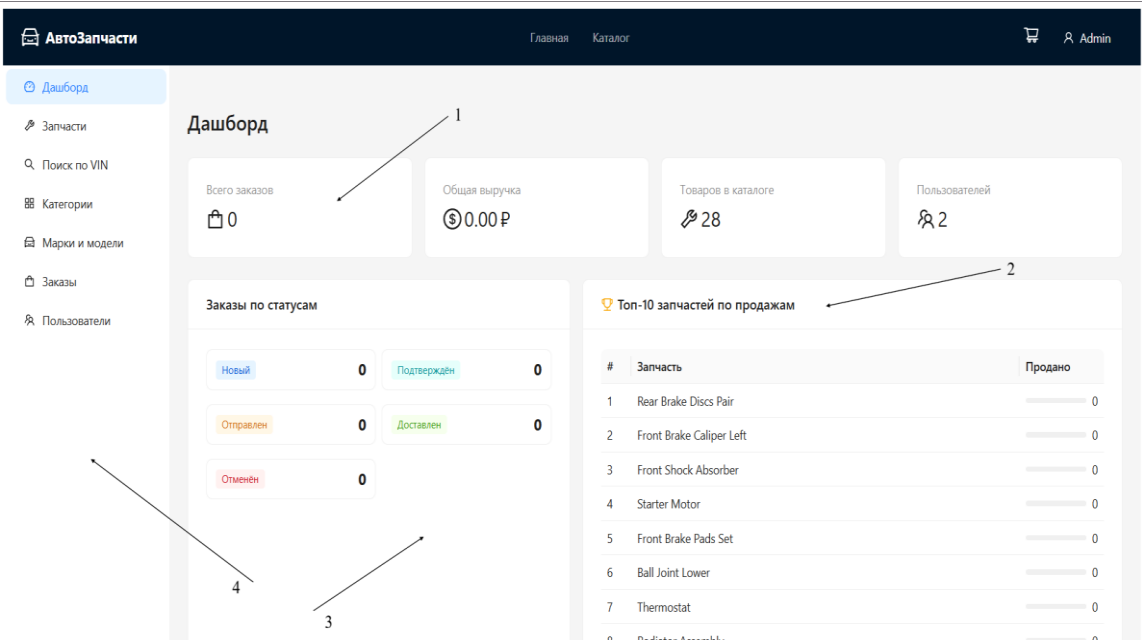


Рисунок 14 – Панель менеджера магазина

На рисунке показаны элементы:

- 1 – общая статистика заказов;
- 2 – статистика по самым продаваемым запчастям;
- 3 – статистика по статусам заказов;
- 4 – элементы навигации.

На рисунке 15 показана страница управления запчастями, доступная менеджеру.

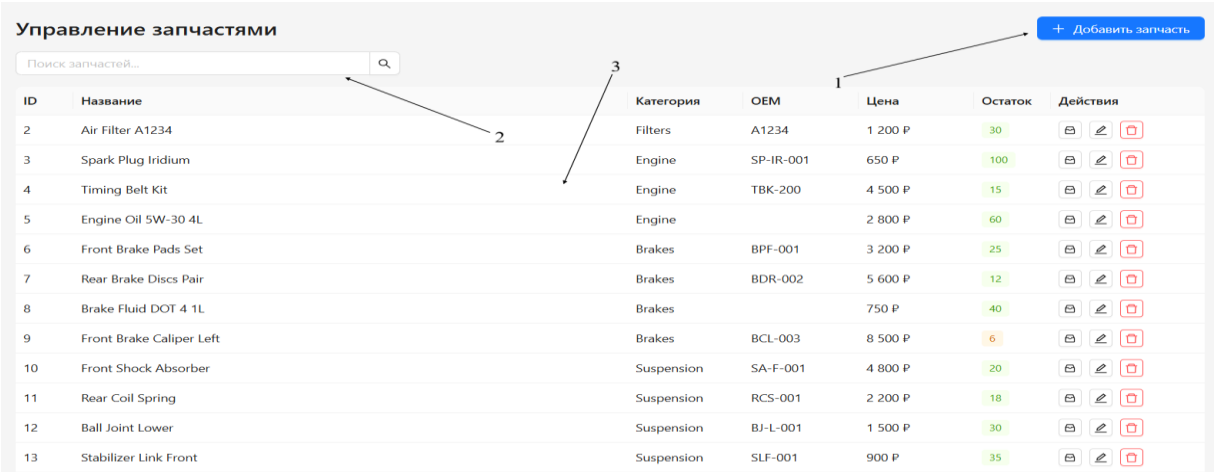


Рисунок 15 – Страница управления запчастями

На рисунке показаны элементы:

- 1– кнопка добавления новой детали в систему при поступлении на склад;
- 2 – поиск по запчастям;
- 3 – все детали, представленные в магазине (в т. ч. на складе).

На рисунке 16 показана страница поиска детали по VIN – номеру.

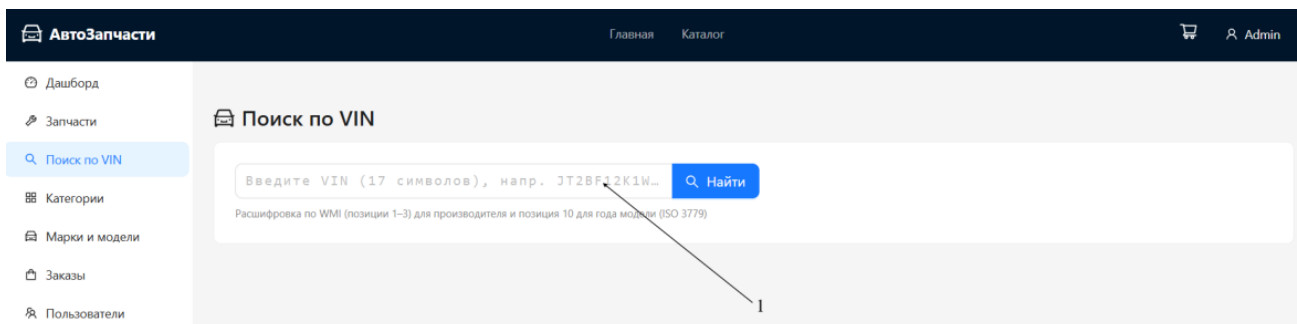


Рисунок 16 – Поиск детали по VIN - номеру

На рисунке 16 показаны следующие элементы:

- 1– поле для поиска детали по VIN – номеру.

На рисунке 17 показана страница управления категориями деталей.

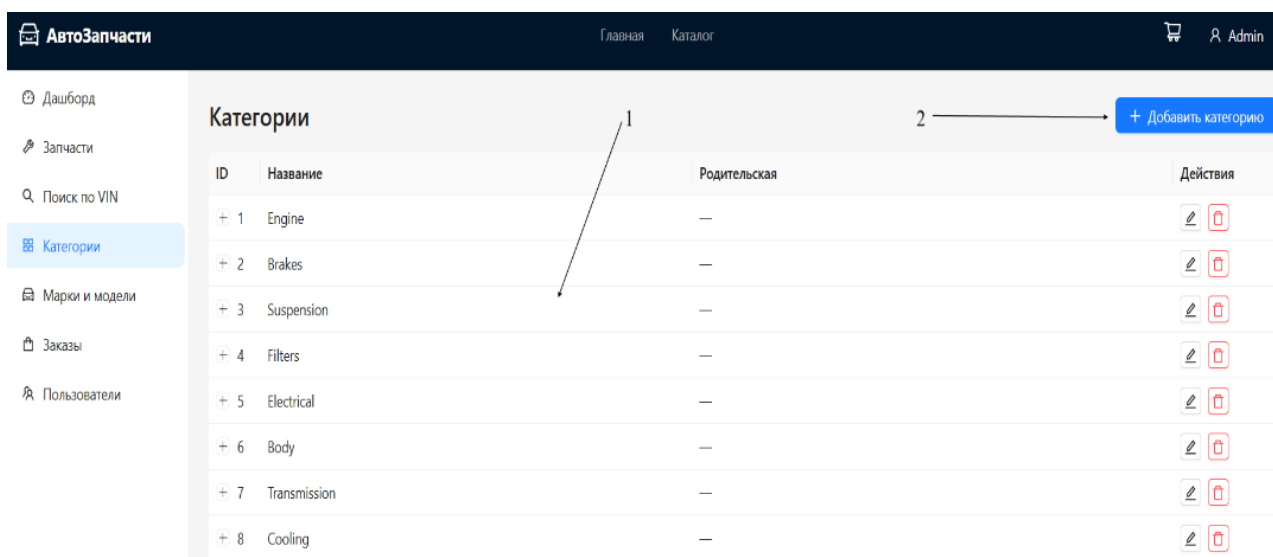


Рисунок 17 – Страница управления категориями деталей

На рисунке показаны следующие элементы:

- 1 – существующие категории деталей;
- 2 – кнопка добавления новой категории.

На рисунке 18 показана страница управления марками и моделями автомобилей, для которых существуют детали.



Рисунок 18 – Страница управления марками и моделями авто

На рисунке показаны следующие элементы:

- 1 – Панель навигации;
- 2 – Кнопка добавления новой марки;
- 3 – Каталог существующих марок.

На рисунке 19 показан общий каталог деталей.

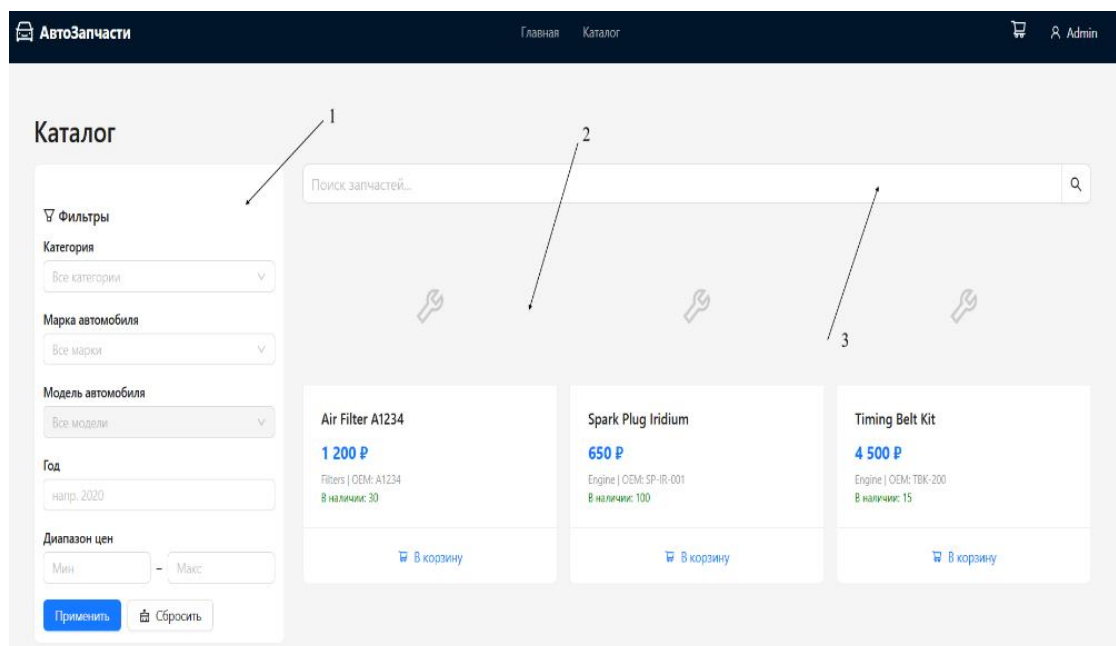


Рисунок 19 – Общий каталог деталей

На рисунке показаны следующие элементы:

- 1– Фильтры по деталям;

2 – Каталог деталей;

3 – Поиск по деталям.

Таким образом был разработан пользовательский интерфейс для информационной системы фирмы по продаже и подбору автозапчастей.

3.2 Разработка программного обеспечения клиентского приложения

Программное обеспечение клиентского приложения разрабатывалось на языке программирования JavaScript с использованием библиотеки React JS.

В React компоненты можно создавать двумя способами: в функциональном стиле или в виде классов (объектно-ориентированный подход). Классовый стиль чаще применяют при разработке на TypeScript [13].

В данной работе клиентское приложение писалось на JavaScript в функциональном стиле – с использованием функциональных компонентов.

На рисунке 20 показана диаграмма компонентов клиентского приложения.

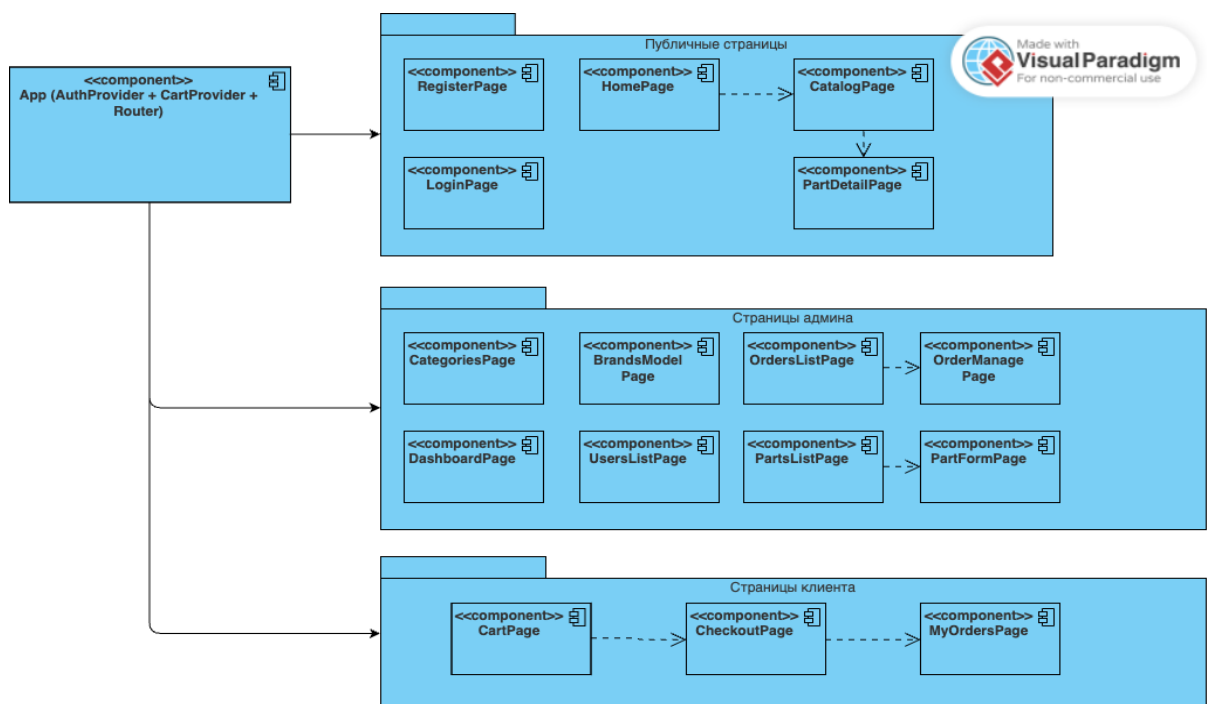


Рисунок 20 – Диаграмма компонентов клиентского приложения

Приведем описание основных компонентов:

- компоненты LoginPage/RegisterPage – формы для входа и регистрации;
- компоненты HomePage/CatalogPage/PartDetailPage – формы для работы с каталогом;
- компоненты CartPage/CheckoutPage/MyOrdersPage – формы оформления заказов;
- компоненты PartsListPage/CategoriesPage/BrandsModelPage/OrdersListPage/UsersListPage – формы менеджера для управления каталогом деталей и т. д.

Готовые элементы интерфейса (формы, таблицы, кнопки) реализованы на основе библиотеки компонентов Ant Design [14].

Код главного компонента App.tsx представлен в листинге 1.

Листинг 1 – Код главного компонента App.js

```
export default function App() {
  return (
    <ConfigProvider
      theme={{
        token: {
          colorPrimary: '#1677ff',
        },
      }}
    >
      <Routes>
        { /* Public routes */ }
        <Route element={ <AppLayout /> } >
          <Route path="/" element={ <HomePage /> } />
          <Route path="/catalog" element={ <CatalogPage /> } />
          <Route path="/catalog/:id" element={ <PartDetailPage /> } />
        </Route>
        <Route path="/login" element={ <LoginPage /> } />
        <Route path="/register" element={ <RegisterPage /> } />
        { /* Client-protected routes */ }
        <Route element={ <ProtectedRoute /> } >
          <Route path="/cart" element={ <CartPage /> } />
          <Route path="/checkout" element={ <CheckoutPage /> } />
          <Route path="/orders" element={ <MyOrdersPage /> } />
          <Route path="/orders/:id" element={ <OrderDetailPage /> } />
        </Route>
        <Route path="/profile" element={ <ProfilePage /> } />
      </Route>
    </Route>
  )
}
```

```

        { /* Admin routes */
        <Route element={<AdminRoute />}>
            <Route element={<AdminLayout />}>
                <Route path="/admin" element={<DashboardPage />} />
                <Route path="/admin/parts" element={<PartsListPage />}
            />
                <Route path="/admin/parts/new" element={<PartFormPage
            />} />
                <Route path="/admin/parts/:id/edit"
            element={<PartFormPage />} />
                <Route path="/admin/categories"
            element={<CategoriesPage />} />
                <Route path="/admin/brands-models"
            element={<BrandsModelsPage />} />
                <Route path="/admin/orders" element={<OrdersListPage
            />} />
                <Route path="/admin/orders/:id"
            element={<OrderManagePage />} />
                <Route path="/admin/users" element={<UsersListPage />}
            />
                <Route path="/admin/vin-search" element={<VinSearchPage
            />} />
            </Route>
        </Route>
    </Routes>
</ConfigProvider>
);
}

```

Полный листинг исходного кода клиентской и серверной части приложения приведён в Приложении А.

Таким образом была разработана клиентская часть информационной системы фирмы по продаже и подбору автозапчастей.

3.3 Разработка серверной части приложения

Серверная часть была реализована на Java с использованием фреймворка Spring Boot. В таблице 5 представлена работа API модуля аутентификации. Аутентификация пользователей и разграничение прав доступа реализованы средствами Spring Security на основе JWT-токенов [15].

Таблица 5 – API модуля аутентификации

Название функции	Строка адреса	Метод REST API	Описание
register	/api/auth/register	POST	Регистрация нового пользователя

Продолжение таблицы 5

Название функции	Строка адреса	Метод REST API	Описание
login	/api/auth/login	POST	Вход по email и паролю
getCurrentUser	/api/auth/me	GET	Получение данных текущего авторизованного пользователя
updateProfile	/api/auth/me	PUT	Обновление профиля и/или смена пароля

В таблице 6 представлена работа API модуля запчастей.

Таблица 6 – API модуля запчастей

Название функции	Строка адреса	Метод REST API	Описание
search	/api/parts	GET	Поиск запчастей с фильтрами и пагинацией
getById	/api/parts/{id}	GET	Получение детальной информации о запчасти по ID
searchByVin	/api/parts/by-vin?vin=	GET	Поиск совместимых запчастей по VIN – коду автомобиля
create	/api/parts	POST	Создание новой запчасти
update	/api/parts/{id}	PUT	Обновление данных запчасти
delete	/api/parts/{id}	DELETE	Удаление запчасти
addCompatibility	/api/parts/{id}/compatibility/{modelId}	POST	Добавление совместимости запчасти с моделью авто
removeCompatibility	/api/parts/{id}/compatibility/{modelId}	DELETE	Удаление совместимости запчасти с моделью авто
receiveStock	/api/admin/parts/{id}/stock	POST	Приемка товара на склад (увеличение остатка)

В таблице 7 представлена работа API модуля заказов клиента.

Таблица 7 – API модуля заказов клиентов

Название функции	Строка адреса	Метод REST API	Описание
createOrder	/api/orders	POST	Создание нового заказа текущего пользователя
getMyOrders	/api/orders	GET	Получение списка заказов текущего пользователя
getOrderById	/api/orders/{id}	GET	Получение заказа по ID
updateStatus	/api/admin/orders/{id}/status	PUT	Изменение статуса заказа

В таблице 8 представлена работа API модуля марки автомобилей.

Таблица 8 – API модуля марки автомобилей

Название функции	Строка адреса	Метод REST API	Описание
getAll	/api/brands	GET	Получение списка всех марок
getById	/api/brands/{id}	GET	Получение марки по ID
getModelsByBrand	/api/brands/{brandId}/models	GET	Получение моделей по ID марки
create	/api/brands	POST	Создание новой марки
update	/api/brands/{id}	PUT	Обновление марки
delete	/api/brands/{id}	DELETE	Удаление марки

В таблице 9 представлена работа API модели автомобилей.

Таблица 9 – API модуля модели автомобилей

Название функции	Строка адреса	Метод REST API	Описание
getAll	/api/models	GET	Получение всех моделей (с фильтром по brandId)
getById	/api/models/{id}	GET	Получение модели по ID
create	/api/models	POST	Создание новой модели
update	/api/models/{id}	PUT	Обновление модели
delete	/api/models/{id}	DELETE	Удаление модели

В таблице 10 представлена работа API категорий автомобилей.

Таблица 10 – API модуля категорий

Название функции	Строка адреса	Метод REST API	Описание
getAll	/api/categories	GET	Получение категорий в виде дерева или плоского списка (?flat=true)
getById	/api/categories/{id}	GET	Получение категорий по ID
create	/api/categories	POST	Создание новой категории
update	/api/categories/{id}	PUT	Обновление категории
delete	/api/categories/{id}	DELETE	Удаление категории

В таблице 11 представлена работа API отчетов.

Таблица 11 – API модуля отчетов

Название функции	Строка адреса	Метод REST API	Описание
getReport	/api/admin/reports	GET	Сводный отчет: выручка, заказы по статусам, топ продаж, низкий остаток, статистика за 30 дней

В листинге 2 в качестве примера реализации одного из функционала приложения приведен код метода createOrder REST API интерфейса серверного

приложения. Для создания заказа первым шагом является получение данных пользователя для оформления его заказа.

Листинг 2 – Код REST API интерфейса приложения

```
@Service
@RequiredArgsConstructor
public class OrderService {
    private final OrderRepository orderRepository;
    private final PartRepository partRepository;
    @Transactional
    public OrderResponse createOrder(User user, OrderCreateRequest
request) {
        Order order = Order.builder()
            .user(user)
            .deliveryAddress(request.getDeliveryAddress())
            .totalPrice(BigDecimal.ZERO)
            .build();
        BigDecimal totalPrice = BigDecimal.ZERO;
```

Далее идет цикл по заказу и для каждой детали в заказе проверяется ее доступность на складе и уменьшается ее количество на один в случае успешности оформления заказа:

```
for (OrderItemRequest itemRequest : request.getItems()) {
    Part part = partRepository.findById(itemRequest.getPartId())
        .orElseThrow(() -> new ResourceNotFoundException("Part
not found: " + itemRequest.getPartId()));
    if (part.getQuantityInStock() < itemRequest.getQuantity()) {
        throw new InsufficientStockException(
            "Insufficient stock for part '" + part.getName() +
            "'. Available: " + part.getQuantityInStock() +
            ", requested: " + itemRequest.getQuantity());
    }
    part.setQuantityInStock(part.getQuantityInStock() -
itemRequest.getQuantity());
    partRepository.save(part);
    OrderItem orderItem = OrderItem.builder()
        .order(order)
        .part(part)
        .quantity(itemRequest.getQuantity())
        .priceAtOrder(part.getPrice())
        .build();
    order.getItems().add(orderItem);
    totalPrice =
totalPrice.add(part.getPrice().multiply(BigDecimal.valueOf(itemRequ
est.getQuantity())));
}
```

Последним шагом является сохранение в базу данных успешного оформленного заказа:

```

order.setTotalPrice(totalPrice);
    Order saved = orderRepository.save(order);
    return toResponse(saved);
}

```

Обмен данными между клиентом и сервером реализуется с помощью использования форма JSON. JSON (Javascript Object Notation) – это текстовый формат обмена данными, представляющий информацию в виде пары «ключ – значение», который легко читается человеком и обрабатывается программами. В данном случае клиентское приложение формирует JSON – строку с необходимыми полями и отправляет ее на сервер. Серверная часть принимает эти данные и автоматически преобразует JSON в объекты Java. Преобразование выполняет встроенная в Spring библиотека Jackson, которая сопоставляет поля JSON с полями соответствующего класса. Для этого были разработаны классы-модели (DTO), описывающие структуру передаваемых данных и содержащие необходимые типы полей (листинг 3). После десериализации полученные объекты передаются в сервисный слой приложения, где над ними выполняется бизнес-логика работы фирмы.

Листинг 3 – Код модели, содержащий информацию о заказе

```

@Data
public class OrderCreateRequest {
    @NotBlank(message = "Delivery address is required")
    private String deliveryAddress;
    @NotEmpty(message = "Order must contain at least one item")
    @Valid
    private List<OrderItemRequest> items;
}

```

Для взаимодействия с базой данных в сервисном слое DTO преобразует в Entity.

Entity (сущность) в Java – это класс, который соответствует таблице в базе данных: один объект такого класса представляет одну строку таблицы, а его поля – столбцы. По сути это связующее звено между объектами в коде и данными в БД, используемое технологией ORM (Object-Relational Mapping), в экосистеме Java – JPA/Hibernate [16]. Помечает класс аннотацией @Entity. В листинге 4 приведен код Entity заказа.

Листинг 4 – Код Entity, содержащий информацию о заказе

```

@Entity
@Table(name = "orders")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    @Builder.Default
    private OrderStatus status = OrderStatus.NEW;
    @Column(nullable = false, precision = 10, scale = 2)
    private BigDecimal totalPrice;
    @Column(columnDefinition = "TEXT")
    private String deliveryAddress;
    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL,
orphanRemoval = true)
    @Builder.Default
    private List<OrderItem> items = new ArrayList<>();
    @Column(nullable = false, updatable = false)
    private LocalDateTime createdAt;
    @Column(nullable = false)
    private LocalDateTime updatedAt;
}

```

Таким образом, была реализована серверная часть приложения информационной системы фирмы по продаже и подбору автозапчастей.

3.4 Разработка базы данных фирмы по продаже и подбору автодеталей

Для хранения данных о деталях, заказах и статистика была развернута база данных. Информация в ней организована в виде связанных между собой таблиц, каждая из которых соответствует определённой сущности предметной области: клиентам, автомобилям, запчастям, заказам и позициям заказов.

Концептуальная модель базы данных представлена на рисунке 21.

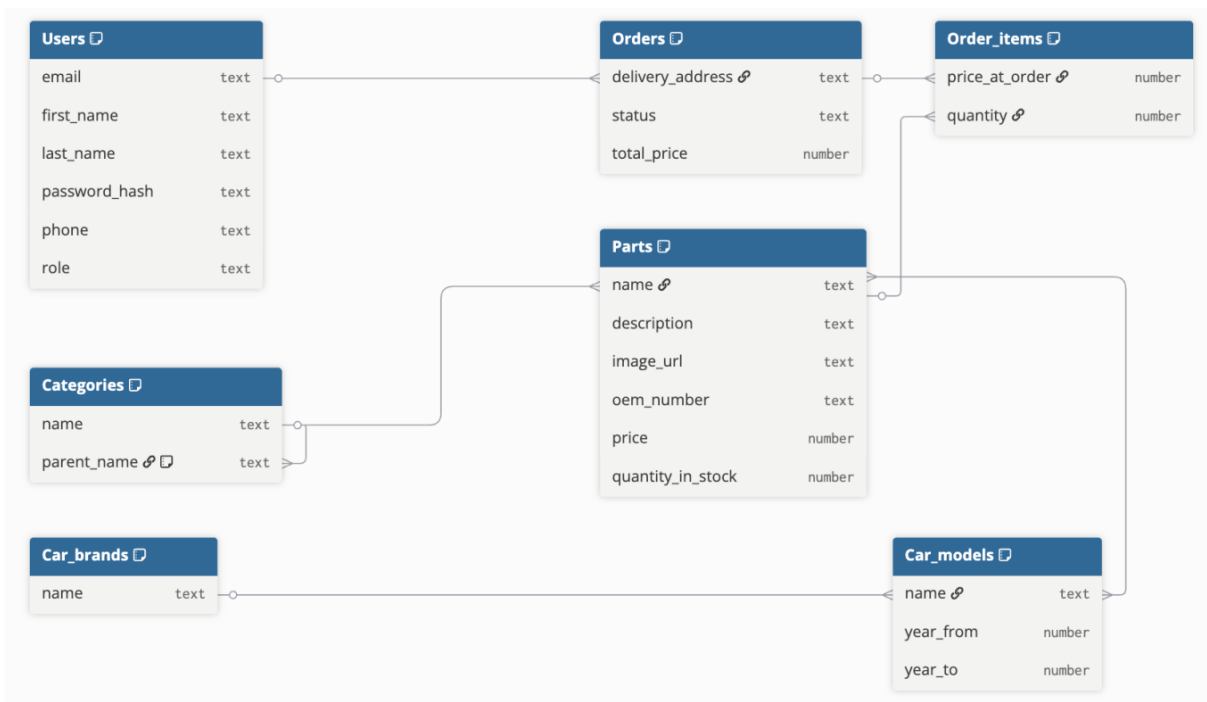


Рисунок 21 – Концептуальная модель базы данных

Концептуальная модель базы данных информационной системы фирмы по продаже и подбору автозапчастей формирует целостное представление о предметной области и позволяет на верхнем уровне выделить основные сущности и определить связь между ними.

Логическая модель базы данных представлена на рисунке 22.

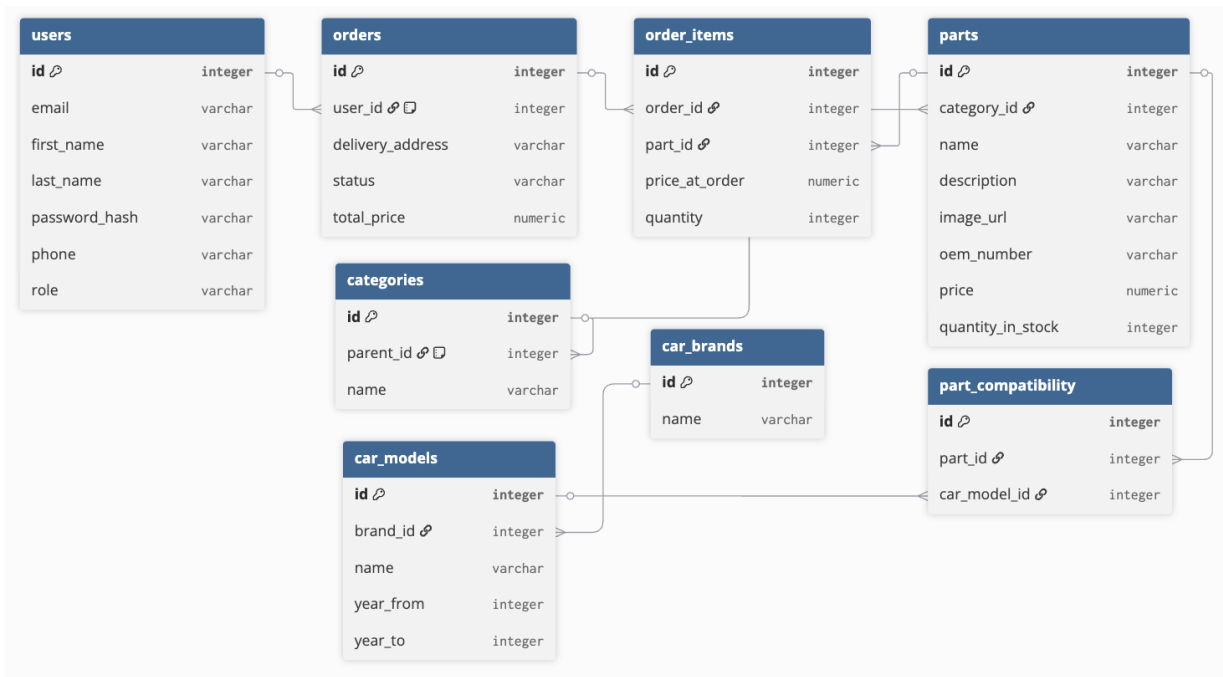


Рисунок 22 – Логическая модель базы данных

Логическая модель базы данных информационной системы фирмы по продаже и подбору автозапчастей способствует определению хранимых типов данных и служит основой для разработки физической модели.

Физическая модель развернутой базы данных представлена на рисунке 23.

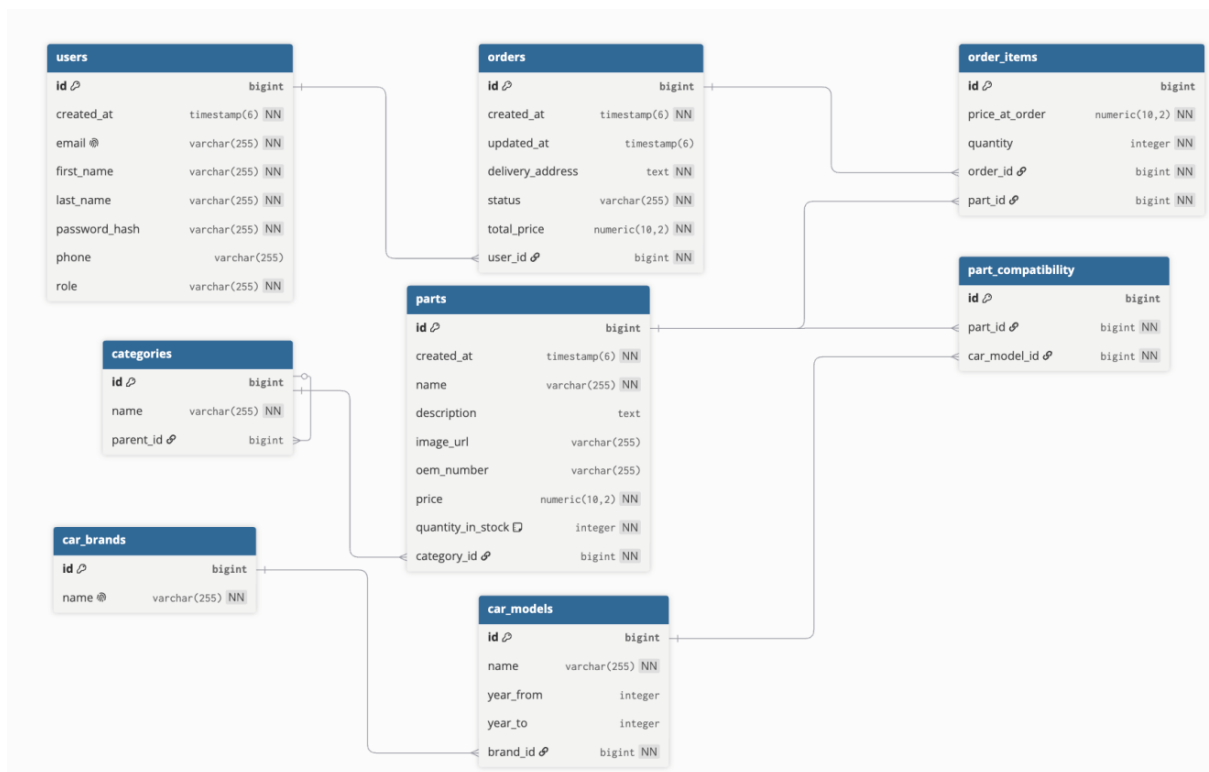


Рисунок 23 – Физическая модель базы данных

В качестве СУБД для хранения данных веб-приложения фирмы по продаже и подбору автозапчастей была выбрана PostgreSQL – реляционная система управления базами данных [17]. Она отличается надёжностью, поддерживает согласованность связей между сущностями и работает под любой операционной системой. PostgreSQL обеспечивает целостность хранимых данных и сохраняет высокую производительность даже при одновременной работе большого числа пользователей.

Описание сущностей представлены в таблицах 12 – 18.

Таблица 12 – Описание сущностей заказов

Атрибут	Тип данных	Ограничения	Описание
id	Integer	PK	Уникальный идентификатор заказа
created_at	timestamp(6)	Not nul	Время создания заказа

Продолжение таблицы 12

Атрибут	Тип данных	Ограничения	Описание
delivery_address	text		Адрес доставки
status	text	Not null	Статус заказа
total_price	numeric(10,2)	Not null	Общая цена заказа
updated_at	timestamp(6)	Not null	Время обновления заказа
user_id	bigint		Внешний ключ – уникальный идентификатор пользователя

Таблица 13 – Таблица сущностей деталей

Атрибут	Тип данных	Ограничения	Описание
id	bigint	PK	Уникальный идентификатор детали
created_at	timestamp(6)	Not null	Время добавления детали
description	text		Описание детали
image_url	varchar(255)	Not null	Ссылка на изображение детали
name	text	Not null	Название детали
oem_number	varchar(255)		ОЕМ номер детали
price	numeric(10,2)	Not null	Цена детали
quantity_in_stock	integer	Not null	Количество деталей на складе
category_id	bigint	Not null	Внешний ключ – уникальный идентификатор категории

Таблица 14 – Сущности договора модели

Атрибут	Тип данных	Ограничения	Описание
id	bigint	PK	Уникальный идентификатор модели
name	varchar(255)	Not null	Название модели
year_from	integer	Not null	Год начала выпуска модели
year_to	integer		Год окончания выпуска модели
brand_id	bigint	Not null	Внешний ключ – уникальный идентификатор марки

Таблица 15 – Описание сущностей марки автомобилей

Атрибут	Тип данных	Ограничения	Описание
id	bigint	PK	Уникальный идентификатор марки
name	Varchar(255)	Not null	Название марки

Таблица 16 – Сущности категории деталей

Атрибут	Тип данных	Ограничения	Описание
id	bigint	PK	Уникальный идентификатор категории
name	varchar(255)	Not null	Название категории
parent_id	bigint		Ключ родительское категории (в случае если это подкатегория)

Таблица 17 – Сущности деталей в оформленном заказе

Атрибут	Тип данных	Ограничения	Описание
id	bigint	PK	Уникальный идентификатор
price_at_order	numeric(10,2)	Not null	Цена детали, зафиксированная на момент оформления заказа
quantity	integer	Not null	Количество деталей, зафиксированные на момент оформления заказа
order_id	bigint	Not null	Внешний ключ – уникальный идентификатор заказа
part_id	bigint	Not null	Внешний ключ – уникальный идентификатор детали

Таблица 18 – Сущности пользователей

Атрибут	Тип данных	Ограничения	Описание
id	bigint	PK	Уникальный идентификатор пользователя
created_at	timestamp(6)	Not null	Время создания записи о пользователе
email	varchar(255)	Not null	Email пользователя
first_name	varchar(255)		Имя пользователя

Продолжение таблицы 18

Атрибут	Тип данных	Ограничения	Описание
last_name	varchar(255)		Фамилия пользователя
password_hash	varchar(255)	Not null	Хэш пароля пользователя
phone	varchar(255)		Телефон пользователя
role	varchar(255)	Not null	Роль пользователя в системе

Представленные сущности формируют основу реляционной базы данных информационной системы и отражают ключевые объекты предметной области. Связи между таблицами обеспечивают целостность данных и позволяют реализовать основные бизнес-процессы системы, включая оформление заказов, хранение информации о пользователях и управление каталогом автозапчастей. Разработанная структура базы данных является нормализованной и обеспечивает эффективное хранение и обработку информации.

4 Контроль качества программного обеспечения информационной системы фирмы по продаже и подбору автомобильных запчастей

4.1 Планирование процесса контроля качества программного обеспечения

4.1.1 Мероприятия по контролю качества программного обеспечения

Любое приложение перед тем, как будет выпущено для использования конечными пользователями должно быть протестировано, чтобы сразу на его старте исправить возможные найденные баги и повысить качество и надежность продукта. Контроль качества разрабатываемого ПО был выполнен с помощью покрытия тестами основного функционала приложение и оценки трудозатрат на написание кода.

Существует несколько моделей тестирования: функциональное тестирование, нефункциональное тестирование.

Функциональное тестирование состоит из проверки функций приложения, которые уже готовы, но могут показывать некорректную работу в разных вариантах поведения приложения.

Можно выделить две основные модели тестирования: функциональное и нефункциональное. Функциональное тестирование чаще всего проводится ручным методом и покрывает различные сценарии использования приложения. Нефункциональное тестирование имеет большое количество подвидов, рассмотрим их.

К нефункциональному тестированию можно отнести:

- юнит-тестирование;
- тестирование с помощью интеграционных тестов;
- нагрузочное тестирование;
- тестирование пользовательского интерфейса;

- тестирование с помощью автотестов.

Разработанное ПО было протестировано с помощью юнит – тестов и интеграционных тестов.

Помимо тестирования, был проведён этап оценки качества кода. Он заключается в анализе исходного кода для выявления его слабых мест с целью последующего рефакторинга и улучшения. В качестве метрики оценки кода была выбрана метрика SLOC (Source Lines of Code), отражающая объём программного кода по числу строк.

4.2 Обоснование выбора модели тестирования

Для тестирования автоматизированной информационной системы фирмы по продаже и подбору автозапчастей было выбрано юнит-тестирование и интеграционное тестирование.

Юнит-тестирование (модульное) – это вид тестирования, при котором проверяется работоспособность отдельных модулей, методов или небольших фрагментов кода в изоляции от остальной системы. Такой подход позволяет быстро находить и устранять ошибки, в том числе при доработке и обновлении приложения.

Преимущества использования модели юнит-тестирования:

- легкое и быстрое написание тестов с использованием обширного числа библиотек;
- возможность разработчикам параллельно писать тесты для покрытия различного функционала;
- возможность найти точечную проблему в ПО с использованием логгирования приложения;
- можно использовать при регрессионном тестировании для проверки работоспособности ранее написанного функционала.

Недостатки использования юнит-тестирования:

- может не покрывать полностью работоспособность ПО если не были написаны тесты на определенные сценарии;

- тестируются отдельные точечные единицы функционала, а не все ПО;
- чаще всего используются заглушки для той части функционала, которая не проверяется в тестировании.

Интеграционное тестирование – это вид тестирования, при котором проверяется совместная работа нескольких модулей или компонентов системы [18]. Если юнит-тесты проверяют каждый модуль по отдельности, то интеграционные тесты убеждаются, что эти модули правильно взаимодействуют друг с другом.

В ходе такого тестирования проверяются зависимости между компонентами, корректность передачи и целостность данных при их движении между слоями приложения, а также взаимодействие программы с базой данных и внешними системами. Тем самым воспроизводится поведение приложения в условиях, близких к реальным, и выявляются ошибки, которые проявляются именно на стыках компонентов и не видны при изолированной проверке отдельных модулей.

Основная цель интеграционного тестирования – убедиться, что связанные между собой части системы работают согласованно, данные сохраняются и передаются без потерь, а обращения к базе данных выполняются корректно.

Преимущества сквозного тестирования:

- проверка взаимодействия совместной работы моделей;
- выявление ошибок на стыках компонентов и при работе с БД;

Недостатки сквозного тестирования:

- сложность и затрата большого количества ресурсов при написании;
- сложность локализации причины ошибки, когда задействовано несколько компонентов;
- необходимость постоянно поддерживать и обновлять тесты при изменении системы.

Таким образом, выбранные модели тестирования обеспечили проверку информационной системы фирмы по продаже и подбору автозапчастей.

4.3 Обоснование выбора метрик программного кода

Использование метрик даёт возможность оценить различные характеристики программного обеспечения: его качество, сложность, надёжность, объём, а также трудозатраты на разработку.

Одной из самых простых метрик является SLOC (Source Lines of Code) – подсчёт количества строк исходного кода. Она была разработана для оценки объёма выполненной по проекту работы.

В качестве основной метрики выбрана именно SLOC, поскольку она позволяет оценить объём написанного кода.

4.4 Тестирование программных средств

4.4.1 Тестирование с помощью юнит-тестов

Для тестирования серверной части приложения были реализованы юнит-тесты. В таблице 19 представлен список юнит-тестов.

Данные тесты были написаны на Java с помощью библиотеки Junit [19].

Таблица 19 – Юнит тесты серверной части приложения

Название юнит-теста	Описание
register_succes	Успешная регистрация: пользователь сохраняется в БД
register_duplicateEmail_throws Exception	Попытка регистрации с уже занятым email – выбрасывается исключение, запись в БД не создается
login_wrongCredentials_throwsException	Вход с неверным паролем – выбрасывается исключение BadCredentialsException, поиск по пользователю не выполняется
createPart_success	Успешное добавление запчастей менеджером: данные сохраняются корректно
createOrder_success	Успешное оформление заказа: сумма рассчитана верно, остаток на складе не уменьшается
createOrder_insufficientStock_throwsException	Оформление заказа при нехватке товара – выбрасывается исключение, заказ не сохраняется
updateOrderStatus_success	Смена статуса заказа администратором: статус обновляется и сохраняется в БД

Данные тесты были нацелены на проверку бизнес-логики информационной системы фирмы по продаже и подбору автозапчастей.

Основной функционал приложения покрыт тестами на 40%.

Общий результат выполнения юнит-тестов представлен на рисунке 24.

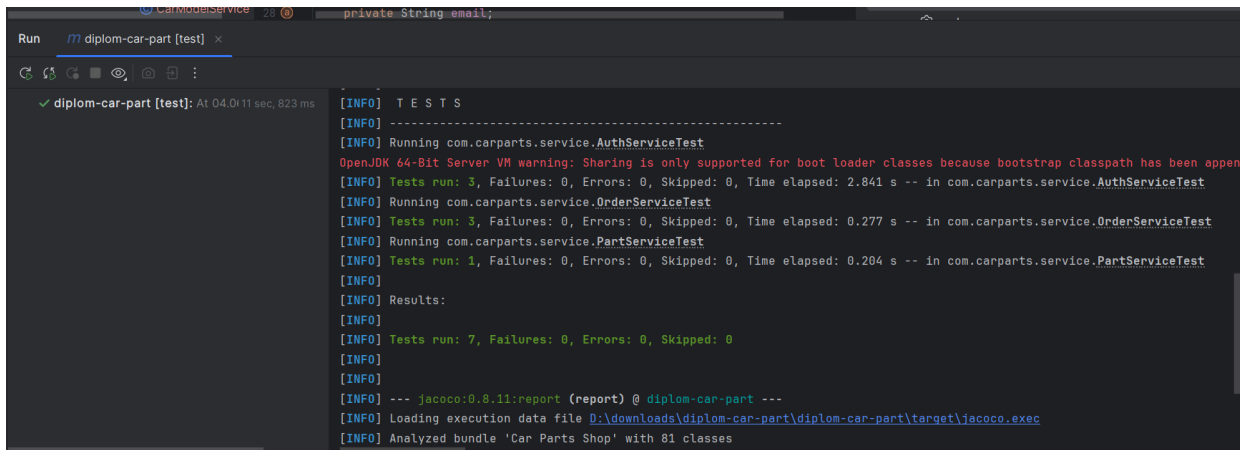


Рисунок 24 – Результат выполнения юнит-тестов

Оставшаяся часть функционала была протестирована с помощью интеграционного тестирования.

4.4.2 Автоматическое интеграционное тестирование

Интеграционные тесты покрывают взаимодействие клиентской части с серверной частью. Тесты написаны на языке javascript(jsx) с использованием библиотеки Vitest.

Таблица 20 – Спецификации автотестов

Название	Описание
login_success_redirects_to_home	Пользователь вводит корректные email и пароль, нажимает «Войти». Проверяется, что API входа вызван с правильными данными и произошёл редирект на главную страницу
login_invalid_credentials_stays_on_page	Пользователь вводит неверные учётные данные. Проверяется, что API был вызван, но редиректа не произошло и форма входа по-прежнему отображается

Продолжение таблицы 20

Название	Описание
register_success_redirects_to_home	Пользователь заполняет форму регистрации. Проверяется, что API регистрации вызван с введенными данными и произошёл редирект на главную страницу
catalog_add_to_cart_updates_context	Страница каталога загружает товары через API. Пользователь нажимает «В корзину». Проверяется, что счётчик корзины увеличился и товар сохранился в localStorage
cart_unauthenticated_checkout_redirects_to_login	Корзина содержит товар, пользователь не авторизован. При нажатии «Оформить заказ» проверяется редирект на страницу входа
checkout_success_clears_cart_and_shows_confirmation	Пользователь заполняет адрес доставки и оформляет заказ. Проверяется, что API вызван с корректными данными, корзина очищена и отображается экран подтверждения

Общий результат выполнения интеграционных тестов представлен на рисунке 25.

```

✓ src/test/integration.test.jsx (6 tests) 10733ms
✓ Интеграционные тесты клиентской части (6)
  ✓ Тест 1: успешный вход перенаправляет на главную страницу 1791ms
  ✓ Тест 2: неверный пароль – нет редиректа, форма остаётся 1019ms
  ✓ Тест 3: успешная регистрация отправляет данные и перенаправляет на главную 2918ms
  ✓ Тест 4: каталог загружает товары из API и добавление обновляет корзину 2035ms
  ✓ Тест 5: неавторизованный пользователь при нажатии «Оформить заказ» попадает на страницу входа 1169ms
  ✓ Тест 6: успешное оформление заказа очищает корзину и показывает экран подтверждения 1799ms

Test Files 1 passed (1)
Tests 6 passed (6)
Start at 02:59:04
Duration 28.34s (transform 5.66s, setup 197ms, collect 15.13s, tests 10.73s, environment 1.47s, prepare 250ms)

```

Рисунок 25 – Результат выполнения интеграционных тестов

Результаты проведённого тестирования подтверждают корректность взаимодействия между клиентской и серверной частями приложения.

4.5 Оценка качества программного кода на основе метрик

4.5.1 Метрика SLOC

Метрика SLOC (Source Line Of Code) – это способ измерения объема программы путем подсчета количества строк написанного кода. Это одна из

самых простых и старейших метрик в разработке программного обеспечения, применяемая для оценки размера проекта и трудозатрат на его создание.

Расчет метрики удобным способом позволяет выполнить консольная утилита cloc [20]. Расчет был выполнен для серверной части приложения и представлен на рисунке 26.

```
----- Result -----  
Physical : 2878  
Source : 2333  
Comment : 18  
Single-line comment : 18  
Block comment : 0  
Mixed : 0  
Empty block comment : 0  
Empty : 527  
To Do : 0  
  
Number of files read : 67  
-----
```

Рисунок 26 – Результат измерений серверной части приложения

Из приведенных данных можно выделить:

- физические строки кода – 2878;
- логические строки кода – 233;
- комментарии – 18;
- пустые строки – 527.

Можно сделать вывод о том, что серверная часть приложения довольно объемна, самодокументирована (малый объем комментариев), малое количество пустых строк обеспечивает большую читаемость кода.

Заключение

В ходе выполнения работы была изучена предметная область, проведен сравнительный анализ различных аналогов приложения, разработаны требования к приложению, выполнен анализ инструментов для реализации приложения.

Приложение состоит из клиентской и серверной части. Серверная часть приложения была реализована с использованием программной среды для разработки IntelliJ Idea на языке программирования Java, фреймворк – Spring. Клиентская часть приложения была реализована в редакторе VSCode с использованием языка JavaScript и его фреймворка React.

Приложение было протестировано с помощью юнит и интеграционных тестов, были посчитаны метрики SLOC. Результаты тестирования свидетельствуют о работоспособности и надежности приложения.

Скриншоты работы готового приложения представлены в Приложении Б.

Все поставленные цели работы успешно выполнены, web – приложение информационной системы фирмы по продаже и подбору автозапчастей успешно разработано и готово к использованию конечным потребителем.

Список использованных источников

1. Гражданский кодекс Российской Федерации (часть вторая) [Электронный ресурс]. URL: https://www.consultant.ru/document/cons_doc_LAW_9027/ (дата обращения: 22.05.2026).
2. Закон РФ «О защите прав потребителей» [Электронный ресурс]. URL: https://www.consultant.ru/document/cons_doc_LAW_305/ (дата обращения: 22.05.2026).
3. Microsoft Project: справка и обучение [Электронный ресурс]. URL: <https://support.microsoft.com/ru-ru/project> (дата обращения: 22.05.2026).
4. UML Diagrams: язык и типы диаграмм [Электронный ресурс]. URL: <https://www.uml-diagrams.org/> (дата обращения: 22.05.2026).
5. Use Case Diagrams [Электронный ресурс]. URL: <https://www.uml-diagrams.org/use-case-diagrams.html> (дата обращения: 22.05.2026).
6. Клиент-серверная архитектура [Электронный ресурс]. URL: <https://servergate.ru/articles/klient-servernaya-arkhitektura/> (дата обращения: 22.05.2026).
7. Что такое REST API [Электронный ресурс]. URL: <https://restfulapi.net/> (дата обращения: 22.05.2026).
8. Формат данных JSON [Электронный ресурс]. URL: <https://www.json.org/json-ru.html> (дата обращения: 22.05.2026).
9. Документация языка Java (Oracle) [Электронный ресурс]. URL: <https://docs.oracle.com/en/java/> (дата обращения: 22.05.2026).
10. Документация Spring Boot [Электронный ресурс]. URL: <https://docs.spring.io/spring-boot/index.html> (дата обращения: 22.05.2026).
11. Spring Initializr [Электронный ресурс]. URL: <https://start.spring.io/> (дата обращения: 22.05.2026).
12. React [Электронный ресурс]. URL: <https://react.dev/learn> (дата обращения: 22.05.2026).
13. TypeScript Documentation [Электронный ресурс]. URL:

<https://www.typescriptlang.org/docs/> (дата обращения: 22.05.2026).

14. Ant Design – A UI Design Language [Электронный ресурс]. URL: <https://ant.design/docs/react/introduce> (дата обращения: 22.05.2026).

15. Spring Security Reference [Электронный ресурс]. URL: <https://docs.spring.io/spring-security/reference/> (дата обращения: 22.05.2026).

16. Документация Hibernate ORM [Электронный ресурс]. URL: <https://hibernate.org/orm/documentation/> (дата обращения: 22.05.2026).

17. Документация PostgreSQL [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения: 22.05.2026).

18. Integration Testing [Электронный ресурс]. URL: <https://www.guru99.com/integration-testing.html> (дата обращения: 22.05.2026).

19. JUnit 5 User Guide [Электронный ресурс]. URL: <https://junit.org/junit5/docs/current/user-guide/> (дата обращения: 22.05.2026).

20. cloc – Count Lines of Code [Электронный ресурс]. URL: <https://github.com/AlDanial/cloc> (дата обращения: 22.05.2026).

Приложение А

(обязательное)

Листинг программы

Листинг А.1 – Роутер App.jsx

```

export default function App() {
  return (
    <ConfigProvider
      theme={{
        token: {
          colorPrimary: '#1677ff',
        },
      }}
    >
      <Routes>
        {/* Public routes */}
        <Route element={<AppLayout />} />
        <Route path="/" element={<HomePage />} />
        <Route path="/catalog" element={<CatalogPage />} />
        <Route path="/catalog/:id" element={<PartDetailPage />} />

        <Route path="/login" element={<LoginPage />} />
        <Route path="/register" element={<RegisterPage />} />

        {/* Client-protected routes */}
        <Route element={<ProtectedRoute />} />
        <Route path="/cart" element={<CartPage />} />
        <Route path="/checkout" element={<CheckoutPage />} />
        <Route path="/orders" element={<MyOrdersPage />} />
        <Route path="/orders/:id" element={<OrderDetailPage />} />

        <Route path="/profile" element={<ProfilePage />} />
      </Route>
    </Route>

    {/* Admin routes */}
    <Route element={<AdminRoute />} />
    <Route element={<AdminLayout />} />
    <Route path="/admin" element={<DashboardPage />} />
    <Route path="/admin/parts" element={<PartsListPage />} />

    <Route path="/admin/parts/new" element={<PartFormPage />} />
    <Route path="/admin/parts/:id/edit" element={<PartFormPage />} />
    <Route path="/admin/categories" element={<CategoriesPage />} />
    <Route path="/admin/brands-models" element={<BrandsModelsPage />} />
    <Route path="/admin/orders" element={<OrdersListPage />} />
    <Route path="/admin/orders/:id" element={<OrderManagePage />} />
    <Route path="/admin/users" element={<UsersListPage />} />

    <Route path="/admin/vin-search" element={<VinSearchPage />} />
  )
}

```

```

/>} />
    </Route>
  </Route>
</Routes>
</ConfigProvider>
);
}

```

Листинг A.2 – Страница CatalogPage.jsx

```

const { Search } = Input;
const { Title } = Typography;

export default function CatalogPage() {
  const [searchParams] = useSearchParams();
  const navigate = useNavigate();
  const { addToCart } = useCart();
  const [parts, setParts] = useState([]);
  const [total, setTotal] = useState(0);
  const [loading, setLoading] = useState(true);
  const [page, setPage] = useState(0);
  const [filters, setFilters] = useState({
    search: searchParams.get('search') || undefined,
  });

  useEffect(() => {
    setLoading(true);
    const params = { ...filters, page, size: 12 };
    // Remove undefined values
    Object.keys(params).forEach((k) => params[k] === undefined &&
delete params[k]);

    getParts(params)
      .then((res) => {
        setParts(res.data.content);
        setTotal(res.data.totalElements);
      })
      .catch(() => {})
      .finally(() => setLoading(false));
  }, [filters, page]);

  const onSearch = (value) => {
    setPage(0);
    setFilters((prev) => ({ ...prev, search: value || undefined
})));
  };

  const onFilter = (newFilters) => {
    setPage(0);
    setFilters(newFilters);
  };

  const handleAddToCart = (e, part) => {
    e.stopPropagation();
  };

```

```

    addToCart(part);
    message.success(`${part.name} добавлен в корзину`);
  };

  return (
    <div>
      <Title level={2}>Каталог</Title>
      <Row gutter={24}>
        <Col xs={24} md={6}>
          <PartFilter filters={filters} onFilter={onFilter} />
        </Col>
        <Col xs={24} md={18}>
          <Search
            placeholder="Поиск запчастей..."
            defaultValue={filters.search}
            onSearch={onSearch}
            size="large"
            style={{ marginBottom: 24 }}
            allowClear
          />

          {loading ? (
            <div style={{ textAlign: 'center', padding: 60 }}><Spin
size="large" /></div>
          ) : parts.length === 0 ? (
            <Empty description="Запчасти не найдены" />
          ) : (
            <>
              <Row gutter={[16, 16]}>
                {parts.map((part) => (
                  <Col xs={24} sm={12} lg={8} key={part.id}>
                    <Card
                      hoverable
                      onClick={() =>
navigate(`/catalog/${part.id}`)}
                      cover={
                        part.imageUrl ? (
                          <img alt={part.name} src={part.imageUrl}
style={{ height: 160, objectFit: 'cover' }} />
                        ) : (
                          <div style={{ height: 160, background:
'#f5f5f5', display: 'flex', alignItems: 'center', justifyContent:
'center' }}>
                            <ToolOutlined style={{ fontSize: 40,
color: '#ccc' }} />
                          </div>
                        )
                      )
                    </div>
                  )
                }
              </Row>
            </>
          )
        }
      </div>
      <div>
        <Button
          key="add"
          type="link"
          icon={<ShoppingCartOutlined />}
        />
      </div>
    </div>
  );

```

```

        onClick={ (e) => handleAddToCart(e, part)}
        disabled={part.quantityInStock === 0}
      >
        {part.quantityInStock === 0 ? 'Нет в
наличии' : 'В корзину'}
      </Button>,
    ]}
  >
    <Card.Meta
      title={part.name}
      description={
        <div>
          <div style={{ color: '#1677ff',
fontSize: 18, fontWeight: 'bold' }}>
            {formatPrice(part.price)}
          </div>
          <div style={{ color: '#999', fontSize:
12, marginTop: 4 }}>
            {part.categoryName}
            {part.oemNumber && ` | OEM:
${part.oemNumber}`}
          </div>
          <div style={{ color:
part.quantityInStock > 0 ? 'green' : 'red', fontSize: 12 }}>
            {part.quantityInStock > 0 ? `В
наличии: ${part.quantityInStock}` : 'Нет в наличии'}
          </div>
        </div>
      }
    </Card>
  </Col>
)}
</Row>
<div style={{ textAlign: 'center', marginTop: 24 }}>
  <Pagination
    current={page + 1}
    total={total}
    pageSize={12}
    onChange={ (p) => setPage(p - 1)}
    showSizeChanger={false}
  />
</div>
</>
)}
</Col>
</Row>
</div>
);
}

```

Листинг А.3 – Страница HomePage.jsx

```

const { Title, Paragraph } = Typography;
const { Search } = Input;

export default function HomePage() {
  const [parts, setParts] = useState([]);
  const [loading, setLoading] = useState(true);
  const navigate = useNavigate();

  useEffect(() => {
    getParts({ size: 8, sort: 'createdAt,desc' })
      .then((res) => setParts(res.data.content))
      .catch(() => {})
      .finally(() => setLoading(false));
  }, []);

  const onSearch = (value) => {
    if (value.trim()) {
      navigate(`/catalog?search=${encodeURIComponent(value.trim())}`);
    } else {
      navigate('/catalog');
    }
  };

  return (
    <div>
      {/* Hero Section */}
      <div style={{ textAlign: 'center', padding: '60px 0 40px' }}>
        <Title level={1}>Магазин автозапчастей</Title>
        <Paragraph style={{ fontSize: 18, color: '#666',
marginBottom: 32 }}>
          Найдите нужные запчасти для вашего автомобиля. Поиск по
названию, марке или модели.
        </Paragraph>
        <Search
          placeholder="Поиск по названию или OEM-номеру..."
          enterButton={<><SearchOutlined /> Найти</>}
          size="large"
          onSearch={onSearch}
          style={{ maxWidth: 600 }}
        />
      </div>

      {/* Features */}
      <Row gutter={[24, 24]} style={{ marginBottom: 48 }}>
        <Col xs={24} md={8}>
          <Card hoverable style={{ textAlign: 'center' }}>
            <CarOutlined style={{ fontSize: 40, color: '#1677ff',
marginBottom: 16 }} />
            <Title level={4}>Поиск по автомобилю</Title>
            <Paragraph>Выберите марку, модель и год выпуска для

```

```

поиска подходящих запчастей.</Paragraph>
    </Card>
  </Col>
  <Col xs={24} md={8}>
    <Card hoverable style={{ textAlign: 'center' }}>
      <ToolOutlined style={{ fontSize: 40, color: '#1677ff',
marginBottom: 16 }} />
      <Title level={4}>Качественные запчасти</Title>
      <Paragraph>ОЕМ и аналоги от проверенных
производителей.</Paragraph>
    </Card>
  </Col>
  <Col xs={24} md={8}>
    <Card hoverable style={{ textAlign: 'center' }}>
      <ShoppingOutlined style={{ fontSize: 40, color:
'#1677ff', marginBottom: 16 }} />
      <Title level={4}>Удобный заказ</Title>
      <Paragraph>Добавьте в корзину и оформите заказ с
доставкой на ваш адрес.</Paragraph>
    </Card>
  </Col>
</Row>

  { /* Latest Parts */ }
  <Title level={3}>Новые поступления</Title>
  { loading ? (
    <div style={{ textAlign: 'center', padding: 40 }}><Spin
size="large" /></div>
  ) : (
    <Row gutter={[16, 16]}>
      {parts.map((part) => (
        <Col xs={24} sm={12} md={8} lg={6} key={part.id}>
          <Card
            hoverable
            onClick={() => navigate(`/catalog/${part.id}`)}
            cover={
              part.imageUrl ? (
                <img alt={part.name} src={part.imageUrl}
style={{ height: 180, objectFit: 'cover' }} />
              ) : (
                <div style={{ height: 180, background:
'#f5f5f5', display: 'flex', alignItems: 'center', justifyContent:
'center' }}>
                  <ToolOutlined style={{ fontSize: 48, color:
'#ccc' }} />
                </div>
              )
            }
          >
            <Card.Meta
              title={part.name}
              description={
                <div>

```

```

        <div style={{ color: '#1677ff', fontSize: 18,
fontWeight: 'bold', margin: '8px 0' }}>
            {formatPrice(part.price)}
        </div>
        <div style={{ color: '#999', fontSize: 12
}}>{part.categoryName}</div>
        </div>
    }
    />
</Card>
</Col>
    )}
</Row>
)}

    <div style={{ textAlign: 'center', margin: '32px 0' }}>
        <Button type="primary" size="large" onClick={() =>
navigate('/catalog')}>
            Смотреть весь каталог
        </Button>
    </div>
</div>
);
}

```

Листинг А.4 – Страница LoginPage.jsx

```

const { Title } = Typography;

export default function LoginPage() {
    const [loading, setLoading] = useState(false);
    const { login } = useAuth();
    const navigate = useNavigate();

    const onFinish = async (values) => {
        setLoading(true);
        try {
            await login(values.email, values.password);
            message.success('Вход выполнен!');
            navigate('/');
        } catch (err) {
            message.error(err.response?.data?.message || 'Неверный email
или пароль');
        } finally {
            setLoading(false);
        }
    };

    return (
        <div style={{ maxWidth: 400, margin: '60px auto' }}>
            <Card>
                <Title level={3} style={{ textAlign: 'center'
}}>Вход</Title>
                <Form layout="vertical" onFinish={onFinish}>

```

```

        <Form.Item
          name="email"
          rules={[
            { required: true, message: 'Введите email' },
          ]}
        >
          <Input prefix={<MailOutlined />} placeholder="Email"
size="large" />
        </Form.Item>
        <Form.Item
          name="password"
          rules={[{ required: true, message: 'Введите пароль' }]}
        >
          <Input.Password prefix={<LockOutlined />}
placeholder="Пароль" size="large" />
        </Form.Item>
        <Form.Item>
          <Button type="primary" htmlType="submit"
loading={loading} block size="large">
            Войти
          </Button>
        </Form.Item>
      </Form>
      <div style={{ textAlign: 'center' }}>
        Нет аккаунта? <Link
to="/register">Зарегистрируйтесь</Link>
      </div>
    </Card>
  </div>
);
}

```

Листинг А.5 – Страница PartDetailPage.jsx

```

const { Title } = Typography;

export default function PartDetailPage() {
  const { id } = useParams();
  const navigate = useNavigate();
  const { addToCart } = useCart();
  const [part, setPart] = useState(null);
  const [loading, setLoading] = useState(true);
  const [quantity, setQuantity] = useState(1);

  useEffect(() => {
    getPartById(id)
      .then((res) => setPart(res.data))
      .catch(() => message.error('Запчасть не найдена'))
      .finally(() => setLoading(false));
  }, [id]);

  if (loading) {
    return <div style={{ textAlign: 'center', padding: 100 }}><Spin
size="large" /></div>;
  }
}

```

```

    }

    if (!part) {
      return <div style={{ textAlign: 'center', padding: 100 }}>Запчасть не найдена</div>;
    }

    const handleAdd = () => {
      addToCart(part, quantity);
      message.success(`${part.name} (x${quantity}) добавлен в корзину`);
    };

    const compatColumns = [
      { title: 'Марка', dataIndex: 'brandName', key: 'brand' },
      { title: 'Модель', dataIndex: 'name', key: 'model' },
      {
        title: 'Годы',
        key: 'years',
        render: (_, record) => `${record.yearFrom}-${record.yearTo} || '...'\``,
      },
    ];

    return (
      <div>
        <Button icon={<ArrowLeftOutlined />} onClick={() => navigate(-1)} style={{ marginBottom: 16 }}>
          Назад
        </Button>

        <Row gutter={32}>
          <Col xs={24} md={10}>
            {part.imageUrl ? (
              <img src={part.imageUrl} alt={part.name} style={{
width: '100%', borderRadius: 8 }} />
            ) : (
              <div style={{
                height: 300, background: '#f5f5f5', borderRadius: 8,
                display: 'flex', alignItems: 'center',
justifyContent: 'center',
              }}>
                <ToolOutlined style={{ fontSize: 80, color: '#ccc' }} />
              </div>
            )}
          </Col>

          <Col xs={24} md={14}>
            <Title level={2}>{part.name}</Title>

            <div style={{ fontSize: 28, color: '#1677ff', fontWeight: 'bold', marginBottom: 16 }}>

```

```

        {formatPrice(part.price)}
      </div>

      <Descriptions column={1} size="small" bordered>
        <Descriptions.Item
label="Категория">{part.categoryName}</Descriptions.Item>
        {part.oemNumber && (
          <Descriptions.Item label="OEM-
номер">{part.oemNumber}</Descriptions.Item>
        )}
        <Descriptions.Item label="В наличии">
          <Tag color={part.quantityInStock > 0 ? 'green' :
'red'}>
            {part.quantityInStock > 0 ?
`$${part.quantityInStock} шт.` : 'Нет в наличии'}
          </Tag>
        </Descriptions.Item>
      </Descriptions>

      {part.description && (
        <Card style={{ marginTop: 16 }} size="small">
          <p>{part.description}</p>
        </Card>
      )}

      {part.quantityInStock > 0 && (
        <div style={{ marginTop: 24, display: 'flex', gap: 16,
alignItems: 'center' }}>
          <InputNumber
            min={1}
            max={part.quantityInStock}
            value={quantity}
            onChange={setQuantity}
            size="large"
          />
          <Button type="primary" size="large"
icon={<ShoppingCartOutlined />} onClick={handleAdd}>
            В корзину
          </Button>
        </div>
      )}
    </Col>
  </Row>

  {part.compatibleModels && part.compatibleModels.length > 0 &&
(
    <div style={{ marginTop: 32 }}>
      <Title level={4}>Совместимые автомобили</Title>
      <Table
        dataSource={part.compatibleModels}
        columns={compatColumns}
        rowKey="id"
        pagination={false}

```

```

        size="small"
      />
    </div>
  )}
</div>
);
}

```

Листинг А.6 – Страница RegisterPage.jsx

```

const { Title } = Typography;

export default function RegisterPage() {
  const [loading, setLoading] = useState(false);
  const { register } = useAuth();
  const navigate = useNavigate();

  const onFinish = async (values) => {
    setLoading(true);
    try {
      await register(values);
      message.success('Регистрация прошла успешно!');
      navigate('/');
    } catch (err) {
      message.error(err.response?.data?.message || 'Ошибка
регистрации');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div style={{ maxWidth: 450, margin: '60px auto' }}>
      <Card>
        <Title level={3} style={{ textAlign: 'center' }}>Регистрация</Title>
        <Form layout="vertical" onFinish={onFinish}>
          <Form.Item
            name="firstName"
            label="Имя"
            rules={[{ required: true, message: 'Введите имя' }] }
          >
            <Input size="large" />
          </Form.Item>
          <Form.Item
            name="lastName"
            label="Фамилия"
            rules={[{ required: true, message: 'Введите фамилию'
}}}
          >
            <Input size="large" />
          </Form.Item>
          <Form.Item
            name="email"

```

```

        label="Email"
        rules={[
          { required: true, message: 'Введите email' },
          { type: 'email', message: 'Неверный формат email' },
        ]}
      >
        <Input size="large" />
      </Form.Item>
      <Form.Item name="phone" label="Телефон">
        <Input size="large" placeholder="+7-XXX-XXX-XXXX" />
      </Form.Item>
      <Form.Item
        name="password"
        label="Пароль"
        rules={[
          { required: true, message: 'Введите пароль' },
          { min: 6, message: 'Минимум 6 символов' },
        ]}
      >
        <Input.Password size="large" />
      </Form.Item>
      <Form.Item>
        <Button type="primary" htmlType="submit"
loading={loading} block size="large">
          Зарегистрироваться
        </Button>
      </Form.Item>
    </Form>
    <div style={{ textAlign: 'center' }}>
      Уже есть аккаунт? <Link to="/login">Войти</Link>
    </div>
  </Card>
</div>
);
}

```

Листинг А.7 – Страница RegisterPage.jsx

```

export default function CartPage() {
  const { items, totalPrice, updateQuantity, removeFromCart } =
useCart();
  const { isAuthenticated } = useAuth();
  const navigate = useNavigate();

  if (items.length === 0) {
    return (
      <div style={{ textAlign: 'center', padding: 60 }}>
        <Empty description="Корзина пуста">
          <Button type="primary" onClick={() =>
navigate('/catalog')}>
            Перейти в каталог
          </Button>
        </Empty>
      </div>
    );
  }
}

```

```

    );
  }

const columns = [
  {
    title: 'Запчасть',
    key: 'name',
    render: (_, record) => (
      <span
        style={{ cursor: 'pointer', color: '#1677ff' }}
        onClick={() => navigate(`/catalog/${record.part.id}`)}
      >
        {record.part.name}
      </span>
    ),
  },
  {
    title: 'Цена',
    key: 'price',
    render: (_, record) => formatPrice(record.part.price),
    width: 150,
  },
  {
    title: 'Количество',
    key: 'quantity',
    width: 120,
    render: (_, record) => (
      <InputNumber
        min={1}
        value={record.quantity}
        onChange={(val) => updateQuantity(record.part.id, val)}
        size="small"
      />
    ),
  },
  {
    title: 'Сумма',
    key: 'subtotal',
    width: 150,
    render: (_, record) => formatPrice(record.part.price *
record.quantity),
  },
  {
    title: '',
    key: 'actions',
    width: 60,
    render: (_, record) => (
      <Button
        danger
        type="text"
        icon={<DeleteOutlined />}
        onClick={() => removeFromCart(record.part.id)}
      />
    )
  }
];

```

```

    ),
  },
];

return (
  <div>
    <Title level={2}>Корзина</Title>
    <Table
      dataSource={items}
      columns={columns}
      rowKey={(record) => record.part.id}
      pagination={false}
    />
    <Card style={{ marginTop: 24, textAlign: 'right' }}>
      <Title level={3}>Итого: {formatPrice(totalPrice)}</Title>
      <Button
        type="primary"
        size="large"
        icon={<ShoppingOutlined />}
        onClick={() => {
          if (!isAuthenticated) {
            navigate('/login');
          } else {
            navigate('/checkout');
          }
        }}
      />
      Оформить заказ
    </Button>
  </Card>
</div>
);
}

```

Листинг A.8 – Страница CartPage.jsx

```

export default function CartPage() {
  const { items, totalPrice, updateQuantity, removeFromCart } =
    useCart();
  const { isAuthenticated } = useAuth();
  const navigate = useNavigate();

  if (items.length === 0) {
    return (
      <div style={{ textAlign: 'center', padding: 60 }}>
        <Empty description="Корзина пуста">
          <Button type="primary" onClick={() =>
            navigate('/catalog')}>
            Перейти в каталог
          </Button>
        </Empty>
      </div>
    );
  }
}

```

```

const columns = [
  {
    title: 'Запчасть',
    key: 'name',
    render: (_, record) => (
      <span
        style={{ cursor: 'pointer', color: '#1677ff' }}
        onClick={() => navigate(`/catalog/${record.part.id}`)}
      >
        {record.part.name}
      </span>
    ),
  },
  {
    title: 'Цена',
    key: 'price',
    render: (_, record) => formatPrice(record.part.price),
    width: 150,
  },
  {
    title: 'Количество',
    key: 'quantity',
    width: 120,
    render: (_, record) => (
      <InputNumber
        min={1}
        value={record.quantity}
        onChange={(val) => updateQuantity(record.part.id, val)}
        size="small"
      />
    ),
  },
  {
    title: 'Сумма',
    key: 'subtotal',
    width: 150,
    render: (_, record) => formatPrice(record.part.price *
record.quantity),
  },
  {
    title: '',
    key: 'actions',
    width: 60,
    render: (_, record) => (
      <Button
        danger
        type="text"
        icon={<DeleteOutlined />}
        onClick={() => removeFromCart(record.part.id)}
      />
    ),
  },
],

```

```

];

return (
  <div>
    <Title level={2}>Корзина</Title>
    <Table
      dataSource={items}
      columns={columns}
      rowKey={(record) => record.part.id}
      pagination={false}
    />
    <Card style={{ marginTop: 24, textAlign: 'right' }}>
      <Title level={3}>Итого: {formatPrice(totalPrice)}</Title>
      <Button
        type="primary"
        size="large"
        icon={<ShoppingOutlined />}
        onClick={() => {
          if (!isAuthenticated) {
            navigate('/login');
          } else {
            navigate('/checkout');
          }
        }}
      />
      Оформить заказ
    </Card>
  </div>
);
}

```

Листинг А.9 – Страница CheckoutPage.jsx

```

const { Title } = Typography;
const { TextArea } = Input;

export default function CheckoutPage() {
  const { items, totalPrice, clearCart } = useCart();
  const navigate = useNavigate();
  const [loading, setLoading] = useState(false);
  const [success, setSuccess] = useState(false);

  if (items.length === 0 && !success) {
    navigate('/cart');
    return null;
  }

  if (success) {
    return (
      <Result
        status="success"
        title="Заказ успешно оформлен!"
        subTitle="Вы можете отслеживать заказ в разделе «Мои

```

```

заказы»."
      extra={[
        <Button type="primary" key="orders" onClick={() =>
navigate('/orders')}>
          Мои заказы
        </Button>,
        <Button key="catalog" onClick={() =>
navigate('/catalog')}>
          Продолжить покупки
        </Button>,
      ]}
    />
  );
}

const onFinish = async (values) => {
  setLoading(true);
  try {
    await createOrder({
      deliveryAddress: values.deliveryAddress,
      items: items.map((item) => ({
        partId: item.part.id,
        quantity: item.quantity,
      })),
    });
    clearCart();
    setSuccess(true);
  } catch (err) {
    message.error(err.response?.data?.message || 'Не удалось
оформить заказ');
  } finally {
    setLoading(false);
  }
};

const columns = [
  { title: 'Запчасть', dataIndex: ['part', 'name'], key: 'name'
},
  { title: 'Кол-во', dataIndex: 'quantity', key: 'qty', width: 80
},
  {
    title: 'Цена',
    key: 'price',
    width: 120,
    render: (_, record) => formatPrice(record.part.price),
  },
  {
    title: 'Сумма',
    key: 'subtotal',
    width: 120,
    render: (_, record) => formatPrice(record.part.price *
record.quantity),
  },

```

```

];

return (
  <div style={{ maxWidth: 700, margin: '0 auto' }}>
    <Title level={2}>Оформление заказа</Title>

    <Card title="Состав заказа" style={{ marginBottom: 24 }}>
      <Table
        dataSource={items}
        columns={columns}
        rowKey={(r) => r.part.id}
        pagination={false}
        size="small"
      />
      <div style={{ textAlign: 'right', marginTop: 16, fontSize:
18, fontWeight: 'bold' }}>
        Итого: {formatPrice(totalPrice)}
      </div>
    </Card>

    <Card title="Информация о доставке">
      <Form layout="vertical" onFinish={onFinish}>
        <Form.Item
          name="deliveryAddress"
          label="Адрес доставки"
          rules={[{ required: true, message: 'Введите адрес
доставки' }]}
        >
          <TextArea rows={3} placeholder="Город, улица, дом,
квартира..." />
        </Form.Item>
        <Form.Item>
          <Button type="primary" htmlType="submit"
loading={loading} size="large" block>
            Оформить заказ
          </Button>
        </Form.Item>
      </Form>
    </Card>
  </div>
);
}

```

Листинг А.10 – Страница MyOrdersPage.jsx

```

const { Title } = Typography;

const statusColors = {
  NEW: 'blue',
  CONFIRMED: 'cyan',
  SHIPPED: 'orange',
  DELIVERED: 'green',
  CANCELLED: 'red',
};

```

```

const statusLabels = {
  NEW: 'Новый',
  CONFIRMED: 'Подтверждён',
  SHIPPED: 'Отправлен',
  DELIVERED: 'Доставлен',
  CANCELLED: 'Отменён',
};

export default function MyOrdersPage() {
  const [orders, setOrders] = useState([]);
  const [total, setTotal] = useState(0);
  const [page, setPage] = useState(0);
  const [loading, setLoading] = useState(true);
  const navigate = useNavigate();

  useEffect(() => {
    setLoading(true);
    getMyOrders({ page, size: 10 })
      .then((res) => {
        setOrders(res.data.content);
        setTotal(res.data.totalElements);
      })
      .catch(() => {})
      .finally(() => setLoading(false));
  }, [page]);

  const columns = [
    { title: 'Заказ №', dataIndex: 'id', key: 'id' },
    {
      title: 'Дата',
      dataIndex: 'createdAt',
      key: 'date',
      render: (val) => formatDate(val),
    },
    {
      title: 'Позиции',
      key: 'items',
      render: (_, record) => record.items.length,
    },
    {
      title: 'Сумма',
      dataIndex: 'totalPrice',
      key: 'total',
      render: (val) => formatPrice(val),
    },
    {
      title: 'Статус',
      dataIndex: 'status',
      key: 'status',
      render: (val) => <Tag
color={statusColors[val]}>{statusLabels[val] || val}</Tag>,
    },
  ],

```

```

];

if (loading) {
    return <div style={{ textAlign: 'center', padding: 60 }}><Spin
size="large" /></div>;
}

return (
    <div>
        <Title level={2}>Мои заказы</Title>
        {orders.length === 0 ? (
            <Empty description="У вас пока нет заказов" />
        ) : (
            <Table
                dataSource={orders}
                columns={columns}
                rowKey="id"
                onRow={ (record) => ({
                    onClick: () => navigate(`/orders/${record.id}`),
                    style: { cursor: 'pointer' },
                })}
                pagination={{
                    current: page + 1,
                    total,
                    pageSize: 10,
                    onChange: (p) => setPage(p - 1),
                }}
            />
        )}
    </div>
);
}

```

Листинг A.11 – Страница CarPartsApplication.java

```

@SpringBootApplication
public class CarPartsApplication {

    public static void main(String[] args) {
        SpringApplication.run(CarPartsApplication.class, args);
    }

}

```

Листинг A.12 – Страница AdminOrderController.java

```

@RestController
@RequestMapping("/api/admin/orders")
@RequiredArgsConstructor
public class AdminOrderController {

    private final OrderService orderService;

    @GetMapping
    public ResponseEntity<Page<OrderResponse>> getAllOrders(

```

```

        @RequestParam(required = false) OrderStatus status,
        @PageableDefault(size = 20) Pageable pageable) {
    if (status != null) {
        return
        ResponseEntity.ok(orderService.getOrdersByStatus(status,
        pageable));
    }
    return
    ResponseEntity.ok(orderService.getAllOrders(pageable));
}

@GetMapping("/{id}")
public ResponseEntity<OrderResponse> getOrderById(@PathVariable
Long id) {
    return
    ResponseEntity.ok(orderService.getOrderByIdAdmin(id));
}

@PutMapping("/{id}/status")
public ResponseEntity<OrderResponse> updateStatus(
    @PathVariable Long id,
    @RequestBody Map<String, String> body) {
    OrderStatus status =
    OrderStatus.valueOf(body.get("status"));
    return ResponseEntity.ok(orderService.updateStatus(id,
    status));
}
}

```

Листинг A.13 – OrderService.java

```

@Service
@RequiredArgsConstructor
public class OrderService {

    private final OrderRepository orderRepository;
    private final PartRepository partRepository;

    @Transactional
    public OrderResponse createOrder(User user, OrderCreateRequest
    request) {
        Order order = Order.builder()
            .user(user)
            .deliveryAddress(request.getDeliveryAddress())
            .totalPrice(BigDecimal.ZERO)
            .build();

        BigDecimal totalPrice = BigDecimal.ZERO;

        for (OrderItemRequest itemRequest : request.getItems()) {
            Part part =
            partRepository.findById(itemRequest.getPartId())
                .orElseThrow(() -> new
            ResourceNotFoundException("Part not found: " +

```

```

itemRequest.getPartId())));

        if (part.getQuantityInStock() <
itemRequest.getQuantity()) {
            throw new InsufficientStockException(
                "Insufficient stock for part '" +
part.getName() +
                "'. Available: " +
part.getQuantityInStock() +
                ", requested: " +
itemRequest.getQuantity());
        }

        part.setQuantityInStock(part.getQuantityInStock() -
itemRequest.getQuantity());
        partRepository.save(part);

        OrderItem orderItem = OrderItem.builder()
            .order(order)
            .part(part)
            .quantity(itemRequest.getQuantity())
            .priceAtOrder(part.getPrice())
            .build();

        order.getItems().add(orderItem);
        totalPrice =
totalPrice.add(part.getPrice().multiply(BigDecimal.valueOf(itemRequ
est.getQuantity())));
    }

    order.setTotalPrice(totalPrice);
    Order saved = orderRepository.save(order);

    return toResponse(saved);
}

public Page<OrderResponse> getMyOrders(Long userId, Pageable
pageable) {
    return
orderRepository.findByUserIdOrderByCreatedAtDesc(userId, pageable)
        .map(this::toResponse);
}

public OrderResponse getOrderById(Long orderId, Long userId) {
    Order order = orderRepository.findById(orderId)
        .orElseThrow(() -> new
ResourceNotFoundException("Order not found: " + orderId));

    if (!order.getUser().getId().equals(userId)) {
        throw new ResourceNotFoundException("Order not found: "
+ orderId);
    }
}

```

```

        return toResponse(order);
    }

    public Page<OrderResponse> getAllOrders(Pageable pageable) {
        return
orderRepository.findAllByOrderByCreatedAtDesc(pageable)
                .map(this::toResponse);
    }

    public Page<OrderResponse> getOrdersByStatus(OrderStatus
status, Pageable pageable) {
        return
orderRepository.findByStatusOrderByCreatedAtDesc(status, pageable)
                .map(this::toResponse);
    }

    public OrderResponse getOrderByIdAdmin(Long orderId) {
        Order order = orderRepository.findById(orderId)
                .orElseThrow(() -> new
ResourceNotFoundException("Order not found: " + orderId));
        return toResponse(order);
    }

    @Transactional
    public OrderResponse updateStatus(Long orderId, OrderStatus
status) {
        Order order = orderRepository.findById(orderId)
                .orElseThrow(() -> new
ResourceNotFoundException("Order not found: " + orderId));
        order.setStatus(status);
        return toResponse(orderRepository.save(order));
    }

    private OrderResponse toResponse(Order order) {
        return OrderResponse.builder()
                .id(order.getId())
                .status(order.getStatus())
                .totalPrice(order.getTotalPrice())
                .deliveryAddress(order.getDeliveryAddress())
                .createdAt(order.getCreatedAt())
                .updatedAt(order.getUpdatedAt())
                .userId(order.getUser().getId())
                .userEmail(order.getUser().getEmail())
                .userName(order.getUser().getFirstName() + " " +
order.getUser().getLastName())

.items(order.getItems().stream().map(this::toItemResponse).toList()
)

                .build();
    }

    private OrderItemResponse toItemResponse(OrderItem item) {
        return OrderItemResponse.builder()

```

```

        .id(item.getId())
        .partId(item.getPart().getId())
        .partName(item.getPart().getName())
        .partImageUrl(item.getPart().getImageUrl())
        .quantity(item.getQuantity())
        .priceAtOrder(item.getPriceAtOrder())
        .build();
    }
}

```

Листинг A.14 – OrderRepository.java

```

public interface OrderRepository extends JpaRepository<Order, Long>
{
    Page<Order> findByUserIdOrderByCreatedAtDesc(Long userId,
    Pageable pageable);

    Page<Order> findAllByOrderByCreatedAtDesc(Pageable pageable);

    Page<Order> findByStatusOrderByCreatedAtDesc(OrderStatus
    status, Pageable pageable);

    long countByStatus(OrderStatus status);

    @Query("SELECT COALESCE(SUM(o.totalPrice), 0) FROM Order o")
    BigDecimal getTotalRevenue();

    @Query(value = "SELECT TO_CHAR(created_at, 'YYYY-MM-DD'),
COUNT(*), COALESCE(SUM(total_price), 0) " +
        "FROM orders WHERE created_at >= :since " +
        "GROUP BY TO_CHAR(created_at, 'YYYY-MM-DD') " +
        "ORDER BY TO_CHAR(created_at, 'YYYY-MM-DD')",
nativeQuery = true)
    List<Object[]> findDailyStats(@Param("since") LocalDateTime
    since);
}

```

Приложение Б

(рекомендуемое)

Скриншоты работы приложения

Регистрация

* Имя

* Фамилия

* Email

Телефон

* Пароль

Зарегистрироваться

Уже есть аккаунт? [Войти](#)

Рисунок Б.1 – Форма регистрации пользователя

Вход

 Email

 Пароль

Войти

Нет аккаунта? [Зарегистрируйтесь](#)

Рисунок Б.2 – Форма входа пользователя

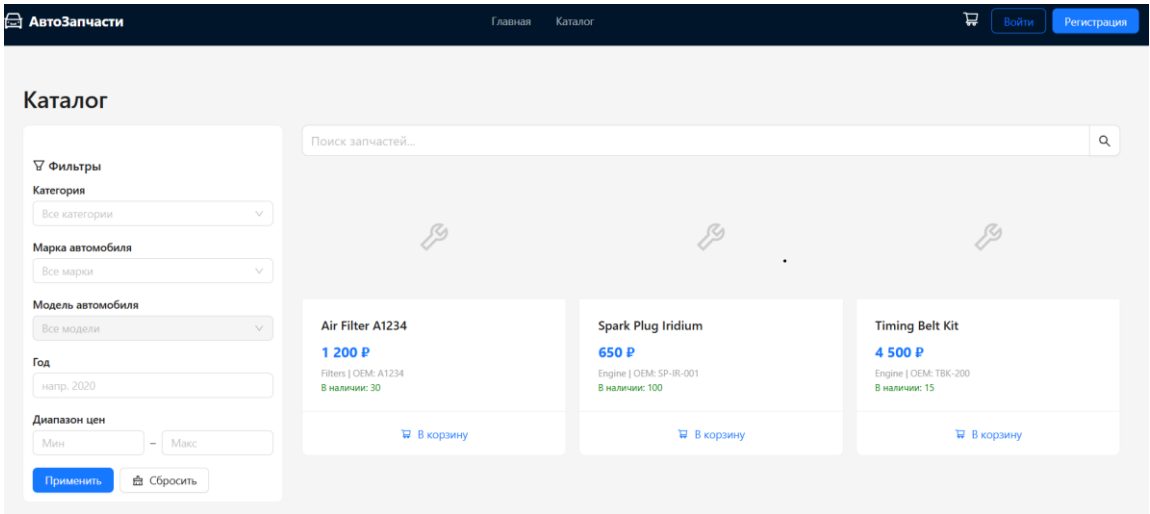


Рисунок Б.3 – Каталог фирмы

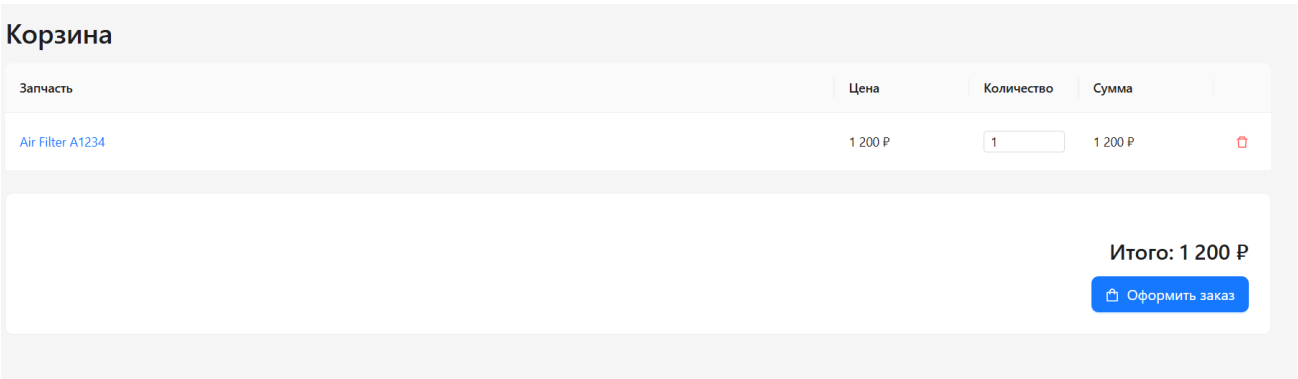


Рисунок Б.4 – Форма оформления заказа

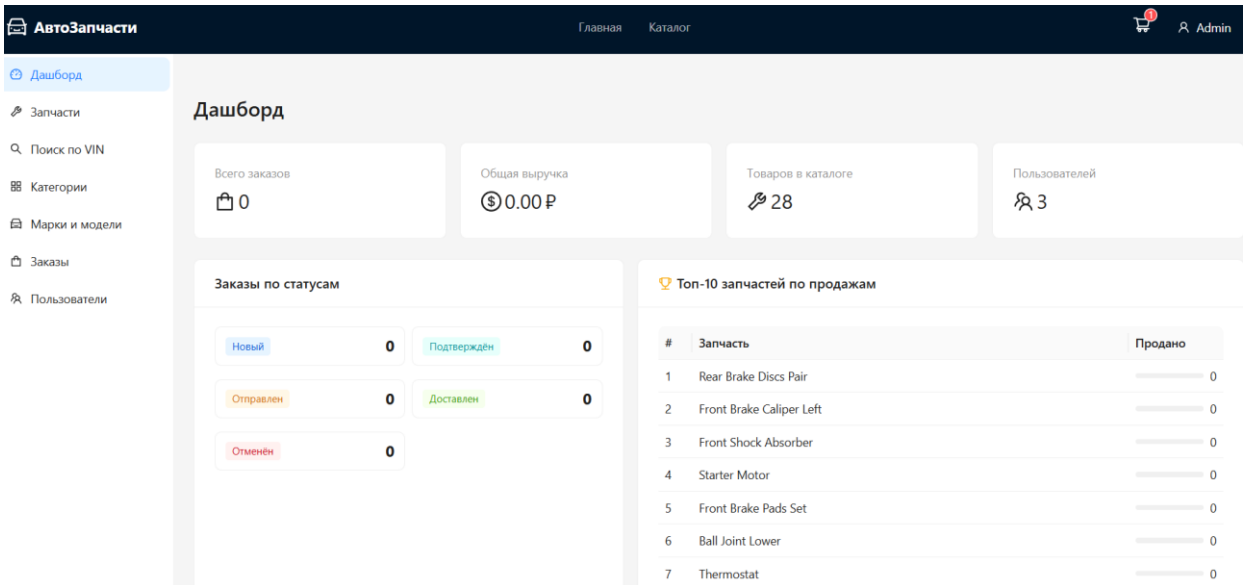


Рисунок Б.5 – Личный кабинет менеджера