

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Web-приложение для управления дронами-
доставщиками**

Студент _____ Костюшин Александр Романович
(подпись, дата) (ФИО полностью)

Руководитель _____ Шашков Б.Д.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

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

Пенза, 2023

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
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Костюшин Александр Романович гр. 19ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Web-приложение для управления дронами-доставщиками

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., профессор Шашков Борис Дмитриевич

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

1) Назначение: программа предназначена для управления дронами-доставщиками

2) Средства разработки: язык программирования Java, JavaScript, интегрированная среда разработки (IDE) IntelliJ IDEA

3) Основные функции программы:

а) хранение профилей ботов с информацией о модели, характеристиках, состоянии, статусе;

б) получение и регистрация заказов;

в) выбор подходящего по характеристикам и состоянию робота для исполнения заказа;

г) получение сигналов и запуск протоколов реагирования на них;

д) расчет стоимости доставки заказа;

е) возможность анализа заказов, автоматическое подтверждение или отказ.

5. Перечень подлежащих разработке вопросов

Анализ предметной области и постановка задачи

Анализ требований на разработку

Проектирование приложения
Реализация приложения
Тестирование и контроль качества приложения
Планирование разработки и оценка бюджета приложения
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	5%	01.04.2023 – 08.04.2023	
2	Анализ требований на разработку	5%	08.04.2023 – 17.04.2023	
3	Проектирование приложения	20%	17.04.2023 – 26.04.2023	
4	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
5	Тестирование и контроль качества приложения	10%	12.05.2023 – 19.05.2023	
6	Планирование разработки и оценки бюджета	10%	19.05.2023 – 2.05.2023	
7	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022г.

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 67 листов, 20 таблиц, 30 рисунков, 16 использованных источников и 4 приложения.

WEB – ПРИЛОЖЕНИЕ, УПРАВЛЕНИЕ ДРОНАМИ ДОСТАВЩИКАМИ,
ООП, JAVA, INTELLIJ IDEA, UML

Объект разработки – Web-приложение для управления дронами доставщиками.

Цель работы – провести анализ предметной области и поставить задачу на разработку, провести сравнительный анализ конкурирующего программного обеспечения, определиться с инструментальными средствами, спроектировать диаграммы UML и пользовательский интерфейс, разработать программное обеспечение, провести его тестирование и анализ кода с помощью различных метрик, спланировать бюджет разработки.

Средства разработки: язык программирования Java, JavaScript, интегрированная среда разработки IntelliJ IDEA.

Полученные результаты: в ходе выполнения выпускной квалификационной работы была спроектирована программа, которая позволяет управлять дронами доставщиками.

					ПГУ 09.03.04 – 08БР191.10 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Web-приложение для управления дронами доставщиками» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Костюшин А.Р.					4	67
Провер.		Шашков Б.Д.						
Н. Контр.		Такташкин Д.В.				Группа 19ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	6
1 Постановка задачи и анализ требований	7
1.1 Постановка задачи.....	7
1.2 Анализ программного обеспечения аналогичного назначения.....	8
1.2.1 Анализ приложения Яндекс Доставка	8
1.2.2 Анализ приложения СберЛогистика	9
1.2.3 Сравнение аналогов	10
1.3 Анализ функциональных требований	11
1.4 Анализ требований к пользовательскому интерфейсу	17
2 Разработка информационного и программного приложения.....	18
2.1 Концептуальная модель предметной области.....	18
2.2 Схема базы данных	19
2.3 Разработка серверного программного обеспечения.....	21
2.4 Разработка пользовательского программного обеспечения.....	25
3 Тестирование программного обеспечения	31
3.1 Обоснование выборов режимов тестирования	31
3.2 Интеграционное тестирование серверного программного обеспечения	32
3.3 Функциональное тестирование клиентского программного обеспечения ...	33
3.4 Метрики кода.....	34
3.4.1 Метрика SLOC.....	34
3.4.2 Метрики Холстеда.....	35
3.4.3 Метрики устойчивости и абстрактности компонент.....	36
3.4.4 Комплексный показатель качества кода	38
4 Планирование и оценка бюджета проекта.....	39
Заключение	43
Список использованных источников	44
Приложение А. Описание таблиц базы данных.....	46
Приложение Б. Серверное тестирование	51
Приложение В. Тестирование клиентского ПО	56
Приложение Г. Код программы.....	61

Введение

Массовое и активное внедрение роботов в нашу жизнь и бизнес началось в 21 веке. Сегодня они довольно широко используются в системах искусственного интеллекта, навигаторах, автопилотах, дронах, логистике. Для многих известных компаний применение роботов стало мощным скачком в развитии и производства.

Роботизация снижает себестоимость доставки, ведь роботу не надо платить, также она повышает скорость и безопасность доставки, а это рост уровня сервиса. Дроны, которые в большей степени являются сложными роботами, хороши и в решении сложных задач – дроны ищут повреждения газопроводов без риска угрозы жизням сотрудников, доставляют лекарства в горные районы и т. д.

Так же существенный интерес к технологиям роботизация подогрела пандемия коронавируса COVID-19, вызвав бум спроса на услуги онлайн доставки еды и прочих покупок, стимулируя «бесконтактную экономику». Интерес инвесторов и конструкторов к роботам-доставщикам существенно возрос.

По данным одной из статистических компаний, в 2019 году рынок автономных роботов составлял около 20 млн долларов. Ожидается, что к 2026 году он достигнет 250 млн долларов. Из вышеперечисленных фактов можно сделать вывод о том, что система дронов доставщиков является перспективным направлением на данный момент времени [1].

Задачами данного проекта являются:

- анализ требований на разработку системы;
- разработка информационного и программного приложения;
- тестирование программного обеспечения.

1 Постановка задачи и анализ требований

1.1 Постановка задачи

Целью данного проекта является разработка и создание сервиса онлайн-заказа ботов-доставщиков через веб-приложение.

Веб-приложение может использоваться как администратором, так и обычным пользователем. Администратор контролирует работу роботов и следит за их состоянием, получает сигналы, а также контролирует выполнение заказов. Пользователь может регистрироваться и авторизовываться в системе, а также создавать заказ и получать более подробную информацию о статусе его выполнения.

Сервис передает заказ боту, который подбирается на основе различных параметров, таких как вес, хрупкость, объем груза, а также расстояние, погодные условия и т. д. и предусматривает неблагоприятные ситуации, которые могут произойти с ним в дороге.

Для разработки веб-приложения необходимо разбить его на 2 основные части, а именно серверную часть и клиентскую.

Для достижения поставленных задач были выделены следующие цели:

- разработка базы данных;
- разработка серверного программного обеспечения;
- разработка клиентского программного обеспечения.

Для достижения цели были использованы следующие инструменты:

- postgresSQL для создания базы данных;
- язык программирования Java для реализации серверного программного обеспечения;
- javascript для создания клиентского программного обеспечения;
- intelliJ IDEA – среда разработки серверного программного обеспечения.

1.2 Анализ программного обеспечения аналогичного назначения

В русскоязычном сегменте сети Интернет существует множество аналогов веб-приложений, которые организуют доставку различных товаров.

1.2.1 Анализ приложения Яндекс Доставка

Яндекс Доставка – логистический сервис для компаний и физических лиц, организующий доставку посылок, которая осуществляется как в черте города, так и между городами [2]. Доставка осуществляется партнёрскими курьерскими службами и самозанятыми курьерами.

Преимущества:

- кроссплатформенность;
- различные виды доставки;
- служба поддержки.

Недостатки:

- доставка осуществляется людьми;
- большие финансовые расходы на доставку;
- неудобный интерфейс приложения.

На рисунке 1 представлен интерфейс данного программного обеспечения.

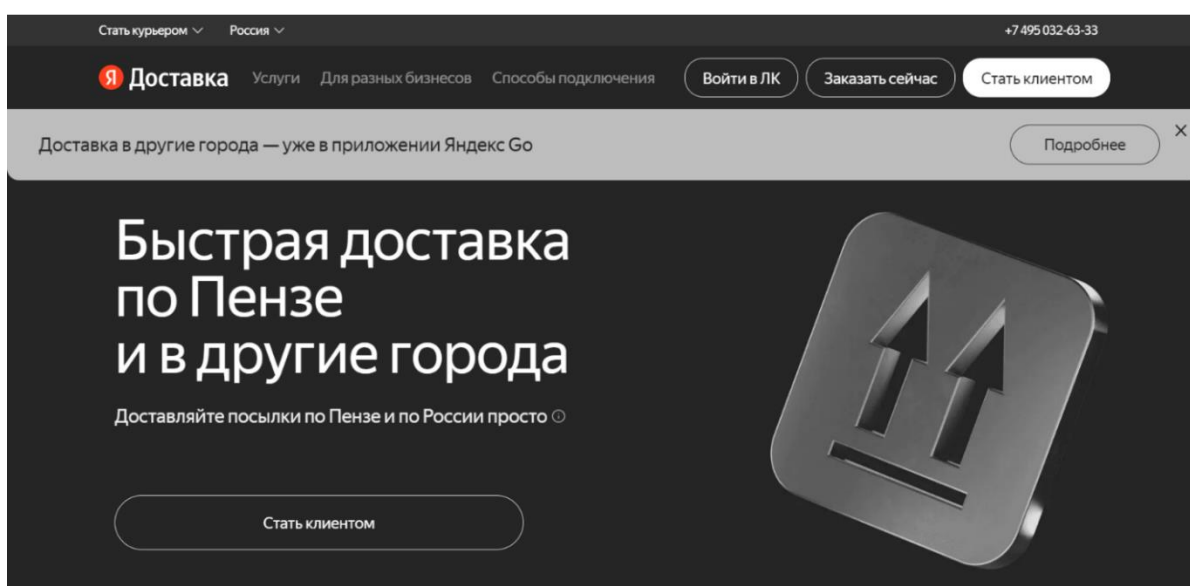


Рисунок 1 – Интерфейс веб-приложения Яндекс Доставка

Главным достоинством данного приложения является наличие разнообразных видов доставок, таких как: «Экспресс – доставка», «По городу», «По России». Кроме того, доставка осуществляется как на дом, так и в отделения пунктов выдачи.

Недостатком приложения является его неудобный интерфейс, что приводит к усложнению его использования. Кроме того, в данном приложении человек выступает в качестве курьера, что приводит к дополнительным расходам.

1.2.2 Анализ приложения СберЛогистика

СберЛогистика – сервис, предоставляющий полный комплекс логистических услуг, включая доставку через пункты приема и выдачи посылок, курьерскую доставку и грузоперевозки [3].

Преимущества:

- кроссплатформенность;
- удобный интерфейс;
- круглосуточная поддержка.

Недостатки:

- доставка осуществляется людьми;
- большие финансовые расходы на доставку.

На рисунке 2 представлен интерфейс ПО.

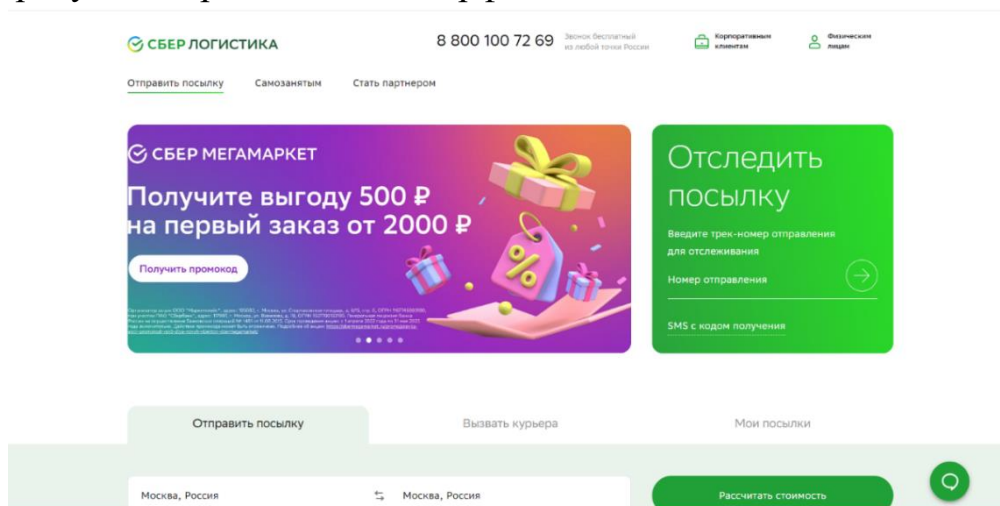


Рисунок 2 – Интерфейс веб-приложения СберЛогистика

К достоинствам данного приложения относятся кроссплатформенность, удобный и приятный интерфейс, что позволяет упростить работу в данном приложении. Также стоит отметить, что в данном приложении имеются различные виды доставки:

- «СберКурьер» (курьерская доставка);
- «СберПосылка» (доставка отправлений от пункта приема до пункта выдачи).

Главным недостатком, как и в предыдущем аналоге, является человеческий фактор, который заключается в том, что именно человек осуществляет доставку товара.

1.2.3 Сравнение аналогов

Для представленных выше аналогов был проведен сравнительный анализ. Результаты приведены в таблице 1.

Таблица 1 – Сводная сравнительная таблица критериев оценки продуктов-аналогов

Веб-приложение	Удобство интерфейса	Кроссплатформенность	Различные виды доставки	Радиус доставки	Курьер	Поддержка
Яндекс Доставка	Неудобный	Да	Да	Доставка по городу и по России	Человек	Есть
СберЛогистика	Удобный	Да	Нет	Доставка по городу и по России	Человек	Есть
Разрабатываемое приложение	Удобный	Да	Да	Доставка по городу	Робот(бот)	Есть

В результате сравнительного анализа можно сделать вывод о том, что в отличие от разрабатываемого приложения аналоги используют в качестве курьеров людей, что создает дополнительные расходы. К таким расходам

относятся: заработная плата для курьера, затраты на обслуживание автомобиля и бензина и т. д.

В будущем в разрабатываемом приложении возможно расширение радиуса доставки, которое связано непосредственно с развитием его инфраструктуры, увеличением количества парков, а также увеличением количества ботов и их работоспособностью.

Создаваемое приложение предназначено для минимизации расходов и участия человека в процессе доставки, что позволит впоследствии снизить цену доставки и получить максимальную прибыль. Кроме того, вышеперечисленные приложения осуществляют доставку товаров непосредственно физическим и юридическим лицам, а разрабатываемое приложение может использоваться как для доставки товаров простым гражданам, так и в военной сфере.

1.3 Анализ функциональных требований

Функциональные требования:

- хранение профилей ботов с информацией о модели, характеристиках, состоянии, и его статусе;
- получение и регистрация заказов, выбор подходящего по характеристикам и состоянию робота для исполнения определенного заказа (по габаритам, возможностям перевозить хрупкий/тяжелый груз, холодное или горячее);
- мониторинг роботов, получение сигналов и запуск протоколов реагирования на них.
- расчет стоимости доставки выполняемого заказа, учитывать расстояние П-А, А-Б, Б-П, где П – парк, А – место хранения посылки, Б – точка доставки заказа, помимо этого, стоит учесть необходимые характеристики бота: модель, вес, а также погодные условия и загруженность парка;
- система должна предусматривать возможность анализа заказов, автоматическое подтверждение заказа или отказ от него;

- расчёт возможных вариантов оптимизации парка, расчет в соответствии с загруженностью парка, количеством ботов, со стоимостью обслуживания ботов, с содержанием и прибылью с заказов и т. п.;

- прибыль рассчитывается как $D - (Э + O)$, где D – доход от доставки заказов, $Э$ – расходы на электроэнергию, затраченную на зарядку бота, O – расходы на обслуживание бота;

- удобный интерфейс администратора для отслеживания всех важных показателей ботов, для получения уведомлений, а также для принятия необходимых решений.

Существуют следующие сигналы: «Застрял», «Сигнал потерян», «Низкий уровень заряда батареи», «Повреждение», «Отклонение от маршрута». Они должны отправлять информацию о местоположении (координаты) бота, который отправил данный сигнал. По сигналам «Начал заказ», «Завершил заказ», «Вернулся в парк» необходимо фиксировать время работы бота, количество совершенных «сессий», запускать протоколы техобслуживания, зарядки и т. п.;

Дополнительные системы:

- генератор случайных событий, к примеру: застрял, разрядился, сбила машина, украли, произошел сбой в системе;

- генератор заказов: расстояние от места хранения посылки до точки доставки заказа, а также вес посылки и ее характеристики;

- интеграция с картой: работа с реальными координатами по г. Пензе, реальный расчет расстояний.

На основе составленных функциональных требований для разрабатываемого приложения была построена диаграмма вариантов использования, которая изображена на рисунке 3.

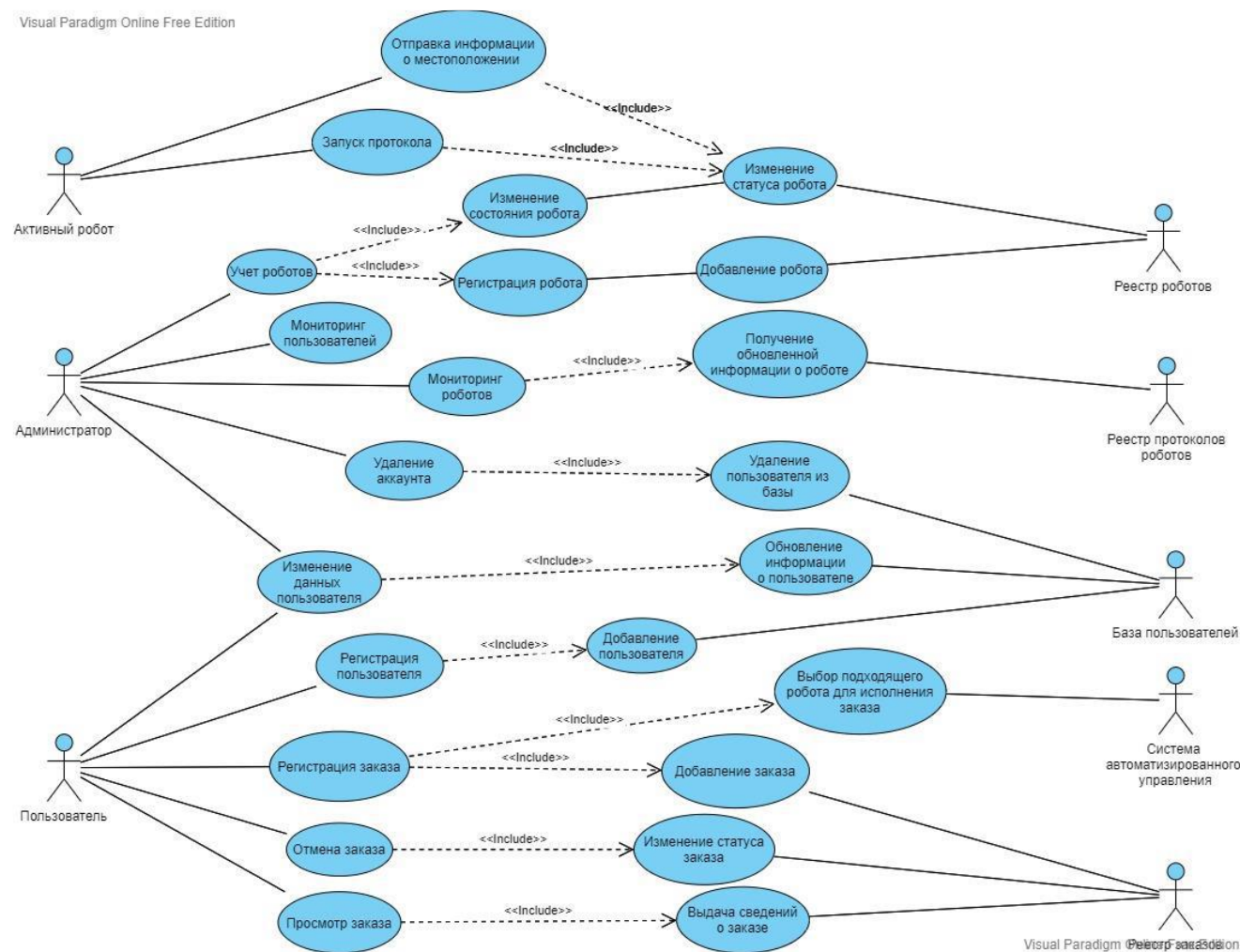


Рисунок 3 – Диаграмма вариантов использования

В таблице 2 приведена спецификация варианта использования «Регистрация робота».

Таблица 2 – Спецификация прецедента «Регистрация робота»

Прецедент: Регистрация робота
ID:1
Краткое описание: Занесение информации о новом роботе в реестр роботов.
Главные актеры: Администратор.
Второстепенные актеры: Реестр роботов
Предусловия: 1. Администратор вошел в систему.
Основной поток: 1. Если робот содержит информацию о том, что он принадлежит несуществующему парку или относится к несуществующей модели то 1.1 Система сообщает, что данного парка или модели робота не существует. 2. Иначе 2.1 Система добавляет нового робота в реестр. 2.2 Система уведомляет администратора об успешном добавлении робота.
Постусловия: 1. Робот добавлен в реестр роботов
Альтернативные потоки: Нет

В таблице 3 приведена спецификация варианта использования «Мониторинг роботов».

Таблица 3 – Спецификация прецедента «Мониторинг роботов»

Прецедент: Мониторинг роботов
ID:2
Краткое описание: Получение информации о состоянии и местоположении робота
Главные актеры: Администратор.
Второстепенные актеры: Реестр протоколов роботов, робот
Предусловия: 1. Администратор вошел в систему.
Основной поток: 1. Если запрошенный робот не существует 1.1. То система сообщает, что данного робота не существует. 2. Иначе 2.1. Выбор определенного робота для получения информации. 2.2. Вывод информации на экран 3. Реестр протоколов роботов запрашивает данные по таймеру и динамически обновляет информацию.
Постусловия: 1. Информация доступна администратору
Альтернативные потоки: Нет.

В таблице 4 приведена спецификация варианта использования «Регистрация заказа».

Таблица 4 – Спецификация прецедента «Регистрация заказа»

Прецедент: Регистрация заказа
ID:3
Краткое описание: Пользователь создает заказ
Главные актеры: Пользователь
Второстепенные актеры: Реестр заказов, система автоматизированного управления
Предусловия: 1.Пользователь авторизован
Основной поток: 1. Пользователь вводит данные нового заказа. 2. Система автоматизированного управления подбирает подходящего робота для исполнения заказа. 2.1. Иначе информирует пользователя о том, что в данный момент все роботы заняты 3. Заказ добавлен в реестр заказов
Постусловия: 1. Заказ зарегистрирован
Альтернативные потоки: Нет.

Для реализации приложения за основу были взяты представленные варианты использования и актеры.

В результате анализа функциональных требований были успешно выявлены основные функции разрабатываемого программного продукта. Для

разрабатываемого приложения была построена диаграмма вариантов использования, а также для некоторых прецедентов были составлены спецификации.

1.4 Анализ требований к пользовательскому интерфейсу

Веб приложение разрабатывается с целью взаимодействия пользователя/администратора с роботами(ботами), осуществляющими доставку.

Администратор отслеживает все важные показатели, получает уведомления, принимает решения относительно роботов и их состояния, осуществляет за ними контроль.

Система должна предусматривать получение и регистрацию заказов. Выбор подходящего по характеристикам и состоянию робота для исполнения заказа (по габаритам, возможностям перевозить хрупкий/тяжелый груз, холодное или горячее).

Администратор может выполнять следующие действия:

- мониторинг роботов;
- получение сигналов, запуск протоколов реагирования на них.

Сигналы «Застрял», «Сигнал потерян», «Низкий уровень заряда батареи», «Повреждение», «Отклонение от маршрута» должны отправлять информацию о местоположении (координаты). По сигналам: «Начал заказ», «Завершил заказ», «Вернулся в парк» нужно фиксировать время работы бота, количество совершенных «сессий», запускать протоколы техобслуживания, зарядки и т. п.

Пользователь может выполнять следующие действия:

- регистрация заказа;
- получение информации о заказе.

Помимо вышеизложенных действий система должна предоставлять возможность анализа заказов, расчет стоимости, автоматическое подтверждение или отказ.

2 Разработка информационного и программного приложения

Современная разработка – это командная и кросс-функциональная деятельность высокой сложности. Для корректного и эффективного взаимодействия всех участников процесса используется та или иная модель, различные инструменты и методологии разработки.

Данный раздел представляет информацию об этапах построения приложения и методах этих построений.

2.1 Концептуальная модель предметной области

Концептуальная модель предметной области представляет собой абстрактное представление предметной области, которое помогает понять ее основные концепции, сущности и их взаимосвязи. Она служит основой для проектирования и разработки.

Концептуальная модель описывает сущности в предметной области и их атрибуты, а также связи между этими сущностями.

Прежде чем проектировать и создавать базу данных, необходимо построить концептуальную модель предметной области. Она представлена на рисунке 4 в виде упрощенной диаграммы сущностей баз данных.

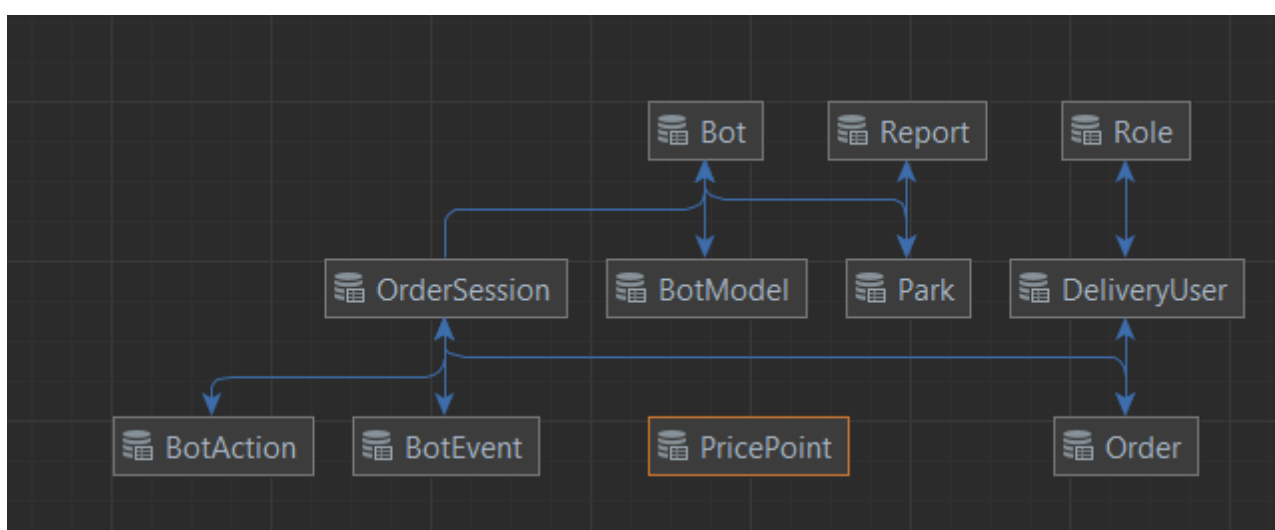


Рисунок 4 – Упрощенная диаграмма сущностей

Данная диаграмма описывает основные сущности, применяемые в приложении, а также связи между ними.

2.2 Схема базы данных

Схема базы данных представляет собой структуру данных, которая определяет организацию и связи между данными в базе данных. Она является важным компонентом проектирования базы данных и служит основой для создания, хранения и управления данными.

Построение схемы базы данных является первым шагом в создании приложения. При создании схемы базы данных были применены принципы нормализации, которые помогают организовать данные таким образом, чтобы минимизировать избыточность информации, обеспечить целостность данных и улучшить эффективность запросов. Использовались следующие принципы нормализации:

- все атрибуты в таблице должны быть неделимы и атомарны, т. е. каждый атрибут содержит только одно значение;
- все данные строго простые (одно поле, одно свойство);
- нет повторяющихся строк (достигаются с помощью создания первичных ключей);
- каждый неключевой атрибут должен зависеть от всего первичного ключа;
- множественные многозначные зависимости должны быть удалены;
- неключевые атрибуты не должны зависеть от других неключевых атрибутов, все зависимости должны быть функционально связаны с первичным ключом.

На рисунке 5 показана схема базы данных.

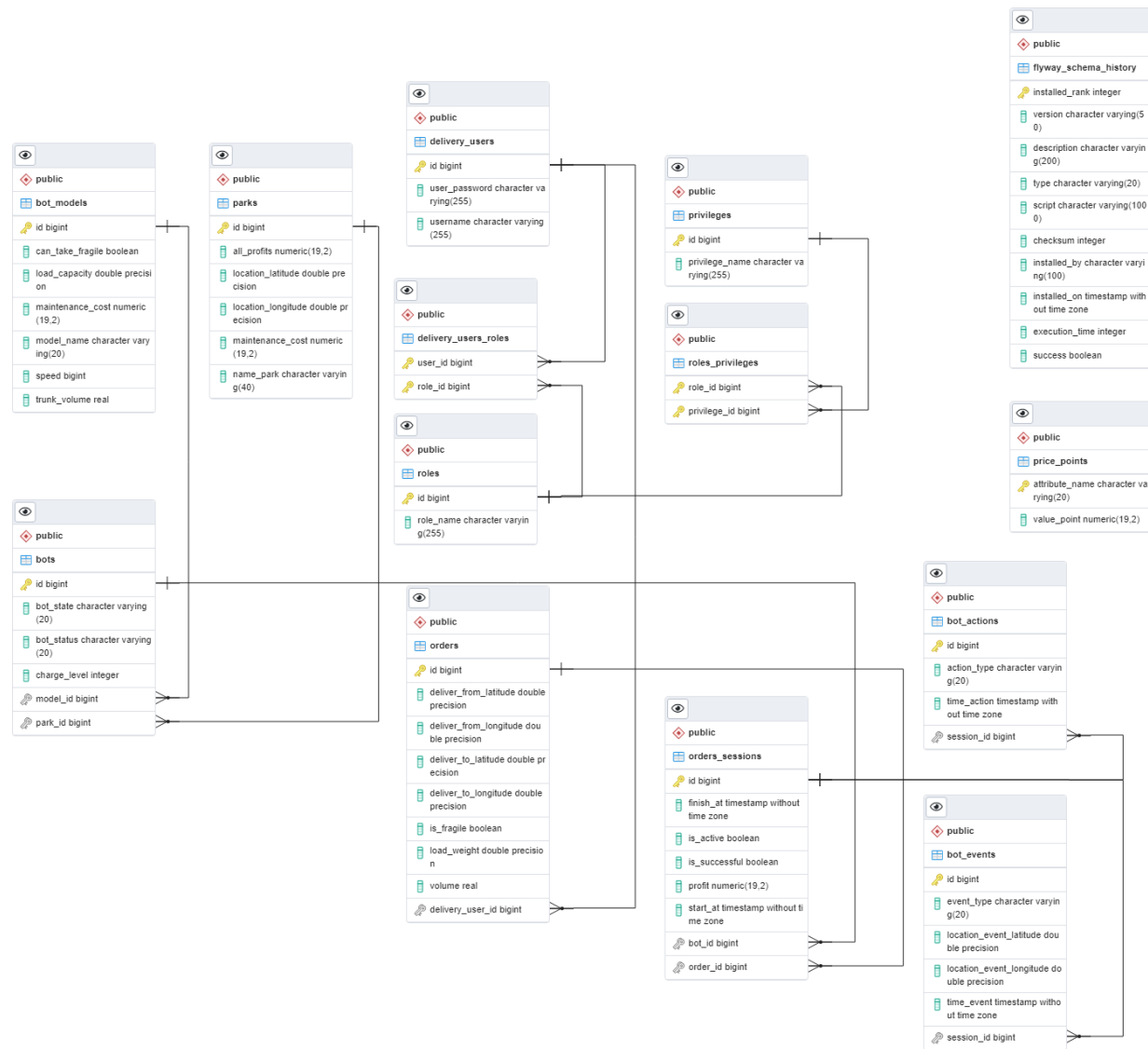


Рисунок 5 – Схема базы данных

Подробное описание каждой сущности базы данных представлено в приложении А.

2.3 Разработка серверного программного обеспечения

Серверное программное обеспечение в информационных технологиях – программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам [2].

На рисунке 6 представлена подробная схема архитектуры реализуемого приложения, а на рисунке 7 – схема взаимодействия слоев приложения по сущности Bot. Схема содержит структуру сервисных слоев приложения в серверной и клиентской частях, а также структуру сервисов удаленного взаимодействия.

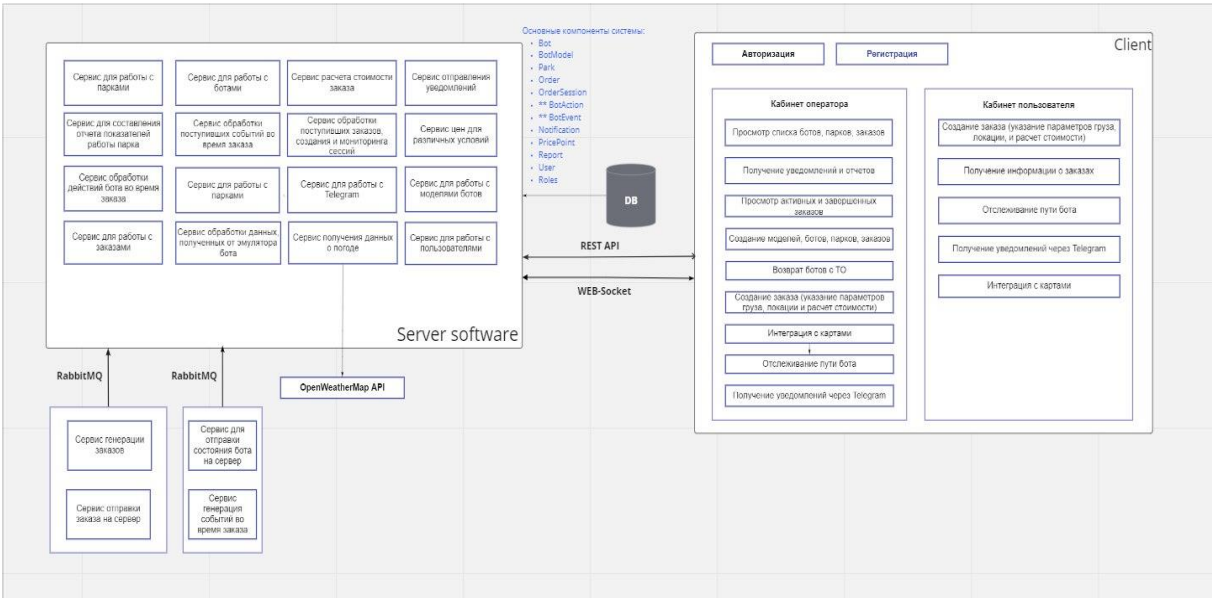


Рисунок 6 – Схема архитектуры реализуемого приложения

В качестве примера логики портов и адаптеров представлены слои, которые предназначены для обработки сущности бота.

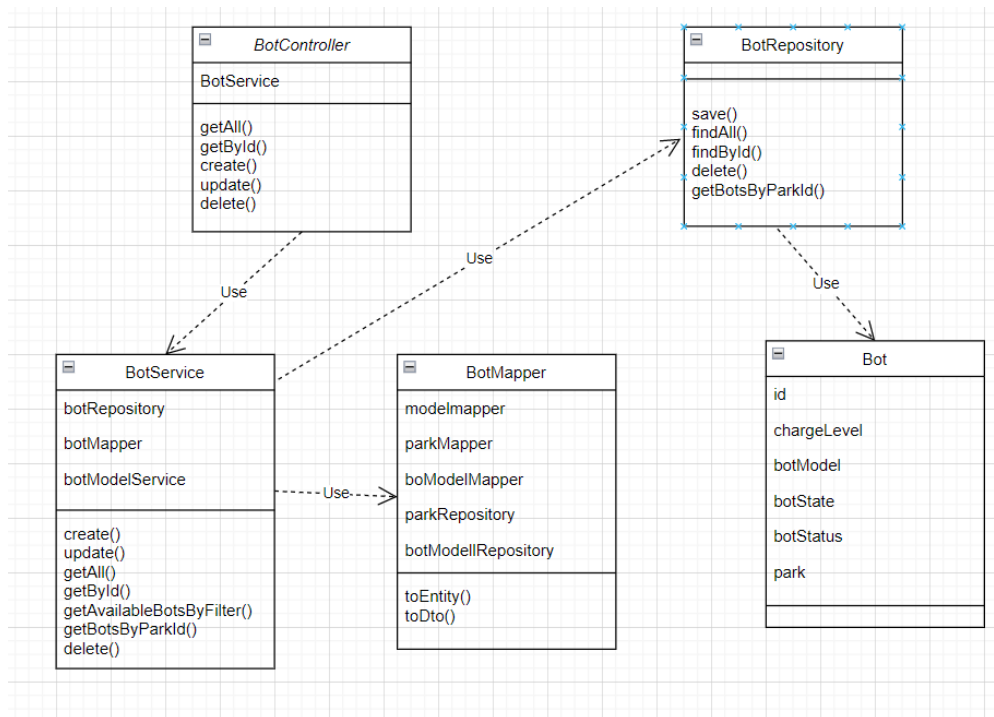


Рисунок 7 – Схема взаимодействия

В таблице 5 приведен класс контроллер BotController.

Таблица 5 – Контроллер для ботов (Rest слой)

BotController
BotService – класс бизнес-логика
<p>Методы:</p> <p>getAll() – получить сущность в виде Json</p> <p>getById() – получить сущность по id Json</p> <p>create() – создание сущности в виде Json;</p> <p>update() – отправка данных для обновления в виде Json;</p> <p>delete() – запрос на удаление сущности из базы данных;</p>

Класс контроллер, отвечает за обработку входящих HTTP запросов и перенаправление их в нужный сервис с бизнес логикой.

В таблице 6 приведен класс – сервис BotService.

Таблица 6 – Класс-сервис для бота(бизнес-логика)

BotService
<p>Поля:</p> <p>botRepository – слой для сохранения данных в БД;</p> <p>botMapper – класс для конвертации данных;</p> <p>botModelService – класс сервиса модели бота;</p>
<p>Методы:</p> <p>getAll() – получить сущность в виде Json;</p> <p>getId() – получить сущность по id Json;</p> <p>create() – создание сущности в виде Json;</p> <p>update() – отправка данных для обновления в виде Json;</p> <p>getAvailableBotsByFilter() – получить доступные сущности бота;</p> <p>getBotsByParkId() -получить сущности бота по id парка;</p> <p>delete() – запрос на удаление сущности из базы данных;</p>

BotService – сервис класс, в таких классах содержится основная логика работы приложений.

В таблице 7 приведен класс – репозиторий BotRepository.

Таблица 7 – Класс репозитория для бота(бизнес-логика)

BotRepository (Repository слой)
<p>Методы:</p> <p>findAll() – найти сущность в базе данных;</p> <p>findById() – найти сущность в базе данных по id;</p> <p>save() – запрос на сохранение сущности в базе данных;</p> <p>getBotsByParkId() -получить сущности бота по id парка;</p> <p>delete() – запрос на удаление сущности из базы данных;</p>

Класс репозиторий – осуществляет доступ к базе данных, предоставляет несколько абстрактных методов из «коробки».

В таблице 8 приведен класс – маппер для конвертации сущности бота.

Таблица 8 – Класс – маппер для бота

BotMapper
<p>Поля:</p> <p>modelmapper – общий класс конфигурации Mapper;</p> <p>parkMapper – класс конвертации сущности парка;</p> <p>botModelMapper – класс конвертации сущности бота;</p> <p>parkRepository – слой класса park для доступа к базе данных к сущности парк;</p> <p>botModelRepository – слой класса botModel для доступа к базе данных к сущности парк;</p>
<p>Методы:</p> <p>toEntity() – метод конвертации сущности из REST слоя в бизнес слой;</p> <p>toDto – метод конвертации сущности в объект представления;</p>

В таблице 9 приведен класс бота.

Таблица 9 – Класс Bot

Bot
<p>Поля:</p> <p>id – уникальный id бота;</p> <p>chargeLevel – уровень зарядки бота;</p> <p>botModel – модель бота;</p> <p>botState – расположение бота;</p> <p>botStatus – статус бота;</p> <p>park – парк, к которому относится бот;</p>

Архитектура, используемая в приложении, допускает зависимости от внешних уровней к внутренним, но не наоборот. Для конечной точки REST поток запроса может выглядеть следующим образом:

- запрос отправляется контроллеру на уровне «инфраструктуры»;
- контроллер десериализует запрос и – в случае успеха запрашивает результат у соответствующего сервиса на уровне сервисы;
- служба проверяет, есть ли у текущего пользователя разрешение на вызов функции, и инициализирует транзакцию базы данных (при необходимости);
- затем он извлекает данные из репозитория домена, манипулирует ими и, возможно, сохраняет их обратно в репозиторий;
- сервис также может вызывать несколько репозитория, преобразовывать и агрегировать результаты;
- репозиторий на уровне домена возвращает бизнес-объекты, этот уровень отвечает за поддержание всех необходимых объектов в допустимом состоянии;
- в зависимости от ответа сервиса, который является допустимым результатом или исключением, уровень инфраструктуры сериализует ответ.

Архитектурное решение «Портов» и «Адаптеров» позволяет связывать слои приложения через абстрактные точки, которые в свою очередь легко заменяются и масштабируются, но это может увеличить время прохода запроса через слои.

2.4 Разработка пользовательского программного обеспечения

Следующим этапом был разработан пользовательский интерфейс разработанного приложения, который представляет собой схему, изображенную на рисунке 8.

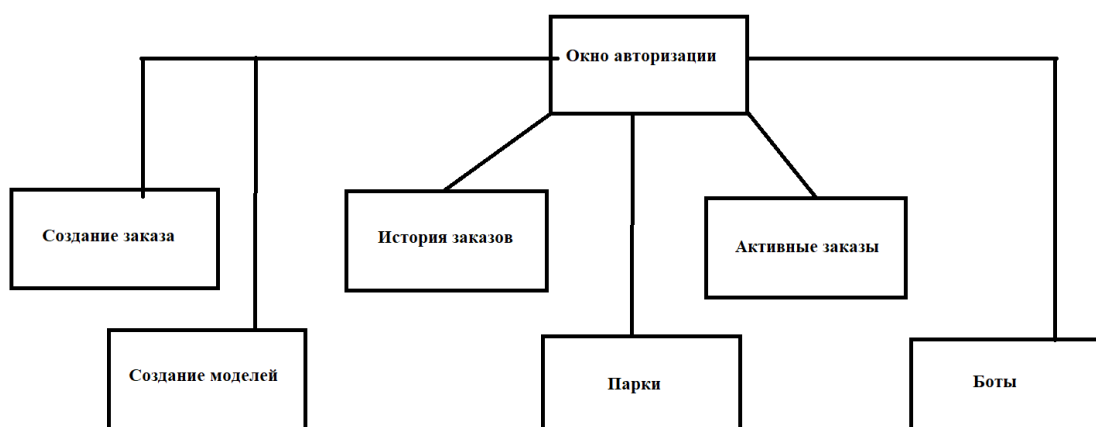


Рисунок 8 – Схема приложения

Разработка пользовательского интерфейса велась под технологией React.

Была создана базовая обертка над нужными компонентами для взаимодействия с пользователем. Схема архитектуры ПО представлена на рисунке 9.

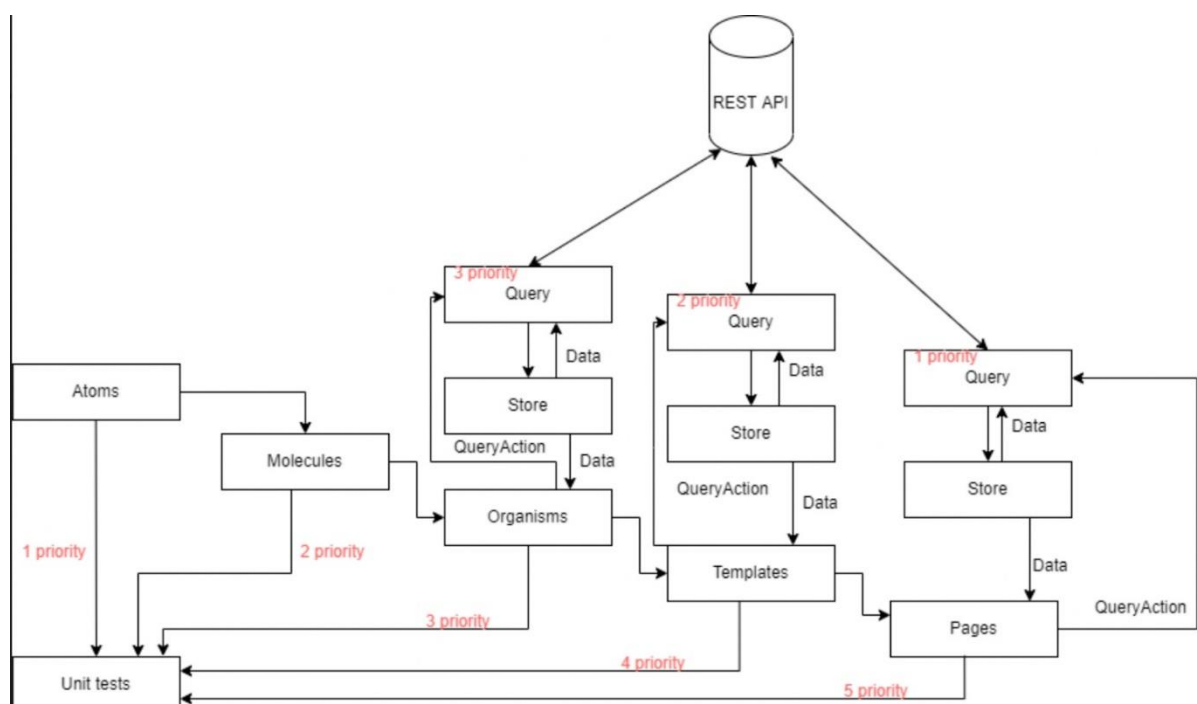


Рисунок 9 – Схема архитектуры пользовательского ПО

React – это инструмент для создания пользовательских интерфейсов. Его главная задача – обеспечение вывода на экран того, что можно видеть на веб-

страницах. React значительно облегчает создание интерфейсов благодаря разбиению каждой страницы на небольшие фрагменты.

В качестве архитектурного проектирования была взята технология Бреда Фроста – атомарный веб-дизайн.

Подход основывается на естественных науках. Все вещества (будь то твердые тела, жидкости, газы, простые, сложные и т.д) состоят из атомов. Атомы связываются между собой и образуют молекулы, которые, в свою очередь, комбинируются и формируют организмы. В конечном счете так создается все вещество в нашей вселенной.

Аналогично интерфейсы состоят из меньших компонент. Это означает, что можно разбить интерфейсы на фундаментальные блоки и комбинировать их, постепенно наращивая сложность. Это суть атомного веб дизайна.

Атомарный дизайн даёт возможность переходить от абстрактного к конкретному. Поэтому можно создавать системы, являющиеся масштабируемыми и обладающие единым, постоянным стилем, и в то же время показывающие итоговую общую картину. Процесс вместо деструктурирования превращается в сборку – создаём структуру с самого начала, а не тщательно выбираем подходящие шаблоны после.

Атомарный дизайн – это методология создания систем дизайна. В атомном дизайне есть пять отчётливых уровней:

- атомы;
- молекулы;
- организмы;
- шаблоны;
- страницы.

Атомы в природе – это основные строительные блоки материи. В контексте веб-интерфейсов атомы – это HTML тэги, такие как форма, поле ввода, или кнопка.

Всё становится интересней и оощутимей, когда начинаем комбинировать атомы. Молекулы – это совокупности связанных между собой атомов. Они имеют собственные свойства и играют роль скелета нашей системы дизайна.

Например – метка, поле ввода и кнопка не так уж применимы сами по себе, но могут быть действительно полезными, если их соединить.

Молекулы дают нам некоторые строительные блоки для работы. Теперь можем соединять их вместе для формирования организмов. Организмы – это объединения молекул, которые образуют довольно сложные, отдельные разделы интерфейса.

Организмы могут состоять из похожих или разных типов молекул. К примеру, заголовок страницы может состоять из разных компонент – логотипа, главного меню, формы поиска и списка медиаканалов. Но организм, отображающий сетку товаров, может содержать одну молекулу (изображение продукта, название и цена), которая повторяется много раз.

Переход от молекул к организмам способствует созданию самостоятельных компонент, готовых к повторному использованию.

На этой стадии перестанем проводить аналогии с химией, чтобы общаться на более осмысленном для наших клиентов языке. Шаблоны чаще всего состоят из групп организмов, связанных вместе и формирующих страницы. Именно на этом этапе становится видна общая картина.

Шаблоны очень конкретные и предоставляют контекст ко всем довольно абстрактным молекулам и организмам. Именно шаблон позволяет видеть конечный дизайн.

Страницы – это конкретные экземпляры шаблонов. В них для точной передачи того, что увидит пользователь, «заглушечный» контент заменён настоящим [3].

В листингах 1, 2 приведены коды атома для компонента `ButtonProps`. `ButtonProps` – кнопки расположенные в левой колонке (рисунок В.2).

Листинг 1 – Описание атома для компонента ButtonProps на языке JavaScript

```
import { memo } from 'react';

import styles from './button.module.css';

type ButtonProps = {
  children?: React.ReactNode;
  disabled?: boolean;
  onClick?: () => void;
  theme?: 'default';
  type?: 'submit' | 'reset' | 'button';
};

function CustomButton({
  onClick,
  disabled = false,
  children = '',
  theme = 'default',
  type = 'button',
}: ButtonProps) {
  return (
    <button
      className={`${styles.button} ${styles[theme]}
upperTitle`}
      onClick={onClick}
      disabled={disabled}
      type={type}
    >
      {children}
    </button>
  );
}

export const Button = memo(CustomButton);
```

Листинг 2 – Стили атома для компонента ButtonProps на языке css

```
.button {
  width: 100%;
  box-sizing: border-box;
  min-height: 48px;
  border-radius: 8px;
  padding: 12px 24px;
  transition: 0.5s;
  cursor: pointer;
}

.default {
  color: white;
  background: var(--orange);
}
```

```
border: none;  
}
```

Код серверной части приложения приведен в приложении Г.

3 Тестирование программного обеспечения

3.1 Обоснование выборов режимов тестирования

Существует четыре метода тестирования:

- модульное тестирование;
- интеграционное тестирование;
- системное тестирование;
- приемочные испытания.

Существующие технологии были исследованы и проанализированы. Из перечисленных выше методов был выбран метод интеграционного тестирования.

Интеграционное тестирование – это одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе, состоящей из двух и более модулей. Основная задача интеграционного тестирования – поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями.

Интеграционное тестирование отличается от других видов тестирования тем, что он сосредоточен в основном на интерфейсах и потоке данных (между модулями). Здесь приоритет проверки присваивается интегрирующим ссылкам, а не функциям блока, которые уже проверены.

При интеграционном тестировании важен только результат взаимодействия, а не детали и порядок вызовов. Поэтому на тесты не влияет рефакторинг кода. Не происходит избыточного или недостаточного тестирования – тестируются только те взаимодействия, которые встречаются при обработке реальных данных. Сами тесты легко поддерживать, так как спецификация хорошо читается и ее просто изменять в соответствии с новыми требованиями [6].

3.2 Интеграционное тестирование серверного программного обеспечения

Для интеграционного тестирования нашего приложения были использованы фреймворки JUnit и Spring Test.

Фреймворк Spring TestContext обеспечивает последовательную загрузку Spring ApplicationContext и WebApplicationContext, а также кэширование этих контекстов [7].

Поддержка кэширования загруженных контекстов важна, потому что время запуска может стать проблемой – не из-за накладных расходов самой Spring, а потому, что объекты, созданные контейнером Spring, требуют времени для создания экземпляра. Например, проекту с 50-100 файлами сопоставления Hibernate может потребоваться от 10 до 20 секунд для загрузки файлов сопоставления, и эти затраты перед запуском каждого теста в каждом тестовом устройстве приводят к более медленным общим запускам тестов, что снижает производительность разработчиков.

Тестовая платформа Spring MVC, также известная как MockMvc, обеспечивает поддержку тестирования приложений Spring MVC. Он выполняет полную обработку запросов Spring MVC, но через объекты mock request и response вместо запущенного сервера.

MockMvc можно использовать самостоятельно для выполнения запросов и проверки ответов. Его также можно использовать через WebTestClient, где MockMvc подключен в качестве сервера для обработки запросов. Преимуществом WebTestClient является возможность работать с объектами более высокого уровня вместо необработанных данных, а также возможность переключаться на полные, сквозные HTTP-тесты на живом сервере и использовать один и тот же тестовый API [8].

В приложении Б представлены реализации нескольких тестов и реальное применение описанных технологий.

3.3 Функциональное тестирование клиентского программного обеспечения

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения (ПО) исходным функциональным требованиям компании клиента. То есть проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям.

В таблице 10 приведены основные тесты и их описание.

Таблица 10 – Функциональное тестирование

Состав теста	Ожидаемый результат	Наблюдаемый результат
Тестирование работоспособности страницы для регистрации и авторизации пользователя.	Тест пройден	Тест пройден (рисунок В.1)
Тестирование работоспособности страницы для просмотра истории, выполненных заказов.	Тест пройден	Тест пройден (рисунок В.2)
Тестирование работоспособности страницы для просмотра карточки заказа.	Тест пройден	Тест пройден (рисунок В.3)
Тестирование работоспособности страницы для просмотра списка существующих парков.	Тест пройден	Тест пройден (рисунок В.4)
Тестирование работоспособности страницы для просмотра списка ботов.	Тест пройден	Тест пройден (рисунок В.5)
Тестирование работоспособности страницы для создания заказа.	Тест пройден	Тест пройден (рисунки В.6, В.7)

В приложении В приведены скриншоты, отражающие результаты функционального тестирования.

3.4 Метрики кода

Мера представляет собой числовой показатель, который отражает степень, количество или размер определенных атрибутов продукта или процесса. Например, количество ошибок может быть мерой.

Метрика программного кода является количественной мерой, которая позволяет оценить, насколько система, компоненты или процесс обладают определенным атрибутом. Метрики собираются до и после проведения рефакторинга. Примером метрики может быть количество обнаруженных ошибок в отношении человек/час, затраченных на разработку.

Метрики программного кода могут быть разделены на следующие группы:

- метрики размера программы (например, метрика строк исходного кода – SLOC);
- метрики сложности управления потоком программы (например, метрики Холстеда, цикломатическая сложность, метрики Чепина).

Рассчитаем соответствующие метрики кода для варианта использования «Обновить модель бота»

3.4.1 Метрика SLOC

Мерой называется количественный показатель, который позволяет оценить степень, количество или размеры кода. SLOC-метрика была разработана для оценки трудозатрат по проекту. Она используется для подсчета физических и логических строк кода. Физические строки включают все непустые строки в исходном файле программы, а логические строки кода отражают количество команд, используемых в программе. Кроме того, определение команды зависит от конкретного языка программирования [9].

На рисунке 10 приведен код для обновления данных модели бота.

```

public BotModelResponseDTO botUpdate(Long botId, BotRequestDTO
botRequestDTO) {
    Bot bot = BotRepository.findById(botId).orElseThrow(
        () -> new NotFoundException(ERROR_MESSAGE_NOT_FOUND_BOT));
    if(!bot.getStatus().getName().equals(BotRequestDTO.getStatus())) {
        if (BotRequestDTO.getStatus().equals(BotStatus.AVAILABLE.getName())
            ||
            BotRequestDTO.getStatus().equals(BotStatus.MAINTENANCE.getName())) {
            List<Bot> botList =
            userRepository.findById(BotRequestDTO.getUserId()).getBots();
            botList.remove(BotModelMapper.toDTO(BotRequestDTO));
            User user =
            userRepository.findById(BotRequestDTO.getUserId()).orElseThrow(
                () -> new
                NotFoundException(ERROR_MESSAGE_NOT_FOUND_USER));
            user.setBots(botList);
        }
    }
    Bot newBot = BotModelMapper.toDTO(botRequestDTO);
    newBot.setId(botId);
    return BotModelMapper.toDTO(BotRepository.save(newBot));
}

```

Рисунок 10 – Код для обновления данных модели бота

Применим SLOC – метрику к нашему программному коду. Результаты данной метрики представлены в таблице 11.

Таблица 11 – SLOC – метрики

Число физических строк	22
Число логических строк	13
Число пустых строк	3
Число строк-комментариев	0

Была применена SLOC – метрика. Следующим этапом измерим метрику Холстеда для нашего программного кода.

3.4.2 Метрики Холстеда

Метрики Холстеда измеряются следующими характеристиками программы:

- количество уникальных операторов, включая символы-разделители, имена процедур и знаки операций;
- количество уникальных операндов;
- общее количество операторов в программе;
- общее количество операндов в программе.

На основе этих метрик можно получить следующие оценки:

- словарь программы: сумма количества уникальных операторов и уникальных операндов;
- длина программы: сумма общего количества операторов и операндов;
- объем программы: произведение длины программы на логарифм по основанию 2 от словаря программы;
- сложность программы: произведение половины количества уникальных операторов на отношение общего количества операндов к количеству уникальных операндов;
- показатель усилий программиста: произведение сложности программы на объем программы.

Применим метрики Холстеда к нашему программному коду. Результаты представлены в таблице 12.

Таблица 12 – Метрики Холстеда

Число операторов (уникальных)	Число операндов (уникальных)
6 (3)	31 (18)

Получим следующие оценки:

- словарь программы = $3 + 18 = 21$;
- длина программы = $6 + 31 = 37$;
- объем программы = $37 \log_2 21 = 162,5$;
- сложность программы = $(3/2) * (31/18) = 2,6$;
- показатель усилий программиста = $422,5$.

3.4.3 Метрики устойчивости и абстрактности компонент

Один из способов определения устойчивости компонент – подсчитать количество входящих и исходящих зависимостей. Вместе они позволяют вычислить позиционную устойчивость компонента:

$$I \text{ (неустойчивость): } I = \frac{Ce}{Ca + Ce},$$

где C_e (исходящие связи) – количество классов внутри данного компонента, зависящих от классов вне компонента;

C_a (входящие связи) – количество классов вне данного компонента, зависящих от классов внутри компонента.

Пример результата вычисления метрики для компонента сервисного слоя, предназначенного для обработки информации о ботах, который находится в классе BotServiceImpl.

В данном случае величина $C_a = 3$, $C_e = 1$, $I = 1/4$.

Метрика A , являющаяся мерой абстрактности компонента, равна отношению количества абстрактных классов к общему количеству классов в компоненте.

$$A \text{ (абстрактность): } A = \frac{N_a}{N_c},$$

где N_a – количество абстрактных классов в компоненте;

N_c – количество классов в компоненте.

В коде программы абстракцией для класса BotServiceImpl является BotService. Соответственно метрика $A = 0.5$.

Соотношение между устойчивостью (I) и абстрактностью (A) компонента обработки информации о ботах можно представить на следующем графике, изображенном на рисунке 11.

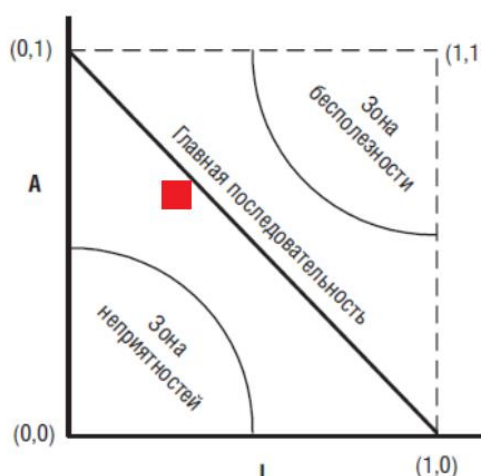


Рисунок 11 – Соотношение между устойчивостью (I) и абстрактностью (A) для компонента BotServiceImpl

На рисунке 11 максимально устойчивые и абстрактные компоненты находятся в левом верхнем углу, в точке (0,1), а максимально неустойчивые и конкретные – в правом нижнем углу, в точке (1,0). Желательно, чтобы компонент находился на главной последовательности или близко к ней. Для измерения, насколько далеко компонент отстоит от идеала, определяют следующую метрику:

D (расстояние от главной последовательности):

$$D = \frac{|A + I - 1|}{\sqrt{2}},$$

где A – абстрактность;

I – неустойчивость.

Величина D изменяется в диапазоне $[0; \sim 0,707]$. Для компонента BotServiceImpl это приблизительно 0,18.

Для удобства анализа можно использовать нормированное расстояние:

$$D' = |A + I - 1|$$

Метрика D' изменяется в диапазоне $[0,1]$. При этом 0 означает, что компонент находится на главной последовательности, а 1 – что он максимально удален от нее. Для компонента BotServiceImpl $D'=0,25$.

3.4.4 Комплексный показатель качества кода

Рассмотрим подсчет данной метрики на примере метода получения всех ботов, привязанных к пользователю. Данный метод находится в классе UserService.

HV = 8.

CC = 4.

LoC = 7.

MI = 93,14.

Данное значение показывает, что метод легко поддерживать.

4 Планирование и оценка бюджета проекта

Для выполнения поставленных задач проекта и достижения цели необходимо учитывать время, бюджет, ресурсы, а также все риски данного проекта. Именно контроль этих факторов называется управлением проектом.

Управление проектом – деятельность по решению задач и достижению поставленных целей проект. Управление проектом характеризуется как организация процесса по успешному достижению поставленной цели (за отведенное время, выделенный бюджет, имеющиеся в наличии ресурсы) и нахождение путей развития проекта в условиях кризиса или непредвиденных ситуаций. При осуществлении управления нужно учитывать правило треугольника проекта, которое характеризует связь между параметрами ВРЕМЯ-КАЧЕСТВО-БЮДЖЕТ и гласит, что изменение любой из этих составляющих влияет на две другие.

При планировании разработки было выделено 22 задачи. В таблице 13 представлены наименование, предшественник и длительность каждой из задач.

Таблица 13 – Перечень задач

№	Наименование	Длительность	Предшественник
1	Начало реализации проекта	-	
2	Программирование	9 дней	
3	Разработка серверной части приложения	9 дней	
4	Постановка задачи	1 день	1
5	Разработка структуры базы данных	3 дня	4
6	Заполнение базы данных	2 дня	5
7	Разработка триггеров	3 дня	6
8	Разработка серверной части завершена	-	7
9	Разработка клиентской части приложения	5 дней	
10	Постановка задачи	1 день	1

Продолжение таблицы 13

№	Наименование	Длительность	Предшественник
11	Разработка интерфейса	4 дня	10
12	Разработка клиентской части завершена	-	11
13	Программирование завершено	-	8;12
14	Отладка	5 дней	
15	Тестирование и исправление ошибок	3 дня	13
16	Составление программной документации	2 дня	15
17	Отладка завершена	-	16
18	Документирование	2 дня	
19	Создание установочного файла	1 день	17
20	Написание руководства пользования	1 день	19
21	Документация завершена	-	20
22	Конец проекта	-	21

Планирование разработки и оценка бюджета разрабатываемого проекта были выполнены в программе Microsoft Project 2007. Даная программа обеспечивает разработку планов, распределение ресурсов по задачам, отслеживание прогресса и анализ объёмов работ, а также позволяет генерировать расписания критического пути.

Ниже представлены две части представления диаграммы Ганта: табличная часть, изображенная на рисунке 12, и графическая часть, проиллюстрированная на рисунке 13. Табличное представление диаграммы Ганта содержит подробное описание задач. Графическое представление диаграммы Ганта отображает график выполнения работ по проекту [10].

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Ср 15.03.23	Ср 15.03.23		
2	Программирование	9 дней?	Ср 15.03.23	Пн 27.03.23		Интернет[800,00 P]
3	Разработка серверной части приложения	9 дней	Ср 15.03.23	Пн 27.03.23		
4	Постановка задачи	1 день	Ср 15.03.23	Ср 15.03.23	1	Ведущий программист; Программист
5	Разработка структуры базы данных	3 дней	Чт 16.03.23	Пн 20.03.23	4	Ведущий программист
6	Заполнение базы данных	2 дней	Вт 21.03.23	Ср 22.03.23	5	Ведущий программист
7	Разработка триггеров	3 дней	Чт 23.03.23	Пн 27.03.23	6	Ведущий программист
8	Разработка серверной части завершена	0 дней	Пн 27.03.23	Пн 27.03.23	7	
9	Разработка клиентской части приложения	5 дней?	Ср 15.03.23	Вт 21.03.23		
10	Постановка задачи	1 день?	Ср 15.03.23	Ср 15.03.23	1	Ведущий программист; Программист
11	Разработка интерфейса	4 дней	Чт 16.03.23	Вт 21.03.23	10	Программист; Ведущий программист[50%]
12	Разработка клиентской части завершена	0 дней	Вт 21.03.23	Вт 21.03.23	11	
13	Программирование завершено	0 дней	Пн 27.03.23	Пн 27.03.23	8,12	
14	Отладка	5 дней	Вт 28.03.23	Пн 03.04.23		Интернет[800,00 P]
15	Тестирование и исправление ошибок	3 дней	Вт 28.03.23	Чт 30.03.23	13	Тестировщик
16	Составление программной документации	2 дней	Пт 31.03.23	Пн 03.04.23	15	Программист; Тестировщик; Ведущий программист[50%]
17	Отладка завершена	0 дней	Пн 03.04.23	Пн 03.04.23	16	
18	Документирование	2 дней	Вт 04.04.23	Ср 05.04.23		Интернет[800,00 P]
19	Создание установочного файла	1 день	Вт 04.04.23	Вт 04.04.23	17	Тестировщик
20	Написание руководства пользования	1 день	Ср 05.04.23	Ср 05.04.23	19	Тестировщик
21	Документация завершена	0 дней	Ср 05.04.23	Ср 05.04.23	20	
22	Конец проекта	0 дней	Ср 05.04.23	Ср 05.04.23	21	

Рисунок 12 – Табличное представление диаграммы Ганта

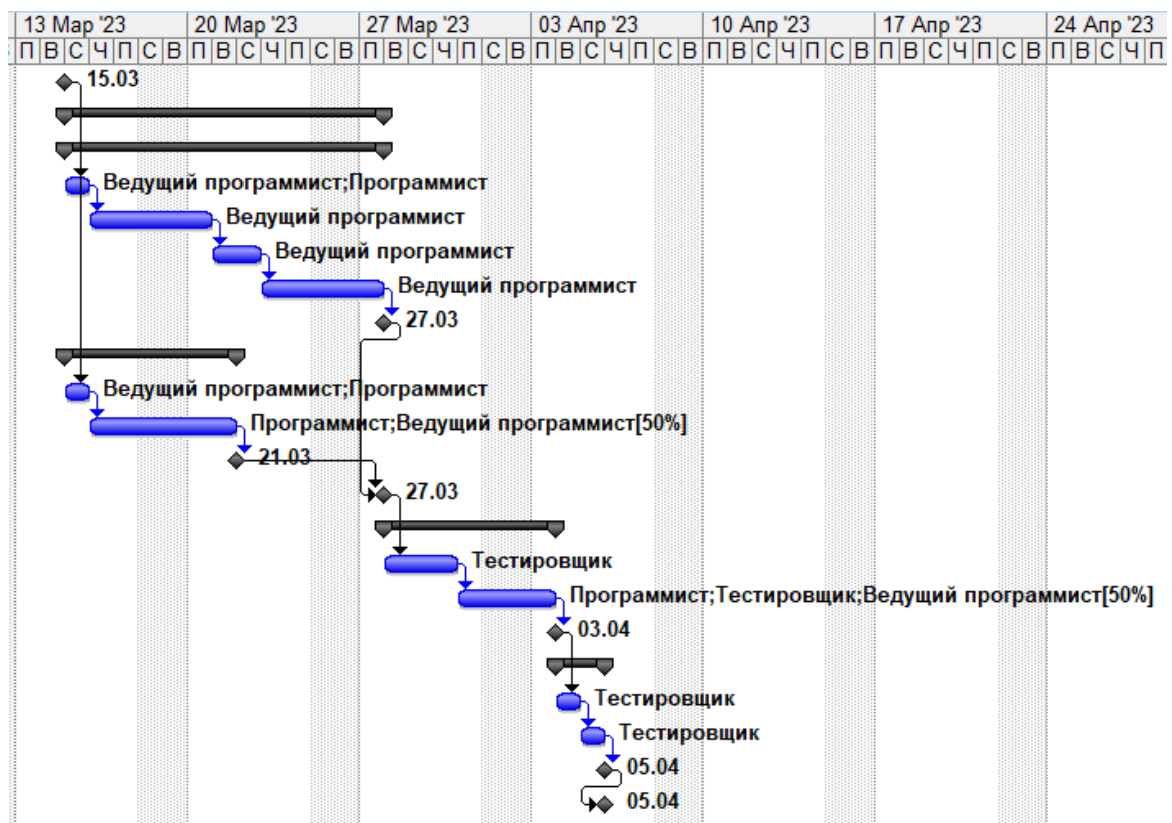


Рисунок 13 – Графическое представление диаграммы Ганта

На рисунке 14 представлен график ресурсов. Синим цветом – отображены нормы рабочего времени, красным – переработки.

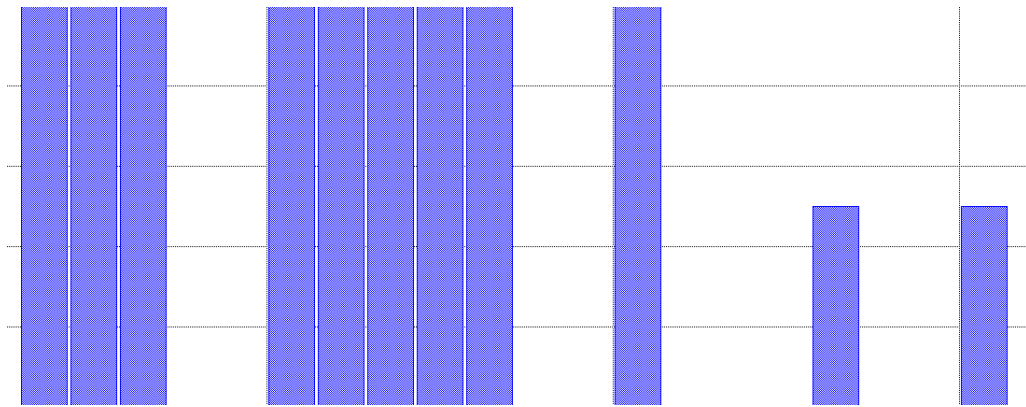


Рисунок 14 – График ресурсов

Исходя из рисунка, можем сделать вывод о том, что переработок нет. Кроме того, в некоторые дни, программист занят только на 50% времени. Соответственно данный план можно считать корректным.

Итоговая стоимость проекта составила: 41025 рублей. В неё входят заработные платы работников, а также траты на электричество и интернет.

Затраты проекта изображены на рисунке 15.

Затраты			
Текущий план:	41 025,00 Р	Оставшиеся:	41 025,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	41 025,00 Р		

Рисунок 15 – Затраты проекта

В результате планирования разработки и оценки бюджета была построена диаграмма Ганта и посчитана итоговая стоимость проекта.

Заключение

В данном проекте были выполнены поставленные задачи на анализ требований, разработку системы и тестирование продукта.

Разработка приложения выполнялась в IntelliJ IDEA 2021.1.3 на языке программирования Java, при разработке приложения были применены принципы SOLID, позволяющие проще поддерживать, наследовать и изменять программное обеспечение.

Также, на основе представленной архитектуры и готового решения можно сделать вывод о том, что «Система дронов доставщиков» является перспективным направлением на данный момент времени.

В будущем разработанное приложение может быть улучшено и внедрено в сферу доставки.

Список использованных источников

1. Сервисы доставки ждут 10-кратного роста за счёт // InvestFuture. [Электронный ресурс]. URL: <https://investfuture.ru/articles/id/servisy-dostavki-jdut-10-kratnogo-rosta-za-schet-robotov> (дата обращения 28.04.2023).
2. ЯндексДоставка // ЯндексДоставка. [Электронный ресурс]. URL: <https://dostavka.yandex.ru/> (дата обращения 28.04.2023).
3. СберЛогистика // СБЕРЛОГИСТИКА. [Электронный ресурс]. URL: <https://sberlogistics.ru/> (дата обращения 28.04.2023).
4. Программные возможности передачи данных. Сервер. // Studfile. [Электронный ресурс]. URL: <https://studfile.net/preview/9331540/page:15/> (дата обращения 29.04.2024).
5. Атомарный веб – дизайн // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/articles/249223/> (дата обращения 01.05.2023).
6. Тестирование программного обеспечения // Studfile. [Электронный ресурс]. URL: <https://studfile.net/preview/16543027/> (дата обращения 07.05.2023)
7. Уоллс Крейг. Spring в действии. / ДМК Пресс, 2015. – 752 с.
8. Юлиана Козмина, Роб Харроп, Крис Шефер, Кларенс Хо. Spring 5 для профессионалов / Диалектика, 2019. – 521 с.
9. Программный код и его метрики // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 13.05.2023).
10. Андрей Шкрыль. MS Project 2007 современное управление проектами. / БХВ-Петербург, 2007. – 257 с.
11. Мартин Роберт. Чистый код. Создание, анализ и рефакторинг. / Питер, 2018. – 464 с.
12. Кэти Сьерра и Берт Бейтс. Изучаем Java. / Эксмо, 720. – 720 с.
13. Бэнкс, Порселло. React. Современные шаблоны для разработки приложений. / Питер, 2021. – 320 с.

14. Э. Гамма, Р.Хелм, Р.Джонсон Приемы объектно-ориентированного программирования. Паттерны проектирования – Питер, 2001. – 368 с.
15. Принципы SOLID // Metanit. [Электронный ресурс]. URL: <https://metanit.com/sharp/patterns/> (Дата обращения: 13.05.2023)
16. Л. В. Гурьянов [и др.], под ред. П. П. Макарычева. Выполнение выпускной квалификационной работы: учебное пособие для бакалавров. Пенза: Издательство Пензенского государственного университета, 2015. – 119 с.

Приложение А

ОПИСАНИЕ ТАБЛИЦ БАЗЫ ДАННЫХ

(рекомендуемое)

Таблица А.1 – Спецификация класса «Модель робота»

Класс: Модель робота
Назначение: класс, описывающий модель робота и его характеристики
Члены класса: <ul style="list-style-type: none"> • Возможность защиты – наличие защиты • Наименование – название модели • Скорость – параметр, показывающий возможную скорость • Объем багажника – вместимость багажника • Грузоподъемность – возможная нагрузка • Затраты ТО – затраты на техническое обслуживание
Отношения: <p>Связь «один ко многим» с классом «Робот».</p>

Таблица А.2 – Спецификация класса «Робот»

Класс: Робот
Назначение: класс, описывающий робота и его параметры
Члены класса: <ul style="list-style-type: none"> • Статус – параметр, показывающий статус робота. • Состояние – параметр, показывающий состояние робота. • Уровень зарядки – параметр, показывающий уровень зарядки робота. • Модель робота – параметр, показывающий модель робота. • Парк – параметр, показывающий к какому парку принадлежит робот.
Отношения: <p>Связь «многие к одному» с классом «Парк».</p> <p>Связь «многие к одному» с классом «Модель робота».</p> <p>Связь «один ко многим» с классом «Сессия заказа».</p>

Таблица А.3 – Спецификация класса «Парк»

Класс: Парк
Назначение: класс, описывающий парк и его параметры
Члены класса: <ul style="list-style-type: none"> • Широта – параметр, показывающий широту местоположения парка. • Долгота – параметр, показывающий долготу местоположения парка. • Стоимость ТО – параметр, показывающий стоимость технического обслуживания. • Наименование – название парка. • Прибыль – параметр, показывающий прибыль парка.
Отношения: <p>Связь «один ко многим» с классом «Робот».</p>

Таблица А.4 – Спецификация класса «Заказ»

Класс: Заказ
Назначение: класс, описывающий заказ
Члены класса: <ul style="list-style-type: none"> • Широта пункта выдачи – параметр, показывающий широту местоположения пункта выдачи. • Долгота пункта выдачи – параметр, показывающий долготу местоположения пункта выдачи. • Широта места назначения – параметр, показывающий широту местоположения места назначения. • Долгота места назначения – параметр, показывающий долготу местоположения места назначения.
Отношения: <p>Связь «один ко многим» с классом «Сессия заказа».</p> <p>Связь «многие к одному» с классом «Пользователь».</p>

Таблица А.5 – Спецификация класса «Сессия заказа»

Класс: Сессия заказа
Назначение: класс, описывающий сессию заказа и его параметры
Члены класса: <ul style="list-style-type: none"> • Время открытия сессии – параметр, показывающий время открытия сессии. • Время закрытия сессии – параметр, показывающий время закрытия сессии. • Активность сессии – параметр, показывающий активность сессии. • Робот – параметр, показывающий робота, который выполняет заказ • Заказ – параметр, показывающий заказ, принадлежащий к сессии. • Успешность заказа – параметр, показывающий выполнен ли заказ. • Прибыль заказа – параметр, показывающий прибыль, полученную с заказа.
Отношения: <p>Связь «многие к одному» с классом «Заказ».</p> <p>Связь «многие к одному» с классом «Робот».</p> <p>Связь «один ко многим» с классом «Действия робота».</p> <p>Связь «один ко многим» с классом «События робота».</p>

Таблица А.6 – Спецификация класса «События робота»

Класс: События робота
Назначение: класс, описывающий события, происходящие с роботом
Члены класса: <ul style="list-style-type: none"> • Вид события – параметр, показывающий вид события. • Широта события – параметр, показывающий широту местоположения произошедшего события • Долгота события – параметр, показывающий долготу местоположения произошедшего события • Время события – параметр, показывающий время произошедшего события • Сессия заказа – параметр, показывающий сессию заказу.
Отношения: <p>Связь «многие к одному» с классом «Сессия заказа».</p>

Таблица А.7 – Спецификация класса «Действия робота»

Класс: Действия робота
Назначение: класс, описывающий действия, выполненные роботом.
Члены класса: <ul style="list-style-type: none"> • Вид действия – параметр, показывающий вид действия. • Время действия – параметр, показывающий время действия. • Сессия заказа – параметр, показывающий сессию действия.
Отношения: <p>Связь «многие к одному» с классом «Сессия заказа».</p>

Приложение Б
СЕРВЕРНОЕ ТЕСТИРОВАНИЕ
(рекомендуемое)

```

@SpringBootTest
@ActiveProfiles({"test"})
@AutoConfigureMockMvc(addFilters = false)
class BotAPITest {

    @Autowired
    private MockMvc mockMvc;

    /*@Autowired
    private BotDataset botDataset;*/

    @Autowired
    private BotService botService;

    @Autowired
    private ParkService parkService;

```

Рисунок Б.1 – Конфигурация окружения теста для слоев сущности Bot

```

@Test
@DisplayName("Создание бота")
void create() throws Exception {
    //final var botModel = MARK_45;
    final var chargeLevel = 100;
    final var botState : BotState = BotState.EFFICIENT;
    final var botStatus : BotStatus = BotStatus.AVAILABLE;
    final var parkId : Long = getOnePark().getId();

    final var botRqDto = BotRqDto.builder()
        .botModel(botModel)
        .botState(botState)
        .chargeLevel(chargeLevel)
        .botStatus(botStatus)
        .parkID(parkId)
        .build();

    mockMvc.perform(postJson(botRqDto, COMMON_API_URI))
        .andExpect(status().isCreated())
        .andExpect(jsonPath("chargeLevel", equalTo(chargeLevel)))
        // .andExpect(jsonPath("botModel", equalTo(botModel.toString())))
        .andExpect(jsonPath("expression: botState", equalTo(botState.toString())))
        .andExpect(jsonPath("expression: botStatus", equalTo(botStatus.toString())))
        .andExpect(jsonPath("expression: park.id", equalTo(parkId.intValue())));
}

```

Рисунок Б.2 – Тест создания сущности

```

@Test
@DisplayName("Создание бота, пустой запрос")
void createNullRequest() throws Exception {
    mockMvc.perform(postJson(body: null, COMMON_API_URI))
        .andDo(print())
        .andExpect(status().isBadRequest());
}

@Test
@DisplayName("Создание бота пустой статус")
void createBuilderNullField() throws Exception {

    final var chargeLevel = 100;
    final var parkId : Long = getOnePark().getId();
    final var botRqDto : BotDto.BotRqDto = BotRqDto.builder()
        // .botModel(BotModel.MARK_45)
        // .botState(BotState.EFFICIENT)
        .chargeLevel(chargeLevel)
        .parkID(parkId)
        .build();

    mockMvc.perform(postJson(botRqDto, COMMON_API_URI))
        .andDo(print())
        .andExpect(status().isBadRequest());
}

```

Рисунок Б.3 – Тест создания сущности с пустыми запросами

```

@Test
@DisplayName("Создание бота с неизвестным парком")
void createUnknownPark() throws Exception {
    final var botRqDto : BotDto.BotRqDto = BotRqDto.builder()
        .botModelId(1L)
        // .botState(EFFICIENT)
        .chargeLevel(100)
        .parkID(1111141151L)
        // .botStatus(AVAILABLE)
        .build();

    mockMvc.perform(postJson(botRqDto, COMMON_API_URI))
        .andDo(print())
        .andExpect(status().isNotFound());
}

```

Рисунок Б.4 – Тест создания сущности Дрона с неизвестным парком

```

@Test
@DisplayName("Изменение бота")
void update() throws Exception {

    final var botRsDto : BotDto.BotRsDto = getOneBot();
    final var botId : Long = botRsDto.getId();
    //final var botModel = ATOM_96;
    final var chargeLevel = 70;
    final var botState : BotState = OUT_OF_SERVICE;
    final var botStatus : BotStatus = AVAILABLE;
    final var parkId : Long = getOnePark().getId();

    final var botRqDto : BotDto.BotRqDto = BotRqDto.builder()
        // .botModel(botModel)
        // .botState(botState)
        .chargeLevel(chargeLevel)
        // .botStatus(botStatus)
        .parkID(parkId)
        .build();

    mockMvc.perform(putJson(botRqDto, uriTemplate: COMMON_API_URI +("/{id}", botId))
        .andDo(MockMvcResultHandlers.print())
        .andExpect(status().isOk())
        .andExpect(jsonPath("id", equalTo(botId.intValue())))
        .andExpect(jsonPath("chargeLevel", equalTo(chargeLevel)))
        // .andExpect(jsonPath("botModel", equalTo(botModel.toString())))
        .andExpect(jsonPath("botState", equalTo(botState.toString())))
        .andExpect(jsonPath("botStatus", equalTo(botStatus.toString())))
        .andExpect(jsonPath("park.id", equalTo(parkId.intValue())));
}

```

Рисунок Б.5 – Тест изменения сущности Дрона

```

@Test
@DisplayName("Изменение бота пустой запрос")
void updateNullRequest() throws Exception {
    final var botId : Long = getOneBot().getId();

    mockMvc.perform(putJson( body: null, uriTemplate: COMMON_API_URI + "/" + botId, API_VERSION))
        .andDo(MockMvcResultHandlers.print())
        .andExpect(status().isBadRequest());
}

@Test
@DisplayName("Изменение бота по id = null")
void updateNullField() throws Exception {
    final var botRqDto : BotDto.BotRqDto = BotRqDto.builder().build();

    mockMvc.perform(putJson(botRqDto, uriTemplate: COMMON_API_URI + "/" + null, API_VERSION))
        .andDo(MockMvcResultHandlers.print())
        .andExpect(status().isBadRequest());
}

```

Рисунок Б.6 – Тест изменения сущности с помощью пустых запросов

```

@Test
@DisplayName("Получение бота по id")
void getById() throws Exception {
    final var botRsDto : BotDto.BotRsDto = getOneBot();
    final var botId : Long = botRsDto.getId();
    //final var botModel = botRsDto.getBotModelRsDto();
    final var chargeLevel : Integer = botRsDto.getChargeLevel();
    final var botState : BotState = botRsDto.getBotState();
    final var botStatus : BotStatus = botRsDto.getBotStatus();
    //final var botParkId = botRsDto.getParkRsDto().getId();

    mockMvc.perform(get( urlTemplate: COMMON_API_URI +("/{id}", botId))
        .andDo(MockMvcResultHandlers.print())
        .andExpect(jsonPath( expression: "id", equalTo(botId.intValue())))
        .andExpect(jsonPath( expression: "chargeLevel", equalTo(chargeLevel)))
        // .andExpect(jsonPath("botModel", equalTo(botModel.toString())))
        .andExpect(jsonPath( expression: "botState", equalTo(botState.toString())))
        .andExpect(jsonPath( expression: "botStatus", equalTo(botStatus.toString())));
    // .andExpect(jsonPath("park.id", equalTo(botParkId.intValue())));
}

```

Рисунок Б.7 – Тест получения сущности по id

```

@Test
@DisplayName("Получение несуществующего бота по id")
void getByIncorrectId() throws Exception {
    mockMvc.perform(get( urlTemplate: COMMON_API_URI + "/" + 104L, API_VERSION)
        .contentType(MediaType.APPLICATION_JSON)
        .andDo(MockMvcResultHandlers.print())
        .andExpect(status().isNotFound());
}

@Test
@DisplayName("Получение бота по id = null")
void getByNullId() throws Exception {
    mockMvc.perform(get( urlTemplate: COMMON_API_URI + "/" + null, API_VERSION)
        .contentType(MediaType.APPLICATION_JSON)
        .andDo(MockMvcResultHandlers.print())
        .andExpect(status().isBadRequest());
}

```

Рисунок Б.8 – Тест получения сущности с помощью пустых запросов

Приложение В

ТЕСТИРОВАНИЕ КЛИЕНТСКОГО ПО

(рекомендуемое)

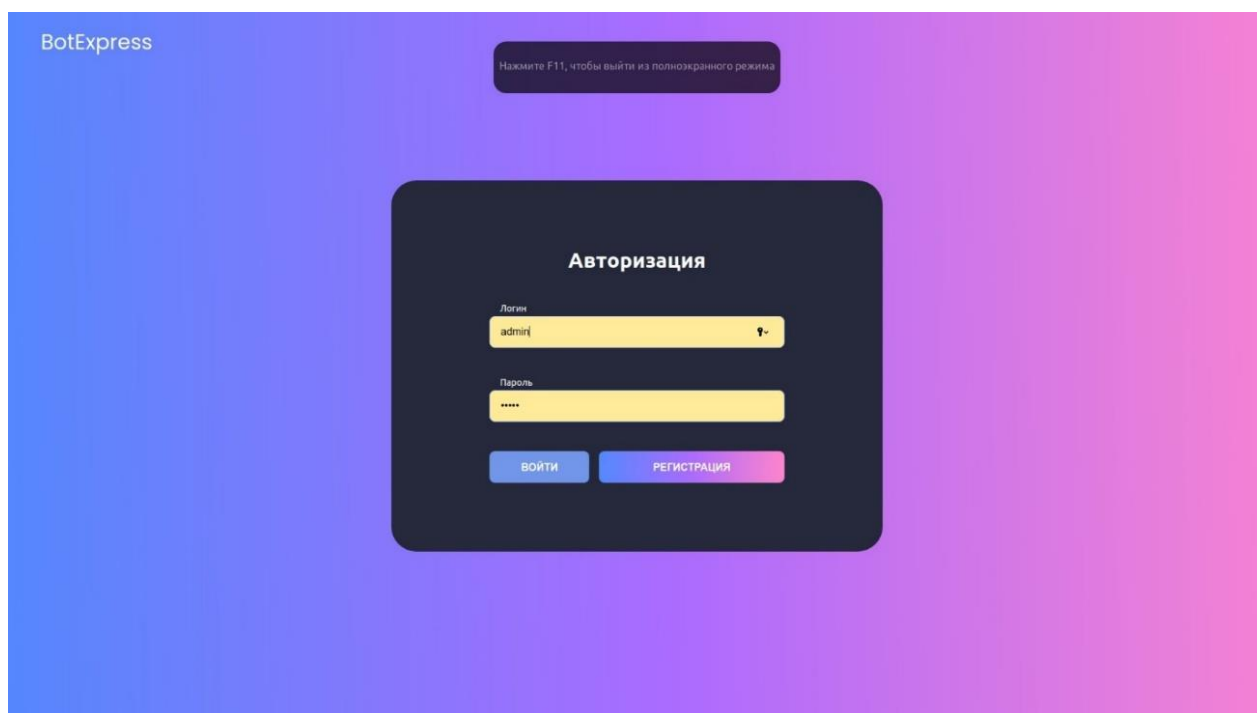


Рисунок В.1 – Окно авторизации

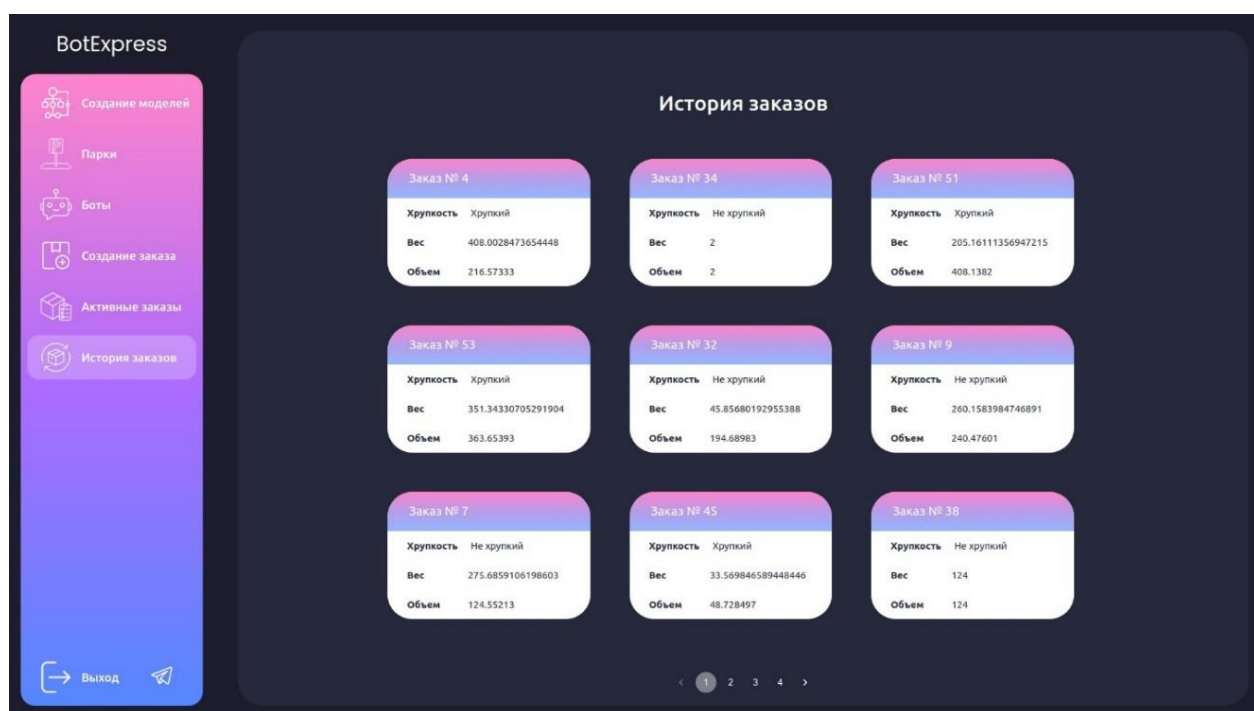


Рисунок В.2 – История заказов

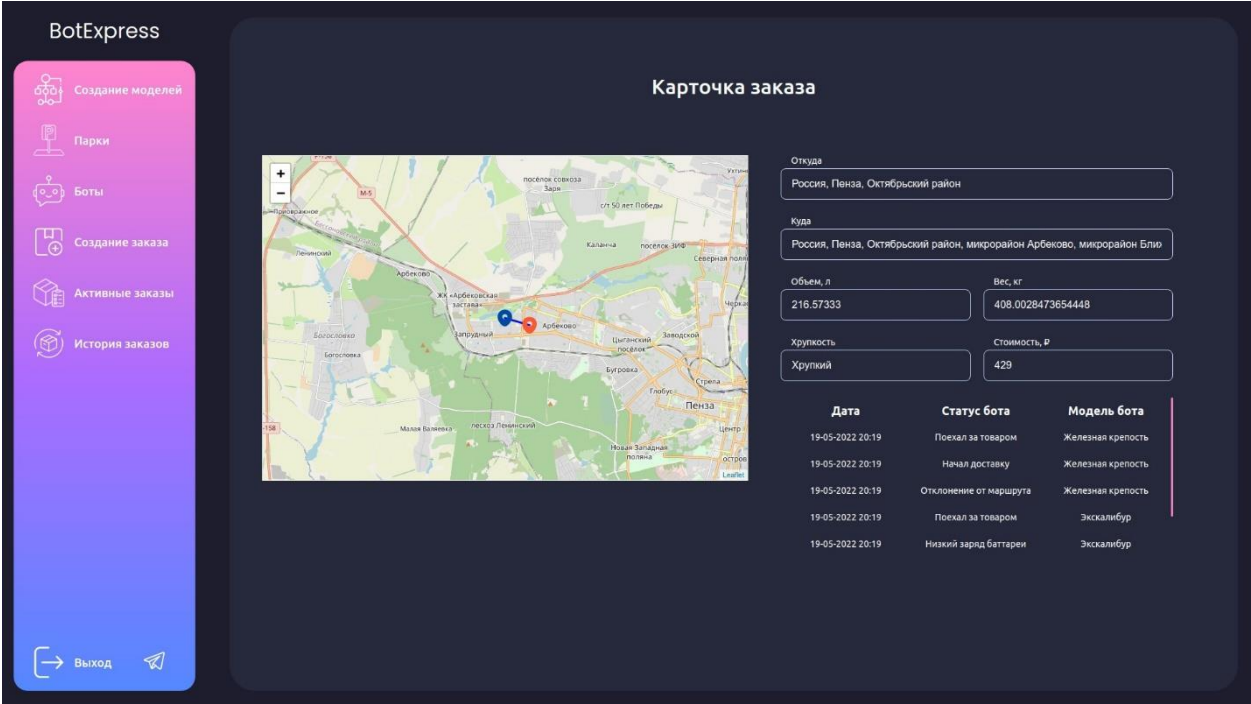


Рисунок В.3 – Карточка заказа

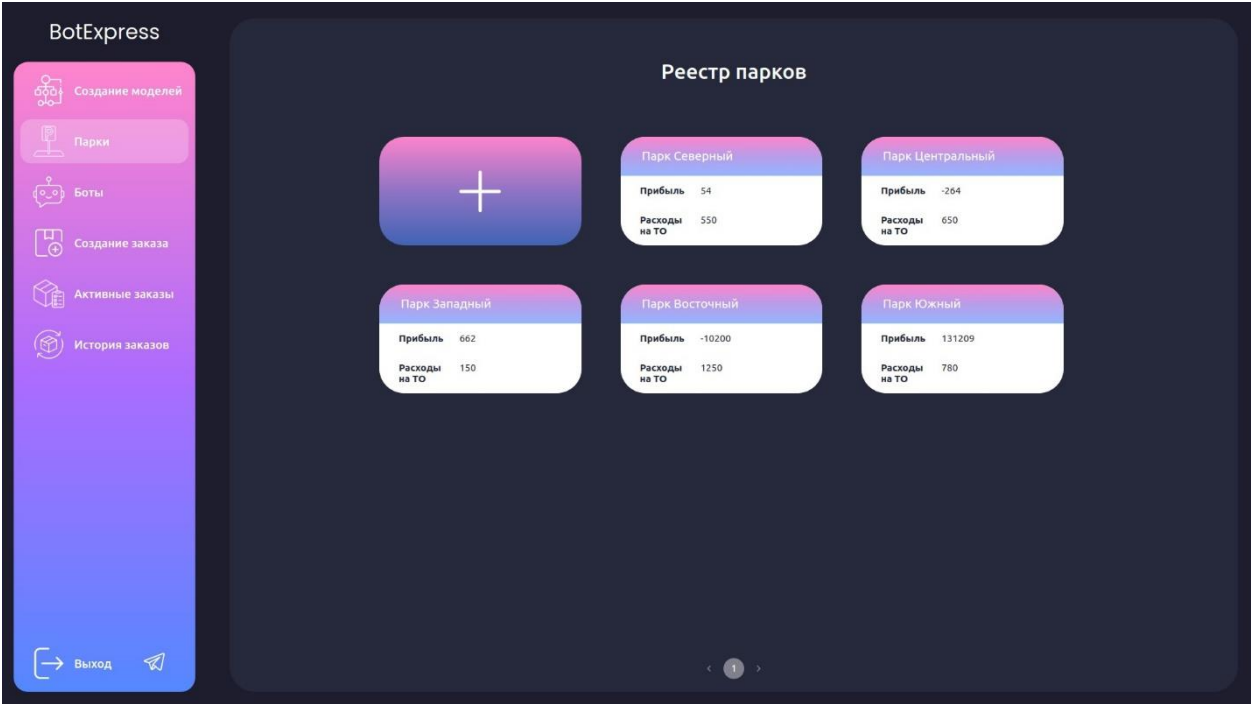


Рисунок В.4 – Список существующих парков

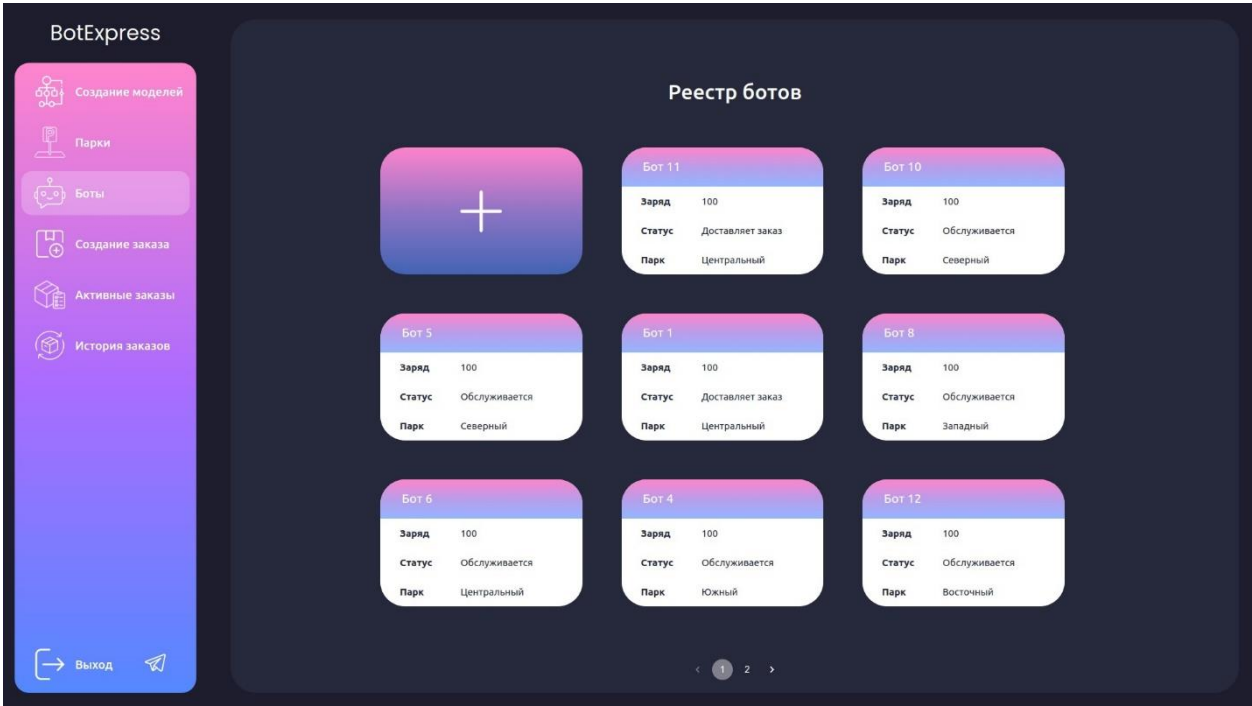


Рисунок В.5 – Список ботов

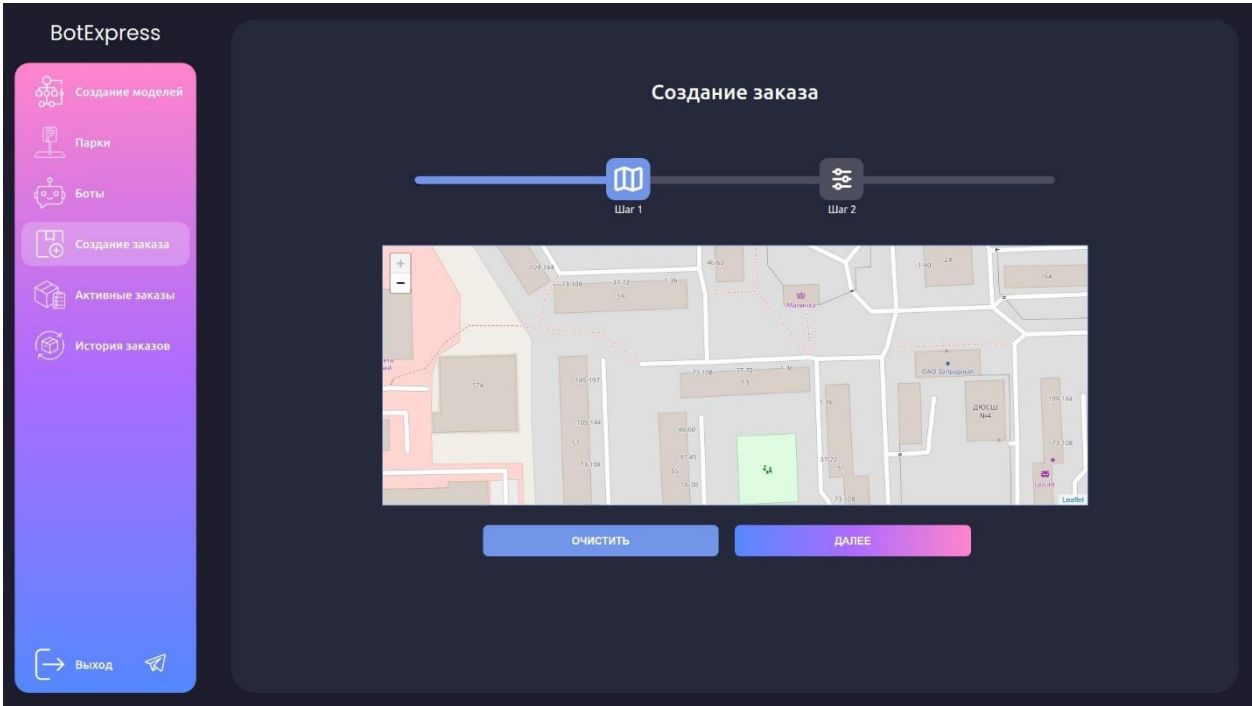


Рисунок В.6 – Создание заказа (Шаг 1)

BotExpress

Создание моделей
Парки
Боты
Создание заказа
Активные заказы
История заказов

Выход

Создание заказа

Шаг 1 Шаг 2

Откуда	Куда
Россия, Пенза, проспект Строителей, 59	Россия, Пенза, проспект Строителей, 45
Вес, кг	Хрупкость
20	Не хрупкий
Объем, л	Стоимость, Р
56	279

НАЗАД ОФОРМИТЬ ЗАКАЗ

Рисунок В.7 – Создание заказа (Шаг 2)

Приложение Г
КОД ПРОГРАММЫ
(рекомендуемое)

Листинг Г.1 – Файл OpenAPIConfig.java

```

@SecurityScheme(
    name = "basic-auth", //@SecurityRequirement(name =
"basic-auth")
    type = SecuritySchemeType.HTTP,
    scheme = "basic" //basic - имя базовой аутентификации
)
@OpenAPIDefinition(
    info = @Info(title = "Bot API", version = "1.0",
description = "Bot Express")
)
@Configuration
class OpenAPIConfig {
}

@Validated
@RequestMapping("/bots")
@SecurityRequirement(name = "basic-auth")
@Tag(name = "Боты", description = "Контроллер для управления
ботами")
@PreAuthorize("hasRole(T(ru.kostyushin.botexpress.security.store.e
ntity.enums.RoleName).ROLE_ADMIN)")

```

Листинг Г.2 – Файл BotAPI.java

```

public interface BotAPI {

    @Operation(summary = "Метод создания бота", method = "POST",
        responses = {
            @ApiResponse(description = "CREATED",
responseCode = "201",
                headers = @Header(name = "content-
type", description = "тип контента"),
                content = @Content(mediaType =
MediaType.APPLICATION_JSON_VALUE,
                    schema =
@Schema(implementation = BotResponseDTO.class))),
            @ApiResponse(description = "BAD REQUEST",
responseCode = "400"),
            @ApiResponse(description = "UNAUTHORIZED",
responseCode = "401"),
            @ApiResponse(description = "FORBIDDEN",
responseCode = "403")
        }
    )

    ResponseEntity<BotResponseDTO> create(@RequestBody
BotRequestDTO botReqDto);

    @Operation(summary = "Метод получения бота по идентификатору",
method = "GET",
        responses = {
            @ApiResponse(description = "OK", responseCode
= "200",

```

```

        headers = @Header(name = "content-
type", description = "тип контента"),
        content = @Content(mediaType =
MediaType.APPLICATION_JSON_VALUE,
        schema =
@Schema(implementation = BotResponseDTO.class))),
        @ApiResponse(description = "BAD REQUEST",
responseCode = "400"),
        @ApiResponse(description = "UNAUTHORIZED",
responseCode = "401"),
        @ApiResponse(description = "FORBIDDEN",
responseCode = "403"),
        @ApiResponse(description = "NOT FOUND",
responseCode = "404")
    }
)
BotResponseDTO getById(@PathVariable @Positive Long id);

@Operation(summary = "Получение всех ботов по идентификатору
парка", method = "GET",
    responses = {
        @ApiResponse(description = "OK", responseCode
= "200"),
        @ApiResponse(description = "BAD REQUEST",
responseCode = "400"),
        @ApiResponse(description = "UNAUTHORIZED",
responseCode = "401"),
        @ApiResponse(description = "FORBIDDEN",
responseCode = "403"),
        @ApiResponse(description = "NOT FOUND",
responseCode = "404")
    }
)

Page<BotResponseDTO> getAll(@RequestParam @Positive Long
parkId,
                           @ParameterObject Pageable
pageable);
}

```

Листинг Г.3 – Файл BotController.java

```

@Slf4j
@RestController
@RequiredArgsConstructor
public class BotController implements BotAPI {

    private final BotService botService;

    @Override
    ResponseEntity<BotResponseDTO> create(BotRequestDTO botRqDto)
    {
        log.info("POST:/bots : botRqDto={}", botRqDto);
    }
}

```

```

        var botRsDto = botService.create(botRqDto);
        var url = ServletUriComponentsBuilder.fromCurrentRequest()
            .path("/{id}")
            .buildAndExpand(botRsDto.getId())
            .toUri();
        return ResponseEntity.created(url).body(botRsDto);
    }

    @Override
    BotResponseDTO getById(Long id) {
        log.info("GET:/bots/{id} : id={}", id);
        return botService.getById(id);
    }

    @Override
    Page<BotResponseDTO> getAll(Long parkId, Pageable pageable) {
        log.info("GET:/bots/from-park/{parkId} : parkId={},
pageable={}", parkId, pageable);
        return botService.getAll(parkId, pageable);
    }
}

```

Листинг Г.4 – Файл BotService.java

```

interface BotService {

    -----
    //REST-----

    /**
     * Создание записи о боте
     *
     * @param botRqDTO сущность бота с данными для сохранения
     * @return сохраненная DTO бота
     */
    BotResponseDTO create(BotRequestDTO botRqDTO);

    /**
     * Получение записи о боте по идентификатору
     *
     * @param id идентификатор бота
     * @return DTO бота
     */
    BotResponseDTO getById(Long id);

    /**
     * Получение всех записей о ботах по идентификатору парка
     *
     * @param parkId идентификатор парка

```

```

    * @param pageable параметры страницы
    * @return страница с DTO ботов
    */
    Page<BotResponseDTO> getAll(Long parkId, Pageable pageable);

    /**
     * Поиск ботов, подходящих под параметры доставки
     *
     * @param filter модель поиска
     * @return список подходящих ботов
     */
    List<Bot> getAll(BotFilter filter);
}

```

Листинг Г.5 – Файл BotExpressMain.java

```

@SpringBootApplication
class BotExpressMain {
    static void main(String[] args) {
        SpringApplication.run(BotExpressMain.class);
    }
}

```

Листинг Г.6 – Файл DatabaseContainer.java

```

class DatabaseContainer extends
PostgreSQLContainer<DatabaseContainer> {

    static final String IMAGE_VERSION = "postgres:12.7";
    static final String DB_URL = "DB_URL";
    static final String DB_USER = "DB_USER";
    static final String DB_PASSWORD = "DB_PASSWORD";
    static final String DB_DRIVER = "DB_DRIVER";

    static final DatabaseContainer INSTANT = new
DatabaseContainer();

    DatabaseContainer() {
        super(IMAGE_VERSION);
    }

    static DatabaseContainer getInstance() {
        return INSTANT;
    }

    @Override
    void start() {
        //тестовый контейнер сам генерирует и заполняет значения
        внешних переменных
        super.start();
    }
}

```

```

        System.setProperty(DB_URL, INSTANT.getJdbcUrl());
        System.setProperty(DB_USER, INSTANT.getUsername());
        System.setProperty(DB_PASSWORD, INSTANT.getPassword());
        System.setProperty(DB_DRIVER,
INSTANT.getDriverClassName());
    }

    @Override
    void stop() {
    }
}

```

Листинг Г.7 – Файл BotAPITest.java

```

@Slf4j
@Testcontainers
@ActiveProfiles("test")
@WithUserDetails(value = "admin")
@AutoConfigureMockMvc(addFilters = false)
@AutoConfigureTestDatabase(replace =
AutoConfigureTestDatabase.Replace.NONE)
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
class BotAPITest {

    static final Integer API_VERSION = 1;
    static final String COMMON_API_URI = "/bots";

    @Container
    static PostgreSQLContainer<DatabaseContainer> container =
DatabaseContainer.getInstance();

    @Autowired
    MockMvc mockMvc;
    @Autowired
    BotDataset botDataset;
    @Autowired
    BotModelRepository modelRepository;
    @Autowired
    ParkRepository parkRepository;
    @Autowired
    BotRepository botRepository;

    @BeforeEach
    void setUp() {
        botDataset.createData();
    }

    @AfterEach
    void tearDown() {
    }
}

```

```

        botDataset.removeData();
    }

    /**
     * Метод получения сущности модели бота
     */
    @Transactional(readOnly = true)
    BotModel getAnyModel() {
        return modelRepository.findAll().stream()
            .findAny()
            .orElseThrow(EntityNotFoundException::new);
    }

    /**
     * Метод получения сущности парка
     */
    @Transactional(readOnly = true)
    Park getAnyPark() {
        return parkRepository.findAll().stream()
            .findAny()
            .orElseThrow(EntityNotFoundException::new);
    }

    /**
     * Метод получения сущности бота
     */
    @Transactional(readOnly = true)
    Bot getAnyBot() {
        return botRepository.findAll().stream()
            .findAny()
            .orElseThrow(EntityNotFoundException::new);
    }

```