

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

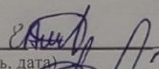
Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального
назначения»
Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

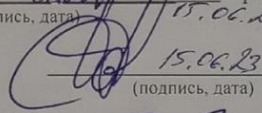
на тему

ПРИЛОЖЕНИЕ ДЛЯ ОБРАБОТКИ ОНЛАЙН-ЗАКАЗОВ
НА БАЗЕ ОБЛАЧНЫХ СЕРВИСОВ

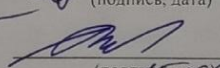
Студент

 Ерина Алина Германовна
(подпись, дата) 15.06.2023 (ФИО полностью)

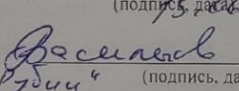
Руководитель

 15.06.23 Федюнин Р.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

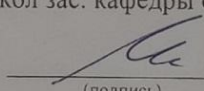
 15.06.23 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент

з.п.с.с.ч.  15.06.23 Васильков А.В.
(должность, место работы) АО «НП «Рубин» (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 15.06.2023 г. № 14)

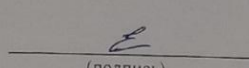
Заведующий кафедрой

 Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично

(протокол заседания ГЭК от 19.06.2023 № 3)

Секретарь ГЭК

 Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2023

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

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»
Заведующий кафедрой
Митрохин М.А.
«26» апреля 2023 г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

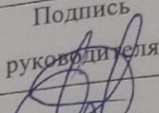
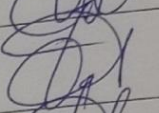
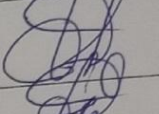
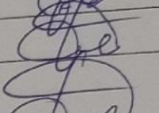
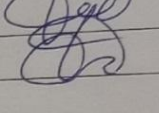
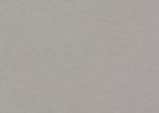
1. Студент Ерина Алина Германовна гр. 18ВОЭ1 факультета вычислительной техники.
2. Тема работы: «Приложение для обработки онлайн-заказов на базе облачных сервисов».
- Тема утверждена приказом ПГУ № 438/0 от 26.04.2023 г.
3. Руководитель работы к.т.н., доцент каф. ВТ «ПГУ» Федюнин Р.Н.
4. Задание на работу (назначение разработки, исходные данные и т.п.).

Назначение разработки – Программа предназначена для сбора, хранения и обработки заказов на ремонт компьютерной техники. В программе необходимо предусмотреть возможность многопользовательского интерфейса, для работы исполнителя и заказчика ремонтных работ. Возможность выполнения операций с записями по схеме CRUD. Программа должна быть реализована таким образом, чтобы иметь возможность работы в облачной экосистеме AWS.

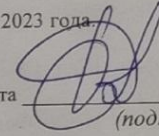
5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметной области, определить функциональный состав программы, выбрать инструменты реализации программы и развертывания приложения в облачной среде исполнения. Необходимо разработать алгоритм работы программы, разработать UML диаграммы с последующим написанием программы.

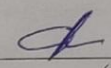
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	26.04.2023 - 28.04.2023	
2.	Выбор инструментов для разработки программы	10	29.04.2023 - 30.04.2023	
3.	Проектирование и составление UML диаграмм для реализации программы	20	31.04.2023 - 03.05.2023	
4.	Реализация базы данных	10	04.05.2023 - 08.05.2023	
5.	Реализация программы	10	09.05.2023 - 13.05.2023	
6.	Развертывание приложения в облачной среде и отладка	10	16.05.2023 - 23.05.2023	
7.	Оформление выпускной работы	30	24.05.2023 - 02.06.2023	

Дата выдачи задания « » 2023 года

Руководитель дипломного проекта  Федюнин Р.Н.
(подпись, дата)

Задание к исполнению принял студент  Ерина А.Г.
(подпись, дата)

Работу к защите допустить
Декан факультета  19.06.2023 Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 108 лист., 42 рис., 21 табл., 6 ист., 3 прилож.

ОБЛАЧНОЕ ХРАНИЛИЩЕ, ПАТТЕРН MVC, MySQL, CRUD, WEB СЕРВИС

Цель работы – разработать многопользовательское программное обеспечение для формирования заказов на ремонт вычислительной техники с последующей их обработкой.

В ходе выполнения ВКР было разработано:

Программное приложение для обработки заказов на основе архитектуры облачных сервисов в соответствии с техническим заданием на дипломное проектирование.

В результате ВКР разработано клиент-серверное приложение, полностью соответствующее поставленной задаче, а именно:

- разработано приложение, которое размещено в облачном хранилище, обрабатывает и хранит информацию о заказах на ремонт компьютерной техники;
- разработана СУБД для хранения информации о заказах и пользователях, которые сделали заказ;
- клиентская часть представлена в виде «тонкого» клиента.

В пояснительной записке приведен анализ предметной области, процесс разработки программного обеспечения, обзор технологии разработки и отладки программы.

Графическая часть содержит диаграммы, полностью описывающие представление СУБД и реализованного программного обеспечения.

					ПГУ 1.090501.03.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Приложение для обработки онлайн-заказов на базе облачных сервисов Пояснительная записка	Лит.	Лист	Листов	
Разраб.	Ерина А.Г.						2	108	
Провер.	Федюнин Р.Н.								
Н. Контр.	Бычков А.С.								
Утверд.	Митрохин М.А.								
					ФВТ, каф. ВТ гр. 18ВОЭ1				

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Предпроектный анализ предметной области	8
1.1 Анализ предметной области	8
1.2 Техническое задание	12
1.2.1 Назначение разработки	12
1.2.2 Требования к функциональным характеристикам	12
1.2.3 Требования к надежности системы.....	13
1.2.4 Требования к составу и параметрам технических средств.....	14
1.2.5 Требования к информационной и программной совместимости	14
1.3 Стадии и этапы разработки.....	14
1.4 Анализ и выбор инструментов для реализации проекта	14
1.4.1 Выбор способа реализации среды для функционирования приложения.....	15
1.4.2 Сравнительный анализ архитектуры реализации серверной части приложения.....	16
1.4.3 Сравнительный анализ и выбор языка программирования для реализации приложения	21
1.4.4 Сравнительный анализ и выбор СУБД.....	23
1.5 Проектирование разрабатываемого приложения	23
1.6 Выводы по первой главе	29
2 Реализация программного приложения для обработки заказов	30
2.1 Проектирование и реализация базы данных	30
2.2 Проектирование и реализация серверной части приложения.....	42
2.3 Проектирование пользовательского интерфейса	48
3 Технология размещения программного приложения для обработки заказов в среде исполнения	60
4 Техничко-экономическое обоснование разработки программы.....	67

5 Безопасность жизнедеятельности. Оценка травмобезопасности на рабочем месте оператора ПК	72
Заключение	76
Список использованных источников	77
Приложение А – UML-диаграммы.....	78
Приложение А.1 – Диаграмма вариантов использования	79
Приложение А.2 – Диаграмма развертывания.....	80
Приложение А.3 – Диаграмма последовательности	81
Приложение А.4 – Диаграмма развертывания.....	82
Приложение А.5 – Диаграмма данных	83
Приложение Б – Листинг программы	84
Приложение В – Презентация.....	95

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ПЗ – пояснительная записка

ПИ – пользовательский интерфейс

ПО – программное обеспечение

ПЭВМ – персональная электронно-вычислительная машина

СУБД – система управления базами данных

API (Application Programming Interface) – программный интерфейс приложения

AWS (Amazon Web Services) – облачное хранилище и исполнительная среда
Amazon

MVC (Model-View-Controller) – паттерн проектирования Модель-Вид-Контроллер

UML (Unified Modeling Language) – унифицированный язык моделирования

					ПГУ 1.090501.03.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Сайты в настоящий момент являются самым популярным способом получения информации и оказания онлайн-услуг.

Разработка и использование web-приложений в десктопном или мобильном варианте до сих пор является самым популярным вариантом реализации сайта (интернет-ресурса).

С точки зрения разработчиков и инженеров классический вариант реализации и поддержания web-приложений в рабочем состоянии заключается в масштабировании кода и инфраструктуры, находящихся в зоне ответственности команды.

Для сайтов, использующих архитектуру стека LAMP, задача «горизонтального» и/или «вертикального» масштабирования программы или аппаратуры приводит часто к смене стека технологий, а иногда и к смене архитектуры приложения.

Как следствие масштабирования программы – переход от монолитного ядра к микросервисной архитектуре или сохранение монолитного ядра с расширенным и измененным функционалом, что влечет увеличение сроков реализации конечного программного продукта. Результатом такого подхода является потеря заказчиком инвестиционных средств в IT-проекте.

Избежать описанных проблем можно, используя облачную архитектуру приложения.

Облачная архитектура реализована таким образом, что команда разработки акцентирует внимание на реализации программной части приложения, а аппаратную часть инфраструктуры приложения, СУБД и даже архитектуру проекта предлагает как услугу оператор (Google, Amazon, Microsoft), который берет на себя ответственность за работоспособность и масштабирование «экосистемы» проекта.

Данный подход позволяет не брать в команду сетевых инженеров и администраторов, ограничивая команду наличием программистов и специалистов по отладке программного обеспечения.

					ПГУ 1.090501.03.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Данная дипломная работа посвящена разработке классического интернет-ресурса на базе паттерна MVC, реализованного на базе облачной инфраструктуры.

В дипломном проекте будет использован следующий стек технологий:

- инфраструктура для развертывания и функционирования приложения – облачный сервис Amazon AWS;
- серверная часть логики приложения реализована с помощью фреймворка языка Java;
- фронтенд приложения реализован с помощью JavaScript-библиотеки с открытым исходным кодом для разработки пользовательских интерфейсов.

Дипломная работа состоит из пяти глав.

В первой главе представлен анализ предметной области и анализ стека технологий. С помощью аналитики и схем показана основа реализации программы обработки заказов в режиме онлайн.

Во второй главе с помощью UML-диаграмм и элементов программного кода описана реализация приложения и протоколов взаимодействия модулей приложения в рамках паттерна MVC.

В третьей главе описана технология отладки программы клиента и облачного сервера.

В четвертой главе описана экономическая целесообразность реализации проекта.

В пятой главе описаны элементы безопасности для программиста при использовании персонального компьютера.

					ПГУ 1.090501.03.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Предпроектный анализ предметной области

1.1 Анализ предметной области

По состоянию на январь 2023 года в мире насчитывается 5,16 миллиардов пользователей Интернета [1], что составляет 64,4 процента мирового населения. Из них 4,76 миллиарда человек, или 59,4 процентов населения земного шара, являются пользователями социальных сетей и различных интернет-ресурсов. Это показывает, насколько велик и перспективен рынок онлайн с точки зрения реализации WEB-приложений для обработки услуг в сети интернет. Анализ источников [2] показывает стабильный рост обработки заказов в режиме онлайн на протяжении последних 10 лет.

В настоящий момент обработка заказов производится с помощью специального программного обеспечения – систем управления заказами (Order Management System, или OMS).

Системы OMS в классической реализации являются сторонними платформами между клиентами и исполнителями для обработки заказов в режиме онлайн. С технической точки зрения OMS реализуются в виде WEB-портала и/или мобильного приложения, реализованного в рамках трехзвенной архитектуры клиент-сервер-СУБД (стек LAMP).

Классическими примерами OMS можно считать решения 1С Предприятие, Generix OMS, АИС Парус в «коробочном» или облачном варианте.

Плюсы и минусы данных платформ приведены в таблице 1.1.

					ПГУ 1.090501.03.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1.1 – Сравнительный анализ аналогов реализуемой программы

Наименование	Плюс решения	Минус решения
1С Предприятие	<ul style="list-style-type: none"> - возможность изменения стандартной конфигурации под текущие потребности; - технологическое сопровождение; - ежемесячные обновления; - российская разработка; - стандартная возможность формирования отчетов. 	<ul style="list-style-type: none"> - для доработки решения из «коробки» нужен специалист среднего уровня и выше; - отсутствие инструментария работы с графиками; - отсутствие кроссплатформенной реализации; - очень специфичный язык реализации; - специфичная СУБД.
Generix OMS	<ul style="list-style-type: none"> - возможность разграничения доступа пользователей; - шифрование данных; - российская разработка; - не требует установки на компьютер пользователя. 	<ul style="list-style-type: none"> - существует только «облачное» решение; - для работы требуется устойчивый интернет; - оптимален для малых и средних объемов заказов; - не совершенная техническая поддержка; - отсутствует конструктор отчетов.
АИС Парус	<ul style="list-style-type: none"> - используется только в государственном секторе; - доступна интеграция с иными OMS; - высокий уровень технической поддержки. 	<ul style="list-style-type: none"> - право проводить модификацию модулей конкретного предприятия могут только разработчики АИС Парус; - стоимость программы на порядок выше аналогов; - отсутствует «тонкий» клиент.

В связи с тем, что разработка и использование систем OMS [3] для обработки заказов на ремонт вычислительной техники является перспективным направлением, было принято решение реализовать в рамках дипломного проекта программу для сбора и обработки заказов на ремонт

вычислительной техники, с учетом недостатков популярных аналогов (таблица 1.1).

Так как для разработчиков программного обеспечения содержание хостинга и инфраструктуры разрабатываемого программного обеспечения не является прямой обязанностью, содержание серверной части приложения и СУБД возлагают на специальных провайдеров, для которых создание и обслуживание облачной и серверной экосистемы является прямой обязанностью. Данный подход позволяет иметь качественную техническую поддержку инфраструктуры проекта в режиме 24/7/365, при отсутствии собственных DevOps инженеров в команде разработки.

Исходя из выше изложенного, дипломный проект реализован в рамках облачной платформы.

Проведенный анализ недостатков существующих OMS привел к решению создать программное приложение, главной целью которого является обработка заказов клиента в режиме «онлайн».

Программное приложение для обработки заказов на основе архитектуры облачных сервисов дает возможность пользователю-исполнителю (рисунок 1.1) иметь программный инструмент для приема и обработки заказов, как облачную услугу. В случае прекращения контракта на услуги с провайдером облачных услуг пользователь-исполнитель имеет возможность развернуть свой сервис в другом облачном сервисе. Для этого исполнитель должен иметь резервную копию приложения и базы данных.

Развернутое в облачной системе приложение дает возможность исполнителю (рисунок 1.1) иметь доступ к клиентской базе, просматривать список поступивших заказов с возможностью обработки заказа.

Обработка заказа исполнителем включает в себя возможность: принять заказ от пользователя-заказчика, редактировать заказ. Редактирование заказа является составной функцией и включает в себя возможность добавить или удалить детали для ремонта.

					ПГУ 1.090501.03.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Исполнитель имеет право изменить статус заказа, сделав его принятым в работу или выполненным. До того как заказ принят, статус заказа – в очереди.

Все вышеописанные функции пользователь-исполнитель имеет право выполнять только в авторизованном виде. Для этого в программе предусмотрено две функции: регистрация и авторизация пользователя. Авторизация пользователя это функции логин и логгаут соответственно.

Пользователь-заказчик (рисунок 1.2) только после регистрации в системе имеет право на авторизацию и доступ к функциям заказа: создание, редактирование и удаление. Создание заказа является составной функцией, которая включает выбор возможных работ по ремонту и выбор исполнителей. В системе предусмотрена возможность поиска исполнителей по Ф.И.О.

В реализованной программе могут работать следующие категории пользователей: исполнитель заказов (рисунок 1.1) и заказчик ремонта вычислительной техники (рисунок 1.2).

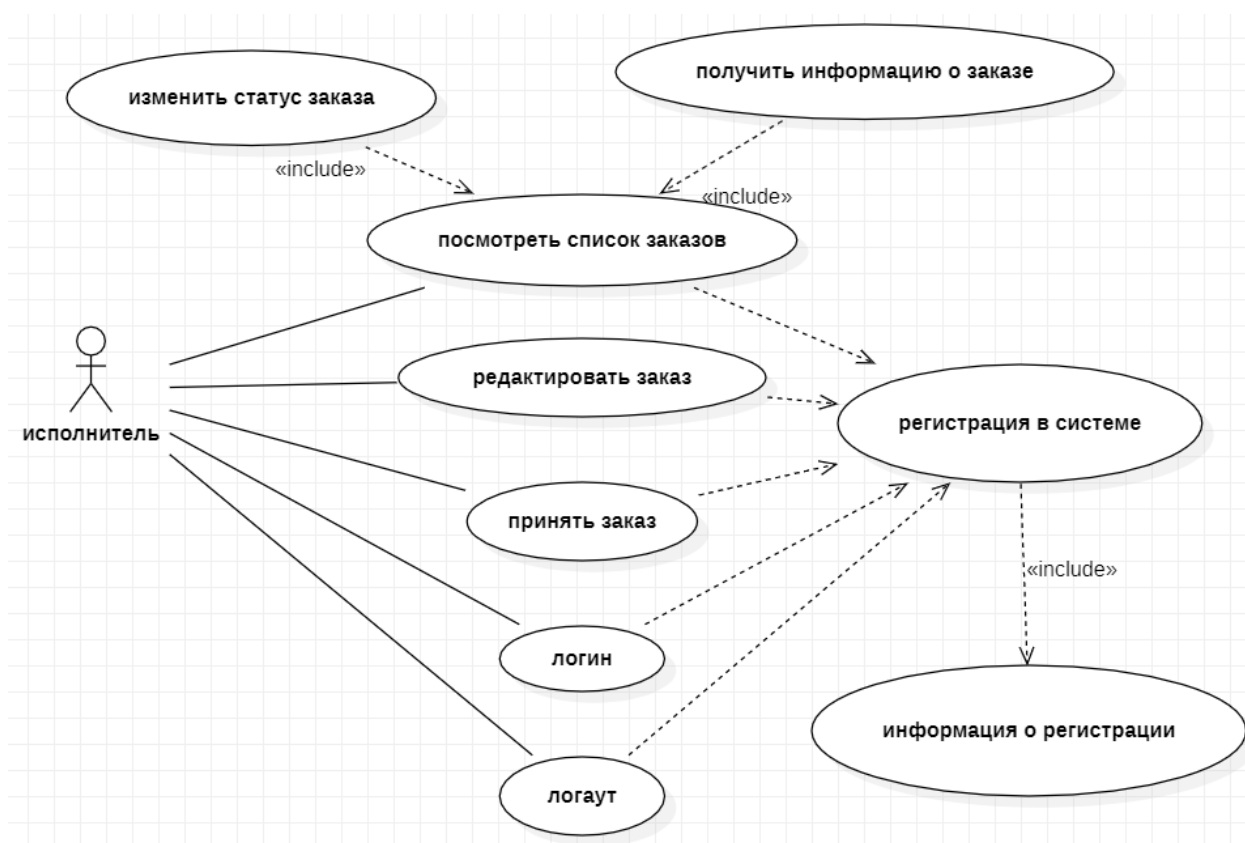


Рисунок 1.1 – Диаграмма вариантов использования для исполнителя заказов

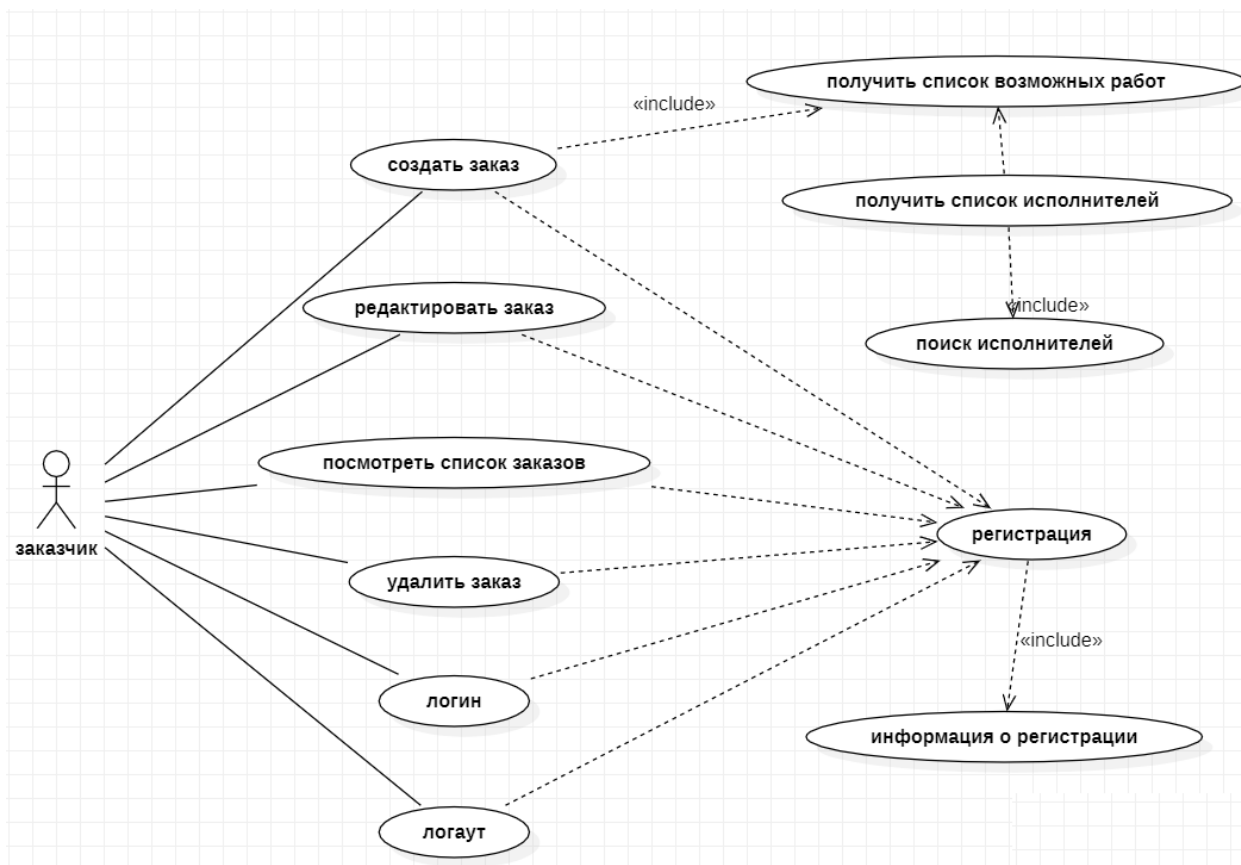


Рисунок 1.2 – Диаграмма вариантов использования для заказчика ремонта вычислительной техники

1.2 Техническое задание

1.2.1 Назначение разработки

Приложение для обработки онлайн-заказов на базе облачных сервисов предназначено для работы двух категорий пользователей: исполнитель и заказчик ремонта вычислительной техники. Программа предназначена для размещения заказчиком заказа на ремонт ВТ с последующей обработкой заказа исполнителем.

Программа имеет трехзвенную архитектуру, реализованную на базе паттерна MVC с помощью фреймворков и библиотек для реализации современного web-приложения.

1.2.2 Требования к функциональным характеристикам

Программное обеспечение со стороны исполнителя заказа должно обеспечивать выполнение следующих функций (рисунок 1.1):

- регистрация в системе;
- функция «логин/логаут»;
- функция принятия заказа на исполнение;
- функция удаления заказа;
- функция редактирования заказа;
- функция просмотра списка заказов и их статусов.

Программное обеспечение со стороны заказчика должно обеспечивать выполнение следующих функций (рисунок 1.2):

- регистрация в системе;
- функция «логин/логаут»;
- функция создания заказа, с возможностью детализации списка работ и выбором исполнителя;
- функция поиска по требуемым критериям;
- функция удаления заказа;
- функция редактирования созданных данным пользователем заказов;
- функция просмотра списка своих заказов.

Из вышеизложенного списка функциональных характеристик входными данными программы являются:

- сведения о видах и типах техники;
- сведения о заказе;
- сведения о пользователе.

Выходными данными являются кортежи СУБД, обработанные логическим уровнем приложения для визуализации на уровне интерфейса приложения.

1.2.3 Требования к надежности системы

При функционировании системы должен осуществляться контроль входных данных.

В случае возникновения ошибки приложение должно формировать соответствующие сообщения и предоставлять пользователю возможность исправления ошибок для продолжения работы.

					ПГУ 1.090501.03.001 ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

1.2.4 Требования к составу и параметрам технических средств

Система должна быть предназначена для работы в виде «тонкого» клиента на x86 совместимых персональных компьютерах со следующей минимальной конфигурацией: процессор не ниже Intel Core 3, оперативная память не ниже 8 Гб, жесткий диск 500 Гб, монитор 24 дюйма.

1.2.5 Требования к информационной и программной совместимости

Приложение для обработки онлайн-заказов на базе облачных сервисов работает на базе взаимодействия «тонкого клиента» и облачной платформы.

1.3 Стадии и этапы разработки

Стадии и этапы разработки приложения для обработки онлайн-заказов на базе облачных сервисов описаны в таблице 1.2.

Таблица 1.2 – Стадии и этапы разработки ПО

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения
1	Анализ предметной области	10%	
2	Составление технического задания	4%	
3	Выбор инструментов разработки	6%	
4	Проектирование системы	14%	
5	Разработка программы	30%	
6	Разработка СУБД	20%	
7	Выполнение задания по БЖД	4%	
8	Выполнение задания по экономике	4%	
9	Оформление пояснительной записки	8%	

1.4 Анализ и выбор инструментов для реализации проекта

Одним из требований, которому должно удовлетворять разработанное приложение – кроссплатформенность.

Кроссплатформенность – способность ПО работать с разными аппаратными платформами и операционными системами.

					ПГУ 1.090501.03.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

Кроссплатформенность обеспечивается использованием высокоуровневых языков программирования, поддерживающих условную компиляцию, компоновку и выполнение кода для различных платформ.

1.4.1 Выбор способа реализации среды для функционирования приложения

Существует несколько способов организации экосистемы кроссплатформенного приложения, начиная от традиционного подхода, заканчивая облачным подходом (рисунок 1.3).

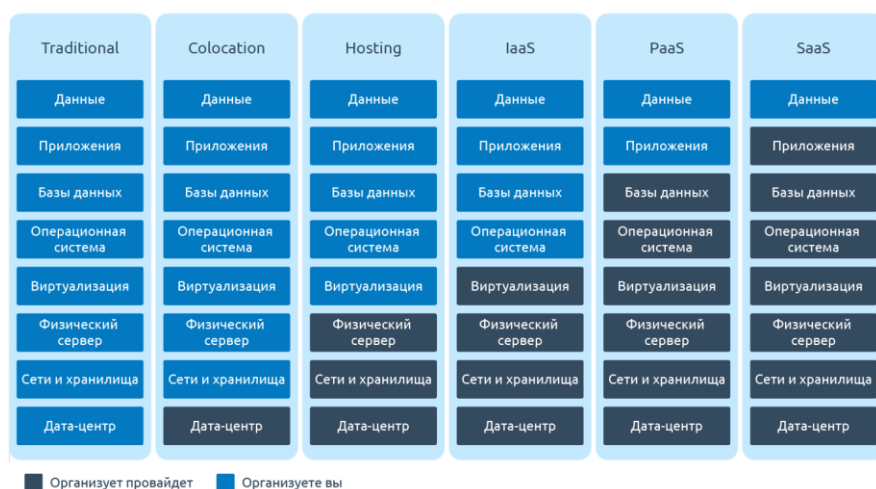


Рисунок 1.3 – Способы организации экосистемы приложения

В основном команды разработчиков используют или традиционную архитектуру (используется в государственных организациях), или способ hosting (используется в крупных коммерческих организациях). В обоих случаях команда разработчиков усилена командой DevOps инженеров для настройки и поддержания системы функционирования и настройки работы готового программного обеспечения.

Если команде разработки ПО требуется направить усилия на реализацию проекта без поддержки и настройки экосистемы приложения, требуется выбрать способ организации экосистемы приложения – PaaS, платформа как сервис. Способ PaaS предоставляет инфраструктуру и подготовленный для разработки приложений программный инструментарий с дополнительной технической поддержкой.

В данном дипломном проекте за основу реализации выбрана PaaS архитектура, на базе облачной платформы.

1.4.2 Сравнительный анализ архитектуры реализации серверной части приложения

Реализуемое web-приложение состоит из серверной части, СУБД и пользовательской части (frontend). Ниже представлены три архитектурных варианта реализации приложения, с указанием достоинств и недостатков каждого из них.

Монолитная архитектура (рисунок 1.4) – модель программного обеспечения, в которой функциональность приложения размещена в одном большом блоке (монолите), который связан с клиентской частью приложения. Данная архитектура является простой в реализации и понимании, но имеет проблемы масштабирования по мере роста приложения и невозможности быстрой сборки модулей приложения, что ограничивает работу с обновлениями функционала ПО.

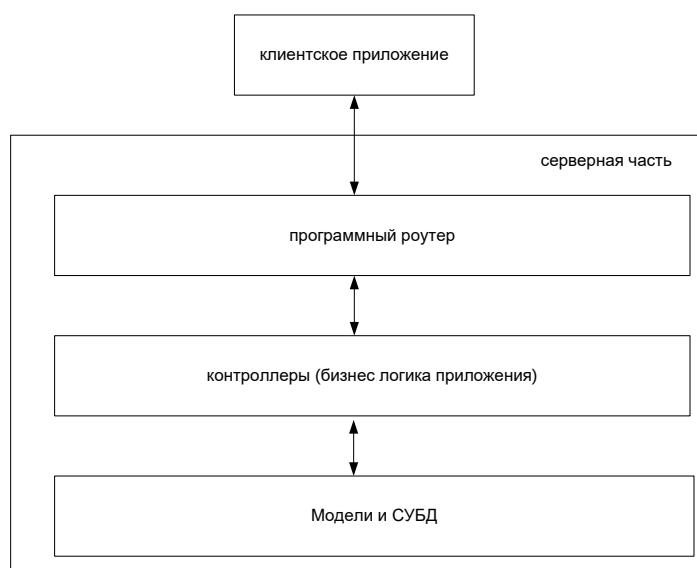


Рисунок 1.4 – Монолитная архитектура программного приложения

К преимуществам монолитной архитектуры можно отнести следующие особенности:

- простое развертывание. Использование одного исполняемого файла или каталога упрощает развертывание приложения;

- разработка. Приложение легче разрабатывать, когда оно создано с использованием одной базы кода;

- удобная отладка. Весь код находится в одном месте, благодаря чему становится легче выполнять запросы и находить ошибки.

К недостаткам монолитной архитектуры можно отнести следующие особенности:

- масштабируемость. Невозможно масштабировать отдельные компоненты без изменений всего проекта;

- надежность. Ошибка в одном модуле может повлиять на работоспособность всего приложения;

- развертывание. При внесении небольшого изменения требуется повторное развертывание всего приложения.

Микросервисная архитектура (рисунок 1.5) – архитектура, основанная на независимо развертываемых сервисах. У каждого сервиса есть собственная бизнес-логика и база данных. Обновление, тестирование, развертывание и масштабирование функционала выполняются внутри каждого сервиса независимо.

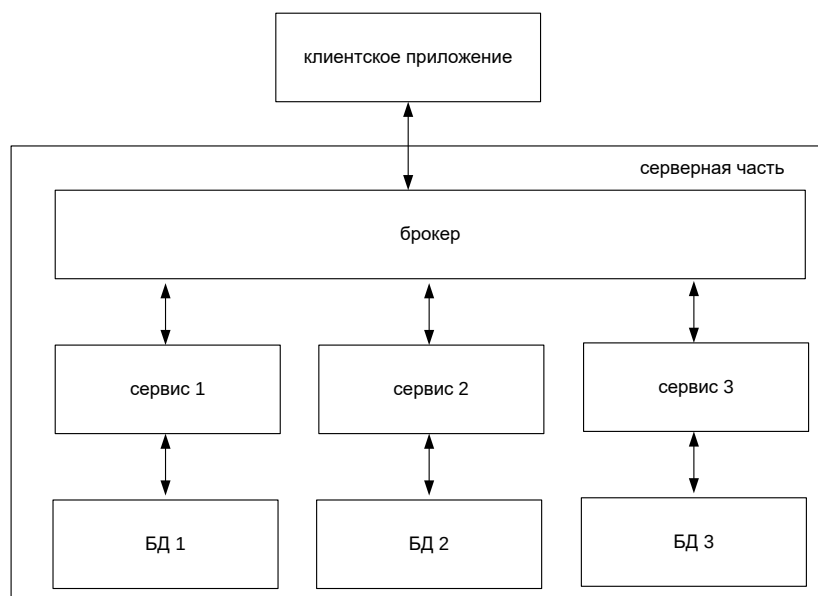


Рисунок 1.5 – Микросервисная архитектура программного приложения

Реализация приложения в виде микросервисной архитектуры дает следующие преимущества:

- гибкое масштабирование. Возможность развертывания новых экземпляров сервисов в кластере без повторной сборки и запуска всей системы;
- непрерывное развертывание. Наличие в инфраструктуре регулярных циклов релиза приложения без повторной сборки и запуска всей системы.

К недостаткам микросервисов можно отнести следующие пункты:

- экспоненциальный рост расходов на инфраструктуру. Каждый новый сервис имеет свой комплект тестов, инструкций по развертыванию, инфраструктуры, инструментов мониторинга, что ведет к увеличению стоимости расходов на содержание программного обеспечения в целом;
- проблемы при отладке. У каждого сервиса свой независимый набор функций, что делает процесс отладки уникальным для каждого сервиса.

Все вышеперечисленные недостатки устраняет использование облачной архитектуры (рисунок 1.6).

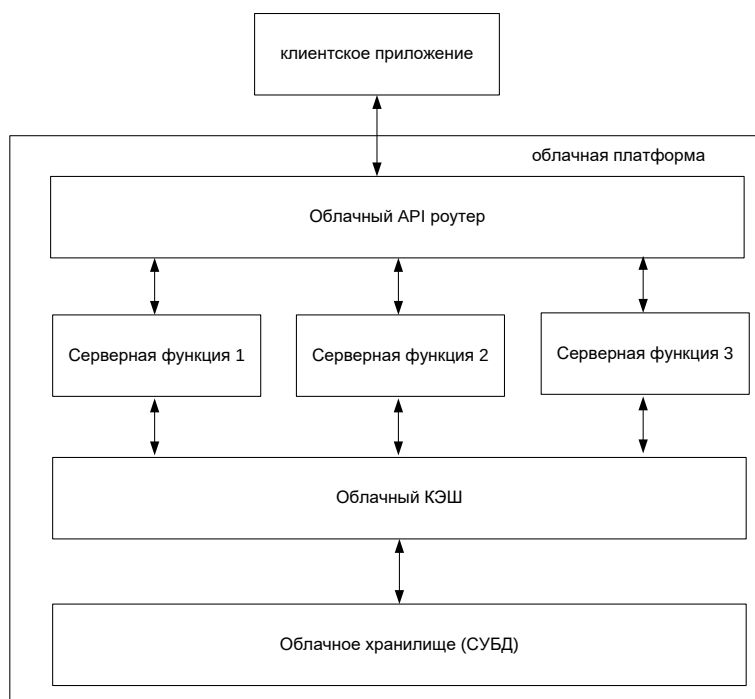


Рисунок 1.6 – Архитектура облачного приложения

На рисунке 1.6 показана архитектура облачного ПО, доступная разработчикам через специальных провайдеров облачных услуг: Google, Amazon, Microsoft и т.д.

При реализации данного проекта была задача найти облачную платформу, которая позволяет реализовать монолитное MVC приложение и поместить его для исполнения в рамках облачных ресурсов штатными инструментами провайдера облачных услуг.

Для этого был проведен анализ инструментов, предоставляемых провайдерами облачных услуг – таблица 1.3.

Таблица 1.3 – Сравнительный анализ провайдеров облачных услуг

Наименование функционала	Google Cloud	Amazon Web Services (AWS)	Microsoft Azure
Наличие встроенной реляционной облачной СУБД с возможностью импорта готовой СУБД	Нет	Имеется	Имеется
Инструменты автоматического развертывания монолитного приложения в облачной платформе	Нет	Имеется	Имеется
Виртуальные выделенные серверы в рамках облачной платформы	Имеется	Имеется	Имеется
Сервис собственных доменных имен (DNS)	Имеется	Имеется	Имеется
Поддержка языка Java с функцией распознавания WAR и JAR файлов	Имеется	Имеется	Нет

Полным набором инструментов с функционалом, представленном в таблице 1.3, обладает платформа Amazon Web Services в следующем исполнении:

- AWS Elastic Beanstalk – инструмент автоматического развертывания монолитного приложения в облачной платформе;
- Amazon RDS – группа реляционных облачных СУБД в составе Amazon Web Services;
- Amazon Elastic Compute Cloud – виртуальные выделенные серверы в рамках облачной платформы;
- Amazon Route 53 – сервис доменных имен (DNS).

Данный набор инструментов позволит в автоматическом режиме развернуть монолитное MVC приложение и сделать сайт приложения доступным для пользователей (рисунок 1.7).

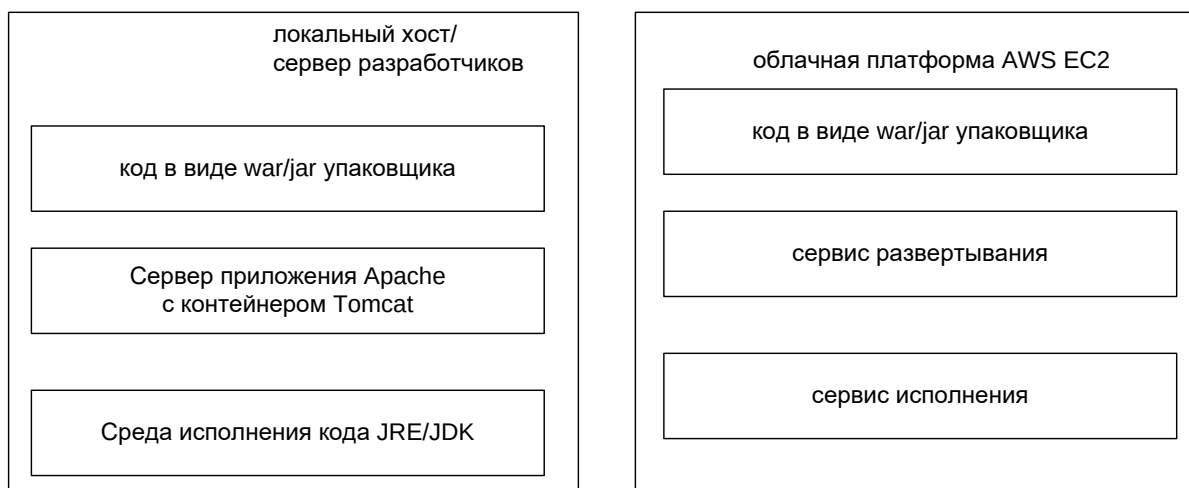


Рисунок 1.7 – Развертывание приложения на исполнение

Из описанного выше сделан вывод, что серверная часть программного кода приложения будет реализована в рамках AWS с настройкой AWS на развертывание программного решения с монолитным ядром.

Так как AWS поддерживает язык программирования Java, было принято решение выбрать язык Java как основу реализации приложения (рисунок 1.8).

1.4.3 Сравнительный анализ и выбор языка программирования для реализации приложения

На сайте сбора глобальной статистики [4] показано, что язык Java входит в тройку самых популярных языков разработки интернет-приложений.

Язык Java поддерживает фреймворки для реализации монолитных программ на базе паттерна MVC с интеграцией в AWS.

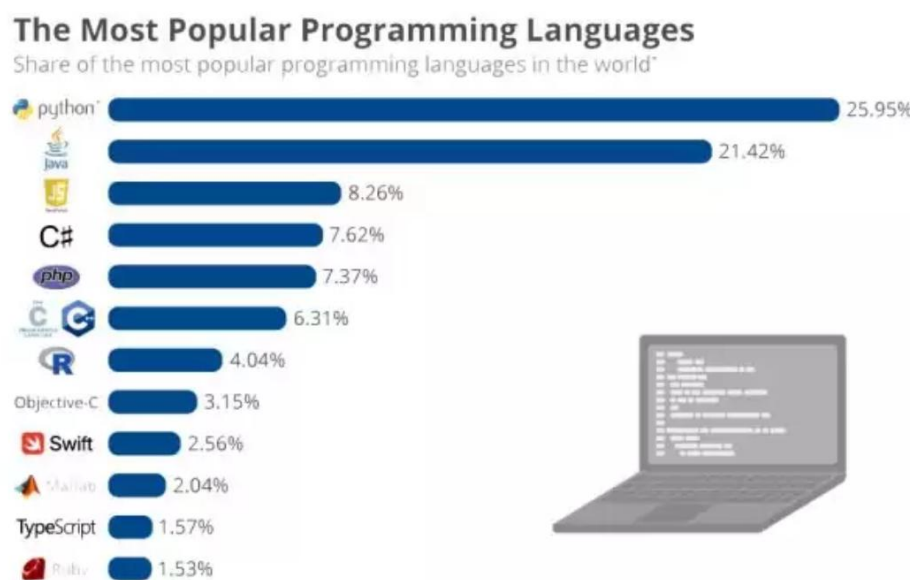


Рисунок 1.8 – Популярность языков программирования для реализации мобильных приложений

В таблице 1.4 представлен сравнительный анализ фреймворков языка Java, которые имеют интеграцию и поддержку с сервисами Amazon Web Services.

Таблица 1.4 – Сравнительный анализ фреймворков языка Java

Наименование фреймворка	Достоинства	Недостатки
Spring Boot	Наличие структурированной документации и технической поддержки AWS	Отсутствие инструментов тестирования приложения в облачной среде
	Наличие набора поддерживаемых технологий и интеграций с AWS	
	Наличие автоматической конфигурации для облачной платформы AWS	
Play Framework	Наличие инструментов для создания RESTful-API приложений	Отсутствие набора поддерживаемых интеграций, в частности с AWS
		Отсутствие прямой интеграции с СУБД
Vert.x	Наличие высокой производительности благодаря асинхронной обработке запросов	Отсутствие набора поддерживаемых интеграций, в частности с AWS
	Наличие поддержки нескольких языков программирования	Отсутствие прямой интеграции с СУБД
		Отсутствие структурированной и обширной документации и технической поддержки
Dropwizard	Наличие поддержки создания RESTful-API приложений	Отсутствие набора поддерживаемых интеграций, в частности с AWS
	Наличие интеграции с Hibernate, JDBI и другими ORM	Отсутствие автоматической конфигурации для облачных платформ

На основании анализа (таблица 1.4) выбран фреймворк Java Spring Boot для создания веб-приложения, так как он обладает инструментами интеграции с облачными платформами.

1.4.4 Сравнительный анализ и выбор СУБД

Существует два подхода к реализации хранения данных: реляционные СУБД и NoSQL. Так как облачная платформа AWS предъявляет критическое требование к транзакциям CRUD, то в реализованном приложении используется реляционная СУБД.

СУБД MySQL, Microsoft SQL Server и PostgreSQL являются самыми популярными реляционными СУБД в мире [5]. В таблице 1.5 представлен сравнительный анализ данных СУБД.

Таблица 1.5 – Сравнительный анализ реляционных СУБД

Наименование функционала	MySQL	Microsoft SQL Server	PostgreSQL
Производительность	Нет	Да	Да
Свободное программное обеспечение	Да	Нет	Да
Кроссплатформенность	Да	Нет	Да
Интеграция с AWS	Да	Нет	Да
Безопасность	Да	Да	Да

Так как основным критерием реализуемого приложения является кроссплатформенность и интеграция с облачной платформой AWS, в качестве хранилища выбрана СУБД MySQL.

1.5 Проектирование разрабатываемого приложения

Проектирование программного приложения сделано с помощью UML-диаграмм. UML-диаграммы – это нотации, используемые для визуального представления программных систем, такие как их структура, поведение и процессы.

Существуют аналоги UML-диаграмм, которые также используются для моделирования программно-инженерных систем: блок-схемы, диаграммы потоков данных, таблицы принятий решений, диаграммы состояний, диаграммы потоков работ, сетевые диаграммы. Каждый из этих способов имеет свои преимущества и недостатки. Однако UML-диаграммы являются наиболее распространенным и удобным методом, который используется в программной инженерии.

Преимуществами UML-диаграмм по сравнению с аналогами являются:

- UML-диаграммы являются стандартным методом документирования и коммуникации между разработчиками, аналитиками, тестерами и другими участниками проекта. Аналоги используются для внутреннего использования и ведения заметок;

- UML-диаграммы имеют более широкий спектр типов диаграмм, включая диаграммы классов, диаграммы взаимодействия, диаграммы состояний и многие другие, что позволяет моделировать различные аспекты системы. Аналоги ограничены использованием только в контексте процесса или части системы.

На рисунках 1.9 – 1.15 с помощью диаграмм последовательности показаны этапы функционирования программы.

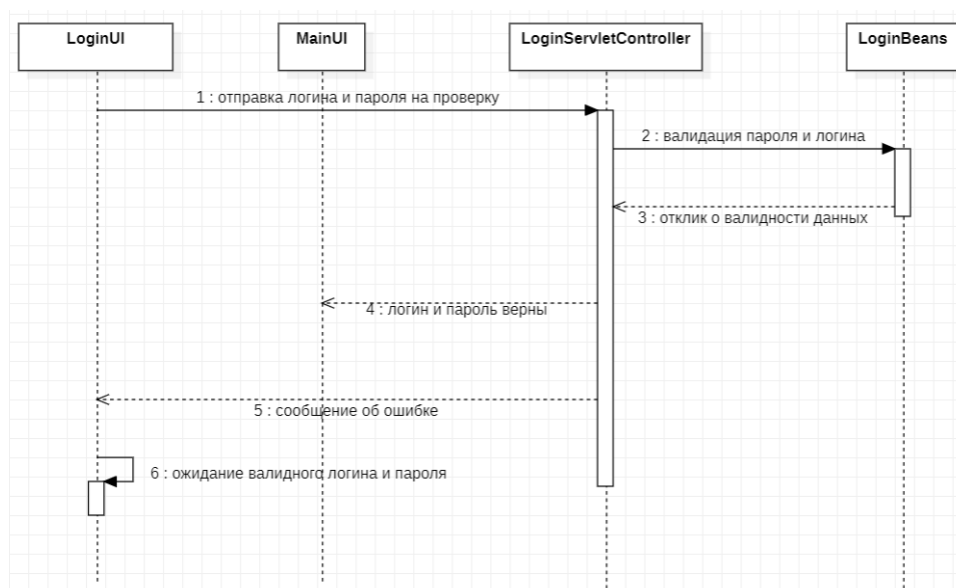


Рисунок 1.9 – Диаграмма последовательности входа в систему

Вход в систему заключается в выполнении следующих шагов:

- отображение страницы авторизации;
- ввод учетных данных пользователя;
- если данные верны, то переход на главную страницу;
- иначе вывод информации об ошибке.

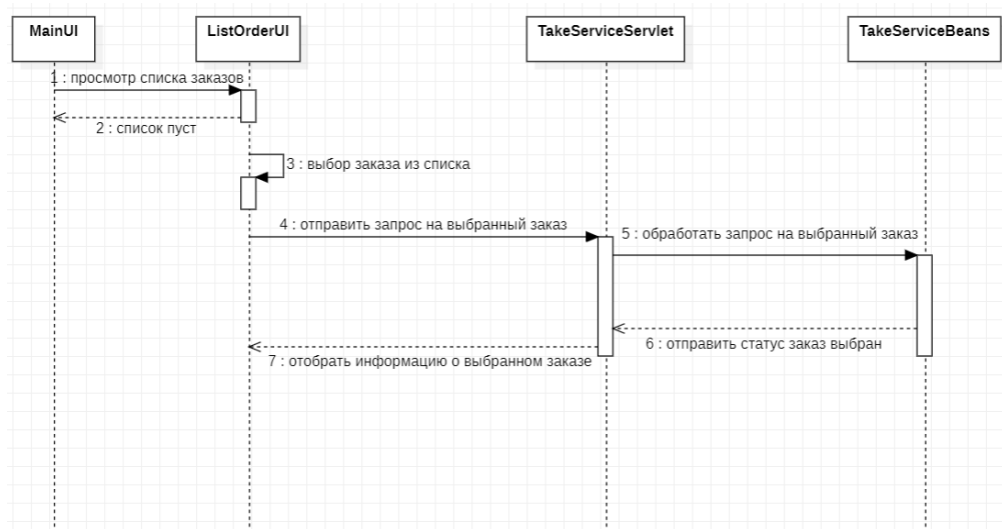


Рисунок 1.10 – Диаграмма последовательности принятия заказа на исполнение

Принятие заказа на исполнение заключается в выполнении следующих шагов:

- переход на страницу заказов;
- просмотр списка заказов, если он существует;
- выбор требуемого заказа;
- изменить статус заказа на выбранный;
- отобразить информацию о выбранном заказе.

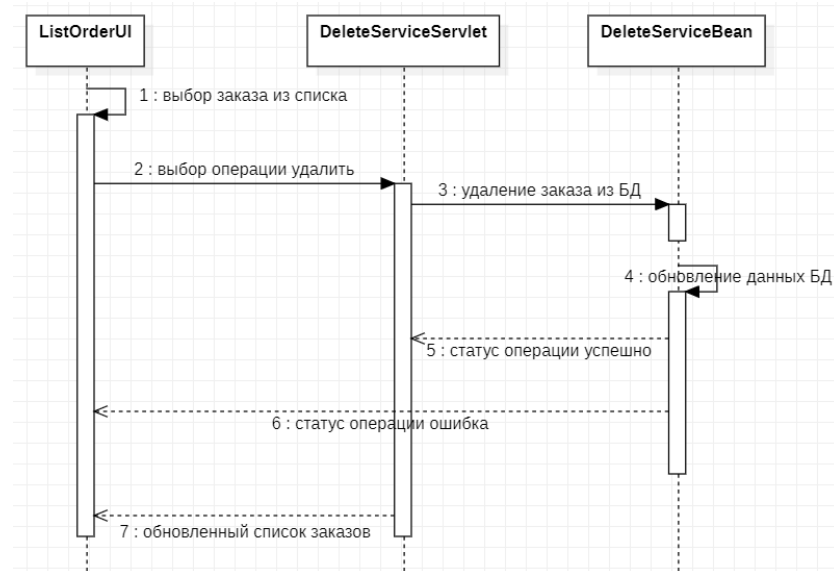


Рисунок 1.11 – Диаграмма последовательности удаления заказа

Удаление заказа заключается в выполнении следующих шагов:

- переход на страницу заказов;
- просмотр списка заказов, если он существует;
- выбор требуемого заказа;
- отправить запрос на удаление заказа;
- отобразить измененный список заказов.

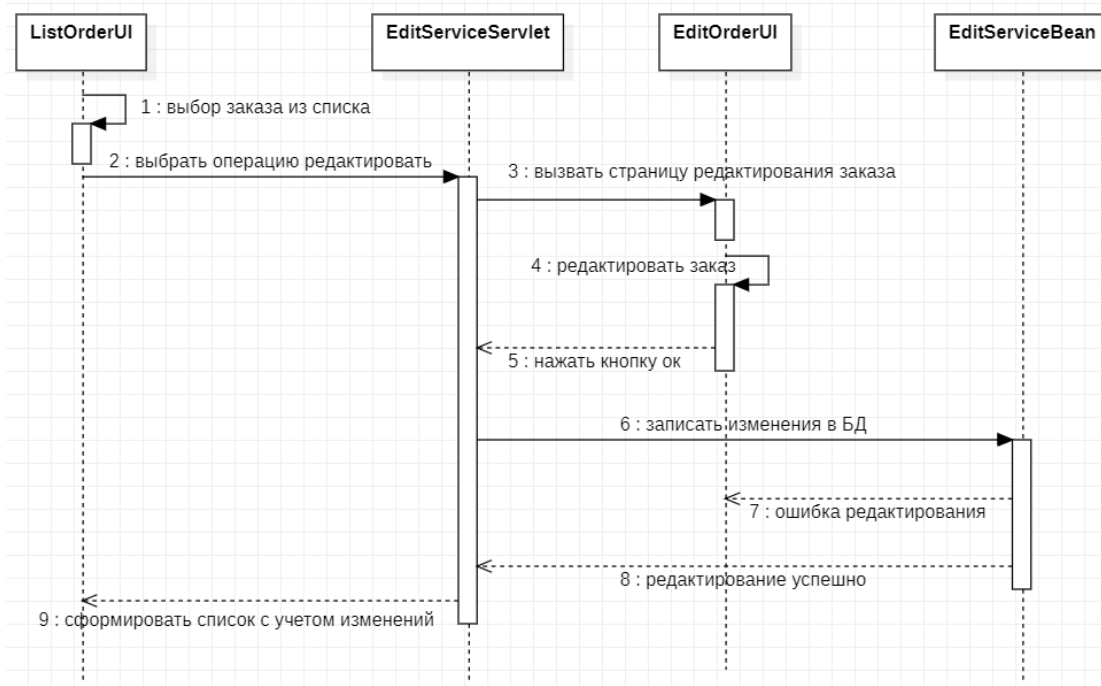


Рисунок 1.12 – Диаграмма последовательности редактирования заказа

Редактирование заказа заключается в выполнении следующих шагов:

- переход на страницу заказов;
- просмотр списка заказов, если он существует;
- выбор требуемого заказа;
- отправить запрос на редактирование;
- отобразить страницу редактирования заказа;
- изменение данных заказа;
- отправить запрос на сохранение изменений;
- сформировать и отобразить список заказов с учетом изменений.

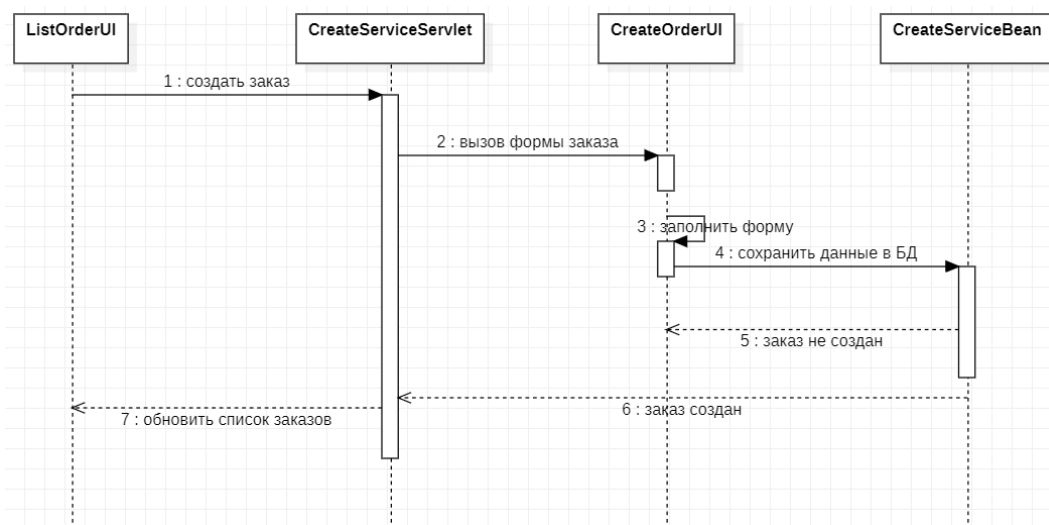


Рисунок 1.13 – Диаграмма последовательности создания заказа

Создание заказа заключается в выполнении следующих шагов:

- переход на страницу заказов;
- отправить запрос на создание нового заказа;
- отобразить страницу создания нового заказа;
- внесение данных нового заказа;
- отправить запрос на сохранение нового заказа в БД;
- сформировать и отобразить список заказов с учетом изменений.

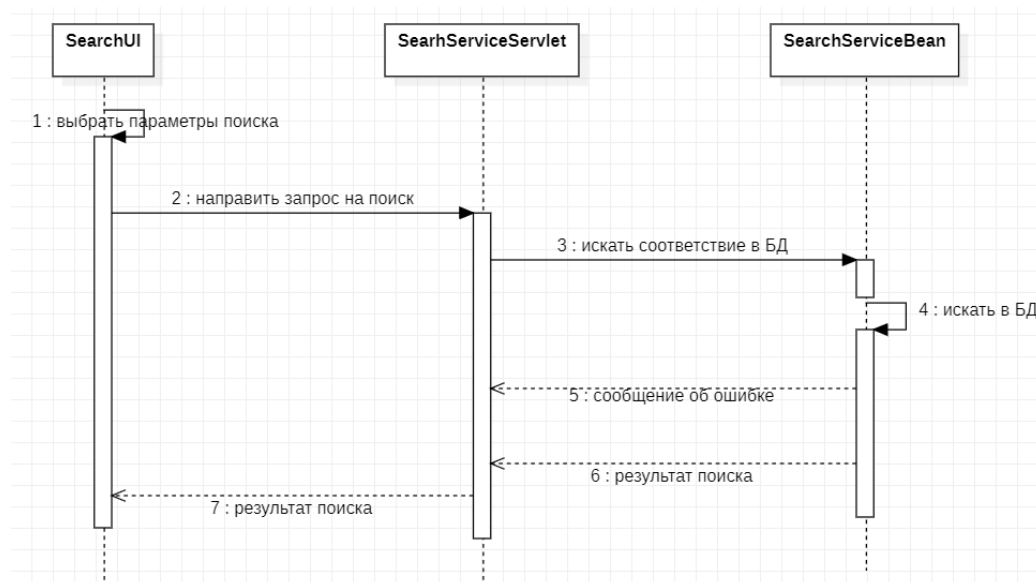


Рисунок 1.14 – Диаграмма последовательности поиска

Поиск заключается в выполнении следующих шагов:

- переход на страницу поиска;
- выбрать критерии поиска;
- осуществить поиск в БД;
- сформировать и отобразить результат поиска.

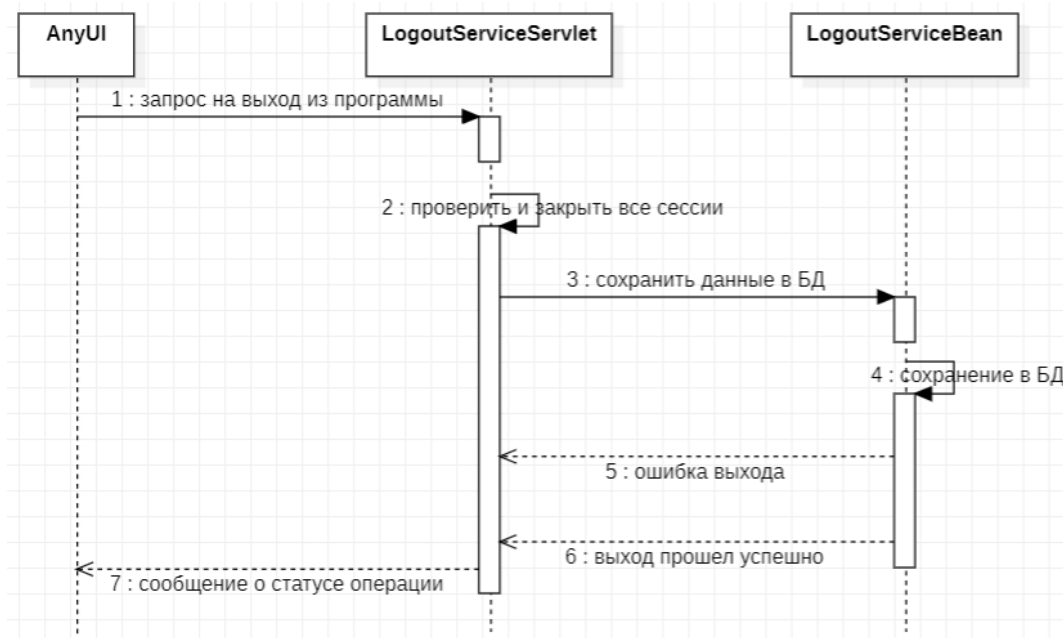


Рисунок 1.15 – Диаграмма последовательности выхода из программы

Выход из программы заключается в выполнении следующих шагов:

- отображение любой страницы программы;
- запрос пользователя на выход;
- сохранение текущих данных в БД;
- выход из программы.

1.6 Выводы по главе

В результате исследования в первой главе проведен анализ предметной области, определен функционал реализуемого приложения и выбран стек технологий для реализации приложения во второй главе.

					ПГУ 1.090501.03.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

2 Реализация программного приложения для обработки заказов

Программное приложение для обработки заказов реализовано на базе паттерна MVC (модель-вид-контроллер) и состоит из трех основных частей (рисунок 2.1), каждая из которых может передать команды и данные:

- клиентская часть (пользовательский интерфейс, вид);
- серверная часть (уровень контроллеров и обработчиков данных и команд);
- схема базы данных (модель).

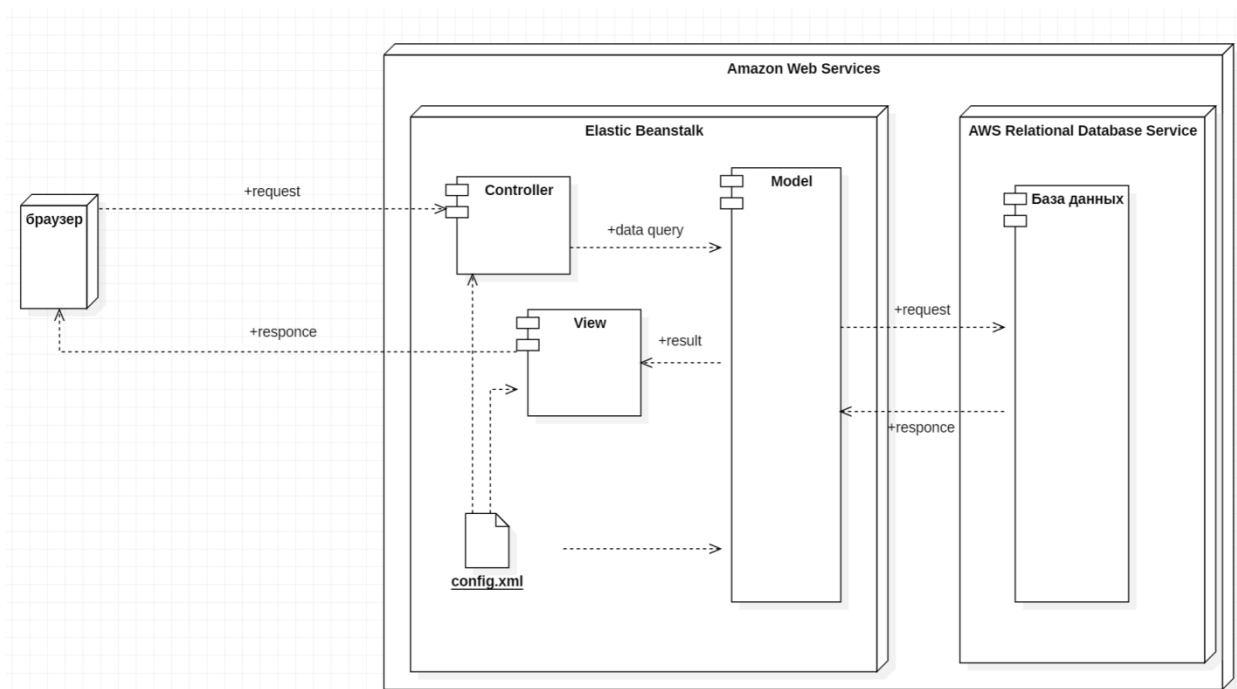


Рисунок 2.1 – Диаграмма развертывания. Архитектура приложения

2.1 Проектирование и реализация базы данных

Реализованная база данных (рисунок 2.2) является реляционной, что позволяет: избежать избыточных, дублирующихся и недопустимых данных, структуру базы данных легко изменять и поддерживать, есть возможность масштабировать базу данных в целом и таблицы в частности.

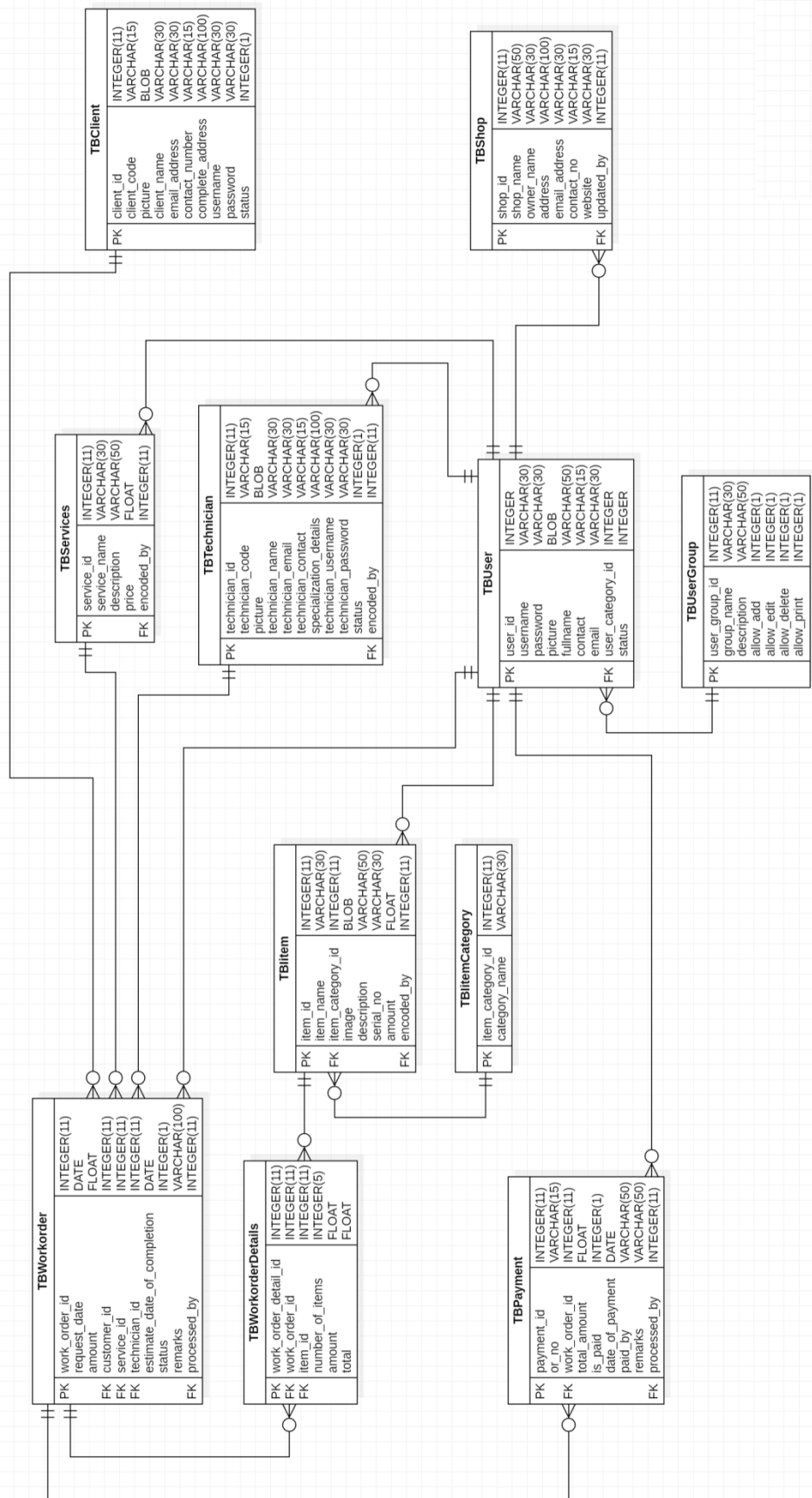


Рисунок 2.2 – Диаграмма базы данных
программного приложения для обработки заказов

Реализованная база данных состоит из одиннадцати таблиц. При проектировании БД сначала применяют логическую модель проектирования, которая дает представление о сущностях БД и связях между ними, затем логическую модель экспортируют в физический вариант БД.

Так как количество сущностей небольшое, было принято решение реализовать только физическую модель с последующей ее реализацией в программе phpMyAdmin.

В таблицах 2.1 – 2.11 представлено описание полей таблиц БД.

Таблица 2.1 – Описание полей таблицы «TBUser»

Имя поля	Тип	Примечание
user_id	integer	Первичный ключ, обязательный
username	varchar(30)	Логин пользователя, обязательный
password	varchar(30)	Пароль в зашифрованном виде, обязательный
picture	blob	Фотография пользователя, необязательный
fullname	varchar(50)	ФИО пользователя, обязательный
contact	varchar(15)	Номер телефона, обязательный
email	varchar(30)	Адрес электронной почты, обязательный
user_category_id	integer	Группа пользователей, обязательный
status	integer	Активен пользователь в системе или нет, обязательный

Для создания таблицы 2.1 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```
CREATE TABLE IF NOT EXISTS TBUser (
  user_id int(11) NOT NULL AUTO_INCREMENT,
  username varchar(30) NOT NULL,
  password` varchar(30) NOT NULL,
  picture blob NOT NULL,
  fullname varchar(50) NOT NULL,
  contact varchar(15) NOT NULL,
  email varchar(30) NOT NULL,
  user_category_id int(1) NOT NULL,
```

```

status int(1) NOT NULL,
PRIMARY KEY (user_id),
KEY user_category_id (user_category_id)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;

```

Таблица 2.2 – Описание полей таблицы «TBUserGroup»

Имя поля	Тип	Примечание
user_group_id	integer	Первичный ключ, обязательный
group_name	varchar(30)	Наименование групп пользователей, обязательный
description	varchar(30)	Информация о группе, обязательный
allow_add	blob	Разрешение на операцию добавление записи в БД, необязательный
allow_edit	varchar(50)	Разрешение на операцию редактирования записи в БД, обязательный
allow_delete	varchar(15)	Разрешение на операцию удаления записи в БД, обязательный
allow_print	varchar(30)	Разрешение на операцию печатать отчет, обязательный

Для создания таблицы 2.2 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```

CREATE TABLE IF NOT EXISTS TBUserGroup (
user_group_id int(11) NOT NULL AUTO_INCREMENT,
group_name varchar(30) NOT NULL,
description varchar(50) NOT NULL,
allow_add int(1) NOT NULL,
allow_edit int(1) NOT NULL,
allow_delete int(1) NOT NULL,
allow_print int(1) NOT NULL,
PRIMARY KEY (user_group_id)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1 ;

```

Таблица 2.3 – Описание полей таблицы «TBClient»

Имя поля	Тип	Примечание
client_id	integer	Первичный ключ, обязательный
client_code	varchar(15)	Уникальный код, предоставляемый клиенту, обязательный
picture	blob	Фотография пользователя, обязательный
client_name	varchar(15)	ФИО клиента, обязательный
email_address	varchar(15)	Адрес электронной почты, обязательный
contact_number	varchar(15)	Мобильный телефон, обязательный
complete_address	varchar(15)	Адрес клиента, обязательный
username	varchar(15)	ФИО клиента, обязательный
password	varchar(30)	Пароль в зашифрованном виде, обязательный
status	integer	Активен пользователь в системе или нет, обязательный

Для создания таблицы 2.3 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```
CREATE TABLE IF NOT EXISTS TBClient (
  client_id int(11) NOT NULL AUTO_INCREMENT,
  client_code varchar(15) NOT NULL,
  avatar blob NOT NULL,
  client_name varchar(30) NOT NULL,
  email_address varchar(30) NOT NULL,
  contact_number varchar(15) NOT NULL,
  complete_address varchar(100) NOT NULL,
  username varchar(30) NOT NULL,
  password varchar(30) NOT NULL,
  status int(1) NOT NULL,
  PRIMARY KEY (client_id)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;
```

Таблица 2.4 – Описание полей таблицы «TBServices»

Имя поля	Тип	Примечание
service_id	integer	Первичный ключ, обязательный
service_name	varchar(15)	Название услуги, предлагаемой ремонтным сервисом, обязательный
description	varchar(50)	Описание услуги, обязательный
price_ed	float	Цена единицы услуги, обязательный
encoded_by	integer	Внешний ключ, указывает на пользователя, который завел запись услуги, обязательный

Для создания таблицы 2.4 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```
CREATE TABLE IF NOT EXISTS tblservices (
  service_id int(11) NOT NULL AUTO_INCREMENT,
  service_name varchar(30) NOT NULL,
  description varchar(50) NOT NULL,
  price_ed float NOT NULL,
  encoded_by int(11) NOT NULL,
  PRIMARY KEY (service_id),
  KEY encoded_by (encoded_by)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;
```

Таблица 2.5 – Описание полей таблицы «TBItem»

Имя поля	Тип	Примечание
item_id	integer	Первичный ключ, обязательный
item_name	varchar(30)	Название детали или элемента для замены, обязательный
item_category_id	integer	Внешний ключ, ссылается на таблицу TBItemcategory, обязательный
picture	blob	Изображение детали или элемента для замены, обязательный
description	varchar(50)	ТТХ детали или элемента для замены, обязательный

Имя поля	Тип	Примечание
serial_no	varchar(30)	Серийный номер детали или элемента для замены, обязательный
price_ed	float	Стоимость единицы детали или элемента для замены, обязательный
encoded_by	integer	Внешний ключ, указывает на пользователя, который завел запись услуги, обязательный

Для создания таблицы 2.5 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```
CREATE TABLE IF NOT EXISTS tblitem (
  item_id int(11) NOT NULL AUTO_INCREMENT,
  item_name varchar(30) NOT NULL,
  item_category_id int(11) NOT NULL,
  picture blob NOT NULL,
  description varchar(50) NOT NULL,
  serial_no varchar(30) NOT NULL,
  price_ed float NOT NULL,
  encoded_by int(11) NOT NULL,
  PRIMARY KEY (item_id),
  KEY item_category_id (item_category_id, encoded_by),
  KEY encoded_by (encoded_by)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1 ;
```

Таблица 2.6 – Описание полей таблицы «TBItemcategory»

Имя поля	Тип	Примечание
item_category_id	integer	Первичный ключ, обязательный
category_name	varchar(30)	Категория услуги, обязательный

Для создания таблицы 2.6 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```
CREATE TABLE IF NOT EXISTS `tblitemcategory` (
  item_category_id int(11) NOT NULL AUTO_INCREMENT,
  category_name varchar(30) NOT NULL,
```

```
PRIMARY KEY (item_category_id)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;
```

Таблица 2.7 – Описание полей таблицы «TBTechnician»

Имя поля	Тип	Примечание
technician_id	integer	Первичный ключ, обязательный
technician_code	varchar(15)	Уникальный номер ремонтного специалиста, обязательный
picture	blob	Фотография ремонтного специалиста, обязательный
technician_name	varchar(30)	ФИО ремонтного специалиста, обязательный
technician_email	varchar(30)	Адрес электронной почты ремонтного специалиста, обязательный
technician_contact	varchar(15)	Контактная информация ремонтного специалиста, обязательный
specialization_details	varchar(100)	Специализация, обязательный
technician_username	varchar(30)	Логин ремонтного специалиста, обязательный
technician_password	varchar(30)	Пароль ремонтного специалиста, обязательный
status	integer	Статус, активен или заблокирован, обязательный
encoded_by	integer	Внешний ключ

Для создания таблицы 2.7 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```
CREATE TABLE IF NOT EXISTS TBTechnician (
  technician_id int(11) NOT NULL AUTO_INCREMENT,
  technician_code varchar(15) NOT NULL,
  picture blob NOT NULL,
  technician_name varchar(30) NOT NULL,
  technician_email varchar(30) NOT NULL,
  technician_contact varchar(15) NOT NULL,
  specialization_details varchar(100) NOT NULL,
  technician_username varchar(30) NOT NULL,
  technician_password varchar(30) NOT NULL,
  status int(1) NOT NULL,
  encoded_by int(11) NOT NULL,
```

```
PRIMARY KEY (technician_id),
KEY encoded_by (encoded_by)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;
```

Таблица 2.8 – Описание полей таблицы «TBWorkorder»

Имя поля	Тип	Примечание
work_order_id	integer	Первичный ключ, обязательный
request_date	date	Дата начала работы, обязательный
service_id	varchar(30)	ФИО ремонтного специалиста, обязательный
price	varchar(30)	Адрес электронной почты ремонтного специалиста, обязательный, обязательный
customer_id	integer	Внешний ключ, который указывает на информацию о клиенте, который запрашивает ремонт или обслуживание, обязательный
technician_id	integer	Внешний ключ, который ссылается на специалиста, назначенного на выполнение работы по заказу, обязательный
estimate_date_of_completion	date	Дата завершения, обязательный
status	integer	Статус выполнения работы, обязательный
remarks	varchar(100)	Комментарии и информация о работе, обязательный
processed_by	integer	Внешний ключ, указывает пользователя, взявшего заказ, обязательный

Для создания таблицы 2.8 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```
CREATE TABLE IF NOT EXISTS TBWorkorder (
work_order_id int(11) NOT NULL AUTO_INCREMENT,
request_date date NOT NULL,
service_id int(11) NOT NULL,
price float NOT NULL,
customer_id int(11) NOT NULL,
```

```

    technician_id int(11) NOT NULL,
    estimate_date_of_completion date NOT NULL,
    status int(1) NOT NULL,
    remarks varchar(100) NOT NULL,
    processed_by` int(11) NOT NULL,
    PRIMARY KEY (`work_order_id`),
    KEY service_id (service_id, customer_id, technician_id,
processed_by),
    KEY processed_by (processed_by),
    KEY customer_id (customer_id),
    KEY technician_id (technician_id)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;

```

Таблица 2.9 – Описание полей таблицы «TBWorkorderDetails»

Имя поля	Тип	Примечание
work_order_detail_id	integer	Первичный ключ, обязательный
work_order_id	integer	Внешний ключ, который ссылается на таблицу заказов на выполнение работ, обязательный
item_id	integer	Внешний ключ, который указывает на информацию о деталях для ремонта в таблице TBItem, обязательный
number_of_items	integer	Количество деталей, которые должны быть включены в заказ на выполнение работ, обязательный
price	float	Цена каждой детали, которые должны быть включены в заказ на выполнение работ, обязательный
total	float	Общая сумма заказа

Для создания таблицы 2.9 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```

CREATE TABLE IF NOT EXISTS TBWorkorderDetails (
work_order_detail_id int(11) NOT NULL AUTO_INCREMENT,
work_order_id int(11) NOT NULL,

```

```

item_id int(11) NOT NULL,
number_of_items int(5) NOT NULL,
price float NOT NULL,
total float NOT NULL,
PRIMARY KEY (work_order_detail_id),
KEY work_order_id (work_order_id, item_id),
KEY item_id (item_id)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;

```

Таблица 2.10 – Описание полей таблицы «ТВPayment»

Имя поля	Тип	Примечание
payment_id	integer	Первичный ключ, обязательный
or_no	varchar(15)	Официальный номер квитанции, обязательный
work_order_id	integer	Внешний ключ, обязательный
total_amount	float	Сумма услуги и сумма деталей, обязательный
is_paid	integer	Статус платежа (оплачен или нет) , обязательный
date_of_payment	date	Дата платежа, обязательный
paid_by	varchar(50)	ФИО плательщика, обязательный
remarks	varchar(50)	Детали и комментарии, обязательный
processed_by	integer	Внешний ключ, который указывает на пользователя, который принял платеж

Для создания таблицы 2.10 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```

CREATE TABLE IF NOT EXISTS TBPayment (
payment_id int(11) NOT NULL AUTO_INCREMENT,
or_no varchar(15) NOT NULL,
work_order_id int(11) NOT NULL,
total_amount float NOT NULL,
is_paid int(1) NOT NULL,
date_of_payment date NOT NULL,
paid_by varchar(50) NOT NULL,
remarks varchar(50) NOT NULL,
processed_by int(11) NOT NULL,

```

```

PRIMARY KEY (payment_id),
KEY work_order_id (work_order_id, processed_by),
KEY processed_by (processed_by)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;

```

Таблица 2.11 – Описание полей таблицы «TBRepairShop»

Имя поля	Тип	Примечание
shop_id	integer	Первичный ключ, обязательный
shop_name	varchar(50)	Официальный номер квитанции, обязательный
owner_name	varchar(30)	Внешний ключ, обязательный
address	varchar(100)	Сумма услуги и сумма деталей, обязательный
email_address	varchar(30)	Статус платежа (оплачен или нет), обязательный
contact	varchar(15)	Контактная информация, обязательный
website	varchar(30)	Веб-сайт компании, обязательный
updated_by	integer	Пользователь, обновивший информацию, обязательный

Для создания таблицы 2.11 в программе phpMyAdmin требуется выполнить следующие команды программы создания таблиц БД:

```

CREATE TABLE IF NOT EXISTS TBRepairShop (
shop_id int(11) NOT NULL AUTO_INCREMENT,
shop_name varchar(50) NOT NULL,
owner_name varchar(30) NOT NULL,
address varchar(100) NOT NULL,
email_address varchar(30) NOT NULL,
contact_no varchar(15) NOT NULL,
website varchar(30) NOT NULL,
updated_by int(11) NOT NULL,
PRIMARY KEY (shop_id),
KEY updated_by (updated_by)
) ENGINE=InnoDB DEFAULT AUTO_INCREMENT=1;

```

2.2 Проектирование и реализация серверной части приложения

Приложение реализовано с помощью инструментов Spring Boot Web на базе архитектурного шаблона MVC, который предоставляет три логических компонента: модель, вид и контроллер (рисунок 2.3).

Модель – это данные и логика, обрабатывающая данные. Она представляет данные, которые передаются между контроллерами или любой другой логикой. Контроллер может извлекать данные (модель) из базы данных и/или от пользователя.

Вид – это часть приложения, которая представляет данные модели. Данные имеют форму базы данных или получены из пользовательского ввода. Это выходное представление пользователям обработанных данных, таких как таблицы, диаграммы и прочее.

Контроллер отвечает за управление взаимодействием пользователя с приложением. Он получает входные данные от пользователей и изменяет модель и представление относительно входных данных.

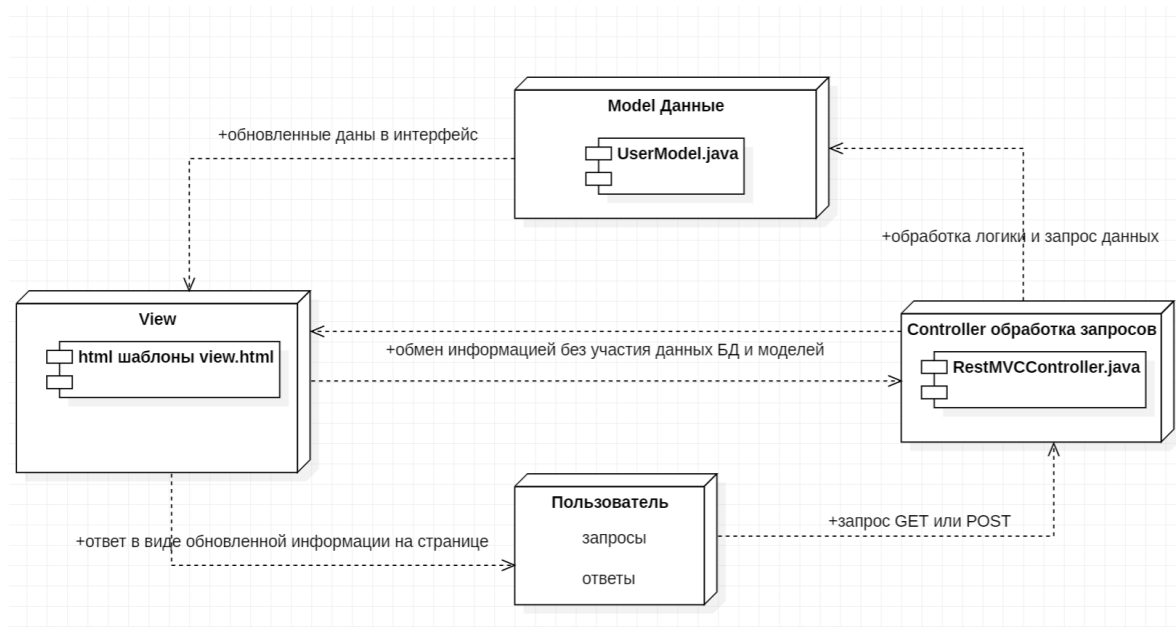


Рисунок 2.3 – Диаграмма развертывания. Паттерн MVC

В данной работе шаблоны проекта были созданы специальными инструментами Spring Doot Web – Spring Initializr (рисунки 2.4 – 2.5), который

создает основу приложения. Созданное приложение облегчает этап настройки и позволяет сразу приступить к реализации основной логики приложения.

The screenshot shows the Spring Initializr interface. At the top, there's a green banner with the text "Meet the Spring team this week". Below it is the "spring initializr" logo. 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 "3.1.1 (SNAPSHOT)", "3.1.0" (selected), "3.0.8 (SNAPSHOT)", "3.0.7", "2.7.13 (SNAPSHOT)", and "2.7.12".
- Project Metadata:** Includes input fields for "Group" (com.computerorder), "Artifact" (erinadiplom), "Name" (erinadiplom), "Description" (Project Spring Boot for AWS), and "Package name" (com.computerorder.erinadiplom). There is also a "Packaging" section with radio buttons for ".Jar" (selected) and "War".
- At the bottom, there are radio buttons for "Java" (selected), "20", "17" (selected), "11", and "8".

Рисунок 2.4 – Создание шаблона проекта в Spring Initializr

The screenshot shows the "WEB" section of the Spring Initializr interface. It lists several project templates:

- Spring Web:** Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container. (Highlighted with a green background)
- Spring Reactive Web:** Build reactive web applications with Spring WebFlux and Netty.
- Spring for GraphQL:** Build GraphQL applications with Spring for GraphQL and GraphQL Java.
- Rest Repositories:** Exposing Spring Data repositories over REST via Spring Data REST.
- Spring Session:** Provides an API and implementations for managing user session information.
- Rest Repositories HAL Explorer:** Browsing Spring Data REST repositories in your browser.

Рисунок 2.5 – Дополнительные настройки Spring Initializr

Для работы в системе WEB с поддержкой базы данных требуется выбрать три зависимости: Spring Web, Spring Data JPA, Database.

После выбора желаемых зависимостей требуется сгенерировать проект, нажав кнопку «Сгенерировать». Нажатие на кнопку загрузит zip-файл. После распаковки zip-файла и импорта проекта в IDE Eclipse можно приступить к разработке основной логики проекта.

Далее требуется разобраться с зависимостями проекта (dependencies), благодаря которым происходит взаимодействие частей проекта, и знание проекта, какая часть с кем и в каких условиях взаимодействует.

Веб-зависимости Spring предоставляет основные функции Spring (Spring MVC, серверный контейнер для локального запуска). С помощью веб-зависимости Spring можно создавать классы контроллеров с логикой, представленной на рисунке 2.3. Контроллер в WEB-приложении выполняет следующие действия: принимает входящие HTTP-запросы, проверяет и сортирует входные данные, отправляет данные на уровень логики обработки, сортирует выходные данные из уровня бизнес логики, обрабатывает исключения и возвращает пользователю ответ.

Построение уровня доступа к данным реализовано с помощью Spring Data JPA. JPA (Java Persistence API) – это набор концепций, который помогает писать код для базы данных. Поскольку JPA – это всего лишь набор спецификаций, для него требуется реализация, подобная Hibernate ORM. Hibernate – это решение, которое формирует сопоставление между классами Java и таблицами в базе данных. Это позволяет редактировать данные в таблицах БД без написания SQL кода.

Далее требуется проверить, что создан файл общей настройки проекта pom.xml. В pom.xml определяются все зависимости, которые используются в проекте. Зависимость – это пакет, содержащий фрагмент кода, необходимый проекту для успешного запуска. Фрагмент pom.xml файла проекта приведен ниже

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org//XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
```

Далее требуется настроить доступ к базе данных. Для доступа кода приложения к базе данных требуется определить конфигурацию для расположения базы данных, регистрационной информации БД. В приложении Spring Boot файл application.yml – это контейнер со служебной информацией о базе данных. Настройки для подключения БД показаны ниже:

```
spring.datasource.url=hibernate:mysql:mem:localdb
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.datasource.driver-class-name=org.MySQL.Driver
```

Для определения учетной записи доступа к БД требуется ввести следующую информацию:

```
spring.datasource.username=username
spring.datasource.password=password
spring.h2.console.enabled=true
```

После реализации настроек проекта необходимо описать все модули слоев вид, модель, контроллер для реализации логики приложения.

Построение уровня модели начинается с создания классов сущностей, которые фактически являются аналогом таблиц в базе данных. Ниже представлен пример класса сущности Services

```
@Entity(name = " TBServices")
public class Book {
    @Id
```

```

@GeneratedValue(strategy = GenerationType.AUTO)
private long id;
@Column(name = " service_name ")
private String service_name;
@Column(name = " description ")
private String description;
@Column(name = " price_ed ")
private float price_ed;

```

После создания сущностей требуется разработать уровень доступа к данным. Уровень доступа к данным позволяет использовать методы для манипулирования данными в базе данных. Уровень доступа к данным, создается с помощью шаблона репозитория.

```

public interface BookRepository extends JpaRepository<Service,
Long> {
@Query( value = "SELECT service_name FROM book Service where
service_name.numberOfInstances like 'Комп' ")
List<Service> findWithMoreInstancesThenFiveJPQL();
}

```

JpaRepository превращает интерфейс в компонент репозитория, который добавляется в группу ApplicationContext и становится доступным для каждого слоя приложения. Для работы без синтаксиса базы данных в приложении используется синтаксис JPQL (Java Persistence Query Language).

После создания классов обработки уровня Модели, необходимо реализовать классы уровня контроллеров и бизнес-логики. Для этого требуется реализовать классы создания, удаления, редактирования заказов, так как это представлено ниже в виде сервисов.

```

@Service
public class UpdateService {
    private final GetUserService getUserService;
    private final SeviceRepository serviceRepository;

    public UpdateService(
        GetUserService getUserService,

```

```

        ServiceRepository serviceRepository) {
            this.getUserService = getUserService;
            this.serviceRepository = serviceRepository;
        }

        public void takeOrder(long serviceId, long userId){
            User user = getUserService.getById(userId);
            if(user.getTakeService().stream()
                .anyMatch(service -> service.getId()==
serviceId)){
                throw new IllegalStateException("Пользователь уже
заказал сервис");
            }
        }
    }
}

```

Далее требуется создать класс контроллера, связанного с сервисом. Данный контроллер отвечает за отображение по запросу всех заказов, которые находятся в базе данных.

```

@RestController
@RequestMapping("/services")
public class ServicesRestController {
    private final GetService getService;
    @Autowired
    public ServicesRestController(GetService getService) {
        this.getService = getService;
    }
    @GetMapping
    List<ServiceResponse> fetchAllServices(){
        return getService.getAllServices();
    }
}

```

После реализации проекта его требуется скомпилировать и проверить на локальном компьютере, для этого с помощью инструмента командной строки браузера URL требуется набрать команду: *'http://localhost:8080/orders'*. Будет выполнен GET-запрос к приложению, и в ответ появится список заказов, доступных в БД.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		47

Все запросы к реализуемому приложению будут выполняться в формате GET и POST-запросов.

2.3 Проектирование пользовательского интерфейса

Структура сайта относится к организации и упорядочению страниц и контента интернет-ресурса. Структура сайта определяет информационную иерархию внутри сайта и служит дорожной картой для поисковых систем. Хорошо структурированный сайт облегчает навигацию, улучшает пользовательский опыт и помогает поисковым системам, таким как Google, понимать и эффективно индексировать содержимое сайта. Это, в свою очередь, может повысить производительность сайта, облегчая пользователям поиск информации и взаимодействие с функционалом ресурса. В конечном счете, оптимизированная структура сайта помогает достичь более высокого рейтинга, большего трафика и более высоких показателей конверсии. Ниже представлены рисунки, на которых представлена структура сайта, реализуемого проекта.

Структура сайта логически делится на страницы заказчика и исполнителя (рисунки 2.6 и 2.7). Реализованы они по одному шаблону на основании html кода и css стилей.

					ПГУ 1.090501.03.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

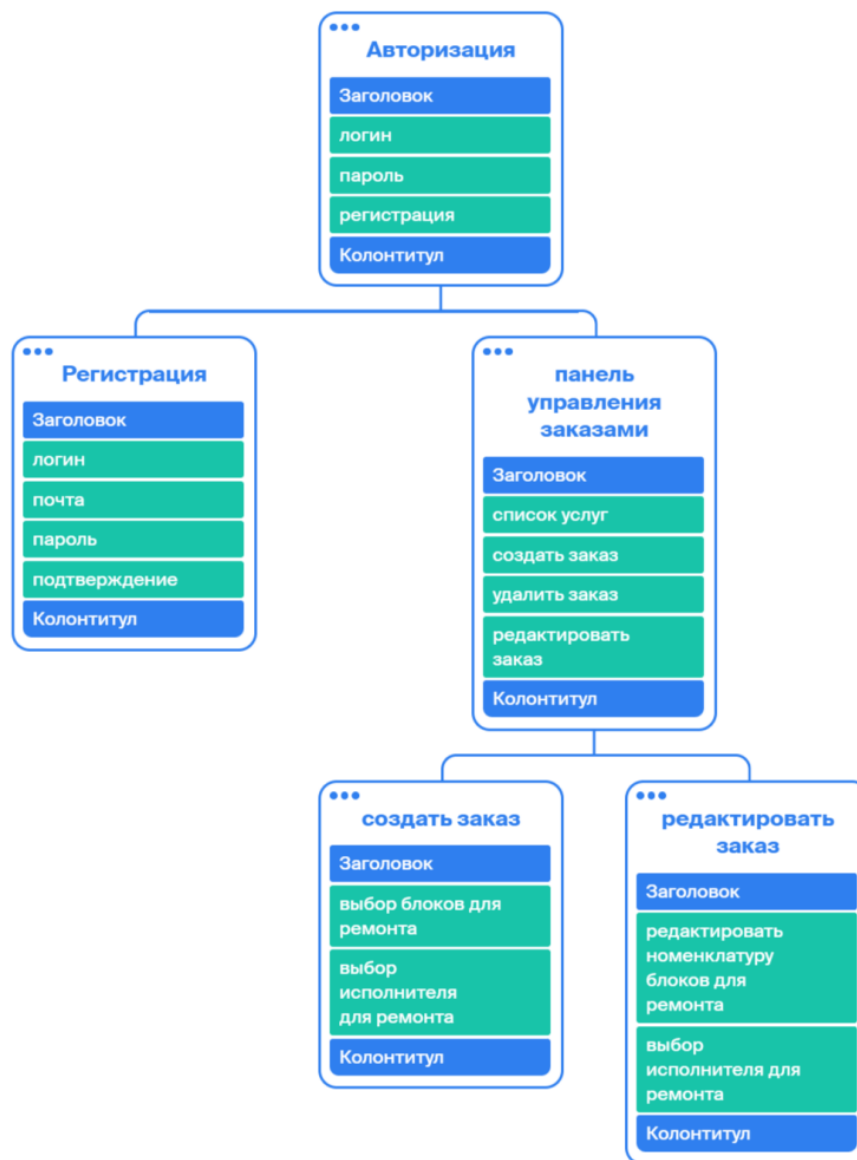


Рисунок 2.6 – Структура страниц для роли «Заказчик»

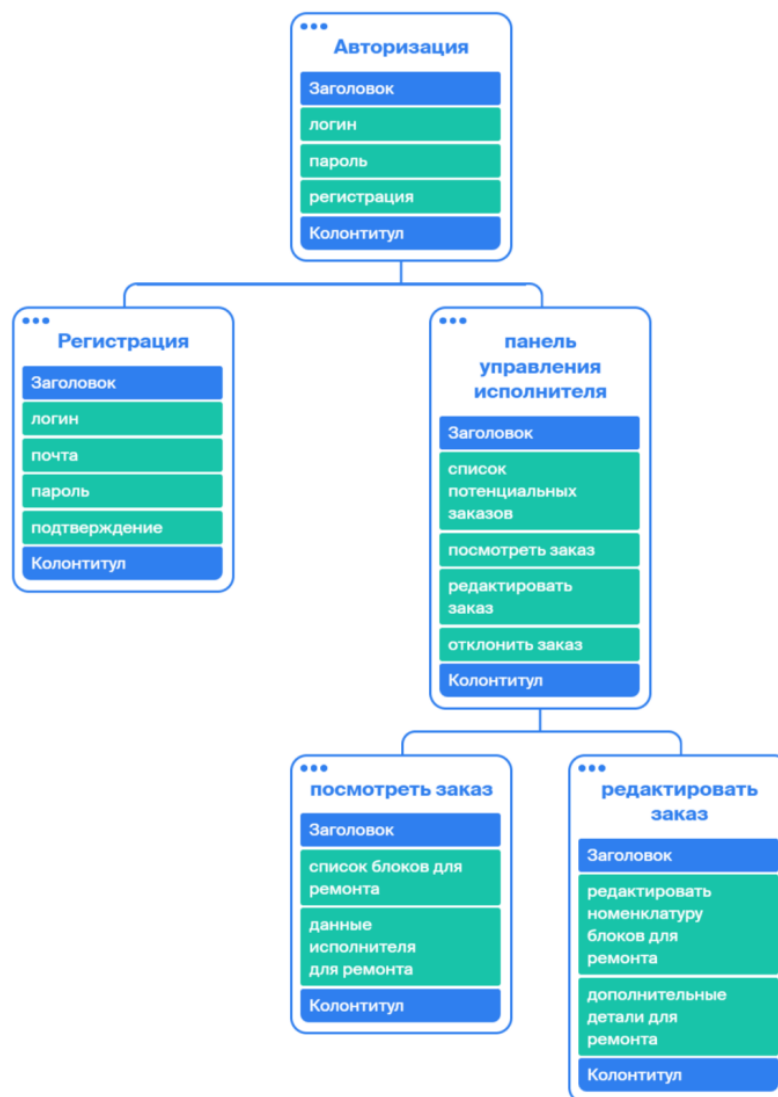


Рисунок 2.7 – Структура страниц для роли «Исполнитель»

Пользовательский интерфейс объединяет концепции взаимодействия, визуального дизайна и информационной архитектуры.

Элементы интерфейса включают:

- элементы управления вводом: кнопки, текстовые поля, флажки, переключатели, выпадающие списки, поля со списком, переключатели, поле даты;
- навигационные компоненты: разметку, поле поиска, разбивку на страницы, иконки;
- информационные компоненты: подсказки, значки, индикатор выполнения, уведомления, окна сообщений, модальные окна.

При проектировании и реализации интерфейса следует придерживаться общепринятых рекомендаций:

- реализовывать интерфейс простым и понятным конечному пользователю;
- избегать ненужных элементов;
- соблюдать согласованность и использование общих элементов пользовательского интерфейса;
- реализовывать качественную верстку страницы и продуманное размещение элементов;
- уместное использование цвета и текстуры.

В результате использования вышеописанных рекомендаций интерфейс приложения получится лаконичным и интуитивно понятным конечному пользователю. Ниже представлены страницы пользовательского интерфейса приложения на рисунках 2.8 – 2.17.

Регистрация пользователя в системе

Пользователь Исполнитель

Логин: erina_ag

Почта: er2023@gmail.com

Пароль: *****

Подтверждение пароля: *****

Подписаться

Отменить

Рисунок 2.8 – Регистрация пользователя в системе

Элементами страницы регистрации являются (рисунок 2.8):

- форма регистрации – тип Form;
- поле ввода логина – тип Input type;

- поле ввода электронной почты – тип Input type;
- поле ввода пароля – тип Input type;
- поле ввода подтверждения пароля – тип Input type;
- кнопка регистрации – тип Button;
- ссылка на форму авторизации – тип href.

Рисунок 2.9 – Вход пользователя в систему

Элементами страницы авторизации являются (рисунок 2.9):

- форма регистрации – тип Form;
- поле ввода логина – тип Input type;
- поле ввода пароля – тип Input type;
- элемент определения роли пользователя – тип Tab;
- кнопка авторизации – тип Button;
- ссылка на форму регистрации – тип href.

Панель управления заказами

Компьютеры	Ноутбуки	МФУ А4	МФУ А3
Создать	Создать	Создать	Создать
Удалить	Удалить	Удалить	Удалить
Редактировать	Редактировать	Редактировать	Редактировать

Рисунок 2.10 – Информационная панель пользователя системы до заказа

Элементами страницы Информационная панель пользователя системы являются (рисунок 2.10):

- форма информационной панели – тип Form;
- информационное поле вывода логина пользователя – тип href;
- информационное поле вывода информации о заказах – тип input (в режиме чтения);
- элемент завершения сеанса пользователя – тип href;
- информативная панель заказов – тип label;
- ссылки на формы «Создать», «Удалить», «Редактировать» – тип href.

Пользователь: eгiпa_ag

Заказов в корзине: 0

Выход

Что сломалось, на основании диагностики

✓

Материнская плата

1

5000

✓

Планки памяти

1

1000

✓

Процессор

1

10000

✓

Куллер

1

500

✓

Блок питания

1

1000

Всего позиций для ремонта: 5

Общая сумма: 17500

Добавить

Отменить

Доступные исполнители

☒ Персик Кирилл Валерьевич

☐ Иванов Иван Иванович

☐ Дримтим Петр Васильевич

Рисунок 2.11 – Информационная панель заказа на ремонт компьютера

Элементами страницы Информационная панель заказа на ремонт компьютера являются (рисунок 2.11):

- форма информационной панели – тип Form;
- информационное поле ввода информации о блоках требующих замену – тип input;
- информационное поле выбора исполнителя – тип input;
- кнопка добавить заказ – тип button;
- кнопка отменить заказ – тип button;
- информация о сумме и объеме ремонта – тип input (в режиме чтения).

Панель управления заказами

Заказы:

Компьютеры: 1

Ноутбуки: 1

Компьютеры	Ноутбуки	МФУ А4	МФУ А3
Создать	Создать	Создать	Создать
Удалить	Удалить	Удалить	Удалить
Редактировать	Редактировать	Редактировать	Редактировать

Рисунок 2.12 – Информационная панель пользователя системы после заказа

На рисунке 2.12 показаны изменения, произошедшие на информационной панели после оформления заказов. Информация о количестве заказов появилась в режиме для чтения.

Пользователь	Исполнитель
Логин	<input type="text" value="persik"/>
Почта	<input type="text" value="perskirill@gmail.com"/>
Пароль	<input type="password" value="*****"/>
Подтверждение пароля	<input type="password" value="*****"/>
<input type="button" value="Подписаться"/>	
Отменить	

Рисунок 2.13 – Регистрация исполнителя

Элементами страницы регистрации являются (рисунок 2.13):

- форма регистрации – тип Form;
- поле ввода логина – тип Input type;
- поле ввода электронной почты – тип Input type;
- поле ввода пароля – тип Input type;
- поле ввода подтверждения пароля – тип Input type;
- кнопка регистрации – тип Button;
- ссылка на форму авторизации - тип href.

Рисунок 2.14 – Регистрация исполнителя

Элементами страницы авторизации являются (рисунок 2.14):

- форма регистрации – тип Form;
- поле ввода логина – тип Input type;
- поле ввода пароля – тип Input type;
- элемент определения роли пользователя – тип Tab;
- кнопка авторизации – тип Button;
- ссылка на форму регистрации – тип href.

Пользователь: persik		Заказов в корзине: 1		Выход	
Заказ	Заказчик	Кол-во деталей	Действие		
Компьютер	erina_ag	5	Посмотреть	Редактировать	Отклонить

Рисунок 2.15 – Главная страница исполнителя

Элементами страницы информационная панель администратора системы являются (рисунок 2.15):

- форма информационной панели – тип Form;
- информационное поле вывода логина пользователя – тип href;
- информационное поле вывода информации о заказах и заказчиках – тип input (в режиме чтения);
- элемент завершения сеанса пользователя – тип href;
- информативная панель заказов – тип label;
- ссылки на формы «Посмотреть», «Редактировать», «Отклонить» – тип href.

Пользователь: persik		Редактировать заказ №1		Выход	
Требуемые детали, с учетом замечаний Заказчика:			Дополнительные детали от исполнителя:		
☒	Материнская плата	<input type="text" value="1"/>	5000	CD/DVD	<input type="text" value="0"/>
☒	Планки памяти	<input type="text" value="2"/>	2000	Видеокарта	<input type="text" value="0"/>
☒	Процессор	<input type="text" value="1"/>	10000		
☒	Куллер	<input type="text" value="1"/>	500		
☒	Блок питания	<input type="text" value="1"/>	1000		
<p>Всего позиций для ремонта: 6 Общая сумма: 18500</p>					
<input type="button" value="Сохранить"/>		<input type="button" value="Отклонить"/>		<input type="button" value="Добавить"/>	

Рисунок 2.16 – Редактирование заказа

Элементами страницы информационная панель заказа на ремонт компьютера в режиме администратора являются (рисунок 2.16):

- форма информационной панели – тип Form;
- информационное поле ввода информации о блоках требующих замену – тип input;
- информационное поле выбора дополнительных деталей – тип input;
- кнопка сохранения заказа – тип button;
- кнопка отмены заказ – тип button;
- кнопка добавления дополнительных элементов заказа – тип button;
- информация о сумме и объеме ремонта – тип input (в режиме чтения).

Пользователь: persik

Заказов в корзине: 1

Выход

Заказ	Заказчик	Кол-во деталей	Действие		
Компьютер	erina_ag	6	Посмотреть	Редактировать	Закрыть

Рисунок 2.17 – Главная страница исполнителя после редакции заказа.

На рисунке 2.17 показаны изменения, произошедшие на информационной панели исполнителя после редактирования заказа. Информация о количестве заказов появилась в режиме для чтения.

3 Технология размещения программного приложения для обработки заказов в среде исполнения

Для развертывания MVC приложения на исполнение существует несколько способов:

- вариант размещения на хостинге, с инсталляцией и настройкой всей инфраструктуры и среды исполнения для выполнения своего приложения (рисунок 3.1);
- вариант использования облачного сервиса с уже настроенными модулями для размещения и исполнения java кода и БД (рисунок 3.2).

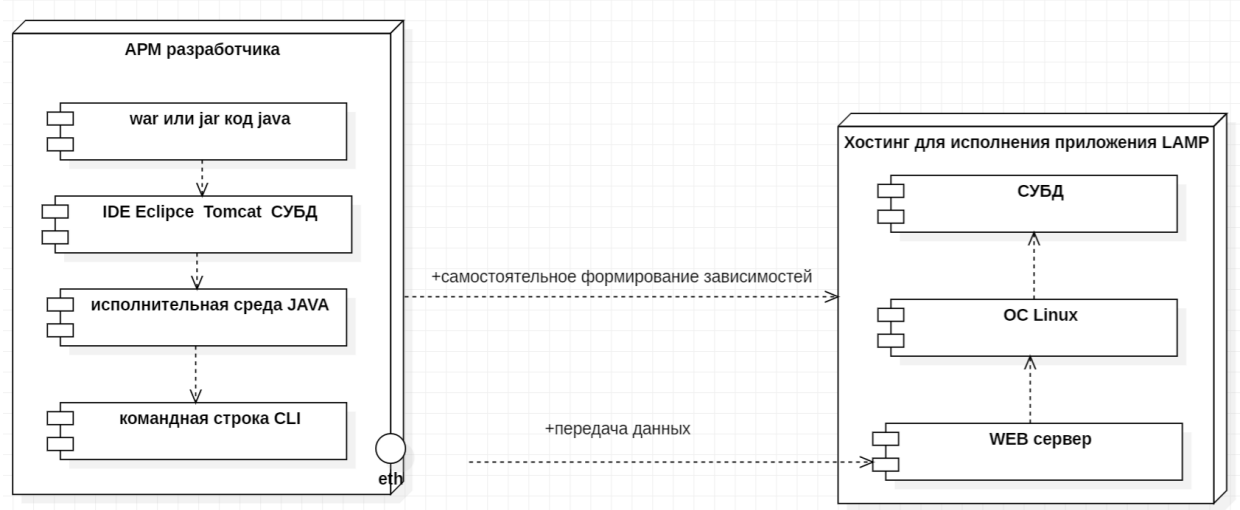


Рисунок 3.1 – Диаграмма развертывания приложения (хостинг)

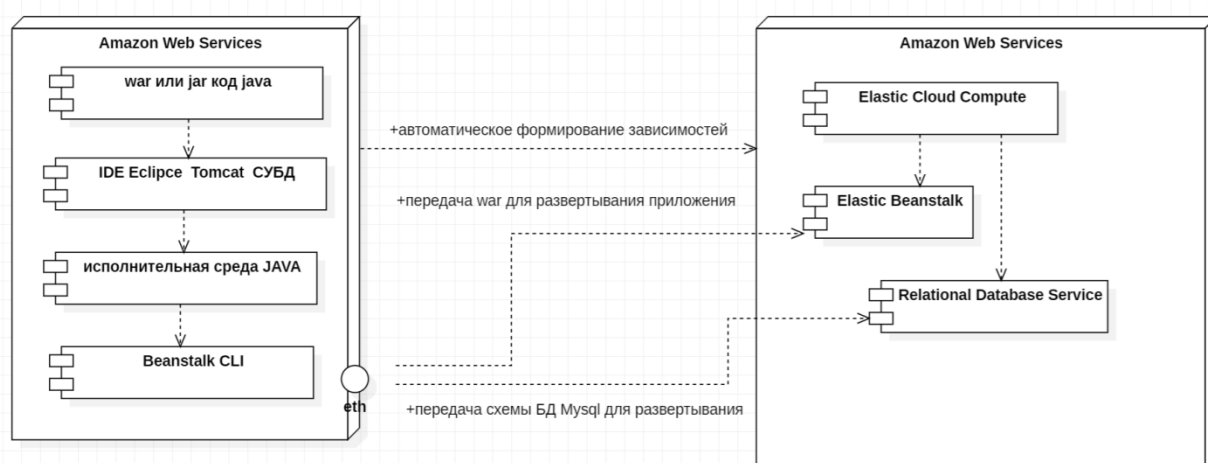


Рисунок 3.2 – Диаграмма развертывания приложения
(облачные сервисы)

Облачная платформа Amazon Web Services имеет (рисунок 3.2) специальные модули:

- Elastic Beanstalk – для исполнения java кода;
- RDS – модуль развертывания и исполнения базы данных;
- EC2 – среда исполнения с встроенными операционными системами.

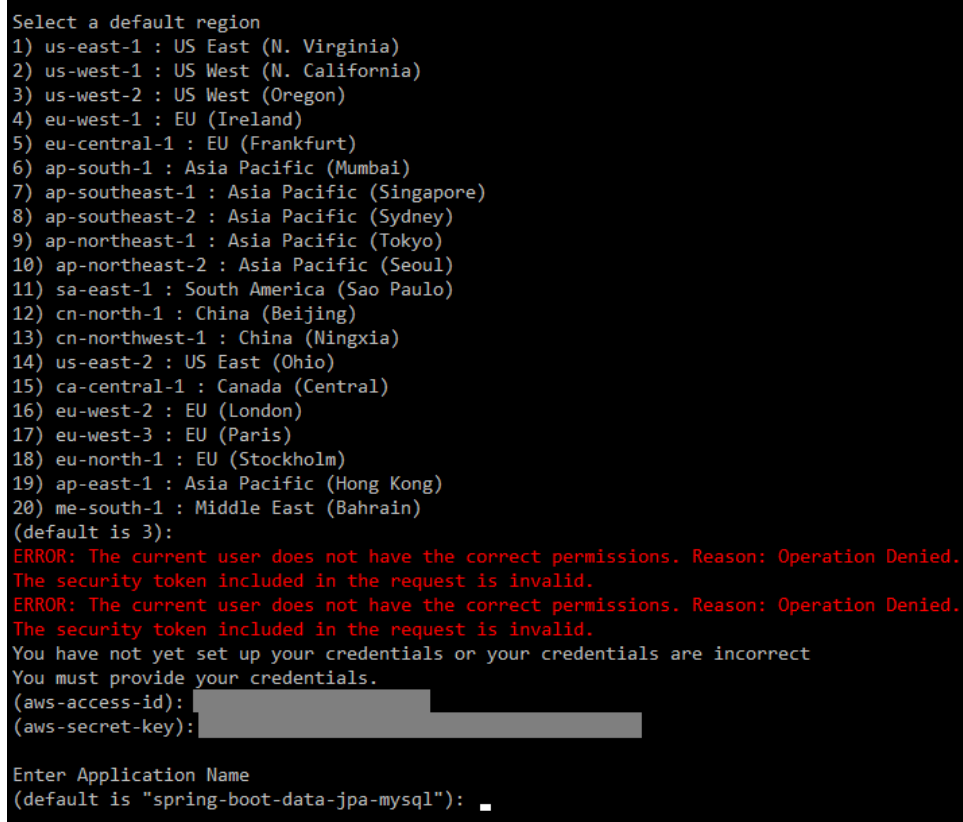
Для миграции с APM разработки исполнительного кода (war или jar архива исполнительного кода java) требуется установить EB CLI (интерфейс командной строки Elastic Beanstalk). Для этого требуется выполнить команду:

```
pip install awsebcli -upgrade
```

Инициализация проекта Beanstalk для приложения Java производится командой:

```
eb init
```

Затем необходимо использовать идентификатор доступа AWS и секретный ключ AWS. Для этого требуется перейти в консоль управления идентификацией и доступом (IAM), как представлено на рисунке 3.3.



```
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : EU (Ireland)
5) eu-central-1 : EU (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
8) ap-southeast-2 : Asia Pacific (Sydney)
9) ap-northeast-1 : Asia Pacific (Tokyo)
10) ap-northeast-2 : Asia Pacific (Seoul)
11) sa-east-1 : South America (Sao Paulo)
12) cn-north-1 : China (Beijing)
13) cn-northwest-1 : China (Ningxia)
14) us-east-2 : US East (Ohio)
15) ca-central-1 : Canada (Central)
16) eu-west-2 : EU (London)
17) eu-west-3 : EU (Paris)
18) eu-north-1 : EU (Stockholm)
19) ap-east-1 : Asia Pacific (Hong Kong)
20) me-south-1 : Middle East (Bahrain)
(default is 3):
ERROR: The current user does not have the correct permissions. Reason: Operation Denied.
The security token included in the request is invalid.
ERROR: The current user does not have the correct permissions. Reason: Operation Denied.
The security token included in the request is invalid.
You have not yet set up your credentials or your credentials are incorrect
You must provide your credentials.
(aws-access-id): 
(aws-secret-key): 
Enter Application Name
(default is "spring-boot-data-jpa-mysql"): _
```

Рисунок 3.3 – Доступ в консоли к среде AWS

Далее требуется выбрать платформу исполнительской среды, в данном случае это будет Java (рисунок 3.4).

```
Select a platform.
1) Python (BETA)
2) Docker (BETA)
3) Ruby (BETA)
4) Go (BETA)
5) Node.js (BETA)
6) PHP (BETA)
7) Node.js
8) PHP
9) Python
10) Ruby
11) Tomcat
12) IIS
13) Docker
14) Multi-container Docker
15) GlassFish
16) Go
17) Java
18) Corretto (BETA)
19) Packer
(default is 1): 17
```

Рисунок 3.4 – Выбор исполнительской среды

Далее требуется создать пару ключей SSH для постоянного доступа среды разработки и облачного сервиса (рисунок 3.5).

```
Select a platform version.
1) Java 8
2) Java 7
(default is 1):
Do you wish to continue with CodeCommit? (y/N) (default is n):
Do you want to set up SSH for your instances?
(Y/n): Y

Type a keypair name.
(Default is aws-eb): bezkoder-app-key
Generating public/private rsa key pair.
```

Рисунок 3.5 – Генерация ключей SSH

					ПГУ 1.090501.03.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

В результате генерации ключей SSH Elastic beanstalk автоматически создаст файл конфигурации в каталоге проекта .elasticbeanstalk /config.yml (рисунок 3.6).

```
branch-defaults:
  master:
    environment: null
    group_suffix: null
global:
  application_name: Application
  branch: null
  default_ec2_keyname: app-key
  default_platform: Java 8
  default_region: ru
  include_git_submodules: true
  instance_profile: null
  platform_name: null
  platform_version: null
  profile: eb-cli
  repository: null
  sc: git
  workspace_type: Java-jpa-mysql
```

Рисунок 3.6 – Содержимое файла конфигурации config.yml

Далее можно в консоли экспортировать файл приложения jar в облачную систему:

```
eb create --single -database // создать канал APM – облачная среда
```

```
artifact: target/Java-jpa-mysql.jar
```

```
mvn clean package spring-boot:repackage
```

После чего требуется запустить на исполнение базу данных в управляющем окне AWS (рисунок 3.7 и рисунок 3.8). В качестве импортируемой СУБД требуется выбрать MySQL.

					ПГУ 1.090501.03.001 ПЗ	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

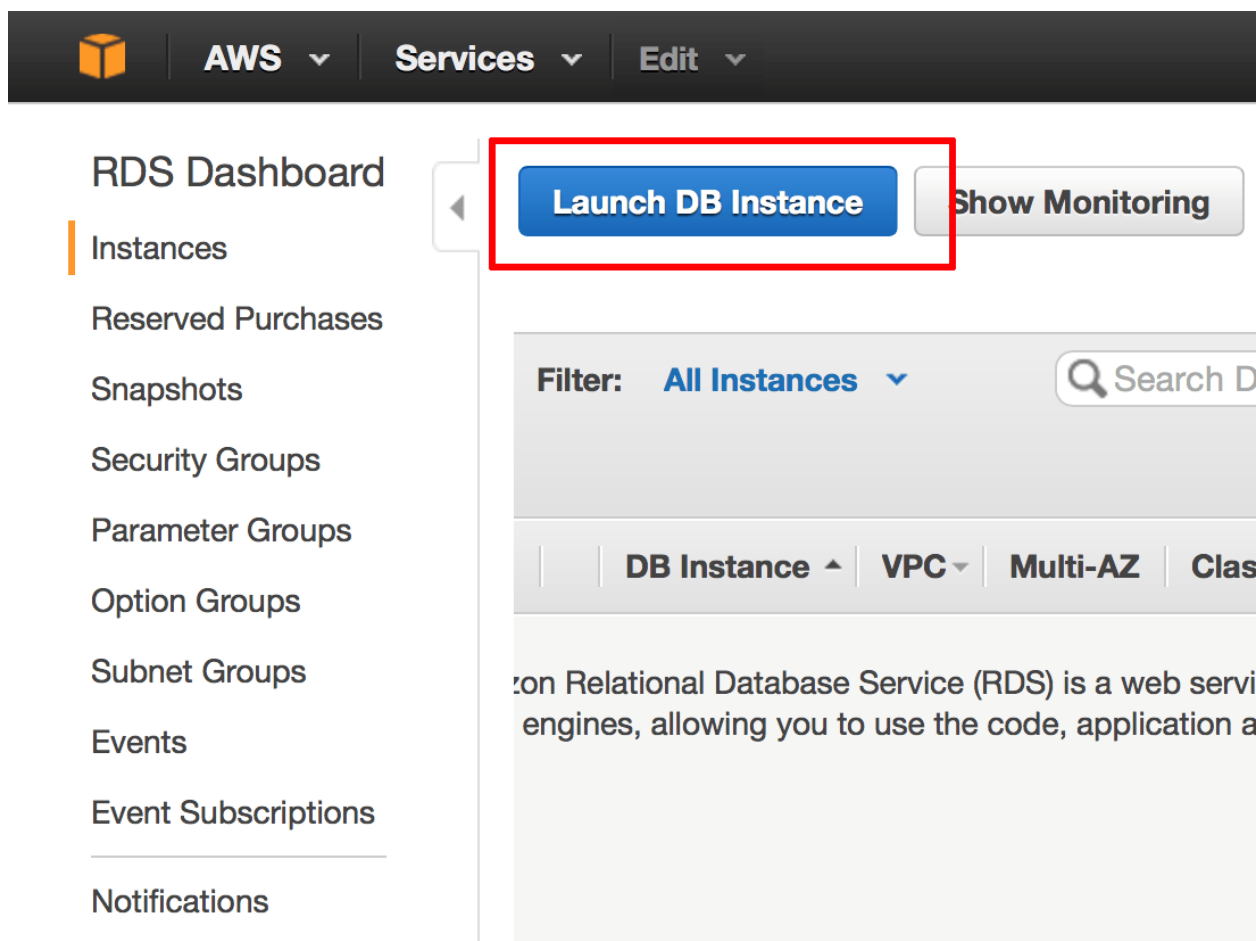


Рисунок 3.7 – Импортрование БД в облачную среду (шаг 1)

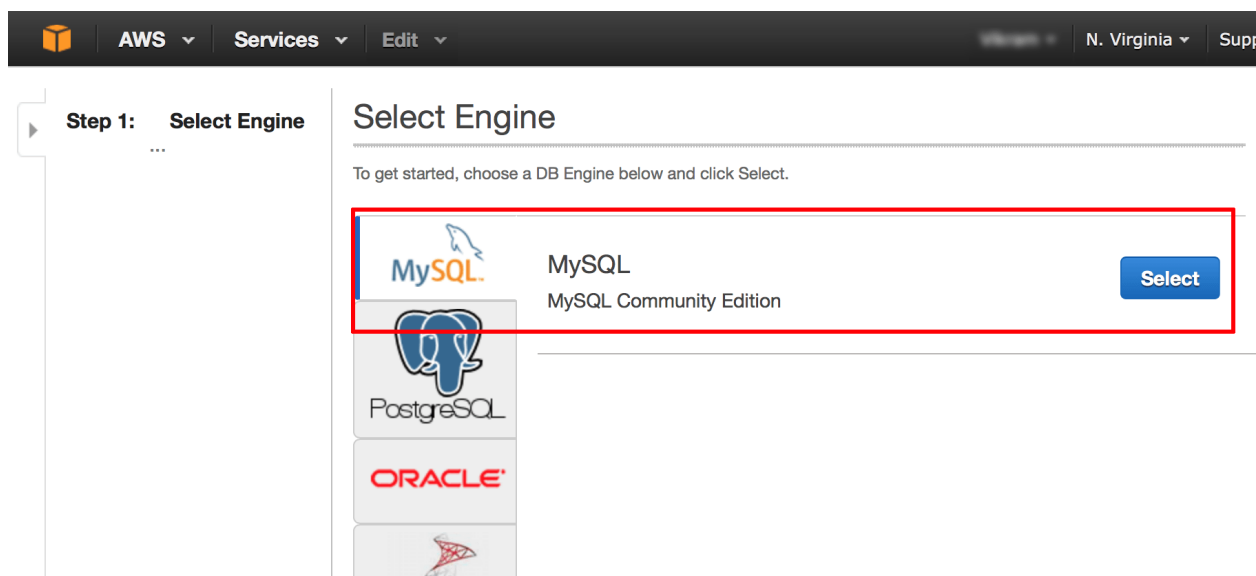


Рисунок 3.8 – Импортрование БД в облачную среду (шаг 2)

После выбора кнопкой «Select» в консоли IDE Eclipse появится сообщение

```

Uploading: [#####] 100%
Done...

Environment details for: java-boot-dev
Application name: java-jpa-mysql
Region: ru
Deployed Version: app-cf24-230529_112238
Environment ID: e-tbmssip4un
Platform: aws:elasticbeanstalk:ru:platform/Java 8 running on 64bit
Amazon Linux/2.10.4
Printing Status: ready
INFO      createEnvironment is starting.
INFO      Created RDS database named: eradpl

```

Открыть консоль AWS, в которой можно запустить развернутое в облаке приложение на исполнение, необходимо командой

```
$eb console
```

Для этого в появившейся web-консоли проверяются параметры запуска приложения (рисунок 3.9).

Name	Value
GRADLE_HOME	/usr/local/gradle
JAVA_HOME	/usr/lib/jvm/java
JAVA_HOME	/usr/local/apache-maven/bin
JAVA_HOME	/usr/local/apache-maven
SERVER_PORT	5000
SPRING_DATASOURCE_PAS	12345678
SPRING_DATASOURCE_URL	jdbc:mysql://
SPRING_DATASOURCE_USE	root
SPRING_JPA_DATABASE_PL	org.hibernate.dialect.MySQL5C
SPRING_JPA_HIBERNATE_D	update

Рисунок 3.9 – Настройки AWS для запуска Java-приложения

Далее в консоли требуется выполнить «Запуск на исполнение» (рисунок 3.10) с помощью команды

```
$eb run
```

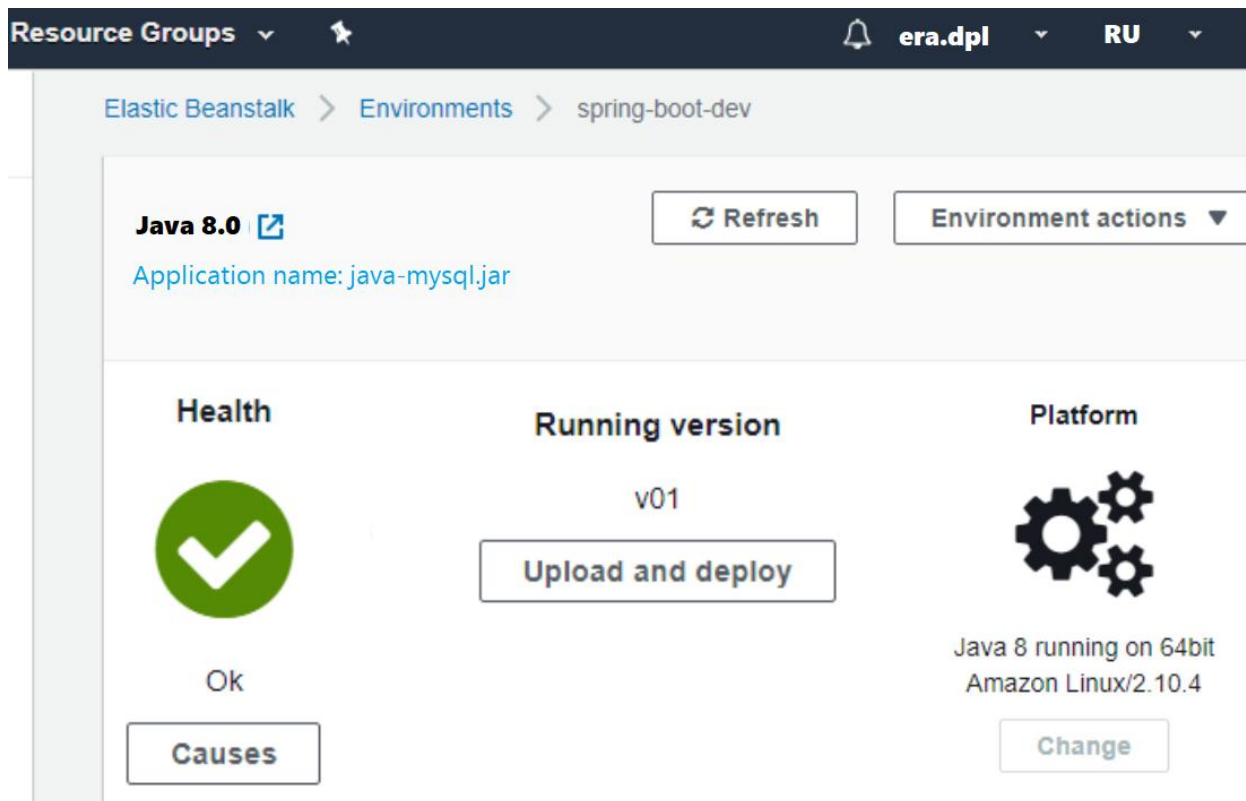


Рисунок 3.10 – Запущенное приложение

Проверкой правильной настройки «экосистемы» приложения будет запуск приложения в браузере, если откроется приветственная страница, значит, приложение установлено корректно.

4 Технико-экономическое обоснование разработки программы

Разработка программного обеспечения осуществлена на базе собственной производственной площади общей площадью 45 кв.м. Содержание помещения в месяц составляет 7 000 рублей.

Для разработки программного обеспечения приобретена вычислительная техника, лицензионное программное обеспечение и расходные материалы (обучающие материалы).

Данные по потребности в оборудовании и лицензионном программном обеспечении представлены соответственно в таблицах 4.1 и 4.2.

Таблица 4.1 – Потребность в оборудовании

Оборудование	Источник	Кол-во	Ориентировочная цена за единицу, руб.	Потребляемая мощность, Ватт
Компьютер iRU 525	ООО «ДНС»	5	60000	500
Клавиатура и мышь	ООО «ДНС»	5	890	2,5
Монитор Samsung GMN27	ООО «ДНС»	5	35600	30

Таблица 4.2 – Потребность в лицензионном программном обеспечении

Лицензионное программное обеспечение	Источник	Цена, рублей	Срок действия лицензии
Подписка Amazon Web Services Base	Производитель Amazon	80000	3 года
Антивирус Kaspersky 2023/2024	Интернет-магазин ООО «Первый Бит»	1999	365 дней
Microsoft Windows Pro 11	Интернет-магазин ООО «Первый Бит»	32000	Бессрочная

Перечень дополнительных материалов, приобретение которых не требуется, приведены в таблице 4.3.

Таблица 4.3 – Потребность в дополнительных материалах

Материал	Источник	Количество, штук
Стол рабочий с надстройкой	ООО «Канцлер»	1
Настольная лампа	ООО «Канцлер»	5
Стол компьютерный	ООО «ДНС»	4
Кресло	ООО «ДНС»	4
Канцтовары	ООО «Канцлер»	5
Бумага	ООО «Канцлер»	5

Разработка программного обеспечения осуществлялась командой разработчиков. Состав коллектива разработчиков и информация по заработной плате разработчиков программного обеспечения и их количественному составу приведены в таблице 4.4.

Таблица 4.4 – Занятость и заработная плата разработчиков

Специалист	Количество, человек	Сумма заработной платы, рублей
Руководитель проекта	1	100000
Системный архитектор	1	80000
DevOps со знанием облачных технологий и их интеграцией	1	120000
Программист	1	80000
Тестировщик	1	60000

Себестоимость разработки программного продукта включает следующие статьи затрат:

- затраты на приобретение оборудования;
- затраты на приобретение лицензионного программного обеспечения;
- затраты на дополнительные материалы;
- затраты на заработную плату;
- затраты на обслуживание.

Затраты на оборудование в период разработки составят 482450 рублей на основании данных в таблице 4.1.

Затраты на программное обеспечение в период разработки составят 113999 рублей и представлены в таблице 4.2.

Затраты на дополнительные материалы в период разработки составят 0 рублей в соответствии с условием, что потребности в закупке нет.

Затраты на заработную плату включают затраты на основную и дополнительную заработную плату разработчиков программного обеспечения и отчисления на страховые взносы.

Затраты на основную заработную плату разработчиков определяются исходя из установленной заработной платы, количества специалистов и их занятости в процессе разработки программного продукта (продолжительность работы в днях).

Дополнительная заработная плата составляет 15% от затрат на основную заработную плату разработчиков.

С учетом выходных дней при 5-дневной рабочей неделе количество рабочих дней в месяце в среднем составляет 21 день.

Затраты на основную и дополнительную заработную плату разработчиков программного обеспечения представлены в таблице 4.5.

					ПГУ 1.090501.03.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 4.5 – Затраты на основную и дополнительную заработную плату разработчиков

Исполнитель	Количество, человек	Сумма заработной платы, рублей	Период занятости, дней	Основная заработная плата, рублей	Дополнительная заработная плата, рублей	Всего рублей
Руководитель проекта	1	100000	42	240000	30000	230000
Системный архитектор	1	80000	42	180000	24000	184000
DevOps со знанием облачных технологий и их интеграцией	1	120000	42	240000	36000	276000
Программист	1	80000	42	180000	24000	184000
Тестировщик	1	60000	42	150000	18000	138000
Итого	5	440000	210	880000	132000	1012000

Таким образом, затраты на заработную плату разработчиков составят 1012000 рублей.

Отчисления на страховые взносы по нормам, установленным законодательством в ПРФ, ФОМС, ФСС РФ вычисляются, исходя из данных, представленных ниже.

Отчисления на страховые взносы производятся согласно «Налоговому кодексу Российской Федерации (часть вторая)» от 05.08.2000 № 117-ФЗ (ред. от 26.03.2022).

Действующие тарифы на страховые взносы работодателей установлены статьей 34 НК РФ и составляют:

- ОПС (обязательное пенсионное страхование) – 22%;

- ВНиМ (страхование временной нетрудоспособности и в связи с материнством) – 2,9%;

- ОМС (обязательное медицинское страхование) – 5.1%.

Общая сумма отчислений на страховые взносы составит 303600 рублей.

Общая сумма затрат является суммой: затрат на оплату труда, включая налоги и сборы, затраты на закупку оборудования и лицензионного ПО, и составляет 1912049 рублей.

При этом заказчик получает масштабируемый гибкий программный продукт размещение и доступ к которому не зависят от локации клиентов.

					ПГУ 1.090501.03.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

5 Безопасность жизнедеятельности. Оценка

травмобезопасности на рабочем месте оператора ПК

Травмобезопасность – свойство рабочих мест соответствовать требованиям безопасности труда, исключающим травмирование работающих в условиях, установленных нормативно-правовыми актами.

Травмобезопасность рабочих мест обеспечивается исключением повреждений частей тела человека, которые могут быть получены в результате воздействия:

- движущихся предметов механизмов или машин, а также неподвижными их элементами на рабочем месте (при механическом воздействии). Такими предметами являются подвижные столы, вращающиеся детали, органы управления;
- электрического тока, источником поражения могут быть незащищенные и неизолированные электропровода, поврежденные электродвигатели, открытые коммутаторы, не заземленное оборудование;
- повреждения, полученные при падениях.

Падения подразделяются на два вида: падения на человека различных предметов и падения человека в результате подскользывания, запинания, падения с высоты или внезапного ухудшения здоровья.

При эксплуатации ЭВМ травмы могут быть получены в результате воздействия электрического тока и повреждений, полученных при падениях.

Электрические установки, к которым относится практически все оборудование ЭВМ, представляют для человека большую потенциальную опасность, так как в процессе эксплуатации или проведении профилактических работ человек может коснуться частей, находящихся под напряжением.

Специфическая опасность электроустановок заключается в том, что токоведущие проводники, корпуса стоек ЭВМ и прочего оборудования, оказавшегося под напряжением в результате повреждения (пробоя) изоляции, не подают каких-либо сигналов, которые предупреждают человека об

					ПГУ 1.090501.03.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

опасности. Реакция человека на электрический ток возникает лишь при протекании последнего через тело человека. Важное значение для предотвращения электротравматизма имеет правильная организация обслуживания действующего оборудования ЭВМ, проведения ремонтных, монтажных и профилактических работ.

При этом под правильной организацией понимается строгое выполнение ряда организационных и технических мероприятий и средств, установленных действующими «Правилами технической эксплуатации электроустановок потребителей и правилами техники безопасности при эксплуатации электроустановок потребителей» (ПТЭ и ПТБ потребителей) и «Правилами установки электроустановок» (ПУЭ). В зависимости от категории помещения необходимо принять определенные меры, обеспечивающие электробезопасность при эксплуатации и ремонте электрооборудования.

Помещение вычислительных центров и офисов по опасности поражения электрическим током можно отнести к 1 классу, т.е. это помещение без повышенной опасности (сухое, беспыльное, с нормальной температурой воздуха, изолированными полами и малым числом заземленных приборов).

На рабочем месте оператора ЭВМ из всего оборудования металлическим является лишь корпус системного блока компьютера, но здесь используются системные блоки, в которых кроме рабочей изоляции предусмотрен элемент для заземления и провод с заземляющей жилой для присоединения к источнику питания.

Системный блок компьютера имеет цифровые и аналоговые микросхемы запитываются постоянными напряжениями ± 5 и ± 12 В, которые получаются путем преобразования переменного напряжения 220В в блоке питания.

Блок питания содержит в себе схемы преобразования напряжения, схемы стабилизации и схему защитного отключения при коротком замыкании.

					ПГУ 1.090501.03.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

Так как корпус компьютера выполнен из металла, то существует опасность пробоя фазы на корпус. Мониторы современных компьютеров практически всегда изготавливаются из пластика, поэтому несмотря на большое напряжение, присутствующее в мониторе, поражение током человека практически исключено.

Электробезопасность помещения обеспечивается в соответствии с ПУЭ. Опасное и вредное воздействие на людей электрического тока, электрической дуги и электромагнитных полей проявляется в виде электротравм и профессиональных заболеваний.

Степень опасного и вредного воздействия на человека электрического тока, электрической дуги и электромагнитных полей зависит от:

- рода и величины напряжения и тока;
- частоты электрического тока;
- пути тока через тело человека;
- продолжительности воздействия на организм человека.

Рассмотрим основные причины поражения человека электрическим током на рабочем месте:

- прикосновение к металлическим нетоковедущим частям (корпусу, периферии компьютера), которые могут оказаться под напряжением в результате повреждения изоляции;
- нерегламентированное использование электрических приборов;
- отсутствие инструктажа сотрудников по правилам электробезопасности.

Электробезопасность на рабочем месте оператора ЭВМ обеспечивается техническими способами и средствами защиты, а также организационными и техническими мероприятиями.

Для обеспечения защиты от поражения электрическим током при прикосновении к металлическим нетоковедущим частям, которые могут оказаться под напряжением в результате повреждения изоляции, необходимо применять защитное заземление.

					ПГУ 1.090501.03.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

Заземление корпуса ЭВМ обеспечено подведением заземляющей жилы к питающим розеткам. Сопротивление заземления 4 Ом, согласно (ПУЭ) для электроустановок с напряжением до 1000 В.

Основным организационным мероприятием является инструктаж и обучение безопасным методам труда, а так же проверка знаний правил безопасности и инструкций в соответствии с занимаемой должностью применительно к выполняемой работе.

					ПГУ 1.090501.03.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения дипломного проекта было сформировано техническое задание на разработку программного приложения для обработки заказов на основе архитектуры облачных сервисов.

Был произведен выбор инструментария, в частности выбор облачного сервиса как контейнера среды исполнения проекта.

Выполнено проектирование архитектуры приложения и базы данных, в результате чего были разработаны UML-диаграммы разрабатываемого ПО и структура базы данных.

В результате реализации модулей приложения, были разработаны блоки интерфейса, логического слоя приложения и модули взаимодействия с СУБД.

Приложение реализовано в соответствии с техническим заданием.

В разделе безопасности жизнедеятельности была произведена оценка травмобезопасности на рабочем месте оператора ПК.

В экономическом разделе был произведен расчет трудоемкости и затрат на разработку программного обеспечения с размещением в облачном контейнере исполнения.

Задание на дипломное проектирование выполнено в полном объеме.

					ПГУ 1.090501.03.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Кораблев В.А. Самоучитель Spring Boot. - СПб.: Мир Издательская группа BHV, 2019. - 528 с.
2. Лобанов Д. С. Программирование на Java. - Москва: Символ-Плюс, 2020.- 688 с.
3. Майо Д. MySQL. Энциклопедия программиста.- Москва: «ДивоСофт», 2019.- 656 с.
4. Матросов Д. С. Web разработка. Быстрый старт.- М.: Бином, 2018.- 384 с.
5. Микелсен К. IDE Eclipse инструменты упражнения. Учебник. - Екатеринбург: «Град», 2019. - 656 с.
6. Михеев П.А. phpMyAdmin для администраторов - Москва, БХВ, 2017 г.- 144 с.

					ПГУ 1.090501.03.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А

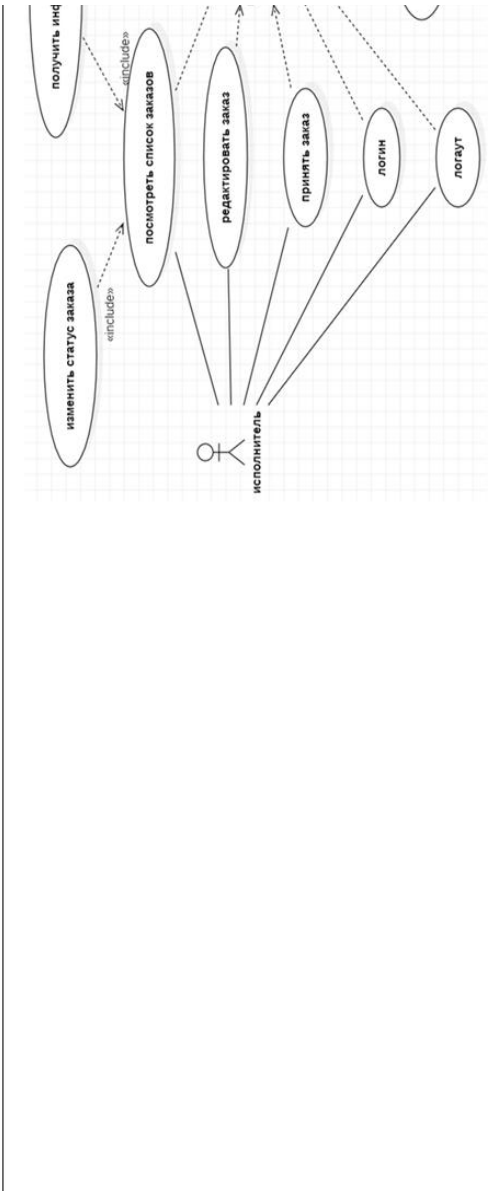
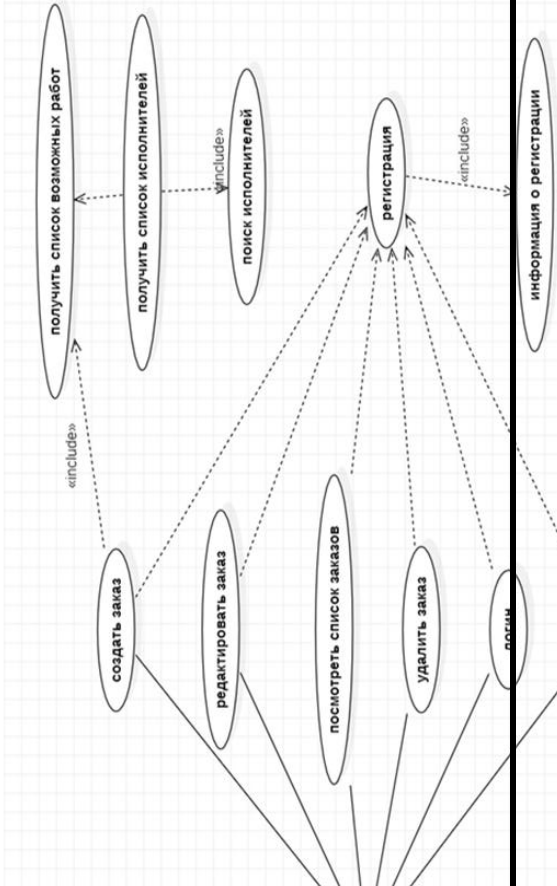
(обязательное)

Приложение для обработки онлайн-заказов на базе облачных сервисов

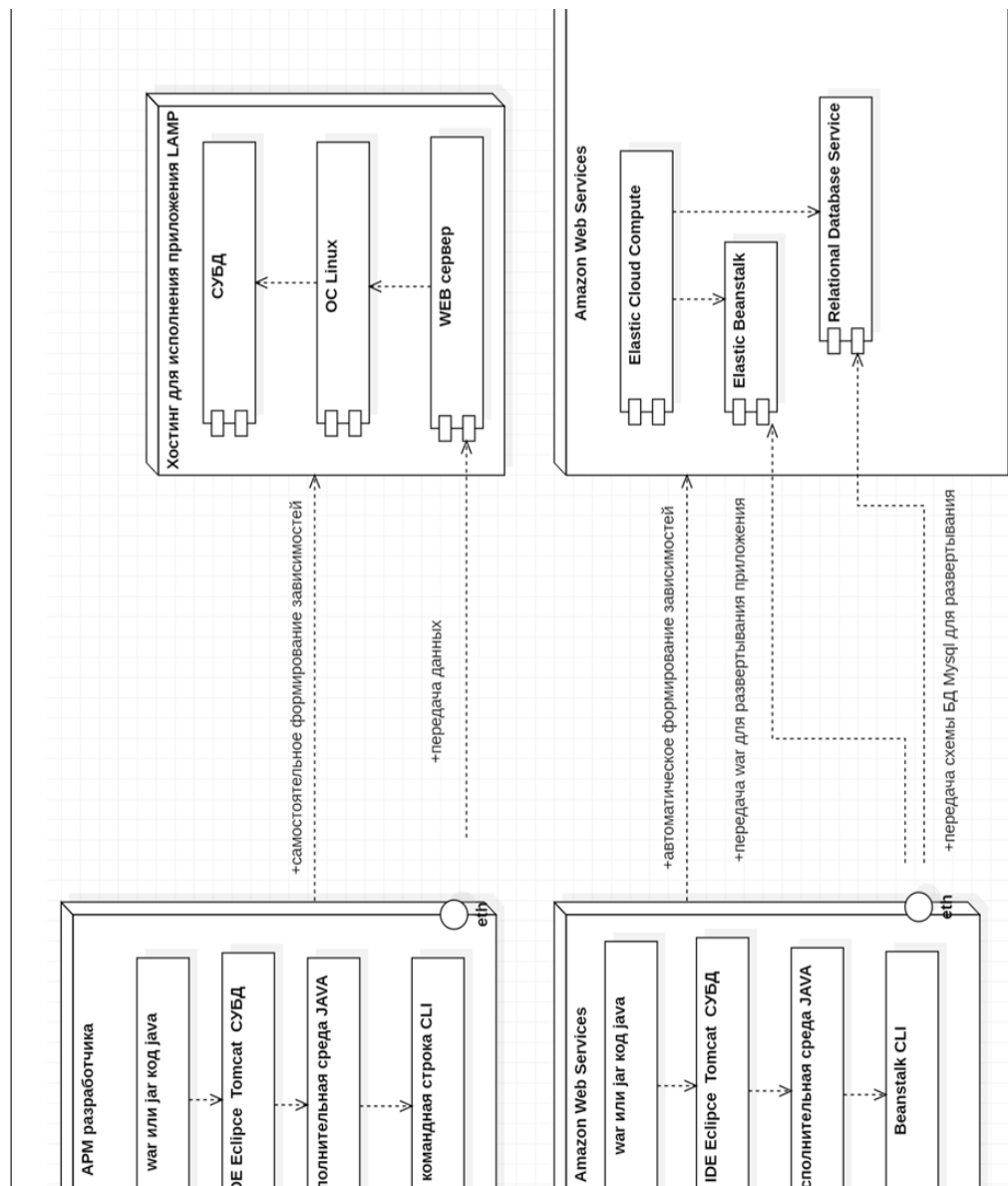
UML-диаграммы

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

Приложение А.1 – Диаграмма вариантов использования



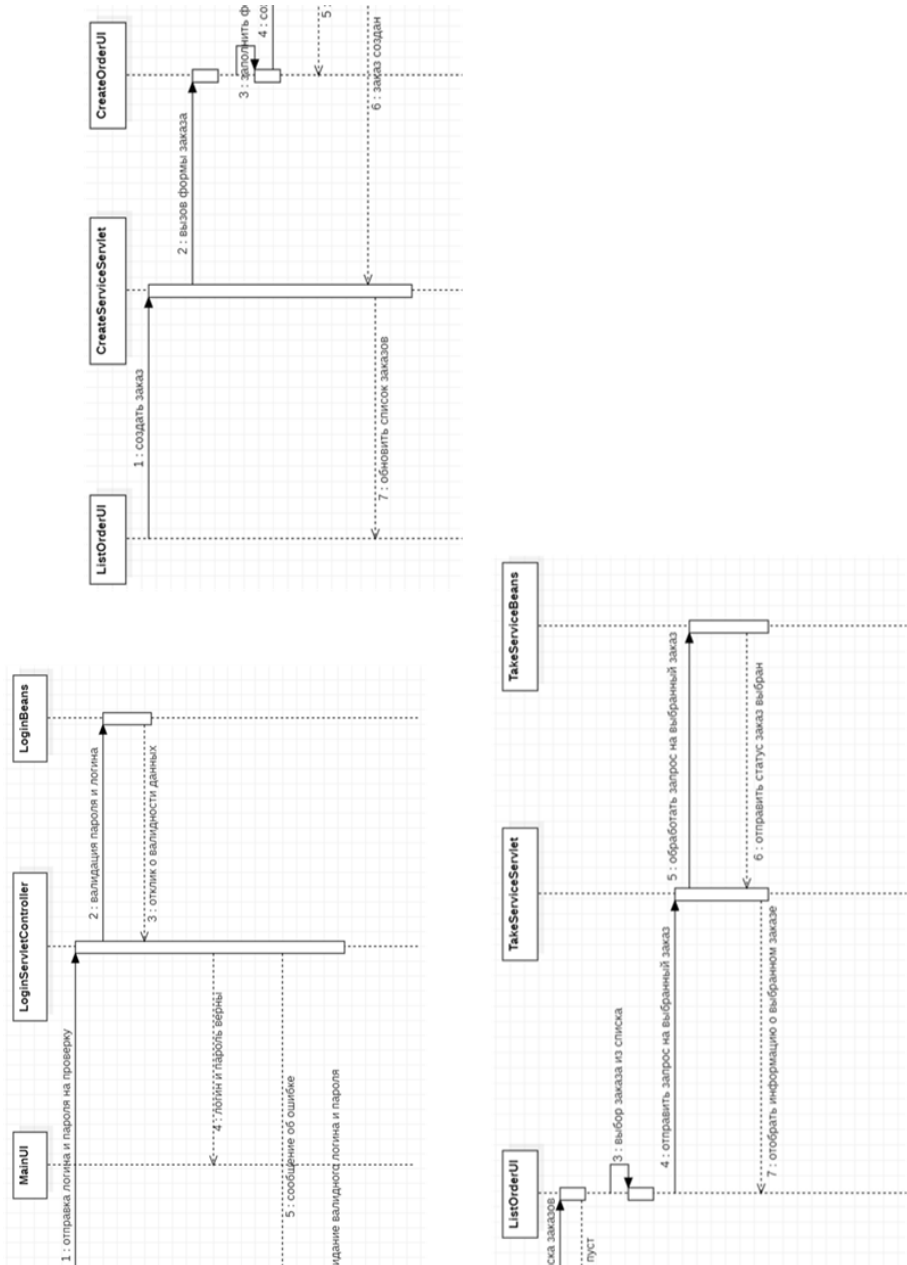
Приложение А.2 – Диаграмма развертывания



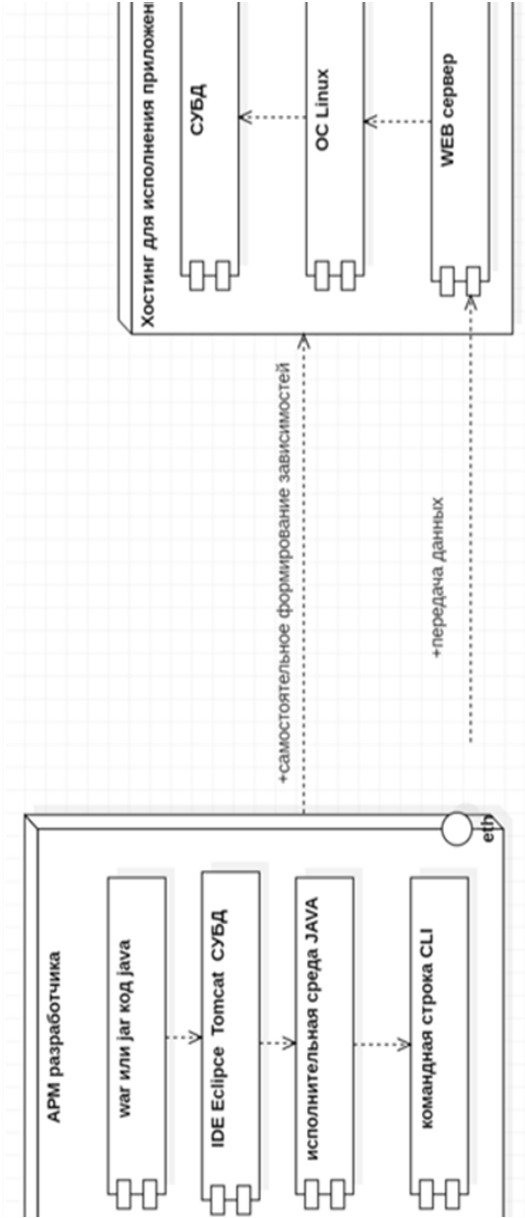
Приложение А.3 – Диаграмма последовательности

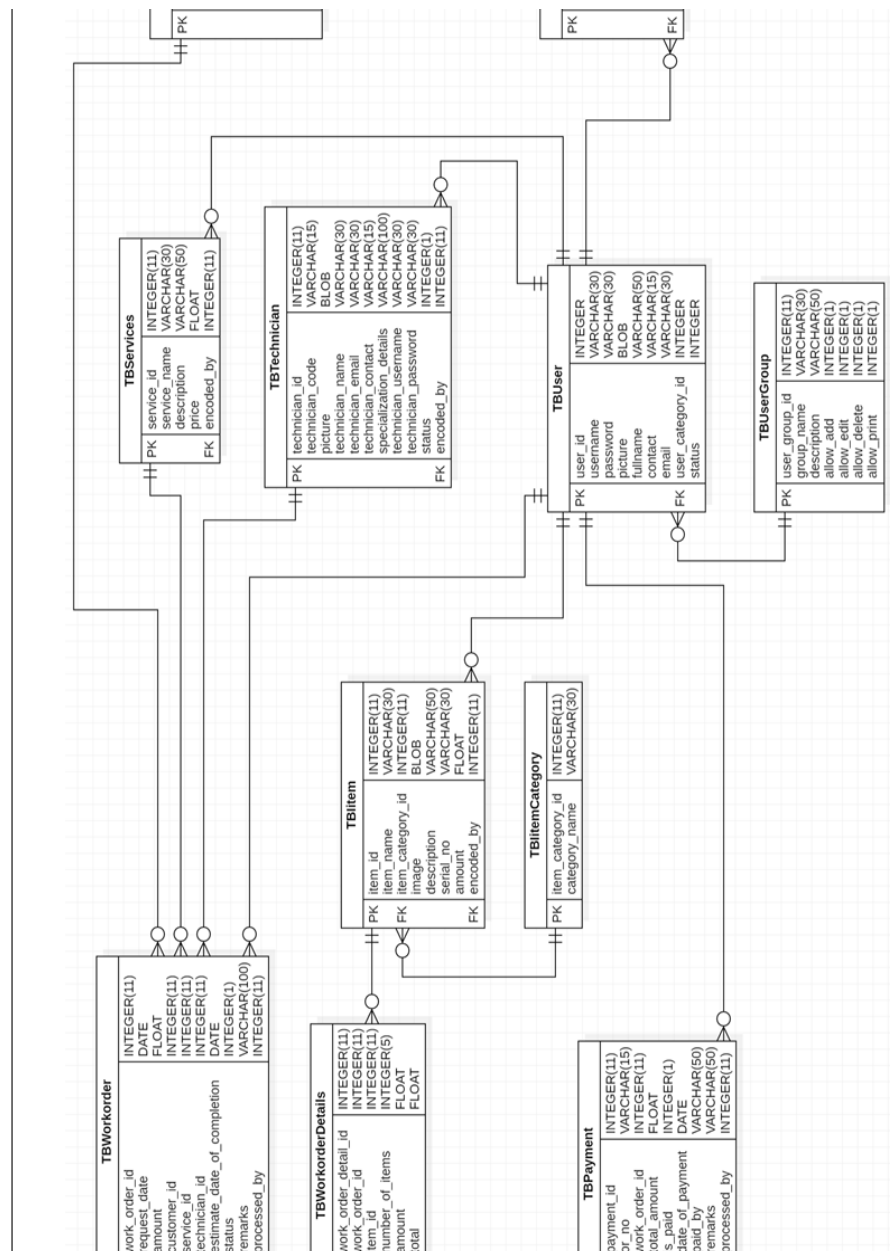
Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

Приложение А.4 – Диаграмма развертывания



Приложение А.5 – Диаграмма данных





Приложение Б

(обязательное)

Приложение для обработки онлайн-заказов на базе облачных сервисов

Листинг программы

Изм.	Лист	№ докум.	Подпись	Дата

Модуль UserController.java

```
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.security.Principal;

@Controller
public class UserController {
    private final UserService userService;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/user")
    public String userPanel(Principal principal, Model
model){
        User user =
userService.findByUsername(principal.getName());

        if (user != null) {
            model.addAttribute("user", user);
        }else {
            return "error/404";
        }

        return "user";
    }
}
```

Модуль LoginController.java

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
```

					ПГУ 1.090501.03.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

```

import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class LoginController {

    @GetMapping("/login")
    public String login(Model model, String error){
        if (error != null)
            model.addAttribute("error", "Некорректный логин
или пароль.");
        return "login";
    }
}

```

Модуль RegisterController.java

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class RegisterController {
    private static final Logger logger =
LoggerFactory.getLogger(RegisterController.class);
    private final UserService userService;
    private final UserValidator userValidator;

    @Autowired
    public RegisterController(UserService userService,
UserValidator userValidator) {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

        this.userService = userService;
        this.userValidator = userValidator;
    }
    @GetMapping("/register")
    public String registration(Model model) {
        model.addAttribute("userForm", new User());
        return "register";
    }
    @PostMapping("/register")
    public String registration(@ModelAttribute("userForm")
User userForm, BindingResult bindingResult) {
        userValidator.validate(userForm, bindingResult);
        if (bindingResult.hasErrors()) {

logger.error(String.valueOf(bindingResult.getFieldError()));
            return "register";
        }
        userService.save(userForm);
        userService.login(userForm.getUsername(),
userForm.getPasswordConfirm());
        return "redirect:/home";
    }
}

```

Модуль ServiceImpl.java

```

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class tServiceImpl implements ServService {
    private final ServiceRepository servicetRepository;
    @Autowired
    public ServiceServiceImpl(ServiceRepository
serviceRepository){
        this.sericeRepository = serviceRepository;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

    @Override
    public void save(Service service) {
        serviceRepository.save(service);
    }

    @Override
    public void edit(long id, Service newService) {
        Service found = serviceRepository.findOne(id);
        found.setName(newService.getName());
        found.setImageUrl(newService.getImageUrl());
        found.setDescription(newService.getDescription());
        found.setPrice(newProduct.getPrice());
        save(newService);
    }

    @Override
    public void delete(long id) {
        productRepository.delete(findById(id));
    }

    @Override
    public Service findById(long id) {
        return serviceRepository.findById(id);
    }

    @Override
    public List<Service> findAllByOrderByIdAsc() {
        return serviceRepository.findAllByOrderByIdAsc();
    }

    @Override
    public List<Service> findAllByCategoryId(long categoryId)
    {
        return
serviceRepository.findAllByCategoryId(categoryId);
    }

    @Override
    public long count() {
        return serviceRepository.count();
    }

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

    Модуль UserDetailsServiceImpl.java

    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    import
org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.security.core.GrantedAuthority;
    import
org.springframework.security.core.authority.SimpleGrantedAuthority
;
    import
org.springframework.security.core.userdetails.UserDetails;
    import
org.springframework.security.core.userdetails.UserDetailsService;
    import
org.springframework.security.core.userdetails.UsernameNotFoundException;
    import org.springframework.stereotype.Service;
    import
org.springframework.transaction.annotation.Transactional;

    import java.util.HashSet;
    import java.util.Objects;
    import java.util.Set;

    @Service
    public class UserDetailsServiceImpl implements
UserDetailsService {
        private static final Logger logger =
LoggerFactory.getLogger(UserDetailsServiceImpl.class);
        private final UserRepository userRepository;
        @Autowired
        public UserDetailsServiceImpl(UserRepository
userRepository) {
            this.userRepository = userRepository;
        }

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

@Override
@Transactional(readonly = true)
public UserDetails loadUserByUsername(String username) {
    User user = userRepository.findByUsername(username);
    if (user != null) {
        Set<GrantedAuthority> authorities = new
HashSet<>();
        if (Objects.equals(username, "заказчик")) {
            authorities.add(new
SimpleGrantedAuthority("ROLE_ADMIN"));
        }else {
            authorities.add(new
SimpleGrantedAuthority("ROLE_USER"));
        }
        logger.debug(String.format("User with name: %s
and password: %s created.", user.getUsername(),
user.getPassword()));
        return new
org.springframework.security.core.userdetails.User(user.getUsenam
e(), user.getPassword(), authorities);
    }else{
        throw new
UsernameNotFoundException("Пользователь" + username + " not
found!");
    }
}
}

```

Модуль OrderItem.java

```

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;
import
org.springframework.data.annotation.PersistenceConstructor;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.math.BigDecimal;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

```

import java.util.Calendar;

@Entity
@Table(name = "items")
class OrderItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @OneToOne
    private Product product;
    @NotNull
    private int quantity;
    @NotNull
    private BigDecimal price;
    @CreationTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    private Calendar createdAt;
    @UpdateTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    private Calendar updatedAt;

    @PersistenceConstructor
    protected OrderItem() {
    }
    OrderItem(final int quantity, final Product product,
final BigDecimal price) {
        this.quantity = quantity;
        this.product = product;
        this.price = price;
    }
    int getId() {
        return id;
    }
    void setId(final int id) {
        this.id = id;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

int getQuantity() {
    return quantity;
}

void setQuantity(final int quantity) {
    this.quantity = quantity;
}

Product getProduct() {
    return product;
}

void setProduct(final Product product) {
    this.product = product;
}

BigDecimal getPrice() {
    return price;
}

void setPrice(final BigDecimal price) {
    this.price = price;
}

Calendar getCreatedAt() {
    return createdAt;
}

void setCreatedAt(final Calendar createdAt) {
    this.createdAt = createdAt;
}

Calendar getUpdatedAt() {
    return updatedAt;
}

void setUpdatedAt(final Calendar updatedAt) {
    this.updatedAt = updatedAt;
}
}

```

Модуль DeleteController.java

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

```

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.core.annotation.AuthenticationPrincip
al;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/orders")
class OrderController {
    private final Logger logger =
LoggerFactory.getLogger(OrderController.class);
    private final OrderFacade facade;
    private final OrderQueryRepository queryRepository;

    OrderController(final OrderFacade facade, final
OrderQueryRepository queryRepository) {
        this.facade = facade;
        this.queryRepository = queryRepository;
    }

    @GetMapping("/{id}")
    ResponseEntity<OrderDto> readSomeOrder(
        @AuthenticationPrincipal Customer customer,
        @PathVariable int id
    ) {
        var result = queryRepository.findDtoByIdAndCustomer(id,
customer)
            .orElseThrow(() -> new ApiNotFoundException("Заказ не
существует"));
        return new ResponseEntity<>(result, HttpStatus.OK);
    }

    @GetMapping
    ResponseEntity<List<OrderDto>> readAllCustomerOrders(

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

    @AuthenticationPrincipal Customer customer
    ) {
        var result = queryRepository.findAllDtoByCustomer(customer);
        return new ResponseEntity<>(result, HttpStatus.OK);
    }

    @PostMapping
    ResponseEntity<OrderDto> createOrder(@AuthenticationPrincipal
Customer customer) {
        var result = facade.placeOrder(customer);
        logger.info("Заказчик(id): " + customer.getId() + " удалить
заказ");
        return new ResponseEntity<>(result, HttpStatus.CREATED);
    }

    @PostMapping(value = "/checkout", params = "orderId")
    ResponseEntity<StripeDto> createCheckout(@RequestParam int
orderId) throws StripeException {
        var result = facade.createSession(orderId);
        var response = new StripeDto(result.getId());
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    @PatchMapping(value = "/check-session", params = "orderId")
    ResponseEntity<Void> checkSession(@RequestParam int orderId)
throws StripeException {
        facade.checkSession(orderId);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }

    @DeleteMapping("/{id}")
    ResponseEntity<Void> deleteOrder(
    @AuthenticationPrincipal Customer customer,
    @PathVariable int id
    ) {
        facade.cancelOrder(customer, id);
        logger.info("Заказчик(id): " + customer.getId() + " заказ
отменен!");
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(обязательное)

Приложение для обработки онлайн-заказов на базе облачных сервисов

Презентация

					ПГУ 1.090501.03.001 ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

работы:
студент гр. 18ВОЭ1 Ерина Алина Германовна
ный руководитель:
к.т.н., доцент Федюнин Роман Николаевич

					ПГУ 1.090501.03.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

Цель работы и объекты

Цель работы – Разработать приложение приема и обработки заказов мастерской по ремонту вычислительной и орг. техники.

Объектом проекта является клиент-серверное приложение, выполняющие следующие функции:

- Многопользовательский доступ к функциям приложения;
- работа в режиме заказчик/исполнитель.

Задачи приложения

Реализовать приложение с возможностью размещения и работой на хостинге облачного хранилища.

Приложение должно исполняться в облачном хранилище.

База данных должна быть размещена в облачном хранилище.

Связь между приложением и БД должна быть обеспечена инструментарием хранилища.

Сравнение аналогов

Наименование	Плюс решения	Минус решения
1С Предприятие	возможность изменения стандартной конфигурации под текущие потребности; технологическое сопровождение; - ежемесячные обновления; - российская разработка; - стандартная возможность формирования отчетов	для доработки решения из «коробки» нужен специалист уровня и выше; отсутствие инструментария работы с графиками; отсутствие кроссплатформенной реализации; очень специфичный язык реализации; специфичная СУБД
Generix OMS	возможность разграничения доступа пользователей; шифрование данных; российская разработка; не требует установки на компьютер пользователя;	существует только «облачное» решение; для работы требуется устойчивый интернет; оптимален для малых и средних объемов заказов; - не совершенная техническая поддержка; - отсутствует конструктор отчетов;

Аналоги обладают недостатками: отсутствие решения для быстрого и мобильного развертывания приложения в облачном сервисе, для создания сервисов «клонов».

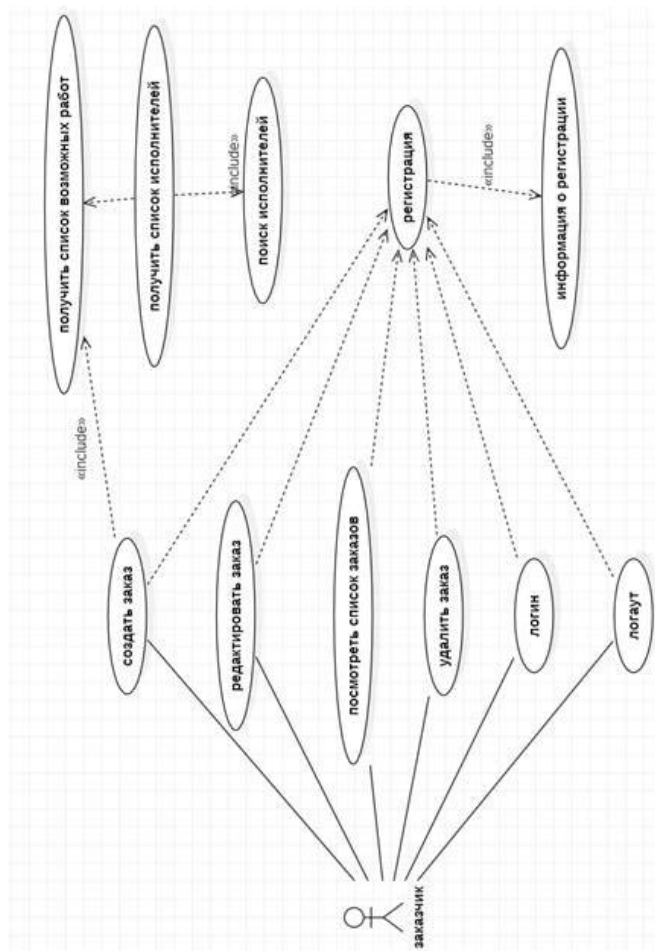
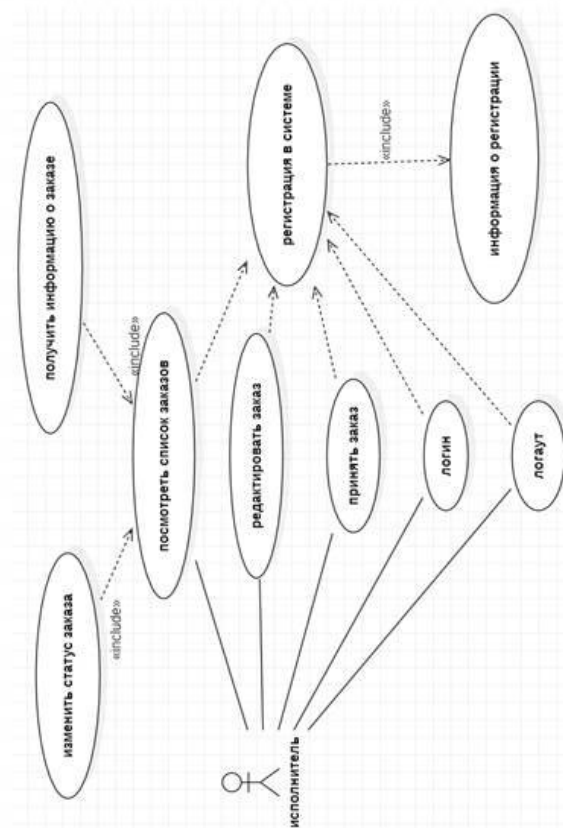
которые зачастую необходимы при развитии франчайзинговой сети.

Стек технологий

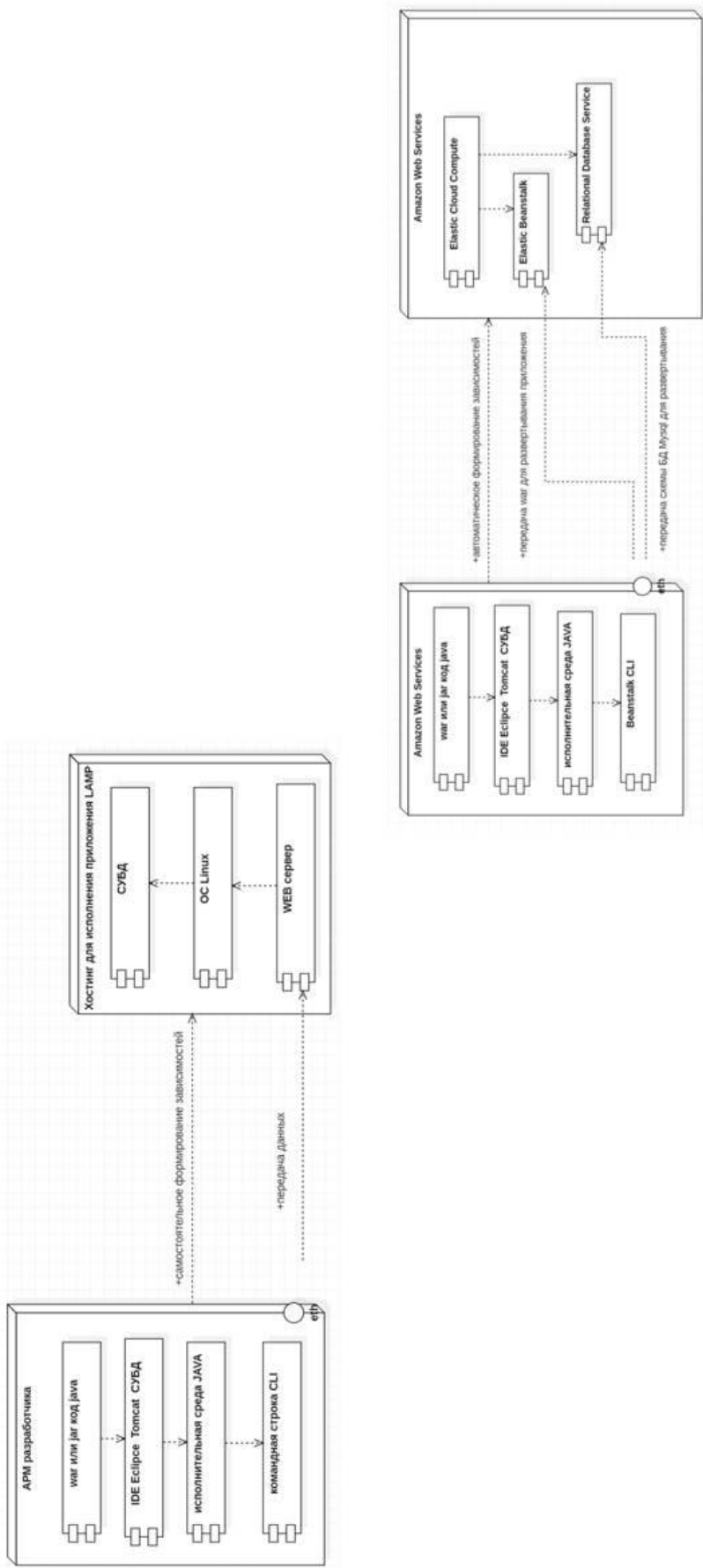


Изм.	Лист	№ докум.	Подпись	Дата

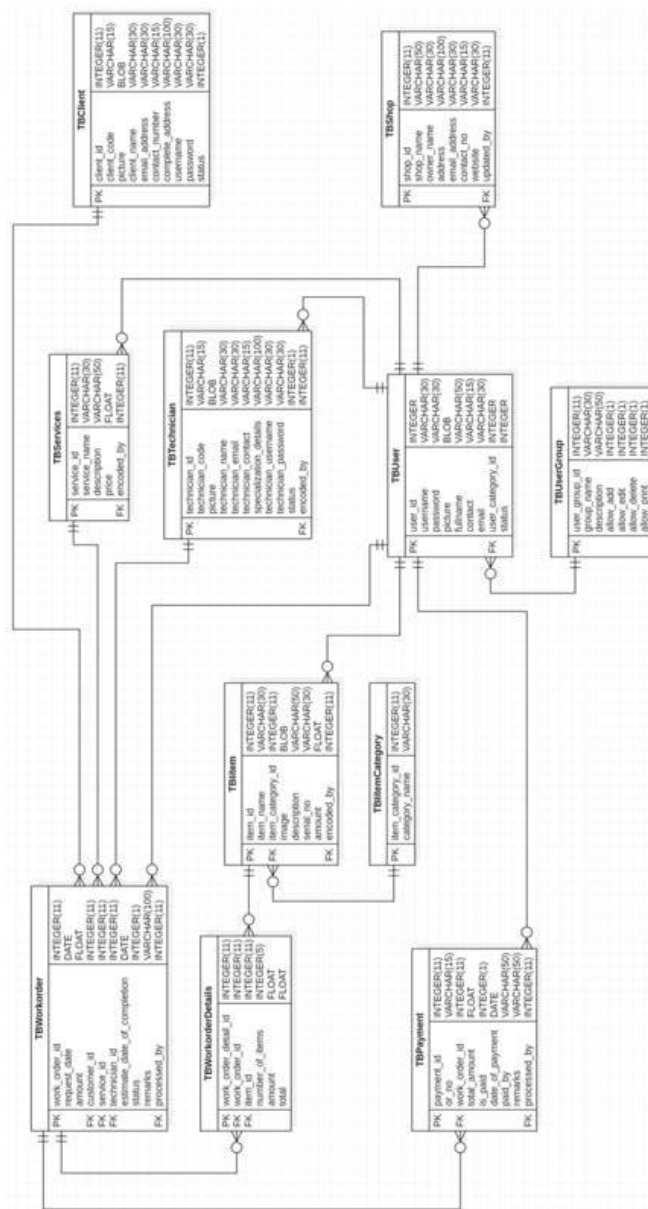
Основные функции приложения



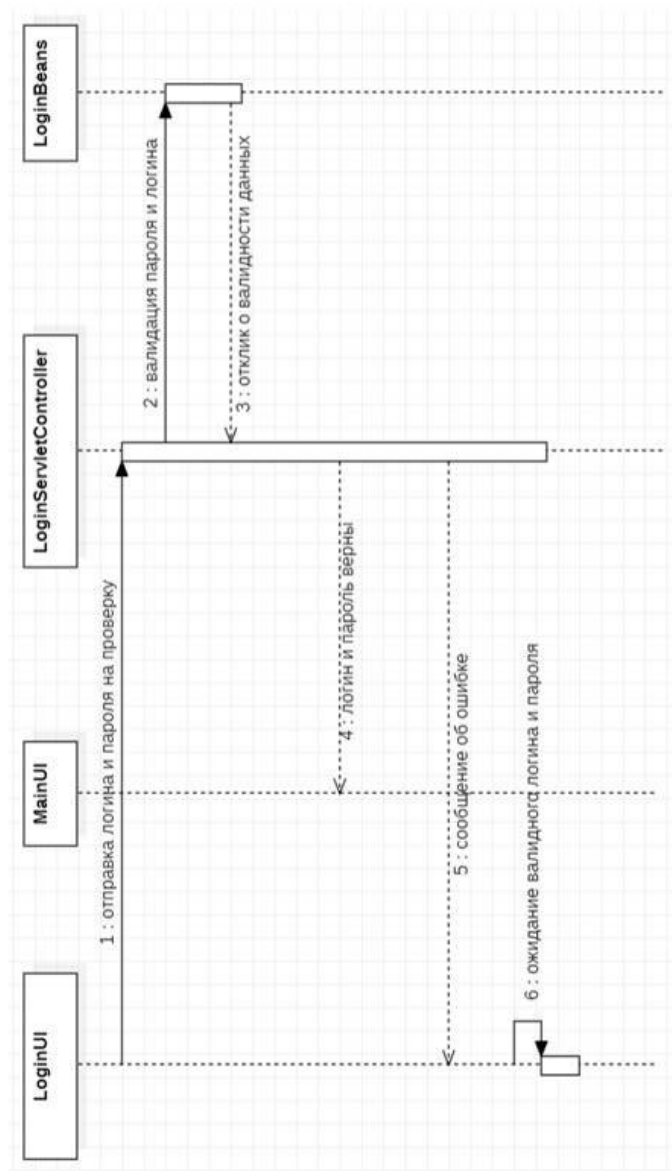
Архитектура приложения



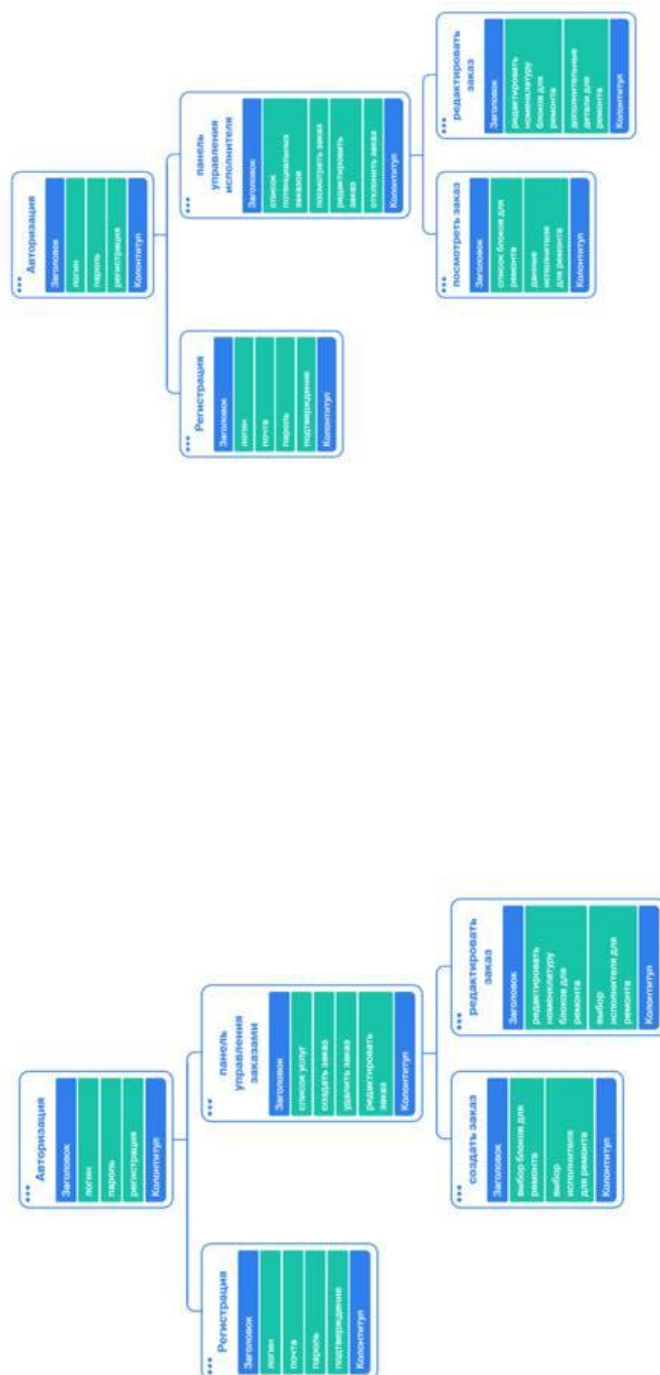
База данных приложения



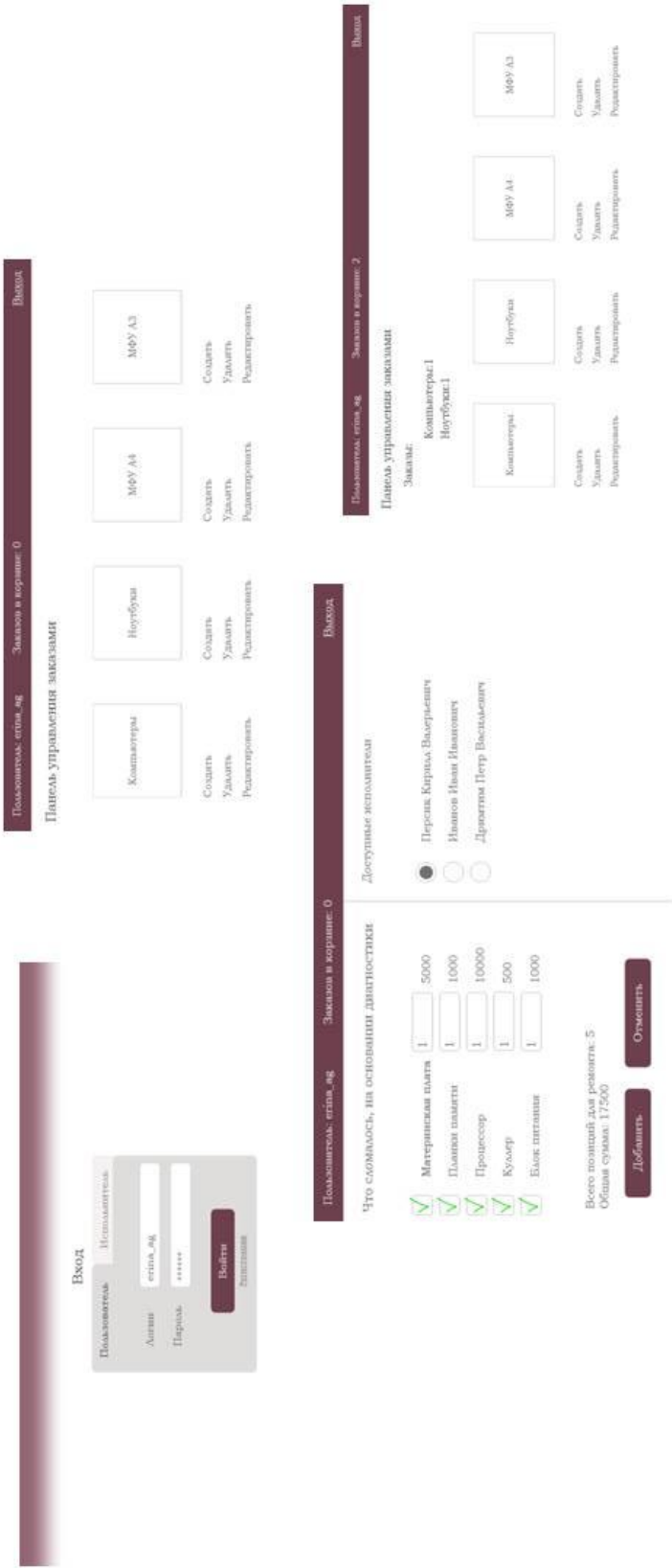
Работа приложения на примере авторизации пользователя



Структура сайта



Скриншоты запущенного приложения



Результаты работы

- Разработано мобильное приложение полностью соответствующее техническому заданию;
- Проведено функциональное тестирование (успешно);
- Намечены задачи по развитию функциональных возможностей программы.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		106

Спасибо за внимание!

					ПГУ 1.090501.03.001 ПЗ	Лист
						107
Изм.	Лист	№ докум.	Подпись	Дата		