МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет «Вычислительная техника»
(наименование)

Кафедра «Системы автоматизированного проектирования»
(наименование)

Направление подготовки 02.03.03 Математическое обеспечение и АИС
(код и наименование)

Профиль Администрирование информационных систем
(наименование)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему:

Разработка web-сервиса для записи клиентов в автомастерскую

Студент 25.05.24 (подпись, дата) Лысак С.В. (фамилия, инициалы)

Руководитель 28.05.24 (подпись, дата) Эпп В.В. (фамилия, инициалы)

Нормоконтролёр проф. каф. САПР (должность, место работы) 31.05.24 (подпись, дата) Бождай А.С. (фамилия, инициалы)

Работа допущена к защите

Заведующий кафедрой (подпись) Бурукинв И.П. (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от_____№____)

Секретарь ГЭК (подпись) Финогеев А.А. (фамилия, инициалы)

Пенза 2024

УТВЕРЖДАЮ
зав. кафедры САПР
_____ И.П. Бурукина
«11» 06 2024 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

1. Студент гр. 20ВА1 факультета ВТ направление 02.03.03 «Математическое обеспечение и администрирование информационных систем»

Лысак Стефан Владимирович
(фамилия, имя, отчество)

2. Руководитель ВКР к.т.н., доцент кафедры САПР
(Должность и место работы)

Эпп Виталина Викторовна
(фамилия, имя, отчество)

3. Время выполнения бакалаврской работы с г._____ по г._____
4. Тема бакалаврской работы «Разработка web-сервиса для записи клиентов в автомастерскую»

Тема утверждена приказом ПГУ №_____ от _____

5. Задание на бакалаврскую работу: предмет и назначение разработки, функциональные, программные, технические требования и т.п.

5.1. Предмет: запись клиентов в автомастерскую.
5.2. Назначение разработки: Разрабатываемое web-сервисное приложение предназначено для автоматизации работы автомастерской.
5.3. Функциональные требования: 1) Создание базы данных для хранения информации 2) Разработка интерфейса для связи между клиентом и администратором системы 3) Разработка функционала со стороны сервера для ведения отчетности автомастерской 4) Разработка функционала со стороны сервера для редактирования системной информации 5) Разработка функционала со стороны сервера для составления статистики автомастерской 6) Разработка функционала со стороны клиента для заполнения и отправки данных формы 7) Разработка функционала со стороны клиента для авторизации клиентов 8) Разработка функционала со стороны клиента для хранения данных об автомобилях клиента.
5.4. Программные требования: среда разработки IntelliJ IDEA, PostgreSQL, Draw.io.
5.5. Технические требования: Персональный компьютер должен быть с операционной системой Microsoft Windows 7 и выше, с частотой процессора 3,2 ГГц. ;

6. Содержание пояснительной записки бакалаврской работы: перечень подлежащих разработке, расчетов, обоснований, описаний

6.1. Анализ предметной области
6.2. Проектирование приложения
6.3. Реализация приложения
6.4. Тестирование приложения

7. Календарный график выполнения бакалаврской работы

Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1. Анализ ТЗ	5 %	27.03 – 31.03	
2. Пред проектные исследования	5%	01.04 – 10.04	
3. Эскизное и техническое проектирование	10%	11.04 – 24.04	
4. Разработка ПО	35%	25.04 – 12.05	
5. Тестирование и опытная эксплуатация программы	35%	13.05 – 15.05	
6. Оформление ПЗ	10%	16.05 – 21.05	
7. Нормоконтроль		31.05	

Дата выдачи задания «25» 03 2024

Руководитель бакалаврской работы _____
(фамилия, имя, отчество)

Задание к исполнению принял «27» 03 2024

Студент Лисак С.В
(фамилия, имя, отчество)

Бакалаврскую работу к защите допустить

Декан ФВТ _____ 14.06.2024 / Л.Р. Фионова
(Подпись, дата)

Реферат

Пояснительная записка содержит 75 листов, 55 рисунков, 20 таблиц, 8 использованных источников и 1 приложение.

БАЗА ДАННЫХ, ИНФОРМАЦИОННАЯ СИСТЕМА, WEB – СЕРВИС, СУБД, АВТОМАСТЕРСКАЯ, POSTGRESQL.

Объект исследования – онлайн – сервисы записи клиентов в автомастерскую.

Цель выпускной квалификационной работы – автоматизировать работу автомастерской.

С помощью web-сервиса необходимо усовершенствовать запись клиентов в автомастерскую.

Технологии разработки – разработка сервера приложений проводилась на языке программирования Java с использованием фреймворка Spring, СУБД PostgreSQL 12. В качестве среды разработки использовался редактор программного кода IntelliJI IDEA и менеджер баз данных DataGrip.

Среда исполнения – Windows.

Результаты работы – в ходе выполнения выпускной квалификационной работы был создан web - сервис для записи клиентов в автомастерскую.

Содержание

Введение 6
1.Анализ предметной области 7
1.1 Описание автомастерской 7
1.2. Описание бизнес-процессов 8
1.3. Сравнительный анализ аналогичных разработок 9
1.4 Требования к Web - сервису 15
2. Проектирование приложения 19
2.1. Диаграмма прецедентов 19
2.3 Моделирование базы данных 25
2.4. Описание среды разработки 33
3.Реализация приложения 34
3.1.1 Развертывание базы данных с помощью ORM 34
3.2 Разработка серверной части 40
3.2.1. Наполнение базы данных тестовыми данными 40
3.2.2. Разработка запросов к базе данных 42
3.2.3. Описание JSON- объектов 46
3.3. Описание клиентской части 51
4.Тестироваие приложения 56
4.1. Выбор режима тестирования приложения 56
4.2. Тестирование Web – сервиса 57
Заключение 61
Список использованных источников 62
Приложение А ЛИСТИНГ ПРОГРАММЫ 63

Введение

У автомехаников в процессе основной работы возникает потребность самим отвечать на звонки, вести учет клиентов и записывать потенциальных заказчиков в очередь. Чтобы автоматизировать данный процесс, им потребуется специальный инструмент, который позволит это организовать.

Одним из таких решений стал онлайн-сервис по обслуживанию автомобилей – удобный метод записаться на услуги по ремонту и техническому осмотру автомобиля без необходимости посещения физического сервисного центра, а также исключить не нужные звонки операторам. Клиенты могут общаться с автомеханиками, получать консультации и следить за процессом работы через онлайн-платформу [1].

К сожалению, существующие сервисы обладают рядом недостатков, что не позволяет им в полной мере удовлетворять потребности автовладельца.

В качестве решаемых задач системы стоит выделить реализацию удобного функционала, как быстрой записи в автомастерскую со стороны автовладельца, так и комфортный учет ресурсов с места администратора-автомеханика.

Для достижения поставленных целей необходимо выполнить следующие задачи:

- проанализировать предметную область;
- спроектировать приложение;
- реализовать приложение;
- протестировать приложение.

Использовано 8 источников и 1 приложение.

1.Анализ предметной области

1.1 Описание автомастерской

Онлайн-сервис обработки заказов в автомастерской предлагает возможность получить качественную услугу без необходимости приезжать на место или названивать операторам с целью спросить свободно ли в конкретное время. В целом, несмотря на популярность онлайн-автомастерских, у них все еще есть ряд проблем, которые можно автоматизировать.

На рисунке 1 представлена организационная схема онлайн-автомастерской.

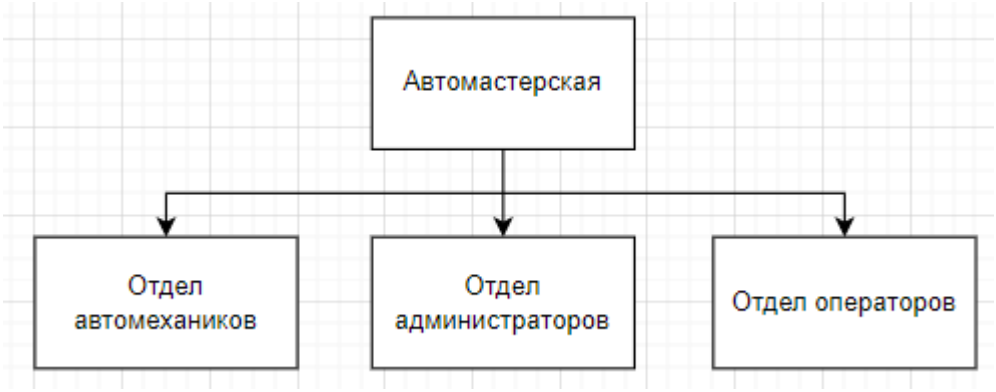


Рисунок 1 – Организационная схема онлайн-автомастерской

В данной схеме видны отделы, каждый из которых отвечает за свою деятельность:

– отдел автомехаников отвечает за оказание качественной услуги по ремонту автомобиля;

– отдел телефонных операторов отвечает за прием заявок как по телефонным звонкам, так и онлайн;

– отдел администрации является ключевым в данной схеме. В его обязанности входит: корректировка статусов работ, введение статистики автомастерской, а также редактирование списка услуг.

Автоматизация отдела телефонных операторов и администраторов позволяет значительно увеличить эффективность работы, сократить время обработки звонков без потери качества обслуживания клиентов.

Отсутствие современных технологий и автоматизированных систем может привести к повышенной нагрузке на операторов и замедлению обработки звонков. Кроме того, автоматизация отдела телефонных операторов и администраторов позволяет сократить расходы на персонал, так как часть операций может выполняться автоматически без участия человека. Это позволяет снизить затраты на оплату труда и снизить вероятность ошибок. С помощью данной разработки можно переложить обязанности администратора автомеханикам и также полностью исключить отдел телефонных операторов, заменив их онлайн-бронированием через web – сервис, где будет видно актуальное расписание автосервиса и список доступных услуг.

Таким образом, автоматизация отдела телефонных операторов является необходимым шагом. Правильно внедренные технологии автоматизации помогут сократить время обработки звонков, улучшить процессы работы отдела и сэкономить финансовые ресурсы компании.

1.2. Описание бизнес-процессов

На основе организационной схемы онлайн-автомастерской были выделены бизнес-процессы для онлайн-сервиса обработки заказов автомастерской (Рисунок 2).

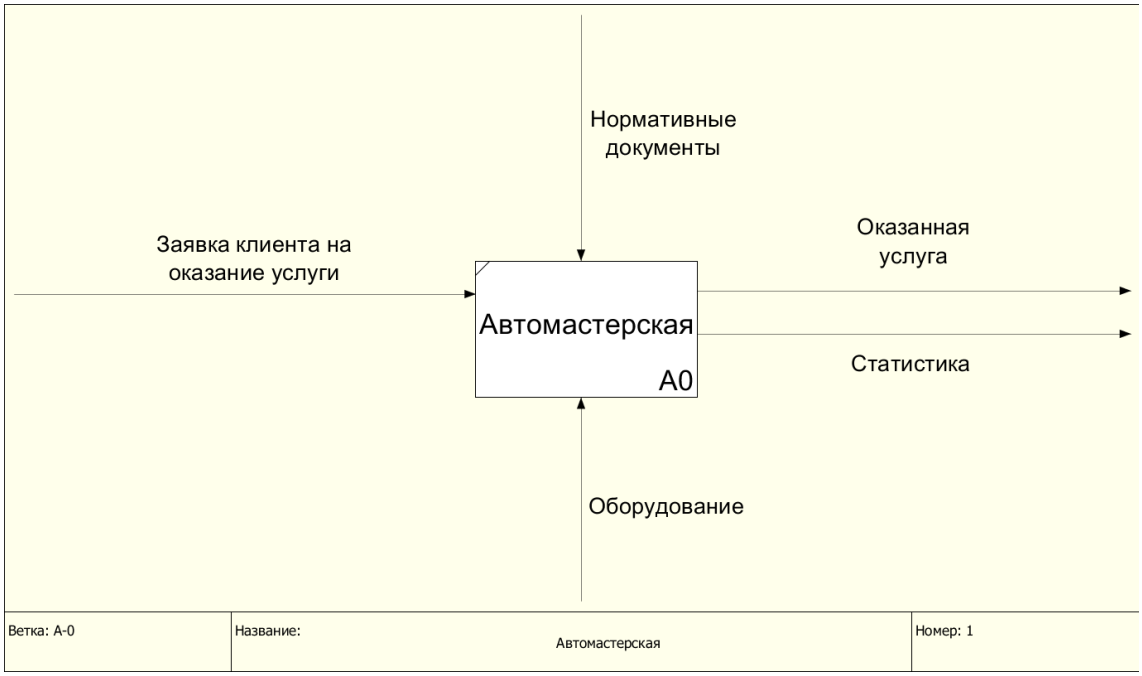


Рисунок 2 – Схема бизнес-процессов 1-го уровня

Далее представлена схема бизнес процессов 2-го уровня (Рисунок 3). На ней обозначены все три отдела автомастерской до автоматизации.

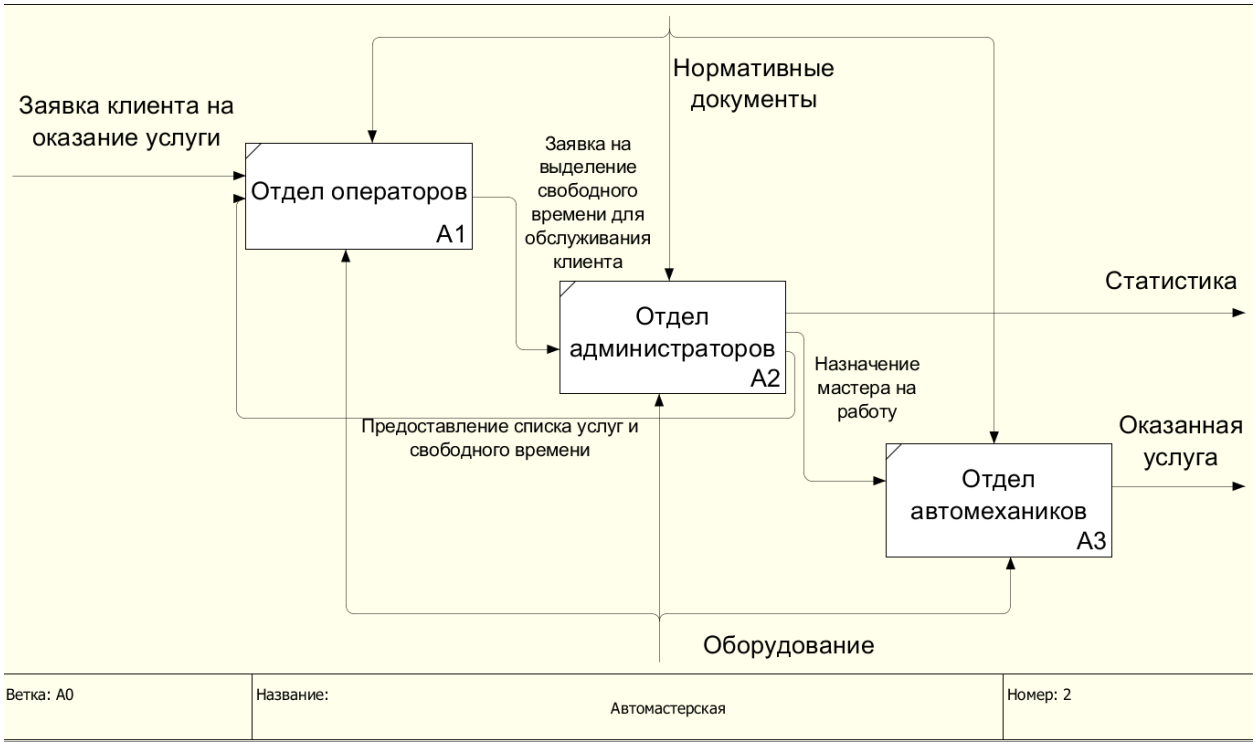


Рисунок 3 – Схема бизнес процессов 2-го уровня

Прежде чем выполнить любой бизнес-процесс в рамках системы, пользователь должен пройти авторизацию.

1.3. Сравнительный анализ аналогичных разработок

Проанализировав рынок программных продуктов для оформления и обработки заказов клиентов автомастерской сделан вывод, что прямые аналоги у разрабатываемого продукта есть.

Были отобраны системы, позволяющие проводить общее тестирование:

1. Autoservice58 – онлайн – запись на СТО [2];

На рисунке 4 приведена формы регистрации.

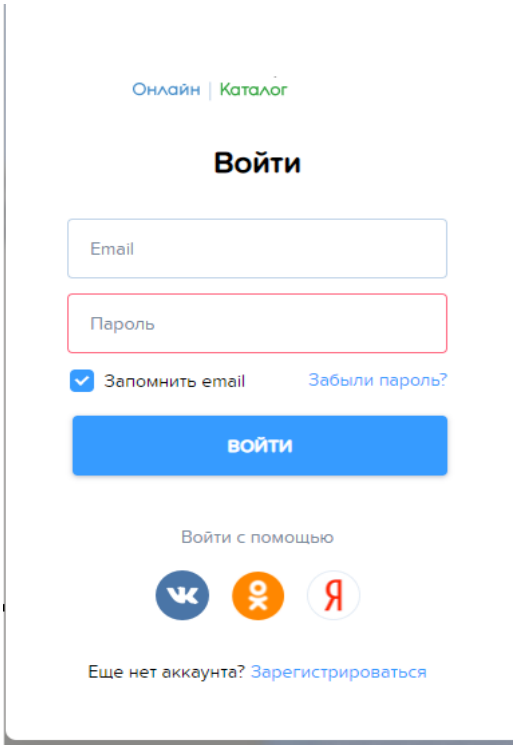


Рисунок 4 – Форма регистрации

На рисунке 5 приведено оформления заявки на данном сайте

Рисунок 5 – Форма оформления заявки

2. Euroauto– программа для онлайн - записи в автомастерскую [3];

На рисунке 6 приведена форма регистрации.

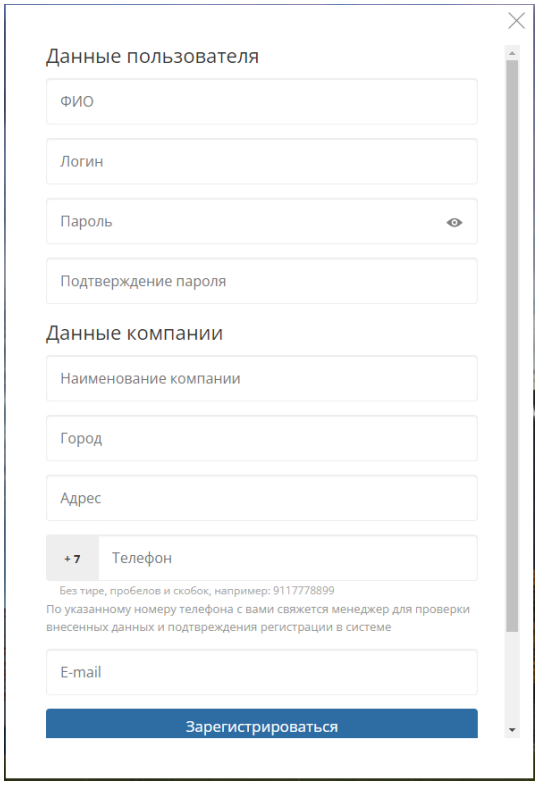


Рисунок 6 – Форма регистрации

На рисунке 7 представлено оформление заявки на данном сайте

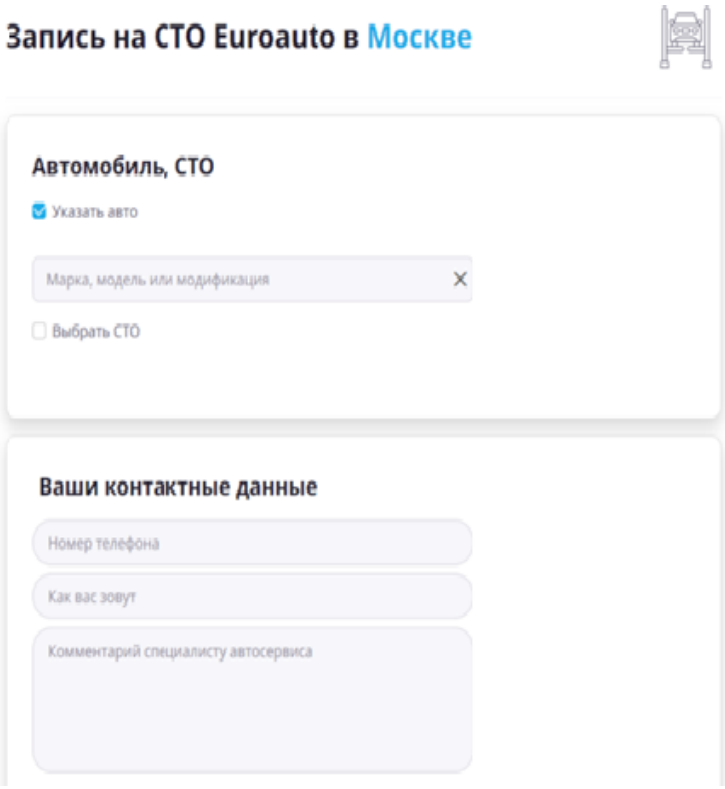


Рисунок 7 – Форма оформления заявки

Для выполнения сравнительного исследования, в роли критериев, выбраны особенно необходимые аспекты, которые открыто воздействуют на функционал сервиса, а именно:

а) функциональные возможности:

- возможность просмотра актуального календаря;
- возможность записи через оператора;
- расчет стоимости работ;
- администрирование пользовательской базы.

б) нефункциональные возможности:

- бесплатность продукта.

в) надёжность и безопасность:

- хранение данных в базе данных (БД).

Проанализировав бизнес-процессы, которые образовываются в момент работы систем онлайн-записи в автосервис, были определены показатели важности критериев, представленные в таблице 1.

Таблица 1 – Критерии оценки систем онлайн- автосервисов

№	Наименование	Показатель важности
	Функциональные возможности	
1	Возможность просмотра актуального календаря	1
2	Возможность записи через оператора	0,75
3	Расчет стоимости работ	0,5
4	Администрирование пользовательской базы	0,5
	Нефункциональные возможности	
5	Бесплатность продукта	0,75
	Надежность и безопасность	
6	Хранение данных в БД	0,5

Для анализа упомянутых выше критериев используется пятибалльная шкала соответствия, от 0 – «полное несоответствие», до 4 – «полное соответствие» требованиям. В таблице 2 приведены характеристики оценки.

Таблица 2 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) отсутствует, реализовать ее невозможно
1	функция (качество) отсутствует, но ее возможно реализовать путем доработки системы
2	схожие, но отличные от требуемых, функции или качества системы находятся на этапе разработки
3	реализованы схожие, но несколько отличные от требуемых, функции или качества системы
4	функция или качество полностью реализовано

Таблица 3 заполнена на основе анализа упомянутых выше систем. Расчет аддитивной суммы интегральной оценки для каждой сопоставляемой системы проведен по следующей формуле:

$$O_{\text{пп}} = \sum_{i=1}^{n} (Z_i \cdot K_i),$$

где $O_{\text{пп}}$ – интегральная оценка программного продукта;

n – количество критериев сравнения заявленных продуктов согласно таблице критериев;

Z_i – значение выполнения критерия (4 – выполнен, 0 – не выполнен);

K_i – показатель важности критерия сравнения (от 0 до 1).

Таблица 3 – Критериальная оценка программных продуктов

№	Критерий	Программный продукт №		Показатель важности критерия сравнения
		Autoservice58	Euroauto	
1	Возможность просмотра актуального календаря	2	4	1
2	Возможность записи через оператора	4	4	0,75
3	Расчет стоимости работ	2	2	0,5
4	Администрирование пользовательской базы	4	0	0,75
5	Бесплатность продукта	0	0	0,5
6	Хранение данных в СУБД	2	3	0,5
	Оценка программного продукта	10	9,5	

Подводя итог, все сравненные сервисы не способны в полной мере выполнить поставленные задачи. Такие как:

– администрирования пользовательской базы;

– правильный расчет стоимости работ (производится после обслуживания автомобиля. Что может привести к конфликтным ситуациям);

– возможность просмотра актуального календаря;

– продукт не является бесплатным.

Таким образом, существующие системы онлайн-записи на прием в автомастерские нуждаются в усовершенствовании. Интеграция передовых технологий может ускорить процесс записи.

Проанализировав основной цикл взаимодействия с Web-сервисом, можно перейти к составлению требований к онлайн сервису записи клиентов в автомастерскую.

1.4 Требования к Web - сервису

Разрабатываемое Web – приложение обязательно должно обеспечивать пользователей следующими функциями:

– формирование (создание/редактирование) заявки в автосервис;

– выбор удобного времени (если удобного времени для автовладельца нет, то скорректировать с автомехаником);

– выбор нужной для клиента услуги.

Требования к разделению доступа.

Для любого действия в системе необходима авторизация пользователя.

Если пользователь авторизован под ролью администратор-автомеханик, то ему доступна административная часть Web-сервера, которая подразумевает под собой: подтверждение/отмена входящей от клиента заявки, обновление статусов работ, а также создание/обновление цен на услуги. Если же пользователь авторизован под ролью Клиента, то ему доступно формирование заявки и назначение времени на выполнение услуги.

В иных случаях, клиентское приложение попытается использовать функции, которые не предназначены для конкретной роли, будет выдана ошибка 403 (доступ запрещен).

Если клиентское приложение попробует получить доступ к REST-серверу без авторизации, будет выдана ошибка 401 (отказ в доступе).

На основе требований к Web-сервису была составлена таблица 4. В ней описаны процессы, которые есть в системе, а также входные и выходные данные.

Таблица 4 – Описание входных и выходных данных

№	Процесс	Входные данные		Выходные данные	
		Поставщик	Содержание	Потребитель	Содержание
1	Просмотреть статусы всех работ	Администратор-автомеханик	Просмотр работ(идентификатор работы, идентификатор статуса работы)	Администратор-механик	Список работ с обновленными статусами и датами
2	Просмотреть список услуг	Администратор-автомеханик	Сведения о предоставляемых услугах автомастерской (идентификатор услуги, цена, длительность). Добавление новой услуги	Клиент	Список всех возможных услуг
3	Оформление заявки на ремонт	Клиент	Заполнение заявки на ремонт (время, идентификатор работы, идентификатор машины)	Клиент	Запись клиента в сервис, назначенный за этой записью мастер
4	Просмотреть работы мастеров	Администратор-автомеханик	Список работ мастеров(идентификатор мастера, идентификатор услуги и количество закрепленных за мастером услуг)	Администратор-автомеханик	Список работ мастеров(можно просмотреть работы определенного мастера)

Продолжение таблицы 4

№	Процесс	Входные данные		Выходные данные	
		Поставщик	Содержание	Потребитель	Содержание
5	Просмотреть клиентов автомастерской	Администратор-автомеханик	Идентификатор работы, идентификатор клиента, идентификатор автомобиля)	Администратор-автомеханик	Список клиентов автомастерской.
6	Просмотреть список автомобилей	Клиент	Просмотр закрепленных в автомастерской автомобилей авторизованного клиента(идентификатор автомобиля)	Клиент	Список автомобилей
7	Просмотреть список прошлых работ	Клиент	Идентификатор работы, идентификатор статуса работы	Клиент	Список прошлых и текущих работ.

По таблице 4 можно сделать вывод, что в системе существует несколько различных процессов, каждый из которых имеет определенные входные данные и выходные данные. Из анализа таблицы видно, что каждый процесс преобразует определенные входные данные в определенные выходные данные, что является основным функционалом системы.

После доработки бизнес – процессов был полностью автоматизирован отдел операторов, а автомеханики взяли на себя функции отдела администратора (Рисунок 8).

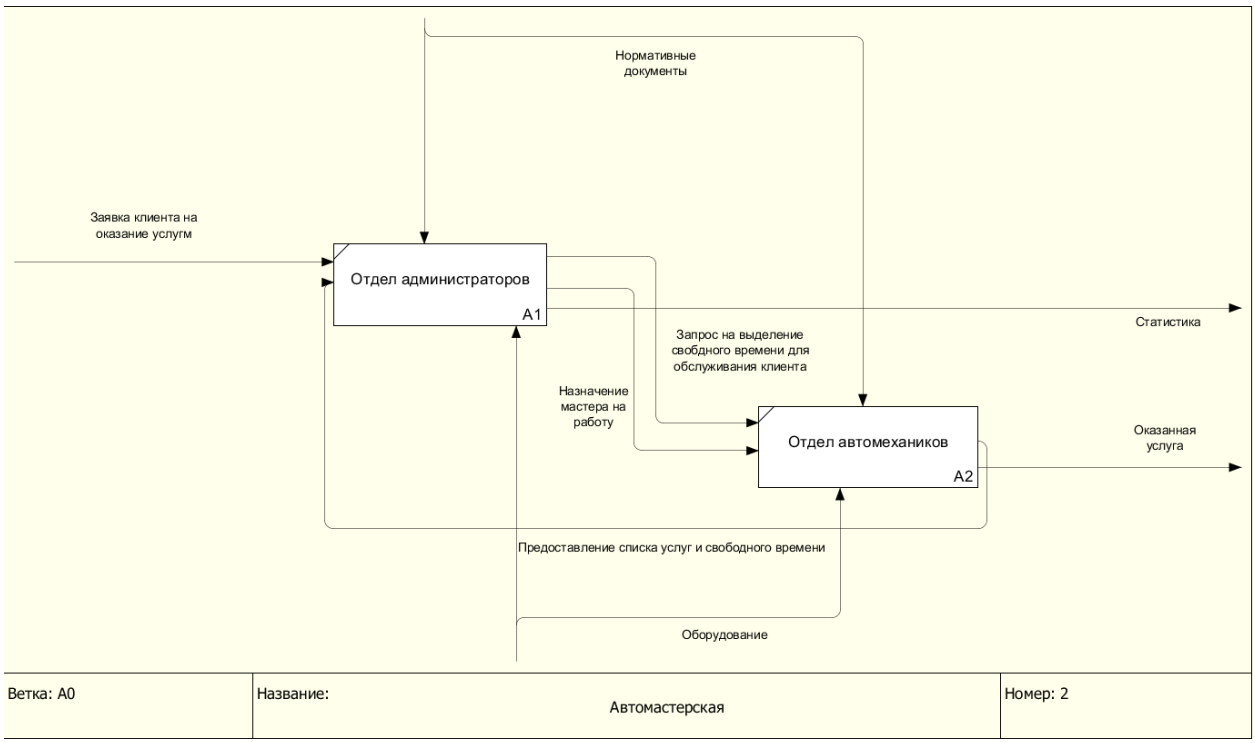


Рисунок 8 – Организация бизнес – процессов после доработки

Требования к программному обеспечению серверной части.

Для успешной работы REST-сервера требуется:

- операционная система Windows Server 2003 или Linux;
- СУБД PostgreSQL версии не ниже 13;
- среда запуска JRE версии не ниже 8 версии 256 обновления.

Требования к программному обеспечению серверной части.

Для нормального REST-серверного приложения требуются техническое обеспечение со следующими минимальными характеристиками:

- процессор Intel Core i3 – 3240;
- оперативная память 4096 Mb RAM;
- жесткий диск (HDD) – 10Gb свободного места.

Также для эффективной работы серверной части программного обеспечения рекомендуется использовать выделенный сервер с поддержкой высокоскоростного интернет-соединения.

2. Проектирование приложения

Построение Web – приложения зависит от используемой технологии разработки. Это связано с использованием определенных инструментов, языков программирования, фреймворков и других компонентов, которые влияют на процесс разработки и функциональность приложения. В данной работе использован язык программирования Java совместно с интегрированной средой разработки IntelliJ IDEA

Она содержит множество удобных функций и инновационных решений для повышения производительности и качества кода, что делает ее идеальным выбором для разработчиков всех уровней.

2.1. Диаграмма прецедентов

Web-сервис предназначен для записи автовладельцев на ремонт, диагностику и разного рода малярных работ посредством бронирования через онлайн-сервис. Также система направлена на упрощение работы автомехаников и операторов. С точки зрения операторов разрабатываемое приложение позволит избавить их от входящих телефонных звонков и в дальнейшем полностью автоматизировать их работу, путем прямого «общения» автомехаников и автовладельцев.

Информационная система должна обеспечить пользователя следующими возможностями:

- просмотр актуального календаря автомастерской;
- выбор удобного времени для записи в автомастерскую;
- формирование (создание/редактирование) заявки;
- обратная связь с автомехаником.

На основе сформулированных задач создана диаграмма прецедентов [4] (Рисунок 9).

Рисунок 9 – Диаграмма вариантов использования

Составлена спецификация для основных прецедентов из диаграммы вариантов использования (Таблица 5).

Таблица 5 – Спецификация основных прецедентов

Прецедент	Краткое описание	Главные актеры:	Предусловие:	Основной поток:
Просмотреть список услуг	Администратор-автомеханик создает список возможных услуг, которые будут доступны в автосервисе	Администратор-автомеханик	Выполнить вход в систему от имени «Администратор-автомеханик»	1.Администратор-автомеханик просматривает каталог услуг в автомастерской 2.Администратор-автомеханик при надобности добавляет новую услугу 3.Система сохраняет изменения в базе данных

Продолжение таблицы 5

Прецедент	Краткое описание	Главные актеры:	Предусловие:	Основной поток:
Просмотреть список автомобилей	Клиент просматривает список своих автомобилей	Клиент	Выполнить вход в систему от имени «Клиент»	1.Клиент просматривает список автомобилей 2.Клиент добавляет новый автомобиль в систему (при необходимости). 3.Система сохраняет результаты в базу данных
Просмотреть статусы всех работ	Просмотр статусов работ в автомастерской	Администратор-автомеханик	Выполнить вход в систему от имени «Администратор-автомеханик»	1.Администратор-автомеханик заходит на вкладку «Статусы работ» 1.1.Администратор-автомеханик просматривает текущие работы 2.Администратор-автомеханик меняет статус работы на актуальный 3.Система сохраняет изменения в базе данных
Оформление заявки на ремонт	Клиенту предоставляется форма для оформления заявки	Клиент	Выполнить вход в систему от имени «Клиент»	1.Клиенту предоставляется календарь свободного времени и перечень предоставляемых услуг 2.Клиент заполняет форму заявки 2.1.Клиент выбирает подходящий для него день 2.2Клиент выбирает подходящее для него время 2.3.Клиент выбирает подходящую для него услугу. 3.Если вся форма заполнена, данные отправляются на подтверждение/отмену администратору. 4.Система сохраняет результаты выполнения теста в базу данных

2.2 Диаграмма деятельности

На основе диаграммы вариантов использования, была сформирована диаграмма деятельности, которая помогает лучше понять все возможные действия в системе как для клиента, так и для администратора-автомеханика (Рисунок 10 – 12).

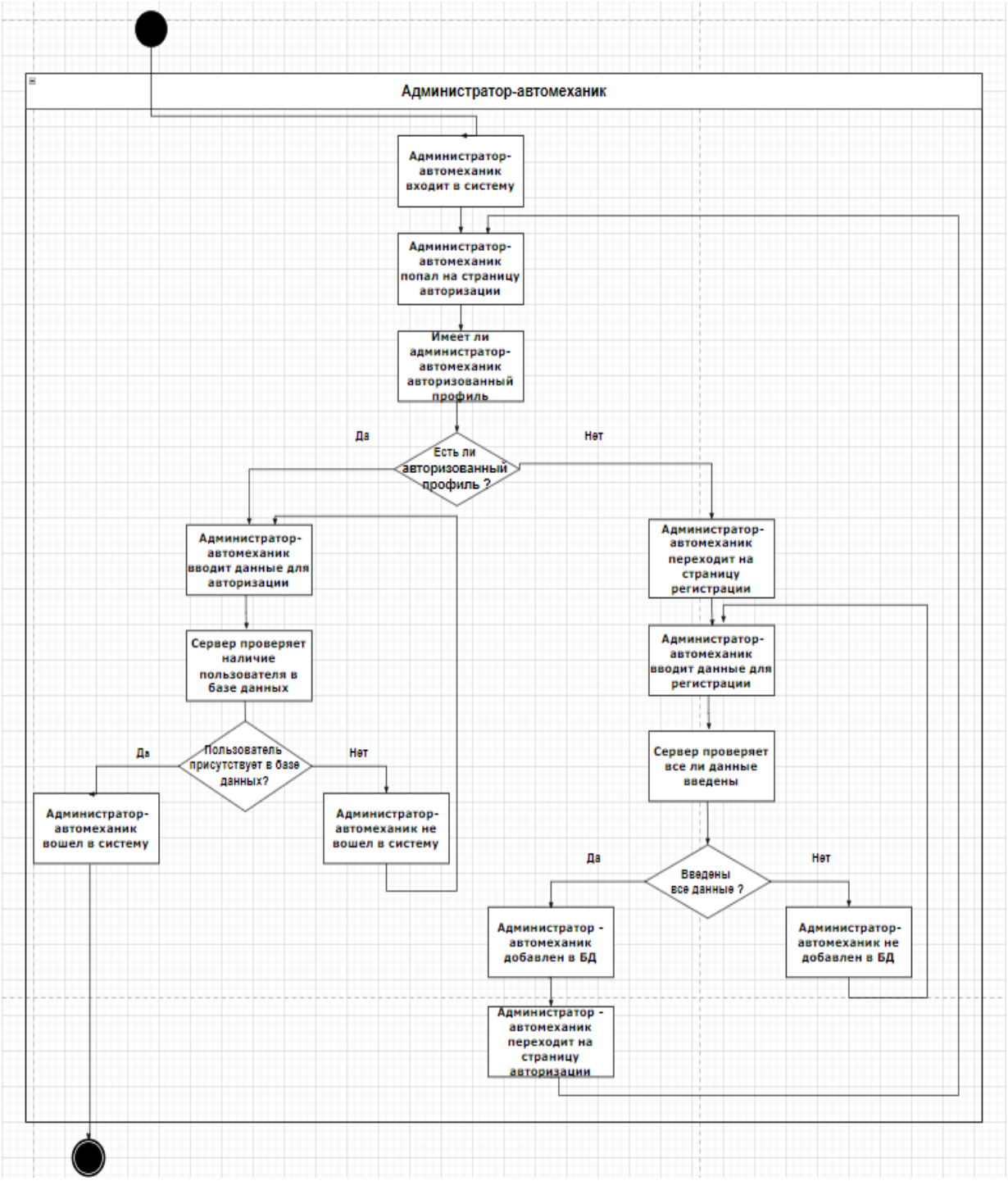


Рисунок 10 – Диаграмма деятельности (вход в систему со стороны администратора - автомеханика)

На рисунке 10 показан вход в систему со стороны администратора – автомеханика

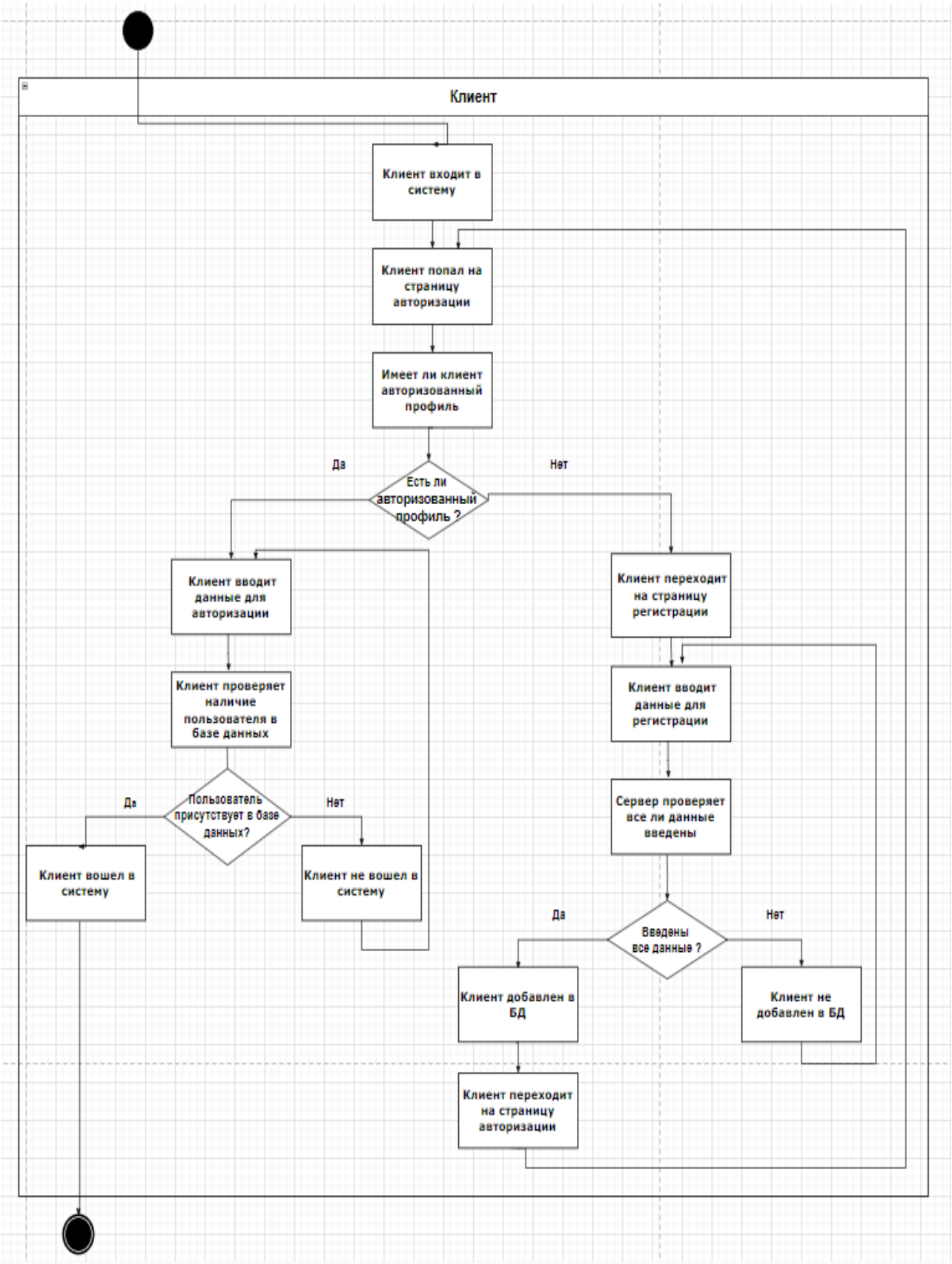


Рисунок 11 – Диаграмма деятельности (вход в систему со стороны клиента)

На рисунке 11 показан вход в систему со стороны клиента

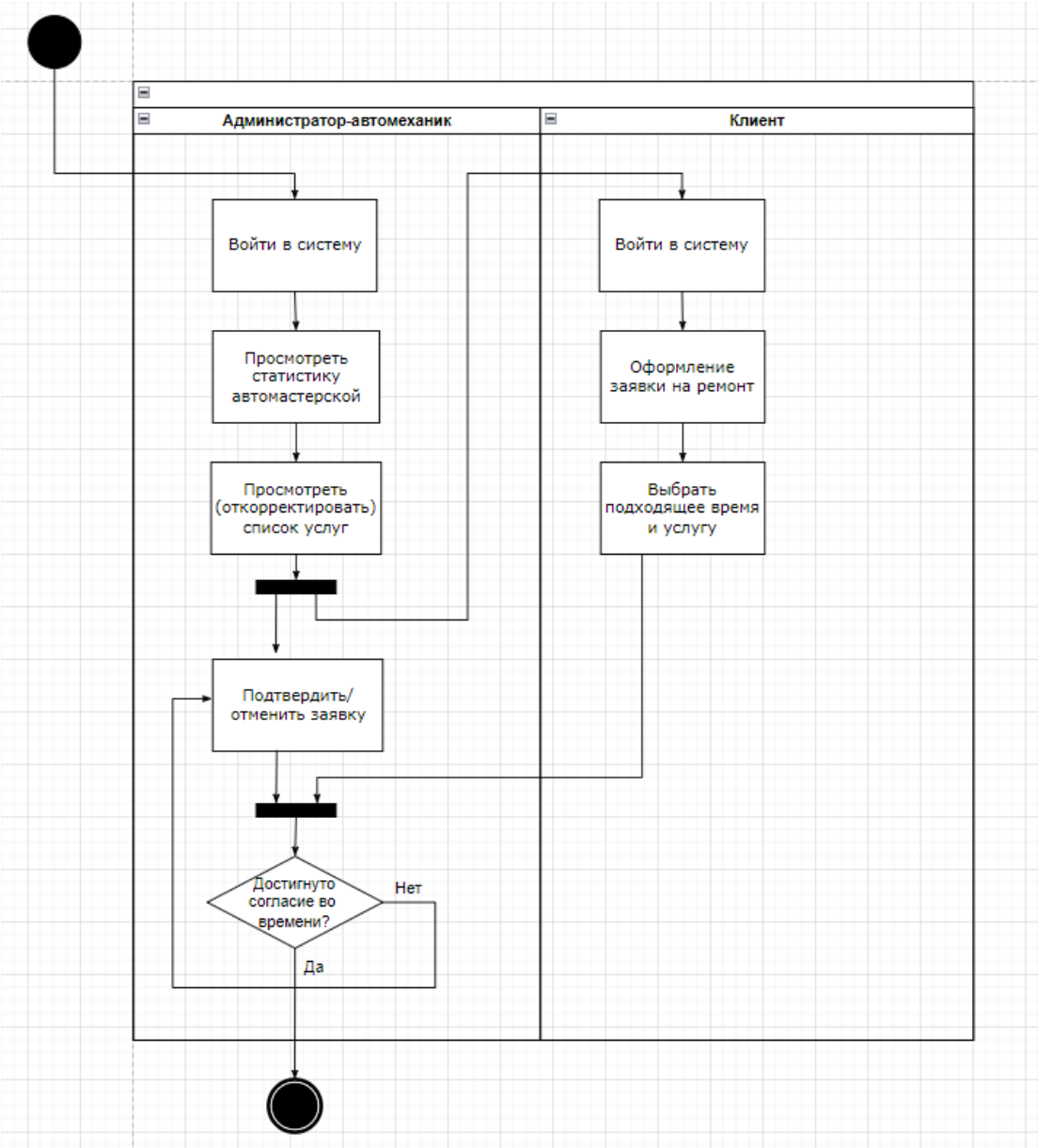


Рисунок 12 – Диаграмма деятельности основных процессов в системе

На рисунке 12 отображена последовательность выполнения основных процессов в системе.

Проанализировав основной цикл взаимодействия с REST-сервером, можно перейти к моделированию базы данных онлайн сервиса обработки заказов автомастерской.

2.3 Моделирование базы данных

Концептуальная модель представляет собой описание ключевых сущностей (таблиц) и их взаимосвязей, независимо от используемой БД и синтаксиса целевой СУБД. Обычно в такой модели указываются только имена сущностей (таблиц), не рассматривая их атрибутов.

На рисунке 13 представлена диаграмма концептуальной модели в формате «сущность-связь» для отражения структуры информационной системы. Эта модель была создана в программном обеспечении draw.io с использованием стандарта IDEF1X [5].

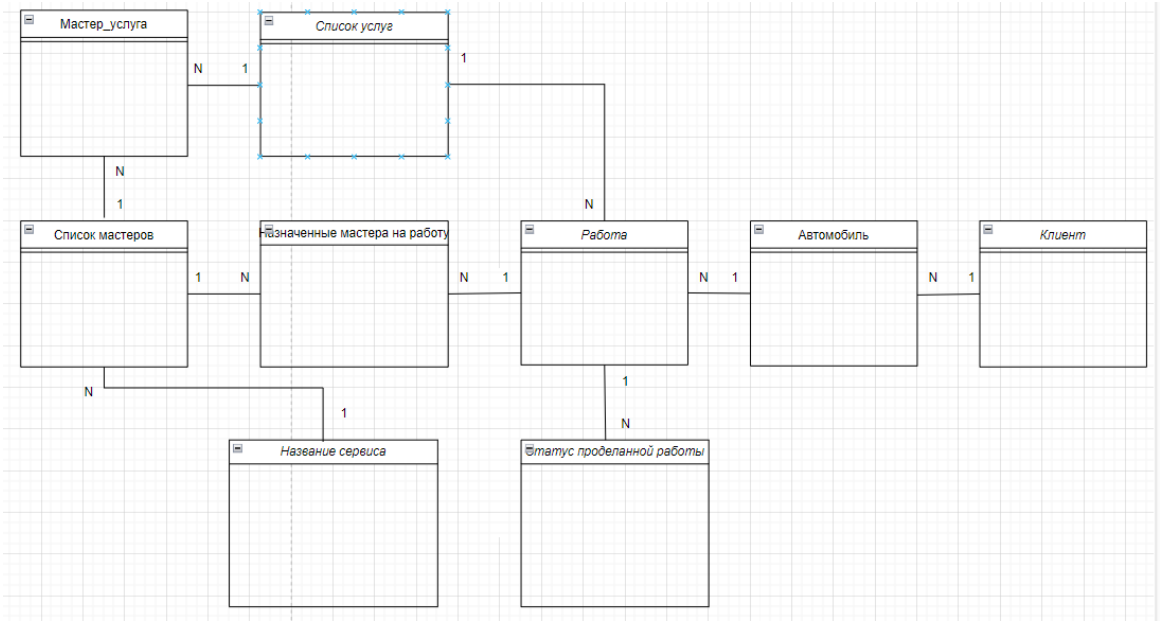


Рисунок 13 – Графическое представление концептуальной модели

В таблице 6 представлены сущности, сформированные на основании концептуальной модели

Таблица 6 – Описание сущностей

№	Наименование	Описание
1	Автомобиль	Перечень автомобилей
2	Клиент	Перечень клиентов в конкретном сервисе

Продолжение таблицы 6

№	Наименование	Описание
3	Назначить мастера на работу	Перечень мастеров, назначенных на конкретную работу
4	Работа	Перечень работ
5	Список услуг	Перечень услуг, которые предоставляет данный сервис
6	Название сервиса	Перечень автосервисов
7	Статус проделанной работы	Статусы заказа, которые выполняются/готовы/отменены
8	Мастер_услуга	Перечень мастеров закрепленных за конкретной услугой

Для сущностей предметной области были сформированы связи (таблица 7).

Таблица 7 – Описание типов связей между сущностями

№	Тип сущности 1	Тип сущности 2	Описание	Мощность
1	Клиент	Автомобиль	У клиента может быть много автомобилей/У автомобиля есть один клиент	1:M
2	Автомобиль	Работа	У автомобиля может быть много работ/ У работы есть один автомобиль	1:M
3	Работа	Список услуг	У одной выполняемой работы одна услуга из списка	M:1
4	Работа	Статус проделанной работы	Множество работ закрепляются за одним статусом	1:M
5	Работа	Назначить мастера на работу	У одной работы может быть несколько назначенных мастеров/ У несколько назначенных мастеров может быть одна работа	1:M
6	Назначить мастера на работу	Список мастеров	Из списка мастеров могут быть назначены несколько мастеров на работу	M:1
7	Список мастеров	Название сервиса	У сервиса есть много мастеров/ У мастера есть один сервис	M:1

Продолжение таблицы 7

№	Тип сущности 1	Тип сущности 2	Описание	Мощность
8	Мастер_услуга	Список услуг	Множество мастеров могут выполнять одну услугу	M:1

Логическая модель – это модель базы данных, излагаемая в понимании модели данных. Логическая модель расширяет концептуальную путем определения для сущностей их атрибутов, описаний и ограничений, уточняет состав сущностей и взаимосвязи между ними [6].

На основании концептуальной модели была собрана логическая модель базы данных (Рисунок 14).

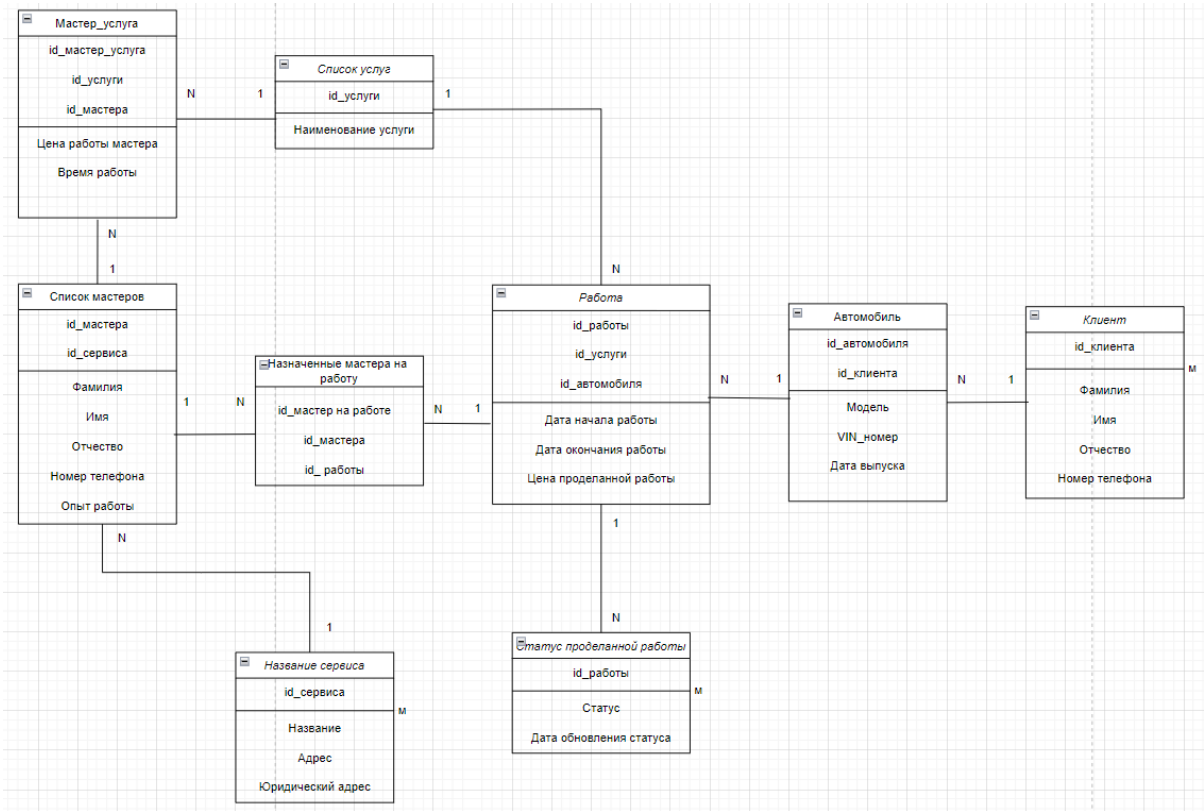


Рисунок 14 – Логическая модель базы данных

Описание атрибутов для сущности «Клиент» представлено в таблице 8.

Таблица 8 – Описание атрибутов сущности «Клиент»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор клиента	Уникальный номер клиента	Целый	Нет	PK	
Фамилия	Фамилия клиента	Символьный	Нет		
Имя	Имя клиента	Символьный	Нет		
Отчество	Отчество клиента	Символьный	Да		
Номер телефона	Номер телефона клиента	Символьный	Нет		Используется для входа в систему

Описание атрибутов для сущности «Автомобиль» представлено в таблице 9.

Таблица 9 – Описание атрибутов сущности «Автомобиль»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор автомобиля	Уникальный номер автомобиля (С точки зрения хранения данных в БД)	Целый	Нет	PK	
Идентификатор клиента	Уникальный номер клиента	Целый	Нет	FK	
Модель	Модель обслуживаемой машины	Символьный	Нет		
VIN_номер	Идентификационный номер автомобиля (С точки зрения самого автомобиля)	Символьный	Нет		Нужен для того, чтобы отличать один автомобиль от другого, а также для заказов деталей для автомобиля
Дата выпуска	Дата выпуска автомобиля				

Описание атрибутов для сущности «Название сервиса» представлено в таблице 10.

Таблица 10 – Описание атрибутов сущности «Название сервиса»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор сервиса	Уникальный номер сервиса	Целый	Нет	PK	
Название	Название сервиса	Символьный	Нет		
Адрес	Адрес сервиса	Символьный	Нет		
Юридический адрес	Юридический адрес				

Описание атрибутов для сущности «Список мастеров» представлено в таблице 11.

Таблица 11 – Описание атрибутов сущности «Список мастеров»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор мастера	Уникальный номер мастера	Целый	Нет	PK	
Идентификатор сервиса	Уникальный номер сервиса	Целый	Нет	FK	
Фамилия	Фамилия мастера	Символьный	Нет		
Имя	Имя мастера	Символьный	Нет		
Отчество	Отчество мастера	Символьный	Да		
Номер телефона	Номер телефона мастера	Символьный	Нет		Используется для входа в систему
Опыт работы	Трудовой стаж мастера	Символьный	Нет		

Описание атрибутов для сущности «Назначенные мастера на работу» представлено в таблице 12.

Таблица 12 – Описание атрибутов сущности «Назначенные мастера на работу»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор мастера на работе	Уникальный номер мастера на работе	Целый	Нет	PK	
Идентификатор мастера	Уникальный номер мастера	Целый	Нет	FK	
Идентификатор работы	Уникальный номер работы	Целый	Нет	FK	

Описание атрибутов для сущности «Работа» представлено в таблице 13.

Таблица 13 – Описание атрибутов сущности «Работа»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор работы	Уникальный номер работы	Целый	Нет	PK	
Идентификатор услуги	Уникальный номер услуги	Целый	Нет	FK	
Идентификатор автомобиля	Уникальный автомобиля	Целый	Нет	FK	
Дата начала работы	Фиксированное время, когда начинается конкретная работа по обслуживанию автомобиля	Timestamp	Нет		
Дата окончания работы	Фиксированное время, когда заканчивается конкретная работа по обслуживанию автомобиля	Timestamp	Нет		
Цена проделанной работы	Цена проделанной работы	Double			

Описание атрибутов для сущности «Статус проделанной работы» представлено в таблице 14.

Таблица 14 – Описание атрибутов сущности «Статус проделанной работы»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор работы	Уникальный номер работы	Целый	Нет	PK	
Статус	Статус заказа	Символьный	Нет		
Дата обновления заказа	Дата, когда сменился статус заказа	Timestamp	Нет		

Описание атрибутов для сущности «Список услуг» представлено в таблице 15.

Таблица 15 – Описание атрибутов сущности «Список услуг»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор услуги	Уникальный номер услуги	Целый	Нет	PK	
Наименование услуги	Перечень предоставляемых услуг	Символьный	Нет		

Описание атрибутов для сущности «Мастер_услуга» представлено в таблице 16.

Таблица 16 – Описание атрибутов сущности «Мастер_услуга»

Наименование	Описание	Тип данных	Возможность значения NULL	Роль	Примечание
Идентификатор мастер_услуга	Уникальный номер мастера, которого назначили на выполнение услуги (или услуг)	Целый	Нет	PK	
Идентификатор мастера	Уникальный номер мастера	Целый	Нет	FK	
Идентификатор услуги	Уникальный номер услуги	Целый	Нет	FK	
Цена услуги	Цена услуги	Double	Нет		
Время выполнение работы	Время, за которое выполнится работа	Целый	Нет		

Заполнение таблиц происходит на основе анализа концептуальной модели и ее элементов. Каждая сущность представлена в отдельной таблице, где каждый столбец соответствует атрибуту сущности, а каждая строка содержит информацию об отдельной записи. Связи между сущностями отражаются в таблицах с помощью внешних ключей. Это позволяет установить связь между записями в разных таблицах и обеспечить целостность данных. Заполнение таблиц на основе графического представления концептуальной модели является важным этапом процесса разработки базы данных. Оно позволяет создать основу для хранения и управления данными.

Логическая модель базы данных представляет собой абстрактное представление структуры данных, которая будет храниться в базе данных. Она описывает сущности, атрибуты и связи, которые определяют, как данные будут организованы и взаимодействовать друг с другом.

Логическая модель базы данных помогает разработчикам понять структуру данных и взаимосвязи между ними. Она служит основой для создания физической модели базы данных, которая определяет, как данные будут храниться на конкретной платформе или СУБД.

2.4. Описание среды разработки

PostgreSQL – это бесплатная СУБД с открытым исходным кодом. С содействием PostgreSQL можно формировать, хранить базы данных и работать с данными с помощью запросов на языке SQL.

JRE (Java Runtime Environment) – среда выполнения Java. Она действует как посредник между программой и операционной системой, позволяет выполнять программу на разных устройствах и ОС. Благодаря ей разработчик может создавать приложение, не задумываясь о том, где оно будет выполняться.

IntelliJ IDEA – это интегрированная среда разработки (IDE), то есть система программных средств для создания проектов на разных языках программирования. Она содержит множество удобных функций и инновационных решений для повышения производительности и качества кода. Также данная среда разработки обладает мощным интеллектуальным анализом кода, автодополнением, отладкой, рефакторингом и другими функциями, которые значительно упрощают и ускоряют процесс разработки.

React - это библиотека для создания пользовательских интерфейсов для веб- и мобильных приложений. Она позволяет разрабатывать интерактивные и масштабируемые пользовательские интерфейсы, разбивая их на небольшие компоненты, которые могут быть повторно использованы.

Spring Framework - это мощный инструмент для разработки приложений на языке Java. Он предоставляет обширную поддержку для создания современных приложений, а также инновационные решения для улучшения производительности и качества кода.

3.Реализация приложения

3.1.1 Развертывание базы данных с помощью ORM

ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Рисунок диаграммы сущность-связь, построенной на основе таблиц БД представлен на рисунке 15.

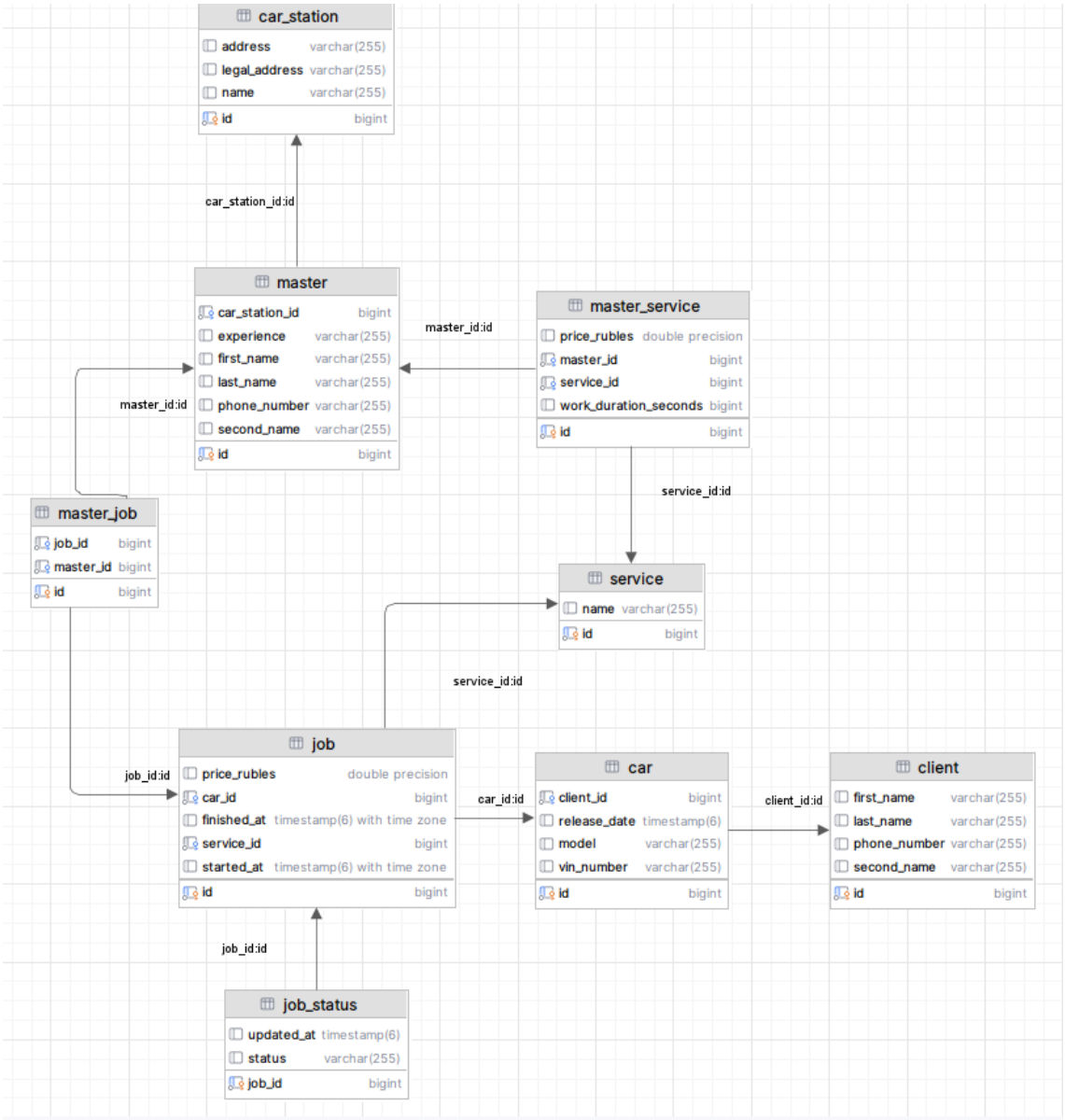


Рисунок 15 – Диаграмма сущность-связь

Исходный код Java классов, отвечающих за создание базы данных, представлен в приложении А.

Сущность базы данных описывается декларативно, при помощи аннотаций. Характерной аннотацией, которая объявляет класс сущностью базы данных, является аннотация @Entity. Аннотация @Table позволяет задать иное название таблицы или создать индексы и т.п.

Содержимое сущности ClientEntity рассмотрено на рисунке 16.

```
@Entity
@Table(name = "client")
public class ClientEntity {

  @Id
  @GeneratedValue
  private Long id;

  private String firstName;

  private String secondName;

  private String lastName;

  private String phoneNumber;

  @OneToMany(mappedBy = "client")
  private List<CarEntity> cars;

}
```

Рисунок 16 – Класс ClientEntity

Содержимое сущности CarEntity рассмотрено на рисунке 17.

```java
@Entity
@Table(name = "car")
public class CarEntity {

  @Id
  @GeneratedValue
  private Long id;

  private String model;

  private String vinNumber;

  private Date releaseDate;

  @JsonIgnore
  @ManyToOne(optional = false)
  private ClientEntity client;
}
```

Рисунок 17 – Класс CarEntity

Содержимое сущности JobEntity рассмотрено на рисунке 18.

```java
@Entity
@Table(name = "job")
public class JobEntity {

  @Id
  @GeneratedValue
  private Long id;

  private ZonedDateTime startedAt;
  private ZonedDateTime finishedAt;

  private Double priceRubles;

  @ManyToOne(optional = false)
  private ServiceEntity service;

  @ManyToOne(optional = false)
  private CarEntity car;
}
```

Рисунок 18 – Класс JobEntity

Содержимое сущности JobStatusEntity рассмотрено на рисунке 19.

```
@Entity
@Table(name = "job_status")
public class JobStatusEntity {

  @Id
  private Long jobId;

  @Enumerated(EnumType.STRING)
  private JobStatus status;

  private Date updatedAt;
}
```

Рисунок 19 – Класс JobStatusEntity

Содержимое сущности CarStationEntity рассмотрено на рисунке 20.

```
@Entity
@Table(name = "car_station")
public class CarStationEntity {

  @Id
  @GeneratedValue
  private Long id;

  private String name;

  private String address;

  private String legalAddress;

  @OneToMany(mappedBy = "carStation")
  private List<MasterEntity> masters;

}
```

Рисунок 20 – Класс CarStationEntity

Содержимое сущности MasterEntity рассмотрено на рисунке 21.

```java
@Entity
@Table(name = "master")
public class MasterEntity {

  @Id
  @GeneratedValue
  private Long id;

  private String firstName;

  private String secondName;

  private String lastName;

  private String phoneNumber;

  private String experience;

  @JsonIgnore
  @ManyToOne(optional = false)
  private CarStationEntity carStation;
}
```

Рисунок 21 – Класс MasterEntity

Содержимое сущности MasterServiceEntity рассмотрено на рисунке 22.

```java
@Entity
@Table(name = "master_service")
public class MasterServiceEntity {

    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne(optional = false)
    private MasterEntity master;

    @ManyToOne(optional = false)
    private ServiceEntity service;

    private Long workDurationSeconds;

    private Double priceRubles;
}
```

Рисунок 22 – Класс MasterServiceEntity

Содержимое сущности ServiceEntity рассмотрено на рисунке 23.

```
@Entity
@Table(name = "service")
public class ServiceEntity {

  @Id
  @GeneratedValue
  private Long id;

  private String name;
}
```

Рисунок 23 – Класс ServiceEntity

Содержимое сущности MaterJobEntity рассмотрено на рисунке 24.

```
@Entity
@Table(name = "master_job")
public class MasterJobEntity {

    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne(optional = false)
    private MasterEntity master;

    @ManyToOne(optional = false)
    private JobEntity job;
}
```

Рисунок 24 – Класс MasterJobEntity

Также у таблиц указаны отношения с другими сущностями. Они могут быть заданы с помощью аннотаций или XML-конфигурации. Например, сущности могут иметь отношения @OneToOne, @OneToMany, @ManyToOne, и @ManyToMany. Эти отношения определяются через аннотации, указывающие связи между различными сущностями.

3.2 Разработка серверной части

3.2.1. Наполнение базы данных тестовыми данными

За тестовое наполнение данными отвечает класс «DatasetCreator». В контексте предметной области этот класс отвечает за наполнение данными всех таблиц.

Содержимое таблицы «car» после наполнения тестовыми данными представлено на рисунке 25.

	client_id	id	release_date	model	vin_number
1	1	1	2015-07-10 00:00:00.000000	Lada Vesta	JH4DA3462JS027249
2	1	2	2011-06-14 00:00:00.000000	Toyota Rav 4	WH4233462JS027249
3	2	3	2009-04-23 00:00:00.000000	Honda Civic	JK42339762JS07679
4	3	4	2014-11-01 00:00:00.000000	Chevrolet Aveo	OP10339762JS07679
5	4	5	2016-12-09 00:00:00.000000	Lada Granta	HY471093JS0761279
6	5	6	2013-10-12 00:00:00.000000	Hyunday Tucson	JK42339762J041941
7	6	7	2015-10-01 00:00:00.000000	Nissan Note	NM42339701JS07679
8	6	8	2017-04-23 00:00:00.000000	Ford Fushion	VR13019762JS06779
9	6	9	2020-08-28 00:00:00.000000	BMW X4	JK4233947120US076
10	6	10	2019-11-06 00:00:00.000000	Toyota Tundra	PQ42340762JS09179
11	7	11	2021-02-16 00:00:00.000000	Kia Rio	XB30384701JS09145
12	8	12	2013-05-08 00:00:00.000000	Lada Kalina	EW42340762JS26810
13	9	13	2022-04-06 00:00:00.000000	Porsche Cayenne	PQ45240762JS09132
14	9	14	2023-11-26 00:00:00.000000	Chevrolet Camaro	NZ4230762JS017901
15	10	15	2019-10-06 00:00:00.000000	Skoda Octavia	PQ42340723QS09179

Рисунок 25 – Содержимое таблицы «car»

Содержимое таблицы «client» после наполнения тестовыми данными представлено на рисунке 26.

	id	first_name	last_name	phone_number	second_name
1	1	Стеф	Лысак	88005553535	Олеговна
2	2	Андрей	Васильев	89867261324	Аркадьевич
3	3	Станислав	Шишкин	89237507215	Николаевич
4	4	Никита	Дронин	89670123481	Олегович
5	5	Леонид	Храмов	89098765090	Владимирович
6	6	Александр	Золотов	89871975109	Иванович
7	7	Елена	Анисимова	89782906791	Артемовна
8	8	Захар	Романов	89456751742	Павлович
9	9	Федор	Пушкарев	89987333441	Игнатьевич
10	10	Дмитрий	Тарасов	89761987645	Романович

Рисунок 26 – Содержимое таблицы «client»

Содержимое таблицы «job» после наполнения тестовыми данными представлено на рисунке 27.

	price_rubles	car_id	finished_at	id	service_id	started_at
1	1000	1	2024-05-22 18:58:48.961000 +00:00	1	2	2024-05-22 18:28:48.961000 +00:00
2	1200	1	2024-05-22 18:58:48.961000 +00:00	2	1	2024-05-22 18:28:48.961000 +00:00
3	2500	3	2024-05-22 18:58:48.961000 +00:00	3	3	2024-05-22 18:28:48.961000 +00:00
4	700	4	2024-05-22 18:58:48.961000 +00:00	4	4	2024-05-22 18:28:48.961000 +00:00
5	700	9	2024-05-22 18:58:48.961000 +00:00	5	4	2024-05-22 18:28:48.961000 +00:00

Рисунок 27 – Содержимое таблицы «job»

Содержимое таблицы «job_status» после наполнения тестовыми данными представлено на рисунке 28.

	job_id	updated_at	status
1	1	2024-05-22 21:28:48.962000	CREATED
2	2	2024-05-22 21:28:48.988000	FINISHED
3	3	2024-05-22 21:28:48.989000	IN_PROCESS
4	4	2024-05-22 21:28:48.990000	REJECTED
5	5	2024-05-22 21:28:48.991000	IN_PROCESS

Рисунок 28 – Содержимое таблицы «job_status»

Содержимое таблицы «car_station» после наполнения тестовыми данными представлено на рисунке 29.

	id	address	legal_address	name
1	1	г. Пенза, ул. Московская 65	г. Пенза, ул. Московская 66	Золотые времена

Рисунок 29 – Содержимое таблицы «car_station»

Содержимое таблицы «master» после наполнения тестовыми данными представлено на рисунке 30.

	car_station_id	id	experience	first_name	last_name	phone_number	second_name
1	1	1	10 лет	Анатолий	Палено	89595675825	Терентьевич
2	1	2	7 лет	Александр	Герасимов	89895310157	Никитич
3	1	3	13 лет	Тихон	Смирнов	8967089730	Константинович
4	1	4	11 лет	Борис	Солдатов	89019496142	Вадимович
5	1	5	5 лет	Олег	Желтов	89781100129	Евгеньевич
6	1	6	2 года	Никита	Лукьянов	89012312281	Денисович
7	1	7	10 лет	Дмитрий	Иванов	89791743492	Захарович
8	1	8	15 лет	Кирилл	Климешин	89510117823	Николаевич

Рисунок 30 – Содержимое таблицы «master»

Содержимое таблицы «master_service» после наполнения тестовыми данными представлено на рисунке 31.

	price_rubles	id	master_id	service_id	work_duration_seconds
1	2000	1	1	1	1800
2	800	2	1	2	900
3	1000	3	2	3	2700
4	3000	4	3	4	900
5	4500	5	4	5	4500
6	4000	6	5	5	5700
7	4000	7	5	5	5700

Рисунок 31 – Содержимое таблицы «master_service»

Содержимое таблицы «service» после наполнения тестовыми данными представлено на рисунке 32.

	id	name
1	1	Шиномонтаж
2	2	Замена масла
3	3	Обслуживание тормозной системы
4	4	Заправка кондиционера
5	5	Ремонт КПП
6	6	Обслуживание подвески

Рисунок 32 – Содержимое таблицы service

Содержимое таблицы «master_job» после наполнения тестовыми данными представлено на рисунке 33.

	id	job_id	master_id
1	1	1	1
2	2	2	1
3	3	3	2
4	4	4	3
5	5	5	4

Рисунок 33 – Содержимое таблицы master_job

Листинг класса «DatasetCreator» приведен в приложении А.

3.2.2. Разработка запросов к базе данных

Аннотация @Query позволяет обозначить SQL – запрос в самом Java коде. Используя данную разработку, можно сочинить SQL – запрос прямо в

методе интерфейса репозитория, чтобы получить данные из БД. Это может быть крайне важно, если нужно осуществить разного рода запросы.

На рисунке 34 показан пример запроса в классе «ClientRepository»

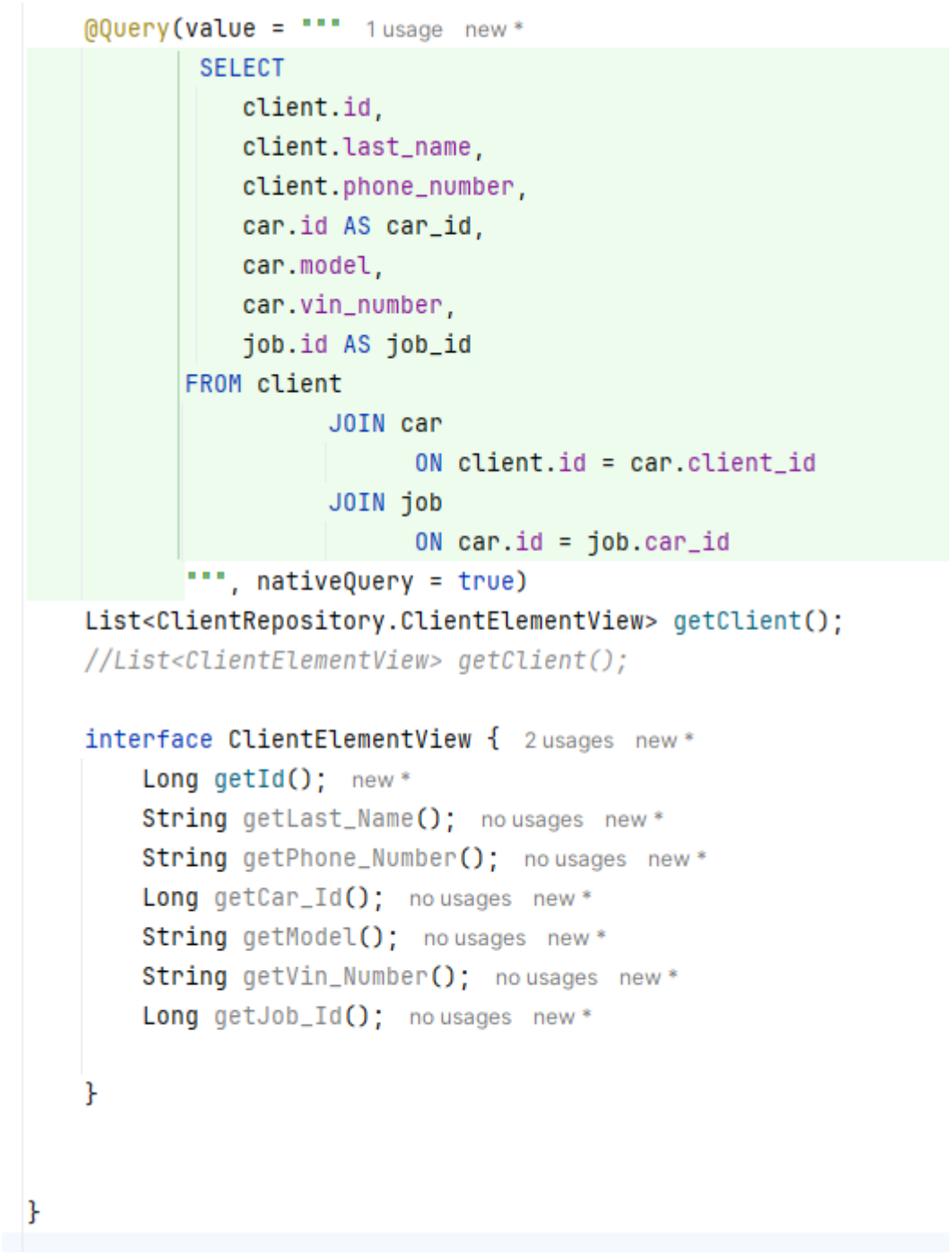


Рисунок 34 – Пример запроса в классе «ClientRepository»

В данном примере формируется список автомобилей, с указанием уникального идентификатора, модели и VIN-номера, которые посещали автомастерскую. Далее указаны фамилии владельцев автомобилей, их

мобильные номера и идентификаторы. Также указаны id-работ, чтобы точно определить с какой машиной проводится работа по её обслуживанию.

Результат выполнения запроса представлен на рисунке 35.

	id	last_name	phone_number	car_id	model	vin_number	job_id
1	1	Лысак	88005553535	1	Lada Vesta	JH4DA3462JS027249	1
2	1	Лысак	88005553535	1	Lada Vesta	JH4DA3462JS027249	2
3	2	Васильев	89867261324	3	Honda Civic	JK42339762JS07679	3
4	3	Шишкин	89237507215	4	Chevrolet Aveo	0P10339762JS07679	4
5	6	Золотов	89871975109	9	BMW X4	JK4233947120US076	5

Рисунок 35 – Результат запроса в базу данных

На рисунке 36 показан пример запроса в классе «JobRepository»

```
    @Query(value = """  1 usage  new *
SELECT
    job.id AS id,
    job.started_at AS startedAt,
    job.finished_at AS finishedAt,
    job_status.updated_at AS updatedAt,
    job_status.status AS status
FROM job
         JOIN job_status
              ON job.id = job_status.job_id""", nativeQuery = true)
    List<JobStatusView> getJobStatus();

    interface JobStatusView {  2 usages  new *

        Long getId();  new *

        Date getStartedAt();  no usages  new *

        Date getFinishedAt();  no usages  new *

        Date getUpdatedAt();  no usages  new *

        JobStatus getStatus();  no usages  new *
    }
}
```

Рисунок 36 – Пример запроса в классе «JobRepository»

Результат выполнения запроса представлен на рисунке 37

	id	started	finished	updated	status
1	2	2024-05-29 19:45:00.997000 +00:00	2024-05-29 20:15:00.997000 +00:00	2024-05-29 22:45:01.125000	ЗАВЕРШЕНА
2	5	2024-05-29 19:45:00.997000 +00:00	2024-05-29 20:00:00.997000 +00:00	2024-05-29 22:45:01.156000	В_ПРОЦЕССЕ
3	1	2024-05-29 19:45:00.997000 +00:00	2024-05-29 20:00:00.997000 +00:00	2024-05-29 22:45:01.001000	СОЗДАНА
4	4	2024-05-29 19:45:00.997000 +00:00	2024-05-29 20:00:00.997000 +00:00	2024-05-29 22:45:01.142000	ОТМЕНЕНА
5	3	2024-05-29 19:45:00.997000 +00:00	2024-05-29 20:30:00.997000 +00:00	2024-05-29 22:45:01.134000	В_ПРОЦЕССЕ

Рисунок 37 – Результат запроса в базу данных

Данный запрос выводит информацию о текущих работах. Отмечает дату начала, дату завершения работ, дату обновления статуса работ и текущий статус. Эта информация может быть полезна администратору-автомеханику при ведении статистики автомастерской.

На рисунке 38 показан пример запроса в классе «ServiceRepository»

```
@Query(value = """  1 usage  new *
        Select service.name,
               master_service.price_rubles AS priceRubles
                    from service
                            JOIN master_service
                                on service.id = master_service.service_id
        group by service.name, master_service.price_rubles;""", nativeQuery = true)
List<ServiceListElementView> getServiceList();

interface ServiceListElementView {  2 usages  new *

    Long getId();  new *
    String getName();  new *
}
}
```

Рисунок 38 – Пример запроса в классе «ServiceRepository»

Результат выполнения запроса представлен на рисунке 39

	name	pricerubles
1	Замена масла	800
2	Обслуживание тормозной системы	1000
3	Ремонт КПП	4500
4	Заправка кондиционера	3000
5	Шиномонтаж	2000

Рисунок 39 – Результат запроса в базу данных

Данный запрос выводит список доступных услуг.

На рисунке 40 показан пример запроса в классе «MasterRepository»

```
@Query(value = """ 1 usage  new *
    SELECT
        master_service.master_id AS masterId,
        master.last_name AS lastName,
        service.name AS serviceName,
        count(*) AS count
    FROM master_service
    JOIN service
         ON master_service.service_id = service.id
    Join master
         ON master_service.master_id = master.id
    group by master_service.master_id, service.name, master.last_name
    order by count DESC
""", nativeQuery = true)

List<MasterJobReportView> getJobReport();

interface MasterJobReportView {  2 usages  new *

    Long getMasterId();  no usages  new *

    String getLastName();  no usages  new *

    String getServiceName();  no usages  new *

    Long getCount();  no usages  new *
}
```

Рисунок 40 – Пример запроса в классе «MasterRepository»

Результат выполнения запроса представлен на рисунке 41

	masterid	lastname	servicename	count
1	4	Солдатов	Ремонт КПП	1
2	3	Смирнов	Заправка кондиционера	1
3	2	Герасимов	Обслуживание тормозной системы	1
4	5	Желтов	Ремонт КПП	2
5	1	Палено	Шиномонтаж	1
6	1	Палено	Замена масла	1

Рисунок 41 – Результат запроса в базу данных

С помощью данного запроса можно просматривать мастеров, которые закреплены за той или иной работой.

Таким образом, при помощи данных механизмов можно реализовать любой, специфический для выбранной СУБД запрос и ORM не накладывает ограничений на вариативность запроса, она лишь служит вспомогательным инструментом, который берет на себя большой объем шаблонной работы.

3.2.3. Описание JSON- объектов

Обозначение объектов JavaScript (JSON – JavaScript Object Notation) – стандартный текстовый формат для представления структурированных данных на основе синтаксиса объекта JavaScript. JSON имеет простой и

понятный синтаксис, что делает его удобным для чтения и записи как людьми, так и компьютерами. Он широко используется для передачи данных между клиентской и серверной частями веб-приложений, а также для хранения и обмена данными в различных форматах.

JSON-объекты состоят из пар "ключ-значение" и оформляются в фигурные скобки {}. Ключи являются строками, а значения могут быть строками, числами, логическими значениями (true или false), null, объектами или массивами.

Далее будет приведены листинги основных JSON объектов системы.

Часть JSON объекта [7] пользовательского представления приведен на Листинге 1

```
[
 {
  "name":
    "Обслуживание тормозной системы",
    "id":3
 },
 {
  "name":
    "Ремонт КПП",
    "id":5
 },
 {
  "name":
    "Заправка кондиционера",
    "id":4
 },
 {
  "name":
    "Обслуживание подвески",
    "id":6
},
{
```

```
 "name":
   "Заменамасла",
   "id":2
},
{
 "name":
   "КПП",
   "id":7
},
{
 "name":
   "Шиномонтаж",
   "id":1
}
]
```


Как видно из JSON представления система корректно составила список услуг в автомастерской.

Часть JSON объекта пользовательского представления приведен на Листинге 2


```
[
   {
      "id": null,
      "model": "Lada Vesta",
      "vin_Number": "JH4DA3462JS027249",
      "phone_Number": "88005553535",
      "last_Name": "Лысак",
      "job_Id": 1
   },
   {
      "id": null,
      "model": "Lada Vesta",
      "vin_Number": "JH4DA3462JS027249",
      "phone_Number": "88005553535",
      "last_Name": "Лысак",
```

```
    "job_Id": 2
  },
  {
    "id": null,
    "model": "Honda Civic",
    "vin_Number": "JK42339762JS07679",
    "phone_Number": "89867261324",
    "last_Name": "Васильев",
    "job_Id": 3
  },
  {
    "id": null,
    "model": "Chevrolet Aveo",
    "vin_Number": "OP10339762JS07679",
    "phone_Number": "89237507215",
    "last_Name": "Шишкин",
    "job_Id": 4
  },
  {
    "id": null,
    "model": "BMW X4",
    "vin_Number": "JK4233947120US076",
    "phone_Number": "89871975109",
    "last_Name": "Золотов",
    "job_Id": 5
  }
]
```


Как видно из JSON представления система корректно составила всех клиентов и их автомобили.

Часть JSON объекта пользовательского представления приведен на Листинге 3


```
[
  {
    "count": 2,
```

```
    "lastName": "Желтов",
    "serviceName": "Ремонт КПП",
  },
  {
    "count": 1,
    "lastName": "Солдатов",
    "serviceName": "Ремонт КПП",
  },
  {
    "count": 1,
    "lastName": "Смирнов",
    "serviceName": "Заправка кондиционера",
  },
  {
    "count": 1,
    "lastName": "Герасимов",
    "serviceName": "Обслуживание тормозной системы",
  },
  {
    "count": 1,
    "lastName": "Палено",
    "serviceName": "Шиномонтаж",
  },
  {
    "count": 1,
    "lastName": "Палено",
    "serviceName": "Замена масла",
  }
]
```


Как видно из представления JSON, система успешно составила данные о всех мастерах и их работах. Это свидетельствует о корректной работе системы.

Таким образом, JSON является удобным и популярным форматом для представления структурированных данных. Он обеспечивает простоту

использования и интеграции с различными языками программирования, что делает его незаменимым инструментом в современной веб-разработке.

3.3. Описание клиентской части

Клиентская часть – это одна из ключевых компонентов веб-приложения, которая отвечает за взаимодействие с пользователем и предоставление ему удобного интерфейса для работы с приложением.

Современные технологии разработки клиентской части, такие как фреймворки и библиотеки JavaScript (например, React) предоставляют разработчикам мощные инструменты для создания пользовательских интерфейсов. Они упрощают разработку, обеспечивают повторное использование кода и повышают производительность приложений.

При запуске приложения, пользователь попадает на страницу авторизации.

Далее при входе в систему за администратора – автомеханика, мы попадаем на его главную страницу (Рисунок 42). Здесь можно посмотреть все текущие работы и их статусы.

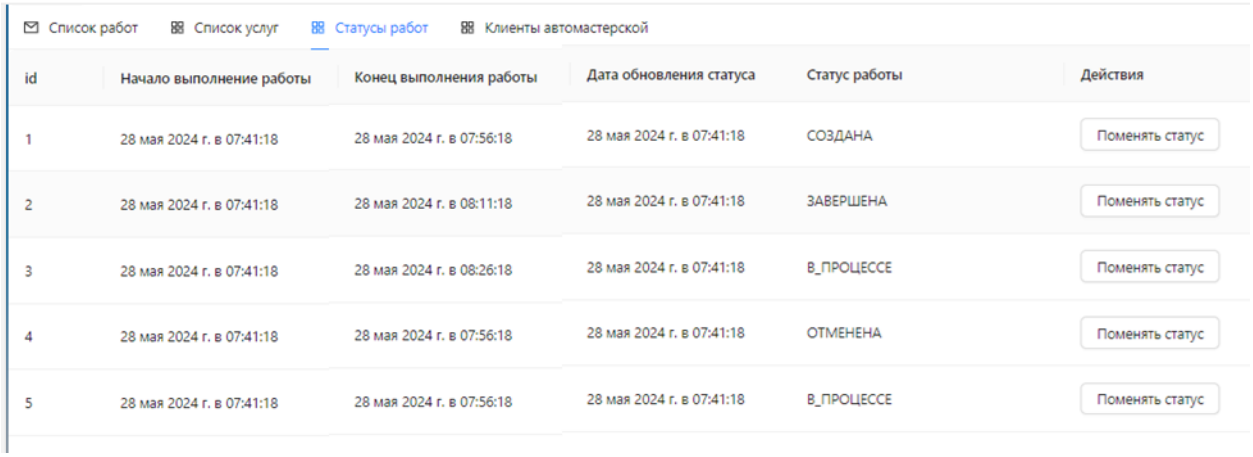


Рисунок 42 – Главная страница администратора – автомеханика

Далее можно увидеть «Список работ» (Рисунок 43). В данной вкладке отображена занятость мастера за конкретной работой, а также количество работ закрепленное за автомехаником.

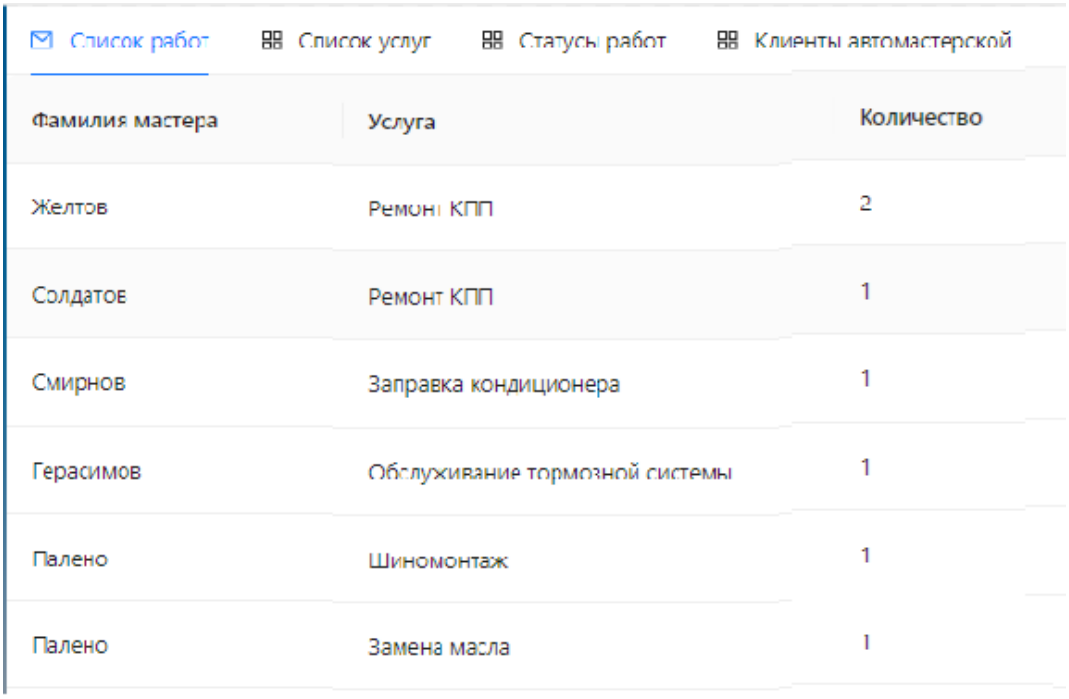


Рисунок 43 – Список работ

Затем перемещаемся на вкладку «Список услуг» (Рисунок 44). В ней отображены все услуги, с возможностью корректировки названия, цены и длительности выполнения конкретной услуги.

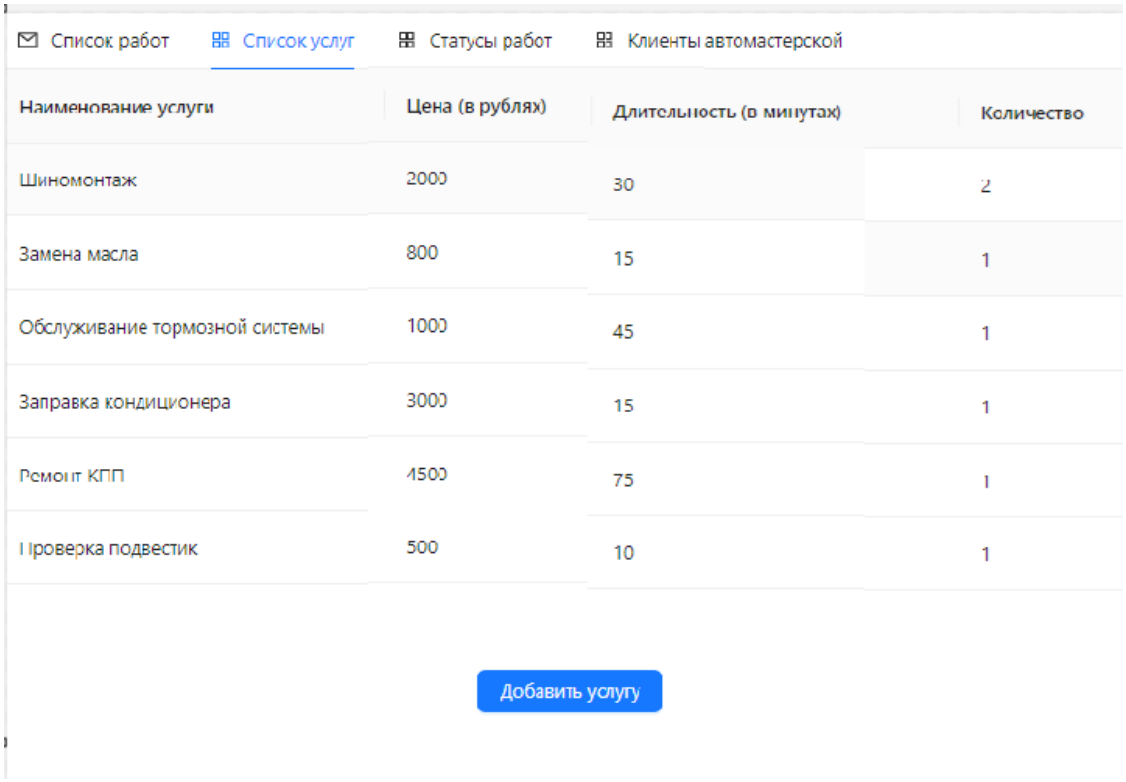
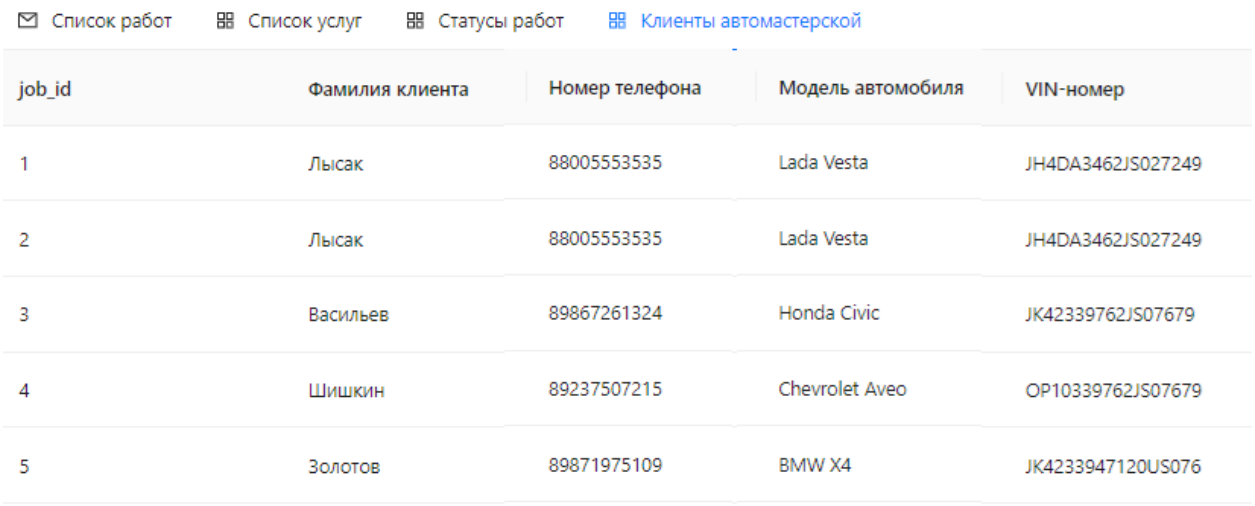


Рисунок 44 – Список услуг

Следующая вкладка носит статистический характер. На ней отображены текущие клиенты, их автомобили, а также идентификатор работы. (Рисунок 45).



job_id	Фамилия клиента	Номер телефона	Модель автомобиля	VIN-номер
1	Лысак	88005553535	Lada Vesta	JH4DA3462JS027249
2	Лысак	88005553535	Lada Vesta	JH4DA3462JS027249
3	Васильев	89867261324	Honda Civic	JK42339762JS07679
4	Шишкин	89237507215	Chevrolet Aveo	OP10339762JS07679
5	Золотов	89871975109	BMW X4	JK4233947120US076

Рисунок 45 – Текущие клиенты

Далее, если мы авторизуемся на страницу клиента, то сможем увидеть список машин конкретного клиента (Рисунок 46).

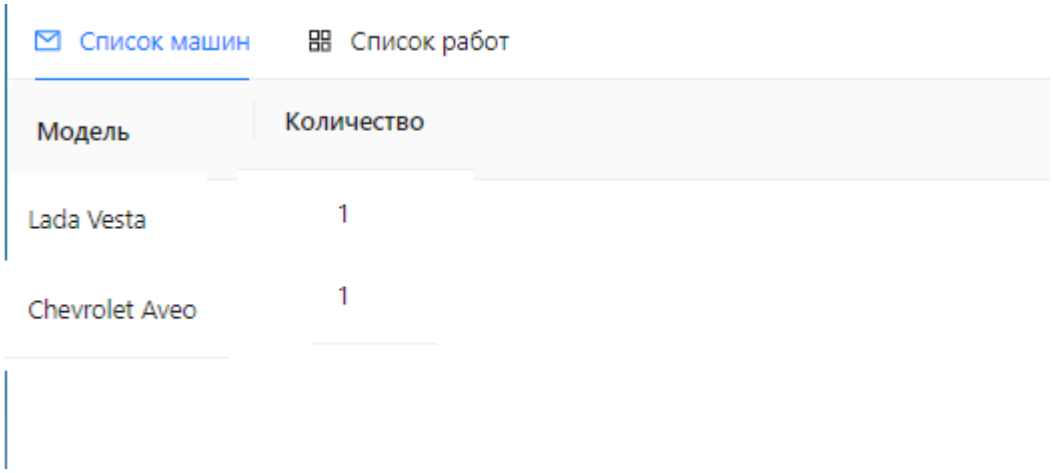


Рисунок 46 – Список машин клиента

Затем по клику на вкладку «Список работ» можно подать заявку на работу и просмотреть прошлые работы в данном сервисе. (Рисунок 47).

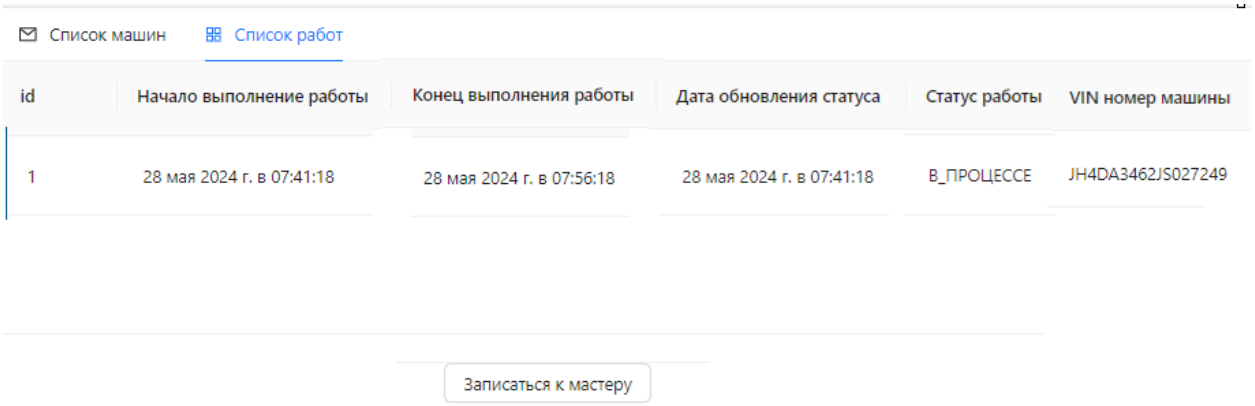


Рисунок 47 – Список работ

Также можно просмотреть загруженность определенного мастера со стороны «Администратора - автомеханика» (Рисунок 48).

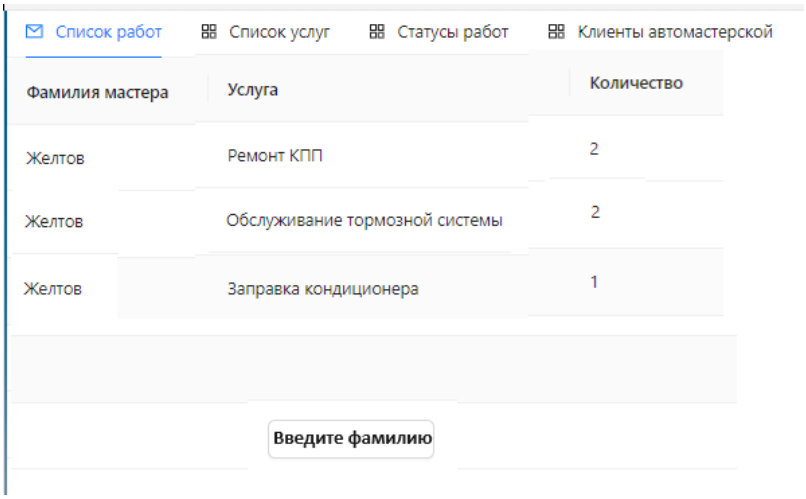


Рисунок 48 – Загруженность определенного мастера

Затем клиент может просмотреть загруженность сервиса на определенный день, выбрав удобную дату и услугу, отправить заявку на рассмотрение. (Рисунок 49).

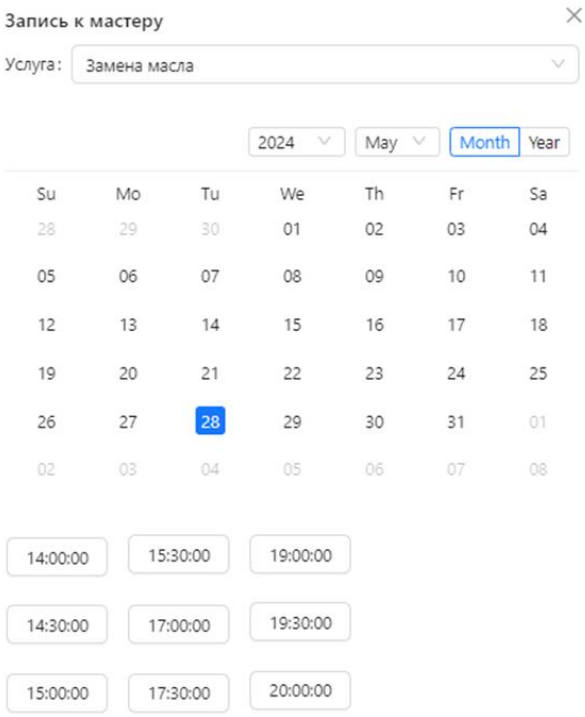


Рисунок 49 – Время приема на определенный день

Далее можно вывести информацию об определенном клиенте, его автомобилях и идентификаторе работы. (Рисунок 50).

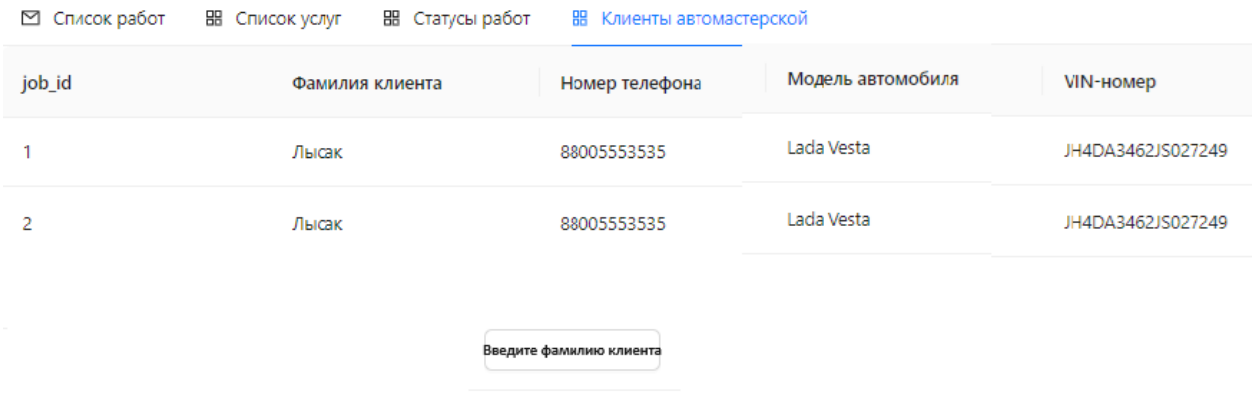


Рисунок 50 – Информация об определенном клиенте

В данной разработке присутствует как клиентская часть, так и административная часть. Система проверяет наличие пользователя по номеру телефона и паролю.

4.Тестироваие приложения

4.1. Выбор режима тестирования приложения

Существует достаточно много видов тестирования программного продукта. Были рассмотрены следующие виды тестирования [8]:

- функциональное тестирование;
- нефункциональное тестирование;
- интеграционное тестирование.

Функциональное тестирование – направлено на проверку того, какие функции приложения реализованы, и что они работают верным образом. Данный метод тестирования является одним из самых применяемых, так как позволяет наиболее полно проверить программу на соответствие требованиям.

Нефункциональное тестирование – вид тестирования, направленный на проверку нефункциональных особенностей приложения, таких как удобство использования, совместимость, производительность, безопасность и т.д.

Интеграционное тестирование – направлено на проверку взаимодействия между несколькими частями приложения. Этот вид тестирования позволяет проверить корректность взаимодействия частей ПО между собой.

Каждый из этих видов тестирования имеет свою цель и помогает обеспечить качество и надежность программного продукта. Выбор конкретных видов тестирования зависит от требований проекта и его особенностей.

Принято решение осуществить проверку программы с помощью функционального тестирования. Такой выбор обусловлен тем, что к программе были предъявлены в основном функциональные требования, выполнение которых нужно проверить.

4.2. Тестирование Web – сервиса

Для основных вариантов использования были выполнены функциональные тесты (таблица 17).

Таблица 17 – Функциональные тесты Web - сервиса

№	Назначение теста	Описание теста	Результат теста
1	Тест предназначен для проверки аутентификации в систему под ролью «администратор-автомеханик»	Входные данные: Логин – 89595675825 Пароль – test Выходные данные: Переход на страницу «администратора - автомеханика»	Тест пройден успешно. Был выполнен переход на страницу «администратора-автомеханика» (Рисунок 51)
2	Тест предназначен для проверки аутентификации в систему под ролью «администратор-автомеханик» при вводе неправильного номера телефона (логина) и пароля	Входные данные: Логин – указать неверный номер телефона (который не зарегистрирован в системе) Пароль – указать неверный пароль (который не зарегистрирован в системе) Выходные данные: Остаться на странице авторизации	Тест пройден успешно. Система сообщила о том, что доступ к ресурсу запрещен и необходимо авторизоваться (Рисунок 52)
3	Тест предназначен для добавления новой услуги на странице «Список услуг»	Входные данные: Ввести в поле ввода услугу, цену и длительность выполнения Выходные данные: Обновленный список услуг	Тест пройден успешно (Рисунок 53).
4	Тест предназначен изменения статуса работы на вкладке «Статусы работ»	Входные данные: Перейти на страницу «Статусы работ», кликнуть по кнопке «Поменять статус»	Тест успешно пройден. Статус поменялся с «В_Процессе» на «Завершен»(Рисунок 54)

Продолжение таблицы 17

№	Назначение теста	Описание теста	Результат теста
5	Тест предназначен для записи в автосервис под ролью «Клиент»	Входные данные: Авторизоваться под ролью «Клиента» Нажать на кнопку «Записаться к мастеру» Выбрать нужную услугу и любую свободную, подходящую дату и время	Тест успешно пройден (Рисунок 55). Была создана запись на ремонт.

Ниже приведены рисунки выполнения функциональных тестов, представленных в таблице 20.

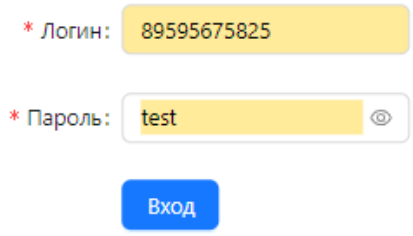


Рисунок 51 – Авторизация под ролью «администратор-автомеханик»

На рисунке 52 показано, что пользователь не смог авторизоваться

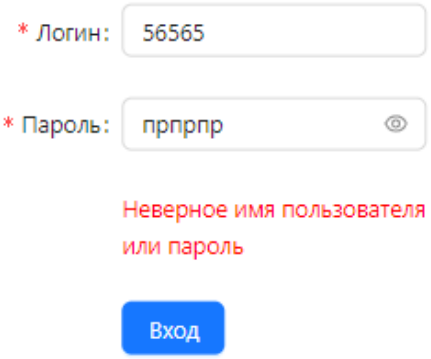


Рисунок 52 – Авторизация прошла не успешно

На рисунке 53 видно, как добавилась услуга «Новая услуга»

Наименование услуги	Цена (в рублях)	Длительность (в минутах)
Шиномонтаж	2000	30
Замена масла	800	15
Обслуживание тормозной системы	1000	45
Заправка кондиционера	3000	15
Ремонт КПП	4500	75
Проверка подвестик	500	10
Новая услуга	500	60

Рисунок 53 – Добавление новой услуги прошло успешно

Администратор-автомеханик может менять статусы работ, исходя из их процесса. (Рисунок 54).

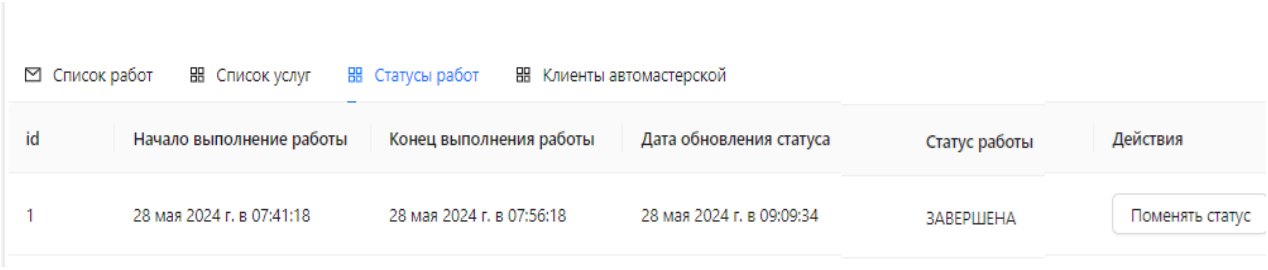


Рисунок 54 – Изменение статуса работ

Можно выбрать любой подходящий день и увидеть свободное время. (Рисунок 55).

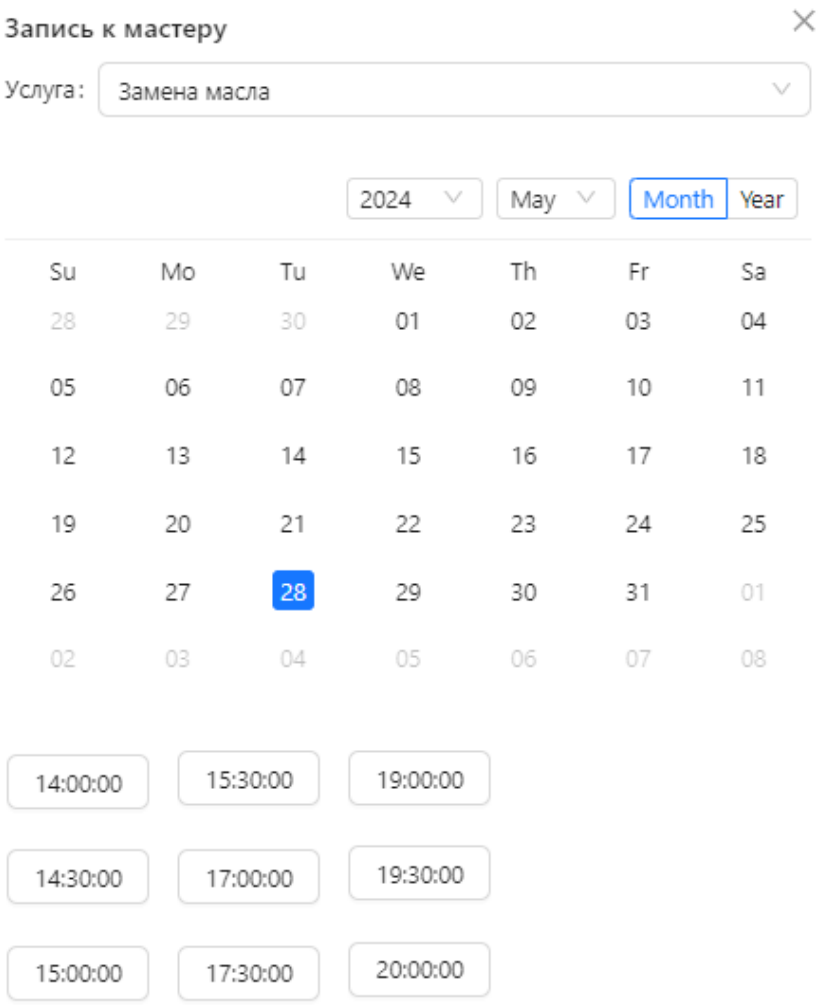


Рисунок 55 – Запись на ремонт

В целом, тестирование является важным этапом разработки программного продукта, который помогает обеспечить его качество, надежность и соответствие требованиям пользователей. Результаты тестирования позволяют выявить и исправить ошибки, а также повысить удовлетворенность пользователей продуктом. Все тесты были успешно выполнены. Ошибок не обнаружено.

Заключение

В ходе выполнения выпускной квалификационной работы был разработан Web-сервис записи клиентов в автомастерскую.

Проведено исследование предметной области. На основе сопоставления текущих аналогов и составления требований была изложена задача на разработку Web – сервиса, а также заданы функциональные требования к будущей системе.

Программные средства спроектированы и реализованы в соответствие с MVC архитектурой. Для работы с базой данных был разработан набор классов, работающий по принципу объектно-реляционного отображения.

Разработка приложения велась в среде разработки IntelliJI IDEA на языке программирования Java в операционной системе Windows. Тестирование программного средства проводилось в аналогичной операционной системе. Все тесты были пройдены успешно и в результате тестирования не выявлено ошибок.

Таким образом требования ТЗ выполнено в полном объеме в соответствии с календарным планом.

Список использованных источников

1. Козар А.Н. Совершенствование услуг автосервисных предприятий / А.Н. Козар. - Москва: Русайнс, 2017. - 352 с.
2. Autoservice58 [Электронный ресурс]. URL: https://avto-servis58.ru/diagnostika-avtomobilya.php# (дата обращения 20.04.2024).
3. Euroauto сервис [Электронный ресурс]. URL: https://euroauto.ru/zapis_na_sto/| (дата обращения 20.04.2024).
4. Арлоу Д., UML 2 и Унифицированный процесс, 2-е издание. Практический объектно-ориентированный анализ и проектирование. / Д. Арлоу, А. Нейштадт – Санкт Петербург: Символ Плюс, 2007. – 624 с.
5. Верников Г., Основы методологии IDEF1X [Электронный ресурс]. URL: https://www.cfin.ru/vernikov/idef/idef1x.shtml (дата обращения 29.04.2022)
6. Крикунов М.М., Основы баз данных / М.М Крикунов – Москва: ДМК Пресс, 2019. – 652с.
7. Что такое JSON [Электронный ресурс]. URL: https://habr.com/ru/post/554274/ (дата обращения 30.04.2022).
8. Куликов С.С., Тестирование программного обеспечения. Базовый курс (3-изд.) / С.С. Куликов – Минск: EPAM Systems, 2021. – 300 с.

Приложение А
ЛИСТИНГ ПРОГРАММЫ

Листинг – Файл CarEntity.java

```
package org.pgu.stefan.store.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.*;

import java.util.Date;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity // указывает, что класс является сущностью, которая будет
отображаться в базе данных.
@Table(name = "car") //указывает имя таблицы, в которой будет храниться
информация о сущности.
public class CarEntity {
  @Id // указывает, что поле id является первичным ключом сущности
  @GeneratedValue // указывает, что значение id будет генерироваться
автоматически.
  private Long id;

  private String model;

  private String vinNumber;

  private Date releaseDate;

  @JsonIgnore
  @ManyToOne(optional = false) // указывает на отношение "многие-к-одному"
  private ClientEntity client;
}
```

Листинг – Файл ClientEntity.java

```
package org.pgu.stefan.store.entity;

import jakarta.persistence.*;
import lombok.*;

import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity // указывает, что класс является сущностью, которая будет
отображаться в базе данных.
@Table(name = "client") //указывает имя таблицы, в которой будет храниться
информация о сущности.
public class ClientEntity {
  @Id // указывает, что поле id является первичным ключом сущности
  @GeneratedValue // указывает, что значение id будет генерироваться
автоматически.
  private Long id;
```

```
  private String firstName;

  private String secondName;

  private String lastName;

  private String phoneNumber;

  @OneToMany(mappedBy = "client") // указывает на отношение "один-ко-многим"
  private List<CarEntity> cars;

}
```

Листинг – Файл JobEntity.java

```
package org.pgu.stefan.store.entity;

import jakarta.persistence.*;
import lombok.*;

import java.time.ZonedDateTime;
import java.util.Date;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity // указывает, что класс является сущностью, которая будет
отображаться в базе данных.
@Table(name = "job") //указывает имя таблицы, в которой будет храниться
информация о сущности.
public class JobEntity {

  @Id // указывает, что поле id является первичным ключом сущности
  @GeneratedValue // указывает, что значение id будет генерироваться
автоматически.
  private Long id;

  private ZonedDateTime startedAt;
  private ZonedDateTime finishedAt;

  private Double priceRubles;

  @ManyToOne(optional = false)
  private ServiceEntity service;

  @ManyToOne(optional = false) // указывает на отношение "многие-к-одному"
  private CarEntity car;
}
```

Листинг – Файл CarStationEntity.java

```
package org.pgu.stefan.store.entity;

import jakarta.persistence.*;
import lombok.*;

import java.util.List;
```

```
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity // указывает, что класс является сущностью, которая будет
отображаться в базе данных.
@Table(name = "car_station")//указывает имя таблицы, в которой будет
храниться информация о сущности.
public class CarStationEntity {

  @Id // указывает, что поле id является первичным ключом сущности
  @GeneratedValue // указывает, что значение id будет генерироваться
автоматически.
  private Long id;

  private String name;

  private String address;

  private String legalAddress;

  @OneToMany(mappedBy = "carStation") // указывает на отношение "один-ко-
многим"
  private List<MasterEntity> masters;

}
```

Листинг – Файл MasterEntity.java

```
package org.pgu.stefan.store.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.*;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity // указывает, что класс является сущностью, которая будет
отображаться в базе данных.
@Table(name = "master") //указывает имя таблицы, в которой будет храниться
информация о сущности.
public class MasterEntity {

  @Id // указывает, что поле id является первичным ключом сущности
  @GeneratedValue // указывает, что значение id будет генерироваться
автоматически.
  private Long id;

  private String firstName;

  private String secondName;

  private String lastName;

  private String phoneNumber;

  private String experience;
```

```
  @JsonIgnore
  @ManyToOne(optional = false) // указывает на отношение "многие-к-одному"
  private CarStationEntity carStation;
}
```

Листинг – Файл MasterJobEntity.java

```
package org.pgu.stefan.store.entity;

import jakarta.persistence.*;
import lombok.*;

@Setter
@Getter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity // указывает, что класс является сущностью, которая будет
отображаться в базе данных.
@Table(name = "master_job")//указывает имя таблицы, в которой будет храниться
информация о сущности.
public class MasterJobEntity {

    @Id // указывает, что поле id является первичным ключом сущности
    @GeneratedValue // указывает, что значение id будет генерироваться
автоматически.
    private Long id;

    @ManyToOne(optional = false) // указывает на отношение "многие-к-одному"
    private MasterEntity master;

    @ManyToOne(optional = false) // указывает на отношение "многие-к-одному"
    private JobEntity job;
}
```

Листинг – Файл ServiceEntity.java

```
package org.pgu.stefan.store.entity;

import jakarta.persistence.*;
import lombok.*;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity // указывает, что класс является сущностью, которая будет
отображаться в базе данных.
@Table(name = "service") ")//указывает имя таблицы, в которой будет храниться
информация о сущности.
public class ServiceEntity {

  @Id // указывает, что поле id является первичным ключом сущности
  @GeneratedValue // указывает, что значение id будет генерироваться
автоматически.
  private Long id;
```

```
  private String name;
}
```

Листинг – Файл CarRepository.java

```
package org.pgu.stefan.store.repository;

import org.pgu.stefan.store.entity.CarEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import java.util.List;

@RepositoryRestResource(collectionResourceRel = "cars", path = "cars")
public interface CarRepository extends JpaRepository<CarEntity, Long> {
// Этот метод аннотирован аннотацией Query и определяет пользовательский SQL-запрос для получения отчета о работе мастеров. Запрос выполняется нативно с помощью аннотации//
    @Query(value = """
        SELECT
            car.model AS model,
            count(*) AS count
        FROM job
                JOIN car
                        ON job.car_id = car.id
        group by car.model
        ORDER BY count DESC""", nativeQuery = true)
    List<TopCarElementView> getTopCars();
//Этот вложенный интерфейс определяет проекцию для отчета о работе мастеров. Он определяет методы для получения именованных полей из результата запроса//
    interface TopCarElementView {
        String getModel();
        Long getCount();
    }
}
```

Листинг – Файл ClientRepository.java

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import java.util.List;

@RepositoryRestResource(collectionResourceRel = "clients", path = "clients")
public interface ClientRepository extends JpaRepository<ClientEntity, Long> {
// Этот метод аннотирован аннотацией Query и определяет пользовательский SQL-запрос для получения отчета о работе мастеров. Запрос выполняется нативно с помощью аннотации//
    @Query(value = """
            SELECT
               client.last_name,
               client.phone_number,
               car.model,
               car.vin_number,
               job.id AS job_id
           FROM client
                     JOIN car
                           ON client.id = car.client_id
```

```
                    JOIN job
                         ON car.id = job.car_id""", nativeQuery = true)
    List<ClientRepository.ClientElementView> getClient();
    //List<ClientElementView> getClient();
//Этот вложенный интерфейс определяет проекцию для отчета о работе мастеров.
Он определяет методы для получения именованных полей из результата запроса//
    interface ClientElementView {
        Long getId();
        String getLast_Name();
        String getPhone_Number();
        String getModel();
        String getVin_Number();
        Long getJob_Id();

    }


}
```

Листинг – Файл MasterRepository.java

```
package org.pgu.stefan.store.repository;

import jakarta.transaction.Transactional;
import org.pgu.stefan.api.controller.MasterController;
import org.pgu.stefan.store.entity.MasterEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.config.Projection;

import java.util.Date;
import java.util.List;

@RepositoryRestResource(collectionResourceRel = "masters", path = "masters")
public interface MasterRepository extends JpaRepository<MasterEntity, Long> {
// Этот метод аннотирован аннотацией Query и определяет пользовательский SQL-
запрос для получения отчета о работе мастеров. Запрос выполняется нативно с
помощью аннотации//
    @Query(value = """
        SELECT
            master_service.master_id AS masterId,
            master.last_name AS lastName,
            service.name AS serviceName,
            count(*) AS count
        FROM master_service
        JOIN service
             ON master_service.service_id = service.id
        Join master
             ON master_service.master_id = master.id
        group by master_service.master_id, service.name, master.last_name
        order by count DESC""", nativeQuery = true)

    List<MasterJobReportView> getJobReport();
//Этот вложенный интерфейс определяет проекцию для отчета о работе мастеров.
Он определяет методы для получения именованных полей из результата запроса//
    interface MasterJobReportView {

        Long getMasterId();
```

```
        String getLastName();

        String getServiceName();

        Long getCount();
    }
}
```

Листинг – Файл ServiceRepository.java

```
package org.pgu.stefan.store.repository;

import org.pgu.stefan.store.entity.CarEntity;
import org.pgu.stefan.store.entity.JobEntity;
import org.pgu.stefan.store.entity.MasterEntity;
import org.pgu.stefan.store.entity.ServiceEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import java.util.List;

@RepositoryRestResource(collectionResourceRel = "service", path = "service")
public interface ServiceRepository extends JpaRepository<ServiceEntity, Long>
// Этот метод аннотирован аннотацией Query и определяет пользовательский SQL-запрос для получения отчета о работе мастеров. Запрос выполняется нативно с помощью аннотации//
{

    @Query(value = """

            Select service.name,
                   master_service.price_rules AS priceRules
                        from service
                                JOIN master_service
                                    on service.id = master_service.service_id
            group by service.name, master_service.price_rules;""",
nativeQuery = true)
    List<ServiceListElementView> getServiceList();
//Этот вложенный интерфейс определяет проекцию для отчета о работе мастеров. Он определяет методы для получения именованных полей из результата запроса//
    interface ServiceListElementView {

        Long getId();
        String getName();
    }
}
```

Листинг – Файл DatasetCreator.java

```
package org.pgu.stefan.inital;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.pgu.stefan.domain.JobStatus;
import org.pgu.stefan.store.entity.*;
import org.pgu.stefan.store.repository.*;
import org.springframework.boot.context.event.ApplicationReadyEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
```

```
import java.time.Duration;
import java.time.ZonedDateTime;
import java.util.Date;
import java.util.GregorianCalendar;

@RequiredArgsConstructor
@Component
public class DatasetCreator {

    private final CarRepository carRepository;
    private final JobRepository jobRepository;
    private final ClientRepository clientRepository;
    private final MasterRepository masterRepository;
    private final ServiceRepository serviceRepository;
    private final JobStatusRepository jobStatusRepository;
    private final MasterJobRepository masterJobRepository;
    private final CarStationRepository carStationRepository;
    private final MasterServiceRepository masterServiceRepository;

    @EventListener(ApplicationReadyEvent.class)
    @Transactional
    public void initStatuses() {

        // Создание клиентов
        ClientEntity client = clientRepository.save(
                ClientEntity.builder()
                        .firstName("Стеф")
                        .lastName("Лысак")
                        .secondName("Олеговна")
                        .phoneNumber("88005553535")
                        .build()
        );
        ClientEntity client2 = clientRepository.save(
                ClientEntity.builder()
                        .firstName("Андрей")
                        .lastName("Васильев")
                        .secondName("Аркадьевич")
                        .phoneNumber("89867261324")
                        .build()
        );
        ClientEntity client3 = clientRepository.save(
                ClientEntity.builder()
                        .firstName("Станислав")
                        .lastName("Шишкин")
                        .secondName("Николаевич")
                        .phoneNumber("89237507215")
                        .build()
        );
        // Создание клиентов
        // Создание машин
        CarEntity carVesta = carRepository.save(
                CarEntity.builder()
                        .model("Lada Vesta")
                        .vinNumber("JH4DA3462JS027249")
                        .releaseDate(new GregorianCalendar(2015, 6,
10).getTime())
                        .client(client)
                        .build()
        );

        CarEntity carToyota = carRepository.save(
                CarEntity.builder()
```

```
                        .model("Toyota Rav 4")
                        .vinNumber("WH4233462JS027249")
                        .releaseDate(new GregorianCalendar(2011, 5,
14).getTime())
                        .client(client)
                        .build()
        );

        // Создание машин
        // Создание СТО
        CarStationEntity carStation = carStationRepository.save(
                CarStationEntity.builder()
                        .name("Золотые времена")
                        .address("г. Пенза, ул. Московская 65")
                        .legalAddress("г. Пенза, ул. Московская 66")
                        .build()
        );
        // Создание СТО
        // Создание мастеров
        MasterEntity master = masterRepository.save(
                MasterEntity.builder()
                        .firstName("Анатолий")
                        .lastName("Палено")
                        .secondName("Терентьевич")
                        .phoneNumber("89595675825")
                        .experience("10 лет")
                        .carStation(carStation)
                        .build()
        );
        // Создание мастеров
        // Создание услуг
        ServiceEntity tireService = serviceRepository.save(
                ServiceEntity.builder()
                        .name("Шиномонтаж")
                        .build()
        );
        ServiceEntity carsuspensionService = serviceRepository.save(
                ServiceEntity.builder()
                        .name("Обслуживание подвески")
                        .build()
        );
        // Создание услуг

        // Закрепление услуг за мастерами
        masterServiceRepository.save(
                MasterServiceEntity.builder()
                        .master(master)
                        .service(tireService)
                        .priceRubles(2000D)

.workDurationSeconds(Duration.ofMinutes(30).toSeconds())
                        .build()
        );
        masterServiceRepository.save(
                MasterServiceEntity.builder()
                        .master(master)
                        .service(oilChangeService)
                        .priceRubles(800D)

.workDurationSeconds(Duration.ofMinutes(15).toSeconds())
                        .build()
        );
        masterServiceRepository.save(
```

```
                MasterServiceEntity.builder()
                        .master(master2)
                        .service(brakesystemService)
                        .priceRubles(1000D)

.workDurationSeconds(Duration.ofMinutes(45).toSeconds())
                        .build()
        );
        masterServiceRepository.save(
                MasterServiceEntity.builder()
                        .master(master3)
                        .service(conditionerService)
                        .priceRubles(3000D)

.workDurationSeconds(Duration.ofMinutes(15).toSeconds())
                        .build()
        );
        masterServiceRepository.save(
                MasterServiceEntity.builder()
                        .master(master4)
                        .service(transmissionService)
                        .priceRubles(4500D)

.workDurationSeconds(Duration.ofMinutes(75).toSeconds())
                        .build()
        );
        // Закрепление услуг за мастерами

        ZonedDateTime date = new GregorianCalendar().toZonedDateTime();

        // Назначение работы за местером
        JobEntity vestaOilChangeJob = jobRepository.save(
                JobEntity.builder()
                        .startedAt(date)
                        .finishedAt(date.plus(Duration.ofMinutes(30)))
                        .service(oilChangeService)
                        .car(carVesta)
                        .priceRubles(1000D)
                        .build()
        );

        masterJobRepository.save(
                MasterJobEntity.builder()
                        .master(master)
                        .job(vestaOilChangeJob)
                        .build()
        );

        jobStatusRepository.save(
                JobStatusEntity.builder()
                        .jobId(vestaOilChangeJob.getId())
                        .status(JobStatus.СОЗДАН)
                        .updatedAt(new GregorianCalendar().getTime())
                        .build()
        );
        // Назначение работы за местером
        // Назначение работы за местером
        JobEntity vestatireJob = jobRepository.save(
                JobEntity.builder()
                        .startedAt(date)
                        .finishedAt(date.plus(Duration.ofMinutes(30)))
                        .service(tireService)
                        .car(carVesta)
```

```
                        .priceRubles(1200D)
                        .build()
        );
        masterJobRepository.save(
                MasterJobEntity.builder()
                        .master(master)
                        .job(vestatireJob)
                        .build()
        );
        jobStatusRepository.save(
                JobStatusEntity.builder()
                        .jobId(vestatireJob.getId())
                        .status(JobStatus.ЗАВЕРШЕН)
                        .updatedAt(new GregorianCalendar().getTime())
                        .build()
        );
        // Назначение работы за местером

            }
}
```

Листинг – Файл ClientCar.js

```
import '../../App.css';
import { Table } from "antd";

import { useState, useEffect } from 'react';

const columns = [
    {
        title: 'Модель',
        dataIndex: 'model',
        key: 'model',
    },
    {
        title: 'Количество',
        dataIndex: 'count',
        key: 'count',
    },
];

function ClientCar() {

    const [clientCar, setClientCar] = useState(null);

    useEffect(() => {

        // Проверяем, если данные уже загружены, то прекращаем выполнение
        if (clientCar) {
            return;
        }

        // Загружаем данные с сервера
        fetch('http://localhost:8080/api/client/car')
            .then((response) => response.json())
            .then((clientCarJson) => {
                setClientCar(clientCarJson);
            })
            .catch((err) => {
```

```
                console.log(err.message);
            });
    }, []); // Пустой массив зависимостей, чтобы useEffect выполнился только
один раз

    return (
        <div className="App">
            <Table dataSource={clientCar} columns={columns} />
        </div>
    );
}

export default ClientCar;
```