МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт машиностроения
(наименование института полностью)

Кафедра Проектирование и эксплуатация автомобилей
(наименование)

23.03.03 Эксплуатация транспортно-технологических машин и комплексов
(код и наименование направления подготовки, специальности)

Автомобили и автомобильное хозяйство
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО

Студент Д. С. Гипиков
(И.О. Фамилия) (личная подпись)

Руководитель к. э. н., доцент, Л.Л. Чумаков
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Выпускная квалификационная работа на тему «Разработка программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО» состоит из: пояснительной записки 46 страниц, 18 рисунков, 2 таблиц, 20 источников и 1 приложения.

В выпускной квалификационной работе разработано программное обеспечения для управления и мониторинга цехов ТО и ТО на СТО.

Целью работы является автоматизация процессов управления и обеспечение мониторинга цехов ТО и ТР на СТО.

В работе представлены: введение, анализ и разработка требований к программному обеспечению для мониторинга и управления цехами ТО и ТР на СТО, проектирование архитектуры программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО, разработка и реализация программного обеспечения, дальнейшее возможное масштабирование и развитие, разработка участка диагностики, заключение и приложение.

Содержание

Введение

Ежегодно растущее количество автомобилей на дорогах страны требует все большего количества станций технического обслуживания и ремонта для поддержания их в хорошем техническом состоянии.

В свою очередь такое положение способствует побуждению малых предпринимателей занимать вакантные места на растущем рынке и организовывать станции технического обслуживания.

Для повышения эффективности и помощи предпринимателям в ведении малых станций технического обслуживания необходимо применять программное обеспечение.

Программное обеспечения выполняет следующие функции:

- хранение информации о клиентах и их автомобилях,
- обработки информации о клиентах и их автомобилях,
- хранения истории выполненных работ на автомобиле,
- сбора статистики выполненных работ,
- мониторинг загруженности цехов и постов.

Программное обеспечение в свою очередь должно быть легко внедряемым в процессы малой станции технического обслуживания, чтобы его могло позволить себе как можно больше предпринимателей.

Целью данной работы является разработка программного обеспечения для мониторинга и управления цехами технического обслуживания и ремонта на станциях технического обслуживания, которое будет доступно для малых предпринимателей.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- анализ и разработка требований к программному обеспечению,
- выбор инструментов разработки программного обеспечения,
- проектирование архитектуры программного обеспечения,
- проектирование логической модели базы данных,
- проектирование физической модели базы данных,

- реализация спроектированного программного обеспечения,
- тестирование программного обеспечения,
- разработка дальнейшего возможного развития,
- разработка участка для использования программного обеспечения.

Разрабатываемое программное обеспечение должно автоматизировать процессы выполнения работ на малых станциях технического обслуживания в цехах технического обслуживания и текущего ремонта.

Программное обеспечения необходимо разработать с возможностью масштабирования, и развития для использования его на станциях технического обслуживания больших размерах и мощностей, а также добавления в него новой функциональности для взаимодействия со складом, другими цехами и управления персоналом.

1 Анализ и разработка требований к программному обеспечению для мониторинга и управления цехами ТО и ТР на СТО

1.1 Описание и анализ задачи по разработке программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО

Предметной областью выпускной квалификационной работы является разработка программного обеспечения, web-сервиса (далее - сервис), для автоматизации процессов управления и мониторинга цехами технического обслуживания (ТО) и текущего ремонта (ТР) на станциях технического обслуживания (СТО) малых размеров и мощностей.

Автоматизация процессов на СТО способствует повышению эффективности, определению четких этапов и увеличению прозрачности выполняемых работ в цехах ТО и ТР.

Без автоматизации процессов в цехах ТО и ТР может быть выбран неоптимальный способ выполнения работ, что увеличивает количество затрачиваемых ресурсов.

В рамках выполнения выпускной квалификационной работы должен быть разработан сервис, предоставляющий функции:

− регистрация нового автомобиля,

− отображение списка всех зарегистрированных автомобилей,

− автоматическое управление потоком автомобилей в цехах ТО и ТР в зависимости от необходимых работ над автомобилем и длительностью их выполнения,

− мониторинг очереди выдачи автомобилей клиентам и загруженности цехов ТО и ТР,

− сбор и хранение статистики по выполненным работам на постах за день/неделю/месяц/год,

− хранение и отображение истории выполненных работ по каждому автомобилю.

1.2 Поиск и анализ аналогов разрабатываемого программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО

В настоящее время существует ряд программных решений от компании «Фирма 1С» [5] для автоматизации процессов на СТО, аналогичных по предоставляемой функциональности разрабатываемого сервиса.

Программное решение: 1С: Предприятие 8. Автосервис [6].

Программное решение предназначено для автоматизации процессов и повышения эффективности на малых СТО, автомойках и автосервисах. Разработано на основе функционала программного продукта «1С:Управление нашей фирмой» с учетом особенностей предприятий автобизнеса и обеспечивает следующие возможности:

− наличие в программе специализированных для описания услуг автосервиса справочников, таких как: «Модели автомобилей», «Автомобили», «Варианты комплектации автомобиля», «Виды ремонта», «Цеха», «Нормо-часы» и «Причины обращения» (рисунок 1),

− предварительная запись и планирование загрузки ресурсов автосервиса с помощью Обработки «Планировщик» (рисунок 2),

− отражение обслуживания или ремонта автомобиля с помощью документа «Заказ-наряд», который включает в себя прием автомобиля у клиента, поиск и подбор запчастей, передачу транспортного средства в ремонтную зону, назначение исполнителей для каждой работы, возврат отремонтированной техники клиенту (рисунок 3),

− отображение в программе этапов проведения работ, путем изменения состояний документа «Заказ-наряда»: «Заявка», «В работе», «Выполнен», «Закрыт».

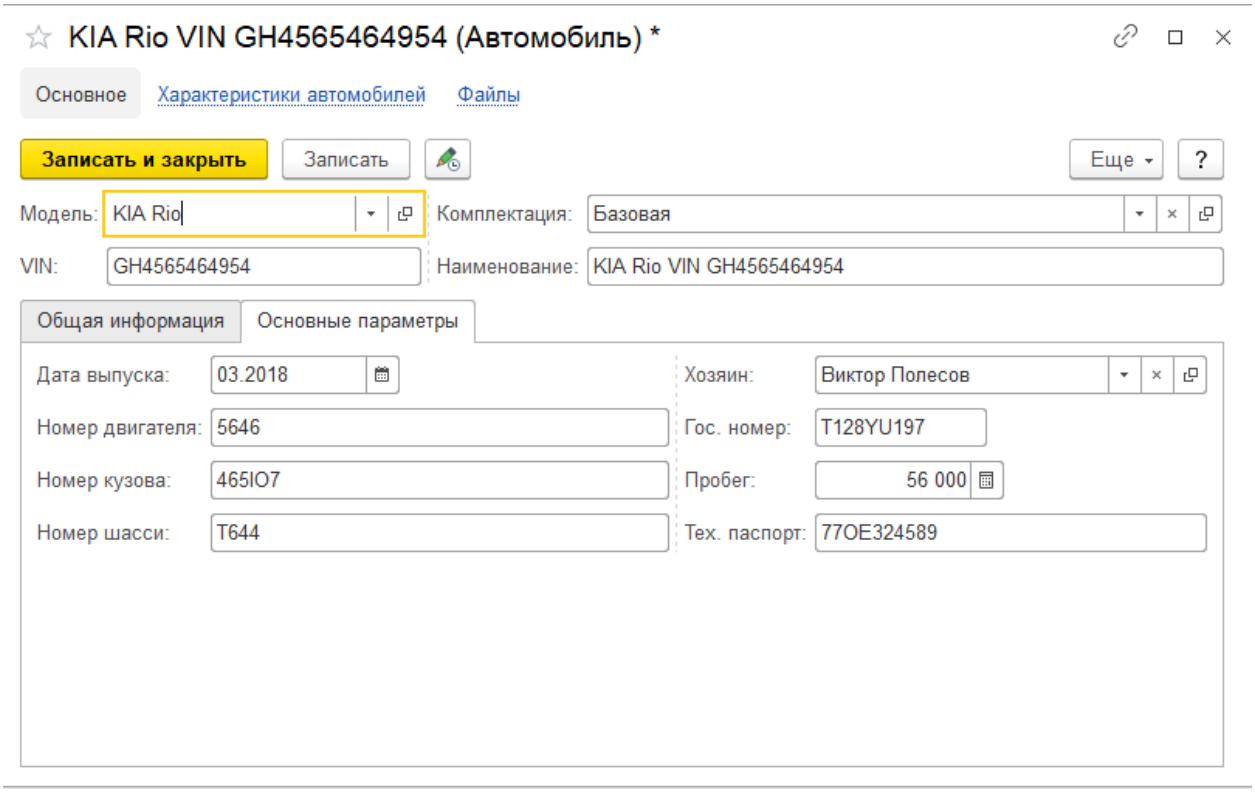

Рисунок 1 – Карточка автомобиля

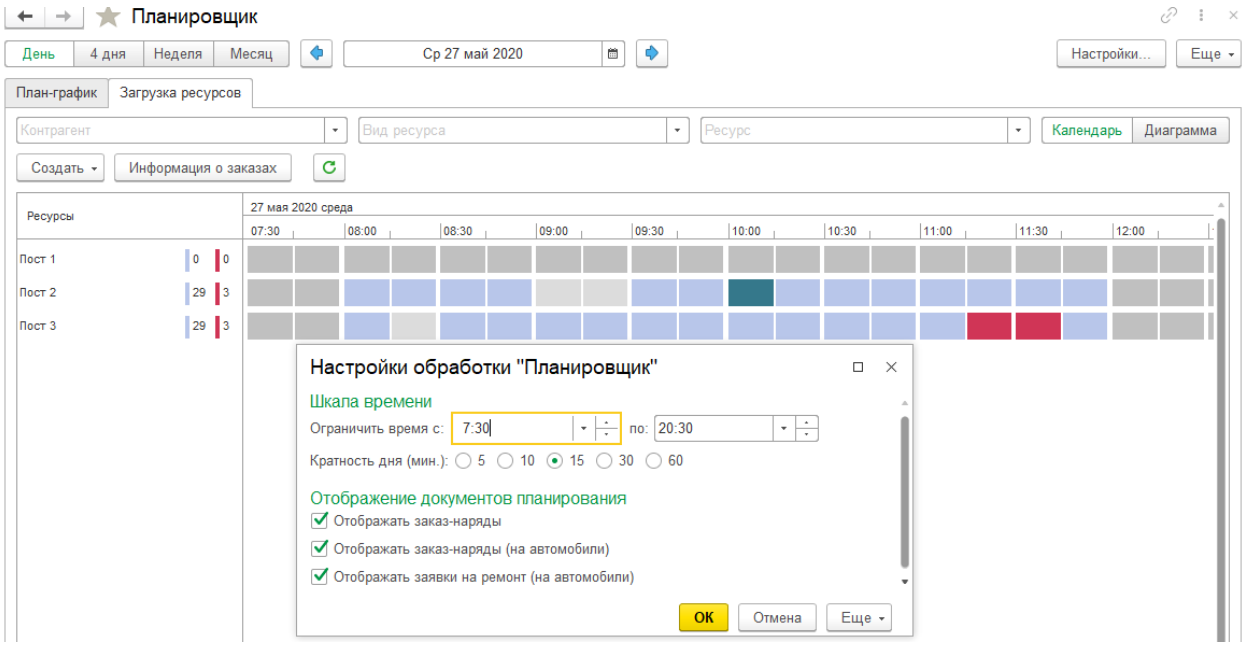

Рисунок 2 – Планировщик

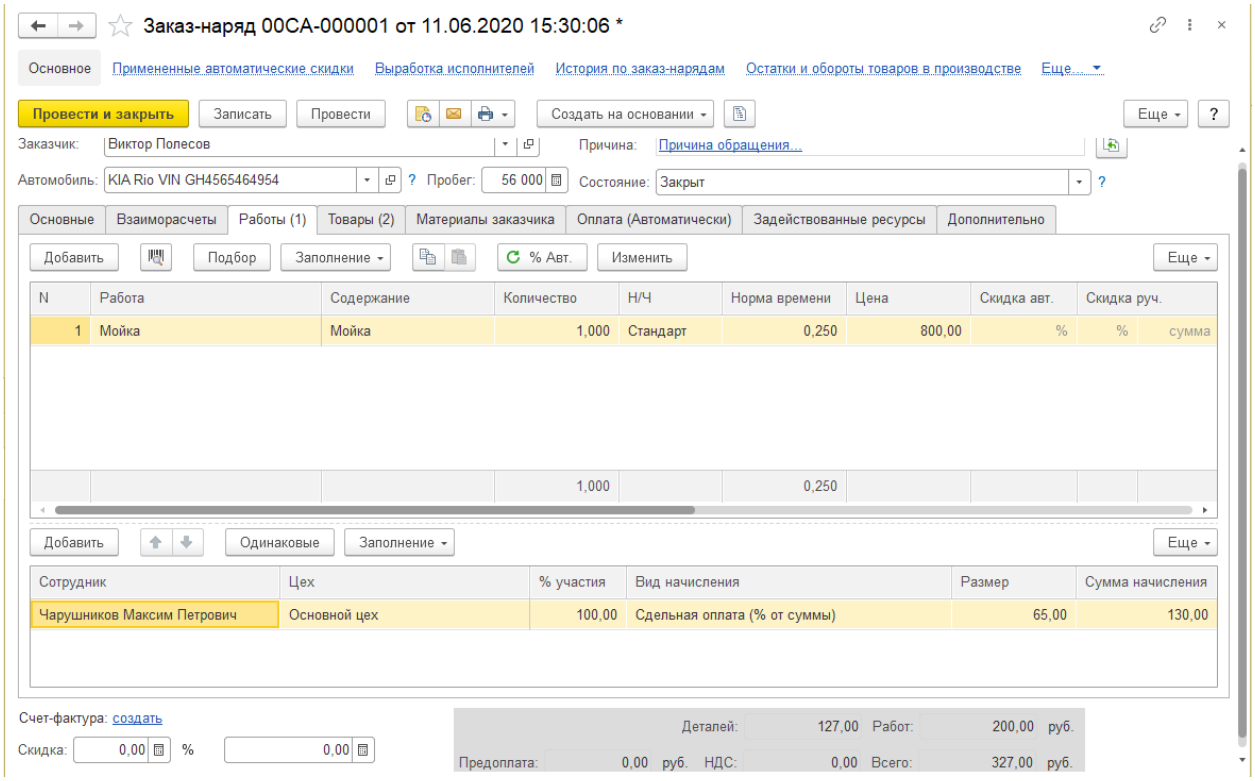

Рисунок 3 – Заказ-наряд

Программное решение: 1С БИТ.Автосервис. [1]

Программа 1С БИТ.Автосервис базируется на базе «1С:Управление торговлей ред. 11», что дает ей весь имеющийся функционал типового продукта 1С. Предоставляемые решением функции:

- планирование технического обслуживания и ремонта техники,
- планирование и анализ загрузки мощностей и механиков (рисунок 4),
- хранение истории обслуживания по каждому автомобилю,
- справочник «Автомобили» (рисунок 5),
- поиск информации по любым реквизитам клиента, автомобиля.

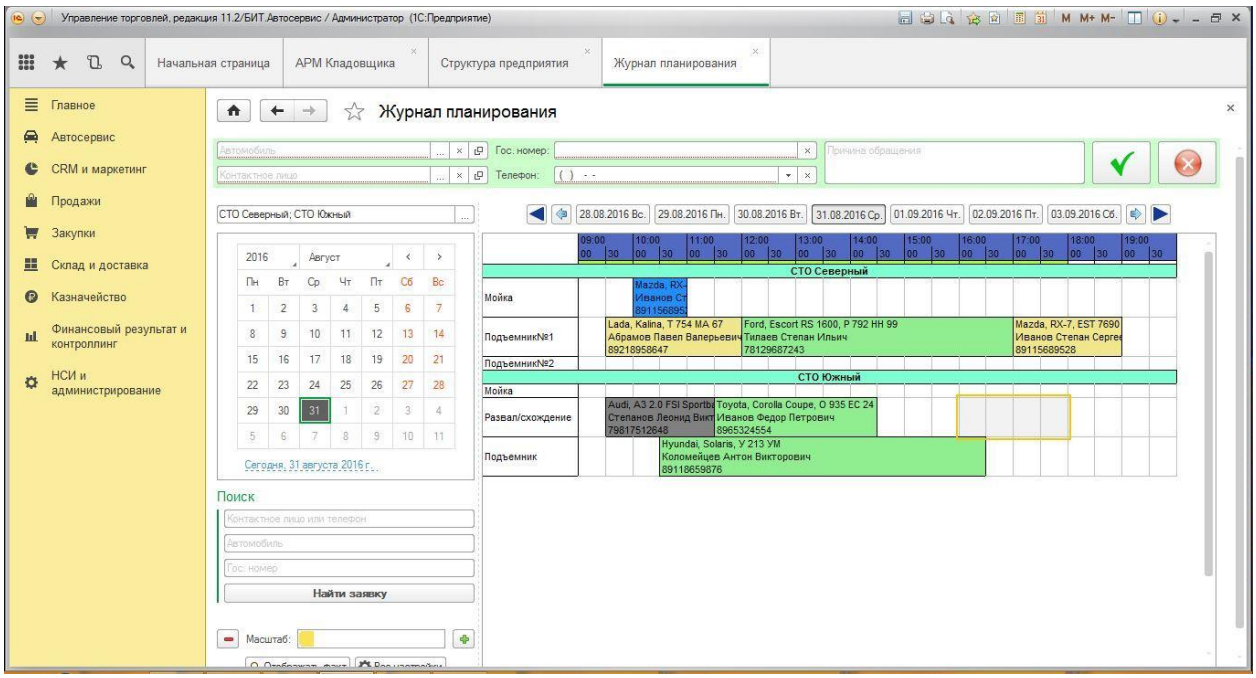

Рисунок 4 – Журнал планирования

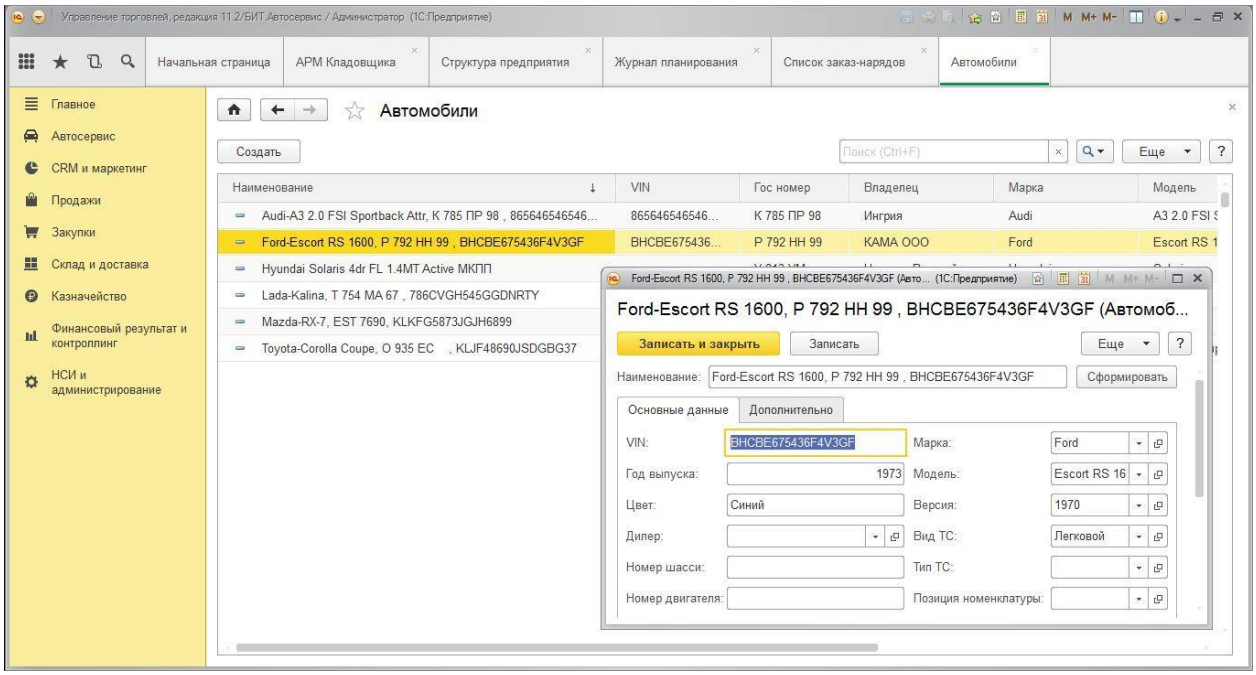

Рисунок 5 – Справочник «Автомобили»

Таблица 1 – Сравнительный анализ рассмотренных решений-аналогов

Требование / Аналог	1С: Предприятие 8. Автосервис	1С БИТ.Автосервис
Удобный пользовательский интерфейс	-	-
Дешевизна и простота внедрения решения	-	-
Доступность в программному решению через браузер с любого устройства	-	-
Список автомобильных марок, моделей и двигателей	+	+
Отображение этапов проведения работ	+	+
Хранение истории работ	+	+
Отображение статистики по СТО	+	+
Возможность мониторинга цехов и постов	+	+
Итог	5/8	5/8

В ходе анализа решений-аналогов в таблице 1, было определено, что ни одно из рассмотренных решений не подходит для использования на СТО малых размеров, так как они являются дорогостоящими и не соответствуют всем требованиям, предъявленным к сервису.

Исходя из этого было принято решение о разработке собственного сервиса для автоматизации процессов СТО.

1.3 Анализ сценариев использования программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО

Для описания сценариев использования сервиса решено использовать методологию графического моделирования UML [18], так как она позволяет подробно описать возможности сервиса.

Язык UML объектно-ориентирован, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования.

В качестве описания возможных вариантов сценария использования будет использована диаграмма вариантов использования (use case diagram), которая отражает отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Актер в UML представляет собой сущность, взаимодействующую с функциональностью сервиса, тем самым достигая решения задач и поставленных целей.

Прецедент (use case) [19] в UML описывает последовательность действий, выполняемых системой для достижения поставленных целей и получения результата актером.

Диаграмма вариантов использования автоматизированного сервиса для СТО представлена на рисунке 6.

Рисунок 6 – Диаграмма вариантов использования

В разрезе выбранной для описания вариантов использования сервиса методологии, на представленной диаграмме на рисунке 6 являются актерами:

- мастер-приемщик,
- мастер цеха,
- механик.

В свою очередь прецедентами являются:

- просмотр истории работ,
- просмотр списка автомобилей,
- регистрация нового автомобиля,

- прием автомобиля и запуск его в работу,
- отправка автомобиля на выдачу,
- выполнение работ,
- мониторинг цехов и постов,
- просмотр статистики,
- поиск автомобиля в системе,
- выдача автомобиля.

Мастер-приемщик использует сервис для регистрации нового автомобиля клиент и запуска его в работу, выбрав необходимые для выполнения работы, цех и пост, на котором будут выполняться работы и для окончания работы после выдачи автомобиля клиенту.

Мастер использует сервис для отслеживания загруженности подконтрольных ему цехов и постов на них, так же для просмотра и анализа статистики выполненных работ.

Механик использует систему для просмотра списка необходимых для выполнения работ на автомобиле и после завершения работ отправить автомобиль на выдачу клиенту.

Таким образом, в данном разделе был выполнен анализ и разработка требований, описаны и проанализированы задачи, так же выполнен поиск систем аналогов разрабатываемого сервиса, в результате которого, было принято решение разрабатывать собственное решение.

2 Проектирование архитектуры программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО

2.1 Проектирование компонентов сервиса

Для удовлетворения установленных требований, разрабатываемый сервис должен представлять собой клиент-серверное [3] приложение с доступом к нему пользователей посредством браузера с любой платформы: ПК, мобильные устройства.

Клиент и сервер должны быть отдельными, независимыми компонентами, которые взаимодействуют между собой через REST API [15].

REST API представляет собой архитектурный стиль взаимодействия компонентов [2] распределённого приложения в сети, при котором важными свойствами являются:

- производительность,
- масштабируемость,
- прозрачность связей между компонентами сервиса,
- выраженная устойчивость к отказам на уровне сервиса при наличии отказов отдельных компонентов.

Клиент в данном случае состоит из одного компонента и является web-сайтом, доступным с любого браузера по адресу, на котором будет размещен после разработки. Сервер в свою очередь состоит из двух компонент: сервера обрабатывающего запросы клиента и базы данных. Схема взаимодействия клиента и сервера отображена на рисунке 7. Такая архитектура сервиса способствует простому масштабированию системы, высокой отказоустойчивости и высокой производительности, что в свою очередь очень важно для СТО.

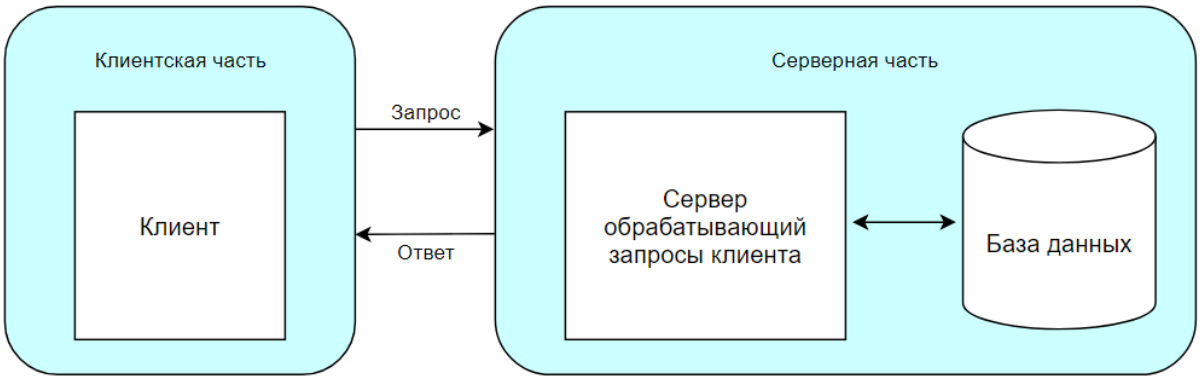

Рисунок 7 – Диаграмма взаимодействия клиента и сервера

2.2 Проектирование логической модели базы данных программного обеспечения для мониторинга и управления цехами ТО и ТР на СТО

Исходя из предъявляемых требований к сервису необходимо спроектировать логическую модель базы данных, которая описывает понятия предметной области, связь между ними и вводит ограничения, наложенные предметной областью.

Для проектирования логической модели базы данных будет использована методология IDEF1X [12], в концепции которой используется диаграмма сущность-связь.

Логическая модель базы данных сервиса представлена на рисунке 8.

В соответствии с построенной логической моделью, были выделены следующие сущности:

- марка,
- модель,
- двигатель,
- модель - двигатель,
- автомобиль,
- владелец автомобиля,

- заказ-наряд,
- работа,
- заказ-наряд - работа,
- цех,
- пост,
- очередь работ.

Сущность «Марка» была добавлена для хранения информации о марках автомобилей, имеющихся в системе.

Сущность «Модель» содержит информацию о моделях, имеющихся в системе для каждой конкретной марки автомобилей.

Сущность «Двигатель» содержит информацию о названии двигателей всех моделей и марок, имеющихся в системе.

«Модель - Двигатель» – ассоциирующая сущность, ограничивающая список двигателей для каждой конкретной модели и марки автомобиля.

Сущность «Автомобиль» содержит VIN-код, марку, модель, название двигателя, ФИО владельца и пробег всех автомобилях, зарегистрированных в системе.

Сущность «Владелец автомобиля» содержит ФИО всех владельцах автомобилей, зарегистрированных в системе.

Сущность «Заказ-наряд» содержит информацию всех заказ-нарядах, зарегистрированных в системе, а именно: информацию об автомобиле, пробег автомобиля на момент запуска в работу и дату открытия конкретного заказ-наряда.

Сущность «Работа» содержит информацию о работах, которые могут быть выполнены на автомобиле в цехах ТО и ТР на конкретных постах.

«Заказ-наряд - Работа» – это ассоциирующая сущность, добавленная для описания всех работ на автомобиле, выбранных в рамках конкретного заказ-наряда.

Сущность «Цех» содержит информацию о всех цехах, организованных на СТО и список постов в каждом цехе.

Сущность «Пост» содержит информацию о всех постах, организованных в рамках цехов на СТО.

«Очередь работ» - это служебная сущность, хранящая в себе список всех когда-либо запущенных в работу заказ-нарядов, их актуальный статус, а также пост, на котором проводились работы и даты начала и завершения работ по конкретному заказ-наряду.

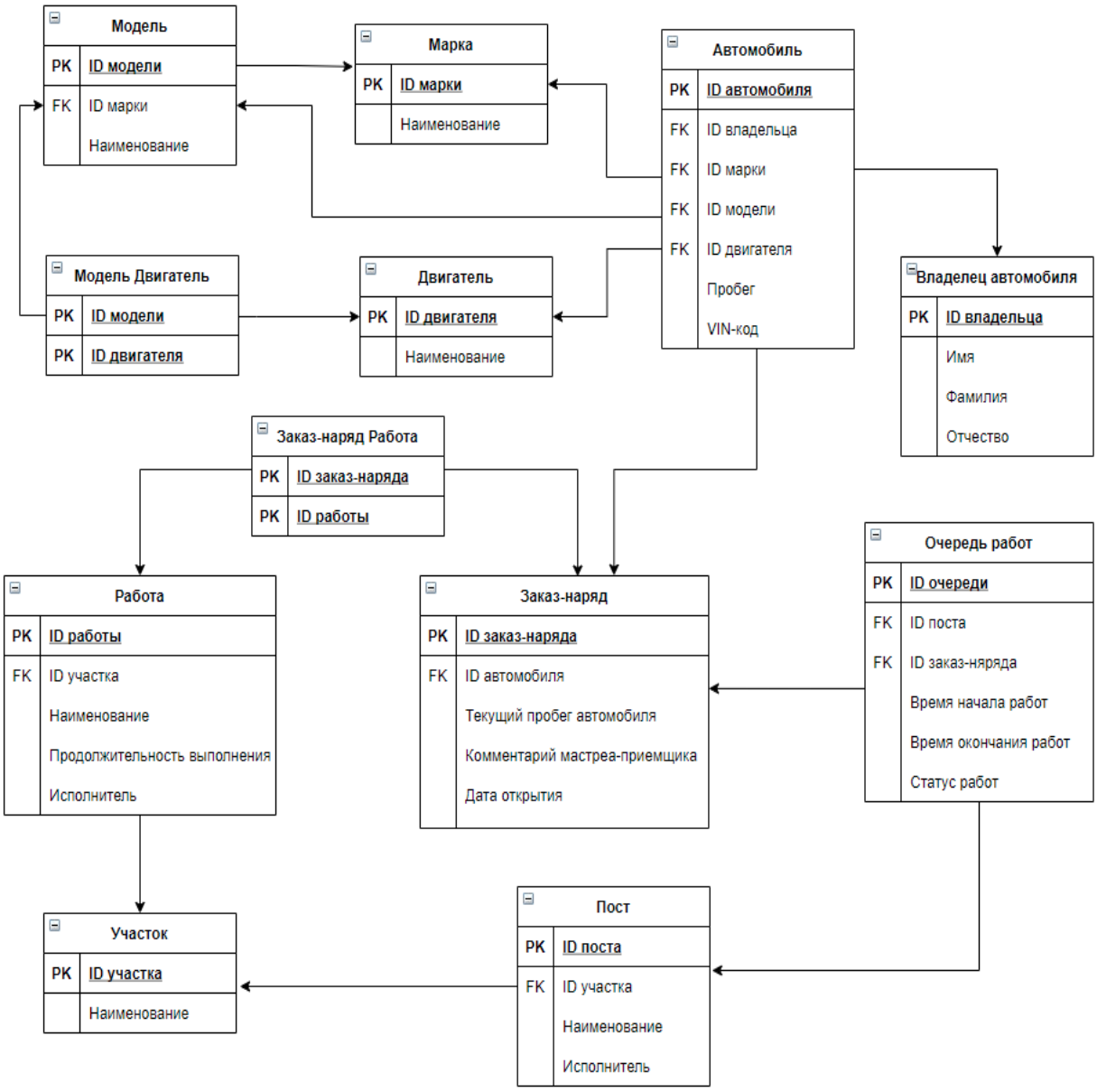

Рисунок 8 – Логическая модель базы данных

2.3 Выбор и обоснование программных инструментов для разработки сервиса

В качестве операционной системы была выбрана Windows 10 [20] от компании Microsoft, преимуществами которой являются высокая надежность, стабильность и производительность.

Для реализации клиентской части сервиса была выбрана программная платформа Angular 11 [9], последней на текущий момент версии. Программная платформа написана на языке программирования TypeScript [17], активно разрабатывается и поддерживается компанией Google. Является open source проектом и выпускается по лицензии MIT, что позволяет свободно использовать ее в коммерческих целях.

Для реализации серверной части сервиса была выбрана программная платформа NestJS 7 [11]. Является open source проектом и выпускается по лицензии MIT, что позволяет свободно использовать ее в коммерческих целях, активно поддерживается и разрабатывается сообществом. Программная платформа написана на языке программирования TypeScript.

Языком программирования выбран TypeScript, разрабатываемый компаний Microsoft. TypeScript является языком программирования, расширяющим возможности языка программирования JavaScript, добавляя явное статическое назначение типов данных и поддержку объектно-ориентированного программирования, что в свою очередь ускоряет и упрощает разработку.

В качестве системы управления базой данных (СУБД) выбрана PostgreSQL [14]. Это свободная объектно-реляционная СУБД, реализует максимально стандарт SQL, что позволяет при необходимости заменить ее другой СУБД с минимальными затратами.

Средой разработки программного обеспечения была выбрана IntelliJ IDEA [7]. Это интегрированная среда разработки, подходящая для разработки на языке программирования TypeScript, разработанная компанией JetBrains.

2.4 Проектирование физической модели базы данных

Физическая модель базы данных описывает реализацию сущностей на выбранной для разработки СУБД.

Физическая модель базы данных представлена на рисунке 9.

В результате выполнения раздела было выполнено проектирование компонентов сервиса, логической и физической моделей базы данных и выбраны программные инструменты для разработки.

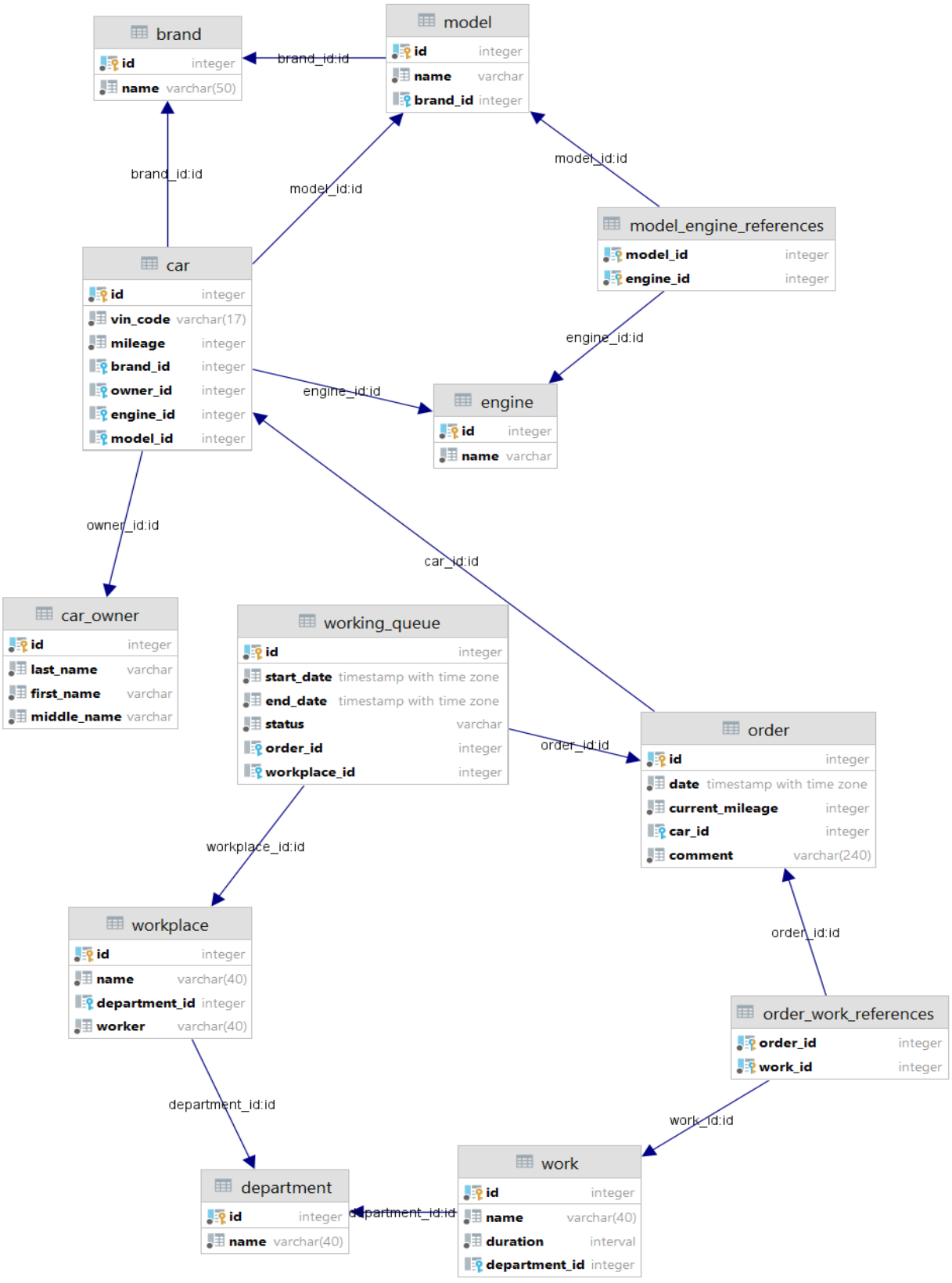

Рисунок 9 – Физическая модель базы данных

3 Разработка и реализация программного обеспечения

3.1 Описание пользовательского интерфейса сервиса

Под пользовательским интерфейсом понимается совокупность средств для ввода данных, отображения информации и элементов управления, с помощью которых пользователь сервиса будет с ним взаимодействовать.

Сервис имеет дружелюбный интерфейс с общепринятыми и понятными элементами управления для удобства использования.

Пользовательский интерфейс спроектирован таким образом, чтобы выполнение действий не было осложнено перегруженностью элементами управления.

Основные страницы сервиса: страница мониторинга, страница списка автомобилей и страница со статистикой. Так же на странице списка автомобилей имеются всплывающие диалоговые окна с формами для заполнения и отображения информации.

Чтобы использовать сервис, необходимо в браузере перейти по URL-адресу, на котором он размещен (в случае локального компьютера – это http://localhost:4200).

Программный код основных модулей сервиса представлен в Приложении А.

3.2 Описание страницы мониторинга загруженности цехов и постов ТР и ТР на СТО

После открытия сервиса, пользователю отображается страница с мониторингом цехов и постов ТО и ТР на СТО. Страница мониторинга разделена на два функциональных блока: блок со списком автомобилей,

готовых к выдаче и блок со списком цехов. Общий вид страницы мониторинга представлен на рисунке 10.

В блоке со списком автомобилей на выдачу клиентам отображаются автомобили, над которыми были выполнены все работы и которые ожидают выдачи клиенту. На каждом элементе с автомобилем есть кнопка «Выдано», которую должен нажать мастер-приемщик после выдачи автомобиля клиенту, после чего машина пропадет из списка. Так же каждый элемент имеет в себе марку, модель VIN-код и ФИО владельца.

Вид блока со списком автомобилей на выдачу клиентам представлен на рисунке 11.

В блоке со списком цехов, под каждым цехом отображаются все посты, имеющиеся в этом цехе. Под каждым элементом с постом на странице имеется автомобиль, на котором выполняются работы, список самих работ и время начала и примерное время окончания выполнения работ исходя из суммы их длительности выполнения.

На каждом элементе с автомобилем имеется кнопка «Завершить», которую должен нажать механик после окончания работ, после чего автомобиль отправляется перемещается в блок со списком автомобилей на выдачу клиенту.

Вид блока со списком цехов представлен на рисунке 12.

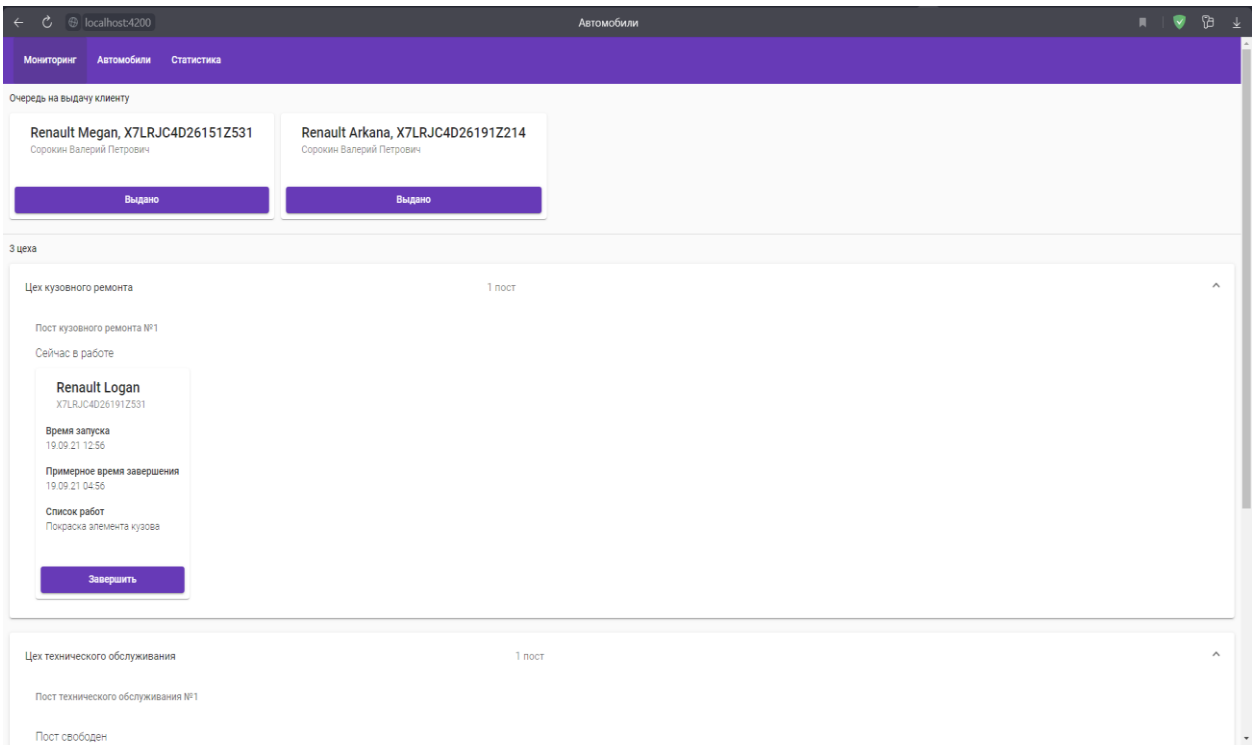

Рисунок 10 – Общий вид страницы мониторинга

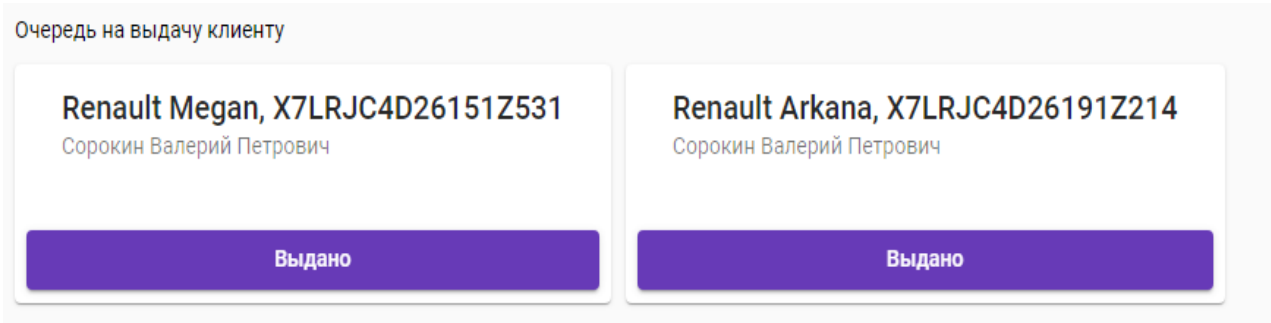

Рисунок 11 – Блок со списком автомобилей на выдачу клиентам

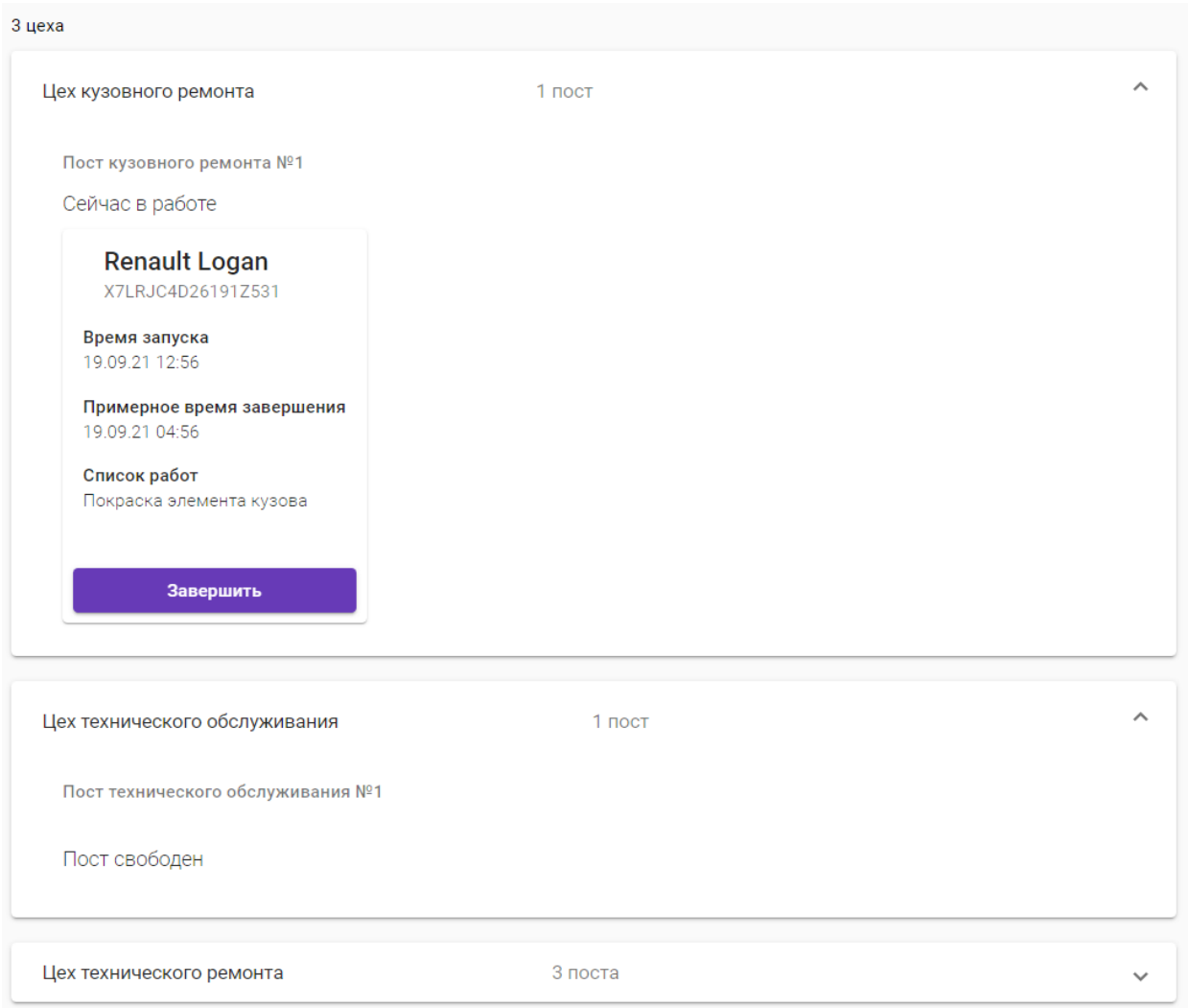

Рисунок 12 – Блок со списком цехов и постов

3.3 Описание страницы списка автомобилей

Используя панель навигации вверху страницы, можно перейти на страницу списка автомобилей. Страница списка автомобилей состоит из:

- строки поиска автомобиля,
- списка автомобилей,
- кнопки регистрации автомобиля в системе,
- кнопки просмотра истории работ на автомобиле,
- кнопки запуска зарегистрированного в системе автомобиля в работу.

Общий вид страницы со списком автомобилей представлен на рисунке 13.

С помощью строки поиска осуществляется поиск зарегистрированного автомобиля в системе по марке, модели, ФИО владельца или VIN-коду. Поиск производится по мере ввода пользователем данных в строку поиска.

Для регистрации нового автомобиля в системе необходимо нажать кнопку «Зарегистрировать автомобиль». В открывшемся после нажатия на кнопку диалоговом окне следует ввести ФИО владельца, марку, модель, VIN-код и пробег автомобиля на момент регистрации.

При необходимости запуска в работу автомобиля сразу после регистрации следует выбрать пункт «В работу». Диалоговое окно для регистрации автомобиля представлено на рисунке 14.

Список автомобилей представляет собой таблицу, в колонках которой отображается вся информация об автомобиле: марка, модель, ФИО владельца и VIN-код. У каждого автомобиля в таблице имеется две кнопки: «История» и «В работу».

С помощью кнопки «История» открывается диалоговое окно со списком заказ-нарядов, которые были зарегистрированы на конкретном автомобиле. Диалоговое окно с историей на автомобиле представлено на рисунке 15.

Кнопка «В работу» открывает диалоговое окно, с помощью которого мастер-приемщик может запустить автомобиль в работу, предварительно выбрав цех, в котором будет расположен необходимый пост, конкретный пост, на котором будут производиться работы, необходимые для выполнения работы, текущий пробег автомобиля и заполнив комментарий при его наличии.

Диалоговое окно для запуска автомобиля в работу представлено на рисунке 16.

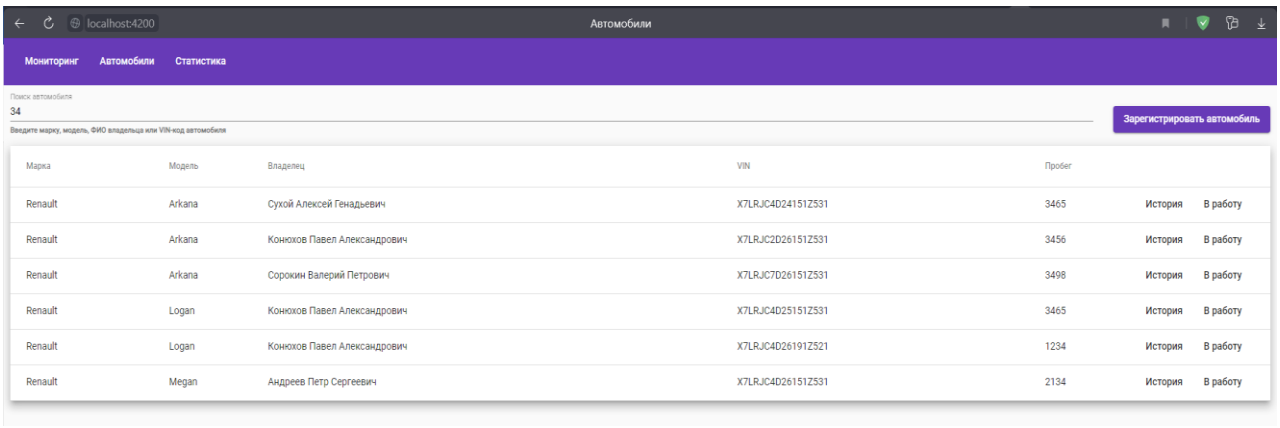

Рисунок 13 – Общий вид страницы списка автомобилей

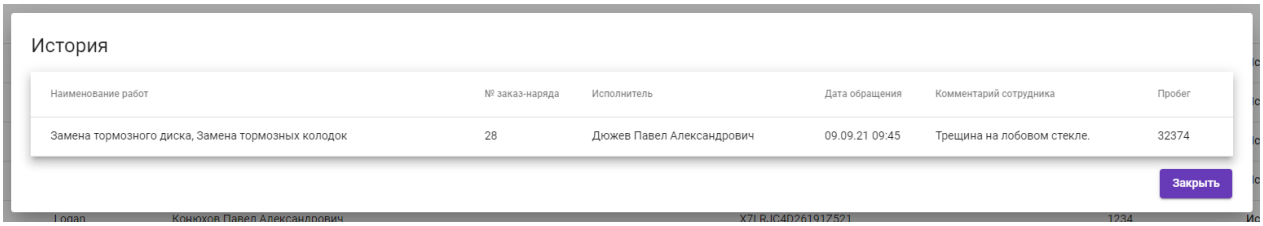

Рисунок 14 – Диалоговое окно с историей работ на автомобиле

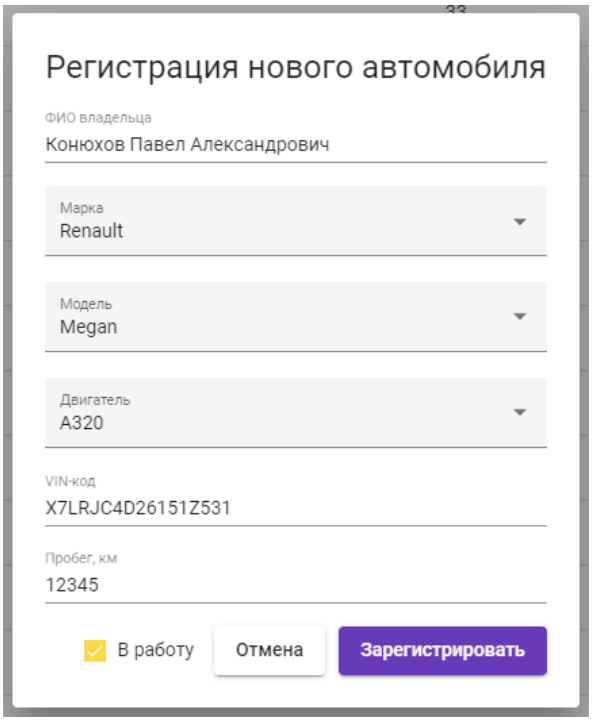

Рисунок 15 – Диалоговое окно с формой регистрации автомобиля

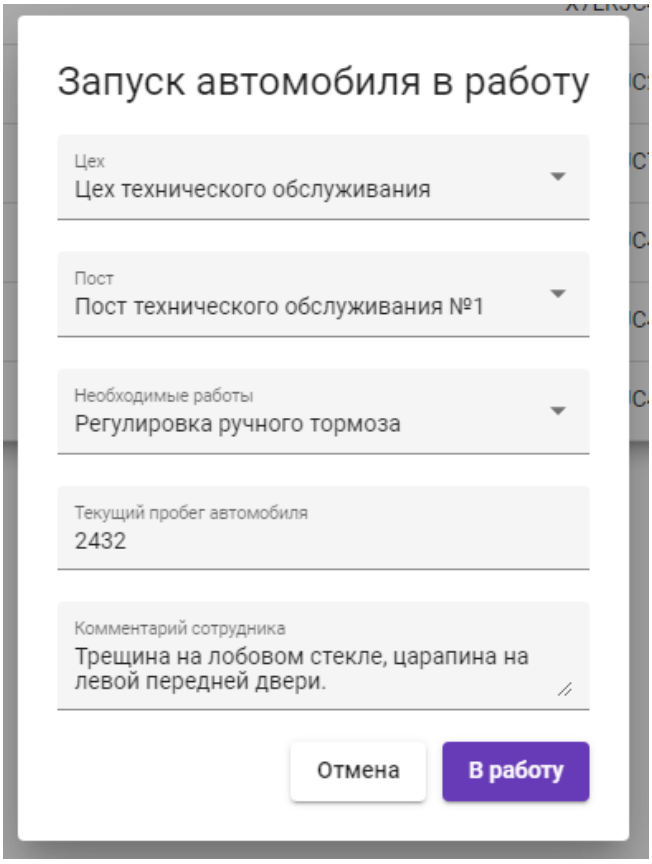

Рисунок 16 – Диалоговое окно с формой запуска автомобиля в работу

3.4 Описание страницы со статистикой выполненных работ

Для просмотра статистики по выполненным работам постах на СТО необходимо в меню навигации вверху страницы выбрать пункт «Статистика».

Статистика собирается и отображается сервисом за определенный промежуток времени, который указывается нажатием на панель с кнопками «День», «Неделя», «Месяц», «Год» и соответствует названию кнопок.

В текущей реализации сервис отображает статистику по закрытым заказ-нарядам, что соответствует количеству выполненных работ.

Общий вид страницы со статистикой представлен на рисунке 17.

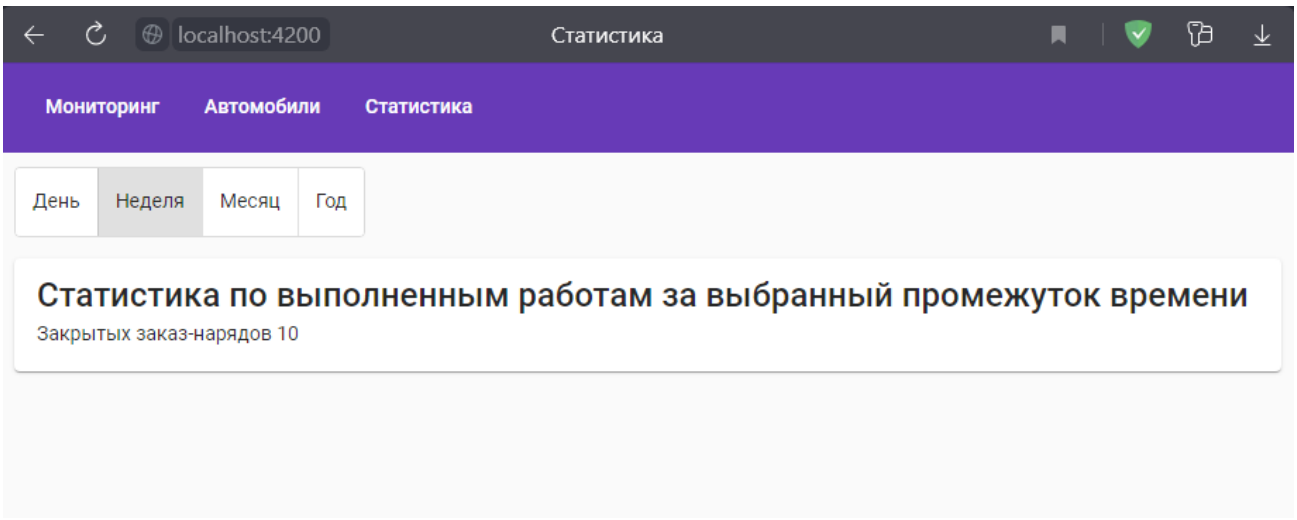

Рисунок 17 – Общий вид страницы со статистикой

3.5 Тестирование сервиса

В рамках тестирования будут проверены следующие функции сервиса:

1. Регистрация нового автомобиля в системе;
2. Отображение списка автомобилей;
3. Отображение истории работ на автомобилях;
4. Отображение списка цехов и постов на СТО для мониторинга;
5. Запуск автомобиля в работу;
6. Отображение списка автомобилей для выдачи клиенту;
7. Завершение работ на автомобиле;
8. Выдача автомобиля клиенту;
9. Отображение статистики выполненных работ за день, неделю, месяц, год.

Все запланированные функции сервиса были проверены и работают согласно требованиям.

Итогом выполнения раздела является разработанный пользовательский интерфейс сервиса и его тестирование.

4 Дальнейшее возможное масштабирование и развитие

Дальнейшим возможным масштабированием для сервиса будет расширение целевых СТО до средних, больших и крупных. С функциональной точки зрения требования средних, больших и крупных СТО относительно малых повышаются, что было учтено при проектировании и заложено в архитектуру программного обеспечения.

Возможные расширения существующей и будущей функциональности программного обеспечения:

- календарь планирования,
- расширенная статистика,
- генерация заказ-нарядов для печати,
- акт осмотра автомобиля,
- расширение хранимых параметров автомобиля,
- каталог запчастей,
- каталог цен на выполняемые работы,
- система управления складом,
- SMS и email рассылки,
- система учета зарплат работников,
- обеспечение соблюдения законодательства по части сбора и хранения информации о клиентах.

Добавление календаря планирования дает возможность записывать клиентов СТО на удобное им время и позволит лучше мастерам контролировать загруженность постов.

Внедрение расширенной статистики позволит отслеживать по следующим параметрам количество:

- выполненных работ конкретным работником,
- новых зарегистрированных автомобилей в системе,
- выполненных работ из отдельных категорий,

− проданных запчастей и деталей.

Генерация заказ-нарядов для печати позволит выдавать клиенту бумажную версию заказ-наряда.

Добавление новых хранимых параметров автомобиля таких как: государственный номер автомобиля, характеристики используемого моторного и трансмиссионного масел способствуют повышению эффективности работы персонала и улучшению качества сервиса СТО.

Каталог автомобильных запчастей значительно повысит скорость и эффективность поиска необходимых для работы запчастей и деталей на конкретный автомобиль.

Акт осмотра автомобиля необходим для фиксирования всех замечаний на автомобиле, обнаруженных в момент приемки автомобиля. Позволяет мастеру-приемщику отметить в нем найденные дефекты, которые были нанесены автомобилю до приезда на СТО, что в свою очередь освободит от необоснованных обвинений со стороны клиента в случае обнаружения им дефекта после окончания работ.

В акт осмотра автомобиля возможно вносить неисправности, обнаруженные мастером-приемщиком после первичного осмотра автомобиля.

Система учета зарплат работников позволит автоматически рассчитывать заработную плату работников исходя из количества и сложности выполненных работ.

Каталог цен на выполняемые работы предоставит возможность зафиксировать стоимость конкретных работ в системе, что в свою очередь облегчит расчет заработной платы работникам, а также консультирование клиентов по стоимости работ во время составления заказ-наряда.

Система управления складом позволит отслеживать актуальное наличие на складе запчастей, деталей, расходных материалов и прочей продукции необходимой для работы СТО.

SMS и email рассылка необходимы для оповещения клиентов об окончании работ и необходимости забрать автомобиль с СТО. Рассылки будут

полезны в рекламных целях, например, для рассылки SMS и email клиентам, предлагая скидки на услуги и прочие акции.

Обеспечение соблюдения законодательства [4] по части сбора и хранения информации о клиентах позволит хранить персональные данные всех клиентов в соответствии с федеральным законодательством страны, в которой используется сервис.

Таким образом, сервис имеет обширный потенциал для масштабирования и развития.

5 Разработка участка диагностики

5.1 Назначение участка

Участок диагностики предназначается для оценки технических показателей и локализации неисправностей.

Разрабатываемый участок диагностики для малой СТО может быть автоматизирован с помощью разработанного программного обеспечения.

5.2 Планировка участка

Разрабатываемый участок диагностики состоит из следующих производственных постов:

1. диагностики тормозной системы,
2. определения состояния амортизаторов и комплексной оценки технического состояния ходовой части автомобиля,
3. проверки состояния элементов подвески,
4. работ по системам освещения и световой сигнализации,
5. определения токсичности отработавших газов.

План участка диагностики представлен на рисунке 18.

5.3 Оборудование участка

Наименование оборудования и его количество, расположенного на спроектированном участке диагностики, представлено в таблице 2.

В результате выполнения раздела был проектирован участок диагностики малой СТО, процессы которого могут быть автоматизированы с помощью разработанного сервиса.

Таблица 2 – Оборудование участка

№	Наименование оборудования	Количество, шт.
1	Стенд для проверки тормозных систем транспортных средств	1
2	Стенд для диагностирования состояния передней подвески автомобиля по боковому уводу в сторону от прямолинейного движения	1
3	Стенд проверки амортизаторов	1
4	Стенд контроля состояния передней подвески и рулевого управления	1
5	Система управления, сбора и обработки данных	1
6	Шкаф инструментальный	2
7	Прибор контроля света фар	1
8	Верстак слесарный	1
9	Газоанализатор четырехпараметровый	1
10	Подъемник канавный передвижной	1

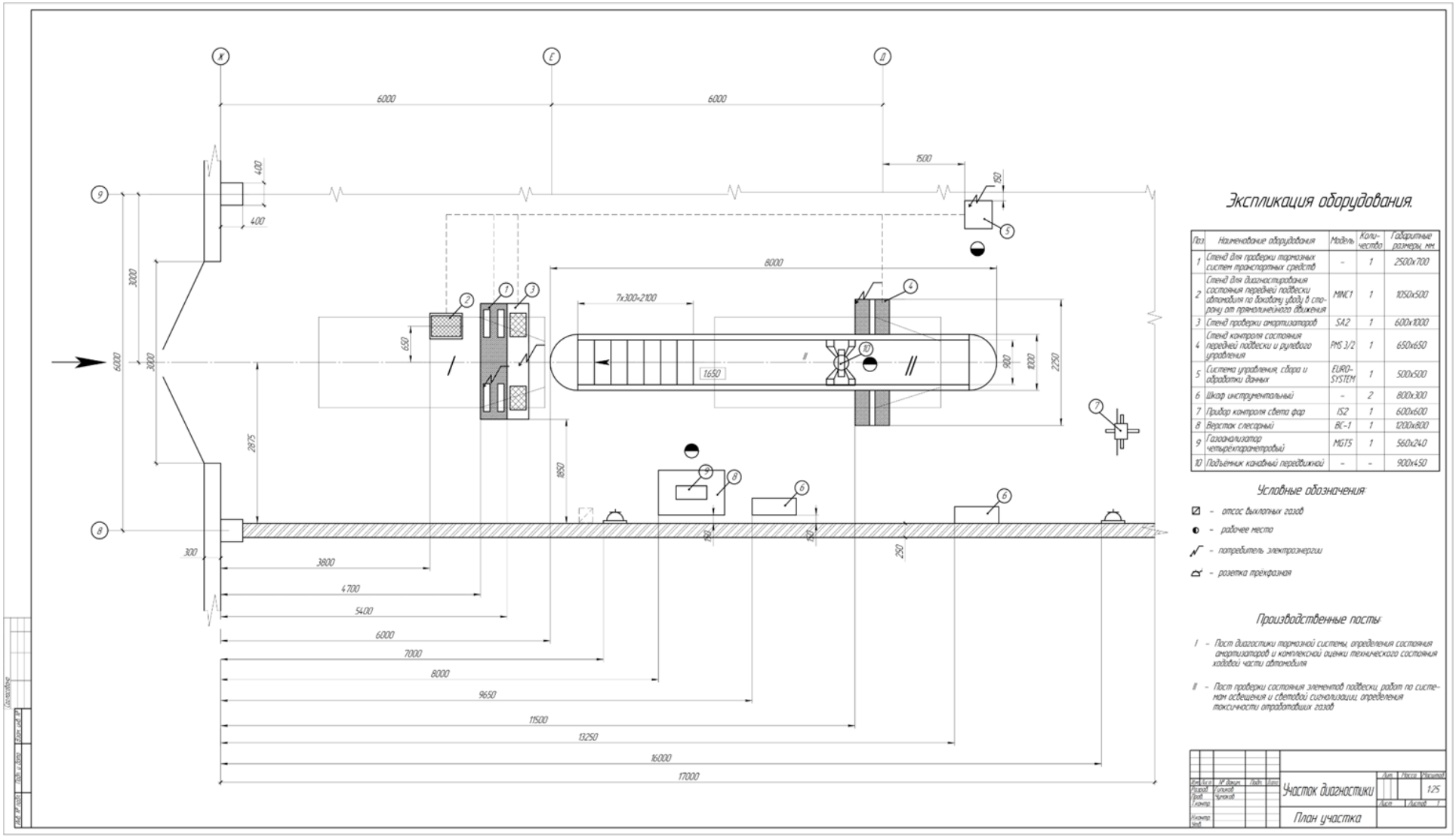

Рисунок 18 – План участка диагностики

Заключение

В процессе написания выпускной квалификационной работы было разработано программное обеспечения для управления и мониторинга цехов и постов ТО и ТР на СТО. Программное обеспечение отвечает всем требованиям задания и реализует следующие запланированные функции:

- мониторинг загруженности цехов и постов ТО и ТР на СТО,
- сбор и отображение статистики по выполненным работам за день, неделю, месяц и год,
- автоматическое управление потоком автомобилей на постах ТО и ТР на СТО,
- хранение и отображение истории работ на автомобилях.

Выполнен анализ и разработка требований к программному обеспечению.

Проведен анализ систем-аналогов разрабатываемого программного обеспечения для выявления необходимости создания нового продукта.

Спроектированы компоненты программного обеспечения и его архитектура, логическая и физическая модель базы данных.

Проведено тестирование программного обеспечения для подтверждения работоспособности решения.

Разработано дальнейшее возможное масштабирование и развитие программного обеспечения.

Разработан участок диагностики малой СТО в цехе ТО и ТР для автоматизации процесса с помощью разработанного программного обеспечения.

Таким образом, следующие цели выпускной квалификационной работы полностью выполнены:

- анализ и разработка требований к программному обеспечению,
- выбор инструментов разработки программного обеспечения,
- проектирование архитектуры программного обеспечения,
- проектирование логической модели базы данных,

- проектирование физической модели базы данных,
- реализация спроектированного программного обеспечения,
- тестирование программного обеспечения,
- разработка дальнейшего возможного развития,
- разработка участка для использования программного обеспечения.

Программное обеспечения было спроектировано, разработано, протестировано и удовлетворяет поставленным требованиям выпускной квалификационной работы.

Список используемых источников

1. БИТ. Автосервис | Программа для автоматизации автосервиса [Электронный ресурс], — URL: https://samara.1cbit.ru/1csoft/bit-avtoservis (дата обращения: 15.06.2021).

2. Диаграмма компонентов [Электронный ресурс], — URL: https://ru.wikipedia.org/wiki/Диаграмма_компонентов (дата обращения: 12.06.2021).

3. Клиент - сервер [Электронный ресурс], — URL: https://ru.wikipedia.org/wiki/Клиент_—_сервер (дата обращения: 15.08.2021).

4. Федеральный закон о персональных данных [Электронный ресурс], — URL: http://www.consultant.ru/document/cons_doc_LAW_61801/ (дата обращения: 10.09.2021).

5. Фирма «1С» [Электронный ресурс], — URL: https://1c.ru (дата обращения: 12.06.2021).

6. Фирма 1С: Предприятие 8. Автосервис [Электронный ресурс], — URL: https://rarus.ru/1c-auto/1c8-avtoservis (дата обращения: 15.06.2021).

7. Функциональность IntelliJ IDEA [Электронный ресурс], — URL: https://www.jetbrains.com/ru-ru/idea/features (дата обращения: 23.06.2021).

8. Что такое 1С. О сложной системе простыми словами [Электронный ресурс], — URL: https://habr.com/ru/company/trinion/blog/250893 (дата обращения: 16.06.2021).

9. Angular – Introduction to the Angular Docs [Электронный ресурс], — URL: https://angular.io/docs (дата обращения: 27.06.2021).

10. Components | Angular Material [Электронный ресурс], — URL: https://material.angular.io/components/categories (дата обращения: 19.08.2021).

11. Documentation | NestJS – A progressive Node.js framework [Электронный ресурс], — URL: https://docs.nestjs.com (дата обращения: 19.08.2021).

12. IDEF1X [Электронный ресурс], — URL: https://ru.wikipedia.org/wiki/IDEF1X (дата обращения: 12.06.2021).

13. pgAdmin – PostgreSQL Tools [Электронный ресурс], — URL: https://www.pgadmin.org (дата обращения: 21.06.2021).

14. PostgreSQL: Documentation: 12: PostgreSQL 12.8 [Электронный ресурс], — URL: https://www.postgresql.org/docs/12/index.html (дата обращения: 20.08.2021).

15. REST [Электронный ресурс], — URL: https://en.wikipedia.org/wiki/REST (дата обращения: 10.09.2021).

16. typeorm: ORM for TypeScript [Электронный ресурс], — URL: https://github.com/typeorm/typeorm (дата обращения: 25.06.2021).

17. TypeScript [Электронный ресурс], — URL: https://github.com/microsoft/TypeScript (дата обращения: 20.06.2021).

18. UML [Электронный ресурс], — URL: https://en.wikipedia.org/wiki/UML (дата обращения: 12.06.2021).

19. User case [Электронный ресурс], — URL: https://en.wikipedia.org/wiki/Use_case (дата обращения: 10.09.2021).

20. Windows 10 [Электронный ресурс], — URL: https://en.wikipedia.org/wiki/Windows_10 (дата обращения: 10.08.2021).

Приложение А

Программный код основных модулей сервиса

entities.ts

```
import {Column, Entity, JoinTable, ManyToMany, ManyToOne, OneToMany,
PrimaryGeneratedColumn} from "typeorm";
import {IPostgresInterval} from "postgres-interval";

@Entity()
export class Brand {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column({type: "varchar", length: 50})
    public name: string;

    @OneToMany(type => Car, car => car.brand)
    public cars: Car[];

    @OneToMany(type => Model, model => model.brand)
    public models: Model[];

}


@Entity()
export class CarOwner {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column({name: "last_name"})
    public lastName: string;

    @Column({name: "first_name"})
    public firstName: string;

    @Column({name: "middle_name"})
    public middleName: string;

    @OneToMany(type => Car, car => car.owner)
    public cars: Car[];

}

@Entity()
export class Model {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column()
    public name: string;

    @ManyToMany(type => Engine, engine => engine.models)
    @JoinTable({name: "model_engine_references"})
    public engines: Engine[];

    @ManyToOne(type => Brand)
    public brand: Brand;
```

```
}

@Entity()
export class Engine {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column()
    public name: string;

    @ManyToMany(type => Model, model => model.engines)
    public models: Model[];

}


@Entity()
export class Car {

    @PrimaryGeneratedColumn("increment")
    public id: number

    @Column({type: "varchar", length: 17, name: "vin_code"})
    public vinCode: string;

    @Column("int")
    public mileage: number;

    @ManyToOne(type => Brand, brand => brand.models)
    public brand: Brand;

    @ManyToOne(type => CarOwner, owner => owner.cars)
    public owner: CarOwner;

    @ManyToOne(type => Engine)
    public engine: Engine;

    @ManyToOne(type => Model)
    public model: Model;

    @OneToMany(type => Order, order => order.car)
    public orders: Order[];


}

@Entity()
export class Department {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column({type: "varchar", length: 40})
    public name: string;

    @OneToMany(type => Workplace, workplace => workplace.department)
    public workplaces: Workplace[];

    @OneToMany(type => Work, work => work.department)
    public works: Work[];
```

```
}

@Entity()
export class Workplace {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column({type: "varchar", length: 40})
    public name: string;

    @Column({type: "varchar", length: 40})
    public worker: string;

    @ManyToOne(type => Department, department => department.workplaces)
    public department: Department;

    @OneToMany(type => WorkingQueue, workingQueue => workingQueue.workplace)
    public workingQueues: WorkingQueue[];

}

@Entity()
export class Order {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column({type: "timestamptz"})
    public date: Date;

    @Column({type: "varchar", length: 240})
    public comment: string;

    @Column({name: "current_mileage", type: "int"})
    public currentMileage: number;

    @Column({type: "varchar", length: 40})
    public worker: string;

    @ManyToOne(type => Car, car => car.orders)
    public car: Car;

    @ManyToMany(type => Work, work => work.orders)
    @JoinTable({name: "order_work_references"})
    public works: Work[];

    @OneToMany(type => WorkingQueue, workingQueue => workingQueue.workplace)
    public workingQueues: WorkingQueue[];

}

@Entity()
export class WorkingQueue {

    @PrimaryGeneratedColumn("increment")
    public id: string;

    @Column({type: "timestamptz"})
    public startDate: Date;
```

```
    @Column({type: "timestamptz"})
    public endDate: Date;

    @Column({type: "varchar"})
    public status: string;

    @ManyToOne(type => Workplace, workplace => workplace.workingQueues)
    public workplace: Workplace;

    @ManyToOne(type => Order, order => order.workingQueues)
    public order: Order;

}


@Entity()
export class Work {

    @PrimaryGeneratedColumn("increment")
    public id: number;

    @Column({type: "varchar", length: 40})
    public name: string;


    @Column({type: "interval"})
    public duration: IPostgresInterval;

    @ManyToMany(type => Order, order => order.works)
    public orders: Order[];

    @ManyToOne(type => Department, workplace => workplace.works)
    public department: Department;

}
```

work.controller.ts

```
import {Controller, Get, Next, Post, Res} from "@nestjs/common";
import {WorkService} from "../services/work.service";
import {NextFunction, Request, Response} from "express";
import {Req} from "@nestjs/common/decorators";

@Controller('work')
export class WorkController {

    constructor(private workService: WorkService) {
    }

    @Get("getAllWorksByDepartment/:id")
    public getAllWorksByDepartment(@Res() res: Response, @Next() next:
NextFunction, @Req() req: Request): void {
        const id = req.params.id;
        this.workService.getAllWorksByDepartment(id)
            .subscribe(works => {
                res.send(works);
            });
    }
```

```
    @Get("getAllFreeWorkplacesByDepartment/:id")
    public getAllFreeWorkplacesByDepartment(@Res() res: Response, @Next()
next: NextFunction, @Req() req: Request): void {
        const id = req.params.id;
        this.workService.getAllFreeWorkplacesByDepartmentId(id)
            .subscribe(works => {
                res.send(works);
            });
    }

    @Get("getAllDepartments")
    public getAllDepartments(@Res() res: Response, @Next() next:
NextFunction, @Req() req: Request): void {
        this.workService.getAllDepartments()
            .subscribe(departments => {
                res.send(departments);
            });
    }

    @Post("carToWork")
    public carToWork(@Res() res: Response, @Next() next: NextFunction, @Req()
req: Request): void {
        const request = req.body;
        this.workService.addCarToWork(request)
            .then(response => {
                res.send(response)
            })
    }

}
```

work.service.ts

```
import {Injectable} from "@nestjs/common";
import {InjectRepository} from "@nestjs/typeorm";
import {Car, Department, Order, Work, WorkingQueue, Workplace} from
"../entities/car.entity";
import {from, Observable} from "rxjs";
import {Repository} from "typeorm";

var moment = require('moment'); // require

@Injectable()
export class WorkService {

    constructor(
        @InjectRepository(Work) private workRepository: Repository<Work>,
        @InjectRepository(Department) private departmentRepository:
Repository<Department>,
        @InjectRepository(Car) private carRepository: Repository<Car>,
        @InjectRepository(Workplace) private workplaceRepository:
Repository<Workplace>,
        @InjectRepository(WorkingQueue) private workingQueueRepository:
Repository<WorkingQueue>,
        @InjectRepository(Order) private orderRepository: Repository<Order>
    ) {
    }

    public getAllDepartments(): Observable<Department[]> {
```

```
        const departments = this.departmentRepository.find();
        return from(departments);
    }

    public getAllFreeWorkplacesByDepartmentId(id: string):
Observable<Workplace[]> {
        const freeWorkplaces =
this.workplaceRepository.createQueryBuilder('workplace')
            .distinct(true)
            .leftJoin('workplace.workingQueues', 'workingQueue')
            .where("(workingQueue.status != :status OR workingQueue.id is
NULL)", {status: "in_progress"})
            .andWhere("workplace.department.id = :id", {id: id})
            .getMany();

        return from(freeWorkplaces);
    }

    public async addCarToWork(request): Promise<WorkingQueue> {
        const car = await this.carRepository.findOne(request.carId);
        car.mileage = request.mileage;
        const order = this.orderRepository.create({
            car: car,
            currentMileage: request.mileage,
            date: new Date(),
            comment: request.comment,
        });
        order.works = request.works.map(work =>
this.workRepository.create(work));
        const updatedCar = await this.carRepository.save(car);
        const savedOrder = await this.orderRepository.save(order);


        const duration = {hour: 0, minute: 0};
        request.works.forEach(work => {
            const hours = work.duration?.hours;
            if (hours) {
                duration.hour = duration.hour + hours;
            }
            const minutes = work.duration?.minutes;
            if (minutes) {
                duration.minute = duration.minute + minutes;
            }
        });
        const queue = this.workingQueueRepository.create(
            {
                status: "in_progress",
                workplace: request.workplace,
                startDate: new Date(),
                order: order,
                endDate: moment(new Date).add(duration).toDate()
            }
        );
        return this.workingQueueRepository.save(queue);
    }

    public getAllWorksByDepartment(id: string): Observable<Work[]> {
        const works = this.workRepository.find({
            relations: ["department"],
            where: {
                department: {
                    id
                }
            }
```

```
        });
        return from(works);
    }

}
```

main-page.component.html

```
<div style="padding: 10px 0 10px 10px">
    <span>Очередь на выдачу клиенту</span>
    <div style="display:flex;flex-direction: row;">
        <ng-container *ngFor="let model of carsToClient">
            <div style="padding: 0 10px 0 0">

                <mat-card style="margin: 10px 0;">
                    <mat-card-header>
                        <mat-card-title>{{model?.car?.brand?.name}}
{{model?.car?.model?.name}}, {{model?.car?.vinCode}}</mat-card-title>
                        <mat-card-subtitle>{{model?.car?.owner?.lastName}}
{{model?.car?.owner?.firstName}} {{model?.car?.owner?.middleName}}</mat-card-
subtitle>
                    </mat-card-header>
                    <mat-card-content>

                    </mat-card-content>
                    <mat-card-actions>
                        <button style="width: 100%" mat-raised-button
                                color="primary"
                                (click)="issuedToClient(model?.car?.id,
model?.queueId)"
                        >Выдано
                        </button></mat-card-actions>
                </mat-card>
            </div>
        </ng-container>
    </div>
</div>
<mat-divider></mat-divider>
<div style="padding: 10px 10px 0">{{departments?.length | multiple: 'цех':
'цеха'}}</div>

<ng-container *ngFor="let department of departments">
    <div style="padding: 10px">

        <mat-accordion>
            <mat-expansion-panel [expanded]="true">
                <mat-expansion-panel-header>

                    <mat-panel-title>
                        {{department?.name}}
                    </mat-panel-title>

                    <mat-panel-description>
                        {{department?.workplaces?.length | multiple: 'пост':
'поста'}}
                    </mat-panel-description>

                </mat-expansion-panel-header>
```

```
                <ng-container *ngFor="let workplace of
department?.workplaces; let i = index;">

                    <mat-list>
                        <div mat-subheader>{{workplace?.name}}</div>
                        <mat-list-item style="height: auto">
                            <div style="display: flex; flex-direction:
column;">
                                <car-in-progress
[workplaceId]="workplace?.id"

(completedWorkOnCar)="loadCarsToClientQueue()"
                                ></car-in-progress>
                            </div>
                        </mat-list-item>

                        <mat-divider
                            *ngIf="department?.workplaces?.length > 1 &&
department?.workplaces?.length - 1 !== i"></mat-divider>
                    </mat-list>
                </ng-container>

            </mat-expansion-panel>

        </mat-accordion>

    </div>
</ng-container>
```

main-page.component.html

```
import {Component, OnInit} from '@angular/core';
import {CarService} from "../../serivces/car.service";

@Component({
    selector: 'app-main-page',
    templateUrl: './main-page.component.html',
    styleUrls: ['./main-page.component.less']
})
export class MainPageComponent implements OnInit {

    public departments = [];
    public carsToClient = [];

    constructor(
        private carService: CarService
    ) {}

    ngOnInit(): void {
        this.loadDepartments();
        this.loadCarsToClientQueue();
    }

    private loadDepartments(): void {
        this.carService.getAllDepartmentsWithPosts()
            .subscribe(departments => {
                this.departments = departments;
            })
    }
```

```
    public issuedToClient(carId: number, queueId: number): void {
        const request = {carId, queueId};
        this.carService.issuedToClient(request)
            .subscribe(car => {
                this.loadCarsToClientQueue();
            });
    }

    public loadCarsToClientQueue(): void {
        this.carService.getAllCarsToClient()
            .subscribe(carsToClient => {
                this.carsToClient = carsToClient;
            })
    }
}
```