

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Интернет технологии и веб программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему

**ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ УПРАВЛЕНИЯ УСЛУГАМИ
АВТОМОБИЛЬНОЙ МАСТЕРСКОЙ**

БГУИР КП 1-53 01 02 10 ПЗ

Студент

С. В. Лосич

Руководитель

Н. В. Хаджинова

Минск 2024

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ
Заведующий кафедрой

(подпись)

2024г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Лосичу Семёну Викторовичу

1. Тема проекта Веб-приложение для управления услугами автомобильной мастерской

2. Срок сдачи студентом законченного проекта 16 декабря 2024 г.

3. Исходные данные к проекту Веб-приложение для управления услугами автомобильной мастерской помогает оптимизировать различные бизнес-аспекты в управлении автомобильной мастерской. Система хранит в базе данных и позволяет редактировать различную информацию о клиентах, автомобилях, работниках, бортовых ошибках, товарах и услугах предоставляемых асосервисом. Система предоставляет пользователю возможность просматривать различную информацию. Так же система предоставляет возможность ознакомиться с новостями о сервисе, отзывах и оценках. В системе функции должны быть разделены по ролям в зависимости от статуса пользователя. Роль администратора должна позволять управлять всей системой и просматривать важную информацию, которая предоставляет дополнительные сведения для управления бизнесом. Роль менеджера должна позволять управлять заказами и каталогами товаров, если таковые имеются. Роль механика должна предоставлять доступ к инструментам диагностики и расписанию ближайших ремонтов.

Роль клиента должна дать возможность ознакомиться как с каталогами, так и с общей информацией о сервисе или состоянии. Система должна представлять из себя Web-сайт с регистрацией и входом. Взаимодействовать с офисными программами такими как Excel. Система должна обеспечивать безопасное хранение паролей посредством хеширования.

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение

1. Выбор программных средств для реализации веб-приложения

2. Проектирование веб-приложения

3 Реализация веб-приложения

4 Руководство пользователя

Заключение

Приложения

7. Дата выдачи задания 15 сентября 2024 г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

раздел 1 к 2.10 – 30 %;

раздел 2 к 30.10 – 60 %;

раздел 3 к 4.12 – 90 %;

оформление пояснительной записки и графического материала к 16. 12 – 100 %.

Защита курсового проекта с 12.12 по 20.12.2024 г.

Руководитель _____ Н. В. Хаджинова
(подпись)

Задание приняла к исполнению _____ С.В.Лосич
(дата и подпись студента)

РЕФЕРАТ

ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ УПРАВЛЕНИЯ УСЛУГАМИ
АВТОМОБИЛЬНОЙ МАСТЕРСКОЙ: курсовой проект / С. В. Лосич. – Минск:
БГУИР, 2024, – п.з. – 53 с.

Курсовой проект на тему «Веб-приложение для управления услугами автомобильной мастерской» разработан с целью создания программы, для демонстрации механизма работы системы для сервисов по обслуживанию автомобилей.

Пояснительная записка к курсовому проекту состоит из введения, выбора программных средств для реализации веб-приложения, проектирования веб-приложения, а именно создания различных диаграмм, разработки программного кода веб-приложения, инструкции пользователя, заключения, списка использованных источников и приложения.

Для разработки системы выбран язык *PHP*, распространённый язык программирования общего назначения с открытым исходным кодом. *PHP* специально сконструирован для веб-разработок. Результат проекта представляет собой веб-приложение для управления услугами автомобильной мастерской, который так же можно использовать для управления автосервисом. В проекте присутствуют различные диаграммы, каждая из которых демонстрирует определенный аспект проекта.

В результате работы над курсовым проектом получена рабочая модель веб-программы.

Результат, полученный в ходе курсового проектирования, может использоваться участниками рынка малого бизнеса для прямого использования или в качестве примера, для написания собственного веб-приложения, которое сможет автоматизировать многие процессы, которые ранее выполнялись вручную.

СОДЕРЖАНИЕ

Введение.....	6
1 Выбор программных средств для реализации веб-приложения	8
1.1 Выбор базы данных.....	8
1.2 Выбор средств реализации клиентской и серверной частей	9
2 Проектирования веб-приоожения	11
2.1 Ппроектирование баз данных	11
2.2 Проектирования веб-приложения	12
3 Реализация веб-приложения	18
3.1 Реализация базы данных	18
3.2 Программная реализация модулей веб-приложения.....	22
4 Руководство пользователя.....	32
Заключение	46
Список использованных источников	47
Приложение А (обязательное) Листинг кода.....	48

ВВЕДЕНИЕ

В современном мире, когда в минуту производится огромное количество информации, трудно обойтись только человеческим ресурсом. За информацией уже не надо посылать гонца или часами проводить время в библиотеках. Все знания мира находятся буквально в наших карманах. На сегодняшний день алгоритмы стали куда эффективнее человека. Искусственный интеллект способен, хоть и в ограниченном формате, анализировать большие объемы однотипных данных и помогать лицу принимающему решение сделать правильный выбор.

Однако искусственный интеллект сам по себе довольно сложный для простого обывателя инструмент, поэтому его необходимо встроить в информационную систему. Которую сможет без труда освоить любой человек. В том числе работник предприятия.

Таким образом, отдав на обработку алгоритмам данные по ошибкам автомобиля, перечень доступного оборудования, списки деталей, добавив интерфейс для удобного взаимодействия получим автоматизированную информационную систему.

База данных играет важную роль в хранении и управлении информацией об автомобилях, сервисных работах, отзывах клиентов и других данных, связанных с обслуживанием транспортных средств. Проектирование эффективной и надежной структуры базы данных позволяет оптимизировать процессы поиска и выборки информации о техническом состоянии автомобилей, а также обеспечивает быстрый доступ к необходимым данным.

Для разработки информационной систем будет применяться язык программирования *PHP* совместно с *JavaScript*. *PHP* представляет собой мощный инструмент для создания веб-приложений и обеспечивает эффективную обработку данных. В данном проекте предполагается создание системы для диагностики и ремонта автомобилей, где используются различные технологии разработки программного обеспечения, основанные на *PHP*. Это обеспечивает возможность создания системы, которая легко интегрируется в рабочие процессы предприятия и обеспечивает эффективное взаимодействие с пользователями.

Цель курсовой работы состоит в проектировании и создании веб-приложения для управления услугами автомобильной мастерской, основанной на современных принципах и технологиях разработки программного обеспечения с использованием языка *PHP*. Данное веб-приложение будет

способствовать повышению эффективности сервисов по обслуживанию автомобилей и сокращению времени ожидания клиентов.

Тема создания веб-приложения для управления услугами автомобильной мастерской актуальна по нескольким причинам.

Во-первых, быстрое развитие автомобильной технологии: современные автомобили оснащены сложной электроникой, сенсорами и системами управления, что делает диагностику и ремонт более сложными и требующими специальных знаний. Информационная система, специализированная на диагностике и ремонте автомобилей, становится необходимым инструментом для облегчения работы автомехаников и повышения эффективности.

Во-вторых, увеличение количества автомобилей на дорогах: постоянно растущее число автомобилей создает повышенный спрос на услуги, оказываемые автосервисом. Информационная система позволяет оптимизировать процесс обслуживания автомобилей, сокращая время на поиск и устранение неисправностей, что позволяет повысить пропускную способность автосервисов и улучшить качество обслуживания.

В-третьих, требования безопасности и экологичности: в современном мире все больше внимания уделяется безопасности дорожного движения и экологической совместимости автомобилей. Информационная система диагностики и ремонта может помочь выявить и устранить потенциальные проблемы, связанные с безопасностью и экологическими нормами, что способствует повышению уровня безопасности на дорогах и снижению вредного влияния автомобилей на окружающую среду.

1 ВЫБОР ПРОГРАММНЫХ СРЕДСТВ ДЛЯ РЕАЛИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЯ

На рынке существует множество средств, которые стоит рассмотреть при выборе подходящих для реализации проекта.

1.1 Выбор базы данных

В контексте клиент-серверных приложений, СУБД (система управления базами данных) играет важную роль, так как она отвечает за хранение, организацию и обработку данных, используемых приложением. СУБД обеспечивает надежность, целостность и безопасность данных, а также позволяет клиентской и серверной частям приложения эффективно обмениваться информацией.

В области баз данных конкурирующими между собой решениями являются *MySQL*, *PostgreSQL* и *Microsoft SQL Server*.

MySQL – это популярная открытая реляционная система управления базами данных (СУБД), которая имеет преимущества для разработки системы оценки кинофильмов, такие как возможность бесплатного использования и имеет открытый исходный код и высокая производительность и эффективность. Кроме того, *MySQL* является одной из самых популярных СУБД и имеет огромное сообщество разработчиков и пользователей. Помимо прочего, эта СУБД предлагает широкий набор функциональных возможностей, которые могут быть использованы для разработки системы.

PostgreSQL является мощной реляционной системой управления базами данных с богатым набором функций и широкими возможностями масштабирования. Он обладает множеством функций, поддерживает сложные запросы и обеспечивает высокую производительность. *MySQL* и *PostgreSQL* оба могут использоваться для различных типов приложений, но в зависимости от требований и характеристик приложения одна СУБД может быть более подходящей, чем другая. *MySQL* часто используется для разработки веб-сайтов и приложений, особенно тех, которые требуют быстрого чтения и записи данных, таких как системы управления контентом, интернет-магазины и социальные сети. Если приложение имеет высокую нагрузку транзакций и требует быстрой обработки множества одновременных запросов на чтение и запись данных, *MySQL* может быть хорошим выбором. Примерами таких приложений могут быть банковские системы, системы учета и системы бронирования. А *PostgreSQL*, в свою очередь, обладает богатыми

возможностями для аналитической обработки данных и поддержкой сложных запросов. Примерами таких приложений могут быть системы бизнес-аналитики и системы управления данными. *PostgreSQL* имеет встроенную поддержку географических данных и функций, что делает его идеальным выбором для приложений, связанных с геолокацией, картографией и географическими информационными системами.

При сравнении *MySQL* и *Microsoft SQL Server* следует учитывать, что *MySQL* является открытой системой управления базами данных с бесплатной версией *Community Edition*, которая доступна для использования без ограничений. С другой стороны, *Microsoft SQL Server* имеет коммерческую лицензию с различными редакциями, включая платные версии с дополнительными функциями и возможностями. Обе системы обладают богатым набором функций и возможностей, таких как поддержка транзакций, хранимых процедур, триггеров и индексов. Обе системы обеспечивают хорошую производительность при обработке запросов и манипулировании данными. Однако производительность может различаться в зависимости от конкретных сценариев использования и настроек. *MySQL* известен своей высокой производительностью при работе с простыми запросами и большими объемами данных, в то время как *Microsoft SQL Server* может предлагать более высокую производительность для сложных запросов и аналитических операций.

В итоге, рассмотрев характеристики различных СУБД и учитывая требования к разрабатываемой системе, рационально использовать *MySQL*.

1.2 Выбор средств реализации клиентской и серверной частей

Для реализации серверной и клиентской частей веб-приложения были выбраны *PHP* и *JavaScript*, соответственно. Каждый из них обладает своими уникальными особенностями и преимуществами, что делает их подходящими для разработки различных частей приложения.

PHP является широко используемым языком программирования для серверной разработки. Он обеспечивает эффективную обработку запросов на стороне сервера и простоту взаимодействия с базой данных. Благодаря своей долгой истории использования в веб-разработке, *PHP* предоставляет множество библиотек и инструментов для облегчения создания серверных приложений.

PHP, в качестве серверного языка программирования, также отлично интегрируется с различными системами управления базами данных, такими как *MySQL* или *PostgreSQL*. Это позволяет разработчикам легко создавать, модифицировать и извлекать данные из баз данных, обеспечивая стабильность и эффективность работы веб-приложений.

Важным преимуществом *PHP* является его распространенность и широкий спектр поддерживаемых хостинг-платформ. Это обеспечивает гибкость в выборе платформы для размещения приложения и упрощает развертывание на различных серверах.

Кроме того, активное сообщество *PHP*-разработчиков предоставляет множество ресурсов, обучающих материалов и сторонних библиотек, что способствует ускоренному развитию проектов и решению различных задач.

Интеграция *JavaScript* с *PHP* в проекте также играет важную роль. Она позволяет создавать динамичные веб-приложения, взаимодействуя с серверной частью и обновляя содержимое страницы без необходимости полной перезагрузки. *AJAX* (асинхронный *JavaScript* и *XML*) — это технология, позволяющая обмениваться данными между браузером и сервером асинхронно. Это особенно полезно в автомобильном сервисе, где пользователи могут оставлять свои отзывы и запросы, и их данные могут обновляться на странице в режиме реального времени без перезагрузки.

Таким образом, *JavaScript* дополняет функциональность веб-приложения, обеспечивая не только эффектные интерфейсные решения, но и возможность взаимодействия с серверной частью, что повышает удобство использования и привлекательность приложения для пользователей автомобильного сервиса.

Использование *JavaScript* для простых анимаций в веб-приложении предоставляет возможность создавать динамичные и привлекательные эффекты, обогащающие пользовательский интерфейс. Одним из простых способов внедрения анимаций – это использование функции *setTimeout* или *setInterval* для изменения стилей элементов на странице через определенные промежутки времени. Например, это может быть изменение цвета фона или плавное перемещение элемента.

Применение таких простых анимаций помогает сделать веб-приложение более интерактивным, привлекая внимание пользователя к ключевым элементам и улучшая визуальный опыт использования.

Выбор данных технологий обусловлен их популярностью, активным сообществом разработчиков и способностью обеспечивать требования проекта..

2 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

2.1 Проектирование базы данных

В ходе проектирования базы данных (БД), важной частью является определение структуры и организации данных, обеспечивая эффективное хранение, доступ и манипуляции информацией. Этот процесс включает в себя выделение сущностей (таблиц), атрибутов (столбцов), связей и установление правил для поддержания целостности данных.

Система управления автомобилями включает следующие ключевые сущности: "Пользователь" (*User*), "Роли" (*Roles*), "Отзывы" (*Reviews*), "Услуги" (*Services*), "Автомобили" (*Cars*), "Ошибки автомобилей" (*CarErrors*), "Запчасти для автомобилей" (*CarParts*), "Текущие ремонты" (*CurrentRepairs*) и "История ремонтов" (*HistoryRepairs*).

Сущность «Пользователь» содержит уникальный идентификатор, зашифрованный пароль, имя, электронную почту.

Сущность «Роли» связана с пользователями и содержит информацию о ролях, присвоенных им. Поля «username» и «role» используются для связи с именем пользователя и определения роли.

Сущность «Отзывы» предназначена для хранения информации о пользовательских отзывах о системе. Уникальный идентификатор, имя пользователя, текст отзыва, рейтинг и дата создания отзыва определяют эту сущность.

Сущность «Услуги» содержит информацию о предоставляемых услугах, включая уникальный идентификатор, название, описание и цену.

Сущность «Автомобили» отражает информацию о транспортных средствах, включая марку, регистрационный номер, цвет, год выпуска и VIN-номер.

Сущность «Ошибки автомобилей» содержит данные об ошибках в транспортных средствах, включая код, название и описание ошибки.

Сущность «Запчасти для автомобилей» содержит информацию о запчастях, включая уникальный идентификатор, *SKU*, название, описание, цену и количество в наличии.

Сущности «Текущие ремонты» и «История ремонтов» предназначены для отслеживания информации о текущих и завершенных ремонтах, включая данные о механиках, менеджерах, клиентах, марках автомобилей, VIN-номерах, датах начала и завершения, стоимости и статусе ремонта.

Таким образом, эти сущности и связи между ними образуют основу для эффективного функционирования системы управления автомобилями, обеспечивая надежное хранение и управление разнообразной информацией.

2.2 Проектирование веб-приложения

Приложение является клиент-серверным. Клиент-серверное приложение – это архитектурный подход, в котором приложение разделено на два основных компонента: клиентскую часть и серверную часть. Клиентская часть обычно представляет собой интерфейс пользователя, который взаимодействует с приложением, в то время как серверная часть отвечает за обработку запросов клиента, выполнение бизнес-логики и управление данными. Клиент и сервер взаимодействуют посредством *HTTP*.

Приложение написано с использованием *RНР*, что отражается на его архитектуре и спецификах. *RНР* позволяет реализовать серверную часть, обрабатывающую запросы от клиента и взаимодействующую с базой данных для выполнения операций.

Клиентская часть приложения, взаимодействуя с сервером, отправляет запросы на обработку. Сервер, в свою очередь, выполняет необходимые операции с данными в базе данных и отправляет результаты обратно клиенту. Взаимодействие между клиентом и сервером организовано с использованием *HTTP*.

Для обеспечения эффективной разработки и обновления приложения на *RНР* используется структура, соответствующая принципам компонентной архитектуры. *RНР* также позволяет создавать множество маленьких и переиспользуемых компонентов, что способствует лучшей организации кода и повторному использованию функциональности.

В *RНР*-приложении вместо *React* используются технологии, специфичные для *RНР*, для построения пользовательского интерфейса. Архитектурные принципы, такие как передача данных по иерархии компонентов в одном направлении и управление состоянием, также остаются актуальными, хотя реализация может отличаться.

Таким образом, архитектура клиент-серверного приложения, написанного с использованием *RНР*, обеспечивает разделение ответственности между клиентской и серверной частями, упрощает разработку и поддержку приложения, а также обеспечивает эффективное взаимодействие между компонентами и сервером через *API*. Для описания

структуры приложения будем использовать диаграмму вариантов использования, диаграмму классов и схему алгоритма.

2.2.1 Диаграмма вариантов использования (*Use Case Diagram*) является одним из ключевых инструментов в разработке программного обеспечения. Она представляет собой графическое представление функциональных требований системы и позволяет понять, как различные акторы (пользователи или внешние системы) взаимодействуют с системой в рамках конкретных сценариев.

Для построения модели вариантов использования необходимо определить действующие лица и варианты использования.

Действующие лица представляют сущности, которые взаимодействуют с системой. Другими словами, это роли, которые пользователи играют по отношению к системе.

Варианты использования предоставляют функциональные возможности системы и описывают конкретные действия, которые акторы могут выполнять, взаимодействуя с системой. Каждый вариант использования описывает один конкретный сценарий взаимодействия между актором и системой.

В системе оценки кинофильмов можно выделить следующие действующие лица:

- «Менеджер» – человек, который использует систему в качестве менеджера;
- «Механик» – человек, который использует систему для диагностики;
- «Пользователь» – человек, который использует систему;
- «Администратор» – человек, который управляет системой в целом.

Исходя из потребностей действующих лиц, выделяются следующие варианты использования:

- Авторизоваться;
- Оставить комментарий;
- Оценить обслуживание;
- Отсортировать услуги;
- Посмотреть информацию о услуге;
- Поиск услуги;
- Выйти;
- Записаться на услугу.
- Работа с услугами;
- Редактирование каталога услуг;

- Пригласить на обслуживание;
- Просмотреть аналитику;
- Работа с отзывами;
- Просмотр истории обслуживания.
- Работа с системой диагностики автомобилей (*OBD*);
- Просмотр ближайших обслуживаний;
- Просмотр истории обслуживания;
- Работа с услугами;
- Установка ролей;
- Удаление аккаунтов;
- Добавление ошибок в данные о диагностике;
- Работа с отзывами;
- Просмотр текущих обслуживаний и истории обслуживания;
- Отсортировать фильмы;
- Выйти;
- Изменить роли;
- Экспортировать данные.

Диаграмма вариантов использования для автоматизированной системы диагностики и ремонта автомобилей представлена на рисунке 2.1.

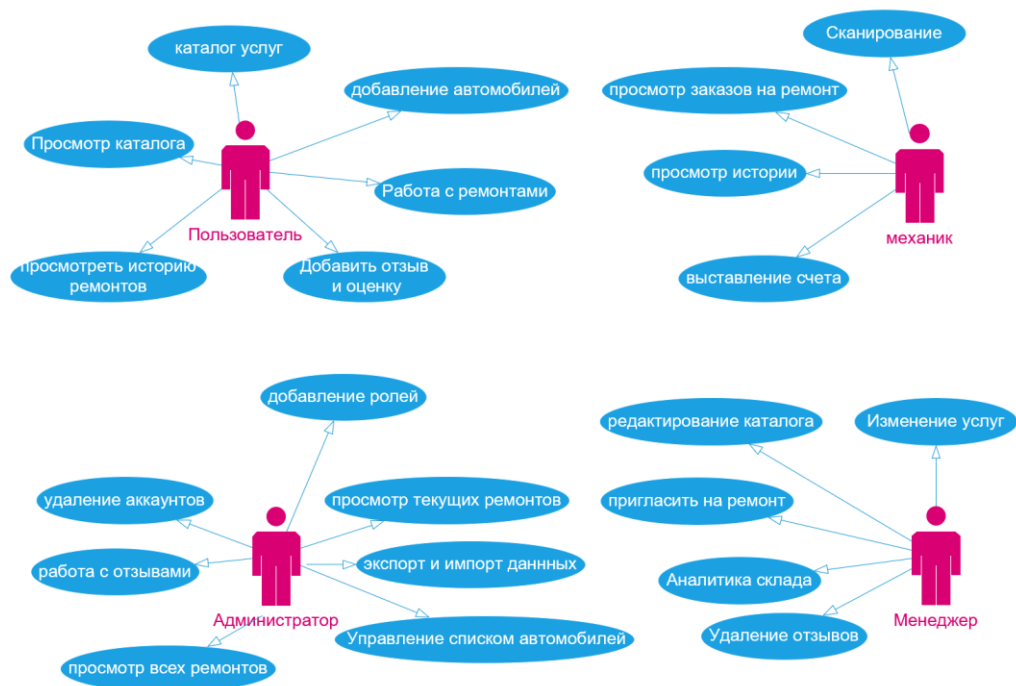


Рисунок 2.1 – Диаграмма вариантов использования

На диаграмме вариантов использования показано взаимодействие между вариантами использования и действующими лицами. Она отражает требования к системе.

Между вариантами использования и действующими лицами использовались связи разных типов: связи коммуникации и расширения.

Связь коммуникации в контексте диаграммы вариантов использования отображает взаимодействие между действующими лицами и вариантами использования. Она показывает, как действующие лица используют варианты использования для достижения определенных целей или выполнения определенных задач.

На диаграмме вариантов использования связь коммуникации представлена стрелками, указывающими направление взаимодействия между действующим лицом и вариантом использования. Стрелка указывает, что действующее лицо инициирует определенное действие или запрос, и этот запрос обрабатывается или обслуживается соответствующим вариантом использования.

Связь расширения используется для показа дополнительного поведения, которое может быть добавлено в основной вариант использования при определенных условиях. Она позволяет моделировать ситуации, когда один вариант использования "расширяет" другой вариант использования, добавляя дополнительные шаги или функциональность.

Связь расширения полезна для моделирования вариантов использования, которые имеют общую базовую функциональность, но могут варьироваться в зависимости от определенных условий или сценариев. Она помогает улучшить модульность и гибкость системы, позволяя легко добавлять или изменять функциональность в зависимости от потребностей.

2.2.2 Диаграмма классов является типом диаграммы, который используется в объектно-ориентированном программировании (ООП) для визуализации структуры классов и их отношений в системе. Она представляет собой графическое представление классов, интерфейсов, атрибутов, методов и связей между ними.

Методы класса представляют его поведение и обычно представлены в виде функций внутри класса. Они могут быть отображены с указанием возвращаемого типа, параметров и видимости.

Диаграмма классов также позволяет отображать различные типы отношений между классами, такие как ассоциация, наследование, реализация

интерфейсов и агрегация. Эти отношения могут быть представлены с помощью стрелок и различных маркеров на линиях связей между классами.

Данную диаграмму классов можно применить для проектирования более сложной версии данной автоматизированной системы.

Диаграмма классов продемонстрирована на рисунке 2.2.

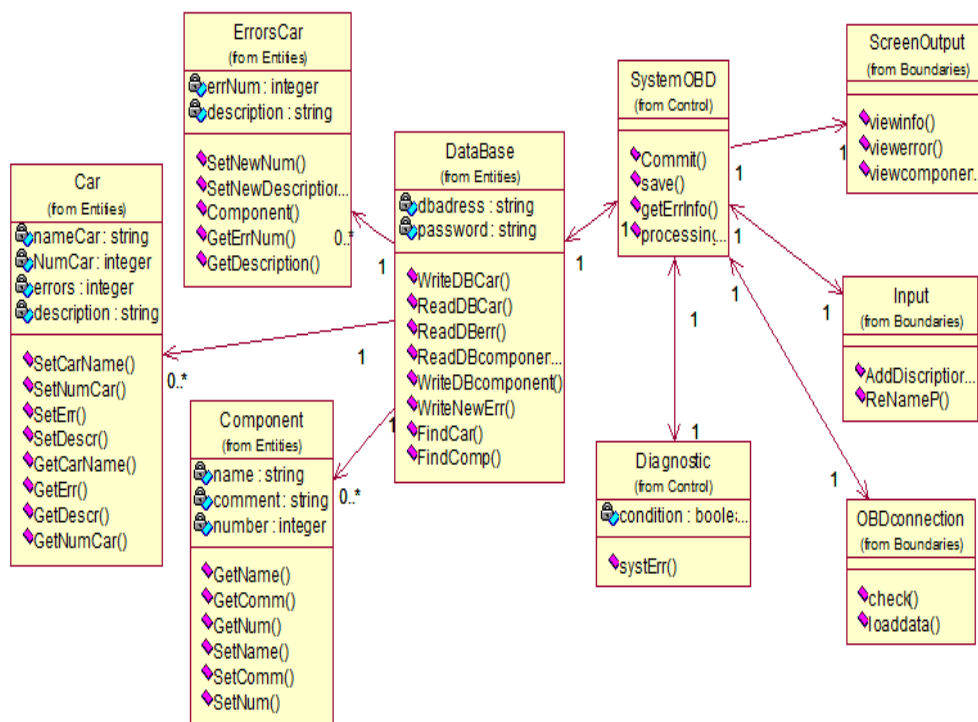


Рисунок 2.2 – Диаграмма классов

2.2.3 Автоматизированная система диагностики и ремонта предоставляет множество возможностей для пользователей, в частности для механика. Для полноценного функционирования системы к ней надо подключить диагностическое оборудование.

Алгоритм работы системы для выявления неполадок во время подключения и загрузки информации через диагностическое оборудование, такое как *OBD*-сканер:

- Проверка подключения автомобиля;
- Загрузка данных;
- Вывод информации.

Ключевая функция системы состоит в предоставлении диагностики в автоматическом режиме. Блок-схема алгоритма выставления оценки продемонстрирована на рисунке 2.3.

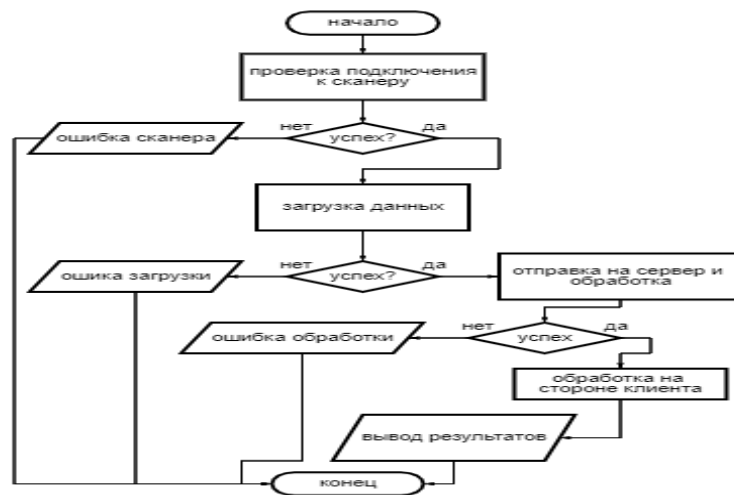


Рисунок 2.3 – Схема проверки готовности диагностического устройства

3 РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ

3.2 Реализация базы данных

Физическая схема базы данных представляет собой конкретные детали о структуре и хранении данных на физическом уровне. Она является основой для создания самой базы данных и ее объектов. В физической схеме определяются таблицы, которые будут созданы в базе данных, а также указываются названия столбцов в таблицах и их типы данных.

Ограничения также могут быть указаны для столбцов в таблицах. Например, можно определить первичные ключи, внешние ключи, ограничения уникальности и ограничения на значения столбцов. Эти ограничения помогают поддерживать целостность данных и связей между таблицами.

Еще одним аспектом физической схемы является физическое расположение данных на диске. Данные могут быть организованы в файлы, разделы или таблицы пространства таблиц. Это включает решения о физическом размещении таблиц и индексов на диске, а также выбор оптимальных методов хранения данных.

Физическая схема базы данных разрабатывается на основе логической схемы и учитывает требования к производительности, доступности данных и эффективности запросов. Она может быть оптимизирована для конкретной системы управления базами данных (СУБД) и используемого аппаратного обеспечения.

Для реализации базы данных использовалась система управления базами данных *MySQL*, поэтому типы данных на физической схеме соответствуют доступным типам данных в выбранной СУБД.

Опишем подробнее таблицы, поля, типы данных и диапазоны допустимых значений.

Таблица *users* содержит информацию о пользователях:

- поле *id* является целочисленным (*INT*) с автоинкрементом и является первичным ключом (*PRIMARY KEY*), оно представляет уникальный идентификатор пользователя в диапазоне от 1 до 2147483647;
- Поле *username* является текстовым (*VARCHAR(255)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения имени пользователя;
- Поле *email* является текстовым (*VARCHAR(255)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения адреса электронной почты пользователя;

- Поле *password* является текстовым (*VARCHAR(255)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения захешированного пароля пользователя.;

Таблица *roles* содержит информацию о ролях пользователей:

- Поле *username* является текстовым (*VARCHAR(255)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения имени пользователя, которому присвоена определенная роль;

- Поле *role* является текстовым (*VARCHAR(255)*) и может содержать значение по умолчанию *NULL*. Оно используется для хранения роли, присвоенной пользователю. Роль может отсутствовать (*NULL*) или иметь конкретное значение в соответствии с бизнес-логикой системы;

Таблица *reviews* предназначена для хранения информации о пользовательских отзывах:

- поле *id* является целочисленным (*INT*) с автоинкрементом и является первичным ключом;

- Поле *username* является текстовым (*VARCHAR(255)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения имени пользователя, оставившего отзыв.

- Поле *review_text* является текстовым (*TEXT*) и не может быть пустым (*NOT NULL*). Оно предназначено для хранения текста отзыва пользователя;

- Поле *rating* является целочисленным (*INT(10)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения рейтинга, присвоенного отзыву.

- Поле *review_date* является датой (*DATE*) и не может быть пустым (*NOT NULL*). Оно предназначено для хранения даты, когда был оставлен отзыв

- Таблица *services* предназначена для хранения информации о предоставляемых услугах:

- Поле *id* является целочисленным (*INT(10)*) с автоинкрементом и представляет собой уникальный идентификатор услуги.

- Поле *service_name* является текстовым (*VARCHAR(255)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения названия услуги.

- Поле *description* является текстовым (*TEXT*) и может содержать *NULL*. Оно предназначено для хранения описания услуги.

- Поле *price* является числовым с фиксированной точностью (*DECIMAL(10,2)*) и не может быть пустым (*NOT NULL*). Оно используется для хранения стоимости услуги.

Таблица *cars* предназначена для хранения информации о транспортных средствах:

- поле *id* является целочисленным (*INT*) с автоинкрементом и является первичным ключом;
- Поле *username* является текстовым (*VARCHAR(255)*) и может содержать *NULL*. Оно используется для связи с пользовательским именем, указывая, кому принадлежит транспортное средство;
- Поле *brand* является текстовым (*VARCHAR(50)*) и может содержать *NULL*. Оно используется для хранения марки (бренда) транспортного средства;
- Поле *license_plate* является текстовым (*VARCHAR(20)*) и может содержать *NULL*. Оно предназначено для хранения регистрационного номера транспортного средства;
- Поле *color* является текстовым (*VARCHAR(20)*) и может содержать *NULL*. Оно используется для хранения цвета транспортного средства;
- Поле *year* является целочисленным (*INT(10)*) и может содержать *NULL*. Оно используется для хранения года выпуска транспортного средства;
- Поле *vin_number* является текстовым (*VARCHAR(50)*) и может содержать *NULL*. Оно используется для хранения уникального идентификационного номера транспортного средства (*VIN*);

Таблица *car_errors* предназначена для хранения информации об ошибках транспортных средств:

- поле *id* является целочисленным (*INT*) с автоинкрементом и является первичным ключом;
- Поле *error_code* является текстовым (*VARCHAR(10)*) и может содержать *NULL*. Оно используется для хранения кода ошибки;
- Поле *error_name* является текстовым (*VARCHAR(100)*) и может содержать *NULL*. Оно предназначено для хранения названия ошибки;
- Поле *error_description* является текстовым (*TEXT*) и может содержать *NULL*. Оно предназначено для хранения подробного описания ошибки;

Таблица *car_parts* предназначена для хранения информации о запчастях для транспортных средств:

- Поле *part_id* является целочисленным (*INT(10)*) с автоинкрементом и представляет собой уникальный идентификатор запчасти;
- Поле *part_sku* является текстовым (*VARCHAR(50)*) и не может содержать *NULL*. Оно предназначено для хранения уникального кода *SKU* (*Stock Keeping Unit*) запчасти и используется в качестве уникального индекса;

- Поле *part_name* является текстовым (*VARCHAR(255)*) и не может содержать *NULL*. Оно предназначено для хранения названия запчасти;
- Поле *part_description* является текстовым (*TEXT*) и может содержать *NULL*. Оно предназначено для хранения подробного описания запчасти;
- Поле *part_price* является числовым (*DECIMAL(10,2)*) и не может содержать *NULL*. Оно предназначено для хранения цены запчасти.
- Поле *part_stock* является целочисленным (*INT(10)*) и не может содержать *NULL*. Оно предназначено для отслеживания количества запчастей в наличии.

Таблицы *current_repairs* и *history_repairs* предназначены для отслеживания текущих ремонтов транспортных средств и истории ремонтов:

- Поле *id_repair* является целочисленным (*INT(10)*) с автоинкрементом и представляет собой уникальный идентификатор текущего ремонта;
- Поле *mech_name* является текстовым (*VARCHAR(255)*) и может содержать *NULL*. Оно предназначено для хранения имени механика, выполняющего ремонт;
- Поле *manager_name* является текстовым (*VARCHAR(255)*) и может содержать *NULL*. Оно предназначено для хранения имени менеджера, управляющего ремонтом;
- Поле *client_name* является текстовым (*VARCHAR(255)*) и может содержать *NULL*. Оно предназначено для хранения имени клиента, заказавшего ремонт;
- Поле *car_brand* является текстовым (*VARCHAR(255)*) и может содержать *NULL*. Оно предназначено для хранения марки транспортного средства, подвергающегося ремонту;
- Поле *vin_number* является текстовым (*VARCHAR(17)*) и может содержать *NULL*. Оно предназначено для хранения уникального идентификационного номера *VIN* транспортного средства;
- Поле *start_date* является типом *DATE* и может содержать *NULL*. Оно предназначено для отражения даты начала ремонта.
- Поле *finish_date* является типом *DATE* и может содержать *NULL*. Оно предназначено для отражения даты завершения ремонта.
- Поле *cost* является числовым (*DECIMAL(10,2)*) и может содержать *NULL*. Оно предназначено для хранения стоимости ремонта.
- Поле *status* является целочисленным (*INT(10)*) и по умолчанию имеет значение '0'. Оно предназначено для отслеживания статуса ремонта (например, "в процессе", "завершен" и т. д.)

Каждая таблица имеет определенные поля с соответствующими типами данных и ограничениями, которые обеспечивают эффективное хранение и организацию информации в базе данных.

Физическая схема данных продемонстрирована на рисунке 3.1.

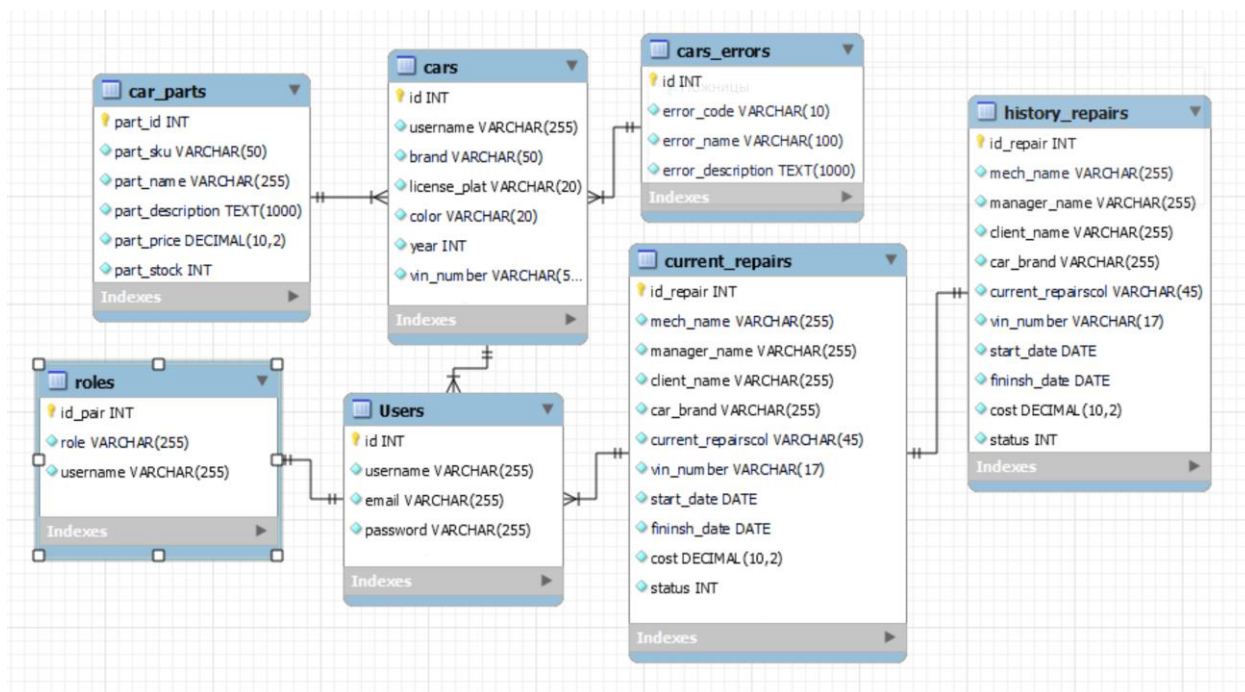


Рисунок 3.1 – Физическая схема базы данных

3.1 Программная реализация модулей веб-приложения

Приведем программный код основных функций приложения, где видна реализация работы с БД с кратким пояснением.

Регистрация в системе, *PHP* код:

```

<?php
// Подключение к базе данных
require_once('database_config.php');
$conn = new mysqli($servername, $username, $password, $dbname);
// Проверка соединения
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
// Получение данных из формы регистрации
$username = $_POST['username'];
    
```

```

$email = $_POST['email'];
$password = $_POST['password'];
$confirm_password = $_POST['confirm_password'];
$response = array();
// Проверка, существует ли уже такой логин в базе данных
$sql_check_username = "SELECT * FROM users WHERE username =
'$username'";
$result_check_username = $conn->query($sql_check_username);
// Проверка, существует ли уже такая почта в базе данных
$sql_check_email = "SELECT * FROM users WHERE email = '$email'";
$result_check_email = $conn->query($sql_check_email);
if ($result_check_username->num_rows > 0) {
    $response['username_error'] = "Такой логин уже используется.";
} elseif ($result_check_email->num_rows > 0) {
    $response['email_error'] = "Такая почта уже используется.";
} elseif ($password != $confirm_password) {
    $response['password_error'] = "Пароли не совпадают.";
} else {
    // Хэширование пароля
    $hashed_password = password_hash($password,
PASSWORD_DEFAULT);
    // SQL запрос для вставки данных в таблицу users
    $sql = "INSERT INTO users (username, email, password) VALUES
('$username', '$email', '$hashed_password')";
    if ($conn->query($sql) === TRUE) {
        $response['success'] = "Пользователь зарегистрирован успешно";
    } else {
        $response['error'] = "Ошибка: " . $sql . "<br> " . $conn->error;
    }
}
echo json_encode($response);
$conn->close();
?>

```

В начале кода устанавливается соединение с базой данных, используя предварительно определенные параметры для сервера, имени пользователя, пароля и имени базы данных. Это обеспечивает доступ к базе данных для выполнения необходимых операций.

Далее происходит извлечение данных из формы регистрации, таких как логин, электронная почта, пароль и его подтверждение. Эти данные присваиваются соответствующим переменным.

Затем осуществляется проверка уникальности логина и электронной почты в базе данных. Для этого выполняются *SQL*-запросы для поиска совпадений среди уже зарегистрированных пользователей. Если обнаруживается, что логин или электронная почта уже заняты, код формирует сообщение об ошибке.

Далее код проверяет совпадение введенных паролей. Если пароли не совпадают, также формируется сообщение об ошибке.

Если все проверки пройдены успешно, пароль хэшируется с использованием функции *password_hash()*, что обеспечивает безопасное хранение пароля в базе данных. После чего данные нового пользователя добавляются в базу данных через *SQL*-запрос.

В случае возникновения ошибок на любом этапе процесса, код формирует соответствующие сообщения об ошибках. В противном случае возвращается сообщение об успешной регистрации пользователя.

Этот код также следит за безопасностью данных, хешируя пароль перед сохранением, что повышает уровень безопасности системы.

Сортировка автомобилей *PHP* код:

```
<?php
// Подключение к базе данных
require_once('database_config.php');
$conn = new mysqli($servername, $username, $password, $dbname);
// Проверка подключения
if ($conn->connect_error) {
    die("Ошибка подключения: " . $conn->connect_error);
}
// Получение фильтра, если предоставлен
$filter = $_POST['filter'] ?? '';
// SQL-запрос для получения данных о машинах с применением фильтра
$sql = "SELECT * FROM cars";
// Добавление условия WHERE только при наличии фильтра
if (!empty($filter)) {
```



```

    $sql .= " WHERE brand LIKE '%$filter%' OR license_plate LIKE
'%$filter%' OR color LIKE '%$filter%' OR vin_number LIKE '%$filter%' OR
username LIKE '%$filter%'";
}
$sql .= " ORDER BY year";
$result = $conn->query($sql);
// Вывод данных о машинах
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "<tr>";
        echo "<td>" . $row['username'] . "</td>";
        echo "<td>" . $row['brand'] . "</td>";
        echo "<td>" . $row['license_plate'] . "</td>";
        echo "<td>" . $row['color'] . "</td>";
        echo "<td>" . $row['year'] . "</td>";
        echo "<td>" . $row['vin_number'] . "</td>";
        echo " <td><button class='delete-button' onclick='deleteCar(\"" .
$row['vin_number'] . "\" )>Удалить</button></td>";
        echo "</tr>";
    }
} else {
    echo "<tr><td colspan='7'>Нет доступных данных</td></tr>";
}
$conn->close();
?>

```

Данный *PHP*-код представляет собой скрипт для извлечения и вывода информации о машинах из базы данных с учетом фильтрации. Рассмотрим основные шаги его работы.

Сначала код устанавливает соединение с базой данных, используя параметры, предоставленные в файле *'database_config.php'*. В случае неудачного подключения, код выводит сообщение об ошибке и прерывает выполнение.

Далее происходит получение параметра фильтрации из *POST*-запроса. Если параметр не предоставлен, устанавливается значение по умолчанию — пустая строка.

Формируется *SQL*-запрос для извлечения данных о машинах из таблицы *'cars'*. Если указан фильтр, добавляется условие *WHERE*, фильтрующее

результаты по нескольким столбцам, таким как 'brand', 'license_plate', 'color', 'vin_number' и 'username'.

SQL-запрос завершается добавлением условия *ORDER BY* для сортировки результатов по столбцу 'year'.

После выполнения запроса, полученные данные обрабатываются в цикле. Если в результате запроса есть строки, код формирует *HTML*-таблицу, выводя информацию о машинах, включая 'username', 'brand', 'license_plate', 'color', 'year', 'vin_number' и добавляет кнопку 'Удалить' для каждой строки, привязанной к соответствующему 'vin_number'. В случае отсутствия данных, выводится строка с сообщением о их отсутствии.

Обновление данных о запчастях *PHP* код:

```
<?php
// Подключение к базе данных
require_once('database_config.php');
// Получение данных из POST-запроса
// $partId = $_GET['partId'];
$partId = $_POST['partId'];
$partName = $_POST['partName'];
$partDescription = $_POST['partDescription'];
$partPrice = $_POST['partPrice'];
$partStock = $_POST['partStock'];
// Проверка соединения с базой данных
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die("Ошибка подключения к базе данных: " . $conn->connect_error);
}
// SQL-запрос для обновления данных о запчасти
$sql = "UPDATE car_parts SET part_name = '$partName', part_description
= '$partDescription', part_price = $partPrice, part_stock = $partStock WHERE
part_id = $partId";
if ($conn->query($sql) === TRUE) {
    echo "Данные успешно обновлены";
} else {
    echo "Ошибка при обновлении данных: " . $conn->error;
}
// Закрытие соединения с базой данных
```

```
$conn->close();  
?>
```

Данный *PHP*-скрипт представляет собой обработчик *POST*-запроса для обновления данных о запчастях в базе данных. Рассмотрим его функционал по шагам.

Сначала происходит подключение к базе данных, используя параметры из файла *'database_config.php'*. В случае неудачного подключения, скрипт выводит сообщение об ошибке и прерывает выполнение.

Затем извлекаются данные из *POST*-запроса, включая идентификатор запчасти (*'partId'*), название (*'partName'*), описание (*'partDescription'*), цену (*'partPrice'*) и количество на складе (*'partStock'*).

Далее выполняется проверка соединения с базой данных. Если соединение успешно, формируется *SQL*-запрос для обновления данных о запчасти в таблице *'car_parts'*. Запрос использует переданные значения для обновления соответствующих полей.

После выполнения запроса, код проверяет успешность операции. В случае успеха, выводится сообщение "Данные успешно обновлены". В противном случае, выводится сообщение об ошибке с описанием ошибки.

Серверная часть обработки услуг, набор функций *PHP*:

```
<?php  
// Функция для подключения к базе данных  
function connectToDatabase() {require_once('database_config.php');  
    $conn = new mysqli($servername, $username, $password, $dbname);  
    // Проверка соединения  
    if ($conn->connect_error) {  
        die("Connection failed: " . $conn->connect_error);  
    }  
    return $conn;  
}  
// Функция для закрытия соединения с базой данных  
function closeDatabaseConnection($conn) {  
    $conn->close();  
}  
function getServices() {  
    $conn = connectToDatabase();  
    $sql = "SELECT * FROM services ORDER BY id DESC";
```

```

$result = $conn->query($sql);
$services = array();
// Проверка наличия записей в таблице
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $services[] = $row;
    }
} else {
    echo "Нет услуг";
}
closeDatabaseConnection($conn);
return $services;
}

function addService($service_name, $description, $price)
{
    // Подключение к базе данных
    $conn = connectToDatabase();
    try {
        // Подготовка SQL-запроса
        $sql = "INSERT INTO services (service_name, description, price)
VALUES (?, ?, ?)";
        $stmt = $conn->prepare($sql);
        // Выполнение SQL-запроса с передачей данных
        $stmt->bind_param("ssd", $service_name, $description, $price);
        $stmt->execute();
    } catch (Exception $e) {
        // Обработка ошибок, если необходимо
        echo "Error: " . $e->getMessage();
    }
    // Закрытие соединения с базой данных
    closeDatabaseConnection($conn);
}

function editServiceByName($service_name, $new_description, $new_price)
{
    // Подключение к базе данных
    $conn = connectToDatabase();
    try {
        // Подготовка SQL-запроса

```

```

        $sql = "UPDATE services SET description = ?, price = ? WHERE
service_name = ?";
        $stmt = $conn->prepare($sql);
        // Выполнение SQL-запроса с передачей данных
        $stmt->bind_param("sds",          $new_description,          $new_price,
        $service_name);
        $stmt->execute();
    } catch (Exception $e) {
        // Обработка ошибок, если необходимо
        echo "Error: " . $e->getMessage();
    }
    // Закрывание соединения с базой данных
    closeDatabaseConnection($conn);
}
// Функция для поиска услуги по названию
function searchServices($searchTerm) {
    // Подключение к базе данных
    $conn = connectToDatabase();
    // Подготовка SQL-запроса с использованием параметров
    $sql = "SELECT * FROM services WHERE service_name LIKE ?";
    $stmt = $conn->prepare($sql);
    // Привязка параметра
    $searchTerm = "%" . $searchTerm . "%";
    $stmt->bind_param("s", $searchTerm);
    // Выполнение SQL-запроса
    $stmt->execute();
    // Получение результатов
    $result = $stmt->get_result();
    $services = $result->fetch_all(MYSQLI_ASSOC);
    // Закрывание соединения с базой данных
    closeDatabaseConnection($conn);
    return $services;
}
function deleteServicesByServiceName($service_name)
{
    // Подключение к базе данных
    $conn = connectToDatabase();
    try {

```

```

// Подготовка SQL-запроса
$sql = "DELETE FROM services WHERE service_name = ?";
$stmt = $conn->prepare($sql);
// Выполнение SQL-запроса с передачей данных
$stmt->bind_param("s", $service_name);
$stmt->execute();
// Проверяем, была ли удалена хотя бы одна запись
if ($stmt->affected_rows > 0) {
    return ["success" => true];
} else {
    return ["success" => false, "error" => "Услуга не найдена"];
}
} catch (Exception $e) {
    // Обработка ошибок, если необходимо
    return ["success" => false, "error" => "Error: " . $e->getMessage()];
} finally {
    // Закрываемое соединение с базой данных
    closeDatabaseConnection($conn);
}
};

// Поиск услуги по названию
function searchServiceByName($service_name) {
    $conn = connectToDatabase();
    $sql = "SELECT * FROM services WHERE service_name LIKE ?";
    $stmt = $conn->prepare($sql);
    $service_name_param = "%" . $service_name . "%";
    $stmt->bind_param("s", $service_name_param);
    $stmt->execute();
    $result = $stmt->get_result();
    $services = array();
    // Проверка наличия записей в результате поиска
    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            $services[] = $row;
        }
    } else {
        echo "Нет результатов поиска";
    }
}

```

```
closeDatabaseConnection($conn);  
return $services;  
}  
?>
```

Данный набор *PHP*-функций представляет собой модульную систему для взаимодействия с базой данных в контексте управления услугами в автосервисе.

Функция ``connectToDatabase`` отвечает за установление соединения с базой данных, используя параметры, предоставленные в файле `'database_config.php'`. В случае неудачного подключения, скрипт завершает выполнение, выводя сообщение об ошибке.

Функция ``closeDatabaseConnection`` служит для закрытия соединения с базой данных.

Функция ``getServices`` предназначена для извлечения данных об услугах из базы данных и возвращает массив услуг. В случае отсутствия записей, выводится сообщение о недоступности услуг.

Функция ``addService`` позволяет добавлять новую услугу в базу данных, принимая параметры, такие как название услуги, описание и цена.

Функция ``editServiceByName`` обновляет данные о существующей услуге по её названию, принимая новые значения для описания и цены.

Функция ``searchServices`` выполняет поиск услуг по заданному термину, применяя параметры к *SQL*-запросу и возвращая результат в виде массива. Если результатов нет, выводится сообщение об их отсутствии.

Функция ``deleteServicesByServiceName`` удаляет услугу из базы данных по её названию.

Функция ``searchServiceByName`` выполняет поиск услуги по частичному совпадению с названием и возвращает результаты в виде массива. Если ничего не найдено, выводится сообщение об отсутствии результатов.

Эти функции обеспечивают управление информацией об услугах в автосервисе, а асинхронные операции с базой данных выполняются в синхронном порядке благодаря использованию ключевых слов ``async`` и ``await``.

Данная коллекция функций представляет собой надежный и удобный инструмент для управления базой данных в автосервисе, где внимание к деталям и гибкость в обработке данных являются ключевыми аспектами.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При входе на сайт, перед пользователем отображается главная страница, продемонстрированная на рисунке 4.1. По середине есть кнопки для входа в систему, регистрации и просмотра отзывов. Не авторизованный пользователь может просматривать отзывы и новостную ленту. А для того, чтобы взаимодействовать с системой необходимо зарегистрироваться и зайти в систему.

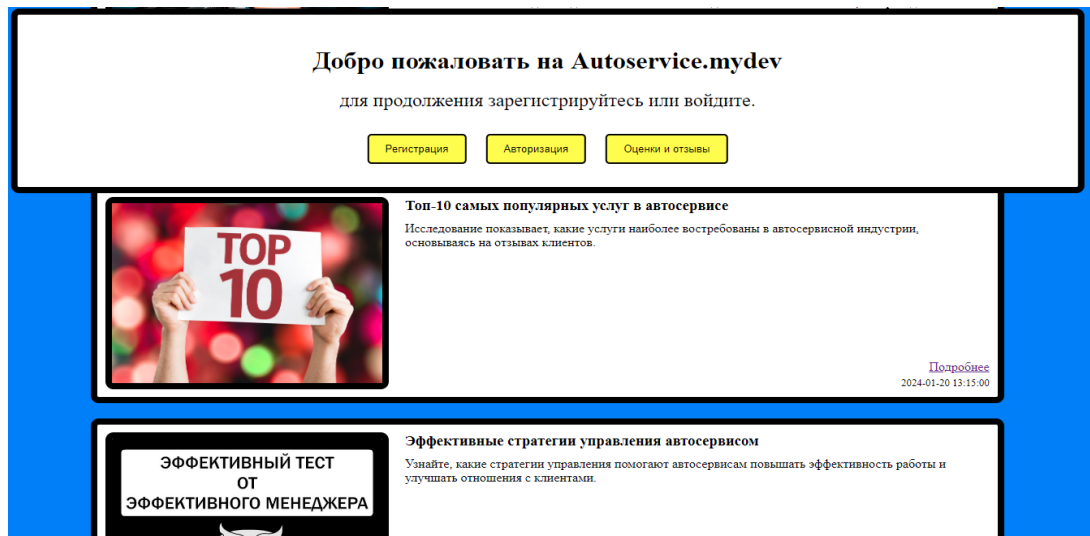


Рисунок 4.1 – Главная страница

Окно авторизации представлено на рисунке 4.2. Если пользователь еще не зарегистрирован на сайте, то ему необходимо это сделать, нажав на кнопку регистрации.

Окно регистрации продемонстрировано на рисунке 4.3.

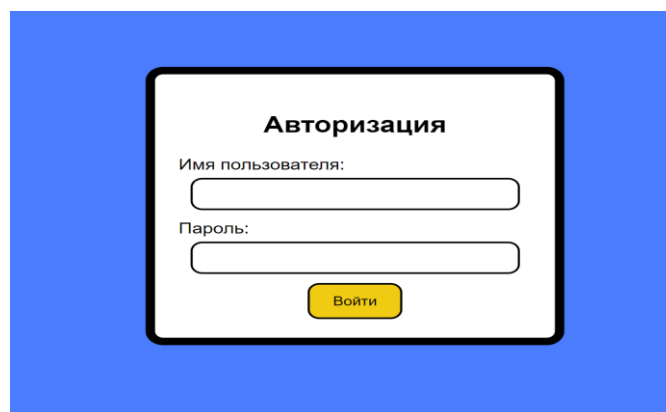


Рисунок 4.2 – Окно авторизации

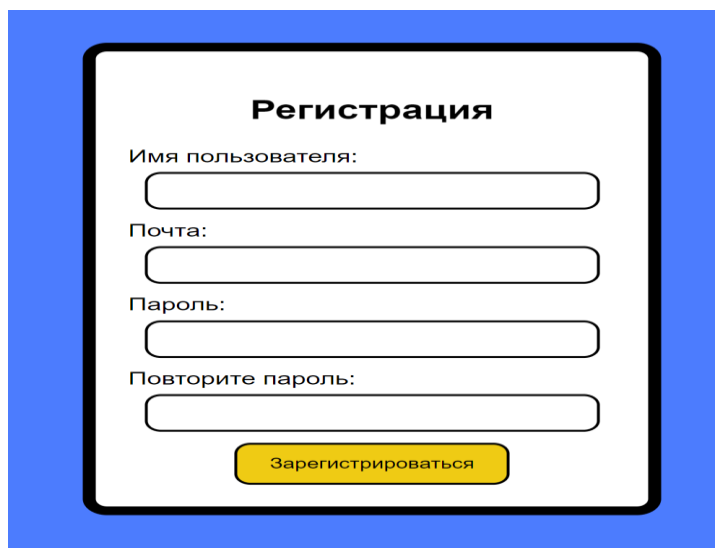
A registration form titled "Регистрация" (Registration) is displayed within a white rounded rectangle with a black border, set against a blue background. The form contains four input fields: "Имя пользователя:" (Username), "Почта:" (Email), "Пароль:" (Password), and "Повторите пароль:" (Repeat password). Below these fields is a yellow button labeled "Зарегистрироваться" (Register).

Рисунок 4.3 – Окно регистрации

После авторизации в качестве нового пользователя открывается страница, изображенная на рисунке 4.4.

Новому пользователю доступно изменение пароля в настройках, и просмотр информации о сайте. При переходе на вкладку «А что делать?» получает дополнительную инструкцию и ожидает подтверждения у администратора.

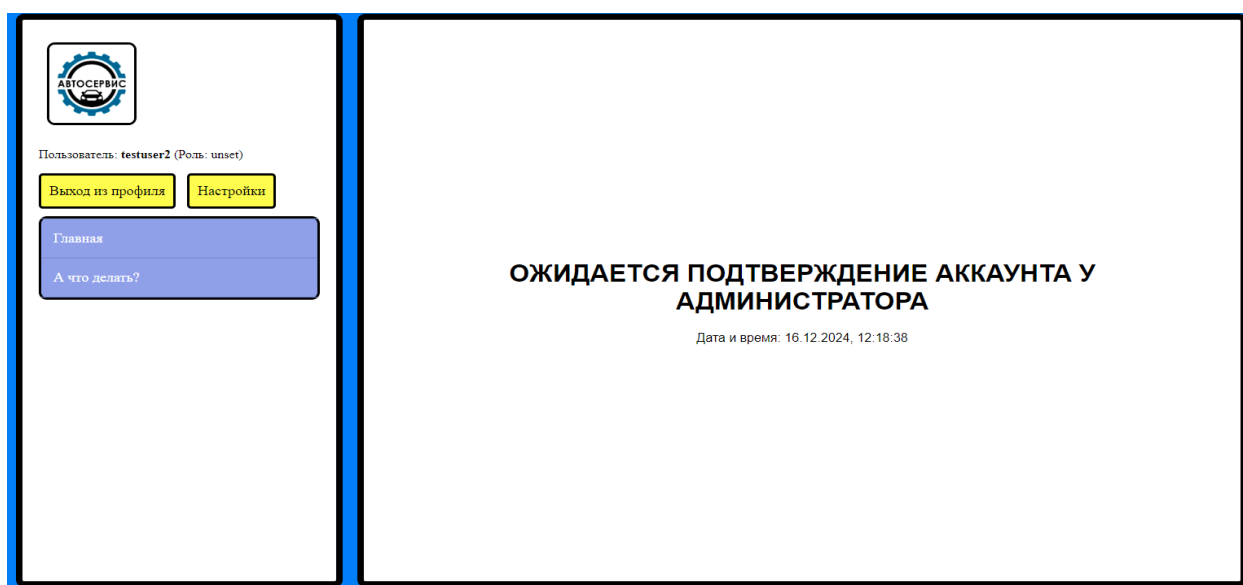
The page is divided into two main sections. The left section contains a sidebar with a logo "АВТОСЕРВИС" (Auto Service) and a user profile for "testuser2 (Роль: unset)". It includes buttons for "Выход из профиля" (Log out of profile) and "Настройки" (Settings), and a menu with "Главная" (Home) and "А что делать?" (What to do?). The right section features a large message: "ОЖИДАЕТСЯ ПОДТВЕРЖДЕНИЕ АККАУНТА У АДМИНИСТРАТОРА" (Waiting for account confirmation from the administrator), with a timestamp "Дата и время: 16.12.2024, 12:18:38" below it.

Рисунок 4.4 – Страница нового пользователя

После подтверждения аккаунта у администратора системы. Мы авторизируемся вновь, но уже с новым функционалом, активированным в

соответствии с нашей ролью. Для установки пользователям доступны роли: администратора, клиента, менеджера и механика. Рассмотрим функционал системы доступный для клиентов изображенный на рисунке 4.5. Доступен следующий функционал: просмотр каталога деталей, просмотр каталога услуг, работа с машинами в системе, текущие ремонты, оценки и отзывы, история ремонтов.

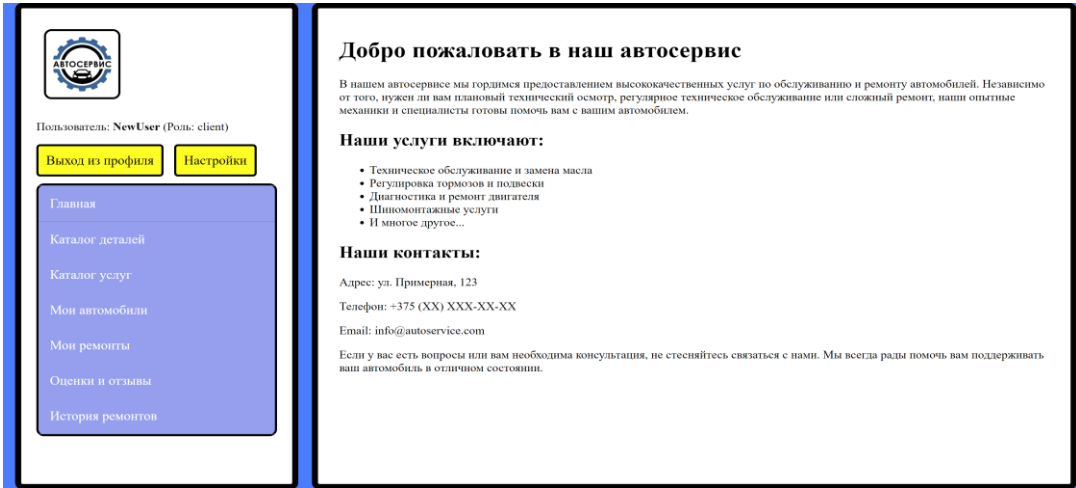


Рисунок 4.5 – Функционал для клиентов

При нажатии «Каталог деталей» появится страница с каталогом деталей. В каталоге деталей можно воспользоваться формой для поиска или сортировки. Внешний вид показан на рисунке 4.6. При нажатии «Каталог услуг» пользователь сможет просмотреть доступные услуги, предоставляемые автосервисом, а также найти информацию по интересующей его услуге используя поиск по названию. Данная страница показана на рисунке 4.7.

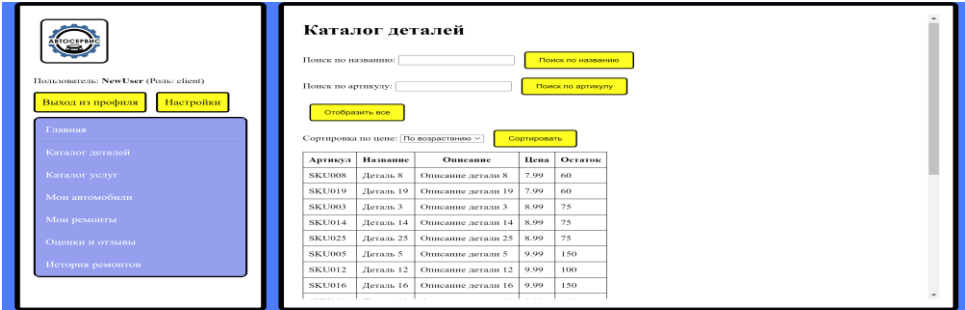


Рисунок 4.6 – Каталог деталей для пользователя

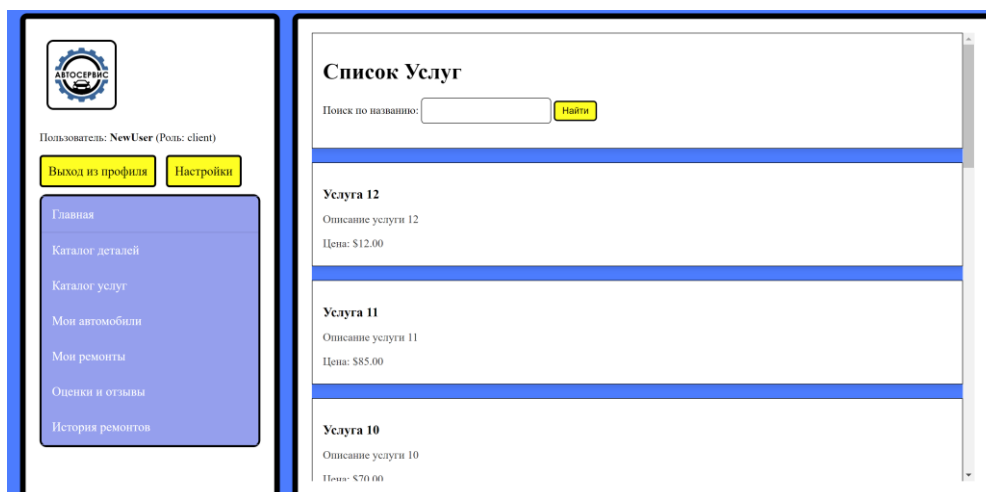


Рисунок 4.7 – Каталог услуг для пользователя

Во вкладке «Мои автомобили» отображены автомобили и есть интуитивно понятная форма для добавления нового автомобиля в систему. Данная вкладка изображена на рисунке 4.8. Так же там доступна кнопка удаления автомобиля если владелец заполнил неправильные данные или перестал являться владельцем.

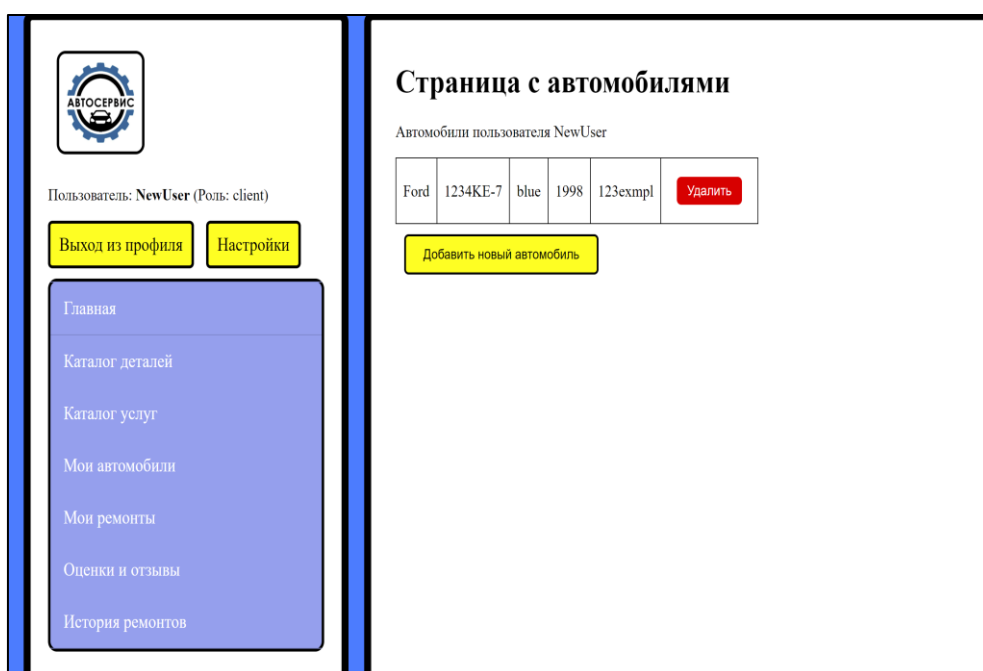


Рисунок 4.8 – Мои автомобили

Когда пользователь авторизован он так же может оставить отзыв и оценку от 1 до 5. Во вкладке «Оценки и отзывы» пользователь может

ознакомиться со всеми отзывами и по кнопке там доступна форма для заполнения отзыва. Вкладка «Оценки и отзывы» изображена на рисунке 4.9.

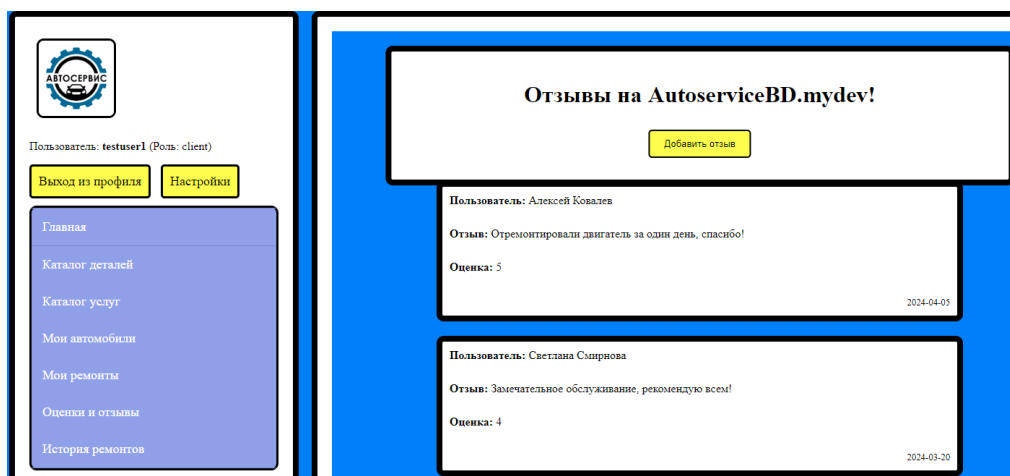


Рисунок 4.9 – Оценки и отзывы

Далее рассмотрим варианты действий от роли «Механика». Интерфейс для непосредственно работника с жезлом очень прост. Доступны следующие разделы: главная, *OBD*, заказы на ремонт, история ремонтов. Интерфейс системы изображен на рисунке 4.10.

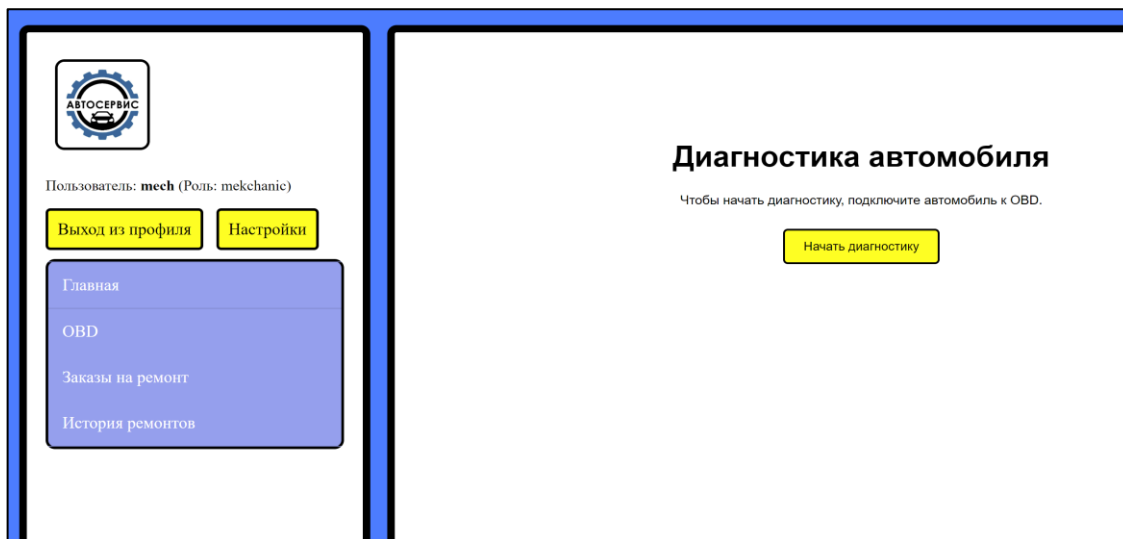


Рисунок 4.10 – Интерфейс системы
для роли «Механик»

Открыв вкладку *OBD* мы попадаем на страницу диагностики автомобиля. При нажатии на кнопку появляются диагностированные ошибки. В данном

проекте ошибки генерируются случайным образом, чтобы продемонстрировать работу системы. Процесс диагностики показан на рисунке 4.11.



Рисунок 4.11 – Процесс диагностики

Во вкладке заказы на ремонт изображенной на рисунке 4.12, пользователь с ролью механика может посмотреть предоставленные ему ремонты и принять заказ. Отдельно надо отметить, что отображаются те ремонты, в которых пользователь «Mech» отмечен в качестве механика и статус ремонта «1». Форма принятия заказа показана на рисунке 4.13.

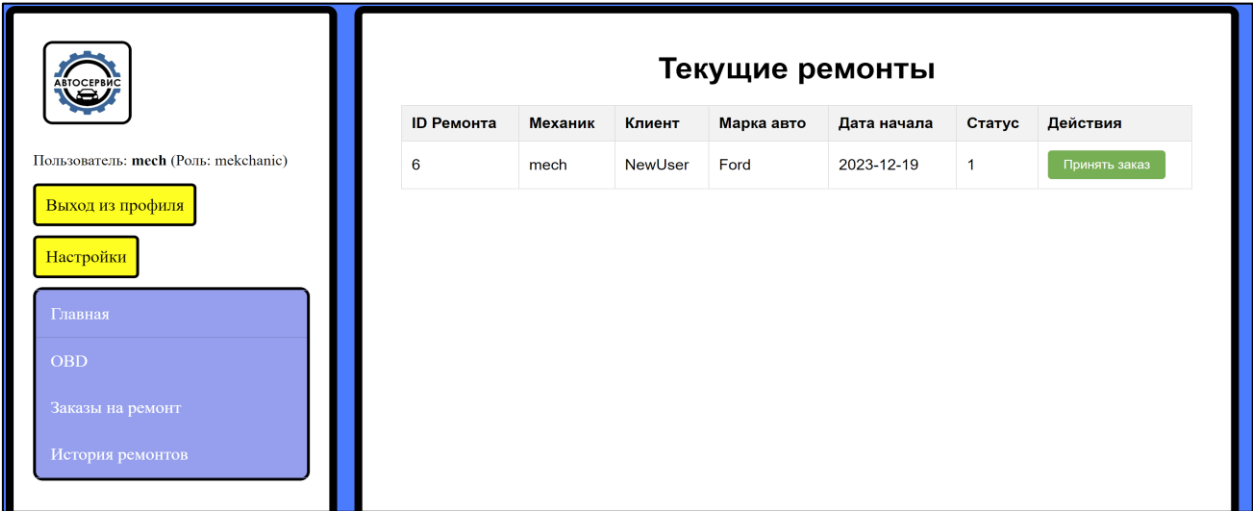


Рисунок 4.12 – Заказы на ремонт

Рисунок 4.13 – Форма принятия заказа

На странице с историей ремонтов показана страница с ремонтами которые выполнял механик с именем «*Mech*». В других ролях, стараница работает аналогичным образом и дальше рассматриваться не будет. Страница с историей изображена на рисунке 4.14.

ID Ремонта	Механик	Менеджер	Клиент	Марка авто	VIN-номер	Дата начала	Дата завершения	Стоимость	Статус
1	mech	manager	testuser1	lada	123123n342	2024-12-16	2024-12-16	650.00	3

Рисунок 4.14 – Форма принятия заказа

Рассмотрим функционал менеджера. Страница от роли «*Manager*» показана на рисунке 4.15. Менеджеру доступны следующие функции: работа с услугами, редактирование каталога, пригласить на ремонт, аналитика склада, рабта с отзывами, история ремонтов. История ремонтов работает точно так же как в выше описанном сценарии. Особенностью роли является диаграмма загруженности склада, которая располагается во вкладке аналитика склада и изображена на рисунке 4.16.

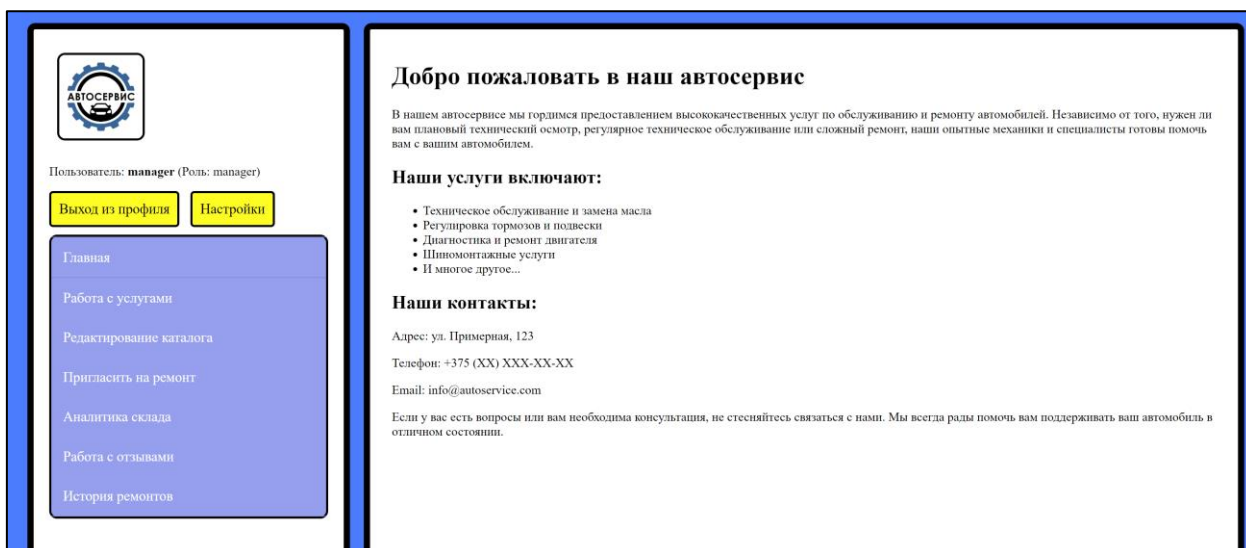


Рисунок 4.15 – Страница менеджера

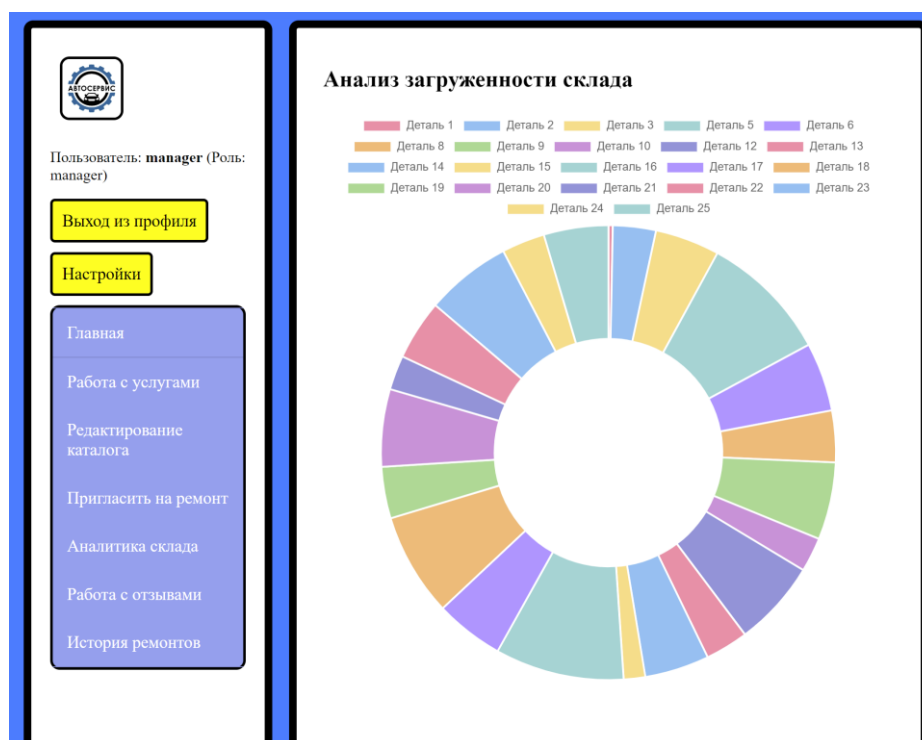


Рисунок 4.16 – Анализ загрузки склада

Рассмотрим интерфейс работы с услугами, показанный на рисунке 4.17. Страница представляет собой список с окном поиска и кнопками добавления новых услуг. Форма добавления услуг содержит проверки на ввод числовых значений. Форма добавления услуг на рисунке 4.18.

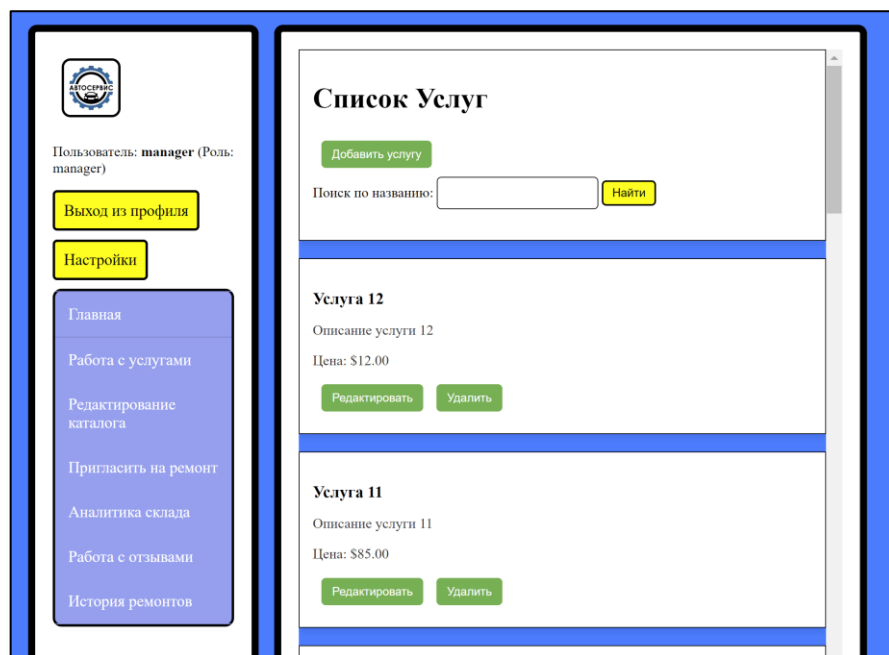


Рисунок 4.17 – Интерфейс работы с услугами

Добавить новую услугу

[Назад к списку услуг](#)

Название услуги:

Описание:

/

Цена:

Добавить услугу

Рисунок 4.18 – Форма добавления услуг

На старанице редактирование каталога доступно управление каталогом. Доступны сортировки и поиск. Так же можно удлить или редактиовать

существующий товар. Интерфейс адаптирован под управление менеджером, из-за этого он значительно отличается от клиентской версии.

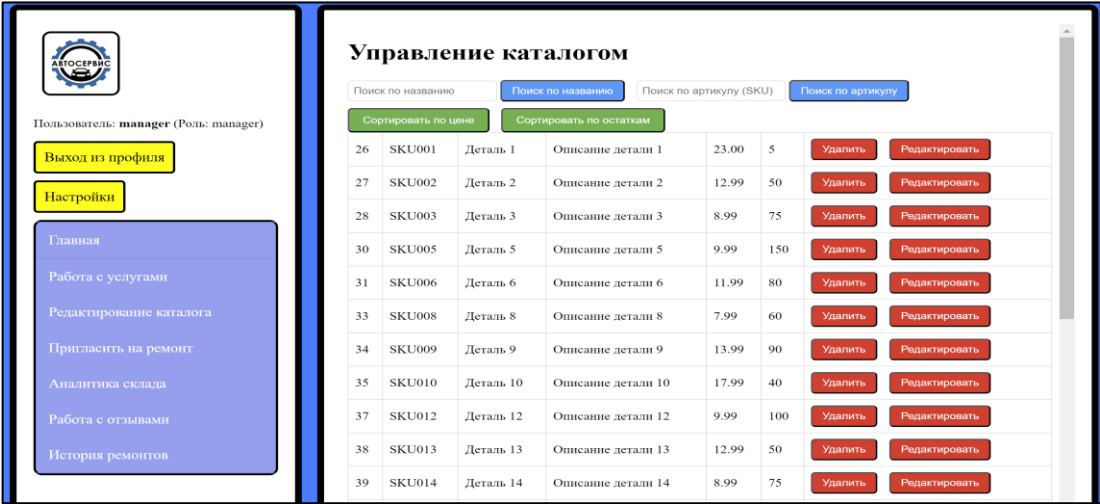


Рисунок 4.19 – Форма добавления услуг

Вкладка работа с отзывами содержит в себе страницу с отзывами и формой как добавления, так и удаления отзывов конкретного пользователя. Вкладка отображена на рисунке 4.20.

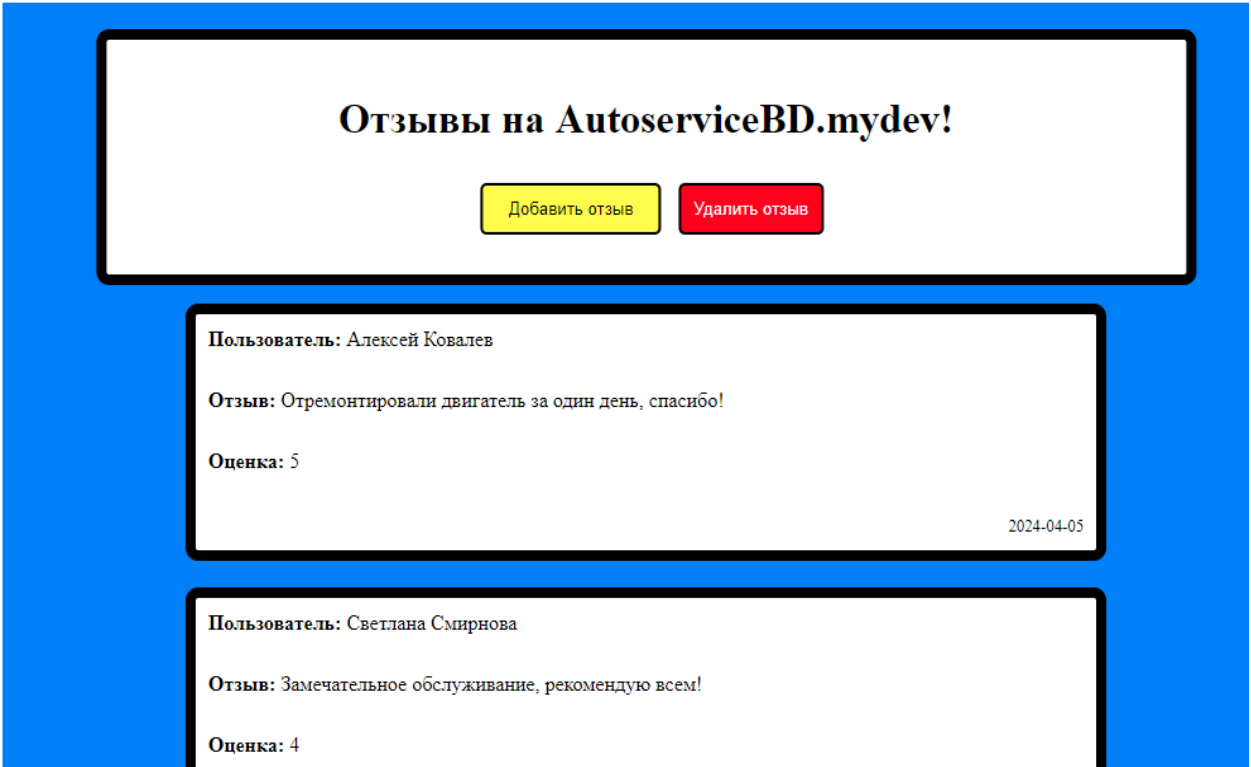


Рисунок 4.20 – Работа с отзывами

Во вкладке «Пригласить на ремонт» менеджер может реализовать сценарии пригласения пользователя на ремонт и подтверждать заказы. В интуитивно понятной форме для создания ремонта выбирается клиент и механик. Далее заказ должен проходить подтверждения в аккаунтах у каждого участника цепочки менеджер-клиент-механик-клиент-менеджер. На этапе подтверждения выбираются заказы со статусом 3, статус 3 означает что заказ исполнен и принят заказчиком. В цепочке клиент выбирает нужный для ремонта автомобиль и принимает работу. Механик исполняет ремонт и выставляет счет для дальнейшего подтверждения. Данная вкладка изображена на рисунке 4.21.

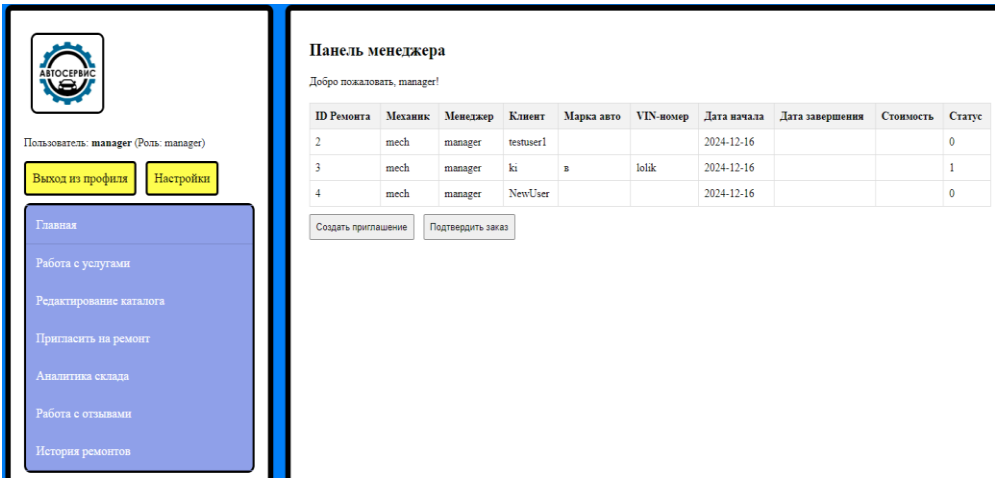


Рисунок 4.21 – Панель менеджера

Далее рассмотрим функционал для администратора сайта. Функционал изображен на рисунке 4.22. Страница администратора имеет такие опции, как «Работа с услугами», «Установка ролей», «Удаление аккаунтов», «Экспорт/импорт ошибок в систему», «Работа с отзывами», «История ремонтов» и «Текущие ремонты».

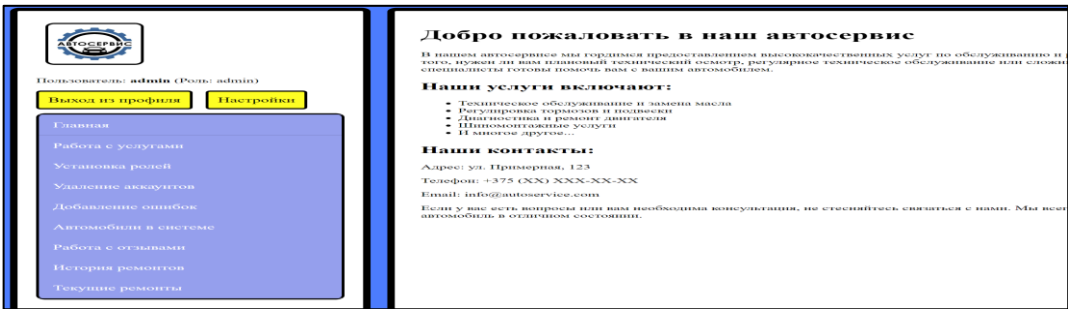


Рисунок 4.22 – Страница администратора

При выборе «Работа с услугами» и «Работа с отзывами» откроются страницы аналогичные по функционалу страниц менеджера. «История ремонтов» покажет все завершённые ремонты автосервиса и имеет кнопку «Удалить все».

Рассмотрим установку ролей. Администратор может установить роль найдя с помощью поиска нужных пользователей. Доступные для установки роли: администратор, механик, менеджер и клиент. Стараница изображена на рисунке 4.23.

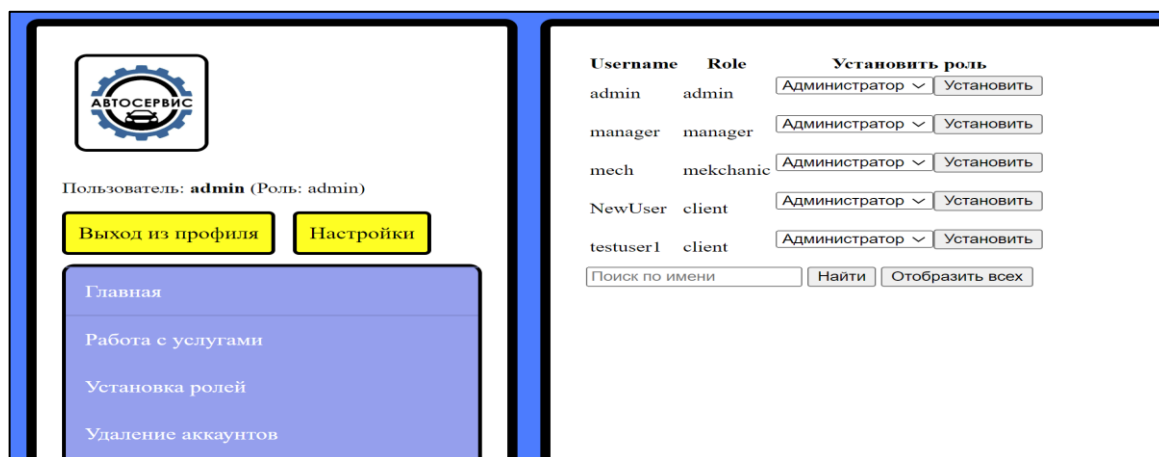


Рисунок 4.23 – Установка ролей

Рассмотрим удаление аккаунтов. Перейдя на страницу, мы получим список пользователей с поиском и возможностью удалить интересующий нас аккаунт. Удаление аккаунтов изображено на рисунке 4.24.

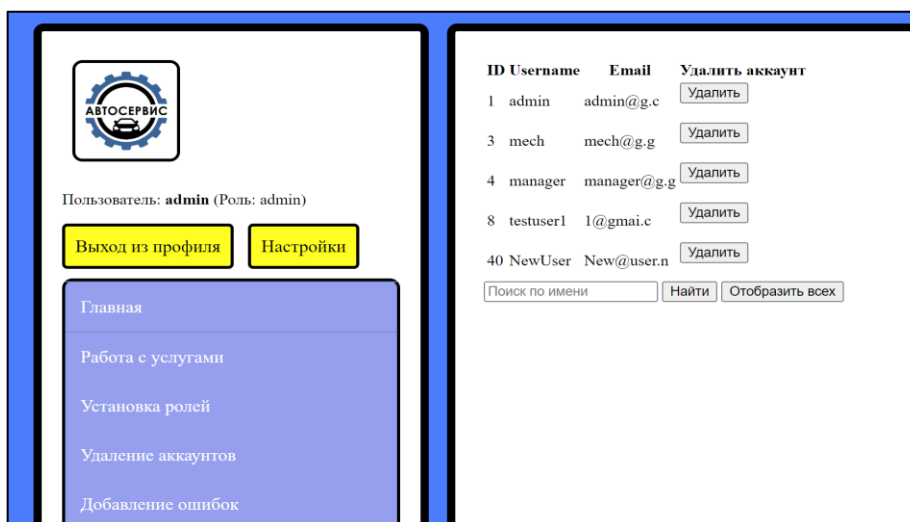



Рисунок 4.24 – Удаление аккаунтов

Добавление ошибок в систему представляет собой загрузку файла формата *Xlsx*. Файл с ошибками изображен на рисунке 4.25.

	A	B	C	D	E	F	G	H	I
1	Код ошибки	Название ошибки	Описание ошибки						
2	P0016	Crankshaft Position - Camshaft Position Correlation (Bank 1 Sensor A)							
3	P0171	System Too Lean (Bank 1)							
4	P0300	Random/Multiple Cylinder Misfire Detected							
5	P0420	Catalyst System Efficiency Below Threshold (Bank 1)							
6	P0455	Evaporative Emission System Leak Detected (Large Le	The powertrain control module (PCM) has detected a large leak in the evaporative emission control system.						
7	P0562	System Voltage Low	The powertrain control module (PCM) has detected that the system voltage is below the specified threshold.						
8	P0700	Transmission Control System Malfunction	The powertrain control module (PCM) has detected a malfunction in the transmission control system.						
9	P1128	Closed Loop Fueling Not Achieved - Bank 1	The powertrain control module (PCM) has detected that the fuel system is not operating in closed loop mode for bank 1.						
10	P1131	Lack of Upstream Heated Oxygen Sensor Switch - Ser	The powertrain control module (PCM) has detected that the upstream heated oxygen sensor is indicating a lean air/fuel ratio.						
11	P1234	Turbocharger Boost Control Position Exceeded Learn	The powertrain control module (PCM) has detected that the turbocharger boost control position has exceeded the learned range.						
12	P1400	DPFE Sensor Circuit Low Voltage Detected	The powertrain control module (PCM) has detected a low voltage condition in the DPFE (differential pressure feedback) sensor circuit.						
13	P1443	Evaporative Emission Control System Control Valve	The powertrain control module (PCM) has detected a malfunction in the evaporative emission control system control valve.						
14	P1518	Intake Manifold Runner Control (IMRC) Circuit Open	The powertrain control module (PCM) has detected an open circuit in the intake manifold runner control (IMRC) circuit.						
15	P1633	Keep Alive Power Voltage Too Low	The powertrain control module (PCM) has detected that the keep alive power voltage is too low.						
16	P1780	Transmission Control Switch Out of Self-Test Range	The powertrain control module (PCM) has detected that the transmission control switch is out of the self-test range.						
17	P2004	Intake Manifold Runner Control (IMRC) Stuck Open	The powertrain control module (PCM) has detected that the intake manifold runner control (IMRC) is stuck open for bank 1.						
18	P2104	Throttle Actuator Control System - Forced Idle	The powertrain control module (PCM) has detected a forced idle condition in the throttle actuator control system.						
19	P2135	Throttle/Pedal Position Sensor/Switch A/B Voltage Correlation	The powertrain control module (PCM) has detected a voltage correlation problem between the throttle position sensor and the pedal position sensor.						
20	P2195	O2 Sensor Signal Stuck Lean (Bank 1 Sensor 1)	The powertrain control module (PCM) has detected that the oxygen sensor signal for bank 1 sensor 1 is stuck lean.						
21	P2251	O2 Sensor Negative Current Control Circuit/Open (Bank 1)	The powertrain control module (PCM) has detected an open circuit or a fault in the negative current control circuit for bank 1.						
22	P2299	Brake Pedal Position/Accelerator Pedal Position Incompatible	The powertrain control module (PCM) has detected an incompatible relationship between the brake pedal position and the accelerator pedal position.						
23	P2402	EVAP Leak Detection Pump Control Circuit High	The powertrain control module (PCM) has detected a high voltage condition in the EVAP leak detection pump control circuit.						
24	P2646	A Rocker Arm Actuator System Performance or Stuck Off	The powertrain control module (PCM) has detected a performance problem or a stuck off condition in the rocker arm actuator system.						
25	P2703	Transmission Friction Element D Apply Time Range/Performance	The powertrain control module (PCM) has detected a problem with the apply time range or performance of the transmission friction element D.						
26	U0100	Lost Communication with ECM/PCM "A"	The powertrain control module (PCM) has lost communication with the engine control module (ECM) or powertrain control module (PCM) "A".						
27									

Рисунок 4.25 – Файл с ошибками

Страница «Автомобили в системе» имеет поиск по имени и *Vin*-номеру кузова и возможность удаления. Так же можно отсортировать по году выпуска. Страница изображена на рисунке 4.26.



Пользователь: **admin** (Роль: admin)

[Выход из профиля](#) [Настройки](#)

- Главная
- Работа с услугами
- Установка ролей
- Удаление аккаунтов
- Добавление ошибок
- Автомобили в системе
- Работа с отзывами
- История ремонтов
- Текущие ремонты

Страница с автомобилями

Поиск по VIN номеру [Искать](#)
 Поиск по имени пользователя [Искать](#)
[Сортировать по году \(возрастание\)](#)

[Сортировать по году \(убывание\)](#)
[Отобразить все](#)

Все автомобили пользователей

Имя	Марка	Модель	Цвет	Год	VIN	Действия
testuser1	audi	1232AE-4	black	1999	111n323	Удалить
testuser1	lada	4992OO-5	gray	2004	123123n342	Удалить
ik	ford	1232PO-7	green	2010	23234mk343	Удалить
NewUser	Ford	1234KE-7	blue	1998	434nn32	Удалить

Рисунок 4.26 – Автомобили в системе

Страница «Текущие ремонты» отобразит администратору все текущие ремонты вне зависимости от их статуса в текущий момент времени. Таблица

имеет цветовые обозначения в зависимости от статуса выполнения. Администратор имеет возможность только просматривать данную страницу. Страница текущие ремонты изображена на рисунке 4.27.

Текущие ремонты									
ID Ремонта	Механик	Менеджер	Клиент	Марка авто	VIN-номер	Дата начала	Дата завершения	Стоимость	Статус
2	mech	manager	testuser1			2024-12-16			0
3	mech	manager	ki	в	lolik	2024-12-16			1
4	mech	manager	NewUser	Ford	434nn32	2024-12-16	2024-12-16	2323.00	2

Рисунок 4.27 – Текущие ремонты

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было разработано веб-приложение для управления услугами автомобильной мастерской. В отчете подробно описаны ключевые этапы процесса разработки, включая проектирование интерфейса, определение функциональности, выбор используемых технологий и инструментов, создание базы данных и серверной части, а также реализация клиентской части.

Также рассмотрена проблематика, важность и актуальность такого типа систем. В рамках проектирования системы определены требования к функциональности и производительности. Для визуализации и описания архитектуры системы построена логическая схема базы данных, *ER*-диаграмма в нотации Питера Чена и диаграмма вариантов использования.

Разработанная система позволяет обеспечить оптимальное взаимодействие между участниками рабочих процессов в автосервисе. Далее будут описаны основные возможности для каждого из участников.

Администратор обладает возможностью просматривать рабочие процессы с удобным красочным выводом в таблицы. Отдельно следует отметить механизм импорта и экспорта информации списка с ошибками автомобиля, используемого при диагностике.

Менеджер автосервиса имеет возможность через приложение пригласить клиентов на обслуживание, а также управлять складом, используя круговые диаграммы.

Механики автосервиса могут подключить к системе *OBD*-сканер (*on-board diagnostic*) и получить достоверную информацию по ошибкам в автомобиле.

Со стороны клиента доступен просмотр каталогов услуг и запчастей, а также доступны отзывы. В каталогах реализованы алгоритмы поиска и сортировки в соответствии с определенными запросами, такими как название, цены или остатки на складе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Разработка функциональных веб-приложений [Электронный ресурс]. – Режим доступа: <https://web-help.com>
- [2] Секреты HTML CSS [Электронный ресурс]. – Режим доступа: <https://skysmart.ru/articles/programming/secretu-HTML-CSS>
- [3] Что такое нормализация данных? [Электронный ресурс]. – Режим доступа: <https://appmaster.io/ru/blog/chto-takoe-normalizatsiia-dannykh/>
- [4] Основы проектирования баз данных [Электронный ресурс]. – Режим доступа: <https://appmaster.io/ru/blog/osnovy-proektirovaniia-baz-dannykh>
- [5] Шилдт Г. *Java*. Полное руководство. 10-е издание/Шилдт г. Издательство Вильямс, 2019. – 1488 с.
- [6] Лерен, Р. PHP и MySQL. Разработка веб-приложений / Р. Лерен. – Москва: ДМК Пресс, 2019. – 704 с.
- [7] Пратт, М. SQL и реляционные базы данных: теория и практика / М. Пратт. – Москва: ДМК Пресс, 2020. – 416 с
- [8] Шилдт Г. *Java*. Полное руководство. 10-е издание/Шилдт г. Издательство Вильямс, 2019. – 1488 с
- [9] Основы проектирования баз данных [Электронный ресурс]. – Режим доступа: <https://appmaster.io/ru/blog/osnovy-proektirovaniia-baz-dannykh>
- [10] СТП – 01-2017 – Стандарт предприятия. Дипломные проекты (работы). Общие требования – Минск, БГУИР. – 174 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Профиль пользователя</title>
  <link rel="stylesheet" href="styles/profile.css">
</head>

<body>
  <div class="container">
    <div class="profile-container">
      <div class="profile-info">
        
        <p class="login-caption">Пользователь: <strong id="username"></strong> (Роль: <span
id="role"></span>)</p>
        <!-- Добавленные кнопки -->
        <div class="profile-buttons">
          <a href="index.php" class="profile-button">Выход из профиля</a>
          <a href="upload_avatar.php" class="profile-button">Настройку</a>
        </div>
      </div>
      <div class="menu">
        <ul id="menu-list">
          <li id="adminLink" style="display: none;"></li>
          <li id="managerLink" style="display: none;"></li>
          <li id="mechanicLink" style="display: none;"></li>
          <li id="userLink" style="display: none;"></li>
          <li><a onclick="loadPage('page1.php', 'Главная')">Главная</a></li>
          <!-- Добавленные ссылки в зависимости от роли -->
        </ul>
      </div>
    </div>
    <div class="page-container">
      <!-- Здесь будет отображаться содержимое страниц, выбранных в меню -->
    </div>
  </div>
</body>
</html>
```



```

<iframe id="iframeContainer" width="100%" height="100%" frameborder="0"></iframe>
</div>
</div>
<script>
document.addEventListener("DOMContentLoaded", function() {
    // Получаем имя пользователя из localStorage
    var storedUsername = localStorage.getItem("username");
    // Получаем роль пользователя из localStorage
    var storedRole = localStorage.getItem("role");
    // Обновляем роль пользователя на странице
    if (storedRole) {
        document.getElementById("role").innerText = storedRole;

        // Проверяем роль пользователя и отображаем соответствующие ссылки в меню
        if (storedRole === "admin") {
            document.getElementById("adminLink").style.display = "block";
            addLinksToMenu("admin", [
                { url: "uslugi_view.php", name: "Работа с услугами" },
                { url: "role/role_main.php", name: "Установка ролей" },
                { url: "admin_fun/deleteUserAdminMain.php", name: "Удаление аккаунтов" },
                { url: "exel_exp_err.php", name: "Добавление ошибок" },
                { url: "car_admin_view.php", name: "Автомобили в системе" },
                { url: "otzivi_main.php", name: "Работа с отзывами" },
                { url: "otobr_ist_admins.php", name: "История ремонтов" },
                { url: "tekukhie_rem_admink.php", name: "Текущие ремонты" }
            ]); // Добавляем ссылки для админа
        } else if (storedRole === "client") {
            document.getElementById("managerLink").style.display = "block";
            addLinksToMenu("manager", [
                { url: "katalog_user.php", name: "Каталог деталей" },
                { url: "uslugi_view.php", name: "Каталог услуг" },
                { url: "car_user_view.php", name: "Мои автомобили" },
                { url: "remont_order_client.php", name: "Мои ремонты" },
                { url: "otzivi_main.php", name: "Оценки и отзывы" },
                { url: "otobr_ist_client.php", name: "История ремонтов" }
            ]); // Добавляем ссылки для менеджера
        }
        else if (storedRole === "manager") {
            document.getElementById("managerLink").style.display = "block";

```

```

addLinksToMenu("manager", [
    { url: "uslugi_view.php", name: "Работа с услугами" },
    { url: "manager_catalog.php", name: "Редактирование каталога" },
    { url: "manager_dashboard.php", name: "Пригласить на ремонт" },
    { url: "analitica_main.php", name: "Аналитика склада" },
    { url: "otzivi_main.php", name: "Работа с отзывами" },
    { url: "otobr_ist_manager.php", name: "История ремонтов" }
]); // Добавляем ссылки для менеджера
} else if (storedRole === "mekchanic") {
    document.getElementById("mechanicLink").style.display = "block";
    addLinksToMenu("mechanic", [
        { url: "OBD_scannerMain.php", name: "OBD" },
        { url: "remont_order_mech.php", name: "Заказы на ремонт" },
        { url: "otobr_ist_mech.php", name: "История ремонтов" }
    ]); // Добавляем ссылки для механика
} else {
    document.getElementById("userLink").style.display = "block";
    addLinksToMenu("user", [
        { url: "unset.php", name: "А что делать?" }
    ]); // Добавляем ссылки для обычного пользователя
}
}

// Функция для добавления ссылок в меню ``html
// Функция для добавления ссылок в меню
function addLinksToMenu(role, links) {
    var menuList = document.getElementById("menu-list");
    var roleLink = document.createElement("li");
    roleLink.id = role + "Link";

    for (var i = 0; i < links.length; i++) {
        var link = document.createElement("a");
        link.href = "javascript:void(0);";
        link.textContent = links[i].name;
        link.setAttribute("onclick", "loadPage('\" + links[i].url + \"', '\" + links[i].name + \"')");
        roleLink.appendChild(link);
    }
    menuList.appendChild(roleLink);
}

```

```

// Загрузка имени пользователя из localStorage
if (storedUsername) {
    document.getElementById("username").innerText = storedUsername;
}
});

// Загрузка страницы в iframe
function loadPage(url, pageTitle) {
    var iframeContainer = document.getElementById("iframeContainer");
    iframeContainer.src = url;
    iframeContainer.onload = function() {
        document.title = pageTitle;
    };
}

</script>
</body>

</html>
<?php
// Подключение к базе данных (замените значения переменных на ваши)
require_once('database_config.php');

$conn = new mysqli($servername, $username, $password, $dbname);

// Проверка соединения
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Получение значения username из URL
$username = $_GET['username'];

// SQL-запрос для получения данных из базы данных
$sql = "SELECT id_repair FROM current_repairs WHERE status = 2 AND client_name = '$username'";
$result = $conn->query($sql);
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
  body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
  }
  form {
    max-width: 400px;
    margin: 50px auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  }
  label {
    display: block;
    margin-bottom: 8px;
  }
  select {
    width: 100%;
    padding: 8px;
    margin-bottom: 16px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
  }
  input[type="submit"] {
    background-color: #4caf50;
    color: #fff;
    padding: 10px 15px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
  }

  input[type="submit"]:hover {
    background-color: #45a049;
  }
</style>

```

```

</head>
<body>
    <form action="repairs_fin_proc_cl.php" method="post"> <!-- Укажите обработчик формы -->
        <label for="repair_id">Выберите ремонт:</label>
        <select name="repair_id" id="repair_id">
            <?php
                // Вывод данных из SQL-запроса в выпадающий список
                while ($row = $result->fetch_assoc()) {
                    echo '<option value="' . $row['id_repair'] . '">' . $row['id_repair'] . '</option>';
                }
            ?>
        </select>
        <input type="submit" value="Подтвердить">
    </form>
</body>
</html>
<?php
    // Закрытие соединения с базой данных
    $conn->close();
?>

```

