

Федеральное государственное образовательное бюджетное
учреждение высшего образования
**«Финансовый университет
при Правительстве Российской Федерации»
(Финансовый университет)**

Кафедра информационных технологий
Факультет информационных технологий и анализа больших данных

Пояснительная записка к курсовой работе по дисциплине
«Современные технологии программирования»
на тему:

**«Информационно-справочная система для управления автосервисом:
Создание системы для учёта клиентов, управления запчастями и
планирования ремонтов»**

Выполнил:
Студент группы ПИ22-1в
Ерхан М. И.

Научный руководитель:
к.т.н., доцент
Журавлева М. Г.

Москва – 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ УПРАВЛЕНИЯ АВТОСЕРВИСОМ	5
1.1 Описание предметной области	5
1.2 Принципы работы систем с клиент-серверной архитектурой	6
1.3 Средства разработки	7
1.4 Вывод по первой главе	9
ГЛАВА 2. РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ АВТОСЕРВИСОМ	10
2.1 Алгоритмические решения	10
2.2 Описание интерфейса программы	10
2.3. Архитектура приложения	19
2.4 Схема взаимодействия с системой	19
2.5 Используемые технологии	20
2.6 Разработка системы	20
2.6.1 Контроллеры	20
2.6.2 Сервисы	20
2.6.4 Модели данных	21
2.7 База данных	22
2.8 Безопасность системы	23
2.8.1 Настройка шифрования паролей	23
2.8.2 Настройка менеджера аутентификации	24
2.8.3 Обработка доступа и ошибок	25
2.8.4 Конфигурация фильтров безопасности	25
2.8.5 Общие преимущества и безопасность	26
2.8.6 Настройки приложения	27
ЗАКЛЮЧЕНИЕ	29
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	32
ПРИЛОЖЕНИЕ	33

ВВЕДЕНИЕ

Информационные технологии в настоящее время являются неотъемлемой частью бизнес-процессов, помогая компаниям различных отраслей добиваться конкурентных преимуществ за счет оптимизации и автоматизации своих операций. В автомобильной сфере, особенно в сфере автосервисов, растет потребность в удобных и гибких информационных системах, которые могут поддерживать учет клиентов, управление запасными частями и планирование ремонтных работ. Применение таких систем позволяет сократить время на обработку данных, минимизировать ошибки при планировании и увеличивать скорость обслуживания клиентов. Традиционные методы ведения учета на бумажных носителях и в таблицах не только отнимают много времени, но и усложняют контроль за запасами и планированием работ, что в конечном итоге негативно сказывается на качестве обслуживания и уровне удовлетворенности клиентов. Таким образом, создание информационно-справочной системы для управления автосервисом является актуальной и востребованной задачей.

Целью данной курсовой работы является разработка прототипа информационно-справочной системы, позволяющий сотрудникам автосервиса организовать удобное управление клиентами, запасными частями и процессами технического обслуживания. Разрабатываемая система повысит прозрачность и структурированность процессов автосервиса, улучшит взаимодействие с клиентами и оптимизирует использование ресурсов. В системе предполагается создание удобного интерфейса для сотрудников, который позволит без большого труда добавлять и редактировать данные о клиентах, отслеживать наличие запчастей, а также управлять графиком ремонтных работ.

Для достижения поставленной цели будет выполнено несколько задач:

1. Анализ предметной области и выявление потребностей автосервисов в автоматизации.

2. Проектирование структуры базы данных для хранения информации о клиентах, заказах, запчастях и ремонтах.
3. Разработка функционала для добавления, редактирования и удаления данных о клиентах.
4. Создание модуля управления запасами автосервиса с возможностью учёта остатков и создания отчётов.
5. Реализация интерфейса для записи клиентов на обслуживание и распределения нагрузки специалистов.
6. Настройка системы безопасности и авторизации для защиты данных.

Объектом исследования является информационно-справочная система для управления автосервисом, включающая функционал для учёта клиентов, управления складом запчастей и планирования ремонтов.

Предметом исследования данной курсовой работы являются методы создания информационно-справочной системы для управления автосервисом, включающая учёт клиентов, управление запасными частями и планирование ремонтных работ. Особое внимание уделяется разработке программной архитектуры, обеспечивающей взаимодействие между серверной и клиентской частями, а также созданию пользовательского интерфейса для удобного взаимодействия с системой.

В процессе разработки будут рассматриваться ключевые аспекты клиент-серверного взаимодействия, организация хранения данных в реляционной базе данных PostgreSQL, а также применение современных технологий веб-разработки, таких как Spring Boot, HTML, CSS, Java и JavaScript.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ УПРАВЛЕНИЯ АВТОСЕРВИСОМ

1.1 Описание предметной области

Автосервис – это предприятие, предоставляющее услуги по диагностике, техническому обслуживанию и ремонту автомобилей. Для эффективной работы и конкурентоспособности необходимо, чтобы его бизнес-процессы были оптимизированы и организованы на высоком уровне. В условиях возрастающей конкуренции информационные технологии играют важную роль в упрощении управления, оптимизации работы с клиентами и улучшении общего качества обслуживания.

Основные аспекты предметной области включают:

1. **Учёт клиентов:** автосервису необходимо хранить информацию о клиентах, которая может включать их контактные данные, историю ремонтов, предпочтительные услуги и особенности автомобилей. Ведение такой информации позволяет обеспечить индивидуальный подход к каждому клиенту и упростить запись на обслуживание.
2. **Управление запчастями:** автосервис должен поддерживать необходимый запас запчастей или возможность быстрой доставки той или иной запчасти для бесперебойной и оперативной работы. Это требует регулярного учёта, отслеживания остатков и планирования поставок. В системе важно иметь функционал, позволяющий сотрудникам быстро найти необходимую деталь, видеть их количество на складе и отслеживать историю заказов.
3. **Планирование ремонтных работ:** для эффективного управления загрузкой сотрудников автосервиса необходимо вести учет записей на обслуживание и планировать распределение задач. Это позволяет оптимизировать рабочее время специалистов, избегать накладок в графике и своевременно выполнять все запланированные работы.

Внедрение информационной системы позволит автоматизировать эти аспекты, создавая единую цифровую платформу для хранения данных о клиентах,

управления складом запчастей и ведения графика ремонтов. Это облегчит доступ к необходимой информации, повысит точность учета и улучшит взаимодействие между сотрудниками и клиентами.

1.2 Принципы работы систем с клиент-серверной архитектурой

Клиент-серверная архитектура — один из наиболее популярных и широко используемых подходов для построения информационных систем. В данном подходе система делится на два основных компонента: сервер, отвечающий за хранение данных и обработку запросов, и клиент, представляющий интерфейс для взаимодействия с пользователем. В рамках данной курсовой работы разработка информационно-справочной системы для автосервиса будет базироваться именно на клиент-серверной архитектуре, что обеспечит гибкость, масштабируемость и надежность решения.

Основные принципы клиент-серверной архитектуры:

1. **Разделение ролей:** клиент-серверная архитектура четко разделяет задачи между клиентом и сервером. Сервер отвечает за обработку данных, хранение информации в базе данных, выполнение бизнес-логики и предоставление необходимых данных по запросам клиента. Клиент же представляет собой пользовательский интерфейс, через который происходит взаимодействие с системой. Такой подход упрощает поддержку и модификацию системы, так как позволяет изменять логику на стороне сервера, не затрагивая клиентскую часть, и наоборот.
2. **Передача данных по сети:** обмен данными между клиентом и сервером осуществляется через сеть. В данном проекте планируется использовать REST API — архитектурный стиль для взаимодействия между клиентом и сервером с использованием HTTP-запросов.
3. **Многоуровневость и масштабируемость:** клиент-серверные системы легко масштабируются, так как сервер может обслуживать множество клиентов одновременно. Если нагрузка на сервер возрастает, его можно

масштабировать, например, добавляя новые серверы в систему или распределяя нагрузку. В случае информационно-справочной системы для автосервиса это значит, что система может поддерживать множество одновременных пользователей, таких как администраторы, менеджеры и технические специалисты.

4. **Безопасность и авторизация:** в клиент-серверной архитектуре важным аспектом является обеспечение безопасности данных, особенно в случае систем, работающих с личной и конфиденциальной информацией клиентов. Сервер реализует механизмы аутентификации и авторизации, что позволяет контролировать доступ к данным. В рамках данной работы планируется реализовать авторизацию на стороне сервера, используя стандартные средства Spring Security для защиты информации о клиентах и доступе к записям на обслуживание.
5. **Упрощенное обновление и поддержка:** благодаря разделению на клиентскую и серверную части, обновления и исправления ошибок можно вносить на сервере без необходимости обновлять клиентскую часть. Это особенно важно для систем, которые работают в режиме реального времени и требуют высокой доступности.

Клиент-серверная архитектура предоставляет возможность построения надежной и гибкой системы, способной эффективно поддерживать бизнес-процессы автосервиса. Разработка такой системы с использованием Spring Boot для серверной части и HTML, CSS, JavaScript для клиентской позволит обеспечить удобный и безопасный доступ к данным, необходимый для управления автосервисом.

1.3 Средства разработки

При разработке информационной системы будут использованы современные технологии, который позволяют эффективно реализовать функционал для веб-приложений. Основным языком программирования – Java. Он

предоставляет стабильную платформу для построения корпоративных приложений и совместим с большинством реляционных баз данных. Для создания серверной части будет использоваться Spring Boot – фреймворк для быстрого и удобного создания приложений на основе Java, который обеспечивает простую интеграцию с базой данных и поддержку различных модулей для работы с безопасностью, авторизацией и ORM. С помощью Spring Boot можно легко реализовать REST API и подключение к базе данных.

```
@RestController
@RequestMapping("/clients")
public class ClientController {

    @GetMapping("/{id}")
    public String getClientById(@PathVariable Long id) {
        return "Информация о клиенте с ID " + id;
    }

    @PostMapping
    public String addClient(@RequestBody String clientData) {
        return "Клиент успешно добавлен!";
    }
}
```

Рисунок 1 - Пример использования Spring Boot и REST API

ORM используется для эффективного взаимодействия с базой данных MySQL, это позволит легко управлять записями, клиентами, запчастями и ремонтами. Мы будем использовать Spring Data JPA, которая позволяет работать с объектами в Java-коде, автоматически преобразуя их в запросы к базе данных. Сама база данных MySQL выбрана как одно из наиболее популярных и производительных решений для хранения данных.

```
@Entity
public class Client {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String fullName;
    private String contactNumber;
}
```

Рисунок 2 - Пример работы ORM с базой данных

Пользовательский интерфейс будет разработан с применением HTML для разметки, CSS для стилизации, и JavaScript для динамики и интерактивности, что

обеспечит простоту использования и удобную навигацию по системе. Для создания интерактивных элементов и графиков, таких как гистограммы, будут использоваться JavaScript-библиотеки.

```
<form id="clientForm">
  <label for="name">Имя клиента:</label>
  <input type="text" id="name" name="name" required>

  <label for="phone">Телефон:</label>
  <input type="text" id="phone" name="phone" required>

  <button type="submit">Добавить клиента</button>
</form>

<script>
  document.getElementById("clientForm").addEventListener("submit", function(event) {
    event.preventDefault();
    alert("Клиент добавлен!");
  });
</script>
```

Рисунок 3 - Пример работы с HTML

1.4 Вывод по первой главе

В заключение теоретической части можно сделать следующий вывод, что информационно-справочные системы существенно повышают эффективность работы предприятий сферы услуг. Введение такой системы в автосервисе обеспечит автоматизацию ключевых процессов, улучшение контроля над запасами, упрощение взаимодействия с клиентами и более рациональное планирование рабочих процессов. Это позволит минимизировать вероятность ошибок, повысит уровень обслуживания и позволит оперативно реагировать на потребности клиентов.

ГЛАВА 2. РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ АВТОСЕРВИСОМ

2.1 Алгоритмические решения

Программа использует архитектуру клиент-сервер. Серверная часть отвечает за обработку запросов клиентов, хранение данных в базе данных и выполнение бизнес-логики. Клиентская часть — это веб-интерфейс, через который пользователь взаимодействует с сервером.

Для обеспечения безопасности используется Spring Security. Все запросы проходят проверку авторизации, а пароли пользователей шифруются с помощью BCrypt.

Все операции с данными, такие как добавление, редактирование или удаление записей, выполняются в модальных окнах. Это позволяет избежать перезагрузки страницы и сделать интерфейс более удобным.

```
@PostMapping("/add")
public String addClients(@ModelAttribute Clients clients) {
    clientsService.saveClients(clients);
    return "redirect:/clients/clients";
}
```

Рисунок 4 - Пример обработки запроса на добавление клиента

2.2 Описание интерфейса программы

Главная страница содержит элементы навигации по разделам системы: "Управление запчастями", "Планирование ремонта". При переходе внутрь одного из элементов в верхней панели можно найти ссылки на другие разделы системы: "Панель администратора", "Панель управления клиентами", "Об авторе".

Раздел "Клиенты" предоставляет таблицу с данными, где можно:

1. Добавить нового клиента через модальное окно.
2. Отредактировать данные клиента через модальное окно.
3. Удалить запись. При удалении всплывает окно подтверждения удаления.

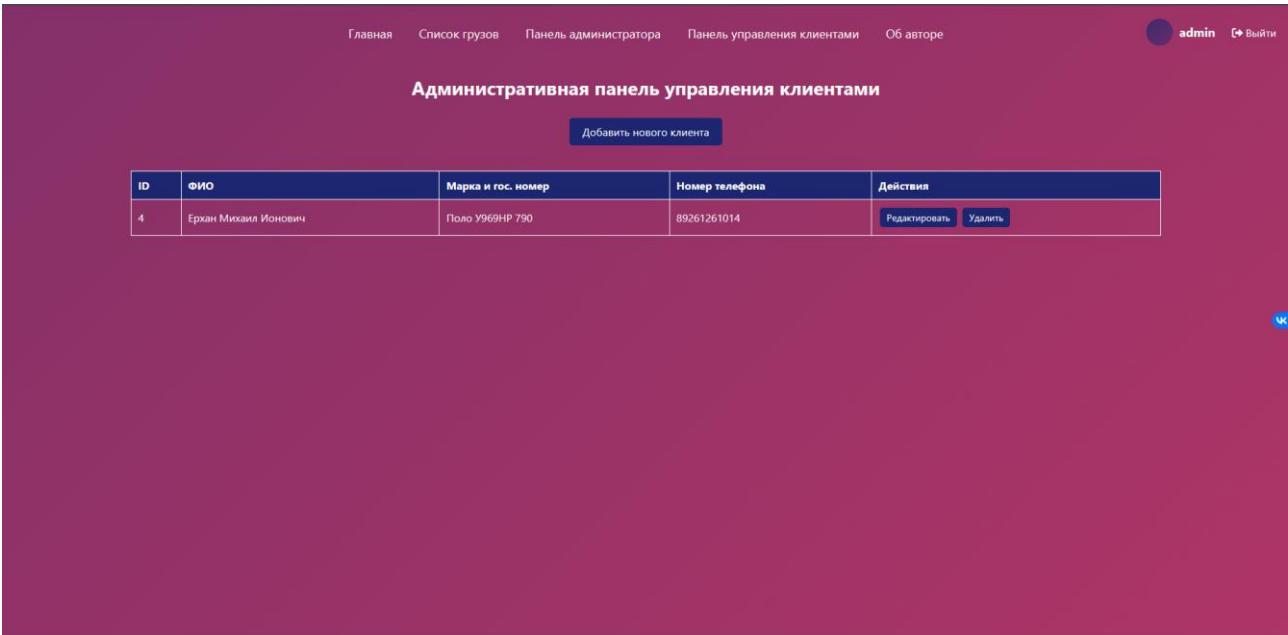


Рисунок 5 - Интерфейс управления клиентами

Раздел «Управление запчастями» включает в себя:

1. Таблицу со всеми заказанными запчастями.
2. Возможность добавить, редактировать, удалить запись с модальным окном.
3. Возможность поиска по любым параметрам.
4. Возможность посмотреть плотность распределения доставок по дням.
5. Сортировка по дате прибытия груза.

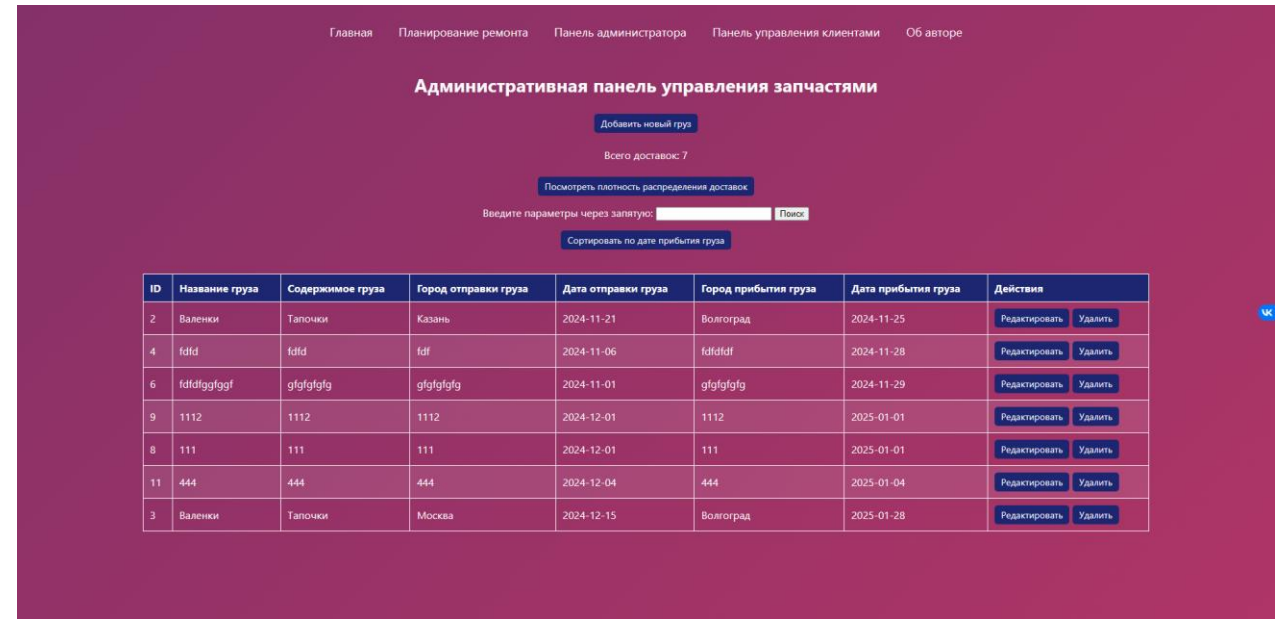


Рисунок 6 - Интерфейс управления запчастями и доставкой

Раздел «Планирование ремонта» включает в себя:

- 1. Таблицу со всеми записями на ремонт.
- 2. Возможность добавить, редактировать, удалить запись с модальным окном.
- 3. Возможность поиска по любым параметрам.

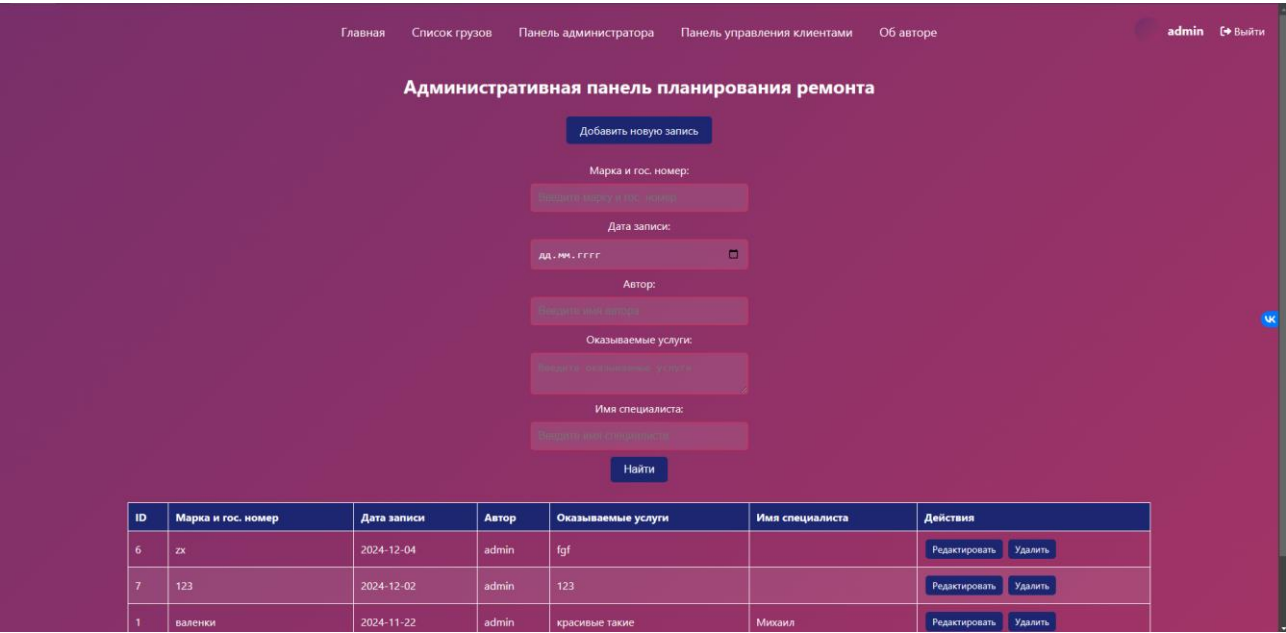


Рисунок 7 - Интерфейс планирования ремонтов

Раздел «Панель администратора» включает в себя:

- 1. Таблицу со всеми пользователями системы.
- 2. Возможность удалить пользователя.

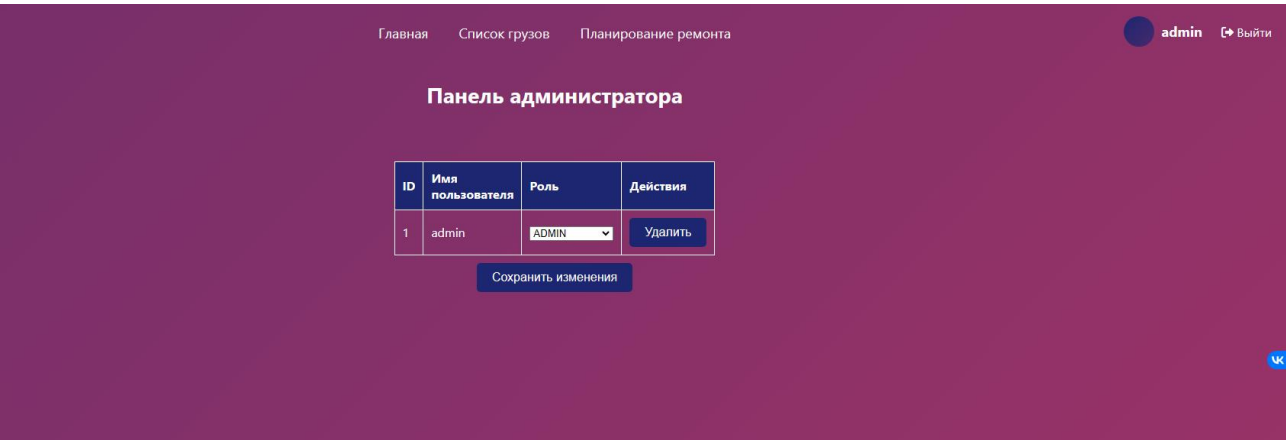


Рисунок 8 - Интерфейс панели администратора

Раздел «Об авторе» включает в себя:

1. Информацию об авторе и группе.
2. Информация о технологиях, использованных в проекте.
3. Информация о сроках выполнения работы

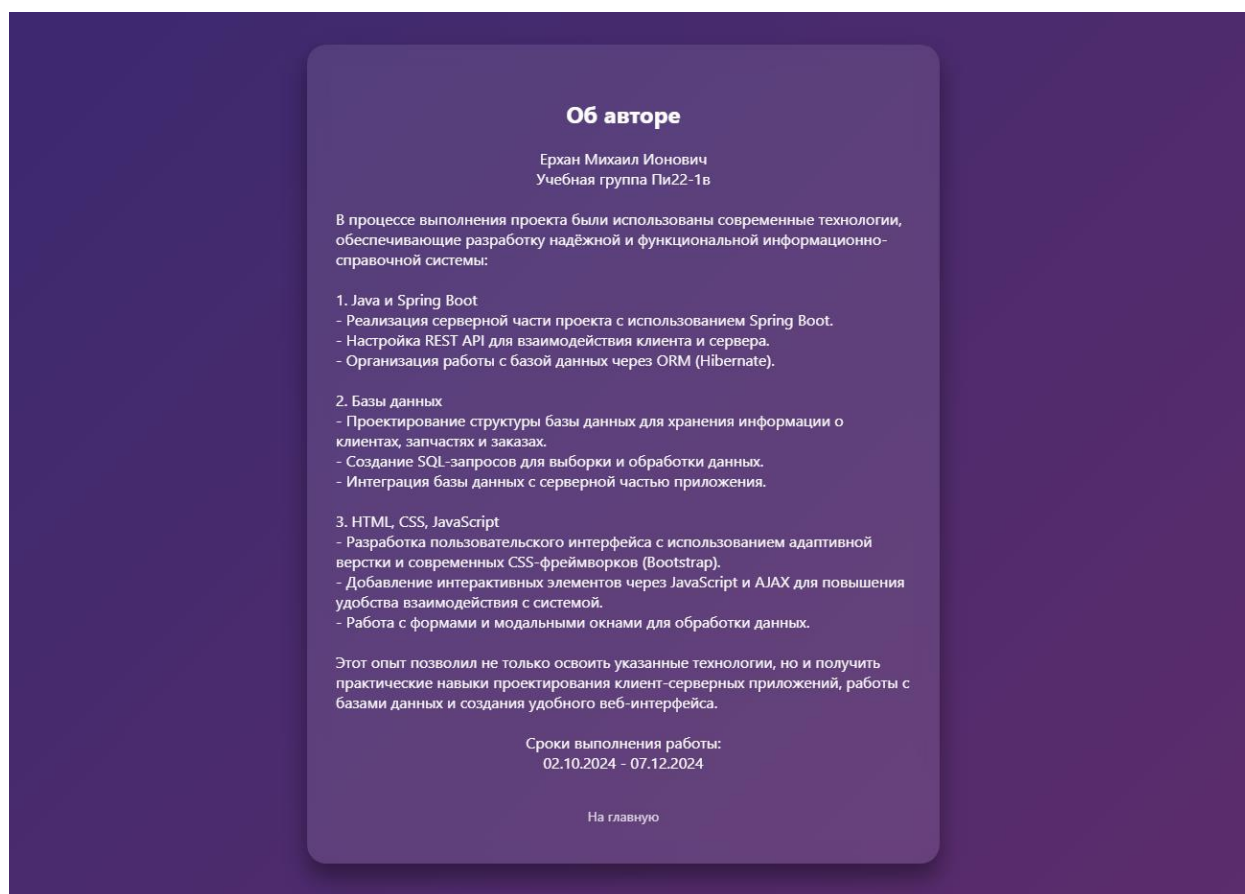


Рисунок 9 - Интерфейс об авторе

Раздел «Авторизация» позволяет пользователю войти в систему по его логину и паролю. При вводе неправильных данных всплывает подсказка, что логин или пароль неверные. Пользователю также доступна регистрация, после которой данные попадают в БД и в разделе «Панель администратора».

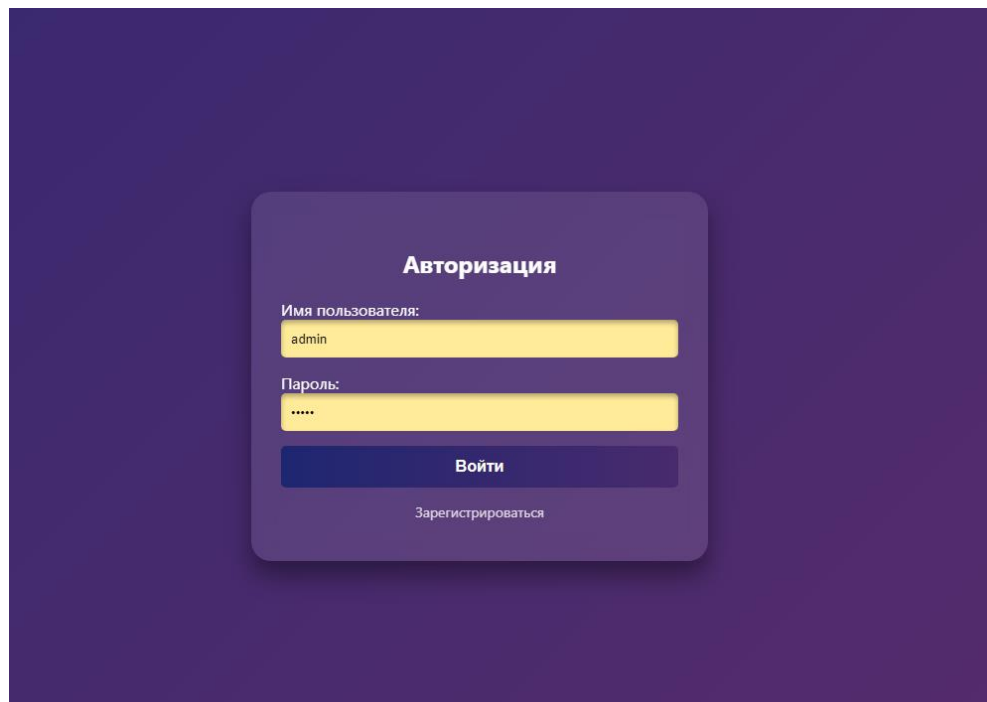
The image shows a login form titled "Авторизация" (Authorization) centered on a dark purple background. The form itself is a lighter purple rounded rectangle. It contains two input fields: "Имя пользователя:" (Username) with the text "admin" and "Пароль:" (Password) with masked characters "*****". Below these fields is a dark blue button labeled "Войти" (Login). At the bottom of the form is a link labeled "Зарегистрироваться" (Register).

Рисунок 10 - Интерфейс авторизации

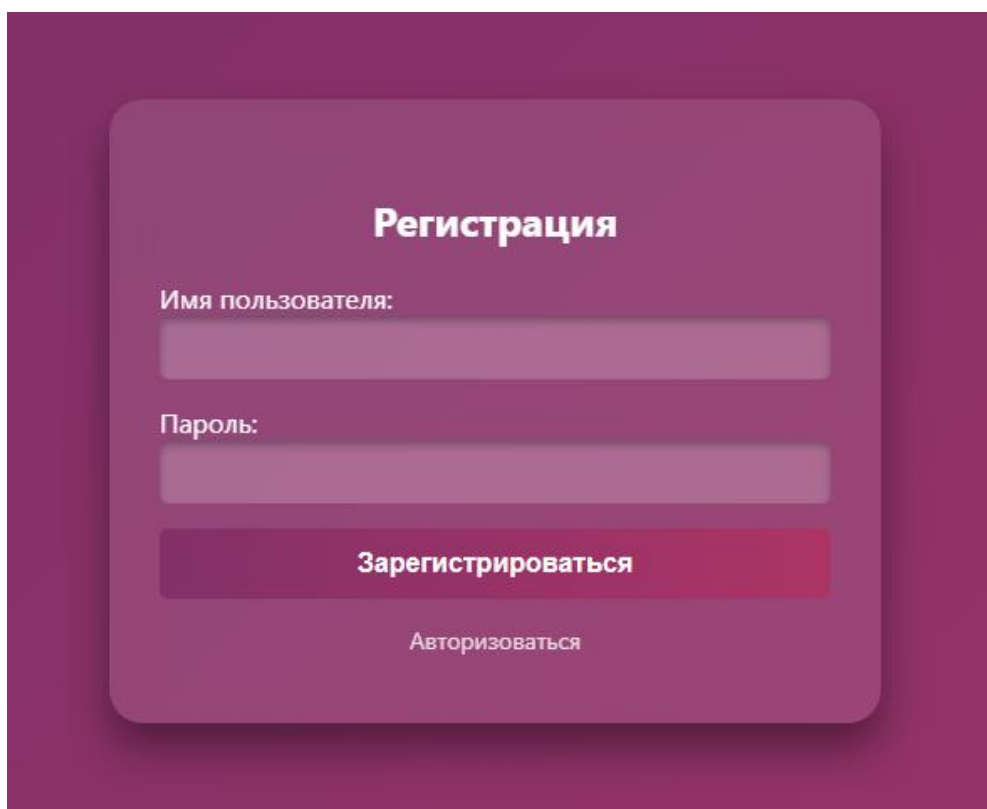
The image shows a registration form titled "Регистрация" (Registration) centered on a dark magenta background. The form is a lighter magenta rounded rectangle. It contains two input fields: "Имя пользователя:" (Username) and "Пароль:" (Password). Below these fields is a dark magenta button labeled "Зарегистрироваться" (Register). At the bottom of the form is a link labeled "Авторизоваться" (Login).

Рисунок 11 - Интерфейс регистрации

При вводе верных данных или при регистрации пользователь попадает на главную страницу, где есть выбор основных разделов.

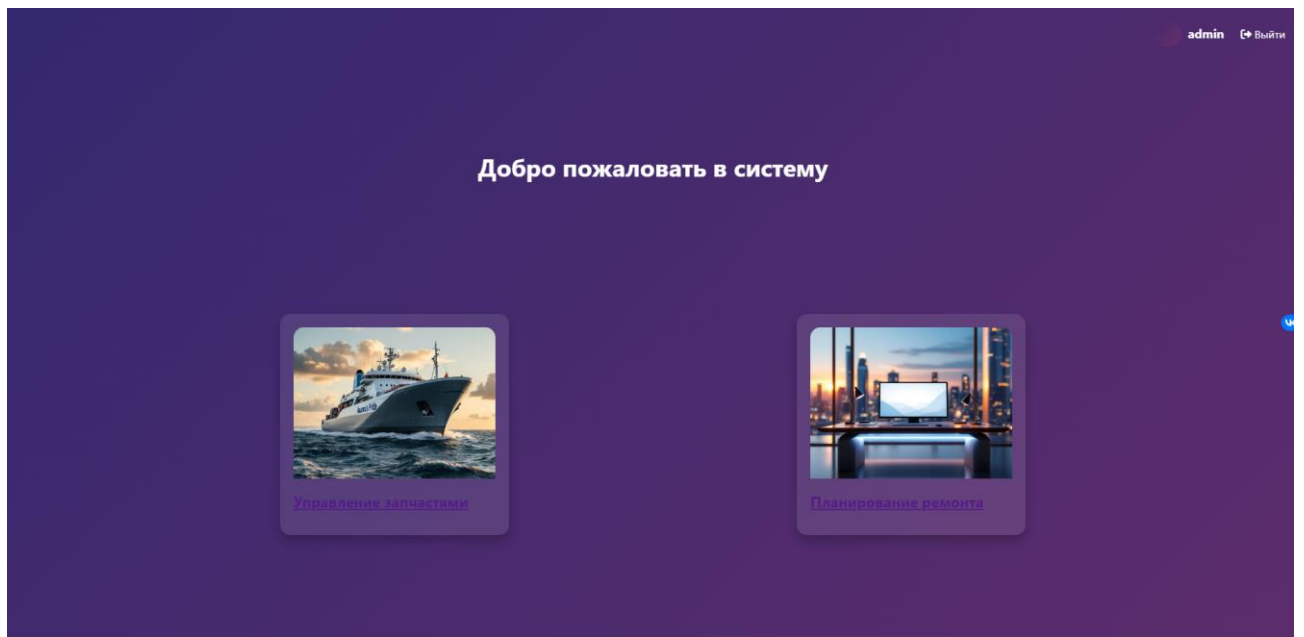


Рисунок 12 - Интерфейс главной страницы

Модальное окно редактирования открывается при выборе кнопки "Редактировать" напротив конкретной доставки или конкретной записи на ремонт.

1. Поля заполняются текущими данными.
2. Пользователь (модератор или администратор) может изменить любую информацию.

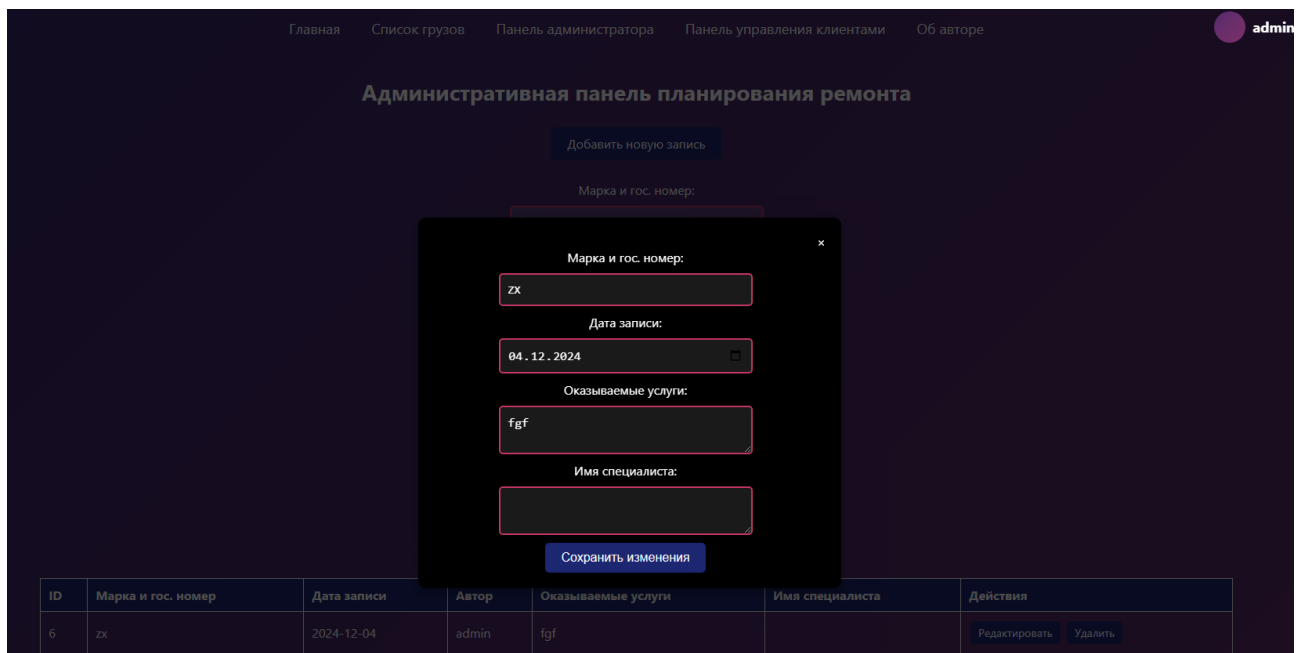


Рисунок 13 - Интерфейс модального окна редактирования записей ремонта

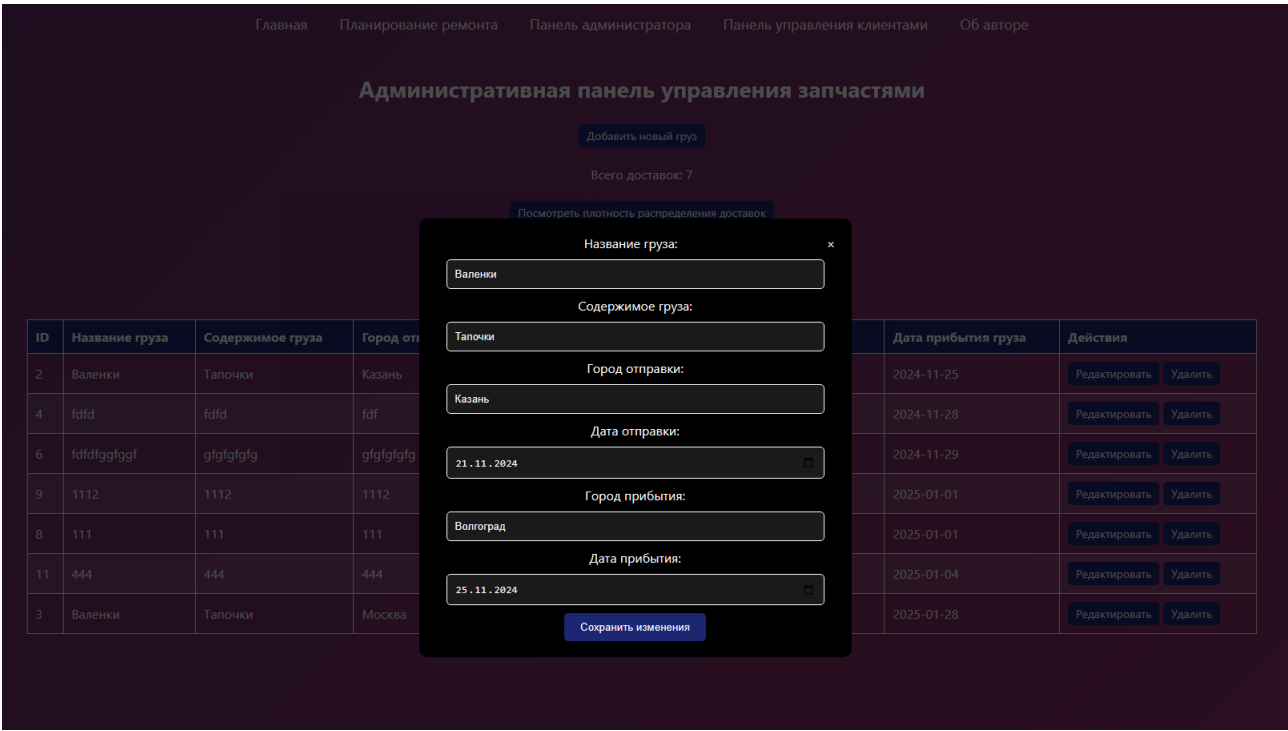


Рисунок 14 - Интерфейс модального окна редактирования доставки запчастей

Модальное окно добавления открывается при нажатии на кнопку «Добавить новый груз» в разделе управления запчастями или «Добавить новую запись» в разделе планирования ремонта или «Добавить нового клиента» в разделе управления клиентами.

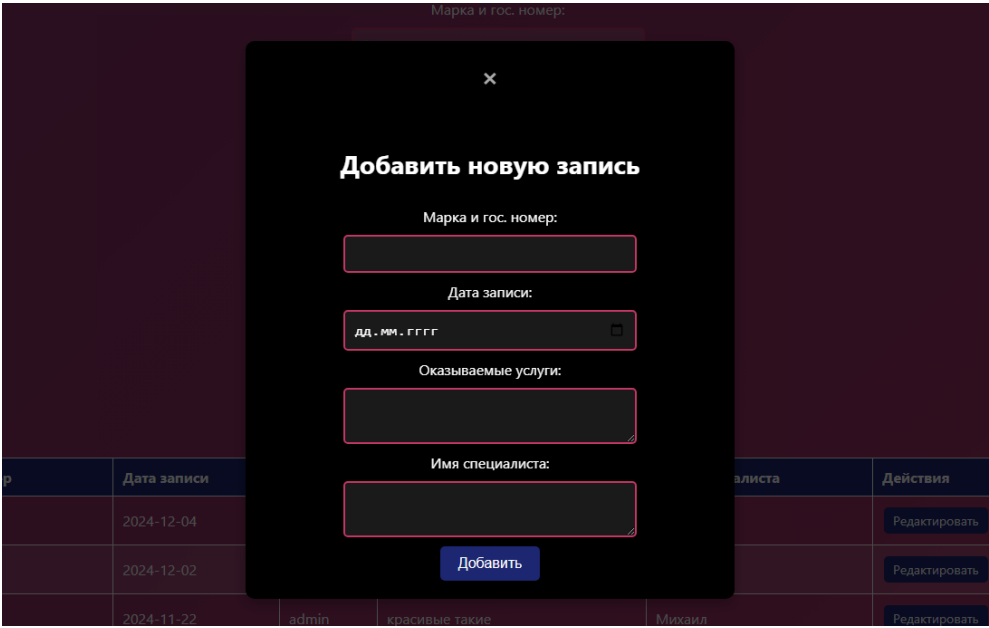


Рисунок 15 - Интерфейс модального окна добавления записи на ремонт

марка и гос. номер | Номер телефона

По

×

Добавить новую запись

ФИО:

Марка и гос. номер:

Номер телефона:

Добавить

Рисунок 16 - Интерфейс модального окна добавления клиента в базу данных

Добавить новый груз

×

Добавить груз

Название груза:

Содержимое груза:

Город отправки:

Дата отправки:

Город прибытия:

Дата прибытия:

Добавить

Рисунок 17 - Интерфейс модального окна добавления доставки запчастей

Модальное окно подтверждения удаления записи открывается при нажатии на кнопку «Удалить запись».

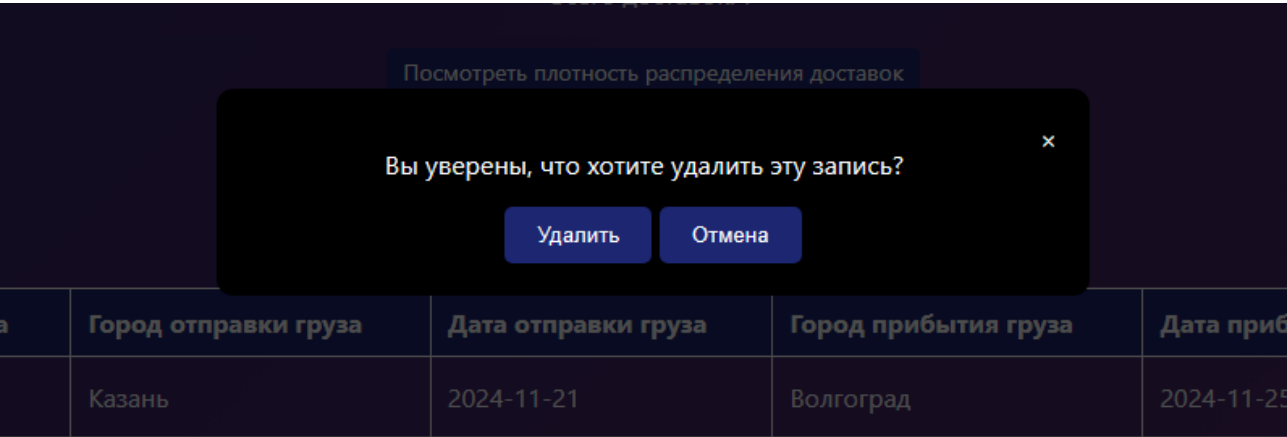


Рисунок 18 - Интерфейс модального окна подтверждения удаления

В разделе «Управления запчастями» есть возможность посмотреть плотность распределения доставок по дням. Также есть возможность посмотреть плотность распределения на конкретную дату.

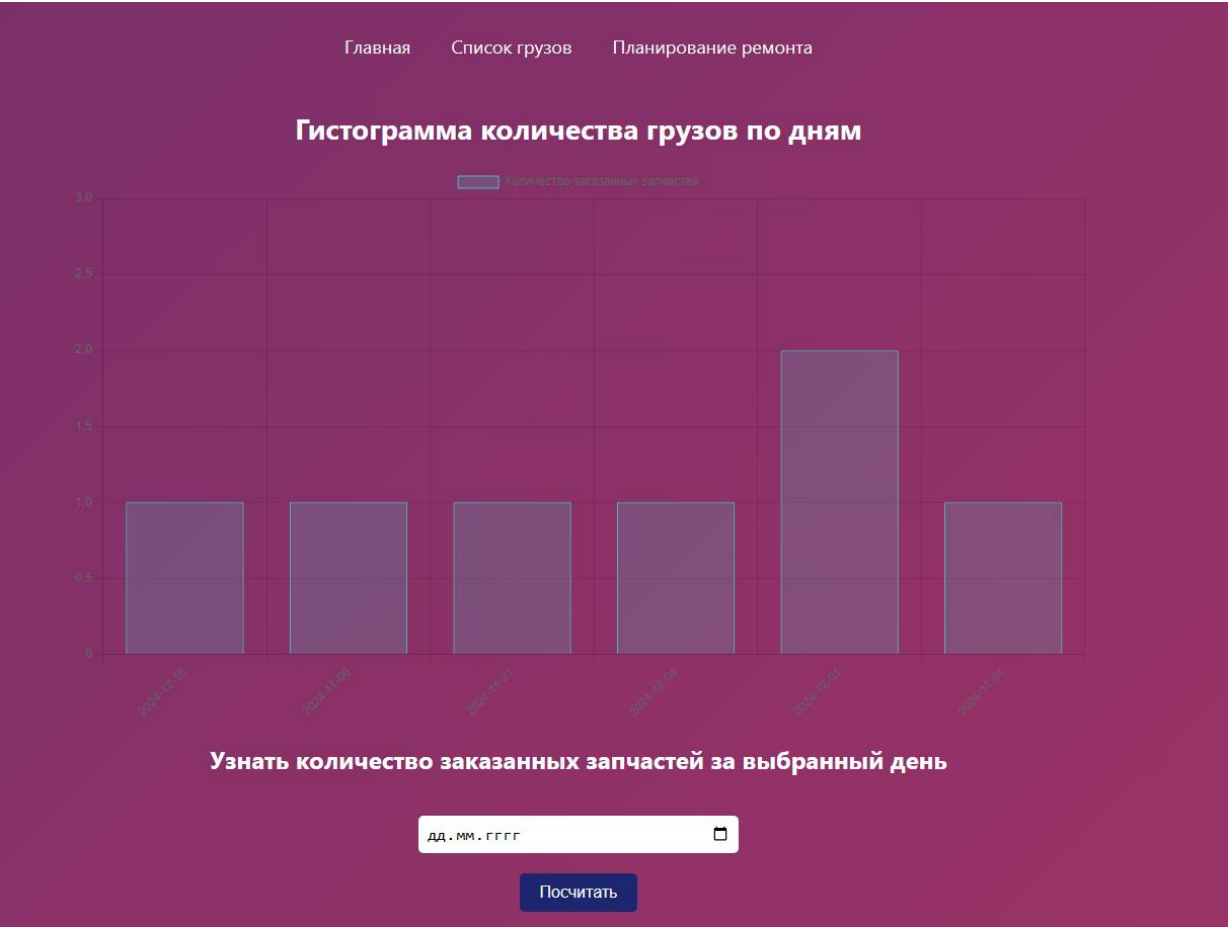


Рисунок 19 - Гистограмма количества грузов по дням

2.3. Архитектура приложения

Разрабатываемая система построена на основе трёхзвенной архитектуры:

1. Клиентский слой

1. Реализован с использованием HTML, CSS и JavaScript.
2. Обеспечивает взаимодействие пользователя с системой через веб-браузер.
3. Включает функционал для отправки запросов на сервер через AJAX и отображения полученных данных.

2. Серверный слой

1. Реализован с использованием Spring Boot.
2. Обрабатывает HTTP-запросы, выполняет бизнес-логику и взаимодействует с базой данных.
3. Включает REST API для работы с клиентскими запросами.

3. Слой данных

1. Используется реляционная база данных PostgreSQL.
2. Хранит данные о клиентах, запчастях, заказах и расписании.
3. Обеспечивает целостность данных через ограничения и связи между таблицами.

2.4 Схема взаимодействия с системой

1. Пользователь через веб-интерфейс (клиент) отправляет запрос (например, добавление клиента).
2. Сервер принимает запрос через REST API и передает его в соответствующий контроллер.
3. Контроллер вызывает сервис для выполнения бизнес-логики.
4. Сервис взаимодействует с базой данных через репозиторий, выполняет необходимые операции и возвращает результат.
5. Результат передается пользователю в виде обновления интерфейса.

2.5 Используемые технологии

1. Spring Boot: фреймворк для создания серверной части.
2. Spring Data JPA: обеспечивает взаимодействие с базой данных через ORM.
3. Thymeleaf: шаблонизатор для динамического формирования HTML-страниц.
4. Bootstrap: CSS-фреймворк для стилизации интерфейса.
5. AJAX: для выполнения асинхронных запросов к серверу.

2.6 Разработка системы

2.6.1 Контроллеры

Контроллеры отвечают за обработку HTTP-запросов и передачу данных между клиентом и сервером.

```
@GetMapping("/moderate/add")
public String showAddCargoForm(Model model) {
    model.addAttribute("cargo", new CargoModel());
    return "redirect:/moderate";
}
```

Рисунок 20 - Пример реализации контроллера

2.6.2 Сервисы

Сервисы содержат бизнес-логику и обрабатывают данные перед их передачей в репозиторий или клиенту.

```

@Service 6 usages
public class CargoService {

    private final CargoRepository cargoRepository; 8 usages

    @Autowired
    public CargoService(CargoRepository cargoRepository) { this.cargoRepository = cargoRepository; }

    // Сохраняем груз в базе данных
    public void saveCargo(CargoModel cargoModel) { cargoRepository.save(cargoModel); }

    // Находим груз по ID
    public Optional<CargoModel> findCargoById(Long id) { return cargoRepository.findById(id); }

    // Находим все грузы, отсортированные по дате
    public List<CargoModel> findAllCargosSortedByDateTime() { 4 usages
        return cargoRepository.findAllByOrderByArrivalDateAsc(); // Сортировка по возрастанию
    }

    public List<CargoModel> findAllCargosSortedByDateTimeDesc() { 2 usages
        return cargoRepository.findAllByOrderByArrivalDateDesc(); // Сортировка по убыванию
    }
}

```

Рисунок 21 - Пример реализации сервиса

2.6.3 Репозитории

Репозитории обеспечивают доступ к данным, хранящимся в базе данных.

```

@Repository 3 usages
public interface CargoRepository extends JpaRepository<CargoModel, Long> {

    @Query("SELECT p FROM CargoModel p WHERE " +
        + "(COALESCE(:cargoName, '') = '' OR LOWER(p.cargoName) LIKE LOWER(CONCAT('%', :cargoName, '%')))) AND " +
        + "(COALESCE(:cargoContents, '') = '' OR LOWER(p.cargoContents) LIKE LOWER(CONCAT('%', :cargoContents, '%')))) AND " +
        + "(COALESCE(:departureCity, '') = '' OR LOWER(p.departureCity) LIKE LOWER(CONCAT('%', :departureCity, '%')))) AND " +
        + "(COALESCE(:departureDate, '') = '' OR LOWER(CAST(p.departureDate AS string)) LIKE LOWER(CONCAT('%', :departureDate, '%')))) AND " +
        + "(COALESCE(:arrivalCity, '') = '' OR LOWER(p.arrivalCity) LIKE LOWER(CONCAT('%', :arrivalCity, '%')))) AND " +
        + "(COALESCE(:arrivalDate, '') = '' OR LOWER(CAST(p.arrivalDate AS string)) LIKE LOWER(CONCAT('%', :arrivalDate, '%'))))")
    List<CargoModel> searchByQuery(@Param("cargoName") String cargoName,
        @Param("cargoContents") String cargoContents,
        @Param("departureCity") String departureCity,
        @Param("departureDate") String departureDate,
        @Param("arrivalCity") String arrivalCity,
        @Param("arrivalDate") String arrivalDate);

    List<CargoModel> findAllByOrderByArrivalDateAsc(); 1 usage
    List<CargoModel> findAllByOrderByArrivalDateDesc(); 1 usage
}

```

Рисунок 22 - Пример реализации репозитория

2.6.4 Модели данных

Модели представляют сущности, связанные с таблицами базы данных.

```

@Setter 25 usages
@Entity
@Table(name = "cargo")
public class CargoModel {
    private Long id;

    @Getter
    private String cargoName;

    @Getter
    private String cargoContents;

    @Getter
    private String departureCity;//

    @Getter
    private LocalDate departureDate;

    @Getter
    private String arrivalCity;//

    @Getter
    private LocalDate arrivalDate;

    // Конструкторы
    public CargoModel() {}

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Long getId() { return id; }

```

Рисунок 23 - Пример реализации модели данных

2.7 База данных

Для реализации программы используется база данных PostgreSQL.
Основные таблицы:

1. Таблица «Грузы»
 1. id – уникальный идентификатор
 2. cargoName – название груза
 3. cargoContents – содержимое груза
 4. departureCity – город отправки груза
 5. departureDate – дата отправки груза
 6. arrivalCity – город прибытия груза
 7. arrivalDate – дата прибытия груза
2. Таблица «Клиенты»
 1. id – уникальный идентификатор

2. FIO – Фамилия Имя Отчество клиента
 3. BrandNumberAuto – марка и гос. номер автомобиля
 4. Number – номер телефона клиента
3. Таблица «Пользователи»
1. userName – имя пользователя
 2. password – пароль пользователя
 3. role – роль в системе
4. Таблица «Планирование ремонта»
1. id – уникальный идентификатор
 2. title – марка и гос. номер автомобиля (FK с таблицей Пользователи)
 3. publicationDate – дата записи клиента
 4. content – оказываемые услуги
 5. specialist – имя специалиста (механика)
 6. author – имя автора (пользователя, создавшего запись)

2.8 Безопасность системы

Класс `SecurityConfig` в приложении обеспечивает конфигурацию безопасности на основе Spring Security. Он отвечает за настройку авторизации, аутентификации, управления доступом и других аспектов защиты системы.

2.8.1 Настройка шифрования паролей

Для шифрования паролей в системе используется «BCryptPasswordEncoder». Этот алгоритм обеспечивает надёжное хеширование, что делает пароли безопасными при хранении в базе данных.

Принцип работы алгоритма довольно простой:

1. При регистрации пользователя пароль преобразуется в хэш и сохраняется в базе данных.
2. Во время аутентификации введенный пароль сравнивается с хэшем и, если они совпадают, авторизация проходит успешно.

```
// Настраиваем шифрование паролей
@Bean
public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Рисунок 24 - Настройка шифрования паролей

2.8.2 Настройка менеджера аутентификации

«AuthenticationManager» используется для управления процессом аутентификации. Он автоматически интегрируется с реализацией «UserDetailsService» через «CustomUserDetailsService».

Принцип работы менеджера аутентификации тоже несложный:

1. Запрос аутентификации направляется «AuthenticationManager».
2. А он использует «CustomUserDetailsService» для поиска пользователя и сравнения паролей.

```
// Настраиваем шифрование паролей
@Bean
public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Рисунок 25 - Настройка шифрования паролей

Запрос		История запросов		
1	SELECT	*	FROM	public.users
2	ORDER BY	id	ASC	

Data Output		Сообщения			Notifications		
	id	password	role	user_name			
	[PK] bigint	character varying (255)	character varying (255)	character varying (255)			
1	1	\$2a\$10\$uhoCIIQRabXCB7YHoKQu7ulj32ilZigTv83.H7P7zXg9tp.DvAp...	ADMIN	admin			

Рисунок 26 - Пример хранения шифрованного пароля в БД

2.8.3 Обработка доступа и ошибок

Обработчик «AccessDeniedHandler» обрабатывает ситуации, когда пользователь пытается получить доступ к запрещённым страницам. В этом случае пользователь перенаправляется на кастомную страницу ошибки (/error/403)

```
@Bean
public AccessDeniedHandler accessDeniedHandler() {
    return (request, response, accessDeniedException) -> {
        response.sendRedirect("/error/403"); // Перенаправляем на кастомную страницу
    };
}
```

Рисунок 27 - Пример обработчика ошибок

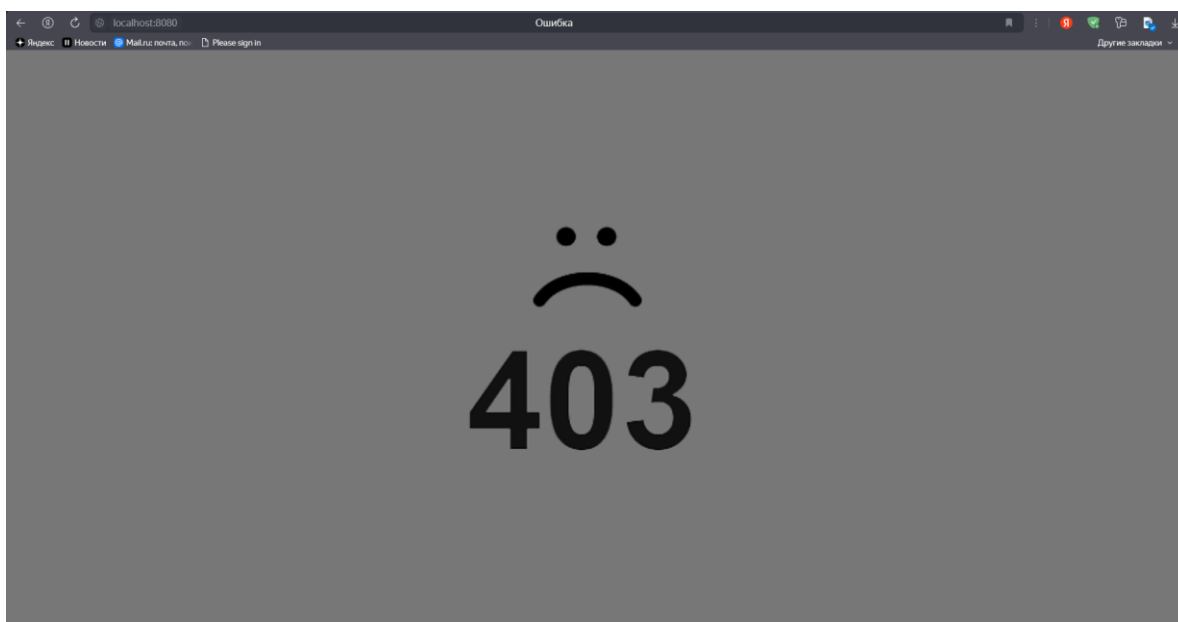


Рисунок 28 - Страница ошибки

2.8.4 Конфигурация фильтров безопасности

Описание авторизации пользователей:

1. Админ-панель доступна только для пользователей с ролью ADMIN.
2. Модераторы могут работать с разделами /moderate/**.
3. Все пользователи должны быть авторизованы для доступа к /blog/**.
4. Неавторизованные запросы к другим страницам разрешены.

Аутентификация через форму входа:

1. Пользователь перенаправляется на /login для авторизации.
2. После успешного входа он попадает на главную страницу, доступную исходя из роли пользователя.

Выход из системы:

1. URL выхода /logout завершает сессию.
2. После выхода пользователь перенаправляется на /login.

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests(authz -> authz
            .requestMatchers("/admin_panel/**").hasRole("ADMIN") // Только для админа
            .requestMatchers("/moderate/**").hasAnyRole("ADMIN", "MODERATOR") // Для модераторов и админов
            .requestMatchers("/blog/admin/**").hasAnyRole("ADMIN", "MODERATOR") // Для модераторов и админов
            .requestMatchers("/cargolist/**").authenticated() // Только для авторизованных пользователей
            .requestMatchers("/blog/**").authenticated() // Только для авторизованных пользователей
            .requestMatchers("/").authenticated() // Только для авторизованных пользователей
            .anyRequest().permitAll() // Все остальные запросы доступны всем
        )
        .formLogin(form -> form
            .loginPage("/login")
            .defaultSuccessUrl("/", alwaysUse: true) // Кастомная страница логина
            .permitAll()
        )
        .logout(logout -> logout
            .logoutUrl("/logout")
            .logoutSuccessUrl("/login") // Перенаправление после выхода
            .permitAll()
        )
        .exceptionHandling(exceptionHandling -> exceptionHandling
            .accessDeniedHandler(accessDeniedHandler())
        )
        .csrf(AbstractHttpConfigurer::disable // Отключение CSRF-защиты
        );

    return http.build();
}
```

Рисунок 29 - Конфигурация фильтров безопасности

2.8.5 Общие преимущества и безопасность

1. Гибкость в настройке

Использование SecurityFilterChain позволяет задать сложные правила доступа для разных частей приложения.

2. Роли и права

Благодаря механизму ролей, доступ можно чётко разграничить для администраторов, модераторов и пользователей.

3. Шифрование паролей

«BCryptPasswordEncoder» обеспечивает надёжное хеширование, защищая пароли от компрометации.

4. Кастомизация

Возможность создавать собственные обработчики ошибок (например, страница 403) и формы входа.

5. Интеграция с сервисом пользователей

Использование «CustomUserDetailsService» для получения данных о пользователях и их ролях из базы.

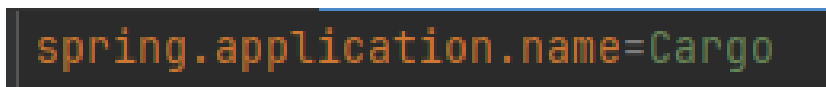
Безопасность — это ключевая часть системы, обеспечивающая защиту приложения и удобство для пользователей.

2.8.6 Настройки приложения

Файл «application.properties» содержит настройки, определяющие поведение приложения и его взаимодействие с базой данных.

1. Основные настройки системы

1. Устанавливает имя приложения
2. Имя используется в различных службах Spring для идентификации приложения.



```
spring.application.name=Cargo
```

Рисунок 30 - Основные настройки системы

2. Настройки подключения к базе данных

1. spring.datasource.url: Адрес базы данных PostgreSQL, к которой подключается приложение.
2. spring.datasource.username: Имя пользователя базы данных.
3. spring.datasource.password: Пароль для подключения.
4. spring.datasource.driver-class-name: Класс драйвера для работы с PostgreSQL.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/cargo_db
spring.datasource.username=postgres
spring.datasource.password=root
spring.datasource.driver-class-name=org.postgresql.Driver
```

Рисунок 31 - Настройки подключения к БД

3. Настройки JPA и Hibernate

1. spring.jpa.hibernate.ddl-auto: определяет стратегию работы с базой данных:
 - i. update — автоматически вносит изменения в структуру базы данных на основе сущностей Java.
 - ii. Подходит для разработки, но не рекомендуется для продакшн-среды.
2. spring.jpa.show-sql: Включает вывод SQL-запросов в консоль для отладки.
3. spring.jpa.properties.hibernate.dialect: Определяет диалект SQL, используемый Hibernate. Для PostgreSQL используется PostgreSQLDialect.

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Рисунок 32 - Настройка JPA и Hibernate

4. Настройки ресурсов веб-приложения

1. Включает автоматическое добавление маппингов для статических ресурсов, таких как CSS, JavaScript и изображения.
2. Если статические файлы размещены в папке src/main/resources/static, они автоматически становятся доступными для приложения.

```
spring.web.resources.add-mappings=true
```

Рисунок 33 - Настройки ресурсов веб-приложения

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана информационно-справочная система для управления автосервисом, включающая ключевые модули для учёта клиентов, управления запасными частями и планирования ремонтных работ. Основной целью работы являлось создание удобной и функциональной системы, которая обеспечивает учёт клиентов, управление запасными частями и планирование ремонтных работ. Все задачи, поставленные перед началом разработки, были успешно решены.

1. Анализ предметной области

В рамках работы были изучены основные процессы, связанные с управлением автосервисом, включая запись клиентов, планирование работ и учёт складских запасов. Это позволило определить требования к функционалу системы и заложить основу для её проектирования.

2. Проектирование архитектуры системы

Для реализации системы была выбрана клиент-серверная архитектура с использованием современных технологий, таких как Spring Boot для серверной части и PostgreSQL для базы данных. Было спроектировано удобное взаимодействие между сервером и клиентом, обеспечивающее высокую производительность и масштабируемость.

3. Разработка пользовательского интерфейса

Интерфейс системы был создан с использованием HTML, CSS и JavaScript. Благодаря этому пользователи получили интуитивно понятный инструмент для управления клиентами, запчастями и ремонтными заказами.

4. Настройка безопасности системы

Система была защищена с использованием Spring Security. Это позволило реализовать разграничение прав доступа для администраторов, модераторов и обычных пользователей.

5. Тестирование и внедрение функционала

Проведено тестирование всех основных модулей системы, включая работу с базой данных, авторизацию и переключение языков интерфейса. Все функции работают корректно, а система готова к внедрению в рабочую среду.

Результатом выполнения работы стала универсальная система, которая упрощает учёт клиентов, управление складом и планирование ремонтов. Решение соответствует современным требованиям и может быть использовано для оптимизации работы автосервисов.

Основные достижения проекта

1. Автоматизация процессов

1. Упрощение записи клиентов, отслеживания ремонтов и управления складскими запасами запчастей.
2. Значительное сокращение времени на обработку данных благодаря централизованному хранению и обработке информации.

2. Применение современных технологий

1. Использование Spring Boot и PostgreSQL для создания надёжной серверной части с гибкой архитектурой.
2. Реализация интерактивного пользовательского интерфейса с помощью HTML, CSS и JavaScript, что обеспечивает удобство работы с системой.
3. Интеграция механизма безопасности на основе Spring Security для защиты данных и управления доступом.

3. Масштабируемость и гибкость

1. Внедрение клиент-серверной архитектуры позволяет системе масштабироваться в зависимости от увеличения числа пользователей или расширения функционала.

4. Улучшение пользовательского опыта

1. Удобная работа с модальными окнами для редактирования и добавления записей.
2. Визуализация данных, таких как распределение доставок, что помогает в принятии управленческих решений.

Практическая ценность

Разработанная система предоставляет автосервису эффективный инструмент для упрощения учёта, планирования и анализа. Это способствует повышению качества обслуживания клиентов, снижению количества ошибок и более рациональному распределению ресурсов.

Перспективы развития

1. Расширение функционала системы, например, добавление модуля для автоматического напоминания клиентам о запланированных ремонтах.
2. Интеграция с внешними поставщиками запчастей для автоматизации заказа.
3. Оптимизация производительности за счёт внедрения асинхронной обработки данных.
4. Разработка мобильной версии приложения для удобного доступа сотрудников автосервиса.

Таким образом, данная курсовая работа позволила не только достичь поставленных целей, но и получить ценные навыки проектирования клиент-серверных приложений. Выполненный проект является основой для дальнейшего совершенствования и практического применения в реальной среде.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Хеклер М. Spring Boot по-быстрому / М. Хеклер. – 2021. – 272 с.
2. Раджпут Д. Освоение Spring Boot 2.0 / Д. Раджпут. – 2020. – 356 с.
3. Лонг Д., Бастани, К. Java в облаке. Spring Boot, Spring Cloud, Cloud Foundry / Д. Лонг К. Бастани. – 2018. – 310 с.
4. Шилдс У. SQL: быстрое погружение / У. Шилдс. – 2019. – 256 с.
5. Рогов Е. PostgreSQL изнутри / Е. Рогов. – 2020. – 400 с.
6. Нестеров С.А. Базы данных: учебник и практикум для вузов / С.А. Нестеров. – 2017. – 448 с.
7. Дакетт Дж. HTML и CSS: Разработка и дизайн веб-сайтов / Дж. Дакетт. – 2014. – 490 с.
8. Фрейн Б. HTML5 и CSS3: Разработка современных веб-сайтов / Б. Фрейн. – 2018. – 384 с.
9. Мейе Э. CSS. Полное руководство / Э. Мейер. – 2020. – 528 с.
10. Хавербеке М. Выразительный JavaScript / М. Хавербеке. – 2020. – 472 с.
11. Браун Э. Изучаем JavaScript / Э. Браун. – 2018. – 320 с.
12. Закас Н. Современный JavaScript для веб-разработчиков / Н. Закас. – 2017. – 512 с.
13. Буайе Л. Spring Security в действии / Л. Буайе. – 2019. – 368 с.
14. Бозетти Л. Spring Microservices Security in Action / Л. Бозетти. – 2020. – 400 с.
15. Уоллс К. Spring в действии / К. Уоллс. – М.: ДМК Пресс, 2018. – 472 с.

ПРИЛОЖЕНИЕ

Весь код разработанной системы расположен на GitHub:

<https://github.com/MichaelErhan/Kurovaya>