

Geekbrains

**Разработка веб-приложения RoadAnalyzer для учета автомобильных расходов и обслуживания на языке программирования JAVA с использованием фреймворка Spring Boot**

Программа:

Java-разработчик

Громов Максим Евгеньевич

Новосибирск

2024

## **Содержание**

Введение ………………………………………………………………..стр.2

Теоретическая часть ……………………………………………...……стр.3

Обоснование выбора технологии……….……………………………..стр.3

Теоретические основы………………………………………………….стр.7

Практическая часть……………………………………………….…….стр.9

Сущности базы данных…………………………………………………стр.9

Эндпоинты API………………………………………………………….стр.15

Тестирование эндпоинтов с помощью Swagger……………………….cтр.39

Настройки приложения………………………………………….………стр.40

Описание фронтенда…………………………………………………….стр.44

Принцип работы приложения……………………………….…………..стр.48

Трудности и решения…………………………………………………….стр.50

Тестирование……………………………………………………………..стр.52

Выводы……………………………………………………………………стр.55

Список используемых литературных источников……………….……..стр.58

## 

## **Введение**

## 

RoadAnalyzer — это веб-приложение, предназначенное для контроля расходов и технического обслуживания автомобилей. Оно помогает пользователям отслеживать затраты на топливо, обслуживание и другие расходы, связанные с эксплуатацией автомобиля, а также управлять техническим состоянием транспортного средства.

## Цели и задачи проекта:

Цель проекта RoadAnalyzer заключается в создании эффективного инструмента для владельцев автомобилей, который позволит:

* Оптимизировать расходы: Пользователи смогут отслеживать и анализировать свои расходы на топливо и техническое обслуживание, что поможет им принимать более обоснованные финансовые решения.
* Управлять техническим состоянием: Приложение будет напоминать пользователям о предстоящих технических обслуживании и предоставлять информацию о состоянии их автомобилей.
* Генерировать отчеты: Пользователи смогут получать детализированные отчеты о своих расходах, что поможет в планировании бюджета.

Аудитория приложения включает владельцев автомобилей, автосервисы и компании, управляющие автопарками. Оно будет полезно как частным лицам, так и организациям, стремящимся к оптимизации затрат на эксплуатацию транспортных средств.

## Обзор существующих решений

На рынке существует несколько аналогичных приложений, таких как Drivvo, Driver Notes и Мой Авто. Однако RoadAnalyzer выделяется среди них благодаря следующим преимуществам:

* Интуитивно понятный интерфейс: Приложение предлагает простой и удобный интерфейс для ввода данных и получения отчетов, что делает его доступным для пользователей с разным уровнем технической подготовки.
* Комплексный подход: В отличие от многих аналогов, которые фокусируются только на учете топлива или расходов, RoadAnalyzer объединяет функции учета расходов с управлением техническим обслуживанием. Это позволяет пользователям не только отслеживать затраты, но и контролировать состояние автомобиля.
* Персонализированные уведомления: Приложение предлагает индивидуальные напоминания о необходимости обслуживания на основе пробега и времени, что позволяет пользователям не забывать о важных процедурах и поддерживать автомобиль в отличном состоянии.

## Актуальность приложения

В условиях растущих затрат на топливо и обслуживание автомобилей, а также необходимости эффективного управления техническим состоянием транспортных средств, приложение RoadAnalyzer становится особенно актуальным. Оно предоставляет пользователям инструменты для оптимизации расходов и повышения надежности их автомобилей.

Контроль технического состояния автомобиля является критически важным аспектом для обеспечения безопасности на дороге и долгосрочной эксплуатации транспортного средства. Регулярный мониторинг состояния позволяет выявлять потенциальные неисправности на ранних стадиях, что помогает избежать серьезных поломок и высоких затрат на ремонт. В современных условиях, когда автомобили становятся все более сложными с точки зрения технологий и систем, необходимость в систематическом подходе к диагностике и обслуживанию возрастает.

Приложение RoadAnalyzer не только помогает пользователям отслеживать затраты на топливо и обслуживание, но и предоставляет возможность контролировать техническое состояние автомобиля. Оно включает функции для ведения истории обслуживания, отслеживания использованных запчастей и получения персонализированных напоминаний о необходимости проведения технических работ. Это способствует поддержанию автомобиля в исправном состоянии и продлевает срок его службы.

## Основные функции

* Учет расходов на топливо и обслуживание: Пользователи могут вносить данные о своих расходах на топливо, услуги по ремонту и техническому обслуживанию, что позволяет отслеживать финансовые затраты на эксплуатацию автомобиля.
* Напоминания о техническом обслуживании: Приложение автоматически генерирует напоминания о предстоящих технических обслуживании на основе пробега и времени, что помогает пользователям не забывать о важных процедурах и поддерживать автомобиль в отличном состоянии.
* Генерация отчетов по расходам: Пользователи могут получать детализированные отчеты о своих расходах, что позволяет анализировать финансовую эффективность эксплуатации автомобиля и планировать бюджет.
* Контроль за техническим состоянием автомобиля: Приложение предоставляет возможность отслеживать текущее состояние автомобиля, включая данные о проведенных работах, что способствует более эффективному управлению его обслуживанием.
* Отслеживание использованных запчастей: Пользователи могут вести учет всех использованных запчастей, что упрощает процесс обслуживания и позволяет заранее планировать закупки необходимых материалов.
* История обслуживания: Приложение сохраняет историю всех проведенных работ и замененных запчастей, что помогает пользователям лучше понимать потребности своего автомобиля и принимать обоснованные решения о дальнейшем обслуживании.

**Теоретическая часть**

## **Обоснование выбора технологий**

В процессе разработки приложения RoadAnalyzer были выбраны следующие технологии, каждая из которых обладает уникальными преимуществами:

* Java: Основной язык программирования, который обеспечивает высокую производительность и надежность. Java является кроссплатформенным, что позволяет запускать приложение на различных операционных системах без необходимости внесения изменений в код.
* Spring Boot: Фреймворк для создания веб-приложений, который значительно упрощает процесс разработки благодаря встроенной конфигурации и поддержке множества библиотек. Он позволяет быстро разрабатывать приложения с минимальными затратами времени на настройку и поддерживает создание RESTful API, что идеально подходит для нашего проекта.
* MySQL/PostgreSQL: Системы управления базами данных, обеспечивающие надежное хранение и управление данными. Они поддерживают сложные запросы и транзакции, что критично для учета расходов и технического обслуживания автомобилей. PostgreSQL также предлагает расширенные возможности работы с данными и поддержку JSON.
* Thymeleaf: Шаблонизатор для создания веб-интерфейсов, который интегрируется с Spring Boot и позволяет легко создавать динамические страницы. Thymeleaf обеспечивает чистый и понятный синтаксис, что упрощает разработку пользовательского интерфейса.
* Bootstrap: CSS-фреймворк для стилизации интерфейса, который помогает быстро создавать адаптивные и привлекательные веб-страницы. Он предоставляет готовые компоненты и сетки, что значительно ускоряет процесс разработки.
* jQuery: Библиотека JavaScript, которая упрощает манипуляции с DOM и обработку событий. jQuery позволяет писать меньше кода для выполнения сложных задач на стороне клиента.
* Spring Security: Модуль безопасности для Spring-приложений, который обеспечивает аутентификацию и авторизацию пользователей. Он защищает приложение от несанкционированного доступа и позволяет управлять ролями пользователей.
* Flyway: Инструмент для управления миграциями базы данных. Flyway позволяет легко обновлять структуру базы данных и управлять версиями схемы, что критично для поддержания целостности данных в процессе разработки.
* OpenCSV: Библиотека для работы с CSV-файлами, используемая для импорта и экспорта данных. Это удобно для миграции данных между системами или для создания отчетов.
* Spring Boot Mail: Библиотека для отправки электронных писем. Она предоставляет простой способ интеграции функциональности отправки почты в приложении, что полезно для уведомлений пользователей о важных событиях.
* Spring Boot Starter Test: Этот стартер включает все необходимые библиотеки для тестирования Spring Boot приложений, такие как JUnit, Mockito и AssertJ. Он упрощает процесс написания тестов, позволяя разработчикам легко создавать юнит-тесты и интеграционные тесты без сложной настройки. Использование этого стартера обеспечивает высокое качество кода за счет автоматизированного тестирования функциональности приложения.
* H2 Database: Легковесная встроенная база данных, используемая в процессе разработки и тестирования. H2 позволяет быстро проверять функциональность приложения без необходимости настройки полноценной базы данных. Она идеально подходит для юнит-тестов и локальной разработки благодаря своей простоте использования.
* Springdoc OpenAPI Starter WebMVC UI: Этот инструмент автоматически генерирует документацию API на основе аннотаций в коде. Он упрощает создание пользовательской документации для RESTful сервисов, что делает API более доступным для разработчиков. Это особенно полезно при работе с внешними системами или при предоставлении API другим разработчикам.
* Spring Boot DevTools: Набор инструментов для улучшения процесса разработки. Он включает функции автоматической перезагрузки приложения при изменении кода, что значительно ускоряет цикл разработки. Это позволяет разработчикам быстрее видеть результаты своих изменений без необходимости вручную перезапускать сервер.

## **Теоретические основы**

В приложении RoadAnalyzer используются несколько ключевых концепций:

* RESTful API: Архитектурный стиль проектирования API, который использует HTTP-запросы (GET, POST, PUT, DELETE) для взаимодействия с ресурсами. RESTful API обеспечивает простоту использования и масштабируемость приложения. Он позволяет клиентам взаимодействовать с сервером через стандартные HTTP-запросы, что делает интеграцию с другими системами более удобной. Поддержка формата JSON упрощает обмен данными между клиентом и сервером.
* Работа с базами данных: Приложение использует реляционные базы данных (MySQL/PostgreSQL) для хранения информации о расходах, техническом обслуживании и пользователях. SQL-запросы позволяют эффективно управлять данными и выполнять сложные операции по выборке информации. Использование Flyway обеспечивает автоматическое управление миграциями базы данных, что упрощает обновление структуры данных и поддерживает целостность базы.
* Аутентификация и авторизация: Эти процессы обеспечивают безопасность приложения. Аутентификация подтверждает личность пользователя (например, через логин и пароль), а авторизация определяет его права доступа к различным ресурсам приложения. Использование Spring Security позволяет легко реализовать эти функции и защищает приложение от несанкционированного доступа.
* Отправка электронной почты: В приложении используется библиотека Spring Boot Mail для отправки электронных писем. Эта функциональность применяется для верификации новых пользователей, что позволяет обеспечить безопасность регистрации и предотвратить создание фальшивых аккаунтов. Пользователи получают письмо с подтверждением, что повышает уровень доверия к приложению.
* Работа с CSV-файлами: Библиотека OpenCSV используется для работы с CSV-файлами, что позволяет удобно импортировать и экспортировать данные. Это особенно полезно при миграции данных с помощью Flyway, а также для создания отчетов или резервного копирования информации. Пользователи могут загружать данные о расходах или техническом обслуживании из CSV-файлов, что упрощает процесс ввода информации в систему.
* Шаблонизация интерфейса: Использование Thymeleaf позволяет динамически генерировать HTML-страницы на основе данных из базы данных. Это улучшает взаимодействие пользователя с приложением и делает интерфейс более интерактивным.
* Тестирование приложения: Библиотека Spring Boot Starter Test предоставляет инструменты для написания юнит-тестов и интеграционных тестов приложения. Это обеспечивает высокое качество кода за счет автоматизированного тестирования функциональности и позволяет выявлять ошибки на ранних стадиях разработки.

Эти технологии и концепции обеспечивают надежную основу для разработки приложения RoadAnalyzer, позволяя эффективно управлять данными о расходах и техническом обслуживании автомобилей, а также обеспечивать безопасность пользователей и удобство работы с данными.

## **Практическая Часть**

## 

**Сущности базы данных**:

## USERS - Пользователи

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор пользователя |
| username | VARCHAR | Имя пользователя |
| password | VARCHAR | Пароль пользователя (в зашифрованном виде) |
| role | VARCHAR | Роль пользователя (например, admin/user) |
| email | VARCHAR | Электронная почта пользователя |
| database\_identifier | VARCHAR | Идентификатор для связи с базой данных |
| created\_at | DATE | Дата создания пользователя |
| verification\_code | VARCHAR | Код для подтверждения электронной почты |
| is\_active | BOOLEAN | Статус подтвержденного пользователя |

## 

## SESSIONS - Сессии пользователей

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор записи |
| session\_id | VARCHAR | Открытый идентификатор сессии (например, токен, хранящийся в куках) |
| user\_id | INT (FK) | Идентификатор пользователя (ссылка на USERS) |
| created\_at | DATE | Дата и время создания сессии |
| last\_accessed\_at | DATE | Дата и время последнего доступа пользователя |
| is\_active | BOOLEAN | Статус активности сессии (false - завершена; true - активна) |

## SPENDING - Расходы

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор расхода |
| date | DATE | Дата, когда был произведен расход |
| part\_and\_service\_id | INT (FK) | Идентификатор связанного товара или услуги (ссылка на таблицу PART\_AND\_SERVICE) |
| description | VARCHAR | Описание расхода (например, детали услуги или комментарий к расходу) |
| count | DOUBLE | Количество единиц товара или услуг, связанных с расходом |
| amount | DOUBLE | Общая сумма расхода |

## PART\_AND\_SERVICE - Товары и услуги

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор товара или услуги |
| description | VARCHAR | Описание товара или услуги |
| type | INT (FK) | Идентификатор типа расхода (ссылка на таблицу EXPENSE\_TYPE) |

## EXPENSE\_TYPE - Тип расхода

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор типа расхода |
| description | VARCHAR | Описание типа расхода (например, "топливо", "обслуживание") |

## INFO - Информация автомобиля

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор записи |
| key\_info | VARCHAR | Наименование ключа записи (например, "год выпуска") |
| value\_info | VARCHAR | Значение ключа записи (например, "2020") |

## PARTS\_STOCK - Запчасти на складе (в наличии)

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор записи |
| spending\_id | INT (FK) | Идентификатор связанного расхода |
| date | DATE | Дата поступления запчасти на склад |
| part\_description | VARCHAR | Название запчасти |
| description | VARCHAR | Дополнительное описание запчасти |
| oem | VARCHAR | Оригинальный номер запчасти |
| count | DOUBLE | Количество запчастей на складе |
| price | DOUBLE | Цена за одну единицу запчасти |
| amount | DOUBLE | Общая сумма стоимости запчастей на складе |
| status | VARCHAR | Статус запчасти ("установлена", "в наличии") |
| part\_group\_id | INT (FK) | Идентификатор категории запчасти (ссылка на таблицу PART\_GROUP) |

## PART\_GROUP - Категории запчастей

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор категории запчастей |
| description | VARCHAR | Описание категории (например, "тормоза", "освещение") |

## MAINTENANCE\_SCHEDULE - Регламентное обслуживание

| **Поле** | **Тип данных** | **Описание** |
| --- | --- | --- |
| id | INT (PK) | Уникальный идентификатор записи о регламентном обслуживании |
| title | VARCHAR | Наименование регламентного обслуживания |
| description | VARCHAR(500) | Подробное описание работ, которые необходимо выполнить |
| requiredMaterials | TEXT | Список необходимых материалов для выполнения работ |
| mileageInterval | INT | Период регламентного обслуживания по пробегу (км) |
| timeInterval | INT | Период регламентного обслуживания по времени (дни) |
| lastExecutionDate | DATE | Дата последнего выполнения регламентного обслуживания |
| lastOdometerReading | INT | Показания одометра при проведении последней регламентной работы |
| cost | DOUBLE | Стоимость проведения работ |
| notes | VARCHAR | Дополнительные заметки или комментарии |

## 

## 

## Доступ к базе данных

Для удобного доступа к базе данных вы можете использовать встроенный H2 Console. Это веб-интерфейс, который позволяет подключаться к базе данных и выполнять SQL-запросы для проверки и управления данными. H2 Console предоставляет возможность:

* Выполнять произвольные SQL-запросы для выборки, обновления и удаления данных.
* Просматривать структуру таблиц и их содержимое.
* Удобно тестировать запросы и проверять корректность работы приложения.

**Эндпоинты API:**

## **Аутентификация**

## POST /auth

Эндпоинт для аутентификации и регистрации пользователя.

* Если передан параметр email, происходит регистрация нового пользователя.
* Если параметр email не передан, выполняется авторизация существующего пользователя.

## Параметры запроса:

* Тело запроса: (String, обязательный): Логин пользователя.

## Пример запроса:

POST /auth

Content-Type: application/json

{

"login": "user123",

"pswd": "securepassword",

"email": "user@example.com"

}

## Ответы:

Успешный ответ (201 CREATED):

{

"success": "Пользователь успешно зарегистрирован."

}

Ошибка регистрации (409 CONFLICT):

{

"message": "Пользователь с таким логином уже существует."

}

Ошибка сервера (500 INTERNAL SERVER ERROR):

{

"message": "Произошла ошибка. Попробуйте позже."

}

Успешный ответ (200 OK):

{

"redirect": "/dashboard"

}

Ошибка аутентификации (401 UNAUTHORIZED):

{

"message": "Неверный логин или пароль."

}

## Описание ответов:

1. 201 CREATED: Возвращается, когда новый пользователь успешно зарегистрирован. В ответе содержится сообщение о успешной регистрации.
2. 409 CONFLICT: Возвращается, если при регистрации возникла ошибка, например, если пользователь с таким логином уже существует. Сообщение объясняет причину конфликта.
3. 500 INTERNAL SERVER ERROR: Возвращается в случае непредвиденной ошибки на сервере. Это может произойти по различным причинам, и пользователю предлагается попробовать позже.
4. 200 OK: Возвращается при успешной аутентификации существующего пользователя. В ответе может содержаться URL для перенаправления пользователя на главную страницу или панель управления.
5. 401 UNAUTHORIZED: Возвращается, если аутентификация не удалась из-за неверного логина или пароля. Сообщение информирует пользователя о некорректных учетных данных.

## **Выход**

## POST /logout

Эндпоинт для выхода пользователя из системы.

* Этот эндпоинт завершает сессию пользователя и очищает все связанные данные.
* Параметры запроса отсутствуют. После успешного выхода пользователь будет перенаправлен на главную страницу.

## Пример запроса:

POST /logout

## Ответы:

Успешный ответ (200 OK):

{

"message": "Вы успешно вышли из системы.",

"redirect": "/"

}

Ошибка выхода (500 INTERNAL SERVER ERROR):

{

"message": "Произошла ошибка при выходе. Попробуйте позже."

}

## Описание ответов:

1. 200 OK: Возвращается при успешном выходе пользователя из системы. В ответе содержится сообщение о том, что пользователь успешно вышел, а также URL для перенаправления на главную страницу.
2. 500 INTERNAL SERVER ERROR: Возвращается в случае непредвиденной ошибки на сервере во время процесса выхода. Это может произойти по различным причинам, и пользователю предлагается попробовать позже.

## **Подтверждение электронной почты**

## POST /verify

Эндпоинт для подтверждения электронной почты пользователя.

* Этот эндпоинт принимает код подтверждения, отправленный на электронную почту пользователя, и проверяет его.

## 

## Параметры запроса:

* Тело запроса: (String, обязательный): Код подтверждения, который пользователь получил на свою электронную почту.

## Пример запроса

POST /verify

Content-Type: application/json

{

"confirmationCode": "123456"

}

## Ответы:

Успешный ответ (200 OK):Описание: Возвращается при успешной верификации. Тело ответа пустое.

Ошибка верификации (400 BAD\_REQUEST):

{

"message": "Неверный код верификации."

}

## Описание ответов

1. 200 OK: Возвращается при успешной верификации кода подтверждения. Это означает, что введенный пользователем код совпадает с тем, который был отправлен на его электронную почту. В этом случае тело ответа пустое, так как нет необходимости передавать дополнительную информацию.
2. 400 BAD\_REQUEST: Возвращается, если предоставленный код подтверждения неверен. Сообщение информирует пользователя о том, что код не соответствует ожидаемому значению и предлагает проверить введенные данные.

## **Информация о сессии**

## GET /session-info

Эндпоинт для получения информации о текущей сессии пользователя.

* Этот эндпоинт возвращает данные о пользователе и его сессии, включая идентификатор пользователя, идентификатор сессии и статус активности.

## Пример запроса:

GET /session-info

## Ответы:

Успешный ответ (200 OK):

{

"userId": 1,

"sessionId": "35861F6236E28E4F49E88472B3906F52",

"email": "user@example.com\_1",

"databaseId": "user\_db\_1",

"createdAt": "2024-11-21T10:16:43.926171",

"active": false

}

## Описание ответа:

1. 200 OK: Возвращается при успешном запросе к эндпоинту. В теле ответа содержится следующая информация:

* userId: Уникальный идентификатор пользователя, который позволяет системе идентифицировать конкретного пользователя.
* sessionId: Идентификатор текущей сессии, который используется для управления состоянием сессии.
* email: Электронная почта пользователя, что может быть полезно для отображения информации о пользователе.
* databaseId: Идентификатор пользователя в базе данных, который может использоваться для связи с другими таблицами или данными.
* createdAt: Дата и время создания текущей сессии, что позволяет отслеживать время активности пользователя.
* active: Булевое значение, указывающее на статус активности сессии (например, true - активна, false - завершена).
* (LocalDate, необязательный): Дата, до которой будут включены расходы.

## 

## **Работа с расходами**

## **Получение всех расходов**

## GET /api/spending

Эндпоинт для получения списка всех расходов.

* Этот эндпоинт возвращает список расходов с возможностью фильтрации по дате создания.

## Параметры запроса:

* Путь: (LocalDate, необязательный): Дата, до которой будут включены расходы.

## 

## Пример запроса:

GET /api/spending?createdAtBefore=2024-01-01&createdAtAfter=2023-01-01

## Ответы:

Успешный ответ (200 OK):

[

{

"id": 1,

"date": "2023-10-15",

"partAndServiceId": 1,

"description": "",

"count": 10,

"amount": 20

},

{

"id": 2,

"date": "2023-10-16",

"partAndServiceId": 2,

"description": "",

"count": 5,

"amount": 20

}

]

## 

## Описание ответа:

1. 200 OK: Возвращается при успешном запросе к эндпоинту. В теле ответа содержится массив объектов, каждый из которых представляет отдельный расход. Поля включают:

* id: Уникальный идентификатор расхода.
* date: Дата, когда был произведен расход.
* partAndServiceId: Идентификатор связанного товара или услуги.
* description: Описание расхода (может быть пустым).
* count: Количество единиц товара или услуг, связанных с данным расходом.
* amount: Общая сумма расхода.

## **Получение всех расходов (подробно)**

## GET /api/spending/full

Эндпоинт для получения списка всех расходов в подробном формате.

* Этот эндпоинт возвращает подробный список расходов с возможностью фильтрации по дате создания.

## Параметры запроса

* Путь: (LocalDate, необязательный): Дата, до которой будут включены расходы.

## Пример запроса:

GET /api/spending/full?createdAtBefore=2024-01-01&createdAtAfter=2023-01-01

## 

## 

## 

## Ответы:

Успешный ответ (200 OK):

[

{

"id": 1,

"date": "2023-10-15",

"partAndServiceId": 1,

"description": "",

"count": 10,

"amount": 20,

"partDescription": "Болт М4",

"partType": 3,

"partTypeDescription": "запчасти"

},

{

"id": 2,

"date": "2023-10-16",

"partAndServiceId": 2,

"description": "",

"count": 5,

"amount": 20,

"partDescription": "Болт М5",

"partType": 3,

"partTypeDescription": "запчасти"

}

]

## Описание ответа:

1. 200 OK: Возвращается при успешном запросе к эндпоинту. В теле ответа содержится массив объектов, каждый из которых представляет отдельный расход с подробной информацией. Поля включают:

* id: Уникальный идентификатор расхода.
* date: Дата, когда был произведен расход.
* partAndServiceId: Идентификатор связанного товара или услуги.
* description: Описание расхода (может быть пустым).
* count: Количество единиц товара или услуг, связанных с данным расходом.
* amount: Общая сумма расхода.
* partDescription: Название запчасти или услуги, связанной с расходом.
* partType: Идентификатор типа запчасти или услуги (например, категория).
* partTypeDescription: Описание типа запчасти или услуги.

## **Создание нового расхода**

## POST /api/spending

Эндпоинт для создания нового расхода.

* Этот эндпоинт принимает данные о расходе в подробном формате и создает новый расход в базе данных, или обновляет существующий.

## Параметры запроса:

* Тело запроса: (SpendingDTO, обязательный): Объект, содержащий информацию о новом расходе.

## Пример запроса:

## POST /api/spending

## Content-Type: application/json

## {

## "date": "2023-12-20",

## "partAndServiceId": 1,

## "description": "",

## "count": 2.0,

## "amount": 100.0,

## "partDescription": "Болт М5",

## "partType": 3,

## "partTypeDescription": "запчасти"

## }

## Ответы:

## Успешный ответ (201 CREATED):

## {

## "message": "Spending saved successfully",

## "id": 2

## }

## Ошибка (400 BAD\_REQUEST):

## {

## "message": "Не удалось создать расходы: наименование не задано",

## "id": null

## }

## Ошибка (500 INTERNAL SERVER ERROR):

## {

## "message": "Не удалось создать расход: ошибка создания нового наименования",

## "id": null

## }

## Описание ответов:

## 201 CREATED: Возвращается при успешном создании нового расхода. В теле ответа содержится сообщение о том, что расходы успешно сохранены, а также идентификатор созданного расхода.

## 400 BAD\_REQUEST: Возвращается в случае ошибки валидации данных при создании расхода. Сообщение информирует пользователя о том, что наименование запчасти или услуги не задано, что является обязательным полем.

## 500 INTERNAL SERVER ERROR: Возвращается в случае непредвиденной ошибки на сервере во время создания нового расхода. Сообщение указывает на возникшую ошибку и рекомендует пользователю попробовать позже.

## **Удаление расхода**

## DELETE /api/spending/{id}

## Эндпоинт для удаления расхода по идентификатору.

## Этот эндпоинт удаляет расход из системы по указанному идентификатору.

## Параметры запроса

## Путь: (Long, обязательный): Идентификатор расхода, который необходимо удалить.

## Пример запроса:

## DELETE /api/spending/1

## Ответы:

## Успешный ответ (204 NO CONTENT):Описание: Возвращается при успешном удалении расхода. Тело ответа пустое.

## Описание ответа:

## 204 NO CONTENT: Возвращается при успешном удалении расхода. Это означает, что запрашиваемый ресурс был успешно удален, и нет необходимости возвращать дополнительную информацию в теле ответа.

## 

## 

## 

## **Работа с запчастями и услугами**

## **Получение всех запчастей и услуг**

## GET /api/parts-and-service

## Эндпоинт для получения списка всех частей и услуг.

## Этот эндпоинт возвращает список всех доступных запчастей и услуг из базы данных.

## Пример запроса:

## GET /api/parts-and-service

## Ответы:

## Успешный ответ (200 OK):

## [

## {

## "id": 1,

## "description": "Болт М4",

## "type": 3

## },

## {

## "id": 2,

## "description": "Болт М5",

## "type": 3

## }

## ]

## Описание ответа:

## 200 OK: Возвращается при успешном запросе к эндпоинту. В теле ответа содержится массив объектов, каждый из которых представляет отдельную запчасть или услугу. Поля включают:

* id: Уникальный идентификатор запчасти или услуги.
* description: Описание запчасти или услуги, что позволяет пользователю понять, что именно представлено в списке.
* type: Идентификатор типа запчасти или услуги, который может использоваться для дальнейшей классификации или фильтрации.

## **Создание новой запчасти или услуги**

## POST /api/parts-and-service

Эндпоинт для создания новой запчасти или услуги.

* Этот эндпоинт принимает данные о запчасти или услуге и создает новую запись в системе, или обновляет существующую.

## Параметры запроса:

* Тело запроса: (PartAndService, обязательный): Объект, содержащий информацию о новой запчасти или услуге.

## Пример запроса:

POST /api/parts-and-service

Content-Type: application/json

{

"id": 1,

"description": "Новое описание",

"type": 3

}

## Ответы:

Успешный ответ (201 CREATED):

{

"message": "Part and service created successfully",

"id": 2

}

Ошибка (400 BAD\_REQUEST):

{

"message": "Failed to create part and service: Invalid data or conflict",

"id": null

}

Описание ответов:

1. 201 CREATED: Возвращается при успешном создании новой запчасти или услуги. В теле ответа содержится сообщение о том, что запись успешно создана, а также идентификатор созданной записи.
2. 400 BAD\_REQUEST: Возвращается в случае ошибки валидации данных при создании запчасти или услуги. Сообщение информирует пользователя о том, что данные некорректны или возник конфликт при создании записи.

## 

## **Удаление запчасти или услуги**

## DELETE /api/parts-and-service/{id}

Эндпоинт для удаления запчасти или услуги по идентификатору.

* Этот эндпоинт удаляет запись о запчасти или услуге из системы по указанному идентификатору.

## Параметры запроса:

* Путь: (Long, обязательный): Идентификатор запчасти или услуги, которую необходимо удалить.

## Пример запроса:

DELETE /api/parts-and-service/1

## Ответы:

Успешный ответ (204 NO CONTENT):Описание: Возвращается при успешном удалении запчасти или услуги. Тело ответа пустое.

## Описание ответа:

1. 204 NO CONTENT: Возвращается при успешном удалении запчасти или услуги. Это означает, что запрашиваемый ресурс был успешно удален, и нет необходимости возвращать дополнительную информацию в теле ответа.

## **Работа с типами запчастей или услуг**

## Получение всех типов запчастей или услуг

## GET /api/expense\_type

Эндпоинт для получения списка всех типов запчастей или услуг.

* Этот эндпоинт возвращает список всех доступных типов запчастей или услуг в системе.

## Пример запроса:

GET /api/expense\_type

## Ответы:

Успешный ответ (200 OK):

[

{

"id": 3,

"description": "запчасти"

},

{

"id": 1,

"description": "топливо"

},

{

"id": 2,

"description": "услуги"

}

]

## Описание ответа:

1. 200 OK: Возвращается при успешном запросе к эндпоинту. В теле ответа содержится массив объектов, каждый из которых представляет отдельный тип запчасти или услуги. Поля включают:

* id: Уникальный идентификатор типа запчасти или услуги.
* description: Описание типа, что позволяет пользователю понять, к какому классу относится данный элемент (например, "запчасти", "топливо", "услуги").

## **Создание нового типа запчасти или услуги**

## POST /api/expense\_type

Эндпоинт для создания нового типа запчасти или услуги.

* Этот эндпоинт принимает данные о типе запчасти или услуги и создает новую запись в системе, или изменяет существующую.

## Параметры запроса:

* Тело запроса: (ExpenseType, обязательный): Объект, содержащий информацию о новом типе расхода.

## Пример запроса:

POST /api/expense\_type

Content-Type: application/json

{

"description": "новый тип"

}

## Ответы:

Успешный ответ (201 CREATED):

{

"message": "Expense type created successfully",

"id": 4

}

Ошибка (400 BAD\_REQUEST):

{

"message": "Failed to create expense type: Invalid data or conflict",

"id": null

}

## Описание ответов:

1. 201 CREATED: Возвращается при успешном создании нового типа запчасти или услуги. В теле ответа содержится сообщение о том, что тип расхода успешно создан, а также идентификатор созданной записи.
2. 400 BAD\_REQUEST: Возвращается в случае ошибки валидации данных при создании типа. Сообщение информирует пользователя о том, что данные некорректны или возник конфликт при создании записи.

## 

## **Работа с прочей информацией**

## **Получение всей информации**

## GET /api/info

Эндпоинт для получения списка всей информации.

* Этот эндпоинт возвращает список всех доступных информационных записей в системе.

## Пример запроса:

GET /api/info

## Ответы:

Успешный ответ (200 OK):

[

{

"id": 1,

"keyInfo": "марка авто",

"valueInfo": "Марка"

},

{

"id": 2,

"keyInfo": "модель авто",

"valueInfo": "Модель"

},

{

"id": 3,

"keyInfo": "дата выпуска",

"valueInfo": ""

}

]

## Описание ответа:

1. 200 OK: Возвращается при успешном запросе к эндпоинту. В теле ответа содержится массив объектов, каждый из которых представляет отдельную информационную запись. Поля включают:

* id: Уникальный идентификатор записи.
* keyInfo: Наименование ключа информации, которое описывает тип данных (например, "марка авто", "модель авто").
* valueInfo: Значение ключа информации, которое может содержать конкретные данные или быть пустым, если значение не задано.

## **Получение информации по идентификатору**

## GET /api/info/{id}

Эндпоинт для получения информации по указанному идентификатору.

* Этот эндпоинт возвращает информацию о конкретной записи на основе переданного идентификатора.

## Параметры запроса

* Путь: (Long, обязательный): Идентификатор информационной записи, которую необходимо получить.

## Пример запроса:

GET /api/info/1

## Ответы:

Успешный ответ (200 OK):

{

"id": 1,

"keyInfo": "марка авто",

"valueInfo": "Марка"

}

Ошибка (404 NOT FOUND): Описание: Возвращается, если информация по данному идентификатору не найдена. Тело ответа пустое.

## 

## Описание ответов:

1. 200 OK: Возвращается при успешном запросе к эндпоинту. В теле ответа содержится объект, представляющий информацию о запрашиваемой записи. Поля включают:id
2. 404 NOT FOUND: Возвращается, если запись с указанным идентификатором не найдена в системе. Это означает, что пользователь запрашивает информацию о несуществующем объекте.

## **Сохранение новой информации**

## POST /api/info

Эндпоинт для сохранения новой информационной записи.

* Этот эндпоинт принимает данные о новой записи и сохраняет их в системе.

## Параметры запроса

* Тело запроса: (Info, обязательный): Объект, содержащий информацию о новой записи.

## Пример запроса:

POST /api/info

Content-Type: application/json

{

"keyInfo": "Цвет машины",

"valueInfo": "белый"

}

## Ответы:

Успешный ответ (200 OK):

{

"id": 4,

"keyInfo": "Цвет машины",

"valueInfo": "белый"

}

Ошибка (400 BAD REQUEST):Описание: Возвращается, если сохранить запись не удалось. Тело ответа пустое.

## Описание ответов:

1. 200 OK: Возвращается при успешном сохранении новой информационной записи. В теле ответа содержится объект с идентификатором созданной записи, а также ключом и значением информации.

* id: Уникальный идентификатор новой записи.
* keyInfo: Наименование ключа информации.
* valueInfo: Значение ключа информации.

1. 400 BAD REQUEST: Возвращается в случае ошибки валидации данных при попытке сохранить новую запись. Это может произойти, если одно из обязательных полей не заполнено или данные некорректны. В этом случае тело ответа пустое.

## **Тестирование эндпоинтов с помощью Swagger**

К проекту подключен Swagger, который служит для автоматической генерации документации и упрощения тестирования эндпоинтов API.

1. Доступ к Swagger UI: После запуска Spring Boot приложения вы можете получить доступ к интерфейсу Swagger по следующему URL: http://localhost:8080/swagger-ui/index.html.
2. Навигация по документации: В интерфейсе Swagger представлен список всех доступных эндпоинтов API. Разверните каждый эндпоинт, чтобы увидеть его описание, параметры запроса и возможные ответы.
3. Тестирование эндпоинтов: Используйте интерфейс для отправки запросов к API. Заполните необходимые параметры и нажмите кнопку "Execute", чтобы увидеть ответ от сервера.

Интеграция Swagger в приложение значительно упрощает процесс разработки и тестирования API, позволяя разработчикам быстро проверять функциональность без необходимости использовать сторонние инструменты, такие как Postman или cURL. Это делает Swagger не только полезным инструментом для документирования, но и мощным средством для взаимодействия с API в реальном времени.

## 

## **Настройки приложения**

В приложении RoadAnalyzer используется файл конфигурации application.yml для управления настройками. Этот файл позволяет удобно настраивать различные параметры, такие как конфигурация базы данных, параметры безопасности, настройки почтового сервера и другие важные аспекты приложения.

## Пример структуры application.yml

spring:

thymeleaf:

cache: false # Отключение кеширования шаблонов Thymeleaf

application:

name: roadanalyzer # Название приложения

datasource:

url: jdbc:h2:mem:test # URL для подключения к базе данных H2 в памяти

username: sa # Имя пользователя для базы данных

password: # Пароль (пустой для H2)

driver-class-name: org.h2.Driver # Драйвер для H2

h2:

console:

enabled: true # Включает консоль H2

path: /h2-console # Путь к консоли H2

jpa:

show-sql: true # Выводить SQL-запросы в консоль

hibernate:

ddl-auto: none # Не управлять схемой базы данных автоматически

flyway:

enabled: true # Включает миграции Flyway

baseline-on-migrate: true # Устанавливает базовую версию при миграции

placeholders:

proto\_user\_db: admin\_db # Плейсхолдер для миграций Flyway

mail:

verification-required: false # Не требуется подтверждение почты

host: smtp.rambler.ru # SMTP-сервер для отправки почты

port: 587 # Порт SMTP-сервера

username: YOUR\_EMAIL@gmail.com # Учетная запись электронной почты

password: YOUR\_APP\_PASSWORD # Пароль для учетной записи электронной почты

properties:

mail.smtp.auth: true # Требуется аутентификация на SMTP-сервере

mail.smtp.starttls.enable: true # Включает TLS для шифрования соединения

mail.smtp.ssl.enable: false # Отключает SSL (если не требуется)

# mail.debug: true # Включение отладки почтового протокола (закомментировано)

# mail.smtp.ssl.trust: smtp.rambler.ru # Доверенный сертификат для SSL (закомментировано)

# mail.transport.protocol: smtp # Протокол передачи (закомментировано)

config:

import: "optional:classpath:application-secret.yml" # Импорт дополнительных секретных настроек

logging:

level:

org:

springframework:

security: DEBUG # Уровень логирования для Spring Security

server:

servlet:

session:

timeout: 30d # Время жизни сессии (30 дней)

cookie:

max-age: 30d # Время жизни cookies (30 дней)

## 

## Описание ключевых разделов

## Конфигурация Spring

* thymeleaf.cache: Отключает кеширование шаблонов Thymeleaf, что полезно при разработке.
* application.name: Указывает название приложения.

## Конфигурация базы данных

* datasource: Строка подключения к базе данных H2 в памяти.
* h2.console: Включает консоль H2 для управления базой данных через веб-интерфейс.
* jpa: Если установлено в
* flyway: Включает миграции Flyway для управления версиями базы данных.

## Настройки почтового сервера

Для настройки отправки электронных писем в приложении RoadAnalyzer рекомендуется использовать почту Rambler, так как она предлагает высокий лимит на отправку сообщений — до 200 писем в час. Чтобы настроить отправку писем через SMTP-сервер Rambler, необходимо добавить соответствующие параметры в файл конфигурации application.yml или application.properties вашего Spring Boot приложения.

* mail: Указывает, требуется ли подтверждение электронной почты.
* properties: Настройки аутентификации и шифрования соединения при отправке почты.

## Логирование

* logging.level.org.springframework.security: Устанавливает уровень логирования для Spring Security на DEBUG, что позволяет отслеживать события безопасности.

## Настройки сервера

* server.servlet.session: Устанавливает время жизни сессии на уровне сервера (30 дней).

## 

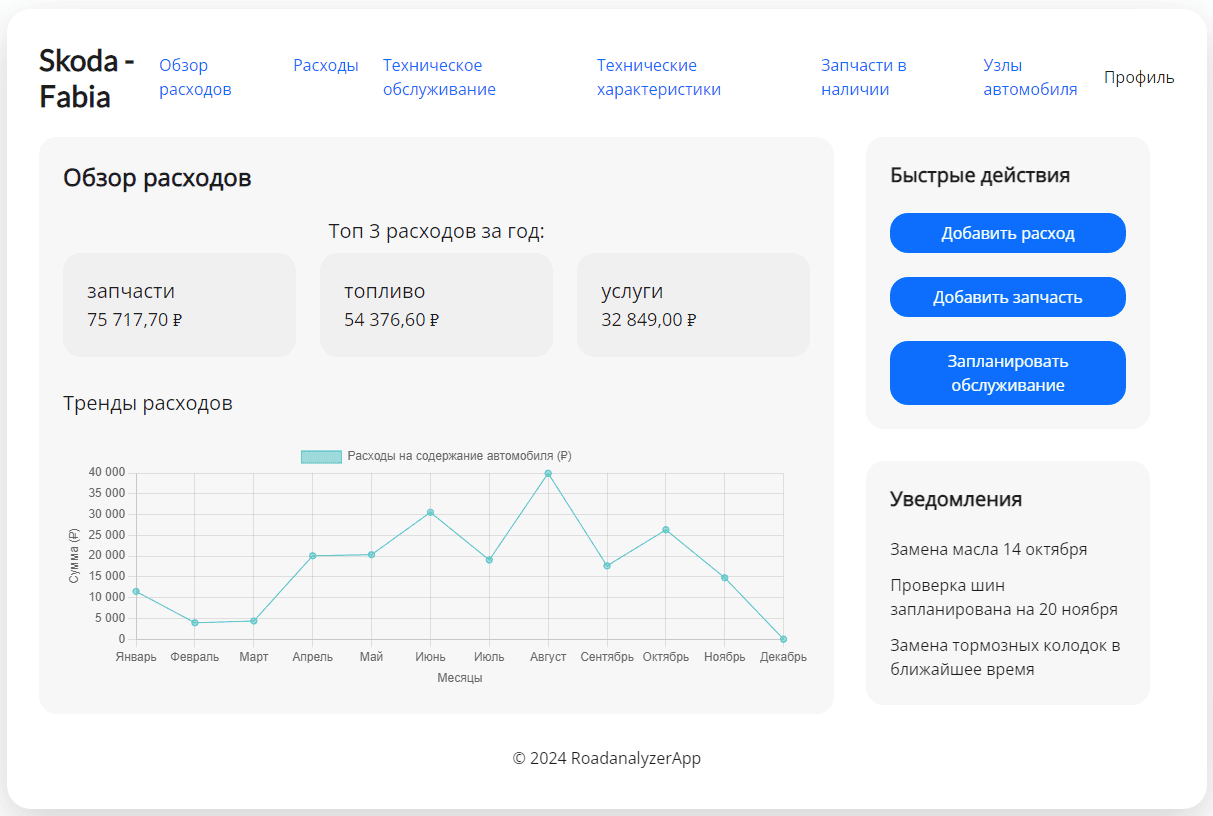
## **Описание фронтенда**

## Структура главной страницы

Главная страница включает в себя:

* Навигацию с ссылками на различные разделы (обзор расходов, техническое обслуживание и т.д.), что позволяет пользователям легко перемещаться по приложению.
* Основной контентный раздел, который загружает данные в зависимости от выбранного таба. Это обеспечивает динамическое обновление информации без необходимости перезагрузки страницы.
* Модальные окна для редактирования информации о марке и модели автомобиля, что улучшает взаимодействие с пользователем.

**Главная страница**

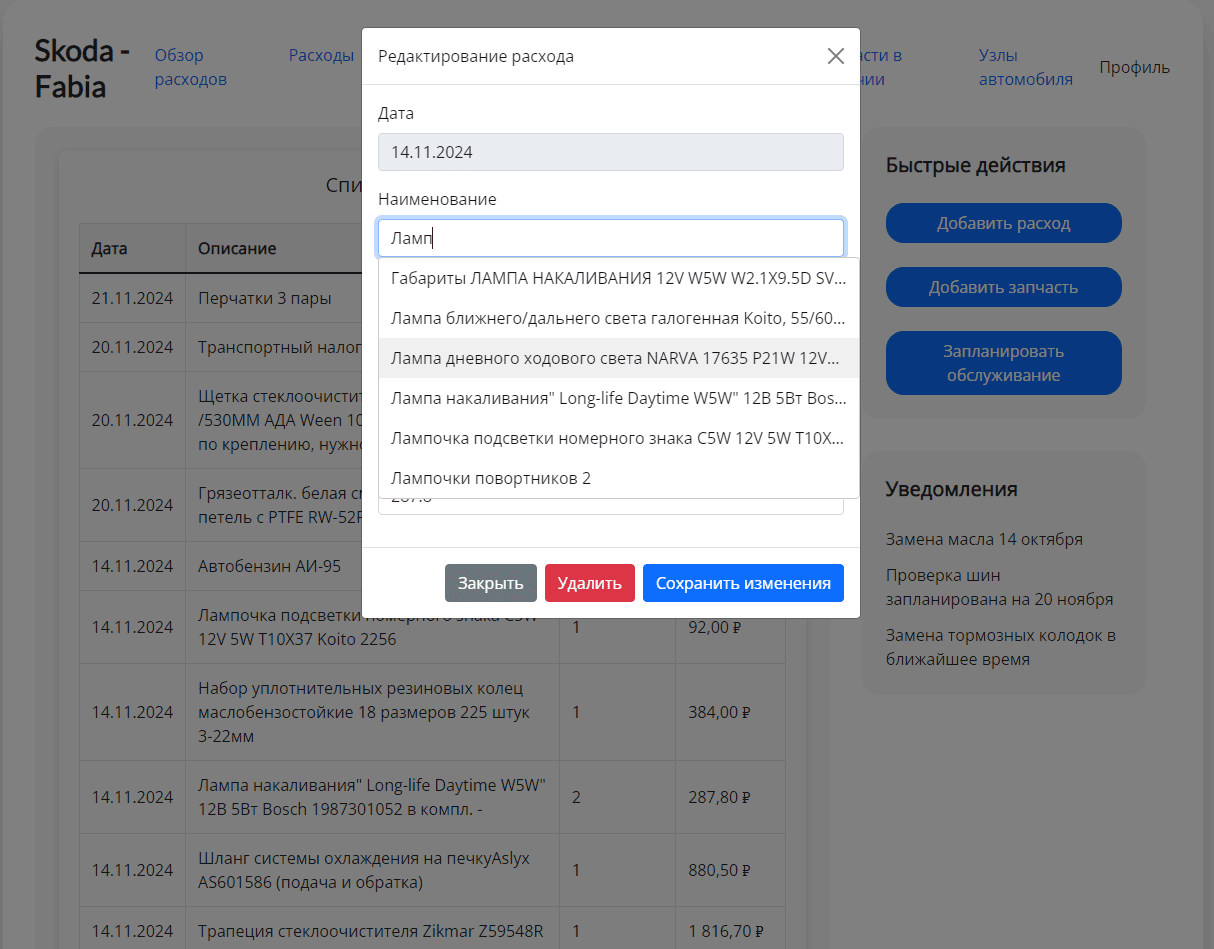


## Интерактивные элементы

Приложение включает интерактивные элементы, такие как:

* Формы для редактирования информации о машине. Формы имеют встроенную валидацию, что помогает избежать ошибок при вводе данных.
* Кнопки для загрузки контента без перезагрузки страницы, что обеспечивает более плавный пользовательский опыт.
* Автозаполнение полей: При вводе новых данных в некоторых полях пользователю предоставляется помощь при вводе. Система автоматически подбирает подходящие данные из базы данных, что упрощает процесс ввода и минимизирует ошибки.

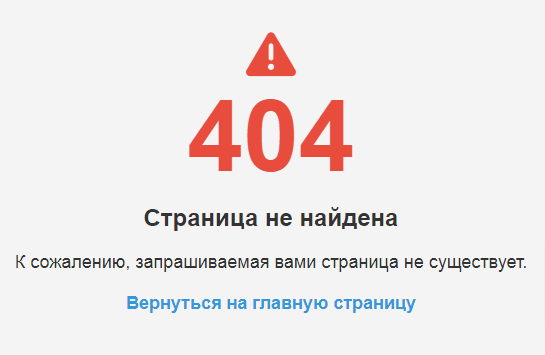
**Новый расход**



## Обработка ошибок

Все ошибки, такие как ответ 404 (страница не найдена), имеют свои шаблоны с простым и понятным стилем. Эти шаблоны не перегружают пользователя лишней информацией и помогают быстро понять, что произошло.

**Ошибка 404**



## Технологии

Для разработки фронтенда приложения RoadAnalyzer используются следующие технологии:

* HTML/CSS/JavaScript: Это основные технологии для создания веб-страниц. HTML используется для структурирования контента, CSS — для стилизации и оформления, а JavaScript — для добавления интерактивности и динамического поведения на страницах.
* jQuery: Библиотека JavaScript, которая упрощает манипуляцию с DOM, обработку событий и выполнение AJAX-запросов. jQuery позволяет быстро и эффективно взаимодействовать с элементами страницы и управлять ими.
* Bootstrap: CSS-фреймворк, который используется для быстрой разработки адаптивных и мобильных веб-приложений. Bootstrap предоставляет готовые компоненты и стили, что значительно ускоряет процесс разработки интерфейса и обеспечивает его современный вид.

## Взаимодействие с API

Фронтенд приложения RoadAnalyzer взаимодействует с бэкендом через HTTP-запросы, используя jQuery:

* jQuery AJAX-запросы: Для выполнения запросов к API используются методы jQuery, такие как $.ajax(), $.get(), и $.post(). Это позволяет отправлять данные на сервер и получать ответы без перезагрузки страницы.
* Обработка ответов: Полученные данные отображаются пользователю в соответствующих компонентах интерфейса. В случае возникновения ошибок (например, при неудачном запросе) используются специальные уведомления или сообщения для информирования пользователя о проблемах.

## Заключение

* Использование HTML, CSS, JavaScript, jQuery и Bootstrap позволяет создать интуитивно понятный и функциональный интерфейс для приложения RoadAnalyzer. Интерактивные элементы и эффективное взаимодействие с бэкендом через AJAX-запросы обеспечивают плавный пользовательский опыт, что делает приложение удобным для пользователей.

## 

## **Принцип работы приложения**

Приложение RoadAnalyzer разработано с учетом удобства пользователей и обеспечения безопасности их данных. При первом входе на страницу происходит несколько ключевых шагов, которые обеспечивают функциональность и изолированность данных для каждого пользователя.

1. Определение текущего пользователя: При загрузке страницы вызывается метод SessionUtils.getUser(request), который определяет, авторизован ли пользователь. Если пользователь не авторизован, создается временная сессия и временный пользователь.
2. Создание временной сессии: Для неавторизованных пользователей создается временный пользователь, которому привязывается новая временная база данных. Это позволяет пользователю работать с приложением без необходимости сразу проходить регистрацию или авторизацию. В рамках текущей сессии пользователь может использовать все функции приложения, такие как добавление расходов, редактирование информации о транспортных средствах и т.д.
3. Регистрация и верификация: Если временный пользователь решает зарегистрироваться, его данные фиксируются в базе данных. После завершения регистрации пользователь получает возможность авторизоваться и работать со своей постоянной базой данных. Это обеспечивает доступ к данным с любого устройства после авторизации.
4. Изолированность данных: Для каждого пользователя в базе данных создается своя группа таблиц. Это обеспечивает изолированность данных при работе с множеством пользователей, что является важным аспектом безопасности и целостности информации. Если одна база данных пользователя повреждена (например, из-за неправильных данных), другие базы данных остаются целыми и доступными для своих владельцев.
5. Экспорт и импорт данных: Каждый пользователь имеет возможность экспортировать свои данные (например, в формате CSV или Excel) для дальнейшего анализа или хранения. Также предусмотрена функция импорта данных, что упрощает процесс переноса информации из других источников.
6. Интерактивный интерфейс: Приложение использует HTML, CSS и JavaScript для создания интуитивно понятного интерфейса. Пользователь может легко взаимодействовать с приложением благодаря формам для ввода данных, кнопкам для выполнения действий и уведомлениям об ошибках или успехах.
7. Взаимодействие с бэкендом: Фронтенд взаимодействует с бэкендом через AJAX-запросы с использованием jQuery, что позволяет пользователю получать актуальную информацию без перезагрузки страницы. Данные отображаются в реальном времени, что улучшает общий пользовательский опыт.

## 

## **Трудности и решения**

В процессе разработки приложения RoadAnalyzer возникло несколько значительных трудностей, каждая из которых требовала тщательного анализа и поиска оптимальных решений. Ниже приведены основные проблемы и способы их преодоления.

## 1. Разделение данных между пользователями

Трудность: Одной из первых проблем, которую пришлось решить, было разделение данных между пользователями. Изначально рассматривался вариант хранения в таблицах принадлежности пользователя к каждой записи. Однако это решение показалось ненадежным по нескольким причинам:

* Безопасность данных: Хранение идентификатора пользователя в каждой записи увеличивает риск несанкционированного доступа к данным. В случае ошибки в логике приложения или SQL-запросах данные одного пользователя могут быть случайно доступны другому.
* Сложность запросов: При таком подходе запросы к базе данных усложняются, так как необходимо постоянно фильтровать данные по идентификатору пользователя, что может привести к ухудшению производительности при большом объеме записей.

Решение: Был выбран путь полной изолированности пользователей, при котором каждому пользователю соответствует своя группа таблиц в базе данных. Это обеспечило высокий уровень безопасности и целостности данных, так как каждый пользователь работает только со своей базой данных. Такой подход также упростил логику запросов и уменьшил вероятность ошибок.

## 2. Отправка электронной почты для верификации пользователей

Трудность: В процессе реализации функции отправки электронной почты для верификации пользователей возникли сложности, связанные с ограничениями многих почтовых сервисов при использовании SMTP-протокола. Многие из них накладывают лимиты на количество отправляемых писем, что может стать проблемой при увеличении числа пользователей.

Решение: После анализа различных вариантов выбор пал на использование почтового сервиса Rambler, который предлагает оптимальное бесплатное решение с высоким лимитом на отправку сообщений — до 200 писем в час. Это позволило обеспечить надежную и стабильную отправку писем без риска превышения лимитов.

## 3. Парсинг CSV-файлов

Трудность: При реализации функции импорта данных возникли сложности с парсингом CSV-файлов. Этот процесс требует специфичного кода под определенный формат таблиц, что усложняет поддержку и расширяемость приложения.

Решение: Был разработан отдельный модуль для обработки CSV-файлов, который можно легко настроить для парсинга любых csv файлов, с разными разделителями и дальнейшей миграции в flyway.

## 

## 4. Работа с JPA

Трудность: Поскольку у каждого пользователя свой идентификатор базы данных и уникальный набор таблиц, стандартные методы работы с JPA не подходили для приложения. Это создало дополнительные трудности при выполнении запросов к базе данных.

Решение: Пришлось решать эту проблему, написав собственные нативные SQL-запросы для работы с данными каждого пользователя. Это дало возможность более гибко управлять запросами и оптимизировать их под конкретные требования приложения.

## 

## **Тестирование**

Тестирование является важной частью разработки приложения RoadAnalyzer, так как оно позволяет обеспечить качество, надежность и функциональность системы. В процессе тестирования использовалось несколько методов и инструментов, включая юнит-тесты и интеграционные тесты.

## 1. Юнит-тесты

Описание: Юнит-тесты предназначены для проверки отдельных компонентов приложения, таких как методы, классы и функции. Они помогают выявить ошибки на ранних стадиях разработки и гарантируют, что каждый элемент работает так, как задумано.

Инструменты:

* JUnit: Основной фреймворк для написания и выполнения юнит-тестов в Java. Были использованы JUnit для создания тестов на методы сервисов и репозиториев приложения.
* Mockito: Библиотека для создания мок-объектов, которая позволяет изолировать тестируемые компоненты от их зависимостей. С помощью Mockito можно имитировать поведение зависимостей (например, базы данных или внешних сервисов), что упрощало тестирование бизнес-логики.

Примеры тестирования:

* Тестирование методов сервиса для проверки правильности обработки данных.
* Проверка исключений при некорректных входных данных.

## 2. Интеграционные тесты

Описание: Интеграционные тесты проверяют взаимодействие между различными компонентами приложения, такими как сервисы, контроллеры и базы данных. Они помогают убедиться, что все части системы работают вместе корректно.

Инструменты:

* Spring Test: Модуль Spring для поддержки интеграционного тестирования. Был использован для конфигурации контекста приложения и запуска тестов с реальными компонентами.

Примеры тестирования:

* Проверка работы REST API с использованием MockMvc для эмуляции HTTP-запросов.
* Тестирование сценариев взаимодействия между контроллерами и сервисами, включая проверку корректности ответов API.

## 3. Тестирование пользовательского интерфейса

Описание: Хотя основное внимание уделялось серверной части приложения, также проводилось тестирование пользовательского интерфейса для проверки его функциональности и удобства использования.

## 4. Тестирование API через Postman и Swagger

Описание: Для проверки работы API использовались инструменты Postman и Swagger. Эти инструменты позволяют выполнять HTTP-запросы к API и анализировать ответы.

* Postman: Использовался для ручного тестирования API. С помощью Postman можно отправлять различные запросы к серверу, изменять параметры и заголовки запросов, а также проверять корректность ответов от сервера. Это облегчало процесс отладки и проверки функциональности отдельных эндпоинтов.
* Swagger: Интегрированный интерфейс Swagger UI позволял визуализировать доступные эндпоинты API и их параметры. Можно быстро отправлять запросы к API прямо из интерфейса Swagger, что значительно ускоряет процесс тестирования и документирования API.

## 

## 

## **Выводы**

В процессе разработки приложения RoadAnalyzer была достигнута значительная цель — создание интуитивно понятного и функционального инструмента для учета расходов и управления техническим обслуживанием автомобилей. Были реализованы ключевые функции, включая регистрацию пользователей, управление данными о транспортных средствах, отслеживание расходов.

Основные достижения проекта включают:

* Безопасность данных: Реализована полная изолированность данных пользователей, что обеспечивает безопасность и целостность информации. Каждый пользователь работает со своей базой данных, что минимизирует риск утечки данных.
* Интерактивный интерфейс: Создан удобный и интуитивно понятный пользовательский интерфейс с использованием современных технологий, таких как HTML, CSS, jQuery и Bootstrap.
* Надежное тестирование: Приложение прошло тестирование, включая юнит-тесты, интеграционные тесты, что обеспечило высокое качество кода и стабильность работы системы.

На основе проведенной работы можно сделать вывод о том, что приложение RoadAnalyzer можно использовать в реальных условиях. Оно предоставляет пользователям все необходимые инструменты для эффективного управления расходами на автомобили и технического обслуживания.

## **Перспективы развития**

Несмотря на достигнутые результаты, проект имеет множество направлений для дальнейшего развития:

1. Расширение функциональности:
   * Аналитика расходов: Внедрение инструментов для визуализации данных о расходах (графики, диаграммы), что поможет пользователям лучше анализировать свои финансовые потоки.
   * Уведомления о предстоящих сервисах: Реализация системы уведомлений о необходимости проведения технического обслуживания на основе пробега или времени.
2. Интеграция с внешними сервисами:
   * Интеграция с банковскими API: Позволит пользователям автоматически импортировать данные о расходах с банковских счетов, упрощая процесс учета.
   * Подключение к сервисам GPS: Для отслеживания маршрутов и автоматического учета расходов на топливо.
3. Мобильное приложение:
   * Разработка мобильной версии приложения для iOS и Android, что позволит пользователям управлять своими данными в любое время и в любом месте.
4. Улучшение системы безопасности:
   * Внедрение многофакторной аутентификации для повышения уровня безопасности учетных записей пользователей.
5. Поддержка нескольких языков:
   * Расширение аудитории за счет добавления поддержки различных языков интерфейса.

## 

## Заключение

Проект RoadAnalyzer представляет собой удобное решение для учета расходов на автомобили и управления техническим обслуживанием. Достигнутые результаты открывают множество возможностей для дальнейшего развития приложения, что позволит улучшить пользовательский опыт и расширить функциональность системы. Думаю, что реализация предложенных направлений, сделает приложение еще более полезным и востребованным среди пользователей.

## **Список Используемых Литературных Источников**

В процессе разработки приложения RoadAnalyzer были использованы различные литературные источники, которые помогли углубить понимание языка Java и принципов программирования. Ниже приведен список ключевых источников:

1. Философия Java (Bruce Eckel)  
   Эта книга является классическим трудом по языку Java и охватывает основные проблемы программирования, подходы к их решению и философию языка. Она помогает читателям понять не только синтаксис Java, но и концептуальные основы, которые делают его уникальным. Книга подходит как для начинающих, так и для опытных программистов, желающих углубить свои знания.  
   ISBN: 978-0132281220
2. Thinking in Java (Bruce Eckel)  
   Еще одно важное произведение Бруса Эккеля, которое рассматривает язык Java в контексте объектно-ориентированного программирования. Книга содержит множество примеров и упражнений, что делает ее отличным ресурсом для изучения языка.  
   ISBN: 978-0131872486
3. Java: A Beginner's Guide (Herb Schildt)  
   Это руководство для начинающих охватывает основы языка Java, включая создание, компиляцию и выполнение программ. Книга также рассматривает более сложные темы, такие как многопоточность и обработка исключений.  
   ISBN: 978-1260463552
4. Effective Java (Joshua Bloch)  
   Эта книга предоставляет рекомендации по лучшим практикам программирования на Java и помогает разработчикам избегать распространенных ошибок. Она полезна как для новичков, так и для опытных программистов, стремящихся улучшить качество своего кода.  
   ISBN: 978-0134686097
5. Java Concurrency in Practice (Brian Goetz)  
   Книга посвящена многопоточности в Java и предлагает глубокое понимание работы с потоками и синхронизацией. Она полезна для разработчиков, работающих с многопоточными приложениями.  
   ISBN: 978-0321349606

Эти источники сыграли важную роль в процессе разработки приложения RoadAnalyzer, предоставляя теоретическую базу и практические рекомендации по использованию языка Java и его возможностей.