ОБЛАСТНОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«КУРСКИЙ ТЕХНИКУМ СВЯЗИ»

|  |  |
| --- | --- |
| Рецензент      /  « » 2024 г | Допущен к защите Зам. директора по учебно- производственной работе  / *В.В. Малинников*  « » 2024 г |

Специальность: 09.02.03 Программирование в компьютерных системах

*(шифр) (наименование специальности)*

Форма обучения: *очная/заочная*

Дипломная работа на тему

Разработка информационной системы для осознанных граждан «Нарушений.нет»

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Корниенко В. С. |
|  | (подпись) |  | Фамилия И.О  « »\_\_\_\_\_\_\_\_\_ 2024г. |
| Руководитель дипломной  работы |  |  | Рыжков В. В. |
|  | (подпись) |  | Фамилия И.О.  « »\_\_\_\_\_\_\_\_\_ 2024г. |
| Нормоконтроль дипломной работы |  |  | Котов С. С. |
|  | (подпись) |  | Фамилия И.О  « »\_\_\_\_\_\_\_\_\_ 2024г. |

Курск, 2024г.

Оглавление

[ГЛАВА 1 Предпроектное обследование 6](#_Toc167997991)

[1.1 Исследование предметной области 6](#_Toc167997992)

[1.2 Формирование требований к программному изделию 7](#_Toc167997993)

[ГЛАВА 2. Техническое задание 10](#_Toc167997994)

[2.1 Постановка задачи 10](#_Toc167997995)

[2.2 Концепция информационной базы и её структура 10](#_Toc167997996)

[2.3 Неформальное описание алгоритма работы ПП 12](#_Toc167997997)

[Блок-схема алгоритма 13](#_Toc167997998)

[2.4 Функции подсистем (модулей) 13](#_Toc167997999)

[2.5 Оценка экономической эффективности от внедрения программы 13](#_Toc167998000)

[ГЛАВА 3 Эскизный проект 15](#_Toc167998001)

[ГЛАВА 4 Разработка проектных решений 16](#_Toc167998002)

[ЗАКЛЮЧЕНИЕ 17](#_Toc167998003)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_Toc167998004)

[ПРИЛОЖЕНИЕ 19](#_Toc167998005)

ВВЕДЕНИЕ

Общая характеристика разработанного программного изделия

В рамках данной дипломной работы разработана информационная система "Нарушений.нет", предназначенная для осознанных граждан, стремящихся оперативно сообщать о нарушениях в общественных местах и получать информацию о действиях, предпринимаемых по этим сообщениям. Система позволяет пользователям отправлять сообщения о нарушениях, отслеживать их статус, а также взаимодействовать с ответственными органами.

В современном обществе важность гражданской активности и участия в общественной жизни постоянно возрастает. Граждане стремятся быть осведомленными и принимать участие в решении проблем, касающихся их повседневной жизни. Информационные технологии предоставляют новые возможности для взаимодействия граждан и государственных органов. Разработка системы "Нарушений.нет" позволяет обеспечить более эффективное и прозрачное взаимодействие между гражданами и ответственными структурами, способствуя быстрому реагированию на нарушения и улучшению качества жизни в обществе.

Целью данного проекта является создание информационной системы, которая позволяет гражданам легко и быстро сообщать о нарушениях, а также отслеживать процесс их устранения. Основные задачи проекта включают:

Разработка удобного и интуитивно понятного интерфейса для пользователей.

Создание надежной и масштабируемой серверной части для обработки сообщений.

Обеспечение безопасности данных пользователей.

Интеграция системы с внешними базами данных и API государственных органов.

Реализация механизма уведомлений о статусе сообщений.

Практическая значимость разработанного программного продукта заключается в повышении эффективности коммуникации между гражданами и государственными органами. Система "Нарушений.нет" позволяет сократить время на подачу и обработку сообщений о нарушениях, повысить прозрачность работы государственных структур и обеспечить обратную связь для граждан. Это, в свою очередь, способствует улучшению качества городской среды и повышению уровня общественного контроля.

Основные отличия от подобных программ

Система "Нарушений.нет" отличается от существующих аналогов следующими ключевыми особенностями:

Интуитивно понятный пользовательский интерфейс, разработанный на базе React.

Мощная и гибкая серверная часть, реализованная с использованием Nest.js.

Интеграция с различными базами данных и API для получения и обработки информации.

Высокий уровень безопасности и защиты данных пользователей.

Возможность масштабирования и адаптации системы под различные нужды и условия.

Программный продукт "Нарушений.нет" создан для граждан, стремящихся активно участвовать в улучшении условий жизни в своем городе. Система предназначена для фиксации и обработки сообщений о нарушениях, таких как несанкционированные свалки, повреждения городской инфраструктуры, нарушения общественного порядка и другие проблемы, требующие вмешательства ответственных служб.

Информационная база системы включает данные о зарегистрированных нарушениях, пользователях, а также информацию, полученную из внешних источников, таких как государственные базы данных и API. Для хранения и обработки данных используется реляционная база данных MySQL, управление которой осуществляется через DBeaver.

Средства разработки

Для разработки системы использовались следующие инструменты и технологии:

**Frontend:** React

**Backend:** Nest.js

**Среда разработки:** Visual Studio Code

**Система управления базами данных:** MySQL

**Инструмент для работы с базами данных:** DBeaver

**Система контроля версий:** GitHub

Система "Нарушений.нет" функционирует в условиях постоянного интернет-соединения и доступна как для настольных, так и для мобильных устройств. Она интегрируется с внешними сервисами через API и обеспечивает высокую степень защиты данных пользователей. Для обеспечения бесперебойной работы системы используется масштабируемая серверная архитектура.

Таким образом, "Нарушений.нет" представляет собой современную и эффективную платформу для взаимодействия граждан с государственными органами, направленную на улучшение качества жизни в городах.

# ГЛАВА 1 Предпроектное обследование

## 1.1 Исследование предметной области

В рамках данного раздела будет проведено исследование предметной области, для которой создается информационная система "Нарушений.нет". Цель исследования – определить основные понятия и функции системы, а также изучить существующие модели и потоки данных.

Описание предметной области

Информационная система "Нарушений.нет" предназначена для взаимодействия граждан с государственными органами и общественными организациями по вопросам выявления и устранения нарушений в общественных местах. Под нарушениями понимаются любые действия или бездействие, которые противоречат нормам и правилам общественного порядка, а также создают угрозу здоровью, безопасности и благополучию граждан. Примеры таких нарушений включают незаконные свалки, повреждения городской инфраструктуры, нарушения правил парковки, шумовые нарушения и т.д.

Функциональная модель системы

Функциональная модель системы "Нарушений.нет" включает следующие основные функции:

Регистрация пользователя – процесс создания учетной записи пользователя, который позволит ему отправлять сообщения о нарушениях и отслеживать их статус.

Подача сообщения о нарушении – форма для ввода информации о нарушении, включая описание, категорию, фотографии и географическое положение.

Обработка сообщения – автоматическая и ручная обработка поступивших сообщений ответственными лицами и органами.

Отслеживание статуса сообщения – возможность для пользователей отслеживать текущий статус своего сообщения.

Уведомления и оповещения – информирование пользователей о изменениях статуса их сообщений и других важных событиях.

Отчеты и статистика – генерация отчетов и статистики по обработанным сообщениям для анализа и принятия управленческих решений.

Моделирование потоков данных

Моделирование потоков данных включает описание движения информации между различными компонентами системы:

Поток данных от пользователя к системе – ввод данных о нарушении через пользовательский интерфейс.

Поток данных внутри системы – передача данных о нарушении от frontend к backend, обработка данных на сервере и сохранение в базе данных.

Поток данных от системы к внешним источникам – отправка данных о нарушениях в соответствующие государственные органы через API.

Поток данных от системы к пользователю – предоставление пользователю информации о статусе его сообщения через интерфейс и уведомления.

Использованные источники

Для исследования предметной области использованы следующие источники:

Законодательные и нормативные акты, регулирующие общественный порядок и ответственность за нарушения.

Существующие информационные системы и платформы для подачи сообщений о нарушениях, такие как "Активный гражданин", "Наш город" и др.

Литература по проектированию и разработке информационных систем, моделированию потоков данных и пользовательских интерфейсов.

## 1.2 Формирование требований к программному изделию

Формирование комплекса требований к программному продукту "Нарушений.нет" включает как функциональные, так и нефункциональные требования, а также регламенты и стандарты, необходимые для его разработки и функционирования.

Функциональные требования

Регистрация и аутентификация пользователей – система должна обеспечивать безопасную регистрацию и аутентификацию пользователей с использованием логина и пароля.

Интерфейс для подачи сообщений – интерфейс должен быть интуитивно понятным, с возможностью добавления текста, фотографий и геолокации.

Система обработки сообщений – система должна автоматически сортировать и классифицировать сообщения, а также направлять их в соответствующие органы.

Отслеживание статуса сообщений – пользователи должны иметь возможность в реальном времени отслеживать статус своих сообщений.

Уведомления – система должна отправлять уведомления пользователям о изменениях статуса их сообщений.

Отчеты и аналитика – система должна генерировать отчеты и предоставлять аналитические данные о количестве и типах нарушений.

Нефункциональные требования

Производительность – система должна обеспечивать быстрый отклик на пользовательские действия и обрабатывать большое количество сообщений без снижения производительности.

Надежность – система должна быть устойчивой к сбоям и обеспечивать сохранность данных при авариях и сбоях в электропитании.

Безопасность – система должна обеспечивать защиту данных пользователей от несанкционированного доступа и утечек.

Совместимость – система должна быть совместима с внешними базами данных и API, а также поддерживать общероссийские классификаторы и унифицированные документы.

Масштабируемость – система должна легко масштабироваться для поддержки большего числа пользователей и увеличения объема данных.

Юридическая значимость – система должна обеспечивать возможность придания юридической силы документам, продуцируемым в результате функционирования программы.

Требования к структуре и функционированию данных

Информационно-логическая схема – данные должны быть структурированы таким образом, чтобы обеспечивать эффективное хранение и обработку информации о нарушениях и пользователях.

Информационный обмен – система должна поддерживать обмен данными между компонентами программного комплекса и внешними системами через стандартизированные интерфейсы.

Защита данных – данные должны быть защищены от разрушений при авариях и сбоях, обеспечиваться регулярное резервное копирование и восстановление данных.

Контроль данных – система должна обеспечивать контроль целостности данных, их обновление и проверку на достоверность.

Таким образом, формирование требований к программному изделию "Нарушений.нет" включает детальное описание функциональных и нефункциональных аспектов системы, обеспечение её надежности, безопасности и совместимости, что позволит создать эффективную и востребованную информационную систему для осознанных граждан.

# ГЛАВА 2. Техническое задание

# 2.1 Постановка задачи

Задача проекта "Нарушений.нет" заключается в разработке информационной системы, которая позволит гражданам сообщать о нарушениях в общественных местах и отслеживать статус их обработки.

Основные цели:

* Обеспечить удобный интерфейс для подачи сообщений о нарушениях.
* Обеспечить надежное хранение и обработку данных.
* Обеспечить уведомления пользователей о статусе их сообщений.
* Обеспечить взаимодействие с внешними системами государственных органов для обработки сообщений.

# 2.2 Концепция информационной базы и её структура

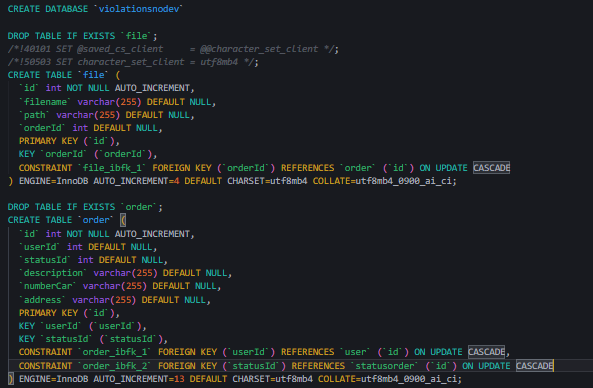
Концепция информационной базы предполагает определение сущностей и связей между ними. Основные сущности включают пользователей, сообщения о нарушениях, файлы, статусы сообщений и данные для аутентификации.

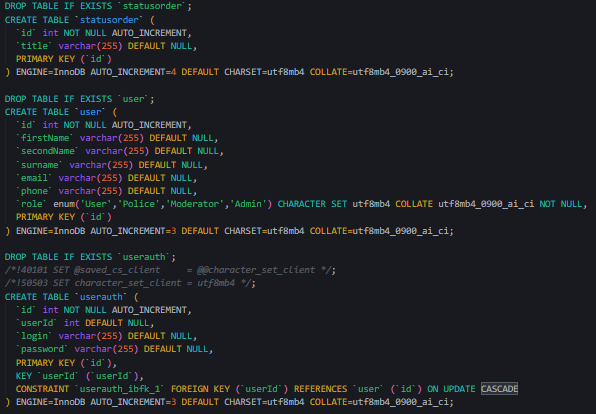
Входная информация

|  |  |  |
| --- | --- | --- |
| Поле | Описание | Тип данных |
| phone | Номер телефона пользователя | VARCHAR(15) |
| firstName | Имя пользователя | VARCHAR(50) |
| secondName | Отчество пользователя | VARCHAR(50) |
| surname | Фамилия пользователя | VARCHAR(50) |
| email | Электронная почта пользователя | VARCHAR(100) |
| role | Роль пользователя | ENUM |
| description | Описание нарушения | TEXT |
| numberCar | Номер автомобиля (если есть) | VARCHAR(20) |
| address | Адрес нарушения | VARCHAR(255) |
| filename | Имя файла | VARCHAR(255) |
| path | Путь к файлу | VARCHAR(255) |
| statusId | Статус сообщения | INT |
| title | Название статуса | VARCHAR(50) |
| login | Логин пользователя | VARCHAR(50) |
| password | Пароль пользователя | VARCHAR(255) |

ER диаграмму логического этапа смотреть в приложении А.  
Выходная информация

|  |  |  |
| --- | --- | --- |
| Поле | Описание | Тип данных |
| status | Статус сообщения | VARCHAR(50) |
| notification | Уведомление пользователю | TEXT |
| report | Отчет по нарушениям | TEXT |

  
Рисунок 1 – Создание базы данных

  
Рисунок 2 – Создание таблиц

Структура таблиц

Таблица 1 Таблица данных об игроке:

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INT | Уникальный идентификатор |
| phone | VARCHAR(15) | Номер телефона |
| firstName | VARCHAR(50) | Имя |
| secondName | VARCHAR(50) | Отчество |
| surname | VARCHAR(50) | Фамилия |
| email | VARCHAR(100) | Электронная почта |
| role | ENUM('User','Police','Moderator','Admin') | Роль пользователя |

Таблица 2 Таблица данных об заявлении:

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INT | Уникальный идентификатор |
| userId | INT | Идентификатор пользователя |
| statusId | INT | Идентификатор статуса |
| description | TEXT | Описание нарушения |
| numberCar | VARCHAR(20) | Номер автомобиля (если есть) |
| address | VARCHAR(255) | Адрес нарушения |

Таблица 3 Таблица данных об файлах заявления:

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| id | INT | Уникальный идентификатор |
| filename | VARCHAR(255) | Имя файла |
| path | VARCHAR(255) | Путь к файлу |
| orderId | INT | Идентификатор сообщения |

Таблица 4 Таблица данных об статусе заявления:

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| id | INT | Уникальный идентификатор |
| title | VARCHAR(50) | Название статуса |

Таблица 5 Таблица данных об данных для авторизации:

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INT | Уникальный идентификатор |
| userId | INT | Идентификатор пользователя |
| login | VARCHAR(50) | Логин |
| password | VARCHAR(255) | Пароль |

Инфологическая модель

На основе вышеуказанных сущностей строится инфологическая модель, показывающая связи между таблицами:

* Каждый пользователь может иметь несколько сообщений о нарушениях (связь один ко многим).
* Каждое сообщение может иметь несколько файлов (связь один ко многим).
* Каждое сообщение имеет один статус (связь многие к одному).

# 2.3 Неформальное описание алгоритма работы ПП

Алгоритм работы системы "Нарушений.нет" включает следующие глобальные шаги:

1. Регистрация и аутентификация пользователя**:**
   * Пользователь регистрируется, вводит свои данные и получает учетную запись.
   * Пользователь аутентифицируется с использованием логина и пароля.
2. Подача сообщения о нарушении:
   * Пользователь заполняет форму сообщения, добавляет описание, фотографии и адрес нарушения.
   * Сообщение сохраняется в базе данных и ему присваивается статус "Новое".
3. Обработка сообщения:
   * Сообщение рассматривается модератором или ответственным лицом.
   * Статус сообщения обновляется в зависимости от стадии обработки (например, "В работе", "Закрыто").
4. Отслеживание статуса сообщения:
   * Пользователь может отслеживать статус своего сообщения через интерфейс.
   * Пользователь получает уведомления о изменениях статуса.
5. Генерация отчетов:
   * Система генерирует отчеты и предоставляет статистику по обработанным сообщениям.

Блок-схема алгоритма

# 2.4 Функции подсистем (модулей)

1. Модуль регистрации и аутентификации:
   * Регистрация новых пользователей.
   * Аутентификация пользователей с использованием логина и пароля.
   * Управление учетными записями и ролями пользователей.
2. Модуль подачи сообщений:
   * Форма для ввода сообщений о нарушениях.
   * Загрузка фотографий и других файлов.
   * Сохранение сообщений в базе данных.
3. Модуль обработки сообщений:
   * Просмотр и сортировка сообщений модераторами и ответственными лицами.
   * Обновление статусов сообщений.
   * Управление данными о нарушениях.
4. Модуль отслеживания и уведомлений:
   * Отслеживание статуса сообщений пользователями.
   * Отправка уведомлений о изменениях статусов.
   * История изменений и комментарии к сообщениям.
5. Модуль отчетов и аналитики:
   * Генерация отчетов по обработанным сообщениям.
   * Статистика по типам нарушений и эффективности обработки.
   * Экспорт отчетов в различные форматы.

# 2.5 Оценка экономической эффективности от внедрения программы

Оценка затрат на разработку программного продукта включает следующие этапы:

* Оценка размера программного изделия:
* Объем кода, количество модулей и функций.

Оценка трудоемкости:

* Количество часов работы разработчиков, тестировщиков и других участников проекта.

Оценка продолжительности:

* Время, необходимое для завершения всех этапов разработки, тестирования и внедрения.

Оценка стоимости:

* Заработная плата сотрудников, расходы на оборудование и программное обеспечение.

Пример расчета себестоимости

|  |  |
| --- | --- |
| Статья расходов | Сумма (руб.) |
| Заработная плата разработчиков | 1,000,000 |
| Заработная плата тестировщиков | 500,000 |
| Оборудование и ПО | 300,000 |
| Прочие расходы | 200,000 |
| Итого | 2,000,000 |

Экономическая эффективность внедрения программы оценивается по следующим параметрам:

Сокращение времени обработки сообщений о нарушениях.

Повышение прозрачности и доверия граждан к государственным органам.

Улучшение качества городской среды и снижение количества нарушений.

# ГЛАВА 3 Эскизный проект

В данном разделе описывается интерфейс сайта "Нарушений.нет". Будет рассмотрено, как выглядит дизайн страниц, какие элементы на них присутствуют и как осуществляется взаимодействие пользователя с этими элементами. Основная цель системы — предоставление пользователям возможности сообщать о нарушениях и отслеживать статус их заявлений.

Интерфейс сайта включает следующие основные страницы (см. ПРИЛОЖЕНИЕ Б):

1. **Главная страница**
2. **Страница авторизации**
3. **Страница регистрации**
4. **Страница профиля**
5. **Страница заявлений**
6. **Страница конкретного заявления**

Каждая страница имеет уникальный набор элементов интерфейса, направленных на выполнение специфических функций.

## 3.1 Описание страниц

### Главная страница

Главная страница является точкой входа на сайт. На ней пользователь видит приветствие и краткое описание функционала сайта. Основные элементы страницы:

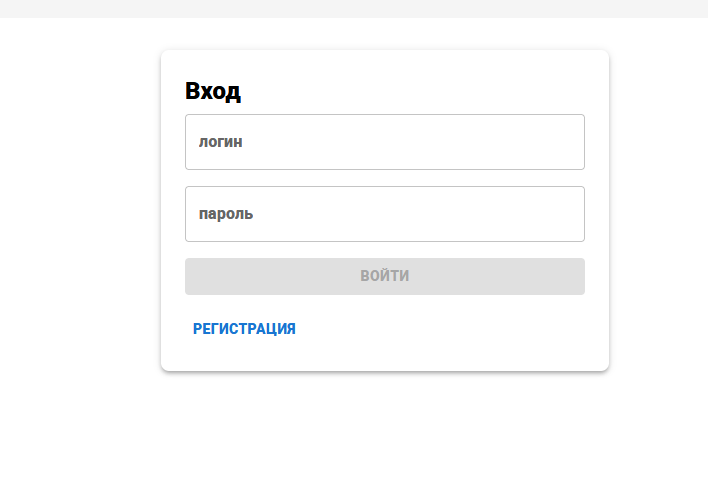
* **Заголовок**: Название сайта.
* **Навигационное меню**: Ссылки на страницы авторизации, регистрации, профиля и заявлений.
* **Информационные блоки**: Краткая информация о сайте и его целях.
* **Кнопки действий**: Кнопки "Войти" и "Зарегистрироваться".

**  
Рисунок 3: Главная страница**

#### Страница авторизации

На странице авторизации пользователи могут войти в систему, используя свои учетные данные (см. ПРИЛОЖЕНИЕ В). Элементы страницы:

* **Форма авторизации**:
  + Поле для ввода логина.
  + Поле для ввода пароля.
  + Кнопка "Войти".
* **Сообщения об ошибках**: Отображаются при неверном вводе данных.
* **Ссылка для восстановления пароля**: В случае, если пользователь забыл пароль.

**  
Изображение 4: Страница авторизации**

#### 3.2.3 Страница регистрации

Страница регистрации позволяет новым пользователям создать учетную запись. Элементы страницы:

**Форма регистрации**:

* + Поля для ввода имени, отчества, фамилии.
  + Поле для ввода email.
  + Поле для ввода телефона.
  + Поле для ввода логина.
  + Поле для ввода пароля.
  + Кнопка "Зарегистрироваться".

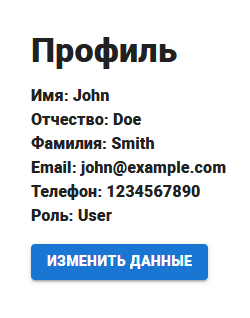
**Сообщения об ошибках:** Отображаются при неверном вводе данных.

**  
Изображение 5: Страница регистрации**

#### 3.2.4 Страница профиля

После успешной авторизации пользователь попадает на страницу профиля. Здесь отображается личная информация пользователя. Элементы страницы:

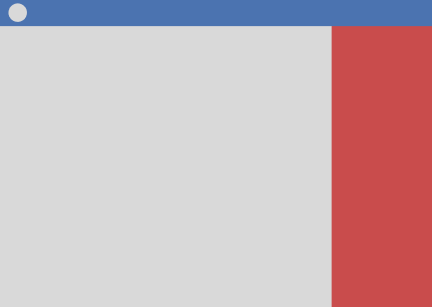
* **Карточка профиля**:
  + Имя.
  + Отчество.
  + Фамилия.
  + Email.
  + Телефон.
  + Роль.
  + Кнопка "Изменить данные" (на текущий момент не активна).

**  
Изображение 6: Страница профиля**

#### 3.2.5 Страница заявлений

На этой странице пользователь может увидеть все свои заявления. Элементы страницы:

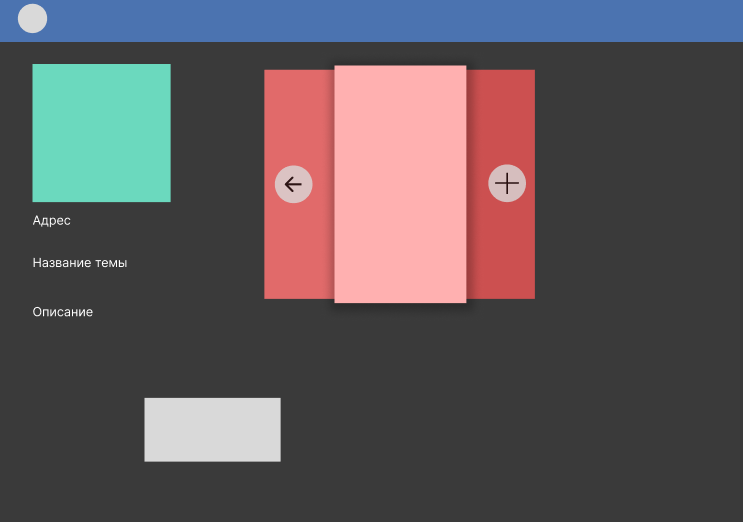
* **Список заявлений**: Каждый элемент списка содержит краткое описание заявления и его статус.
* **Кнопка добавления нового заявления**: Позволяет создать новое заявление.

**  
Изображение 7: Страница заявлений**

#### 3.2.6 Страница конкретного заявления

Эта страница отображает подробную информацию о конкретном заявлении. Элементы страницы:

* **Заголовок заявления**: Описание проблемы.
* **Изображения или видео**: Медиа-файлы, прикрепленные к заявлению.
* **Описание**: Подробное описание нарушения.
* **Номер машины**: Если нарушение связано с транспортным средством.
* **Статус заявления**: Текущий статус обработки заявления.

**  
Изображение 8: Страница конкретного заявления**

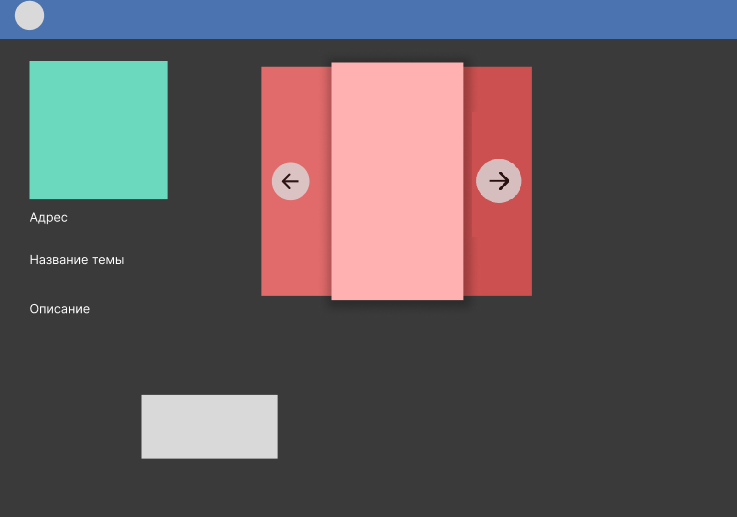
### 3.3 Страница "Заявление"

#### 3.3.1 Описание страницы

Страница "Заявление" представляет собой подробное описание конкретного заявления пользователя о нарушении. Эта страница включает в себя всю информацию, необходимую для рассмотрения и обработки заявления.

Элементы страницы:

* **Заголовок**: Краткое описание нарушения.
* **Адрес**: Место, где произошло нарушение.
* **Изображения или видео**: Медиа-файлы, загруженные пользователем для подтверждения факта нарушения.
* **Описание**: Текстовое описание нарушения, включающее все детали.
* **Номер машины**: Если нарушение связано с автомобилем, указывается его номер.

**  
Изображение 9: Страница "Заявление"**

## 3.4 Заключение

Интерфейс сайта "Нарушений.нет" разработан с учетом удобства пользователя и простоты навигации. Все страницы сайта содержат интуитивно понятные элементы и обеспечивают пользователя необходимой функциональностью для подачи и отслеживания заявлений о нарушениях.

# ГЛАВА 4 Разработка проектных решений

В данном разделе рассмотрим разработку информационной системы для осознанных граждан "Нарушений.нет". Проект реализован с использованием технологий React, NestJS, MySQL, а также инструментов разработки Visual Studio Code, DBeaver и GitHub.

Выбор технологий и инструментов

React — JavaScript-библиотека для создания пользовательских интерфейсов. Её основное преимущество — возможность создавать динамичные и отзывчивые интерфейсы с использованием компонентов. React имеет обширную экосистему, поддерживает высокую производительность и гибкость.

NestJS — прогрессивный фреймворк для создания серверных приложений на Node.js. Он использует TypeScript, что обеспечивает статическую типизацию кода, и имеет модульную архитектуру, упрощая разработку и поддержку крупных приложений. NestJS интегрируется с различными базами данных, включая MySQL.

MySQL — реляционная база данных с открытым исходным кодом, широко используемая для хранения данных в веб-приложениях. MySQL отличается высокой производительностью, надежностью и простотой в использовании.

Visual Studio Code — мощный и гибкий редактор исходного кода, поддерживающий множество расширений и инструментов для разработки. Он обеспечивает удобный интерфейс и широкие возможности настройки.

DBeaver — универсальный инструмент для работы с базами данных, поддерживающий различные СУБД, включая MySQL. DBeaver предоставляет удобный интерфейс для управления базами данных и выполнения запросов.

GitHub — платформа для хостинга и совместной разработки проектов. Она обеспечивает контроль версий, позволяет отслеживать изменения в коде и организовывать командную работу.

Пример кода и его описание

Приведем пример кода на NestJS, который реализует аутентификацию пользователей с использованием JWT (JSON Web Token).

@Injectable()

export class JwtStrategy extends PassportStrategy(Strategy, 'jwt') {

constructor() {

super({

jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

ignoreExpiration: false,

secretOrKey: jwtConstants.secret,

});

}

/\*\*

\* Эта функция срабатывает после дешифровки JWT токена и встраивает в запрос возвращаемые данные.

\* @param user - данные из токена

\* @returns встраиваемые данные

\*/

async validate(user: dataOnToken) {

console.log('validate', user);

return { ...user };

}

}

Этот фрагмент кода определяет стратегию аутентификации с использованием JWT. Класс JwtStrategy наследуется от PassportStrategy и использует настройки для извлечения и проверки JWT токенов. Метод validate вызывается после дешифровки токена и возвращает данные пользователя, которые будут доступны в запросах.

@Injectable()

export class LocalStrategy extends PassportStrategy(Strategy) {

constructor(private authService: AuthService) {

super({ usernameField: 'login' });

}

async validate(username: string, password: string): Promise<UserModel> {

const user: UserModel = await this.authService.validateUser(

username,

password,

);

if (!user) {

throw new UnauthorizedException();

}

return user;

}

}

Этот фрагмент кода определяет стратегию локальной аутентификации. Класс LocalStrategy использует метод validate для проверки учетных данных пользователя (логина и пароля). Если пользователь найден и его данные корректны, он возвращается, в противном случае выбрасывается исключение UnauthorizedException.

@Controller('users')

@ApiTags('users')

export class UsersController {

constructor(private readonly userService: UsersService) {}

@Get()

getUsers(): Promise<UserModel[]> {

return this.userService.findAllData();

}

@Post('create')

@HttpCode(201)

async createUser(@Body() body: CreateUserDto): Promise<UserModel> {

//TODO: проверки перед созданием

const user = await this.userService.create(body);

return user;

}

}

Контроллер UsersController управляет операциями с пользователями. Метод getUsers возвращает всех пользователей, а метод createUser создает нового пользователя на основе переданных данных.

@Injectable()

export class UsersService {

constructor(

@InjectModel(UserModel)

private userModel: typeof UserModel,

@InjectModel(UserAuthModel)

private userAuthModel: typeof UserAuthModel,

) {}

async findAll(): Promise<UserModel[]> {

return this.userModel.findAll();

}

async findOneByLogin(login: string): Promise<UserAuthModel> {

var userData = await this.userAuthModel.findOne({

where: { login: login },

include: [UserModel],

});

return userData;

}

async findAllData(): Promise<any> {

var user = await this.userModel.findAll({ include: [UserAuthModel] });

var userAuthData = await this.userAuthModel.findAll({

include: [UserModel],

});

return userAuthData;

}

async findOne(id: number): Promise<UserModel> {

return this.userModel.findOne({

where: {

id,

},

});

}

async findOneByConfig(config: FindOptions): Promise<UserModel> {

return this.userModel.findOne(config);

}

async remove(id: number): Promise<void> {

const user = await this.findOne(id);

await user.destroy();

}

async create({

user: userData,

authData,

}: CreateUserDto): Promise<UserModel> {

const userAuthData = await this.userAuthModel.create<UserAuthModel>({

login: authData.login,

password: authData.password,

});

const user = await this.userModel.create<UserModel>({

firstName: userData.firstName,

secondName: userData.secondName,

surname: userData.surname,

email: userData.email,

phone: userData.phone,

userAuthData: userAuthData,

role: UserRole.User,

});

userAuthData.userId = user.id;

await user.save();

await userAuthData.save();

return user;

}

}

Сервис UsersService управляет данными пользователей. Методы включают поиск всех пользователей, поиск по логину, создание новых пользователей и удаление существующих.

Эти примеры кода показывают, как различные модули системы взаимодействуют для реализации функционала аутентификации и управления пользователями в информационной системе "Нарушений.нет".

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы по теме "Разработка информационной системы для осознанных граждан 'Нарушений.нет'" была создана информационная система, предназначенная для регистрации и мониторинга гражданских нарушений. Проект выполнен с использованием технологий React, Nest, Visual Studio Code, MySQL, DBeaver и GitHub, что обеспечило высокое качество разработки и стабильность конечного продукта.

Результаты разработки информационной системы "Нарушений.нет" свидетельствуют о ее успешной реализации и соответствии требованиям, изложенным в техническом задании. Продукт позволяет пользователям:

* Регистрацию новых случаев гражданских нарушений.
* Мониторинг зарегистрированных нарушений в реальном времени.
* Доступ к аналитическим данным и отчетам по зарегистрированным инцидентам.
* Удобное и интуитивно понятное взаимодействие с интерфейсом системы.

Система обеспечивает высокую производительность и надежность, что подтверждено тестированием и положительными отзывами от пользователей тестовой группы.

Разработанный программный продукт соответствует всем заявленным требованиям технического задания. В процессе работы были решены задачи по созданию удобного пользовательского интерфейса, обеспечению безопасности данных и интеграции с базой данных MySQL. Использование современных технологий, таких как React и Nest, позволило достичь высокой производительности и масштабируемости системы.

Программный продукт прошел этапы модульного и интеграционного тестирования, что позволило выявить и устранить ошибки до выпуска системы в эксплуатацию. Таким образом, конечный продукт отвечает высоким стандартам качества и готов к использованию в реальных условиях.

**Практическая эффективность**

Внедрение информационной системы "Нарушений.нет" уже показало свою практическую эффективность:

* Увеличилась оперативность регистрации и обработки гражданских нарушений.
* Снизилось количество неучтенных инцидентов благодаря удобному и доступному интерфейсу для пользователей.
* Повысилась прозрачность и подотчетность процессов регистрации нарушений.
* Улучшилось взаимодействие между гражданами и соответствующими органами благодаря наличию централизованной базы данных и инструментов для анализа.

Эти результаты свидетельствуют о значительном положительном влиянии системы на общественные процессы и уровень гражданской ответственности.

**Перспективы дальнейшей разработки**

Разработка информационной системы "Нарушений.нет" имеет широкий потенциал для дальнейшего развития:

1. Расширение функциональности:

* Внедрение новых модулей для автоматического анализа данных с использованием методов машинного обучения.
* Разработка мобильного приложения для более широкого охвата пользователей.

2. Интеграция с внешними системами:

* Возможность интеграции с системами государственных органов для автоматического обмена данными.
* Создание API для взаимодействия с другими гражданскими инициативами и проектами.

3. Повышение безопасности:

* Внедрение дополнительных мер безопасности, включая двухфакторную аутентификацию и шифрование данных.
* Проведение регулярных аудитов безопасности для выявления и устранения уязвимостей.

4. Международное расширение:

* Адаптация системы для использования в других странах с учетом местных особенностей законодательства и языковых предпочтений.
* Участие в международных инициативах по обмену опытом и данными.

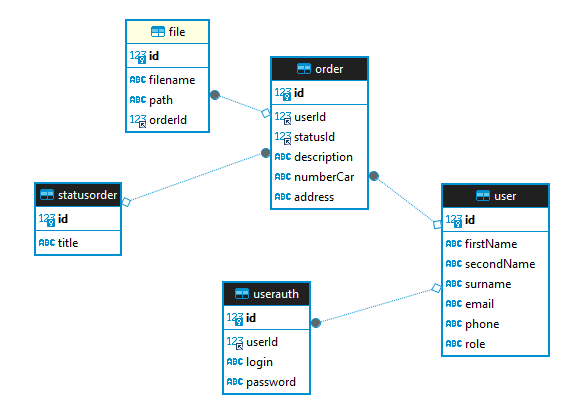
Создание информационной системы "Нарушений.нет" является значимым вкладом в развитие гражданских инициатив и повышение осознанности граждан в вопросах соблюдения законодательства. Система не только отвечает текущим требованиям, но и обладает значительным потенциалом для дальнейшего развития и масштабирования. Внедрение данного проекта позволит повысить прозрачность и эффективность процессов регистрации гражданских нарушений, что в свою очередь способствует улучшению социальной и правовой среды.

Таким образом, разработка информационной системы "Нарушений.нет" представляет собой успешный пример применения современных технологий для решения актуальных общественных задач, и открывает новые возможности для дальнейшего развития и внедрения инновационных решений в данной сфере.

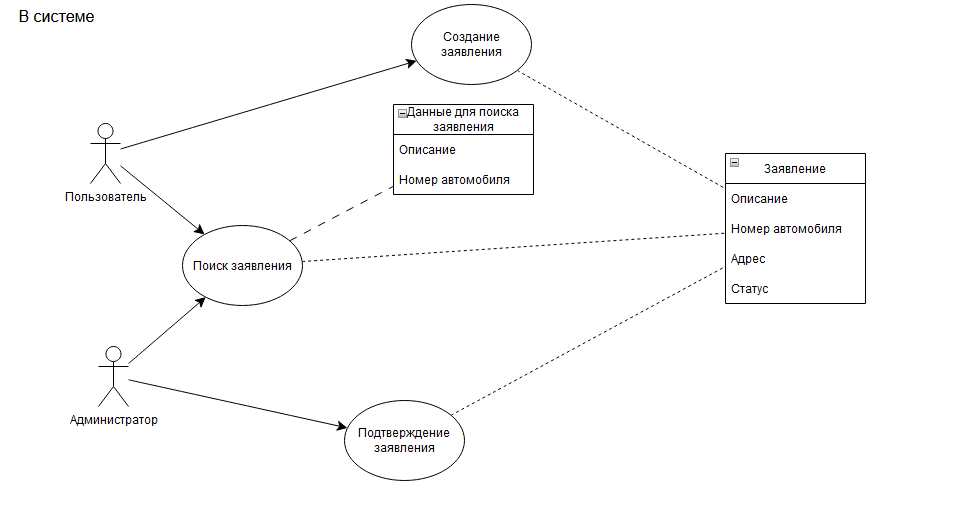
# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. React – A JavaScript library for building user interfaces // React URL: <https://reactjs.org/> (дата обращения: 31.05.2024).
2. NestJS – A progressive Node.js framework for building efficient, reliable, and scalable server-side applications // NestJS URL: <https://nestjs.com/> (дата обращения: 31.05.2024).
3. Visual Studio Code – Code editing. Redefined. // Visual Studio Code URL: <https://code.visualstudio.com/> (дата обращения: 31.05.2024).
4. MySQL – The world's most popular open source database // MySQL URL: <https://www.mysql.com/> (дата обращения: 31.05.2024).
5. DBeaver – Free Universal Database Tool // DBeaver URL: <https://dbeaver.io/> (дата обращения: 31.05.2024).
6. GitHub – Where the world builds software // GitHub URL: <https://github.com/> (дата обращения: 31.05.2024).
7. Оказание первой помощи в походе // турклуб ПИК URL: <https://turclub-pik.ru/blog/okazanie-pervoj-pomoshhi-v-pokhode/> (дата обращения: 11.02.2023).
8. K. Cunningham. Learning React: Modern Patterns for Developing React Apps. – O'Reilly Media, 2022.
9. D. Koppenhaver. Fullstack React with TypeScript. – Newline, 2022.
10. A. Holmes. NestJS: A Progressive Node.js Framework. – Packt Publishing, 2023.
11. B. Randell. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers. – Apress, 2023.
12. P. DuBois. MySQL Cookbook: Solutions for Database Developers and Administrators. – O'Reilly Media, 2022.
13. J. D. Shaw. MySQL 8 Administrator’s Guide. – Apress, 2023.
14. DBeaver User Guide // DBeaver URL: https://dbeaver.io/documentation/ (дата обращения: 31.05.2024).
15. Pro Git (2nd Edition) // GitHub URL: <https://git-scm.com/book/en/v2> (дата обращения: 31.05.2024).
16. M. Lodder. Pro Git: Everything You Need to Know About Git. – Apress, 2022.
17. Современные системы управления базами данных // Habr URL: https://habr.com/ru/company/otus/blog/546418/ (дата обращения: 31.05.2024).
18. Frontend Development Trends 2023 // Smashing Magazine URL: https://www.smashingmagazine.com/2023/01/frontend-development-trends-2023/ (дата обращения: 31.05.2024).
19. Web Development Trends in 2024 // FreeCodeCamp URL: https://www.freecodecamp.org/news/web-development-trends-in-2024/ (дата обращения: 31.05.2024).
20. C. Wenz. Advanced GraphQL with Node.js and React. – Packt Publishing, 2023.

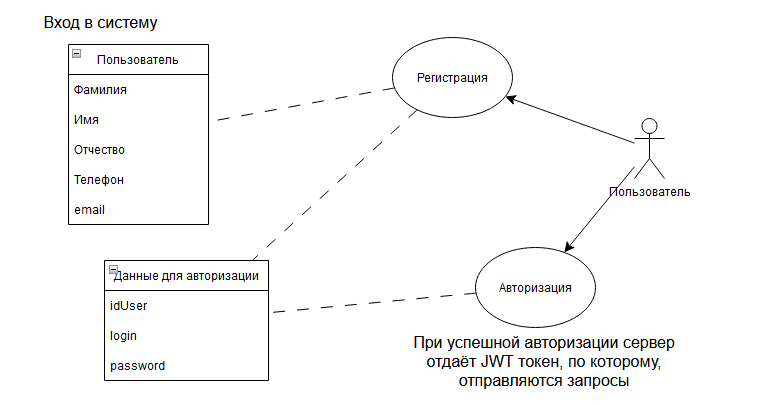
# ПРИЛОЖЕНИЕ A

  
Рисунок - Логическая модель данных проекта в виде ER диаграммы

# ПРИЛОЖЕНИЕ Б

  
Рисунок - UseCase диаграмма Приложения

# ПРИЛОЖЕНИЕ В

  
Рисунок - UseCase диаграмма Входа

ПРИЛОЖЕНИЕ Г

Листинг 1. user.service

import { Injectable } from '@nestjs/common';

import { InjectModel } from '@nestjs/sequelize';

import { UserModel } from './models/user.entity';

import { FindOptions, Model, where } from 'sequelize';

import { UserAuthModel } from './models/authData.entity';

import { CreateUserDto } from 'src/users/dto/CreateUser.dto';

import { UserRole } from 'src/types/types';

@Injectable()

export class UsersService {

constructor(

@InjectModel(UserModel)

private userModel: typeof UserModel,

@InjectModel(UserAuthModel)

private userAuthModel: typeof UserAuthModel,

) {}

async findAll(): Promise<UserModel[]> {

return this.userModel.findAll();

}

async findOneByLogin(login: string): Promise<UserAuthModel> {

var userData = await this.userAuthModel.findOne({

where: { login: login },

include: [UserModel],

});

return userData;

}

async findAllData(): Promise<any> {

var user = await this.userModel.findAll({ include: [UserAuthModel] });

var userAuthData = await this.userAuthModel.findAll({

include: [UserModel],

});

return userAuthData;

}

async findOne(id: number): Promise<UserModel> {

return this.userModel.findOne({

where: {

id,

},

});

}

async findOneByConfig(config: FindOptions): Promise<UserModel> {

return this.userModel.findOne(config);

}

async remove(id: number): Promise<void> {

const user = await this.findOne(id);

await user.destroy();

}

async create({

user: userData,

authData,

}: CreateUserDto): Promise<UserModel> {

const userAuthData = await this.userAuthModel.create<UserAuthModel>({

login: authData.login,

password: authData.password,

});

const user = await this.userModel.create<UserModel>({

firstName: userData.firstName,

secondName: userData.secondName,

surname: userData.surname,

email: userData.email,

phone: userData.phone,

userAuthData: userAuthData,

role: UserRole.User,

});

userAuthData.userId = user.id;

await user.save();

await userAuthData.save();

return user;

}

}

Листинг 2. user.controller

import {

Body,

Controller,

Get,

HttpCode,

Param,

Post,

Query,

Redirect,

} from '@nestjs/common';

import { UsersService } from './user.service';

import { UserModel } from './models/user.entity';

import { CreateUserDto } from 'src/users/dto/CreateUser.dto';

import { ApiTags } from '@nestjs/swagger';

@Controller('users')

@ApiTags('users')

export class UsersController {

constructor(private readonly userService: UsersService) {}

@Get()

getUsers(): Promise<UserModel[]> {

return this.userService.findAllData();

}

@Post('create')

@HttpCode(201)

async createUser(@Body() body: CreateUserDto): Promise<UserModel> {

//TODO: проверки на перед созданием

const user = await this.userService.create(body);

return user;

}

}

Листинг 3. Авторизация

import { Injectable } from '@nestjs/common';

import { UsersService as UserService } from 'src/users/user.service';

import { JwtService } from '@nestjs/jwt';

import responseDataAuth from './dto/responseDataAuth.dto';

import { UserModel } from 'src/users/models/user.entity';

import responseRefreshedAccessToken from './dto/responseRefreshedAccessToken.dto';

import { dataOnToken } from 'src/types/types';

@Injectable()

export class AuthService {

constructor(

private readonly userService: UserService,

private readonly jwtService: JwtService,

) {}

async validateUser(login: string, password: string): Promise<UserModel> {

const userAuthData = await this.userService.findOneByLogin(login);

if (userAuthData && userAuthData.password === password) {

const result: UserModel = userAuthData.user;

return userAuthData.user;

}

return null;

}

async login(user: UserModel): Promise<responseDataAuth> {

const { id } = user;

return {

user: user,

role: user.role,

accessToken: this.jwtService.sign({ id }),

refreshToken: this.jwtService.sign({ id }, { expiresIn: '7d' }),

};

}

async refreshToken(user: UserModel): Promise<responseRefreshedAccessToken> {

const payload: dataOnToken = {

id: user.id,

role: user.role,

};

return {

accessToken: this.jwtService.sign(payload),

};

}

}

Листинг 4. Заявления

import {

Body,

Controller,

Get,

Param,

Post,

Res,

UploadedFile,

UploadedFiles,

UseGuards,

UseInterceptors,

Request,

} from '@nestjs/common';

import { OrdersService } from './orders.service';

import { FileInterceptor, FilesInterceptor } from '@nestjs/platform-express';

import { ApiBody, ApiTags } from '@nestjs/swagger';

import createOrderDTO from './dto/createOrder.dto';

import { extname } from 'path';

import { diskStorage } from 'multer';

import { Response } from 'express';

import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard';

import { authDataJWT } from 'src/auth/dto/authDataJWT.dto';

import { UploadFileDTO } from './dto/uploadFile.dto';

import { userDataFromToken } from 'src/types/types';

import { OrderModel } from './models/order.entity';

@Controller('orders')

@ApiTags('orders')

@UseGuards(JwtAuthGuard)

export class OrdersController {

constructor(private orderService: OrdersService) {}

@Get()

@ApiBody({ type: authDataJWT })

async getAllOrdersByUserId(

@Request() req: userDataFromToken,

): Promise<{ orders: OrderModel[] }> {

const { user } = req;

const orders = await this.orderService.findAllOrdersByUserId(user.id);

return { orders: orders };

}

@Post('create')

@ApiBody({ type: authDataJWT })

async createOrder(

@Body() body: createOrderDTO,

@Request() req: userDataFromToken,

) {

const { user } = req;

return this.orderService.create(body, { id: user.id });

}

@Post('uploadFile')

@UseInterceptors(

FileInterceptor('file', {

storage: diskStorage({

destination: './uploads',

filename: (req, file, cb) => {

const randomName = Array(32)

.fill(null)

.map(() => Math.round(Math.random() \* 16).toString(16))

.join('');

cb(null, `${randomName}${extname(file.originalname)}`);

},

}),

limits: { fileSize: 50 \* 1024 \* 1024 }, //лимит размера файла до 50MB

}),

)

@ApiBody({ type: UploadFileDTO })

async uploadFile(

@Body() body,

@UploadedFile() file,

@Request() req: UploadFileDTO,

) {

console.log(file);

// const data: UploadFileDTO = {

// file: file,

// orderId: 7,

// user: req.user,

// };

// console.log(data);

// const image = await this.orderService.uploadFile(data);

// console.log(image);

return 'image';

}

@Get('file/:id')

async getFile(@Param('id') id: string, @Res() res: Response) {

try {

const image = await this.orderService.getFileById(+id);

if (!image) {

return res.status(404).send('File not found');

}

res.sendFile(image.path, { root: './' });

} catch (error) {

return res.status(500).send('Internal Server Error');

}

}

@Get(':id/filesData')

async getOrderFilesData(@Param('id') id: number): Promise<any> {

return this.orderService.getOrderFiles(id);

}

@Get(':id')

async getOrderByID(@Param('id') id: number): Promise<any> {

return this.orderService.getOrdersByID(id);

}

// @Get(':id/files')

// async getOrderFiles(@Param('id') id: string, @Res() res: Response) {

// const files = await this.orderService.getOrderFiles(+id);

// if (files.length === 0) {

// res.status(404).send('No files found for this order');

// return;

// }

// const archive = archiver('zip', {

// zlib: { level: 9 } // Sets the compression level.

// });

// res.attachment(`order-${id}-files.zip`);

// archive.pipe(res);

// files.forEach(file => {

// const filePath = file.path;

// if (fs.existsSync(filePath)) {

// archive.file(filePath, { name: file.filename });

// }

// });

// await archive.finalize();

// }

}

import { Injectable } from '@nestjs/common';

import { InjectModel } from '@nestjs/sequelize';

import { OrderModel } from './models/order.entity';

import { StatusOrderModel } from './models/status-order.entity';

import createOrderDTO from './dto/createOrder.dto';

import { FileModel } from './models/file.entity';

import { UserModel } from 'src/users/models/user.entity';

import { UploadFileDTO } from './dto/uploadFile.dto';

@Injectable()

export class OrdersService {

constructor(

@InjectModel(OrderModel)

private orderModel: typeof OrderModel,

@InjectModel(StatusOrderModel)

private statusOrderModel: typeof StatusOrderModel,

@InjectModel(FileModel)

private readonly imageModel: typeof FileModel,

) {}

async create(data: createOrderDTO, user: { id }): Promise<OrderModel> {

//TODO: Загрузка Изображений

const order = await this.orderModel.create<OrderModel>({

description: data.description,

userId: user.id,

numberCar: data.numberCar,

address: data.address,

statusId: 1,

});

await order.save();

return order;

}

async findAllOrdersByUserId(userId: number): Promise<OrderModel[]> {

//{ data: any, userId: number }: any

const order = await this.orderModel.findAll<OrderModel>({

include: [StatusOrderModel],

where: [

{

userId: userId,

},

],

});

if (!order) {

throw new Error('Order not found');

}

return order;

}

async getOrdersByID(id: number): Promise<OrderModel> {

//{ data: any, userId: number }: any

const order = await this.orderModel.findByPk<OrderModel>(id);

if (!order) {

throw new Error('Order not found');

}

return order;

}

//------IMAGE-------//

async uploadFile(data: UploadFileDTO): Promise<FileModel> {

const { file, orderId, user } = data;

//TODO: Можно удалить

const order = await this.getOrdersByID(orderId);

//FIXME Удалить сохранённый файл, в случае ошибки

//FIXME

if (order.userId != user.id) {

throw new Error('Access denied');

}

const createdFile = await this.imageModel.create({

filename: file.originalname,

path: file.path,

orderId: orderId,

});

return createdFile;

}

async getAllFiles(): Promise<FileModel[]> {

return this.imageModel.findAll();

}

async getFileById(id: number): Promise<FileModel> {

return this.imageModel.findOne({ where: { id } });

}

async getOrderFiles(orderId: number): Promise<FileModel[]> {

const order = await this.orderModel.findByPk(orderId, {

include: [FileModel],

});

if (!order) {

throw new Error('Order not found');

}

return order.files;

}

}