ОБЛАСТНОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«КУРСКИЙ ТЕХНИКУМ СВЯЗИ»

|  |  |
| --- | --- |
| Рецензент  *Аббакумова Екатерина*  *Виктороевна-ведущий инженер -программист ОАО*  «Курскрезинотехника» /  « » 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

[ВВЕДЕНИЕ 3](#_Toc169009971)

[ГЛАВА 1 ПРЕДПРОЕКТНОЕ ОБСЛЕДОВАНИЕ 6](#_Toc169009972)

[1.1 Исследование предметной области 6](#_Toc169009973)

[1.2 Формирование требований к программному изделию 7](#_Toc169009974)

[ГЛАВА 2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 10](#_Toc169009975)

[2.1 Постановка задачи 10](#_Toc169009976)

[2.2 Концепция информационной базы и её структура 11](#_Toc169009977)

[2.2.1 Структура информационной базы 11](#_Toc169009978)

[2.2.2 Разработка базы данных 14](#_Toc169009979)

[2.2.3 Модели и связи 14](#_Toc169009980)

[2.3 Неформальное описание алгоритма работы ПП 18](#_Toc169009981)

[2.4 Функции подсистем (модулей) 20](#_Toc169009982)

[2.5 Оценка экономической эффективности от внедрения программы 21](#_Toc169009983)

[ГЛАВА 3 ЭСКИЗНЫЙ ПРОЕКТ 26](#_Toc169009984)

[3.1 Верстка фронтенда 26](#_Toc169009985)

[ГЛАВА 4 РАЗРАБОТКА ПРОЕКТНЫХ РЕШЕНИЙ 31](#_Toc169009986)

[4.1 Разработка серверных решений 31](#_Toc169009987)

[ЗАКЛЮЧЕНИЕ 36](#_Toc169009988)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_Toc169009989)

[ПРИЛОЖЕНИЕ A 40](#_Toc169009990)

[ПРИЛОЖЕНИЕ Б 41](#_Toc169009991)

[ПРИЛОЖЕНИЕ В 43](#_Toc169009992)

# ВВЕДЕНИЕ

Разработка информационной системы для осознанных граждан "Нарушений.нет" представляет собой важное и актуальное направление в современном обществе, где проблема нарушений правил дорожного движения остается острой. Данный проект нацелен на создание удобного и эффективного инструмента, который позволит гражданам активно участвовать в фиксации и предотвращении правонарушений на дорогах, тем самым способствуя повышению безопасности и порядка.

Актуальность выбранной темы обусловлена растущей необходимостью вовлечения граждан в процесс обеспечения правопорядка. Современные технологии предоставляют уникальные возможности для создания платформ, которые позволяют оперативно сообщать о нарушениях и предоставлять достоверную информацию правоохранительным органам. Система "Нарушений.нет" призвана стать таким инструментом, облегчающим взаимодействие граждан с полицией и улучшая эффективность обработки заявлений о нарушениях.

Целью проекта является разработка информационной системы, которая обеспечит возможность гражданам регистрировать и отправлять заявления о нарушениях правил дорожного движения, а также позволит администраторам эффективно обрабатывать и подтверждать эти заявления. Для достижения поставленной цели решаются следующие задачи:

1. Создание удобного интерфейса для регистрации и входа пользователей.
2. Разработка функционала для формирования и отправки заявлений о нарушениях.
3. Обеспечение безопасного хранения и обработки данных пользователей.
4. Внедрение системы администрирования для подтверждения или отклонения заявлений.

Практическая значимость проекта заключается в создании платформы, которая облегчит процесс фиксации нарушений и повысит прозрачность взаимодействия между гражданами и полицией. Ожидается, что внедрение такой системы приведет к снижению количества нарушений на дорогах и улучшению общей ситуации с безопасностью дорожного движения.

Основные отличия "Нарушений.нет" от аналогичных программ заключаются в простоте использования, высокой степени защиты данных пользователей, а также в возможности оперативного администрирования заявлений. Система разработана с использованием современных технологий, таких как React для создания пользовательского интерфейса, Nest для серверной части, MySQL для управления базой данных, и DBeaver для ее администрирования. Среда разработки Visual Studio Code и система контроля версий GitHub обеспечивают удобство и эффективность процесса разработки.

Предметная область проекта охватывает сферу фиксации нарушений правил дорожного движения. Гражданин, желающий воспользоваться услугами портала, должен пройти процедуру регистрации, в ходе которой указывает свои персональные данные, логин и пароль. Войдя в систему, пользователь получает возможность формировать заявления, указывая номер автомобиля нарушителя и описание нарушения. Все заявления хранятся в системе и проходят процесс администрирования, где каждому заявлению присваивается статус: новое, подтверждено или отклонено.

В качестве информационной базы проекта используется MySQL, которая обеспечивает надежное хранение и быстрый доступ к данным. Средства разработки включают React и Nest, что позволяет создать гибкую и масштабируемую систему, а использование Visual Studio Code и DBeaver обеспечивает удобство разработки и администрирования базы данных. Система функционирует в условиях, требующих высокой надежности и безопасности, что достигается благодаря использованию современных технологий и методик разработки программного обеспечения.

Таким образом, разработанная информационная система "Нарушений.нет" представляет собой важный шаг к повышению безопасности дорожного движения и вовлечению граждан в процесс обеспечения правопорядка.

# 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.

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 «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 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.

## 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.

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.

Страница регистрации. На данной странице необходимо предусмотреть добавление пользователя в систему. Пользователю необходимо предоставить возможность ввести уникальный логин, пароль (минимум 6 символов), ФИО (символы кириллицы и пробелы), телефон (в формате +7(XXX)-XXX-XX-XX) и адрес электронной почты (формат электронной почты). Все поля обязательны для заполнения. Ошибки валидации должны отображаться на форме. По кнопке «Зарегистрироваться» пользователь должен заноситься в базу если поля прошли валидацию.

Страница авторизации. На данной странице необходимо предусмотреть возможность ввода логина и пароля для зарегистрированных пользователей. Попытки некорректного ввода логина и пароля должны сопровождаться сообщениями.

Страница заявлений. На данной странице авторизованный пользователь имеет возможность просмотреть свои заявления со статусами, а также оставить новое заявление

Страница формирования заявления. Гражданин указывает: государственный регистрационный номер автомобиля и описание нарушения. Все поля обязательны.

Панель администратора. Доступ в панель администратора осуществляется по логину copp и паролю password. В панели администратора видны все заявления (ФИО подавшего, описание нарушения, номер автомобиля и статус заявления). Администратор может сменить статус на подтверждено или отклонено (только для заявлений со статусом новое).

## 2.2 Концепция информационной базы и её структура

Информационная система «Нарушениям.Нет» предназначена для помощи полиции в фиксации нарушений правил дорожного движения. Эффективная организация информационной базы этой системы играет ключевую роль в её функционировании. Данная глава описывает концепцию информационной базы, её структуру и технологии, используемые для реализации базы данных.

### 2.2.1 Структура информационной базы

Информационная база системы включает следующие основные сущности: пользователи, заявления, статусы заявлений и файлы. Каждая из этих сущностей имеет свои поля и взаимосвязи, что позволяет организовать данные и обеспечить их целостность.

Пользователь (UserModel):

* **Идентификатор (id)**: Уникальный идентификатор пользователя.
* **Имя (firstName)**: Имя пользователя.
* **Фамилия (secondName)**: Фамилия пользователя.
* **Отчество (surname)**: Отчество пользователя.
* **Email (email)**: Адрес электронной почты.
* **Телефон (phone)**: Номер телефона.
* **Роль (role)**: Роль пользователя (User, Police, Moderator, Admin).

**Таблица 1** - Структура таблицы пользователей.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INTEGER | Уникальный идентификатор |
| firstName | STRING | Имя пользователя |
| secondName | STRING | Фамилия пользователя |
| surname | STRING | Отчество пользователя |
| email | STRING | Адрес электронной почты |
| phone | STRING | Номер телефона |
| role | ENUM | Роль пользователя в системе |

Аутентификационные данные пользователя (UserAuthModel):

* Идентификатор (id): Уникальный идентификатор записи.
* Логин (login): Логин пользователя.
* Пароль (password): Хэшированный пароль пользователя.
* Идентификатор пользователя (userId): Ссылка на соответствующего пользователя.

Таблица 2 - Структура таблицы аутентификационных данных пользователей.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INTEGER | Уникальный идентификатор |
| login | STRING | Логин пользователя |
| password | STRING | Хэшированный пароль пользователя |
| userId | INTEGER | Ссылка на идентификатор пользователя |

Заявление (OrderModel):

* Идентификатор (id): Уникальный идентификатор заявления.
* Идентификатор пользователя (userId): Ссылка на пользователя, подавшего заявление.
* Статус заявления (statusId): Ссылка на статус заявления.
* Описание (description): Описание нарушения.
* Номер автомобиля (numberCar): Номер автомобиля, указанный в заявление.
* Адрес (address): Адрес нарушения.
* Файлы (files): Связанные с заявлением файлы.

Таблица 3 - Структура таблицы заявлений.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INTEGER | Уникальный идентификатор |
| userId | INTEGER | Ссылка на пользователя |
| statusId | INTEGER | Ссылка на статус заявления |
| description | STRING | Описание нарушения |
| numberCar | STRING | Номер автомобиля |
| address | STRING | Адрес нарушения |

Статус заявления (StatusOrderModel):

* Идентификатор (id): Уникальный идентификатор статуса.
* Название (title): Название статуса (новое, подтверждено, отклонено).

Таблица 4 - Структура таблицы статусов заявлений.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INTEGER | Уникальный идентификатор |
| title | STRING | Название статуса |

Файл (FileModel):

* Идентификатор (id): Уникальный идентификатор файла.
* Имя файла (filename): Имя файла.
* Путь (path): Путь к файлу.
* Идентификатор заявления (orderId): Ссылка на заявление, к которой относится файл.

**Таблица 5 -** Структура таблицы файлов.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | INTEGER | Уникальный идентификатор |
| filename | STRING | Имя файла |
| path | STRING | Путь к файлу |
| orderId | INTEGER | Ссылка на идентификатор заявления |

### 2.2.2 Разработка базы данных

Разработка базы данных является важной частью создания информационной системы. Для реализации базы данных системы «Нарушениям.Нет» были выбраны следующие технологии и подходы:

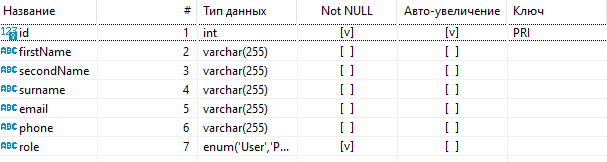
* MySQL: Реляционная база данных, обеспечивающая надежное хранение данных и поддерживающая сложные запросы и транзакции.
* Sequelize ORM: ORM (Object-Relational Mapping) для работы с базой данных, что упрощает процесс управления схемами базы данных и выполнением запросов.
* NestJS: Фреймворк для разработки серверной части приложения, обеспечивающий модульность, масштабируемость и удобную интеграцию с другими инструментами.

Использование Sequelize ORM позволяет определить модели, описывающие структуру и связи между таблицами в базе данных. Эти модели включают все необходимые поля и связи для реализации функциональности системы.

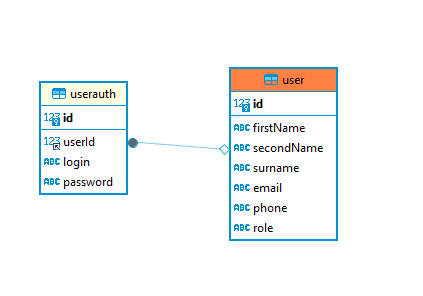
### 2.2.3 Модели и связи

Основные модели базы данных, используемые в системе «Нарушениям.Нет», включают модели пользователей, заявлений, статуса заявлений и файлов. Каждая модель определяет структуру таблицы и взаимосвязи между сущностями.

Модель пользователя (UserModel):

  
Рисунок 1 – таблица пользователь

Модель пользователя включает в себя следующие поля: идентификатор, имя, фамилия, отчество, email, телефон и роль. Эта модель также связана с моделью аутентификационных данных

  
Рисунок 2 – Связь таблиц

@Table({ tableName: 'user' })

export class UserModel extends Model<UserModel> {

@Column({ autoIncrement: true, primaryKey: true })

id: number;

@Column firstName: string;

@Column secondName: string;

@Column surname: string;

@Column @IsEmail() email: string;

@Column @IsPhoneNumber() phone: string;

@Column({ type: DataType.ENUM, values: Object.keys(UserRole), allowNull: false })

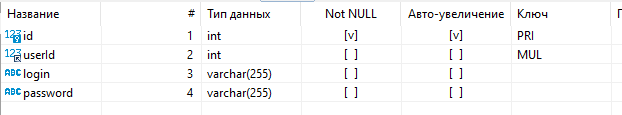
role: UserRole;

@HasOne(() => UserAuthModel) userAuthData: UserAuthModel;

}

Модель аутентификационных данных пользователя (UserAuthModel):

Модель аутентификационных данных пользователя содержит идентификатор, логин, пароль и идентификатор пользователя, к которому эти данные относятся. Эта модель используется для обеспечения безопасности и управления доступом к системе.

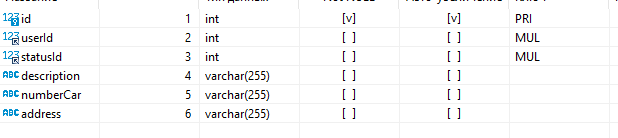
  
Рисунок 3 – таблица аутентификационных данных

Модель заявления (OrderModel):

Модель заявления включает идентификатор, идентификатор пользователя, статус заявления, описание нарушения, номер автомобиля, адрес и файлы, связанные с заявлением. Эта модель обеспечивает управление и отслеживание заявлений пользователей.

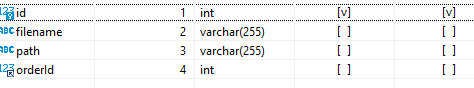
Модель статуса заявления (StatusOrderModel):

Модель статуса заявления содержит идентификатор и название статуса. Эта модель используется для отслеживания текущего состояния заявлений и управления их жизненным циклом

  
Рисунок 4 – таблица заявлений

Модель файла (FileModel):

Модель файла включает идентификатор, имя файла, путь к файлу и идентификатор заявления, к которой относится файл. Эта модель позволяет сохранять и управлять файлами, связанными с заявлениями пользователей.

  
Рисунок 5 – таблица файлов заявления

2.2.4 Описание моделей и их взаимодействий

Информационная база системы «Нарушениям.Нет» построена таким образом, чтобы обеспечить эффективное управление данными и поддерживать все необходимые операции. Модели базы данных определяют структуру и взаимосвязи между сущностями, что позволяет организовать данные и обеспечить их целостность.

Каждая модель базы данных включает в себя определённые поля и связи с другими моделями. Например, модель пользователя связана с моделью аутентификационных данных, что позволяет хранить и управлять информацией о пользователях и их данных для входа в систему. Модель заявления включает ссылки на пользователя, который подал заявление, и статус заявления, что позволяет отслеживать текущий статус и историю заявлений.

Модель статуса заявления определяет различные статусы, которые может иметь заявление, такие как "новое", "подтверждено" или "отклонено". Это позволяет легко управлять жизненным циклом заявлений и обеспечивать их актуальность.

Модель файла позволяет сохранять и управлять файлами, связанными с заявлениями. Это может включать фотографии, видеозаписи или другие документы, которые пользователи прикрепляют к своим заявлениям.

Таким образом, концепция информационной базы и её структура обеспечивают эффективное управление данными и поддерживают все необходимые операции для функционирования системы «Нарушениям.Нет». Использование современных технологий и подходов, таких как MySQL, Sequelize ORM и NestJS, позволяет создать надёжную и масштабируемую информационную систему.

С полной версией разработанной ER диаграммы базы данных, можно ознакомиться в приложении Б, а также с полным листингом кода схемы базы данных вы можете ознакомиться в приложении А.

## 2.3 Неформальное описание алгоритма работы ПП

Алгоритм работы информационной системы "Нарушениям.нет" можно разделить на несколько модулей, которые взаимодействуют между собой. Основные модули и их функциональность:

Модуль регистрации и авторизации:

* Регистрация новых пользователей: Гражданин вводит данные, такие как ФИО, телефон, адрес электронной почты, логин и пароль. Модуль проверяет уникальность логина и сохраняет данные в базе данных.
* Аутентификация пользователей: Пользователи могут входить в систему с помощью уникальных учетных данных (логин и пароль). Модуль проверяет введенные данные, сверяет их с данными в базе данных и предоставляет доступ в случае успешной авторизации.

Модуль подачи заявлений:

* Создание заявлений: Пользователь вводит номер автомобиля нарушителя и описание нарушения. Модуль предоставляет интерфейс для ввода этих данных.
* Сохранение заявлений: Сохранение созданных заявлений в базе данных с указанием номера автомобиля, описания нарушения и текущего статуса (новое, подтверждено или отклонено).

Модуль обработки заявлений:

* Просмотр и изменение статусов заявлений: Администратор просматривает поступившие заявления и изменяет их статус на "подтверждено" или "отклонено".

Модуль базы данных:

* Хранение данных пользователей: Сохранение данных, введенных пользователями при регистрации.
* Хранение заявлений: Сохранение всех созданных заявлений и их текущих статусов.

Модуль интерфейса пользователя:

* Интерфейс регистрации и входа: Обеспечение удобного интерфейса для регистрации и входа.
* Интерфейс подачи заявлений: Предоставление интерфейса для подачи заявлений о нарушениях.
* Интерфейс администратора: Предоставление интерфейса для просмотра и управления заявлениями.

Эти модули взаимодействуют друг с другом для обеспечения полной функциональности системы "Нарушениям.нет". В приложении В представлена блок-схема работы системы.

## 2.4 Функции подсистем (модулей)

В информационной системе "Нарушениям.нет" присутствуют клиентский и серверный модули.

Клиентский модуль представляет собой пользовательский интерфейс, через который пользователи взаимодействуют с системой. Он обеспечивает выполнение различных функций и предоставление информации.

Функции клиентского модуля включают:

* Авторизация и аутентификация: Пользователи могут входить в систему, используя свои учетные данные. Модуль обеспечивает процесс аутентификации и проверку прав доступа.
* Просмотр и редактирование данных: Пользователи могут просматривать информацию о своих заявлениях, а также редактировать их.
* Подача заявлений: Пользователь может создать новое заявление, указав номер автомобиля и описание нарушения.
* Просмотр статусов заявлений: Пользователи могут отслеживать статус своих заявлений (новое, подтверждено, отклонено).
* Управление профилем пользователя: Возможность управлять профилем, например, изменять пароль.
* Серверный модуль выполняет централизованную обработку данных, бизнес-логику и взаимодействие с клиентскими модулями.

Функции серверного модуля включают:

* Управление пользователями и аутентификация: Обработка запросов на авторизацию, проверка данных и предоставление доступа.
* Взаимодействие с базой данных: Выполнение операций чтения, обновления и удаления данных, обеспечение целостности и безопасности информации.
* Бизнес-логика системы: Правила и процессы обработки данных, формирование ответов на запросы клиентского модуля.
* Обработка запросов от клиентского модуля: Прием и обработка запросов, выполнение операций, получение данных из базы данных и отправка ответов.
* Обеспечение безопасности: Проверка прав доступа, шифрование данных и защита от несанкционированного доступа.

Эти модули работают совместно, обеспечивая надежную и эффективную работу системы "Нарушениям.нет".

# 2.5 Оценка экономической эффективности от внедрения программы

Оценка экономической эффективности проекта "Разработка информационной системы для осознанных граждан 'Нарушений.нет'" включает в себя анализ затрат и выгод, связанных с разработкой и внедрением данной системы. Эта оценка помогает определить, насколько рационально использованы ресурсы и оправданы ли вложенные средства.

Для начала необходимо составить перечень всех этапов работ, которые будут выполнены в ходе разработки проекта. Это поможет структурировать процесс и учесть все необходимые действия и затраты.

Таблица 6 – Распределение работ по этапам и видам

|  |  |  |
| --- | --- | --- |
| Этап проведения | Вид работы на данном этапе | Трудоемкость выполнения, чел.-ч. |
| Исследование предметной области | Изучение требований и характеристик системы | 5 |
| Проектирование | Разработка архитектуры системы | 15 |
| Разработка | Написание кода | 250 |
| Тестирование | Проверка и отладка кода | 30 |
| Внедрение | Установка и настройка системы | 10 |
| Итого |  | 310 |

Для определения затрат на разработку проекта нужно составить смету со следующими статьями: затраты на оплату труда, отчисления на социальные нужды, амортизация основных фондов и прочие затраты.

Таблица 7 – Затраты на оплату труда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория работника | Квалификация | Трудоемкость, чел.-ч. | Часовая ставка, руб/ч | Сумма, руб |
| Разработчик | Программист | 310 | 300 | 93 000 |
| Руководитель проекта | Менеджер | 20 | 500 | 10 000 |
| Итого |  |  |  | 103 000 |

Общая сумма затрат на оплату труда рассчитывается по формуле (1):

Обозначения данной формулы:

* – часовая ставка i-го работника, руб.;
* - время на разработку игры, час.;
* i– категория работника;
* n – количество работников, разрабатываемых игру.

Среднечасовая заработная плата разработчика рассчитывается по формуле (2):

, (2)

Расчет затрат на оплату труда:

Отчисления на социальные нужды включают в себя единый социальный налог и взносы на страхование от несчастных случаев, составляющие 30% и 0,2% от затрат на оплату труда:

Общая сумма амортизационных отчислений определяется по формуле (3):

, (3)

Обозначения данной формулы:

* – стоимость i-го оборудования, руб.;
* – годовая норма амортизации i-го оборудования, %;
* i – вид оборудования;
* n – количество оборудования.

Сумма амортизационных отчислений составит:

Таблица 8 – Расчет амортизационных отчислений

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование оборудования | Стоимость оборудования, руб. | Годовая норма амортизации, % | Сумма, руб. |
| Компьютер (корпус) | 50 000 | 20 | 10 000 |
| Монитор | 15 000 | 20 | 3 000 |
| Итого |  |  | 13 000 |

Прочие затраты включают расходы на содержание административного и вспомогательного персонала, коммунальные услуги и другие хозяйственные расходы, и рассчитываются как 30% от затрат на оплату труда:

Итоговая смета затрат на разработку проекта "Нарушений.нет" приведена в таблице 4.

Таблица 9 – Смета затрат на разработку проекта

|  |  |
| --- | --- |
| Статьи затрат | Сумма, руб. |
| Затраты на оплату труда | 103 900 |
| Отчисления на социальные нужды | 31 577,8 |
| Амортизация основных фондов | 13 000 |
| Прочие затраты | 31 170 |
| Итого по смете | 179 647,8 |

Договорная цена для сайта рассчитывается по формуле (4):

, (4)

Обозначения данной формулы:

* – затраты на разработку сайта, руб.;
* – средний уровень рентабельности, 25%.

Исходя из этого, цена данного сайта, будет следующей:

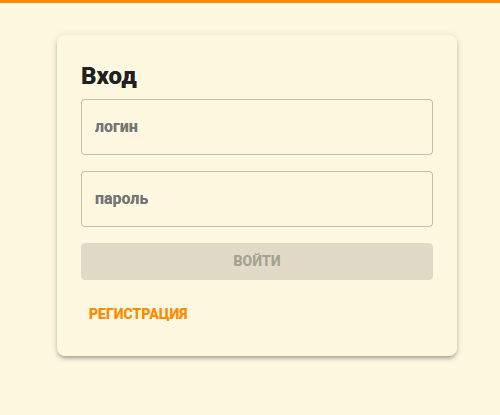
Итоговая стоимость проекта составит:

# ГЛАВА 3 ЭСКИЗНЫЙ ПРОЕКТ

## 3.1 Верстка фронтенда

В этом разделе рассматриваются связи между страницами веб-приложения «Нарушениям.Нет», а также детально описываются страницы, из которых состоит фронтенд.

На рисунке ниже изображена форма входа, которая является одной из первых страниц, доступных пользователю.

  
Рисунок 6 - Форма входа

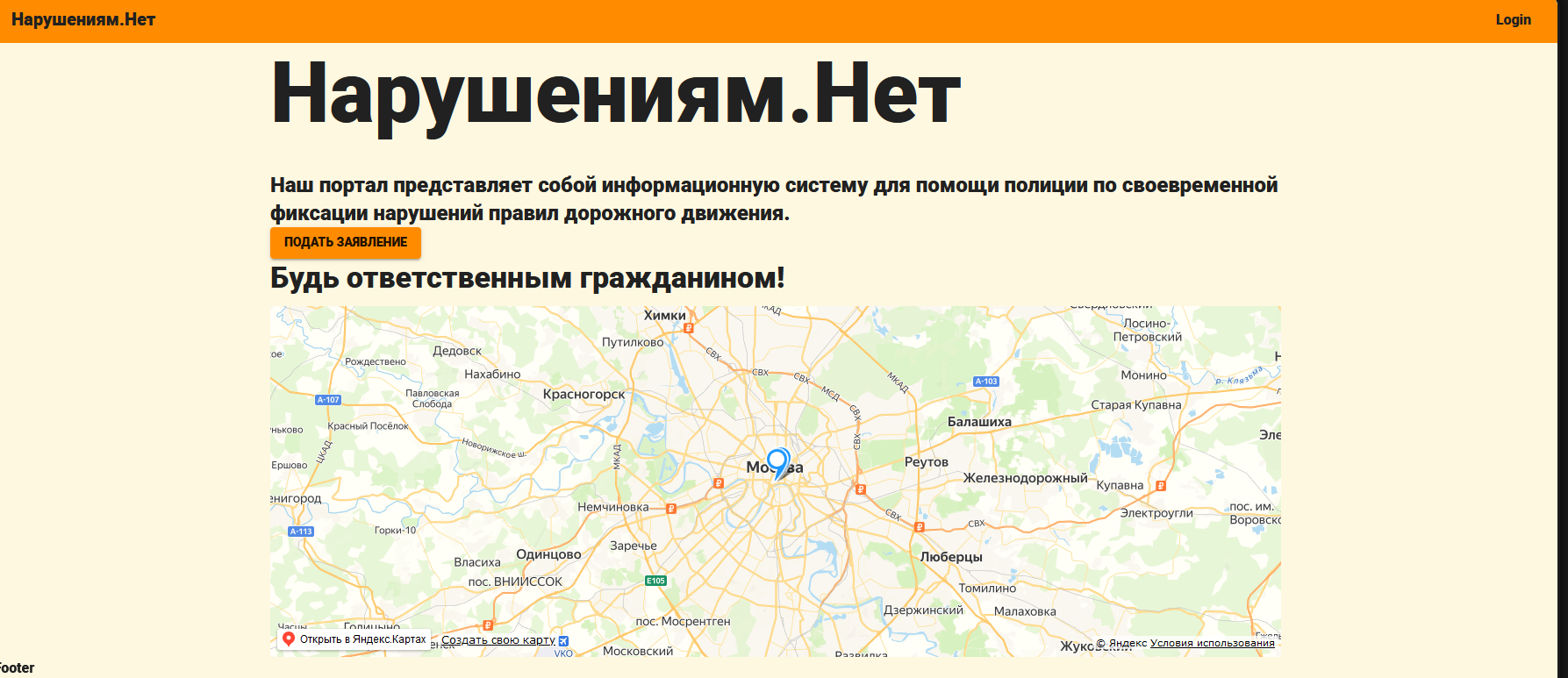
Пользователь имеет начальный доступ к трем основным страницам: вход, регистрация и главная страница. Страница входа включает поля для ввода логина и пароля, а также кнопку для перехода на страницу регистрации. При нажатии на кнопку "Войти" происходит проверка введенных данных и, в случае успешной авторизации, пользователь перенаправляется на главную страницу.

На рисунке ниже изображена форма регистрации, которая запрашивает у пользователя ввод данных, таких как ФИО, телефон, адрес электронной почты, логин и пароль.

****  
Рисунок 7 - Форма регистрации

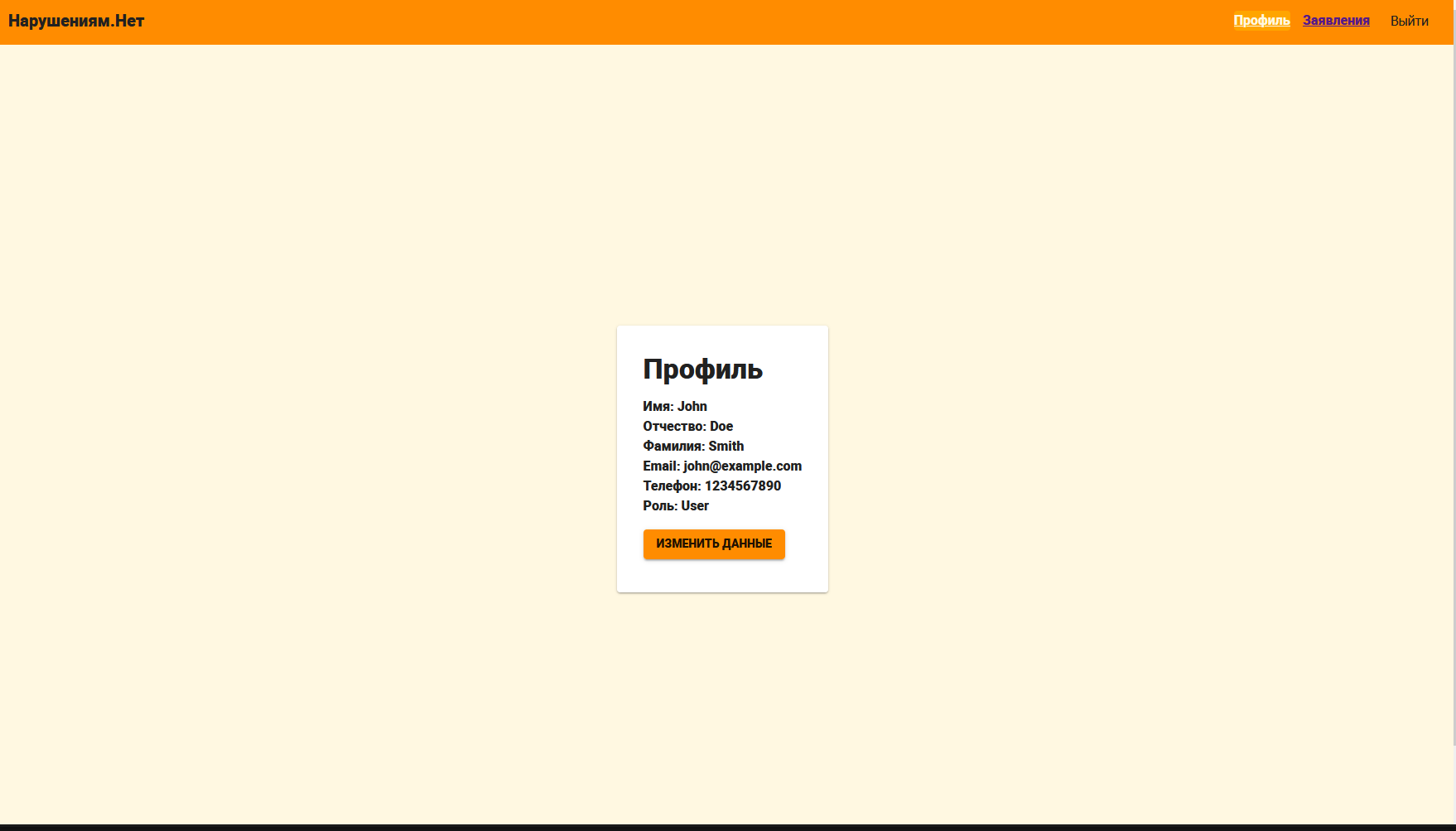
При успешной регистрации пользователь может войти в систему, используя указанный логин и пароль. Главная страница приветствует пользователя и предоставляет доступ к другим разделам сайта, таким как профиль пользователя и список заявлений.

На рисунке ниже изображена главная страница после авторизации, где отображаются приветственное сообщение и ссылки на другие разделы.

  
Рисунок 8 - Главная страница

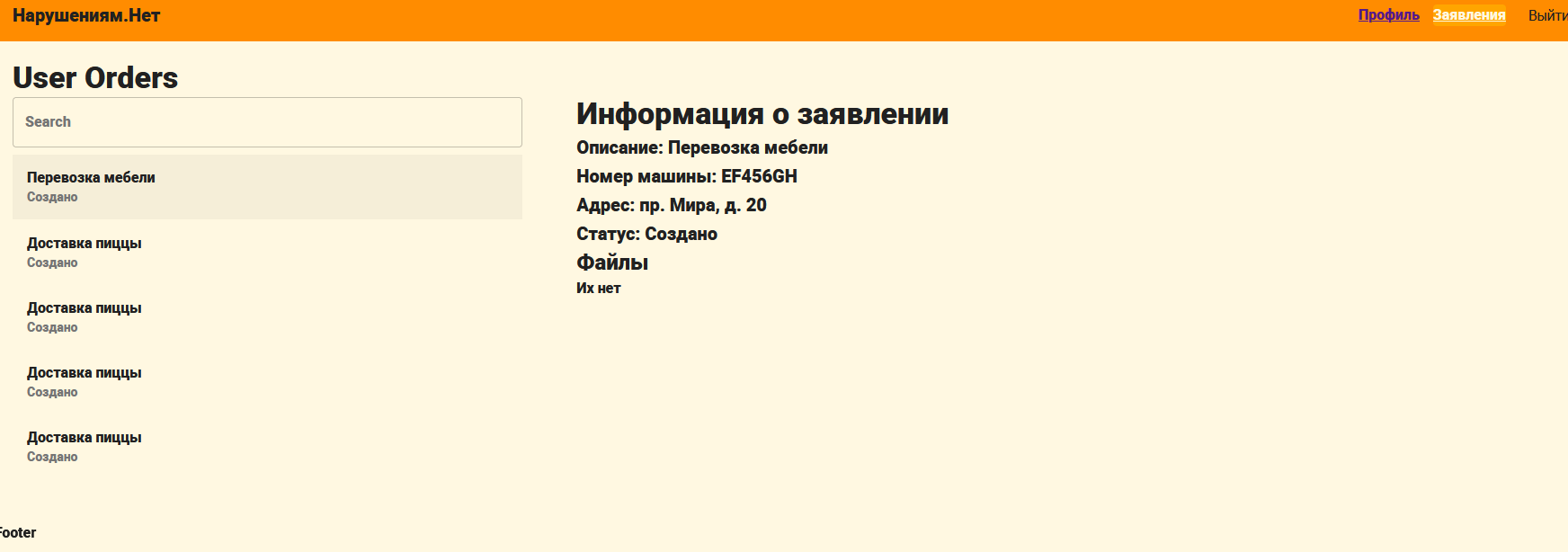
После авторизации пользователь получает доступ к своему профилю, где отображается общая информация о нем, а также список заявлений. В профиле можно увидеть список всех созданных пользователем заявлений. При нажатии на любое заявление открывается подробная информация о нем, включающая номер автомобиля, описание нарушения и статус заявки (новое, подтверждено, отклонено).

На рисунке ниже показан интерфейс профиля пользователя, включающий список заявлений и основную информацию о пользователе.

  
Рисунок 9 - Профиль пользователя

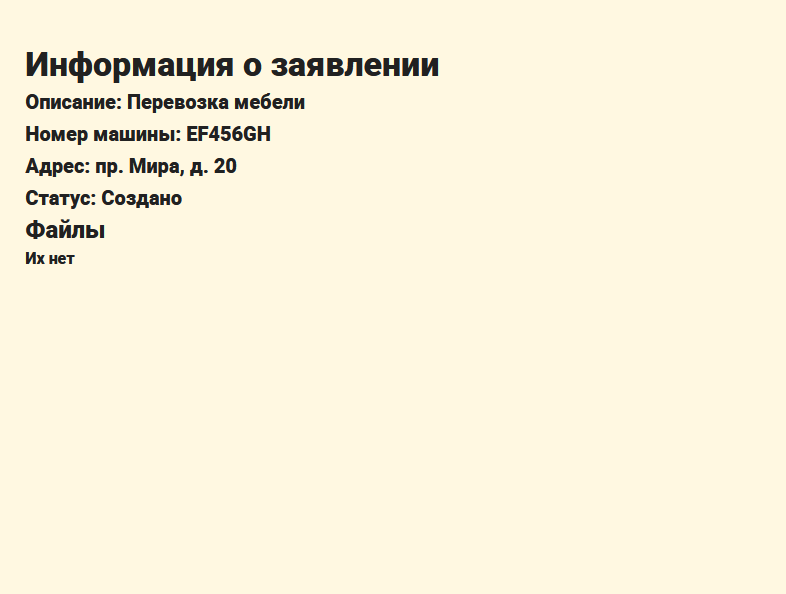
На этой странице можно просмотреть данные пользователя и при необходимости изменить данные

На рисунке ниже показан интерфейс заявлений пользователя, включающий список заявлений и статус.

  
Рисунок 10 - Профиль пользователя

При нажатии на элемент списка заявлений открывается страница с детальной информацией о конкретном заявлении. Эта страница включает все важные данные о заявлении, а также кнопку для возврата к списку заявлений.

На рисунке ниже представлена страница с подробной информацией о заявлении.

  
Рисунок 11 - Подробное заявление

Таким образом, навигация между страницами организована логично и последовательно, позволяя пользователю легко перемещаться между формами входа и регистрации, профилем и списком заявлений, а также просматривать и управлять поданными заявлениями.

Информационная система «Нарушениям.Нет» предназначена для помощи гражданам и полиции в фиксации нарушений правил дорожного движения. Граждане могут зарегистрироваться, входить в систему, подавать заявления о нарушениях, а администраторы могут обрабатывать эти заявления, подтверждая или отклоняя их.

# ГЛАВА 4 РАЗРАБОТКА ПРОЕКТНЫХ РЕШЕНИЙ

## 4.1 Разработка серверных решений

Разработка серверных решений является ключевым этапом при создании информационных систем, таких как портал сознательных граждан «Нарушениям.Нет». Этот этап включает проектирование и реализацию серверной части приложения, обеспечивающей хранение и обработку данных, а также взаимодействие с клиентскими приложениями. Для достижения этой цели используются современные технологии и инструменты, такие как NestJS для построения сервера, MySQL для управления базой данных, и различные инструменты для обеспечения безопасности и масштабируемости системы.

@Injectable()

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

constructor() {

super({

jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

ignoreExpiration: false,

secretOrKey: jwtConstants.secret,

});

}

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> {

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

Разработка информационной системы для осознанных граждан "Нарушений.нет" завершена успешно, и ее внедрение демонстрирует значительный потенциал в улучшении процесса фиксации нарушений правил дорожного движения. В ходе работы были достигнуты все поставленные цели и решены задачи, что подтверждает соответствие разработанного программного продукта требованиям и техническому заданию.

Одним из основных результатов работы является создание интуитивно понятного интерфейса для регистрации пользователей и формирования заявлений о нарушениях. Применение современных технологий React и Nest позволило обеспечить высокую производительность и надежность системы. База данных, разработанная на основе MySQL, гарантирует безопасность и целостность данных пользователей, что является важным аспектом при работе с конфиденциальной информацией.

Практическая значимость проекта подтверждается его потенциалом для повышения эффективности фиксации нарушений правил дорожного движения. В будущем внедрение системы позволит гражданам оперативно сообщать о нарушениях, что, в свою очередь, поможет правоохранительным органам быстрее реагировать и принимать меры. Предполагается, что система "Нарушений.нет" значительно облегчит процесс взаимодействия между гражданами и полицией, улучшая общую безопасность на дорогах.

С точки зрения соответствия разработанного программного продукта требованиям и техническому заданию, проект выполняется в полном соответствии с заданными параметрами. Все разработанные модули прошли предварительное тестирование и показали свою работоспособность и надежность.

Перспективы дальнейшей разработки темы включают расширение функционала системы и интеграцию новых технологий для улучшения пользовательского опыта. В будущем планируется добавить мобильное приложение, что позволит гражданам еще удобнее использовать сервис на своих мобильных устройствах. Также рассматривается возможность внедрения автоматических систем распознавания нарушений на основе видеозаписей и фотографий, что позволит повысить точность и оперативность фиксации нарушений.

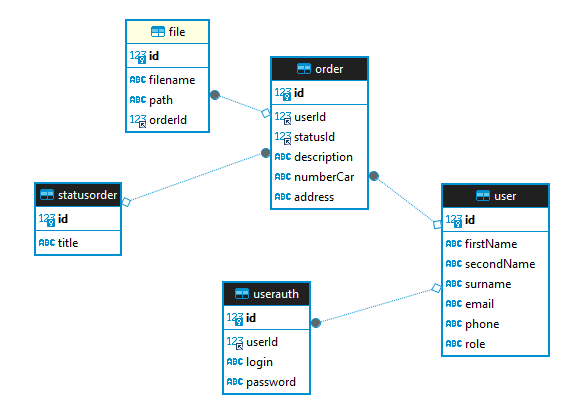
Кроме того, возможно интеграция с другими государственными информационными системами, что облегчит обмен данными и сделает процесс администрирования еще более эффективным. Внедрение элементов машинного обучения и искусственного интеллекта позволит автоматизировать часть процессов, связанных с обработкой заявлений и улучшением их анализа.

Таким образом, информационная система "Нарушений.нет" представляет собой значимый вклад в обеспечение правопорядка на дорогах. Ее внедрение способствует улучшению взаимодействия между гражданами и правоохранительными органами, что в конечном итоге ведет к повышению общей безопасности и снижению количества правонарушений. Перспективы дальнейшего развития системы позволяют надеяться на ее дальнейшее совершенствование и расширение функциональных возможностей.

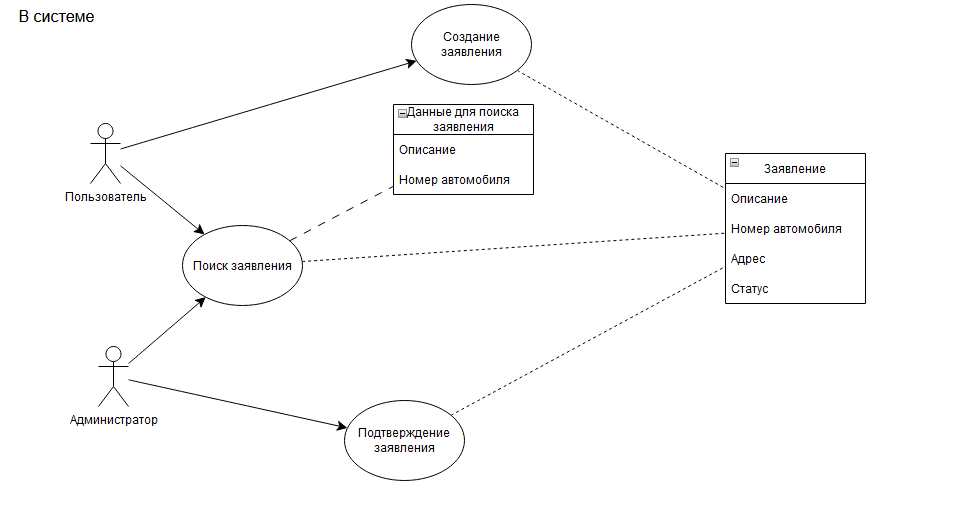
# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

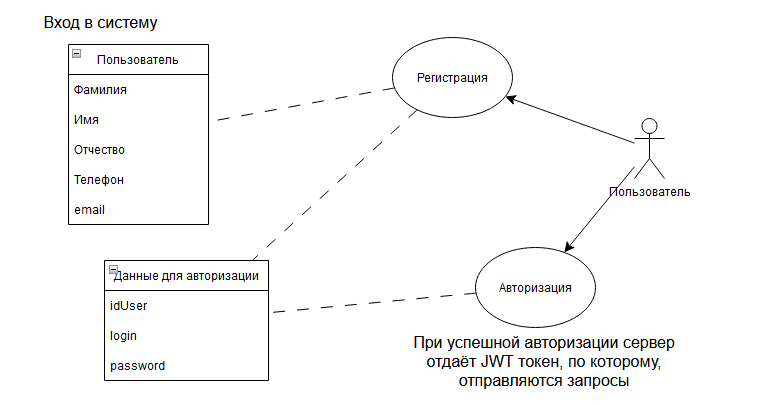
1. A. Holmes. NestJS: A Progressive Node.js Framework. – Packt Publishing, 2023.
2. B. Randell. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers. – Apress, 2023.
3. C. Wenz. Advanced GraphQL with Node.js and React. – Packt Publishing, 2023.
4. D. Koppenhaver. Fullstack React with TypeScript. – Newline, 2022.
5. DBeaver – Free Universal Database Tool // DBeaver URL: <https://dbeaver.io/> (дата обращения: 2.05.2024).
6. DBeaver User Guide // DBeaver URL: https://dbeaver.io/documentation/ (дата обращения: 11.05.2024).
7. Frontend Development Trends 2023 // Smashing Magazine URL: https://www.smashingmagazine.com/2023/01/frontend-development-trends-2023/ (дата обращения: 15.05.2024).
8. GitHub – Where the world builds software // GitHub URL: <https://github.com/> (дата обращения: 31.05.2024).
9. J. D. Shaw. MySQL 8 Administrator’s Guide. – Apress, 2023.
10. K. Cunningham. Learning React: Modern Patterns for Developing React Apps. – O'Reilly Media, 2022.
11. M. Lodder. Pro Git: Everything You Need to Know About Git. – Apress, 2022.
12. MySQL – The world's most popular open source database // MySQL URL: <https://www.mysql.com/> (дата обращения: 31.05.2024).
13. NestJS – A progressive Node.js framework for building efficient, reliable, and scalable server-side applications // NestJS URL: <https://nestjs.com/> (дата обращения: 21.04.2024).
14. P. DuBois. MySQL Cookbook: Solutions for Database Developers and Administrators. – O'Reilly Media, 2022.
15. Pro Git (2nd Edition) // GitHub URL: <https://git-scm.com/book/en/v2> (дата обращения: 14.05.2024).
16. React – A JavaScript library for building user interfaces // React URL: <https://reactjs.org/> (дата обращения: 31.05.2024).
17. Visual Studio Code – Code editing. Redefined. // Visual Studio Code URL: <https://code.visualstudio.com/> (дата обращения: 21.03.2024).
18. Web Development Trends in 2024 // FreeCodeCamp URL: https://www.freecodecamp.org/news/web-development-trends-in-2024/ (дата обращения: 31.04.2024).
19. Оказание первой помощи в походе // турклуб ПИК URL: <https://turclub-pik.ru/blog/okazanie-pervoj-pomoshhi-v-pokhode/> (дата обращения: 11.02.2023).
20. Современные системы управления базами данных // Habr URL: https://habr.com/ru/company/otus/blog/546418/ (дата обращения: 1.04.2024).

# ПРИЛОЖЕНИЕ A

  
Рисунок 12- Логическая модель данных проекта в виде ER диаграммы

# ПРИЛОЖЕНИЕ Б

  
Рисунок 13 - UseCase диаграмма Приложения

  
Рисунок 14 - UseCase диаграмма Входа

# ПРИЛОЖЕНИЕ В

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;

}

}