# ДОКУМЕНТАЦИЯ К ПРОЕКТУ

**Описание проекта**

Данный проект представляет собой ASP.NET Core Web API, разработанный для управления списком пользователей и их ролями. Проект включает в себя реализацию операций CRUD (Create, Read, Update, Delete) для управления пользователями и их ролями. Каждый пользователь имеет уникальный идентификатор (Id), имя (Name), возраст (Age), электронную почту (Email), а также связанные роли (Role). Роли включают в себя следующие значения: User, Admin, Support и SuperAdmin. Каждый пользователь может иметь несколько ролей.

Технический стек

- ASP.NET Core (Web API);
- Entity Framework Core;
- PostgreSQL (база данных);
- JWT-токены для аутентификации и авторизации;
- LINQ для запросов к базе данных;
- Swagger для документирования API;
- Serilog для логирования действий в API.

Проводится валидация данных в контроллерах, включая проверку наличия обязательных полей (Имя, Возраст, Email), уникальности Email и положительности возраста. Обработка ошибок и возврат соответствующих статусных кодов HTTP (например, 404 при отсутствии пользователя). Используемые технологии:

Entity Framework Core для доступа к данным и сохранения пользователей и ролей в базе данных. Создание миграций для создания необходимых таблиц в базе данных. Документирование API с использованием инструментов Swagger.

При взаимодействии с API в проекте используются стандартные статус-коды HTTP, чтобы информировать клиентов о результатах выполнения запросов. Вот некоторые из использованных при создании Web Api статус-кодов и их значения:

Успешные запросы (2xx)

200 OK: Запрос успешно выполнен. Возвращается при успешном выполнении операции, например, при создании, обновлении или успешном поиске ресурсов.

Ошибки клиента (4xx)

400 Bad Request: Недопустимый запрос. Возвращается, когда запрос клиента некорректен, например, если отправлены неверные данные.

401 Unauthorized: Неавторизованный доступ. Возвращается, когда пользователь не предоставил достаточных аутентификационных данных или у него нет прав на выполнение операции.

403 Forbidden: Доступ запрещен. Возвращается, когда пользователь авторизован, но у него нет прав на выполнение операции.

404 Not Found: Ресурс не найден. Возвращается, когда запрашиваемый ресурс не существует.

Ошибки сервера (5xx)

500 Internal Server Error: Внутренняя ошибка сервера. Возвращается, когда на сервере произошла ошибка, которую невозможно обработать.

**Схема базы данных**

База данных использует PostgreSQL и содержит следующие таблицы:

1. Пользователи (Users):
   - Id (уникальный идентификатор пользователя)
   - Name (имя пользователя)
   - Age (возраст пользователя)
   - Email (электронная почта пользователя)
2. Роли (Roles):
   - Id (уникальный идентификатор роли)
   - Role (наименование роли)
3. Таблица для связи пользователей и ролей (UserRoles):
   - UserId (идентификатор пользователя)
   - RoleId (идентификатор роли)

Схема базы данных приведена на рисунке 1.

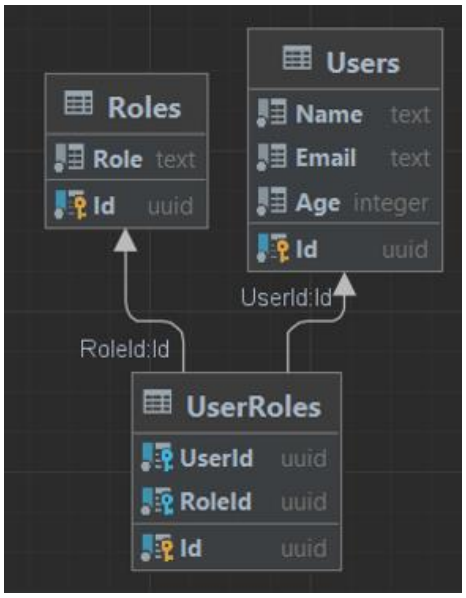


Рисунок 1 – Схема базы данных

Связь между пользователями и их ролями реализована через таблицу UserRoles в модели «Многие ко многим» (Many-to-Many).

JWT Токены

Для аутентификации и авторизации в API используются JWT-токены (JSON Web Tokens). Для создания JWT-токена необходимо предоставить Email пользователя, и производится проверка уникальности этого Email. Токен содержит информацию о ролях пользователя, что позволяет управлять доступом к определенным ресурсам и действиям в API.

**Документация API**

API документирована с использованием инструмента Swagger. Вы можете использовать Swagger для получения полной информации о доступных методах, запросах и ответах. Здесь приведены некоторые из доступных эндпоинтов:

**Получение списка всех пользователей**

- **Метод**: GET
- **Путь**: /api/user/getUsers
- **Описание**: Получение списка всех пользователей с поддержкой пагинации, сортировки и фильтрации по различным атрибутам.
- **Пример запроса**:

```json
{
    "pageNumber": 1,
    "pageSize": 10
}
```

- **Пример ответа**:

```json
[
    {
        "id": "1",
        "name": "John Doe",
        "age": 30,
        "email": "john.doe@example.com",
        "roles": ["User"]
    },
    {
        "id": "2",
        "name": "Admin User",
        "age": 35,
        "email": "admin@example.com",
        "roles": ["Admin"]
    }
]
```

**Создание нового пользователя**

- **Метод**: POST
- **Путь**: /api/user/create
- **Описание**: Создание нового пользователя.
- **Пример запроса**:

```json
{
```

```
            "name": "Alice",
            "age": 25,
            "email": "alice@example.com"
        }
        ```
    - **Пример ответа**:
        ```
        "User created successfully"
        ```
```

### Получение пользователя по Id

```
    - **Метод**: GET
    - **Путь**: /api/user/getUser/{id}
    -  **Описание**: Получение информации о пользователе по его уникальному идентификатору.
    - **Пример ответа**:
        ```json
        {
            "id": "1",
            "name": "John Doe",
            "age": 30,
            "email": "john.doe@example.com",
            "roles": ["User"]
        }
      ```
```

### Добавление роли пользователю

```
    - **Метод**: POST
    - **Путь**: /api/user/addRole
    - **Описание**: Добавление новой роли к существующему пользователю.
    - **Пример запроса**:
        ```json


        {
            "userId": "1",
            "roleId": "2"
        }
        ```
    - **Пример ответа**:
        ```
        "Role added to user successfully"
      ```
```

### Редактирование информации о пользователе

```
    - **Метод**: PUT
    - **Путь**: /api/user/editUser
    - **Описание**: Редактирование информации о пользователе.
    - **Пример запроса**:
        ```json
        {
            "id": "1",
            "name": "Updated Name",
            "age": 32,
            "email": "updated@example.com"
        }
```

```
    - **Пример ответа**:
      ```
      "User with Id 1 has been updated."
      ```

    Удаление пользователя по Id
    - **Метод**: DELETE
    - **Путь**: /api/user/deleteUser/{id}
    - **Описание**: Удаление пользователя по его уникальному идентификатору.
    - **Пример ответа**:
      ```
      "User(1) has been deleted."
      ```
```

**Заключение**

Этот проект предоставляет мощный инструмент для управления пользователями и их ролями с помощью удобного API. Он также обеспечивает безопасность с использованием JWT-токенов и логирование действий с помощью Serilog. Ссылка на репозиторий проекта будет предоставлена для оценки и дальнейшего рассмотрения.