МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение видеохостинг «BYTUBE»

Выполнил студент Окулич Дмитрий Юрьевич

(Ф.И.О.)

Руководитель проекта асс. Некрасова Н.П.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доцент Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2024

Содержание

[Введение 3](#_Toc185857113)

[1 Проектирования веб-приложения 4](#_Toc185857114)

[1.1 Разработка функционала 4](#_Toc185857115)

[1.2 Архитектура приложения 6](#_Toc185857116)

[1.3 Архитектура база данных 8](#_Toc185857117)

[1.4 Вывод 12](#_Toc185857118)

[2 Реализация веб-приложения 13](#_Toc185857119)

[2.1 Реализация базы данных 13](#_Toc185857120)

[2.2 Реализация серверной части 14](#_Toc185857121)

[2.2.1 Просмотра публичных видео 14](#_Toc185857122)

[2.2.2 Поиск публичных видео 15](#_Toc185857123)

[2.2.3 Регистрации, аутентифицироваться и авторизации 16](#_Toc185857124)

[2.2.4 Загрузка, редактирование и удаление видео 17](#_Toc185857125)

[2.2.5 Добавление, редактирование, удаление комментариев 18](#_Toc185857126)

[2.2.6 Создание, просмотр жалоб 18](#_Toc185857127)

[2.2.7 Создания или удаление плейлистов 19](#_Toc185857128)

[2.2.8 Подписка на канал 20](#_Toc185857129)

[2.2.9 Скрытия видео администратором 20](#_Toc185857130)

[2.3 Реализация клиентской части 20](#_Toc185857131)

[2.4 Вывод 21](#_Toc185857132)

[3 Тестирование веб-приложения 22](#_Toc185857133)

[3.1 Функциональное тестирование 22](#_Toc185857134)

[3.2 Автоматизированное тестирование 25](#_Toc185857135)

[3.3 Нагрузочное тестирования 25](#_Toc185857136)

[3.4 Вывод 26](#_Toc185857137)

[4 Руководство программиста 27](#_Toc185857138)

[Заключение 28](#_Toc185857139)

[Список используемых источников 29](#_Toc185857140)

[Приложение А 30](#_Toc185857141)

[Приложение Б 33](#_Toc185857142)

[Приложение В 35](#_Toc185857143)

[Приложение Г 52](#_Toc185857144)

[Приложение Д 54](#_Toc185857145)

# Введение

Цель данной работы заключается в создании веб-приложения для видеохостинга "Bytube".

Цель веб-приложения — обеспечить пользователям удобную платформу для загрузки, просмотра и взаимодействия с видеоконтентом. Приложение предлагает возможности для создания и управления каналами, плейлистами, а также для общения через комментарии.

Для разработки веб-приложения была выбрана монолитная архитектура. Серверная часть реализована на платформе ASP.NET Core, клиентская часть — с использованием библиотеки React и MobX. В качестве системы управления базами данных используется PostgreSQL, с подключением через Entity Framework Core. Для передачи данных используется REST API. Приложение развёрнуто в Docker.

Основные задачи приложения:

* обеспечить удобный и безопасный доступ к видеоконтенту;
* предоставить пользователям возможности взаимодействия с видеоматериалами, включая их загрузку, редактирование и удаление;
* создать интуитивно понятный и адаптивный интерфейс для пользователей;
* обеспечить быструю и надёжную обработку запросов между клиентом и сервером.

Для обеспечения безопасности пользователей используется валидация данных, управление ролями и политики авторизации.

Основные функциональные требования к приложению:

Роли пользователей:

* Гость: регистрация, аутентификация и авторизация; просмотр и поиск публичных видео;
* Клиент: загрузка, редактирование и удаление своих видео; добавление, редактирование и удаление комментариев к своим видео; создание и удаление плейлистов; подписка на каналы; подача жалоб;
* Администратор: просмотр и обработка жалоб; скрытие и удаление видео; управление комментариями.

Приложение предназначено для широкого круга пользователей, включая:

* обычных зрителей (гостей), которые хотят найти и просмотреть необходимый им видеоконтент;
* авторов контента, заинтересованных в создании и публикации своих видео;
* администраторов, отвечающих за модерацию и поддержание контента.

В пояснительной записке содержится информация о схожих продуктах, архитектуре и реализации проекта, тестировании веб-приложения, а также инструкции по его использованию.

# Проектирования веб-приложения

# Разработка функционала

Диаграмма вариантов использования иллюстрирует ключевые действия, доступные различным ролям пользователей веб-приложения «Bytube». Она помогает визуализировать взаимодействие пользователей с системой в зависимости от их роли (гость, клиент, администратор). Диаграмма вариантов использования предоставлена на рисунке 1.1.

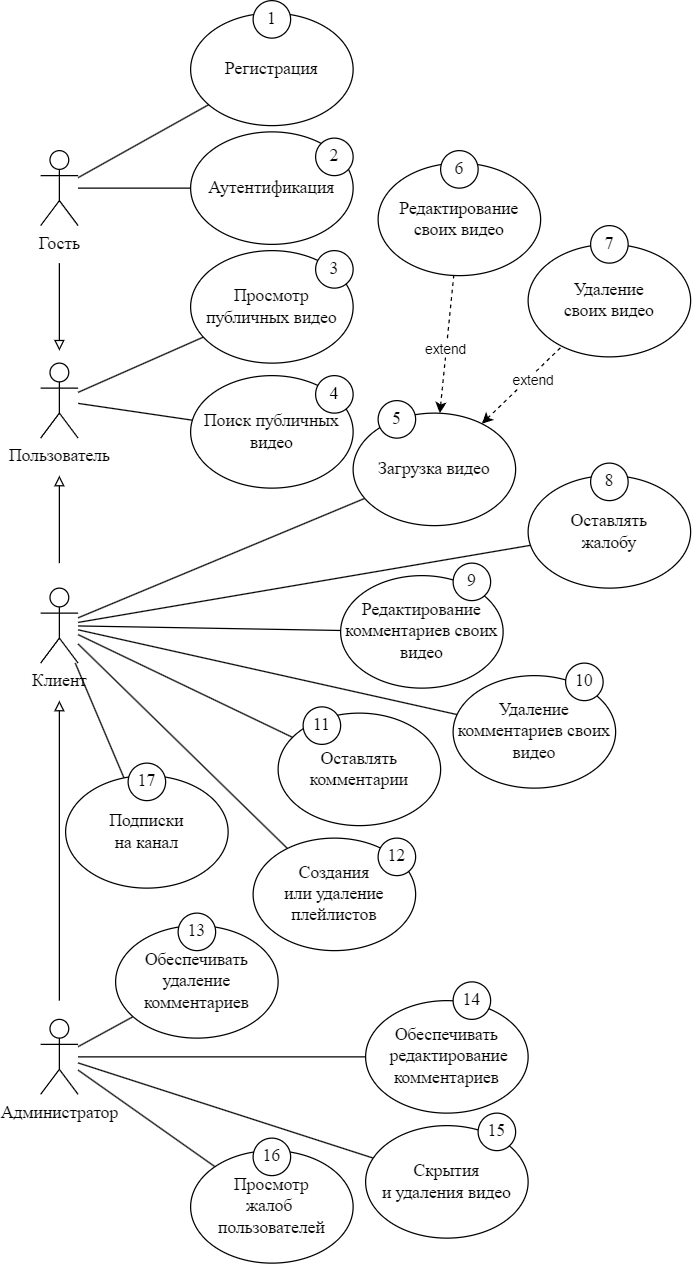


Рисунок 1.1 – Диаграмма вариантов использования

Таким образом на диаграмме видно, что какой пользователь делает. Описание ролей предоставлена на таблице 1.1.

Таблица 1.1 – описание ролей

|  |  |
| --- | --- |
| Роль | Описание |
| Гость | Пользователь, не прошедший регистрацию, с доступом к публичным видео и базовым функциям. |
| Клиент | Зарегистрированный пользователь, имеющий возможность взаимодействовать с контентом. |
| Администратор | Уполномоченный пользователь, отвечающий за модерацию и управление приложением. |

Роли в системе разделены таким образом, чтобы разграничить доступ к функционалу приложения и обеспечить безопасное использование платформы. Каждая роль наделена строго определённым набором возможностей, что упрощает управление и поддержку системы.

Описание функций предоставлена на таблице 1.2.

Таблица 1.2 – описание функций

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание | Доступна ролям |
| 1 | Регистрация | Возможность создания нового аккаунта для получения доступа к расширенному функционалу. | Гость |
| 2 | Аутентификация | Вход в систему для использования персонализированного функционала. | Гость |
| 3 | Просмотр публичных видео | Возможность смотреть видео, доступные для всех пользователей. | Гость, Клиент, Администратор |
| 4 | Поиск публичных видео | Возможность находить видео, по ключевым словам, или тегам. | Гость, Клиент, Администратор |
| 5 | Загрузка видео | Возможность загружать свои видеоролики на платформу. | Клиент, Администратор |
| 6 | Редактирование своих видео | Изменение загруженных видеороликов (название, описание, настройки). | Клиент, Администратор |
| 7 | Удаление своих видео | Удаление ранее загруженных видеороликов. | Клиент, Администратор |
| 8 | Оставлять жалобу | Возможность отправить жалобу на видеоролики или другой контент. | Клиент, Администратор |
| 9 | Редактирование комментариев своих видео | Изменение комментариев, оставленных под своими видеороликами. | Клиент, Администратор |

Продолжение таблицы 1.2

|  |  |  |  |
| --- | --- | --- | --- |
| № | Функция | Описание | Доступна ролям |
| 10 | Удаление комментариев своих видео | Удаление комментариев, оставленных под своими видеороликами. | Клиент, Администратор |
| 11 | Оставлять комментарии | Добавление комментариев под видеороликами. | Клиент, Администратор |
| 12 | Создание или удаление плейлистов | Возможность организовывать видео в плейлисты или удалять их. | Клиент, Администратор |
| 13 | Обеспечивать удаление комментариев | Удаление нежелательных комментариев под любыми видео на платформе. | Администратор |
| 14 | Обеспечивать редактирование комментариев | Изменение комментариев на платформе для устранения нарушений. | Администратор |
| 15 | Скрытие и удаление видео | Возможность скрывать или удалять видеоролики, нарушающие правила платформы. | Администратор |
| 16 | Просмотр жалоб пользователей | Обзор поступивших жалоб для модерации и принятия мер. | Администратор |
| 17 | Подписка на канал | Возможность пользователю подписаться на канал. | Клиент, Администратор |

Функции приложения разделены между ролями таким образом, чтобы обеспечить удобство использования для каждой категории пользователей. Например, гости могут только просматривать и искать публичные видео, что снижает нагрузку на систему, так как им не нужно хранить данные о своих действиях. Клиенты, в свою очередь, обладают более широким спектром возможностей, включая управление собственным контентом и подписки. Администраторы сосредоточены на обеспечении качества контента и модерации пользовательских жалоб.

Диаграмма вариантов использования позволяет визуализировать все основные функции и роли веб-приложения «Bytube». Разделение на роли обеспечивает структурированный доступ к функционалу и улучшает безопасность приложения. Таблицы с описанием ролей и функций дополняют диаграмму, предоставляя детальное представление о возможностях каждой категории пользователей.

# Архитектура приложения

Для обеспечения работы приложения «Bytube» используется архитектура, включающая серверную часть и базу данных, размещенные в отдельных Docker-контейнерах. Этот подход позволяет изолировать сервисы друг от друга, что упрощает управление и масштабирование системы. Следующая диаграмма, предоставленная на рисунке 1.2 визуализирует основные компоненты архитектуры приложения и их размещение.

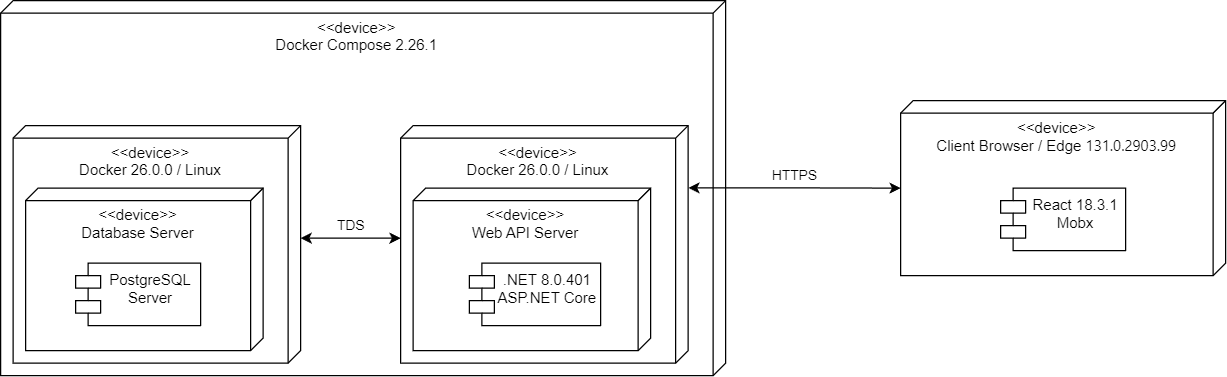


Рисунок 1.2 – Диаграмма размещения

Серверная часть приложения расположена внутри докер контейнеров «Bytube». Это решение предоставляет стабильность, минимальный объём занимаемого пространства и высокую производительность. Решение является оптимальным выбором для развертывания серверных приложений.

Для взаимодействия между клиентской и серверной частями приложения применяется протокол HTTPS (HTTP Secure), основанный на шифровании TLS версии 1.3. Это гарантирует защиту передаваемых данных, включая персональные данные пользователей и пароли, от перехвата и взлома. Использование HTTPS также является стандартом для современных веб-приложений, что повышает доверие пользователей и их уверенность в безопасности платформы.

В клиентской части приложения используется React 18.3.1 с управлением сборками через Webpack 5.88.0. Этот инструмент позволяет эффективно организовывать и оптимизировать код, обеспечивая минимальный размер файлов для передачи клиенту. Стилизация интерфейса выполнена с помощью MUI, что упрощает адаптацию приложения под различные устройства и экраны.

Для развертывания приложения "Bytube" используются два изолированных Docker-контейнера:

* Контейнер сервера на основе официального образа ASP.NET Core 8.0, предоставляемого Visual Studio, что обеспечивает высокую совместимость с инструментами разработки и стабильность при выполнении серверного приложения;
* Контейнер базы данных PostgreSQL версии 15, который позволяет эффективно работать с хранимыми данными.

Composer в проекте не используется, так как приложение построено на технологиях, не требующих его интеграции. Вместо этого в проекте применяются собственные механизмы управления зависимостями через NuGet для серверной части и npm для клиентской.

Таким образом архитектура приложения «Bytube» построена на современных технологиях, обеспечивающих производительность, безопасность и масштабируемость. Размещение серверной части в Docker контейнере, позволяет добиться стабильности и эффективности работы приложения.

Применение протокола HTTPS гарантирует безопасность передачи данных между клиентом и сервером, что особенно важно для защиты пользовательской информации. Клиентская часть, разработанная с использованием [1] React 18.2.0, и управление сборкой через [2] Webpack 5.88.0 обеспечивают быструю загрузку и интуитивно понятный интерфейс для пользователей.

Разделение серверной части и базы данных на отдельные Docker-контейнеры упрощает развертывание, обслуживание и масштабирование приложения. Этот подход позволяет легко управлять сервисами, обеспечивая их независимость друг от друга.

По итогу выходит продуманная архитектура и выбор технологий делают приложение «Bytube» надежным инструментом для видеохостинга, отвечающим современным требованиям безопасности и удобства использования.

# Архитектура база данных

Для реализации функциональности веб-приложения видеохостинга BYTUBE разработана реляционная база данных. Диаграмма базы данных предоставлена на рисунке 1.3.

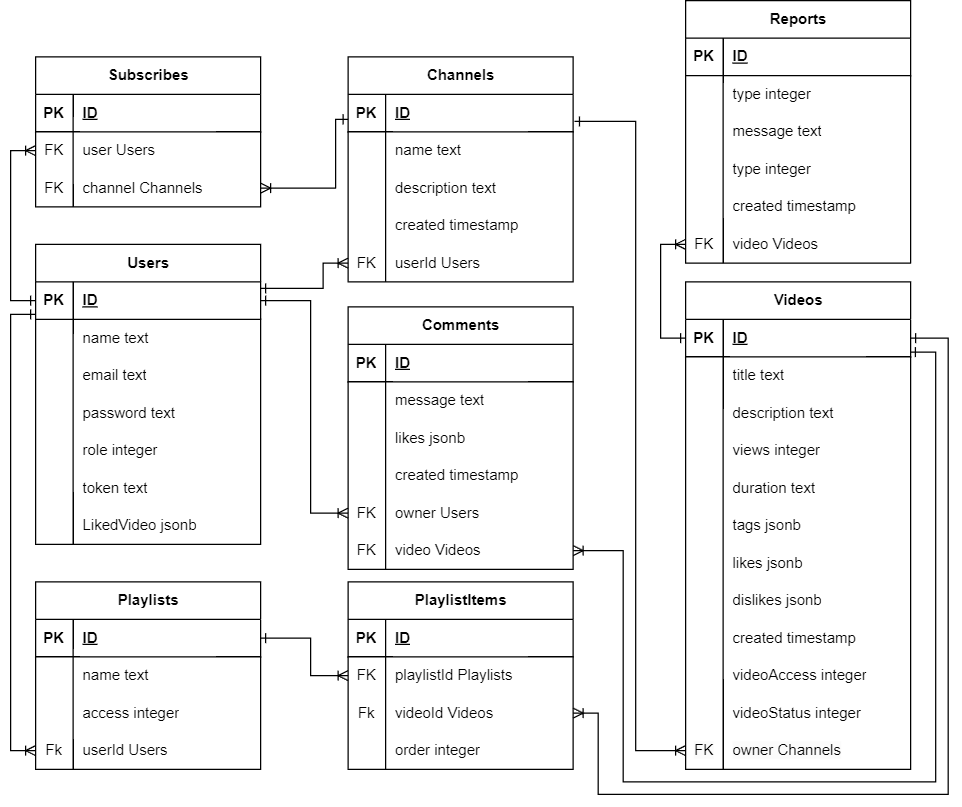


Рисунок 1.3 – Диаграмма базы данных

Структура базы данных спроектирована с учетом обеспечения гибкости и производительности при хранении и обработке данных. Основной акцент сделан на сохранении информации о пользователях, видео, каналах, комментариях, плейлистах и жалобах. В данной главе приведены основные таблицы, описаны их поля, а также связи между таблицами. Описание таблиц базы данных предоставлено в таблице 1.3.

Таблица 1.3 – Описание таблиц БД

|  |  |
| --- | --- |
| Таблица | Описание |
| Users | Хранение информации о пользователях |
| Channels | Хранение информации о каналах |
| Videos | Хранение данных о загруженных видео |
| Comments | Хранение комментариев к видео |
| Playlists | Хранение данных о плейлистах |
| PlaylistItems | Хранение информации о содержимом плейлистов |
| Subscribes | Хранение информации о подписках пользователей |
| Reports | Хранение жалоб пользователей на видео |

Каждая таблица базы данных включает набор полей, которые обеспечивают хранение и обработку данных, необходимых для работы приложения. Ниже представлены подробные описания всех полей для каждой таблицы.

Описание таблицы Users предоставлена в таблице 1.4.

Таблица 1.4 – Описание таблицы Users

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | integer PK | Уникальный идентификатор пользователя |
| name | text | Имя пользователя |
| email | text | Электронная почта пользователя |
| password | text | Хэш пароля |
| role | integer | Роль пользователя (гость, клиент, администратор) |
| token | text | Токен для аутентификации |
| LikedVideo | jsonb | Список понравившихся видео |

Описание таблицы Channels предоставлена в таблице 1.5.

Таблица 1.5 – Описание таблицы Channels

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | integer PK | Уникальный идентификатор канала |
| name | text | Название канала |
| description | text | Описание канала |
| created | timestamp | Дата создания канала |
| userId | integer FK | Внешний ключ, ссылающийся на таблицу Users |

Таблица Channels напрямую связана с таблицей Users, так как каждый канал создается и управляется конкретным пользователем.

Описание таблицы Videos предоставлена в таблице 1.6.

Таблица 1.6 – Описание таблицы Videos

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | integer PK | Уникальный идентификатор видео |
| title | text | Название видео |
| description | text | Описание видео |
| views | integer | Количество просмотров видео |
| duration | text | Продолжительность видео |
| tags | jsonb | Теги видео |
| likes | jsonb | Лайки видео |
| dislikes | jsonb | Дизлайки видео |
| created | timestamp | Дата загрузки видео |
| videoAccess | integer | Доступность видео |
| videoStatus | integer | Статус видео |
| owner | integer FK | Внешний ключ, ссылающийся на таблицу Channels |

Описание таблицы Comments предоставлена в таблице 1.7

Таблица 1.7 – Описание таблицы Comments

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | integer PK | Уникальный идентификатор комментария |
| message | text | Содержание комментария |
| likes | jsonb | Список пользователей, поставивших лайки |
| created | timestamp | Дата создания комментария |
| owner | integer FK | Внешний ключ, ссылающийся на таблицу Users |
| video | integer FK | Внешний ключ, ссылающийся на таблицу Videos |

Таблица Comments связана сразу с двумя таблицами: Users и Videos. Это обеспечивает хранение информации о том, кто оставил комментарий и к какому видео он относится.

Описание таблицы Playlists предоставлена в таблице 1.8

Таблица 1.8 – Описание таблицы Playlists

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | integer PK | Уникальный идентификатор плейлиста |
| name | text | Название плейлиста |
| access | integer | Уровень доступа (публичный или приватный) |
| userId | integer FK | Внешний ключ, ссылающийся на таблицу Users |

Таблица Playlists предназначена для хранения плейлистов. Каждому плейлисту назначен владелец, что реализовано через связь с таблицей Users.

Описание таблицы PlaylistItems предоставлена в таблице 1.9

Таблица 1.9 – Описание таблицы PlaylistItems

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | Integer PK | Уникальный идентификатор элемента плейлиста |
| playlistId | integer FK | Внешний ключ, ссылающийся на таблицу Playlists |
| videoId | integer FK | Внешний ключ, ссылающийся на таблицу Videos |
| order | integer | Порядковый номер видео в плейлисте |

Таблица PlaylistItems реализует связь между плейлистами и видео. Это позволяет пользователям добавлять видео в определенном порядке, который фиксируется через поле order.

Описание таблицы Subscribes предоставлена в таблице 1.10.

Таблица 1.10 – Описание таблицы Subscribes

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | integer PK | Уникальный идентификатор подписки |
| user | integer FK | Внешний ключ, ссылающийся на таблицу Users |
| channel | integer FK | Внешний ключ, ссылающийся на таблицу Channels |

Таблица Subscribes хранит информацию о подписках пользователей на каналы. Она связывает таблицы Users и Channels, реализуя функционал подписок.

Описание таблицы Reports предоставлена в таблице 1.11.

Таблица 1.11 – Описание таблицы Reports

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ID | integer PK | Уникальный идентификатор жалобы |
| type | integer | Тип жалобы (нарушение, спам и т.д.) |
| message | text | Содержание жалобы |
| created | timestamp | Дата подачи жалобы |
| video | integer FK | Внешний ключ, ссылающийся на таблицу Videos |

Таблица Reports связана с таблицей Videos, что позволяет фиксировать жалобы на конкретные видео. Дополнительное поле type дает возможность классифицировать жалобы для их последующей обработки.

Для обеспечения целостности данных в базе реализованы связи между таблицами. Описание связей представлена в таблице 1.12.

Таблица 1.12 – Описание связей таблиц.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица-источник | Таблица-цель | Тип связи | Описание |
| Users | Channels | Один ко многим | Один пользователь может создать много каналов |
| Channels | Videos | Один ко многим | Один канал может содержать много видео |
| Users | Comments | Один ко многим | Один пользователь может оставлять много комментариев |
| Videos | Comments | Один ко многим | Одно видео может иметь много комментариев |
| Users | Playlists | Один ко многим | Один пользователь может создать много плейлистов |
| Playlists | PlaylistItems | Один ко многим | Один плейлист может содержать много видео |
| Users | Subscribes | Один ко многим | Один пользователь может подписываться на несколько каналов |

В данной главе подробно описана структура базы данных приложения BYTUBE. Приведены основные таблицы, их поля и типы данных, а также связи между таблицами. Данная структура обеспечивает надежное хранение данных и позволяет эффективно обрабатывать запросы, необходимые для реализации функционала видеохостинга.

# Вывод

Таким образом в рамках проектирования веб-приложения было выполнено:

* Разработана диаграмма вариантов использования, благодаря которой был определён какой функционал требуется приложению, а также какие роли будут иметь к им доступ.
* Спроектирована архитектура приложения. Было определено каким образам будут размещены компоненты приложения и какие протоколы будут использоваться.
* Спроектирована база данных приложения. Были определены таблицы, а также связи между ими.

# Реализация веб-приложения

# Реализация базы данных

Для обеспечения функциональности приложения BYTUBE требуется продуманная структура базы данных. В данном разделе представлена реализация базы данных, которая описывает необходимые таблицы и связи между ними. В процессе разработки используется подход Code First, предоставляемый [3] Entity Framework Core. Это позволяет разрабатывать структуру базы данных на основе моделей, написанных на C#, а миграции автоматически создают соответствующую схему в базе данных. Код контекста БД предоставлен в листинге 4.1.

public class PostgresDbContext : DbContext

{

public DbSet<User> Users { get; set; }

public DbSet<Channel> Channels { get; set; }

public DbSet<Comment> Comments { get; set; }

public DbSet<Video> Videos { get; set; }

public DbSet<Subscribe> Subscriptions { get; set; }

public DbSet<Report> Reports { get; set; }

public DbSet<Playlist> Playlists { get; set; }

public DbSet<PlaylistItem> PlaylistItems { get; set; }

public PostgresDbContext(DbContextOptions<PostgresDbContext> options) : base(options)

{

Database.EnsureCreated();

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<User>().HasIndex(u => u.Email).IsUnique();

base.OnModelCreating(modelBuilder);

}

}

Листинг 2.1 – Код контекста БД

Однако для демонстрации структуры и генерации базы данных вручную в приложении А приведен SQL-код, описывающий создание всех таблиц и их связей.

# Реализация серверной части

Для реализации серверной части приложения BYTUBE используется платформа ASP.NET Core. Эта платформа является кроссплатформенным и высокопроизводительным фреймворком для создания веб-приложений, который поддерживает разработку REST API. Серверное приложение запускается на встроенном веб-сервере Kestrel, который предоставляет поддержку HTTPS и интеграцию с Docker-контейнерами для удобного развертывания. Версия платформы: ASP.NET Core 8.0.

Для реализации серверной части используется набор библиотек, которые обеспечивают различные аспекты функциональности приложения. Основные используемые библиотеки:

* Entity Framework Core. Используется для взаимодействия с базой данных по принципу ORM (Object-Relational Mapping). Версия 8.0;
* JWT Bearer Authentication. Используется для реализации аутентификации на основе токенов JSON Web Token (JWT). Версия: 8.0;
* Xabe.FFmpeg. Используется изменения, чтения медиа файлов на сервере. Версия 5.2.6;
* Npgsql. Библиотека драйвера для PostgreSQL. Версия: 8.0.8.

В серверной части приложения маршруты (endpoints) определяются с помощью [4] REST API. В таблица с соответствием маршрутов, контроллеров и функций приложения предоставлен в приложении Б.

В процессе разработки серверной части был разработан весь необходимый функционал.

# 2.2.1 Просмотра публичных видео

Для просмотра публичных видео было разработана два метода в рамка контроллера «VideoController» это «Get» и «StreamVideo».

Метод «Get» возвращает полную информацию о видео по его идентификатору, включая метаданные, ссылки на файл видео и превью, а также данные о канале. Параметр id определяет идентификатор видео. Передается как параметр строки запроса. При успешном ответе возвращает объект VideoFullModel в формате JSON, содержащий данные видео (заголовок, описание, статус, метки, количество просмотров), а также данные о канале, к которому относится видео. Сообщение ответа предоставлена в листинге 2.2.

{

"id": 1,

"title": "Пример видео",

"description": "Описание видео",

"duration": "5:00",

"created": "2024-12-19T12:00:00",

"views": 100,

"tags": ["tutorial", "video"],

"videoUrl": "/data/videos/1/video.mp4",

"previewUrl": "/data/videos/1/preview.jpg",

"videoAccess": 1,

"videoStatus": 0,

"channel": {

"id": 1,

"name": "Канал пользователя",

"isSubscripted": false,

"subscribes": 10,

"iconUrl": "/data/channels/1/icon.jpg"

}

}

Листинг 2.2 – Ответ метода «Get»

Метод «StreamVideo» отвечает за потоковую передачу видеофайла. Поддерживает range-запросы для частичного скачивания видео. Параметр id определяет идентификатор видео. Возвращает потоковый файл видео в формате «video/mp4». Поддерживается заголовок Range для воспроизведения определенных частей файла.

Листинг методов предоставлен в приложении В.

# 2.2.2 Поиск публичных видео

Для поиска видео в рамках контроллера «VideoController» был реализован метод «Select». Метод реализует выборку видео из базы данных с поддержкой различных фильтров (по тегам, подпискам, избранным), сортировки (по дате создания, просмотрам, жалобам), а также пагинации (с помощью параметров Skip и Take). Метод принимает объект SelectOptions через строку запроса, который определяет условия поиска. Возвращает список объектов VideoModel в формате JSON. Каждый объект включает данные о видео (заголовок, продолжительность, статус, количество просмотров) и данные о канале (название, подписчики, иконка). Пример ответа предоставлен в листинге 2.3.

[

{

"id": 1,

"title": "Обучающее видео",

"duration": "10:00",

"created": "2024-12-19T12:00:00",

"videoAccess": 1,

"videoStatus": 0,

"views": 150,

"reportsCount": 0,

"previewUrl": "/data/videos/1/preview.jpg",

"channel": {

"id": 1,

"name": "Мой канал",

"isSubscripted": true,

"subscribes": 200,

"iconUrl": "/data/channels/1/icon.jpg"

}

}

]

Листинг 2.3 – Ответ метода «Select»

Листинг метода предоставлен в приложении В.

# 2.2.3 Регистрации, аутентифицироваться и авторизации

Для регистрации пользователя в рамка контроллера «AuthController» был разработан метод «Register». Для регистрации метод принимает имя пользователя, почту, пароль и файл иконки пользователя. Создается новый пользователь с указанным именем, электронной почтой и захешированным паролем. Пользователь сохраняется в базе данных. Если всё прошло хорошо возвращает пользователю код 200 OK.

Для аутентифицироваться и авторизации в рамках контроллера был разработан метод «signinJwt». Метод выполняет вход пользователя с использованием JWT (JSON Web Token). Принимает в качестве параметров почти и пароль, пример запроса предоставлен в листинге 2.4. Возвращает 200 OK при успешной авторизации.

{

"Email": "johndoe@example.com",

"Password": "SecurePassword123"

}

Листинг 2.4 – Данные запроса метода «signinJwt»

Листинг методов «Register» и «singinJwt» предоставлен в приложении В.

# 2.2.4 Загрузка, редактирование и удаление видео

Для загрузки видео в рамка контроллера «VideoController» был разработан метод «Post». Метод добавляет новое видео в указанный канал, который принадлежит авторизованному пользователю. Также выполняется сохранение видеофайла и создание метаданных. В запросе получает специальную модель для создания видео в формате JSON, которая предоставлена в листинге 2.5.

{

"title": "My First Video",

"description": "This is a description!",

"tags": ["fun", "vlog", "travel"],

"videoAccess": 1,

"videoStatus": 1,

"previewFile": "/file.png

"videoFile": "/file.mp4"

}

Листинг 2.5 – Данные запроса метода «Post»

Так же из строки запроса получает параметр channelId идентификатор канала, в который загружается видео.

Для изменения видео в рамка контроллера «VideoController» был разработан метод «Put». Метод обновляет данные существующего видео, принадлежащего указанному каналу пользователя. В запросе получает специальную модель для создания видео в формате JSON, которая предоставлена в листинге 2.6.

{

"title": "Updated Video Title",

"description": "This is the updated description for the video.",

"tags": ["updated", "education", "tutorial"],

"videoAccess": 0,

"previewFile": "/file.png

}

Листинг 2.6 – Данные запроса метода «Put»

Так же из строки запроса получает параметры channelId идентификатор канала, в который загружается видео и id идентификатор видео для редактирования.

Для удаления видео в рамка контроллера «VideoController» был разработан метод «Delete». Метод удаляет видео из указанного канала, принадлежащего текущему пользователю. Удаление может происходит только от имени автора видео либо администратора. Так же из строки запроса получает параметры channelId идентификатор канала, в который загружается видео и id идентификатор канала, в котором находится удаляемое видео.

Листинг Методов «Post», «Put», «Delete» контроллера «VideoController» предоставлены в приложении В.

# 2.2.5 Добавление, редактирование, удаление комментариев

Для добавления комментариев в рамках контроллера «CommentController» был разработан метод «Post». Этот метод позволяет авторизованному пользователю добавлять новый комментарий к видео. Он принимает данные из тела запроса в виде объекта CommentModel в формате JSON и сохраняет новый комментарий в базе данных. В случае успеха возвращается статус 200 OK. Пример запроса метода предоставлен в листинге 2.7.

{

"Message": "Очень интересное видео!",

"VideoId": 123

}

Листинг 2.7 – Данные запроса метода «Post»

Для редактирования комментариев в рамках контроллера «CommentController» был разработан метод «Put». Метод позволяет авторизованному пользователю редактировать свой комментарий. Также доступ предоставляется администратору или владельцу видео, к которому относится комментарий. Он принимает данные из тела запроса в виде объекта CommentModel в формате JSON, пример которое предоставлен в листинге 2.7. В случае успеха возвращается статус 200 OK.

Для удаления комментариев в рамках контроллера «CommentController» был разработан метод «Delete». Метод удаляет указанный комментарий. Авторизованный пользователь может удалить только свои комментарии. Дополнительно доступ имеет администратор или владелец видео. В строке запроса принимает параметр id идентификатор удаляемого комментария. В случае успеха возвращает статус 200 OK.

Листинг Методов «Post», «Put», «Delete» контроллера «CommentController» предоставлены в приложении В.

# 2.2.6 Создание, просмотр жалоб

Для простора жалоб видео в рамках контроллера «ReportController» был разработан метод «GetVideoReports». Метод возвращает список всех жалоб, связанных с указанным видео. Требуется авторизация. В троке запроса передаётся параметр vid идентификатор видео, для которого требуется получить жалобы. В ответ приходит массив из жалоб в формате JSON, пример ответа предоставлен в листинге 2.8

[

{

"id": 1,

"description": "Неприемлемый контент",

"type": 2,

"videoId": 42,

"created": "2024-12-21T12:34:56Z"

},

{

"id": 2,

"description": "Сцены насилия ",

"type": 1,

"videoId": 42,

"created": "2024-12-20T15:22:10Z"

}

]

Листинг 2.8 – Данные ответа метода «GetVideoReports»

Для добавления жалоб видео в рамках контроллера «ReportController» был разработан метод «Post». Метод позволяет создать жалобу на указанное видео. Требуется авторизация. В теле запроса передаются данные о жалобе в формате JSON, пример запроса предоставлен в листинге 2.9. Если запрос отработал без ошибок, то возвращается статут код 200 OK.

{

"description": "Контент нарушает авторские права.",

"type": 1,

"videoId": 42

}

Листинг 2.9 – Данные запроса метода «Post»

Листинг методов «GetVideoReports» и «Post» предоставлен в листинге В.

# 2.2.7 Создания или удаление плейлистов

Для создания плейлистов в рамках контроллера «PlaylistController» был разработан метод «Post». Метод добавляет новый плейлист для авторизованного пользователя. В теле запроса отправляются данные в формате JSON, пример которых предоставлен в листике 2.10. При успешной обработке возвращает код 200 OK.

{

"name": "Мои любимые видео",

"access": 1

}

Листинг 2.10 – Данные запроса метода «Post»

Для удаления плейлистов в рамках контроллера «PlaylistController» был разработан метод «Delete». Метод удаляет указанный плейлист пользователя, включая все его элементы. Требуется авторизация. В строке запроса принимает параметр id идентификатор плейлиста, который нужно удалить. При успешной обработке возвращает код 200 OK.

Листинг методов «Post» и «Delete» предоставлен в листинге В.

# 2.2.8 Подписка на канал

Для подписки пользователя на канал в рамках контроллера «ChannelController» был разработан метод «Subscribe». Метод подписывает текущего авторизованного пользователя на указанный канал. В строке запроса принимает параметр id идентификатор канала, на который нужно подписаться. При успешной обработке возвращает код 200 OK.

Листинг метода «Subscribe» предоставлен в листинге В.

# 2.2.9 Скрытия видео администратором

Для скрытия видео в рамках контроллера «VideoController» был разработан метод «BlockingByAdmin». Метод позволяет администратору заблокировать или разблокировать указанное видео. В строке запроса принимает параметр id идентификатор видео, статус которого нужно изменить. При успешной обработке возвращает код 200 OK.

Листинг метода «Subscribe» предоставлен в листинге В.

# 2.3 Реализация клиентской части

Клиентская часть приложения BYTUBE построена с использованием библиотеки React. Компоненты разбиты на модули в соответствии с функциональными зонами и задачами приложения. Структура представлена в виде набора компонентов, каждый из которых реализует определенную часть пользовательского интерфейса и логики приложения.

Описание основных компонентов:

* Компонент App. Основной компонент, отвечающий за базовую структуру приложения и маршрутизацию;
* Компонент AppHeader. Компонент, реализующий верхнюю навигационную панель. Содержит элементы управления, такие как поиск, логотип и ссылки на основные страницы (например, авторизация, главная страница);
* Компонент AuthPage. Страница, отвечающая за регистрацию и авторизацию пользователей;
* Компонент AdminPage. Специальная страница для администратора. Реализует управление контентом, жалобами;
* Компонент ChannelPage. Страница, отображающая информацию о канале;
* Компонент CommentsViewer. Компонент для отображения списка комментариев к видео;
* Компонент GeneralRoutes. Компонент для настройки маршрутов приложения. Определяет пути к страницам;
* Компонент Logo. Компонент, отображающий логотип приложения;
* Компонент SearchPage. Страница поиска видео;
* Компонент StudioPage. Страница для авторов контента, предоставляющая интерфейс для загрузки и управления видео;
* Компонент VideoMain. Главная страница приложения с лентой видео;
* Компонент VideoPage. Страница, для просмотра определенного видео;
* Компонент VideoPlayer. Компонент, реализующий воспроизведение видео;
* Компонент AddToPlaylistModal. Модальное окно для добавления видео в любой плейлист.

Таким образом, например в компоненте «App» через «GeneralRoutes» осуществляется переход на «VideoMain». На главной странице отображаются видео, которые загружаются в компонентах «VideoMain» и «VideoPlayer».

Такой модульный подход к реализации клиентской части позволяет легко поддерживать и расширять функциональность приложения BYTUBE. Каждый компонент отвечает за строго определенные задачи, что способствует удобству разработки и читаемости кода.

# 2.4 Вывод

Таким образом в рамках реализации веб-приложения было выполнено:

* Реализована база данных используя СУБД Postgres 15;
* Реализован весь основной функционал приложения такой как: просмотр публичных видео, поиск публичных видео, регистрации, аутентифицироваться и авторизации и тд.;

– Реализованы все необходимые компоненты на стороне клиента.

# 3 Тестирование веб-приложения

# 3.1 Функциональное тестирование

Для тестирования функционала были разработаны специальные тесты, которые предоставлены в таблице 3.1.

Таблица 3.1 – Функциональные тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название функции | Как её протестировать | Итог тестирования функции |
| 1 | Просмотр публичных видео | Зайти на главную страницу как гость. Убедиться, что отображаются доступные видео. Нажать на видео для просмотра. | Ошибок не обнаружено |
| 2 | Поиск публичных видео | Ввести запрос в строку поиска как гость. Нажать "Поиск". Убедиться, что результаты поиска соответствуют запросу. | Ошибок не обнаружено |
| 3 | Регистрация | На главной странице нажать "Регистрация". Ввести корректные данные. Нажать "Подтвердить". Убедиться, что аккаунт создан. | Ошибок не обнаружено |
| 4 | Аутентификация | На странице авторизации ввести e-mail и пароль ранее созданного пользователя. Нажать "Подтвердить". Убедиться, что осуществляется вход. | Ошибок не обнаружено |
| 5 | Авторизация | Проверить доступность функций авторизованного пользователя после входа (например, возможность создания плейлиста). | Ошибок не обнаружено |
| 6 | Загрузка видео | Авторизоваться. Перейти в "Студию". Нажать "Загрузить видео". Выбрать файл, ввести метаданные (название, описание, теги). Нажать "Подтвердить". Проверить, что видео появилось в списке. | Ошибок не обнаружено |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название функции | Как её протестировать | Итог тестирования функции |
| 7 | Редактирование своего видео | Перейти в "Студию". Выбрать видео. Нажать "Редактировать". Изменить название/описание/теги. Нажать "Сохранить". Убедиться, что изменения сохранены. | Ошибок не обнаружено |
| 8 | Удаление своего видео | Перейти в "Студию". Выбрать видео. Нажать "Редактирование". Нажать "Удалить". Убедиться, что видео удалено. | Ошибок не обнаружено |
| 9 | Оставление комментария | Авторизоваться. Зайти на страницу видео. Ввести текст комментария. Нажать "Отправить". Убедиться, что комментарий отображается. | Ошибок не обнаружено |
| 10 | Оставление жалобы | На странице видео нажать "Пожаловаться". Ввести причину. Нажать "Подтвердить". Убедиться, что нет ошибок. | Ошибок не обнаружено |
| 11 | Редактирование комментария на своих видео | Зайти на страницу своего видео. Выбрать комментарий. Нажать "Редактировать". Изменить текст. Нажать "Подтвердить". Убедиться, что комментарий обновлён. | Ошибок не обнаружено |
| 12 | Удаление комментария на своих видео | Зайти на страницу своего видео. Выбрать комментарий. Нажать "Удалить". Подтвердить действие. Убедиться, что комментарий удалён. | Ошибок не обнаружено |
| 13 | Подписка на канал | Зайти на страницу канала. Нажать "Подписаться". Убедиться, что статус подписки обновлён. Проверить отображение канала в списке подписок. | Ошибок не обнаружено |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название функции | Как её протестировать | Итог тестирования функции |
| 14 | Создание плейлиста | Зайти в "Плейлисты". Нажать "Создать плейлист". Ввести название. Указать доступ (публичный/приватный). Нажать "Подтвердить". Убедиться, что плейлист создан. | Ошибок не обнаружено |
| 15 | Удаление плейлиста | Зайти в "Плейлисты". Выбрать плейлист. Нажать "Удалить". Убедиться, что плейлист удалён. | Ошибок не обнаружено |
| 16 | Скрытие видео администратором | Зайти в админ-панель. Выбрать видео. Нажать "Скрыть". Убедиться, что видео становится недоступным для пользователей. | Ошибок не обнаружено |
| 17 | Удаление видео администратором | Зайти в админ-панель. Выбрать видео. Нажать "Удалить". Подтвердить действие. Убедиться, что видео удалено. | Ошибок не обнаружено |
| 18 | Просмотр пользовательских жалоб | Зайти в админ-панель. Выбрать видео. Перейти в раздел жалоб. Проверить отображение списка жалоб. | Ошибок не обнаружено |
| 19 | Удаление комментария администратором | Зайти в админ-панель. Выбрать видео. Перейти в список комментариев. Выбрать комментарий. Нажать "Удалить". Убедиться, что комментарий удалён. | Ошибок не обнаружено |
| 20 | Редактирование комментария администратором | Зайти в админ-панель. Выбрать видео. Перейти в список комментариев. Выбрать комментарий. Нажать "Редактировать". Изменить текст. Нажать "Подтвердить". Убедиться, что изменения сохранены. | Ошибок не обнаружено |

Таким образом был протестирован весь основной функционал веб-приложения.

# 3.2 Автоматизированное тестирование

Для автоматизированного теста были разработаны специальные тесты, которые предоставлены в таблице 3.2.

Таблица 3.2 – Автоматизированные тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название теста | Описание | Тест пройден |
| 1 | PasswordHasherTest0 | Тестирование класса PasswordHasher на корректные данные. | Да |
| 2 | PasswordHasherTest1 | Тестирование класса PasswordHasher на корректные данные. | Да |
| 3 | JwtManagerTest0 | Тестирование генерации исправного access токена. | Да |
| 4 | JwtManagerTest1 | Тестирование генерации исправного refresh токена. | Да |
| 5 | VideoMediaServiceTest | Тестирование работоспособности утилиты ffmpeg. | Да |

Таким образом был протестирован компоненты веб-приложения. Листинг тестов предоставлен в приложении Г.

# 3.3 Нагрузочное тестирования

Для проверки стабильности сервера были разработаны специальные нагрузочные тесты, которые предоставлены в таблице 3.3.

Таблица 3.3 – Нагрузочные тесты

|  |  |  |
| --- | --- | --- |
| № | Описание | Результат |
| 1 | Массовое воспроизведение видео. Проверить производительность системы при одновременном просмотре видео большим числом пользователей. | Никаких сбоев |
| 2 | Массовое добавление комментариев к видео. Проверить устойчивость при большом количестве комментариев. | Никаких сбоев |
| 3 | Одновременный просмотр и модерация администратором. Проверить производительность системы при одновременном использовании функций просмотра и модерации. | Никаких сбоев |

Таким образом веб-приложение было протестировано на стрессоустойчивость.

# 3.4 Вывод

В результате тестирования веб-приложение BYTUBE показало высокий уровень надежности, устойчивости и готовности к работе в реальных условиях эксплуатации. Выявленные дефекты были оперативно устранены, а результаты нагрузочного тестирования доказали способность системы справляться с большим числом пользователей и операций одновременно. Это обеспечивает уверенность в том, что приложение удовлетворит ожидания пользователей, обеспечивая стабильность и удобство работы.

# 4 Руководство программиста

Для развёртывания web-приложения применяется инструмент Docker Compose. Перед развёртыванием web-приложения необходимо убедиться, что в системе установлены Docker Engine и Docker Compose при помощи команд docker version и docker compose version. В случае, если любая из указанных технологий не установлена, её необходимо установить согласно подходящей инструкции на официальном сайте, например, для [5] Docker Engine и для [6] Docker Compose.

Требуется создать папку, где будет храниться исходный код приложения, и поместить туда исходных код. Так же требуется создать папку «Data», если папка уже существует, то очистить ее содержимое. После чего создать файлы «docker-compose.yml» и «Dockerfile» содержимое которых предоставлено в приложении Д. После чего открыть корневую папку приложения в терминале и ввести команду «docker-compose up --build». В результате будут созданы два связанных контейнера и веб-приложение будет работать.

Для корректной работы приложения в корневой папке необходимо:

* разместить подписанный сертификат и назвать его «BYTUBE.pfx»;
* создать и настроить файл «appsettings.json» в котором должны находиться соль для хеширования пароля, а так же описание access и refresh токенов;
* В папке «Data» создать папки «users», «channels», «videos». Внутри этих папок создать по папке «template» в которых будут находиться шаблоны для данных пользователей, каналов и видео;
* Заполнить папки «template». В папке «template» пользователя должен быть файл «icon.png», для каналов там файлы «icon.png» и «banner.png», для видео это файл «preview.png».

Таким образом происходить настройка и запуск веб-приложения в Docker.

# Заключение

При выполнении курсового проекта было создано приложение поиска и предложения работы. Сервер был создан при помощи платформы ASP.Core, языка программирования C#. Web-приложение было реализовано при помощи библиотеки React. База данных была реализована в СУБД PostgreSQL. Были реализованы все функциональные требования, а именно:

обеспечивать всем пользователям возможности:

* просмотра публичных видео;
* поиск публичных видео;

обеспечивать гостям возможности:

* регистрации, аутентифицироваться и авторизации;

обеспечивать клиентам возможности:

* загрузки, редактирования и удаление своего видео;
* оставлять комментарии;
* оставлять жалобу;
* удаление или редактирование комментариев своих видео;
* подписки на канал;
* создания или удаление плейлистов;

обеспечивать администраторам возможности:

* скрытия и удаления видео;
* просмотр жалоб пользователей;
* обеспечивать удаление или редактирование комментариев.

Было разработано 8 таблиц базы данных которые позволяют хранить всю необходимую информацию.

По итогам написания кода для веб-приложения получилось около 10000 строк кода, из которых 7000 клиентская часть и 3000 серверная часть.

Также были реализована поддержка протокола HTTPS. Приложение было протестировано на наличие ошибок с использованием ручного и автоматического тестирования. Для ручного тестирования использовался браузер и Postman. Для автоматического тестирования использовалось вспомогательная библиотека xUnit.

Данное приложение готово к эксплуатации и использованию в сети Интернет.

# Список используемых источников

1. React Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://react.dev/learn> – Дата доступа: 22.12.2024;
2. Webpack Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://webpack.js.org/guides](https://webpack.js.org/guides%20) – Дата доступа: 22.12.2024;
3. Metanit Entity Framework Core [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://metanit.com/sharp/efcore/1.2.php](https://metanit.com/sharp/efcore/1.2.php%20) – Дата доступа: 22.12.2024;
4. Habr руководство по REST API [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habr.com/ru/articles/590679/> – Дата доступа: 22.12.2024;
5. Docker Engine Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://docs.docker.com/engine/](https://docs.docker.com/engine/%20) – Дата доступа: 22.12.2024;
6. Docker Compose Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.docker.com/compose/> – Дата доступа: 22.12.2024;

# Приложение А

В данном приложении предоставлен код создания базы данных.

-- Создание таблицы Users

CREATE TABLE Users (

ID SERIAL PRIMARY KEY,

name TEXT NOT NULL,

email TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

role INTEGER NOT NULL,

token TEXT,

LikedVideo JSONB

);

-- Создание таблицы Channels

CREATE TABLE Channels (

ID SERIAL PRIMARY KEY,

name TEXT NOT NULL,

description TEXT,

created TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

userId INTEGER NOT NULL,

CONSTRAINT FK\_Channels\_Users FOREIGN KEY (userId) REFERENCES Users (ID) ON DELETE CASCADE

);

-- Создание таблицы Videos

CREATE TABLE Videos (

ID SERIAL PRIMARY KEY,

title TEXT NOT NULL,

description TEXT,

views INTEGER DEFAULT 0,

duration TEXT NOT NULL,

tags JSONB,

likes JSONB,

dislikes JSONB,

created TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

videoAccess INTEGER NOT NULL,

videoStatus INTEGER NOT NULL,

owner INTEGER NOT NULL,

CONSTRAINT FK\_Videos\_Channels FOREIGN KEY (owner) REFERENCES Channels (ID) ON DELETE CASCADE

);

-- Создание таблицы Comments

CREATE TABLE Comments (

ID SERIAL PRIMARY KEY,

message TEXT NOT NULL,

likes JSONB,

created TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

owner INTEGER NOT NULL,

video INTEGER NOT NULL,

CONSTRAINT FK\_Comments\_Users FOREIGN KEY (owner) REFERENCES Users (ID) ON DELETE CASCADE,

CONSTRAINT FK\_Comments\_Videos FOREIGN KEY (video) REFERENCES Videos (ID) ON DELETE CASCADE

);

-- Создание таблицы Playlists

CREATE TABLE Playlists (

ID SERIAL PRIMARY KEY,

name TEXT NOT NULL,

access INTEGER NOT NULL,

userId INTEGER NOT NULL,

CONSTRAINT FK\_Playlists\_Users FOREIGN KEY (userId) REFERENCES Users (ID) ON DELETE CASCADE

);

-- Создание таблицы PlaylistItems

CREATE TABLE PlaylistItems (

ID SERIAL PRIMARY KEY,

playlistId INTEGER NOT NULL,

videoId INTEGER NOT NULL,

"order" INTEGER NOT NULL,

CONSTRAINT FK\_PlaylistItems\_Playlists FOREIGN KEY (playlistId) REFERENCES Playlists (ID) ON DELETE CASCADE,

CONSTRAINT FK\_PlaylistItems\_Videos FOREIGN KEY (videoId) REFERENCES Videos (ID) ON DELETE CASCADE

);

-- Создание таблицы Subscribes

CREATE TABLE Subscribes (

ID SERIAL PRIMARY KEY,

"user" INTEGER NOT NULL,

channel INTEGER NOT NULL,

CONSTRAINT FK\_Subscribes\_Users FOREIGN KEY ("user") REFERENCES Users (ID) ON DELETE CASCADE,

CONSTRAINT FK\_Subscribes\_Channels FOREIGN KEY (channel) REFERENCES Channels (ID) ON DELETE CASCADE

);

-- Создание таблицы Reports

CREATE TABLE Reports (

ID SERIAL PRIMARY KEY,

type INTEGER NOT NULL,

message TEXT,

created TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

video INTEGER NOT NULL,

CONSTRAINT FK\_Reports\_Videos FOREIGN KEY (video) REFERENCES Videos (ID) ON DELETE CASCADE

);

# Приложение Б

Таблица соответствия маршрутов, контроллеров и функций приложения

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP-метод | Маршрут | Контроллер | Функция |
| GET | /App/\* | AppController | GetView |
| POST | /auth/signin | AuthController | signinJwt |
| GET | /auth/signout | AuthController | logoutJwt |
| POST | /auth/register | AuthController | Register |
| GET | /api/channel | ChannelController | Get |
| GET | /api/channel/check | ChannelController | CheckChannel |
| POST | /api/channel | ChannelController | Post |
| PUT | /api/channel | ChannelController | Put |
| DELETE | /api/channel | ChannelController | Delete |
| GET | /api/channel/user | ChannelController | GetUserChannels |
| POST | /api/channel/subscribe | ChannelController | Subscribe |
| DELETE | /api/channel/subscribe | ChannelController | Unsubscribe |
| GET | /api/comment | CommentController | Get |
| GET | /api/comment/video | CommentController | GetVideoComments |
| POST | /api/comment | CommentController | Post |
| POST | /api/comment/like | CommentController | LikeComment |
| PUT | /api/comment | CommentController | Put |
| DELETE | /api/comment | CommentController | Delete |
| GET | /api/playlist | PlaylistController | Get |
| GET | /api/playlist/user | PlaylistController | GetByUser |
| POST | /api/playlist | PlaylistController | Post |
| POST | /api/playlist/add | PlaylistController | AddVideoToPlaylist |
| DELETE | /api/playlist/remove | PlaylistController | RemoveVideoFromPlaylist |
| PUT | /api/playlist | PlaylistController | Put |
| DELETE | /api/playlist | PlaylistController | Delete |
| GET | /api/report | ReportController | Get |
| POST | /api/report | ReportController | Post |
| DELETE | /api/report | ReportController | Delete |
| GET | /api/report/video | ReportController | GetVideoReports |
| GET | /api/user/geticon | UserController | GetUserIcon |
| GET | /api/user/auth | UserController | Auth |
| GET | /api/user/channelslist | UserController | GetChannelsList |

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP-метод | Маршрут | Контроллер | Функция |
| GET | /api/video | VideoController | Get |
| GET | /api/video/channel | VideoController | GetChannelVideos |
| GET | /api/video/playlist | VideoController | GetPlaylistVideos |
| GET | /api/video/select | VideoController | Select |
| POST | /api/video | VideoController | Post |
| PUT | /api/video | VideoController | Put |
| DELETE | /api/video | VideoController | Delete |
| GET | /data/videos/{id}/video.mp4 | VideoController | StreamVideo |
| GET | /api/video/mark | VideoController | GetMark |
| POST | /api/video/like | VideoController | LikeVideo |
| POST | /api/video/dislike | VideoController | DislikeVideo |
| POST | /api/video/view | VideoController | AddView |
| DELETE | /api/video/delete | VideoController | DeleteByAdmin |
| PUT | /api/video/block | VideoController | BlockingByAdmin |

# Приложение В

Метод «Get» контроллера VideoController

[HttpGet]

public async Task<IResult> Get([FromQuery] int id)

{

try

{

Video? video = await \_db.Videos.FirstOrDefaultAsync(i => i.Id == id);

if (video == null)

throw new ServerException("Видео не найдено", 404);

Channel channel = await \_db.Channels.Include(i => i.Subscribes).FirstAsync(i => i.Id == video.OwnerId);

if (video.VideoAccess == Video.Access.Private)

{

if (!IsAutorize || channel.UserId != UserId)

throw new ServerException("Видео вам не доступно", 403);

}

if (video.VideoStatus == Video.Status.Blocked)

throw new ServerException("Видео более не доспутно", 403);

var videoLocalData = \_localDataManager.GetVideoData(id);

var channelLocalData = \_localDataManager.GetChannelData(channel.Id);

VideoFullModel model = new VideoFullModel()

{

Id = video.Id,

Title = video.Title,

Description = video.Description ?? "",

Duration = video.Duration,

Created = video.Created,

Views = video.Views,

Tags = video.Tags,

VideoUrl = $"/data/videos/{id}/video.{videoLocalData.VideoExtention}",

PreviewUrl = $"/data/videos/{id}/preview.{videoLocalData.PreviewExtention}",

VideoAccess = video.VideoAccess,

VideoStatus = video.VideoStatus,

Channel = new ChannelModel()

{

Id = channel.Id,

Name = channel.Name,

IsSubscripted = false,

Subscribes = channel.Subscribes.Count,

IconUrl = $"/data/channels/{channel.Id}/icon.{channelLocalData.IconExtention}"

}

};

if (IsAutorize)

{

model.Channel.IsSubscripted = channel.Subscribes.Any(i => i.UserId == UserId);

}

return Results.Json(model);

}

catch (ServerException srverr)

{

return Results.Json(srverr.GetModel(), statusCode: srverr.Code);

}

catch

{

return Results.Problem(statusCode: 400);

}

}

Метод «StreamVideo» контроллера «VideoController»

[HttpGet("/data/videos/{id:int}/video.mp4")]

public async Task<IResult> StreamVideo([FromRoute] int id)

{

try

{

string path = $"./Data/videos/{id}/video.mp4";

if (!System.IO.File.Exists(path))

throw new ServerException("Файла больше не существует!", 404);

Video video = await \_db.Videos.Include(i => i.Owner).FirstAsync(i => i.Id == id);

if (video.VideoAccess == Video.Access.Private)

{

if ((IsAutorize && UserId != video.Owner.UserId) || !IsAutorize)

throw new ServerException("Видео файл не доступен!", 403);

}

if (video.VideoStatus == Video.Status.Blocked)

throw new ServerException("Видео файл не доступен!", 403);

HttpContext.Response.Headers.Append("Accept-Ranges", "bytes");

var fileStream = new FileStream(path, FileMode.Open, FileAccess.Read, FileShare.Read | FileShare.Delete);

var fileLength = fileStream.Length;

if (Request.Headers.ContainsKey("Range"))

{

var rangeHeader = Request.Headers["Range"].ToString();

var range = rangeHeader.Replace("bytes=", "").Split('-');

var start = long.Parse(range[0]);

var end = range.Length > 1 && !string.IsNullOrEmpty(range[1])

? long.Parse(range[1])

: fileLength - 1;

var chunkSize = end - start + 1;

fileStream.Seek(start, SeekOrigin.Begin);

return Results.File(

fileStream,

"video/mp4",

enableRangeProcessing: true);

}

return Results.File(fileStream, "video/mp4");

}

catch (ServerException srvErr)

{

return Results.Json(srvErr.GetModel(), statusCode: srvErr.Code);

}

catch (Exception err)

{

return Results.Problem(err.Message, statusCode: 400);

}

}

Метод «Select» контроллера «VideoController»

[HttpGet("select")]

public async Task<IResult> Select([FromQuery] SelectOptions options)

{

try

{

var tagPattern = @"#\w+";

var tags = Regex.Matches(options.SearchPattern, tagPattern)

.Cast<Match>()

.Select(m => m.Value)

.ToList();

var query = \_db.Videos

.Include(video => video.Owner)

.Include(video => video.Owner!.Subscribes)

.Include(video => video.Reports)

.AsQueryable();

if (!(options.AsAdmin && IsAutorize && Role == Entity.Models.User.RoleType.Admin))

{

query = query.Where(video => video.VideoAccess == Video.Access.All);

query = query.Where(video => video.VideoStatus == Video.Status.NoLimit);

}

if (!string.IsNullOrEmpty(options.Ignore))

{

var ignoredIds = options.Ignore.Split(',').Select(int.Parse);

query = query.Where(video => !ignoredIds.Contains(video.Id));

}

if (!string.IsNullOrEmpty(options.SearchPattern))

{

query = query.Where(video => Regex.IsMatch(video.Title, $@"(\w)\*{options.SearchPattern}(\w)\*"));

}

if (IsAutorize)

{

var user = await \_db.Users.FirstAsync(i => i.Id == UserId);

if (options.Subscribes)

{

query = query.Where(video => video.Owner!.Subscribes.Any(sub => sub.UserId == UserId));

}

if (options.Favorite)

{

var likedVideoIds = user.LikedVideo;

query = query.Where(video => likedVideoIds.Contains(video.Id));

}

}

query = options.OrderBy switch

{

SelectOrderBy.Creation => query.OrderBy(video => video.Created),

SelectOrderBy.CreationDesc => query.OrderByDescending(video => video.Created),

SelectOrderBy.Reports => query.OrderBy(video => video.Reports.Count),

SelectOrderBy.ReportsDesc => query.OrderByDescending(video => video.Reports.Count),

SelectOrderBy.Views => query.OrderByDescending(video => video.Views),

\_ => query

};

query = query.Skip(options.Skip).Take(options.Take);

var videos = await query.ToListAsync();

var result = videos.Select(video =>

{

var videoData = \_localDataManager.GetVideoData(video.Id);

var channelData = \_localDataManager.GetChannelData(video.Owner!.Id);

return new VideoModel

{

Id = video.Id,

Title = video.Title,

Duration = video.Duration,

Created = video.Created,

VideoAccess = video.VideoAccess,

VideoStatus = video.VideoStatus,

Views = video.Views,

ReportsCount = video.Reports.Count,

PreviewUrl = $"/data/videos/{video.Id}/preview.{videoData.PreviewExtention}",

Channel = new ChannelModel

{

Id = video.Owner.Id,

Name = video.Owner.Name,

IsSubscripted = false,

Subscribes = video.Owner.Subscribes.Count,

IconUrl = $"/data/channels/{video.Owner.Id}/icon.{channelData.IconExtention}"

}

};

});

Метод «Register» контроллера «AuthController»

[HttpPost("register")]

public async Task<IResult> Register([FromForm] RegisterModel model)

{

try

{

var usr = await \_db.Users.AddAsync(new()

{

Name = model.UserName,

Email = model.Email,

Password = \_passwordHasher.Hash(model.Password)

});

await \_db.SaveChangesAsync();

await \_localDataManager.SaveUserFiles(usr.Entity.Id, model.ImageFile);

}

catch

{

var errorModel = new ServerErrorModel(400);

errorModel.errors.Add("email", ["Пользователь с такой почтой уже сущетсвует"]);

return Results.Json(errorModel, statusCode: 400);

}

return Results.Ok();

}

Метод «signinJwt» контроллера «AuthController»

[HttpPost("signin")]

public async Task<IResult> signinJwt([FromBody] SigninModel model)

{

try

{

var user = await \_db.Users.FirstAsync(i => i.Email == model.Email);

if (!\_passwordHasher.Verify(model.Password, user.Password))

{

throw new ServerException("Пароли не совпадают");

}

HttpContext.Response.Cookies.Append(

"AccessToken",

JwtManager.GenerateJwtToken(\_jwtManager.AccessToken, user),

\_jwtManager.JwtCookieOptions

);

string token = JwtManager.GenerateJwtToken(\_jwtManager.RefreshToken, user);

HttpContext.Response.Cookies.Append(

"RefreshToken",

token,

\_jwtManager.JwtCookieOptions

);

user.Token = token;

\_db.Users.Update(user);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException apperr)

{

return Results.Json(apperr.GetModel(), statusCode: apperr.Code);

}

catch (Exception err)

{

return Results.Problem(err.Message, statusCode: 400);

}

}

Метод «Post» контроллера «VideoController»

[HttpPost, Authorize]

public async Task<IResult> Post([FromForm] CreateVideoModel model, [FromQuery] int channelId)

{

try

{

if (!await \_db.Channels.AnyAsync(c => c.Id == channelId && c.UserId == UserId))

throw new ServerException("Канал вам не пренадлежит", 403);

var video = await \_db.Videos.AddAsync(new Video()

{

Title = model.Title,

Description = model.Description,

Created = DateTime.Now.ToUniversalTime(),

Dislikes = [],

Likes = [],

Views = 0,

Tags = model.Tags,

Duration = "00:00",

OwnerId = channelId,

VideoAccess = model.VideoAccess,

VideoStatus = model.VideoStatus,

});

await \_db.SaveChangesAsync();

await \_localDataManager.SaveVideoFiles(video.Entity.Id, model.PreviewFile!, model.VideoFile!);

var videoInfo = await \_videoMediaService

.GetMediaInfo($"{LocalDataManager.VideosPath}/{video.Entity.Id}/video.{\_localDataManager

.GetVideoData(video.Entity.Id).VideoExtention}");

int minutes = (int)Math.Floor(videoInfo.Duration.TotalSeconds / 60);

int secound = (int)videoInfo.Duration.TotalSeconds - (minutes \* 60);

video.Entity.Duration = $"{minutes}:{secound}";

\_db.Videos.Update(video.Entity);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException srvErr)

{

return Results.Json(srvErr.GetModel(), statusCode: srvErr.Code);

}

Метод «Put» контроллера «VideoController»

[HttpPut, Authorize]

public async Task<IResult> Put([FromForm] EditVideoModel model, [FromQuery] int id, [FromQuery] int channelId)

{

try

{

if (!await \_db.Channels.AnyAsync(c => c.Id == channelId && c.UserId == UserId))

throw new ServerException("Канал вам не пренадлежит", 403);

Video? video = await \_db.Videos.FirstOrDefaultAsync(v => v.Id == id);

if (video == null)

throw new ServerException("Видео не существует", 404);

if (video.OwnerId != channelId)

throw new ServerException("Видео вам не пренадлежит", 403);

video.Title = model.Title;

video.Description = model.Description;

video.Tags = model.Tags;

video.VideoAccess = model.VideoAccess;

if (model.PreviewFile != null)

await \_localDataManager.SaveVideoFiles(video.Id, model.PreviewFile!);

\_db.Videos.Update(video);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException srvErr)

{

return Results.Json(srvErr.GetModel(), statusCode: srvErr.Code);

}

catch

{

return Results.Problem("Error", statusCode: 400);

}

}

Метод «Delete» контроллера «VideoController»

[HttpDelete, Authorize]

public async Task<IResult> Delete([FromQuery] int id, [FromQuery] int channelId)

{

try

{

if (!await \_db.Channels.AnyAsync(c => c.Id == channelId && c.UserId == UserId))

throw new ServerException("Канал вам не пренадлежит", 403);

Video? video = await \_db.Videos.FirstOrDefaultAsync(v => v.Id == id);

if (video == null)

throw new ServerException("Видео не существует", 404);

if (video.OwnerId != channelId)

throw new ServerException("Видео вам не пренадлежит", 403);

Directory.Delete($"{LocalDataManager.VideosPath}/{id}", true);

\_db.Videos.Remove(video);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException srvErr)

{

return Results.Json(srvErr.GetModel(), statusCode: srvErr.Code);

}

catch

{

return Results.Problem("Error", statusCode: 400);

}

}

Метод «Post» контроллера «CommentController»

[HttpPost, Authorize]

public async Task<IResult> Post([FromBody] CommentModel model)

{

try

{

\_db.Comments.Add(new Comment()

{

Message = model.Message,

VideoId = model.VideoId,

UserId = UserId,

Likes = [],

Created = DateTime.Now.ToUniversalTime(),

});

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод «Put» контроллера «CommentController»

[HttpPut, Authorize]

public async Task<IResult> Put([FromBody] CommentModel model, [FromQuery] int id)

{

try

{

Comment? comment = await \_db.Comments

.Include(i => i.Video)

.Include(i => i.Video!.Owner)

.FirstOrDefaultAsync(c => c.Id == id);

if (comment == null)

throw new ServerException("Комментарий не найден!", 404);

if (comment.UserId != UserId &&

Role != Entity.Models.User.RoleType.Admin &&

comment.Video!.Owner!.UserId == UserId)

throw new ServerException("Комментарий вам не пренадлежит", 403);

comment.Message = model.Message;

\_db.Comments.Update(comment);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод «Delete» контроллера «CommentController»

[HttpDelete, Authorize]

public async Task<IResult> Delete([FromQuery] int id)

{

try

{

Comment? comment = await \_db.Comments

.Include(i => i.Video)

.Include(i => i.Video!.Owner)

.FirstOrDefaultAsync(c => c.Id == id);

if (comment == null)

throw new ServerException("Комментарий не найден!", 404);

if (comment.UserId != UserId

&& Role != Entity.Models.User.RoleType.Admin &&

comment.Video!.Owner!.UserId == UserId)

throw new ServerException("Комментарий вам не пренадлежит", 403);

\_db.Comments.Remove(comment);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод «Post» контроллера «ReportController»

[HttpPost, Authorize]

public async Task<IResult> Post([FromBody] ReportModel model)

{

try

{

\_db.Reports.Add(new Report()

{

VideoId = model.VideoId,

Description = model.Description,

Type = model.Type,

});

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод «GetVideoReports» контроллера «ReportController»

[HttpGet("video"), Authorize]

public async Task<IResult> GetVideoReports([FromQuery] int vid)

{

try

{

var reports = await \_db.Reports.Where(r => r.VideoId == vid).ToArrayAsync();

return Results.Json(reports.Select(report => new ReportModel()

{

Id = report.Id,

Description = report.Description,

Type = report.Type,

VideoId = report.VideoId,

Created = report.Created,

}));

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод «Post» контроллера «PlaylistController»

[HttpPost, Authorize]

public async Task<IResult> Post([FromBody] PlaylistModel model)

{

try

{

\_db.Playlists.Add(new Playlist()

{

Name = model.Name,

Access = model.Access,

UserId = UserId,

});

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод «Delete» контроллера «PlaylistController »

[HttpDelete, Authorize]

public async Task<IResult> Delete([FromQuery] int id)

{

try

{

Playlist? playlist = await \_db.Playlists.Include(pl => pl.PlaylistItems).FirstOrDefaultAsync(c => c.Id == id);

if (playlist == null)

throw new ServerException("Плейлист не найден!", 404);

if (playlist.UserId != UserId)

throw new ServerException("Плейлист вам не пренадлежит", 403);

\_db.Playlists.Remove(playlist);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод «Subscribe» контроллера «ChannelController»

[HttpPost("subscribe"), Authorize]

public async Task<IResult> Subscribe([FromQuery] int id)

{

try

{

Channel? channel = await \_db.Channels.FirstOrDefaultAsync(c => c.Id == id);

if (channel == null)

throw new ServerException("Канал не найден", 404);

if (await \_db.Subscriptions.AnyAsync(i => i.UserId == UserId && i.ChannelId == id))

throw new ServerException("Уже подписан");

await \_db.Subscriptions.AddAsync(new Subscribe()

{

UserId = UserId,

ChannelId = channel.Id,

});

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

}

Метод « BlockingByAdmin» контроллера «VideoController»

[HttpPut("block"), Authorize]

public async Task<IResult> BlockingByAdmin([FromQuery] int id)

{

try

{

if (Role != Entity.Models.User.RoleType.Admin)

throw new ServerException("Вы не администратор!", 403);

var video = await \_db.Videos.FirstOrDefaultAsync(x => x.Id == id);

if (video == null)

throw new ServerException("Видео не найдена!", 404);

if (video.VideoStatus == Video.Status.NoLimit)

video.VideoStatus = Video.Status.Blocked;

else

video.VideoStatus = Video.Status.NoLimit;

\_db.Videos.Update(video);

await \_db.SaveChangesAsync();

return Results.Ok();

}

catch (ServerException err)

{

return Results.Json(err.GetModel(), statusCode: err.Code);

}

catch (Exception err)

{

return Results.Json(err.Message, statusCode: 500);

}

}

# Приложение Г

public class Tests

{

private readonly JwtManager jwtManager = new JwtManager(new JwtSettings()

{

Audience = "http://localhost:8081/api",

Issuer = "http://localhost:8081",

SecretKey = "N0ek5pYWMgnq-iaCqz811YNsxNFkhTb8oNEFooIyHBg",

ExpiryMinutes = 1,

},

new JwtSettings()

{

Audience = "http://localhost:8081/api",

Issuer = "http://localhost:8081",

SecretKey = "N0ek5pYWMgnq-iaCqz811YNsxNFkhTb8oNEFooIyHBf",

ExpiryMinutes = 2,

});

[Fact]

public void PasswordHasherTest0()

{

PasswordHasher hasher = new PasswordHasher("salt");

string passwordHashed = hasher.Hash("pravoda01");

Assert.True(hasher.Hash("pravoda01") == passwordHashed);

}

[Fact]

public void PasswordHasherTest1()

{

PasswordHasher hasher = new PasswordHasher("salt");

string passwordHashed = hasher.Hash("pravoda01");

Assert.False(hasher.Hash("Pravoda01") == passwordHashed);

}

[Fact]

public void JwtManagerTest0()

{

string token = JwtManager.GenerateJwtToken(jwtManager.AccessToken, new User()

{

Id = 12,

Name = "Test",

Role = User.RoleType.User

});

ClaimsPrincipal claims = JwtManager.ValidateToken(token, JwtManager.GetParameters(jwtManager.AccessToken));

Assert.Equal("12", claims.Claims.First().Value);

}

[Fact]

public void JwtManagerTest1()

{

string token = JwtManager.GenerateJwtToken(jwtManager.RefreshToken, new User()

{

Id = 13,

Name = "Testsad",

Role = User.RoleType.User

});

ClaimsPrincipal claims = JwtManager.ValidateToken(token, JwtManager.GetParameters(jwtManager.RefreshToken));

Assert.Equal("13", claims.Claims.First().Value);

}

[Fact]

public void VideoMediaServiceTest()

{

VideoMediaService videoMedia = new VideoMediaService("C:\\ffmpeg");

var data = videoMedia.GetMediaInfo("D:\\BYData\\GAMBLECORE.mp4");

Assert.NotNull(data);

}

}

# Приложение Д

Содержимое файла «docker-compose.yml»

version: '3.8'

services:

server:

build:

context: .

dockerfile: Dockerfile

ports:

- "8080:8080"

- "8081:8081"

environment:

- ASPNETCORE\_ENVIRONMENT=Development

- ASPNETCORE\_URLS=https://+:8081;http://+:8080

- FFMpegPath=/usr/bin/

- ConnectionStrings\_\_DefaultConnection=Host=database;Port=5432;Database=BYTUBE;Username=admin;Password=root

depends\_on:

- database

labels:

- "traefik.enable=true"

- "traefik.http.routers.webapp-http.rule=Host(`localhost`)"

- "traefik.http.services.webapp.loadbalancer.server.port=8080"

- "traefik.http.routers.webapp-https.rule=Host(`localhost`)"

- "traefik.http.routers.webapp-https.tls=true"

- "traefik.http.services.webapp-https.loadbalancer.server.port=8081"

volumes:

- ./Data:/app/Data

database:

image: postgres:15

environment:

POSTGRES\_DB: BYTUBE

POSTGRES\_USER: admin

POSTGRES\_PASSWORD: root

ports:

- "5432:5432"

volumes:

- postgres\_data:/var/lib/postgresql/data

volumes:

postgres\_data:

Содержимое файла «Dockerfile»

умолчанию для конфигурации отладки)

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base

USER app

WORKDIR /app

EXPOSE 8080

EXPOSE 8081

USER root

RUN apt-get update && apt-get install -y ffmpeg && rm -rf /var/lib/apt/lists/\*

USER app

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build

ARG BUILD\_CONFIGURATION=Release

WORKDIR /src

COPY ["BYTUBE.csproj", "."]

RUN dotnet restore "./BYTUBE.csproj"

COPY . .

WORKDIR "/src/."

RUN dotnet build "./BYTUBE.csproj" -c $BUILD\_CONFIGURATION -o /app/build

RUN mkdir -p /app/Data && chown -R app:app /app/Data

FROM build AS publish

ARG BUILD\_CONFIGURATION=Release

RUN dotnet publish "./BYTUBE.csproj" -c $BUILD\_CONFIGURATION -o /app/publish /p:UseAppHost=false

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

COPY ./BYTUBE.pfx /https/BYTUBE.pfx

# Настройка переменных среды для использования сертификата

ENV ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Path=/https/BYTUBE.pfx

ENV ASPNETCORE\_Kestrel\_\_Certificates\_\_Default\_\_Password=BYTYBE

ENTRYPOINT ["dotnet", "BYTUBE.dll"]