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:

Веб-приложение видеохостинг «BYTUBE»

Выполнил студент Окулич Дмитрий Юрьевич
(Ф.И.О.)

Руководитель проекта преп.-ст. 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

Веб-приложение – это клиент-серверное приложение, в котором клиент и сервер взаимодействуют по протоколу HTTP.

Видеохостинг – это веб-приложение, позволяющий загружать и просматривать видео в браузере.

Цель данной работы заключается в создании платформы для самовыражения пользователей. Веб-приложения обеспечит 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 8.0, 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 этапы работы:

– постановка задачи и обзор аналогичных решений (раздел 1);

– проектирование веб-приложения (раздел 2);

– реализация веб-приложения (раздел 3);

– тестирования веб-приложения (раздел 4);

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

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.

Приложение будет доступно в сети интернет и будет работать через браузер.

1 Постановка задачи и обзор аналогичных решений

1.1 Постановка задачи

При помощи веб-приложения «BYTUBE» пользователь сможет просматривать видео контент. Пользователю будет дана возможность самому создать канал и загрузить, а потом редактировать свои видео. Так же пользователи смогут подписываться на каналы тех чей контент им понравился. Пользователя смогут оставлять комментарии под видео. Для того что бы сохранить видео, которые им понравились пользователи смогут создавать и редактировать плейлисты.

1.2 Обзор аналогичных решений

1.2.1 Веб-приложение «YouTube»

YouTube [1] обладает широким спектром функций, включая загрузку и просмотр видео, создание плейлистов, подписку на каналы, комментирование и оценку видео, а также возможность проведения прямых трансляций. Этот видеохостинг также позволяет пользователям создавать и управлять своими каналами, а также монетизировать контент через рекламу и спонсорство.

YouTube имеет простой и удобный интерфейс, который легко использовать. Дизайн этого видеохостинга состоит из основных функций в верхней части экрана и списка рекомендованных видео в центральной части. YouTube также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид сайта.

Дизайн YouTube – минималистичный и простой в использовании. Цветовая схема состоит в основном из белого и красного цветов, что создает ощущение легкости и чистоты для пользователя.

Интерфейс приложения представлен на рисунке 1.1.

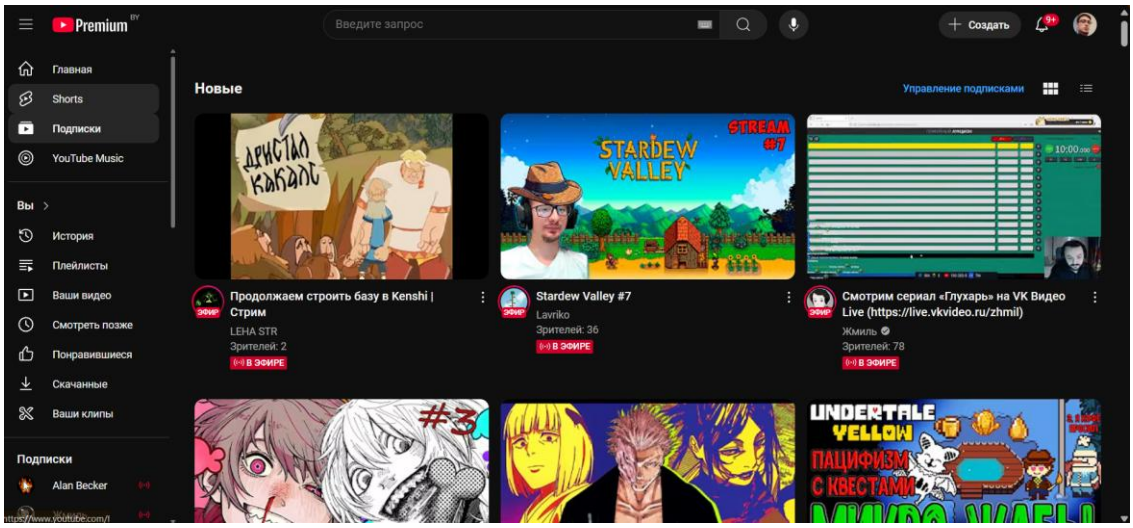

Рисунок 1.1 – Приложение «YouTube»

Проанализировав интерфейс, было принято решение делать что-то похожее на дизайн YouTube так как он достаточно современный и интуитивный.

1.2.2 Веб-приложение «RUTUBE»

RUTUBE [2] обладает широким спектром функций, включая загрузку и просмотр видео, создание плейлистов, подписку на каналы, комментирование и оценку видео, а также возможность проведения прямых трансляций. Этот видеохостинг также позволяет пользователям создавать и управлять своими каналами, а также монетизировать контент через рекламу и спонсорство.

RUTUBE имеет простой и удобный интерфейс, который легко использовать. Дизайн этого видеохостинга состоит из основных функций в верхней части экрана и списка рекомендованных видео в центральной части. RUTUBE также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид сайта.

Дизайн RUTUBE – минималистичный и простой в использовании. Цветовая схема состоит в основном из белого и синего цветов, что создает ощущение легкости и чистоты для пользователя.

Интерфейс приложения представлен на рисунке 1.2.

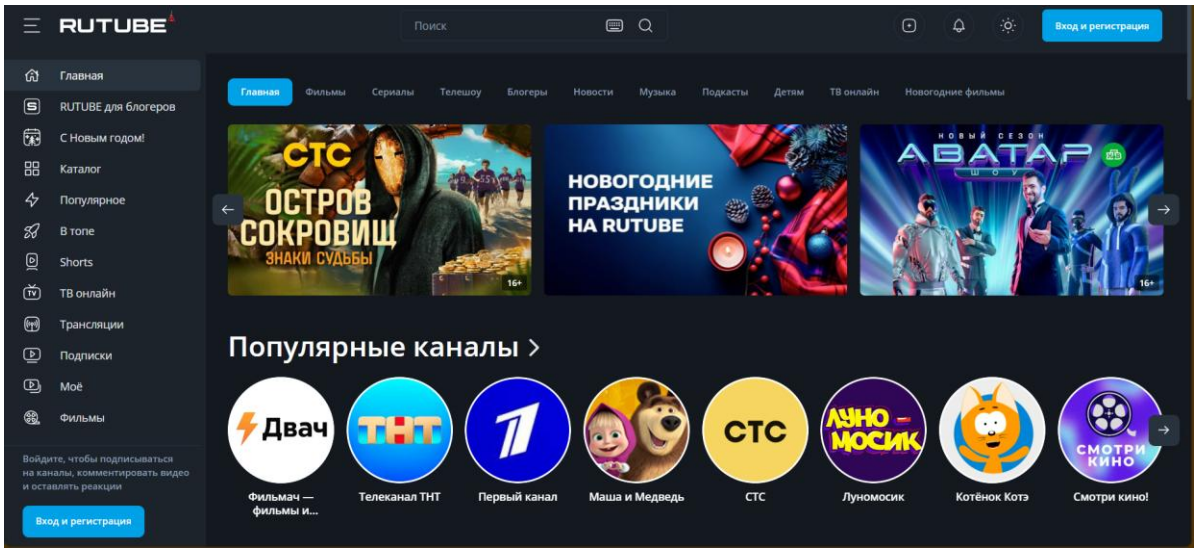

Рисунок 1.2 – Приложение «RUTUBE»

Проанализировав приложение было решено сделать похожим дизайн интерфейса страницы видео.

1.2.3 Веб-приложение «VK Video»

VK Video [3] обладает широким спектром функций, включая загрузку и просмотр видео, создание плейлистов, подписку на каналы, комментирование и оценку видео, а также возможность проведения прямых трансляций. Этот видеохостинг также позволяет пользователям создавать и управлять своими каналами, а также монетизировать контент через рекламу и спонсорство2.

VK Video имеет простой и удобный интерфейс, который легко использовать. Дизайн этого видеохостинга состоит из основных функций в верхней части экрана и списка рекомендованных видео в центральной части. VK Video также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид сайта.

Дизайн VK Video – довольно-таки минималистичный и простой в использовании. Цветовая схема состоит в основном из белого и синего цветов, что создает ощущение легкости и чистоты для пользователя.

Интерфейс приложения представлен на рисунке 1.3.

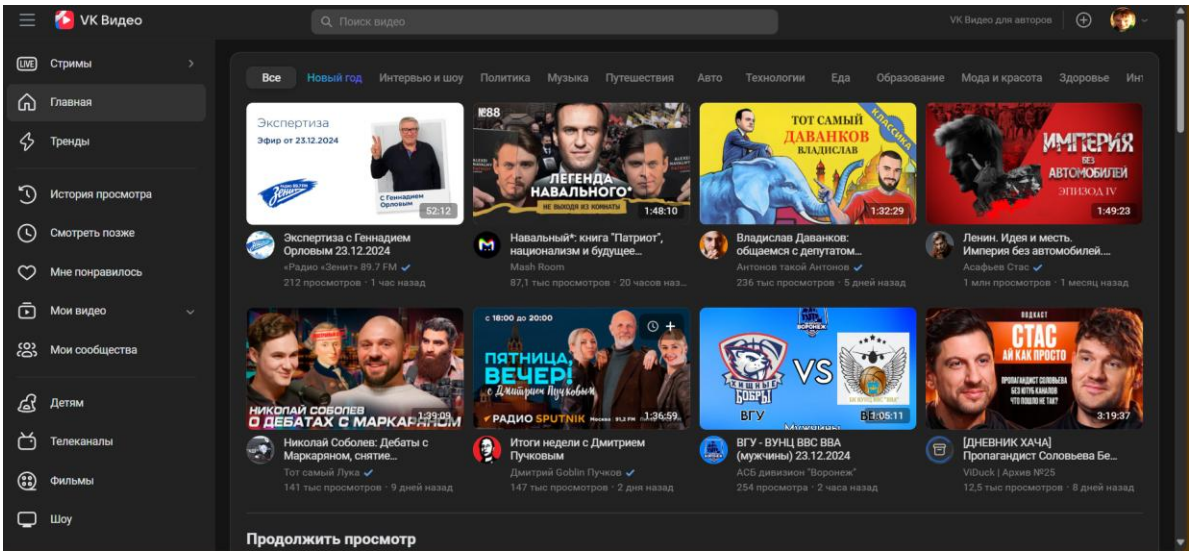

Рисунок 1.3 – Приложение «VK Video»

Проанализировав приложение было решено реализовать похожую боковую панель так как она достаточно интуитивна.

1.3 Вывод

В рамках данной главы было выполнено:

1. Поставлены необходимые задачи для реализации веб-приложения;
2. Разобраны три аналогичных решения: «YouTube», «RUTUBE», «VK Video». Из «YouTube» взят интерфейс, из “RUTUBE” взят основной дизайн приложения, а из «VK Video» взята боковая панель.

2 Проектирования веб-приложения

2.1 Функциональные возможности веб-приложения

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 2.1.

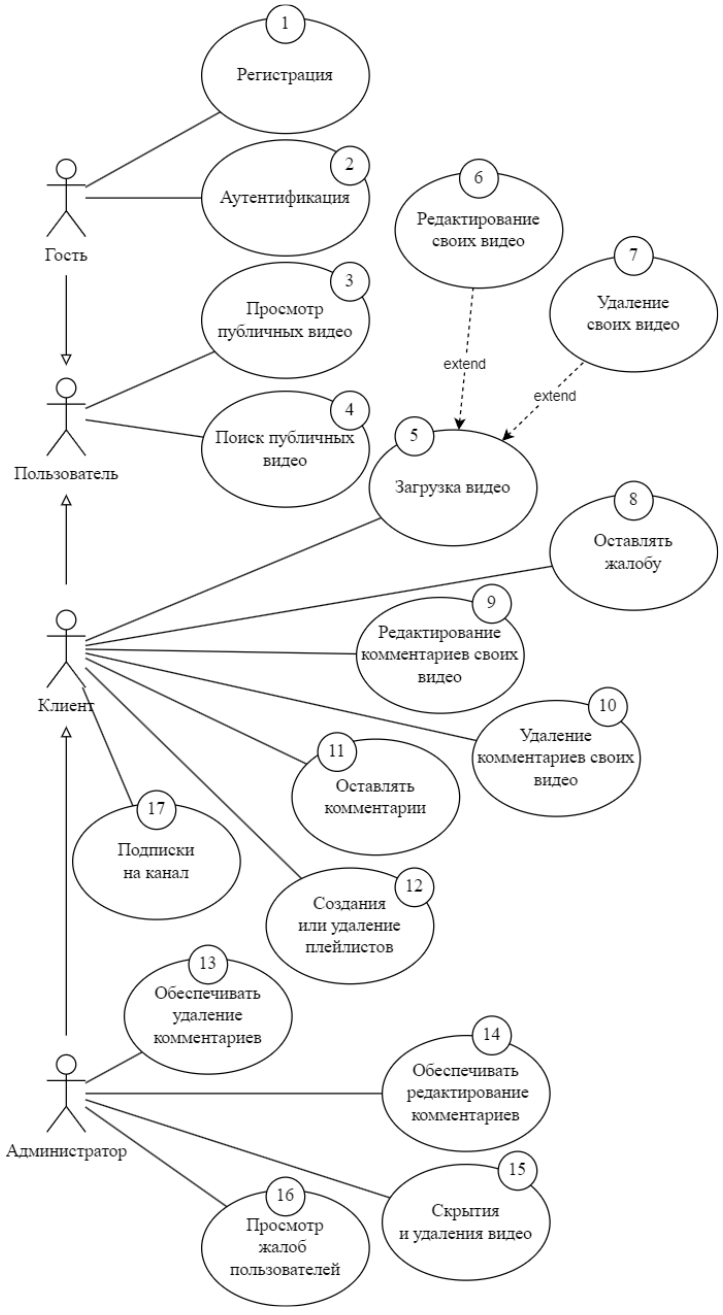

Рисунок 2.1 – Диаграмма вариантов использования

Таким образом на диаграмме видно, что какой пользователь делает. Описание ролей предоставлена на таблице 2.1.

Таблица 2.1 – 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 2.2.

Таблица 2.2 – 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 2.2

№	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.

2.2 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 2.2 визуализирует основные компоненты архитектуры приложения и их размещение.

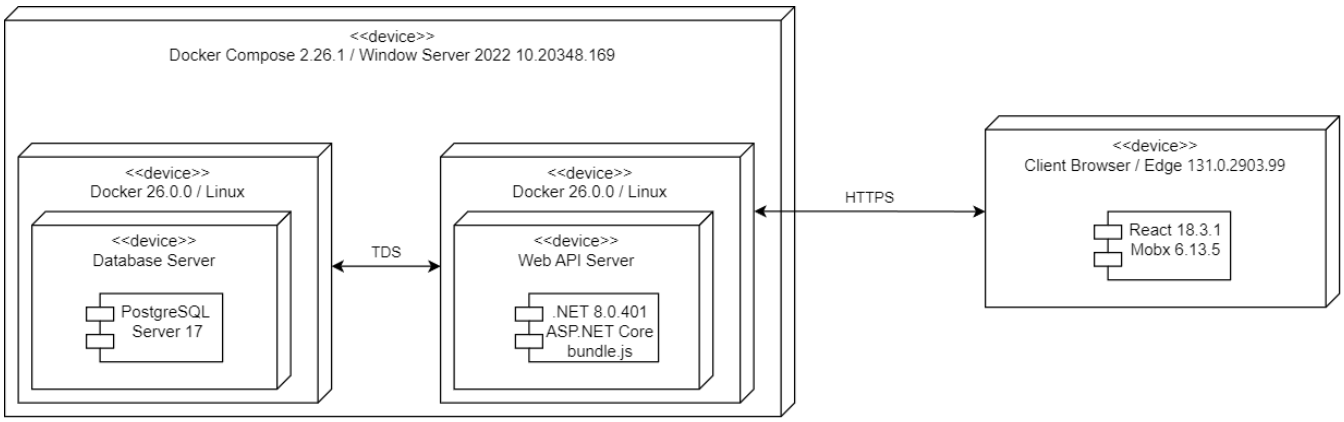

Рисунок 2.2 – Структурная схема веб-приложения

Серверная часть приложения расположена внутри докер контейнеров «Bytube» который завёрнут на [4] Window Server 2022. 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 [5] 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 [6] React 18.3.1 с управлением сборками через [7] 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 [8] ASP.NET Core 8.0, предоставляемого Visual Studio, что обеспечивает высокую совместимость с инструментами разработки и стабильность при выполнении серверного приложения;

– Контейнер базы данных [9] PostgreSQL версии 15, который позволяет эффективно работать с хранимыми данными.

Composer в проекте используется, так как приложение построено на технологиях, требующих его интеграции. А так для установки пакетов в проекте применяется механизм управления зависимостями 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 React 18.2.0, и управление сборкой через Вебpack 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.

2.3 Логическая схема базы данных

Для реализации функциональности веб-приложения видеохостинга BYTUBE разработана реляционная база данных. Логическая схема данных предоставлена на рисунке 2.3.

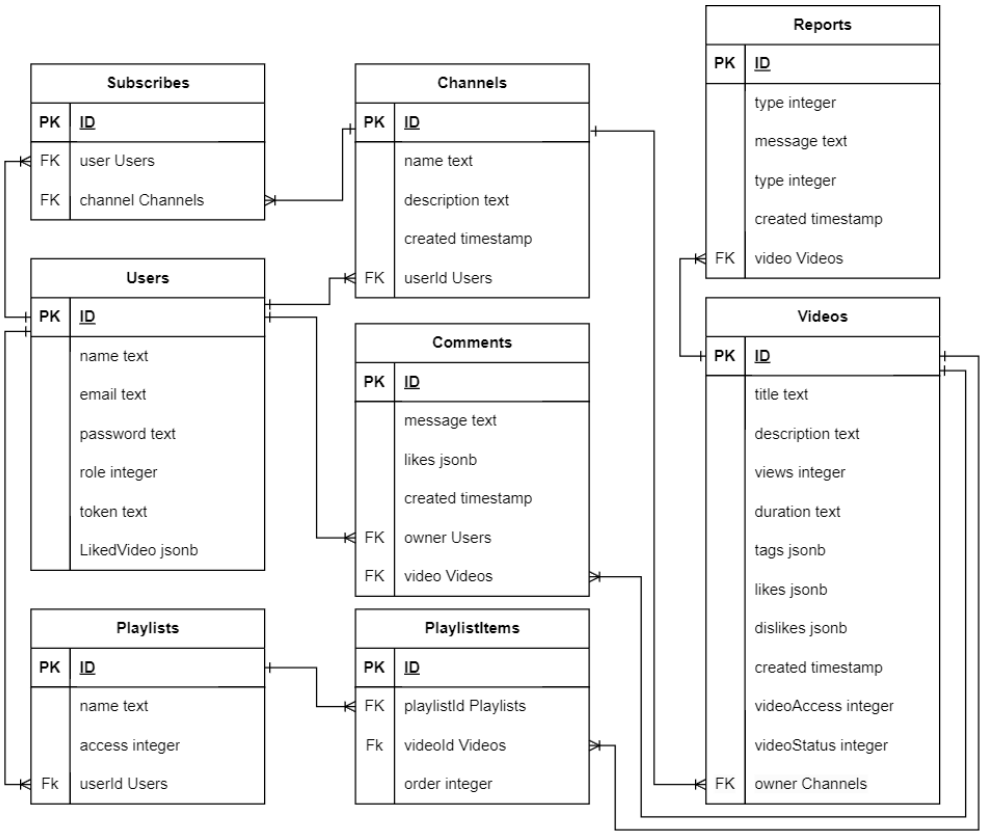

Рисунок 2.3 – Логическая схема базы данных

Логическая схема 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 2.3.

Таблица 2.3 – 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 2.4.

Таблица 2.4 – Описание таблицы Users

Поле	Тип данных	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 2.5.

Таблица 2.5 – Описание таблицы Channels

Поле	Тип данных	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 2.6.

Таблица 2.6 – Описание таблицы Videos

Поле	Тип данных	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 2.7

Таблица 2.7 – Описание таблицы Comments

Поле	Тип данных	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 2.8

Таблица 2.8 – Описание таблицы Playlists

Поле	Тип данных	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 2.9

Таблица 2.9 – Описание таблицы PlaylistItems

Поле	Тип данных	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 2.10.

Таблица 2.10 – Описание таблицы Subscribes

Поле	Тип данных	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 2.11.

Таблица 2.11 – Описание таблицы Reports

Поле	Тип данных	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 2.12.

Таблица 2.12 – 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.

2.4 Вывод

Таким образом в рамках проектирования веб-приложения было выполнено.

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

3 Реализация веб-приложения

3.1 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 [10] 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 3.1.

```
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 3.1 – 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.

3.2 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 [11] 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 Веб 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 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.

3.2.1 Просмотр 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 [12] JSON, содержащий данные видео (заголовок, описание, статус, метки, количество просмотров), а также данные о канале, к которому относится видео. Сообщение ответа предоставлена в листинге 3.2.

```
{
    "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 3.2 – 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.

3.2.2 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 3.3.

```
[
    {
        "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 3.3 – Ответ метода «Select»

Листинг метода предоставлен в приложении В.

3.2.3 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 аутентификации и авторизации в рамках контроллера был разработан метод «signinJwt». Метод выполняет вход пользователя с использованием JWT (JSON Веб Token). Принимает в качестве параметров почти и пароль, пример запроса предоставлен в листинге 3.4. Возвращает 200 OK при успешной авторизации.

```
{
    "Email": "johndoe@example.com",
    "Password": "SecurePassword123"
}
```

Листинг 3.4 – Данные запроса метода «signinJwt»

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

3.2.4 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 3.5.

```
{
  "title": "My First Video",
  "description": "This is a description!",
  "tags": ["fun", "vlog", "travel"],
  "videoAccess": 1,
  "videoStatus": 1,
  "previewFile": "/file.png
  "videoFile": "/file.mp4"
}
```

Листинг 3.5 – 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 3.6.

```
{
  "title": "Updated Video Title",
  "description": "This is the updated description for the video.",
  "tags": ["updated", "education", "tutorial"],
  "videoAccess": 0,
  "previewFile": "/file.png
}
```

Листинг 3.6 – 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.

3.2.5 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 3.7.

```
{
  "Message": "Очень интересное видео!",
  "VideoId": 123
}
```

Листинг 3.7 – 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.

3.2.6 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 предоставлен в листинге 3.8

```
[
  {
    "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 3.8 – 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 3.9. Если запрос отработал без ошибок, то возвращается статут код 200 OK.

```
{
  "description": "Контент нарушает авторские права.",
  "type": 1,
  "videoId": 42
}
```

Листинг 3.9 – Данные запроса метода «Post»

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

3.2.7 Создание 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 3.10. При успешной обработке возвращает код 200 OK.

```
{
  "name": "Мои любимые видео",
  "access": 1
}
```

Листинг 3.10 – 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.

3.2.8 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.

3.2.9 Скрытие 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 «BlockingByAdmin» предоставлен в листинге В.

3.3 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.

3.4 Вывод

Таким образом в рамках реализации веб-приложения было выполнено.

1. Реализована база данных используя СУБД Postgres 15;

2. Реализован весь основной функционал приложения такой как: просмотр публичных видео, поиск публичных видео, регистрации, аутентифицироваться и авторизации и тд.;

3. Реализованы все необходимые компоненты на стороне клиента.

4 Тестирование веб-приложения

4.1 Функциональное тестирование

Для тестирования функционала были разработаны специальные тесты, которые предоставлены в таблице 4.1.

Таблица 4.1 – Функциональные тесты

№	Название функции	Описание теста	Ожидаемый результат	Результат
1	Просмотр публичных видео	Зайти на главную страницу как гость. Убедиться, что отображаются доступные видео. Нажать на видео для просмотра.	Воспроизведение видео	Успешно
2	Поиск публичных видео	Ввести запрос в строку поиска как гость. Нажать "Поиск".	Результаты поиска соответствуют запросу.	Успешно
3	Регистрация	На главной странице нажать "Вход". Ввести корректные данные. Нажать "Подтвердить".	В таблице БД появилась новая запись.	Успешно
4	Аутентификация	На странице авторизации ввести e-mail и пароль ранее созданного пользователя. Нажать "Подтвердить".	Вход осуществлён.	Успешно
5	Авторизация	Проверить доступность функций авторизованного пользователя после входа (например, возможность создания плейлиста).	Функции доступны.	Успешно
6	Загрузка видео	Авторизоваться. Перейти в "Студию". Нажать "Загрузить видео". Выбрать файл, ввести метаданные (название, описание, теги). Нажать "Подтвердить".	Видео появилось в списке ваших видео.	Успешно

Продолжение таблицы 4.1

№	Название функции	Описание теста	Ожидаемый результат	Результат
7	Редактирование своего видео	Перейти в "Студию". Выбрать видео. Нажать "Редактировать". Изменить название/описание/теги. Нажать "Сохранить".	Изменения вступили в силу.	Успешно
8	Удаление своего видео	Перейти в "Студию". Выбрать видео. Нажать "Редактирование". Нажать "Удалить".	Видео удалено.	Успешно
9	Оставление комментария	Авторизоваться. Зайти на страницу видео. Ввести текст комментария. Нажать "Отправить".	Комментарий отображается.	Успешно
10	Оставление жалобы	На странице видео нажать "Пожаловаться". Ввести причину. Нажать "Подтвердить".	Ошибок не возникло.	Успешно
11	Редактирование комментария на своих видео	Зайти на страницу своего видео. Выбрать комментарий. Нажать "Редактировать". Изменить текст. Нажать	Комментарий обновлён.	Успешно
12	Удаление комментария на своих видео	Зайти на страницу своего видео. Выбрать комментарий. Нажать "Удалить". Подтвердить действие.	Комментарий удалён.	Успешно
13	Подписка на канал	Зайти на страницу канала. Нажать "Подписаться". Убедиться, что статус подписки обновлён.	Канал есть в списке подписок.	Успешно

Продолжение таблицы 4.1

№	Название функции	Описание теста	Ожидаемый результат	Результат
14	Создание плейлиста	Зайти в "Плейлисты". Нажать "Создать плейлист". Ввести название. Указать доступ (публичный/приватный). Нажать "Подтвердить".	Плейлист появился в списке плейлистов.	Успешно
15	Удаление плейлиста	Зайти в "Плейлисты". Выбрать плейлист. Нажать "Удалить".	Плейлист удалён.	Успешно
16	Скрытие видео администратором	Зайти в админ-панель. Выбрать видео. Нажать "Скрыть".	Видео не доступно для просмотра	Успешно
17	Удаление видео администратором	Зайти в админ-панель. Выбрать видео. Нажать "Удалить". Подтвердить действие.	Видео удалено.	Успешно
18	Просмотр пользовательских жалоб	Зайти в админ-панель. Выбрать видео. Перейти в раздел жалоб.	Жалобы отображаются.	Успешно
19	Удаление комментария администратором	Зайти в админ-панель. Выбрать видео. Перейти в список комментариев. Выбрать комментарий. Нажать "Удалить".	Комментарий удалён.	Успешно
20	Редактирование комментария администратором	Зайти в админ-панель. Выбрать видео. Перейти в список комментариев. Выбрать комментарий. Нажать "Редактировать". Изменить текст. Нажать "Подтвердить".	Изменения вступили в силу.	Успешно

Таким образом был протестирован весь основной функционал веб-приложения.

4.2 Автоматизированное тестирование

Для автоматизированного теста были разработаны специальные тесты, которые предоставлены в таблице 4.2.

Таблица 4.2 – 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.

4.3 Нагрузочное тестирования

Для проверки стабильности сервера были разработаны специальные нагрузочные тесты, которые предоставлены в таблице 4.3.

Таблица 4.3 – 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.

4.4 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 одновременною. Таким образом покрытость тестами составила около 80%. Это обеспечивает уверенность в том, что приложение удовлетворит ожидания пользователей, обеспечивая стабильность и удобство работы.

5 Руководство пользователя и программиста

5.1 Руководство пользователя

Для навигации нового пользователе либо программиста в веб-приложении «BYTUBE» было разработана руководство пользователя.

5.1.1 Просмотр публичных видео

Для просмотра публичных видео необходимо зайти на главную страницу, скриншот предоставлен на рисунке 5.1.

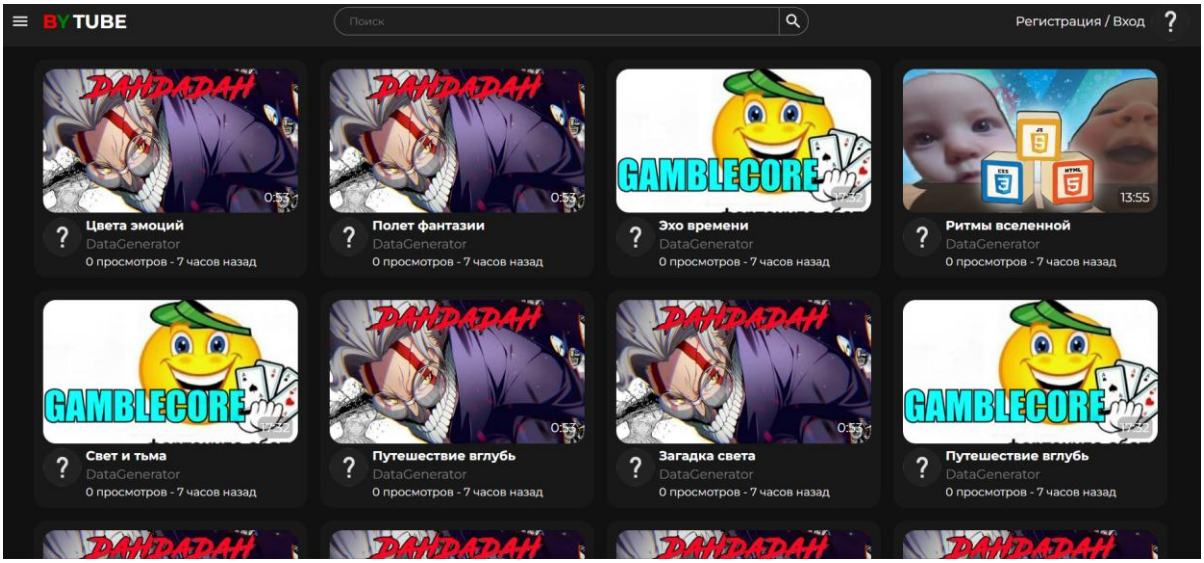

Рисунок 5.1 – Главная страница

Выбрать любой ролик в каталоге, после чего произойдет переход на страницу с выбранным видео. Пример страницы видео предоставлено на рисунок 5.2.

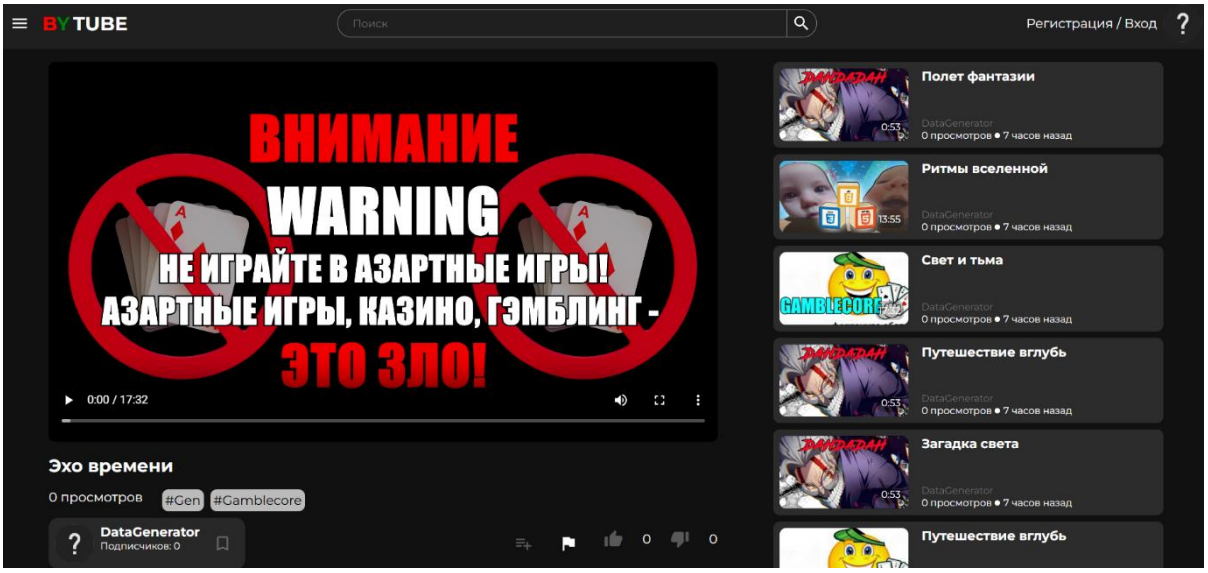

Рисунок 5.2 – Страница видео

5.1.2 Поиск публичных видео

Для поиска публичных видео требуется ввести запрос в строку поиска, которая находится в верхней панели главной страницы. После чего нужно нажать на кнопку поиска. После чего произойдёт переход на страницу поиска. Пример страницы поиска предоставлен на рисунке 5.3.

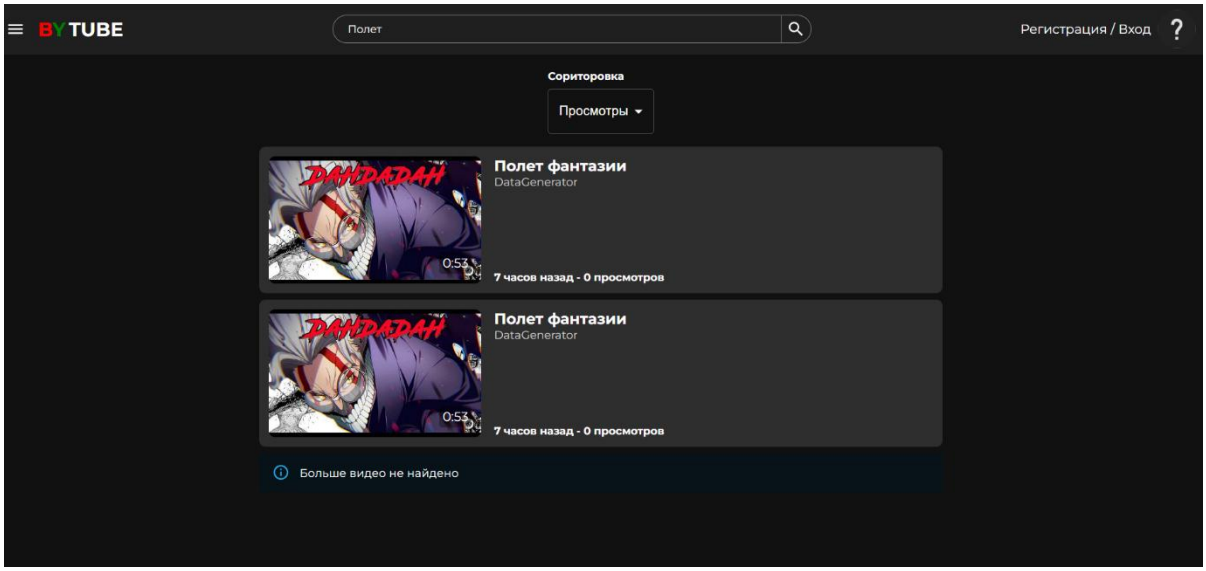

Рисунок 5.3 – Страница поиска

5.1.3 Регистрация пользователя

Для регистрации на главной странице требуется нажать на кнопку регистрация в правом верхнем верху. На странице регистрации требуется заполнить поля никнейм, почту, пароль, повторить пароль. После чего нужно нажать кнопку «подтвердить». Пример страницы предоставлен на рисунке 5.4.

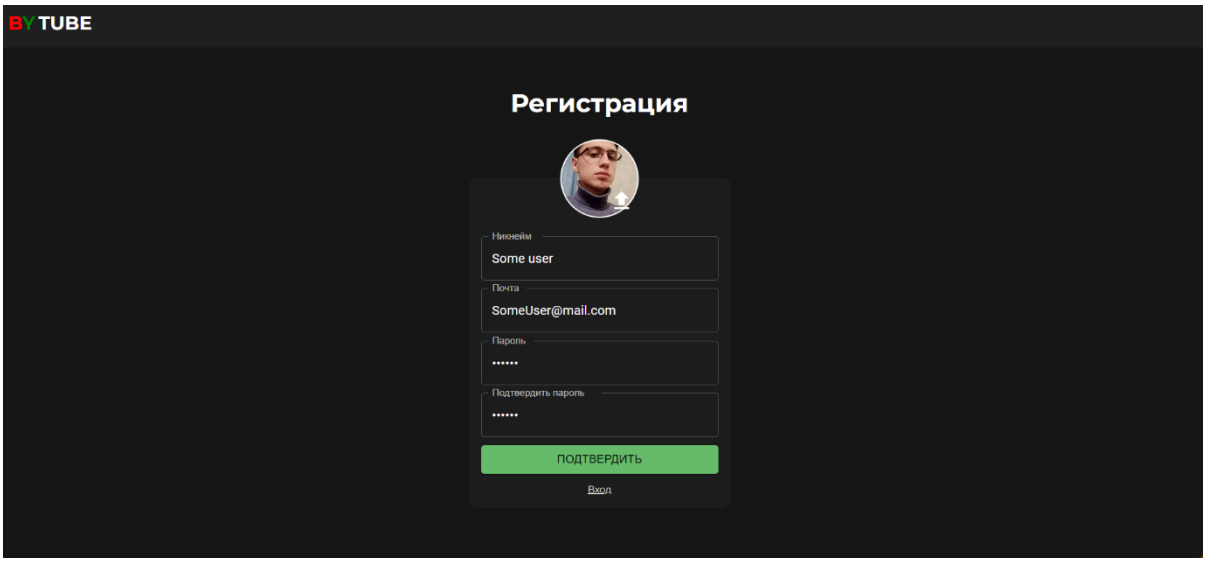

Рисунок 5.4 – Страница регистрации

5.1.4 Аутентификации пользователя

Для аутентификации на главной странице требуется нажать кнопку «войти», в правом верхнем углу. На странице входа требуется сначала заполнить поле почты, нажать кнопку «подтвердить», заполнить поле пароля и опять нажать кнопку «продолжить». Пример входа предоставлен на рисунке 5.5.

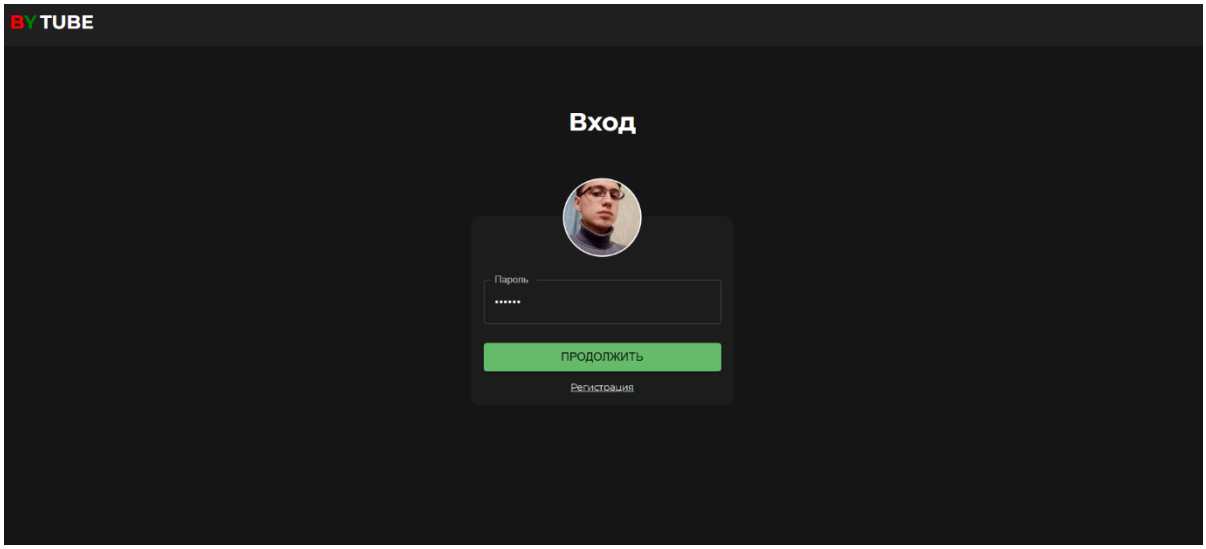

Рисунок 5.5 – Страница входа

5.1.5 Загрузка видео

Для загрузки видео необходимо создать канал. Что бы создать канал в боковой панели нужно нажать кнопку «создать канал». В форме создания канал нужно указать фото превью, фото иконки, название канала, описание. После чего нужно нажать кнопку «подтвердить». Пример формы создание канала предоставлен на рисунке 5.6.

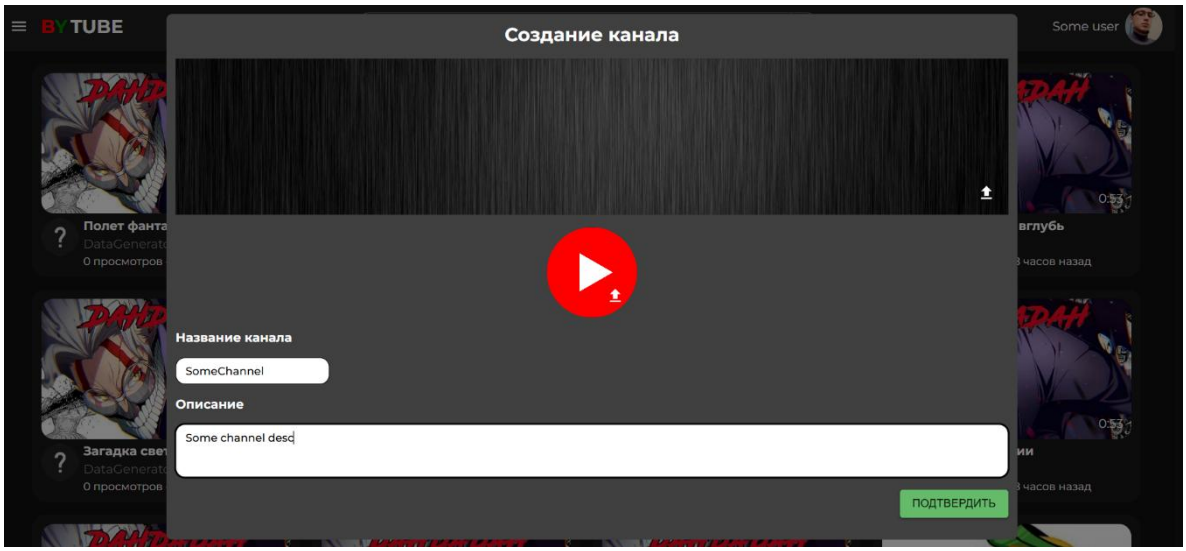

Рисунок 5.6 – Форма создания канала

Для загрузки видео нужно выбрать созданный канал в боковой панели. Далее на странице студии (рисунок 5.8) надо нажать кнопку «загрузить видео». В форме загрузки видео требуется заполнить поля: видео, название, описание, теги, фото превью, доступ. После чего нужно нажать на кнопку «подтвердить» и дождаться загрузки видео. Пример формы создание канала предоставлен на рисунке 5.7.

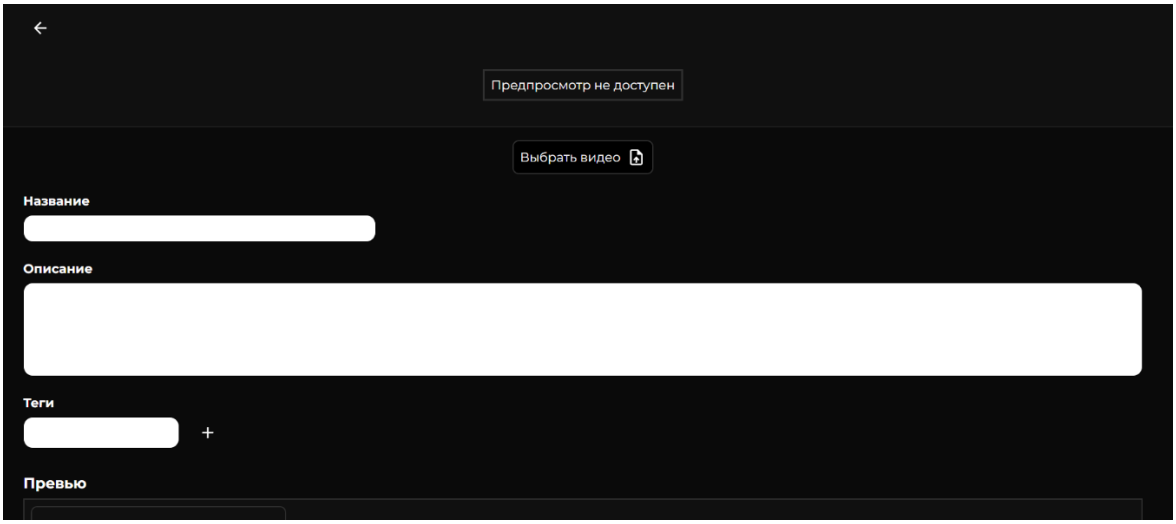

Рисунок 5.7 – Форма загрузки видео

5.1.6 Редактирование видео

Для редактирования видео нужно выбрать созданный канал в боковой панели. Далее на странице студии надо выбрать существующее видео. Пример страницы студии предоставлен на рисунке 5.8.

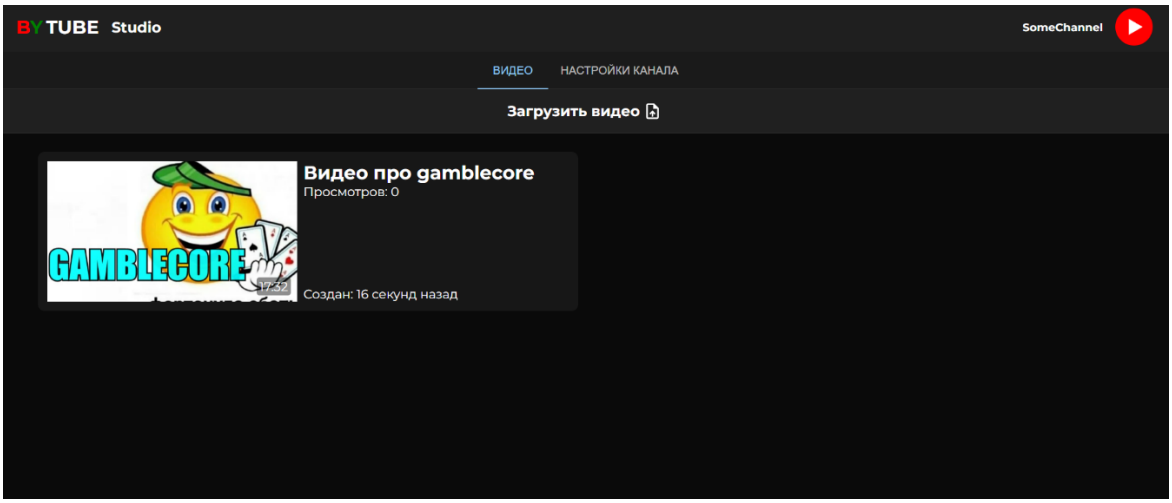

Рисунок 5.8 – Страница студии

Далее требуется нажать на верхней панели «редактирования». В форме редактирования видео можно изменить поля: название, описание, теги, фото превью, доступ. После чего нужно нажать на кнопку «подтвердить». Пример страницы студии предоставлен на рисунке 5.9.

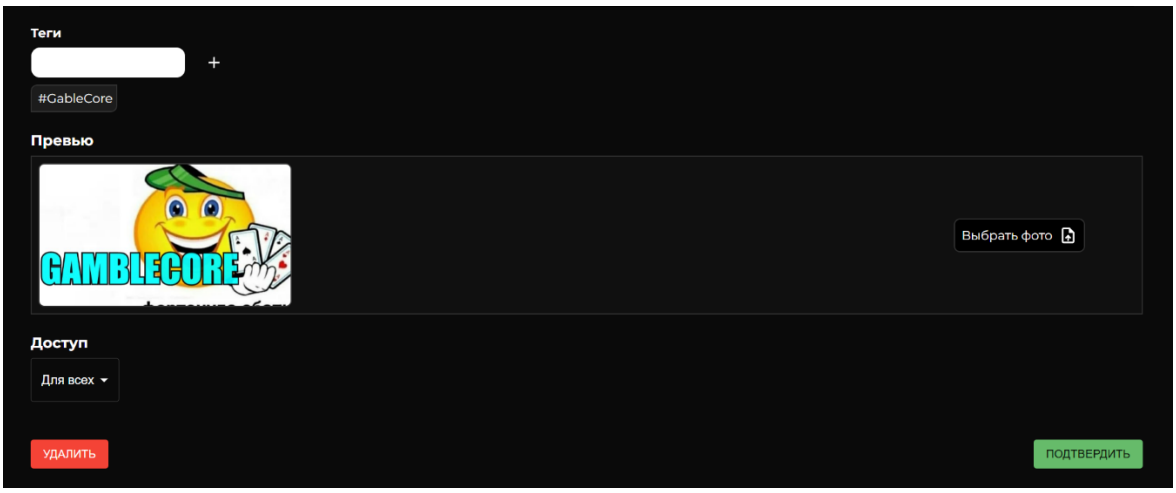

Рисунок 5.9 – Страница студии

5.1.7 Удаление видео

Для удаления видео в той же форме редактирования видео (рисунок 5.9) требуется нажать кнопку «удалить».

5.1.8 Оставление комментария

Для того что бы оставить комментарий нужно перейти на страницу любого видео и нажать панель «комментарии». После чего в текстовом поле нужно ввести текст и нажать на кнопку «отправить». Пример панели комментариев предоставлен на рисунке 5.10.

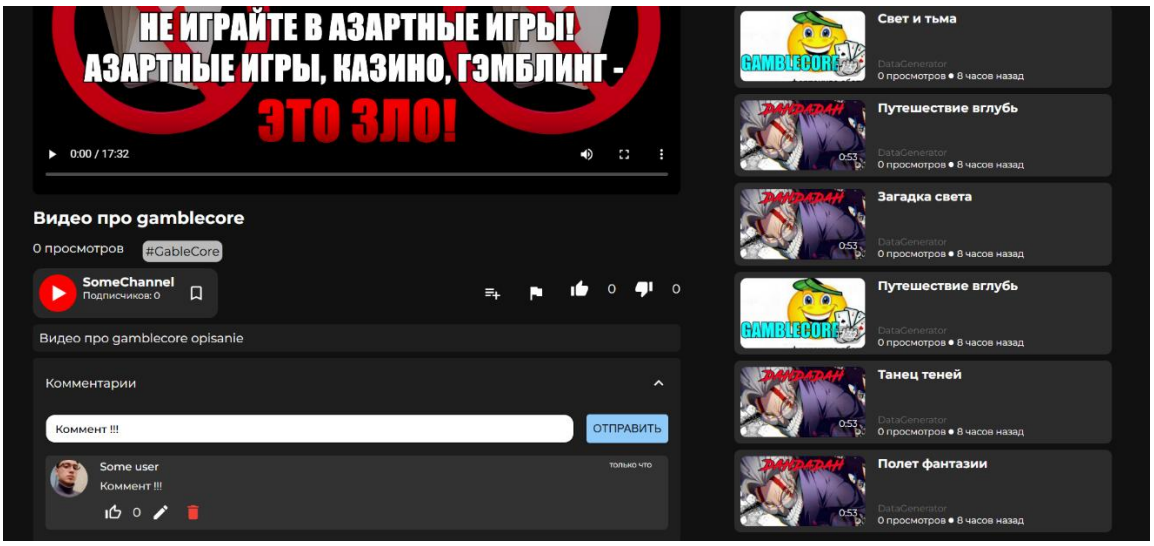

Рисунок 5.10 – Страница видео с панелью комментариев

5.1.9 Отправка жалобы

Для того чтобы отправить жалобу на видео требуется в странице видео нажать на иконку «пожаловаться». Далее в форме жалобы требуется выбрать причину

жалобы и нажать кнопку «подтвердить». Пример формы жалобы предоставлен на рисунке 5.11.

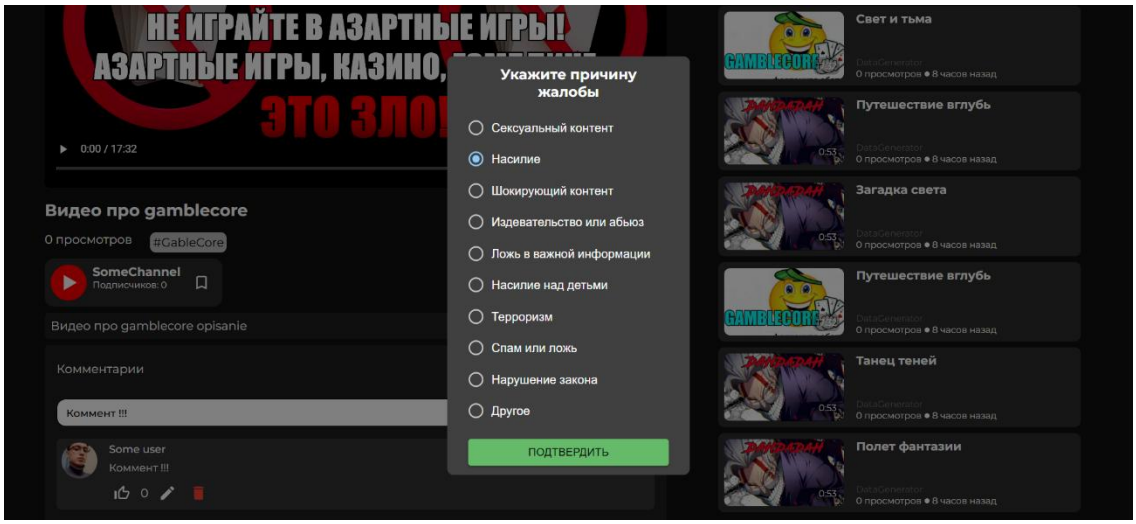

Рисунок 5.11 – Форма жалобы

5.1.10 Редактирование комментариев своих видео

Для того чтобы редактировать комментарии своих видео нужно перейти на страницу своего видео (рисунок 5.10) и нажать панель «комментарии». После чего требуется нажать на кнопку «редактировать», текст станет доступным для изменения. После изменения нужно нажать кнопку «подтвердить».

5.1.11 Удаление комментариев своих видео

Для того чтобы удалить комментарии своих видео (рисунок 5.10) нужно перейти на страницу своего видео и нажать панель «комментарии». После чего требуется нажать на кнопку «удалить».

5.1.12 Подписка на канал

Для того чтобы подписать на канал требуется в странице видео (рисунок 5.10) нажать на иконку «подписаться».

5.1.13 Создание и удаление плейлистов

Для того чтобы создать плейлист требуется на баковой панели главной страницы нажать на кнопку «создать плейлист». После чего требуется заполнить поле название и доступ. Далее нужно нажать кнопку «подтвердить». Пример формы создания плейлиста предоставлен на рисунке 5.12.

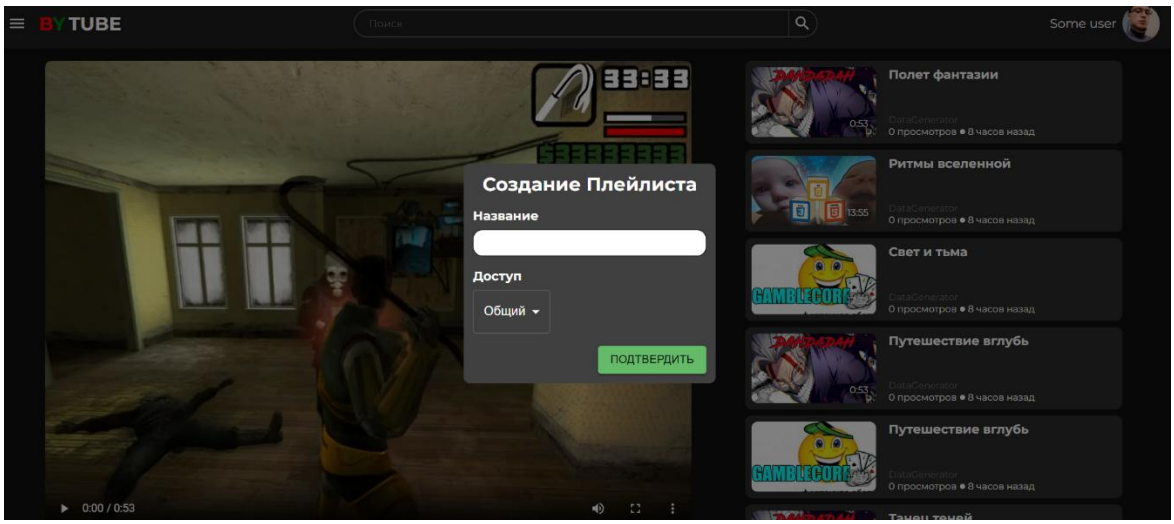

Рисунок 5.12 – Форма создания плейлиста

Для того чтобы удалить плейлист требуется на баковой панели главной страницы выбрать плейлист. В форме плейлиста нужно нажать кнопку «удалить плейлист». Пример формы создания плейлиста предоставлен на рисунке 5.13.

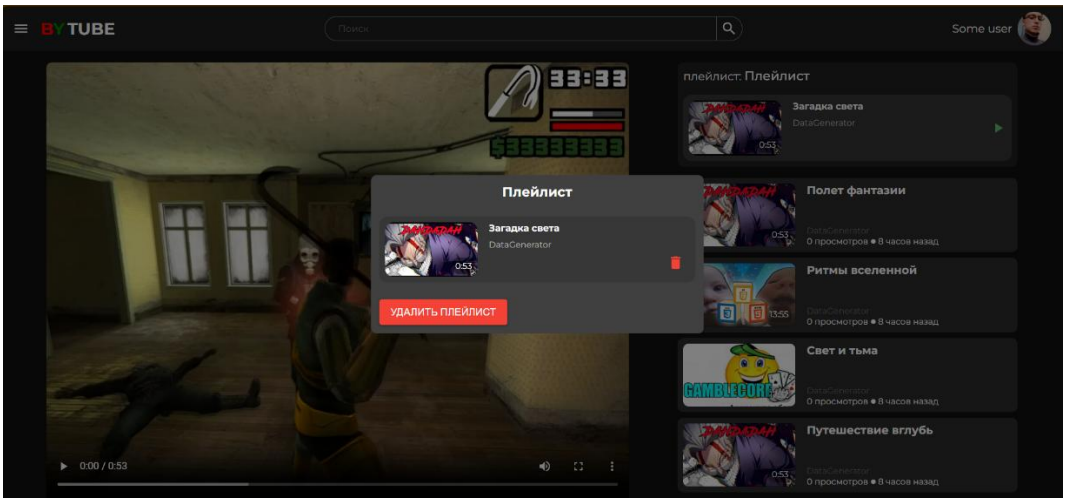

Рисунок 5.13 – Форма плейлиста

5.1.14 Скрытие видео администратором

Для скрытия видео администратором требуется войти в аккаунт администратора, после чего в боковой панели главной страницы нужно нажать кнопку «панель администратора». На странице администратора нужно нажать кнопку «заблокировать» на выборном видео. Пример формы создания плейлиста предоставлен на рисунке 5.14.

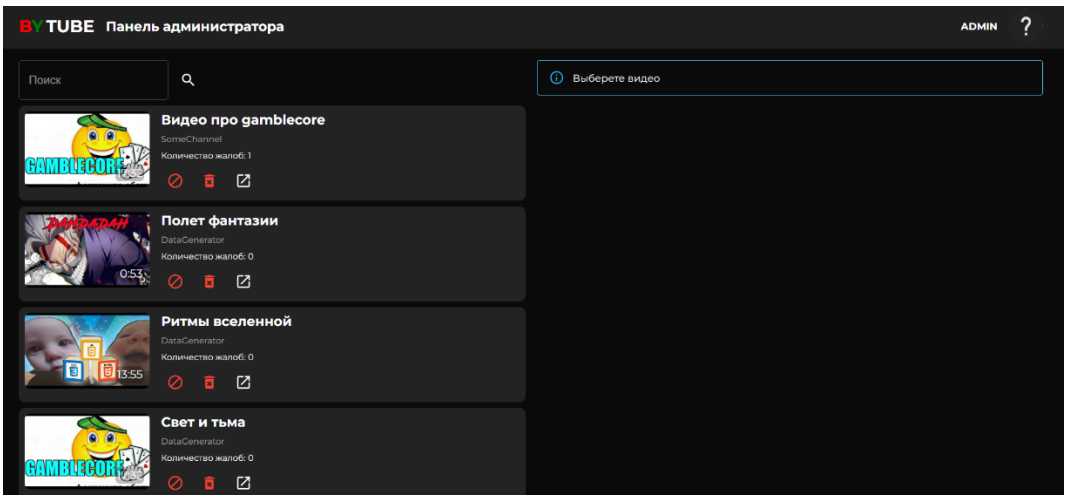

Рисунок 5.14 – Страница администратора

5.1.15 Удаление видео администратором

Для удаления видео администратором требуется войти в аккаунт администратора, после чего в боковой панели главной страницы нужно нажать кнопку «панель администратора». На странице администратора (рисунок 5.14) нужно нажать кнопку «удалить» на выборном видео.

5.1.16 Редактирование комментария администратором

Для того чтобы редактировать комментарии видео администратором требуется войти в аккаунт администратора, после чего нужно перейти на страницу любого видео (рисунок 5.10) и нажать панель «комментарии». После чего требуется нажать на кнопку «редактировать», текст станет доступным для изменения. После изменения нужно нажать кнопку «подтвердить».

5.1.17 Удаление комментария администратором

Для того чтобы удалить комментарии видео администратором требуется войти в аккаунт администратора, после чего нужно перейти на страницу любого видео (рисунок 5.10) и нажать панель «комментарии». После чего требуется нажать на кнопку «удалить».

5.2 Руководство программиста

Для развёртывания веб-приложения применяется инструмент Docker Compose. Перед развёртыванием веб-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 [13] Docker Engine и для [14] 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.

5.3 Вывод

Таким образом в рамка руководства было выполнено:

– разработано руководство пользователя, которое будет необходимым для навигации по веб-приложению;

– разработано руководство программиста, которое будет необходимо разработчикам для развертывания веб-приложения.

Заключение

Результаты выполнения курсового проекта:

– разработано веб-приложение «BYTUBE»;

– использованы технологии: ASP.NET core 8.0, React, Mobx, PostgreSQL.

– веб-приложение поддерживает 3 роли: «Гость», «Клиент», «Администратор»;

– проанализировано 3 аналога;

– разработано 8 таблиц базы данных;

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

– осуществлена поддержка протокола HTTPS;

– проведено ручное, автоматическое, нагрузочное тестирование. Покрытие тестами составило 80%;

– приложение было развёрнуто на Docker;

– было разработано 47 различных методов и маршрутов.

Данное приложение готово к эксплуатации и использованию в сети Интернет.

Список используемых источников

1 YouTube [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.youtube.com – Дата доступа: 22.12.2024;

2 RUTUBE [Электронный ресурс]. – Электронные данные. – Режим доступа: https://rutube.ru – Дата доступа: 22.12.2024;

3 VK Video [Электронный ресурс]. – Электронные данные. – Режим доступа: https://vkvideo.ru – Дата доступа: 22.12.2024;

4 HTTPS RFC 2818 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://datatracker.ietf.org/doc/html/rfc2818 – Дата доступа: 22.12.2024;

5 Windows Server 2022 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.microsoft.com/ru-ru/windows-server – Дата доступа: 22.12.2024;

6 React Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://react.dev/learn – Дата доступа: 22.12.2024;

7 Вебpack Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://вебpack.js.org/guides – Дата доступа: 22.12.2024;

8 ASP .NET 8.0 Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/aspnet/core – Дата доступа: 22.12.2024;

9 Postgres SQL Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.postgresql.org/docs/ – Дата доступа: 22.12.2024;

10 Metanit Entity Framework Core [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/sharp/efcore/1.2.php – Дата доступа: 22.12.2024;

11 Habr руководство по REST API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habr.com/ru/articles/590679/ – Дата доступа: 22.12.2024;

12 JSON RFC 8259 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://datatracker.ietf.org/doc/html/rfc8259 – Дата доступа: 22.12.2024;

13 Docker Engine Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/engine/ – Дата доступа: 22.12.2024;

14 Docker Compose Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/compose/ – Дата доступа: 22.12.2024;

Приложение А

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.вебapp-http.rule=Host(`localhost`)"
      - "traefik.http.services.вебapp.loadbalancer.server.port=8080"
      - "traefik.http.routers.вебapp-https.rule=Host(`localhost`)"
      - "traefik.http.routers.вебapp-https.tls=true"
      - "traefik.http.services.вебapp-
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"]
```