Минобрнауки России

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Санкт-Петербургский государственный технологический институт

(технический университет)»

|  |  |  |
| --- | --- | --- |
| УГНС | 09.00.00 | Информатика и вычислительная техника |
| Направление подготовки | 09.03.01 | Информатика и вычислительная техника |
| Направленность программы |  | Системы автоматизированного проектирования |
| Форма обучения |  | Очная |
| Факультет |  | Информационных технологий и  управления |
| Кафедра |  | Систем автоматизированного  проектирования и управления |
| Учебная дисциплина |  | Интернет-технологии |

**КУРСОВОЙ ПРОЕКТ**

|  |  |
| --- | --- |
| Тема | Разработка веб-приложения для публикации и просмотра видео |

Зав. каф., д-р техн. наук, проф. Т.Б. Чистякова

Руководитель

курсового проекта

доцент, канд. техн. наук А. С. Разыграев

Исполнитель

Студент группы 494 А. М. Казанцев

Санкт-Петербург

2022

СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ 3**](#_Toc104508702)

[**1 Аналитический обзор 4**](#_Toc104508703)

[**1.1 Описание работы видеохостинга. Формирование функциональных требований к информационной системе хранения и воспроизведения видеоматериала 4**](#_Toc104508704)

[**1.2 Обзор и обоснование выбора технологий вёрстки макетов веб-приложения 5**](#_Toc104508705)

[**1.3 Обзор и обоснование выбора архитектуры информационной системы, способа клиент-серверного взаимодействия, используемой СУБД, инструментальных средств разработки 12**](#_Toc104508706)

[**1.4 Выводы по аналитическому обзору 16**](#_Toc104508707)

[**2 Цель и задачи работы 17**](#_Toc104508708)

[**3 Технология разработки программного комплекса 18**](#_Toc104508709)

[**3.1 Разработка функциональной структуры хранения и использования видеоматериала с учётом клиент-серверной архитектуры. Постановка задачи разработки 18**](#_Toc104508710)

[**3.2 Разработка UML-диаграммы вариантов использования веб-приложения для пользователя 19**](#_Toc104508711)

[**3.3 Разработка макета графического интерфейса веб-приложения для видеохостинга 19**](#_Toc104508712)

[**3.4 Разработка блок-схемы обобщенного алгоритма функционирования приложения 21**](#_Toc104508713)

[**3.5 Проектирование серверного интерфейса взаимодействия с клиентом и СУБД 24**](#_Toc104508714)

[**3.6 Разработка базы данных веб-приложения для видеохостинга 26**](#_Toc104508715)

[**3.7 Разработка программного обеспечения веб-приложения для видеохостинга 28**](#_Toc104508716)

[**4 Тестирование веб-приложения 31**](#_Toc104508717)

[**5 Выводы по проекту 33**](#_Toc104508718)

[**Список использованных источников 34**](#_Toc104508719)

# ВВЕДЕНИЕ

На сегодняшний день происходит стремительное развитие СМИ. Скорость, универсальность, мобильность, мультимедийность и интерактивность стало основополагающим для создания информационного контента. Все больше традиционных СМИ переходит в формат интернет-изданий и каналов. Задачей журналиста становится не просто получить и распространить информацию, а сделать это быстро.

Все это происходит благодаря развитию цифровой технологии, которая занимает сейчас одно из первых мест в жизни человека. К традиционным СМИ и традиционным видам коммуникации добавились новые медиа и новые виды коммуникации – блоги, социальные сети, мобильные телефоны, планшеты. Теперь человек может получать новости в любом месте и в любое время. Все это ведет к созданию новых медиаплощадок или развитию уже существующих [1].

В связи с вышесказанным актуальной стала задача разработки веб-приложения для размещения контента, на котором было бы возможно: размещать, просматривать, оценивать и искать нужный материал. Для реализации площадки размещения и просмотра контента был выбран видеохостинг.

# 1 Аналитический обзор

## **1.1 Описание работы видеохостинга. Формирование функциональных требований к информационной системе хранения и воспроизведения видеоматериала**

**Видеохостинг** — веб-сервис, позволяющий загружать и просматривать видео в браузере, например, через специальный проигрыватель. При этом большинство подобных сервисов не предоставляют видео, следуя таким образом принципу «контент генерирует пользователь». Видеохостинг стал набирать популярность вместе с распространением широкополосного доступа в Интернет и развитием жёстких дисков (на которых стало возможно долговременно хранить громадные объёмы информации) [1, 2].

Широкое применение видеохостинги получили в распространении требуемого контента, рекламе продукции, а также для влияния на общественное мнения [1].

Самым популярной платформой для размещения видеоматериала является открытый в 2005 году YouTube. Благодаря интуитивно понятному интерфейсу, а также технологии Flash Video, позволяющей получить относительно хорошее качество записи при небольшом объёме передаваемых данных, он быстро получил своё распространение [2].

В нашей стране существуют собственные видеохостинги, самыми известными из которых являются Rutube и Яндекс.Дзен. К сожалению, они не получили такого большого распространения, как YouTube [2].

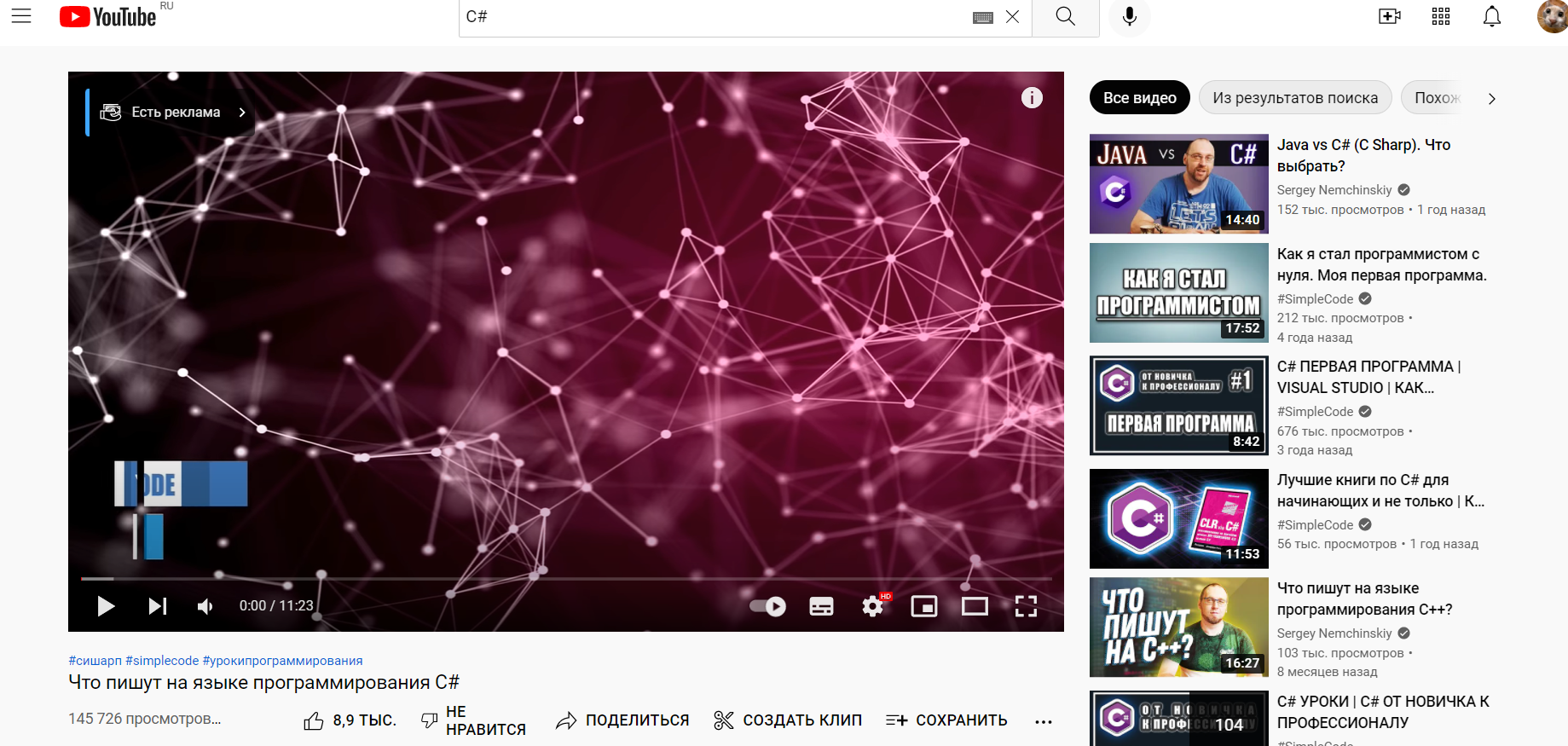


Рисунок 1 – Пример внешнего вида главной страницы Youtube

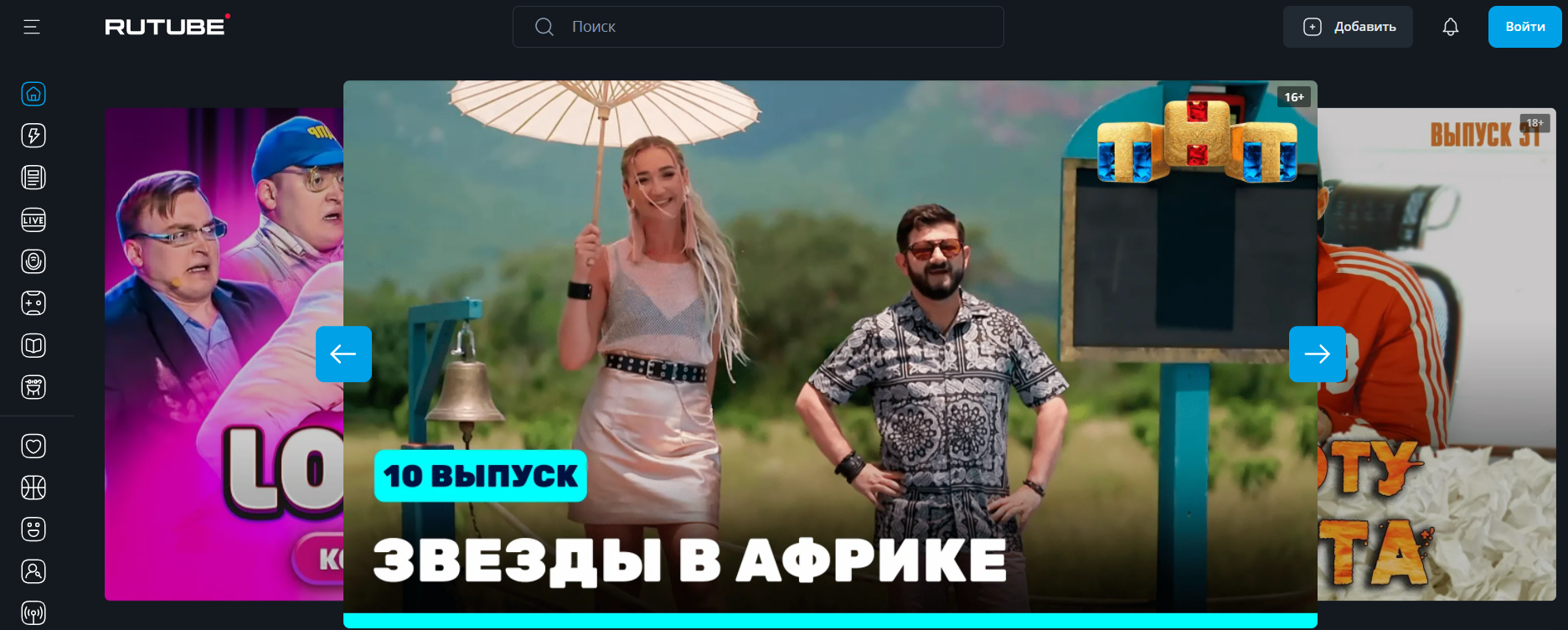


Рисунок 2 – Пример внешнего вида главной страницы Rutube

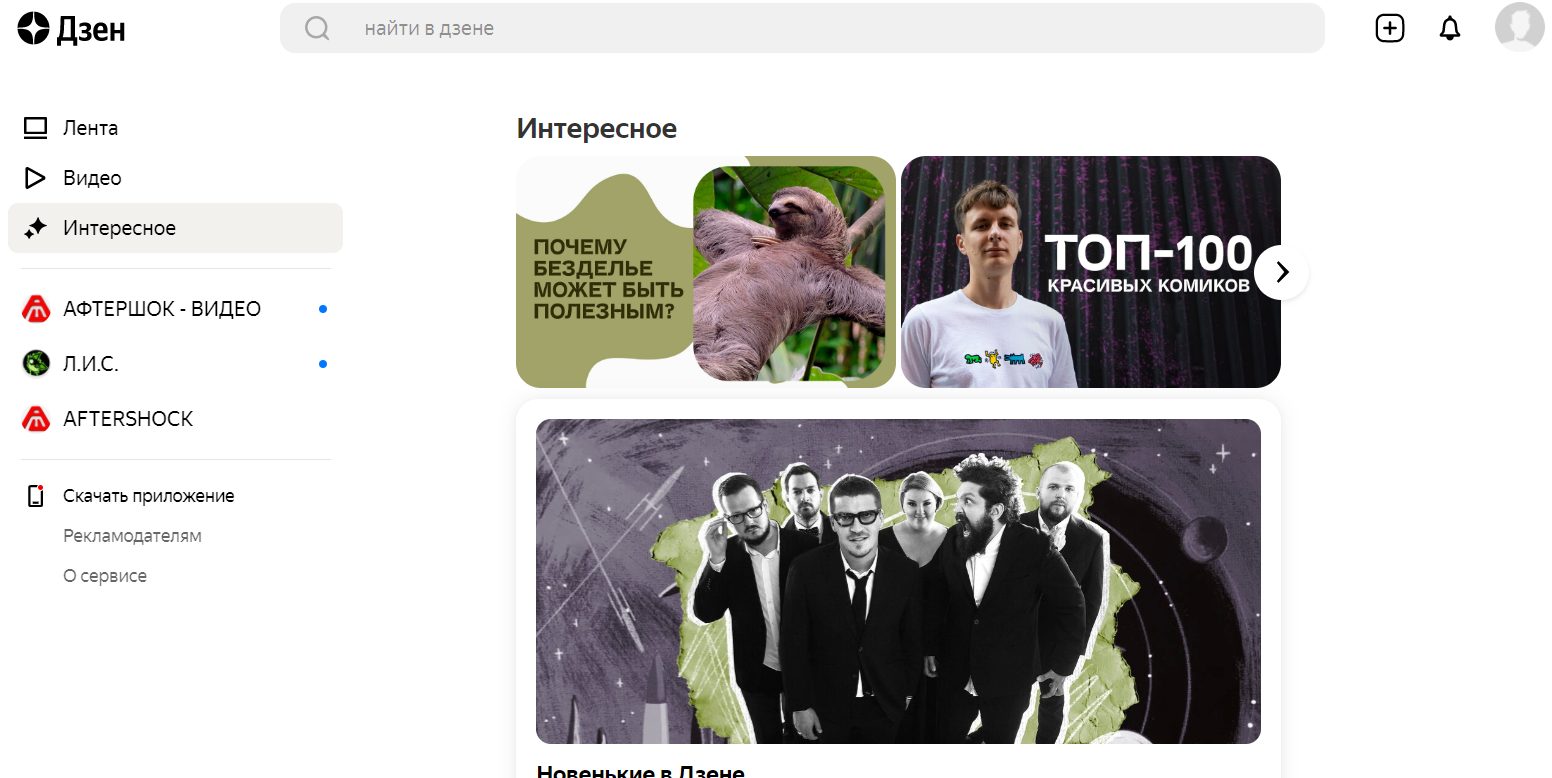


Рисунок 3 – Пример внешнего вида главной страницы Яндекс.Дзен

Разрабатываемое web-приложение позволяет хранить и просматривать видеоматериал. В приложении проводиться оценка и поиск загруженных видео. По результатам оценок пользователь сможет сделать выводы об популярности того или иного видео.

## **1.2 Обзор и обоснование выбора технологий вёрстки макетов веб-приложения**

Сейчас часто выбирают создание сайтов/веб-приложений c использованием CMS систем, чаще всего ими пользуются те, кто не хочет погружаться в веб-разработку. Кроме очевидных плюсов, таких как скорость создания, простота, доступность веб-разработки, способность редактирования контента без нарушения кода, в разработке с использованием CMS систем есть и существенные минусы, а именно: однотипность работ, что затрудняет возможность создания индивидуального контента, низкая безопасность, большой вес, медленная загрузка.

В сравнении с версткой CMS систем, страницы HTML имеют меньший вес, большую безопасность, за счет того, что нет единой базы данных, как в случае CMS систем. Что касается индивидуальности контента, разумеется, при выборе верстки вручную, эта проблема также становится не актуальной, так как возможности при верстке ограничены лишь фантазией и знаниями языка разметки.

HTML – это язык разметки гипертекста. Он позволяет с помощью специальных тегов создавать и структурировать разделы для web-страниц и приложений.

CSS – это язык, используемый в качестве средства оформления веб-страниц. Он используется для работы с оформлением шрифтов, цветовым оформлением, оформлением полей, таблиц, картинок и для расположения элементов на странице в целом.

К плюсам CSS можно отнести

– Возможность различного оформления одной веб-страницы для разных устройств,

– Возможность создания уникальных страниц,

– Простоту изменения дизайна страницы,

– Повышение совместимости с разными платформами за счет использования web-стандартов.

К недостаткам можно отнести различающееся отображение верстки в разных браузерах, что происходит из-за различного восприятия функций CSS браузерами.

Для размещения содержимого сайта в определенной последовательности, т.е. вёрстки, используются следующие методы [4]: нормальный поток; flexbox; grid layout; float; позиционирование; таблицы; многоколоночный макет.

Нормальный поток (Normal flow) – это отображение по умолчанию в порядке написания разметки сверху-вниз (рисунки 4 и 5).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – HTML-код для демонстрации нормального потока

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Результат отображения HTML кода с использованием нормального потока

Flexbox (сокращение от Flexible Box Layout) это модуль, разработанный для облегчения вёрстки в одном из измерений — как ряд или как колонка. Разметка HTML, представленная на рисунке 6, состоит из элемента wrapper, включающего в себя три «div» элемента. По умолчанию все они будут изображаться как блочные, один под другим. При добавлении свойства «display: flex» родительскому элементу, три дочерних сгруппируются в колонки. Дочерние элементы становятся элементами «flex» и наследуют некоторые свойства, установленные контейнеру, в котором они находятся. Они выстраиваются в строку согласно свойству «flex-direction», значением по умолчанию которого является «row». Высота становится равной высоте самого высокого элемента, потому что начальное значение «align-items» установлено как «stretch». Элементы вытягиваются по высоте контейнера, который в этом случае сам принимает высоту самого высокого элемента. Все они группируются в начале контейнера, оставляя пустое пространство в конце строки (рисунки 6 и 7).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 – Разметка методом flexbox

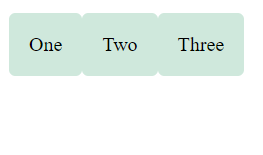


Рисунок 7 – Отображение в браузере разметки методом flexbox

Grid Layout представляет пересекающийся набор горизонтальных и вертикальных линий - один набор определяет столбцы, а другой строки. Элементы могут быть помещены в сетку, соответственно строкам и столбцам. Grid Layout можно использовать для размещения основных областей страницы или небольших элементов пользовательского интерфейса. Пример ниже (рисунки 8 и 9) использует разметку подобную примеру с flex layout, в нём определяются размеры рядов и столбцов в родительском элементе, с помощью свойств «grid-template-rows» и «grid-template-columns». Чтобы воспользоваться данным методом не требуется вводить какие-либо правила для дочерних элементов, т.к. они автоматически помещаются в ячейки, созданные родительским элементом.

Изображение выглядит как текст, стол

Автоматически созданное описание Изображение выглядит как текст, стол

Автоматически созданное описание

Рисунок 8 – Разметка при помощи grid layout

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 9 – Отображение в браузере разметки методом grid layout

Float указывает, что элемент должен быть взят из нормального потока и помещён вдоль левой или правой стороны его контейнера, где текст и встроенные элементы будут обтекать его (рисунки 10 и 11).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10 – Разметка при помощи float

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – Отображение в браузере разметки методом float

Позиционирование – способ перемещать элементы из нормального потока в другое место на странице. Позиционирование не является методом создания основной разметки страницы, его чаще используют для настройки положения определённых элементов на странице.

Существует пять типов позиционирования:

– static positioning – позиционирование по умолчанию в нормальном потоке;

– relative positioning – позиционирование позволяющее задавать положение на странице относительно нормального потока с помощью свойств «top», «right», «bottom» и «left»;

– absolute positioning полностью перемещает элемент из нормального потока разметки страницы так будто он находится на своём собственном отдельном слое, менять его положение можно с помощью свойств «top», «right», «bottom» и «left»;

– fixed positioning очень похоже на абсолютное за исключением того, что он изменяет положение относительно окна просмотра браузера, а не другого элемента;

– sticky positioning – относительно новый метод позиционирования, который заставляет элемент вести себя как position: static пока не достигнет определённой линии окна просмотра и с этого момента будет вести себя как position: fixed.

Таблица – это структурированный набор данных, состоящий из строк и столбцов (табличных данных). HTML таблицы чаще всего используются по прямому их назначению – для отображения табличных данных, но до широкой поддержки CSS браузерами, HTML таблицы использовали для размещения содержания на странице в определенном порядке. То, как таблица выглядит на веб-странице при использовании разметки таблицы, обусловлено набором свойств CSS, которые определяют макет таблицы. Эти свойства могут использоваться для размещения элементов, которые не являются таблицами, использование, которое иногда описывается как «использование CSS таблиц».

Многоколоночный макет – способ располагать содержимое в столбцах, подобно тексту в газете (рисунок 12). Чтобы превратить блок в многоколоночный контейнер используется свойство «column-count», которое определяет количество колонок, либо свойство «column-width», которое указывает браузеру заполнить контейнер как можно большим количеством столбцов, по крайней мере, такой ширины.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Отображение в браузере верстки с помощью многоколоночного макета

Нельзя выбрать какую-либо одну технологию верстки HTML-страниц: они используются вместе друг с другом. Единственной технологией из приведенного выше списка технологий верстки, от которой стоит отказаться, – это верстка при помощи HTML-таблиц. Данный способ вёрстки сильно устарел, т.к. использовался ещё до введения CSS3.

Для обеспечения возможности просмотра графического интерфейса на разных устройствах с экранами различных размеров выбран шаблон «Наиболее резиновый» (рисунок 13). Данный шаблон состоит главным образом из резиновой сетки. На экранах большой или средней ширины ее размер обычно остается неизменным, на больших экранах регулируются только поля. На небольших экранах резиновая сетка вызывает перерасчет макета для основного контента, а столбцы размещаются друг под другом [5].

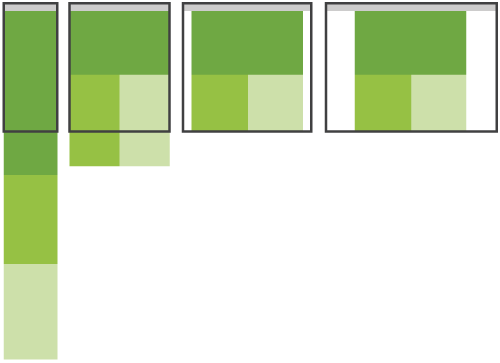


Рисунок 13 – Шаблон «Наиболее резиновый»

Таким образом, для верстки макетов графического интерфейса клиентской подсистемы веб-приложения видеохостинга необходим комплекс технологий верстки HTML-страниц. Для данной задачи выбраны несколько технологий верстки: нормальный поток, flexbox, шаблон «Наиболее резиновый», которые позволяют разработать настраиваемый макет графического интерфейса.

Также для веб-приложения видеохостинга необходим модуль для запуска и работы с видео, которым в HTML выступает тег <video>. Он добавляет, воспроизводит и управляет настройками видеоролика на веб-странице. Путь к файлу задается через атрибут src или вложенный тег <source>. Для универсального воспроизведения в разных браузерах видео кодируют с помощью разных кодеков и добавляют файлы одновременно.

Атрибутами тега <video> являются:

* autoplay – видео начинает воспроизводиться автоматически после загрузки страницы;
* controls – добавляет панель управления к видеоролику;
* height – задает высоту области для воспроизведения видеоролика;
* loop – повторяет воспроизведение видео с начала после его завершения;
* poster – указывает адрес картинки, которая будет отображаться, пока видео не доступно или не воспроизводится;
* preload – используется для загрузки видео вместе с загрузкой веб-страницы;
* src – указывает путь к воспроизводимому видеоролику;
* width – задает ширину области для воспроизведения видеоролика.

Пример реализации тега <video> в браузере Opera, можно увидеть на рисунке 14 [6].



Рисунок 14 – Пример работы тега <video> в браузере Opera

## **1.3 Обзор и обоснование выбора архитектуры информационной системы, способа клиент-серверного взаимодействия, используемой СУБД, инструментальных средств разработки**

Веб-приложение – это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер (в широком смысле). Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты (потребители этих функций). Практические реализации такой архитектуры называются клиент-серверными технологиями.

Наиболее известные подходы к архитектуре:

– Многослойная архитектура (Layered Architecture);

– Многоуровневая архитектура (Tiered Architecture);

– Сервис-ориентированная архитектура (Service Oriented Architecture — SOA);

– Микросервисная архитектура (Microservice Architecture) [10].

Многослойная архитектура работает по принципу разделения ответственностей. Программное обеспечение разделено на слои, лежащие друг на друге, и каждый из них выполняет определенную обязанность. Преимущества: более простая реализация по сравнению с другими подходами; предлагает абстракцию благодаря разделению ответственностей между уровнями; изолирование защищает одни слои от изменений других; повышает управляемость программного обеспечения за счет слабой связанности. Недостатки: не предлагает большой масштабируемости; программное обеспечение, созданное с таким подходом, будет иметь монолитную структуру, усложняющую внесение модификаций; данные должны проходить по каждому слою, даже если нет необходимости передавать их с определенных слоев.

Многоуровневая архитектура разделяет комплекс программного обеспечения на уровни по принципу взаимодействия “клиент-сервер”. Архитектура может иметь один, два и больше уровней, разделяющих ответственности между поставщиком данных и потребителем. Этот подход использует шаблон Request Response для связи между уровнями. В отличие от многослойной архитектуры, он предлагает масштабируемость, которая может быть как горизонтальной (масштабирование сети с помощью высокопроизводительных узлов), так и вертикальной (масштабирование каждого узла путем повышения его производительности).

Сервис-ориентированная архитектура состоит из компонентов и приложений, которые связываются друг с другом с помощью четко определенных сервисов. Клиент отправляет запрос с использованием стандартного протокола и формата данных по сети. Этот запрос обрабатывается ESB (enterprise service bus — сервисная шина предприятия), которая считается сердцем сервис-ориентированной архитектуры и отвечает за оркестровку и маршрутизацию. С помощью сервисного репозитория ESB направляет запрос в специальный сервис, который может взаимодействовать с другими сервисами и базами данных, чтобы составить полезную нагрузку (данные) ответа.

Mикросервисная архитектура разрабатывается как набор небольших сервисов, каждый из которых работает в собственном процессе и связывается с легковесными механизмами, обычно API для HTTP-ресурса. Преимущества:

предлагает слабую связанность благодаря высокой степени изоляции; повышает модульность; cбой в одном сервисе не затронет всю систему, поскольку они изолированы; предлагает высокую гибкость и масштабируемость; простота модификации может ускорить итерации; позволяет реализовать улучшенную систему обработки ошибок; решает проблемы с потоками данных, которые бывают у многослойной архитектуры. Недостатки: повышенный риск сбоя при обмене данными между сервисами; большим количеством сервисов трудно управлять; требует решения таких проблем, как задержки в сети, балансировка нагрузки и прочих трудностей, свойственных распределенной архитектуре; нуждается в комплексном тестировании в распределенной среде; на реализацию потребуется гораздо больше времени.

Для реализации проекта выбрана трёхуровневая архитектура приложения, представленная на рисунке 15.



Рисунок 15 – Схема трехуровневой архитектура

Хранение данных будет осуществляться на стороне севера в серверной системе управления базами данных – это снизит требования к клиентскому оборудованию (требуется меньший объем накопителя).

Обработка запросов на получение статических файлов и данных из системы управления базами данных будет производиться на сервере.

Клиент и сервер осуществляют запросы посредством протокола HTTP. В целях обеспечения большей безопасности рекомендуется использовать расширение протокола HTTPS для поддержки шифрования данных.

Далее, необходимо выбрать систему управления базами данных. Рассмотрим три популярных представителя реализации СУБД, результаты приведены в таблице 1.

Таблица 1 – Сравнение систем управления базами данных MySQL, MongoDB, SQLite

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | MySQL | MongoDB | SQLite |
| Краткое описание | Широко используемая свободная [реляционная система управления базами данных](https://ru.bmstu.wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) | Одно из наиболее популярных документных хранилищ | Быстрая и легкая встраиваемая однофайловая СУБД на языке C, которая не имеет сервера и позволяет хранить всю базу локально на одном устройстве. Для работы SQLite не нужны сторонние библиотеки или службы. |
| Основная модель хранения данных | Реляционная база данных | Документоориентированная база данных | Реляционная база данных |
| Схема данных | Да | Свободная | Да |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | MySQL | MongoDB | SQLite |
| Типизация | Да | Да | Да |
| Триггеры | Да | Нет | Да |
| Методы разбиения | Горизонтальное разбиение, шардинг с MySQL Cluster или MySQL Fabric | Шардинг | При разбиении базы данных SQLITE естественным способом является создание отдельной базы данных для каждого X, где X-это некоторый элемент в системе, представляющий границу запроса, которую вам не нужно пересекать |
| SQL | Да | Нет | Да |
| Концепции согласования | Немедленное согласование | Согласованность в конечном счёте, Немедленное согласование | Немедленное согласование |
| Внешние ключи | Да | Нет | Да |
| Параллелизм | Да | Да | Да |

Для реализации в проекте хранения данных о видео выбрана СУБД SQLite обладающая большим спектром преимуществ:

**Высокая скорость.** Благодаря особенностям архитектуры SQLite работает быстро, особенно на чтение. Компоненты СУБД встроены в приложение и вызываются в том же процессе. Поэтому доступ к ним быстрее, чем при взаимодействии между разными процессами.

**Хранение данных в одном файле.** База данных состоит из табличных записей, связей между ними, индексов и других компонентов. В SQLite они хранятся в едином файле (database file), который находится на том же устройстве, что и программа. Чтобы при работе не возникало ошибок, файл блокируется для сторонних процессов перед записью. Раньше это приводило к тому, что записывать данные в базу мог только один процесс единовременно. Но в новых версиях это решается перенастройкой режима работы СУБД.

**Минимализм.** Создатели SQLite пользуются принципом «минимального полного набора». Из всех возможностей SQL в ней есть наиболее нужные. Поэтому SQLite отличают малый размер, простота решений и легкость администрирования. Для повышения базовой функциональности можно использовать стороннее программное обеспечение и расширения.

**Надежность.** Код на 100% покрыт тестами. Это означает, что протестирован каждый компонент ПО. Поэтому SQLite считается надежной СУБД с минимальным риском непредсказуемого поведения.

**Нулевая конфигурация.** Перед использованием СУБД не нужна сложная настройка или длительная установка. Для решения большинства задач ей можно пользоваться «из коробки», без установки дополнительных компонентов.

**Малый размер.** Полностью сконфигурированный SQLite со всеми настройками занимает меньше 400 Кб. Если использовать СУБД без дополнительных компонентов, размер можно уменьшить до 250 Кб. Он зависит только от количества загруженной информации. Несмотря на малый размер, SQLite поддерживает большинство функций стандарта SQL2 и имеет ряд собственных.

**Доступность.**SQLite находится в публичном доступе. На ее использование нет правовых ограничений, а владельцем считается общество. Можно открывать, просматривать и изменять исходный код установленного ПО.

**Кроссплатформенность.** СУБД подходит для UNIX-подобных систем, MacOS и Windows.

**Автономность.** Система независима от стороннего ПО, библиотек или фреймворков. Чтобы приложение с базой на SQLite работало, дополнительные компоненты не требуются. Также не обязателен доступ в интернет: вся база хранится на устройстве, получить данные можно локально [11, 12].

## **1.4 Выводы по аналитическому обзору**

В ходе проведения аналитического обзора работы видеохостинга было принято решение в веб-приложении учитывать такие характеристики, как название видео, его автор, описание, а также его количественное оценивание (скольким пользователям понравился контент и скольким не понравился).

В ходе обзора средств разработки принято решение об использовании языков HTML и CSS для оформления страниц, языка программирования JavaScript совместно с C#, включающую в себя библиотеку NudeGet – System.Data.SQLite для работы с базой данных. Разработку приложения было решено проводить в среде Visual Studio 2019, а разработку базы данных – в СУБД SQLite с использованием инструмента для визуального проектирования баз данных SQLite SQLiteStudio.

# 2 Цель и задачи работы

Целью курсового проекта является разработка видеохостинга, который облегчит распространение контента обычным пользователям.

Для достижения цели необходимо выполнить следующие задачи:

– разработать функциональную структуру веб-приложения с учётом клиент-серверной архитектуры и сформулировать задачу разработки веб-приложения;

– разработать UML-диаграммы вариантов использования веб-приложения для пользователя;

– разработать макеты графического интерфейса веб-приложения;

– разработать блок-схему алгоритма загрузки видео на сервер видеохостинга;

– спроектировать и описать серверный интерфейс взаимодействия с клиентской подсистемой и СУБД;

– разработать информационное обеспечение веб-приложения;

– произвести тестирование веб-приложения на примере возможности подключения к серверу и просмотра различных видео на нём.

# 3 Технология разработки программного комплекса

## **3.1 Разработка функциональной структуры хранения и использования видеоматериала с учётом клиент-серверной архитектуры. Постановка задачи разработки**

Веб-приложение включает в себя следующие модули:

– Модуль загрузки данных;

– Модули взаимодействия между с собой серверной и клиентской подсистем;

– Модуль выбора данных о видео из базы данных;

– Модуль взаимодействия с базой данных;

– Модуль вывода требуемой страницы;

– Модуль проверки параметров адресной строки;

– Модуль поиска видео;

– Модуль вывода результатов поиска видео;

– Модуль проверки статических файлов веб-приложения;

– Модуль отдачи статических файлов веб-приложения;

– БД с параметрами видео;

– Интерфейс пользователя веб-приложения.

Функциональная структура приложения изображена на рисунке 16.



Рисунок 16 – Функциональная структура приложения

Постановку задачи разработки:

Требуется разработать веб-приложение, позволяющее пользователю:

– загружать в приложение данные о видео;

– поиск видео;

– воспроизведение видео.

## **3.2 Разработка UML-диаграммы вариантов использования веб-приложения для пользователя**

Пользователем веб-приложения является случайный человек, открывший страницу веб-приложения. UML-диаграмма вариантов использования для пользователя представлена на рисунке 17.

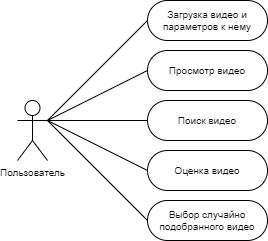


Рисунок 17 – UML-диаграмма вариантов использования для пользователя веб-приложения

При использовании приложения пользователь может:

– загрузить своё видео и параметры к нему на сервер;

– просмотреть выбранное видео;

– искать нужное видео на сайте;

– оценивать видео;

– видеть случайно подобранные 10 видео.

## **3.3 Разработка макета графического интерфейса веб-приложения для видеохостинга**

Разрабатываемое веб-приложение состоит из двух страниц, в обоих можно открыть дополнительное окно «Загрузка видео»:

1. основная страница с просматриваемым видео;
2. страница результата поискового запроса.

На рисунке 18 представлен схематичный макет основной страницы веб-приложения пользовательского интерфейса при.



Рисунок 18 – Содержимое основной страницы

На рисунке 19 представлен схематичный макет страницы результата поиска в веб-приложении.



Рисунок 19 – Содержимое страницы поиска видео в веб-приложении

На рисунке 20 представлен схематичный макет поля, открывающегося поверх остальных окон при нажатии на кнопку «Загрузка видео».

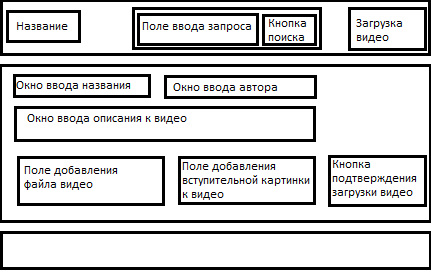


Рисунок 20 – Содержание поля для загрузки видео в веб приложение

На рисунке 21 представлена структура меню веб-приложения.



Рисунок 21 – Структура меню веб-приложения

## **3.4 Разработка блок-схемы обобщенного алгоритма функционирования приложения**

На рисунках 22 представлена блок-схема алгоритма загрузки видео на сайт.

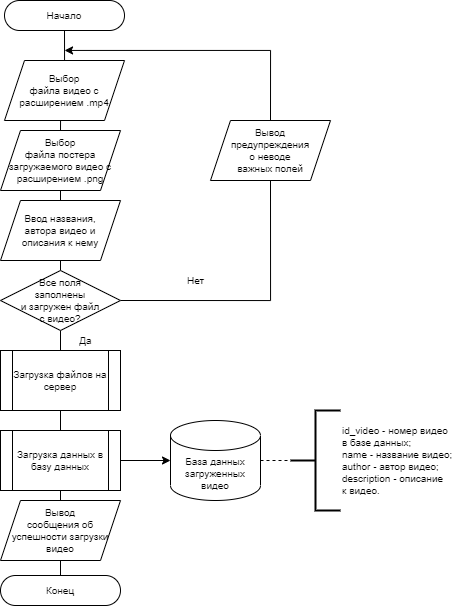


Рисунок 22 – Блок-схемы алгоритма загрузки видео на сайт

На рисунках 23 представлена блок-схема алгоритма подстановки найденного видео по, переданному параметру ‘video\_id’.

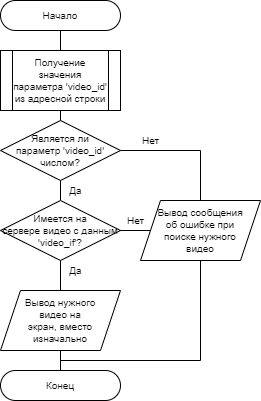


Рисунок 23 – Блок-схемы алгоритма подстановки видео на главной странице

На рисунке 24 представлена блок-схема алгоритма нативной реализации отправки объекта клиенту.

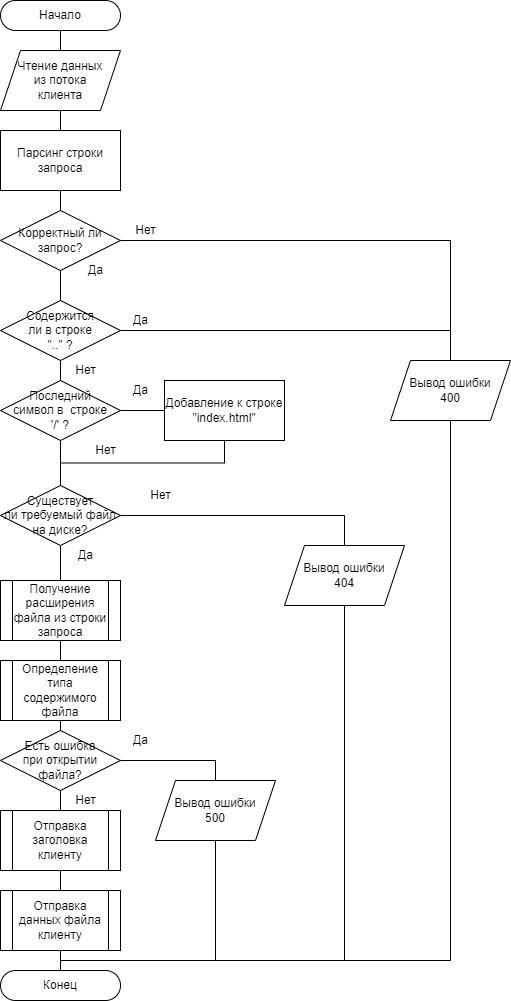


Рисунок 24 – Блок-схема алгоритма нативной реализации отправки объекта клиенту

## **3.5 Проектирование серверного интерфейса взаимодействия с клиентом и СУБД**

В таблице 2 представлена основная часть реализации клиент-серверного взаимодействия – нативная реализация отправки объекта клиенту.

Таблица 2 – Реализация создания веб-сервера на С#

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | using System;  using System.IO;  using System.Text;  using System.Net.Sockets;  using System.Text.RegularExpressions;  using System.Net;  namespace ServerVideoDom  {  class Client  {  public Client(TcpClient Client)  {  string Request = "";  byte[] Buffer = new byte[1024];  int Count;  while ((Count = Client.GetStream().Read(Buffer, 0, Buffer.Length)) > 0)  {  Request += Encoding.ASCII.GetString(Buffer, 0, Count);  if (Request.IndexOf("\r\n\r\n") >= 0 || Request.Length > 4096)  {  break;  }  }  Match ReqMatch = Regex.Match(Request, @"^\w+\s+([^\s\?]+)[^\s]\*\s+HTTP/.\*|");  if (ReqMatch == Match.Empty)  {  SendError(Client, 400);  return;  }  string RequestUri = ReqMatch.Groups[1].Value;  RequestUri = Uri.UnescapeDataString(RequestUri);  if (RequestUri.IndexOf("..") >= 0)  {  SendError(Client, 400);  return;  }  if (RequestUri.EndsWith("/"))  {  RequestUri += "index.html";  }  string FilePath = "www/" + RequestUri; |

Продолжение таблицы 2

|  |  |
| --- | --- |
| 45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94 | if (!File.Exists(FilePath))  {  SendError(Client, 404);  return;  }  string Extension = RequestUri.Substring(RequestUri.LastIndexOf('.'));  string ContentType = "";  switch (Extension)  {  case ".htm":  case ".html":  ContentType = "text/html";  break;  case ".css":  ContentType = "text/css";  break;  case ".js":  ContentType = "text/javascript";  break;  case ".jpg":  ContentType = "image/jpeg";  break;  case ".mp4":  ContentType = "video/\*";  break;  case ".jpeg":  case ".png":  case ".gif":  ContentType = "image/" + Extension.Substring(1);  break;  default:  if (Extension.Length > 1)  {  ContentType = "application/" + Extension.Substring(1);  }  else  {  ContentType = "application/unknown";  }  break;  }  FileStream FS;  try  {  FS = new FileStream(FilePath, FileMode.Open, FileAccess.Read, FileShare.Read);  }  catch (Exception)  { |

Продолжение таблицы 2

|  |  |
| --- | --- |
| 95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134 | SendError(Client, 500);  return;  }  string Headers = "HTTP/1.1 200 OK\nContent-Type: " + ContentType + "\nContent-Length: " + FS.Length + "\n\n";  byte[] HeadersBuffer = Encoding.ASCII.GetBytes(Headers);  Client.GetStream().Write(HeadersBuffer, 0, HeadersBuffer.Length);  while (FS.Position < FS.Length && Client.Connected)  {  Count = FS.Read(Buffer, 0, Buffer.Length);  try  {  Client.GetStream().Write(Buffer, 0, Count);  Client.GetStream().Flush();  }  catch  {  break;  }  }  FS.Close();  Client.Close();  }  private void SendError(TcpClient Client, int Code)  {  string CodeStr = Code.ToString() + " " + ((HttpStatusCode)Code).ToString();  string Html = "<html><body><h1>" + CodeStr + "</h1></body></html>";  string Str = "HTTP/1.1 " + CodeStr + "\nContent-type: text/html\nContent-Length:" + Html.Length.ToString() + "\n\n" + Html;  byte[] Buffer = Encoding.ASCII.GetBytes(Str);  Client.GetStream().Write(Buffer, 0, Buffer.Length);  Client.Close();  }  }  } |

## **3.6 Разработка базы данных веб-приложения для видеохостинга**

Разработка реляционной базы данных проводилась в инструменте для визуального проектирования баз данных SQLite SQLiteStudio.

Информационное обеспечение состоит из базы данных BDVideos.db (параметры видео). Внутри базы данных есть одна таблица DownloadVideos, внутри которых хранятся параметры видео. Одна строка таблицы содержит информацию о номере, названии, авторе, описании, лайках, дизлайках одного видео.

Структура таблицы представлена в таблице 3.

Таблица 3 – Структура таблицы DownloadVideos

|  |  |  |
| --- | --- | --- |
| Имя | Тип данных | Значение |
| id | INTEGER | Уникальный идентификатор видео |
| name | VARCHAR(45) | Название видеофайла |
| autor | VARCHAR(30) | Автор видео |
| description | TEXT | Описание к видео |
| like | INTEGER | Количество людей, которым понравилось видео |
| dislike | INTEGER | Количество людей, которым не понравилось видео |

Для взаимодействия с выбранной СУБД на серверной стороне используется официальный пакет NuGet System.Data.SQLite. В таблице 4 представлены классы, реализованные для работы с СУБД SQLite. В таблице 5 представлен код на C# разработанного класса (DownloadVideo) для работы с объектами СУБД SQLite.

Таблица 4 – Классы для работы с SQLite

|  |  |
| --- | --- |
| Название класса | Описание класса |
| DownloadVideo | Класс представляющий объекты базы данных BDVideos.db |
| APContext | Подключение к базе данных и получение из него объектов класса DownloadVideo |

Таблица 5 – Реализация класса DownloadVideo на C#

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | class DownloadVideo  {  public int id { get; set; }  private string name, autor, description;  private int like, dislike;  public string Name  {  get { return name; }  set { name = value; }  }  public string Autor  {  get { return autor; }  set { autor = value; }  }  public string Description  {  get { return description; } |

Продолжение таблицы 5

|  |  |
| --- | --- |
| 19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | set { description = value; }  }  public int Like  {  get { return like; }  set { like = value; }  }  public int Dislike  {  get { return dislike; }  set { dislike = value; }  }  public DownloadVideo() { }  public DownloadVideo(string name, string autor, string description, int like, int dislike)  {  this.name = name;  this.autor = autor;  this.description = description;  this.like = like;  this.dislike = dislike;  }  public string GetName()  {  return name;  }  } |

## **3.7 Разработка программного обеспечения веб-приложения для видеохостинга**

Веб-приложение разрабатывается в инструментальной среде разработки Visual Studio Code на языках JavaScript, CSS и C#.

База данных параметров хранимых на сервере видео разработана в СУБД SQLite SQLiteStudio на языке SQLite.

Для отправки SQL запросов используется библиотека языка C# – System.Data.SQLite, которая представляет возможность более быстрого взаимодействия с выбранной СУБД.

Разрабатываемое веб-приложение можно использовать на устройствах, оснащенных графическим интерфейсом, имеющих возможность выхода в интернет с ОС Windows 7 и новее, MacOS X и новее, Ubuntu 14.04 (64-разрядная версия) и новее, Fedora Linux 24 и новее. Структура программного обеспечения для видеохостинга представлена на рисунке 25.

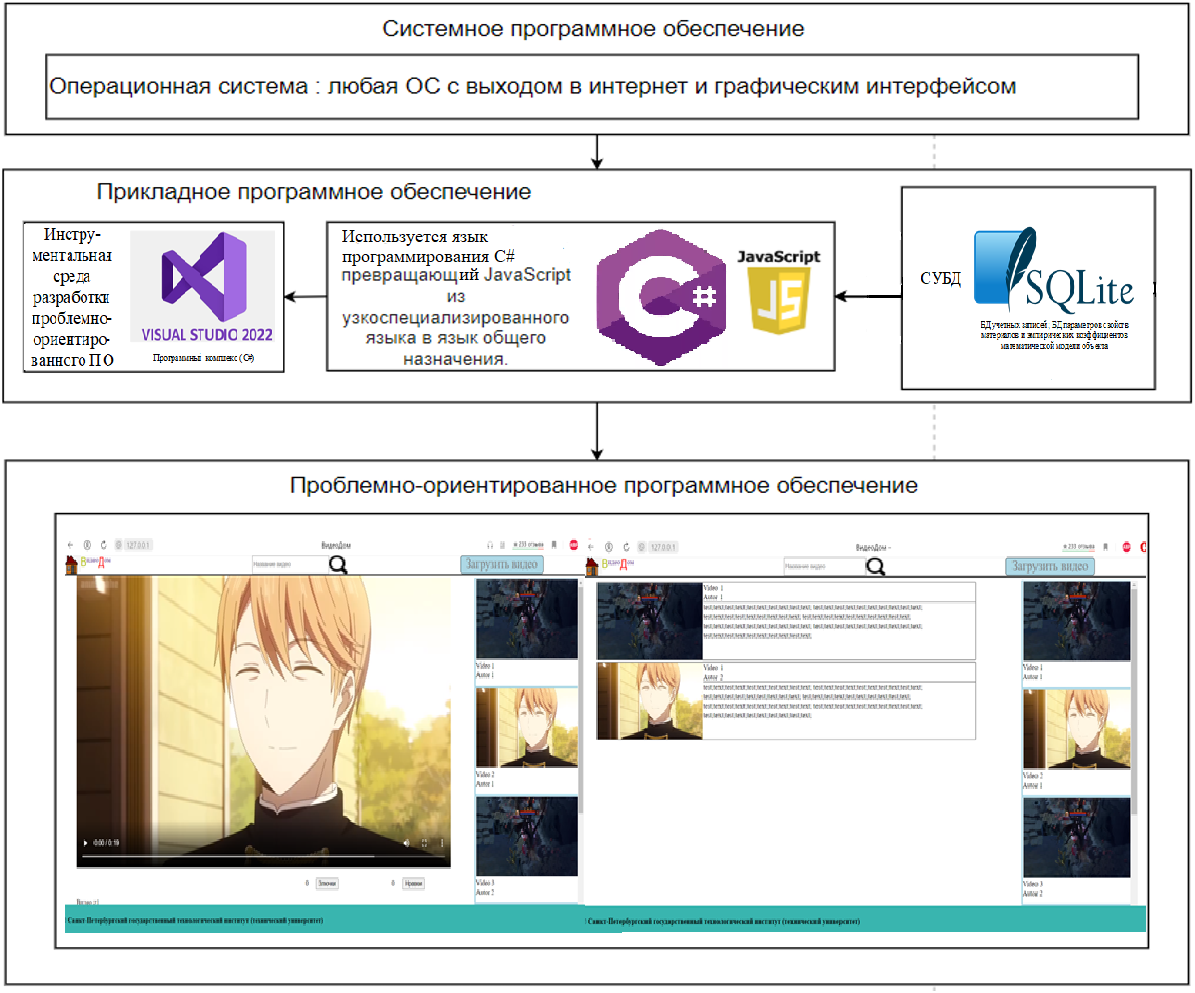


Рисунок 25 – Структура программного обеспечения

В таблице 6 представлен пример кода на JS реализующий подстановку нужного видео на главной странице по параметру ‘video\_id’.

Таблица 6 – Пример кода на JS для реализации подстановки нужного видео на главной странице

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | window.addEventListener('load', (event) => {  var p\_url = location.search.substring(1);  var parametr = p\_url.split("&");  var values = new Array();  for(i in parametr) {  var j=parametr[i].split("=");  values[j[0]]=unescape(j[1]);  }  let main\_video\_block = document.getElementsByClassName('main-video-block')[0];  if('video\_id' in values){  InsertMainVideo();  let video\_data;  //video\_data = loadBytes('/video/'+value['video\_id']+'.mp4',InsertMainVideo());  }  function InsertMainVideo(){ |

Продолжение таблицы 6

|  |  |
| --- | --- |
| 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | document.getElementById('main-video').getElementsByTagName('source')[0].src = 'video/'+values['video\_id']+'.mp4';  // Работа с БД - изменение названия, автора, количества лайков/дизлайков/просмотров видео  document.getElementById('main-video').load();  let like\_number = document.getElementById('quantity-like');  let dislike\_number = document.getElementById('quantity-dislike');  let name\_video = document.getElementById('name-video-string');  let autor\_video = main\_video\_block.getElementsByTagName('p')[2];  let text\_video = main\_video\_block.getElementsByTagName('p')[3];  name\_video.innerHTML = 'Видео ' + values['video\_id'];  autor\_video.innerHTML = 'Autor ' + values['video\_id'];  text\_video.innerHTML = values['video\_id'] + ' ' + values['video\_id'] + ' ' + values['video\_id'] + ' ' + values['video\_id'] + ' ';  }  }); |

В таблице 7 представлен пример кода на JS позволяющий осуществлять поиск видео по сайту.

Таблица 7 – Пример части кода JS для реализации поиска видео

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | window.addEventListener('load', (event) => {  let search\_loop = document.getElementById('search-loop');  search\_loop.onclick = function(){  SearchVideo();  }  function SearchVideo(){  let value\_searching\_input = document.getElementById('searching-video').value;  if( value\_searching\_input != undefined && value\_searching\_input != null && value\_searching\_input != "" ){  document.location.href ="result\_search.html?name\_video="+value\_searching\_input;  }  }  }); |

# 4 Тестирование веб-приложения

На рисунке 26 представлена командная строка, появляющаяся при запуске сервера.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 26 – Пустая командная строка сервера

На рисунке 27 и 28 представлены интерфейсы главной страницы и страницы поиска видео.

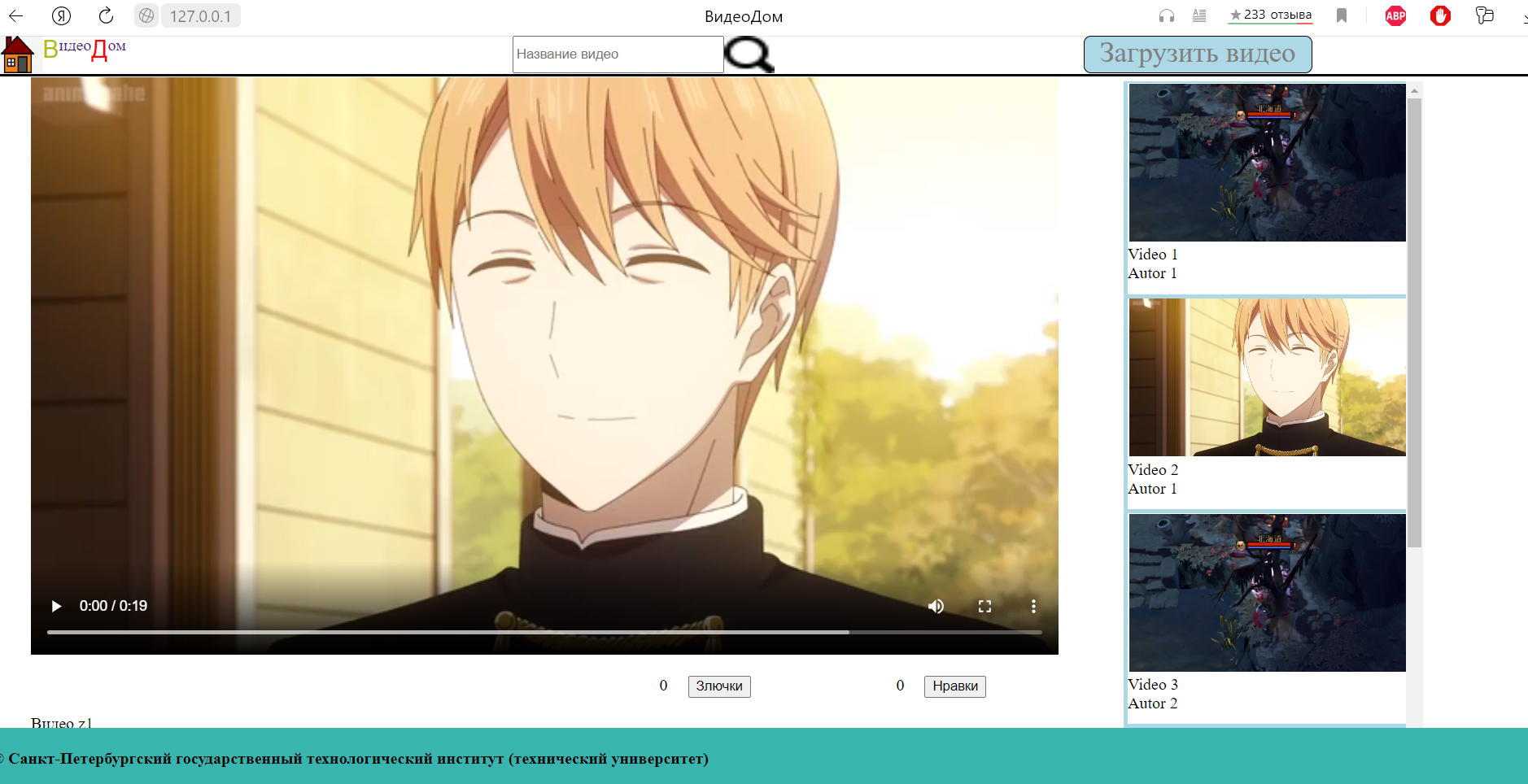


Рисунок 27 – Интерфейс главной страницы

Изображение выглядит как текст, стол

Автоматически созданное описание

Рисунок 28 – Интерфейс страницы поиска видео

Для тестирования приложения выберем случайное видео из подборки, тогда на главной странице должно измениться видео (рисунок 27).

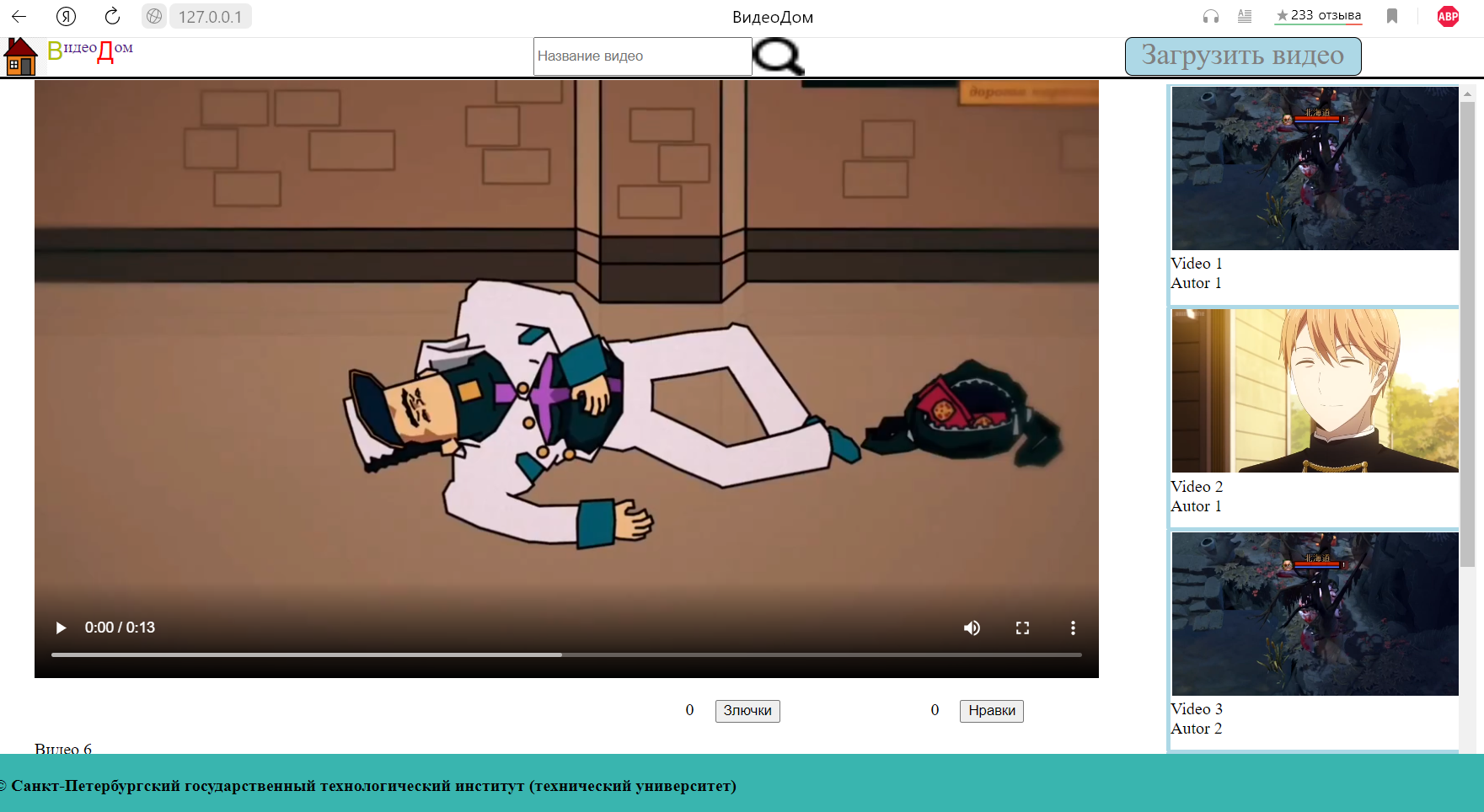


Рисунок 29 – Результат тестирования выбора видео из подборки

На рисунке 30 показан результат проверки обработки сервером адресного запроса к несуществующему на нем файлу.



Рисунок 30 – Результат обращения к несуществующему на сервере файлу

Результаты работы веб-приложения совпадают с поставленными задачами, из этого можно сделать вывод, что работа сервера и клиента проводятся корректно с обработкой возможных ошибок.

# 5 Выводы по проекту

В ходе выполнения курсовой работы было разработано веб-приложение, позволяющее загружать и просматривать видеоматериал, для чего было выполнено следующее:

– На основании аналитического обзора разработана функциональная структура веб-приложения, включающего базу данных основных параметров видео, модуль загрузки данных, модуль взаимодействия с базой данных, модуль проверки параметров в адресной строке, модуль поиска видео, модуль вывода результатов поиска, модуль вывода требуемой страницы, интерфейс пользователя видеохостинга;

– Сформулирована задачи разработки веб-приложения для хранения и просмотра видео;

– Разработана структура интерфейса пользователя;

– Разработаны макеты графического интерфейса и структура меню веб-приложения;

– Описан алгоритм решения задачи загрузки видео, его просмотра, а также нативной реализации отправки объекта клиенту;

– Спроектирован и описан серверный интерфейс взаимодействия с клиентской подсистемой и СУБД SQLite;

– Разработана информационная модель, по которой создано информационное обеспечение веб-приложения, а именно база данных параметров видео;

– Разработано программное обеспечение для решения задачи загрузки и просмотра видеоматериала, тестирование которого было произведено на основании данных заранее подготовленных видео.

# Список использованных источников

1 Танцырева, А. Е. Youtube как медиаплатформа для современных средств массовой информации : специальность 42.03.02 «Журналистика» : выпускная квалификационная работа / А. Е. Танцырева ; Южно-Уральский государственный университет. - Челябинск, 2019. – 52 с.

2 CalltouchBlog | Видеохостинги: обзор самых популярных площадок

: сайт. – URL: https://blog.calltouch.ru/videohostingi-obzor-i-samyh-populyarnyh-ploshhadok/ (дата обращения: 28.04.22).

3 Форум программистов и сисадминов Киберфорум : официальный сайт. – Москва, 2010. –URL: https://www.cyberforum.ru/ (дата обращения 28.04.2022).

4 Introduction to CSS layout // MDN Web Docs : сайт. – URL: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\_layout/Introduction/ (дата обращения: 29.04.22).

5 LukeW | Multi-Device Layout Patterns : сайт. – URL: https://www.lukew.com/ff/entry.asp?1514 (дата обращения: 01.05.22).

6 htmlbook.ru | Тег <video> : сайт. – URL: http://htmlbook.ru/html/video (дата обращения: 01.05.22)

7 learn JavaScript.ru : электронный учебник : сайт. – Сан-Франциско, 2014 – . – URL: https://learn.javascript.ru/ (дата обращения: 15.05.22).

8 JavaScript.com : электронный ресурс : сайт. – Сан-Франциско, 2014 – . – URL: https://www.javascript.com/ (дата обращения: 16.05.22).

9 Mdn web docs\_ | Windows: load event : сайт. – URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/load\_event (дата обращения: 12.05.22).

10 4 типа архитектуры программного обеспечения | by Daria Sidorova | NOP::Nuances of Programming | Medium : сайт. – Хельсинки, 2021 – . – URL: https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724 (дата обращения: 27.02.22).

11 SkillFactory | SQLite : сайт. – URL: https://blog.skillfactory.ru/glossary/sqlite/ (дата обращения: 23.05.22).

12 Хабр | SQLite — замечательная встраиваемая БД (часть 1) : сайт. – URL: https://habr.com/ru/post/149356/ (дата обращения: 24.05.22).