Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Курсовая работа**

По дисциплине: «Новые технологии в РПС»

Тема: «Разработка веб-ресурса для веб-скрапинга ресурсов с отзывами фильмов на языке программирования Python»

|  |
| --- |
| Выполнил:  студент группы  М.ПИН.РИС - 23.06  Быстров А.А |
| Проверила:  старший преподаватель  кафедры ПО  Корнеева Е.И. |

Тверь 2024

Оглавление

[**Введение** 3](#_Toc167365812)

[**Постановка задачи** 4](#_Toc167365813)

[**Цель** 4](#_Toc167365814)

[**Априорные модельные представления** 4](#_Toc167365815)

[**Ожидаемый результат** 4](#_Toc167365816)

[**Критерии оценки результата** 4](#_Toc167365817)

[**Средства проектирования и реализации** 5](#_Toc167365818)

[**Обзор предметной области** 6](#_Toc167365819)

[**Анализ существующих решений** 6](#_Toc167365820)

[**Формирование требований** 6](#_Toc167365821)

[**Проектирование прототипа** 8](#_Toc167365822)

[**Варианты использования** 8](#_Toc167365823)

[**Структура БД** 9](#_Toc167365824)

[**Обзор средств реализации** 11](#_Toc167365825)

[***Flask*** 11](#_Toc167365826)

[**Реализация** 13](#_Toc167365827)

[**Скрапинг** **отзыва** 13](#_Toc167365828)

[**Функциональное тестирование** 15](#_Toc167365829)

[**Заключение** 19](#_Toc167365830)

[**Список литературы** 21](#_Toc167365831)

[**ПРИЛОЖЕНИЕ А. Листинг кода приложения** 22](#_Toc167365832)

# **Введение**

В современном мире, где доступ к информации становится все более значимым, существует растущая потребность в эффективных инструментах для анализа общественного мнения. Одной из таких областей является анализ отзывов о фильмах, которые позволяют не только пользователям делиться своим мнением, но и предоставляют ценные данные для кинокритиков, студий и любителей кинематографа.

В рамках данной курсовой работы будет разработан веб-ресурс, который предназначен для сбора и анализа отзывов о фильмах с популярного кинематографического ресурса "КиноПоиск". Основная цель проекта заключается в создании инструмента, который позволит пользователям получать доступ к обзорам фильмов и агрегированной статистике о них.

Проект будет включать в себя механизм скрапинга данных с веб-страниц, анализ полученных отзывов и предоставление пользователю удобного интерфейса для их просмотра. В результате работы над проектом ожидается создание функционального и эффективного веб-ресурса, который поможет кинолюбителям принимать более информированные решения при выборе фильмов для просмотра.

# **Постановка задачи**

## **Цель**

Целью данной работы является создание веб-ресурса , который будет способен собирать отзывы о фильмах с кинопоиска и предоставлять пользователям удобный доступ к ним. Основной целью проекта является обеспечение пользователям возможности получения обзоров фильмов и анализа общественного мнения о них на основе собранных данных.

## **Априорные модельные представления**

Веб-ресурс позволит пользователям регистрироваться в системе и выбирать фильмы, просматривать отзывы, анализировать сентимент, скачивать отзывы и удобно навигироваться по ресурсу.

## **Ожидаемый результат**

Ожидаемым результатом разработки веб-ресурса является создание функционального инструмента, обеспечивающего пользователям следующие возможности:

* Просмотр и выбор фильмов для просмотра отзывов.
* Анализ тональности отзывов для оценки общего мнения о фильме, а также возможность скачать отзывы.
* Управление данными и настройками профиля пользователя.
* Удобная навигация и интуитивно понятный интерфейс для повышения пользовательского опыта.

## **Критерии оценки результата**

Для оценки соответствия, разработанного веб – ресурса целям проекта и ожидаемым результатам мы будем использовать следующие критерии:

1. Функциональная соответствие:

* Возможность выбора фильма для просмотра отзывов.
* Реализация скрапинга отзывов с выбранного ресурса (в данном случае - с сайта КиноПоиск).
* Автоматизированная аналитика тональности отзывов для определения общего настроения к фильму (позитивное, нейтральное, негативное).
* Предоставление пользователю интерфейса для взаимодействия с системой.
* Возможность сохранения и отображения истории просмотренных фильмов и их отзывов для зарегистрированных пользователей.
* Реализация системы аутентификации и авторизации пользователей для доступа к персонализированным функциям.
* Обеспечение безопасности данных пользователей и конфиденциальности информации, включая шифрование и защиту от несанкционированного доступа.

2.Интуитивно понятный интерфейс:

* Интерфейс приложения должен быть интуитивно понятным и удобным в использовании для пользователей всех уровней навыков.
* Навигация по приложению должна быть логичной и понятной.

## **Средства проектирования и реализации**

1. Язык программирования и фреймворк:

* Язык программирования: *Python* - для разработки скрапера отзывов.
* Фреймворк: *Flask* - для создания веб-приложения, взаимодействующего с базой данных и обрабатывающего запросы пользователей.

1. Интегрированная среда разработки (*IDE*):

* *Visual Studio Code (Code)* - популярная свободно распространяемая среда разработки, обеспечивающая удобное написание кода, отладку и управление проектом.

1. База данных:

* *SQLite* - компактная и простая в использовании реляционная база данных, которая будет хранить отзывы о фильмах и другую необходимую информацию.

# **Обзор предметной области**

## **Анализ существующих решений**

В современном мире существует множество приложений и сервисов, предоставляющих функционал по сбору данных с веб-ресурсов. Некоторые из них ориентированы на обширную аудиторию пользователей, предлагая универсальные инструменты для скрапинга данных, в то время как другие специализируются на определенных.

*Scraper API* - это сервис, который предоставляет *API* для скрапинга данных с веб-страниц. Он позволяет пользователям извлекать данные с веб-сайтов, обходя защитные меры, такие как *CAPTCHA*. *Scraper* *API* поддерживает различные типы данных, включая *HTML*, *JSON* и *XML*.

*Octoparse* - это еще один популярный инструмент для веб-скрапинга, который предлагает широкий набор функций для сбора и анализа данных с веб-сайтов. Он позволяет пользователям создавать и настраивать веб-скраперы без необходимости программирования, что делает его доступным для широкого круга пользователей.

*DataOx* - это инструмент для веб-скрапинга, который специализируется на сборе данных с онлайн-ресурсов. Он предоставляет удобный пользовательский интерфейс и широкий набор функций для извлечения данных с различных типов веб-сайтов.

Так как многие инструменты имеют платную основу, то возможность их использования сильно ограничена.

## **Формирование требований**

Функциональные требования к приложению:

1.Регистрация и авторизация:

• Реализовать механизм регистрации пользователей с использованием по логину и пароля.

• Обеспечить механизм авторизации и контроля доступа пользователей к различным функциональным возможностям приложения.

• Вывод статистики, какие отзывы смотрел пользователь.

2.Просмотр фильмов и возможность скачивания их :

• Реализовать механизм просмотра и скачивания отзывов о фильме.

• Возможность выбрать нужный фильм.

• Просмотр статистики видов отзывов.

Технические требования к приложению:

* Язык программирования и фреймворки:

• Использование языка программирования *Python*;

• Использование фреймворка *Flask* для разработки веб-приложения.

* Системные требования:

• Установленная среда разработки *VS Code* последней версии;

• Настроенный *SQLite* для хранения данных.

# **Проектирование прототипа**

## **Варианты использования**

В системе можно выделить несколько два модуля:

1.Модуль авторизации и регистрации.

2. Модуль просмотра отзывов фильма.

Для описания функциональных возможностей этих модулей были составлены диаграммы вариантов использования.

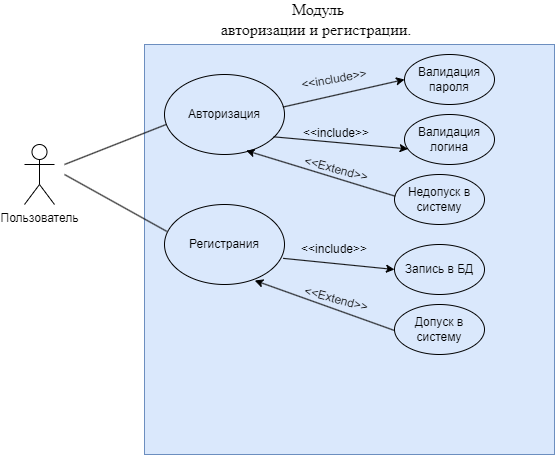


Рисунок 1. Диаграмма вариантов использования для модуля авторизации и регистрации

Для входа в приложение пользователю необходимо пройти процедуру аутентификации. Пользователь должен заполнить соответствующие поля введя логина и пароля. После входа в систему пользователю показывается начальная страница приложения, где пользователь уже может работать с основным его функционалом. Если пользователь впервые зашёл он должен зарегистрироваться. В модуле регистрации он должен записать свой логин и придумать пароль. После регистрации его данные занесутся в базу, и он автоматически попадает на начальную страницу приложения.

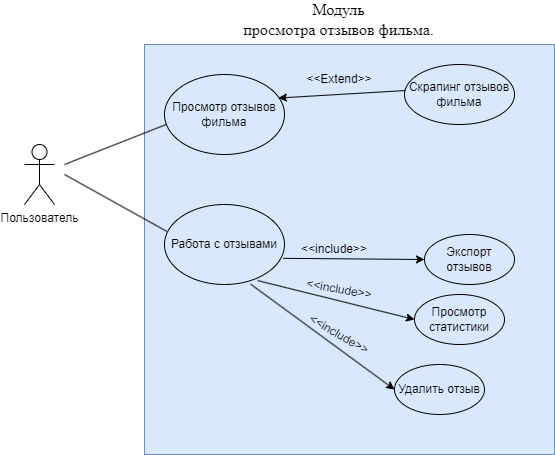


Рисунок 2. Диаграмма вариантов использования для модуля просмотра отзывов фильма

В модуле просмотра отзывов фильма, пользователь может просматриваться отзывы определенного фильма путем выбора из выпадающего списка. При выборе происходит скрапинг отзыва из КиноПоиска. Затем пользователь может посмотреть эти отзывы их статистику, а также удалить определенный отзыв.

## **Структура БД**

В качестве хранения и управления данными используется реляционная база данных *SQLite*. *SQLite* - это легковесная встроенная база данных, которая не требует отдельного серверного программного обеспечения и хорошо подходит для небольших проектов и прототипов.

Структура базы данных состоит из 4 таблиц: *films*, *reviews*, *users*, *user*\_*logs*:

1.*Users*:

* Таблица users хранит информацию о зарегистрированных пользователях.
  + *id* (*INTEGER*, *PRIMARY* *KEY* *AUTOINCREMENT*): Уникальный идентификатор пользователя.
  + username (*TEXT*, *UNIQUE*): Имя пользователя. Это поле должно быть уникальным.
  + password (*TEXT*): Хэшированный пароль пользователя.

2. *User*\_*logs*:

* Таблица users\_*logs* хранит информацию о действиях пользователей, таких как просмотр фильмов.
  + *id* (*INTEGER*, *PRIMARY* *KEY* *AUTOINCREMENT*): Уникальный идентификатор записи лога.
  + *user\_id* (*INTEGER*): Идентификатор пользователя, совершившего действие. Внешний ключ, ссылающийся на таблицу users.
  + *film*\_*name* (*TEXT*): Название фильма, который был просмотрен пользователем.
  + *timestamp* (*DATETIME*, *DEFAULT* *CURRENT\_TIMESTAMP*): Временная метка действия, автоматически устанавливается при создании записи, нужна чтобы узнать, когда пользователь просматривал отзыв.

3. *Films*:

* Таблица *films* хранит информацию о фильмах.
  + *id* (*TEXT*, *PRIMARY* *KEY*): Уникальный идентификатор фильма на КиноПоиске.
  + name (*TEXT*): Название фильма.

4. *Reviews*:

* Таблица reviews хранит информацию об отзывах на фильмы.
  + *id* (*INTEGER*, *PRIMARY* *KEY* *AUTOINCREMENT*): Уникальный идентификатор отзыва.
  + *film\_id* (*TEXT*): Идентификатор фильма, к которому относится отзыв. Это поле является внешним ключом, ссылающимся на таблицу *films*.
  + *review\_type* (*TEXT*): Тип отзыва (позитивный, нейтральный, негативный).
  + *review*\_*text* (*TEXT*): Текст отзыва.

# **Обзор средств реализации**

Для создания приложения был выбран набор технологий и инструментов, обеспечивающий эффективную разработку, удобство использования и широкие возможности для реализации задач. Одним из ключевых инструментов является фреймворк *Flask*, который предоставляет широкие возможности для создания веб-приложений на языке программирования *Python*. Для хранения данных о фильмах, отзывах и пользователях используется реляционная база данных *SQLite*, что обеспечивает надежное и удобное управление данными. Интегрированная среда разработки *Visual* *Studio* *Code* предоставляет удобные инструменты для написания кода и управления проектом, что значительно упрощает процесс разработки.

## ***Flask***

Для реализации веб-ресурса использовался фреймворк *Flask*. *Flask* является легковесным микрофреймворком на языке программирования *Python*, который обеспечивает необходимую функциональность для разработки веб-приложений. Он отличается простотой использования, гибкостью и высокой производительностью.

Особенности *Flask*:

* Легковесность: *Flask* не включает в себя ничего лишнего, что позволяет разработчикам добавлять только те компоненты, которые необходимы для их проекта.
* Гибкость: *Flask* предоставляет возможности для настройки и расширения функционала, что позволяет создавать как простые, так и сложные приложения.
* Расширяемость: *Flask* имеет систему расширений, которая позволяет добавлять дополнительные возможности, такие как аутентификация пользователей, работа с базами данных и многое другое.
* Удобство: *Flask* имеет простой и понятный синтаксис, что делает его доступным для разработчиков с различным уровнем опыта.
* Сообщество: *Flask* имеет большое и активное сообщество, что обеспечивает наличие большого количества обучающих материалов, документации и готовых решений.

В проекте я использовал следующие компоненты *Flask*:

*Flask*: Основной фреймворк, обеспечивающий базовую функциональность веб-приложения.

*render*\_*template*: Используется для рендеринга *HTML*-шаблонов с динамическими данными.

*request*: Позволяет обрабатывать данные, отправленные пользователями через веб-формы.

*redirect*: Обеспечивает перенаправление пользователей на другие страницы веб-приложения.

*send*\_*file*: Используется для отправки файлов пользователям, например, для экспорта отзывов.

*session*: Обеспечивает хранение данных сессии пользователя, таких как аутентификация.

# **Реализация**

В рамках курсовой работы, в приложении было реализовано:

* Регистрация и авторизация пользователя.
* Выбор фильма из выпадающего списка.
* Скрапинг отзывов из КиноПоиска.
* Просмотр отзывов и их статистика.
* Удаление отзывов и экспорт в отдельный файл.
* Просмотр истории, какие отзывы просматривал пользователь.

## **Скрапинг** **отзыва**

В качестве примера посмотрим, код скрапинга отзыва.

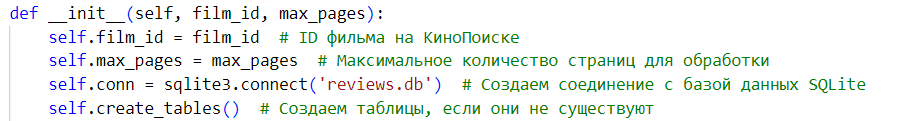


Рисунок 3. Инициализация объекта

Метод \_\_*init*\_\_ инициализирует объект *ReviewScraper*, принимая ID фильма и максимальное количество страниц отзывов, которые нужно обработать. Также устанавливается соединение с базой данных *SQLite* и создаются необходимые таблицы, если они еще не существуют.

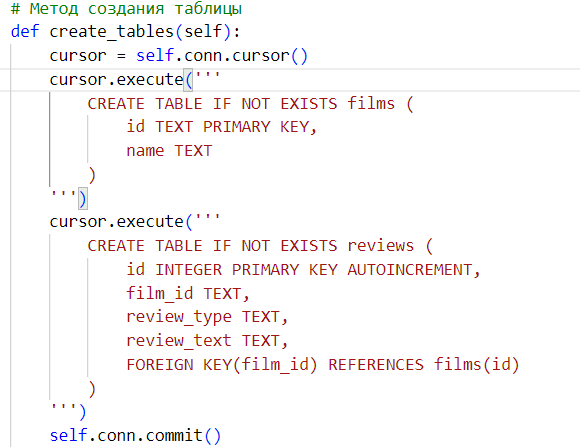
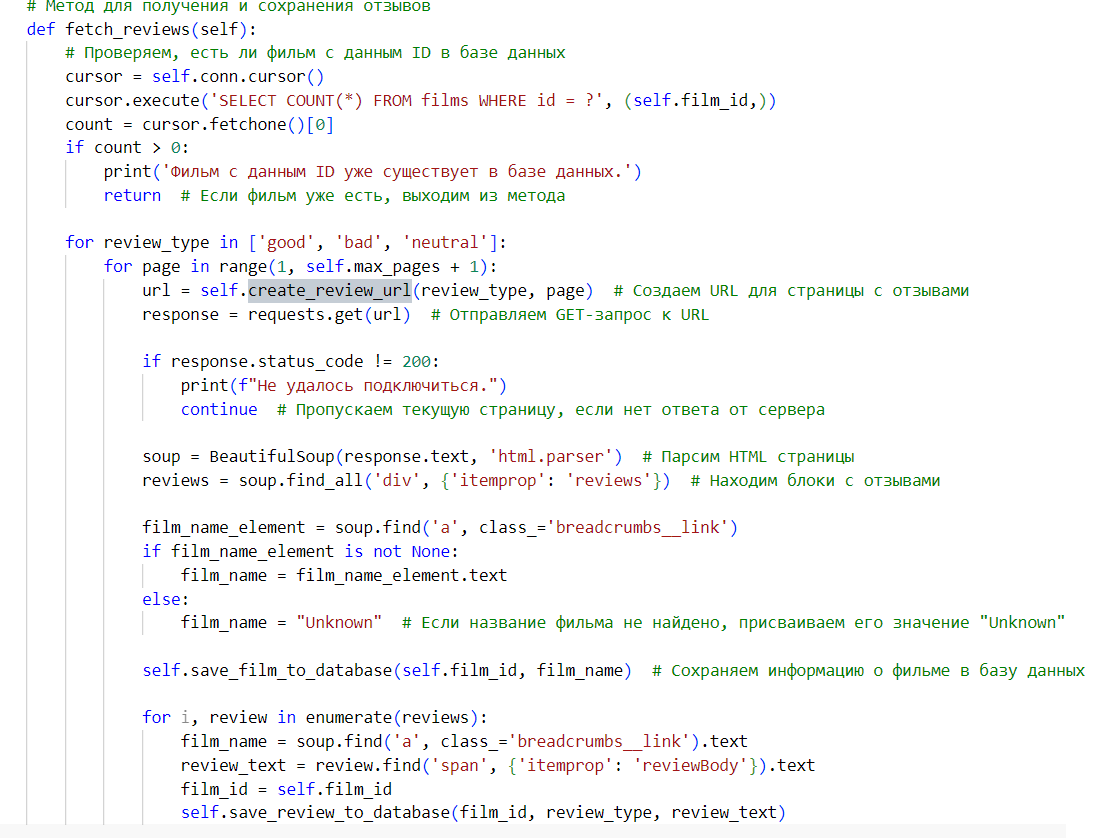


Рисунок 4. Создание таблиц

Метод *create\_table* создает две таблицы в базе данных:

*films* для хранения информации о фильмах (*ID* и название фильма).

*reviews* для хранения отзывов о фильмах (*ID* отзыва, *ID* фильма, тип отзыва, текст отзыва).  Рисунок 5. Получение и сохранение отзывов

Метод *fetch\_reviews* отвечает за получение отзывов о фильме и их сохранение в базу данных:

* Проверяется, существует ли уже фильм с данным *ID* в базе данных. Если да, метод завершает работу.
* Проходится по типам отзывов (*good*, *bad*, *neutral*) и страницам отзывов.
* Формируется *URL* для страницы с отзывами и отправляется *GET*-запрос.
* Если ответ успешный, *HTML*-страница парсится с помощью *BeautifulSoup*.
* Извлекаются блоки с отзывами и название фильма.
* Сохраняется информация о фильме в базу данных.
* Каждый отзыв извлекается и сохраняется в базу данных.

# **Функциональное тестирование**

В рамках тестирования сделаем профиль пользователя и посмотрим отзывы о фильме «Титаник» и «Аватар». Для начала зайдем на веб-страницу, где у нас появиться начальное окно (см. рис.6).

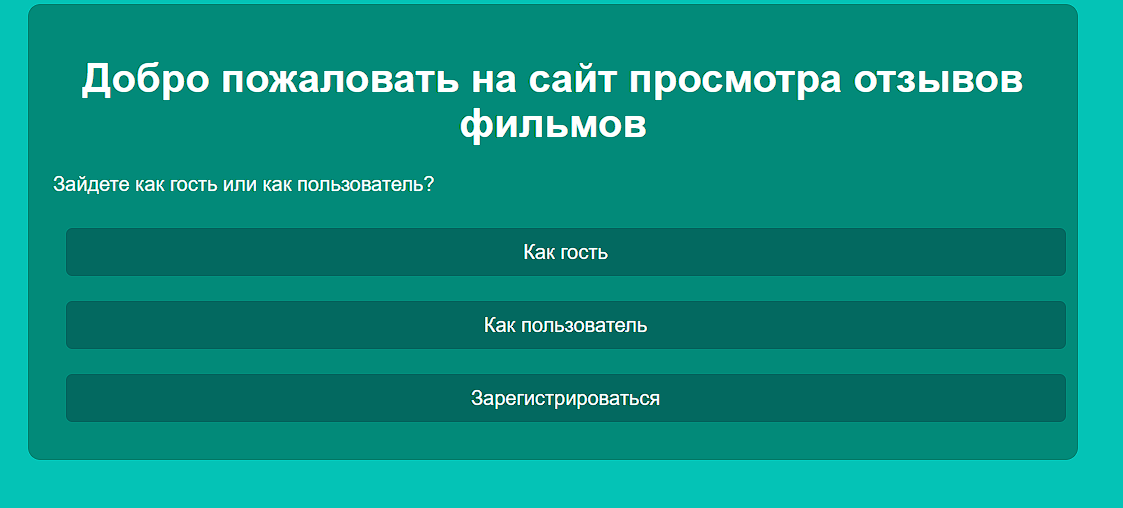


Рисунок 6. Начальное окно

Так как мы новый пользователь, то создадим новый профиль, нажмем на «Зарегистрироваться» и введем новый логин и пароль (см. рис.7).

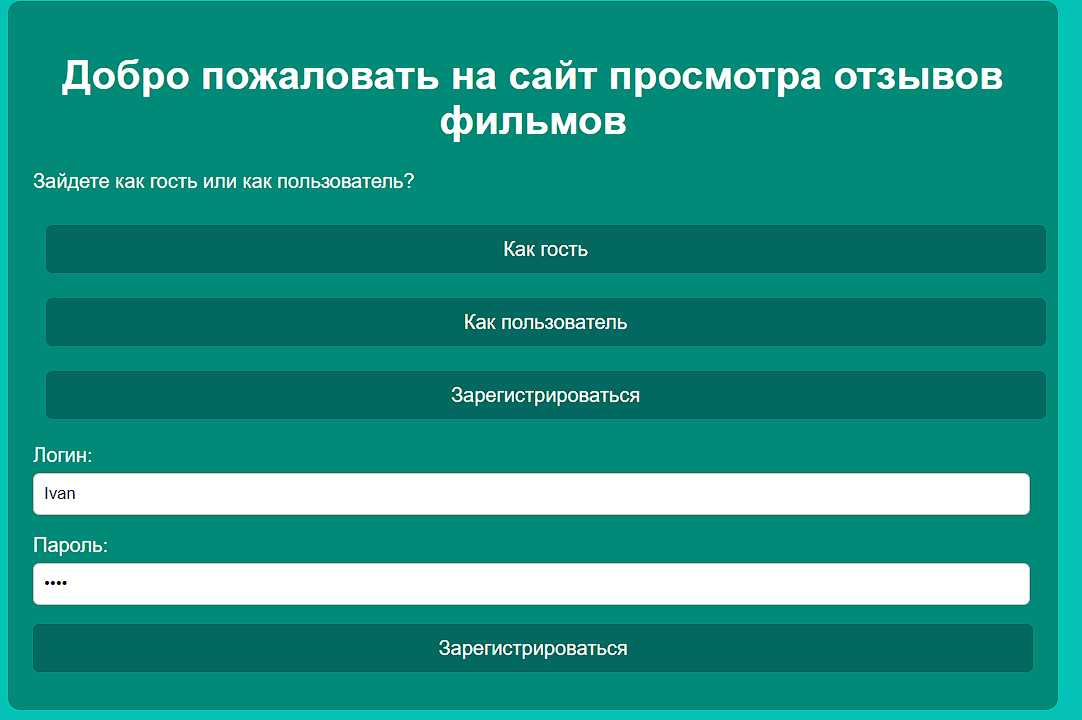


Рисунок 7.Регистрируем нового пользователя

После того, как мы зарегистрируемся, то мы попадем в личный кабинет, где мы можем выбрать нужный фильм и посмотреть его отзывы (см. рис.8).

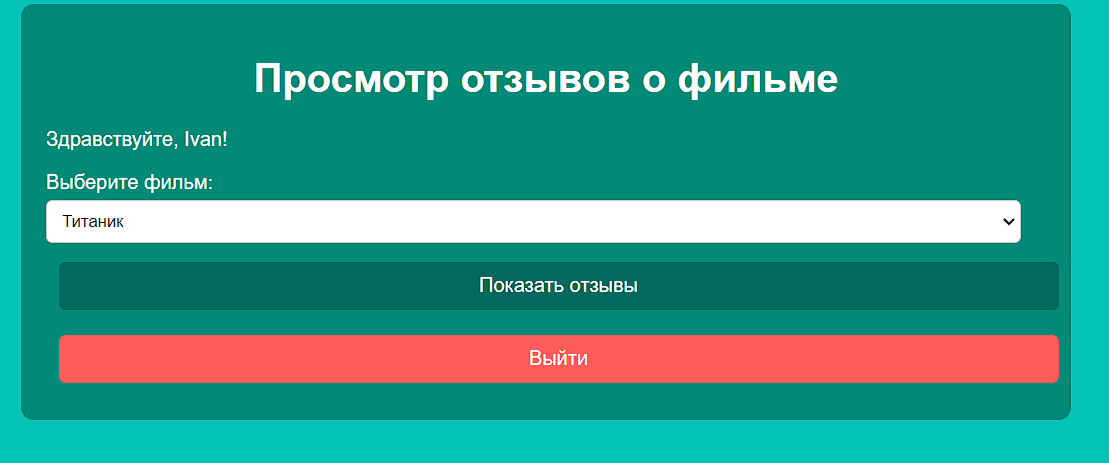


Рисунок 8.Личный кабинет

При нажатии на кнопку «Показать отзывы», то он выводит на страницу с самими отзывами (см. рис.9).

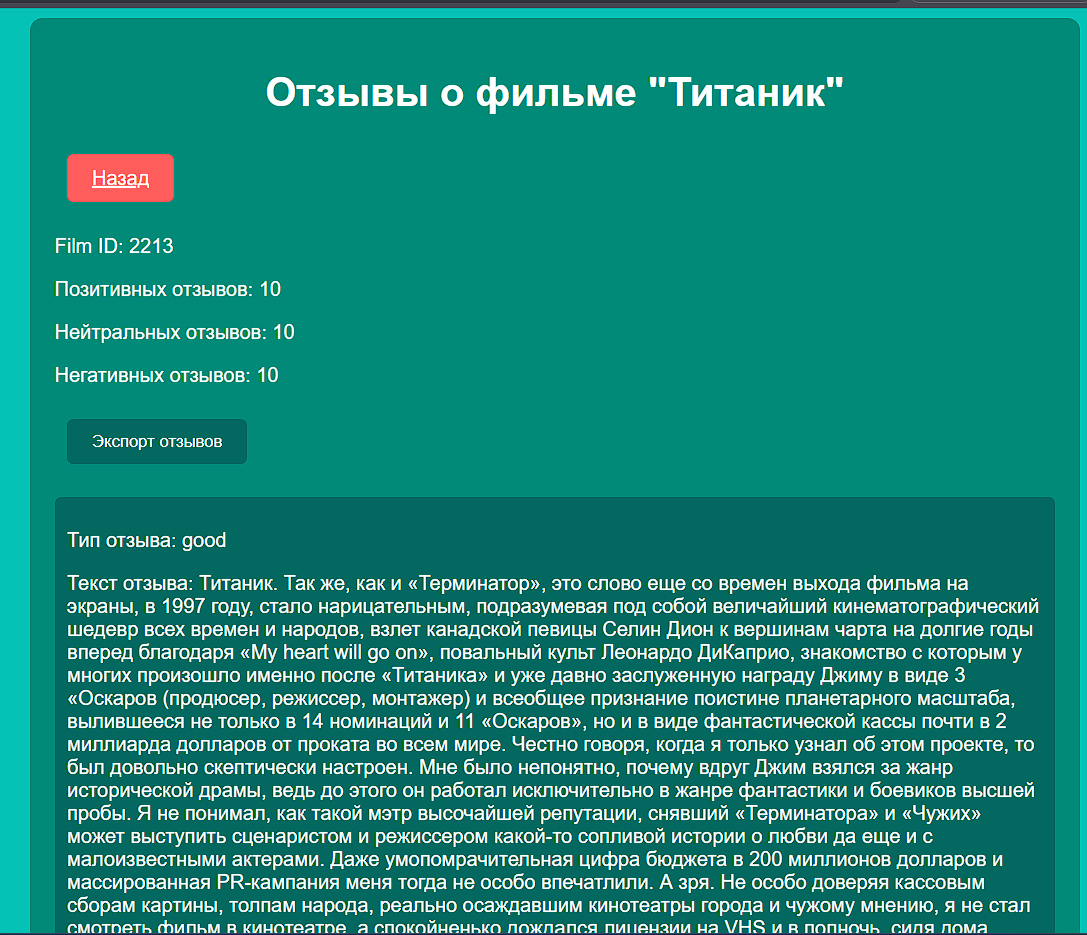


Рисунок 9. Страница с отзывами

Мы можем видеть весь список отзывов, а также его статистику, какие типы отзывов нам представлены, также можем удалить отзывы если он на не нравиться, нажимая на кнопку «Удалить отзыв» под каждым отзывом, страница перезапустится а вместе с ним и статистика (см. рис.10-11).

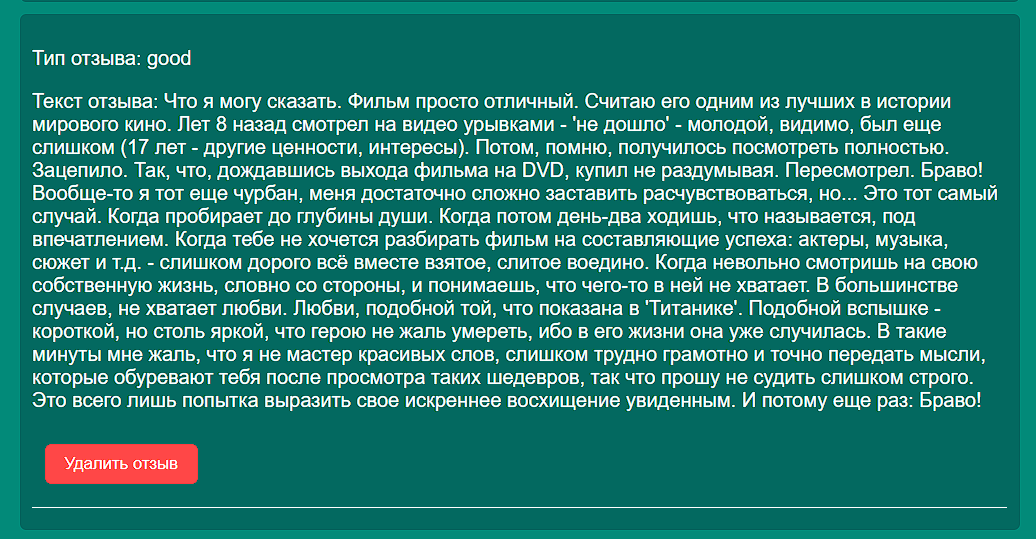


Рисунок 10. Отзыв, который хотим удалить

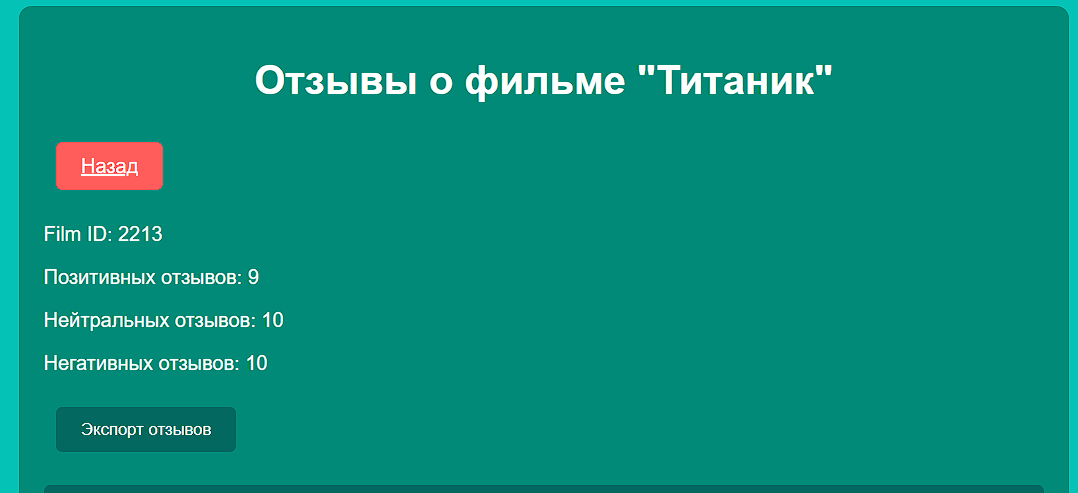


Рисунок 11. Видим, что статистика один положительный отзыв удалился

Так же мы можем экспортировать все отзывы в отдельный TXT файл, при нажатии кнопки «Экспорт отзывов» (см. рис.12).

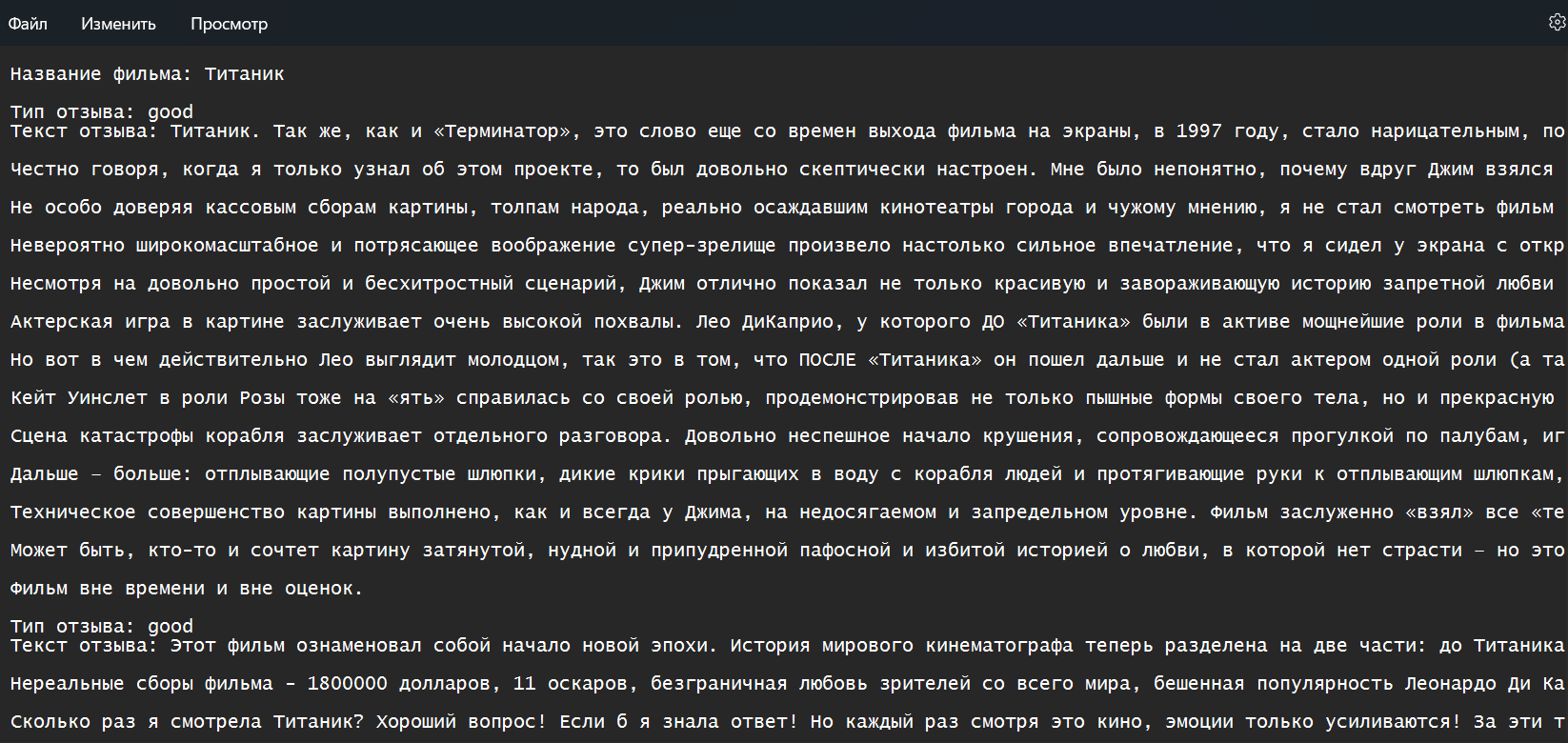


Рисунок 12. Текстовый файл с отзывами

После того как мы сделали все операции с этими отзывами, мы можем вернуться обратно, где запишется информация, с какими отзывами мы работали (см. рис.13).

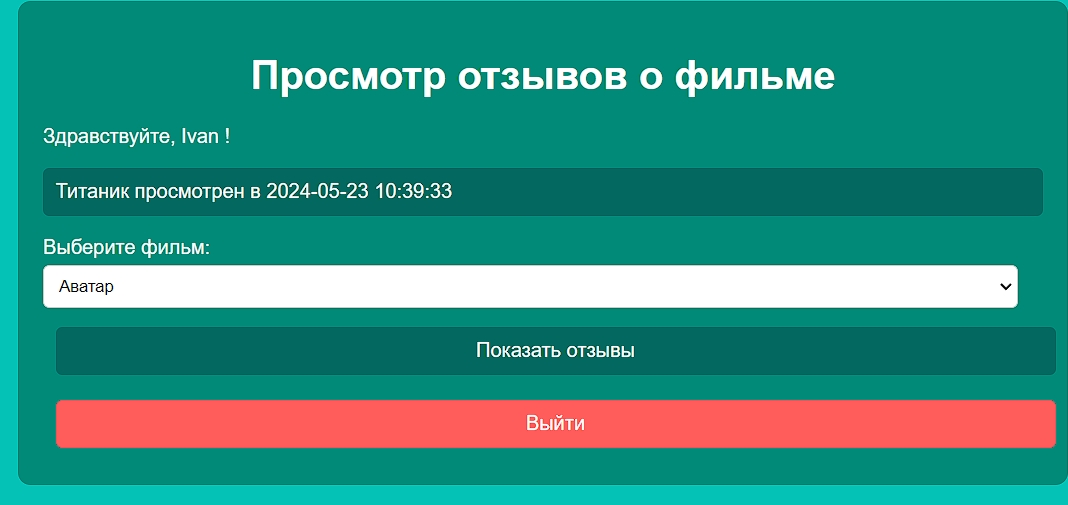


Рисунок 13. Какой фильм мы просмотрели

# **Заключение**

В рамках данной курсовой работы была разработана система для веб-скрапинга отзывов о фильмах с сайта КиноПоиск с использованием языка программирования *Python* и различных инструментов, таких как *Flask* и *SQLite*. Главной целью проекта было создание веб-ресурса, который автоматизирует процесс сбора отзывов, их хранения и последующего анализа.

Рассмотренные этапы разработки включали:

1)Анализ предметной области и существующих решений:

* Были исследованы и проанализированы популярные инструменты для веб-скрапинга, такие как *Scraper* *API*, *Octoparse* и *DataOx*.
* На основании анализа был выбран подход, наиболее подходящий для задач проекта.

2)Проектирование и реализация системы:

* Разработан и реализован веб-ресурс с использованием *Flask*, который обеспечивает взаимодействие пользователя с системой.
* Для хранения данных был выбран *SQLite*, что позволило организовать эффективное и простое управление базой данных.

3)Реализация функционала веб-скрапинга:

* Создан класс *ReviewScraper*, который автоматизирует процесс сбора отзывов с сайта КиноПоиск.
* Разработан механизм сохранения собранных данных в базу данных *SQLite*.

4)Тестирование и отладка системы:

* Проведено тестирование функциональности системы

Результатом данной курсовой работы является функциональная система, позволяющая собирать и анализировать отзывы о фильмах с КиноПоиска. Данный веб-ресурс предоставляет пользователям удобный интерфейс для доступа к собранным данным и их последующего использования.

Перспективы дальнейшего развития проекта включают:

* Расширение функционала системы для поддержки сбора отзывов с других платформ.
* Улучшение интерфейса пользователя и добавление новых возможностей для анализа данных.
* Оптимизация процесса скрапинга для повышения его эффективности и снижения нагрузки на ресурсы.

Разработанная система может быть полезной для кинокритиков, исследователей и всех, кто интересуется анализом отзывов о фильмах. Она предоставляет удобный и автоматизированный инструмент для сбора данных, что значительно упрощает процесс анализа отзывов и делает его более эффективным.

# **Список литературы**

1. Flask документация: [Электронный ресурс] [https://flask.palletsprojects.com/en/3.0.x/](https://flask.palletsprojects.com/en/3.0.x/%20) (дата обращения: 1.04.2024);
2. Python документация: [Электронный ресурс] [https://docs.python.org/3/index.html](https://docs.python.org/3/index.html%20) (дата обращения: 1.04.2024);
3. Sqlite3 документация: [Электронный ресурс] [https://docs.python.org/3/library/sqlite3.html](https://docs.python.org/3/library/sqlite3.html%20) (дата обращения: 1.04.2024);
4. BeautifulSoup документация: [Электронный ресурс] [https://beautiful-soup-4.readthedocs.io/en/latest/](https://beautiful-soup-4.readthedocs.io/en/latest/%20) (дата обращения: 1.04.2024);
5. Requests документация: [Электронный ресурс] [https://requests.readthedocs.io/en/latest/index.html](https://requests.readthedocs.io/en/latest/index.html%20) (дата обращения: 1.04.2024);

# **ПРИЛОЖЕНИЕ А. Листинг кода приложения**

Файл mian.py

# -\*- coding: utf-8 -\*-

from flask import Flask, render\_template, request, redirect, send\_file

import sqlite3

import webbrowser

from threading import Timer

import subprocess

from tempfile import NamedTemporaryFile

from flask import session

from werkzeug.security import generate\_password\_hash, check\_password\_hash

app = Flask(\_\_name\_\_)

app.secret\_key = 'your-secret-key'

# Функция для получения отзывов о фильме из базы данных

def get\_reviews(film\_id):

    conn = sqlite3.connect('reviews.db')

    cursor = conn.cursor()

    cursor.execute('SELECT \* FROM reviews WHERE film\_id = ?', (film\_id,))

    reviews = cursor.fetchall()

    conn.close()

    return reviews

# Маршрут для главной страницы, отображает страницу приветствия или страницу пользователя в зависимости от того, вошел ли пользователь в систему

@app.route('/index', methods=['GET', 'POST'])

def index():

    username = session.get('username')

    user\_logs = []

    if username is not None:

        conn = sqlite3.connect('reviews.db')

        cursor = conn.cursor()

        cursor.execute('SELECT id FROM users WHERE username = ?', (username,))

        user\_row = cursor.fetchone()

        if user\_row:

            user\_id = user\_row[0]

            cursor.execute('SELECT film\_name, timestamp FROM user\_logs WHERE user\_id = ? ORDER BY timestamp DESC', (user\_id,))

            user\_logs = cursor.fetchall()

        conn.close()

    return render\_template('index.html', username=username, user\_logs=user\_logs)

# Маршрут для страницы входа, обрабатывает вход пользователя или гостя

@app.route('/login', methods=['GET', 'POST'])

def login():

    error = None

    if request.method == 'POST':

        user\_type = request.form.get('user\_type')

        if user\_type == 'guest':

            session.pop('username', None)

            return redirect('/index')

        username = request.form.get('username')

        password = request.form.get('password')

        conn = sqlite3.connect('reviews.db')

        cursor = conn.cursor()

        cursor.execute('SELECT password FROM users WHERE username = ?', (username,))

        user = cursor.fetchone()

        conn.close()

        if user is None:

            error = "Ошибка: такого логина нет в базе данных"

        else:

            password\_hash = user[0]

            if check\_password\_hash(password\_hash, password):

                session['username'] = username

                return redirect('/index')

            else:

                error = "Ошибка: неверный пароль"

    return render\_template('welcome.html', error=error)

# Маршрут для начальной страницы, обрабатывает выбор пользователя войти как гость или как пользователь

@app.route('/', methods=['GET', 'POST'])

def welcome():

    if request.method == 'POST':

        user\_type = request.form.get('user\_type')

        if user\_type == 'guest':

            return redirect('/index')

        elif user\_type == 'user':

            return redirect('/login')

    return render\_template('welcome.html')

# Маршрут для страницы регистрации, обрабатывает регистрацию нового пользователя

@app.route('/register', methods=['POST'])

def register():

    username = request.form.get('new\_username')

    password = request.form.get('new\_password')

    password\_hash = generate\_password\_hash(password)

    conn = sqlite3.connect('reviews.db')

    cursor = conn.cursor()

    cursor.execute('INSERT INTO users (username, password) VALUES (?, ?)', (username, password\_hash))

    conn.commit()

    conn.close()

    session['username'] = username  # Сохраняем имя пользователя в сессии

    return redirect('/index')

# Маршрут для страницы отзывов, отображает отзывы о выбранном фильме и обрабатывает добавление отзыва в лог пользователя

@app.route('/reviews', methods=['GET', 'POST'])

def show\_reviews():

    if request.method == 'POST':

        film\_id = request.form.get('film\_id')

    else:

        film\_id = request.args.get('film\_id')

    if film\_id:

        process = subprocess.Popen(["python3", "reviews.py", str(film\_id)])

        process.wait()

        conn = sqlite3.connect('reviews.db')

        cursor = conn.cursor()

        cursor.execute('SELECT name FROM films WHERE id = ?', (film\_id,))

        film\_row = cursor.fetchone()

        if film\_row:

            film\_name = film\_row[0]

            reviews = get\_reviews(film\_id)

            positive\_count = sum(1 for review in reviews if review[2] == 'good')

            neutral\_count = sum(1 for review in reviews if review[2] == 'neutral')

            negative\_count = sum(1 for review in reviews if review[2] == 'bad')

            username = session.get('username')

            if username is not None:

                cursor.execute('SELECT id FROM users WHERE username = ?', (username,))

                user\_row = cursor.fetchone()

                if user\_row:

                    user\_id = user\_row[0]

                    cursor.execute('INSERT INTO user\_logs (user\_id, film\_name) VALUES (?, ?)', (user\_id, film\_name))

                    conn.commit()

            conn.close()

            return render\_template('reviews.html', film\_name=film\_name, film\_id=film\_id, reviews=reviews,

                                   positive\_count=positive\_count, neutral\_count=neutral\_count, negative\_count=negative\_count)

        else:

            conn.close()

            return "Фильм не найден"

    else:

        return "Нет ID фильма"

# Маршрут для удаления отзыва, обрабатывает удаление отзыва из базы данных

@app.route('/delete\_review', methods=['POST'])

def delete\_review():

    review\_id = request.form['review\_id']

    film\_id = request.form['film\_id']

    conn = sqlite3.connect('reviews.db')

    cursor = conn.cursor()

    cursor.execute('DELETE FROM reviews WHERE id = ?', (review\_id,))

    conn.commit()

    conn.close()

    return redirect(f'/reviews?film\_id={film\_id}')

# Маршрут для экспорта отзывов, обрабатывает экспорт отзывов о выбранном фильме в текстовый файл

@app.route('/export\_reviews', methods=['POST'])

def export\_reviews():

    film\_name = request.form.get('film\_name')

    film\_id = request.form.get('film\_id')

    if film\_id:

        reviews = get\_reviews(film\_id)

        # Создаем временный файл для записи отзывов с названием "Отзывы"

        with NamedTemporaryFile(mode='w', delete=False, encoding='utf-8', suffix='.txt', prefix='Отзывы\_') as temp\_file:

            temp\_file.write(f"Название фильма: {film\_name}\n\n")

            for review in reviews:

                temp\_file.write(f"Тип отзыва: {review[2]}\nТекст отзыва: {review[3]}\n\n")

        # Отправляем временный файл в качестве ответа

        return send\_file(temp\_file.name, as\_attachment=True, mimetype='text/plain')

    else:

        return "Ошибка: отсутствует идентификатор фильма"

# Функция для создания таблицы пользователей и таблицы логов пользователей в базе данных

def create\_users\_table():

    conn = sqlite3.connect('reviews.db')

    cursor = conn.cursor()

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS users (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            username TEXT NOT NULL UNIQUE,

            password TEXT NOT NULL

        );

    ''')

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS user\_logs (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            user\_id INTEGER,

            film\_name TEXT,

            timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

        );

    ''')

    conn.commit()

    conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

    create\_users\_table()

    url = "http://127.0.0.1:5000"

    Timer(1, lambda: webbrowser.open(url)).start()  # Открываем браузер через 1 секунду

    app.run()

Файл reviews.py

import sqlite3

import requests

from bs4 import BeautifulSoup

# Инициализация объекта ReviewScraper

class ReviewScraper:

    def \_\_init\_\_(self, film\_id, max\_pages):

        self.film\_id = film\_id  # ID фильма на КиноПоиске

        self.max\_pages = max\_pages  # Максимальное количество страниц для обработки

        self.conn = sqlite3.connect('reviews.db')  # Создаем соединение с базой данных SQLite

        self.create\_tables()  # Создаем таблицы, если они не существуют

    # Метод создания таблицы

    def create\_tables(self):

        cursor = self.conn.cursor()

        cursor.execute('''

            CREATE TABLE IF NOT EXISTS films (

                id TEXT PRIMARY KEY,

                name TEXT

            )

        ''')

        cursor.execute('''

            CREATE TABLE IF NOT EXISTS reviews (

                id INTEGER PRIMARY KEY AUTOINCREMENT,

                film\_id TEXT,

                review\_type TEXT,

                review\_text TEXT,

                FOREIGN KEY(film\_id) REFERENCES films(id)

            )

        ''')

        self.conn.commit()

    # Метод для получения и сохранения отзывов

    def fetch\_reviews(self):

        # Проверяем, есть ли фильм с данным ID в базе данных

        cursor = self.conn.cursor()

        cursor.execute('SELECT COUNT(\*) FROM films WHERE id = ?', (self.film\_id,))

        count = cursor.fetchone()[0]

        if count > 0:

            print('Фильм с данным ID уже существует в базе данных.')

            return  # Если фильм уже есть, выходим из метода

        for review\_type in ['good', 'bad', 'neutral']:

            for page in range(1, self.max\_pages + 1):

                url = self.create\_review\_url(review\_type, page)  # Создаем URL для страницы с отзывами

                response = requests.get(url)  # Отправляем GET-запрос к URL

                if response.status\_code != 200:

                    print(f"Не удалось подключиться.")

                    continue  # Пропускаем текущую страницу, если нет ответа от сервера

                soup = BeautifulSoup(response.text, 'html.parser')  # Парсим HTML страницы

                reviews = soup.find\_all('div', {'itemprop': 'reviews'})  # Находим блоки с отзывами

                film\_name\_element = soup.find('a', class\_='breadcrumbs\_\_link')

                if film\_name\_element is not None:

                    film\_name = film\_name\_element.text

                else:

                    film\_name = "Unknown"  # Если название фильма не найдено, присваиваем его значение "Unknown"

                self.save\_film\_to\_database(self.film\_id, film\_name)  # Сохраняем информацию о фильме в базу данных

                for i, review in enumerate(reviews):

                    film\_name = soup.find('a', class\_='breadcrumbs\_\_link').text

                    review\_text = review.find('span', {'itemprop': 'reviewBody'}).text

                    film\_id = self.film\_id

                    self.save\_review\_to\_database(film\_id, review\_type, review\_text)

    # Метод для создания URL страницы с отзывами

    def create\_review\_url(self, review\_type, page):

        return f'https://www.kinopoisk.ru/film/{self.film\_id}/reviews/ord/rating/status/{review\_type}/perpage/10/page/{page}/'

    # Метод для сохранения отзыва в sql

    def save\_review\_to\_database(self, film\_id, review\_type, review\_text):

        cursor = self.conn.cursor()

        cursor.execute('INSERT INTO reviews (film\_id, review\_type, review\_text) VALUES (?, ?, ?)',

                       (film\_id, review\_type, review\_text))

        self.conn.commit()

    # Метод для сохранения фильма в sql

    def save\_film\_to\_database(self, film\_id, film\_name):

        cursor = self.conn.cursor()

        cursor.execute('SELECT id FROM films WHERE id = ?', (film\_id,))

        existing\_film = cursor.fetchone()

        if existing\_film is None:

            cursor.execute('INSERT INTO films (id, name) VALUES (?, ?)', (film\_id, film\_name))

            self.conn.commit()

# Изменения в методе main()

def main(film\_id):

    max\_pages = 1  # Максимальное количество страниц для обработки (по умолчанию 1)

    scraper = ReviewScraper(film\_id, max\_pages)  # Создаем экземпляр класса ReviewScraper

    scraper.fetch\_reviews()

    print('Готово')

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    film\_id = int(sys.argv[1])  # Получаем film\_id из аргументов командной строки

    main(film\_id)