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 2023

Оглавление

[**Введение** 3](#_Toc169488675)

[**Техническое задание** 4](#_Toc169488676)

[**Обзор предметной области** 6](#_Toc169488677)

[**Анализ существующих решений** 6](#_Toc169488678)

[**Проектирование прототипа** 7](#_Toc169488679)

[**Варианты использования** 7](#_Toc169488680)

[**Структура БД** 8](#_Toc169488681)

[**Обзор средств реализации** 10](#_Toc169488682)

[***Flask*** 10](#_Toc169488683)

[**Используемые библиотеки** 11](#_Toc169488684)

[**Реализация** 13](#_Toc169488685)

[**Скрапинг** **отзыва** 13](#_Toc169488686)

[**Описание роутов и HTTP методов** 15](#_Toc169488687)

[**Функциональное тестирование** 16](#_Toc169488688)

[**Автоматизированное тестирование** 21](#_Toc169488689)

[**Postman** 21](#_Toc169488690)

[**Pytest** 22](#_Toc169488691)

[**Заключение** 23](#_Toc169488692)

[**Список литературы** 25](#_Toc169488693)

[**ПРИЛОЖЕНИЕ А. Листинг кода приложения** 26](#_Toc169488694)

# **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.

# **Техническое задание**

**Тема проекта*:*** Веб-ресурс для скрапинга отзывов фильмов.

**Используемые библиотеки:**

* *Python 3.11.3*
* *requests*
* *flask*
* *webbrowser*
* *threading*
* *routes*
* *os*
* *config*
* *subprocess*
* *pytest*
* *sqlite3*
* *bs4*
* *pandas*
* *tempfile*
* *werkzeug.security*

**Цель и описание работы:**

Целью данной работы является создание веб-ресурса , который будет способен собирать отзывы о фильмах с кинопоиска и предоставлять пользователям удобный доступ к ним.

**Требование к техническому обеспечению:**

* Установленный *Python* и его библиотеки.

**Требование к пользовательскому интерфейсу:**

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

**Требование к обработке ошибок:**

* Обработка ошибок: Неправильного введенного логина и пароля, ошибки 404 и 405 или доступ к несуществующей странице.

**Примерный функционал:**

* 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.

**План работ с датами и основными этапами разработки:**

|  |  |
| --- | --- |
| **Дата** | **Этап** |
| 20.02.2024-02.03.2024 | Техническое задание |
| 03.03.2024-20.05.2024 | Разработка приложения |
| 21.05.2024-10.06.2024 | Пояснительная записка |
| 10.06.2024-17.06.2024 | Сдача работы |

# **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. Также в отличие от традиционных методов, где пользователю приходится самостоятельно находить и выбирать нужные данные на HTML-страницах, Scraper API, Octoparse и DataOx используют передовые алгоритмы для автоматического распознавания и извлечения данных. Эти решения предлагают инструменты для создания скреперов без необходимости программирования, позволяя пользователям настроить правила извлечения данных через интуитивно понятные интерфейсы или API.

Мой проект также включает автоматизацию процесса скрапинга без активного участия пользователя.

# **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.

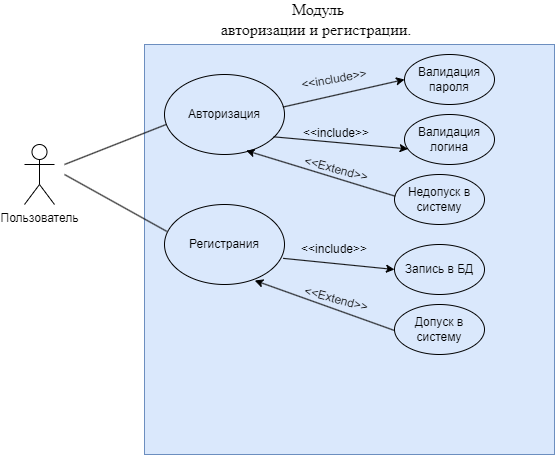


Рисунок 1. Диаграмма вариантов использования для модуля авторизации и регистрации

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.

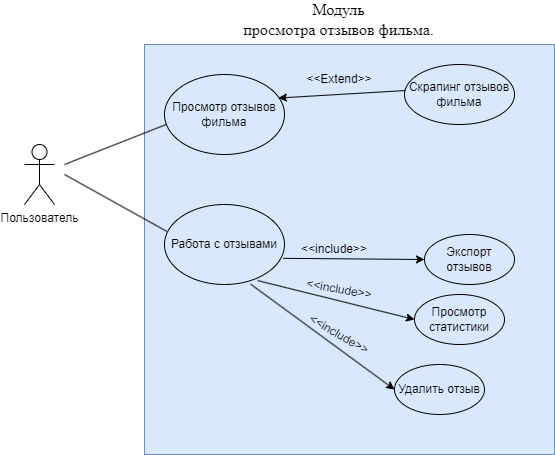


Рисунок 2. Диаграмма вариантов использования для модуля просмотра отзывов фильма

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* (*INTEGER*): 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.

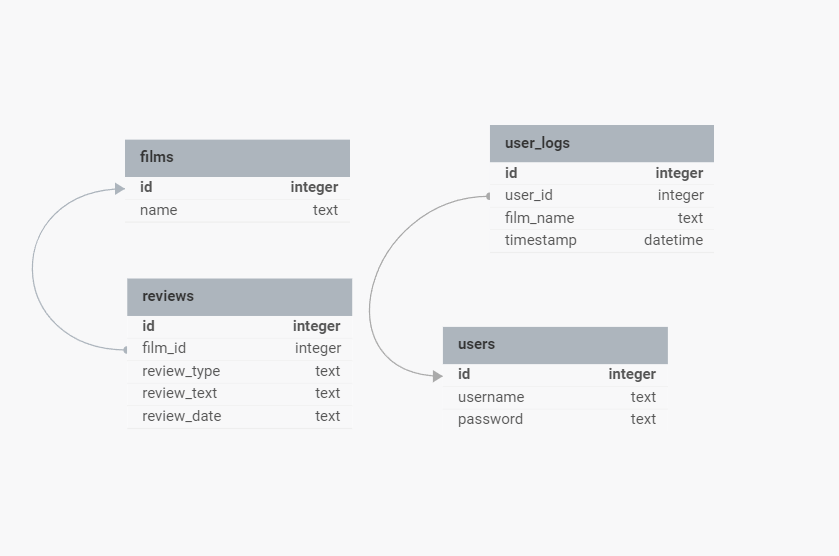


Рисунок 3. Структура БД

# **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.

## **Используемые библиотеки**

* *Requests:* Используется для выполнения HTTP запросов к веб-серверам, получения данных с веб-страниц.
* *Webbrowser:* Используется для открытия URL в стандартном веб-браузере пользователя.
* *threading:* Используется для реализации параллельного выполнения задач, улучшения производительности и отзывчивости приложений.
* *routes:* Используется для определения URL маршрутов и связывания их с обработчиками запросов во Flask и других веб-фреймворках.
* *os:* Используется для выполнения операций с файлами
* *config:* Используется для загрузки, чтения и записи конфигурационной информации приложения.
* *subprocess*: Используется запуска внешних программ.
* *pytest:* Используется для написания и запуска автоматических тестов.
* *bs4:* Используется для извлечения данных из веб-страниц.
* *pandas:* Используется для работы с табличными данными при экспорте файла.
* *tempfile:* Используется для создания временных файлов и каталогов в процессе выполнения программы.
* *werkzeug.security:* Используется для хеширования паролей пользователей, проверки подлинности и обеспечения защиты данных.

# **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.

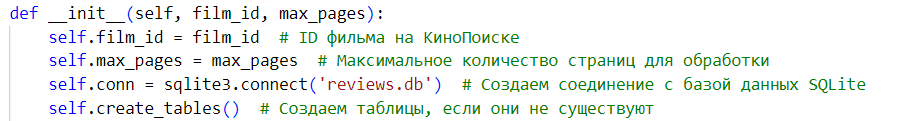


Рисунок 4. Инициализация объекта

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.

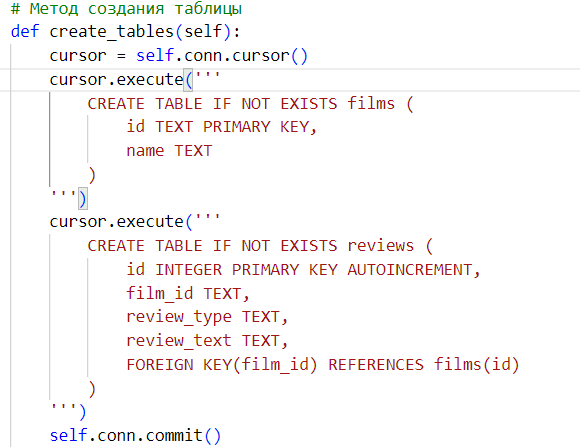
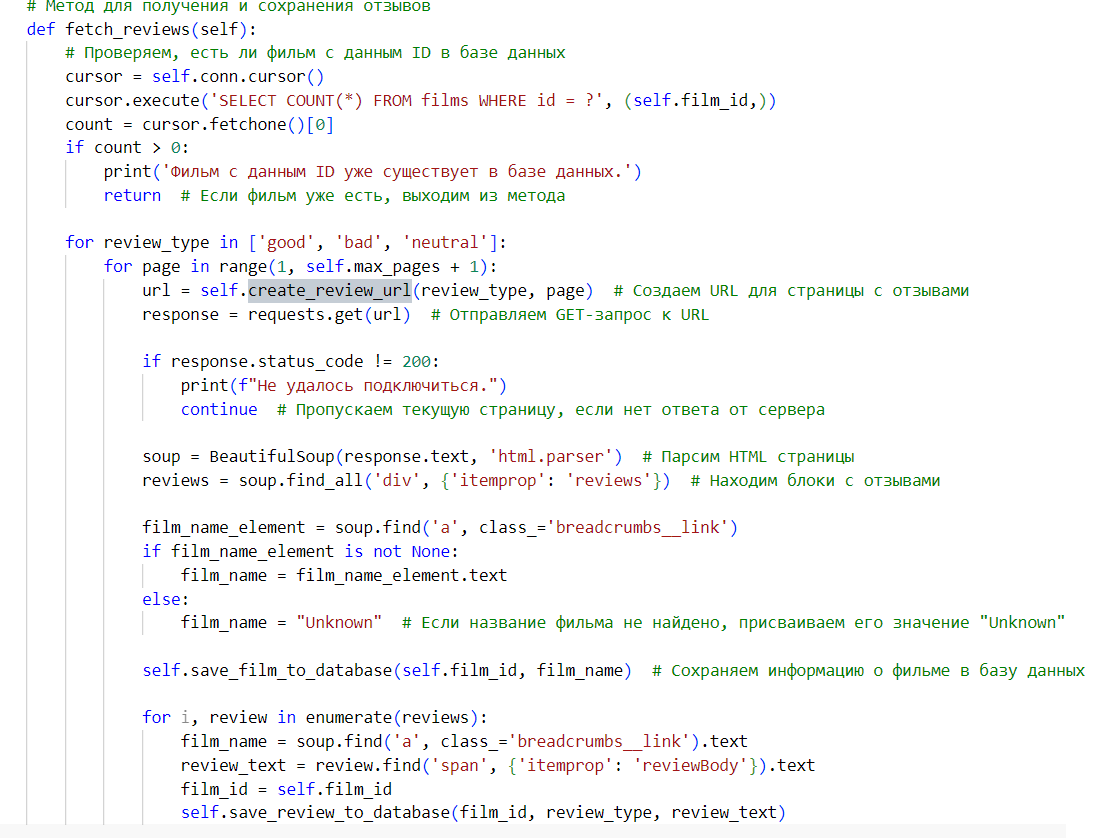


Рисунок 5. Создание таблиц

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 6. Получение и сохранение отзывов

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

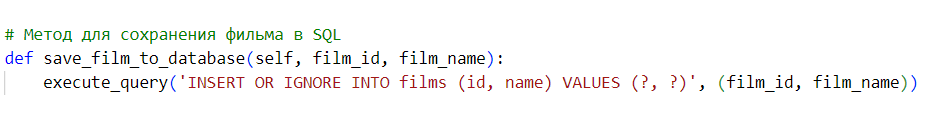
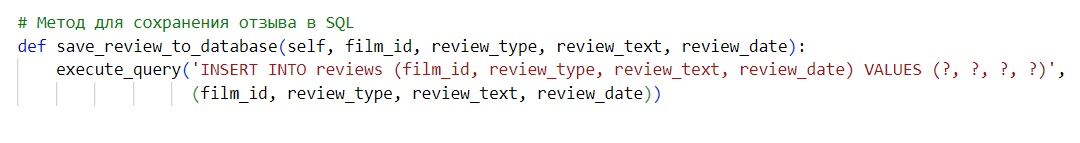
* Проверяется, существует ли уже фильм с данным *ID* в базе данных. Если да, метод завершает работу.
* Проходится по типам отзывов (*good*, *bad*, *neutral*) и страницам отзывов.
* Формируется *URL* для страницы с отзывами и отправляется *GET*-запрос.
* Если ответ успешный, *HTML*-страница парсится с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

Рисунок 7. Сохранение название фильма

* Сохраняется информация о фильме в базу данных.



* Каждый отзыв извлекается и сохраняется в базу данных.

Рисунок 8. Сохранение отзыва о фильме

## **Описание роутов и HTTP методов**

1. Роут «selection/<date>»

* HTTP методы: GET
* Описание: Этот роут принимает дату в формате <date> в качестве параметра и возвращает результаты поиска зрителей на указанную дату.

1. Роут «selection\_range/<start\_date>&<end\_date>»

* HTTP методы: GET
* Описание: Этот роут принимает две даты <start\_date> и <end\_date> в качестве параметров и возвращает результаты поиска зрителей за указанный диапазон дат.

1. Роут /index

* HTTP методы: GET, POST
* Описание: Роут для отображения главной страницы (индексной страницы) и обработки данных формы входа (POST запросы).

1. Роут /login

* HTTP методы: GET, POST
* Описание: Роут для страницы входа в систему. GET запрос отображает форму входа, POST запрос обрабатывает введенные данные для аутентификации.

1. Роут /register

* HTTP методы: POST
* Описание: Роут для обработки данных формы регистрации нового пользователя. Принимает данные пользователя и добавляет их в базу данных.

1. Роут /reviews

* HTTP методы: GET, POST
* Описание: Роут для просмотра и добавления отзывов о фильмах. GET запрос отображает страницу с отзывами, POST запрос добавляет новый отзыв в базу данных.

1. Роут /delete\_review

* HTTP методы: POST
* Описание: Роут для удаления отзыва о фильме. Принимает идентификатор отзыва (review\_id) и удаляет соответствующий отзыв из базы данных.

1. Роут /export\_reviews

* HTTP методы: POST
* Описание: Роут для экспорта отзывов о фильмах. Принимает запрос на экспорт данных и выполняет экспорт в формат CSV.

Пример HTTP запроса: GET <http://127.0.0.1:5000/selection/2024-06-01>

Этот запрос отправляет GET запрос к нашему приложению, ожидая получить данные о пользователях на указанную дату 1 июня 2023 года.

# **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).

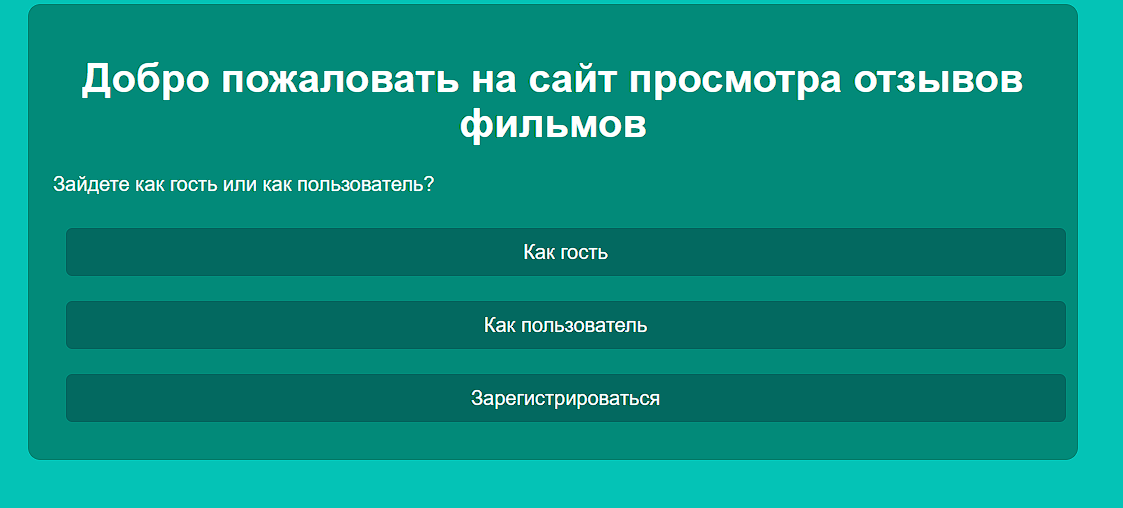


Рисунок 9. Начальное окно

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

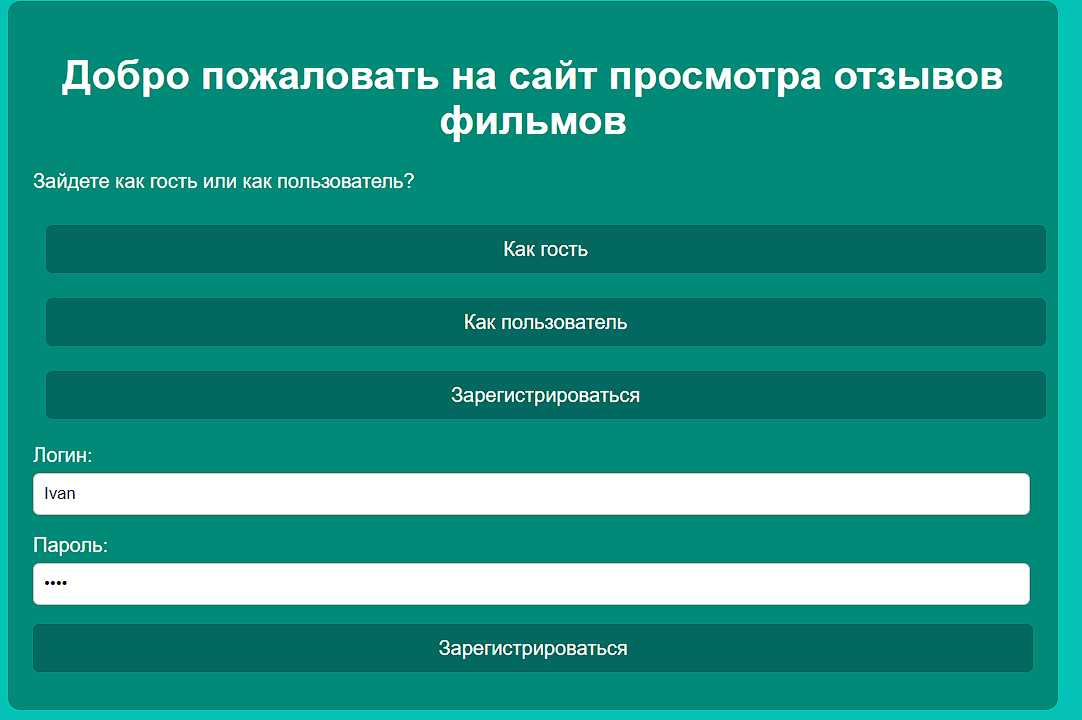


Рисунок 10.Регистрируем нового пользователя

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

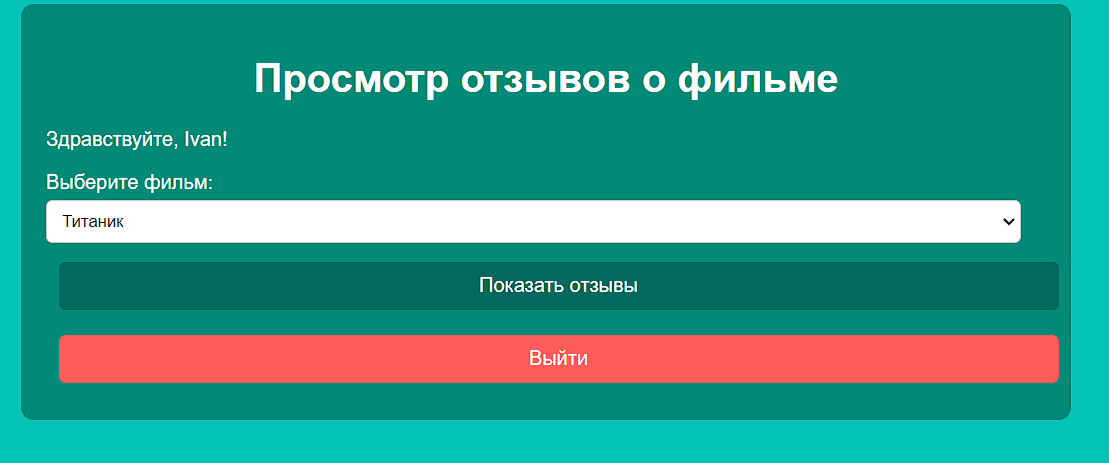


Рисунок 11.Личный кабинет

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

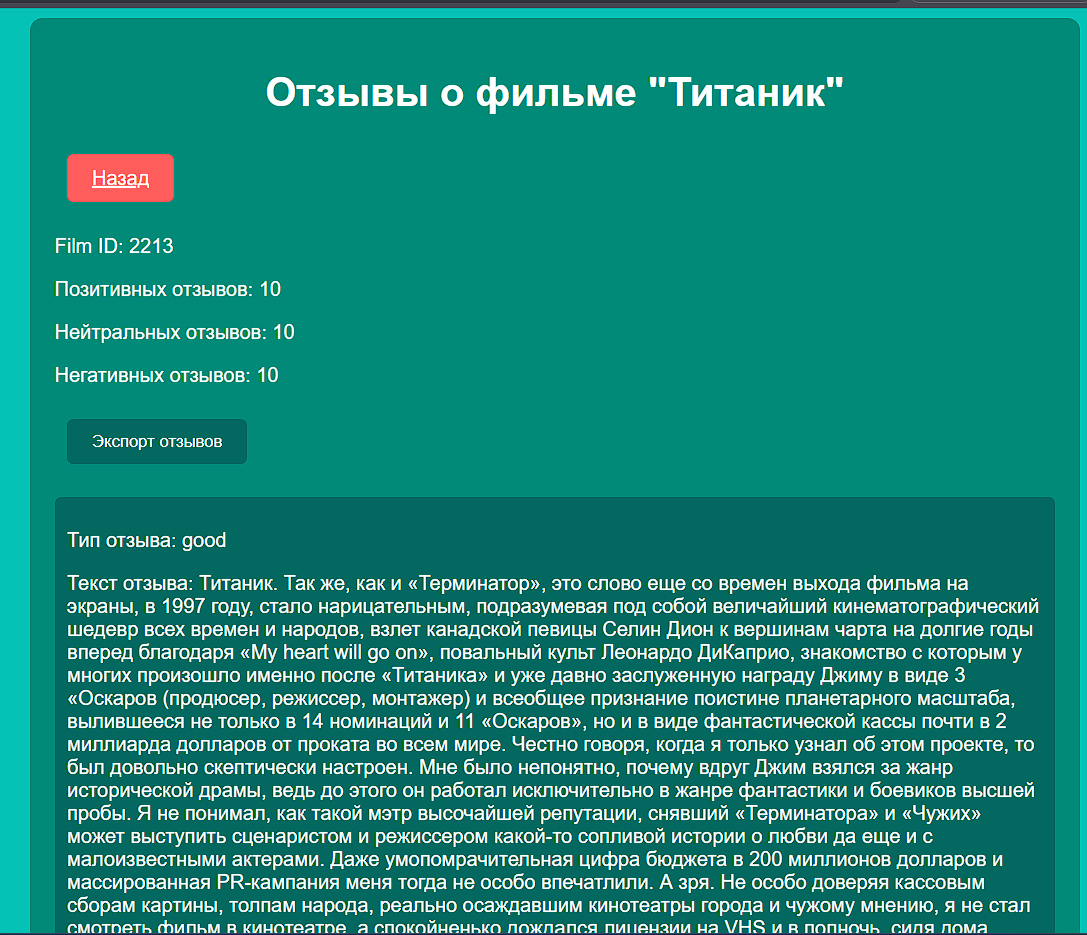


Рисунок 12. Страница с отзывами

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).

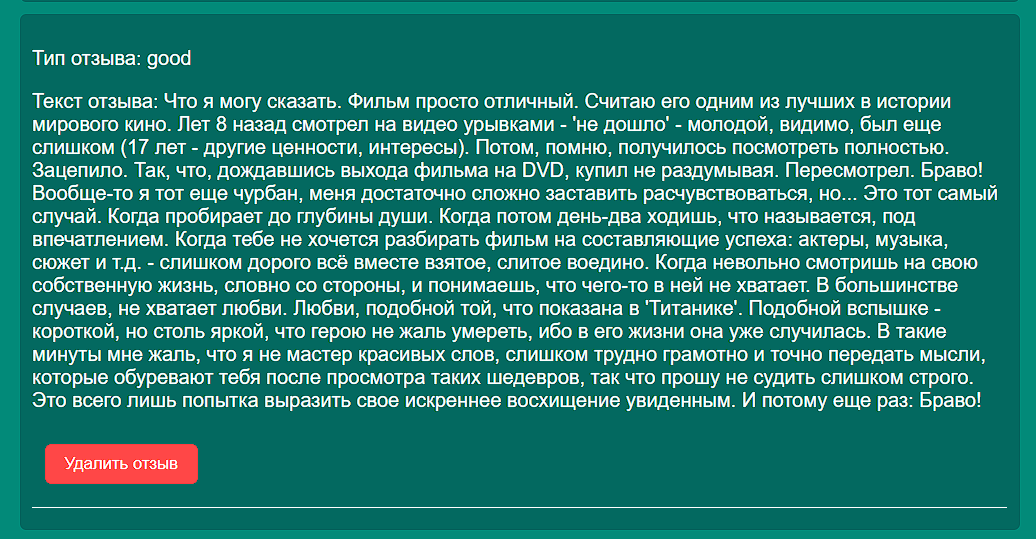


Рисунок 13. Отзыв, который хотим удалить

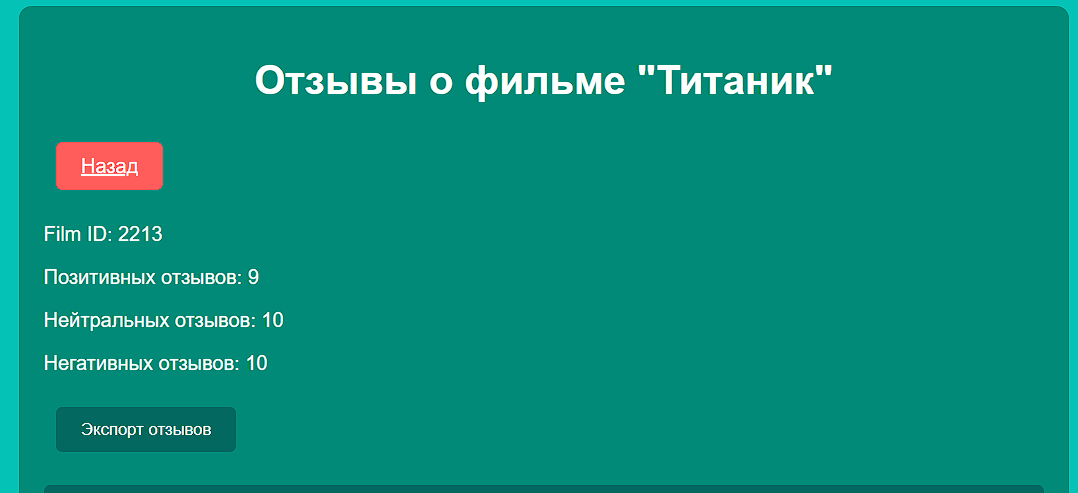


Рисунок 11. Видим, что статистика один положительный отзыв удалился

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

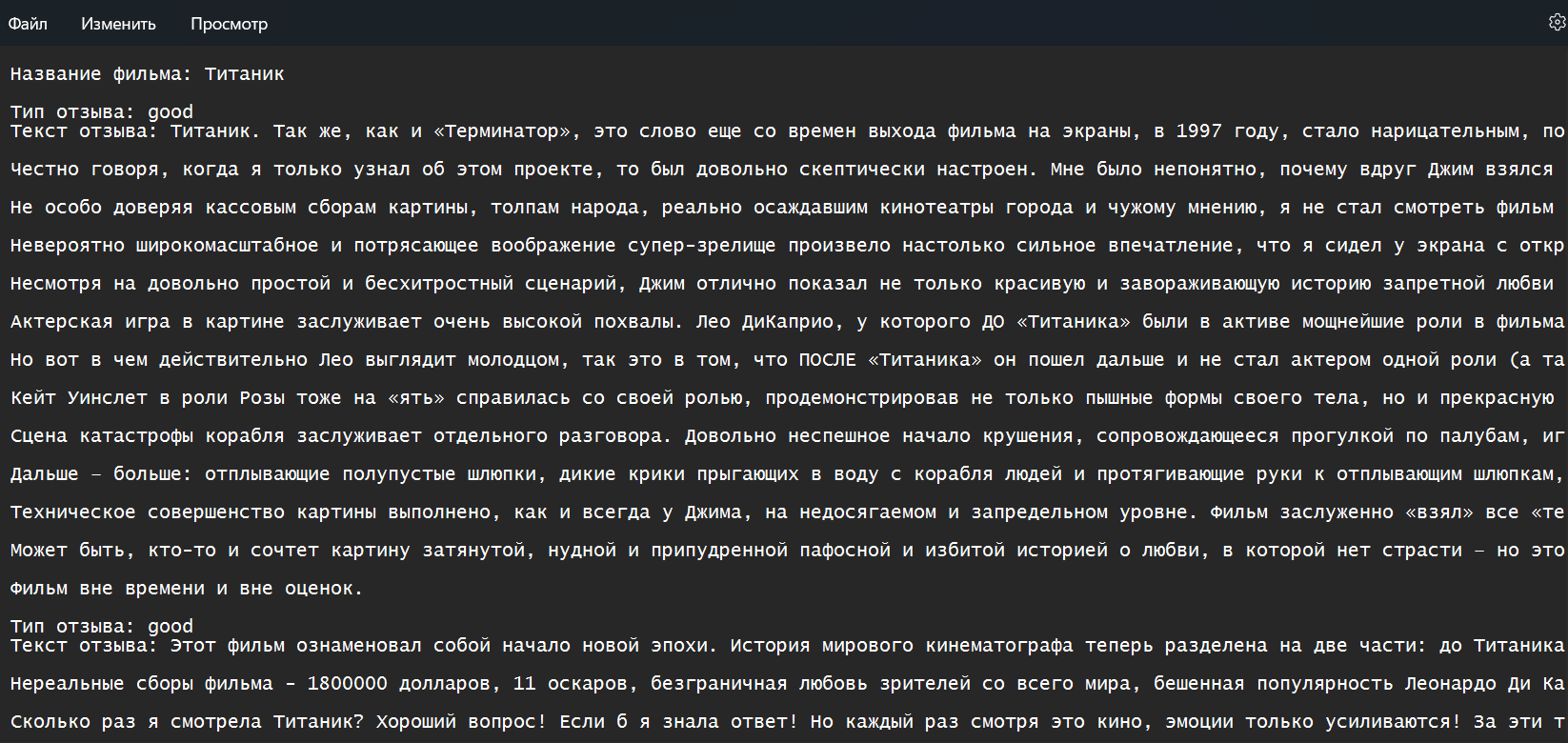


Рисунок 14. Текстовый файл с отзывами

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).

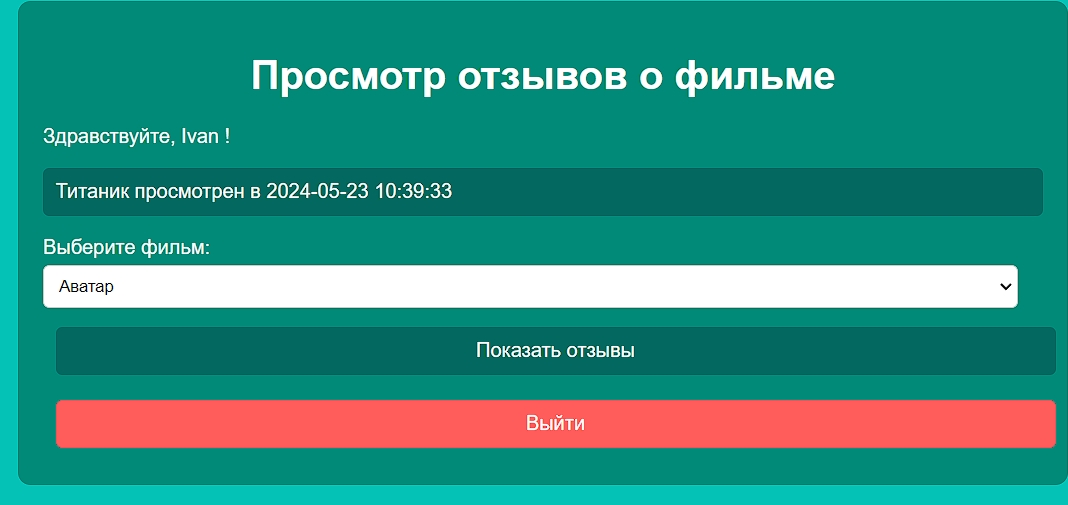


Рисунок 15. Какой фильм мы просмотрели

# **Автоматизированное тестирование**

В качестве автоматического тестирования проведем проверки API через Postman и тест через pytest.

## **Postman**

Для начала сделаем вывод списка пользователей, которые просматривали фильмы за определенную дату.

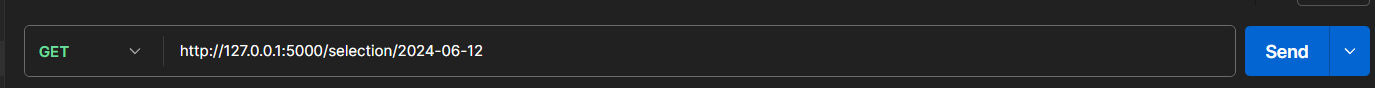


Рисунок 16. GET-запрос

Результатом будет JSON список пользователей.

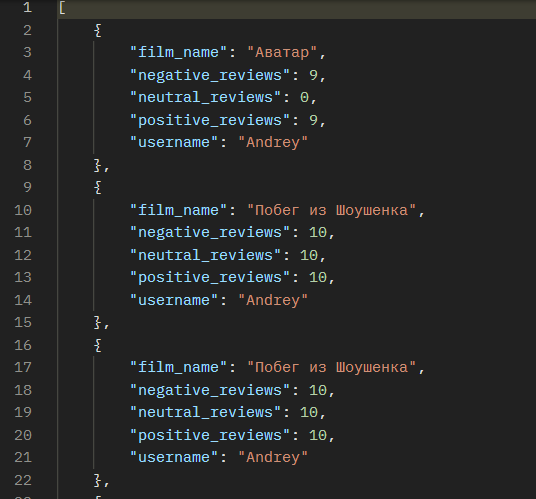


Рисунок 17. Результат запроса

Также проведем проверки на ошибки 404 и 405.

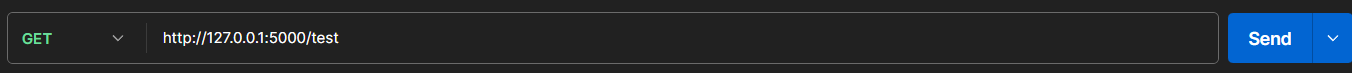


Рисунок 18. GET-запрос

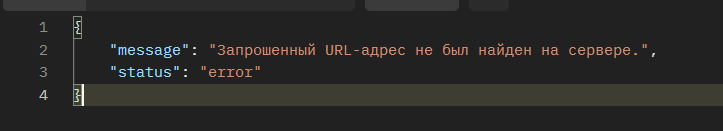


Рисунок 19. Ошибка несуществующего адреса



Рисунок 20. PUT-запрос

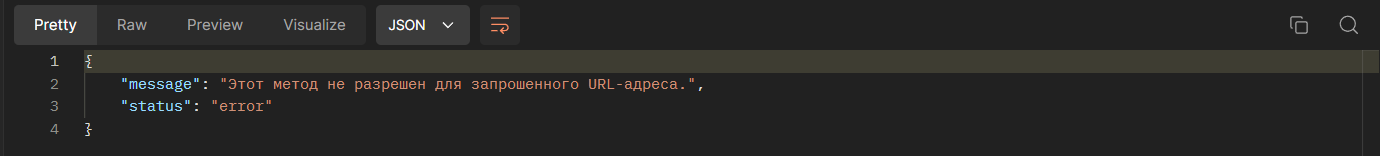


Рисунок 21. Ошибка допуска

## **Pytest**

Через pytest мы проведем проверки следующих составляющих:

* Подключение к базе.
* Вход пользователя, который есть в базе и которого нету.
* Открытия страницы с фильмом, который есть в базе и которого нету на кинопоиске .
* Проверка на ошибку 404.

Листинг теста, представлен в приложении. Запуск осуществляется через команду «*python -m pytest -v -s*». Результат теста.

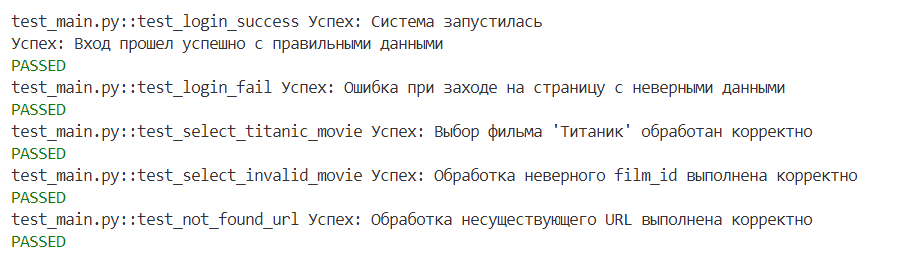


Рисунок 22. Результат теста

# **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);
6. Руководство по веб-скрапингу на Python: [Электронный ресурс] <https://habr.com/ru/companies/ruvds/articles/796885/> (дата обращения: 1.04.2024);
7. Python Web Scraping Tutorial: [Электронный ресурс] <https://www.geeksforgeeks.org/python-web-scraping-tutorial/> (дата обращения: 1.04.2024);

# **ПРИЛОЖЕНИЕ А. Листинг кода приложения**

Файл main.py

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

from flask import Flask

from database import create\_tables

from database import execute\_query

import webbrowser

from threading import Timer

import routes

import os

import config

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

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

# Маршруты

#на одну дату

app.add\_url\_rule('/selection/<date>', 'get\_viewers\_by\_date', routes.get\_viewers\_by\_date, methods=['GET'])

#на диапозон дат

app.add\_url\_rule('/selection\_range/<start\_date>&<end\_date>','selection\_range',routes.selection\_range, methods=['GET'])

#Ошибка404

app.register\_error\_handler(404, routes.not\_found)

#Ошибка405

app.register\_error\_handler(405, routes.method\_not\_allowed)

app.add\_url\_rule('/index', 'index', routes.index, methods=['GET', 'POST'])

app.add\_url\_rule('/login', 'login', routes.login, methods=['GET', 'POST'])

app.add\_url\_rule('/', 'welcome', routes.welcome, methods=['GET', 'POST'])

app.add\_url\_rule('/register', 'register', routes.register, methods=['POST'])

app.add\_url\_rule('/reviews', 'show\_reviews', routes.show\_reviews, methods=['GET', 'POST'])

app.add\_url\_rule('/delete\_review', 'delete\_review', routes.delete\_review, methods=['POST'])

app.add\_url\_rule('/export\_reviews', 'export\_reviews', routes.export\_reviews, methods=['POST'])

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

    # Проверяем наличие файла базы данных

    if not os.path.exists('reviews.db'):

        create\_tables()  # Если файл базы данных не существует, создаем таблицы

    # Проверяем переменную окружения для определения режима тестирования

    if os.getenv('FLASK\_ENV') != 'testing':

        url = f"{config.SERVER\_ADDRESS}:{config.SERVER\_PORT}"

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

    app.run()

Файл routes.py

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

from flask import render\_template, request, redirect, send\_file, session, jsonify

from database import execute\_query

import subprocess

import pandas as pd

from tempfile import NamedTemporaryFile

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

from datetime import datetime

def get\_reviews(film\_id):

    query = 'SELECT \* FROM reviews WHERE film\_id = ?'

    return execute\_query(query, (film\_id,))

def index():

    username = session.get('username')

    user\_logs = []

    if username:

        user\_row = execute\_query('SELECT id FROM users WHERE username = ?', (username,))

        if user\_row:

            user\_id = user\_row[0][0]

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

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

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')

        result = execute\_query('SELECT password FROM users WHERE username = ?', (username,))

        if result:

            password\_hash = result[0][0]

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

                session['username'] = username

                return redirect('/index')

            else:

                error = "Ошибка: неверный пароль"

        else:

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

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

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')

def register():

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

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

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

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

    session['username'] = username

    return redirect('/index')

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:

        # Проверка наличия отзывов в базе данных

        reviews = get\_reviews(film\_id)

        if not reviews:

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

            process.wait()

            reviews = get\_reviews(film\_id)

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

        if film\_row:

            film\_name = film\_row[0][0]

            # Инициализация счетчиков

            positive\_count = 0

            neutral\_count = 0

            negative\_count = 0

            # Один цикл для подсчета всех типов отзывов

            for review in reviews:

                if review[2] == 'good':

                    positive\_count += 1

                elif review[2] == 'neutral':

                    neutral\_count += 1

                elif review[2] == 'bad':

                    negative\_count += 1

            username = session.get('username')

            if username is not None:

                user\_row = execute\_query('SELECT id FROM users WHERE username = ?', (username,))

                if user\_row:

                    user\_id = user\_row[0][0]

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

            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:

            return "Фильм не найден"

    else:

        return "Нет ID фильма"

def delete\_review():

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

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

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

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

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)

        if reviews:

            # Получаем данные о фильме из базы данных

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

            if film\_row:

                film\_name = film\_row[0][0]

                # Создаем DataFrame из отзывов с добавлением столбца film\_name и преобразованием даты

                df = pd.DataFrame(reviews, columns=['id', 'film\_id', 'review\_type', 'review\_text', 'review\_date'])

                df['film\_name'] = film\_name

                df['review\_date'] = pd.to\_datetime(df['review\_date'], format='%Y-%m-%d %H:%M:%S')  # Преобразуем строковое представление даты в формат даты

                # Удаляем столбец "id"

                df.drop(columns=['id'], inplace=True)

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

                temp\_file = NamedTemporaryFile(delete=False, suffix='.csv', prefix='Отзывы\_')

                # Записываем DataFrame в CSV-файл

                df.to\_csv(temp\_file.name, index=False, encoding='utf-16')

                # Отправляем временный файл в качестве ответа

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

            else:

                return "Фильм не найден"

        else:

            return "Нет отзывов для данного фильма"

    else:

        return "Ошибка: отсутствует идентификатор фильма"

def get\_viewers\_by\_date(date):

    try:

        # Запрос к базе данных для извлечения данных о просмотрах за указанную дату

        query = '''

            SELECT users.username,

                   user\_logs.film\_name,

                   (SELECT COUNT(\*) FROM reviews WHERE film\_id = films.id AND review\_type = 'good') AS positive\_reviews,

                   (SELECT COUNT(\*) FROM reviews WHERE film\_id = films.id AND review\_type = 'bad') AS negative\_reviews,

                   (SELECT COUNT(\*) FROM reviews WHERE film\_id = films.id AND review\_type = 'neutral') AS neutral\_reviews

            FROM user\_logs

            JOIN users ON user\_logs.user\_id = users.id

            JOIN films ON user\_logs.film\_name = films.name

            WHERE DATE(user\_logs.timestamp) = ?

        '''

        # Выполнение SQL запроса

        viewers = execute\_query(query, (date,))

        # Формирование списка словарей с данными о пользователях, фильмах и количестве отзывов

        result = [{'username': viewer[0],

                   'film\_name': viewer[1],

                   'positive\_reviews': viewer[2],

                   'negative\_reviews': viewer[3],

                   'neutral\_reviews': viewer[4]} for viewer in viewers]

        # Возвращение результата в формате JSON

        return jsonify(result), 200

    except Exception as e:

        # В случае ошибки возвращаем JSON с сообщением об ошибке и статусом 500

        return jsonify({'error': str(e)}), 500

def selection\_range(start\_date, end\_date):

    try:

        # Выполнение запроса к базе данных для получения данных о просмотрах в заданном диапазоне дат

        query = '''

            SELECT users.username,

                   user\_logs.film\_name,

                   user\_logs.timestamp AS view\_date,

                   (SELECT COUNT(\*) FROM reviews WHERE film\_id = films.id AND review\_type = 'good') AS positive\_reviews,

                   (SELECT COUNT(\*) FROM reviews WHERE film\_id = films.id AND review\_type = 'bad') AS negative\_reviews,

                   (SELECT COUNT(\*) FROM reviews WHERE film\_id = films.id AND review\_type = 'neutral') AS neutral\_reviews

            FROM user\_logs

            JOIN users ON user\_logs.user\_id = users.id

            JOIN films ON user\_logs.film\_name = films.name

            WHERE user\_logs.timestamp BETWEEN ? AND ?

        '''

        viewers = execute\_query(query, (start\_date, end\_date))

        # Формирование списка словарей с данными о просмотрах в заданном диапазоне дат

        result = [{'username': viewer[0],

                   'film\_name': viewer[1],

                   'date': viewer[2],

                   'positive\_reviews': viewer[3],

                   'negative\_reviews': viewer[4],

                   'neutral\_reviews': viewer[5]}

                  for viewer in viewers]

        # Возвращение результатов в формате JSON

        return jsonify(result), 200

    except Exception as e:

        return jsonify({'error': str(e)}), 500

def not\_found(error):

    response = jsonify({

        "status": "error",

        "message": "Запрошенный URL-адрес не был найден на сервере."

    })

    response.status\_code = 404

    return response

def method\_not\_allowed(error):

    response = jsonify({

        "status": "error",

        "message": "Этот метод не разрешен для запрошенного URL-адреса."

    })

    response.status\_code = 405

    return response

Файл test\_main.py:

import requests

def login(username, password):

    url = 'http://127.0.0.1:5000/login'

    data = {'username': username, 'password': password}

    return requests.post(url, data=data, allow\_redirects=False)

def select\_film(film\_id):

    url = 'http://127.0.0.1:5000/reviews'

    data = {'film\_id': film\_id}

    return requests.post(url, data=data)

def test\_login\_success(start\_server):

    response\_success = login('Andrey', '1234')

    assert response\_success.status\_code == 302

    assert response\_success.headers['Location'] == '/index'

    print("Успех: Вход прошел успешно с правильными данными")

def test\_login\_fail(start\_server):

    response\_fail = login('Andrey123', '1234')

    assert response\_fail.headers.get('Location') is None

    print("Успех: Ошибка при заходе на страницу с неверными данными")

def test\_select\_titanic\_movie(start\_server):

    response\_titanic = select\_film('2213')

    assert response\_titanic.status\_code == 200

    assert '<title>Отзывы о фильме Титаник</title>' in response\_titanic.text

    print("Успех: Выбор фильма 'Титаник' обработан корректно")

def test\_select\_invalid\_movie(start\_server):

    response\_invalid = select\_film('999999999999999999999999999999999999999')

    assert 'Фильм не найден' in response\_invalid.text

    print("Успех: Обработка неверного film\_id выполнена корректно")

def test\_not\_found\_url(start\_server):

    url = 'http://127.0.0.1:5000/nosrt'

    response = requests.get(url)

    assert response.status\_code == 404

    assert response.json() == {

        "message": "Запрошенный URL-адрес не был найден на сервере.",

        "status": "error"

    }

    print("Успех: Обработка несуществующего URL выполнена корректно")

Файл reviews.py:

import requests

from bs4 import BeautifulSoup

from database import execute\_query,create\_tables

import sys

import os

import config

from datetime import datetime

# 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

        if not os.path.exists('reviews.db'):

            create\_tables()  # Если файл базы данных не существует, создаем таблицы

    # Словарь для соответствия названий месяцев и их номеров

    months\_ru\_to\_en = {

        'января': 'January',

        'февраля': 'February',

        'марта': 'March',

        'апреля': 'April',

        'мая': 'May',

        'июня': 'June',

        'июля': 'July',

        'августа': 'August',

        'сентября': 'September',

        'октября': 'October',

        'ноября': 'November',

        'декабря': 'December'

    }

    # Метод для получения и сохранения отзывов

    def fetch\_reviews(self):

        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):

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

                    review\_date\_str = review.find('span', class\_='date').text

                    # Заменяем русские названия месяцев на английские

                    for month\_ru, month\_en in self.months\_ru\_to\_en.items():

                        review\_date\_str = review\_date\_str.replace(month\_ru, month\_en)

                    review\_date\_format = "%d %B %Y | %H:%M"  # Предполагаемый формат даты

                    review\_date = datetime.strptime(review\_date\_str, review\_date\_format)

                    film\_id = self.film\_id

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

     # Метод для создания URL страницы с отзывами

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

        return f'{config.SITE\_URL}/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, review\_date):

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

    # Метод для сохранения фильма в SQL

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

        execute\_query('INSERT OR IGNORE INTO films (id, name) VALUES (?, ?)', (film\_id, film\_name))

# 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\_\_":

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

    main(film\_id)