ОЛИМПИАДА ШКОЛЬНИКОВ «ШАГ В БУДУЩЕЕ»

НАУЧНО-ОБРАЗОВАТЕЛЬНОЕ СОРЕВНОВАНИЕ «ШАГ В БУДУЩЕЕ, МОСКВА»

3376

регистрационный номер

Информатика и системы управления

название факультета

Программное обеспечение ЭВМ и информационные технологии (ИУ7)

название кафедры

Веб-приложение для поиска фильма по цитате.

название работы

Автор: **Сиденко Олег Генадьевич**

фамилия, имя, отчество

ГБОУ Школа №1571, 10 Б класс

наименование учебного заведения, класс

Научный руководитель: _____

фамилия, имя, отчество

_____

место работы

_____

звание, должность

_____

подпись научного руководителя

Москва - 2021

АННОТАЦИЯ

Целью данной проектной работы является создание сервиса поиска фильмов по фразе из них. В сервисе также реализовано добавление в избранное, авторизация, настройки пользователя, работа с Telegram.

Ключевые слова: Python, SQLite, Flask, Telegram.

Данный сервис реализуется на языке Python с использованием микрофреймворка Flask и модуля PyTelegramBotApi, а также технологий HTML, CSS, Javascript.

В тексте работы был рассмотрен алгоритм обработки запросов пользователя для устранения случайных ошибок: опечаток, неправильной пунктуации, смещения порядка слов, неправильной их формы.

Полученное в результате работы ПО может использоваться как часть веб сервисов по подбору фильмов, например как Kinopoisk.

Отчёт содержит 22 страницы, 8 рисунков, 3 таблицы, 16 источников, 1 приложение.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Поиск информации в интернете в наше время стал одной из самых обыденных вещей в повседневной жизни.

Но поиск зачастую не затрагивает некоторые ниши, которые бывают нужны при запросах в интернет, например, поиск фильмов по фразе, сказанной там. Например, вы не можете вспомнить название фильма, но помните цитату из него. Но если цитата не стала крылатым выражением или просто обсуждаемой фразой, вы не получите ответа на вопрос, что же это был за фильм. Не даст вам поиск ответа и в том случае, если цитата оборвана, или используется где-то кроме этого фильма, например в книге или статье.

Целью данной проектной работы является разработка программного обеспечения, позволяющего находить фильм по фразе или части фразы из него.

Для достижения данной цели потребуется решить такие задачи как.

— Создать базу данных фильмов и их текстов.

— Разработать алгоритм предобработки запроса пользователя и поиска информации по базе данных.

— Создать сайт, демонстрирующий работу системы.

1 **Основная часть**

1.1 Постановка задачи

Требуется разработать приложение, которое обладает следующей функциональностью.

— Авторизация пользователя.

— Поиск фильмов по фразе.

— Сохранение истории поиска.

— Возможность добавления фильмов в избранное.

— Телеграмм-бот для двухфакторной авторизации.

Система должна иметь веб- и телеграмм- интерфейс для поиска по базе фильмов.

1.2 Алгоритм

На вход подается текст, набранный пользователем, являющийся фрагментом текста одного из фильмов. Однако пользователь мог не точно запомнить фразу или сделать какую-либо ошибку для этого фразы в базе данных и пользовательский текст необходимо предварительно обработать.

Нам необходимо избавить от следующих типов ошибок пользователей.

— Некорректный регистр (например, имя).

— Неправильно расставленные знаки препинания (как пользователем, так и авторская пунктуация).

— Опечатки и орфографические ошибки.

Таким образом, был придуман метод поэтапной обработки текст, представленный на рисунке 1.1.

Рисунок 1.1 — Схема обработки запроса пользователя.

1.3 Выбор технологий

Для выполнения проекта был выбран язык программирования Python. Python [1] – это высокоуровневый язык программирования общего назначения, который используется в том числе и для разработки веб-приложений. Язык ориентирован на повышение производительности разработчика и читаемости кода.

Для разработки веб-приложения используется фреймворк Flask [2].

Flask – фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков – минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

В разработанном ПО были использованы следующие библиотеки.

1. Для разработки телеграмм приложения используется PyTelegramBotApi (telebot). PyTelegramBotApi [3] – обертка Telegram Bot Api [4] для Python.

2. Pymorphy2 [5] используется для приведения текста запросов в начальную форму.

3. SpellChecker [6] используется для устранения опечаток в словах.

Для утранения опечаток в Python существуют несколько библиотек, такие как Spellhecker, AutoCorrrect, JamSpell. JamSpell не поддерживается новыми версиями Python. Сравнение скорости работы библиотек Autocorrect и SpellChecker представлено в таблице 1.1. Время в таблице указано в формате: "минуты:секунды.микросекунды".

Таблица 1.1 — Сравнение библиотек коррекции опечаток.

	1 слово	10 слов	100 слов	1000 слов
SpellChecker	0:00.07	0:01.78	0:38.43	7:44.29
AutoCorrect	0:03.01	0:03.38	0:25.47	3:21.94

Исходя из этого была выбрана библиотекка SpellChecker, так как она работает быстрее на малых количествах слов, а запросы пользователей при поиске фильма небольшие по объему.

4. Random [7] используется для получения случайных чисел.

5. Hashlib [8] используется для шифровки паролей.

6. Datetime [9] используется для хранения времени сессии.

7. OS [10] используется для получения переменных окружения.

8. Requests [11] используется для отправки на сервер Flask запросов из Telegram.

Для работы веб сайта используется язык разметки HTML [12], таблица стилей CSS [13], скриптовый язык для страниц JavaScript [14].

1.4 Анализ типов и выбор СУБД

Как было замечено ранее, для реализации необходимо создать базу данных.

Для начала выберем: реляционная или нереляционная база данных, сравнение представлено в таблице 1.2.

Таблица 1.2 — Сравнение моделей баз данных.

	SQL	NoSQL
Структура и тип хранящихся данных	Однозначно определённая структура хранения данных	Ограничений нет
Запросы	Получение данных при помощи языка SQL	Каждая NoSQL база данных реализует свой способ работы с данными
Масштабируемость	Не подходит для постоянно меняющихся структур данных	Подходит для постоянно меняющихся структур данных
Поддержка	Развитая СУБД, наличие большого сообщества вокруг неё, множество примеров	Только недавно стала популярной

Так как, данные имеют четкую структуру и заранее определены, выбор в пользу реляционной базы данных.

Определимся с СУБД, основываясь на том факте, что для разработки был выбран ЯП Python.

— SQLite является самым простым способом работы с базами данных в Python, не требует сервера и установки модулей, т.к. sqlite3 встроен в Python

— MySQL и PostgreSQL требуют дополнительной установки и использования сервера MySQL.

Поэтому мой выбор пал на SQLite.

SQLite [15] — компактная встраиваемая реляционная база данных. Исходный код библиотеки передан в общественное достояние. Является чисто реляционной базой данных.

Преимуществом SQLite является скорость исполнения, простота кода, встраиваемость в Python.

1.5 База данных

На рисунке 1.2 представлена схема базы данных, а в таблице 1.3 указаны описания сущностей базы данных.

Таблица 1.3 — Описание сущностей базы данных

Users	Пользователи
Requests	Запросы пользователей
Telegram	Телеграмм профили пользователей
Liked	Понравившиеся пользователей
Films	Фильмы
Subtitles	Субтитры

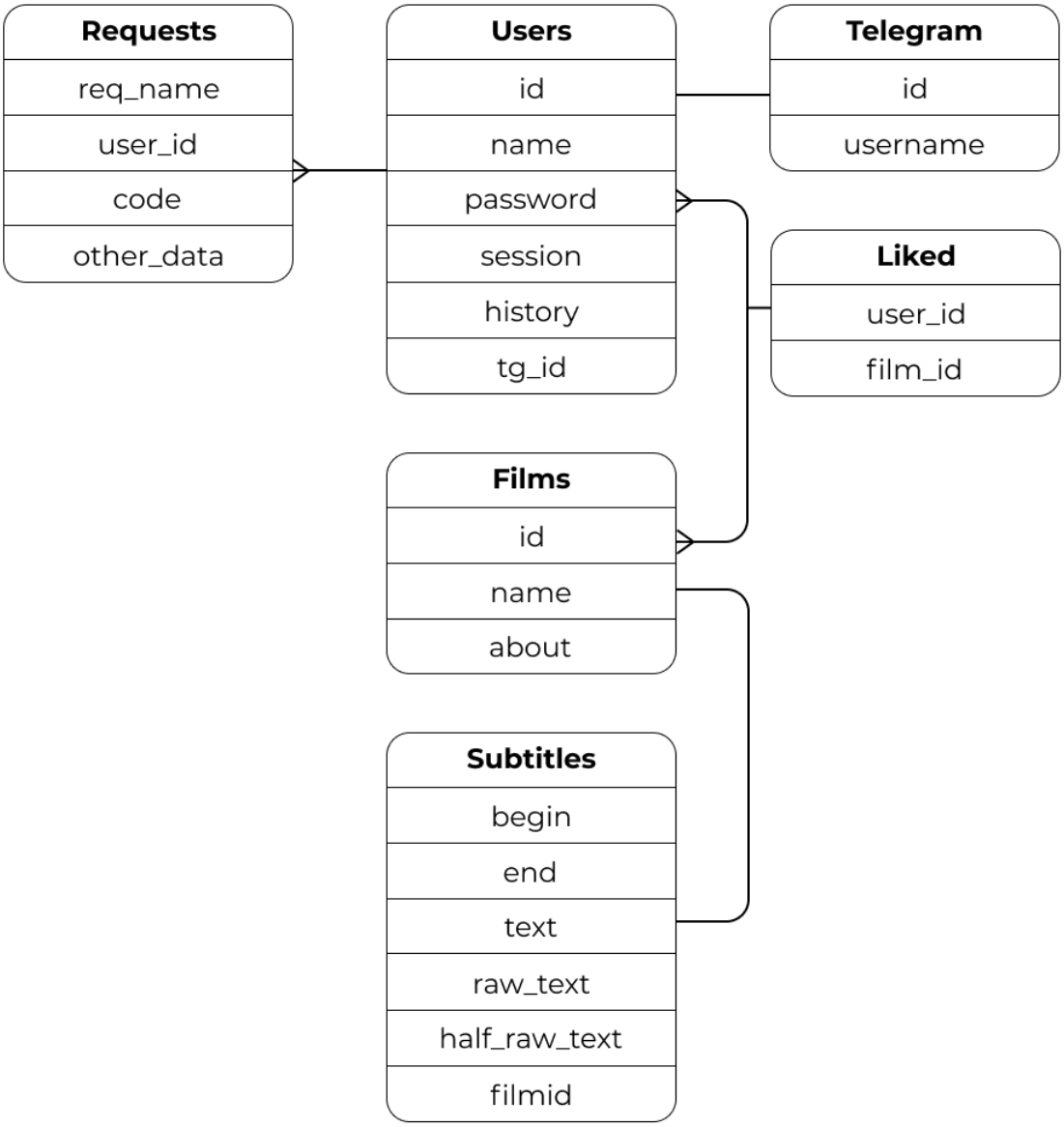

Рисунок 1.2 — Схема базы данных.

1.6 Текущее состояние базы данных и ее расширение

На данный момент в базе содержится 535 разных фильмов и 591274 строки с субтитрами. В телеграмм базе организован постраничный вывод по 5 фильмов, из-за ограничений площадки: сообщения обрезаются до 10000 символов. Таким образом, бот часто превышает лимит длины и вызывает проблемы с читабельностью.

Для получения субтитров был использован сайт с субтитрами: ”subtitry.ru” и парсер, код которого приведен в листинге А.1, в основе которого

лежат библиотеки BeautifulSoup [16] и SQLite. Субтитры обрабатывались по выше привденному алгоритму и записывались в базу данных. Для этого использовались библиотеки os и sqlite.

1.7 Разработка приложения

В разработанном сайте есть такие страницы как.

— Логин и регистрация. Также была реализована двухфакторная аутентификация через Telegram, пользователь сам решает будет ли он ее использовать: для этого необходимо привязать аккаунт телеграмма к профилю. В листинге А.2 представлен код регистрации пользователя на сайте.

— Главная страница, содержащая 3 варианта поиска:

— Стандартный поиск - самый точный поиск по исходным текстам фильмов: сохраняются раскладки, пунтктуация, формы слов.

— Полу-нечеткий поиск - поиск, убирающий пунктуацию и раскладку клавиатуры, но сохраняющий формы слов. Используется как основной, так как увеличивает количество результатов, но при этом не разрушает смысла цитаты.

— Нечеткий поиск - алгоритм данного поиска был представлен на рисунке 1.1. Он предоставляет самое большое количество результатов, но при этом часто жертвует точностью цитаты (пример этого вы можете увидеть в разделе 1.8).

— Результат поиска фразы, сделан с помощью шаблонизатора Jinja2. Его HTML код вы можете увидеть в листинге А.3.

— Понравившиеся фильмы, реализованы с использованием AJAX запросов. Их обработчик в Python, представлен в листинге А.4.

— Профиль пользователя, включающий в себя очистку истории поиска, смену пароля и привязку телеграмма, реализован также с ипользованиемм AJAX запросов.

Также был разработан Telegram бот.

— Он связывается с веб приложением при помощи мини-api. Пример их связи может быть просмотрен в листинге А.5.

— Также возможен поиск фильмов в интерфейсе Telegram.

1.8 Примеры работы сервиса

По запросу «Текст» было найдено 79 результатов с помощью полу-нечеткого поиска, результаты вы можете увидеть на рисунке 1.3.

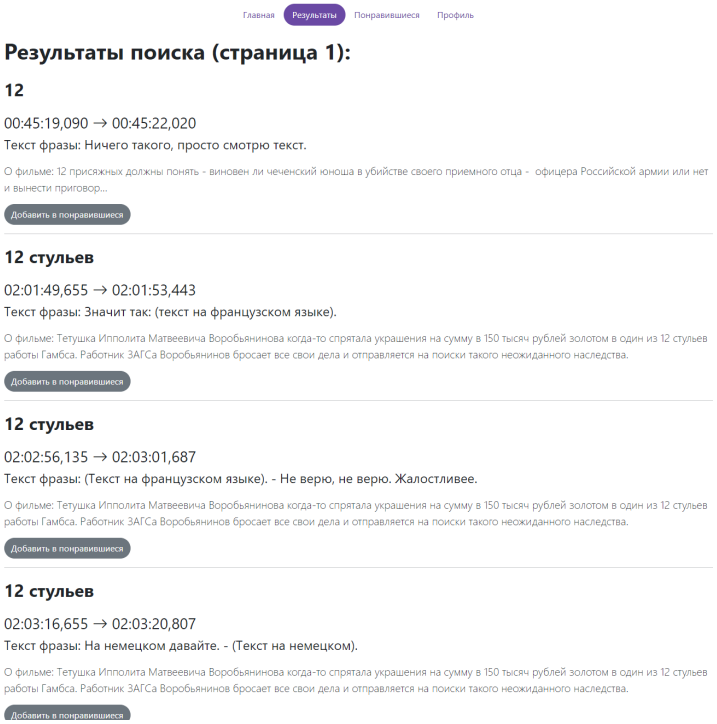

Рисунок 1.3 — Запрос «Текст».

По запросу «Роботал диньги» с помощью нечеткого поиска было найдено 56 вариантов. Ровно столько же было найдено по запросу «Работать деньги» с тем же поиском. Результаты данных запросов представлены на рисунках 1.4 и 1.5.

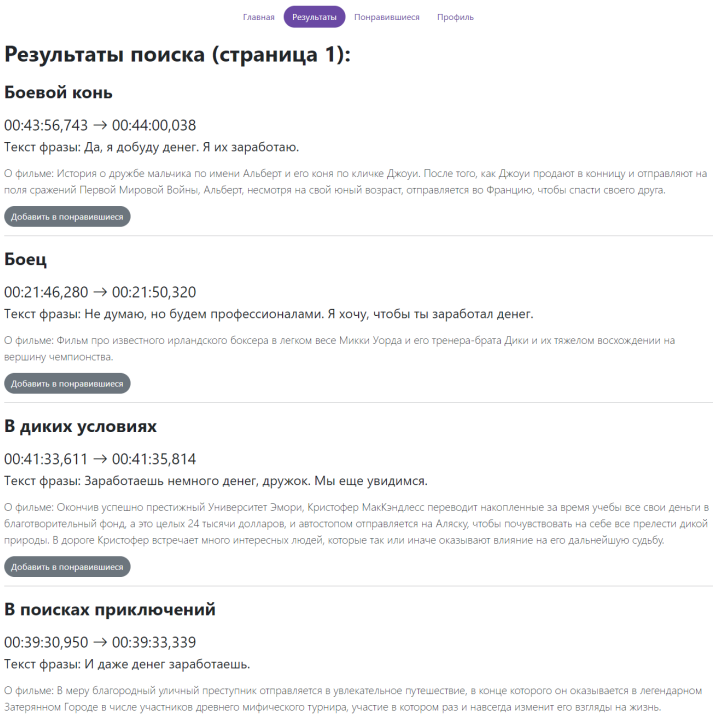

Рисунок 1.4 — Запрос «Роботал деньги».

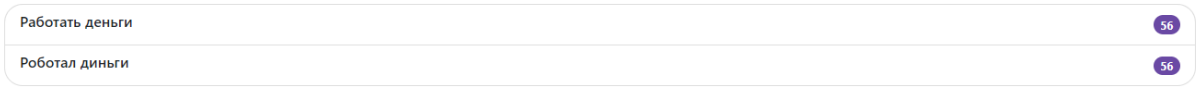

Рисунок 1.5 — Сравнение запросов.

Для показа разницы между 3 видами поиска, используем запрос "О, привет!". Его результаты представлены на рисунке 1.6. Запрос "О, привет!"содержит специфичный восклицательный знак в конце, и поэтому самый четкий поиск дал всего 3 результата. Средний по четкости поиск упразднил этот восклицательный знак и запятую, а также заглавную "О"превтратил в строчную и показал 57 результатов, а самый нечеткий смог убрать предлог "О"и получить 287 результатов.

Рисунок 1.6 — Запросы «О, привет!».

Пример работы Telegram бота представлен на рисунке 1.7.

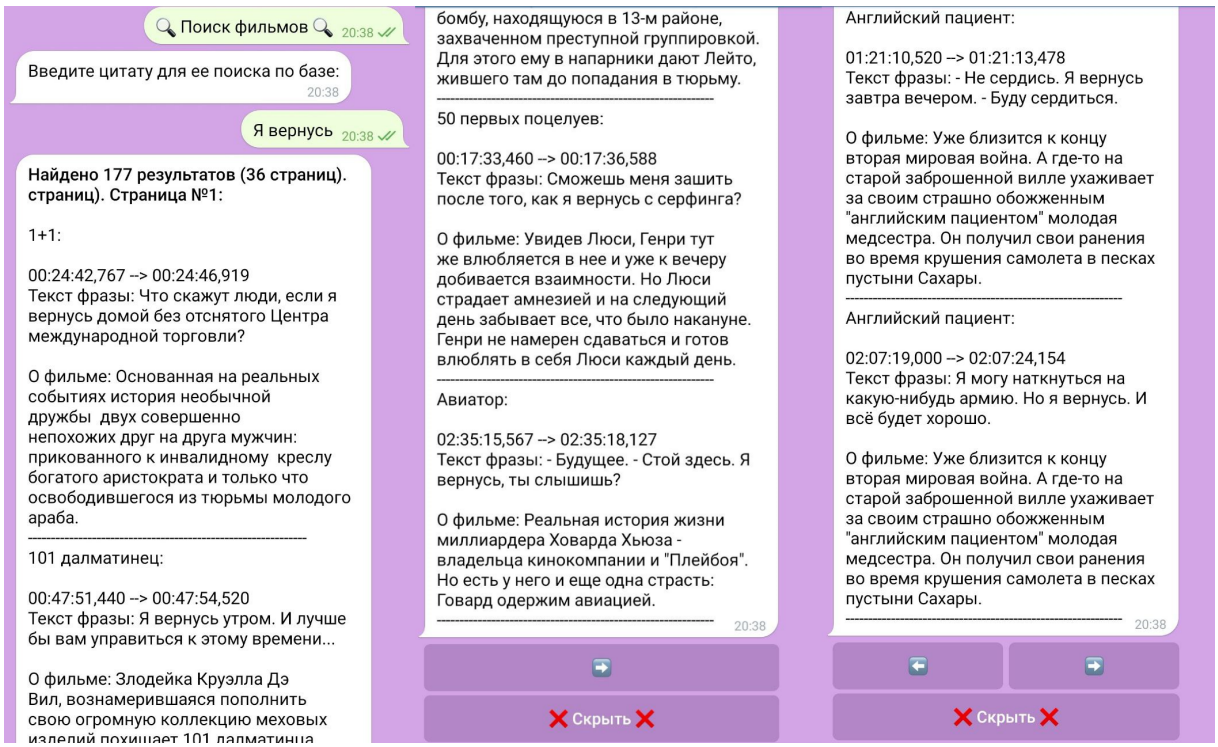

Рисунок 1.7 — Пример работы Telegram бота.

Настройка двухфакторной аутентификации 1.8.

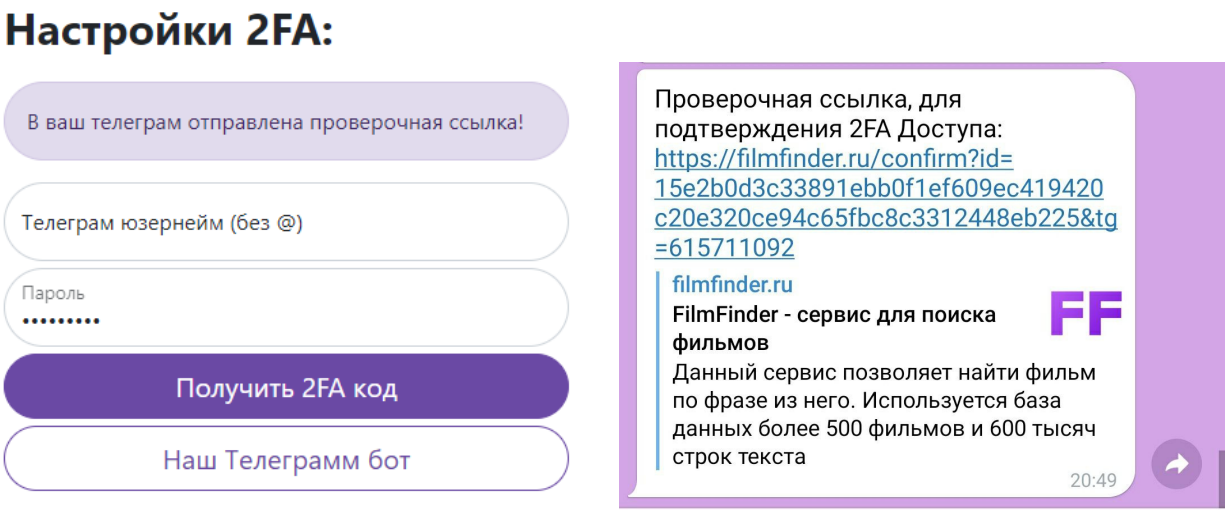

Рисунок 1.8 — Настройка двухфакторной аутентификации.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы достигнута цель работы: реализован сервис по поиску фильмов по фразе из него.

Выполнены такие задачи как.

— Регистрация пользователя.

— Поиск фильмов по фразе.

— Сохранение истории поиска.

— Возможность добавления фильмов в избранное.

— Телеграмм-бот для двухфакторной авторизации.

Реализован сервис на языке Python с использованием микрофреймворка Flask и технологий HTML, CSS, JavaScript.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Python документация. — Режим доступа: https://www.python.org/doc/ (дата обращения: 11.08.2021).

2. Flask документация. — Режим доступа: https://flask.palletsprojects.com/en/2.0.x/ (дата обращения: 11.08.2021).

3. Pytelegrambotapi. — Режим доступа: https://github.com/eternnoir/pyTelegramBotAPI (дата обращения: 3.11.2021).

4. Telegram api. — Режим доступа: https://core.telegram.org/bots/api (дата обращения: 3.11.2021).

5. Pymorphy. — Режим доступа: https://pymorphy2.readthedocs.io/en/stable/ (дата обращения: 22.10.2021).

6. SpellChecker. — Режим доступа: https://github.com/barrust/pyspellchecker (дата обращения: 22.10.2021).

7. Random. — Режим доступа: https://docs.python.org/3/library/random.html (дата обращения: 25.08.2021).

8. Hashlib. — Режим доступа: https://docs.python.org/3/library/hashlib.html (дата обращения: 24.08.2021).

9. Datetime. — Режим доступа: https://docs.python.org/3/library/datetime.html (дата обращения: 26.08.2021).

10. os. — Режим доступа: https://docs.python.org/3/library/os.html (дата обращения: 03.01.2022).

11. Requests. — Режим доступа: https://docs.python-requests.org/en/master/ (дата обращения: 03.01.2022).

12. HTML. — Режим доступа: https://devdocs.io/html/ (дата обращения: 12.08.2021).

13. CSS. — Режим доступа: https://devdocs.io/css/ (дата обращения: 12.08.2021).

14. JavaScript. — Режим доступа: https://devdocs.io/javascript/ (дата обращения: 14.10.2021).

15. SQLite. — Режим доступа: https://lecturesdb.readthedocs.io/databases/sqlite.html (дата обращения: 12.08.2021).

16. BeautifulSoup. — Режим доступа: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата обращения: 12.08.2021).

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ

Листинг А.1 — Код парсера фильмов

```
import sqlite3
import time
import requests
from bs4 import BeautifulSoup

headers = {
    'User-Agent': ('Mozilla/5.0 (Windows NT 6.0; rv:14.0)
       Gecko/20100101 '
                   'Firefox/14.0.1'),
    'Accept':
    'text/html,application/xhtml+xml,'\
    'application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language':
    'ru-ru,ru;q=0.8,en-us;q=0.5,en;q=0.3',
    'Accept-Encoding':
    'gzip, deflate',
    'Connection':
    'keep-alive',
    'DNT':
    '1'

}

conn = sqlite3.connect('date.db')
cur = conn.cursor()
cur.execute(f'SELECT * FROM films')
films = cur.fetchall()
for film in films:
    time.sleep(20)
    try:
        name = film[1].replace(' ', '+')
        url = f'https://subtitry.ru/subtitles/?film={name}'

        response = requests.get(url, headers=headers)
        result = BeautifulSoup(response.text, 'lxml')
```

```
35
36          try:
37              url = result.find_all('a')[66]['href']
38          except Exception as e:
39              print(f'No film: {name}')
40
41          if 'subtitles' in url:
42              response = requests.get(url, headers=headers)
43              result = BeautifulSoup(response.text, 'lxml')
44              film_link = result.find_all('form')[1]['action']
45              f = open(fr'film/{name}.zip', "wb")
46              a = 'https://subtitry.ru' + film_link
47              subtitles = requests.get(a, headers=headers)
48              f.write(subtitles.content)
49              f.close()
50              print(f'Film is done')
51          else:
52              print(f'No film: {name}')
53      except Exception as e:
54          print(f"Error: {str(e)}\nfilm: {name}")
```

Листинг А.2 — Регистрация пользователя

```
1  ip = request.remote_addr
2  q = request.form
3  cur.execute(f'SELECT * FROM users WHERE login = "{q["login"]}"')
4  user = cur.fetchall()
5  cur.execute(f'SELECT * FROM users WHERE session = "{ip}"')
6  res = cur.fetchall()
7  if res:
8      return render_template('register.html', error=4)
9  if user:
10     return render_template('register.html', error=1)
11 elif len(q['password']) < 8:
12     return render_template('register.html', error=2)
13 elif q['password'] != q['password-2']:
14     return render_template('register.html', error=3)
15 today = datetime.date.today()
16
17 pas = shifr(q['password'])
18 cur.execute('INSERT INTO users VALUES(?, ?, ?, ?, ?, ?, ?, ?);',
```

```
19                (q['login'], pas, ip, today, '', '', '', None))
20 conn.commit()
21 return redirect(url_for('index'))
```

Листинг А.3 — HTML шаблон результатов

```
1  <div class="container">
2
3      <p class="fs-1 fw-bold">Результаты поиска:</p>
4      {% if result %}
5      {% set a = {'num': 1} %}
6      {% for r in result %}
7      <p class="fs-2 fw-bold">{{ r[0] }}</p>
8      <div class="fs-3">{{ r[1] }} <i class="bi
           bi-arrow-right"></i> {{ r[2] }}</div>
9      <p class="fs-4">{{ 'Текст фразы: ' + r[3] }}</p>
10     <p class="fs-5 fw-light">{{ 'О фильме: ' + r[4] }}</p>
11
12     <form method="post" name="{{ r[6] }}" id="{{  'form' + r[6]
           }}">
13         {% if r[6] in like %}
14         <button type="submit" class="btn btn-outline-secondary"
               id="{{ 'add4d' + r[6] }}" name="hid-btn">Добавлено в п
               онравившиеся</button>
15         {% endif %}
16         {% if r[6] not in like %}
17         <button type="submit" class="btn btn-secondary" id="{{
               'add4' + r[6] }}" name="btn">Добавить в понравившиеся
               </button>
18         {% endif %}
19         <button type="submit" class="btn btn-outline-secondary"
               id="{{ 'added' + r[6] }}" name="hid-btn"
               style="display:none;">Добавлено в понравившиеся
               </button>
20         <button type="submit" class="btn btn-secondary" id="{{
               'add' + r[6] }}" name="btn"
               style="display:none;">Добавить в понравившиеся
               </button>
21         </form>
22     {% if a.update({'num': a.num + 1}) %} {% endif %}
23     <hr>
```

```
24      {% endfor %}
25      {% endif %}
26      {% if not result %}
27      <p class="fs-2 fw-bold">Ничего не найдено</p>
28      {% endif %}
29  </div>
```

Листинг А.4 — Приемщик Ajax запросов

```
1  usid = request.form['id']
2  conn = sqlite3.connect('users.db')
3  cur = conn.cursor()
4  ip = request.remote_addr
5  cur.execute(f'SELECT * FROM users WHERE session = "{ip}"')
6  res = cur.fetchone()
7  us = res[7]
8  cur.execute(f'SELECT * FROM liked WHERE userId = "{us}"')
9  res = cur.fetchone()
10 if not res:
11     cur.execute(f'INSERT INTO liked VALUES(?, ?, ?)',
12                 (None, us, f'{usid}'))
13     conn.commit()
14     return jsonify({'send': '1'})
15 elif not res[2]:
16     cur.execute(
17         f'UPDATE liked SET filmIds = "{usid}" WHERE userId =
               "{us}"')
18     conn.commit()
19     return jsonify({'send': '1'})
20 else:
21     if usid not in res[2].split('; '):
22         cur.execute(
23             f'UPDATE liked SET filmIds = "{res[2] + "; " +
                   usid}" WHERE userId = "{us}"')
24         conn.commit()
25         return jsonify({'send': '1'})
26     else:
27         films = res[2].split('; ')
28         films.remove(usid)
29         films = '; '.join(films)
30         cur.execute(
```

```
31                f'UPDATE liked SET filmIds = "{films}" WHERE userId
                      = "{us}"')
32            conn.commit()
33            return jsonify({'notsend': '1'})
```

Листинг А.5 — Связь приложений

```
1 us = message.chat.id
2 un = message.chat.username
3 res = requests.post(f'{link}/get_tg?username={un}&id={us}').text
4 if res == 'good':
5     bot.send_message(us, 'Вы уже записаны в базу, нет нужды добав
          лять еще раз')
6 elif res == 'bad':
7     bot.send_message(us, 'Ваш айди зарегистирован, можно подключа
          ть 2FA')
```