**Листинг программы:**

from flask import Flask, request, jsonify, abort

import requests

from bs4 import BeautifulSoup

import Levenshtein

import secrets

import hashlib

import logging

from cachetools import TTLCache, cached

from datetime import datetime

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

# Логирование

logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s %(message)s')

# Кэширование

cache = TTLCache(maxsize=100, ttl=3600)

# Список API токенов для авторизации

api\_tokens = []

app.config['api\_tokens'] = api\_tokens

def generate\_token(length=32):

alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

token = ''.join(secrets.choice(alphabet) for \_ in range(length))

return token

def hash\_token(token):

sha3\_512 = hashlib.sha3\_512()

sha3\_512.update(token.encode('utf-8'))

hashed\_token = sha3\_512.hexdigest()

return hashed\_token

def fetch\_site\_content(url):

headers = {

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'

}

try:

response = requests.get(url)

response.raise\_for\_status()

return response.text, response.status\_code

except requests.RequestException as e:

return str(e), getattr(e.response, 'status\_code', None)

def clean\_text(text):

return ''.join(text.split()).lower()

def find\_phrase(content, phrase):

cleaned\_content = clean\_text(content)

cleaned\_phrase = clean\_text(phrase)

min\_distance = len(cleaned\_phrase)

phrase\_found = False

for i in range(len(cleaned\_content) - len(cleaned\_phrase) + 1):

sub\_string = cleaned\_content[i:i + len(cleaned\_phrase)]

distance = Levenshtein.distance(sub\_string, cleaned\_phrase)

if distance <= 2:

phrase\_found = True

min\_distance = min(min\_distance, distance)

return phrase\_found, min\_distance

@app.route('/generate\_token', methods=['POST'])

def generate\_token\_endpoint():

token = generate\_token()

hashed\_token = hash\_token(token)

api\_tokens.append(hashed\_token)

app.logger.info(f"Generated token: {token} (hashed: {hashed\_token})")

app.logger.info(f"Current tokens: {api\_tokens}")

return jsonify({"token": token})

def log\_request(params, result):

logging.info(f"Params: {params} - Result: {result}")

@cached(cache)

def analyze\_website(url, phrase):

content, status\_code = fetch\_site\_content(url)

if status\_code != 200:

return {

"error": "Ошибка извлечения веб-страницы",

"http\_code": status\_code

}

soup = BeautifulSoup(content, 'html.parser')

text = soup.get\_text(separator=' ', strip=True)

phrase\_found, distance = find\_phrase(text, phrase)

return {

"phrase\_found": phrase\_found,

"levenshtein\_distance": distance

}

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

def analyze\_endpoint():

if 'Authorization' not in request.headers:

app.logger.error("Authorization header missing")

abort(401)

token = request.headers['Authorization']

hashed\_token = hash\_token(token)

if hashed\_token not in api\_tokens:

app.logger.error("Invalid token")

abort(403)

try:

data = request.get\_json()

if not data or 'url' not in data or 'phrase' not in data:

app.logger.error("Invalid JSON data")

abort(400)

except Exception as e:

app.logger.error(f"Failed to decode JSON object: {str(e)}")

abort(400)

url = data['url']

phrase = data['phrase']

app.logger.info(f"Analyzing URL: {url} with phrase: {phrase}")

result = analyze\_website(url, phrase)

log\_request({"url": url, "phrase": phrase}, result)

return jsonify(result)

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

app.run(threaded=True, debug=True)

**Описание кода программы:**

from flask import Flask, request, jsonify, abort

import requests

from bs4 import BeautifulSoup

import Levenshtein

import secrets

import hashlib

import logging

from cachetools import TTLCache, cached

from datetime import datetime

Эти строки импортируют необходимые библиотеки для работы с веб-запросами, создания веб-приложения, парсинга HTML, вычисления расстояния Левенштейна, генерации и хэширования токенов, логирования, кэширования и работы с датой и временем.

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

# Логирование

logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s %(message)s')

Создаёт экземпляр приложения Flask и настраивает логирование, чтобы все логи записывались в файл microservice.log.

# Кэширование

cache = TTLCache(maxsize=100, ttl=3600)

Создаёт кэш с максимальным размером 100 элементов и временем жизни кэша 3600 секунд (60 минут).

# Список API токенов для авторизации

api\_tokens = []

app.config['api\_tokens'] = api\_tokens

Создаёт пустой список для хранения хэшированных API токенов, используемых для авторизации.

def generate\_token(length=32):

alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

token = ''.join(secrets.choice(alphabet) for \_ in range(length))

return token

Функция generate\_token генерирует случайный токен длиной 32 символа, используя буквы и цифры.

def hash\_token(token):

sha3\_512 = hashlib.sha3\_512()

sha3\_512.update(token.encode('utf-8'))

hashed\_token = sha3\_512.hexdigest()

return hashed\_token

Функция hash\_token хэширует переданный токен с использованием алгоритма SHA-3 512 и возвращает хэш.

def fetch\_site\_content(url):

headers = {

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'

}

try:

response = requests.get(url)

response.raise\_for\_status()

return response.text, response.status\_code

except requests.RequestException as e:

return str(e), getattr(e.response, 'status\_code', None)

Функция fetch\_site\_content отправляет GET-запрос на указанный URL и возвращает содержимое страницы и статус-код ответа. В случае ошибки возвращает сообщение об ошибке и статус-код.

def clean\_text(text):

return ''.join(text.split()).lower()

Функция clean\_text удаляет пробелы и приводит текст к нижнему регистру для облегчения сравнения.

def find\_phrase(content, phrase):

cleaned\_content = clean\_text(content)

cleaned\_phrase = clean\_text(phrase)

min\_distance = len(cleaned\_phrase)

phrase\_found = False

for i in range(len(cleaned\_content) - len(cleaned\_phrase) + 1):

sub\_string = cleaned\_content[i:i + len(cleaned\_phrase)]

distance = Levenshtein.distance(sub\_string, cleaned\_phrase)

if distance <= 2:

phrase\_found = True

min\_distance = min(min\_distance, distance)

return phrase\_found, min\_distance

Функция find\_phrase ищет фразу в содержимом страницы, очищая текст и фразу, и вычисляет минимальное расстояние Левенштейна для фраз с допустимой погрешностью в 2 символа.

@app.route('/generate\_token', methods=['POST'])

def generate\_token\_endpoint():

token = generate\_token()

hashed\_token = hash\_token(token)

api\_tokens.append(hashed\_token)

app.logger.info(f"Generated token: {token} (hashed: {hashed\_token})")

app.logger.info(f"Current tokens: {api\_tokens}")

return jsonify({"token": token})

Маршрут /generate\_token генерирует новый API токен, хэширует его, добавляет в список токенов и возвращает сгенерированный токен в формате JSON.

def log\_request(params, result):

logging.info(f"Params: {params} - Result: {result}")

Функция log\_request записывает в лог параметры запроса и результаты обработки.

@cached(cache)

def analyze\_website(url, phrase):

content, status\_code = fetch\_site\_content(url)

if status\_code != 200:

return {

"error": "Ошибка извлечения веб-страницы",

"http\_code": status\_code

}

soup = BeautifulSoup(content, 'html.parser')

text = soup.get\_text(separator=' ', strip=True)

phrase\_found, distance = find\_phrase(text, phrase)

return {

"phrase\_found": phrase\_found,

"levenshtein\_distance": distance

}

Функция analyze\_website кэшируется и анализирует содержимое сайта, возвращая результат поиска фразы и расстояние Левенштейна.

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

def analyze\_endpoint():

if 'Authorization' not in request.headers:

app.logger.error("Authorization header missing")

abort(401)

token = request.headers['Authorization']

hashed\_token = hash\_token(token)

if hashed\_token not in api\_tokens:

app.logger.error("Invalid token")

abort(403)

try:

data = request.get\_json()

if not data or 'url' not in data or 'phrase' not in data:

app.logger.error("Invalid JSON data")

abort(400)

except Exception as e:

app.logger.error(f"Failed to decode JSON object: {str(e)}")

abort(400)

url = data['url']

phrase = data['phrase']

app.logger.info(f"Analyzing URL: {url} with phrase: {phrase}")

result = analyze\_website(url, phrase)

log\_request({"url": url, "phrase": phrase}, result)

return jsonify(result)

Маршрут /analyze проверяет авторизацию, принимает JSON-запрос с URL и фразой, анализирует содержимое сайта, логирует и возвращает результат анализа в формате JSON.

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

app.run(threaded=True, debug=True)

Запускает Flask приложение в многопоточном режиме с включённой отладкой.

**Как должен выглядеть запрос:**

curl -X POST http://127.0.0.1:5000/analyze -L -H "Content-Type: application/json" -H "Authorization: V0mkbcqu0OQvJRxLjJmBHMk60gBIpEJg" -H "User-Agent: Mozilla/5.0" -d "{\"url\": \"https://en.wikipedia.org/wiki/Levenshtein\_distance\", \"phrase\":

\"Levenshtein distance\"}"

Примечание: не используйте одинарные кавычки ‘’ в cmd, для экранирования используйте обратные слэши \ и двойные кавычки “”

**Как будет выглядеть ответ:**  
  
{

"levenshtein\_distance": 0,

"phrase\_found": true

}

**Unit-тесты:**

setUp метод:

* Этот метод выполняется перед каждым тестом.
* Создаётся тестовый клиент Flask.
* Генерируется и хэшируется токен, который добавляется в список api\_tokens.

test\_generate\_token тест:

* Отправляет POST-запрос на /generate\_token.
* Проверяет, что статус ответа 200 (OK).
* Проверяет, что в ответе содержится токен длиной 32 символа.

test\_analyze\_website тест:

* Отправляет POST-запрос на /analyze с валидным токеном и JSON-данными.
* Проверяет, что статус ответа 200 (OK).
* Проверяет, что в ответе содержатся поля phrase\_found и levenshtein\_distance.

test\_analyze\_website\_invalid\_token тест:

* Отправляет POST-запрос на /analyze с невалидным токеном.
* Проверяет, что статус ответа 403 (Forbidden).

test\_analyze\_website\_missing\_token тест:

* Отправляет POST-запрос на /analyze без токена.
* Проверяет, что статус ответа 401 (Unauthorized).

test\_analyze\_website\_invalid\_json тест:

* Отправляет POST-запрос на /analyze с невалидным JSON.
* Проверяет, что статус ответа 400 (Bad Request).

**Результаты тестирования:**

Testing started at 14:36 ...

Launching unittests with arguments python -m unittest

2024-07-09 14:36:18,232 INFO Analyzing URL: https://en.wikipedia.org/wiki/Levenshtein\_distance with phrase: Levenshtein distance

2024-07-09 14:36:18,849 INFO Params: {'url': 'https://en.wikipedia.org/wiki/Levenshtein\_distance', 'phrase': 'Levenshtein distance'} - Result: {'phrase\_found': True, 'levenshtein\_distance': 0}

2024-07-09 14:36:18,850 ERROR Failed to decode JSON object: 400 Bad Request: The browser (or proxy) sent a request that this server could not understand.

2024-07-09 14:36:18,850 ERROR Invalid token

2024-07-09 14:36:18,851 ERROR Authorization header missing

2024-07-09 14:36:18,851 INFO Generated token: 1EcuPcanHOO9HhZDd1RF5ehzcLcBQafz (hashed: b9916f726eb0d96dd44fb14e0a9c80ffbd349878f591eaf971593c4a25d18564664fbe978fc739af1b6c5217c6d9513a3987c1a9e4e8ee8a883afdc87eeb6601)

2024-07-09 14:36:18,851 INFO Current tokens: ['dcbc108349604defed573116ff0e4a327fc0debfde5f53ae74063417fa49f95ce793b1a725518758bc5b6190834287525005c15ecd19a1375c40894cfcaf7aed', '4fd1eb8900ba99ac5a0a87d42923a6d9d4f0a9376f72f2aafd222973cc85fc429d07eb88dc3866695a76729bf5842abfbebf33d5fff683f0ca9c65f1c03f734b', 'fff5f4e60fdab4694c658cf63859e4f252b1d9f5e697321a05bf21def355cd34688d2061bf3a1d63523afd828b6036600410845fdc077677e3749413893ac4ff', '2b82996ff4b6f5da775c995a68d36165bd3bfe609b9a39ac56f7a77c181486bd4fd77910bd421928a08b93179c590fded93b67db11ffb50146f880414db5adc1', '51c44d35d5cbee125fb80d700efbbccbe12a4c7fa81199d825e698be471f828b58bbe493e1892d9fce75946ff991337a2001dd6fbd8d64372295a290ca4455f2', 'b9916f726eb0d96dd44fb14e0a9c80ffbd349878f591eaf971593c4a25d18564664fbe978fc739af1b6c5217c6d9513a3987c1a9e4e8ee8a883afdc87eeb6601']

Ran 5 tests in 0.625s

OK

