МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра Автоматизированных систем управления

**ОТЧЕТ О ВЫПОЛНЕНИИ**

**РАСЧЁТНО-ГРАФИЧЕСКОГО РАБОТЫ**

по теме «*Описание библиотеки Requests для Python*»

Дисциплина «*Разработка программных приложений и WEB-программирование*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Выполнил:** |  | **Проверил:** |  | |
| Студент | *Бузмаков А.И.,* | Преподаватель | *Эстрайх И.В.* | |
| Факультет | *АВТФ* |  |  | |
| Направление (специальность) подготовки | *09.03.01 – Информатика и вычислительная техника* | Балл: \_\_\_\_\_\_\_\_\_ | |  |
| Группа | *АП-227* | Оценка *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | | |
| Шифр |  |  |  | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| подпись | | подпись | | |
| Дата сдачи: | «02» декабря 2024г. | Дата защиты: | «\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_. | |

Новосибирск 2024

**Оглавление**

[ВВЕДЕНИЕ 3](#_Toc184078267)

[ОСНОВНАЯ ЧАСТЬ 4](#_Toc184078271)

[ЗАКЛЮЧЕНИЕ 10](#_Toc184078277)

[СПИСОК ЛИТЕРАТУРЫ 11](#_Toc184078278)

[ПРИЛОЖЕНИЕ 12](#_Toc184078279)

# ВВЕДЕНИЕ

Введение в библиотеку «Requests» представляет собой погружение в мир сетевого программирования на Python. Библиотека «Requests» является одним из самых популярных и эффективных инструментов для работы с HTTP-запросами в экосистеме Python. Созданная с целью максимально упростить взаимодействие с веб-службами и API, она предоставляет разработчикам интуитивно понятный и элегантный способ отправки HTTP/1.1 запросов.

## Актуальность

В современном мире информационных технологий взаимодействие между различными системами и сервисами происходит преимущественно через сеть Интернет. Повсеместное распространение REST API, микросервисной архитектуры и облачных сервисов делает навыки работы с сетевыми запросами критически важными для любого разработчика программного обеспечения. Эффективное и безопасное взаимодействие с удаленными ресурсами становится ключевым фактором успеха при создании современных распределенных приложений, веб-сервисов и систем интеграции

## Цель работы

Целью данной работы является комплексное исследование библиотеки **«**Requests**»**, демонстрация ее возможностей, изучение механизмов работы и практическое применение в различных сценариях сетевого программирования.

## Задачи

1. Провести детальный анализ функционала библиотеки Requests, раскрыть ее основные возможности и механизмы работы с HTTP-запросами.
2. Продемонстрировать практические примеры использования различных методов библиотеки с реальными сценариями взаимодействия с API.
3. Выполнить сравнительный анализ библиотеки Requests с альтернативными решениями для работы с сетевыми запросами в Python.
4. Создать демонстрационный скрипт, показывающий основные возможности библиотеки и служащий практическим пособием по её использованию.
5. Сделать выводы.

# ОСНОВНАЯ ЧАСТЬ

## Скачивание и установка библиотеки

Установка библиотеки Requests является предельно простым процессом благодаря системе управления пакетами pip, которая входит в стандартную поставку Python. Существует несколько способов установки:

Стандартная установка через pip:

|  |
| --- |
| *pip install requests* |

Установка в виртуальном окружении позволяет изолировать зависимости проекта:

|  |
| --- |
| *python -m venv venv*  *source venv/bin/activate # Для Unix-подобных систем*  *venv\Scripts\activate # Для Windows*  *pip install requests* |

При работе над крупными проектами рекомендуется использовать виртуальные окружения для управления зависимостями и предотвращения конфликтов между пакетами.

## Возможности библиотеки

Requests предоставляет широкий спектр возможностей для работы с HTTP-протоколом. Разработчики могут отправлять запросы различных типов, включая GET, POST, PUT, DELETE, HEAD, OPTIONS и PATCH. Библиотека поддерживает работу с параметрами запросов, заголовками, Cookie-файлами и сессиями.

Особенно полезными являются встроенные механизмы обработки ответов. Requests автоматически декодирует содержимое ответа, поддерживает работу с JSON, умеет обрабатывать различные кодировки и предоставляет удобные методы для извлечения данных.

Библиотека также обеспечивает простую аутентификацию, поддерживает прокси-серверы, позволяет загружать и отправлять файлы, а также имеет встроенные средства обработки ошибок и таймаутов.

## Функции библиотеки

Библиотека Requests предоставляет широкий спектр возможностей для работы с HTTP-протоколом, которые значительно упрощают процесс сетевого взаимодействия в Python. Рассмотрим подробно основные возможности с практическими примерами.

**Типы HTTP запросов**

Requests поддерживает все основные типы HTTP-методов:

1. GET-запросы для получения данных

|  |
| --- |
| response = requests.get('https://api.example.com/users') |

1. POST-запросы для создания новых ресурсов

|  |
| --- |
| new\_user = {'name': 'John Doe', 'email': 'john@example.com'}  response = requests.post('https://api.example.com/users', json=new\_user) |

1. PUT-запросы для полного обновления ресурса

|  |
| --- |
| updated\_user = {'name': 'John Smith', 'email': 'john.smith@example.com'}  response = requests.put('https://api.example.com/users/123', json=updated\_user) |

1. DELETE-запросы для удаления ресурсов

|  |
| --- |
| response = requests.delete('https://api.example.com/users/123') |

**Работа с параметрами запроса**

Requests позволяет легко передавать параметры в URL:

|  |
| --- |
| params = {      'page': 1,      'limit': 10,      'sort': 'name'  }  response = requests.get('https://api.example.com/users', params=params) |

**Пользовательские заголовки**

Имеется возможность установки кастомных заголовков:

|  |
| --- |
| headers = {      'Authorization': 'Bearer token123',      'User-Agent': 'MyApp/1.0',      'Content-Type': 'application/json'  }  response = requests.get('https://api.example.com/secure', headers=headers) |

**Работа с cookies**

Автоматическая обработка Cookie-файлов:

|  |
| --- |
| session = requests.Session()  session.get('https://example.com/login')  session.post('https://example.com/profile', data={'update': 'data'}) |

**Загрузка файлов**

Простая загрузка файлов на сервер:

|  |
| --- |
| files = {      'document': **open**('report.pdf', 'rb'),      'image': **open**('screenshot.png', 'rb')  }  response = requests.post('https://api.example.com/upload', files=files) |

**Обработка ответов**

Удобные методы работы с ответами:

|  |
| --- |
| response = requests.get('https://api.example.com/data')  *# Получение статус-кода*  **print**(response.status\_code)  *# Получение текста ответа*  **print**(response.text)  *# Автоматический парсинг JSON*  data = response.json()  *# Получение заголовков ответа*  **print**(response.headers) |

**Обработка ошибок и таймауты**

Встроенные механизмы обработки ошибок и установки таймаутов:

|  |
| --- |
| try:      response = requests.get('https://api.example.com/data', timeout=5)      response.raise\_for\_status()  *# Вызывает исключение для плохих статус-кодов*  except requests.exceptions.RequestException as e:  **print**(f'Ошибка при выполнении запроса: {e}') |

**Прокси-серверы**

Поддержка работы через прокси:

|  |
| --- |
| proxies = {      'http': 'http://10.10.1.10:3128',      'https': 'http://10.10.1.10:1080'  }  response = requests.get('https://example.com', proxies=proxies) |

**Аутентификация**

Встроенные механизмы аутентификации:

|  |
| --- |
| *# Базовая аутентификация*  response = requests.get('https://api.example.com/secure',                          auth=('username', 'password'))  *# OAuth-аутентификация*  response = requests.get('https://api.example.com/user',                          headers={'Authorization': 'Bearer access\_token'}) |

Продемонстрированные возможности показывают, насколько универсальной и удобной является библиотека Requests для работы с HTTP-запросами в Python.

## Достоинства и недостатки библиотеки

Достоинства библиотеки Requests заключаются в ее простоте и удобстве использования. Интуитивный интерфейс позволяет быстро писать код для работы с сетевыми запросами. Автоматическая обработка кодировки и встроенная поддержка JSON существенно упрощают работу разработчика.

Вместе с тем, библиотека имеет и некоторые ограничения. Она не поддерживает нативно HTTP/2, что может быть критично для высокопроизводительных приложений. Requests не является самой быстрой библиотекой для выполнения большого количества параллельных запросов, что может быть недопустимо в определенных сценариях.

Для асинхронных приложений могут потребоваться дополнительные решения, такие как async-requests или переход на более современные библиотеки, поддерживающие асинхронность из коробки.

## Аналоги библиотеки

Для сравнения библиотеки Requests с аналогами представлена следующая таблица:

Таблица 1 - Сравнение основных характеристик библиотеки Requests и ее аналогов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | requests | urllib3 | aiohttp | httpx |
| Простота использования | Высокая | Средняя | Средняя | Высокая |
| Асинхронность | Нет | Нет | Да | Да |
| Производительность | Средняя | Высокая | Высокая | Высокая |
| Поддержка HTTP/2 | Нет | Нет | Нет | Да |
| Встроенная JSON-обработка | Да | Нет | Нет | Да |
| Работа с сессиями | Да | Да | Да | Да |

Проанализировав таблицу, можно сказать, что каждая библиотека имеет свои преимущества и подходит для различных сценариев использования, поэтому выбор зависит от конкретных требований проекта.

# ЗАКЛЮЧЕНИЕ

Библиотека Requests остается одним из самых популярных и удобных инструментов для работы с HTTP-запросами в Python. Ее простота, читаемость и богатый функционал делают ее незаменимым инструментом как для начинающих, так и для опытных разработчиков.

Несмотря на наличие альтернатив, Requests продолжает занимать прочные позиции благодаря своей универсальности и продуманному дизайну API. Для большинства задач, связанных с сетевым взаимодействием, она остается оптимальным выбором.

# СПИСОК ЛИТЕРАТУРЫ

1. Requests: HTTP for Humans – Requests 2.32.3 documentation [Электронный ресурс]. — Kenneth Reitz, 29 мая 2024. URL: https://requests.readthedocs.io/en/latest/ (дата обращения: 28.11.2024)

# ПРИЛОЖЕНИЕ

Приложение 1 – Листинг программы

|  |
| --- |
| requests-demo.py |
| import **requests**  def **get\_weather**(city, lang='ru'):      """      Получение погоды для указанного города        :param city: Название города      :param lang: Язык ответа (ru, en и др.)      :return: Словарь с информацией о погоде      """      try:  *# Формирование URL с параметрами*          url = f'https://wttr.in/{city}?format=j1&lang={lang}'    *# Установка пользовательского заголовка*          headers = {              'User-Agent': 'PythonWeatherApp/1.0',              'Accept': 'application/json'          }    *# Выполнение GET-запроса с таймаутом*          response = **requests**.**get**(url, headers=headers, timeout=10)    *# Проверка успешности запроса*          response.**raise\_for\_status**()    *# Парсинг JSON-ответа*          weather\_data = response.**json**()          return weather\_data        except **requests**.**exceptions**.**RequestException** as e:  **print**(f'Ошибка при получении погоды: {e}')          return None  def **display\_weather**(weather\_data):      """      Вывод информации о погоде        :param weather\_data: Словарь с данными о погоде      """      if not weather\_data:  **print**("Не удалось получить данные о погоде")          return        current = weather\_data['current\_condition'][0]      location = weather\_data['nearest\_area'][0]    **print**(f"Погода в городе {location['areaName'][0]['value']}:")  **print**(f"Температура: {current['temp\_C']}°C")  **print**(f"Ощущается как: {current['FeelsLikeC']}°C")  **print**(f"Влажность: {current['humidity']}%")  **print**(f"Скорость ветра: {current['windspeedKmph']} км/ч")  **print**(f"Осадки: {current['weatherDesc'][0]['value']}")  def **get\_weather\_for\_multiple\_cities**():      """      Демонстрация запросов для нескольких городов      """      cities = ['Moscow', 'Berlin', 'New York', 'Beijing']    *# Получение погоды для нескольких городов*      for city in cities:  **print**(f"\n{'='\*20}")          weather = **get\_weather**(city)          if weather:  **display\_weather**(weather)  def **main**():  *# 1. Получение погоды для одного города с обработкой ошибок*      moscow\_weather = **get\_weather**('Novosibirsk')      if moscow\_weather:  **display\_weather**(moscow\_weather)    *# 2. Получение погоды для нескольких городов*  **get\_weather\_for\_multiple\_cities**()    if \_\_name\_\_ == "\_\_main\_\_":  **main**() |

Приложение 2 – Скриншот работы программы

