

Flask

Знакомство: Werkzeug, Flask

- Flask v1.1.x
 - Python ≥ 3.8
-



На этом уроке

1. Рассмотрим азы работы с Werkzeug и Flask.
2. Попробуем на практике часто используемые инструменты.
3. Создадим базовый view.
4. Поработаем с объектами request, g.
5. Познакомимся с обработкой и передачей ошибок и логгированием.
6. Обработаем входные данные.

Оглавление

[О курсе](#)

[Содержание](#)

[Практическое задание \(проект\)](#)

[Знакомство: Werkzeug, Flask](#)

[HTTP](#)

[Коды ответа \(состояния\) HTTP](#)

[Введение в Werkzeug](#)

[WSGI](#)

[Werkzeug](#)

[XSS](#)

[Jinja2](#)

[Flask](#)

[Установка и настройка проекта](#)

[Первый запуск](#)

[Variable Rules](#)

[Context Locals \(контекст-локальные переменные\)](#)

[Объект request](#)

[Передача данных query string](#)

[Обрабатываем параметры запроса \(query string\)](#)

[Обрабатываем тело запроса и возвращаем кастомные статус-коды](#)

[Обработчики before_request, after_request: объект g](#)

[Логгер, обработка исключений](#)

[Обработка непредвиденных исключений](#)

[Итоги](#)

[Практическое задание](#)

О курсе

Содержание

Приветствуем вас на курсе по Flask. В нём мы рассмотрим наиболее актуальные технологии разработки веб-приложений.

Курс ориентирован на людей, которые хотят научиться создавать свои веб-приложения и реализовывать интеграции с другими сервисами. Вы узнаете, как устроены сайты изнутри, познакомитесь с системами авторизации, способами ограничения прав доступа, создания связей между моделями в базах данных.

За время курса мы с нуля создадим сайт для публикации статей, дадим пользователям возможность регистрироваться, создавать и публиковать собственные статьи. Также научимся работе с JSON API, узнаем, как происходит общение между сервисами, и проработаем защиту от недобросовестных пользователей (поговорим про OWASP).

Сейчас есть много сервисов для создания сайтов без навыков программирования. Но бывает так, что вы хотите сделать что-то уникальное либо просто добавить свои фишки в устоявшиеся категории проектов. В таком случае придётся разрабатывать сайт с нуля. Данный курс позволит вам полностью овладеть навыками, нужными для начала работы в этой сфере.

Практическое задание (проект)

По прохождении курса у студента будет запущен ресурс в общем доступе с возможностью публикации статей, исходный код которого можно будет посмотреть на GitHub. Данный проект отлично подойдёт для портфолио, так как показывает умение работать в различных аспектах веб-разработки с глубоким знанием бэкенда.

Задания составлены таким образом, что позволяют:

1. Применить и закрепить рассмотренный на занятии материал.
2. Создать следующую часть проекта.
3. Понять определённую логику и последовательность разработки проекта с самого начала.

На выходе мы получим современный проект, построенный с учётом актуальных технологий и принципов программирования.

Знакомство: Werkzeug, Flask

HTTP

Большинство разработчиков для создания веб-приложений используют HTTP протокол. URL-адрес, на который мы отправляем запрос, — это ресурс. Тип запроса определяет действие — что необходимо сделать с этим ресурсом.

Основные типы запросов и назначение каждого из них:

- GET — чтение данных;
- POST — создание;
- PUT — создание или полная замена;
- DELETE — удаление;
- OPTIONS — описание параметров соединения с целевым ресурсом;
- HEAD — чтение заголовков запросов (аналогично GET, но без тела);
- PATCH — частичная модификация.

Таким образом, чтобы посмотреть, какие методы доступны по адресу `/animals/`, можно отправить OPTIONS-запрос. Для просмотра списка животных — GET `/animals/`. Создать новое животное — POST `/animals/`. Посмотреть одно животное — GET `/animals/1/`. Для изменения животного — PUT `/animals/1/`. Удалить животное — DELETE `/animals/1/`. Если нам нужно просто проверить, доступен ли адрес, можем отправить на него HEAD-запрос.

HTTP-метод POST предназначен для отправки данных на сервер. Тип тела запроса указывается в заголовке Content-Type.

Разница между PUT и POST состоит в том, что PUT является идемпотентным: повторное его применение даёт тот же результат, что и при первом применении (то есть у метода нет побочных эффектов). Повторный вызов одного и того же метода POST может иметь такие эффекты, например, оформление одного и того же заказа несколько раз.

Запрос POST обычно отправляется через форму HTML и приводит к изменению на сервере. В этом случае тип содержимого выбирается путем размещения соответствующей строки в атрибуте `enctype` элемента `<form>` или `formenctype` атрибута элементов `<input>` или `<button>`:

- `application/x-www-form-urlencoded`: значения кодируются в кортежах с ключом, разделенных символом `'&'`, с `'='` между ключом и значением. Не буквенно-цифровые символы — percent encoded: по этой причине тип не подходит для использования с двоичными данными (вместо этого используйте `multipart/form-data`);
- `multipart/form-data`: каждое значение посылается как блок данных («body part») с заданным пользовательским клиентом разделителем («boundary»), разделяющим каждую часть. Эти ключи даются в заголовки Content-Disposition каждой части;
- `text/plain`.

Когда запрос POST отправляется с помощью метода, отличного от HTML-формы — например, через XMLHttpRequest, — тело может принимать любой тип. Как описано в спецификации HTTP 1.1, POST предназначен для обеспечения единообразного метода для покрытия следующих функций:

- аннотация существующих ресурсов;
- публикация сообщения на доске объявлений, в новостной группе, в списке рассылки или в аналогичной группе статей;
- добавление нового пользователя посредством модальности регистрации;
- предоставление блока данных, например результата отправки формы, процессу обработки данных;
- расширение базы данных с помощью операции добавления.

Коды ответа (состояния) HTTP

Код ответа (состояния) HTTP показывает, был ли успешно выполнен определённый HTTP запрос.

Коды сгруппированы в 5 классов ([источник](#)):

- информационные 100–199;
- успешные 200–299;
- перенаправления 300–399;
- клиентские ошибки 400–499;
- серверные ошибки 500–599.

Введение в Werkzeug

WSGI

WSGI (Web-Server Gateway Interface) является потомком CGI (Common Gateway Interface). Когда веб начал развиваться, CGI разрастался из-за поддержки огромного количества языков и из-за отсутствия других решений. Однако такое решение было медленным и ограниченным. WSGI был разработан как интерфейс для маршрутизации запросов от веб-серверов (Apache, Nginx и т.д.) на веб-приложения.

WSGI-серверы появились потому, что веб-серверы изначально не умели взаимодействовать с приложениями, написанными на языке Python.

Werkzeug

Werkzeug — набор инструментов WSGI, стандартного интерфейса Python для развёртывания веб-приложений и взаимодействия между ними и различными серверами разработки. Отвечает за роутинг, обработку запросов и ответов, а также предоставляет такие возможности, как debugger и reloader.

XSS

XSS (англ. Cross-Site Scripting — «межсайтовый скриптинг») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «Внедрение кода».

Специфика подобных атак заключается в том, что вредоносный код может использовать авторизацию пользователя в веб-системе для получения к ней расширенного доступа или для получения авторизационных данных пользователя. Вредоносный код может быть вставлен в страницу как через уязвимость в веб-сервере, так и через уязвимость на компьютере пользователя.

Для термина используют сокращение XSS, чтобы не было путаницы с каскадными таблицами стилей, использующими сокращение CSS.

Jinja2

Jinja2 — это движок шаблонов и одновременно современный язык шаблонов для Python, созданный по образцу шаблонов Django. Он быстр, широко используется и безопасен, благодаря дополнительной среде отрисовки шаблона в песочнице.

Возможности:

- песочница для исполнения;
- мощная автоматическая HTML-экранирующая система для предотвращения XSS;
- наследование шаблонов;

- компилируется до оптимального кода python just in time;
- опциональная досрочная компиляция шаблонов;
- легко отлаживается. Номера строк исключений непосредственно указывают на нужную строку в шаблоне;
- настраиваемый синтаксис.

Flask

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.

Установка и настройка проекта

Чтобы начать использовать Flask, установим его:

```
pip install Flask
```

В директории нашего проекта создаём модуль blog (папка blog и внутри файл `__init__.py`), который будем постепенно дополнять. Создаём файл `'app.py'`, добавляем в него следующее содержимое:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello web!"
```

Здесь мы:

- создаём Flask app, в него передаём имя текущего файла (таковы правила для создания основного приложения);
- создаем index view: обрабатываем обращение на корень сайта, отдаём обычный текст;
- при помощи декоратора `@app.route("/")` указываем, что данный view должен быть использован для обработки запроса на `/`, то есть корень сайта (или index view).

Первый запуск

Проверим, что получилось. Для этого в корне проекта создаём файл `wsgi.py` со следующим содержимым:

```
from blog.app import app

if __name__ == "__main__":
    app.run(
        host="0.0.0.0",
        debug=True,
    )
```

Через этот файл мы будем запускать наше приложение. При помощи `app.run` запуск выполняется силами встроенного в Werkzeug отладочного веб-сервера, который не подходит для продакшена.

Запускаем веб-сервер:

```
python wsgi.py
```

Переходим на <http://127.0.0.1:5000> (порт по умолчанию 5000). И видим, что нас встречает текст «Hello web!»

Variable Rules

Вы можете добавить переменные части в URL-адрес, пометив эти части в формате `<variable_name>`. Тогда функция будет получать `variable_name` в качестве аргумента ключевого слова. При желании вы можете использовать конвертер, чтобы указать тип аргумента, например `<converter:variable_name>`.

<code>string</code>	(по умолчанию) принимает текст без слеша
<code>int</code>	принимает позитивные целые числа
<code>float</code>	принимает позитивные числа с плавающей точкой
<code>path</code>	как <code>string</code> , но принимает слеша
<code>uuid</code>	принимает строки UUID

Добавим для примера ссылку с приветствием по имени, которое нужно будет вписать в путь:

```
@app.route("/greet/<name>/")
def greet_name(name: str):
    return f"Hello {name}!"
```

Теперь при переходе по ссылке <http://127.0.0.1:5000/greet/GitHub/> будет отображаться «Hello GitHub!»

Context Locals (контекст-локальные переменные)

Некоторые объекты в Flask являются глобальными объектами, но не обычного типа. Эти объекты — на самом деле прокси для объектов, которые являются локальными для определённого контекста. Кажется запутанным, но на самом деле это довольно легко понять. Контекст — это поток обработки запроса. Поэтому такой «глобальный» объект на самом деле является локальным для каждого отдельного запроса. Если вебсервер обрабатывает 1-2-5-10-100 запросов одновременно, всё равно для каждого потока (соответственно, для каждой view-функции, обрабатывающей запрос) будет создан уникальный локальный объект.

Объект request

Объект запроса, используемый по умолчанию в Flask, хранит endpoint, по которому эта view функция и была вызвана, а также прочие параметры запроса: запрашиваемый хост, адрес клиента, заголовки, метод и тело запроса и прочее. Объект запроса (request) является подклассом Request и предоставляет все атрибуты, определённые Werkzeug, плюс несколько специфичных для Flask.

Передача данных query string

Данные на сервер можно передавать через так называемую строку запроса query string. Это параметры ключ=значение, которые располагаются в самой ссылке запроса после вопросительного знака, например `/home?key=value`.

Обрабатываем параметры запроса (query string)

Для обработки параметров query string необходимо использовать объект request. Создадим новый view. Для обращения к query string необходимо использовать `request.args`. Мы используем `.get`, потому что с `request.args` нужно обращаться как со словарём.

```
from flask import request

@app.route("/user/")
def read_user():
    name = request.args.get("name")
    surname = request.args.get("surname")
    return f"User {name or '[no name]'} {surname or '[no surname]'}"
```

Проверяем:

```
> http://127.0.0.1:5000/user/
```

```
- User [no name] [no surname]
```

```
> http://127.0.0.1:5000/user/?name=John&surname=Smith
```

```
- User John Smith
```

Обрабатываем тело запроса и возвращаем кастомные статус-коды

Регистрируем новый view. Необходимо указать допустимые методы через именованную переменную `methods`. Через `request.method` проверяем, что метод GET. Если так, возвращаем инструкцию, как пользоваться данным endpoint. Если же метод POST, то выполняем обработку данных. Сначала проверяем `request.form`, а затем `request.json`. В обоих случаях работаем с объектом как со словарём. Проверяем наличие ключа `code`, и если он там присутствует, отдаём ответ с таким кодом. По умолчанию отдаём пустой ответ с кодом 204.

```
@app.route("/status/", methods=["GET", "POST"])
def custom_status_code():
    if request.method == "GET":
        return """\
        To get response with custom status code
        send request using POST method
        and pass `code` in JSON body / FormData
        """

    print("raw bytes data:", request.data)

    if request.form and "code" in request.form:
        return "code from form", request.form["code"]

    if request.json and "code" in request.json:
        return "code from json", request.json["code"]

    return "", 204
```

```
# GET
> curl --request GET --url http://127.0.0.1:5000/status/
- To get response with custom status code
  send request using POST method
  and pass `code` in JSON body / FormData
```

```
# POST (empty body)
> curl --request POST --url http://127.0.0.1:5000/status/
< HTTP/1.0 204 NO CONTENT
```

```
# POST (multipart/form-data)
> curl --request POST --url http://127.0.0.1:5000/status/ \
  --header 'Content-Type: multipart/form-data' \
  --form code=202
< HTTP/1.0 202
- code from form
```

```
# POST (json)
> curl --request POST --url http://127.0.0.1:5000/status/ \
  --header 'Content-Type: application/json' \
  --data '{"code": 205}'
< HTTP/1.0 205 RESET CONTENT
- code from json
```

Обработчики `before_request`, `after_request`; объект `g`

Добавляем обработчики, вызываемые до запроса и после. Используем объект `g`. Он, как и объект `request`, является локальным для текущего запроса. На него мы можем установить любые атрибуты.

```
from flask import g

@app.before_request
def process_before_request():
    """
    Sets start_time to `g` object
    """
    g.start_time = time()

@app.after_request
def process_after_request(response):
    """
    adds process time in headers
    """
    if hasattr(g, "start_time"):
        response.headers["process-time"] = time() - g.start_time

    return response
```

Например, посещаем `http://127.0.0.1:5000/greet/GitHub/`.

В ответе заголовки будут содержать `process-time: 0.0001270771026611328`.

Логгер, обработка исключений

Импортируем исключение `BadRequest` из `Werkzeug`. Оно нам понадобится для возврата статус-кода 400 и прерывания обработки текущего запроса.

```
from werkzeug.exceptions import BadRequest
```

Создаём view для возведения `x` в степень `y`. Добавляем обработку и валидацию входных данных, логгирование, выброс исключения:

```
@app.route("/power/")
def power_value():
    x = request.args.get("x") or ""
    y = request.args.get("y") or ""
    if not (x.isdigit() and y.isdigit()):
        app.logger.info("invalid values for power: x=%r and y=%r", x, y)
        raise BadRequest("please pass integers in `x` and `y` query params")

    x = int(x)
    y = int(y)
    result = x ** y
    app.logger.debug("%s ** %s = %s", x, y, result)
    return str(result)
```

```
# GET (no values passed) http://127.0.0.1:5000/power/
< HTTP/1.0 400 BAD REQUEST
- Bad Request
  please pass integers in `x` and `y` query params
[log]: INFO in app: invalid values for power: x='' and y=''
```

```
# GET (pass valid data) http://127.0.0.1:5000/power/?x=7&y=3
< HTTP/1.0 200 OK
- 343
[log]: DEBUG in app: 7 ** 3 = 343
```

Обработка непредвиденных исключений

Мы не можем заранее предвидеть все возможные исключения. Поэтому стоит добавить обработку вероятных исключений на самом верхнем уровне, чтобы, если мы их пропустим, в итоге всё же выполнить обработку.

Создаём view `do_zero_division`, который будет вызывать исключение. А также регистрируем обработчик `handle_zero_division_error` при помощи декоратора `@app.errorhandler(ZeroDivisionError)` (передаём туда исключение, которое собираемся ловить).

```
@app.route("/divide-by-zero/")
def do_zero_division():
    return 1 / 0

@app.errorhandler(ZeroDivisionError)
def handle_zero_division_error(error):
    print(error) # prints str version of error: 'division by zero'
    app.logger.exception("Here's traceback for zero division error")
    return "Never divide by zero!", 400
```

Проверяем. Выполняем переход на <http://127.0.0.1:5000/divide-by-zero/>. В ответ получаем:

```
< HTTP/1.0 400 BAD REQUEST
- Never divide by zero!
```

А в логах приложения видим:

```
Process unhandled zero division error
[log]: ERROR in app: Here's traceback for zero division error
Traceback (most recent call last):
  File "/usr/lib/python3.9/site-packages/flask/app.py", line 1950, in
full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/lib/python3.9/site-packages/flask/app.py", line 1936, in
dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)
  File "/apps/flask-lesson/blog/app.py", line 155, in do_zero_division
    return 1 / 0
ZeroDivisionError: division by zero
```

Итоги

На занятии мы поговорили про HTTP методы и статус коды, узнали, как создавать view в Flask, рассмотрели, как применять объекты request, g, как использовать исключения (BadRequest, NotFound, etc), как обрабатывать поступающие данные (request.args, request.json, request.form, request.data). Применили логгирование, обработку и передачу исключений.

Практическое задание

На протяжении всего курса в практическом задании мы по частям будем разрабатывать проект «блог с возможностью регистрации и публикации статей».

После каждого занятия:

- сразу будем применять полученные знания на практике;
- будем добавлять в проект новые возможности, относящиеся к теме занятия.

Последовательность действий:

1. Создайте новый проект на GitHub.
2. Начните новый проект и установите Flask.
3. Создайте базовый index view для обработки посещений на корень сайта.
4. Сдайте работу в виде ссылки на репозиторий с кодом.

Дополнительные материалы

1. [Flask Quickstart](#).
2. List of HTTP status codes ([wiki](#), [mozilla](#)).
3. [JavaScript Object Notation \(JSON\)](#).
4. [HTTP request methods](#).
5. [Flask app.logger](#).
6. [Flask Response](#).
7. [Wiki REST](#).
8. [HTTP Methods](#).
9. https://en.wikipedia.org/wiki/List_of_HTTP_status_codes.
10. [Flask.logger](#).

Используемые источники

1. [Методы HTTP и их назначение.](#)
2. <https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/POST>.
3. https://en.wikipedia.org/wiki/List_of_HTTP_status_codes.
4. <https://flask.palletsprojects.com/en/1.1.x/quickstart/>.
5. [REST и HTTP.](#)
6. [Flask Quickstart.](#)
7. Miguel Grinberg. Flask Web Development.