Flask | Я.Шпора

Установка Flask

```
$ pip install flask==3.0.3
```

Минимальный проект на Flask

Минимальная структура

Нужно создать отдельную директорию. В ней — модуль приложения app.py:

```
from flask import Flask


app = Flask(__name__)


@app.route('/')
def the_view():
    return 'ok'


if __name__ == '__main__':
    app.run()
```

Настройка режима работы

Режим по умолчанию — production, для разработки нужно поменять его на режим debug:

```
# Linux и MacOS.
$ export FLASK_DEBUG=1

# Windows.
$ set FLASK_DEBUG=1
```

Команда запуска

Рекомендованный вызов:

```
$ flask run
```

Если модуль приложения не app.py:

```
# Задать переменную окружения в Linux и MacOS.
$ export FLASK_APP=theapp
# В Windows.
$ set FLASK_APP=theapp

# Запустить приложение.
$ flask run
```

Прямой вызов:

```
$ python theapp.py
```

Адрес сайта

```
http://127.0.0.1:5000/
```

Работа с .env

Для работы необходимо установить библиотеку python-dotenv:

```
pip install python-dotenv==1.0.1
```

Если модуль приложения называется the_app.py, то содержимое файла .env будет таким:

```
FLASK_APP=the_app
FLASK_DEBUG=1
```

Переменные окружения будут применяться при запуске приложения любым способом.

Типичный проект на Flask

Установить библиотеку Flask-Migrate:

```
pip install flask-migrate==4.0.7
```

Структура

```
├── dir_app
    ├── instance/
    ├── migrations/
    ├── the_app
        ├── static/
        ├── templates/
        ├── __init__.py
        ├── cli_commands.py
        ├── the_app.sqlite3
        ├── error_handlers.py
        ├── forms.py
        ├── models.py
        └── views.py
    ├── .env
    └── settings.py
```

Настройки окружения

Содержимое файла .env:

```
FLASK_APP=the_app
FLASK_DEBUG=1
SQLALCHEMY_DATABASE_URI=sqlite:///the_app.sqlite3
SECRET_KEY=THE_SECRET_KEY
```

Содержимое файла settings.py:

```
import os
```

```
class Config:
    SECRET_KEY = os.getenv('SECRET_KEY')
    SQLALCHEMY_DATABASE_URI = os.getenv('SQLALCHEMY_DATABASE_URI')


config = Config()
```

Содержимое файла the_app/__init__.py:

```
from flask import Flask
from flask_migrate import Migrate
from flask_sqlalchemy import SQLAlchemy

from settings import config


app = Flask(__name__)
app.config.from_object(config)
db = SQLAlchemy(app)
migrate = Migrate(app, db)


from import cli_commands, error_handlers, views, api_views
```

Подключение БД через Flask-SQLAchemy

Установка

```
$ pip install flask-sqlalchemy==3.1.1
```

В проекте

```
from flask_sqlalchemy import SQLAlchemy
```

```
app = Flask(__name__)
db = SQLAlchemy(app)
```

Магическая строка подключения

```
# Общий формат
dialect+driver://username:password@host:port/database

# Для абсолютного адреса в ОС Windows
sqlite:///p:\a\t\h\db.sqlite3

# Для абсолютного адреса в ОС Unix/Mac
sqlite:////username/path/db.sqlite3

# Для относительного адреса в ОС Unix/Mac/Windows
sqlite:///db.sqlite3
```

Описания моделей в Flask-SQLAchemy

```
...
db = SQLAlchemy(app)


class TheModel(db.Model):
    the_field = db.Column(
      тип, # из db.Integer, db.String(1234), db.Text, db.DateTi
me, ...
      primary_key=Нет/Да,
      unique=Нет/Да,
      nullable=Да/Нет,
      index=Нет/Да,
      default=...,
      ...
      )
```

Работа во Flask Shell

Запуск Flask Shell:

```
$ flask shell
```

Создание объекта модели:

```
>>> o = TheModel(поля)
>>> db.session.add(o)
>>> db.session.commit()
```

Создание базы, а именно создание файла, если его не было, и таблиц в нём:

```
>>> db.create_all()
```

Удаление таблиц, файл БД при этом останется:

```
>>> db.drop_all()
```

ORM-команды для моделей

Создание таблицы:

```
o = TheModel(поля); db.session.add(o)[; db.session.commit()]
```

Удаление таблицы:

```
db.session.delete(o)[;db.session.commit()]
```

> Значения для `primary_key` и `DateTime` с `default=now` появятся после `.commit()`.

Отмена изменений, произошедших после последнего `.commit()`:

```
db.session.rollback()
```

Объём таблицы или выборки:

```
TheModel.query.count()
```

Извлечение всех объектов:

```
os = TheModel.query.all()
```

Извлечение первых трёх объектов после первых двух:

```
os = TheModel.query.offset(2).limit(3).all()
```

Извлечение набора объектов **по значениям** полей:

```
os = TheModel.query.filter_by(поле=значение, ...).all()
```

Извлечение набора объектов **по условиям** для полей:

```
os = TheModel.query.filter(TheModel.поле.условие(...), ...).all()
```

Извлечение первого объекта из выборки. Для пустой выборки устанавливается значение `None`:

```
o = TheModel.query.first()
o = TheModel.query.first_or_404()
```

Извлечение объекта по ключу. При промахе устанавливается значение `None`:

```
o =  TheModel.query.get(ключ)
o =  TheModel.query.get_or_404(ключ)
```

Извлечение случайного объекта из всех:

```
o = TheModel.query.offset(randrange(TheModel.query.count())).first
```

Роутинг

Извлечение параметров из урла при реакции

```
@app.route('/the_path/<тип:имя>')  # Завершающий / необязателен
def the_view(имя):
    ...
```

Где «тип» это:

- `string` — строка без `/`, устанавливается по умолчанию;
- `int` — положительные целые числа;
- `float` — положительные вещественные числа;
- `path` — строка с `/`;
- `uuid` — UUID.

Вычисление абсолютных ссылок

```
url_for('имя-функции-обработчика'[, параметры-обработчика[, _exte
```

Перенаправления

```
return redirect(url_for(...))
```

Аварийные реакции

Генерация:

```
abort(код возврата)
```

Оформление обработчика:

```
@app.errorhandler(код)
def the_handler(error):
    return render_template('код.html'), 404
```

Где «код» — это 404, 500 и так далее.

Jinja2 в Flask

Размещение

По умолчанию исходники размещаются в следующих директориях:

- static — картинки, шрифты и так далее;
- templates — шаблоны.

Рендеринг

Вызов рендеринга:

```
from flask render_template


return render_template(шаблон[, имя=значение[, имя=значение, ...]]
```

Выражения в шаблонах:

- `{% extends 'имя-базового-блока' %}` — наследовать;
- `{% include 'имя-подключаемого-шаблона' %}` — вставить шаблон;
- `{% block имя-блока %}{% endblock имя-блока %}` — объявить и заменить блок с разметкой;
- `{% if условие %}...{% else %}...{% endif %}` — ветвление;
- `{% for переменная in набор %}...{% endfor %}` — перебор;
- `{{ вычисление }}` — вставить значение;
- `{# ... #}` — комментарий.

Вычисление ссылок:

```
url_for('имя-функции-обработчика'[, параметры-обработчика[, _exte
```

Вычисление мест со статикой:

```
url_for('static', filename='относительный-путь-к-файлу')
```

Флеш-сообщения

В Python:

```
from flask import flash


def the_fun()
    if условие:
        flash('сообщение', 'the-tag')
```

В шаблоне:

```
{% for message in get_flashed_messages() %}
  ...{{ message }}...
{% endfor %}
{% for category, message in get_flashed_messages(with_categorie
s=true) %}
  ...{% if category=='the-tag' }}...{{ message }}...{% endif %}
{% endfor %}
{% for category, message in get_flashed_messages(category_filte
r=['the-tag']) %}
  ...{{ message }}...
{% endfor %}
```

Формы WTForms

Установка

```
$ pip install Flask-WTF==1.2.1
```

Секретный ключ для защиты форм:

```
class Config(object):
    SECRET_KEY = os.getenv('SECRET_KEY')
```

Класс для формы

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, TextAreaField, URLF
from wtforms.validators import DataRequired, Length, Optional
```

```
class TheForm(FlaskForm):
    поле = ТипПоля(
        'Метка',
        validators=[Валидатор, ...]
    )
```

- Тип поля: `StringField`, `SubmitField`, `TextAreaField`, `URLField` и так далее.
- Валидатор:
  - `DataRequired(message='...')`,
  - `Length(от, до, message='...')`,
  - `Optional()`.

Рендеринг формы

Не отличается от других рендерингов:

```
return render_template('шаблон-с-формой.html', form=TheForm(...))
```

Шаблон формы

```
<form ...>
  {{ form.csrf_token }}
  {{ form.поле.label }} {{ form.поле }}
  {{ form.поле(class="CSS-классы", placeholder=form.поле.label.tex
  {% if form.поле.errors %}
    {% for error in form.поле.errors %}
      {{ error }}
    {% endfor %}
  {% endif %}
  {{ form.submit(class="...") }}
</form>
```

Обработка формы

```
@app.route('/path', methods=['GET', 'POST'])
def the_view():
    form = TheForm()
    if form.validate_on_submit():
        db.session.add(TheModel(поле=form.поле.data, ...))
        db.session.commit()
        return redirect(url_for(...))
    return render_template('the_form.html', form=form)
```

Миграции

Установка обёртки над `Alembic` для Flask:

```
$ pip install Flask-Migrate==4.0.7
```

Подключение в проект:

```
from flask_migrate import Migrate
...
db = SQLAlchemy(app)
migrate = Migrate(app, db)
```

Создание репозитория миграций в поддиректории migrations:

```
flask db init
```

Создание одной миграции, исходной или после изменения моделей:

```
flask db migrate -m "комментарий"
```

Применение всех миграций:

```
flask db upgrade
```

Добавление команд

Устанавливать ничего не нужно. Модуль `click` (Command Line Interface Creation Kit) устанавливается вместе с Flask.

В Python:

```
import click


@app.cli.command('имя_комманды')
def the_command():
"""Описание комманды."""
click.echo(f'Сообщение о работе')
```

Применение в консоли:

```
$ flask
Commands:
  ...
  имя_комманды  Описание комманды.
$ flask имя_комманды
Сообщение о работе
```

Роутинг в REST API

```
from flask import jsonify, request


@app.route('/the_path/<тип:имя>/', methods=['GET', ...])  # Зав
ершающий / необязателен
def the_api_view(имя):
    data = request.get_json()
    return jsonify({'ответ': ответ}), 200
```

Обработка ошибок

```
class TheError(Exception):
    status_code = 400
    def __init__(self, message, status_code=None):
        super().__init__()
```

```
        self.message = message
        if status_code is not None:
            self.status_code = status_code
    def to_dict(self):
        return dict(message=self.message)


@app.errorhandler(TheError)
def the_handler(error):
    return jsonify(error.to_dict()), error.status_code
```

Полезные ресурсы

Документация SQLAchemy

Документация Flask-SQLAchemy

Документация WTForms

Практикум