Установка Flask

```
$ pip install flask==2.0.2
```

Минимальный проект на Flask

Минимальная структура

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

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def the_view(): return 'ok'
if __name__ == '__main__':
    app.run()
```

Настройка режима работы

По умолчанию — **production**, поменять его на **development**:

```
# Linux и MacOS
$ export FLASK_ENV=development
# Windows
$ set FLASK_ENV=development
```

Команда запуска

Рекомендованный запуск:

```
$ flask run
```

Прямой вызов:

```
$ python theapp.py
```

Если модуль приложения не **app.py**

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

Адрес сайта

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

Работа с .env

Содержимое файла **.env**, если модуль приложения называется **the_app.py**:

```
FLASK_APP=the_app
FLASK_ENV=development
```

Типичный проект на Flask

```
├── dir_app
    ├── 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
```

Содержимое файла **.env**:

```
FLASK_APP=the_app
FLASK_ENV=development
```

Содержимое файла **settings.py**:

```
import os
class Config(object):
    SECRET_KEY = os .getenv('SECRET_KEY')
    SQLALCHEMY_DATABASE_URI = os.getenv('DATABASE_URI')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

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-SQLAlchemy

Документация SQLAlchemy

Flask-SQL-Alchemy

Установка

```
$ pip install flask-sqlalchemy==2.5.1
```

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 во Flask-SQLAlchemy

Документация UG

API

```
db = SQLAlchemy(app)
class TheModel(db.Model):
    the_field = db.Column(
      тип,                # Из db.Integer, db.String(1234), ...
      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:

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

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

```
>>> from the_app import db
>>> db.create_all()
```

Удаление таблиц, файл БД при этом останется:

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

ORM-команды для моделей

Документация

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 появятся после .commit().

Объём таблицы или выборки:

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

Отмена изменений, произошедших после последнего .commit()

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

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:

```
o = TheModel.query.first() # None для пустой выборки
o = TheModel.query.first_or_404()
```

Извлечение объекта по ключу:

```
o =  TheModel.query.get(ключ) # None при промахе
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:

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

Где «тип» — это:
- string — строка без /, по молчанию;
- int — положительные целые числа;
- float — положительные вещественные числа;
- path — строка с /;
- uuid — UUID.

Абсолютные ссылки:

```
url_for('имя-функци-
и-обработчика'[, пара-
метры-обработчика[,
_external=True]])
```

Оформление обработчика:

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

Перенаправления:

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

Генерация:

```
abort(код возврата)
```

Jinja2 в Flask

По умолчанию исходники размещаются в следующих директориях:
- static — картинки, шрифты и так далее;
- templates — шаблоны.

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

Вызов рендеринга:

```
from flask import render_template
render_template(шаблон[, имя=значение[, имя=значение, ...]])
```

Оформление ссылок:

```
url_for('имя-функции-
обработчика'[, параметры-
обработчика[, _external=True]])
```

Оформление статики:

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

Формы WTForms

Документация WTForms

Установка:

```
$ pip install Flask-WTF==1.0.0
```

Секретный ключ для защиты:

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

Класс для формы:

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, TextAreaField, URLField
from wtforms.validators import DataRequired, Length, Optional
class TheForm(FlaskForm):
   поле = ТипПоля(
      'Метка',
      validators=[Валидатор, ...]
   )
```

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

Рендеринг формы:

```
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.text)
}}
  {% 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:

```
$ pip install
Flask-Migrate==3.1.0
```

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 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_команды')
def the_command():
    """Описание команды."""
    click.echo(f'Сообщение')
```

Применение в консоли:

```
$ flask
Commands:
  ...
  имя_команды  Описание команды.
$ flask имя_команды
Сообщение о работе
```

Роутинг в REST API

```
from flask import jsonify, request
# Завершающий / необязателен
@app.route('/the_path/<тип:имя>/', methods=['GET', ...])
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
```