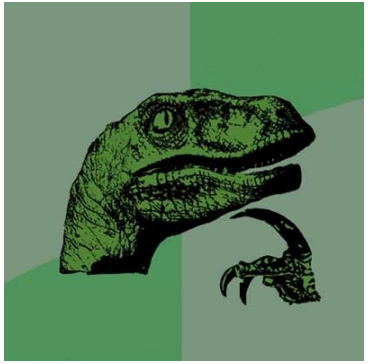
Розробка back-end на Python (Django/Flask)

Якімець Андрій

Нашо?

- Python - це стильно, модно, молодіжно;
- Велика спільнота;
- Розробка займає мало часу і ресурсів.



Як то буде?

1. Переваги і недоліки Django/Flask
2. Що краще обрати?
3. Спільні речі у Django та Flask
4. Приклади Rest API для Django та Flask
5. Як запустити Django/Flask на Linux сервері?
6. Приклад власного реального проекту на Django

“Django makes it easier to build better Web apps more quickly and with less code.”

Плюси;

- Багато речей вже є “з коробки”;
- Суперова документація;
- Легко використовувати;
- Велика спільнота.

Мінуси:

- Заскладне для малих проектів;
- Не вміє справлятися із багатьма запитами одночасно;
- Зав’язане на власному ORM.

Компанії, які використовують:

“Once you have Flask up and running, you’ll find a variety of extensions available in the community to integrate your project for production.”

Плюси:

- Мінімалістичний, без нічого зайвого;
- Масштабований;
- Простий в розробці;

Мінуси:

- Не стандартизований;
- Має не настільки багато додатків, які покривають багато випадків;
- Потрібно робити багато ручної роботи.

Компанії, які використовують:

Що обрати?

- **Flask** – підходить, якщо потріно мати веб-аплікацію без нічого зайвого або мікросервісну архітектуру, яку можна легко масштабувати.
- **Django** – підходить, якщо достаньо інструментів, які воно пропонує для розробки класичного веб-додатку. І це можна зробити за реально стислі терміни.

Спільне. Локальний сервер

- **Сервер розробки** – піднімають тимчасовий локальний сервер, який обробляє запити, статичні файли і автоматично перезавантажується в разі змін у коді.

  Команда запуску для Flask і Django:

  python manage.py runserver

Спільне. Команди

- Підтримують набір команд-утиліт, напр. **Django** – *makemigrations* для створення міграційних файлів, *startproject* для створення нового проекту і т. д.
- **Flask** – напр., *flask shell*, щоб запустити командний рядок.
- Обоє підтримують написання власних команд-утиліт.

Спільне. Власні команди

```python
from django.core.management.base import BaseCommand, CommandError
from polls.models import Question as Poll

class Command(BaseCommand):
    help = 'Closes the specified poll for voting'

    def add_arguments(self, parser):
        parser.add_argument('poll_ids', nargs='+', type=int)

    def handle(self, *args, **options):
        for poll_id in options['poll_ids']:
```

python manage.py closepoll <poll_ids>

```python
import click
from flask import Flask

app = Flask(__name__)

@app.cli.command("create-user")
@click.argument("name")
def create_user(name):
    ...
```

```
$ flask create-user admin
```

Спільне. Маршрутизація

```python
from django.urls import path

from .views import HomeView

urlpatterns = [
    path('home/', HomeView.as_view(), name='home'),
]
```

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/home/')
def home_view():
    """
    View для домашней страницы.
    """
    return render_template("home.html")
```

Спільне. Об’єкт Request

- Об’єкт Request містить дані про запит.
- **Django** – приймає його як параметр для view.
- **Flask** – матиме доступ до об’єкту лише після його імпорту.

Спільне. Аутентифікація та авторизація

- **Django** – має вже додаток аутентифікації із таблицею користувачів. В більшості випадків цього функціоналу достатньо, але є можливість все повністю переписати.
- **Flask** – містить лише аутентифікацію через куки. Для більш широких можливостей, потрібно встановлювати розширення.

Rest API

Flask. Rest API

A Minimal API

A minimal Flask-RESTful API looks like this:

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')

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

Django. Rest API

```python
from django.contrib.auth.models import User, Group
from rest_framework import serializers


class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = ['url', 'username', 'email', 'groups']


class GroupSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Group
        fields = ['url', 'name']
```

Django. Rest API

```python
from django.contrib.auth.models import User, Group
from rest_framework import viewsets
from tutorial.quickstart.serializers import UserSerializer, GroupSerializer


class UserViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows users to be viewed or edited.
    """
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserSerializer


class GroupViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows groups to be viewed or edited.
    """
    queryset = Group.objects.all()
    serializer_class = GroupSerializer
```

Django. Rest API

```python
from django.urls import include, path
from rest_framework import routers
from tutorial.quickstart import views

router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)
router.register(r'groups', views.GroupViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

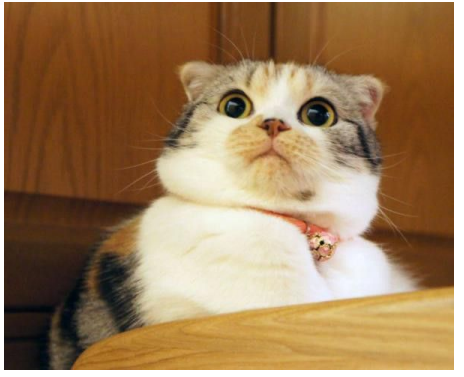
Запускаємось на Linux сервері

Для цього необхідна утиліта **uWSGI**. Вона дає нам змогу запускати веб-сервер для запуску програм на Python через протокол **WSGI** (Web Server Gateway Interface). Тобто, це є інструмент, який стає прошарком, для запуску Python програми на сервері.

Запускаємось на Linux сервері

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type','text/html')])
    return [b"Hello World"]
```

```
uwsgi --http :9090 --wsgi-file foobar.py
```



Django. Запускаємось на Linux сервері

```
[uwsgi]
socket = 127.0.0.1:3031
chdir = /home/foobar/myproject/
wsgi-file = myproject/wsgi.py
processes = 4
threads = 2
stats = 127.0.0.1:9191
```

```
uwsgi yourfile.ini
```

Flask. Запускаємось на Linux сервері

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return "<span style='color:red'>I am app 1</span>"
```

uwsgi --socket 127.0.0.1:3031 --wsgi-file myflaskapp.py --callable app --processes 4 --threads 2 --stats 127.0.0.1:9191

Приклад власного реального проекту на Django

Питання?

Дякую за увагу!

andriyakimets@gmail.com