



Python: RESTful API application

ITFROVN - Python 2

Python Flask

03



[demo] Flask environment prepare

1. Install virtual environment và libpq-dev:

```
apt install python3-venv libpq-dev
```

2. Download Flask boilerplate:

```
git clone -b simple  
https://github.com/abpabab/flask-boilerplate
```

3. Tạo mới virtual environment:

```
cd flask-boilerplate
```

```
python3 -m venv .venv
```

4. Kích hoạt virtual environment:

```
source .venv/bin/activate
```

5. Cài đặt project dependency modules:

```
pip3 install -r requirements.txt
```

```
ubuntu@ubuntu:~/Projects/Python-2$ tree flask-boilerplate -L 3
flask-boilerplate
├── app
│   ├── home
│   │   ├── __init__.py
│   │   └── routes.py
│   ├── __init__.py
│   └── welcome
│       ├── __init__.py
│       └── routes.py
└── config.py
├── README.md
└── REAME.md
├── requirements.txt
└── run.sh
└── simple_app.py

3 directories, 11 files
```

[demo] Flask - run

- Chạy chương trình ở chế độ “development”:

make dev



```
ubuntu@ubuntu:~/Projects/New-Python-2/flask-boilerplate$ make dev
. .venv/bin/activate; \
export FLASK_ENV=development; \
python3 simple_app.py
* Serving Flask app "app" (lazy loading)
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 264-774-482
```

- Test chương trình:

curl http://127.0.0.1:5000/welcome

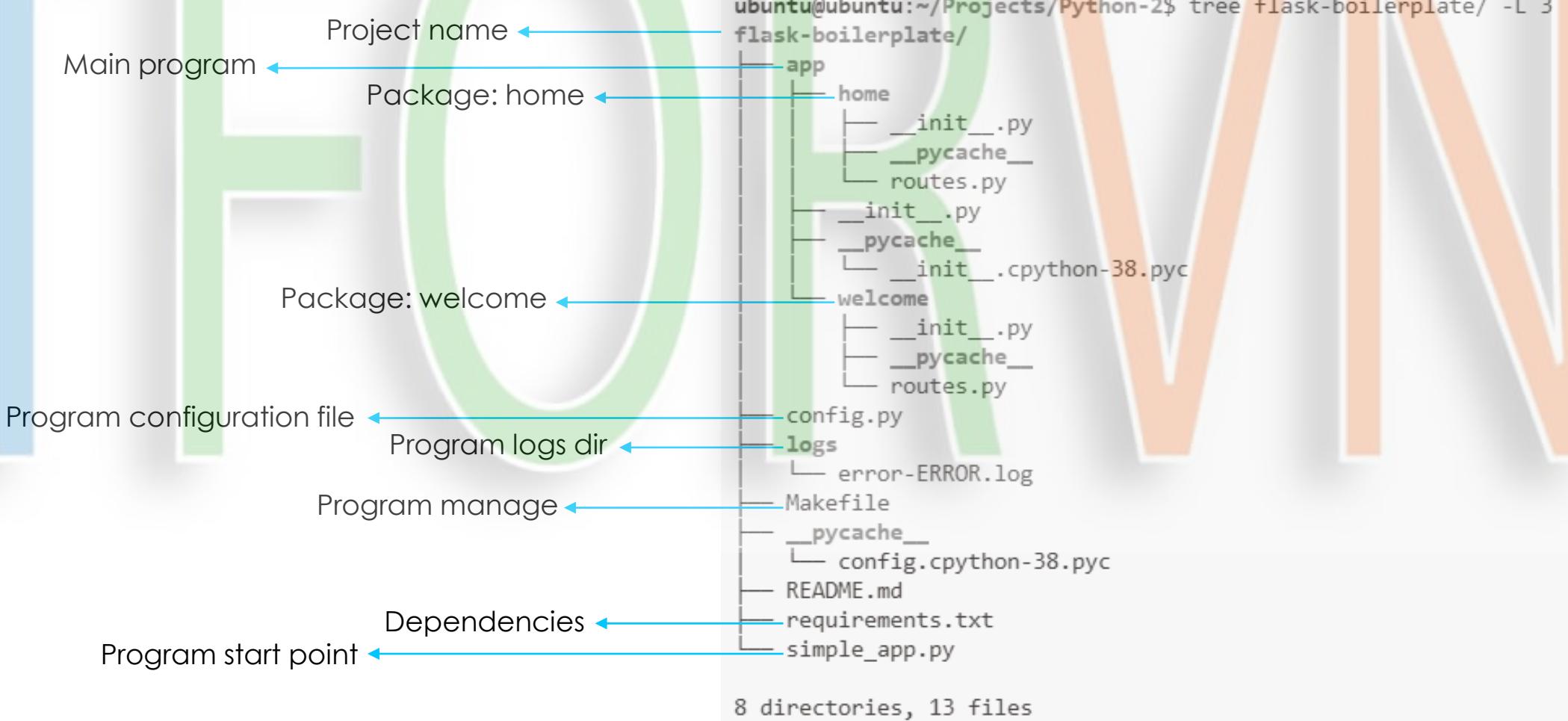


```
ubuntu@ubuntu:~$ curl http://127.0.0.1:5000/welcome
{
  "message": "Welcome to the API",
  "status": true
}
ubuntu@ubuntu:~$
```

Dùng Web browser truy cập:

http://127.0.0.1:5000/welcome

[demo] Simple Flask structure



[demo] via Postman

http://192.168.138.128:5000/welcome

GET http://192.168.138.128:5000/welcome

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Headers (6 hidden)

KEY

Key

VALUE

Value

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize

JSON



```
1   "message": "Welcome to the API",
2   "status": true
3
4
```

Python module vs package

- 1 file python, với các variable và function, có thể xem như là 1 module và có thể được import sử dụng trong các file (module) python khác.

```
ubuntu@ubuntu:~/Projects/Python-2/test$ cat a.py
bien_a = 10
print(bien_a)
```

```
def ham_a():
    print("day la ham a")
```

```
def ham_b():
    print("day la ham b")
```

```
ubuntu@ubuntu:~/Projects/Python-2/test$ cat b.py
import a
```

```
a.ham_a()
a.ham_b()
```

- 1 thư mục với nhiều module, và có tồn tại file `__init__.py` thì được xem như là 1 package và được import sử dụng bình thường.

```
ubuntu@ubuntu:~/Projects/Python-2/test$ tree mp/
mp/
├── a.py
├── b.py
└── __init__.py
    └── __pycache__
        ├── a.cpython-38.pyc
        ├── b.cpython-38.pyc
        └── __init__.cpython-38.pyc
```

1 directory, 6 files

```
ubuntu@ubuntu:~/Projects/Python-2/test$ cat main.py
import mp

if __name__ == '__main__':
    print(f"In main.py, my __name__ is: {__name__}")

    mp.a.ham_a_1()

    mp.a.get_name()
```

Module path & `__name__`

- Khi import module, thì path để Interpreter dùng để tìm module: sys.path.
- Chú ý: khi sử dụng import package, thì phải sử dụng dạng “namespace”, tính từ vị trí thư mục mà file được chạy để import.

- Nếu một module (file) được thực thi trực tiếp: `__name__ == "__main__"`
- Nếu một module (file) được thực thi gián tiếp: `__name__` sẽ có giá trị là tên của module đó, tính theo “namespace” được gọi và bỏ đi extension .py

```
def main():
    # the main code here

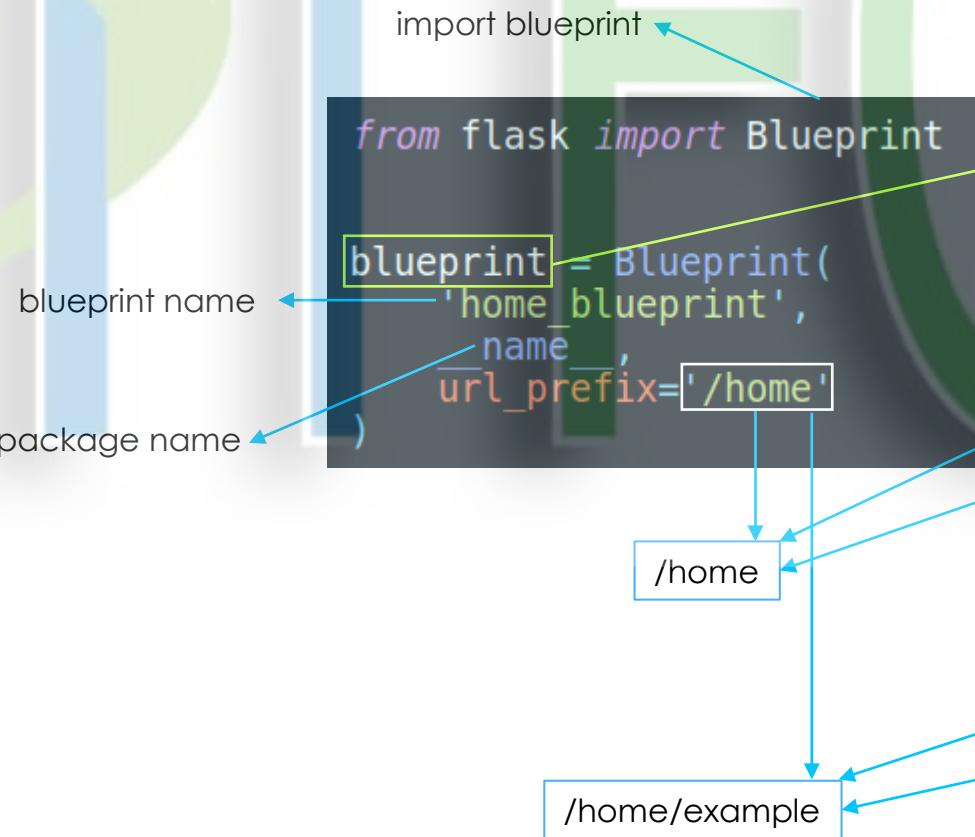
if __name__ == '__main__':
    main()
```

Flask – important things

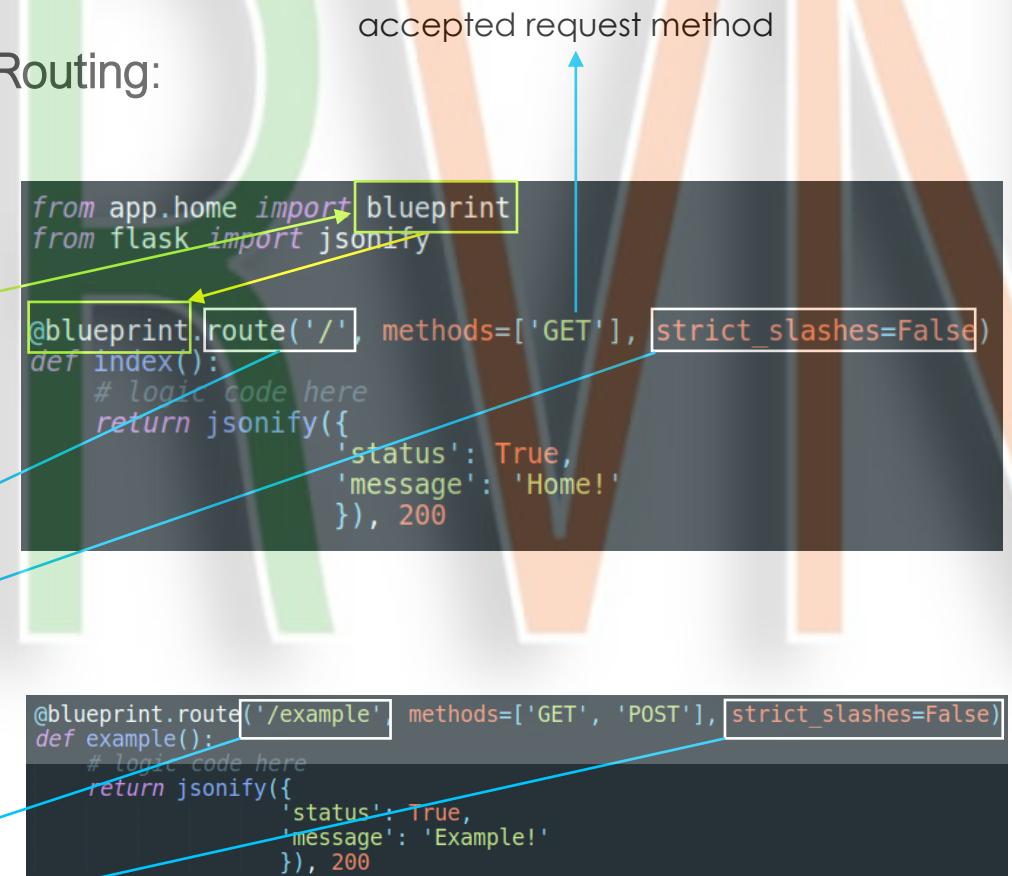
- `app/__init__.py`: file chứa code chính của program, bao gồm các hàm khởi tạo Flask application và các hàm phụ trợ.
- Blueprint: Flask sử dụng blueprint (bản thiết kế) để quản lý và tổ chức các API. Một blueprint bao gồm một API endpoint hoàn chỉnh: `model`, `routing`, `template`, `assets`. Khi sử dụng blueprint, cần các bước sau:
 - Khai báo blueprint: file `app/<package>/__init__.py`
 - Import blueprint: file `app/<package>/routes.py`
 - Register blueprint: file `app/__init__.py`, hàm `register_blueprints()`
- `app/<package>`: mỗi API endpoint sẽ được tổ chức trong một thư mục, bao gồm tất cả các thành phần model, routing, template. Mỗi thư mục package phải có:
 - `__init__.py`: file khai báo blueprint đồng thời cho phép sử dụng thư mục như là 1 package.
 - `routes.py`: file khai báo routing, chính là các endpoint mà API cung cấp và các xử lý logic của endpoint.
 - `models`: sử dụng cho việc khai báo các database model (nếu có).
- `config.py`: là file chứa các cấu hình (configuration) cho toàn bộ program.

Flask – blueprint vs routing

- Blueprint:



- Routing:



API design – naming endpoint

1. Dùng danh từ (dạng số nhiều nếu có thể):
 - <http://myapi.com/users>
 - [http://myapi.com/getUser](http://myapi.com/get-user) <http://myapi.com/getUsers>
2. Sử dụng chữ thường, không dùng chữ in hoa:
 - <http://myapi.com/tasks>
 - <http://myapi.com/Tasks> <http://myapi.com/ĐoTasks>
3. Không sử dụng kí tự đặc biệt
4. Không sử dụng file extension
 - <http://myapi.com/users>
 - <http://myapi.com/users.html> <http://myapi.com/users.api>

API design – CRUD

- Ví dụ API CRUD cho User:
 - Endpoint: <http://myapi.com/users>
 - Tạo mới user: POST <http://myapi.com/users>
 - Lấy danh sách toàn bộ users: GET <http://myapi.com/users>
 - Chi tiết user có ID 10: GET <http://myapi.com/users/10>
 - Update user có ID 10: PUT <http://myapi.com/users/10>
 - Xóa user có ID 10: DELETE <http://myapi.com/users/10>

- Create – POST
- Read – GET
- Update – PUT
- Delete – DELETE

Flask – step to build an application

1. Prepare Flask: boilerplate, environment, configuration.
2. Prepare database: MySQL, MongoDB, Elasticsearch, ...
3. Functional design:
 - a. Ứng dụng có bao nhiêu tính năng, chi tiết mỗi tính năng.
 - b. Input/output ở mỗi tính năng.
 - c. Mỗi quan hệ giữa các tính năng.
4. Endpoint design: dựa trên functional design.
5. Model design: thiết kế lưu trữ database (nếu có), bao gồm:
 - a. Xác định loại CSDL cần sử dụng: SQL or NoSQL
 - b. Schema design: chi tiết từng field name và kiểu dữ liệu của từng field, mối quan hệ giữa các table/schema với nhau.
6. Coding

