



# 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

6. 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
```

7. Test chương trình:

curl http://127.0.0.1:5000/welcome

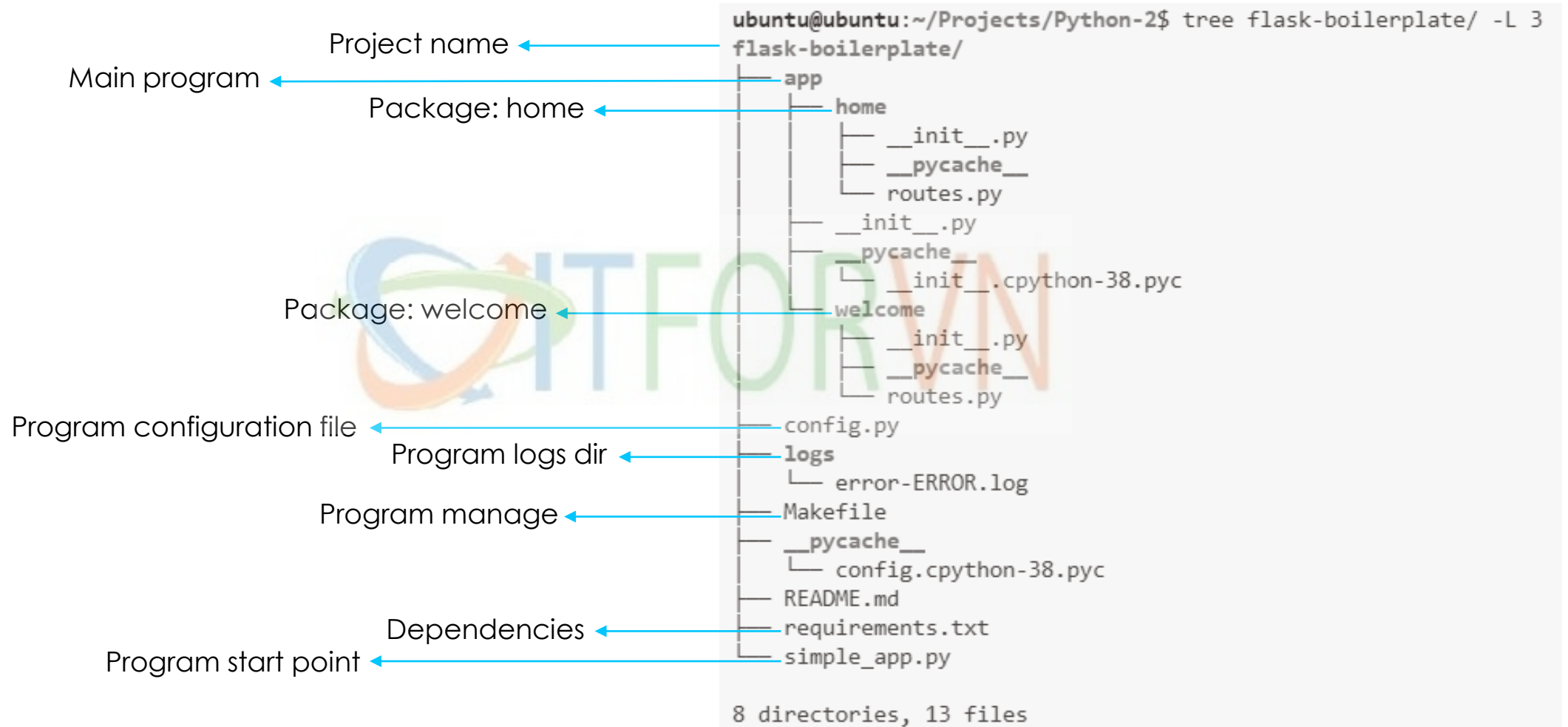


Dùng Web browser truy cập:

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:~$
```

# [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	VALUE
Key	Value



Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Welcome to the API",
3   "status": true
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.
- 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$ 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()
```

```
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()
```



# API design – naming endpoint

1. Dùng danh từ (dạng số nhiều nếu có thể):

- <http://myapi.com/users>
- ~~<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/DoTasks>

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

# Database prepare - MySQL

- MySQL 8.0 on Ubuntu 20.04:
  - Install MySQL server: apt install mysql-server
  - Create Database and User:

```
(root) # mysql
```

```
mysql> CREATE DATABASE tasks_db;
```

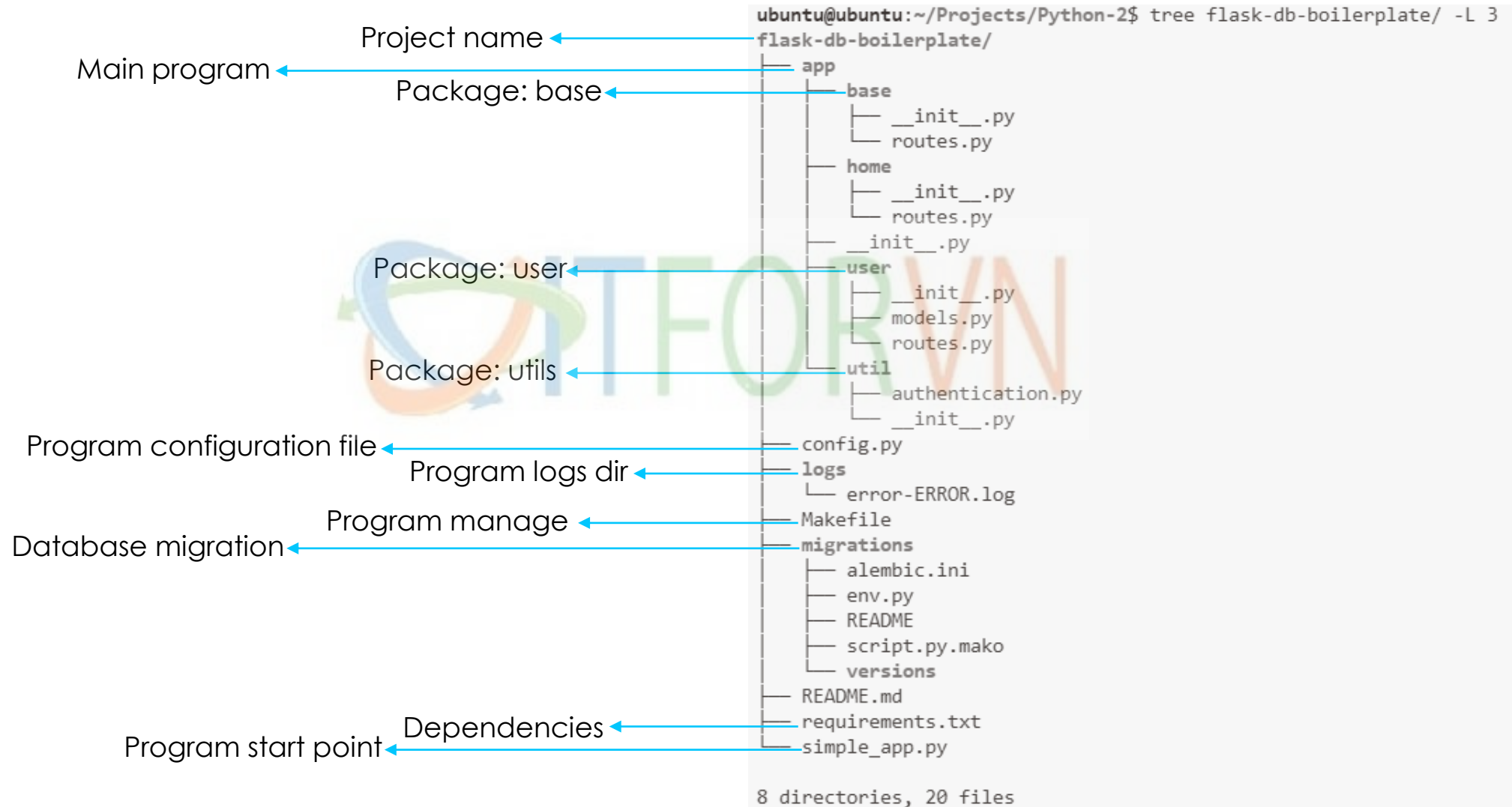
```
mysql> CREATE USER 'tasks_user'@'localhost' \
-> IDENTIFIED WITH mysql_native_password \
-> BY 'MyStrongPassword';
```

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, INDEX, DROP, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES \
-> ON tasks_db.* \
-> TO 'tasks_user'@'localhost';
```



Database nên phân quyền **đúng** và **đủ** cho user, tránh dùng “GRANT ALL”

# [Database] Flask structure



# Jinja template

- Request methods – chỉ định các tác vụ mong muốn mà client muốn thực hiện trên một resource:
  - GET: thực hiện lấy resource mong muốn, và server chỉ trả về resource mà client cần.
  - HEAD: tương tự như GET nhưng không trả về resource data, chỉ có response header được trả về.
  - POST: submit data từ client đến server, thường dùng để tạo/thêm mới dữ liệu ở phía server.
  - PUT: thực hiện việc thay đổi hoàn toàn trên một đối tượng ở 1 resource cụ thể.
  - PATCH: thực hiện việc thay đổi 1 phần trên một đối tượng ở 1 resource cụ thể.
  - **DELETE**: xóa 1 resource cụ thể.
  - CONNECT
  - OPTIONS
  - TRACE

<b>C</b>	→	Create	→	<b>POST</b>
<b>R</b>	→	Read	→	<b>GET</b>
<b>U</b>	→	Update	→	<b>PUT</b>
<b>D</b>	→	Delete	→	<b>DELETE</b>



# 1. HTTP – Response status code

- HTTP response status code – giúp xác định một HTTP request được thực thi thành công hay thất bại, nếu thất bại thì lý do là gì.
- Status code được biểu thị bằng các số thập phân và được chia làm 5 nhóm:
  - Informational responses: 100-199
  - Successful responses: 200-299
  - Redirect: 300-399
  - Client errors: 400-499
  - Server errors: 500-599
- List status code: <https://datatracker.ietf.org/doc/html/rfc2616#section-10>



# 1. HTTP – Security

- HTTP Access control CORS (Cross-Origin Resource Sharing).

```
GET /resources/public-data/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
Origin: https://foo.example
```


```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml

[...XML Data...]
```

- SSL: Secure Socket Layer – Netscape thêm vào HTTP (năm 1994) “additional encrypted transmission layer”, gọi là SSL (phiên bản 1.0 internal company), SSL 2.0 và SSL 3.0.
- TLS: Transport Layer Security.
- CSP: Content-Security-Policy – cho phép web administrator control client được/không được load một trang tài liệu nào đó.

```
Content-Security-Policy: frame-ancestors none;
Content-Security-Policy: frame-ancestors <source> <source>;
```



An aerial photograph of ocean waves, showing a mix of golden-brown water and white foam. A semi-transparent logo and text are overlaid in the center. The logo consists of a stylized 'G' in green and blue, followed by the text 'ITFORVN' in a multi-colored font. Below this, a dark grey rectangular box contains white text defining the term 'Protocol'.

A **Protocol** is a system of rules that define how data is exchanged within or between computers. Communications between devices require that the devices agree on the format of the data that is being exchanged. The set of rules that defines a format is called a protocol.



## 2. HTTP - some

- URL vs URI.

**http(s)://www.vietnam.com.vn:80/path/to/api.html?key1=value1&key2=value2#ChoNaoDo**

The diagram shows a URL with its components labeled in boxes below it: **scheme** (http(s)), **domain name** (www.vietnam.com.vn), **port** (:80), **path** (/path/to/api.html), **query** (?key1=value1&key2=value2), and **anchor** (#ChoNaoDo).

- MIME type.
- www vs non-www

**http(s)://www.vietnam.com.vn:80/path/to/api.html?key1=value1&key2=value2#ChoNaoDo**

The diagram shows a URL with its components labeled in boxes below it: **scheme** (http(s)), **domain name** (www.vietnam.com.vn), **port** (:80), **path** (/path/to/api.html), **query** (?key1=value1&key2=value2), and **anchor** (#ChoNaoDo).

# 1. HTTP - Header

- HTTP client-protocol:
  1. Client khởi tạo TCP connection.
  2. Client gửi request, chờ đợi server gửi trả lại kết quả.
  3. Server process client request, phản hồi lại kết quả đã xử lý cho client: status code và dữ liệu đã xử lý.

