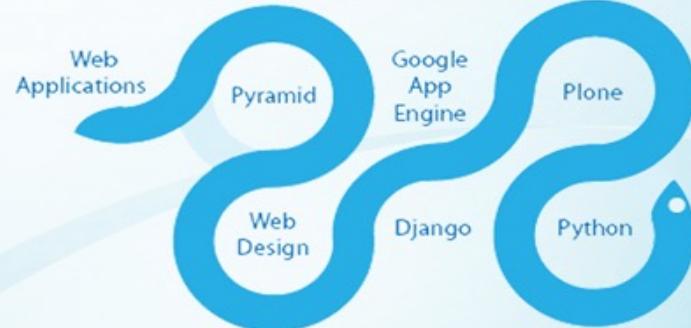


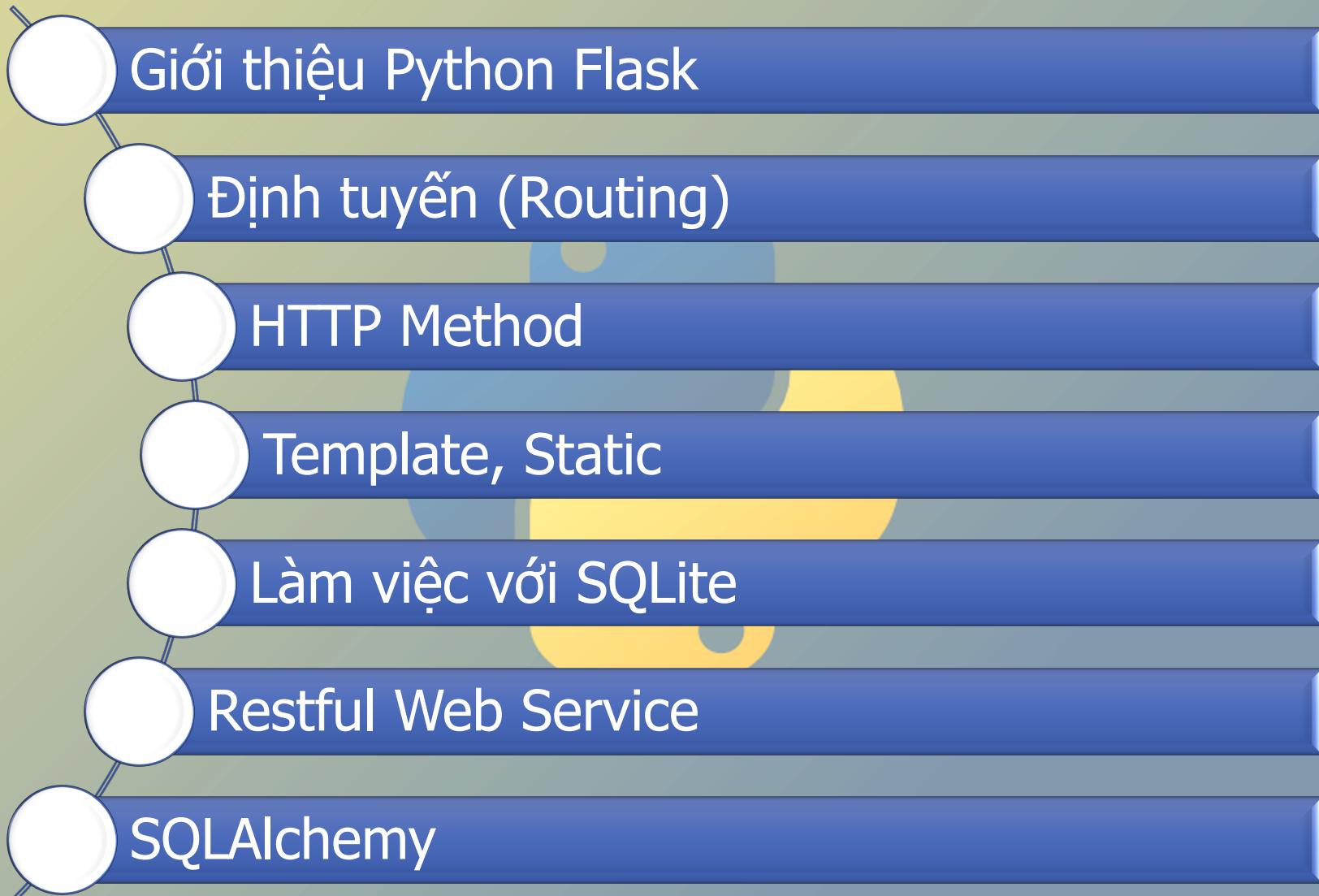
# NGÔN NGỮ PYTHON

## PYTHON FLASK

ThS. Dương Hữu Thành  
[dhthanh@hcmute.edu.vn](mailto:dhthanh@hcmute.edu.vn),  
[dhthannhQA@gmail.com](mailto:dhthannhQA@gmail.com)



# Nội dung chính





# Giới thiệu Python Flask

- Flask là framework bằng python phát triển ứng dụng web được phát triển bởi Armin Ronacher.
- Flask sử dụng bộ công cụ Werkzeug WSGI và bộ template Jinja2.
  - WSGI (Web Server Gateway Interface) là chuẩn phát triển ứng dụng web trong python.
  - Werkzeug thực thi các đối tượng request, response và các chức năng hữu ích khác.
  - Jinja2 là một template engine phổ biến, kết hợp template với một data source nào đó để kết xuất hiển thị trên web động.

# Giới thiệu Flask

- Cài đặt Flask
  - Cài đặt môi trường ảo

```
pip install virtualenv
```

- Tạo môi trường ảo

```
virtualenv venv
```

- Cài đặt Flask

```
pip install flask
```

# Viết ứng dụng Flask đầu tiên

- Cấu trúc cơ bản một project Flask

```
/FlaskApp
    /templates
        -/index.html
        -/hello.html
    /static
        -/images
        -/sty.css
        -/script.js
    -/index.py
```



# Viết ứng dụng Flask đầu tiên

- Tạo tập tin hello.py với nội dung sau:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello_world():
    return "Hello World!!!!"

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

```
Run:  hello ×
/Users/duonghuuthanh/PycharmProjects/FlaskApp/venv/bin/python /Users/duonghuuthanh/P
  * Serving Flask app "hello" (lazy loading)
  * Environment: production
    WARNING: This is a development server. Do not use it in a production deployment.
    Use a production WSGI server instead.
  * Debug mode: off
  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



# Viết ứng dụng Flask đầu tiên

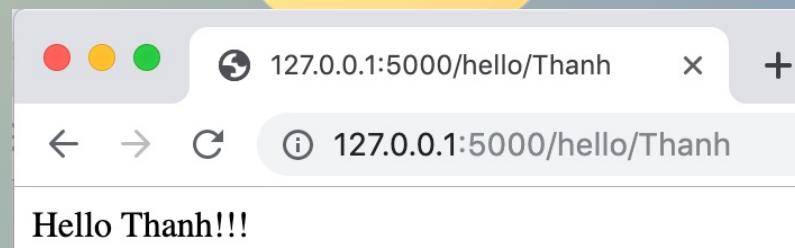
- @app.route(rule, option): chỉ định URL kết hợp với hàm.
- app.run(host, port, debug, options):
  - host: mặc định 127.0.0.1
  - port: mặc định 5000
  - debug: mặc định False
- Truy cập trang web tại địa chỉ
  - <http://127.0.0.1:5000/>

## ▪ URL

```
@app.route("/hello")  
def hello_world():  
    return "Hello World!!!"
```

## ▪ Thêm biến vào URL với cú pháp &lt;tên-biến&gt;

```
@app.route("/hello/<name>")  
def hello_world_2(name):  
    return "Hello %s!!!!" % name
```



- Thêm biến trên URL với kiểu dữ liệu chỉ định

```
@app.route("/details/<int:product_id>/<float:price>")  
def detail(product_id, price):  
    return "Sản phẩm %d: %.1f triệu VNĐ" % (product_id, price)
```





# Routing

- Hàm **use\_for** sử dụng tạo URL động

```
from flask import Flask, redirect
app = Flask(__name__)

@app.route("/hello/<name>")
def hello_world_2(name):
    if name == "client":
        return redirect(url_for("detail",
                               product_id=10,
                               price=3.8))
    else:
        return "Hello %s!!!" % name
```

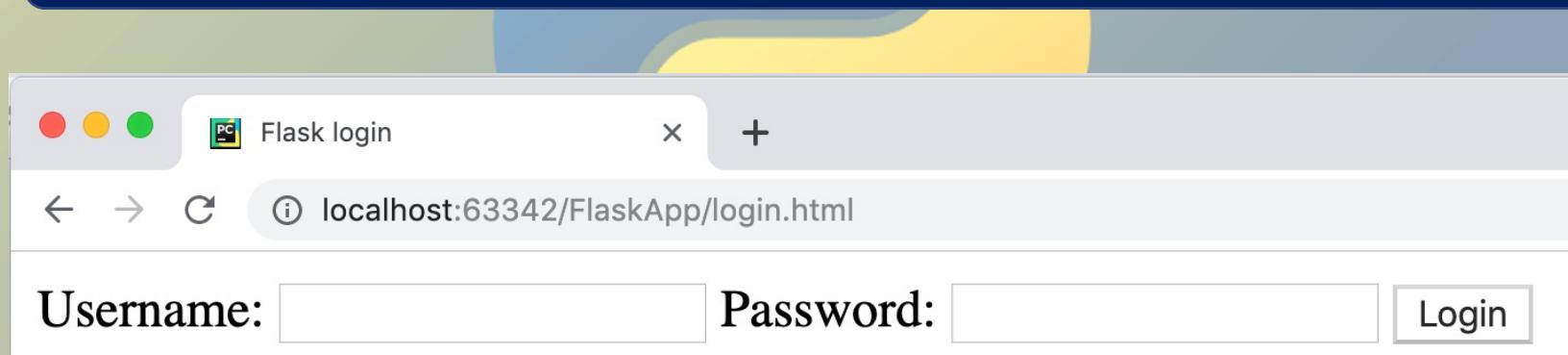
# HTTP Method

- HTTP là giao thức nền tảng giao tiếp dữ liệu trong môi trường WWW. Một số phương thức HTTP:
  - GET: gửi dữ liệu không mã hoá lên server.
  - POST: gửi dữ liệu HTML form lên server, thường dùng tạo mới tài nguyên trên server.
  - PUT: thường dùng cập nhật tài nguyên.
  - DELETE: thường dùng xoá tài nguyên.
- Mặc định Flask sử dụng phương thức GET.

# HTTP Method

- Ví dụ cho trang HTML login có <body> như sau:

```
<form action="/login" method="post">
    Username: <input type="text" name="username" />
    Password: <input type="password" name="password" />
    <input type="submit" value="Login" />
</form>
```





# HTTP Method

## ▪ Đoạn chương trình Python Flask xử lý login

```
@app.route("/login", methods=["POST"])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        if username == 'admin' and password == '123':
            return redirect(url_for("success", name=username))
        else:
            return redirect(url_for("success", name="Anonymous"))
```

Flask login

localhost:63342/FlaskApp/login.html

Username: admin Password: ... Login

127.0.0.1:5000/FlaskApp/login.html

127.0.0.1:5000/success/admin

Welcome admin!!!

127.0.0.1:5000/FlaskApp/login.html

127.0.0.1:5000/success/Admin

Welcome Anonymous!!!

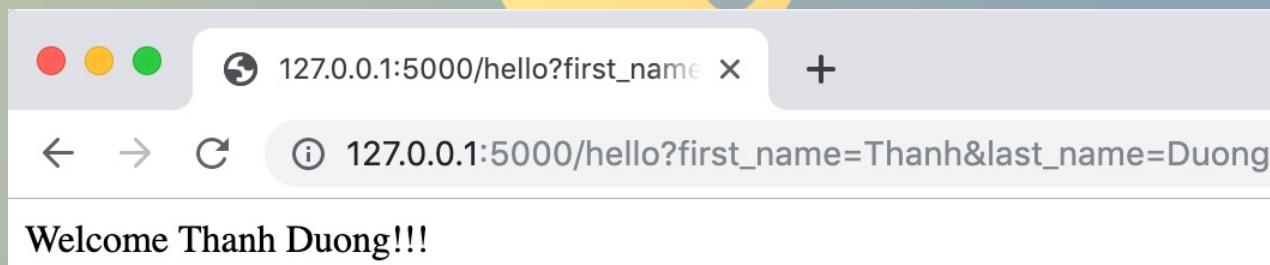
# HTTP Method

- Minh họa lấy dữ liệu từ phương thức GET

```
@app.route("/hello", methods= ["GET"] )  
def hello():  
    first_name = request.args.get("first_name")  
    last_name = request.args.get("last_name")  
  
    return "Welcome %s %s!!!" % (first_name, last_name)
```

- Truy cập URL như sau

```
http://127.0.0.1:5000/hello?first_name=Thanh&last_name=Duong
```



# HTTP Method

- Dữ liệu gửi từ client lên server được lưu trữ trong đối tượng request.
- Các thuộc tính quan trọng của đối tượng request
  - args: dữ liệu từ các tham số truy vấn trên URL.
  - form: dữ liệu từ form data.
  - files: dữ liệu liên quan upload tập tin.
  - method: phương thức thực hiện request.
  - cookies: đối tượng cookies.



# File Uploads

- Để thực hiện upload tập tin lên server thì form phải khai báo **enctype=multipart/form-data**.
- Thuộc tính **files** của đối tượng request chứa dữ liệu (kiểu dictionary) tập tin được gửi lên server.
- Phương thức **save()** cho phép lưu trữ tập tin vào hệ thống tập tin trên server.

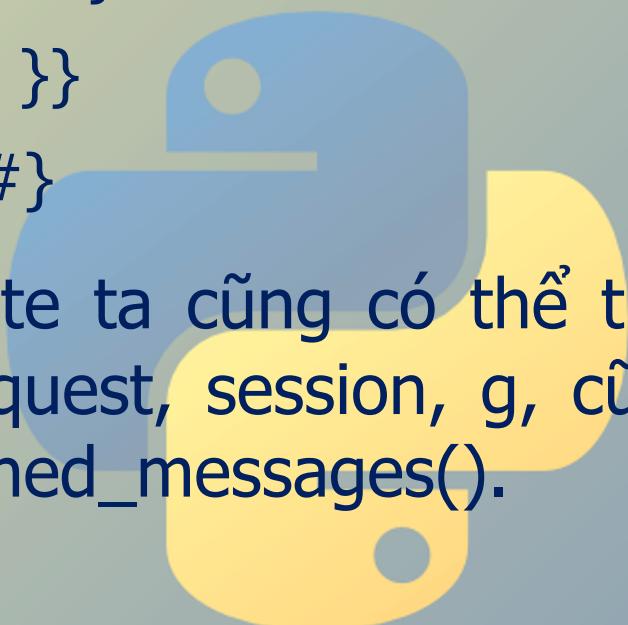
```
from flask import request

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['ten_file']
        f.save(app.root_path + '/static/uploads/uploaded_file.txt')
```

- Flask sử dụng jinja2 làm template engine, template chứa HTML và nhúng vào biến hoặc biểu thức, giá trị của chúng sẽ được thay thế khi kết xuất (render) hiển thị trang web.
- Các tập tin template mặc định đặt trong thư mục **templates** của projet.



- Các cú pháp sử dụng trong template
  - { % câu lệnh % }
  - {{ biểu thức }}
  - { # ghi chú # }
- Trong template ta cũng có thể truy cập vào các đối tượng request, session, g, cũng như phương thức get\_flashed\_messages().



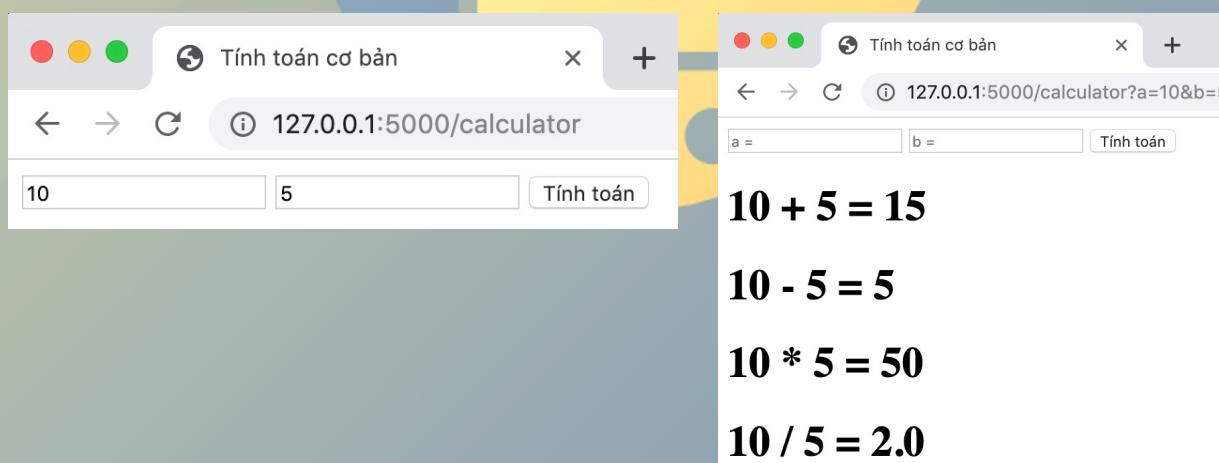
- Ví dụ chương trình tính toán đơn giản với HTML

```
<!DOCTYPE html>
<html lang="en">
<body>
<form action="/calculator" method="get">
    <input type="text" placeholder="a = " name="a" />
    <input type="text" placeholder="b = " name="b" />
    <input type="submit" value="Tính toán" />
</form>
{%
  if a and b %
}
<h1>{{ a }} + {{ b }} = {{ a + b }}</h1>
<h1>{{ a }} - {{ b }} = {{ a - b }}</h1>
<h1>{{ a }} * {{ b }} = {{ a * b }}</h1>
<h1>{{ a }} / {{ b }} = {{ a / b }}</h1>
{%
  endif %
}
</body>
</html>
```

- Chương trình python

```
@app.route("/calculator", methods=[ "GET" ] )
def calculator():
    a = request.args.get("a", None)
    b = request.args.get("b", None)

    return render_template("calculator.html",
                          a=int(a) if a else None,
                          b=int(b) if b else None)
```



- Ví dụ chương trình hiển thị lời chào

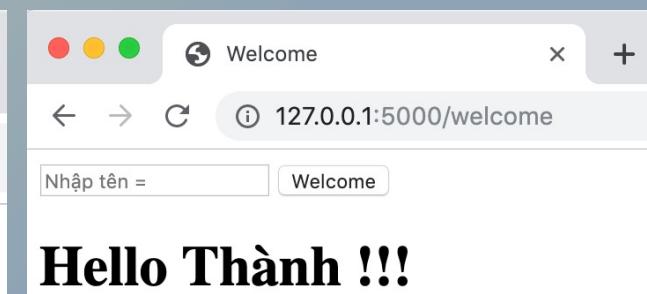
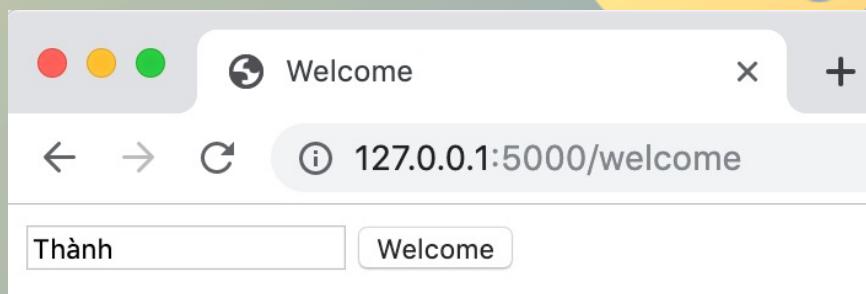
```
<!DOCTYPE html>
<html lang="en">
<body>
    <form action="/welcome" method="post">
        <input type="text" placeholder="Nhập tên = "
               name="username" />
        <input type="submit" value="Welcome" />
    </form>
    {% if name %}
        <h1>Hello {{ name }} !!!</h1>
    {% endif %}
</body>
</html>
```

- Chương trình Python

```
from flask import Flask, request, render_template
app = Flask(__name__)

@app.route("/welcome", methods=["GET", "POST"])
def welcome():
    if request.method == "POST":
        name = request.form["username"]
    else:
        name = ""

    return render_template("hello.html", name=name)
```



- Các biến toàn cục trong Jinja2 template
  - config: đối tượng cấu hình
  - request: đối tượng request hiện hành.
  - session: đối tượng session hiện hành.
  - g: đối tượng kết buộc request cho các biến toàn cục
    - Chú ý: biến session, request và g sẽ không tồn tại nếu template chỉ được render mà không trong một ngữ cảnh request nào.
  - url\_for()
  - get\_flashed\_messages()

# Template Filter

- Tạo bộ lọc riêng cho template

```
@app.template_filter('reverse')
def reverse_filter(s):
    return s[::-1]
```

- Sử dụng trong template

```
{% for x in mylist | reverse %}
{% endfor %}
```

# Template Inheritance

- Cơ chế inheritance cho phép xây dựng template chứa các thành phần chung của trang trang Web và cho phép chỉ định các thành phần được phép ghi đè ở các trang Web con kế thừa lại.
- Điều này cho phép thiết kế template nhanh chóng, nhất quán, tiết kiệm thời gian, dễ dàng bảo trì, sửa đổi.

# Template Inheritance

- Tạo một base template định nghĩa khung chung cho các trang Web. Trong base template, chia thành các thành phần (khối), các khối này có thể được ghi đè ở các trang Web con kế thừa.
- Cú pháp khai báo khối trong template:

```
{% block tên_block %}  
    Nội dung mặc định của block  
{% endblock %}
```

# Template Inheritance

- Các trang Web khác có thể kế thừa lại base template và có thể ghi đè một số các block để chỉ định nội dung cho riêng trang Web đó.
- Cú pháp kế thừa base template, lệnh này đặt ở đầu trang Web

```
{% extends base_template_path %}
```

- Trong một khôi của trang Web con muốn sử dụng nội dung của khôi đó trong template cha có thể sử dụng cú pháp:

```
{ { super() } }
```



# Static files

- Các static file hỗ trợ hiển thị trang web như javascript, css. Các tập tin này đặt mặc định trong thư mục static của project.
- Để lấy url tới static file, ta có thể sử dụng cú pháp

```
url_for("static", filename="style.css")
```

- Tất cả dữ liệu trong cookie được client gửi lên server được lưu trữ trong thuộc tính cookies (kiểu dictionary) của đối tượng request.

```
from flask import request

@app.route('/')
def index():
    username = request.cookies.get('username')
```

- Muốn thiết lập giá trị cho cookie sử dụng phương thức `set_cookie()` của đối tượng `response`.

```
from flask import make_response  
  
@app.route('/')  
def index():  
    resp = make_response(render_template(...))  
    resp.set_cookie('username', 'the username')  
    return resp
```



- Thuộc tính session của đối tượng request lưu trữ thông tin session.
- Để sử dụng session cần thiết lập giá trị secret key

```
app = Flask(__name__)

# some random bytes
# python -c 'import os; print(os.urandom(16))'
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'
```

# Chuyển trang

- Để chuyển người dùng đến một endpoint khác sử dụng hàm `redirect()` và để huỷ bỏ request với một error code nào đó sử dụng hàm `abort()`.

```
from flask import abort, redirect, url_for

@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login')
def login():
    abort(401)
```

- Để chỉ định một số trang error page thay thế các trang mặc định của nó, ta sử dụng decorator `errorhandler()`.

```
from flask import render_template

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

- Giá trị trả về từ hàm view sẽ được chuyển thành đối tượng response.
- Nếu đối tượng response được trả về, nó trực tiếp được trả về từ view.
- Nếu trả về string: nó được chuyển thành đối tượng response, status code 200, mimetype là text/html.
- Nếu trả về dict, jsonify() sẽ được gọi để tạo response.

# Response

- Nếu tuple được trả về, các item trong tuple có thể cung cấp thêm thông tin phụ giống như sau:
  - (response, status)
  - (response, headers)
  - (response, status, headers)



- SQLite được viết bằng ANSI-C, làm việc được trên nhiều hệ điều hành như Linux, MacOS, Windows.
- SQLite hoạt động không cần có server, không cần cấu hình, cài đặt hay quản trị.
- SQLite có đầy đủ các thuộc tính ACID, hỗ trợ ngôn ngữ truy vấn chuẩn SQL.
- Để có thể dễ dàng tương tác cơ sở dữ liệu SQLite ta có thể cài công cụ SQLiteStudio

<https://sqlitestudio.pl/index.rvt>

# Làm việc với SQLite

- Đầu tiên phải import thư viện sqlite3

```
import sqlite3
```

- Tạo kết nối đến cơ sở dữ liệu

```
sqlite3.connect (<sqlite3_path>)
```

- Khi kết nối nếu chưa có cơ sở dữ liệu chỉ định thì nó sẽ được tạo ra.
- Thực thi truy vấn bằng phương thức execute

```
conn.execute (<sql_query>)
```

- Chèn dòng dữ liệu mới vào bảng product

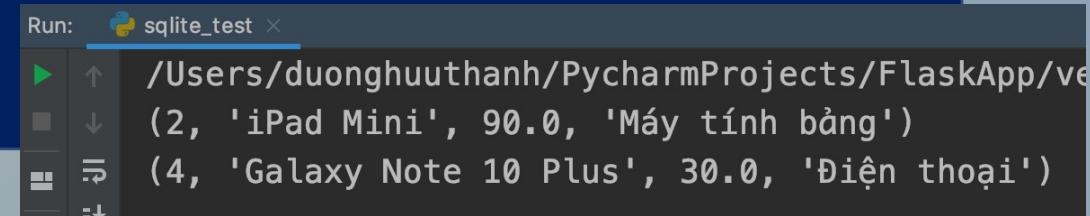
```
conn = sqlite3.connect("data/sale.db")
sql = """
INSERT INTO product(name, price, category_id)
VALUES(?, ?, ?)
"""

if conn.execute(sql, ('iPhone 7 Plus', 17, 1)):
    print("Add successful")
conn.commit()
conn.close()
```

# Làm việc với SQLite

- Lấy danh sách tất cả các sản phẩm, yêu cầu hiển thị thông tin mã sản phẩm, tên sản phẩm, giá bán và tên danh mục có giá lớn hơn 25.

```
conn = sqlite3.connect("data/sale.db")
sql = """
SELECT P.id, P.name, P.price, C.name
FROM product P INNER JOIN category C ON C.id =
P.category_id
WHERE P.price > ?
"""
cursor = conn.execute(sql, (25, ))
for row in cursor:
    print(row)
conn.close()
```



The screenshot shows the PyCharm Run tool window titled "sqlite\_test". It displays the results of the executed SQL query. The results are:

id	name	price	category
2	iPad Mini	90.0	Máy tính bảng
4	Galaxy Note 10 Plus	30.0	Điện thoại

- Web Service thường triển khai một trong hai dạng
  - SOAP (Simple Object Access Protocol):
    - Dạng một giao thức.
    - Định nghĩa nhiều tiêu chuẩn phải tuân theo.
    - Chỉ hỗ trợ XML.
  - REST (Representational State Transfer)
    - Dạng một kiến trúc.
    - Sử dụng URI thể hiện logic nghiệp vụ.
    - Yêu cầu ít băng thông và tài nguyên hơn SOAP.
    - Sử dụng nhiều định dạng dữ liệu như HTML, XML, JSON.

# Các đặc điểm của REST

- Client-server: server cung cấp các dịch vụ và client là nơi nhận sử dụng dịch vụ từ server.
- Stateless: server không lưu thông tin client gửi lên server từ request. Do đó mỗi request từ client phải chứa đầy đủ thông tin yêu cầu từ server để thực hiện.
- Cacheable: kiến trúc REST có nhiều cấp lưu đệm (cache) từ client đến server. Đặc điểm này nhằm tăng hiệu năng REST trong môi trường mạng.
- Uniform Interface: cách thức giao tiếp client và server phải đồng nhất.



# RESTful Web Service

- API lấy danh sách các sản phẩm

```
@app.route('/products', methods=['GET'])
def get_products():
    conn = sqlite3.connect("data/sale.db")
    cursor = conn.execute("SELECT id, name FROM product")
    data = []
    for r in cursor:
        data.append({
            "id": r[0],
            "name": r[1]
        })
    conn.close()

    return jsonify({"products": data})
```



# RESTful Web Service

- API lấy chi tiết thông tin một sản phẩm

```
@app.route('/products/<int:product_id>', methods=['GET'])
def get_product_by_id(product_id):
    conn = sqlite3.connect("data/sale.db")
    sql = """
        SELECT P.name, P.price, C.name
        FROM product P INNER JOIN category C
        ON P.category_id=C.id
        WHERE P.id=?
    """
    r = conn.execute(sql, (product_id, )).fetchone()
    p = {
        "product_name": r[0],
        "product_price": r[1],
        "category_name": r[2]
    }
    conn.close()
    return jsonify({'product': p})
```



# RESTful Web Service

- API tìm kiếm sản phẩm theo tên

```
@app.route("/products/search", methods=['GET'])
def get_products_by_name():
    kw = request.args.get("kw")
    conn = sqlite3.connect("data/sale.db")
    sql = '''
        SELECT * FROM product WHERE name LIKE ?
    '''
    cursor = conn.execute(sql, ('%%%s%%' % kw, ))
    data = []
    for r in cursor:
        data.append({
            "id": r[0],
            "name": r[1]
        })
    conn.close()

    return jsonify({'products': data})
```



# RESTful Web Service

- API thêm sản phẩm mới

```
@app.route('/products/add', methods=["POST"])
def insert_product():
    conn = sqlite3.connect("data/sale.db")
    sql = """
        INSERT INTO product(name, price, category_id)
        VALUES(?, ?, ?)
    """
    if conn.execute(sql, (request.form.get("name"),
                          request.form.get("price", 0),
                          request.form.get("category_id", 1))):
        conn.commit()
        conn.close()
        return jsonify({"success": 1})
    return jsonify({"success": 0})
```



# RESTful Web Service

- Thực thi API thêm sản phẩm

The screenshot shows the Postman application interface. At the top, there is a header bar with the Python logo, the title "RESTful Web Service", and a search bar containing "POST http://127.0.0.1:5000/product...". Below the header are buttons for "+" and "...". To the right of the search bar are dropdown menus for "No Environment", "Comments 0", and settings.

The main area is titled "Untitled Request". It shows a POST request to "http://127.0.0.1:5000/products/add". There are "Send" and "Save" buttons. Below the URL, tabs are visible: "Params", "Authorization", "Headers (8)", "Body", "Pre-request Script", "Tests", "Settings", "Cookies", and "Code".

The "Body" tab is selected, showing options for "form-data" (which is selected) and other formats like "raw", "binary", and "GraphQL". Below this is a table with columns "KEY", "VALUE", and "DESCRIPTION". Three rows are present:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> name	iPhone 8+	
<input checked="" type="checkbox"/> price	29	
<input checked="" type="checkbox"/> category_id	1	

Below the table, there is a row for "Key" with "Value" and "Description" columns. At the bottom of the "Body" tab, there are buttons for "Pretty", "Raw", "Preview", "Visualize", and "JSON".

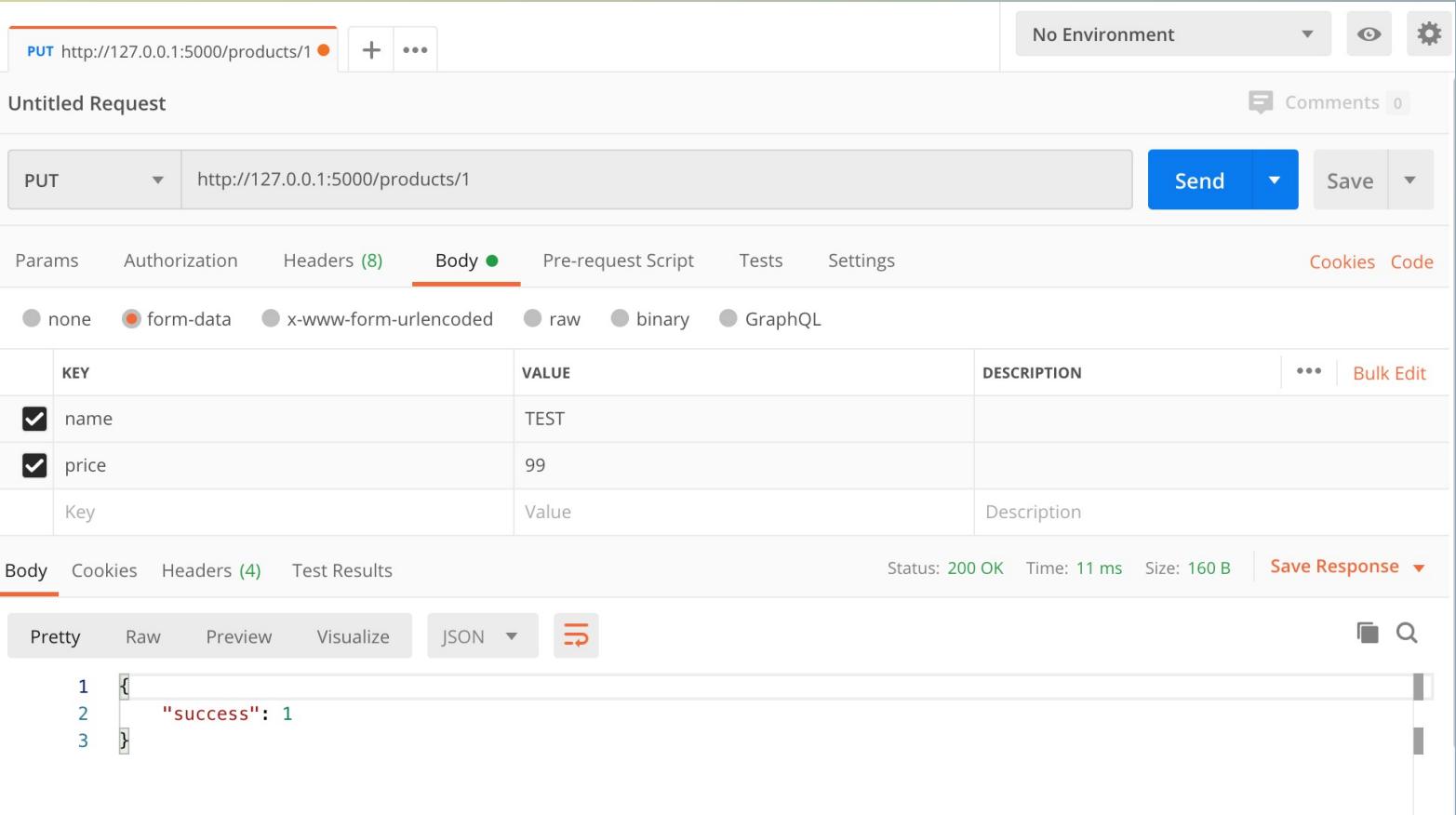
At the bottom of the interface, status information is displayed: "Status: 200 OK", "Time: 11 ms", "Size: 160 B", and a "Save Response" button. The entire interface has a light blue background with dark blue header elements.

- API cập nhật thông tin sản phẩm

```
@app.route('/products/<product_id>', methods=['PUT'])
def update_product(product_id):
    conn = sqlite3.connect("data/sale.db")
    sql = """
        UPDATE product SET name=?, price=?
        WHERE id=?
    """
    if conn.execute(sql, (request.form.get("name"),
                          request.form.get('price'),
                          product_id)):
        conn.commit()
        conn.close()
        return jsonify({'success': 1})
    return jsonify({'success': 0})
```

# RESTful Web Service

- Thực thi API cập nhật sản phẩm



The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `PUT http://127.0.0.1:5000/products/1`, a '+' button, a '...' button, and environment dropdown set to 'No Environment'. To the right are 'Comments' (0), a 'Send' button, a 'Save' button, and settings gear icons.

The main area is titled 'Untitled Request' and contains a 'Body' tab selected, showing the following JSON payload:

```
PUT http://127.0.0.1:5000/products/1
```

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> name	TEST			
<input checked="" type="checkbox"/> price	99			
Key	Value	Description		

Below the body, there are tabs for 'Body', 'Cookies', 'Headers (4)', and 'Test Results'. The status bar at the bottom shows 'Status: 200 OK Time: 11 ms Size: 160 B Save Response'.

The 'Body' tab displays the JSON response:

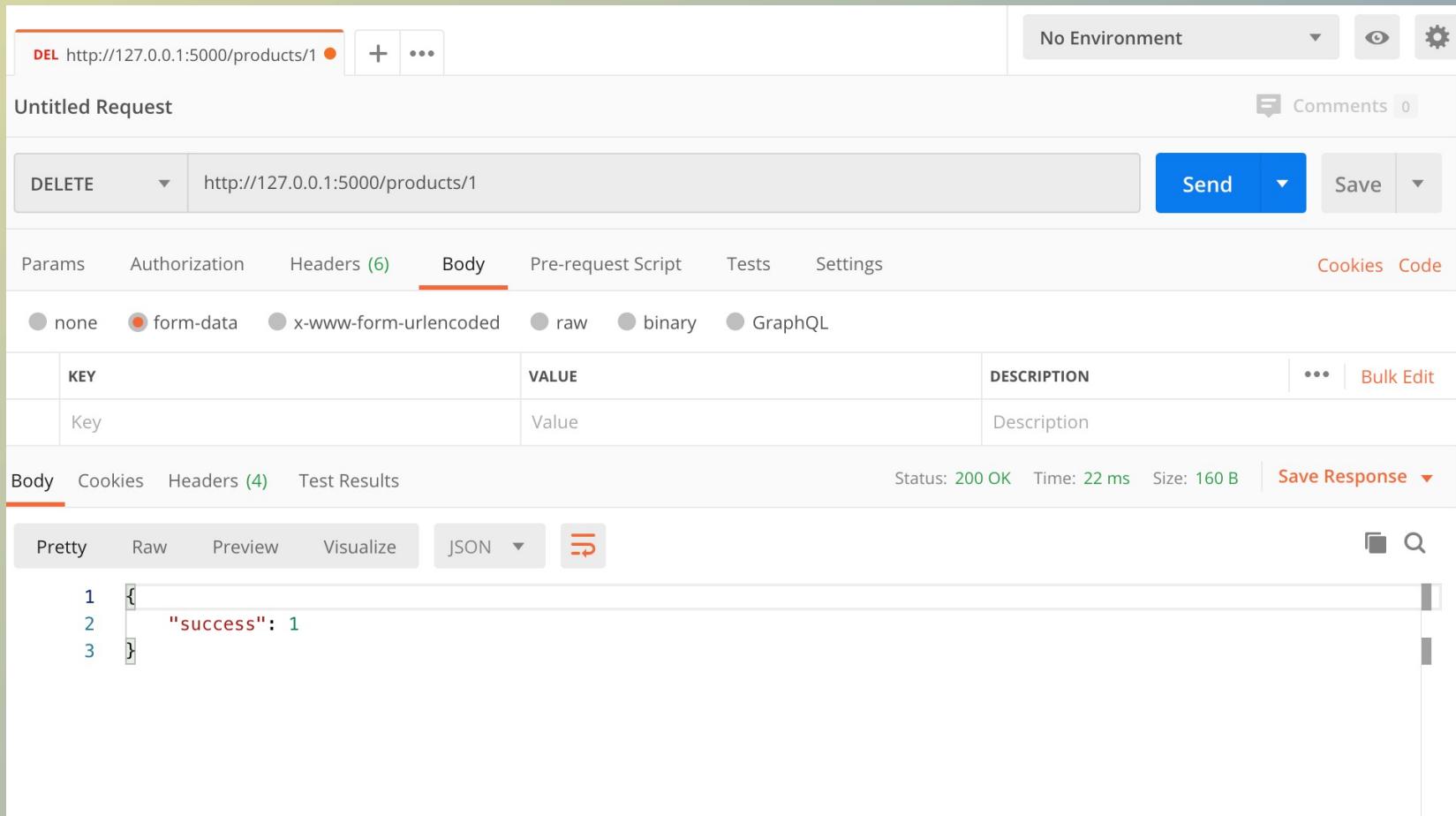
```
1 {  
2     "success": 1  
3 }
```

- API xoá sản phẩm

```
@app.route('/products/<product_id>', methods=['DELETE'])
def delete_product(product_id):
    conn = sqlite3.connect("data/sale.db")

    if conn.execute("DELETE FROM product WHERE id=?",
                   product_id):
        conn.commit()
        conn.close()
        return jsonify({'success': 1})
    return jsonify({'success': 0})
```

- Thực thi API xoá sản phẩm



Untitled Request

DELETE http://127.0.0.1:5000/products/1

No Environment

Comments 0

Send Save

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

Body

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Status: 200 OK Time: 22 ms Size: 160 B Save Response

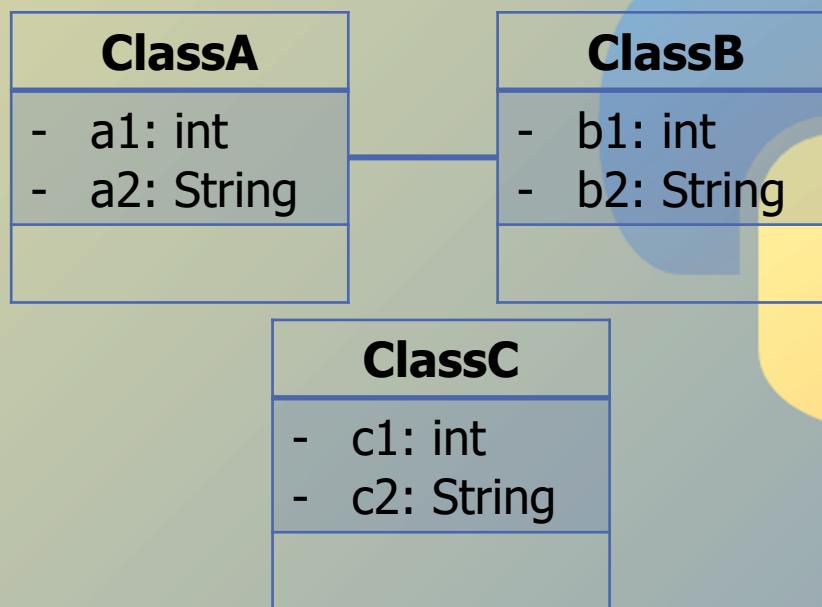
Pretty Raw Preview Visualize JSON

```
1 {  
2     "success": 1  
3 }
```

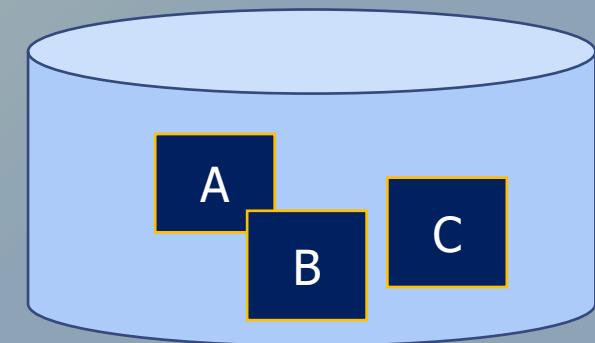
- SQLAlchemy là một công cụ Python, sử dụng kỹ thuật ORM tương tác với cơ sở dữ liệu.
- Để sử dụng SQLAlchemy trong ứng dụng flask ta cài extension flask-sqlalchemy.

```
pip install flask-sqlalchemy
```

- ORM (Object-Relational Mapping) là một kỹ thuật lập trình chuyển dữ liệu giữa các cơ sở dữ liệu quan hệ và các ngôn ngữ lập trình hướng đối tượng như Python, Java, C#, v.v.



Các lớp đối tượng



Cơ sở dữ liệu quan hệ

# Các cấu hình quan trọng

- `SQLALCHEMY_DATABASE_URI`
  - `mysql://username:password@server/db`
  - `postgresql://scott:tiger@localhost/mydatabase`
  - `oracle://scott:tiger@127.0.0.1:1521/sidname`
  - `sqlite:///absolute/path/to/foo.db`
- `SQLALCHEMY_ENGINE_OPTIONS`



# Cấu hình quan trọng

- Trong các ví dụ tiếp theo ta sẽ tương tác với cơ sở dữ liệu MySQL, nên ta cài thư viện pymysql

```
pip install pymysql
```

- Cấu hình kết nối MySQL

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] =
"mysql+pymysql://root:12345678@localhost/flashsaledb?ch
arset=utf8mb4"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = True

db = SQLAlchemy(app)
```

# Các lớp model

- Lớp cơ sở của các lớp model là db.Model.
- Tên bảng mặc định sẽ lấy tên lớp chuyển thành các ký tự thường (lowercase), để chỉ định tên bảng sử dụng thuộc tính `__tablename__` trong lớp model.
- Để tạo cơ sở dữ liệu đã thiết lập thực thi lệnh

```
db.create_all()
```

# Các lớp model

- Sử dụng Column để định nghĩa cột
  - Đối số đầu tiên của Column là kiểu dữ liệu cột
    - String(size)
    - Text
    - DateTime
    - Float
    - Boolean
    - PickleType
    - LargeBinary



# Các lớp model

- Một số thuộc tính Column
  - primary\_key = True chỉ định thuộc tính khoá.
  - autoincrement = True chỉ định thuộc tính tự tăng.
  - nullable = False tạo trường bắt buộc khác null.
- Ví dụ

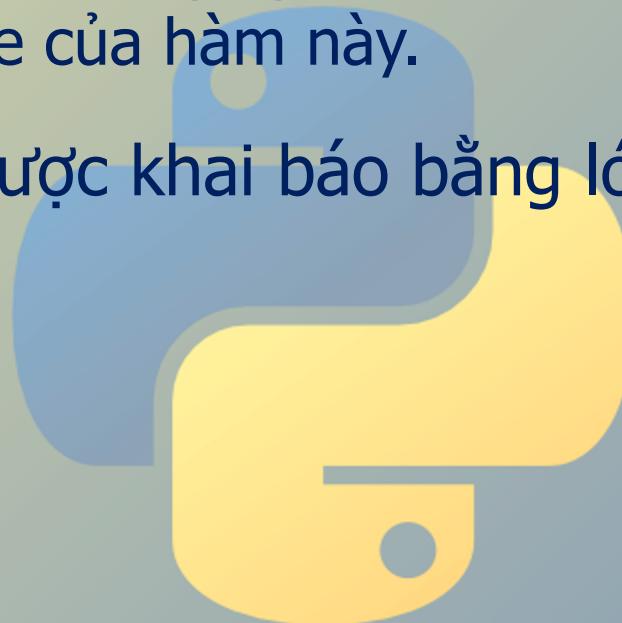
```
from sqlalchemy import Column, Integer, String
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy(app)

class Category(db.Model):
    __tablename__ = 'category'

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(50), nullable=False)
```

# Quan hệ One-To-Many

- Các quan hệ thiết lập bằng hàm relationship().
  - Nếu muốn thiết lập quan hệ 1-1 thì bật thuộc tính uselist=False của hàm này.
- Khoá ngoại được khai báo bằng lớp ForeignKey()





# Quan hệ One-To-Many

```
from sqlalchemy import Column, Integer, String, Float, ForeignKey
from sqlalchemy.orm import relationship
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy(app)

class Category(db.Model):
    __tablename__ = 'category'
    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(50), nullable=False)
products = relationship('Product',
                        backref='category', lazy=True)

class Product(db.Model):
    __tablename__ = 'product'
    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(50), nullable=False)
    price = Column(Float, default=0)
category_id = Column(Integer,
                      ForeignKey(Category.id), nullable=False)
```

# Quan hệ One-To-Many

- Thuộc tính backref và lazy trong relationship()
  - Thuộc tính backref chỉ định tạo thuộc tính category trên lớp Product, ta có thể gọi product.category để lấy danh mục của product đó.
  - Thuộc tính lazy có thể nhận giá trị
    - 'select'/True (mặc định) dữ liệu sẽ được nạp khi cần thiết.
    - 'joined'/False nạp mỗi quan hệ khi thực thi câu truy vấn cha sử dụng lệnh JOIN.
    - 'subquery' làm việc tương tự giá trị 'joined' nhưng sử dụng truy vấn con (subquery) thay vì lệnh JOIN.
    - 'dynamic': thay vì nạp dữ liệu, SQLAlchemy sẽ trả về một đối tượng truy vấn (query) khác, cho phép chỉnh sửa trước khi nạp dữ liệu.

# Quan hệ Many-To-Many

- Để định nghĩa quan hệ many-to-many cần định nghĩa bảng trung gian, bảng này không cần định nghĩa như một lớp model, nhưng nó sẽ là một bảng thực sự dưới cơ sở dữ liệu.
- Ví dụ: giả sử một sản phẩm có thể thuộc nhiều nhà sản xuất và nhà sản xuất sản xuất nhiều sản phẩm.



# Quan hệ Many-To-Many

```
prod_manufactuer = db.Table('prod_manufactuer',
    Column('product_id', Integer,
           ForeignKey('product.id'),
           primary_key=True),
    Column('manufacturer_id', Integer,
           ForeignKey('manufacturer.id'),
           primary_key=True))

class Product(db.Model):
    ...
    manufacturers = relationship('Manufacturer',
        secondary='prod_manufactuer',
        lazy='subquery',
        backref=backref('products', lazy=True))

class Manufacturer(db.Model):
    __tablename__ = 'manufacturer'
    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(50), nullable=False)
    country = Column(String(50), nullable=False)
```

# Tương tác với cơ sở dữ liệu

- Thêm dữ liệu

## # Thêm dòng vào bảng category

```
c = Category(name="Mobile Phone")  
db.session.add(c)  
db.session.commit()
```

## # Thêm dòng vào bảng manufacturer

```
m = Manufacturer(name="Apple", country="America")  
db.session.add(m)  
db.session.commit()
```

## # Thêm dòng vào bảng product

```
p = Product(name="iPhone 7", price=18, category=c)  
p.manufacturers.add(m)  
p.manufacturers.append(m)  
db.session.add(p)  
db.session.commit()
```

# Tương tác với cơ sở dữ liệu

- Cập nhật dữ liệu

```
p = Product.query.get(1)  
p.price = 22  
db.session.add(p)  
db.session.commit()
```

- Xoá dữ liệu

```
p = Product.query.get(1)  
db.session.delete(p)  
db.session.commit()
```

# Tương tác với cơ sở dữ liệu

- Flask-SQLAlchemy cung cấp thuộc tính query trong lớp Model để thực hiện truy vấn dữ liệu.
- Khi truy cập vào thuộc tính này, ta sẽ có một đối tượng truy vấn mới trên tất cả các record.
- Sử dụng phương thức **filter()** để lọc dữ liệu theo điều kiện nào đó trước khi thực thi các phương thức chọn dữ liệu là **all()** hoặc **first()**.
- Sử dụng phương thức **get()** để lấy dữ liệu theo khoá chính.

# Tương tác với cơ sở dữ liệu

```
# Lấy sản phẩm có id = 1
```

```
Product.query.get(1)
```

```
# Lấy sản phẩm có tên bắt đầu 'iPhone'
```

```
Product.query.filter(Product.name.startswith('iPhone'))\n    .all()
```

```
# Lấy sản phẩm có tên chứa từ 'galaxy'
```

```
Product.query.filter(Product.name.contains('galaxy'))\n    .all()
```

```
# Lấy sản phẩm có giá > 15 và < 20
```

```
Product.query.filter(Product.price.__gt__(15),\n                     Product.price.__lt__(20))\n    .order_by(Product.id).all()
```



# Tương tác với cơ sở dữ liệu

```
from sqlalchemy import or_
# Tìm sản phẩm có id=2 hoặc có giá > 30
Product.query.filter(or_(Product.id==2,
                           Product.price.__gt__(30))) \
               .all()

# Tìm sản phẩm có tên danh mục chứa từ 'mobile'
Product.query.join(Category,
                     Product.category_id == Category.id) \
               .filter(Category.name.contains("mobile")) \
               .add_columns(Category.name).all()
```

# Tương tác với cơ sở dữ liệu

```
from sqlalchemy import func
```

**# Tìm sản phẩm có giá cao nhất**

```
db.session.query(func.max(Product.price)).first()
```

**# Đếm số lượng sản phẩm, giá trung bình từng danh mục**

```
db.session.query(Category.id, Category.name,  
                  func.count(Product.id).label('So Luong'),  
                  func.avg(Product.price).label('Gia TB')) \  
    .join(Product,  
          Product.category_id == Category.id,  
          isouter=True) \  
    .group_by(Category.id, Category.name).all()
```

**# Chú ý: isouter = True → LEFT OUTER JOIN**

- Flask Admin nhằm xây dựng phân hệ quản trị Website với mô hình dữ liệu đã xây dựng.
- Flask Admin giúp tiết kiệm thời gian và chi phí xây dựng phân hệ admin, tập trung phần xử lý nghiệp vụ phía người dùng.
- Cài đặt thư viện

```
pip install flask-admin
```

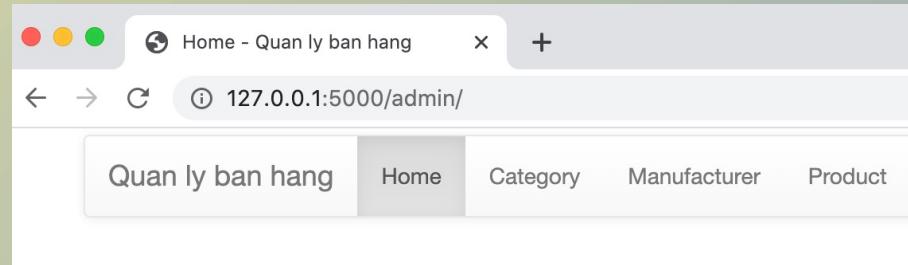
- Khởi tạo Flask Admin và thêm các trang quản trị

```
from flask_admin.contrib.sqla import ModelView

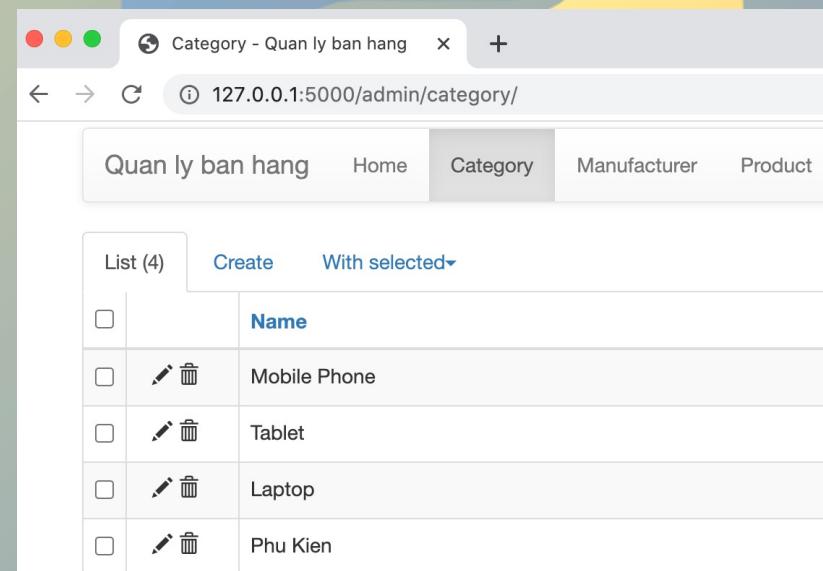
admin = Admin(app, name="Quan ly ban hang",
              template_mode="bootstrap3")
admin.add_view(ModelView(Category, db.session))
admin.add_view(ModelView(Manufacturer, db.session))
admin.add_view(ModelView(Product, db.session))
```

- Chạy ứng dụng và truy cập
  - <http://127.0.0.1:5000/admin>

- Trang chủ



- Truy cập tab Category



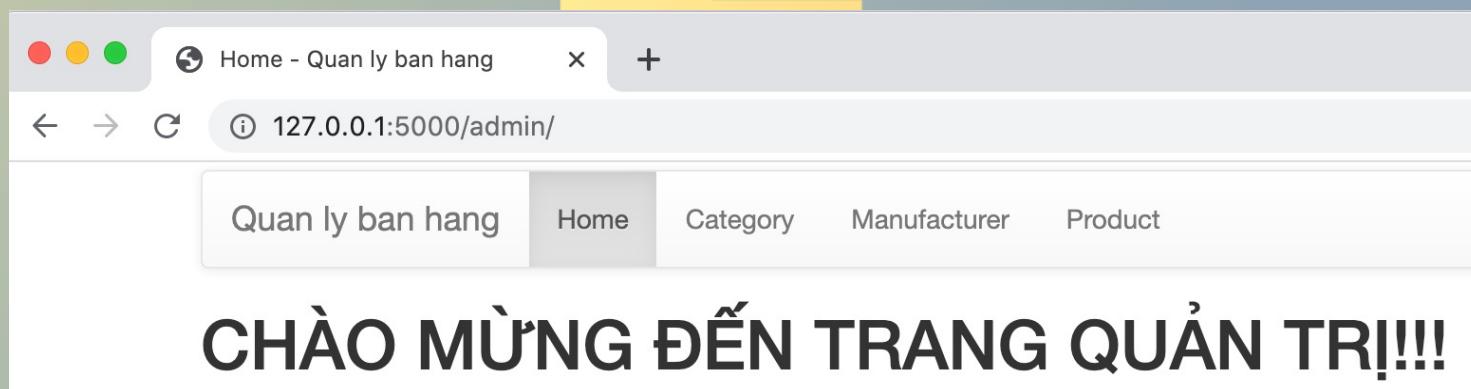
	Name
<input type="checkbox"/>	Mobile Phone
<input type="checkbox"/>	Tablet
<input type="checkbox"/>	Laptop
<input type="checkbox"/>	Phu Kien

# Tuỳ chỉnh các trang quản trị

- Thêm nội dung vào trang chủ
  - Trong templates tạo admin/index.html, trang này kế thừa admin/master.html.

```
{% extends 'admin/master.html' %}

{% block body %}
    <h1>CHÀO MỪNG ĐẾN TRANG QUẢN TRỊ!!!!</h1>
{% endblock %}
```



# Tuỳ chỉnh các trang quản trị

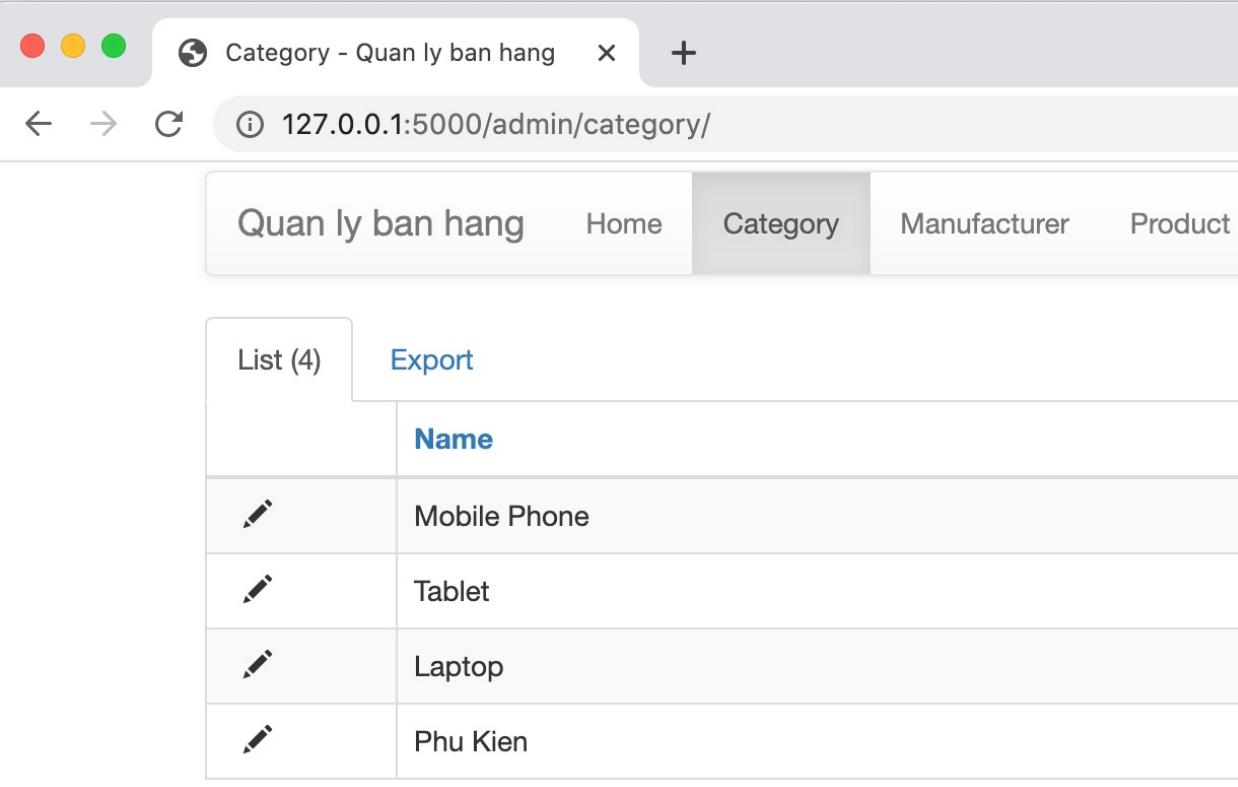
- Để tùy chỉnh các trang quản trị khác, ta tạo lớp con kế thừa ModelView và sử dụng lớp con này khi thêm model vào admin.

```
class CategoryModelView(ModelView):  
    column_display_pk = False  
    can_create = False  
    can_edit = True  
    can_export = True  
    can_delete = False  
    can_export = True  
    form_columns = ('name', )
```

```
admin.add_view(CategoryModelView(Category, db.session))
```

# Tùy chỉnh các trang quản trị

- Truy cập lại vào trang quản trị Category



	Name
	Mobile Phone
	Tablet
	Laptop
	Phu Kien

# Tùy chỉnh các trang quản trị

- Để tạo View riêng không liên quan đến các model, ta tạo một class kế thừa BaseView và sử dụng annotation @expose.

```
from flask_admin import BaseView, expose

class AboutUsView(BaseView):
    @expose('/')
    def index(self):
        return self.render('admin/about-us.html')
```

```
admin.add_view(AboutUsView(name="Giới thiệu"))
```

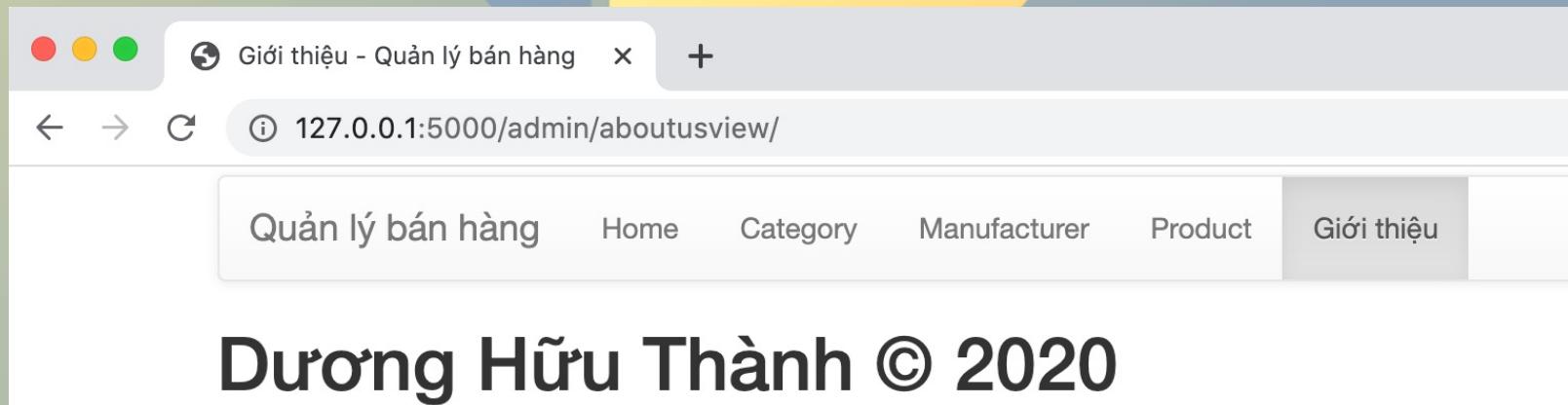
# Tuỳ chỉnh các trang quản trị

- Trang admin/about-us.html

```
{% extends 'admin/master.html' %}

{% block body %}
    <h1>Dương Hữu Thành © 2020</h1>
{% endblock %}
```

- Truy cập lại vào trang quản trị



# Tuỳ chỉnh các trang quản trị

- Xử lý đăng nhập và đăng xuất admin, cài đặt flask-login

```
pip install flask-login
```

- Tạo thể hiện của LoginManager cho ứng dụng

```
from flask_login import LoginManager  
  
...  
login = LoginManager(app)
```

# Tùy chỉnh các trang quản trị

- Tạo một model lưu trữ thông tin user

```
from flask_login import UserMixin

class User(db.Model, UserMixin):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True,
                 autoincrement=True)
    name = Column(String(50), nullable=False)
    active = Column(Boolean, default=True)
    username = Column(String(50), nullable=False)
    password = Column(String(50), nullable=False)
```

# Tùy chỉnh các trang quản trị

- Ta cần cung cấp phương thức callback user\_load để nạp lại đối tượng user từ user id trong session.

```
@login.user_loader  
def load_user(user_id):  
    return User.query.get(user_id)
```

# Tùy chỉnh các trang quản trị

- Chỉnh sửa trang của quản trị admin/index.html

```
{% extends 'admin/master.html' %}

{% block body %}
    {% if current_user.is_authenticated %}
        <h1>CHÀO MỪNG ĐẾN TRANG QUẢN TRỊ!!!</h1>
    {% else %}
        {% include 'admin/login.html' %}
    {% endif %}
{% endblock %}
```

# Tùy chỉnh các trang quản trị

- Trong đó admin/login.html

```
<form action="{{ url_for('login_admin') }} " method="post">
    <div class="form-group">
        <label>Username</label>
        <input type="text" name="username"
               class="form-control" />
    </div>
    <div class="form-group">
        <label>Password</label>
        <input type="password" name="password"
               class="form-control" />
    </div>
    <div class="form-group">
        <input type="submit" value="Đăng nhập"
               class="btn btn-danger" />
    </div>
</form>
```

# Tùy chỉnh các trang quản trị

- Viết action xử lý đăng nhập

```
from flask_login import login_user

@app.route("/login-admin", methods=['GET', 'POST'])
def login_admin():
    if request.method == 'POST':
        username = request.form.get("username")
        password = request.form.get("password")
        user = User.query.filter(username == username,
                                password == password).first()
        if user:
            login_user(user=user)
    return redirect("/admin")
```

# Tùy chỉnh các trang quản trị

- Tạo View xử lý đăng xuất khỏi hệ thống

```
from flask_admin import BaseView, expose  
from flask_login import logout_user  
  
class LogoutView(BaseView):  
    @expose('/')  
    def index(self):  
        logout_user()  
        return redirect('/admin')
```

- Thêm vào admin view

```
admin.add_view(LogoutView(name="Đăng xuất"))
```

# Tuỳ chỉnh các trang quản trị

- Các ModelView muốn bắt buộc người dùng đăng nhập mới được truy cập có thể ghi đè lại phương thức `is_accessible` và trả về trạng thái chứng thực `current_user` của flask-login.
- Ví dụ LogoutView chỉ hiển thị sau khi đăng nhập

```
class LogoutView(BaseView):  
    @expose('/')  
    def index(self):  
        logout_user()  
        return redirect('/admin')  
  
    def is_accessible(self):  
        return current_user.is_authenticated
```

# Tuỳ chỉnh các trang quản trị

- Ta cũng có thể định nghĩa lớp View chung để để các lớp con kế thừa.

```
class AuthenticatedView(ModelView):  
    def is_accessible(self):  
        return current_user.is_authenticated
```

- CategoryModelView yêu cầu phải login

```
class CategoryModelView(AuthenticatedView):  
    ...
```



# Tuỳ chỉnh các trang quản trị

Home - Quản lý bán hàng

127.0.0.1:5000/admin/

Quản lý bán hàng Home Manufacturer Product User Giới thiệu

Username  
admin

Password  
.....

Đăng nhập

Home - Quản lý bán hàng

127.0.0.1:5000/admin/

Quản lý bán hàng

Home Category Manufacturer Product User Giới thiệu Đăng xuất

CHÀO MỪNG ĐẾN TRANG QUẢN TRỊ!!!

# Tuỳ chỉnh các trang quản trị

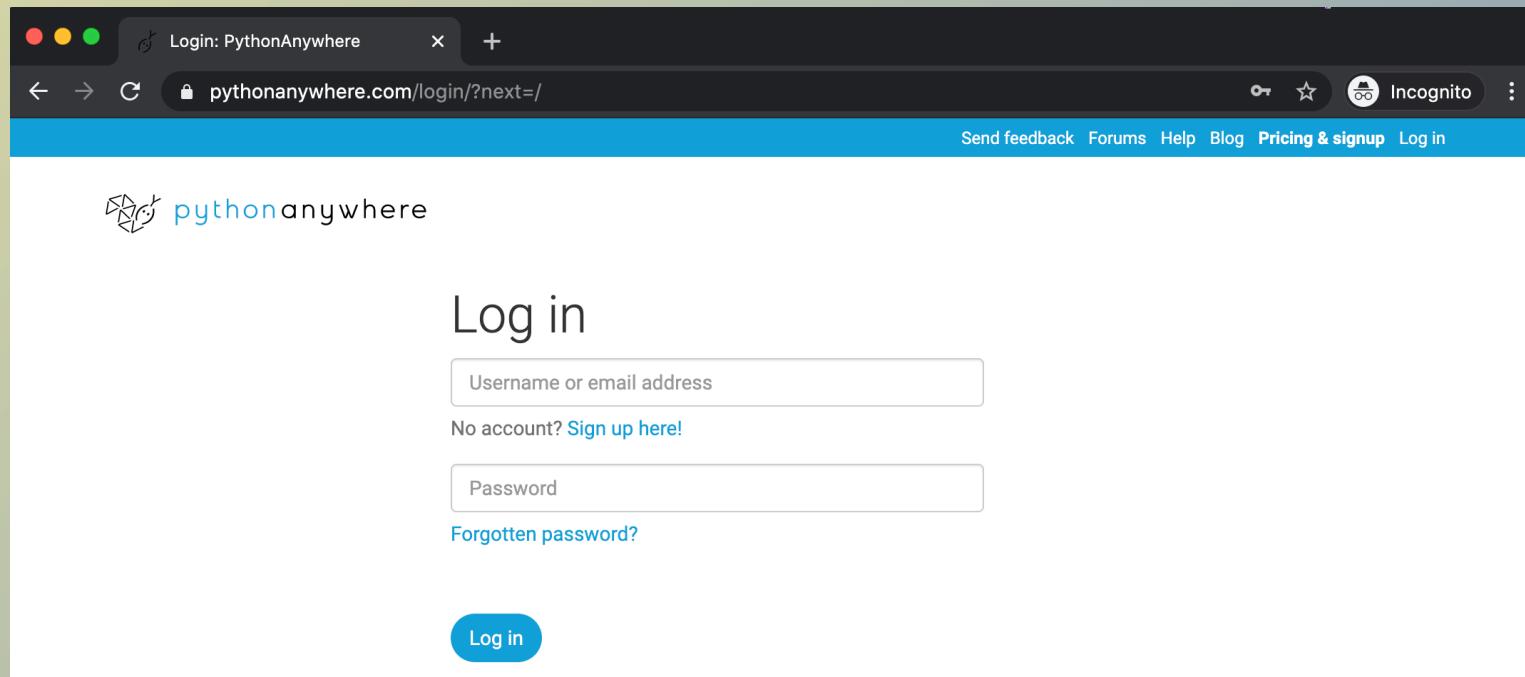
- Ở các view của ứng dụng muốn bắt buộc người dùng phải đăng nhập mới được phép truy cập có thể sử dụng annotation `@login_required`

```
from flask_login import login_required  
  
@app.route("/payment")  
@login_required  
def pay():  
    pass
```



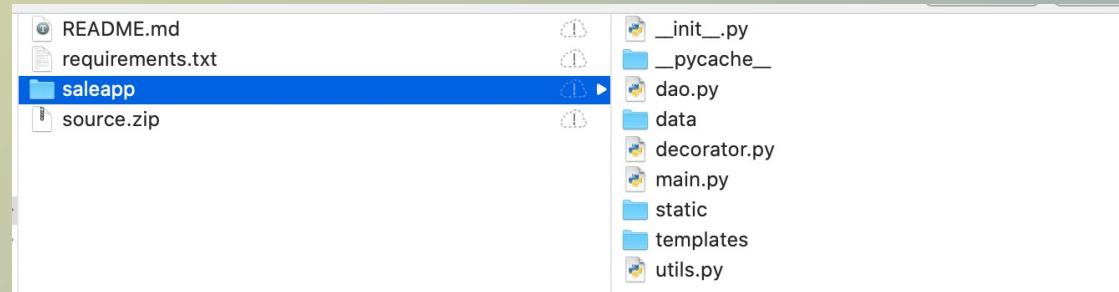
# Triển khai lên Python Anywhere

- Truy cập vào trang pythonanywhere.com

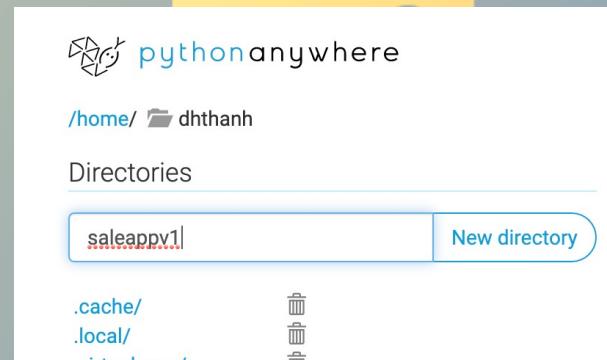


# Triển khai lên Python Anywhere

- Giả sử ta có cấu trúc project như sau:



- Nén mã nguồn cần triển khai đặt tên source.zip
- Trên pythonanywhere tạo thư mục saleappv1 và upload tập tin nén mã nguồn vào thư mục này.



# Triển khai lên Python Anywhere

- Trong bash console truy cập vào thư mục saleappv1 và thực hiện lệnh giải nén:

>> **unzip source.zip**



Bash console 13530082

```
05:07 ~ $ pwd  
/home/dhthanh  
05:07 ~ $ cd saleappv1/  
05:07 ~/saleappv1 $ unzip source.zip  
Archive: source.zip  
  inflating: README.md  
  inflating: requirements.txt  
  creating: saleapp/
```



# Triển khai lên Python Anywhere

- Trong Bash Console tạo môi trường ảo và cài đặt các thư viện (trong tập tin requirements.txt)

```
virtualenv --python=/usr/bin/python3.7 venv  
source venv/bin/activate  
Pip install -r requirements.txt
```

# Triển khai lên Python Anywhere

- Tạo một ứng dụng Web trong tab Web thiết lập các thông tin cần thiết chạy ứng dụng Web
- Chỉ định thư mục source code và môi trường ảo

Code:

What your site is running.

Source code:	<a href="#">/home/dhthanh/saleappv1</a>	<a href="#">↗ Go to directory</a>
Working directory:	<a href="#">/home/dhthanh/</a>	<a href="#">↗ Go to directory</a>
WSGI configuration file:	<a href="#">/var/www/dhthanh_pythonanywhere_com_wsgi.py</a>	
Python version:	3.7 	

Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. [More info here](#). You need to **Reload your web app** to activate it; NB - will do nothing if the virtualenv does not exist.

[/home/dhthanh/saleappv1/venv](#)

 [Start a console in this virtualenv](#)

# Triển khai lên Python Anywhere

- Trong đó sửa tập tin wsgi như sau

```
import sys

path = '/home/dhthanh/saleappv1'
if path not in sys.path:
    sys.path.append(path)

from saleapp.main import app as application
```





# PHÁT TRIỂN ỨNG DỤNG WEB VỚI PYTHON FLASK

Q & A



Dương Hữu Thành