Lập trình Flask tạo WEBSITE

Python – Flask web

Tạ Quang Tùng (Tuvo)

K66A2 – Toán Tin

Khoa Toán – Cơ – Tin học

PISCES KIBO

Mục lục:

1. TẠO WEB ĐƠN GIẢN VỚI FLASK:

- Khai báo thư viện flask bằng cách: *from flask import Flask*
- Khởi tạo app bằng cách: *app = Flask(__name__)*
- Tạo định tuyến: *@app.route('/')*
- Gọi hàm def để tạo chương trình
- Hiển thị trình duyệt web bằng cách:

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

VD:

a) Tạo thẻ tiêu đề trực tiếp trong hàm:

- Kết hợp một số kiến thức về html để tạo tiêu đề in đậm với heading và header:
  - Tạo tiêu đề in đậm với heading (Thẻ tiêu đề):
    - "<hi> Tiêu đề </hi>" với i là kích cở nhỏ dần (h1 > h2 > …)
  - Tạo thẻ văn bản: Tên văn bản
  - Thêm trang trí màu sắc trong header:
    - <hi style = "color: màu sắc;" tên tiêu đề </hi>

b) Cách truyền biến và chuyển hướng trang:

- Thông qua dạng định tuyến: ***@app.route('user/<name>')***
- Kết hợp hàm được khai báo ngay phía sau: (biến name phải giống như trên)

```
def hello_user(name):
    if name == 'admin':
        # Chuyển hướng trang
        return redirect(url_for(hello_admin))
```

```
    return f"<h1> Hello {name}!</h1>"
    # Sửa {name} ngay trên link bằng cách "link/user/tên"
```

- Chuyển hướng trang bằng ***import url_for***
  - Chuyển hướng trang trong hàm bằng cách

return redirect(url_for(tên hàm))

c) Làm việc đối với biến kiểu khác:

VD:

```
# Làm việc đối với biến int
@app.route('/blog/<int:blog_id>')
def blog(blog_id):
    return f"<h3> Blog {blog_id}!</h3>"
```

- Tạo biến kiểu type: int, float, … tại phần định tuyến sẽ có dạng:

@app.route('/blog/<type: tên biến>')

d) Tạo truyền biến Templates và Render Template:

- Khai báo thư viện: ***from flask import Flask, render_template***
- Khởi tạo: ***app = Flask(__name__)***
- Tạo đường định tuyến: ***@app.route('/')***
- Tạo hàm để truyền biến vào render_template:

VD:

```
@app.route('/')
def hello_world():
    #Truyền biến vào render_template
    return render_template('index.html', content = "Tùng đẹp trai",
                           car = ["Vinfast", "BMW", "Mer"])
```

⇨ Render_template có chức năng mở sang file html và thực hiện code html ở file đó

2. KHÁI QUÁT CHUNG VỀ HTML:

- Làm việc với file html
- Định dạng cấu trúc của HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title> TIÊU ĐỀ </title>
</head>
<body>
    <h1 style = "color: MÀU SẮC;"> VĂN BẢN {{content}}</h1>
    {% for x  in range(10) %}
        {% if x % 3 == 0 %}
            <h2>{{x}}</h2>
        {% endif %}
    {% endfor %}

    {% for y in car %}
        <h3>{{y}}</h3>
    {% endfor %}
</body>
</html>
```

  - Chú thích:
    - <h1 style = “color: Màu sắc;”> Văn bản {{giá trị}} </h1> là thẻ tag tiểu đề
    - Có thể dùng câu lệnh điều kiện và vòng lặp trong html với điều kiện:
      - Mỗi 1 dòng phải có: {% câu lệnh %}
      - Giá trị biến: <hi> {{x}} </hi>
      - Kết thúc câu điều kiện hoặc vòng lặp phải có:

        {% endif %} hoặc {% endfor %}
  - Phông dạng:

```
<html>
<head>
        <title> TIÊU ĐỀ </title>
</head>

<body>
        <hi style = “color: MÀU SẮC;”> VĂN BẢN {{content}} </hi>
         Có thể sử dụng các câu lệnh như Python với dấu { }
         <p> TÊN VĂN BẢN PARAGRAPH </p>
</body>
</html>
```

3. KẾ THỪA TEMPLATE VÀ SỬ DỤNG BOOTSTRAP:

a) Kế thừa Template:

- Sử dụng chủ yếu trên file html
- Tạo 1 file định dạng: base.html và chứa code của HTML
  - Mục title thêm: **<title> {% block title %} {% endblock %} </title>**
  - Mục body có thể thêm thẻ <hi> tên </hi> và dưới thẻ chứa

    {% block content %} {% endblock %}
- Tạo 1 file home để kế thừa từ file base.html (không cần code html)
  - Tạo khối lệnh extends bằng cách: **{% extends "base.html" %}**
  - Tiếp tục tạo block title: **{% block title %}Home Page{% endblock %}**
  - Tạo block content: **{% block content %} <thêm thẻ> {% endblock %}**
- Quay trở lại file python để chạy chương trình với hàm chứa

  render_template("home.html")

  VD:

```
#Lấy hai file base.html và file home.html

from flask import Flask, redirect, url_for, render_template

app = Flask(__name__)

@app.route("/")
def hello_world():
    return render_template("home.html")

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

b) Thư viện Bootstrap về CSS:

- Copy link CSS: https://getbootstrap.com/docs/4.3/getting-started/introduction/

⇨ Copy vào trong thẻ **<head> copypaste </head>**

⇨ Copy theo kiểu internal

- Copy link Javascipt:

  https://getbootstrap.com/docs/4.3/getting-started/introduction/

⇨ Để dưới hẳn thẻ </body>

- Copy link Navbar: https://getbootstrap.com/docs/4.3/components/navbar/
  ⇨ Cho dưới thẻ <hi> … </hi> trong phần body

4. HTTP METHODS, POST VÀ GET:

+ Phương thức Get request có tác dụng gửi đến server và nhận về nội dung yêu cầu
+ Phương thức Post có tác dụng gửi post request thông qua biểu mẫu

⇨ **HTTP Methods: (HyperText Transfer Protocol)**

a) **Get Request:** **"GET / HTTP/1.1" 200 -** ***#Ví dụ định dạng***

b) **Post Request:**

- Xuất thư viện request: ***from flask import Flask, request***
- Tạo file kế thừa (vd: login.html) để kế thừa file base.html
- Tạo thêm thẻ div là form bao quát và có dạng: (ví dụ)

```
{% block content %}
<div style = "margin-left: 50px;">
    <form action = "#" method = "name">
            <label for = " "> Name </label>
            <input type = "text" name = "name"> </input>
            <button style = "background-color: màu ô; color: màu chữ;" type =
            "submit"> Login </button>
    </form>
</div>
{% endblock %}
```

- Khởi tạo thêm hàm chứa file html trên tại file python
- Định dạng ví dụ: **"POST /login HTTP/1.1" 302 –**

c) **Kết hợp POST và GET lại với nhau:**

- Mở file python và tại đường định tuyến thêm method chứa POST và GET kiểu list

  @app.route("/login", method = ["POST", "GET"]

- Sử dụng thư viện request trong hàm bằng cách kết hợp câu lệnh điều kiện:

```
def login():
    if request.method == "POST":
        user_name = request.form["name"]
        if user_name:
            return redirect(url_for("hello_user", name = user_name))
    return render_template("login.html")
```

5. SESSION VÀ LIFETIME SESSION:

+ Thiết lập giữa Server và Client

+ Tập trung cho lập trình WEB backend

+ SESSION có chức năng như phiên làm việc định kỳ (không dùng trong khoảng thời gian dài sẽ tự động đăng xuất ra) → Lưu trữ tạm thời (SESSION ID #đặc biệt)

- Xuất thư viện SESSION bằng cách: **from flask import Flask, session**
- Tạo tính bảo mật cho SESSION: **app.config[“SECRECT_KEY”] = “String random”**
- Định dạng của SESSION: **session[“name_user”]**

#Cách thoát ra khỏi lưu trữ của SESSION

- Khởi tạo định tuyến mới và gọi hàm xóa xuất dữ liệu ra:

session.pop(“user”, None)

return redirect(url_for(“tên hàm”))

#Sử dụng LIFETIME SESSION

- Thời gian cho phiên làm việc mặc định 30 – 31 ngày (có thể set lại lifetime cho web)
- Xuất thư viện datetime để import timedelta: ***from datetime import timedelta***
- Tạo thêm set thời gian thoát ra:

```
app.permanent_session_lifetime = timedelta(minutes = phút)      #Set tgian
```

- Chọn hàm cần thiết để thiết lập LIFETIME SESSION bằng cách:

session.permanent = True

6. FLASH MESSAGE:

+ Thiết lập thông báo sau khi xử lý thành công

+ Xuất thư viện flash từ flask: **from flask import flash**

+ Câu lệnh của flash có dạng: **flash ("Văn bản", 'fla')** # "fla" kiểu:

- 'message' → nhiều loại chung
- 'error' → thông báo lỗi
- 'infor' → tin nhắn thông tin
- 'warning' → cảnh báo

+ Tiếp tục sử dụng file html để viết code python với khối {block content} với thứ tự lớn nhất là ***{% with messages = get_flashed_messages() %}*** và dưới câu lệnh này là các câu điều kiện hoặc vòng lặp và khi kết thúc sẽ có thể ***{% endwith %}***

7. TỔNG QUAN ORM, SQLALCHEMY:

a) ORM (Object Relation Mapping):

+ Mỗi khi tương tác với relational database (tạo, đọc, sửa, xóa dữ liệu trong bảng), họ phải viết câu lệnh SQL thuần và execute nó ở trong chương trình sử dụng connector

+ Tưởng tượng có 1 bảng Car với 3 cột name, brand, year trong database và bạn muốn thêm 1 dòng dữ liệu (record) vào đó, ta sẽ phải làm như sau: (câu lệnh SQL python)

```
conn = sqlite3.connect('car.db')
c = conn.cursor()
sql_insert = f"""
    INSERT INTO cars VALUES (:name, :brand, :year)
"""
def insert_car(car):
    with conn:
        c.execute(sql_insert, {'name': car.name, 'brand': car.brand, 'year': car.year})
```

- Viết câu lệnh SQL thuần dưới dạng string rồi dùng connector để exercute nó → SQL thuần thường dài dòng, khó thay đổi, không tối ưu
- Ngoài việc cập nhập, thay đổi cơ sở dữ liệu không dễ dàng. Ví dụ như mỗi lần thay đổi cấu trúc bảng, nếu không có ORM, chúng ta sẽ phải truy cập vào database server và thay đổi bằng câu lệnh SQL, dẫn tới quy trình làm việc khó khăn ở 2 môi trường: application code và database server
- Trong khi đó, sử dụng ORM → có thể thay đổi bảng rồi thực hiện migrate trong application code và tránh làm việc trong database server

⇨ ORM giải quyết vấn đề quản trị hiệu quả hơn (Object Relation Mapping)

⇨ Kỹ thuật lập trình biểu diễn các dòng dữ liệu (record) trong CSDL bằng đối tượng, vật thể trong ngôn ngữ lập trình

b) Các loại ORM:

- **Active Record** → mô hình thiết kế ORM trong mỗi bảng CSDL và gói gọn trong model → mỗi object thuộc model sẽ được gắn với 1 dòng trong bảng
    - Với Active Record, ở trong model thì người dùng không cần ghi rõ properties của model đó hay sự liên hệ của những properties ấy đến database mà object tự động biết chúng bằng cách nhìn vào database schema
    - Các ORM sử dụng Active Record có sẵn CRUD (Create – Read – Update – Delete) method như save(), create(), … → tạo project

- Những ORM nổi tiếng sử dụng Active Record: Eloquent, Ruby on Rails, Django's ORM, Yii

- **Data Mapper** → ORM đóng vai trò như 1 lớp (layer) có chức năng tách biệt như vận chuyển dữ liệu hai chiều giữa CSDL và ứng dụng
  - Các object ở application sẽ không có thông tin gì về database hay những thuộc tính các models
  - Những ORM sử dụng Mapper: Doctrine, Hibernate, SQLAlchemy

c) <u>Ưu điểm và Nhược điểm:</u>

- Ưu điểm:
  - Ngắn gọn: ORM sử dụng ngôn ngữ lập trình với cú pháp ngắn gọn, đơn giản so với SQL và giảm thiểu lượng code
  - Tối ưu: ORM cho phép tận dụng tối ưu phương pháp lập trình hướng đối tượng (Object – Oriented Programming) như kế thừa, đóng gói, khái quát hóa thông qua biểu diễn các record trong database dưới dạng object
  - Linh hoạt: chuyển đổi linh hoạt giữa các hệ quản trị CSDL khác nhau(MySQL, Sqlite3, PostgreSQL, …)
- Nhược điểm: Thời gian học dài framework và sự chủ động ít, kiểm soát hơn đối với database

d) <u>SQLAlchemy:</u>

+ SQLAlchemy là bộ công cụ SQL mã nguồn mở và ORM sử dụng trong ngôn ngữ lập trình Python → quản lý, thao tác với CSDL, cung cấp ORM sử dụng mô hình thiết kế DATA MAPPER

- Kết nối với database:

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///:memory:')
```

- Khai báo mapping và tạo schema:

  + ORM truyền thống gồm 2 bước: mô tả database chúng ta phải xử lý và khai báo class mà sẽ được đối chiếu với các bảng → SQLAlchemy gộp 2 bước này vào một và sử dụng Declarative Base

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()
```

```
class Employee(Base):
    __tablename__ = "employees"
    td = Column(Integer, primary_key = True)
    name = Column(String)
    age = Column(Integer)
```

- Query:

+ Trước khi thực hiện các tác vụ như query hay thay đổi database, chúng ta cần kết nối với database trong 1 session:

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker()
Session.configure(bind = engine)
session = Session()
```

+ Giả sử muốn tìm các Employee có tên Alice thì lấy kết quả đơn giản:

```
res = session.query(Employee).filter_by(name = "Alice").all()
```

+ Nếu quá nhiều người tên Alice và muốn giới hạn số lượng lấy ra xếp theo id:

```
res = session.query(Employee).filter_by(name = 'Alice').order_by(Employee.id).limit(5).all()
```

8. TẠO DATABASE SQLALCHEMY:

a) Config SQLAlchemy:

- Tải Flask-SQLAlchemy: ***pip install Flask-SQLAlchemy***
- Xuất thư viện flask_sqlalchemy: ***from flask_sqlalchemy import SQLAlchemy***
- Xuất thư viện os: ***from os import path***
- Thêm các config đặc trưng của SQLAlchemy:

```
app = Flask(__name__)
app.config["SECRET_KEY"] = "tungdz"                #Chuỗi bảo mật
app.config["SQLALCHEMY_DATABASE_URL"] = "sqlite:///user.db"          #Tạo
thư mục user ngay
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False              #Theo
dõi sự thay đổi
app.permanent_session_lifetime = timedelta(minutes = 1)      #Set thời
gian tự đăng xuất
```

- Khởi tạo: ***db = SQLAlchemy(app)***

b) Khởi tạo ORM thông qua hướng đối tượng: (Tạo bảng):

```
#Tạo bảng qua hướng đối tượng
class User(db.Model):
    #Bảng dữ liệu user
    user_id = db.Column(db.Integer, primary_key = True)
    name = db.Column(db.String(100))
    email = db.Column(db.String(100))

    def __init__(self, name, email):
        self.name = name
        self.email = email
```

c) Custom user:

- Sử dụng trang user.html và có sự sửa đổi
- Tìm kiếm SQLITE DB sẽ ra màn hình tool của SQL

9. ADD, UPDATE, DELETE USER:

a) Thêm USER vào database:

- Làm việc với bảng SQL
- Cách lưu vào database:

```
#Lưu user vào database
user = User(user_name, "temp@gmail.com")
db.session.add(user)
db.session.commit()      #Thực hiện cái thay đổi
```

- Thông báo đã lưu vào database thành công:

flask("Created in DB!")

b) Cập nhật USER vào SQL:

```
@app.route("/user", methods = ["POST", "GET"])
def user():
    email = None
    if "user" in session:
        name = session["user"]
        if request.method == "POST":
            email = request.form["email"]
            #Đưa vào email
            session["email"] = email
            found_user = User.query.filter_by(name = name).first()
            #Gán lại email (cập nhật email)
            found_user.email = email
            #Update có hiệu lực
            db.session.commit()

            #In ra màn hình
            flask("Email updated")

        elif "email" in session:
            email = session["email"]
        return render_template("user.html", user = name, email = email)
    else:
        flash("You haven't logged in!", "infor")
        return redirect(url_for("login"))
```

- Khi dùng flask có thể in ra màn hình bằng cách: ***flask("tên văn bản")***

c) Delete USER:

- Sửa đổi thêm logic button DELETE vào file html

  User.query.filter_by(name=name).delete()

  - User là Class hướng đối tượng
  - Phương thức delete() xóa sạch chương trình

10. CÁCH THÊM STATIC FILE:

a) File link CSS:

- Tạo giao diện WEB cơ bản và file html thông thường

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("house.html")

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

- Tạo folder cùng cấp với folder của các file html (VD: folder Static)
- Tạo file CSS trong folder Static
- Có thể đổi màu thẻ “Home Page”
- Đặt class cho thẻ <body> … </body> trong file html vừa tạo (thêm các thẻ hi, màu sắc, …)
    - Trong thẻ <head> … </head> có thêm link:

<link rel = "stylesheet" href = "{{url_for('tên folder thường', filename = 'styles/style.css')}}">

    - File name chứa tên đường truyền file CSS

b) Gán file ảnh CSS:

- Chọn file ảnh có sẵn của máy
- Tạo folder cùng cấp với file styles để chứa file ảnh
- Sử dụng link ảnh trong folder image tại thẻ <body> … </body>
    - Tạo thẻ img có định dạng link file ảnh:

<img src="{{url_for('tên folder tổng thường', filename = 'image/tên ảnh.jpg')}}" alt = " ">

11. DÙNG BLUEPRINT TÁCH MODULE:

- Chia code thành các Module nhỏ:
    - Tạo thêm các file.py
    - Thêm thư viện BLUEPRINT: ***from flask import Blueprint***
    - Gán qua Blueprint: ***user = Blueprint("name_file", __name__)***
    - Đường định tuyến đổi thành: ***@user.route("/")***

  ⇨ Làm việc trên file.py mới
- Tại file chính sẽ import file.py mới ra và thêm câu lệnh:

  app.register_blueprint(user)
- Chạy chương trình web trên file chính
    - Tại file con file.py thì chú ý tại câu lệnh "***redirect(url_for(user.user_name))***"