TePSIT

Costruire un sito web con Flask+SQLAlchemy

Giulio Angiani
I.I.S. "Blaise Pascal" - Reggio Emilia

Web app con Flask+SQLAlchemy

prima parte

Installazione

Python è un linguaggio multipiattaforma

Funzionamento applicazione identico per Windows e Linux

Installazione leggermente differente

Useremo un **virtual environment**

- ambiente virtuale
- non impatta la macchina ospite
- ottima soluzione per lo sviluppo

Installazione virtual environment su Windows

Installazione virtual environment **Windows**

- Installare python3.x (ultima versione 3.9)
- aprire un terminale
- creazione di una directory dove creare il venv
  - md flaskwebapp
- trasformazione della directory in virtual env
  - python -m venv flaskwebapp
- entrare nella directory
  - cd flaskwebapp
- attivare il venv
  - Scripts\Activate

Installazione virtual environment su Linux

Passi di installazione su **Linux**

- Installare python3.x (ma su Linux è presente di default)
- aprire un terminale
- creazione di una directory dove creare il venv
  - mkdir flaskwebapp
- trasformazione della directory in virtual env
  - python -m venv flaskwebapp
- entrare nella directory
  - cd flaskwebapp
- attivare il venv
  - . bin/activate

Installazione

Dalla shell del terminale digitiamo

- pip install flask

per installare il framework **Flask**

poi

- pip install sqlalchemy

per installare l'ORM **SQLAlchemy**

Accendiamo il server

L'applicazione minima per far partire il server web

e rispondere ad una chiamata http ha il seguente codice :

PYTHON

```python
# file 00_server.py
from flask import Flask     # importa le librerie del framework

app = Flask(__name__)       # crea la web app

@app.route('/')             # route di reindirizzamento alla metodo di risposta
def home():                 # metodo di risposta (deve restituire una stringa)
    return "Hi, guys!"      # restituisce il risultato (content-type di default è text/html)

if __name__ == '__main__':
    app.secret_key = "mysecretkey"
    app.run(
        host="0.0.0.0",     # localhost
        port=9000,          # porta
    )
```

Lanciamo il server e contattiamolo via browser

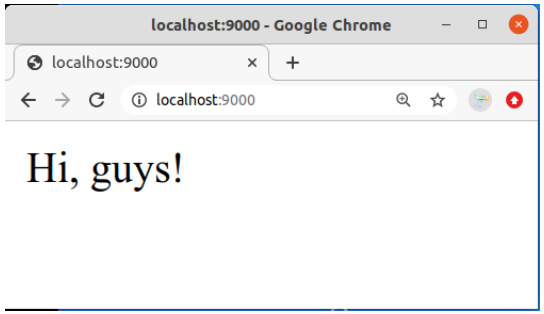
dalla shell dell'ambiente virtuale lanciamo il server

- 00_server.py è il nome del file

SHELL

```
(flaskwebapp) $ python 00_server.py
 * Serving Flask app "00_server" (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://0.0.0.0:9000/ (Press CTRL+C to quit)
```

Apriamo un brower all'indirizzo **http://localhost:9000**



Routes

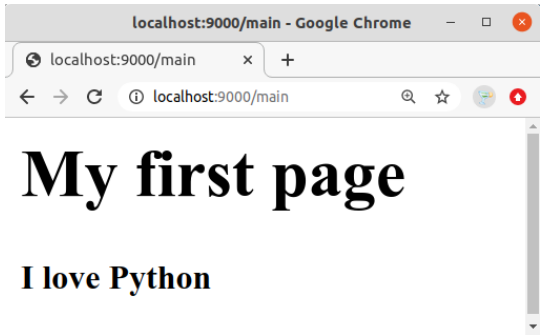
Aggiungiamo una route all'applicazione

- Creazione altro metodo
- aggancio della route tramite il decoratore **@app.route**

PYTHON

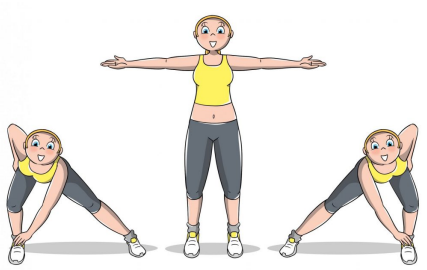
```python
@app.route('/main')
def main():
    content = "I love Python"
    title = "My first page"
    return "<html><body><h1>{title}</h1><h4>{content}</h4></body></html>".format(**vars())
```

Riavviando il server e puntando all'indirizzo **http://localhost:9000/main**



Esercizi - 1

- Creare un metodo **lista()** dell'applicazione associato alla route "/lista"
- A partire dalla lista:



```
articoli = [(1, 'Mouse', 9.99), (2, 'Tastiera', 19.99), (3, 'Tablet', 299.99)]
```

- Restituire una pagina HTML con la lista degli articoli presenti

Si ricorda il ciclo for in python

PYTHON

```
articoli = [(1, 'Mouse', 9.99), (2, 'Tastiera', 19.99), (3, 'Tablet', 299.99)]
for a in articoli:
    print(a[1], a[2])
```

OUTPUT

```
('Mouse', 9.99)
('Tastiera', 19.99)
('Tablet', 299.99)
```

(5 minuti)

Soluzione - 1

PYTHON

```python
@app.route('/lista')
def lista():
    content = "Lista degli articoli presenti"
    title = "Lista articoli"

    lista_articoli = "<ul>"
    articoli = [(1, 'Mouse', 9.99), (2, 'Tastiera', 19.99), (3, 'Tablet', 299.99)]
    for a in articoli:
        lista_articoli += "<li>{} {}".format(a[1], a[2])
    lista_articoli += "</ul>"

    return """<html><body>
                        <h1>{title}</h1>
                        <h4>{content}</h4>
                        {lista_articoli}
                    </body></html>
           """.format(**vars())
```

Templates (Jinja)

Cos'è un template?

- pagina (o parte di pagina) già preimpostata riutilizzabile
- sono presenti dei **placeholder** che vengono sostituiti runtime
- **può contenere logica applicativa**
- può contenere anche dei **blocchi** con un nome identificativo

JINJA TEMPLATE

```
<html>
    <head>
        <title>
            {{ title }}
        </title>
    </head>
    <body>
        <h3>{{ content }}</h3>
        {% block maincontent %}

        {% endblock  %}
        </body>
</html>
```

Ref: https://en.wikipedia.org/wiki/Jinja_(template_engine)

Usare i templates in Flask

In **Flask** esiste la funzione **render_template** che va importata

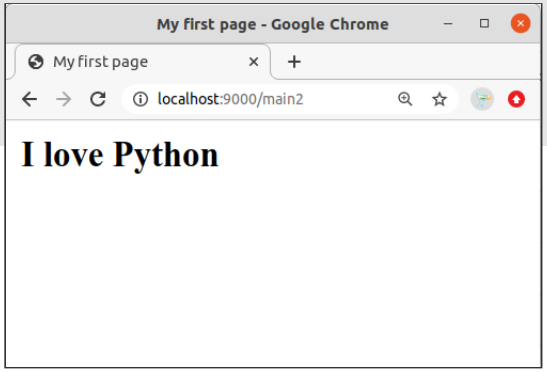

```
from flask import Flask, render_template
```

- in ingresso il file template
- i valori da sostituire
- i placeholder e i blocchi NON passati vengono ignorati

NOTA: i file .html di template **devono** essere posizionati nella directory **templates**

PYTHON

```
@app.route('/main2')
def main2():
    content = "I love Python"
    title = "My first page"
    return render_template("home.html", **vars())
```



**

Usare i templates in Flask

Cambiamo la funzione per mostrare la lista di articoli usando un template


```python
@app.route('/lista2')
def lista2():
    content = "Lista degli articoli presenti"
    title = "Lista articoli"
    articoli = [(1, 'Mouse', 9.99), (2, 'Tastiera', 19.99), (3, 'Tablet', 299.99)]
    return render_template("articoli.html", **vars())
```

nel template inseriamo la logica applicativa del ciclo su **articoli**

JINJA

```
    <table border=1>
        <tr>
            <th>ID</th>
            <th>Descrizione</th>
            <th>Prezzo</th>
        </tr>
    {% for a in articoli %}
        <tr>
            <td>{{ a[0] }}</td>
            <td>{{ a[1] }}</td>
            <td>{{ a[2] }}</td>
        </tr>
    {% endfor %}
    </table>
```

Usare i templates in Flask

Se avessimo una lista di dizionari...


```
@app.route('/elements')
def elements():
    content = "I love Python and Jinja!"
    title = "My first page"
    elements = [
        {
            "id": 1,
            "description": "Mouse",
            "prezzo": 9.99
        },
        {
            "id": 2,
            "description": "Tastiera",
            "prezzo": 19.99
        },
        {
            "id": 3,
            "description": "Tablet",
            "prezzo": 299.99
        }
    ]
    return render_template("elements.html", **vars())
```

**

title: Usare i templates in Flask figure: images/webappflask/flaskjinjia.png

...potremmo usare la dot-notation

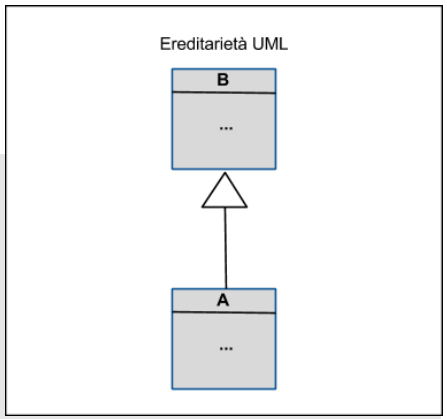
JINJA

```
<table border=1>
    <tr>
        <th>ID</th>
```

Ereditarietà dei templates

Concetto di **estensione** dei templates

Modifico solo il **blocco** che è cambiato rispetto al template padre




```
{% extends "home.html" %}
{% block maincontent %}
    <table border=1>
        <tr>
            <th>ID</th>
            <th>Descrizione</th>
            <th>Prezzo</th>
        </tr>
    {% for e in elements %}
        <tr>
            <td>{{ e.id }}</td>
            <td>{{ e.description }}</td>
            <td>{{ e.prezzo }}</td>
        </tr>

    {% endfor %}
    </table>
{% endblock  %}
```

Inclusione dei templates

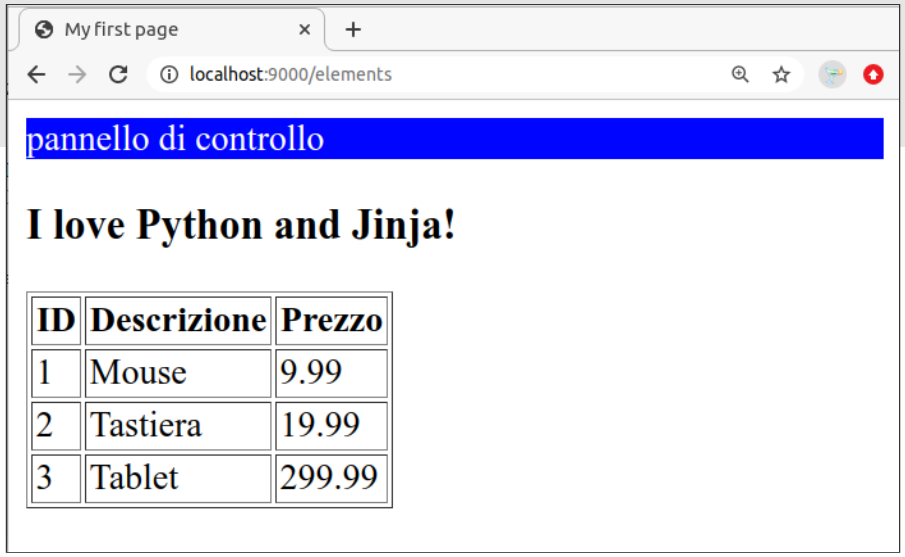
Concetto di **inclusione** dei templates

JINJA (HEADER.HTML)

```
<div style='background-color: blue; color: white; width:100%; margin:0'>
    {{ PANEL }}
</div>
```

JINJA

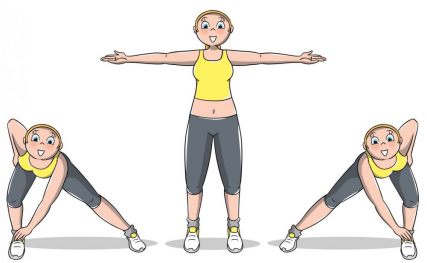
```
{% extends "home.html" %}
{% block maincontent %}
{% include "header.html" %}
    <table border=1>
        [...]
    </table>
{% endblock  %}
```



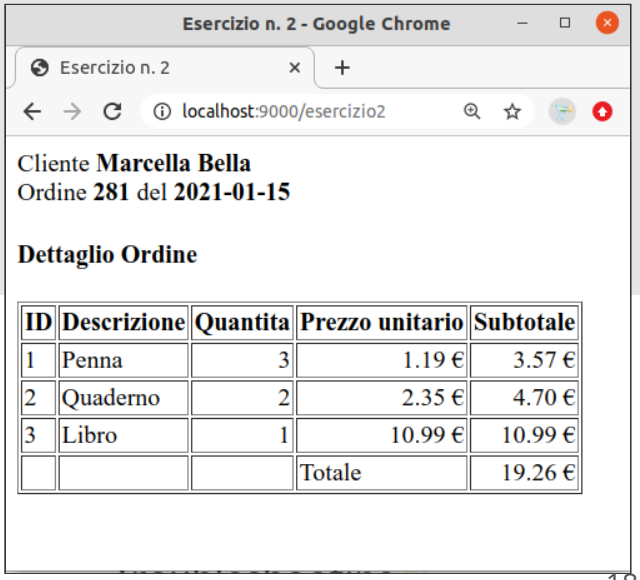
**

Esercizi - 2

Partendo dal seguente snippet python ottenere la pagina in basso a destra progettando il template **ordine.html**



```python
@app.route('/esercizio2')
def elements():
    title = "Esercizio n. 2"
    dati_cliente = { "nome": "Marcella",
                     "cognome": "Bella" }
    dati_ordine = {  "id": 281,
                     "data": "2021-01-15"}
    righe_ordine = [
        {   "id": 1,
            "descrizione": "Penna",
            "qta": 3,
            "prezzo": 1.19 },
        {   "id": 2,
            "descrizione": "Quaderno",
            "qta": 2,
            "prezzo": 2.35 },
        {   "id": 3,
            "descrizione": "Libro",
            "qta": 1,
            "prezzo": 10.99 },
    ]
    totale = sum(elem["prezzo"]*elem["qta"] for elem in righe_ordine)
    return render_template("ordine.html", **vars())
```



Soluzione - 2

JINJA

```
{% extends "home.html" %}
{% include 'header.html' %}
{% block maincontent %}
    <div>Cliente <b>{{ dati_cliente.nome }} {{dati_cliente.cognome }}</b></div>
    <div>Ordine <b>{{ dati_ordine.id }}</b> del <b>{{dati_ordine.data }}</b></div>
    <h4>Dettaglio Ordine</h4>
    <table border=1>
        <tr>
            <th>ID</th>
            <th>Descrizione</th>
            <th>Quantita</th>
            <th>Prezzo unitario</th>
            <th>Subtotale</th>
        </tr>
    {% for r in righe_ordine %}
        <tr>
            <td>{{ r.id }}</td>
            <td>{{ r.descrizione }}</td>
            <td align=right>{{ r.qta }}</td>
            <td align=right>{{ "%.2f"|format(r.prezzo) }} €</td>
            <td align=right>{{ "%.2f"|format(r.prezzo*r.qta) }} €</td>
        </tr>
    {% endfor %}
        <tr><td></td><td></td><td></td><td>Totale</td><td align=right>{{ totale|round(2) }} €</td></tr>
    </table>
{% endblock  %}
```

Giulio Angiani
I.I.S. "Blaise Pascal" - Reggio Emilia