



Big Data e Business Intelligence



Web Frameworks

Giulio Angiani - UniPr

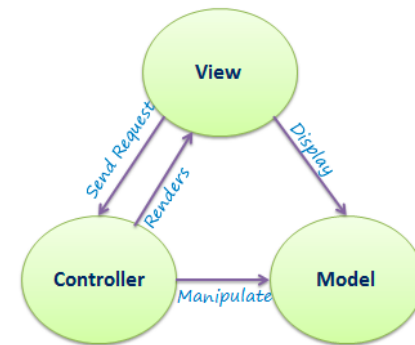


Framework MVC

Pattern MVC

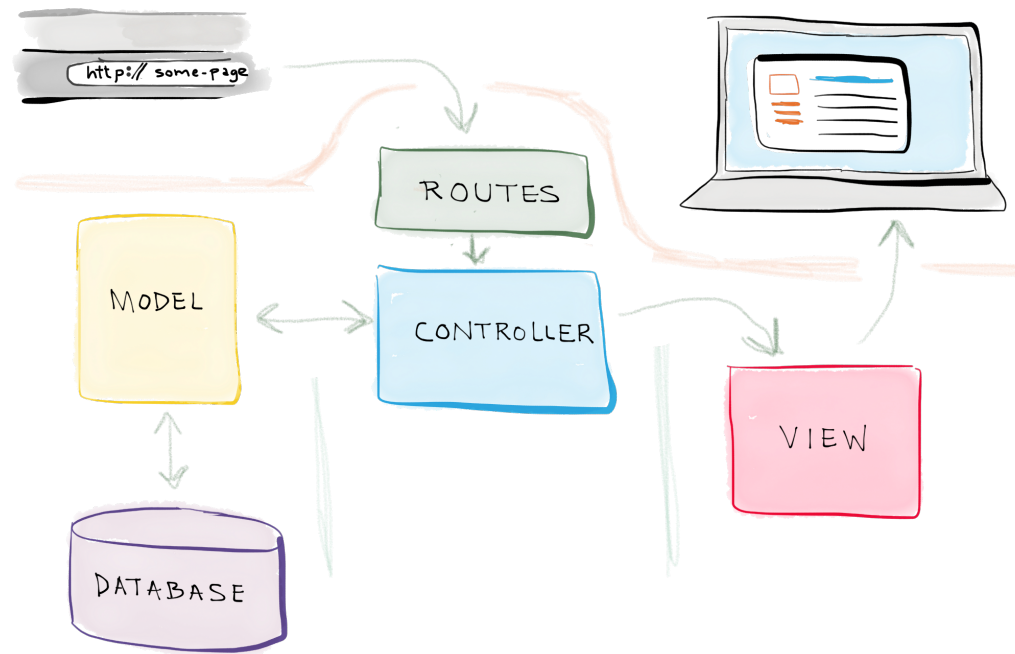
Model-view-controller

- Model: Accesso ai dati.
- View: Visualizzare i dati all'utente - Interazione utente-infrastruttura.
- Controller:
 - gestisce comandi utente tramite View
 - esegue operazioni su Model => cambio stato View.



Pattern MVC

Concetto di Route



Perché MVC

- Disaccoppiare logica applicativa da quella dei dati
- Riusabilità del codice
- Facilità di manutenzione

Regole standard pattern MVC

- Componenti Model, View e Controller codificate su file distinti
- Diverse *view* dello stesso model su file distinti
- La parte View può racchiudere (spesso) anche pseudocodice **per la sola presentazione**

MVC in python

- Django
- Flask
- Pyramid
- CherryPy
- Pylons



ref: <https://hackr.io/blog/python-frameworks>

6/23

MVC in python - Flask

Perché Flask

- Leggero
- Semplice struttura
- Completo

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

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



PYTHON

Flask - Routes

Gestire le route

```
@app.route('/saluta')  
def bye():  
    return "Ciao ragazze e ragazzi!"
```



Flask - Templates

Templates

- creare una subdirectory **templates** nel progetto
- creare un file *base.html* in templates

```
from flask import render_template
```

```
@app.route('/home')
```

```
def home():
```

```
    return render_template("base.html")
```



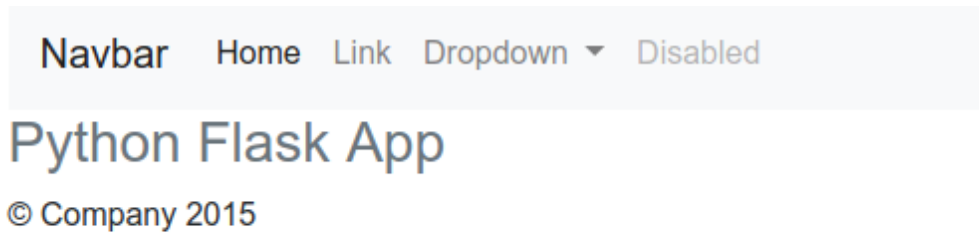
PYTHON

Flask - Templates

file *base.html*

```
<title>Python Flask App</title>
....
<div>
    <h3 class="text-muted">Python Flask App</h3>
</div>
...
```

HTML



10/23

Flask - Templates

modifico file *base.html*

```
<title>Python Flask App</title>
....
<div>
    <h3 class="text-muted">Python Flask App</h3>
</div>
...
<div>
    <h2>{{ content }}</h2>
</div>
```

HTML

11/23

Flask - Templates

cicli in template *base.html*

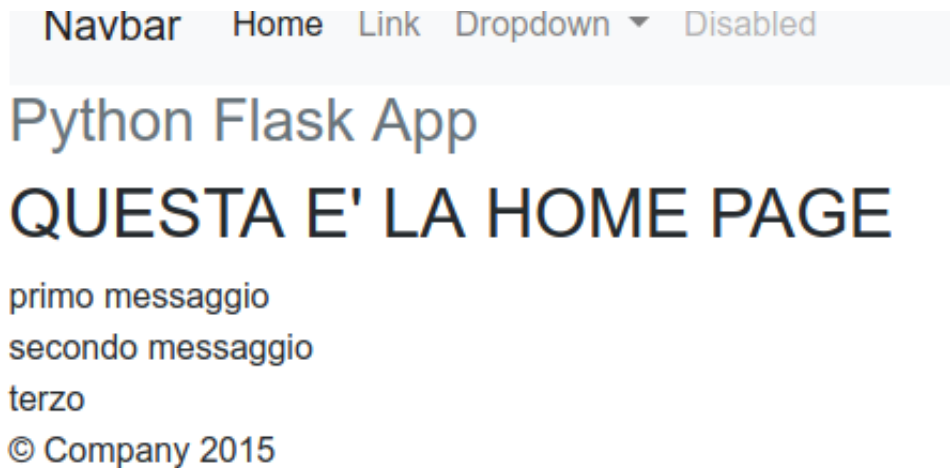
```
<title>Python Flask App</title>
....
<div>
    {% for msg in msgs %}
        <div class="flash">{{ msg }}</div>
    {% endfor %}
</div>
```

HTML

Flask - Templates

PYTHON

```
from flask import render_template
@app.route('/home2')
def home2():
    content = "QUESTA E' LA HOME PAGE"
    messages = [ "primo messaggio",
                  "secondo messaggio",
                  "terzo"]
    return render_template("base.html", content = content, msgs=messages)
```

A screenshot of a web application interface. At the top, there is a light gray navigation bar containing the text 'Navbar', 'Home', 'Link', 'Dropdown' with a downward arrow, and 'Disabled'. Below the navigation bar, the text 'Python Flask App' is displayed in a large, blue, sans-serif font. Underneath that, the text 'QUESTA E' LA HOME PAGE' is shown in a large, bold, black, sans-serif font. Further down, the words 'primo messaggio', 'secondo messaggio', and 'terzo' are listed vertically in a smaller, black, sans-serif font. At the bottom of the visible content, the text '© Company 2015' is displayed in a small, black, sans-serif font.

Navbar Home Link Dropdown ▾ Disabled

Python Flask App

QUESTA E' LA HOME PAGE

primo messaggio
secondo messaggio
terzo

© Company 2015

13/23

Flask - Templates inheritance

- si importa la pagina da ereditare all'inizio del nuovo template
- in base.html aggiungo un blocco...

```
<div>  
    {% block main %}{% endblock %}  
</div>
```

JINJA

- nuovo template : ext1.html

```
{% extends 'base.html' %}  
  
{% block title %}Altro titolo{% endblock %}  
  
{% block main %}  
    Ciao {{ d["nome"] }} {{ d.cognome }}  
{% endblock %}
```

JINJA

Flask - Templates inheritance

- nel controller lo chiamo con un altro nome

```
@app.route('/home3')
def home3():
    content = "Terza pagina"
    var_dict = {
        "nome": "Giulio",
        "cognome": "Angiani"
    }
    return render_template("ext1.html", content = content, d = var_dict)
```

PYTHON

Flask - Connessione al DB

```
import sqlite3  
from flask import g
```

```
DATABASE = 'database.db'
```

PYTHON

Flask - Connessione al DB

```
@app.route('/list')
def lista():
    items = []
    for i in query_db('select * from items'):
        items.append(dict(zip(["code", "description", "price"], i)))

    return render_template("lista.html", items = items)
```

PYTHON

```
[{'code': 'APP001', 'price': 1025.99, 'description': 'iPhone X'},
{'code': 'SAM003', 'price': 899.99, 'description': 'Samsung 10'}]
```

OUTPUT

Rendering su template

- template lista.html

```
{% extends 'base.html' %}
```

JINJA

```
{% block title %}Lista Items{% endblock %}
```

```
{% block main %}
```

```
    {% for i in items %}
```

```
        <div class="flash">{{ i.code }} {{ i.description }} {{ i.price }}</div>
```

```
    {% endfor %}
```

```
{% endblock %}
```

Flask - Model

```
from flask_sqlalchemy import SQLAlchemy
DATABASE = 'database.db'
project_dir = os.path.dirname(os.path.abspath(__file__))
database_file = "sqlite:///{}".format(os.path.join(project_dir, DATABASE))
app.config["SQLALCHEMY_DATABASE_URI"] = database_file
db = SQLAlchemy(app)
```

PYTHON

Flask - Model

- Classe Item (mappa una tabella **items**)

```
class Item(db.Model):  
    code = db.Column(db.String(10), unique=True, nullable=False, primary_key=True)  
    description = db.Column(db.String(80), unique=True, nullable=False, primary_key=False)  
    price = db.Column(db.Float(), unique=True, nullable=False, primary_key=False)  
  
    def __init__(self, code=None, description=None, price=0):  
        self.code = code  
        self.description = description  
        self.price = price  
  
    def __repr__(self):  
        return "<Item:"
```

PYTHON

Flask - Creazione DB

- per creare il DB

```
@app.route('/makedb')
def makedb():
    db.create_all()
    i = Item(code="APP001", description="Apple iPhoneX", price=1200.00)
    db.session.add(i)
    i = Item(code="SAM201", description="Samsung 10", price=95.00)
    db.session.add(i)
    i = Item(code="LG0002", description="LG Q60", price=249.9)
    db.session.add(i)
    db.session.commit()
    return "DB CREATO"
```

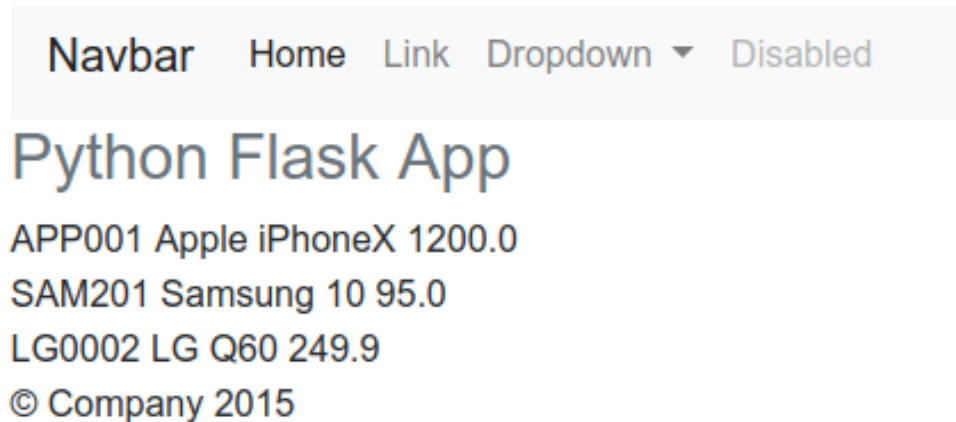
PYTHON

Flask - Query su DB

- Query senza SQL

```
@app.route('/list')
def lista():
    items = Item.query.all()
    return render_template("lista.html", items = items)
```

PYTHON





Giulio Angiani
Universita' degli Studi di Parma