Python Biella Group

Flask Course

Docente

Andrea Guzzo

https://andreaguzzo.com

Anno 2020

Agenda

Incontri e lezioni

01 Introduzione a Flask e Jinja2 base

02 Jinja avanzato, Bootstrap, Forms

03 Flask con Database

04 Review con Andrea

05 Review con Mario

06 Grandi applicazioni con Flask

07 REST Backend e concetti avanzati

Python Biella Group

JOIN US!

- GitHub: *https://github.com/PythonGroupBiella*
- Telegram: *https://t.me/joinchat/AAAAAFGSWcxhSln_SRhseQ*

Tutto questo è stato reso possibile grazie a:

- Tutta la community di P.B.G.
- Maria Teresa Panunzio: https://www.linkedin.com/in/maria-teresa-panunzio-27ba3815/
- Mario Nardi: https://www.linkedin.com/in/mario-nardi-017705100/

Obiettivo del corso

Realizzare una piccola applicazione

- Concetti base di Flask per usarlo in differenti contesti
- Costruire piccole applicazioni web o backend
- Riuscire a "mettere in produzione" il proprio codice

Altri Framework:
Django, FastAPI, Pyramid, Bottle, ...

Cos'è Flask?

Micro-web framework

- Core semplice altamente estendibile e modulare
- Utilizzato per creare siti web e API
- Server side
- Leggero e performante
- Dipendenze:
  - Werkzeug: routing, debugger, WSGI support
  - Jinja2: templating

Features

- Request dispatcher
- Template engine
- Secure cookies
- User sessions
- Unit testing
- In-browser debugger e reloader

Moduli

- Administration
- Email
- Databases
- Caching
- User auth
- ...

Prima di iniziare...

Visual Studio Code: Estensioni

- Python
- GitLens
- GitHistory
- Jinja
- Jinja2 Snippets
- Kite Autocomplete for Python (?)
- Python Docstring
- Python Indent
- vscode-icons
- Visual Studio IntelliCode

Getting Started

- Python: 3.7
- *Repository Github:*

  https://github.com/PythonGroupBiella/MaterialeLezioni

```
#Install Python virtualenv
> pip install virtualenv

#Linux, MAC
> virtualenv venv
> source venv/bin/activate
> pip install -r requirements.txt

#Windows
> virtualenv venv
> venv\scripts\activate
> pip install -r requirements.txt
```

Nella precedente lezione...

Abbiamo visto:

- Impostazione dell'ambiente e strumenti
- Creazione di una semplice app
- Jinja Base
- Esempio di progetto

Per recuperare la lezione precedente:

- Video Youtube: https://www.youtube.com/watch?v=FPI5-oGKiVI
- *Repository Github:* https://github.com/PythonGroupBiella/MaterialeLezioni

Condividete le vostre soluzioni!

Python Biella Group

Lezione 2

Oggi vedremo...

- Costruzione di Forms
- Jinja Avanzato
- Bootstrap

Librerie e requisiti:

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni

Nuove librerie

Flask Form

WTForms

Condividete le vostre soluzioni e fate gli esercizi

HTTP Methods

GET

Leggi (o recupera) una rappresentazione di una sorgente (pagina)

POST

Inserisci o crea una nuova risorsa

PUT (PATCH)

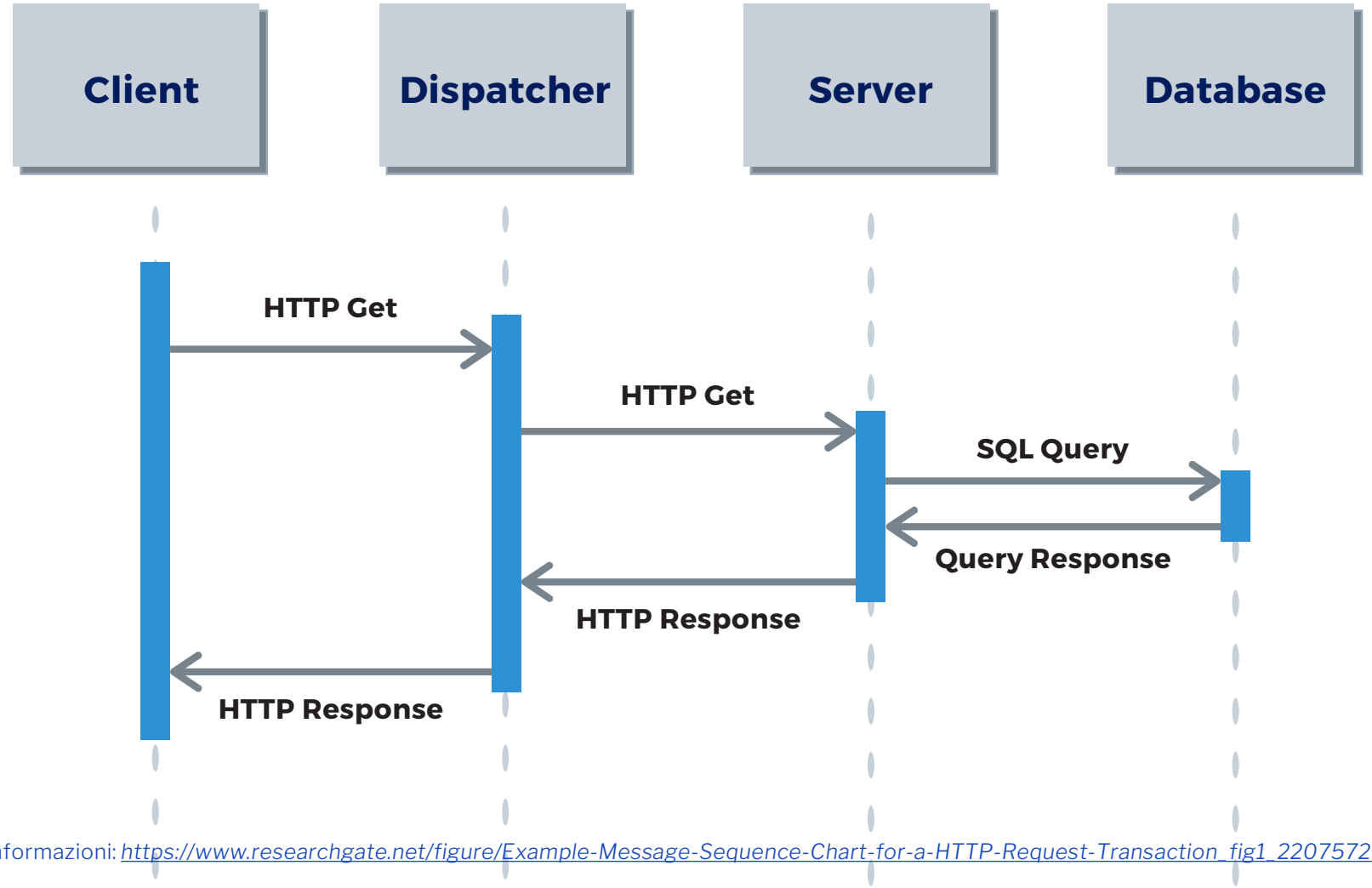
Modifica la rappresentazione corrente con un nuovo aggiornamento (eventualmente crea da zero)

DELETE

Elimina una risorsa esistente

Informazioni: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Sequenza dei messaggi



Informazioni: *https://www.researchgate.net/figure/Example-Message-Sequence-Chart-for-a-HTTP-Request-Transaction_fig1_220757202*

Bootstrap

Front end framework

Componenti

Grafica migliorata

Componenti animati

Grid System

Mobile Ready

Temi

References: *https://getbootstrap.com/*

Flask Bootstrap: *https://pythonhosted.org/Flask-Bootstrap/*

Bootstrap Italia: *https://italia.github.io/bootstrap-italia/*

Bootstrap Themes: *https://bootswatch.com/*

Lezione 3

Oggi vedremo...

- Database con Flask
- Migrazioni
- CRUD

Librerie e requisiti:

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni

Nuove librerie

Flask SQLAlchemy

Flask Migrate

Condividete le vostre soluzioni e fate gli esercizi

SQL Light con Flask

Basic applications (100,000 hits /day)

ORM - Object Relational Mapper

CRUD Operations: Create, Read, Update, Delete

Migrate - sincronizzazione automatica

```
#MacOS/Linux
export FLASK_APP=myapp.py

#Windows
set FLASK_APP=myapp.py
```

Python

Ci sono 4 comandi principali che si possono usare da command line:

```
flask db init #set up the migrations directory
flask db migrate -m "some message" #set up the migration file (is always usefull ins
flask db upgrade #update the database with the migration
```

Python

SQL Alchemy: https://docs.sqlalchemy.org/en/13/

Flask SQLAlchemy: https://flask-sqlalchemy.palletsprojects.com/en/2.x/

Flask Migrate: https://flask-migrate.readthedocs.io/en/latest/

Python Biella Group

Lezione 6

Oggi vedremo...

- Blueprints - riorganizzazione per “Large applications”
- Gestire configurazioni
- Unittest
- Idee per le prossime evoluzioni

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni/tree/master/Flask/Lezione6

Nuove librerie

Unittest

Condividete le vostre soluzioni e fate gli esercizi

Libro di riferimento

Flask Web Development

Progetto di esempio

Flasky

https://github.com/miguelgrinberg/flasky

Riorganizzare per “Large applications”

Blueprints

In Flask, a blueprint is a method of extending an existing Flask app.

They provide a way of combining groups of views with common functionality and allow developers to break their app down into different components.

A blueprint is similar to an application in that it can also define routes and error handlers.

The difference is that when these are defined in a blueprint they are in a dormant state until the blueprint is registered with an application, at which point they become part of it.

Using a blueprint defined in the global scope, the routes and error handlers of the application can be defined in almost the same way as in the single-script application.

Like applications, blueprints can be defined all in a single file or can be created in a more structured way with multiple modules inside a package.

To allow for the greatest flexibility, a subpackage inside the application package can be created.

Nel libro: Cap. 7 - Large Application Structure

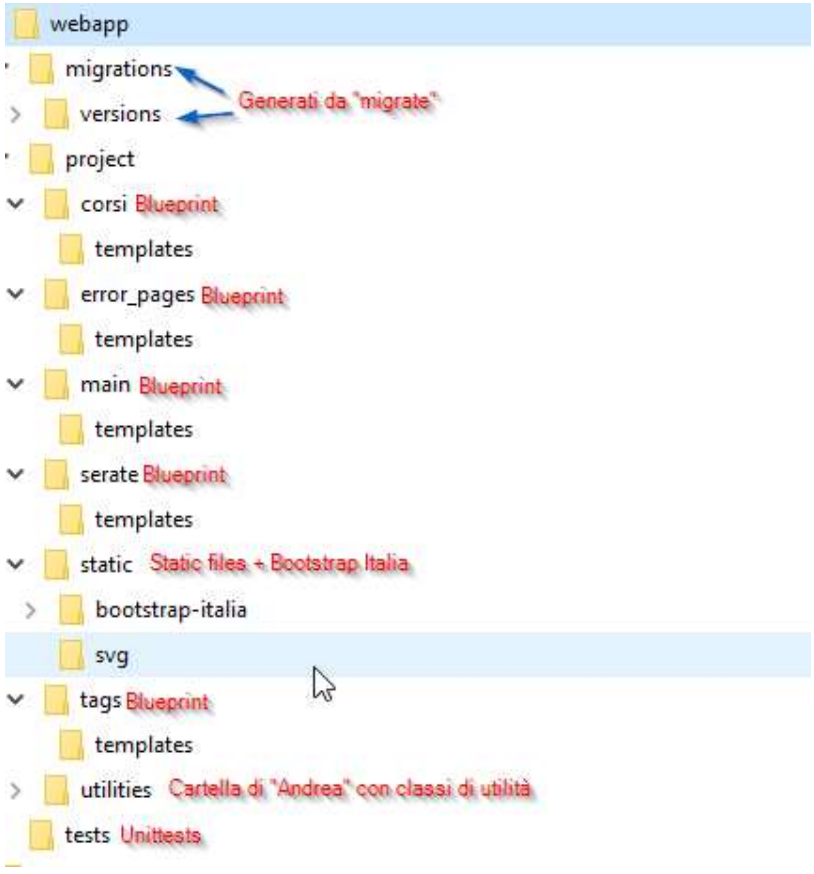
Riorganizzare per “Large applications”

Revisione della struttura

Non ci sono vincoli (al contrario di altri framework come Django)

Quindi non ci sono standard comuni

Nel ns progetto (vedi immagine)



Nel libro: Cap. 7 - Large Application Structure

Python Biella Group

Gestire configurazioni

Config file

create_app()

Application factory –

- Design pattern “creazionale”

- In software engineering, a design pattern is a general repeatable solution to a commonly occuring problem in software design
- It isn't a finished design that can be transformed directly into code, but a **description** or **template** for how to solve a problem that can be used in many different situations

- Design patterns:
  - Provide general solutions, documented in a format that doesn't require specifics tied to a particular problem
  - Can speed up the development process by providing tested, proven development paradigms
  - Help you benefit from the experience of fellow developers
  - Prevent subtle issues that can cause major problems
  - Improve code readability for coders and architects familiar with them

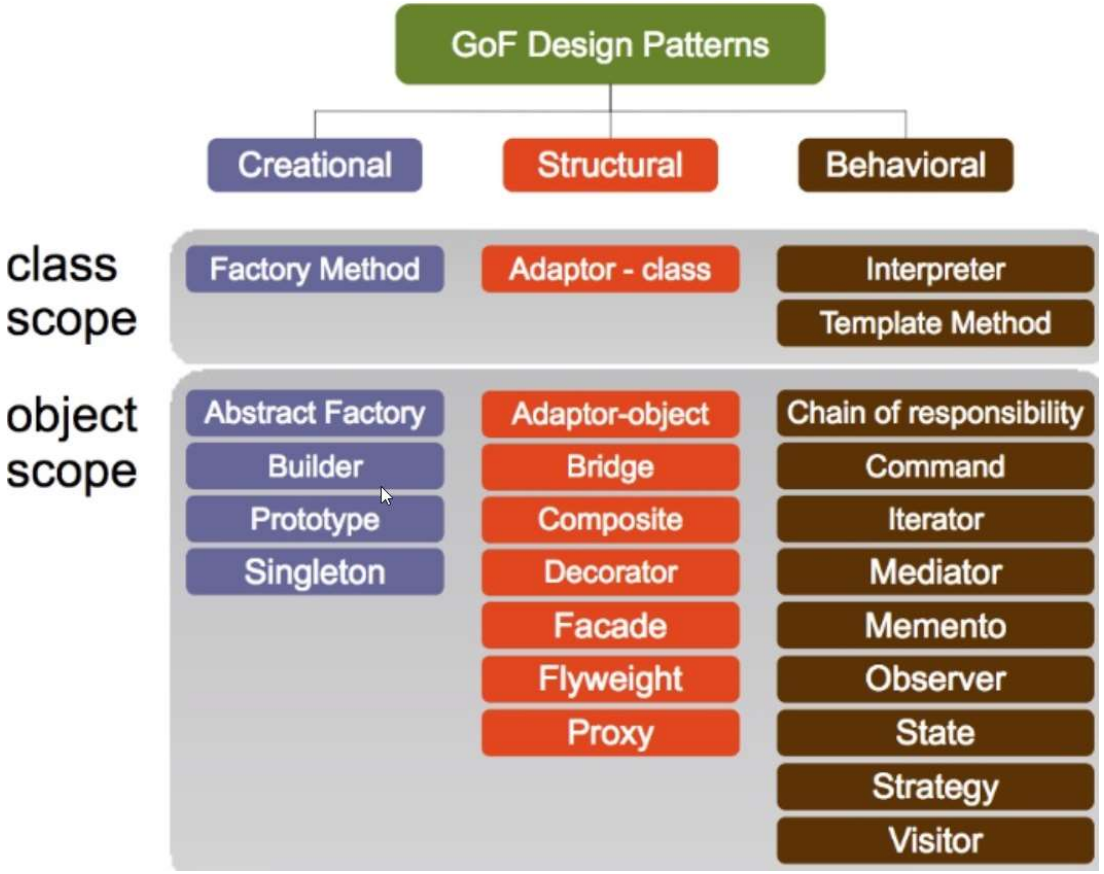
Nel libro: Cap. 7 - Large Application Structure

Design Patterns

GoF Design Patterns

Design Patterns: Definition and Utility

- The **Gang of Four** are the four authors of the book « Design Patterns: Elements of Reusable Object-Oriented Software »
- Defined 23 design patterns for recurrent design issues, called GoF design patterns
- Classified by *purpose*:
  - Structural : Concerns the **composition** of classes and objects
  - Behavioral : Characterizes the **interaction and responsibility** of objects and classes
  - Creational : Concerns the **creation process** of objects and classes
- ... and by *scope:*
  - Class scope: relationship between classes and subclasses, defined statically
  - Object scope: object relationships, dynamic



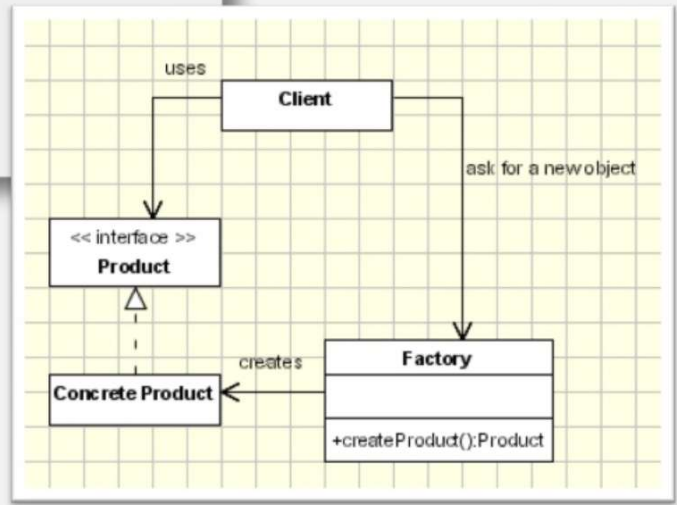
Factory Pattern

Factory

Creational Patterns

- Creates objects without exposing the instantiation logic to the client
- Refers to the newly created object through a common interface.

```
public class ProductFactory{
        public Product createProduct(String ProductID){
                if (id==ID1)
                        return new OneProduct();
                if (id==ID2) return
                        return new AnotherProduct();
                ... // so on for the other Ids

        return null;
    }
    ...
}
```



Factory Pattern - Esempio

Introduction

As per Wikipedia:

"The factory pattern is a creational design pattern used in software development to encapsulate the processes involved in the creation of objects."

Factory pattern involves creating a super class which provides an abstract interface to create objects of a particular type, but instead of taking a decision on which objects get created it defers this creation decision to its subclasses. To support this there is a creation class hierarchy for the objects which the factory class attempts create and return.

Factory pattern is used in cases when based on a "type" got as an input at run-time, the corresponding object has to be created. In such situations, implementing code based on Factory pattern can result in scalable and maintainable code i.e. to add a new type, one need not modify existing classes; it involves just addition of new subclasses that correspond to this new type.

In short, use Factory pattern when:

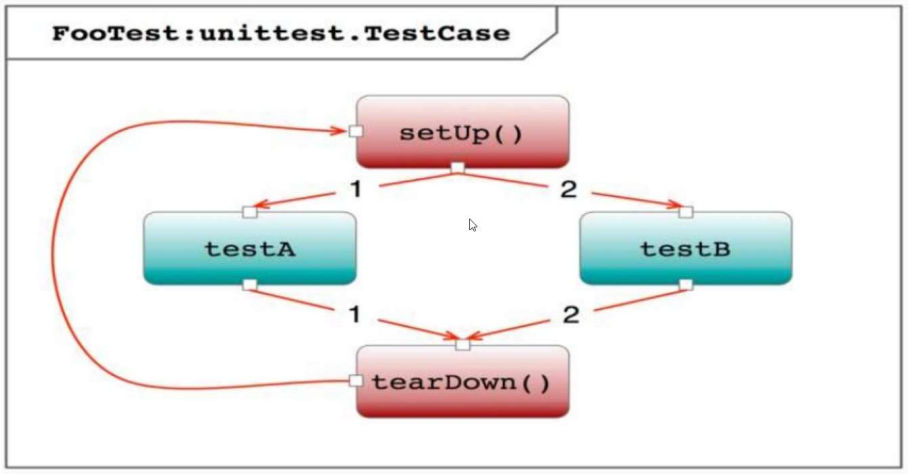
- A class does not know what kind of object it must create on a user's request
- You want to build an extensible association between this creater class and classes corresponding to objects that it is supposed to create.

```
class Person:
        def __init__(self):
                self.name = None
                self.gender = None

        def getName(self):
                return self.name

        def getGender(self):
                return self.gender

class Male(Person):
        def __init__(self, name):
                print "Hello Mr." + name

class Female(Person):
        def __init__(self, name):
                print "Hello Miss." + name

class Factory:
        def getPerson(self, name, gender):
                if gender == 'M':
                        return Male(name)
                if gender == 'F':
                        return Female(name)
```

Unit Test / 1

Some Important Points

- Every test class must be sub class of **`unittest.TestCase`**
- Every test function should start with **test** name.
- to check for an expected result use **assert** functions.
- The **setUp()** method define instructions that will be executed before test case.
- The **tearDown()** method define instructions that will be executed after test case.
- Run Test with **python -m unittest -v test_module**
- Only test single part of code



Unit Test / 2

Assert functions

- assertEqual(a, b)
- assertNotEqual(a, b)
- assertTrue(x)
- assertFalse(x)
- assertIs(a, b)
- https://docs.python.org/2/library/unittest.html#test-cases

Unit Test / 3

Integrazione con Flask

Decorator in app.py -> flask test

Package test

Nel libro: Cap. 7 - Large Application Structure

Prossime evoluzioni dell'app

Autenticazione e modulo login (nuovo blueprint)

Utenti non autenticati:

- Prossime serate
- Lista corsi
- Blog in lettura

Utenti autenticati

- Blog in scrittura

Utenti amministratori

- Gestione tag, serate, corsi

Gestione blog (nuovo blueprint)

Python Biella Group

Lezione 7

Oggi vedremo...

- Autenticazione
  - Blueprint Utenti
  - Blueprint Ruoli
  - Gestione di tutti gli step
    - Hash password
    - Flask-Login
  - Invio mail
    - Token conferma

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni/tree/master/Flask/Lezione7

Nuove librerie

• Flask-Login: Management of user sessions for logged-in users (pip install –U flask-login)

• Werkzeug: Password hashing and verification (pip install -U Werkzeug)

• itsdangerous: Cryptographically secure token generation and verification (pip install -U itsdangerous)

• flask-email: Creazione e invio email (pip install -U flask-email)

Cap. 8 – 9 Libro “Flask Web Development”

Nuovi blueprints

Utenti (cap. 8 del libro) e Ruoli (cap. 9)

- Il modello di sicurezza varia da applicazione ad applicazione in base al contesto
- Quasi sempre è necessario almeno il profilo ADMIN

Auth (blueprint nel libro)

Hash della password

Sicurezza della password

Memorizzare l'hash e non la password

A password hashing function takes a password as input, adds a random component to it (the salt), and then applies several one-way cryptographic transformations to it

https://it.wikipedia.org/wiki/Salt_(crittografia)

Password hashes can be verified in place of the real passwords because hashing functions are repeatable: given the same inputs (the password and the salt), the result is always the same.

Modulo Werkzeug

`generate_password_hash(password, method='pbkdf2:sha256', salt_length=8)`
This function takes a plain-text password and returns the password hash as a string that can be stored in the user database. The default values for `method` and `salt_length` are sufficient for most use cases.

`check_password_hash(hash, password)`
This function takes a password hash previously stored in the database and the password entered by the user. A return value of `True` indicates that the user password is correct.

TestUnit

Autenticazione con Flask-Login

When users log in to the application, their authenticated state has to be recorded in the user session, so that it is remembered as they navigate through different pages.

Flask-Login is a small but extremely useful extension that specializes in managing this particular aspect of a user authentication system, without being tied to a specific authentication mechanism.

Table 8-1. Flask-Login required items

Property/method	Description
`is_authenticated`	Must be `True` if the user has valid login credentials or `False` otherwise.
`is_active`	Must be `True` if the user is allowed to log in or `False` otherwise. A `False` value can be used for disabled accounts.
`is_anonymous`	Must always be `False` for regular users and `True` for a special user object that represents anonymous users.
`get_id()`	Must return a unique identifier for the user, encoded as a Unicode string.

These properties and methods can be implemented directly in the model class, but as an easier alternative Flask-Login provides a **UserMixin** class that has default implementations that are appropriate for most cases

Proteggere le “routes”

login_required decorator

LoginForm, template, route

Esempio di Flask-Bootstrap wtf.quick_form()

Testing logins: if current_user.is_authenticated ?

Logout

Nuove registrazioni

Generazione token di conferma con **itsdangerous**

Invio email

flask_mail

Email templates (text e HTML)

Testare con https://mailtrap.io/

Permessi e Ruoli (nel libro)

Table 9-1. Application permissions

Task name	Permission name	Permission value
Follow users	FOLLOW	1
Comment on posts made by others	COMMENT	2
Write articles	WRITE	4
Moderate comments made by others	MODERATE	8
Administration access	ADMIN	16

Table 9-2. User roles

User role	Permissions	Description
None	None	Read-only access to the application. This applies to unknown users who are not logged in.
User	FOLLOW, COMMENT, WRITE	Basic permissions to write articles and comments and to follow other users. This is the default for new users.
Moderator	FOLLOW, COMMENT, WRITE, MODERATE	Adds permission to moderate comments made by other users.
Administrator	FOLLOW, COMMENT, WRITE, MODERATE, ADMIN	Full access, which includes permission to change the roles of other users.

Verifica dei ruoli

Can

isAdministrator

Permission presente in tutti i templates

```python
@main.app_context_processor
def inject_permissions():
    return dict(Permission=Permission)
```

Test

admin_required decorator

Prossime evoluzioni dell'app

Profili (cap 10)

- Gestione profile
- Avatar Utente

Blog (cap 11)

Python Biella Group

Lezione 8

Oggi vedremo...

- Ruoli, profili e permessi
  - Avatars (gravatar)
- Blog (blueprint)
  - Post e Commenti

Repository Github

https://github.com/PythonGroupBiella/MaterialeLezioni/tree/master/Flask/Lezione8

Nuove librerie

- moment
- markdown / bleach
- pagedown e flask-pagedown
- faker

https://github.com/OmkarPathak/A-Simple-Note-Taking-Web-App

Cap. 10 – 11 – 13 Libro “Flask Web Development”

Python Biella Group

moment

Visualizzazione “da quanto tempo”

- Codice da includere

Example 3-12. templates/base.html: importing the Moment.js library

```
{% block scripts %}
{{ super() }}
{{ moment.include_moment() }}
{% endblock %}
```

- Esempio di utilizzo: utenti/User.html
  - moment(user.member_since).format("DD/MM/YYYY")

    moment(user.last_seen).fromNow()
- Localizzazione

```
{% block scripts %}
{{ super() }}
{{ moment.include_moment() }}
{{ moment.locale('es') }}
{% endblock %}
```

```
from flask import Flask, render_template
from flask_bootstrap import Bootstrap
from flask_mail import Mail
from flask_moment import Moment
from flask_sqlalchemy import SQLAlchemy
from config import config

bootstrap = Bootstrap()
mail = Mail()
moment = Moment()
db = SQLAlchemy()

def create_app(config_name):
    app = Flask(__name__)
    app.config.from_object(config[config_name])
    config[config_name].init_app(app)

    bootstrap.init_app(app)
    mail.init_app(app)
    moment.init_app(app)
    db.init_app(app)

    # attach routes and custom error pages here

    return app
```

Cap. 3 pag. 38 e Cap. 10 pag. 140

Python Biella Group

(gr)avatar

Visualizzazione avatars

- Utilizzo del servizio offerto da https://en.gravatar.com/ che espone l'avatar utente da URL con MD5 hash dell'email (es. https://secure.gravatar.com/avatar/d4c74594d841139328695756648b6bd6)
- Utilizzo: modello Utente

Table 10-1. Gravatar query string arguments

Argument name	Description
s	Image size, in pixels.
r	Image rating. Options are "g", "pg", "r", and "x".
d	The default image generator for users who have no avatars registered with the Gravatar service. Options are "404" to return a 404 error, a URL that points to a default image, or one of the following image generators: "mm", "identicon", "monsterid", "wavatar", "retro", or "blank".
fd	Force the use of default avatars.

```python
# Avatar
def gravatar_hash(self):
    return hashlib.md5(self.email.lower().encode('utf-8')).hexdigest()

def gravatar(self, size=100, default='identicon', rating='g'):
    url = 'https://secure.gravatar.com/avatar'
    hash = self.avatar_hash or self.gravatar_hash()
    return '{url}/{hash}?s={size}&d={default}&r={rating}'.format(
        url=url, hash=hash, size=size, default=default, rating=rating)
```

Cap. 10 pag. 146

Python Biella Group

E’ un linguaggio!

markdown (rich text)/1

Moduli: flask_pagedown / bleach e markdown

- Codice da includere

Example 11-12. app/__init__.py: Flask-PageDown initialization

```
from flask_pagedown import PageDown
# ...
pagedown = PageDown()
# ...
def create_app(config_name):
    # ...
    pagedown.init_app(app)
    # ...
```

```
from flask_pagedown.fields import PageDownField

class PostForm(FlaskForm):
    body = PageDownField("What's on your mind?", validators=[Required()])
    submit = SubmitField('Submit')
```

- Esempio di utilizzo front-end: main/index.html
- Utilizzo back-end (bleach e markdown), pag. 164

Cap. 11 pag. 162

- PageDown, a client-side Markdown-to-HTML converter implemented in JavaScript
- Flask-PageDown, a PageDown wrapper for Flask that integrates PageDown with Flask-WTF forms
- Markdown, a server-side Markdown-to-HTML converter implemented in Python
- Bleach, an HTML sanitizer implemented in Python

markdown (rich text)/2

- FRONT-END
  - Flask-Pagedown

```
{% block scripts %}
{{ super() }}
{{ pagedown.include_pagedown() }}
{% endblock %}
```

    - Permette di usare PageDownField da WTForms
    - La preview “Markdown” è implementata utilizzando il modulo PageDown che va messo nel template
- BACK-END
  - Per ragioni di sicurezza solo il Markdown è mandato con la richiesta POST
    - The HTML code for the rendered blog post is cached in a new field added to the Post model that the template can access directly. The original Markdown source is also kept in the database in case the post needs to be edited

```
from markdown import markdown
import bleach

class Post(db.Model):
    # ...
    body_html = db.Column(db.Text)
    # ...
    @staticmethod
    def on_changed_body(target, value, oldvalue, initiator):
        allowed_tags = ['a', 'abbr', 'acronym', 'b', 'blockquote', 'code',
                        'em', 'i', 'li', 'ol', 'pre', 'strong', 'ul',
                        'h1', 'h2', 'h3', 'p']
        target.body_html = bleach.linkify(bleach.clean(
            markdown(value, output_format='html'),
            tags=allowed_tags, strip=True))

db.event.listen(Post.body, 'set', Post.on_changed_body)
```

Cap. 11 pag. 164

Python Biella Group

Paginazione/1

Grazie a paginate() di SQLAlchemy

- Main/Index.html
  - Paginazione dei post
- Blog/Post.html
  - Paginazione dei commenti

UTILIZZO

- Definizione numero elementi per pagina (config)
- .paginate()
- Classe Pagination

```
@main.route('/', methods=['GET', 'POST'])
def index():
    # ...                                  La pagina da renderizzare è presa da "request.ars" dall'URL; 1 default
    page = request.args.get('page', 1, type=int)
    pagination = Post.query.order_by(Post.timestamp.desc()).paginate(
        page, per_page=current_app.config['FLASKY_POSTS_PER_PAGE'],
        error_out=False)
    posts = pagination.items
    return render_template('index.html', form=form, posts=posts,
                           pagination=pagination)
```

Anzichè .all(), paginate (numero_pagina, numero_elementi_per_pagina) che restituisce un oggetto Pagination

Cap. 11 Libro “Flask Web Development”, da pag. 155

Python Biella Group

Paginazione/2

Classe Pagination

Table 11-1. Flask-SQLAlchemy pagination object attributes

Attribute	Description
`items`	The records in the current page
`query`	The source query that was paginated
`page`	The current page number
`prev_num`	The previous page number
`next_num`	The next page number
`has_next`	`True` if there is a next page
`has_prev`	`True` if there is a previous page
`pages`	The total number of pages for the query
`per_page`	The number of items per page
`total`	The total number of items returned by the query

Table 11-2. Flask-SQLAlchemy pagination object methods

Method	Description
`iter_pages(left_edge=2, left_current=2, right_current=5, right_edge=2)`	An iterator that returns the sequence of page numbers to display in a pagination widget. The list will have `left_edge` pages on the left side, `left_current` pages to the left of the current page, `right_current` pages to the right of the current page, and `right_edge` pages on the right side. For example, for page 50 of 100 this iterator configured with default values will return the following pages: 1, 2, None, 48, 49, 50, 51, 52, 53, 54, 55, None, 99, 100. A None value in the sequence indicates a gap in the sequence of pages.
`prev()`	A pagination object for the previous page.
`next()`	A pagination object for the next page.

Cap. 11 Libro “Flask Web Development”, da pag. 155

Python Biella Group

Paginazione/3

Widget riusabile per la paginazione: _macros.html

```
{% macro pagination_widget(pagination, endpoint, fragment='') %}
<ul class="pagination">
    <li {% if not pagination.has_prev %} class="disabled"{% endif %}>
        <a href="{% if pagination.has_prev %}{{ url_for(endpoint, page=pagination.prev_num, **kwargs)
            &laquo;
        </a>
    </li>
    {% for p in pagination.iter_pages() %}
        {% if p %}
            {% if p == pagination.page %}
            <li class="active">
                <a href="{{ url_for(endpoint, page = p, **kwargs) }}{{ fragment }}">{{ p }}</a>
            </li>
            {% else %}
            <li>
                <a href="{{ url_for(endpoint, page = p, **kwargs) }}{{ fragment }}">{{ p }}</a>
            </li>
            {% endif %}
        {% else %}
        <li class="disabled"><a href="#">&hellip;</a></li>
        {% endif %}
    {% endfor %}
    <li {% if not pagination.has_next %} class="disabled"{% endif %}>
        <a href="{% if pagination.has_next %}{{ url_for(endpoint, page=pagination.next_num, **kwargs)
            &raquo;
        </a>
    </li>
</ul>
{% endmacro %}
```

Cap. 11 Libro “Flask Web Development”, da pag. 159

Python Biella Group

faker

Articoli “fittizi” per verificare la paginazione

- Utilizzo in fake_data.py per creare utenti e post fittizi

Cap. 11 Libro “Flask Web Development”

Prossime evoluzioni dell'app

"No" Followers (cap 12)

Deploy su Heroku (cap 17)

API(cap 14)

INIZIAMO!

- GitHub: https://github.com/PythonGroupBiella/MaterialeLezioni