Python Biella Group

Flask Course

Docenti

Andrea Guzzo

https://andreaguzzo.com

Mario Nardi

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!

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

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

#Windows
set FLASK_APP=myapp.py
```

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

```python
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
```

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

Python Biella Group

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

Python Biella Group

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

Python Biella Group

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

```
@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 e bleach
- pagedown e flask-pagedown
- faker

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

moment

Visualizzazione “da quanto tempo”

- Moment.js
- Codice da includere
- 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 %}
```

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

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

(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.

```
# 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

E' un linguaggio! (readme Github, Jupytern Notebook, etc…)

markdown (rich text)/1

Moduli: flask_pagedown / bleach e markdown

- Codice da includere

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

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

```python
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

- 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

Cap. 11 pag. 162

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

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():
    # ...
    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)
```

La pagina da renderizzare è presa da "request.ars" dall'URL; 1 default

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

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

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

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">&lt;a href="#"&gt;&hellip;&lt;/a&gt;&lt;/li&gt;
        {% 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

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)

Python Biella Group

Lezione 09

Oggi vedremo...

- Config e ambienti
- Deploy in Heruku
  - Heroku CLI
  - Procfile
- Docker containers
  - Docker desktop
  - Dockerfile
  - Docker compose (orchestrazione base)

Repository Github

https://github.com/PythonGroupBiella/SitoPGB

Nuove librerie

- heroku

Cap. 14 Libro “Flask Web Development”

Python Biella Group

Lezione 10

Oggi vedremo...

- REST API
- REST in Flask

Repository Github

https://github.com/PythonGroupBiella/SitoPGB

Nuove librerie

- flask-httpauth
- httpie

Cap. 17 Libro “Flask Web Development”

API (Application Programming Interfaces)

- Trend: Rich Internet Applications (RIAs)

- Client: presentation and validation
- Server: data retrieval and storage services (web service or API)

- Diversi protocolli:
  - RPC/XML-RPOC
  - SOAP (Simplified Object Access Protocol)
  - Representational State Transfer (REST)

- Flask è un framework ottimale per costruire servizi REST

Cap. 11 Libro “Flask Web Development”

REST (caratteristiche)

Client–server - Chiara separazione tra clients e servers

Stateless - Le richieste client devono contenere tutte le info necessarie per eseguirla; il server non deve memorizzare lo stato del client tra una richiesta e l'altra

Cache - Le risposte del server possono essere eventualmente etichettate come memorizzabili in cache

Interfaccia uniforme - Il protocollo usato deve essere coerente definito e standardizzato (uso di identificatori di risorse univoci, rappresentazioni di risorse, messaggi auto-descrittivi)

Sistema a strati - Server proxy, cache o gateway possono essere inseriti tra client e server per migliorare prestazioni, affidabilità e scalabilità.

Codice su richiesta - I client possono facoltativamente scaricare codice dal server per eseguirlo nel loro contesto.

Cap. 11 Libro “Flask Web Development”

REST (risorse)

Il concetto di "risorsa" è fondamentale: elemento di interesse nel dominio dell'applicazione.

Es. nella nostra app: utenti, blog, commenti

Ogni risorsa deve avere identificativo univoco (Rest è HTTP -> identificativo URL)

Es: /api/utenti/1; /api/posts/12; /api/comments/43 dove 1,12 e 43 sono gli identificativi (PK)

Anche una collezione di risorse ha URL.

Es: /api/utenti/ -> tutti gli utenti

Un API può anche definire un sottoinsieme di risorse

Es: /api/posts/12/comments/ -> tutti i commenti al post 12

Cap. 11 Libro “Flask Web Development”

REST (metodi request)

Table 14-1. HTTP request methods in RESTful APIs

Request method	Target	Description	HTTP response status code
GET	Individual resource URL	Obtain the resource.	200
GET	Resource collection URL	Obtain the collection of resources (or one page from it if the server implements pagination).	200
POST	Resource collection URL	Create a new resource and add it to the collection. The server chooses the URL of the new resource and returns it in a `Location` header in the response.	201
PUT	Individual resource URL	Modify an existing resource. Alternatively, this method can also be used to create a new resource when the client can choose the resource URL.	200 or 204
DELETE	Individual resource URL	Delete a resource.	200 or 204
DELETE	Resource collection URL	Delete all resources in the collection.	200 or 204

Esempi:

GET request https://pythonbiella.herokuapp.com/api/v1/corsi/

GET request https://pythonbiella.herokuapp.com/api/v1/corsi/1/

I metodi di richiesta GET, POST, PUT e DELETE non sono gli unici.

Il protocollo HTTP protocollo si basa su altri metodi, come HEAD e OPTIONS,

che sono automaticamente implementati da Flask.

REST (Request and Response)

Content-Type header (in requests e responses) indica il formato di encoding

JavaScript Object Notation (JSON) o Extensible Markup Language (XML)

JSON: più conciso, più leggero, più leggibile, più comodo

Es:

```
{
 "self_url": "https://pythonbiella.herokuapp.com/api/v1/posts/12345",
 "title": "Scrivere RESTful APIs in Python",
 "author_url": "https://pythonbiella.herokuapp.com/api/v1/utenti/2",
 "body": "... testo ...",
 "comments_url": "https://pythonbiella.herokuapp.com/api/v1/posts/12345/comments"
}
```

REST (Versioning)

Nelle app tradizionali, in caso di aggiornamenti c'è sufficiente controllo per aggiornare tutti client affinchè si adattino alle modifiche.

Nelle app con RIA e servizi la situazione è più complicata: i client sono sviluppati in modo indipendente, da persone diverse e spesso chi offre il servizio neppure conosce tutti i clients.
I servizi devono essere più tolleranti e permettere di lavorare con le vecchie versioni.

Pratica comune: dare una versione al servizio, inclusa nell'URL stessa.
Es: /api/**v1**/posts/, il cui aggiornamento potrà essere ad esempio /api/v2/posts/

REST in Flask/1

URLs -> controllate con **routes**

JSON -> request.get_json() e jsonify() helper function

Creare API blueprint

Gestione errori

Table 14-2. HTTP response status codes typically returned by APIs

HTTP status code	Name	Description
200	OK	The request was completed successfully.
201	Created	The request was completed successfully and a new resource was created as a result.
202	Accepted	The request was accepted for processing, but it is still in progress and will run asynchronously.
204	No Content	The request was completed successfully and there is no data to return in the response.
400	Bad Request	The request is invalid or inconsistent.
401	Unauthorized	The request does not include authentication information or the credentials provided are invalid.
403	Forbidden	The authentication credentials sent with the request are insufficient for the request.
404	Not Found	The resource referenced in the URL was not found.
405	Method Not Allowed	The method requested is not supported for the given resource.
500	Internal Server Error	An unexpected error occurred while processing the request.

Python Biella Group

REST in Flask/2

Autenticazione con Flask-HTTPAuth

Autenticazione basata su token

Serializzazione delle risorse da e a JSON

Implementare gli endpoints delle risorse

Paginazione

Testing con HTTPie

Flask HTTPAuth

Flask-HTTPAuth: wrapper per l'autenticazione HTTP

Nasconde le complessità come Flask-Login

Inizializzata solo nel blueprint API, ovvero dove serve

Utente e password sono verificati usando il modello Utente (con decoratore @auth.verify_password)

Gestione errori con @auth.error_handler

Blocco utenti non confermati

```
@api.before_request
@auth.login_required
def before_request():
    if not g.current_user.is_anonymous and \
            not g.current_user.confirmed:
        return forbidden('Iscrizione non confermata')
```

Autenticazione basata su token

Per evitare di dover passare ogni volta le credenziali, generazione di token (con scadenza)

Token: codifica dell'id utente

Route per richiamare la generazione del token

```
@api.route('/tokens/', methods=['POST'])
def get_token():
    if g.current_user.is_anonymous or g.token_used:
        return unauthorized('Invalid credentials')
    return jsonify({'token': g.current_user.generate_auth_token(
        expiration=3600), 'expiration': 3600})
```

```
class User(db.Model):
    # ...
    def generate_auth_token(self, expiration):
        s = Serializer(current_app.config['SECRET_KEY'],
                       expires_in=expiration)
        return s.dumps({'id': self.id}).decode('utf-8')

    @staticmethod
    def verify_auth_token(token):
        s = Serializer(current_app.config['SECRET_KEY'])
        try:
            data = s.loads(token)
        except:
            return None
        return User.query.get(data['id'])
```

(de)serializzazione JSON

De(serializzazione): conversione da rappresentazione interna (oggetto) a JSON (in questo caso)

Modelli

- to_json()
- from_json()
  - Es. Post
  - Gestione eccezioni nella creazione

```python
from app.exceptions import ValidationError

class Post(db.Model):
    # ...
    @staticmethod
    def from_json(json_post):
        body = json_post.get('body')
        if body is None or body == '':
            raise ValidationError('post does not have a body')
        return Post(body=body)
```

Example 14-15. app/exceptions.py: ValidationError exception

```python
class ValidationError(ValueError):
    pass
```

Example 14-16. app/api/errors.py: API error handler for ValidationError exceptions

```python
@api.errorhandler(ValidationError)
def validation_error(e):
    return bad_request(e.args[0])
```

Endpoints risorse

GET sono le più semplici

Gestione dei permessi

Paginazione

```python
@api.route('/posts/', methods=['POST'])
@permission_required(Permission.WRITE)
def new_post():
    post = Post.from_json(request.json)
    post.author = g.current_user
    db.session.add(post)
    db.session.commit()
    return jsonify(post.to_json()), 201, \
        {'Location': url_for('api.get_post', id=post.id)}


@api.route('/posts/<int:id>', methods=['PUT'])
@permission_required(Permission.WRITE)
def edit_post(id):
    post = Post.query.get_or_404(id)
    if g.current_user != post.author and \
            not g.current_user.can(Permission.ADMIN):
        return forbidden('Permessi insufficienti')
    post.body = request.json.get('body', post.body)
    db.session.add(post)
    db.session.commit()
    return jsonify(post.to_json())
```

```python
@api.route('/corsi/')
def get_corsi():
    page = request.args.get('page', 1, type=int)
    pagination = Corso.query.paginate(
        page, per_page=current_app.config['PBG_CORSI_PER_PAGE'],
        error_out=False)
    corsi = pagination.items
    prev = None
    if pagination.has_prev:
        prev = url_for('api.get_corsi', page=page-1)
    next = None
    if pagination.has_next:
        next = url_for('api.get_corsi', page=page+1)
    return jsonify({
        'corsi': [c.to_json() for c in corsi],
        'prev': prev,
        'next': next,
        'count': pagination.total
    })
```

Documentazione API

E' quella utile per chi le vuole chiamare, tipo quella di Flasky

Con framework più moderni, tipo **FastAPI**, generazione automatica!

Table 14-3. Flasky API resources

Resource URL	Method	Description
/users/<int:id>	GET	Return a user.
/users/<int:id>/posts/	GET	Return all the blog posts written by a user.
/users/<int:id>/timeline/	GET	Return all the blog posts followed by a user.
/posts/	GET	Return all the blog posts.
/posts/	POST	Create a new blog post.
/posts/<int:id>	GET	Return a blog post.
/posts/<int:id>	PUT	Modify a blog post.
/posts/<int:id>/comments/	GET	Return the comments on a blog post.
/posts/<int:id>/comments/	POST	Add a comment to a blog post.
/comments/	GET	Return all the comments.

Testing con HTTPie

Da terminale

Richiamo servizi con USER e PWD:

```
http --json --auth USER:PWD GET https://pythonbiella.herokuapp.com/api/v1/corsi/
```

Richiamo servizi con TOKEN:

```
http --auth USER:PWD --json POST https://pythonbiella.herokuapp.com/api/v1/tokens/
```

per ottenerla

```
http --json --auth TOKEN: GET https://pythonbiella.herokuapp.com/api/v1/corsi/
```

```
http --json --auth TOKEN: GET https://pythonbiella.herokuapp.com/api/v1/corsi/1/serate/
```

THE END…

E’ tempo di feedback !

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