# BD 2 - Autenticazione con Flask

Luca Cosmo

Università Ca' Foscari Venezia

Università
Ca'Foscari
Venezia

# Introduzione

In questa lezione mettiamo insieme i pezzi studiati nelle ultime settimane, sviluppando un sistema di autenticazione basato su Flask.

Ingredienti:

- conoscenza base di Python e Flask
- un database di utenti, con cui ci interfacceremo via SQLAlchemy
- una libreria di autenticazione (Flask-Login)

Quanto studieremo oggi potrà essere esteso per diventare il primo pezzo del vostro progetto.

# Database di Utenti

Rappresentiamo gli utenti tramite una tabella `Users`:

- `email`: indirizzo email dell'account (chiave primaria)
- `pwd`: password di accesso dell'account

Esercizio di riscaldamento! Scrivere uno script Python che:

- si connette al vostro DBMS preferito tramite SQLAlchemy
- crea la tabella `Users` con la struttura indicata
- inserisce nella tabella due utenti di prova (Alice e Bob)

# Flask-Login

Flask-Login è una libreria di autenticazione allo stato dell'arte per Flask:

- implementa l'autenticazione tramite le sessioni native di Flask
- consente di limitare l'accesso a certe pagine ai soli utenti autenticati
- indipendente dalla scelta del DBMS usato per gli utenti: in effetti potreste benissimo salvare gli utenti in un file esterno!
- non supporta autorizzazione: è vostro compito implementarla, ma sono disponibili altre estensioni di Flask (per esempio Flask-RBAC) per tale scopo

Installabile tramite: `pip install flask-login`

# Flask-Login: Configurazione

L'inizializzazione di Flask-Login richiede una chiave segreta, che viene utilizzata per firmare un cookie da impostare nel browser degli utenti:

```
from flask_login import *

app = Flask(__name__)
app.config['SECRET_KEY'] = 'ubersecret'

login_manager = LoginManager()
login_manager.init_app(app)
```

Il cookie salva nel client le informazioni della sessione autenticata, fra cui l'identità dell'utente autenticato.

# Flask-Login: Utilizzo

Flask-Login si appoggia ai seguenti ingredienti principali:

1. una classe utente che definisce la struttura degli utenti del sistema, che deve includere un attributo univoco detto identità (tipo: stringa)
2. una funzione `login_user(usr)`, che salva nella sessione l'identità dell'utente autenticato, eseguendo quindi il login
3. una funzione `logout_user()`, che fa l'opposto (logout)
4. una callback che trasforma l'identità precedentemente salvata nella sessione in un'istanza della classe utente, che viene resa accessibile a Flask tramite l'oggetto `current_user`
5. un decoratore `@login_required` per proteggere le route che richiedono autenticazione

E' compito del programmatore definire 1 e 4.

# Struttura degli Utenti

La classe `User` eredita dalla classe `UserMixin` di Flask-Login. E' usata per portare le informazioni sugli utenti dal database a Flask:

```
class User(UserMixin):
    # costruttore di classe
    def __init__(self, id, pwd):
        self.id = id
        self.pwd = pwd
```

E' richiesta la definizione di un attributo di nome `id`, a meno che non vogliate definirvi un metodo `get_id()` appropriato.

# Callback

La callback `load_user` ha il compito di trasformare un'identità (stringa) in un'istanza della classe `User`, si noti l'uso del decoratore:

```
@login_manager.user_loader
def load_user(user_id):
    with Session(engine) as session:
      return session.get(User,user_id)
```

# Struttura degli Utenti

Oltre a ciò che definite voi, ciascuna classe utente mette nativamente a disposizione i seguenti attributi e metodi via ereditarietà da `UserMixin`:

- `is_authenticated`: attivato se l'utente ha presentato credenziali valide, è l'attributo controllato dal decoratore `@login_required`
- `is_active`: può essere disattivato per bloccare un account
- `is_anonymous`: attivato solo per gli utenti non autenticati
- `get_id()`: ritorna l'identità dell'utente

I cambiamenti dell'attributo `is_authenticated` sono effettuati tramite invocazione degli appropriati metodi di login / logout di Flask-Login.

# Implementare la Home

La home ha il compito di rimandare gli utenti autenticati all'area riservata. Gli utenti anonimi sono invece invitati a fare login:

```
@app.route('/')
def home():
    # current_user identifica l'utente attuale
    # utente anonimo prima dell'autenticazione
    if current_user.is_authenticated:
        return redirect(url_for('private'))
    return render_template("base.html")
```

# Implementare l'Area Riservata

L'area riservata è dedicata ai soli utenti autenticati e visualizza la lista di tutti gli utenti registrati nel sistema:

```
@app.route('/private')
@login_required
def private():
    with Session(engine) as session:
      query = select(User)
      all_users = session.scalars(query)
      resp = make_response(render_template("private.html",
    return resp
```

# Implementare il Login

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        with Session(engine) as session:
            stmt = select(User)
                .where(User.email == request.form['user'],\
                    User.pwd == request.form['pass'])
            try:
                res = session.scalars(stmt).one()
                login_user(res) # chiamata a Flask - Login
                return redirect(url_for('private'))
              except sqalchemy.exc.NoResultFound:
                pass
    return redirect(url_for('home'))
```

dove la funzione `load_user` ritorna un'istanza della classe `User` a partire dall'indirizzo email dell'utente (definita in precedenza).

# Implementare il Logout

L'implementazione del logout è molto semplice:

```
@app.route('/logout')
@login_required # richiede autenticazione
def logout():
    logout_user() # chiamata a Flask-Login
    return redirect(url_for('home'))
```

L'effetto della chiamata è la rimozione dell'identità dell'utente dalla sessione corrente.

Esempio: `project.py`

# Note di Sicurezza

## Autenticazione

In una web app tipicamente ci sono due livelli di autenticazione:

- presso la web application (in fase di login via Flask-Login)
- presso il DBMS (in fase di connessione al DBMS)

Usare la stessa connessione al DBMS per tutti gli utenti è sconsigliato!

## Autorizzazione

Flask-Login vi fornisce la base su cui implementare le vostre politiche di autorizzazione: per esempio potreste estendere la classe `User` con un attributo `role` oppure appoggiarvi a Flask-RBAC.

Bonus: `https://pythonhosted.org/Flask-Security/index.html`

# Checkpoint!

Provate ora a ricostruire l'applicazione che abbiamo discusso in queste slides sulla vostra macchina! In particolare, dovreste avere:

- un'applicazione Flask con 4 route: una per la pagina di benvenuto, una per il login, una per l'area riservata ed una per il logout
- una directory con 2 template: uno per la pagina di benvenuto (login form) ed uno per l'area riservata (lista utenti e pulsante di logout)

Riutilizzate il codice presentato a lezione e completate i pezzi mancanti, in particolare l'integrazione col DBMS, per ottenere una prima web app funzionante.