Advanced Programming with Python

ORMs and sessions

Pepe García jgarciah@faculty.ie.edu

2020-04-20

Plan for today

- Review last day's homework
- flask-sqlalchemy
- Using SQL in web applications

Homework

Let's review last days' homework

flask-sqlalchemy

Let's install flask-sqlalchemy in conda

Object Relational Mapping

ORM is a technique we'll use to relate database records to Python objects.

Object Relational Mapping

ORM is a technique we'll use to relate database records to Python objects.

```sql
CREATE TABLE users (
  id INTEGER PRIMARY KEY,
  username TEXT NOT NULL,
  email TEXT NOT NULL
);
```

```python
class User:
    def __init__(self, id, username, email):
        self.id = id
        self.username = username
        self.email = email
```

You can see how records in the users table could be related to objects of the User class

Declaring models

Models are classes that we'll use to interact with the DB. We'll declare them by extending **db.Model**

```
class User(db.Model):
    pass
```

Adding fields to models

We'll need to make model fields have the same type as the DB columns:

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String, nullable=False)
    email = db.Column(db.String, nullable=False)
```

Object Relational Mapping

Using SQLAlchemy ORM, we can use methods in the class instead of raw SQL:

Object Relational Mapping

Using SQLAlchemy ORM, we can use methods in the class instead of raw SQL:

```sql
SELECT * FROM users;
```

```
User.query.all()
```

Object Relational Mapping

Using SQLAlchemy ORM, we can use methods in the class instead of raw SQL:

```sql
SELECT * FROM users;
SELECT * FROM users WHERE email = 'pepe@ie.edu';
```

```
User.query.all()
User.query.filter(User.email == "pepe@ie.edu")
```

Object Relational Mapping

Using SQLAlchemy ORM, we can use methods in the class instead of raw SQL:

```sql
SELECT * FROM users;
SELECT * FROM users WHERE email = 'pepe@ie.edu';
SELECT * FROM users WHERE email = 'pepe@ie.edu'
                      AND username = 'pepegar;'
```

```
User.query.all()
User.query.filter(User.email == "pepe@ie.edu")
User.query.filter(User.email == "pepe@ie.edu")
          .filter(User.username == "pepegar")
```

Practice

Using the ORM

example_2_orm.py

Sessions

There's a feature of most web applications that we haven't yet discussed, sessions.

Session in the web allow to create sections of websites that are private, for which users need to authenticate in order to access.

Sessions & cookies

Sessions are hold by making HTTP use a special kind of header called cookie.

Cookies are headers that the server sends alongside the HTTP response, that the clien **will send back** in subsequent requests!

Sessions & cookies

Whiteboard

Let's whiteboard the whole flow of cookies and sessions

Using sessions in flask

sessions in flask are handled by importing the session object from flask:

```
from flask import session

@app.route("/")
def index():
    if "user_id" in session: # session behaves like a dictionary
        return render_template("index.html")
    else:
        return render_template("login.html")
```

Using sessions in flask

Full login example

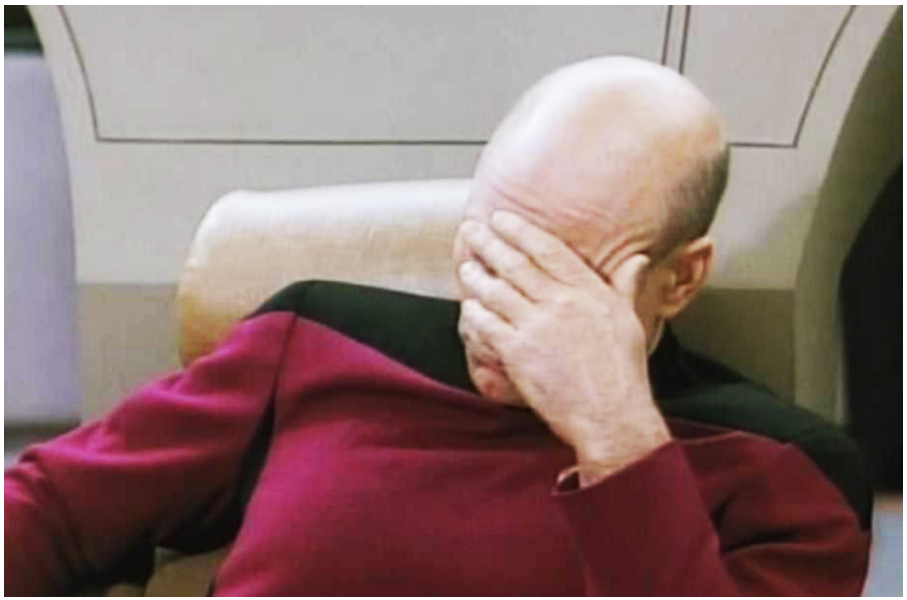
Let's go through a full login example in **example_3_session.py**

Passwords...

There's something **very wrong** about the database we're creating for tweeter...

Passwords...

There's something **very wrong** about the database we're creating for tweeter...



We're storing **PLAIN TEXT PASSWORDS**. That means that anyone with access to the DB can see the passwords right away.

Introducing hash functions

Hash functions are functions that can map arbitrarily sized data to fixed size values.

There are lots of different hash functions, but the ones we'll care about will:

Introducing hash functions

Hash functions are functions that can map arbitrarily sized data to fixed size values.

There are lots of different hash functions, but the ones we'll care about will:

- be **deterministic**

https://en.wikipedia.org/wiki/Hash_function

Introducing hash functions

Hash functions are functions that can map arbitrarily sized data to fixed size values.

There are lots of different hash functions, but the ones we'll care about will:

- be **deterministic**
- make it possible to guess the **plaintext** given the **hash text**

https://en.wikipedia.org/wiki/Hash_function

Introducing hash functions

Hash functions are functions that can map arbitrarily sized data to fixed size values.

There are lots of different hash functions, but the ones we'll care about will:

- be **deterministic**
- make it possible to guess the **plaintext** given the **hash text**
- avoid **collisions**

https://en.wikipedia.org/wiki/Hash_function

hash functions

Hash functions will convert clear text to hashed text.

hash functions

Hash functions will convert clear text to hashed text.

cleartext	hashtext
p4ssw0rd	df984bd56ad2a0df3863b6a0f5230baf520e2b24

hash functions

Hash functions will convert clear text to hashed text.

cleartext	hashtext
p4ssw0rd	df984bd56ad2a0df3863b6a0f5230baf520e2b24
pepegar	5e1249bc5af93d7be8cb9c574bdf5b08e42ebba6

Back to passwords

The approach we'll follow to securely store passwords in our database is that we will **store their hash text** instead of their clear text.

Then, when checking if a user has a specific password, we'll **compare the hashed values**.

Back to passwords

The approach we'll follow to securely store passwords in our database is that we will **store their hash text** instead of their clear text.

Then, when checking if a user has a specific password, we'll **compare the hashed values**.

We'll import `generate_password_hash` and `check_password_hash` from `werkzeug.security`.

Practice

Example 4

Exercise

composing tweets

Make it possible to compose tweets from the frontend of tweeter (`exercise1.py`). It should only be allowed to create tweets for a logged user.