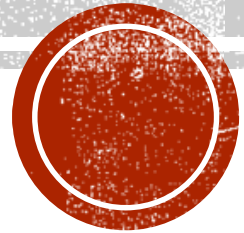
FLASK &DATABASES - II



NEVER SAVE PASSWORDS!

- Why?
- If someone gains access to the database, they will figure out what your users' passwords are!
- Even the staff / development team should not have access to user passwords
- So, we store a hash of the users' passwords
- We use a one-way mathematical function to turn a user's password into a hash
- It's called `*one-way*' because if you know the hash, you cannot (reasonably) recover the password
- **We use two functions called:** `generate_password_hash` **and** `check_password_hash`

MODIFY TWO ROUTES: LOGIN AND SIGNUP

- First, we import the functions into `routes.py`

```
from werkzeug.security import generate_password_hash, check_password_hash
```

- Then insert code into the `signup` route, just before creating the user

```
passw_hash = generate_password_hash(form.password.data)
user = User(username=form.username.data, email=form.email.data, password_hash=passw_hash)
```

- And check the password in the `login` route

```
user_in_db = User.query.filter(User.username == form.username.data).first()
if not user_in_db:
    flash('No user found with username: {}'.format(form.username.data))
    return redirect(url_for('login'))
if (check_password_hash(user_in_db.password_hash, form.password.data)):
    flash('Login success!')
```

SESSIONS

- HTTP is a state-less protocol. This means, that every request to the server is treated independently
- But, sometimes we need to keep track of whether the previous request was successful or not
- Fundamentally, we need to have some notion of the state of interaction [also called *keep state*]
- One way of keeping state is to use a *session* object
- A *session* object is basically an object containing *key-value* pairs, that the server uses to store (and retrieve) information about some user.
- Flask signs the object using the SECRET_KEY that we created in our `config` class

CODE FOR SESSIONS

- Check Firefox using `ctrl + shift + i`. Click on the 'storage' tab, and there should be nothing
- First, import `session` from flask

```
from flask import render_template, flash, redirect, url_for, session
```

- Modify `login` route to add a key-value pair

```
if (check_password_hash(user_in_db.password_hash, form.password.data)):
    flash('Login success!')
    session["USERNAME"] = user_in_db.username
```

- Check Firefox using `ctrl + shift + i`

you should now be able to see a cookie called session

CHECK IF SESSION EXISTS

- Check for a key that you know **must** exist
- If the session object returns a value, then you know that you started a session
- Else, there is no session. Do appropriate error-handling

```
if not session.get("USERNAME") is None:
## session exists, do whatever you need to do
else:
## session does not exist, do error handling
```

IN CODE FOR A NEW ROUTE

```
@app.route('/profile', methods=['GET', 'POST'])
def profile():
    if not session.get("USERNAME") is None:
        if form.validate_on_submit():
```

- If no session exists, handle error

```
    else:
        flash("User needs to either login or signup first")
        return redirect(url_for('index'))
```

FINISHING A SESSION

- Since the session cookie is simply an object, we can delete whatever *key-value* pair we want, using `.pop()`:

```
@app.route('/logout')
def logout():
    session.pop("USERNAME", None)
    return redirect(url_for('login'))
```

CREATE A PROFILE

- We can check for a password
- We can check if user is logged in
- Now, let's create a profile page that should *only* be accessible if the user is logged in

- As usual, when we want to create a new functionality visible to the user, we do the following things:
  - [only if necessary] Create a new table to store data
  - Create a new `Form` class to gather data
  - Create a new template to show the user [add a file in `templates/`]
  - Create a new route in `routes.py`

TO CREATE A PROFILE – II

- Requirements:
  - The profile should only be accessible to a logged-in user
  - The profile should store the date-of-birth, the gender and the CV of the user
  - The CV is a PDF file that can be uploaded by the user, and should be stored by the application

- We create a new table called `Profile`, by adding a class to `models.py`
- We create a new form called `ProfileForm` to store date-of-birth, gender and CV
- We create a template called `profile.html` to show the user
- We add a route called `/profile` to link the form and the template

NEW TABLE? NEW CLASS!

```
class Profile(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    dob = db.Column(db.DateTime, index=True)
    gender = db.Column(db.String(10), index=True)
    cv = db.Column(db.LargeBinary)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))

    def __repr__(self):
        return '<Profile for user: {}, gender: {}, birthday: {}>'.format(self.user_id, self.dob, self.dob)
```

PAY ATTENTION TO RELATIONSHIPS

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), index=True, unique=True)
    email = db.Column(db.String(120), index=True, unique=True)
    password_hash = db.Column(db.String(128))
    posts = db.relationship('Post', backref='author', lazy='dynamic')
    profile = db.relationship('Profile', backref='user', lazy='dynamic')
```

CREATE THE DATABASE

```
(flaskenv) microblog>flask shell
Python 3.7.0b3 (default, Mar 30 2018, 04:35:22)
[GCC 7.3.0] on linux
App: blogapp [production]
Instance: /c/Users/vivek/OneDrive - University College Dublin/ucd/2019/teac
hing/bdic/web-app-dev/lecture-slides/sample-code/week12/microblog/instance
>>> from blogapp import db
>>> db.create_all()
```

CREATE A FORM

```
class ProfileForm(FlaskForm):
    dob = DateField ('Date of Birth', validators = [DataRequired()])
    gender = RadioField('Gender', choices = ['Male', 'Female'], validators=[DataRequired()])
    cv = FileField('Your CV', validators = [FileRequired()])
    submit = SubmitField('Update Profile')
```

CREATE A TEMPLATE

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

{% block content %}
    <h2>Hello, {{ user.username }}!</h2>
    <p> Let's complete your profile:</p>
    <form action="" method="post" enctype="multipart/form-data" novalidate>
    {{ form.hidden_tag() }}
    <p>
        {{ form.dob.label }}<br>
        {{ form.dob(size=32) }}
```

Necessary for file upload!

TEMPLATE HAS CHOICES

```
<p>
    {% for choice in form.gender %}
        <tr>
            <td>{{ choice }}</td>
            <td>{{ choice.label }}</td>
        </tr>
    {% endfor %}

    {% for error in form.gender.errors %}
    <span style="color: red;">[{{ error }}]</span>
    {% endfor %}
</p>
<p>
    {{ form.cv.label }}<br>
    {{ form.cv(size=12) }}

    {% for error in form.cv.errors %}
    <span style="color: red;">[{{ error }}]</span>
    {% endfor %}
</p>
<p>{{ form.submit() }}</p>
```

THEN WE ADD A ROUTE

```
@app.route('/profile', methods=['GET', 'POST'])
def profile():
    form = ProfileForm()
    if not session.get("USERNAME") is None:
        if form.validate_on_submit():
            cv_dir = Config.CV_UPLOAD_DIR
            file_obj = form.cv.data
            cv_filename = session.get("USERNAME") + '_CV.pdf'
            file_obj.save(os.path.join(cv_dir, cv_filename))
            flash('CV uploaded and saved')
            return redirect(url_for('index'))
        return render_template('profile.html', title='Add/Modify your profile', form=form)
    else:
        flash("User needs to either login or signup first")
        return redirect(url_for('login'))
```

MINOR ADJUSTMENTS TO CONFIG

```
class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'

    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
    'sqlite:///' + os.path.join(basedir, 'blogdb.db')

    SQLALCHEMY_TRACK_MODIFICATIONS = False

    CV_UPLOAD_DIR = os.path.join(basedir, 'uploaded_CV')
```

TO-DO IN CLASS

- Download the code shown in class, and make it run
- We have not yet added the profile object to the profile table. Create a profile object [hint: look at how the Profile object was created and linked to the user] and add it to the database
- Inspect the database to check if data was stored correctly