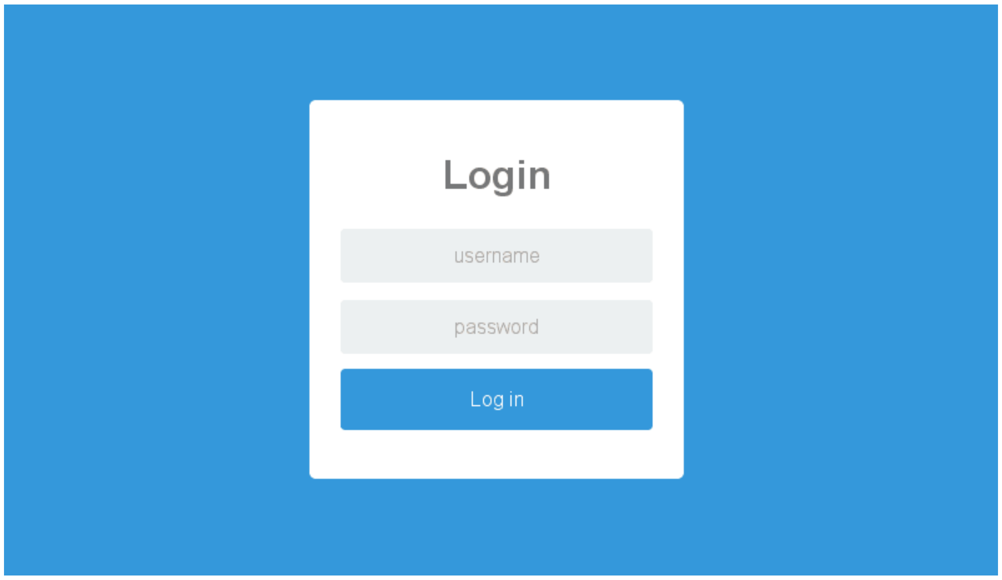
# E-14a: Building Web Applications for Data Analysis

## Lab 6: User Logins and Sessions



*Web form*

### Learning Objectives

- Create Signup and Login forms
- Place content from multiple pages into DB
- Reference content to users in DB
- Understand HTTP methods
- Understand Authentication
- Understand User Sessions

## INTRODUCTION

In previous weeks you learned how to create the user forms and how to work with a database. The next step is understanding user sessions and create a user login system.

Steps:

- Download “lab_6” working files
- Create database
- Create login and signup web forms
- Connect with a database
- Update index page

## Database “lab_6”

Open Postgres and create new database with name “lab_6”. The database should table “users” :

```
CREATE TABLE users (
    uid serial NOT NULL PRIMARY KEY,
    username TEXT NOT NULL,
    password TEXT NOT NULL
    );
```

Populate the tables with some arbitrary information, for example:

```
INSERT INTO users (username, password) VALUES ('Michelle', 'test_pw1');
INSERT INTO users (username, password) VALUES ('Jasmine', 'test_pw2');
```

## Create Web Forms

After creating *users* table in the database, we need to write a user model that can read and write into these tables. First step is to create a *signup* page, so that we can take in user data and save it to the database. User will visit the URL *signup* to create a new account. The page will be retrieved through an HTTP GET request and will load in the browser.

Second, the sign up page will have fields for a username and password. The user will fill out the form and press *Signup* button. Third, when a user presses *Signup*, the forms will submit to the flask app through a POST request and will hit *routes.py*. In *routes.py*, a function will check whether the submitted data is valid. If any of the submitted data does not pass validation, the sign up page will load again, with helpful error messages, so that the user can try again.

Otherwise, if all fields are valid, we'll save the user's credentials to the database. The user will be signed in and redirected to the homepage. We're going to use the *signup* form to take a new user's data and save it to the database. Remember from the previous labs, instead of packing in extra functionality, Flask lets you add it on as needed, using extensions. To add functionality for web forms, we will use Flask-WTF.

In order to make a web form using Flask WTF we need to do two things. First we need to set up a backend where we define the form's fields. Second we need to setup a frontend where we display form to the user.

## Signup Web Form

Create a new file named *forms.py* and write the form there. Then open up *forms.py* and add the following code to it:

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, TextAreaField
from wtforms.validators import DataRequired
```

First we import a few helpful classes from Flask WTF. The base form class, a text field, a password field and a submit button. Next we create a new class with name *SignupForm* and define each field we want in the form:

```
class SignupForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Signup')
```

When the user visits the URL /`signup`, a corresponding page containing this form should show up. This means we need to create a new URL mapping in *routes.py*. Open up *routes.py* and import the newly created forms:

```
from forms import SignupForm
```

Next, type `app.route("/signup")` and create `signup` function. Inside `signup` function, first create a new usable instance of *SignupForm* and then send this form to a web template named *signup.html*.

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    form = SignupForm()
    Return render_template("singup.html", form=form)
```

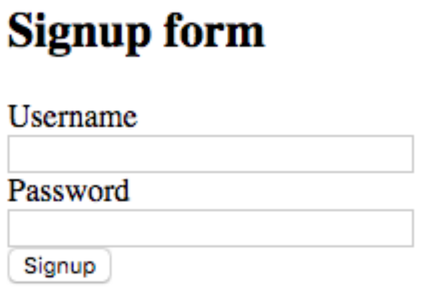
Open up the *signup.html* and add following code to define username and password fields:

```
<div class="form-group">
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>

            {% for error in form.username.errors %}
            <span style="color: red;">[{{ error }}]</span>
{% endfor %} </div>

 <div class="form-group">
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
            <span style="color: red;">[{{ error }}]</span>
{% endfor %} </div>
```

Here we create username and password fields. Now, we have to create a submit button:

```
{{ form.submit(class="btn btn-primary btn-xl") }}
```

You can visit *localhost:5000/signup* in the browser and see the results of your work:



The signup form looks good, however when we click the *Signup* button, we get an error that says *Method Not Allowed*. Go back to *routes.py* in order to solve this problem.

What we need to do is encounter for two possible scenarios: if a user visits `/signup`, this page is retrieved through a GET request. Else, if the user fills in the form and presses *Signup*, the form submits to the Flask app through a POST request. Impact on this logic can be expressed as an `if-else` statement.

In the first line we imported another Flask class that we can use to distinguish between GET and POST requests. It's called `request`. So let's go ahead and use it in the signup function. Add `methods=['GET', 'POST']` and then add an `if-else` statement to distinguish between GET and POST requests. If a form has been submitted, a POST request has happened:

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    form = SignupForm()
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
    else:
        return render_template('signup.html', title='Signup', form=form)
```

Let's visit *localhost:5000/signup* again, fill in the form, and click that *Signup* button. The *signup.html* generated the signup form html, rendered html was sent back to *routes.py* and sent that rendered html back to the browser, which we see when the page loads.

When a form has all valid data and is submitted, we want to save that data into the database. To do this, let's add the following code to *models.py*:

```
class User(db.Model):
    __tablename__ = 'users'
    uid = db.Column(db.Integer, primary_key=True, autoincrement=True)
    username = db.Column(db.String(64), unique=True, nullable=False)
    password = db.Column(db.String(128), nullable=False)
```

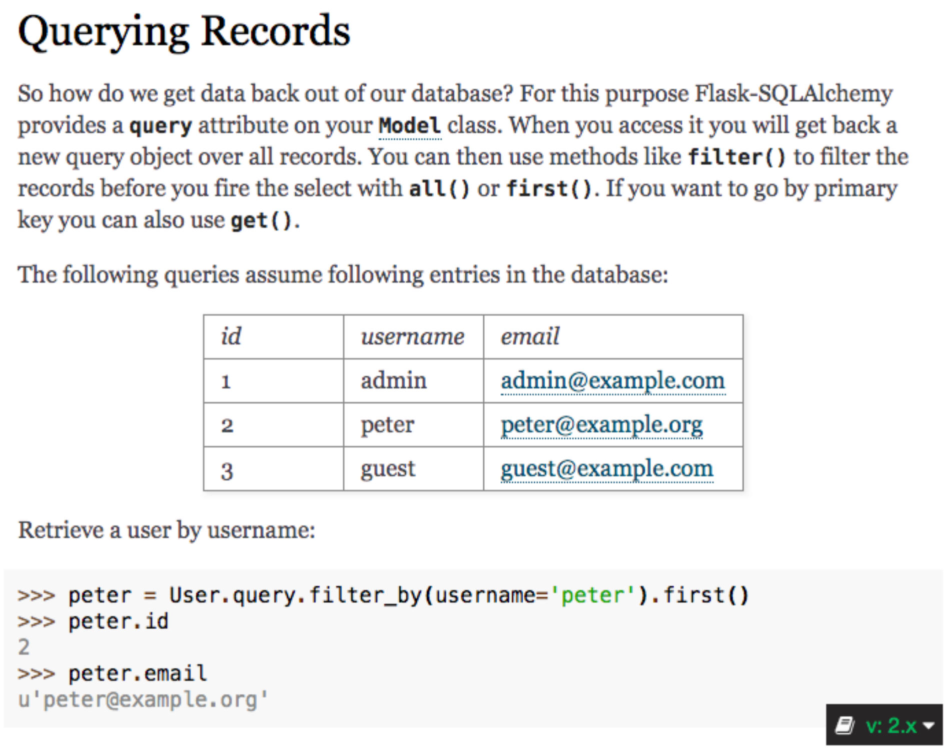
Go back to your *routes.py*. Import the user class from *models.py*:

```
from models import db, User
```

Then create a new instance of the user object, and initialize it with data from the sign up form. In the `signup()` function, `if request.method == 'POST'`, first check if the user is already in the database:

```
        existing_user = User.query.filter_by(username=username).first()
```

Take a look at the following image to better understand the process of querying records.



## Querying Records

So how do we get data back out of our database? For this purpose Flask-SQLAlchemy provides a **query** attribute on your **Model** class. When you access it you will get back a new query object over all records. You can then use methods like **filter()** to filter the records before you fire the select with **all()** or **first()**. If you want to go by primary key you can also use **get()**.

The following queries assume following entries in the database:

| *id* | *username* | *email* |
|---|---|---|
| 1 | admin | admin@example.com |
| 2 | peter | peter@example.org |
| 3 | guest | guest@example.com |

Retrieve a user by username:

```
>>> peter = User.query.filter_by(username='peter').first()
>>> peter.id
2
>>> peter.email
u'peter@example.org'
```

Source: [link]()

If true, inform user and render *signup.html* page again:

```
        if existing_user:
            flash('The username already exists. Please pick another one.')
            return redirect(url_for('signup'))
```

else, populate the database with user's credentials. Note: we are not passing the open password information to a database. Instead we are using `sha256_crypt.hash()` to securely store users passwords in database. After signing up redirect users to the homepage:

```
        else:
            user = User(username=username, password=sha256_crypt.hash(password))
            db.session.add(user)
            db.session.commit()
            flash('Congratulations, you are now a registered user!')
            return redirect(url_for('login'))
```

## Logging in

First, what is a **session**? When you sign in, an app needs some way of identifying page requests you make so it can serve your information. The app looks up whether the credentials are valid or not. If they are, then the app understands that you're a signed in user and returns a response with the relevant information. This combination of having the information stored in your browser that maps to valid user credentials on the server, is called a *session*. Flask has a session object that makes this functionality easy to implement.

Open up *forms.py* and create a new class named login form. The login form class will be similar to the sign up form class we created. So inherit from the base form class. There will be two fields in this class for a username and password. Both fields will have validators on them. The DataRequired validator checks that the field contains data.

```
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit                      =                     SubmitField('Login')
```

Next, open login.html to define the form frontend:

```
<div class="form-group">
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
            <span style="color: red;">[{{ error }}]</span>
       {% endfor %}
</div>

<div class="form-group">
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
            <span style="color: red;">[{{ error }}]</span>
      {% endfor %}
</div>
```

```
{{ form.submit }}
```

We just created a new class named login form in *forms.py*. Let's move on and create a new URL mapping for `/login` in *routes.py*. Open *routes.py* and see that we already imported `session` from Flask at the top. Then, create a new URL mapping for `/login`. It will accept two methods: GET and POST. And then the login function itself will be similar to the sign up function above. First, import:

```
from forms import SignupForm, LoginForm
```

And than add:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
```

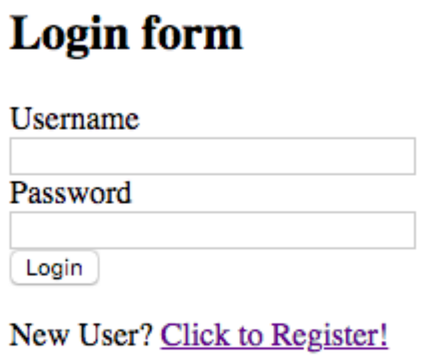
Next we are checking the validation (the code provided below). If the form validates, we'll fetch the data from the login form. So, collect the username and the password data and check whether user exists in the database with these credentials. Here, `User.query.filter_by` searches the database for a user with that email address.

If a user does exist, and the password check works out, then log in the user by creating a new session and redirecting to the home page.

If a user does NOT exist in the database with that username and password, then we need to reload the login form. So what this line does is it redirects to the login route, which will trigger the GET request. If a GET request has been received, we simply render the form.

```
if request.method == 'POST':
   username = request.form['username']
   password = request.form['password']

   user = User.query.filter_by(username=username).first()

      if user is None or not sha256_crypt.verify(password, user.password):
            flash('Invalid username or password')
            return redirect(url_for('login'))
```

```
        else:
            session['username'] = username
            return redirect(url_for('index'))
    else:
        return render_template('login.html', title='Login', form=form)
```



Login form

Username

Password

New User? Click to Register!

Log in with your credentials to see if it all works. Click sign in and check if it redirects to the home page as expected. The login page is up and running.

## Logging out

Logging a user means creating a new session. Creating a new session means setting a cookie in the user's browser, containing an ID and mapping that ID to a user's credentials. Therefore, logging out a user must mean deleting a session. This means clearing cookies in the users browser and removing that mapping to the user's credentials.

Open up *routes.py*, and create a new URL mapping for `/logout`. Type the following:

```
@app.route('/logout', methods=['POST'])
def logout():
    session.clear()
    return redirect(url_for('index'))
```

## Setting up Index page

Now that we've implemented the sign up page functionality, let's update index page. Index page should have two different messages depending on visitors. Open *index.html* and place the following code:

```
{% if session_username is defined %}
<h1>Hi, {{ session_username }} !</h1>
{% else %}
<h1>Hi, guest!</h1>
{% endif %}
```

Next, we want to define a dynamic user interface for a different user groups:

```
{%        if        session_username        is        defined        %}
      <form              action="/logout"              method="POST">
        <button      type="submit">      Log      out      </button>
      </form>

{%                              else                              %}
      <form              action="/login"              method="GET">
        <button      type="submit">      Log      in      </button>
      </form>
      <form              action="/signup"              method="GET">
        <button      type="submit">      Sign      up      </button>
      </form>

{% endif %}
```

To enable this functionality, go back to the URL mapping for `/index` and add:

```
if 'username' in session:
      session_user = User.query.filter_by(username=session['username']).first()
      return render_template('index.html', title='Home',
      session_username=session_user.username)
```

```
else:
      return render_template('index.html', title='Home')
```

## Credits and Additional Resources

The Flask Mega-Tutorial Part III: Web Forms
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iii-web-forms

The Flask Mega-Tutorial Part V: User Logins
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-v-user-logins

The Flask Mega-Tutorial Part IX: Pagination
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-ix-pagination