Final Project: Milestone 3

Matias Cinera
Keylin Sanchez
Devin Parmet

COP 4710
Dr. William Hendrix
University of South Florida

November 29, 2021

Table of Contents

1. Overview

For our group project, we designed and implemented a website for "PWR Fit", a mock gym where members can register for classes and trainers can register to teach classes and view all members registered for the classes they are teaching.

The front-end of this project was developed with python Flask. The DBMS used was MySQL.

2. Design/Schema

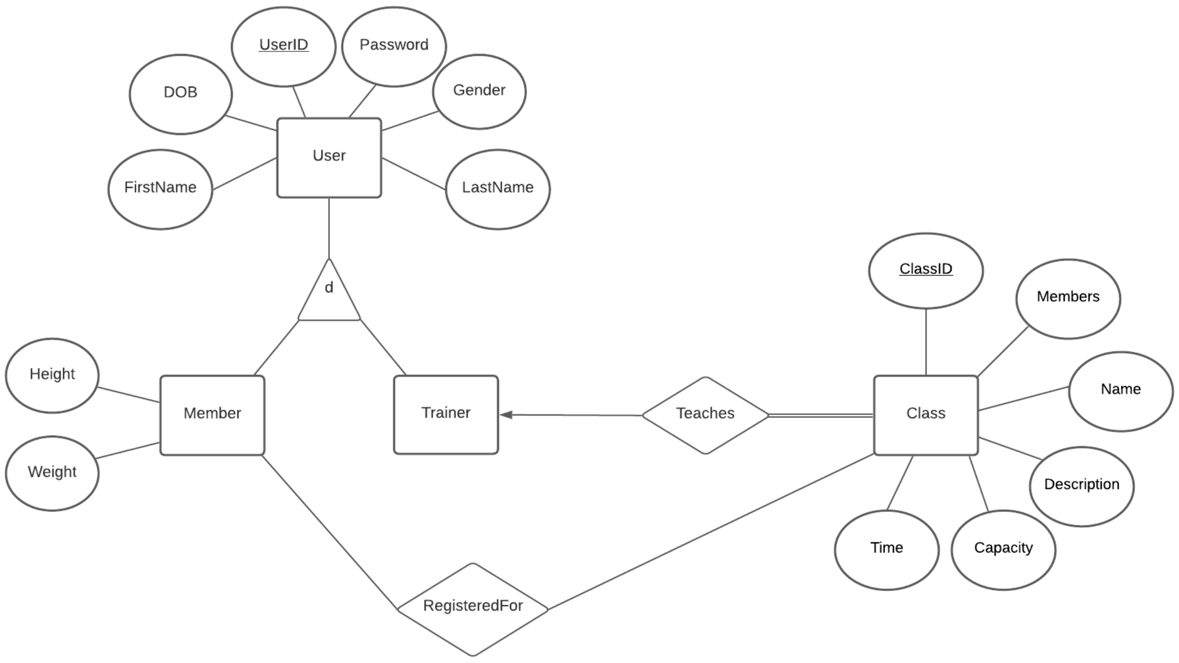


Figure 2.1: The ER diagram constructed that was used to plan out the table schema for each entity/ relationship in the database.

The following SQL statements were used to create the tables within our database:

```
CREATE TABLE member(username VARCHAR(20) PRIMARY KEY,
     password VARCHAR(20),firstname VARCHAR(15),
     lastname VARCHAR(15),dob DATE,height REAL,weight REAL);

CREATE TABLE trainer(username VARCHAR(20) PRIMARY KEY,
    password VARCHAR(20),firstname VARCHAR(15),
    lastname VARCHAR(15),dob DATE);
```

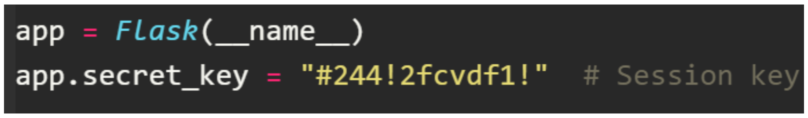
```
CREATE TABLE class(classID INTEGER PRIMARY KEY,
     instructor VARCHAR(20) REFERENCES trainer,
     name VARCHAR(25),description VARCHAR(400),
     capacity INTEGER,time TIMESTAMP(0));

CREATE TABLE RegisteredFor (username VARCHAR(20),
     classID INTEGER REFERENCES class,
     PRIMARY KEY(username, classID), FOREIGN KEY(username)
     REFERENCES member(username) ON DELETE CASCADE);
```

3. Application Logic/Implementation

3.1. Application Structure

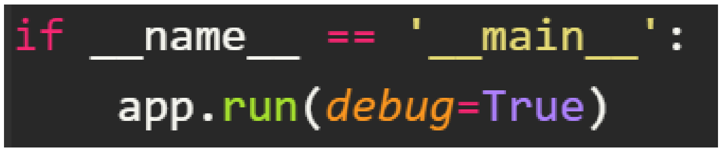
```
.
|--__pycache__/
|--templates/
|  |--***HTML files***
|--***application routes and input validation file***
|--***JSON files for class, member, and trainer data***
|--***SQL Command files***
|--***README file***
```

3.2. Flask Setup



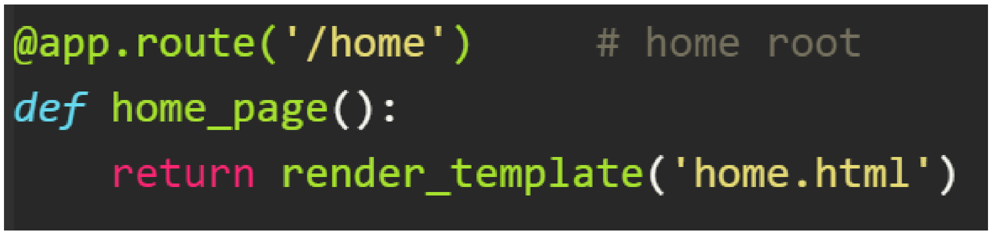
```
app = Flask(__name__)
app.secret_key = "#244!2fcvdf1!"  # Session key
```

Figure 3.2.1: Code to set up flask and the secret key used for input validation (found in app.py)



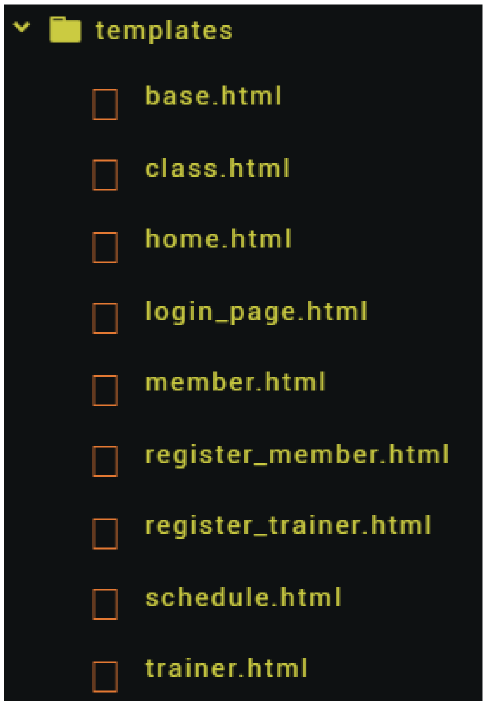
```
if __name__ == '__main__':
    app.run(debug=True)
```

Figure 3.2.2: Code to run flask in main (found in app.py)



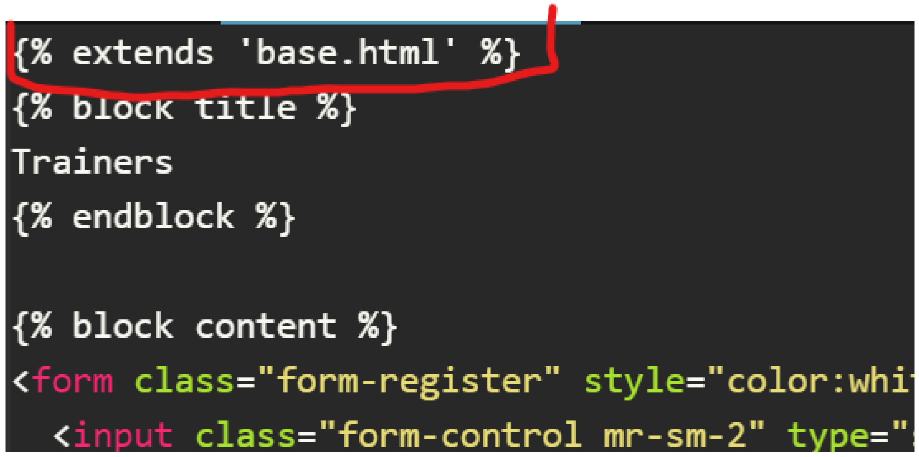
```
@app.route('/home')     # home root
def home_page():
    return render_template('home.html')
```

Figure 3.2.3: Example of an application route (found in app.py)



```
templates
    base.html
    class.html
    home.html
    login_page.html
    member.html
    register_member.html
    register_trainer.html
    schedule.html
    trainer.html
```

Figure 3.2.4: All routes (from the app.py file) reference an HTML file (found in the templates directory)



```
{% extends 'base.html' %}
{% block title %}
Trainers
{% endblock %}

{% block content %}
<form class="form-register" style="color:whi
  <input class="form-control mr-sm-2" type="
```

Figure 3.2.5: All HTML files (found in the templates directory) reference the base.html file (also within the templates directory)

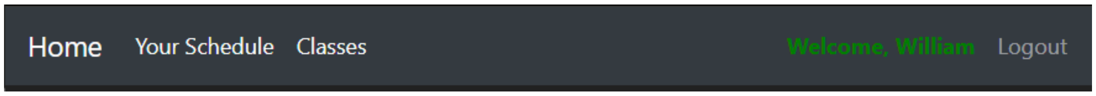
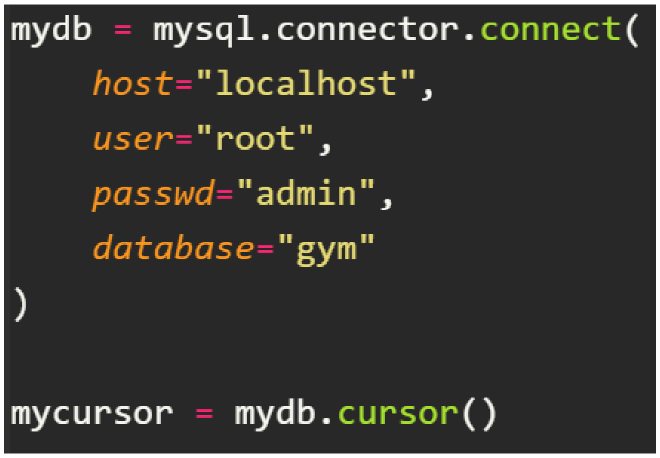


Figure 3.2.6: The base.html file (found in the templates directory) pertains to the navigation bar of the application.

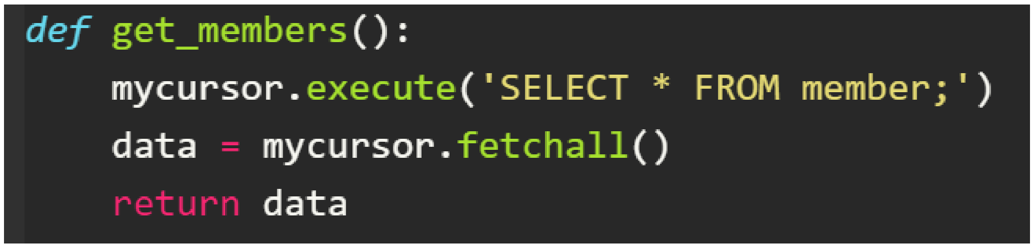
3.3. SQL Connector



```
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="admin",
    database="gym"
)

mycursor = mydb.cursor()
```

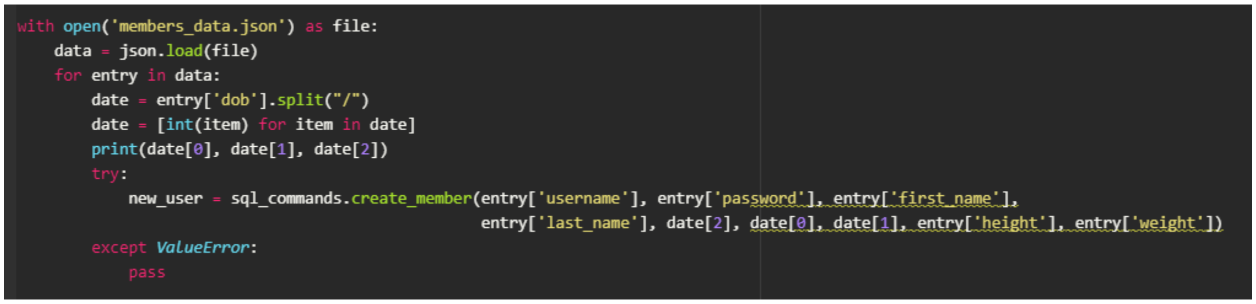
Figure 3.3.1: Setting up the SQL Connector object within the sql_commands.py

3.4. Sample Implementation



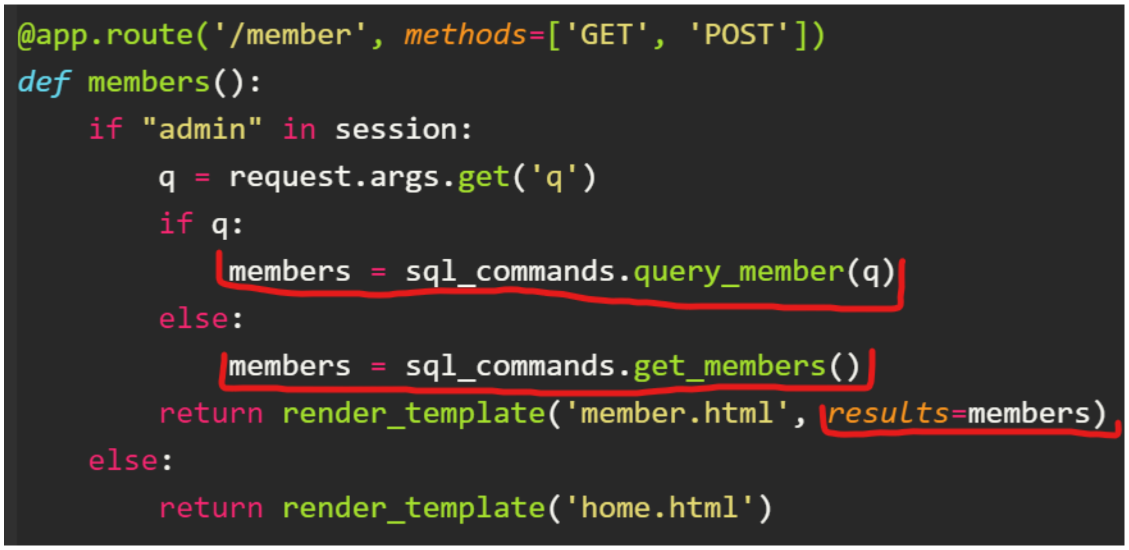
```
def get_members():
    mycursor.execute('SELECT * FROM member;')
    data = mycursor.fetchall()
    return data
```

Figure 3.4.1: Function to return all members of the database with the SQL object. (found in sql_commands.py)



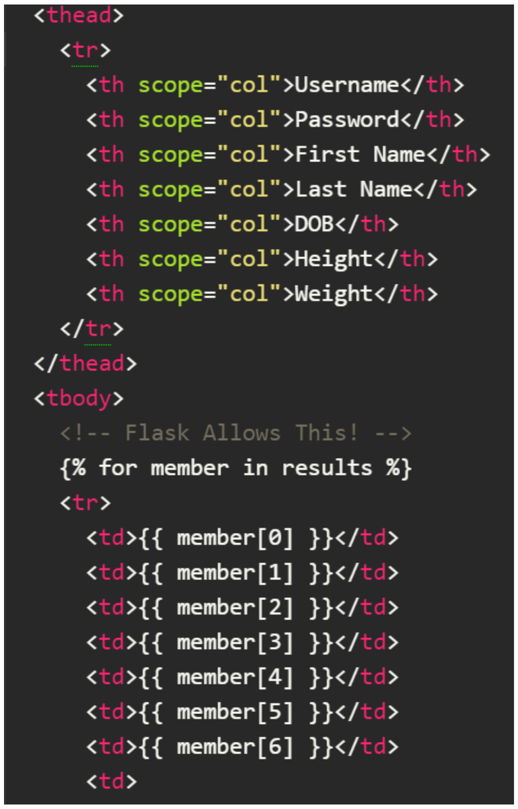
```python
with open('members_data.json') as file:
    data = json.load(file)
    for entry in data:
        date = entry['dob'].split("/")
        date = [int(item) for item in date]
        print(date[0], date[1], date[2])
        try:
            new_user = sql_commands.create_member(entry['username'], entry['password'], entry['first_name'],
                                                  entry['last_name'], date[2], date[0], date[1], entry['height'], entry['weight'])
        except ValueError:
            pass
```

Figure 3.4.2: Adding member data using the members_data.JSON file



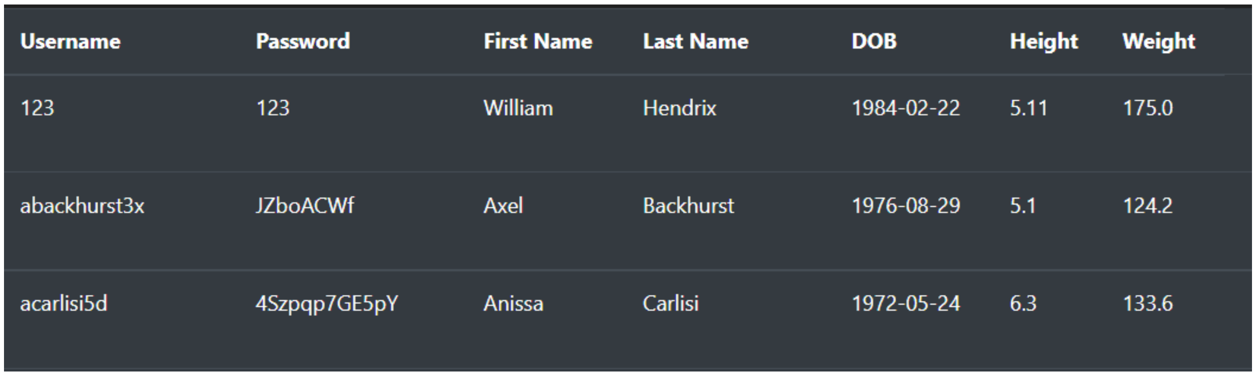
```python
@app.route('/member', methods=['GET', 'POST'])
def members():
    if "admin" in session:
        q = request.args.get('q')
        if q:
            members = sql_commands.query_member(q)
        else:
            members = sql_commands.get_members()
        return render_template('member.html', results=members)
    else:
        return render_template('home.html')
```

Figure 3.4.3: Invoking the get_members function in the flask route function (found in app.py)



```
<thead>
  <tr>
    <th scope="col">Username</th>
    <th scope="col">Password</th>
    <th scope="col">First Name</th>
    <th scope="col">Last Name</th>
    <th scope="col">DOB</th>
    <th scope="col">Height</th>
    <th scope="col">Weight</th>
  </tr>
</thead>
<tbody>
  <!-- Flask Allows This! -->
  {% for member in results %}
  <tr>
    <td>{{ member[0] }}</td>
    <td>{{ member[1] }}</td>
    <td>{{ member[2] }}</td>
    <td>{{ member[3] }}</td>
    <td>{{ member[4] }}</td>
    <td>{{ member[5] }}</td>
    <td>{{ member[6] }}</td>
    <td>
```

Figure 3.4.4: Referencing the results of the function in figure 3.4.3 in the member.html file



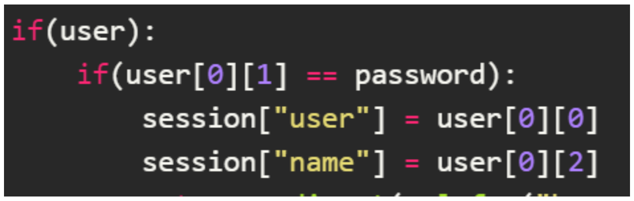
Username	Password	First Name	Last Name	DOB	Height	Weight
123	123	William	Hendrix	1984-02-22	5.11	175.0
abackhurst3x	JZboACWf	Axel	Backhurst	1976-08-29	5.1	124.2
acarlisi5d	4Szpqp7GE5pY	Anissa	Carlisi	1972-05-24	6.3	133.6

Figure 3.4.4: The results displayed on the web application when locally deployed

3.5. Functionality

Our group used Flask Sessions for user validation. **Sessions are cookies stored in the user browser's cache.** In python, sessions are a dictionary (or map).

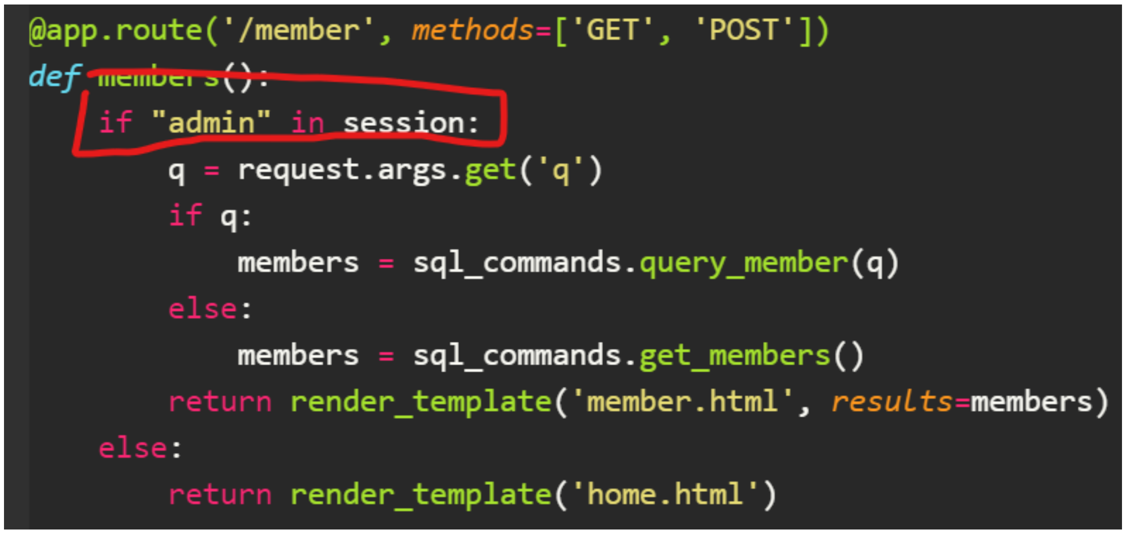
If the member logs in, the application adds a “user” in the dictionary, if a trainer logs in, the application adds an “admin”.



```
if(user):
    if(user[0][1] == password):
        session["user"] = user[0][0]
        session["name"] = user[0][2]
```

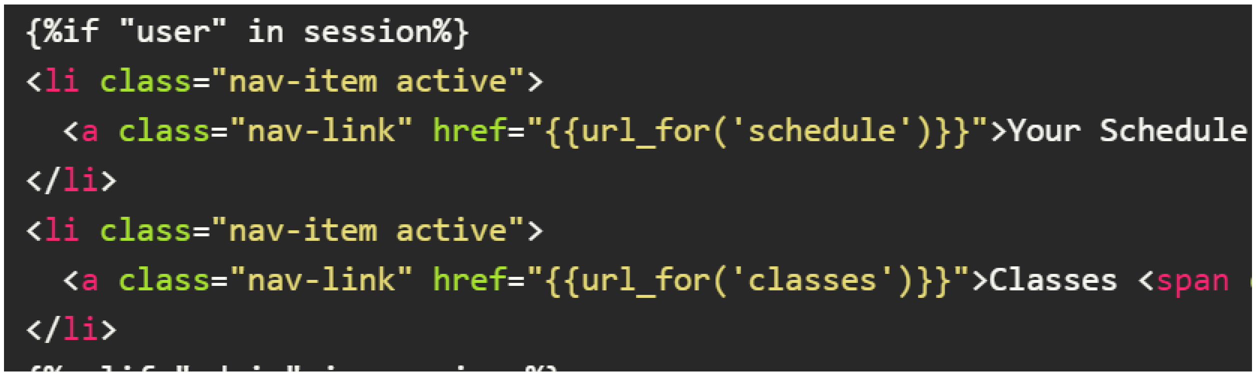
Figure 3.5.1: Sample code depicting how the dictionary adds a user/admin to dictionary

Only “admins” (trainers) can see the members page. So, even if the user had a link to the members page, it will get redirected to the homepage. Session data can also be referenced in the html via flask. This way, the navigation bar can change depending on the user.



```
@app.route('/member', methods=['GET', 'POST'])
def members():
    if "admin" in session:
        q = request.args.get('q')
        if q:
            members = sql_commands.query_member(q)
        else:
            members = sql_commands.get_members()
        return render_template('member.html', results=members)
    else:
        return render_template('home.html')
```

Figure 3.5.2: Function within the app.py file to route members and trainers to the appropriate web pages.



```
{%if "user" in session%}
<li class="nav-item active">
  <a class="nav-link" href="{{url_for('schedule')}}">Your Schedule
</li>
<li class="nav-item active">
  <a class="nav-link" href="{{url_for('classes')}}">Classes <span
</li>
```

Figure 3.5.3: Code from the base.html file referencing session data.

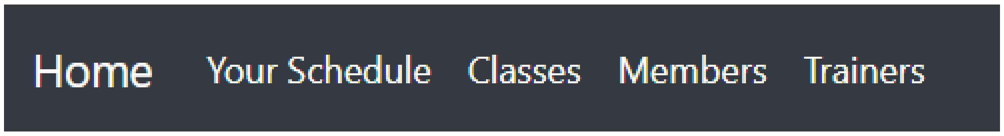


Figure 3.5.4: The navigation bar when logged in as a trainer.

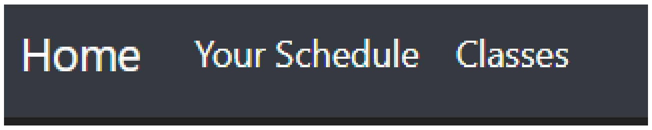


Figure 3.5.5: The navigation bar when logged in as a member.

Log out button will clear session data, and clears the cookie in your browser.

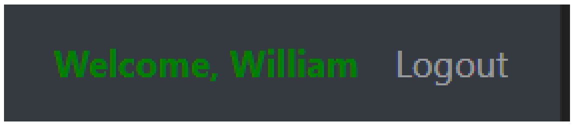
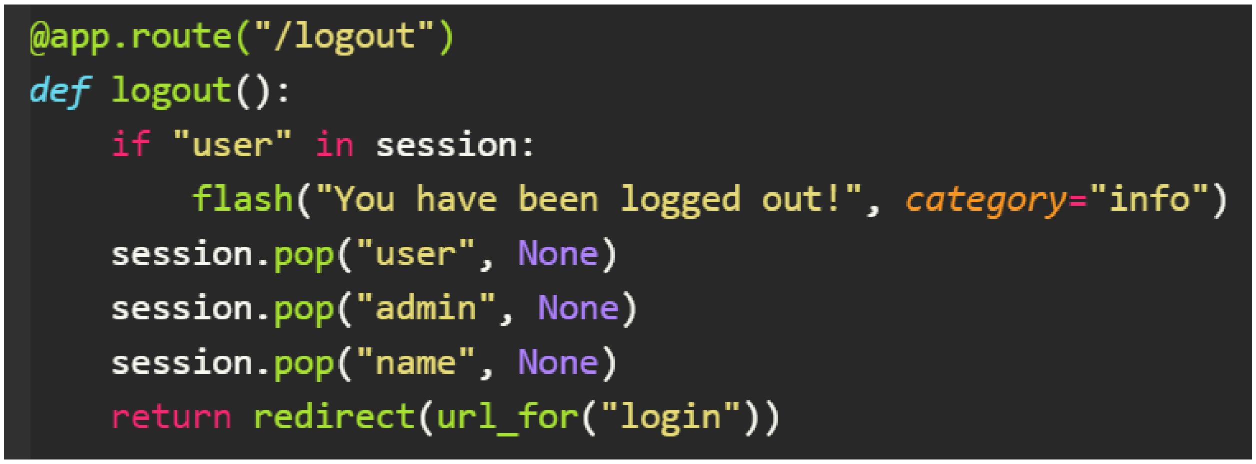


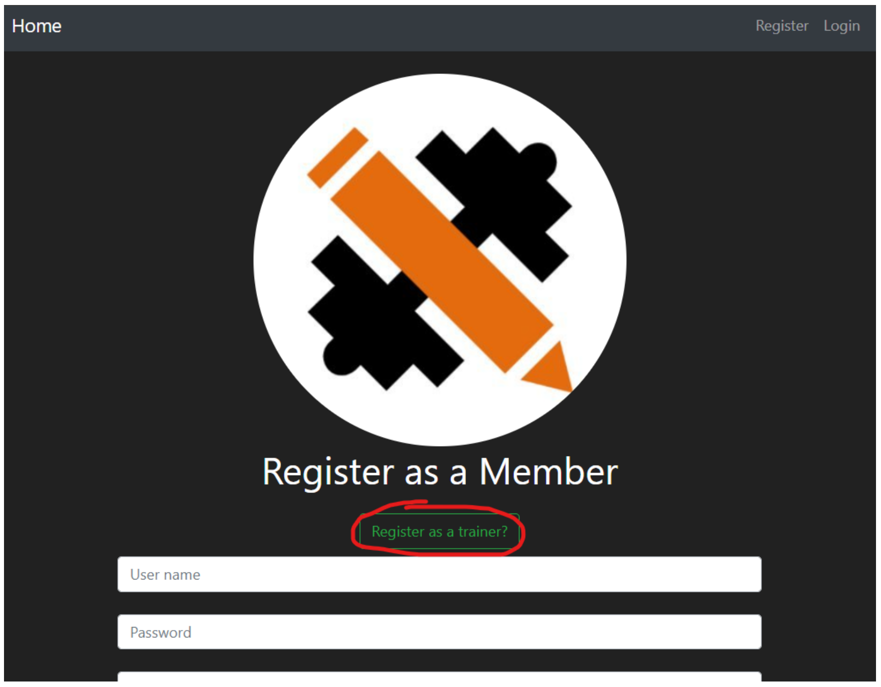
Figure 3.5.6: The option to logout displayed for a user that is logged in.



```python
@app.route("/logout")
def logout():
    if "user" in session:
        flash("You have been logged out!", category="info")
    session.pop("user", None)
    session.pop("admin", None)
    session.pop("name", None)
    return redirect(url_for("login"))
```

Figure 3.5.7: Code from the app.py file that clears the session data and redirects the application user to the login page.

There are two registration forms pertaining to registration as a member and trainer.



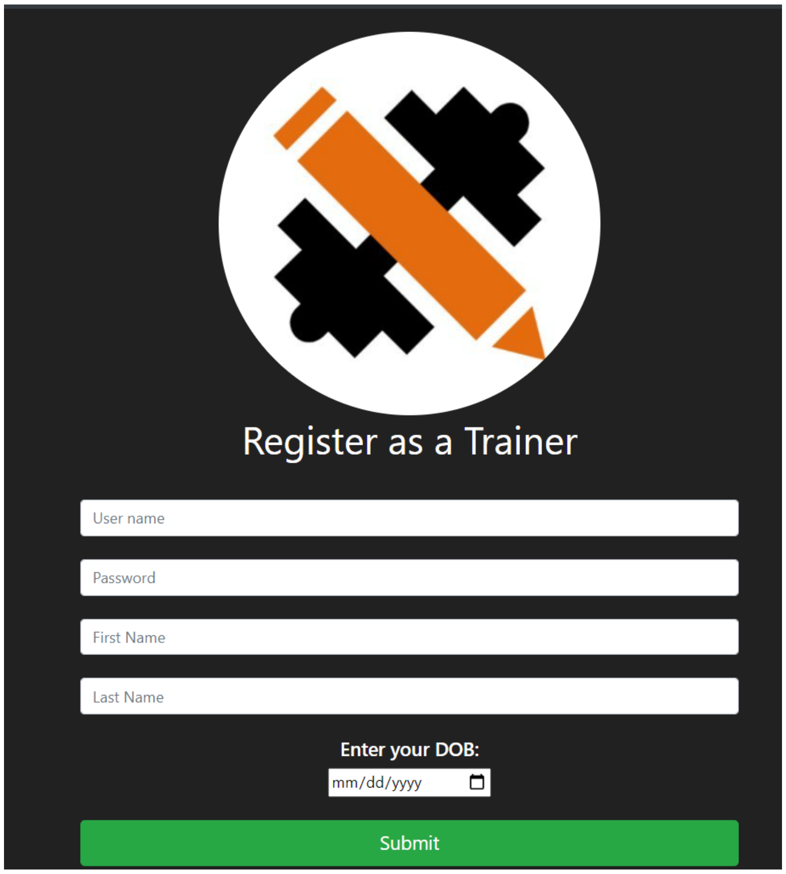


Figure 3.5.8 and 3.5.9: The two forms for registration within the application.

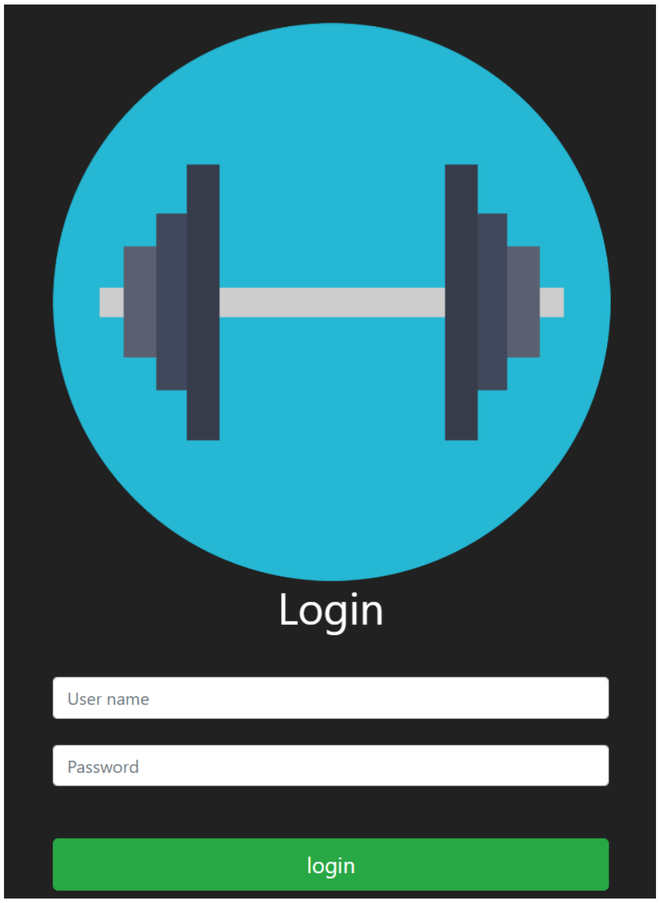


Figure 3.5.10: The login form for the application.

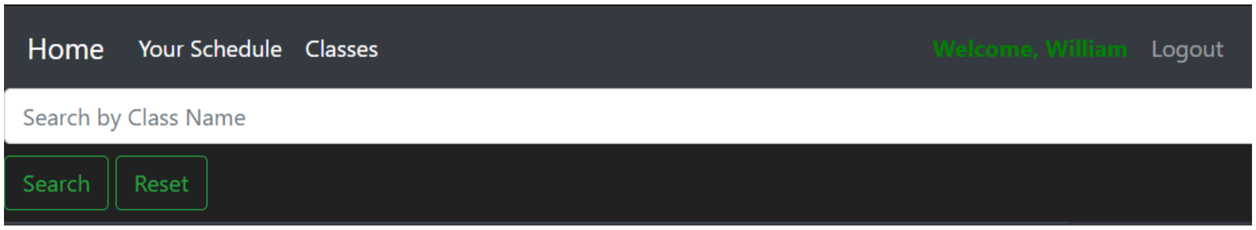
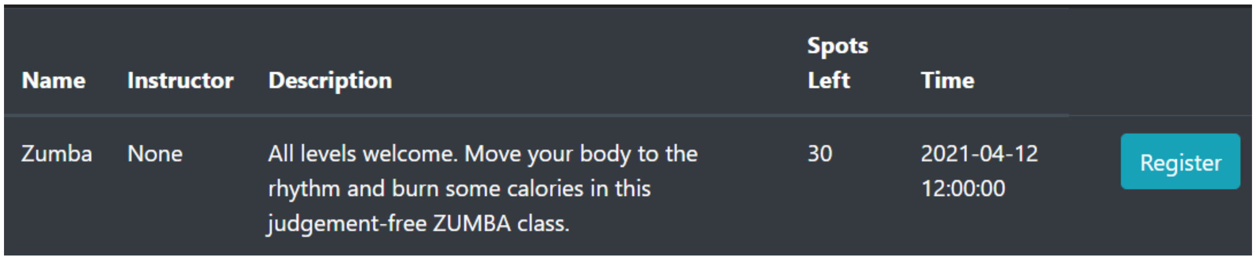
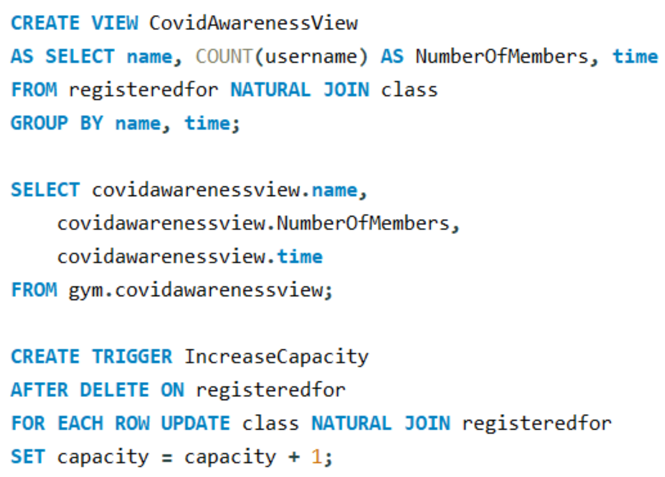


Figure 3.5.11: The search bar to look up classes to register for.



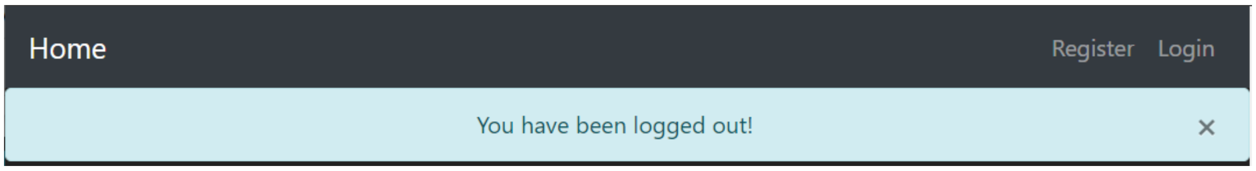
Name	Instructor	Description	Spots Left	Time	
Zumba	None	All levels welcome. Move your body to the rhythm and burn some calories in this judgement-free ZUMBA class.	30	2021-04-12 12:00:00	Register

Figure 3.5.12: A sample class that a member can register for.



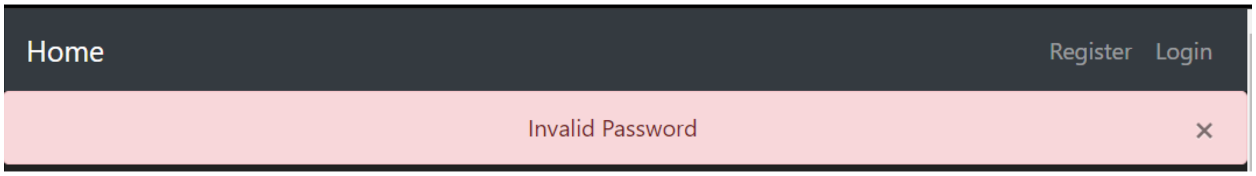
```
CREATE VIEW CovidAwarenessView
AS SELECT name, COUNT(username) AS NumberOfMembers, time
FROM registeredfor NATURAL JOIN class
GROUP BY name, time;

SELECT covidawarenessview.name,
    covidawarenessview.NumberOfMembers,
    covidawarenessview.time
FROM gym.covidawarenessview;

CREATE TRIGGER IncreaseCapacity
AFTER DELETE ON registeredfor
FOR EACH ROW UPDATE class NATURAL JOIN registeredfor
SET capacity = capacity + 1;
```

Figure 3.5.13: SQL statements to implement the use of views and triggers. These statements are referenced within the sql_commands.py file. CovidAwarenessView creates a view of the number of members registered per class, which is displayed within a 'COVID Awareness' tab. The IncreaseCapacity trigger ensures that the class size is updated to reflect the amount of active members registered. When a member is deleted, they are no longer registered for any classes in their schedule.



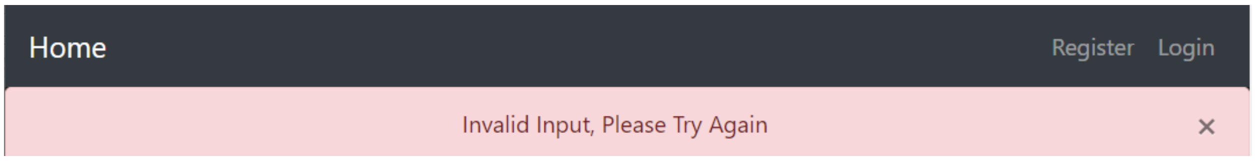
You have been logged out!

Figure 3.5.14: Notification that a user has successfully been logged out.



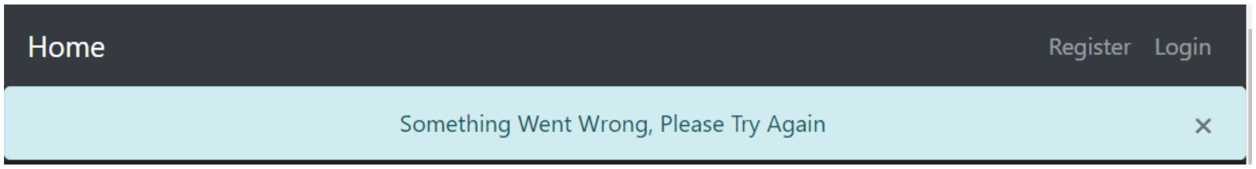
Invalid Password

Figure 3.5.15: Notification that a user has inputted the invalid information for the password field of the login form.



Invalid Input, Please Try Again

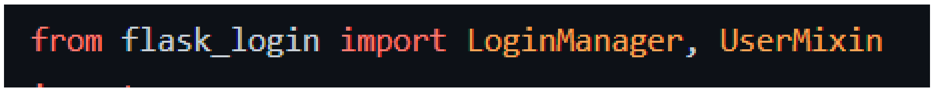
Figure 3.5.16: Notification of a failed login.



Something Went Wrong, Please Try Again

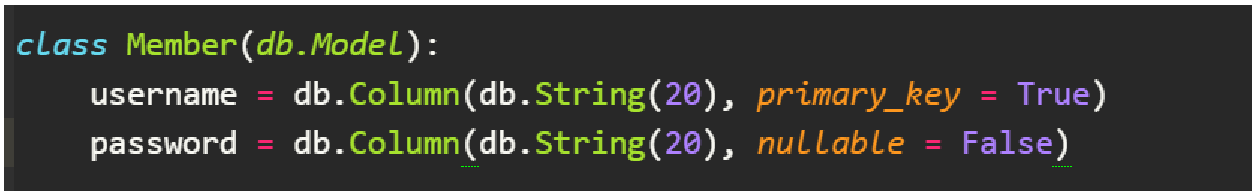
Figure 3.5.17: Notification of a failed registration.

3.6. Notes



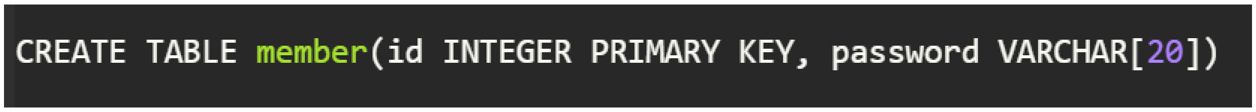
```
from flask_login import LoginManager, UserMixin
```

If this webapp is ever hosted, **USE LOGIN MANAGER AND USER MIXING.** The current implementation does not encrypt the data (no password encryption). This will replace **sessions.** However, if UserMixin is used, the way of querying the database would have to change to using Flask SQL syntax instead of Pure sql syntax.



```
class Member(db.Model):
    username = db.Column(db.String(20), primary_key = True)
    password = db.Column(db.String(20), nullable = False)
```

Figure 3.6.2: Example of using Flask SQL syntax.



```
CREATE TABLE member(id INTEGER PRIMARY KEY, password VARCHAR[20])
```

Figure 3.6.3: Example of Pure SQL syntax.

4. References

Flask Tutorial #5 - Sessions. (2019, November 5). [Video]. YouTube. https://www.youtube.com/watch?v=iIhAfX4iek0&t=143s. Information on flask sessions.

Flask Course - Python Web Application Development. (2021, March 10). [Video]. YouTube.

https://www.youtube.com/watch?v=Qr4QMBUPxWo&t=14350s. Information on flask setup.

J. (2021). *GitHub - jimdevops19/FlaskSeries*. GitHub.

https://github.com/jimdevops19/FlaskSeries. Used the CSS files from this repository.