

P3 MySQL with Python (Week 4 to 6)

Learning Outcomes

In this practical, students will learn how to integrate Flask-Python with MySQL database.

At the end of this practical lesson, students will be able to:

- Prepare database by running database scripts.
- Use **MySQL** to persist (store).
- Integrate MySQL with Python Flask
- Add a **Create User** function to the Web Application.
- Add a **Login** Page to perform authentication.
- Create controller routes and templates to display various views.

Instructions

You should have already installed the “MySQL 8.0 community server” based on the instruction from your **Database Management** module **IT2651**. If you have not done so, please follow the instructions from IT2651 before continuing with this lab. Some of the codes can be downloaded from BB.

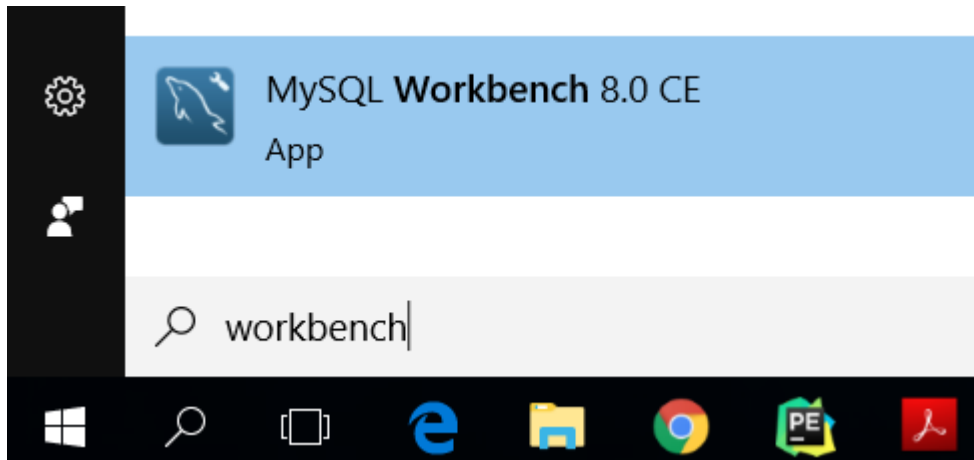
SUBMISSION : Please submit your completed solution into blackboard by end of week 6. You may improve your solution to include WTForms and improved form validation checks.

- Part 1
 - Create login.py (as Controller), login.html (as View), create database (mysql) and run login.py to display login.html.
 - Experience the use of MySQL and DB scripts to populate database, tables and data.
- Part 2
 - Write database codes to query from MySQL server.
 - Create and manage session during login authentication.
 - Invalidate session upon user logout.
- Part 3
 - Create registration form for user registration
 - Insert data into Database
 - ***Optional challenge: add in form validation and check for duplicate account in database.***
- Part 4
 - Create Homepage Page for registered user.
- Part 5
 - Create Profile page for registered user.

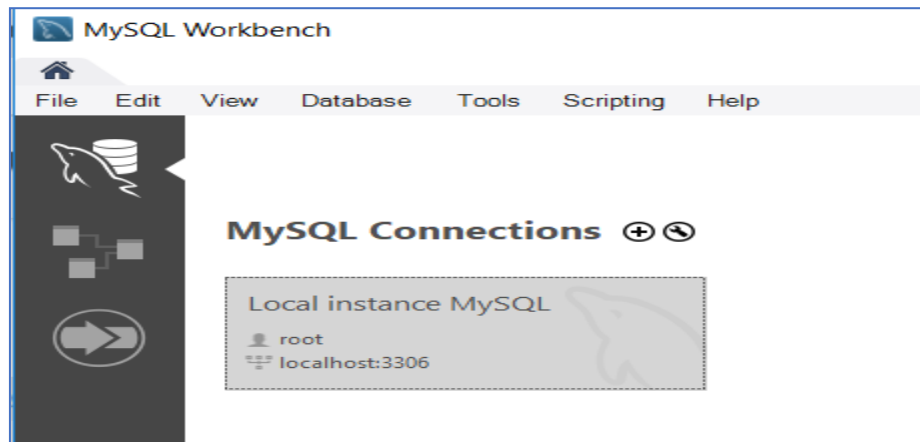
PART 1

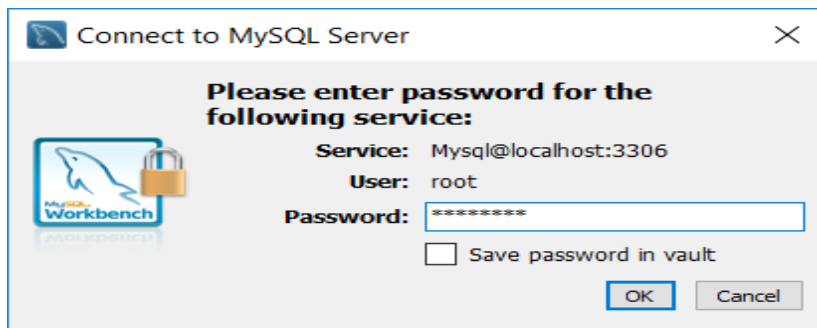
1. Creating the Database (MySQL) : pythonlogin

Start the MySQL workbench 8.0 CE.



Click on the “Local Instance MySQL”. If there isn’t any ready local instance, click on the “+” sign to create one. During the installation of your database, you should have set a password for the user “root”. Create one if you have not done so. (Other username is fine as well).





Load and run the following database scripts into workbench under a Query window:

This script will create a database name **“pythonlogin”** and a table name **“accounts”**.

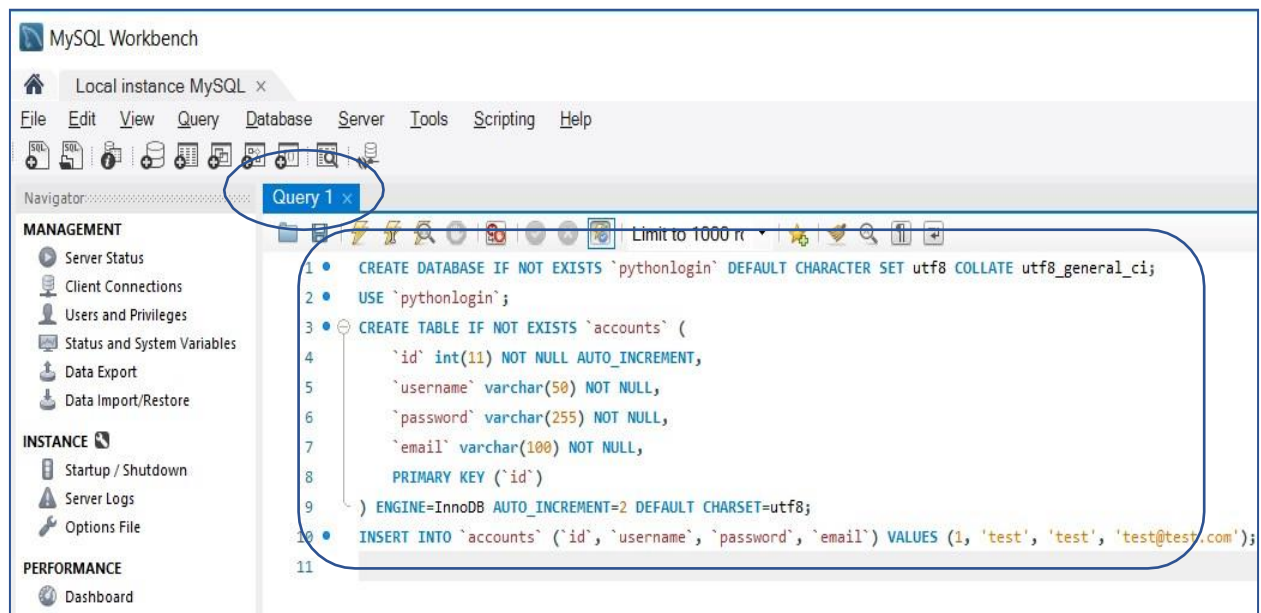
```
CREATE DATABASE IF NOT EXISTS `pythonlogin` DEFAULT CHARACTER SET utf8 COLLATE  
utf8_general_ci;
```

```
USE `pythonlogin`;
```

```
CREATE TABLE IF NOT EXISTS `accounts` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `username` varchar(50) NOT NULL,  
    `password` varchar(255) NOT NULL,  
    `email` varchar(100) NOT NULL,  
    PRIMARY KEY (`id`)
```

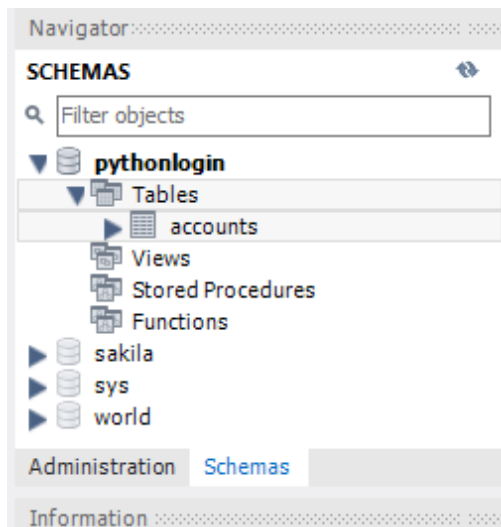
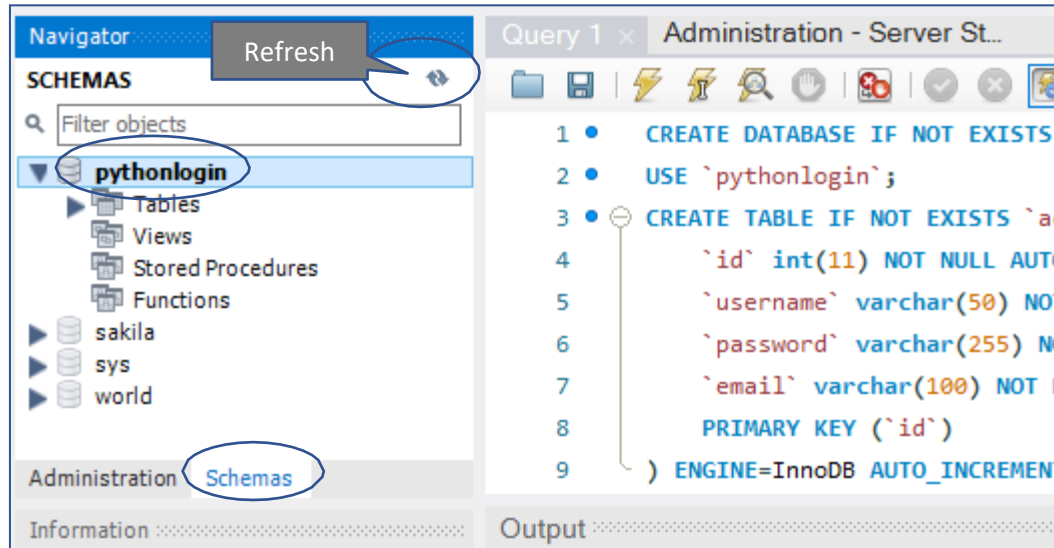
```
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

```
INSERT INTO `accounts` (`id`, `username`, `password`, `email`) VALUES (1, 'test', 'test', 'test@test.com');
```



A schema with the name “pythonlogin” should be created after running the db script. Click on the Refresh button if the schema did not appear.

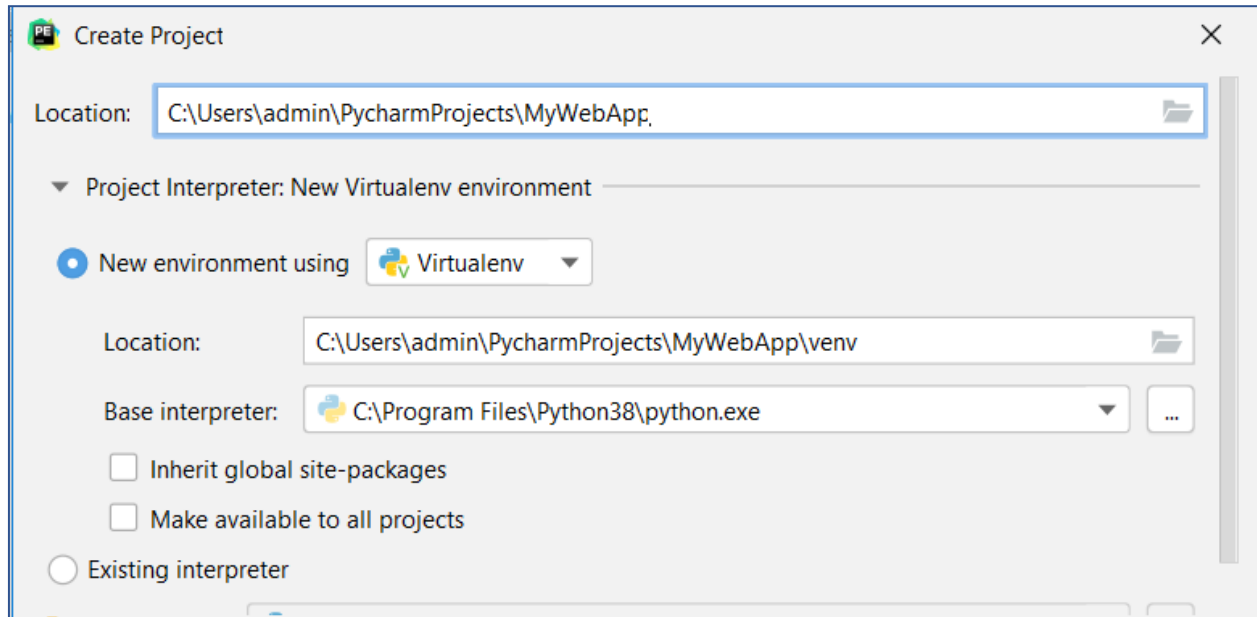
A table by the name “accounts” get created as well. Expand the “Tables” sub menu.



2. Creating the Project : MyWebApp

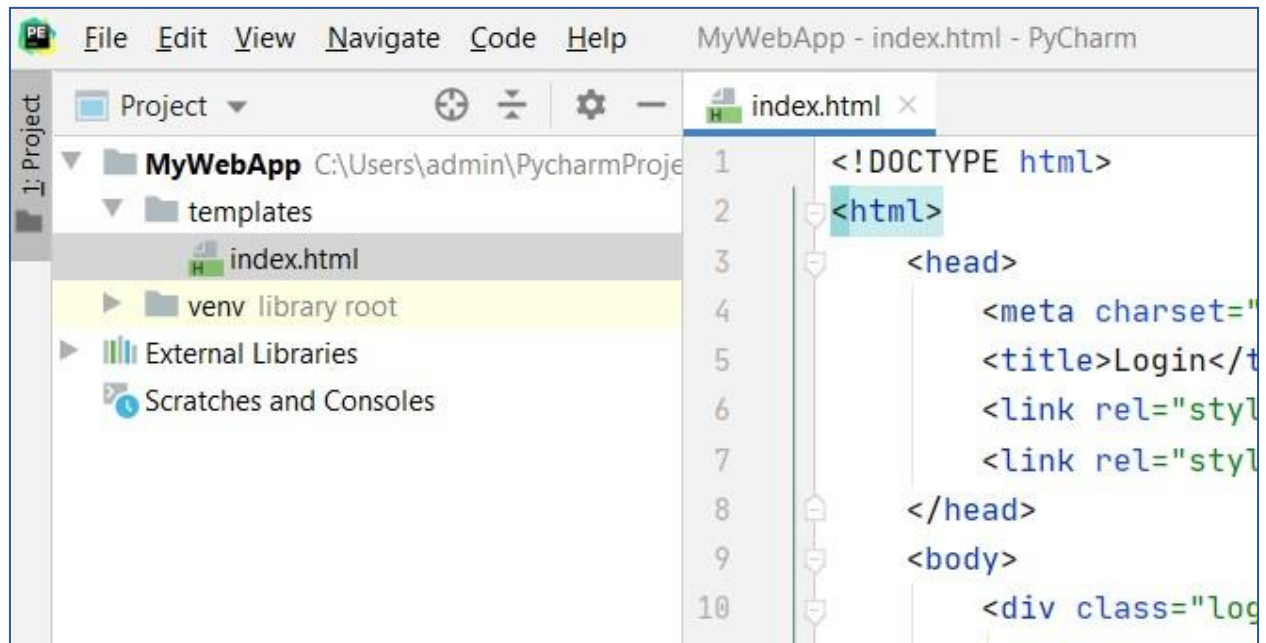
Create a new project :

File  Create new project  "MyWebApp" or any name.



3. Creating the View (MVC) : login.html

Create a folder “templates”. Create an index page with login form (index.html) inside the “templates” folder.



Codes for index.html

```
<!DOCTYPE html>

<html>
<head>
    <meta charset="utf-8">
    <title>Login</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css">
</head>
<body>
    <div class="login">
        <h1>Login</h1>
        <div class="links">
            <a href="{{ url_for('login') }}" class="active">Login</a>
            <a href="#">Register</a>
        </div>

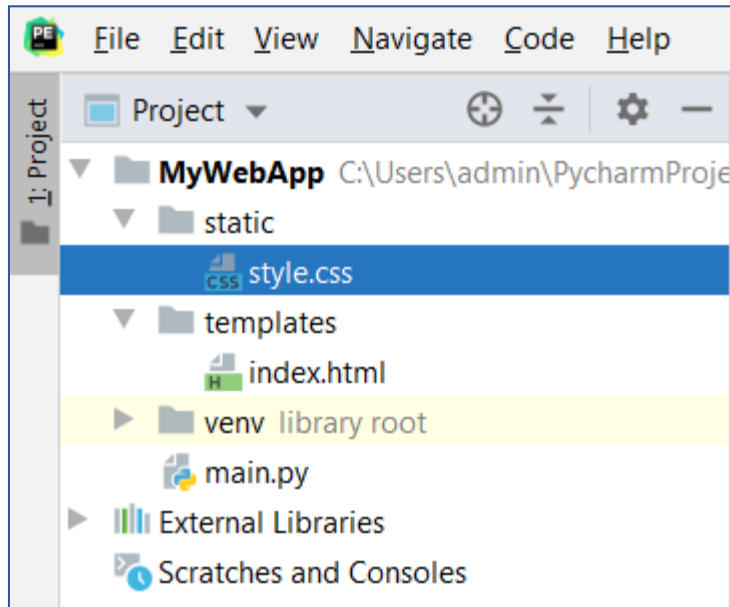
        <form action="{{ url_for('login') }}" method="post">
            <label for="username"><i class="fas fa-user"></i></label>
            <input type="text" name="username" placeholder="Username" id="username"
required>
            <label for="password">|
            <i class="fas fa-lock"></i></label>

            <input type="password" name="password" placeholder="Password" id="password"
required><div class="msg">{{ msg }}</div>
            <input type="submit" value="Login">
        </form>

    </div>
</body>
</html>
```

- The login template above was created with the input fields: username and password, and the form's method is set to post, this will send the form data to our Python Flask server using a POST request.

4. Creating the CSS (Style) : style.css



- Create a folder “Static” and add a new file style.css.
- Download style.css from blackboard, copy-paste the codes into style.css

5. Creating the Login Controller (MVC) : login.py

The first thing we need to do is import the packages we're are going to use.

- **pip install flask:**
 - Successfully installed Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2 flask-1.1.2 itsdangerous-1.1.0
- **pip install flask-mysqldb:**
 - Successfully installed flask-mysqldb-0.2.0 mysqlclient-1.4.6

Create Login.py file and add the following codes:

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_mysql import MySQL
import MySQLdb.cursors
import re

app = Flask(__name__)

# Change this to your secret key (can be anything, it's for extra protection)

app.secret_key = 'your secret key'

# Enter your database connection details below
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'mysql'
app.config['MYSQL_DB'] = 'pythonlogin'

# Initialize MySQL
mysql = MySQL(app)

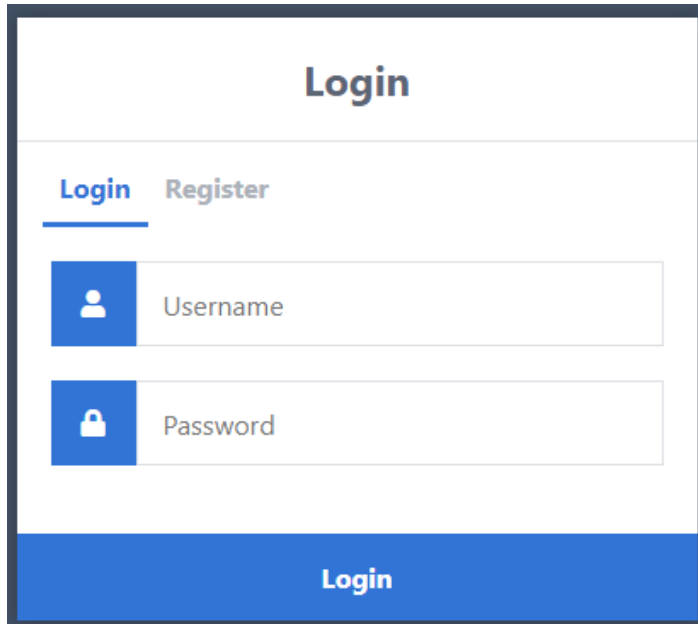
# http://localhost:5000/MyWebApp/ - this will be the login page, we need to use both GET and POST #requests
@app.route('/MyWebApp/', methods=['GET', 'POST'])
def login():
    return render_template('index.html', msg='')

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

Note : Please ensure that you **change the MySQL details to your own details** and change the secret key.

At this point, it is good to check that your app is working by running login.py.

If you navigate to **<http://localhost:5000/MyWebApp/>** in our web browser it will look like the following:



Login

Login Register

Login

If you are able to see this screen, it means that your style.css, main controller routing and your login view is working properly. This test does not mean that your Database connection is successful.

----- End of Part 1 -----

PART 2

6. Adding Authentication codes

Go back to our Login.py file and add the authentication code to our route function that was created.

```
def login():  
    # Output message if something goes wrong...  
  
    msg = "  
  
    # Check if "username" and "password" POST requests exist (user submitted form)  
  
    if request.method == 'POST' and 'username' in request.form and 'password' in request.form:  
        # Create variables for easy access  
  
        username = request.form['username']  
  
        password = request.form['password']
```

- In the above codes, the **IF** statement was used to check if the requested method is POST and check if the username and password exist in the form request, if they both exist the username and password variables will be created.
- Continue to add the codes below to the login.py un the IF statement

```
# Check if account exists using MySQL  
  
cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)  
  
cursor.execute('SELECT * FROM accounts WHERE username = %s AND password = %s', (username,  
password,))  
  
# Fetch one record and return result  
  
account = cursor.fetchone()
```

- The code above will select the account from our **accounts** table in our MySQL database with the username and password that the user entered in the login form.
- The **Cursor** class returns rows as dictionaries and stores the result set in the client.
- **Fetchone ()** method retrieves the next row of a query result set and returns a single sequence, or None if no more rows are available. By default, the returned tuple consists of data returned by the MySQL server, converted to Python objects.

- Continue to add codes below IF statement

```
# Create session data, we can access this data in other routes

session['loggedin'] = True

session['id'] = account['id']

session['username'] = account['username']

# Redirect to home page

return 'Logged in successfully!'

else:

    # Account doesn't exist or username/password incorrect

    msg = 'Incorrect username/password!'

# Show the login form with message (if any)
return render_template('index.html', msg="")
```

- The above will check if the account exists, if it does new session variables will be created and used for keeping track of the user's state (e.g remembered for the user) in order to determine if the user is logged in or not.
- Session variables act like cookies, they will be remembered on the server instead of the user's browser, only the session ID will be stored as a browser cookie.
- If the account doesn't exist we can simply show an error on the login form.

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

- Don't forget to issue the app.run() command regardless of the routes.

Your final login () ROUTE should look something like the screen below:

```
def login():
    # Output message if something goes wrong...
    msg = ''
    # Check if "username" and "password" POST requests exist (user submitted form)
    if request.method == 'POST' and 'username' in request.form and 'password' in request.form:
        # Create variables for easy access
        username = request.form['username']
        password = request.form['password']

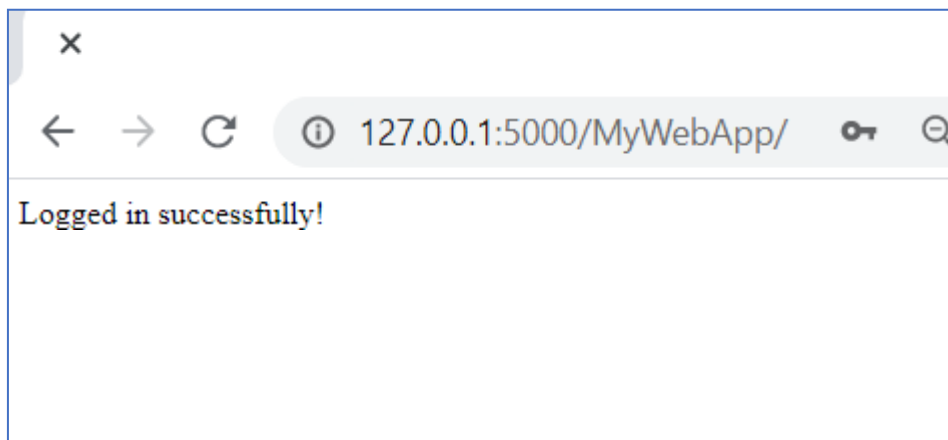
        # Check if account exists using MySQL
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM accounts WHERE username = %s AND password = %s', (username, password,))
        # Fetch one record and return result
        account = cursor.fetchone()

        # If account exists in accounts table in our database
        if account:
            # Create session data, we can access this data in other routes
            session['loggedin'] = True
            session['id'] = account['id']
            session['username'] = account['username']
            # Redirect to home page
            return 'Logged in successfully!'
        else:
            # Account doesnt exist or username/password incorrect
            msg = 'Incorrect username/password!'

    # Show the login form with message (if any)
    return render_template('index.html', msg=msg)

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

- To make sure everything is working correctly navigate to <http://localhost:5000/MyWebApp/> and input "test" in both the username and password fields, and then click the Login button, you should receive a message that outputs "Logged in successfully!".



7. Manage Logout session

- Whenever a user logout from the system, the session variables that were created when the user logs in should be deleted.
- Add a new route : /MyWebApp/logout
- Add the following code to the login.py file:

```
# http://localhost:5000/MyWebApp/logout - this will be the logout page
```

```
@app.route('/MyWebApp/logout')
```

```
def logout():
```

```
    # Remove session data, this will log the user out
```

```
    session.pop('loggedin', None)
```

```
    session.pop('id', None)
```

```
    session.pop('username', None)
```

```
    # Redirect to login page
```

```
    return redirect(url_for('login'))
```

- Try to login to the system again. This time round, after the browser display the message “Login is successful!”
- You can either create a button to issue the command to logout or just type the url :
<http://127.0.0.1:5000/MyWebApp/logout>
- The system should bring you back to the login page.

----- End of Part 2 -----

PART 3

8. Register User

We need a registration system so users can register on your web app. In this section we are going to create a new register route and create the registration template.

- Create register.html and put it under templates section.
- The codes for register.html can be downloaded from the blackboard.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Register</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css">
  </head>
  <body>
    <div class="register">
      <h1>Register</h1>
      <div class="links">
        <a href="{{ url_for('login') }}">Login</a>
        <a href="{{ url_for('register') }}" class="active">Register</a>
      </div>
      <form action="{{ url_for('register') }}" method="post" autocomplete="off">
        <label for="username">
          <i class="fas fa-user"></i>
        </label>
        <input type="text" name="username" placeholder="Username" id="username" required>
        <label for="password">
          <i class="fas fa-lock"></i>
        </label>
        <input type="password" name="password" placeholder="Password" id="password" required>
        <label for="email">
          <i class="fas fa-envelope"></i>
        </label>
        <input type="email" name="email" placeholder="Email" id="email" required>
        <div class="msg">{{ msg }}</div>
        <input type="submit" value="Register">
      </form>
    </div>
  </body>
</html>

```

- Create register.html and put it under templates section.
- The codes for register.html can be downloaded from the blackboard.

- Create the "register" route to handle POST request and insert a new account into the accounts table.
- Validate the data in all the fields.
- Go back to the Login.py file and add a register () route

http://localhost:5000/MyWebApp/register - this will be the registration page, we need to use both GET and POST requests

```
@app.route('/MyWebApp/register', methods=['GET', 'POST'])
```

```
def register():
```

```
    # Output message if something goes wrong...
```

```
    msg = "
```

```
    # Check if "username", "password" and "email" POST requests exist (user submitted form)
```

```
    if request.method == 'POST' and 'username' in request.form and 'password' in request.form and 'email' in request.form:
```

```
        # Create variables for easy access
```

```
        username = request.form['username']
```

```
        password = request.form['password']
```

```
        email = request.form['email']
```

```
    elif request.method == 'POST':
```

```
        # Form is empty... (no POST data)
```

```
        msg = 'Please fill out the form!'
```

```
    # Show registration form with message (if any)
```

```
    return render_template('register.html', msg=msg)
```

- To insert data into the database
- Add in the following codes after : **email = request.form['email']**

Optional challenge : Check for duplicate acct and perform form validation

Account doesnt exists and the form data is valid, now insert new account into accounts table

```
cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
```

```
cursor.execute('INSERT INTO accounts VALUES (NULL, %s, %s, %s)', (username, password, email,))
```

```
mysql.connection.commit()
```

```
msg = 'You have successfully registered!'
```







- Update the index.html to reflect the register link from

```
<a href="#">Register</a>
```

To

```
<a href="{{ url_for('register') }}">Register</a>
```

- Test the registration feature out by using the url : <http://localhost:5000/MyWebApp/register> or click on the register button.

| Register | Register |
|--|---|
| <p>Login <u>Register</u></p> <p> 0</p> <p> .</p> <p> 0@0</p> <p>Register</p> | <p>Login <u>Register</u></p> <p> Username</p> <p> Password</p> <p> Email</p> <p>You have successfully registered!</p> <p>Register</p> |

- Optional challenges:
 - Check for duplicate user id before inserting data into the database.
 - Make changes to the project and perform your own form validation.
 - Valid email address
 - Username can only contain alphabets and numbers
 - Password must be at least 8 char in length.

----- End of Part 3 -----

PART 4

9. Create a Home and profile page for Registered user

In this section, we will create a home page for logged-in users only.

- Edit Login.py and add the following:

```
# http://localhost:5000/MyWebApp/home - this will be the home page, only
accessible for loggedin users
@app.route('/MyWebApp/home')
def home():
    # Check if user is loggedin
    if 'loggedin' in session:
        # User is loggedin show them the home page
        return render_template('home.html', username=session['username'])
    # User is not loggedin redirect to login page
    return redirect(url_for('login'))
```

- Create a home.html inside templates folder. Add the codes below into home.html.

```
{% extends 'layout.html' %}
{% block title %}Home{% endblock %}
{% block content %}
<h2>Home Page</h2>
<p>Welcome back, {{ username }}!</p>
{% endblock %}
```

- Create a layout.html to support home.html. Put it inside templates folder.
- NOTE : **url_for('profile')** will generate an error if you have yet to create the route. If you wish to test the home () then either you change the url to home or disable it first.

```

<!DOCTYPE html>

<html>

  <head>
    <meta charset="utf-8">

    <title>{% block title %}{% endblock %}</title>

    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css">
  </head>
  <body class="loggedin">
    <nav class="navtop">
      <div>

        <h1>Website Title</h1>

        <a href="{{ url_for('home') }}"><i class="fas fa-home"></i>Home</a>
        <a href="{{ url_for('profile') }}"><i class="fas fa-user-circle"></i>Profile</a>
        <a href="{{ url_for('logout') }}"><i class="fas fa-sign-out-alt"></i>Logout</a>

      </div>
    </nav>
    <div class="content">

      {% block content %}{% endblock %}

    </div>
  </body>
</html>

```

- Edit Login.py and add the following:
- Change **login() route** function

From :

return 'Logged in successfully!'

To:

return redirect(url_for('home'))

- The user will now be redirected to the home page when they log in.
- You may proceed with the next section first on creation of a profile () route before you go ahead to test your webapp for home () route.

----- End of Part 4 -----

PART 5

10. Create a profile page for Registered user

In this section, we will create a profile page for logged-in users only. Login users will be able view their username and email.

- Edit Login.py and add a profile () route:

```
# http://localhost:5000/MyWebApp/profile - this will be the profile page, only
accessible for loggedin users
@app.route('/MyWebApp/profile')
def profile():
    # Check if user is loggedin
    if 'loggedin' in session:
        # We need all the account info for the user so we can display it on the
        profile page
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM accounts WHERE id = %s', (session['id'],))
        account = cursor.fetchone()
        # Show the profile page with account info
        return render_template('profile.html', account=account)
    # User is not loggedin redirect to login page
    return redirect(url_for('login'))
```

- Continue to next page to create profile.html page

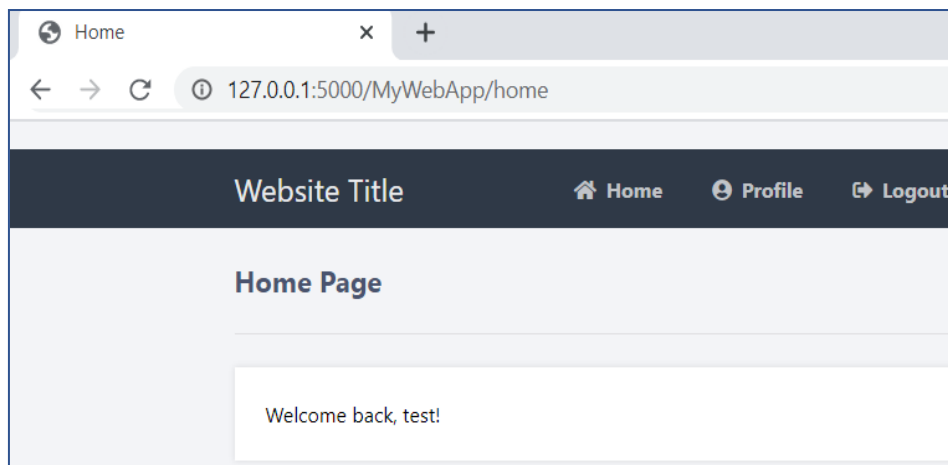
- Create the profile.html page and extend layout.html

```
{% extends 'layout.html' %}

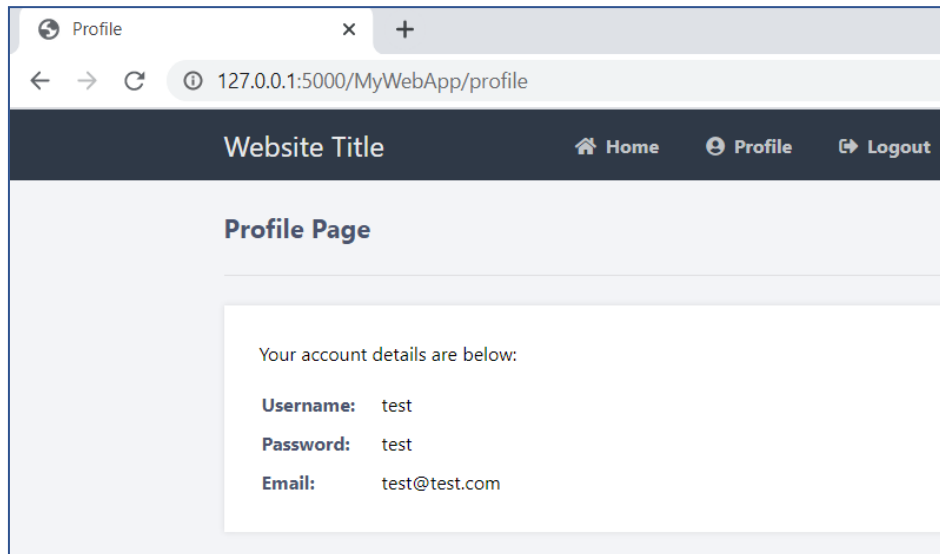
{% block title %} Profile {% endblock %}

{% block content %}
<h2>Profile Page</h2>
<div>
  <p>Your account details are below:</p>
  <table>
    <tr>
      <td>Username:</td>
      <td>{{ account['username'] }}</td>
    </tr>
    <tr>
      <td>Email:</td>
      <td>{{ account['email'] }}</td>
    </tr>
  </table>
</div>
{% endblock %}
```

- Test both home () and profile () routes if they are working fine.
- Home.html page should look like the screen below



- Profile.html page



----- End of Part lab -----