

Contents

1. [Getting Started](#)
 1. [What You Will Learn in this Tutorial](#)
 2. [Requirements](#)
 3. [File Structure & Setup](#)
2. [Creating the Database and setting-up Tables](#)
3. [Creating the Stylesheet \(CSS3\)](#)
4. [Creating the Login System](#)
 1. [Creating the Login Template](#)
 2. [Authenticating Users with Python](#)
 3. [Creating the Logout Script](#)
5. [Creating the Registration System](#)
 1. [Creating the Registration Template](#)
 2. [Registering Users with Python](#)
6. [Creating the Home Page](#)
7. [Creating the Profile Page](#)

1. Getting Started

There are a few steps required before creating and testing the Python login and registration system named **User Web Application (UWA)**. You need to setup a test environment. The test environment should have a software IDE that has access to a database environment, application components, and development language support. To accomplish this starts with downloading and installing Python along with the packages that the UWA app will depend on. If you already have Python installed, confirm that the MySQL database environment is installed too. Use the **Test Environment Setup** document to certify that your test environment is complete and ready for application development and testing before proceeding with this document.

All the files are available for you to download the UWA application from GITHUB. But you can use the instructions below to step through a build application approach that will inform you of the Python code, CSS and HTML assets that create the UWA. If you don't have time, you can take the short route by downloading the files from [here at GITHUB](#).

1.1. What You Will Learn in this Tutorial

- **Form Design** — Design a login and registration form with HTML5 and CSS3.
- **Templates** — Create Flask templates with HTML and Python.
- **Basic Validation** — Validating form data that is sent to the server (username, password, and email).
- **Session Management** — Initialize sessions and store retrieved database results.
- **MySQL Queries** — Select and insert records from/in your database table.
- **Routes** — Routing will enable you to associate the URLs with the functions that you will create.

1.2. Requirements

- See the **Test Environment Setup** document for details.

1.3. File Structure & Setup

You need to create your project directory and files. You can create the directory anywhere on your computer if Python can access it. Create the directories and files below.

File Structure

```
\-- pythonlogin
  |-- main.py
  \-- static
    |-- style.css
  \-- templates
    |-- index.html
    |-- register.html
    |-- home.html
    |-- profile.html
    |-- layout.html
```

Each file will contain the following:

- As you add code to these files, the first line is a label like CSS, PYTHON, SQL, etcetera. **Do not copy that line into your files.**
- **main.py** — This will be the main Python file, which will contain the Python executable code (Routes, MySQL connection, validation, etc.).
- **index.html** — The login form template created with HTML5 and CSS3.
- **register.html** — The registration form template created with HTML5 and CSS3.
- **home.html** — The home template which is restricted to logged-in users.
- **profile.html** — The profile template which is restricted to logged-in users. The user's details will be populated on this page.
- **layout.html** — The layout template for the home and profile templates.
- **style.css** — The CSS3 stylesheet for our login and registration system.

The below instruction will start your web server (Windows):

- Make sure your **MySQL server is up and running**. It should have automatically started if you installed it via the installer. In addition, ensure MySQL is running on port 3306 otherwise you'll encounter connection errors.
- Open **Command Prompt** or Terminal in your IDE and navigate to your project directory. You can do this with the following command statement:
 - `cd c:\your_project_folder_destination` on Windows.
- Run command: `set FLASK_APP=main.py`
- Run command: `set FLASK_DEBUG=1`
- Run command: `flask run`

Alternative methods follow:

- Place the run commands above into a single BAT file and execute.
- Use the arrow pointing to the “IF” statement at the end of the main.py code. It should allow you to RUN the APP.

2. Creating the Database and setting-up Tables

- **MySQL Workbench is a GUI app for managing databases.** See the **Test Environment Setup** document for details.

3. Creating the Stylesheet (CSS3)

Edit the **style.css** file and add the following:

```
CSS
* {
    box-sizing: border-box;
    font-family: -apple-system, BlinkMacSystemFont, "segoe ui", roboto,
oxygen, ubuntu, cantarell, "fira sans", "droid sans", "helvetica neue", Arial,
sans-serif;
    font-size: 16px;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
body {
    background-color: #435165;
    margin: 0;
}
.login, .register {
    width: 400px;
    background-color: #ffffff;
    box-shadow: 0 0 9px 0 rgba(0, 0, 0, 0.3);
    margin: 100px auto;
}
.login h1, .register h1 {
    text-align: center;
    color: #5b6574;
    font-size: 24px;
    padding: 20px 0 20px 0;
    border-bottom: 1px solid #dee0e4;
}
.login .links, .register .links {
    display: flex;
    padding: 0 15px;
}
.login .links a, .register .links a {
    color: #adb2ba;
    text-decoration: none;
    display: inline-flex;
    padding: 0 10px 10px 10px;
    font-weight: bold;
}
.login .links a:hover, .register .links a:hover {
    color: #9da3ac;
}
.login .links a.active, .register .links a.active {
    border-bottom: 3px solid #3274d6;
    color: #3274d6;
}
```

```

}
.login form, .register form {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  padding-top: 20px;
}
.login form label, .register form label {
  display: flex;
  justify-content: center;
  align-items: center;
  width: 50px;
  height: 50px;
  background-color: #3274d6;
  color: #ffffff;
}
.login form input[type="password"], .login form input[type="text"], .login form
input[type="email"], .register form input[type="password"], .register form
input[type="text"], .register form input[type="email"] {
  width: 310px;
  height: 50px;
  border: 1px solid #dee0e4;
  margin-bottom: 20px;
  padding: 0 15px;
}
.login form input[type="submit"], .register form input[type="submit"] {
  width: 100%;
  padding: 15px;
  margin-top: 20px;
  background-color: #3274d6;
  border: 0;
  cursor: pointer;
  font-weight: bold;
  color: #ffffff;
  transition: background-color 0.2s;
}
.login form input[type="submit"]:hover, .register form input[type="submit"]:hover
{
  background-color: #2868c7;
  transition: background-color 0.2s;
}
.navtop {
  background-color: #2f3947;
  height: 60px;
  width: 100%;
  border: 0;
}
.navtop div {
  display: flex;
  margin: 0 auto;
  width: 1000px;
  height: 100%;

```

```

}
.navtop div h1, .navtop div a {
  display: inline-flex;
  align-items: center;
}
.navtop div h1 {
  flex: 1;
  font-size: 24px;
  padding: 0;
  margin: 0;
  color: #eaebed;
  font-weight: normal;
}
.navtop div a {
  padding: 0 20px;
  text-decoration: none;
  color: #c1c4c8;
  font-weight: bold;
}
.navtop div a i {
  padding: 2px 8px 0 0;
}
.navtop div a:hover {
  color: #eaebed;
}
body.loggedin {
  background-color: #f3f4f7;
}
.content {
  width: 1000px;
  margin: 0 auto;
}
.content h2 {
  margin: 0;
  padding: 25px 0;
  font-size: 22px;
  border-bottom: 1px solid #e0e0e3;
  color: #4a536e;
}
.content > p, .content > div {
  box-shadow: 0 0 5px 0 rgba(0, 0, 0, 0.1);
  margin: 25px 0;
  padding: 25px;
  background-color: #fff;
}
.content > p table td, .content > div table td {
  padding: 5px;
}
.content > p table td:first-child, .content > div table td:first-child {
  font-weight: bold;
  color: #4a536e;
  padding-right: 15px;
}

```

```

}
.content > div p {
    padding: 5px;
    margin: 0 0 10px 0;
}

```

The above stylesheet will structure your pages and provide an innovative experience for users. Feel free to customize the stylesheet (change text color, font sizes, content width, etc.).

4. Creating the Login System

You will need to modify the Python module! What will be done in this section is create the login template, connect to our MySQL database, implement login authentication, and define session variables.

The first thing you need to do is import the packages to be used. Edit the **main.py** file, and add the following:

```

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

```

Now that you have imported all the packages to be utilized, create the MySQL and app-related variables, and configure the MySQL connection details.

Add and modify the following:

```

Python
app = Flask(__name__)

# Change this to your secret key (it can be anything, it's for extra protection)
app.secret_key = "secretUWA"

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

```

```
# Intialize MySQL
mysql = MySQL(app)
```

Ensure to configure the MySQL variables to reflect your MySQL details.

Now, you can proceed to create the login page. To do that, create a new route. Routes will enable you to associate your functions with a particular URL.

Add after:

```
Python
# http://localhost:5000/ - the following will be our login page, which will use
both GET and POST requests
@app.route('/', methods=['GET', 'POST'])
def login():
    # Output message if something goes wrong...
    msg = ''
    return render_template('index.html', msg='')

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

4.1 Creating the Login Template

Edit the [index.html](#) file and add:

```
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">
```



```

class="active">Login</a>
    <a href="{{ url_for('login') }}"
    <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>

```

As you can see with the login template, you created the form along with the input fields: `username` and `password`. The form's method is set to `post` which determines the type of request you want to send to your server. You will be using a POST request to send the form data.

```

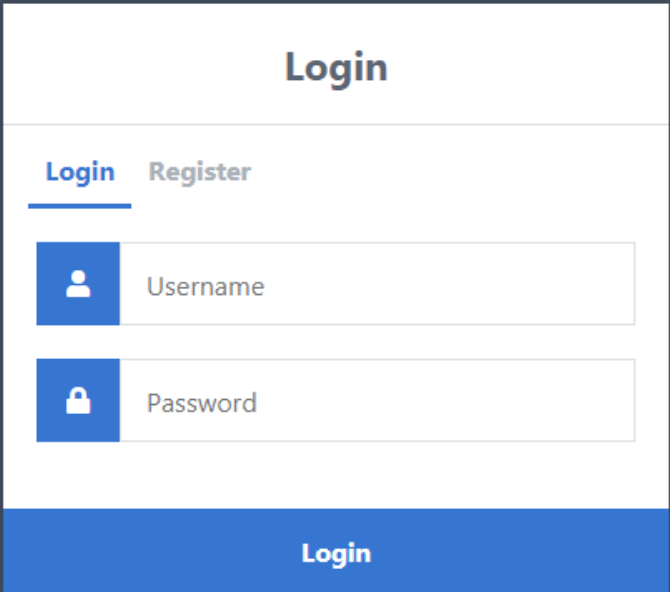
28 ▶ if __name__ == '__main__':
29    = app.run()

```

You should see an arrow on the above shown line in your code. If you hover over the arrow there should be a “RUN APP” selection revealed. Click the arrow and then click that option to execute your main.py module.

If you navigate to <http://localhost:5000> in your web browser or click the localhost link in your IDE terminal window, your browser will look like the following:

```
http://localhost:5000
```



The image shows a web form titled "Login". At the top, the word "Login" is displayed in a large, bold, dark font. Below the title, there are two tabs: "Login" (which is active and underlined) and "Register". The form contains two input fields: "Username" with a person icon and "Password" with a lock icon. At the bottom of the form is a large blue button labeled "Login".

If you click the **Login** button, nothing will happen or will return an error. That's because we haven't implemented the code that handles the POST request.

At this point, if you were able to see a page of the application without error, you have confirmed all components are working together. Proceed to the next steps with confidence.

4.2 Authenticating Users with Python

Now you need to go back to your version of **main.py** file and add the authentication code to your route method implemented. But remove the existing "login()" function. It should be replaced by new "login()" function.

After the code entered so far, please insert the following before the last “IF” statement and make sure the “IF” statement remains the last statement in your code:

```
def login():
    # Output a message if something goes wrong...
    msg = ''
```

Insert the following code:

```
Python
# 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']
    # Retrieve the hashed password
    hash = password + app.secret_key
    hash = hashlib.sha1(hash.encode())
    password = hash.hexdigest()
```

With the code above, it used an **if** statement to check if the requested method is **POST** and check if the **username** and **password** variables exist in the form request. If they both exist, the username and password variables will be created, which will be associated with the form variables.

In addition, the code is leveraging the [hashlib module](#) to create a hashed password, which is good practice and will help prevent the original password from being exposed.

Insert this code next:

```
Python
# 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 the result
account = cursor.fetchone()
```

The code above will execute a SQL query that will retrieve the account details from your **accounts** table in your MySQL database.

The **username** and **password** variables are associated with this query, as that is what is used to find the account.

Now, insert this code next:

```
Python
# If account exists in accounts table in your 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 does not exist or username/password incorrect
    msg = 'Incorrect username/password!'
```

The code above will determine if the account exists. If it does, the session variables are declared. These session variables will be remembered for the user as they will be used to determine whether the user is logged in or not.

Session variables basically act like browser cookies. They are stored on the server as opposed to the user's browser.

If the account doesn't exist, simply output the error on the login form.

Your login route should look like the following:

```
Python
@app.route('/pythonlogin/', methods=['GET', 'POST'])
def login():
    # Output a 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
    msg = 'Logged in successfully!'
    return render_template('index.html', msg=msg)
else:
    # Account does not exist or username/password incorrect
    msg = 'Incorrect username/password!'
# Show the login form with message (if any)
return render_template('index.html', msg=msg)

```

To make sure everything is working correctly, navigate to <http://localhost:5000> 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!".

4.3 Creating the Logout Script

For a user to logout, all you must do is remove the session variables that were created when the user logged in.

Add the following code to the **main.py** file:

```

Python
# http://localhost:5000/python/logout - this will be the logout page
@app.route('/pythonlogin/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'))

```

The above code will remove each session variable associated with the user. Without these session variables, the user cannot be logged in. Subsequently, the user is redirected to the login page.

You can logout by navigating to the following

URL: <http://localhost:5000/pythonlogin/logout>

5. Creating the Registration System

You need a registration system that users can use to register on your app. What you will do in this section is create a new register route and create the registration template, along with the registration form, which will consist of input fields, submit button, etc.

5.1 Creating the Registration Template

Edit the [register.html](#) file and add:

```
HTML
<!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>
        <input type="text" name="role" placeholder="User
Role (Admin/Member)" id="role" required>
        <label for="role">
            <i class="fas fa-lock"></i>
        </label>
        <div class="msg">{{ msg }}</div>
        <input type="submit" value="Register">
    </form>
</div>
</body>
</html>

```

The HTML template above will be used to register users. It's identical to the login template but also includes the **Email** input field.

The form's action attribute is associated with the **"register"** route, as you'll use this route to handle the POST request.

5.2 Registering Users with Python

Now that you have your template created, you can proceed to create the **"register"** route, which will handle the POST request and insert a new account into your **accounts** table, but only if the submitted fields are valid.

Go back to the **main.py** file and insert the following before the last "IF" statement:

```

Python
# http://localhost:5000/pythinlogin/register - this will be the registration
page, we need to use both GET and POST requests
@app.route('/pythonlogin/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)

```

You created the "**register**" route and implement validation that will check if all the form fields exist. If they don't, then output a simple error.

Next, enter the following:

```

activation_code = "activated"
email = request.form['email']
role = request.form['role']

```

Then insert:

```

Python
# Check if account exists using MySQL
cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
cursor.execute('SELECT * FROM accounts WHERE username = %s', (username,))
account = cursor.fetchone()
# If account exists show error and validation checks
if account:
    msg = 'Account already exists!'
elif not re.match(r'^@]+@^[^@]+\.[^@]+', email):
    msg = 'Invalid email address!'
elif not re.match(r'[A-Za-z0-9]+', username):
    msg = 'Username must contain only characters and numbers!'
elif not username or not password or not email:
    msg = 'Please fill out the form!'
else:
    # Hash the password
    hash = password + app.secret_key
    hash = hashlib.sha1(hash.encode())
    hashed_password = hash.hexdigest()
    # Account doesn't exist, and the form data is valid, so insert the
new account into the accounts table

```



```
        cursor.execute(
            'INSERT INTO accounts (username, password, email, activation_code,
            role, ip) VALUES (%s, %s, %s, %s, %s, %s)',
            (username, hashed_password, email, activation_code, role,
            request.environ['REMOTE_ADDR'],))
        mysql.connection.commit()
        msg = 'You have successfully registered!'
        return render_template('register.html', msg=msg)
```

The above code will select an account with the submitted **username** and **password** fields. If the account doesn't exist, you can proceed to validate the input data. Validation will check if the submitted **email** is valid and check if the **username** contains only letters and numbers.

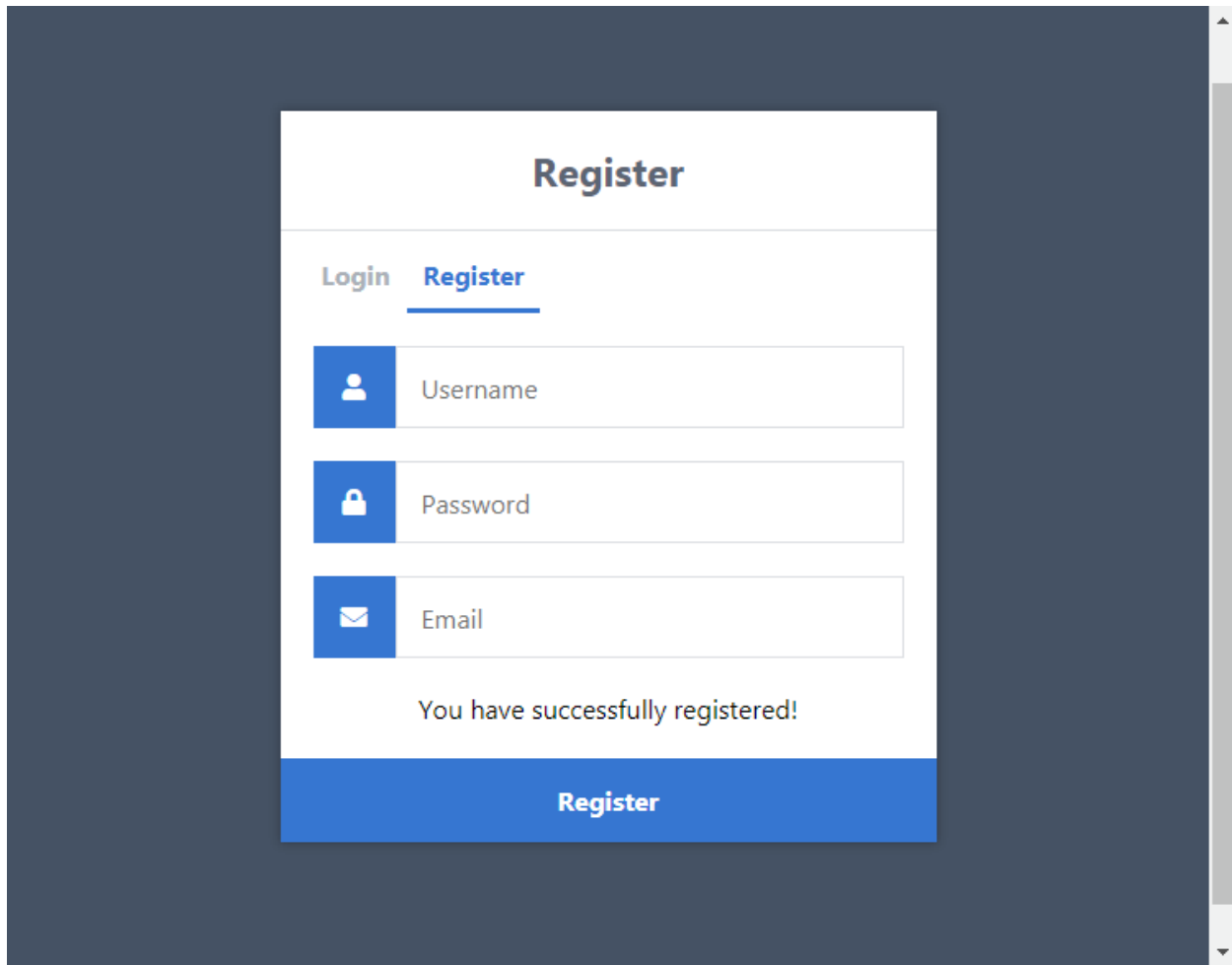
Coding Tip: The %s placeholder in the query string will help prevent SQL injection, as their bound values will be escaped before insertion.

Subsequently, the code will insert a new account into your **accounts** table.

To test that it is working correctly, navigate to <http://localhost:5000/pythonlogin/register> and fill out the form and click the **Register** button.

<http://localhost:5000/pythonlogin/register>

You should receive the following response:



I have added an extra field to support entry of user **role** which can either be “Member” or “Admin”. Think about what this means when you think about verification and validation. HINT:

Now you can go back to your [index.html](#) file and change this line:

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

To:

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

Users can now register and log in to your app. Next, you'll create a basic home page for logged-in users.

6. Creating the Home Page

The home page will be restricted to logged-in users only. Non-registered users cannot access this page. You can adapt this page and create more pages.

Edit the **main.py** file and add the following:

```
Python
# http://localhost:5000/pythonlogin/home - this will be the home page, only
# accessible for logged in users
@app.route('/pythonlogin/home')
def home():
    # Check if the user is logged in
    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'))
```

The above code will create the home route function. If the user is logged in, they will have access to the home page. If not, they will be redirected to the login page.

Edit the **home.html** file and add the following:

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

{% block title %}Home{% endblock %}

{% block content %}
<h2>Home Page</h2>
<p>Welcome back, {{ username }}!</p>
{% endblock %}
```

You also need to create the layout for your logged-in pages. Edit the **layout.html** file and add:

```
HTML
<!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>

```

Now, you can easily extend the same layout for both the home and profile pages.

Currently, when a user logs in, there will be a basic output message. You can now change that to redirect the user to your new home page instead. Find the following code in the login route function:

```
return 'Logged in successfully!'
```




Replace with:

```
Python
return redirect(url_for('home'))
```

The user will now be redirected to the home page when they log in. If you enter the test details into the login form and click the **Login** button, you will see the following:

```
http://localhost:5000/pythonlogin/home
```

Website Title

 Home Profile Logout

Home Page

Welcome back, test!

It's just a simple home page that will output the username. You can implement your own code later.

Next, you need to create the profile page and populate the user's details.

7. Creating the Profile Page

The profile page will populate all details associated with the account (username, password, and email).

Add the following route to the **main.py** file:

Python

```
# http://localhost:5000/pythinlogin/profile - this will be the profile page, only accessible for logged in users
```

```
@app.route('/pythonlogin/profile')
def profile():
    # Check if the user is logged in
    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 logged in redirect to login page
    return redirect(url_for('login'))
```

The above code will create the profile route and retrieve all the account details from the database, but only if the user is logged in.

Edit the **profile.html** file and add:

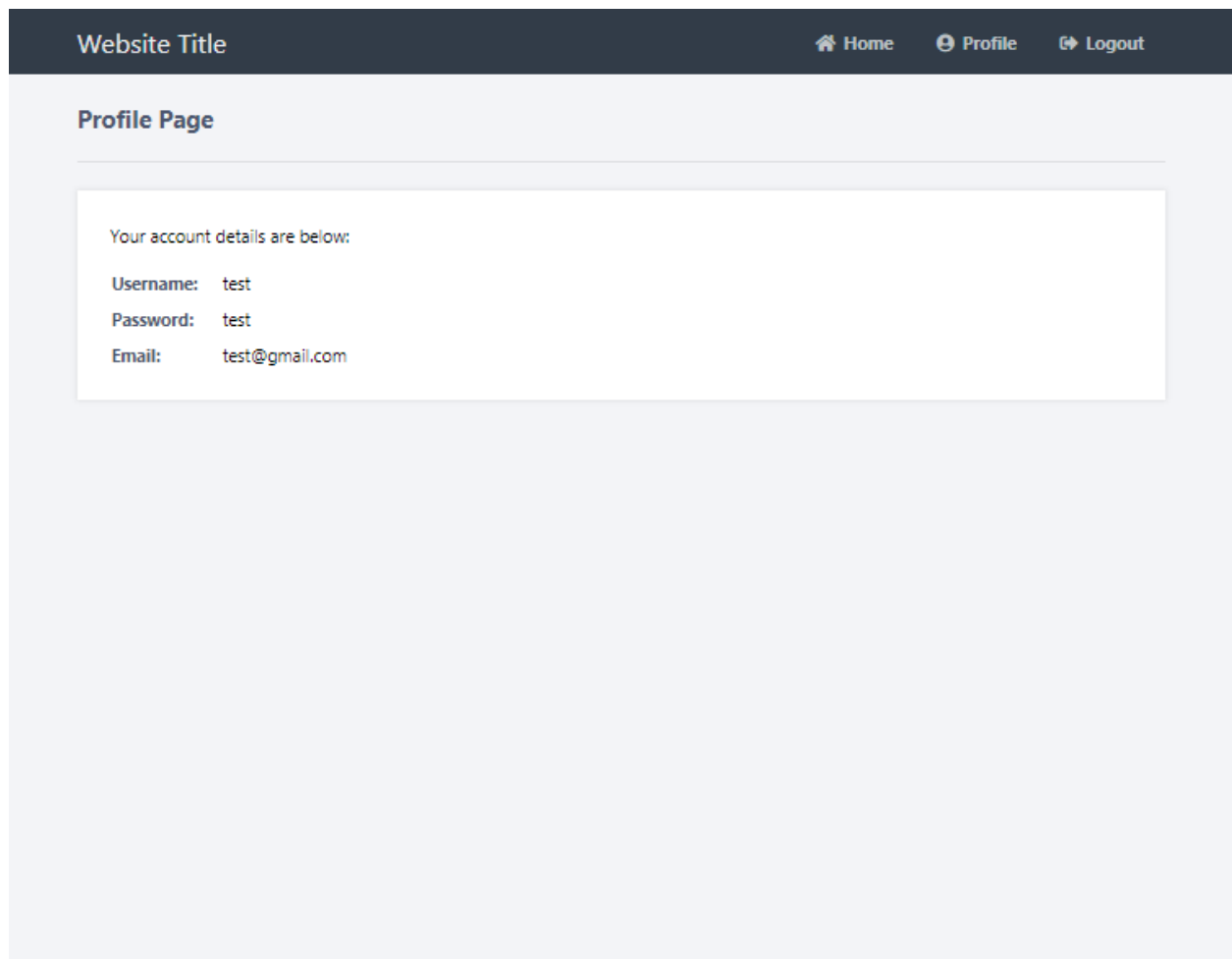
```
HTML
{% extends 'layout.html' %}

{% 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>Password:</td>
            <td>*****</td>
        </tr>
        <tr>
            <td>Email:</td>
            <td>{{ account['email'] }}</td>
        </tr>
    </table>
</div>
{% endblock %}
```

The above code will extend the layout ([layout.html](#)) file that you created earlier. If you navigate to the profile page, with the following address, <http://localhost:5000/pythonlogin/profile>,

it will display like the following:



That's basically it for the home and profile pages. The [Advanced package](#) includes the **edit profile feature** that will enable the user to change their username, password, and email.

Conclusion

Congratulations! You've successfully created your own login and registration system with Python Flask and MySQL. You're free to use the source code from this tutorial in your application(s). You are ready to begin your test project. But I highly recommend you save the project you just completed and use the GitHub version for your testing project.