WeHelp

Assignment - Week 6

This week, we have to build a member system, based on the database tables designed in week 5, with Python FastAPI, MySQL, and any other necessary skills. Refer to W3Schools Python MySQL tutorial for learning how to connect to MySQL by the official mysql-connector-python package.

Note: Don't push code including your daily password to a public GitHub Repository.

Task 1: Pages

We have to build 3 pages for this member system.

Home Page:

- **URL:** http://127.0.0.1:8000/
- **Method:** GET
- **Design Points:** a signup-form with 2 text inputs, 1 password input and a submit button. A login-form with 1 text input and 1 password input and a submit button.

Member Page:

- **URL:** http://127.0.0.1:8000/member
- **Method:** GET
- **Design Points:** a page for logged-in members, including member name.

Error Page:

- **URL:** http://127.0.0.1:8000/ohoh?msg=自訂的錯誤訊息
- **Method:** GET
- **Design Points:** get error message from Query String in URL defined above and show it on this page.

Task 2: Build a procedure for signing up

Users have to sign up for a new member account before logging in.

Signup Endpoint:

- **URL:** http://127.0.0.1:8000/signup
- **Method:** POST

Required Procedure:

1. The user enters name, email and password in the **Home Page**, and then clicks the submit button. Check if there is any empty input in the front-end, if yes, prevent form submission, if no, submit signup form to the **Signup Endpoint**.
2. In the **Signup Endpoint**, get inputs from front-end and execute procedure described below:
    a. Check the website database if there is any repeating email in the member table.
        i. If yes, it means signup failed. Do not insert any data to the member table. Redirect the user to the **Error Page,** show 重複的電子郵件 in the page.
        ii. If no, it means signup succeeds. Insert input data to the member table. Redirect the user to the **Home Page**.

Task 3: Build a procedure for logging in

Users can login in to the member page after signing up.

Login Endpoint:

- **URL:** http://127.0.0.1:8000/login
- **Method:** POST

Required Procedure:

1. The user enters email and password in the **Home Page**, and then clicks the submit button. Check if there is any empty input in the front-end, if yes, prevent form submission, if no, submit login-form to the **Login Endpoint**.
2. In the **Login Endpoint**, get inputs from front-end and execute procedure described below:
    a. Check the website database if there exists an email/password pair in the member table, the same as the inputs.
        i. If yes, it means login succeeds. Record member id, email and name into the user state. Redirect the user to the **Member Page**, show the current member's name in the page.
        ii. If not, it means login failed. Redirect the user to the **Error Page,** show 電子郵件或密碼錯誤 in the page.

Task 4: Build a procedure for logging out

Users can logout from the member page after logging in.

Logout Endpoint:

- **URL:** http://127.0.0.1:8000/logout
- **Method:** GET

Required Procedure:

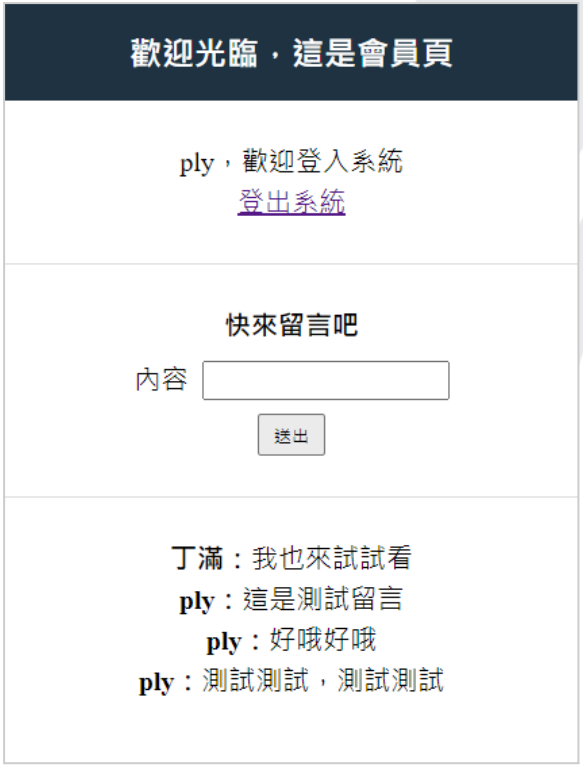
1. In the **Member Page**, we should always verify the recorded user state in the back-end logic. If it does not pass the verification, force redirecting the user to the **Home Page** without showing any content on the member page.
2. Add a logout link/button to the member page. If this logout link/button is clicked, connect to the **Logout Endpoint** where we have to clear recorded member data in the user state and redirect to the **Home Page**.

Task 5: Build a simple message system

In the member page, add a feature for leaving a message, and show all the past messages in the bottom of the page.

Member Page:

- **URL:** http://127.0.0.1:8000/member
- **Method:** GET
- **Design Points:** leaving message form with 1 text input, show past messages including author's name and content.



CreateMessage Endpoint:

- **URL:** http://127.0.0.1:8000/createMessage
- **Method:** POST

Required Procedure:

1. When a user enters the **Member Page**, our backend code gets all the past messages from the database and renders message data to the web page by template engine.

2. Enter message content and click the submit button to connect to the **CreateMessage Endpoint** for leaving a new message.
3. In the **CreateMessage Endpoint**, get message content from front-end, get member id from user state, and insert a new message record to the message table. After all, redirect back or refresh the **Member Page**.

Task 6: Build a procedure to delete message

Users can delete their own message in the member page.

Member Page:

- **URL:** http://127.0.0.1:8000/member
- **Method:** GET
- **Design Points:** only show a delete button next to my own message.

DeleteMessage Endpoint:

- **URL:** http://127.0.0.1:8000/deleteMessage
- **Method:** POST

Required Procedure:

1. Show a delete button next to messages belonging to the current logged-in member.
2. If the delete button is clicked, use the built-in confirm function to confirm this deleting action in the front-end by JavaScript. If canceled, do nothing.
3. If confirming deleting, connect and pass message id to the **DeleteMessage Endpoint** where our backend code gets message id from front-end, and delete corresponding message data in the database. After all, redirect back to the **Member Page**.