Flask Feedback

In this exercise, we'll be creating an application that lets users sign up and log in to their own accounts. Once logged in, users can add feedback, edit their feedback, delete their feedback, and see a list of all feedback that they've given. Many of these routes should be protected, so that for example *user1* can't edit a piece of feedback that *user2* created. (More on this below.)

Part 0: Set up your environment

You know the drill. Make a venv, pip install all the things, put your code on GitHub, etc.

Part 1: Create User Model

First, create a ***User*** model for SQLAlchemy. Put this in a ***models.py*** file.

It should have the following columns:

- ***username*** - a unique primary key that is no longer than 20 characters.
- ***password*** - a not-nullable column that is text
- ***email*** - a not-nullable column that is unique and no longer than 50 characters.
- ***first_name*** - a not-nullable column that is no longer than 30 characters.
- ***last_name*** - a not-nullable column that is no longer than 30 characters.

Part 2: Make a Base Template

Add a base template with slots for the page title and content. Your other templates should use this.

You can use Bootstrap for this project, but don't spend a lot of time worrying about styling — this is **not** a goal of this exercise.

Part 3: Make Routes For Users

Make routes for the following:

GET /

Redirect to /register.

GET */register*

Show a form that when submitted will register/create a user. This form should accept a username, password, email, first_name, and last_name.

Make sure you are using WTForms and that your password input hides the characters that the user is typing!

POST *`/register`*

Process the registration form by adding a new user. Then redirect to ***`/secret`***

GET *`/login`*

Show a form that when submitted will login a user. This form should accept a username and a password.

Make sure you are using WTForms and that your password input hides the characters that the user is typing!

POST *`/login`*

Process the login form, ensuring the user is authenticated and going to ***`/secret`*** if so.

GET *`/secret`*

Return the text “You made it!” (don’t worry, we’ll get rid of this soon)

Part 4: Don’t let everyone go to */secret*

Despite all of this wonderful password hashing that you have been doing, anyone can navigate to ***`/secret`*** and see the text “You made it!”. Let’s protect this route and make sure that only users who have logged in can access this route!

To do that, we’re going to make sure that when we log a user in (and after they register), we store just a little information in the session. When the user successfully registers or logs in, store the ***username*** in the session.

Part 5: Log out users

Make routes for the following:

GET *`/logout`*

Clear any information from the session and redirect to **/**

Part 6: Let’s change */secret* to */users/<username>*

Now that we have some logging in and and logging out working. Let’s add some authorization! When a user logs in, take them to the following route:

GET *`/users/<username>`*

Display a template the shows information about that user (everything except for their password)

You should ensure that only logged in users can access this page.

Part 7: Give us some more feedback!

It's time to add another model.

Create a ***Feedback*** model for SQLAlchemy. Put this in a ***models.py*** file.

It should have the following columns:

- ***id*** - a unique primary key that is an auto incrementing integer
- ***title*** - a not-nullable column that is at most 100 characters
- ***content*** - a not-nullable column that is text
- ***username*** - a foreign key that references the username column in the users table

Part 8: Make/Modify Routes For Users and Feedback

GET *</users/<username>>*

Show information about the given user.

Show all of the feedback that the user has given.

For each piece of feedback, display with a link to a form to edit the feedback and a button to delete the feedback.

Have a link that sends you to a form to add more feedback and a button to delete the user **Make sure that only the user who is logged in can successfully view this page.**

POST *</users/<username>/delete*

Remove the user from the database and make sure to also delete all of their feedback. Clear any user information in the session and redirect to **/**. **Make sure that only the user who is logged in can successfully delete their account**

GET *</users/<username>/feedback/add*

Display a form to add feedback **Make sure that only the user who is logged in can see this form**

POST *</users/<username>/feedback/add*

Add a new piece of feedback and redirect to /users/<username> — **Make sure that only the user who is logged in can successfully add feedback**

GET *</feedback/<feedback-id>/update*

Display a form to edit feedback — ** <#id1>Make sure that only the user who has written that feedback can see this form **

POST *</feedback/<feedback-id>/update*

Update a specific piece of feedback and redirect to /users/<username> — **Make sure that only the user who has written that feedback can update it**

POST *</feedback/<feedback-id>/delete*

Delete a specific piece of feedback and redirect to /users/<username> — **Make sure that only the user who has written that feedback can delete it**

Further Study

- Make sure your registration and authentication logic is being handled in your ***models.py***
- Make sure that if there is already a ***username*** in the session, do not allow users to see the register or login forms
- Add a 404 page when a user or feedback can not be found as well as a 401 page when users are not authenticated or not authorized.
- Add a column to the users table called ***is_admin*** which is a boolean that defaults to false. If that user is an admin, they should be able to add, update and delete any feedback for any user as well as delete users.
- Make sure that if any of your form submissions fail, you display helpful error messages to the user about what went wrong.
- Tests! Having tests around authentication and authorization is a great way to save time compared to manually QA-ing your app.
- **Challenge** Add functionality to reset a password. This will involve learning about sending emails (take a look at the Flask Mail module. You will need to use a transactional mail server to get this to work, gmail is an excellent option) and will require you to add a column to your users table to store a password reset token. **HINT** - here is how that data flow works
  - A user clicks a link and is taken to a form to input their email
  - If their email exists, send them an email with a link and a unique token in the query string (take a look at the built in ***secrets*** module and the ***token_urlsafe*** function. You will create this unique token and store it in the database
  - Once the user clicks on that link, take them to a form to reset their password (make sure that the unique token is valid before letting them access this form)
  - Once the form has been submitted, update the password in the database and delete the token created for that user

Solution

View our solutions <solution/index.html>