Project Part 2

From this part forward, your project is a team assignment. You will use a GitHub repository to track all of your progress and contributions to the project. You should use proper git/GitHub practices in this repo. There is no additional submission for the project as the course staff will check your repo after the deadline to grade your project.

Submit your repo link here:
https://docs.google.com/forms/d/e/1FAIpQLScpaJqGNS-cFGlRbe9WQO3mDHvueBhycIjPcEKsDqTEWZKZWg/viewform?usp=sf_link

You must still use a framework throughout the rest of the project. You and your teammates should discuss which framework you'd like to use from the list provided on the project part 1 handout. You may start part 2 by using one of your part 1 submissions, or start over building a new app as a team.

You will need a database throughout this part, and for the rest of your project. You may use whatever you'd like for your database, however your database must be used through a second container that is separate from your app and started using docker compose.

Objective 1: Authentication

Add authentication to your app. When navigating to you home page, the user should be presented with 2 forms:

- A registration from
    - Used when a user creates an account by entering a username and password
- A login form
    - Used to login after a user creates an account by entering the same username and password that was used when they registered

You are free to design these forms, and the HTTP requests that they create, however you'd like.

When a user is logged in, Their username must be displayed somewhere on your home page.

When a user sends a **registration** request, store their username and a salted hash of their password in your database.

When a user sends a **login** request, authenticate the request based on the data stored in your database. If the [salted hash of the] password matches what you have stored in the database, the user is authenticated.

When a user successfully logs in, set an **authentication token** as a cookie for that user with the HttpOnly directive set. These tokens should be random values that are associated with the user. You must store a **hash** (no salt) of each token in your database so you can verify them on subsequent requests.

The auth token cookie must have an expiration time of 1 hour or longer. It cannot be a session cookie.

Security: Never store plain text passwords. You must only store salted hashes of your users' passwords. It is strongly recommended that you use the bcrypt library to handle salting and hashing.

Security: Only hashes of your auth tokens should be stored in your database (Do not store the tokens as plain text). Salting is not expected.

Security: Set the HttpOnly directive on your cookie storing the authentication token.

Testing Procedure

1. Start your server with "docker-compose up"
2. Open a browser and navigate to http://localhost:8080/
3. Find the registration form and register a username/password
    a. Navigate back to http://localhost:8080/ if a different page loaded after the form submission
4. Find the login form and enter the same username, but an incorrect password
    a. Navigate back to http://localhost:8080/ if a different page loaded after the form submission
5. Verify that your username is not displayed on the page
6. Submit the login form again with the correct username and password from the registration step
    a. Navigate back to http://localhost:8080/ if a different page loaded after the form submission
7. Verify that your username is displayed on the page
8. Restart the server with "docker-compose restart"
9. Navigate to http://localhost:8080/
10. Verify that your username is displayed on the page
11. Open a second browser (Use Chrome and Firefox. Switch to the one you didn't use in the previous steps), register and login with a different username
    a. Navigate back to http://localhost:8080/ if a different page loaded after the form submission
12. Verify that your new username is displayed on the page
13. Refresh the page first browser and verify that it still displayed your first username
14. Delete the cookie containing your authentication token in the first browser and refresh the page
15. Verify that your username is not displayed on the page

16. **Security:** Check the server code to ensure passwords are being salted and hashed before being stored in the database
17. **Security:** Look through the code and verify that the tokens are not stored as plain text
18. **Security:** Verify that the cookies HttpOnly directive is set
19. **Security:** Look through the code to verify that prepared statements are being used to protect against SQL injection attacks [If SQL is being used] (This is true for all project objectives and will not be explicit stated after this objective)
20. Verify that a framework is used to implement these features (This is true for all project objectives and will not be explicit stated after this objective)
21. Verify that a database is being used by communicating with a second docker container that was created using docker compose (This is true for all project objectives and will not be explicit stated after this objective)

Objective 2: Making Posts

Add a way for users to make posts to your page. These posts must contain a title and description chosen by the user along with their username. To build this feature you must include:

- Field on your page where a user can create a post by entering a title and description from your home page
- When a user submits a post, their post will be displayed on the page for all users containing the title entered by the user, the description added by the user, and the username of the user who created the post which should be added by your server (eg. You server will verify that the user was authenticated and are the author of that post)
    - Unauthenticated users cannot make posts (It's ok if the form is still displayed on their page, but submitting the form will not result in the post being added to your page
- Users must see new posts without requiring any action on their part
- Posts must be stored in a database

Security: You must escape any HTML in the users' title, description, and username since all three will be displayed to all users after this objective is complete

Security: If you are using a SQL database, you must protect against SQL injection attacks

Testing Procedure

1. Start your server using docker-compose up
2. Open a browser and navigate to http://localhost:8080/
3. Find the way to create a post, enter a title and description, and submit the post
    a. Verify that the post does not appear on the page
4. Login using an existing account (or register a new account)

5. Submit another post and verify that it appear on the page along with your username
6. Open a second browser (Use Chrome and Firefox. Switch to the one you didn't use in the previous steps), and login with a different account (register a new account if you haven't already)
7. Create several posts in the second browser
   a. Verify that all posts appear on the page along with your second username
8. Go back to the first browser and verify that the new posts appeared without taking any action
9. Make another post from the first browser and verify that it appears in both browsers
10. Restart the server with "docker-compose restart"
11. Navigate to http://localhost:8080/
12. Make another post from each browser and verify that the posts appear as expected (contains the correct usernames)
13. Refresh both browsers and verify that all messages still appear on the page
14. **Security:** Verify that HTML is escaped in **titles**
15. **Security:** Verify that HTML is escaped in **descriptions**
16. **Security:** Verify that HTML is escaped in **usernames**

Objective 3: Liking Posts

Add a "like" button to every post along with the total number of likes for that post. You are free to choose what this button is called (eg. like, dislike, heart, thumbs up, etc.), but it will be called a like in this description for simplicity. The like button should have the following functionality:

- When a user clicks the like button for a post, their like should be recorded in your database
- The current number of likes for each post should should always be displayed on the page and should update for all users without requiring any action on their part
- A user can only like a post once. This needs to be enforced by your server, not your front end (eg. It is not acceptable to simply hide/remove the like button. You need a server-side check that prevents multiple likes)
- When a user likes a post, they should have an option to "unlike" the post. This is expected to be done by clicking the like button a second time, though you are free to design this however you'd like as long as it's clear to the user how they can remove their like
- Your team may decide if users are allowed to like their own posts. By default, it's expected that they would be able to like their own posts, but you add logic to prevent this if you'd like. It will not be tested in the testing procedures

Testing Procedure

1. Start your server using docker-compose up
2. Open a browser and navigate to http://localhost:8080/

3. Register and login using 3 different accounts throughout this procedure using both Firefox and Chrome (You can use incognito windows to have different sets of cookies with the same browser)
4. Create a post with one of the 3 accounts and verify that all users see a like count of 0 on the post
5. Like the post with a second account and verify that all users can see the like count increment to 1
6. Like the post with the third account and verify that all users can see the like count increment to 2
7. Unlike the post with the second account, then the third account and verify that all users see the count go to 1, then to 0
8. Like the post again with the second user
9. Find a way to send another HTTP request from the second user that will like the post a second time
10. Verify that the like is not registered due to a server-side check that prevents multiple likes of the same post from the same user
11. Create several posts by each of the accounts
12. Like and unlike the posts in a variety of combinations and verify that all 3 users see the updates to the like count as expected
    a. Do not have any user like their own post during testing since the expected behavior is undefined by this document
13. Restart the server with "docker-compose restart"
14. Navigate to http://localhost:8080/ and verify that all the posts have the correct number of likes
15. Use the like buttons again and verify that they still work as expected after the server restart

Submission

All of your project files must be in your GitHub repo at the time of the deadline. Please remember to include:

- A docker-compose file in the root directory that exposes your app on port 8080
- All of the static files you need to serve in the public directory (HTML/CSS/JavaScript/images)

It is **strongly** recommended that you download and test your submission. To do this, clone your repo, enter the directory where the project was cloned, run docker-compose up, then navigate to localhost:8080 in your browser. This simulates exactly what the TAs will do during grading.

If you have any Docker or docker-compose issues during grading, your grade for each objective may be limited to a 1/3.

Team Scoring

Each objective will be scored on a 0-3 scale as follows:

3	Clearly correct. Following the testing procedure results in all expected behavior
2	Mostly correct, but with some minor issues. Following the testing procedure does not give the exact expected results
1	Clearly incorrect, but an honest attempt was made to complete the objective. Following the testing procedure gives completely incorrect results or no results at all. This includes issues running Docker or docker-compose even if the code for the objective is correct
0	No attempt to complete the objective or violation of the assignment (Ex. Not using a framework) -or- a security risk was found while testing the objective

3	2 Application Objectives
2	1 Application Objective
1	0 Application Objectives
0	0 Application Objectives

Please note that there is only one chance to earn these application objectives. There will not be a second deadline for project parts 2-4.

Individual Grading

The grading above will be used to determine your team score which is based on the functionality of your project. Your actual grade may be adjusted based on your individual contributions to the project. These decisions will be made on a case-by-case basis at the discretion of the course staff. Factors used to determine these adjustments include:

- Your meeting form submissions: team meeting and eval form
    - You must fill out this form after every meeting. Failure to do so is an easy way to earn a negative individual grade adjustment

    - The quality of your submissions will be taken into account as well (eg. Saying "I'll do stuff" before the next meeting is a low quality submission)
    - You may submit more meeting forms than are required even if there was no meeting (eg. If you want to adjust your evals after a deadline without waiting for the next meeting)
- Your evaluations from the meeting form
    - If you teammates rated you poorly/excellently, your grade may be adjusted down/up respectively
- Your commits in the team repo
    - Your commits may be check to see if you did in fact complete the work you mentioned in the meeting form, as well as compare the amount of work you completed to that of your teammates
    - You **MUST** commit your own code! It is not acceptable for your teammate to commit your code for you. You should have a clear separation between your tasks and commit the code for your task. If a commit is not in your name, you effectively did not write that code. If a teammate is making this difficult, let the course staff know in the meeting form