Homework 4: Authentication and HTTPS

Objective 1: Visit Counting Cookie

Use a cookie to count the number of times a user has visited your home page. When a user first requests your home page (path "/"), set a cookie to 1 to track the number of times they visited your page. If the cookie is already set (Subsequent visits), read the cookie to check the number of times the user visited, increment this value by 1, then set the cookie with the incremented value.

Only set/update the cookie on requests for the root path "/".

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

Add some HTML to your home page that displays the number of visits for the user.

```
<h1>Number of page visits: 5</h1>
```

Note: As always, you can personalize these messages if you'd like as long as it's clear to the grader that the visit count is displayed.

Testing Procedure

1. Start your server with "docker-compose up"
2. Open a browser and navigate to http://localhost:8080/ (or http://localhost or https://localhost if objective 5 is complete AND the app doesn't load on 8080)
3. Verify that somewhere on the page is a visit count of 1
4. Refresh the page
5. Verify that somewhere on the page is a visit count of 2
6. Close the browser window
7. Open a new window of the same browser and navigate to http://localhost:8080/
8. Verify that somewhere on the page is a visit count of 3 (Verifies that the cookie is not a session cookie)
9. Open a different browser and navigate to http://localhost:8080/
10. Verify that somewhere on the page is a visit count of 1

Objective 2: Authentication and Tokens

Add authentication to your app. This must include 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 have much flexibility in how you create these forms. You can use an HTML form element with url or multipart encodings, process the form using JavaScript to send an AJAX request, or any other approach that implements the required features.

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 is authenticated, display a success message on the page that loads when the form is submitted (or on the current page if you're using AJAX). For example:

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** 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.

Whenever a request for your home page is received from a user with a valid authentication token as a cookie, the page must contain a message containing that user's username. For example:

```
<h1>Welcome back <username>!</h1>
```

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 required.

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
4. Navigate back to http://localhost:8080/ if a different page loaded after the form submission
5. Find the login form and enter the same username, but an incorrect password
6. Refresh the home page and verify that your username is not displayed
7. Back on http://localhost:8080/, submit the login form again with the correct username and password from the registration step
8. Refresh the page (Go back to the home page if a different page loaded after the form submission) and verify that you can see a message that contains your username
9. Restart the server with "docker-compose restart"
10. Navigate to http://localhost:8080/
11. Verify that the page still acknowledges that you're logged in and displays your username
12. Open a second browser, register and login with a different username, refresh the page, and verify that your username is displayed
13. Refresh the first browser and verify that your username is still displayed
14. Delete your authentication token in the first browser
15. Refresh the page
16. Verify that your username is no longer displayed

17. **Security:** Check the server code to ensure passwords are being salted and hashed before being stored in the database
18. **Security:** Look through the code and verify that the tokens are not stored as plain text
19. **Security:** Verify that the cookies HttpOnly directive is set

Objective 3: Authenticated Chat

Using any method you'd like (HTML forms, AJAX requests, WebSockets), add authentication to a chat feature on your home page. This chat is not required to be live (ie. It's ok if a refresh is required to see new chat messages).

This chat feature must require users to be authenticated and the username of the sender must be shown next to each of their messages. Verify each user by their authentication token and lookup the user associated with the token if the token is valid. If a request is received that does not contain a valid auth token, do not add the chat message to the page.

The chat history must be served when the page loads.

It is acceptable if the chat history does not persist through a server restart.

Testing Procedure

1. Start your server using docker-compose up
2. Open a browser and navigate to http://localhost:8080/
3. Find the chat feature and send a message
4. Verify that the message does not show up on the page
5. Register an account and login
6. Refresh the home page page (To ensure that a request is made that includes an auth token)
7. Find the chat feature and send several messages
8. Refresh the page
9. Verify that the messages appear on the page with your username shown next to each message
10. Open a second browser, navigate to http://localhost:8080/, register and login with a different username than the one used in step 3
11. Send several messages
12. Refresh both browsers
13. Verify that all messages appear on both browsers with the correct username shown next to each message
14. **Security**: Verify that HTML is escaped in **chat messages**
15. **Security**: Verify that HTML is escaped in **usernames**

Objective 4: Authenticated XSRF Tokens

Add randomly generated XSRF tokens to your chat feature that are associated with the user that made the request for your home page. If a chat submission is received that does not contain a valid token for the user, do not save the submitted data and respond with a 403 Forbidden response notifying the user that their submission was rejected.

These tokens must be associated with a specific user. For example, when an authenticated user (check the auth token) requests your home page, you will generate (or retrieve) an XSRF token for this user. Whenever the user submits a chat message, you must check if their XSRF token matches a token that has been issued to this user. If the token was not issued to this user, you should respond with a 403.

It is ok to reuse a token multiple times for a single user. This way if you are using AJAX calls for your chat feature you can keep reusing the same token since you are not required to issue a new token until the page reloads. It is also ok to only have one token per user.

If a request is made from an unauthenticated user, you are not required to send a XSRF token.

Testing Procedure

1. Start your server using docker-compose up
2. Open a browser and navigate to http://localhost:8080/
3. Register an account and login
4. Refresh the page (To ensure that a request is made that includes an auth token)
5. Inspect the HTML of one of the forms and verify that there is a XSRF token included somewhere as part of the form (Only the form for the authenticated chat needs a XSRF token)
6. Open a second browser, navigate to http://localhost:8080/, register and login with a different username than the one used in step 3
7. Find the XSRF in the second browser and copy it into the browser in the first browser
8. Send a chat message in the first browser and verify that the response is a 403 and the chat message does not appear in the chat (This XSRF token was not issued to this user) (If using WebSockets, you don't have to send a 403 and it's ok to not respond to the message)

Objective 5: HTTPS

Setup your app to listen to HTTP requests on port 80 and HTTPS requests on port 443 using a self-signed certificate.

It is recommended that you use nginx as a reverse proxy that will listen on both ports and forward traffic to your app container.

Testing Procedure

1. Start your server with "docker-compose up"
2. Open a browser and navigate to http://localhost
3. Verify that the home page loads
4. Open a browser and navigate to https://localhost
5. Verify that the home page loads. Accept any warning about a self-signed certificate

Submission

Submit all files for your server to AutoLab in a **.zip** file (A .rar or .tar file is not a .zip file!). Be sure to include:

- A docker-compose.yml file in the root directory that exposes your app on port 8080 and runs a separate container for your database
- A Dockerfile that is used by your docker-compose configuration
- All of the static files you need to serve (HTML/CSS/JavaScript/images)

It is **strongly** recommended that you download and test your submission after submitting. To do this, download your zip file into a new directory, unzip your zip file, enter the directory where the files were unzipped, run docker-compose up, navigate to localhost:8080 in your browser, and go through the testing procedures for all 5 objectives. 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.

Grading

Each objective is equally weighted and will 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. Using an HTTP library) -or- a security risk was found while testing the objective

The following conversions will be used when translating grades on this 0-3 scale to percentage grades:

3	100%
2	80%
1	60%
0	0%