Project Part 1

This project is a team assignment and you will use a public GitHub repository to track all of your progress and contributions to your project. You should use proper git/GitHub practices in this repo. There are no additional submissions for the project as the course staff will check your repo after the deadline to grade your project. Commits made after each deadline will be ignored while grading.

As opposed to the homework, you must use a framework throughout the project. The purpose of the project is to give you experience working on a web app in the way you will outside of this course (You should never build an app starting with TCP outside of an educational environment). In the real world, we use frameworks. Through this experience, you will gain an understanding of what a framework does for you.

You will need a database throughout the 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.

In this part of the project, you will become familiar with the idea of using a framework to build a web application. You will choose a web framework, build an app, and set up docker compose for deployment.

Objective 1: Choosing a Framework and Hosting a Static Page

Build an app. You have freedom in what you build for this objective as long as it contains all of the following that must be hosted by your server:

- HTML hosted at the root path
- CSS hosted at a separate path
- JavaScript hosted at a separate path
- At least one image

Note: You must actually build your own app. You cannot steal someone else's work. If you host, for example, the front end from the homework, your team will not earn any credit for this part of the project.

All of these parts must be hosted by your server (eg. You must serve the image from your server using your framework of choice). Do not add a full url as the src of the image.

Security: All files must be served with the correct MIME type and the X-Content-Type-Options: nosniff header must be set.

You must use a web framework for the project (As opposed to the homework where you effectively build your own framework). You are expected to find and study documentation to learn how to use your framework of choice as it will not be covered in lecture.

You may choose from the following approved frameworks:

- Flask / Python
- Express / Node.js
- Django / Python
- gin / go
- Play / Java;Scala
- Koa / Node.js
- FastAPI / Python
- Elysia / Node.js
- Spring / Java

If you would like to use a framework that is not in this list, let Jesse know and it will be considered for approval. If approved, it will be added to this list.

You may change your framework in later parts (not recommended), but you must use an approved framework for all parts of the project.

Docker: Once you choose a framework, set up your project using Docker and docker compose. Your app must run on local port 8080.

Testing Procedure

1. Start your server using "docker compose up"
2. With the network tab open, navigate to http://localhost:8080/ in a browser
3. Ensure that a web page appears in the browser
4. In the network tab, check that at least 4 HTTP requests were sent to load the page containing HTML, CSS, JavaScript, and an image all in separate requests
5. Check to ensure that each of the 4 parts had some impact on the loaded page (eg. the CSS and JavaScript must do something/anything)
6. **Security:** Verify that the "X-Content-Type-Options: nosniff" header is set on each response
7. Check the code to ensure that an approved framework is being used (No credit will be given on any part of the project if the code uses a TCP socket like the HW code does. For all other project objectives, this will be implied to be part of the testing procedures)

Objective 2: 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 2 passwords (They provide the same password twice)
- 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 may use any approach you'd like to build these forms as long as they contain these features:

- After registration or login, the user should still be on the homepage (Either using AJAX or a redirect). The user should not have to manually navigate back to your app
- Checking if the 2 entered passwords are the same must be done on your server. Do not trust the front end for this verification
- A user cannot register with a username that is already taken by another user

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

A logged in user must have a way to logout that will invalidate their auth token.

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 required) 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/
    a. Clear all cookies stored for localhost
3. Find the registration form and:
    a. Attempt to register with 2 passwords that do not match
        i. Verify that the account is not created
        ii. Check to make sure the check was made by the server (It can also be on the front end for UX, so if you find a front end check do not assume that the back end is not also checking. You must check the server code if you find a front end check)
    b. Register properly with a username and matching passwords
4. Find the login form and enter your chose username, but an incorrect password
    a. Verify that your username is not displayed on the page
5. Submit the login form again with the correct username and password from the registration step
6. Verify that your username is displayed on the page
7. Restart the server with "docker compose restart"
8. Navigate to http://localhost:8080/
9. Verify that your username is still displayed on the page
10. Open a second browser (Use Chrome and Firefox. Switch to the one you didn't use in the previous steps)
    a. Attempt to register with the same username that you've already registered
        i. Verify that an account was not created and that you cannot login with that username and password
    b. Register with a second username and login with this new account
        i. Verify that your new username is displayed on the page
11. Refresh the page on the first browser and verify that it still displays your first username
12. In the first browser:
    a. Find your auth cookie and copy your auth token. Store this token for later use
    b. Modify your auth token and refresh the page
        i. Verify that you are treated as not being logged in
    c. Paste your auth token back into your cookie and refresh the page
        i. Verify that you are logged in again (Your username is displayed on the page)
    d. Find a way to logout through the app's interface and logout
        i. Refresh and verify that you are logged out
    e. Modify your cookies to set an auth token with the value saved earlier to simulate the same state you had when you were logged in, then refresh the page

            i. Verify that you are not logged in
        f. Log in again by entering your username/password
            i. Verify that you are logged in

13. Refresh the second browser and verify that you are still logged in with your second account
14. **Security:** Check the server code to ensure passwords are being salted and hashed before being stored in the database
15. **Security:** Look through the code and verify that the authentication tokens are not stored as plain text **and** that the cookie values are not the same as what's stored in the DB (ie. Whatever value you use in cookies, the DB must store a hash of that value. Setting the cookies to the hashed values defeats the purpose of hashing and is a security risk)
16. **Security:** Verify that the cookies HttpOnly directive is set
17. **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)
18. 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 3: Making Interactive Posts

Note: For this objective, you should start thinking about what your app will eventually accomplish. You are free to design whatever type of app you'd like as long as it meets the criteria of each objective. In future parts you will add multimedia uploads, live WebSocket interactions, at least one feature where timing matters, and you will deploy your app to the world at a domain name of your choosing. Start planning what you want to build with all those features.

Add a way for users to make posts to your page. These posts must contain at least two pieces of user-supplied information (eg. This must be more than a simple chat feature). When a user creates a post, their username must be displayed on that post. You may allow guest posts if you'd like, but it is not required (Guest posts can make testing easier though). To build this feature you must include:

- A clear way for users to create posts (A reasonable person who logs in to your app must be able to figure out how to make a post)
- When a post is created, the server will verify the author and add their username to the post (Do not trust the user to supply their username without proper authentication and verification)
- All logged in users should be able to see all posts (You may decide if guest users can view posts)
- It is ok if a refresh is required to see new posts
- Posts must be stored in your database

Once a post is created, all authenticated users must be able to interact with each post in a way that takes their username and the specific post into account. This is open for interpretation and creativity so it can fit into whatever idea you have for your app. If your entire team is not creative, you can build a like button that prevents a user from liking the same post twice. Any interaction you design must meet the following criteria:

- Interactions are stored in your database
- All interactions should be visible to all [authenticated] users (You decide if guests can see interactions). It is ok if this requires a refresh to see new interactions
- Your server must verify the user who made the interaction and take their username into account in some way (eg. The server checks if this user already liked this post and blocks the second like) (eg. Users can comment on posts and their username is displayed on their comments)
- Interaction must be made on a per-post basis (eg. A user can like or comment on a specific post). It should be clear to all users which interactions are related to which posts (eg. Each post displays it's number of likes)

Security: You must escape any HTML supplied by your users

Testing Procedure

1. Start your server using docker compose up
2. Open a browser and navigate to http://localhost:8080/
3. Create an account and login (You can reuse an account made while testing a previous objective)
4. Find the way to create a post, enter all required information, and submit the post
    a. Verify that the post appear on the page along with your username
5. 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)
    a. Create several posts in the second browser
    b. Verify that all posts appear on the page along with your second username
6. Go back to the first browser, refresh the page, and verify that the new posts appeared as expected
7. Make another post from the first browser and verify that it appears in both browsers
8. Find a way to interact with posts and from each browser, interact with posts made by the other user. If necessary to test interactions effectively, use a 3rd browser/account
    a. This step is more open-ended since each team has much freedom in their design. Do whatever is needed to test the feature (eg. For likes, make sure you cannot like a post twice and ensure this check is made on the back end)
    b. Make sure that the user and post were taken into account for each interaction. Interaction not associated with a specific post, or that make no difference who made the interaction, do not satisfy this objective
9. Restart the server with "docker compose restart"

10. Refresh the page in both browsers and verify that all posts and interactions still appear as expected
11. **Security:** Verify that HTML is escaped in all user supplied strings

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 (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 (Complete)	Clearly correct. Following the testing procedure results in all expected behavior
2 (Complete)	Mostly correct, but with some minor issues. Following the testing procedure does not give the *exact* expected results, but all features are functional
1 (Incomplete)	Not all features outlined in this document are functional, but an honest attempt was made to complete the objective. Following the testing procedure gives an incorrect result, or no results at all, during any step. This includes issues running Docker or docker-compose even if the code for the objective is correct
0.3 (Incomplete)	The objective would earn a 3, but a **security** risk was found while testing
0.2 (Incomplete)	The objective would earn a 2, but a **security** risk was found while testing
0.1 (Incomplete)	The objective would earn a 1, but a **security** risk was found while testing
0 (Incomplete)	No attempt to complete the objective or violation of the assignment (Ex. Using an HTTP library)

Note that for your final grade there is no difference between a 2 and 3, or a 0 and a 1. The numeric score is meant to give you more feedback on your work.

3	Objective Complete
2	Objective Complete
1	Objective Not Complete
0	Objective Not Complete

Please note that there is only one chance to earn these application objectives. There will not be a second deadline for any part of the project.

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. Since you have weekly meetings, you are expected to have as many form submissions as there were weeks for this part of the project
    - 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 your teammates rated you poorly/excellently, your grade may be adjusted down/up respectively
- Your commits in the team repo
    - Your commits may be checked 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
    - If you don't have any commits on the default branch of the repo, you effectively did not work on the project and will earn a 0 after individual grade adjustments

Security Essay

For each objective for which your team earned a 0.3 or 0.2, you will still have an opportunity to earn credit for the objective by submitting an essay about the security issue you exposed. These essays must:

- Be at least 1000 words in length
- Explain the security issue from your submission with specific details about your code
- Describe how you fixed the issue in your submission with specific details about the code you changed
- Explain why this security issue is a concern and the damage that could be done if you exposed this issue in production code with live users

Any submission that does not meet all these criteria will be rejected and you will not earn credit for the objective.

Security essays are individual assignments. It is important that every member of the team understands the importance of preventing security vulnerabilities even if they did not write the offending code. Multiple members of a team submitting the same, or similar, essays is an academic integrity violation.

Due Date: Security essays are due 1-week after grades are released.

Any essay may be subject to an interview with the course staff to verify that you understand the importance of the security issue that you exposed. If an interview is required, you will be contacted by the course staff for scheduling. Decisions of whether or not an interview is required will be made at the discretion of the course staff.