Project Part 2

In this part of the project, you will add multimedia uploads and WebSocket interactions to your app. You will have much freedom in the design of your app and what features you build with these technologies.

In the third objective, you will deploy your app to the world. Notice that this part is due right before presentations start and your app must be deployed during your presentation (This way, every team must deploy by the part 2 deadline regardless of when they will present).

Objective 1: Multimedia Uploads

Add multimedia uploads to your app. You can design your own feature for this objective as long as it meets the following criteria:

- Logged in users are provided a clear way to upload multimedia (A reasonable person who logs in to your app must be able to figure out how to upload multimedia)
    - It is ok if users have to create and/or interact with a post before this feature is apparent
- The uploaded multimedia can be an image, video, and/or audio and can be either static or streamed (eg. A group voice chat feature is acceptable even though it's not a static upload)
- Other users can consume the multimedia that has been uploaded
    - It is acceptable if only a subset of users can consume the media (eg. Image uploads in DMs shouldn't be public to all users)

Any feature that meets these criteria will complete this objective. If you are having trouble thinking of an idea, here are some sample features that would meet all criteria:

- Users can upload profile pictures which are displayed on each of their posts
- Posts can contain images in addition to their text (An imageboard)
- Posts can contain video in addition to their text (A YouTube clone)
- Use WebRTC to build a voice chat feature
- Use WebRTC to build a video conferencing feature (A Zoom clone)
- A live-streaming app (A Twitch/YouTube Live clone)

Testing Procedure

1. Start your server using docker compose up
2. Open two browsers and navigate to http://localhost:8080/
3. Create an account and login in both browsers
4. Find a way to upload multimedia. When you find a way to do this, move on to step 5

    a. If there is no clear way to do this, create and interact with posts and check if there's a way to upload multimedia on the posts
    b. Look for any links to other pages on the app that may contain the feature
    c. Refresh the home page
    d. If there's still no clear way to upload multimedia after following all these steps, the feature is assumed to not be implemented.
5. Upload multimedia and verify that it is displayed to the other user
    a. This step is open-ended as it will depend on the specific feature implemented by each team. Do whatever is needed to test the feature and verify that it meets the required criteria
    b. If necessary to test interactions effectively, use a 3rd/4th/etc. browser/account
6. Restart the server with "docker compose restart"
7. Refresh the page in both/all browsers and verify that all multimedia is still displayed [This step can be skipped if the feature involves streaming multimedia]
8. Check for relevant **security** vulnerabilities

Objective 2: WebSocket Interactions

Add WebSocket interactions to your app. You can design your own feature for this objective as long as it meets the following criteria:

- Logged in users are provided a clear way to interact with each other using WebSockets (A reasonable person who logs in to your app must be able to figure out how to use this feature)
    - It is ok if users have to create and/or interact with a post before this feature is apparent
- Interactions must be both sent and received via WebSockets (eg. You can't have messages sent to the server with WebSockets, but nother received by the user. It must be duplex communication)
    - You must actually use WebSockets. For example, the SocketIO library sometimes falls back to long-polling. You must prevent this from happening and force it to use WebSockets if you use this library
- Other users must see the interaction immediately (Minus network delays) without requiring a refresh or any other action on their part
    - It is acceptable if only a subset of users can see the interactions (eg. DMs over WebSockets that are only shown to sender and recipient)
- WebSocket interaction must be authenticated if the user is logged in and this authentication must matter to other users of your app
    - It's up to you if you want to allow guests to use your WebSocket feature. If they can, they must interact as a guest
    - If a user is logged in, their identity must be taken into account in all their WebSocket interactions and displayed to other users. If 2 users both take an action, a 3rd user must be able to see who took which action

Any feature that meets these criteria will complete this objective. If you are having trouble thinking of an idea, here are some sample features that would meet all criteria:

- Add a user list and a way to send DMs
- Convert your post interactions to use WebSockets (No refresh or polling to see new interactions)
- ~~Convert uploading posts to use WebSockets (No refresh or polling to see new posts)~~ -required!!!
- ~~Add a live chat feature (Either global or separate chat rooms)~~
- Build a game where player actions are sent to all other players in real-time. Usernames should be displayed to other players
- A drawing app where users can all draw on a shared JS canvas with actions shared live. Each user draws in a different color with a key displayed to all users so they know who drew which parts of the drawing
- A Google Docs clone where all users can edit the same text and each user can see everyone else's cursor with their username displayed on their cursor
- An auction app where bids are sent and updated in real-time. The name of the highest bidder is displayed to all users

Note: You will have to authenticate the WebSocket connections. This can take some work depending on the library you use as some of them make it difficult to access the HTTP upgrade request that contains your auth token. Once a connection is authenticated, you can treat all WS frames over that connection as authenticated.

Testing Procedure

1. Start your server using docker compose up
2. Open two browsers and navigate to http://localhost:8080/
3. Create an account and login in both browsers
4. Open the network tab and refresh the page
    a. Verify that there is a 101 response that upgrades to WebSockets
    b. View the messages on this connection while testing this objective
5. Find a way to interact using WebSockets. When you find a way to do this, move on to step 6
    a. While watching for messages on WebSocket connections, use all the features of the app until you find one that sends messages over the connection
    b. Look for any links to other pages on the app that may contain the feature
    c. Refresh the home page
    d. If you've used all apparent features of the app without seeing a message over the WebSocket connection, the feature is assumed to not be implemented.
6. Use the WebSocket feature
    a. Verify that messages are both sent and received by at least one user each (eg. One user can send with another receiving which would be the case for DMs)

    b. Verify that the messages are critical to the functionality of the feature (eg. The messages should not be superficial just to say it uses WebSockets)
    c. Verify that interactions are real-time and do not require a refresh or polling
    d. Verify that each user's account mattered in the WebSocket interactions as was apparent to other users
        i. If two users can take an action and they appear the same to a third user, this criteria is not met and the objective is not complete (eg. A Google doc clone where you can't identify other users earns a 1/3 for this objective) (eg. If post interactions are converted to WebSockets and the interaction is a like button that only displays the total number of likes, that's a 1/3 for this objective since a 3rd user would only see the number go up with no indication of who liked the post)
7. As with other objectives, this is open-ended as it will depend on the specific feature implemented by each team. Do whatever else is needed to test the feature and verify that it meets the required criteria
    a. If necessary to test interactions effectively, use a 3rd/4th/etc. browser/account
8. Check for relevant **security** vulnerabilities

Objective 3: Deployment and Encryption

Deploy your app on a publicly available server with a domain name and HTTPS with a valid certificate. You may use any means available to accomplish this, though it is recommended that you take advantage of the GitHub Student Developer Pack.

When deployed, add a link to your app in the readme of your repository so we can find your deployment.

Note the following implications of this objective:

- You will use the WSS protocol for your WebSocket connection (If you finished the WebSocket objective)
- Your certificate must be valid. It's recommended that you use CertBot to acquire a free certificate
- Any HTTP requests must be redirected to use HTTPS. Do not let users access your app with unencrypted requests

Free Clause: You are not required to spend any money to take this course. If your team is in a situation where you need to spend money to deploy your app (eg. Every member of your team already used your student developer pack credit on other projects), please let me (Jesse) know and I'll work with you to ensure you are not required to spend money on the course requirements. You pay enough in tuition. You do not need to pay more to take a class.

Security: Do not map port 27017:27017 for your MongoDB in your docker-compose file. This is a major security risk that makes your database publicly available. Since you are deploying on the Internet in this objective, that means everyone on the planet would have access to your DB and attackers will steal your data. Some of you have been using this in your development environments, but this MUST be removed before deploying.

Testing Procedure

1. Find the app url in the project repo and navigate to the public deployment
2. Ensure the page loads using an HTTPS connection and no security warnings appear
3. Navigate to the same app using HTTP and verify that you are redirected to the app using HTTPS
4. If objective 2 is complete, verify that the WebSocket connection is encrypted using WSS
5. **Security**: In the repo, check the docker compose file and ensure that port 27017 is not mapped to a local port. If it is, attempt to access the deployed DB and ensure that it is not publicly available. If it is publicly available in production, that's game over for this objective

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)
- Add your public link to your deployment to the README in your repo

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.