Team Project Requirements

GitHub Repository

Create a GitHub repository for your project and add all your team members as collaborators. This repository will contain all the code for your project.

This repository should be public. If you have a convincing reason why you need a private repository, please message Jesse to discuss this reason.

Note: It's recommended that you use GitHub issues (Or a task tracking app) to document what each team member is working on at any given point in time. You can submit links to these issues/tasks on the meeting form to make it very clear what each team member was responsible for completing. If the tasks are not clear, individual grading decisions will be made at the discretion of the course staff.

Web Frameworks

You must use a web framework for the project (As opposed to the homework where you effectively build your own framework).

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

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.

Open-Source Reports

Report template

You are required to write 3 open-source reports as part of your project. These reports will be on the following features of your framework:

- TCP connections
    - Must include the code, in your chosen framework from the list above, where the connections are established or where a library approved on the homework is called
- Parsing HTTP headers

    - Must include the code in your framework from the above list that parses the headers
- WebSockets
    - Be sure to include how the connection is established as well as how frames are parsed in the library you chose for WebSocket connections

Create a directory in your repository named “reports” that contains all the open-source reports for your project. Review the project section of the course website for details on what each report must contain.

You must use the report template to write each report for your project. You should copy this template 3 times, one for each report. In your repository, you can either include a file with public links to your reports as Google Docs, or convert them to PDFs.

For each report, you must trace through the libraries you use until you find the code that actually does the work that you want it to do (eg. You must provide links to the code in the library that resembles your HW code). You will show this entire trace of calls from your code to the code doing the work which may include calls through many classes and libraries. Be prepared to dig deep into these libraries!

Team Project Reports Checkpoint

After week 11, the course staff will review the reports in your repository. You should attempt to have all of your reports complete by this point in the semester so we can provide feedback to you.

Docker-Compose

Your app must be set up to deploy with docker-compose. In the root directory of your repository, include a docker-compose.yml file with everything needed to run your app using docker-compose. At a minimum this must include creating a container for your database and another container for your app that will communicate with the database container.

- You may choose the local port for your app unless otherwise directed

When testing or deploying your app the procedure will be:

1. Clone your repository
2. cd into to the directory of the repository
3. Build the docker image (docker-compose build)
4. Create and run a docker containers (docker-compose up --detach)
5. Open a browser and navigate to http://localhost:<local_port> (We’ll read your docker-compose.yml to find your local port)

Deployment

Your app must be deployed and publicly available with a domain name. 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.

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. you all 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.

Project Requirements

For your project, you may choose one of the following apps to build. You have much freedom in the design of your app, but you must complete the required features for your app of choice.

The requirements will be broken down into these 4 categories. The following apply to all projects:

- [User accounts]
    - A user must have a way to securely create an account and login to that account
    - When logged into their account, a user can view and edit data that is specific to their profile
    - Any information that is part of a user profile must be stored in a database that is created by your docker-compose file
    - Other users must not be able to edit the profiles of other users
    - You must build your own authentication system (OAuth "login with Google/etc." is not allowed). When storing passwords on your server, they must be stored securely
    - Users must be provided a way to log out
- [User Data]
    - Each app will include a way for some users to view information about other users
    - This data does not have to be live (ie. a refresh can be required to view new user data)
- [Communication]
    - Some users will have a way to communicate with each other directly
    - This data does not have to be live (ie. a refresh can be required to view new user data)
- [WebSockets]
    - Each app will include a feature that will require a WebSocket connection to allow live 2-way interaction with your server

    - You are required to use WebSockets for these features! Resorting to polling or long-polling is not allowed (Please note that the SocketIO library often resorts to long-polling. You must disable this and force it to use WebSockets if you are using this library)

Game

Make a multiplayer game with lobby support and leaderboards. The game itself can be very simple as you will be graded on the functionality of the 4 features below, however it must be a multiplier game (think tic-tac-toe, pong, connect 4, rock-paper-scissors). The game can run entirely in the browser via JavaScript while sending/receiving updates on the game state via WebSockets.

- [User accounts]
    - Each user's profile must track at least one statistic related to that user's performance in the game (eg. high score, number of wins)
    - Users must have a way to view this data for their own profile (eg. if they are not on the leaderboard they can still see their statistic on a profile page)
- [User Data] - Leaderboards
    - Your app must include at least one leaderboard that ranks users based on the statistic(s) stored in their profile. The leaderboard must be sorted so the highest performing players are clearly on top
- [Communication] - Lobbies
    - Multiple games can occur simultaneously and users must be provided a way to create, join, and start games
- [WebSockets] - Multiplayer
    - When users are playing the game, all moves and game state updates must be sent via WebSockets to enable a real-time multiplayer experience

eCommerce (auction house)

Make a storefront where users can buy and sell goods. Your store will also have an option for users to put items up for auction

- [User accounts]
    - Each user account must store the items that user has posted for sale, and all past transactions for that user (both sales and purchases)
    - Users must have a way to view their own data, but this data cannot be viewed by anyone except the buyer and seller
- [User Data] - Item Listings
    - Each user can create a new listing of an item for sale and purchase items
    - When a user creates a listing, they must provide an image, a description, and the price of the item
    - Each user can view all the items that are currently for sale along with their image, description, price
    - When an item is purchased, it should no longer be displayed to all users (or you can mark it as sold) and the buyer and seller should be able to view the transaction in their profile

- [Communication] - Shopping Cart
    - Users must have a way to purchase multiple items using a shopping cart
    - For each item that is listed for sale, there must be an option for users to add the item to their shopping cart. This cart must be able to store multiple items for each user
    - The shopping cart must be securely (users can't see other users' carts) stored in your database and persist through a both a server restart and a user logging off and back in
- [WebSockets] - Auctions
    - In addition to listing items for sale with a fixed price, users can also put items up for auction. An image and description must be provided, but instead of a price the user will specify when the auction should end
    - When an auction starts, users will have a way to join the auction
    - Each user in an auction will be able to see the current bid as well as place their own bid. Bids must be communicated in real time over WebSockets
    - When the auction ends, the user with the highest bid purchases the item
    - Multiple auctions must be able to take place simultaneously without interfering with each other

TopHat

Make a question site where instructors can create classes and post questions to their students

- [User accounts]
    - Users can view all of the courses they created, and for each course that they enrolled in they can see their grades. Only the user themselves and the instructor of the course can view a user's grades
- [User Data] - Courses
    - All users can create courses. When a user creates a course, they are the instructor for that course
    - All users can view every course that has been created along with their instructor and be given an option to enroll in each course
    - The instructor of a course must be able to view the roster (users who enrolled) for each of their courses
- [Communication] - Chat
    - Each course must have a way for users enrolled in the course to communicate to each other. This can be chat feature that shows on the page for each course
- [WebSockets] - Questions
    - Instructors can create questions and assign those questions to the class
    - When a question is assigned, the instructor will provide the amount of time for the question
    - When a question is assigned, each user who is enrolled in the course for that question can "join" the question
    - When a user joins the question, they can submit an answer as well as see a clock with the time remaining. Answers and clock information must be sent

via WebSockets (ie. Your server maintains the clock and sends the time remaining to all clients who joined the question)
    - Any answers submitted after time expires do not count
    - When time expires, the answers are graded. Each student can view their grades for each question and the instructor of the course can view all the grades in a gradebook
    - Grading must be automated (ie. when a question is created, the correct answer(s) must be provided)
    - Multiple courses must be able to have questions simultaneously, though a single course can be limited to one question at a time

Fantasy Sports

Make a fantasy sports app

- [User accounts]
    - Users can view their roster(s) as well as their performance throughout the season
- [User Data] - Rosters
    - For each league that a user joined, they must be able to view the rosters of all other users who joined that league. A user cannot view rosters for a league that they did not join
    - Users can also view the performance of each member of each league that they've joined
    - Your app must contain a list of all players available to be added to a roster
- [Communication] - Leagues
    - Your app must support multiple leagues. Any user can view, create, and join any league
    - Your app must have a way to provide results (player stats) and your app should compute the performance of each member of the league based on the provided results
    - Results can be uploaded multiple times (eg. After each week of the season) and total performance must be tracked
- [WebSockets] - Draft
    - For each league, the creator of the league can begin a draft
    - Once a draft starts, users can no longer join that league
    - When the draft begins, the members of the league will be ordered randomly
    - Users will select players for their roster in this order until all their rosters are filled
    - All draft communication must be via WebSockets to provide a live draft
    - Multiple drafts must be able to occur simultaneously

Security

Your site must be secure! If a vulnerability is found, the consequences will be decided based on the severity of the vulnerability. Severe vulnerabilities may result in a 0 for this phase of the project.

At a minimum, we will explicitly check for the following during grading:

- HTML/JS Injection (This is considered a severe vulnerability!)
- SQL Injection (Use prepared statements)
- Securing user accounts (If passwords are stored they must be salted and hashed properly. There must not exist a practical way for a user to authenticate as another user)
- Private content must be private (Do not assume users are using your site as intended. Your server must check that the user is authenticated before sending them private content even if your site doesn't offer a way to request that content without being authenticated)
- Keep your secret information secret. Do not push API keys, passwords, etc. to your repository