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

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!

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, unlike the homework which requires port 8080

When testing or deploying your app, the procedure will be:

1. Clone your repository
2. cd into to the directory of the repository
3. Create and run a docker containers (docker-compose up)
4. 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 and hosted using 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.

^ This is the only way we'll know where to find your app

Note that your deployment is only directly graded during the presentation, though it can help us grade your project features if there are any issues while running it locally.

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 must 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 3 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)
- [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 3 features below, however it must be a multiplier game. 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 value related to that user (eg. display name, avatar, number of wins, etc.)
    - Users must have a way to view this data for their own profile. They do not have to have a way to change this data (eg. display name, avatar can be chosen at account creation and never change)
- [User Data] - Lobbies or MMO
    - If the number of players for your game is limited, multiple games must be able to occur simultaneously. You are free to design the way users start multiple games as long as it is intuitive. Players should be able to start a new game even when other games are already in progress (ex. If you have a 2 player game, 10 users should be able to play 5 different games simultaneously)
    - Alternatively, you may build an MMO (Massively Multiplayer Online) game when any number of users can play in the same game simultaneously
- [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

Auction House

Make an auction house where users can put items up for auction and other users can bid on those items.

- [User accounts]
    - Each user account must store the items that user has posted for auction and all auctions that user has won/purchased
    - Users must have a way to view their own data
- [User Data] - Create Auction
    - Each user can create an auction for an item. When they create an auction, they must have a way to provide: An image of the item, a description of the item, the duration of the auction, the starting price of the item
    - Other users should be able to see that an auction has been created along with all the information for that auction. It is OK if this requires a refresh
- [WebSockets] - Auctions
    - Build the auction functionality
    - When an auction starts, users will have a way to join the auction. This can bring them to a new page for that specific auction, or you can display all auction on a single page
    - 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

    - Each user in the auction must see the current time remaining down to the second. This must be live. This can be communicated over WebSockets each second, or sent once from the server with front end JavaScript controlling the timer
    - When the auction ends, the user with the highest bid purchases the item and all users are notified that the auction has ended (Via WebSockets and something displayed on your front end)
    - 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
- [WebSockets] - Questions
    - Instructors can create questions and assign those questions to the class. When a question is created, the instructor must provide enough information for the question to be automatically graded (eg. a multiple choice question with the correct answer provided by the instructor)
    - Instructors must have a way to start and stop a question. Answers will only be accepted while the question is active. The start and stop messages must be sent via WebSockets
    - When a question is active, each user who is enrolled in the course can provide their answer for the question. These answers must be sent via WebSockets. If the question is stopped, users must be able to immediately see that the question was stopped (Via WebSockets).
    - Any answers submitted after the question is stopped do not count
    - When the question is stopped, 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
    - Multiple courses must be able to have questions simultaneously, though a single course can be limited to one question at a time

Fantasy Sports Draft

Make a fantasy sports draft app. This app will allow users the start a draft and create rosters, but does not have to implement points/scoring/game results/player stats/etc.

- [User accounts]
    - Users can view their roster for each league they've joined
- [User Data] - Rosters
    - Users must have a way to create and join leagues
    - 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 (If a draft has not occurred yet, the rosters will be empty)
    - Your app must contain a list of all players available to be added to a roster
    - Your app must support multiple leagues. Any user can view, create, and join any league
- [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 [for different leagues] 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