CSE 331: Software Design & Implementation

Homework Olympics (due Wednesday, August 14th at 11:00 PM)

In this assignment, you will implement the client and server portions of an application described below. This assignment consists entirely of coding work. Submit your final version to "HW Olympics" on Gradescope. Turning in your work for this assignment is a little trickier than usual, so follow these steps carefully!

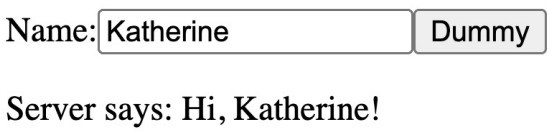
- Create an `extras.md` file in the same directory as the `/client` and `/server` directories. In this file, list any extra credit functionality that you implemented, this is the *only* extra credit that we will grade. A blank file indicates that you implemented no extra credit.
- `cd` into the directory that contains the `/client` and `/server` directories.
- Delete the `node_modules` directories from each directory. You can do this manually or use the command: `rm -r client/node_modules && rm -r server/node_modules`. The `rm` command will remove and the `-r` makes `rm` recursive, so `rm -r` will remove directories and all their contents.
- Generate a zipped file containing both of these folders and your extras.md file
  - **On Mac** you can run: `zip -r submission.zip client/ server/ extras.md` (The `zip` command compresses the specified arguments (`client/ server/`) into a `zip` file under the name `submission.zip` and again uses `-r` to recursively to get the full contents of the folders.)
  - **On Windows** you can select both folders, right click, and select 'Send to' > 'Compressed (zipped) folder'. You'll want to rename the zip that is created to 'submission.zip' so you can find it.
- Go to Gradescope and select the `submission.zip` file that was created in the last step
- Make sure you get all the autograder points! If you decide to work on your app some more after turning it in, you'll need to run `npm install --no-audit` in both of the directories again to get your `node_modules` back.

You can start by checking out the starter code using the command:

```
git clone https://gitlab.cs.washington.edu/cse331-24su/cse331-24su-materials/hw-olympics.git
```

Install the node modules, with `npm install --no-audit`, in both the `client` and `server` directories.

If you start the client and server and open `http://localhost:8080` in Chrome, you will see a dummy application that simply sends your name to the server when the button is pressed and shows the response message that comes back (which just says hi).



We included that code to give you a basic working example of client/server communication, but you can (and should) remove it when you start writing the actual application, which is described on the next page.

Read the entire spec before getting started. Understanding the requirements and making a comprehensive design is essential to being successful on this assignment.

Summer quarter is almost finished! You've decided that you're going to go to the Paris 2024 Olympics to celebrate all of the hard work that you've done in 331. You and your classmates have already bought your plane tickets, so now it is time to plan! There are lots of details to organize, one big task being deciding which events you're going to attend! You need some way to easily plan out your agenda, what better tool for this task than a web application?

Luckily, you all have been practicing building apps all quarter long! So you and your classmates decide to design and build an app to help plan the trip.

1. On Your Marks, Get Set, Node! (60 points)

Write a client side for your Olympics Trip Planning web application, meeting the "Required Functionality" detailed in the following pages.

When implementing your application, it is, as usual, a good idea to crash if the program has a bug, but you are not allowed to crash on invalid inputs from the user. (Those are not bugs!) You must either show an error message *or* prevent the user from submitting invalid inputs.

Thoroughly test your application. Like in HW Squares, the UI portions of this assignment are best tested with manual testing (which we will, of course, perform on your app), but any complex utility functions in your client code (think similar to problem 1 of HW Squares), should be tested with unit tests.

When grading we will be looking for high quality code (including organization, documentation, testing) and app functionality (including error handling).

Note that `let` variable declarations are not allowed on the client side.

2. 331m Dash (40 points)

Our client side app is great for adding all the event information, but the app data is not persistent. If the web page reloads, the whole event list will be lost! We also want to allow users to access the app from a different tab/window, so they can continue to reserve tickets and see event details. To enable this, we need to back up the event data to a server.

Implement a server with routes that allow you to store and load the necessary data. Change your client side app to store and fetch event information from the server.

Write unit tests and document your server. Again, we will be looking for high quality code while grading.

Note that, unlike HW Squares, you need to write all the client-server communication code (fetch requests) yourself. Since you are writing them yourself, you can integrate these requests into your components like we've shown in the lecture demos instead of placing them in a separate file as in HW Squares.

3. Extra Credit: Coding for Gold (0 points)

Add any new features that seem useful! You will get points for any feature that works correctly and seems like it would be valuable to the user.

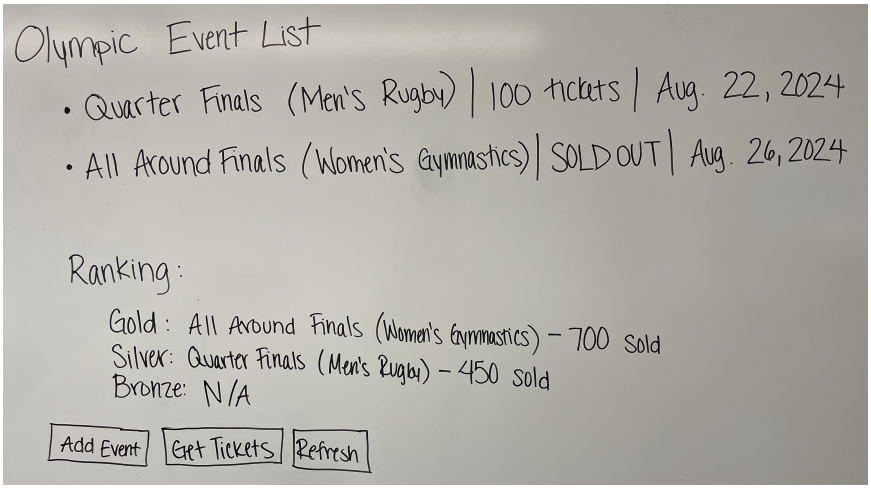
Note: adding CSS alone is not eligible for extra credit. You're definitely encouraged to add CSS, but extra credit will only be awarded to submissions that have additional *functionality* beyond the requirements.

Required Functionality

Katherine sketched out a design for the look of the Olympics Trip Planning app that she'd like you to implement. This section describes the minimum required functionality of the app and includes sketches of one possible way to provide that functionality to the user. It is fine to make changes that make the application *more* functional. Video demo of the app coming soon!

View Olympic Events

Users must be able to view a list of all future Olympic events. Each event name must be listed along with the sport's name, how many tickets are left (if there are none indicate that the event is "Sold Out"), and the date. The events in the list must be in chronological order (events on the same date can be in any order).



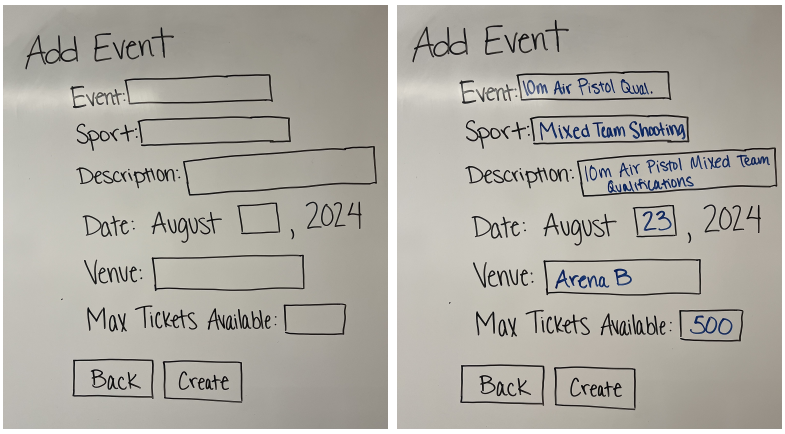
Along with the list of events, there should be a ranking of the top 3 events with the most ticket sales and, for each, the number of tickets that have been sold.

The "Refresh" button should refresh the event list (and each event's number of tickets left) and update the ranking.

(Continued on the next page. . .)

Add an event

As the Olympic committee adds events to the 2024 Olympic Games, you must also be able to add the events to your app! You must be able to add the event and sport name, a description of the event, the day (in August 2024) of the event, the event venue, and the max tickets available. If you decide not to add the event, you should be able to go "back" to the event list page and the event will not be added.



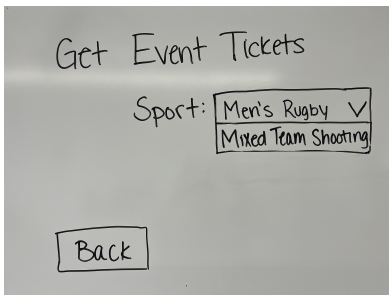
All of these details must be specified and valid in order for the event to be added (excluding the event description). To make validation easier, you may assume that the current date is August 1st, 2024. Otherwise, an error message should appear indicating what's invalid.

Ideally we would prevent duplicate events, but handling this case gets tricky, so we will not require you to handle it.

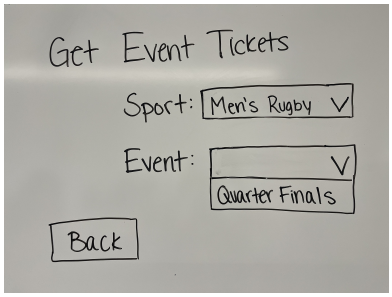
(Continued on the next page. . .)

Get event tickets

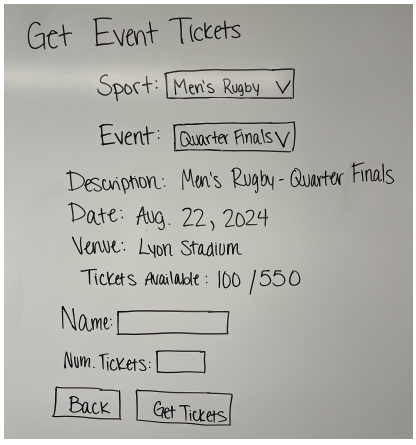
Once events have been added, you should be able to get tickets to any event. If there are no events, you should either not be able to navigate to the get event tickets view *or* the get event tickets view should indicate to the user that there are no future events. When there are events, the get event tickets view should be populated with a field to select the name of the sport that you'd like to attend. The sports that should be shown to the user should have events that have available ticket(s).

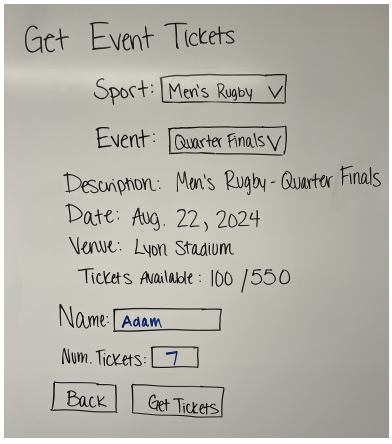


Then, once you select the sport, the view should be updated with an additional field to allow the you to select an event for that sport. The events that should be shown to the user should have available ticket(s).



Finally, once you select the event, all of the event details should be displayed, along with the fields for the reserver's name and the number of tickets you want to reserve.





The name and the number of tickets must be specified, and the number of tickets must be a positive integer that is $\leq$ the number of tickets remaining, for the tickets to be reserved. Otherwise, an error message should appear indicating what's invalid.

If you decide not to get tickets to an event, you should be able to go "back" to the event list page and the count of tickets left for each event will not be changed.

(Continued on the next page...)

Tips & Notes

- Sketch out your design on paper first to divide the app into components and map out how the data will be stored and flow between them *before* you code anything.
- Utilize your resources! The section and lecture code (especially `Auction`) and prior assignments (especially HW Squares) give you amazing examples to borrow from.
- For anything that involves ordering/ranking, you can decide how you would like to break ties!
- Create types and helpers to parse those types! Notice how in the lecture example, we defined a custom type for `Auctions` and created a `parseAuction` function to parse out each field of the record, verify that it was the correct type, and then format the data into an `Auction` object.
- Don't get stuck on the HTML, we care more about testing your skills in other areas, not your knowledge of HTML. If you're unfamiliar with which elements you need to render what you see in our app outline, **please ask**.
- If you are running short on time, you can exclude the event date, sorting the event list, the event sport name, and/or the event sport name selection in the get event tickets view, and only receive a small deduction (per attribute skipped) compared to skipping other features.
- If you are *really* short on time, just focus on the client-only version of the app from problem 1. The client-only version is worth 60% of the points!
- **Have fun and start early!** Students often enjoy the freedom in this assignment. While this is an opportunity to use your skills, there are also many potential bugs. Give yourself time to deal with those bugs, as well as enjoy it.

Congratulations on completing your last app of 331!! You have designed and implemented a functional client-server application starting from only sketches, way to go!

Snoop Dogg cheering you on! Photo credits: BBC