CSE 331: Software Design & Implementation

Homework 9 (due Friday, December 8th 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 "HW9 Coding" on Gradescope. Turning in your work for this assignment is a little trickier than usual, so follow these steps carefully!:

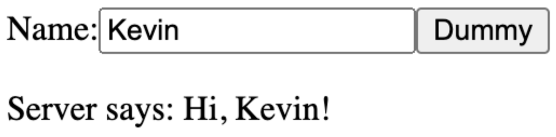
- `cd` into the directory that contains the `/client` and `/server` directories.
- Delete the `node_modules` directories from each directory (you can do this manually or use the command: `rm -r client/node_modules && rm -r server/node_modules`).
- Generate a zipped file containing both of these folders On Mac you can run: `zip -r submission.zip client/ server/`. 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'.
- Go to Gradescope and select the `submission.zip` file that was created by running the last command.
- 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/cse331-23au-materials/hw-polls.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).



Server says: Hi, Kevin!

We included that code to remind you how client/server communication should look, but you can (and should) remove it when you start writing the actual application, which is described on the next page.

1. For Whom The Bell Polls (100 points)

Your friend Cassandra, who just returned from a hike up to Delphi, told you that she had a dream where she learned that tomorrow is your unlucky day. Every decision you make, she says, will turn out badly!

To escape your fate, you decided to avoid decisions altogether by outsourcing them to your friends. Any time you need to make a decision, you will poll your friends and pick the option they like best. Crisis averted! But first, you'll need to write an app to poll your friends for each of your decisions.

Write a web application, with both client and server parts, that allows one user to create and then multiple users to participate in a poll. The required functionality is spelled out the next few 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 homework 8, there are UI portions of this assignment that are best tested with manual testing (which we will, of course, perform on your app), but other areas can and should have unit tests written for them and turned in with your work.

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

2. Extra Credit: Feeding Time at the New (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.

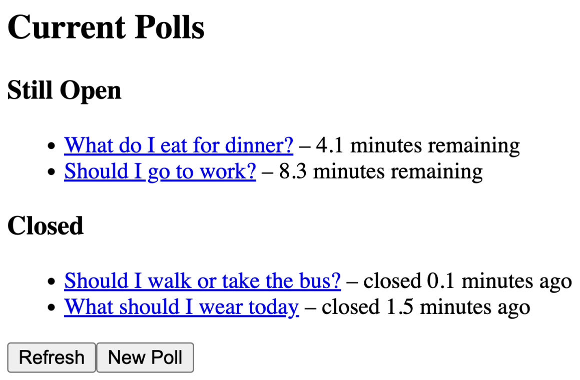
Required Functionality

This section describes the minimum, required functionality, along with screenshots with one way of providing that functionality to the user. You do not need to make your application look exactly like this, nor must the functionality be identical. It is fine to make changes that make the application *more* functional. **This video also includes a walk-through of what our final product looks like**.

View existing polls

Users must be able to view all polls that have been created, ongoing and completed. They should be able to click on each poll listed to open it and see either the results (if the poll has completed) or vote in it (if the poll is ongoing).

Each poll must be listed with the amount of time remaining or the amount of time that has passed since it has completed. The ongoing polls should be listed in order of increasing time remaining and the completed polls should be listed in order of increasing time since completed (as shown below). However, if you are running short on time, listing all the polls together in the order they were created instead of ordering them separately would only result in a small point deduction.


Current Polls

Still Open

- What do I eat for dinner? – 4.1 minutes remaining
- Should I go to work? – 8.3 minutes remaining

Closed

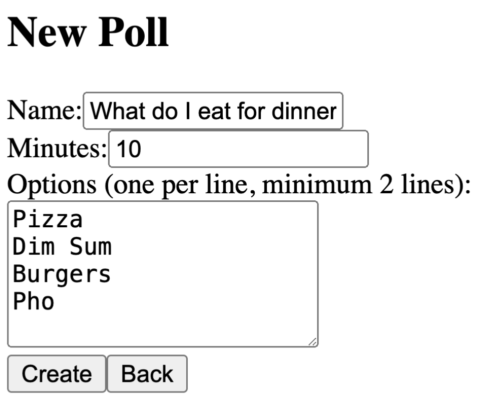
- Should I walk or take the bus? – closed 0.1 minutes ago
- What should I wear today – closed 1.5 minutes ago

(Continued on the next page...)

Create a New Poll

Users must be able to create a new poll, specifying the list of poll options available to vote from and the duration the poll should run for.

No two polls should have the same name, however, handling this case can be a little tricky, so if you are running short on time, skipping handling this case would only result in a small point deduction.


New Poll

Name: What do I eat for dinner

Minutes: 10

Options (one per line, minimum 2 lines):

```
Pizza
Dim Sum
Burgers
Pho
```

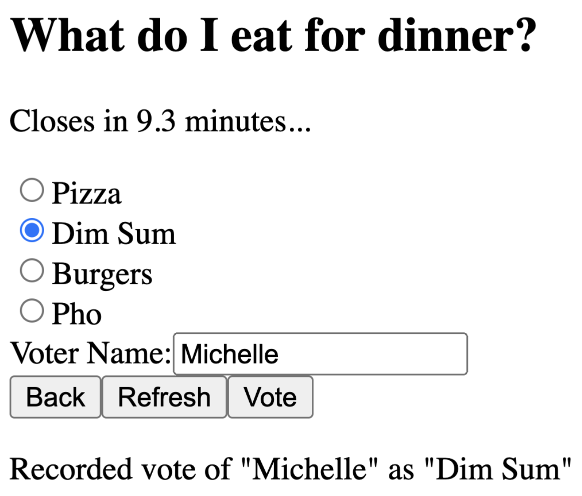
The following html may be a useful example for syntax to create a text area with a set size and a number input area (which is what we used in our solution though not the only possible approach). It uses hardcoded values so it would not currently change if you tried to type in the input areas.

```
<div>
    <label htmlFor="textbox">Enter text:</label>
    <br/>
    <textarea id="textbox" rows={3} cols={40} value={"Change to allow input!"}
        onChange={() => console.log("text updated")}></textarea>
</div>
<div>
    <label htmlFor="num">Enter num:</label>
    <input id="num" type="number" min="5" value={5}
        onChange={() => console.log("num updated")}></input>
</div>
```

(Continued on the next page...)

Vote in an Existing Poll

Users must be able to vote in ongoing polls by picking from the poll options. Each user can vote as many times as they'd like while the poll is ongoing, but only their most recent vote will be recorded (i.e. only one vote per distinctly named user).



What do I eat for dinner?

Closes in 9.3 minutes...

- ○ Pizza
- ● Dim Sum
- ○ Burgers
- ○ Pho

Voter Name: Michelle

Recorded vote of "Michelle" as "Dim Sum"

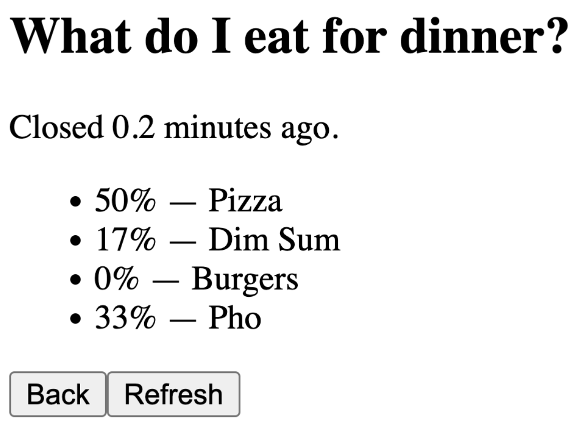
The following html may be a useful example for syntax to create a list of selectable radio items (only one can be selected at a time) which is similar to what we used in our solution (though not the only possible approach). It uses hardcoded values so it will not actually change when items are clicked as is.

```
<ul>
    <div key={"1"}>
        <input type="radio" id={"item 1"} name="item" value={"item 1"}
            checked={true} onChange={() => console.log("item 1 clicked")}/>
        <label htmlFor={"item 1"}>item 1</label>
    </div>
    <div key={"2"}>
        <input type="radio" id={"item 2"} name="item" value={"item 2"}
            checked={false} onChange={() => console.log("item 2 clicked")}/>
        <label htmlFor={"item 2"}>item 2</label>
    </div>
    <div key={"3"}>
        <input type="radio" id={"item 3"} name="item" value={"item 3"}
            checked={false} onChange={() => console.log("item 3 clicked")}/>
        <label htmlFor={"item 3"}>item 3</label>
    </div>
</ul>
```

(Continued on the next page. . .)

View poll results

Once the poll is over (the set poll duration has elapsed), users must be able to see the percentage of votes each option in the poll received.


What do I eat for dinner?

Closed 0.2 minutes ago.

- 50% — Pizza
- 17% — Dim Sum
- 0% — Burgers
- 33% — Pho

Each "refresh" button shown in the above sections should update the poll countdowns, and the "back" buttons should returns the user to the page of the app they were on previously.

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