CSE 331: Software Design & Implementation

Section 8

In this section, we will look at an application that includes both a client and a server. To get started, check out the starter code using the command

```
git clone https://gitlab.cs.washington.edu/cse331-23sp-materials/sec-squares.git
```

To install the modules `cd server` and run `npm install --no-audit`, then `cd ../client` and run the same. Then, `cd server` and start it with `npm run start`. Then, `cd client` and do the same. The server running in the `client` directory is serving JavaScript that runs in the browser, while the server running in the `server` directory is running the server-side JavaScript. Point your browser at `http://localhost:8080`. You should see the same "Animal Trivia" application from Section 6, now rewritten to use React for the client portion.

1. All I'm Asking Is For a Little Correct

Currently, when you submit your answer, nothing changes in the UI.

Open the Chrome developer tools, navigate to the "Network" tab, and refresh the page. Verify that the client is making a `/new` request to the server to check the answer and is receiving a response but is not updating the UI to show the result.

(a) Add a `debugger` statement at the top of the `handleCorrectness` method in `App.tsx`. Confirm that the method is being called with the result of the correctness check.

(b) The `handleCorrectness` method updates the state to include the field called "`correct`", but this information is not currently being used in render.

Update `render` to show some HTML (either a congratulatory message or one saying that the answer is incorrect) when `this.state.correct` is not undefined.

Verify that the HTML now appears on the screen when you submit an answer.

(c) Augment the HTML from part (b) to include a button that lets the user get a new question like this:

```
<button type="button" onClick={this.fetchNewQuestion}>New Question</button>
```

Verify that you can now get a new question after submitting the answer to the current one.

2. Click-state

Add a call to `console.log(JSON.stringify(this.state))` at the top of the `render` method in `App.tsx`. Then, run the application through one cycle from the initial page to a new question. You should see the state printed out multiple times in the browser console.

Draw a diagram showing the sequence of states that were printed out and what events (e.g., a button click or a fetch response) caused each transition between states. No need to write down states that are `undefined`, unless you find it useful to keep track of.

Each character typed into the answer box will generate an update to the state. You don't need to write out every one of those states. Just write the last one.

3. Take Out the Papers and the Flash

You will notice that, when we click on the button for a new question, the UI flashes. As you saw in the previous problem, this occurs because we are redrawing the UI twice: once immediately when the button is clicked and a second time when the fetch completes.

Change the application to skip updating the state after the button click.

Verify that this eliminates the flashing.

4. How Do I Love Tree, Let Me Count the Ways

Recall our definition of a binary search tree from HW4:

$$\begin{aligned} \textbf{type}\ \text{BST} := \ & \text{empty} \\ | \ & \text{node}(x : \mathbb{Z},\ S : \text{BST},\ T : \text{BST}) \quad \text{with conditions A and B} \end{aligned}$$

Suppose that we wanted to have a way to refer to a specific node in a BST. One way to do so would be to give directions from the root to that node. If we define these types:

$$\begin{aligned} \textbf{type}\ \text{Dir} &:= \text{S} \mid \text{T} \\ \textbf{type}\ \text{Path} &:= \text{List}\langle\text{Dir}\rangle \end{aligned}$$

then a Path tells you how to get to a particular node where each step along the path (item in the list) would be a direction pointing you to keep going down the left (S) or right (T) branch of the tree.

For example, $\text{cons}(\text{S}, \text{cons}(\text{T}, \text{nil}))$ says to select the "S" (left) child of the parent and then the "T" (right) child of that node, giving us a grand-child of the root node.

(a) Define a function "find(p : Path, T : BST)" that returns the node (a BST) at the path from the root of T or undefined if there is no such node.

(b) Define a function "remove(p : Path, T : BST)" that returns T except with the node at the given path replaced by empty.