Practice Challenge: WestWorld Command Center

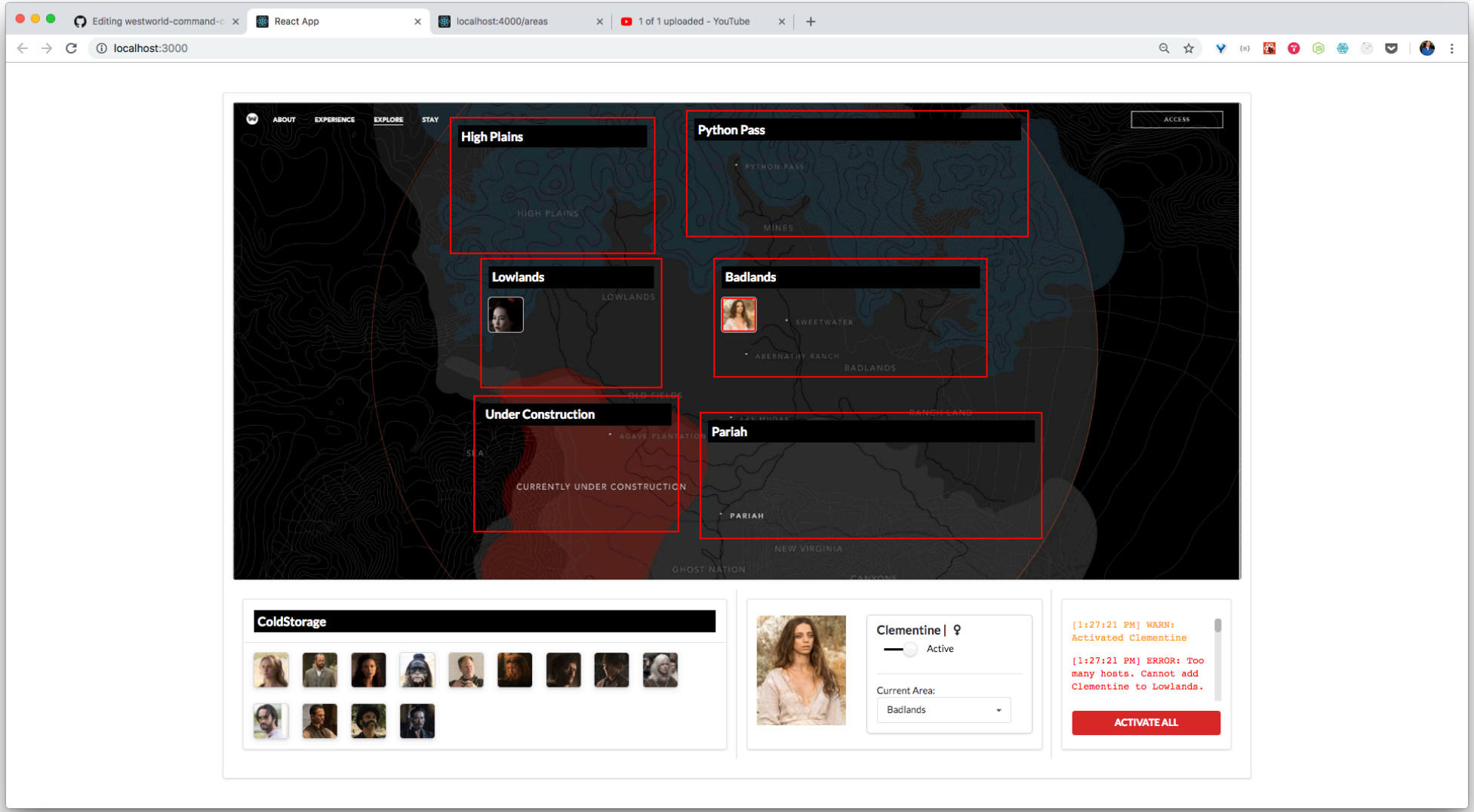
(https://github.com/learn-co-curriculum/react-hooks-practice-westworld-command-center) (https://github.com/learn-co-curriculum/react-hooks-practice-westworld-command-center/issues/new)

Overview

Note: This is a very extensive exercise, much longer than the typical practice challenge! Save this until you're feeling very confident with React and want a larger project to test your skills.

The Executives at Delos Inc. need you to help them build some software for their new theme park: WestWorld. WestWorld is an interactive theme park where guests get to experience life in the Wild Wild West with the help of some AI known as "Hosts". But WestWorld needs a way to deploy these hosts to different areas of the park and bring them back to "Cold Storage" where they can be repaired or retired.

Your job is to create a React based interface that allows you to select Hosts, activate them, and send them to any area of the park or call them back to Cold Storage. Your application should look something like this when you're finished:



A Note on Styling

The styling is a mix of pre-written CSS and Semantic components. Don't worry about it too much. As long as you're using the classNames and id's we suggest everything should be fine. If you have a question about how one of the Semantic components works, search for the component

in the Semantic docs for a complete run down:

Semantic UI React Docs (https://react.semantic-ui.com/)

Setup

Watch a walk through of what's expected to complete this challenge here:

https://youtu.be/GhCazAgsJzw (https://youtu.be/GhCazAgsJzw)

(https://youtu.be/GhCazAgsJzw)

NOTE: The original version of this practice challenge was written before hooks were introduced to React, but you can achieve the same functionality you see in the original demo nonetheless

All the data about can be found in the `db.json` file. We'll be using `json-server` to create a RESTful API for our database.

Run `npm install` to install our dependencies.

Then, run `npm run server` to start up `json-server` on `http://localhost:3001` .

In another tab, run `npm start` to start up our React app at `http://localhost:3000` .

With `json-server` running, you'll have access to these two endpoints:

- `http://localhost:3001/hosts`
- `http://localhost:3001/areas`

Deliverables

The components and styling have already been given to you. It'll be your job to import the components in the right order to build the component tree correctly and add most of the logic. Any conditional styling will be given via changing classNames. For example, if styling on a button should be changed based on a click, you'll be given two classNames to swap in depending on what the current status of that button is.

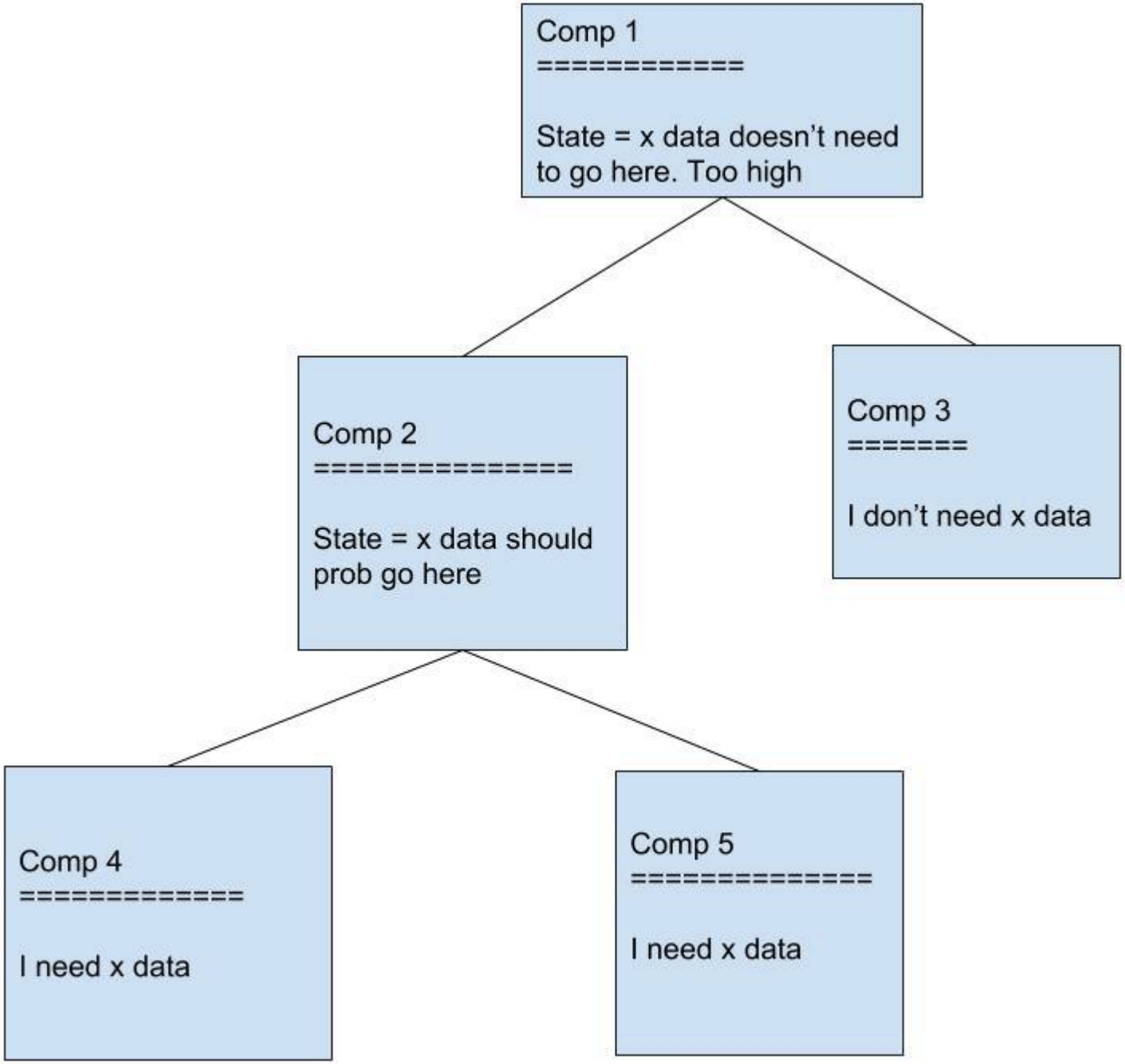
Checkpoint 1: Build the Component Tree

Determine how the component tree should be built. Some of the imports have already been given for you. Before you get started, it is highly suggested to draw your component tree on paper. A couple things to note:

1. Let the visual cue of the application guide you. For example, there are clearly two main sections to this application: The top half (`WestworldMap`) and the bottom half (`Headquarters`). How should each of those components import the components that live inside them?
2. Aside from visual cues, what functional cues can you get from the application? For example, clearly the `Area` component holds hosts in a type of list. So what component does an area need to render that list? Is there another component that also holds hosts in a list that's not an area component?
3. Remember that two separate component branches can import the same component.

Checkpoint 2: Determine Where State Lives

You're going to be fetching information from two endpoints. Where should you be doing that and where should that data live once it's been fetched? Remember the rule: hold state as high as necessary but NO HIGHER. For example:


Comp 1
==========
State = x data doesn't need to go here. Too high
Comp 2
==========
State = x data should prob go here
Comp 3
=======
I don't need x data
Comp 4
==========
I need x data
Comp 5
==========
I need x data

Checkpoint 3: Render the Areas

Area info comes in through the `/areas` endpoint. You'll have to use that to render the right number of area components on the map. Styling is given for you, but you'll have to pass the area name to the `id` attribute to make it appear in the right place on the map. Format the name to remove underscores and capitalize all words for the label. Ex: `high_plains` should be displayed as "High Plains"

Checkpoint 4: Render the Hosts

The `Host` component represents a host Thumbnail. You'll have to render the appropriate number of hosts based on the data fetched from the `/hosts` endpoint with the appropriate imageUrl for each. You'll also need to make sure they render in the right place. Note that their `active` attribute is set to `false`, meaning they come in decommissioned. Decommissioned hosts should always appear in `ColdStorage` no matter what their `area` attribute is set to.

Checkpoint 5: Host Behavior

Follow these rules for selecting and moving hosts:

1. Clicking a `Host` selects them with a red border and displays their information in the `HostInfo` component. Styling has been given via `classNames` (see Host component).
2. Only one `Host` can be selected at a time.
3. Only one `Host` can exist on the screen at a time. If they're in `Cold Storage` then they're not on the `WestworldMap` and vice versa.
4. If a host's `active` attribute is set to `false` then they are decommissioned and should appear in `ColdStorage`. The `HostInfo` radio button should reflect this as well, reading "Active" if `active: true` and "Decommissioned" if `active: false`.
5. The Area dropdown should be pre-selected with the area the host is currently in, even if they are in `ColdStorage`.
6. If a host is Active, selecting a new area from the dropdown should move that host to the corresponding area. If the host is Decommissioned they should not be able to leave `ColdStorage`, but their `area` attribute/dropdown should still update.
7. Setting a host's toggle to Decommissioned should immediately remove them from their area and place them in `ColdStorage`.

Checkpoint 6: Limit Hosts

Each `Area` should only allow the number of hosts given by that area's limit attribute. This includes hosts set to areas in `ColdStorage`. This is a hard deliverable and there are many ways to do this. Think about where you should actually be blocking this action (ie. what component should the rejection happen in).

Checkpoint 7: Activate All/Decommission All

The Activate All/Decommission All button is located in the LogPanel component. If you want, you can extract this into a separate component that LogPanel imports, but it's not necessary.

Clicking the `Activate All` button should activate all hosts. The button should turn green and change to read `Decommission All`. Clicking the `Decommission All` button should decommission all hosts and the button should change red and read `Activate All`. Remember, if all hosts are activated, this should be reflected in a host's activate toggle.

Checkpoint 8: Logging

Last but not least, you should log the actions a user takes. Use the Log service class we've provided (located in: `src/services/Log`). To use the class all you need to do is invoke a particular method (take a look at the class to see what methods are available) and send in the message you want to log as an argument. Don't worry about the styling, that's taken care of. For example, if you want to log an error saying "Something bad happened" you would write:

```
Log.error("Something bad happened")
```

This would return the following object:

```
{type: 'error', msg: '[9:00pm] ERROR: Something bad happened"}
```

You should collect these in some type of array somewhere and give it to the `.map` statement in the `LogPanel` component to get them to render. These should render most recent first (so the first element in the array should have the most recent time stamp).

At the very least you should be logging the following:

1) Setting a host's area:

```
Notify: {first name of host} set in area {formatted area name}
```

2) Activating a host:

```
Warn: Activated {first name of host}
```

3) Decommissioning a host:

```
Notify: Decommissioned {first name of host}
```

4) Activating all hosts:

```
Warn: Activating all hosts!
```

5) Decommissioning all hosts:

```
Notify: Decommissiong all hosts.
```

6) Trying to add too many hosts to an area:

```
Error: Too many hosts. Cannot add {first name of host} to {formatted area name}
```

Finish

If you've completed all the Checkpoints, good for you because that is a ton! It is very rare that people are able to finish this in an all day pairing/solo attempt. It would be awesome if you could share the way you solved it!

Contributing

If you find any bugs or have some suggestions, send a PR and we'll try to incorporate it!