Intro to React Ottergram App

1 Generating Ottergram React Project

We will generate the React app for this project using Create React App (CRA) in the same way that we have done for all our previous labs.

In the top-level folder that you created for this workshop, open a command prompt and generate a new project using CRA with:

```
create-react-app ottergram
```

If you already have an existing development server running on your machine from another existing active React project, then you will need to specify a different port for the development server of CRA for this project.

Open a command prompt in this project folder. Here, we use an alternative port of 5000, but you can use any port that is free on your local machine:

```
SET PORT=5000
```

```
npm start
```

The newly generated app should provide the standard default view.

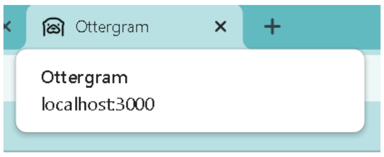
Create another instance of VS Code (File -> New Window) to open this project folder

2 Basic app view

Folder to use: `ottergram-app`
File to use: `App-v1.js`
`index-v1.html`

Folder to use: `ottergram-app/resources`
File to use:
`favicon.ico`
`logo192.png`
`logo512.png`

We update the header in the browser (index.html) and also place our own custom favicon and related images for this app, which should now show up in the browser tab.



3 Adding posts with images

Files in the `public` subfolder are not part of the WebPack build process, and so it will not be fully optimized in the build process that WebPack performs. To allow this optimization we need to place all our image files in the `src` subfolder.

In the `src` subfolder of the current React project, create a new subdirectory called `otters`.
Copy over the 4 PNG files contained in `ottergram-app/resources/otters` to this newly created subdirectory.

Folder to use: `ottergram-app`
File to use: `App-v2.js`

4 Styling posts

Folder to use: `ottergram-app`
File to use: `App-v3.js`
`App-v3.css`

Here we add the relevant style rules to App.css and set the corresponding class names in the various elements in App.js. Note that in the JSX that we return, we use the `className` attribute (instead of the normal `class` attribute that we would use for standard HTML / CSS).

Notice that we using flexbox to perform layout in the CSS style sheet. Flexbox is one of the techniques used to implement responsive web design (RWD), so you can check out how this app renders in various devices using the features for simulating mobile devices in Chrome Dev Tools:
https://developer.chrome.com/docs/devtools/device-mode/

5 Refactoring into smaller components

We will typically start developing an app by implementing all our UI and business logic within a single component (the default root component App). However, at some point, as the app starts to grow in complexity, it makes sense to decompose it into a series of smaller components that can be subsequently combined together in a special way general the overall app view.

Create a `components` subfolder in the `src` directory.

Folder to use: `ottergram-app`

Files to use: `Header-v4.js`
`Post-v4.js`
Folder to place in: `components`

File to use: `App-v4.js`
Folder to place in: `src`

Notice that these new child function components starts with a capital letter so that when it is embedded in JSX to be returned from the root component App, React can distinguish it from the standard built-in component corresponding to normal HTML elements such as <header>
https://www.w3schools.com/tags/tag_header.asp

Even though you did not import the stylesheet directly into the new `Header.js` file, the style rules are still being applied here. This is because all CSS styles are global by default in React App, which means they apply to all components of the app even though they may only be imported by a particular component (in this case, <App>). You can define separate stylesheets for each component if you want as well.
A good rule to use is to ensure that CSS selectors are unique throughout the components of your application. As long as you do not have components with duplicate names, using the component's name as the class prefix usually works well.

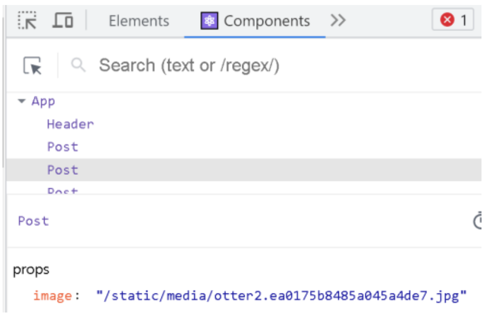
6 Passing props to components

Folder to use: `ottergram-app`

Files to use: `App-v5.js`
`App-v5.css`
`Post-v5.js`

We can further generalize the app by passing the relevant info that the Post component needs (such as the name of the Post and the image associated with it) to it from the parent App. We can do this for all the images since the Post UI is going to display the same behavior for all otters that are being displayed.

If you have already installed the React Developer Tools from a previous lab, you should be able to see the various properties of the `props` object being passed to the respective child components.



```
image: "/static/media/otter2.ea0175b8485a045a4de7.jpg"
```

Notice that the image name that is being passed as one of these properties references the actual jpg file in a different location and with a different name than the original images in `src/otters`. This is just part of the optimization that the WebPack build process performs on all files in the `src` subfolder to ensure the most optimum and smallest JavaScript bundle is delivered to the browser on the initial load.

You can try changing the `name` property in the Components panel view and see the app update dynamically. This demonstrates how the React Developer Tools can be used experiment with changes to your React app in real time. Once you refresh / reload the app, these changes will be gone.

We can make a slight change to the Post component to use object destructuring to simplify the syntax for obtaining the relevant properties from the `props` object.

Files to use: `App-v6.js`

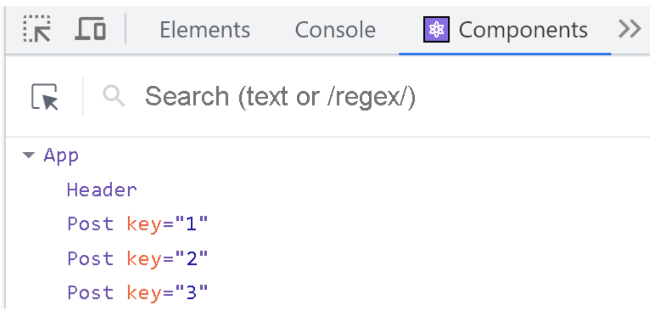
7 Rendering a list from items in an array

Folder to use: `ottergram-app`

Files to use: `App-v7.js`

We can store all the information that we wish to pass down as props to the various Post components in an array and then use the `map` array function to place them appropriately into an appropriate array of <Post> components. Since Post is a child component of a top-level array returned from the map function, it is recommended to provide it a key value to uniquely identify it. React uses keys so that when the component in the list is added, removed, or changed, it can render only that item and not the full list.

Each key must be unique among its sibling components and must be consistently associated with the same component. It is generally not a good idea to use the array index as the value for the key since the key value should be fixed throughout the duration of the app life time, and the array indices for a specified component will change if components are inserted into the array or deleted from it.



```
▾ App
    Header
    Post key="1"
    Post key="2"
    Post key="3"
```

Note the although the key value appears in Post component in the Components panel of React Dev Tools in Chrome, it does not actually appear in the DOM tree since it is not accessed or used in the Post child component itself. The key is a feature that React uses to optimize performance and hence it is not part of the JSX returned from the component.

8 Handling events

Folder to use: `ottergram-app`

Files to use: `SelectedItem-v8.js`
Folder to place in: `components`

Files to use: `App-v8.js`

We first create another new component to display a single Otter image and access this from the main root App component.

Files to use: `Post-v8.js`

Next we can add a basic event handler in the Post component to do a basic console log when the button holding the image in it is clicked. Experiment with this in the browser view with the Console Panel active in Chrome Developer tools.

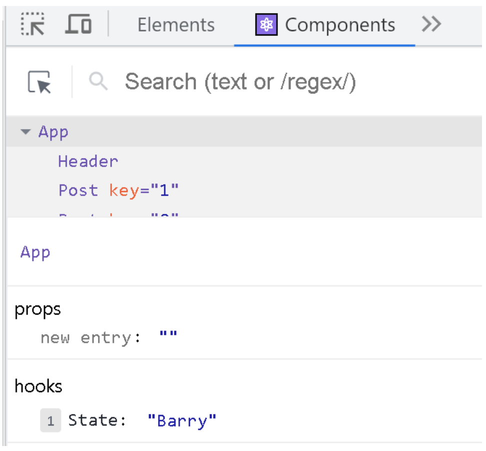
9 Maintaining state

Folder to use: `ottergram-app`
Files to use: `Post-v9.js`
`App-v9.js`

We use the `useState` Hook to provide a state variable that keeps track of the current selected otter name and a corresponding state setter function. We use another supplementary variable to obtain the image related to that otter. The state setter function is then passed as props (with the same name) to the Post child component, where clicking on the in-built Button component will trigger it and select the specific otter name of that Post component.

Clicking on the button corresponding to the various Post components will now make the image for that Post render on the right hand side of the app as the SelectedItem.

Notice now that the state for the App component is visible in the Components panel of React Dev Tools in Chrome.



```
▾ App
    Header
    Post key="1"

App

props
  new entry: ""

hooks
  1 State: "Barry"
```