20. React Developer Workflow

Github Repository | Vimeo Video Recording

Topics to cover

- [x] Development environment review/best practices
- [x] React development tools
- [x] How to develop a React project?
  - [x] Create a project with Create React App
  - [x] How to understand and clean the initial folder
  - [x] Convert interface designs to component-based UI
  - [x] Components and JSX
  - [x] Using Mock Data
  - [x] Props
  - [x] Inputs
  - [x] Event Handlers
  - [x] State
  - [x] Handling state between parents and children

Development environment review

Even though it's possible to create a React project that has a web server, JavaScript transpilation and other features from scratch, it's a lot of work to build and maintain it.

The one we will use in the bootcamp is `create-react-app`, it contains Babel, Webpack and a lot of other tools to make sure we can create a proper development environment for our future app.

However, `create-react-app` is not the only package out there, here is some other that you may see out there.

- Vite (`npm create vite@latest`)
- CRA (`npx create-react-app nameOfApp`)
- Next App (`npx create-next-app nameOfApp`)
- ...

At the end of the day, all of those options will achieve the same goal!

React development tools

Working with a lot of components and different JavaScript files can be confusing and overwhelming sometimes, especially if we try to debug using console.log(). Since React re-renders everytime the state values changes, it can be hard to keep track of which console.log is which.

Enter React Developer Tools, a pretty cool extension that will help us debug and interact with our components.

More information here: https://react.dev/learn/react-developer-tools

Project best practices

When building a project with a library like React, we do not have specific instructions on organization, naming, what type of function, etc.

- How should we structure our files?
- How should we name our components?
- Where do we put mock data?
- Should the frontend and the backend be in two different repos?

Component-based development

It's possible to do a React project with one or two components, but often we will have between 5-15 components for a small project. Trying to make them all work together while integrating new ones can be a nightmare. One of the solutions that we can implement is to build components in isolation.

By running them one at a time, we can make sure that a component works by itself first, then we can integrate it in the component tree. It's possible for small projects to work in isolation without extra tools or packages, but we may see other solutions when working on future projects after the bootcamp.

React file structure/best practices

There isn't a specific instruction on organization, naming props, and even components. There are the general accepted practices. Such as leaving your data in a folder named `data`. Any images being rendered in your components should be stored inside of the folder `src`. If the images are not being used by the components, then they can safely be stored in the folder `public`. All images should be organized in one location so it's easier to keep track of where they are located. It can a folder named `images` or `assests`. Naming components should revolve around the purpose of the component

Component-based development

The two most important questions we should answer during development is what does our data look like, and what is the purpose of this component. Diving into development of our components without having an idea of what data the component is supposed to look like will make our work more challenging. By understanind this idea, you will have a better idea of the structure of the HTML, and the purpose of the component.

Each component should serve a single purpose. We can make a React app with a few (1-3) components but it will be come a wall of code. To keep things organized, and easier to understand, we should develop our app based on each component having its own purpose.

Example:

- A component to render a single Grocery List Item

```
function GroceryListItem(props) {
    return (
        <li>
            <h2>Item name:{props.item}</h2>
            <h2>Quantity: {props.quantity}</h2>
        </li>
     );
}
```

- Another component that houses all the Grocery List Items

```
function GroceryList(props) {
    return (
        <ul>
            <h1>Bryan's Grocery List</h1>
            {props.groceryList.map((element) => {
                return (
                    <GroceryListItem
                        key={element.id}
                        item={element.item}
                        quantity={element.quantity}
                    />
                )
            })}
        </ul>
    );
}
```

View on Compass | Leave Feedback