A React App Example

New Attempt

Due No Due Date **Points** 1 **Submitting** a website url

FORK (https://github.com/learn-co-curriculum/react-hooks-react-app-example/fork) (https://github.com/learn-co-curriculum/react-hooks-react-app-example) (https://github.com/learn-co-curriculum/react-hooks-react-app-example/issues/new)

Learning Goals

- Understand the folder structure and key files for running a React application
- Explore the structure of a basic React component
- Debug React components using the React Developer Tools

Introduction

Before we dive in and start working on specific parts of a React project, let's take a look at a simple React application so we can understand it at a high level.

In this lesson, we will start with a basic React app, examine the default directory structure, and gain a better understanding of how these apps are built.

Starting Up a React App

This lesson has all the necessary parts of a fully working React app. To check the app out, fork and clone this lesson onto your computer, navigate into the lesson's directory, and run:

```
$ npm install
```

This will get and install all the required dependencies for React.

Next, we need to start up a server for the app to run on:

```
$ npm start
```

This will host the app and open a browser window to display it. If the server started correctly but the browser doesn't open, you can use the links that appear in the terminal to access the app. They should look something like this:

```
Local:            http://localhost:3000
On Your Network:  http://192.168.1.5:3000
```

Help

You can use the `Local` link to open the app in your own browser. The second is for any *other* computers on your network that you want to access your app from (this is particularly useful if you want to test out your app in a mobile browser on your phone).

If everything has worked correctly, you should see a page with the exact time it was loaded, along with a small amount of text and a GIF.

If we make any changes to our app while the server is running, it will 'hot reload,' and update the app in the browser. If there are app-breaking errors in your code, the browser will display those errors instead.

We'll start by exploring the JavaScript code for this sample app in the `src` directory.

index.js

The "entry point" into our application — the first JavaScript code that will run when our app starts up — is in the `src/index.js` file. Open that file in your text editor. Inside, you'll see something like this:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./components/App";
import "./index.css";

ReactDOM.render(<App />, document.getElementById("root"));
```

We'll talk about the `import` statements in a bit, but for now, let's have a look at `ReactDOM.render()` .

This function comes from the `react-dom` npm package. It takes in two arguments:

- A **React component** to render (typically, we'll render our top-level `App` component here).
- A **DOM element** where we want that component to be rendered (by convention, a `div` with the ID of `root`).

`ReactDOM.render()` will always be used in your applications. This one small function is how the rest of our application — all the components we'll write — will eventually make its way onto the DOM!

Even though React is a modern, complex framework, it still relies on a regular `index.html` file to load the JavaScript! The file can be found in the `public` folder. Take a look at it now.

In general, when you're given a React project to work with, it's a good idea to start by reading the `index.js` file then work your way down from there into the rest of the components.

App.js

Next, open up `src/components/App.js` in a text editor. This file contains our `App` component. Within the `App` component is a section of code that looks *very much* like HTML:

Help

```
<div className="App">
  <h1>{format(new Date(), "MMMM do yyyy, h:mm:ss a")}</h1>
  <p className="App-intro">
    In React apps, we write JSX - it looks like HTML, and uses a lot of HTML
    syntax. JSX lets us include JavaScript functions right along with the HTML,
    and also allows us to add in components, which are separate, self-contained
    chunks of JSX.
  </p>
  <ExampleComponent />
</div>
```

There's also some *JavaScript* code mixed in with this HTML-like syntax: `format(new Date(), "MMMM do yyyy, h:mm:ss a")`.

As it turns out, this is actually *all* JavaScript. This syntax is called JSX. It lets us write code that looks nearly identical to HTML, but allows us to mix in vanilla JavaScript and other neat things.

Reading through the JSX code, we've got one `div` that contains three child elements, `<h1>`, `<p>` and `<ExampleComponent />`. In your browser, *these* are the elements being displayed! The `<h1>` provides a timestamp of the exact time the app was loaded, and the `<p>` section includes the brief text on JSX.

The `ExampleComponent` contains the sunglasses GIF. In the `src` folder, take a look at `ExampleComponent.js`. You'll see a file very similar to `App.js`, containing `<img>` and `<p>` elements.

By including `<ExampleComponent />` in `App.js`'s JSX, we are able to use the contents of the component. If you copy and paste `<ExampleComponent />` so it is listed two times in a row, *two* GIFs will appear on the page. Try this now.

What about the rest of `App.js`, though? Moving out from the middle, we see this JSX code is the *return* value of a function called `App`:

```
function App() {

  return (
    // JSX goes here!
  )
}
```

The key thing to understand is that all of the *visible content* of our app is returned from this `App` function.

We've already seen that it is possible to have multiple files that contain visible content, i.e., by using both `App` and `ExampleComponent`. `ExampleComponent`, however, is used within `App`. `App` is at the top-most level; it is the *parent component* of our React app content.

Help

Importing, Exporting, and the Dependency Tree

There are two other sections in the `App.js` file we haven't touched on:

```
import React from "react";
import { format } from "date-fns";
import ExampleComponent from "./ExampleComponent";

// function App() { etc }

export default App;
```

`react` and `date-fns` are both npm packages, so what is happening here? `App.js` is *pulling in* specific content from these two packages! `react` and `date-fns` are being *imported* from the `node_modules` folder, which was created when we ran `npm install`.

You can see in the `App` function that `format` from the `date-fns` library is being used in the return statement when we call `format(...)`. `react` is also being used, even though you can't see it written in the code! Anywhere you write JSX inside a component, it is actually *transpiled* to JavaScript code that looks like this: `React.createElement(tagName, props, children)` (more on that later).

> Version 17 of React, released October 2020, introduced a new JSX transformation. So instead of `React.createElement()`, the JSX is transpiled into `_jsx()`. You can read more about the change **here** **(https://reactjs.org/blog/2020/09/22/introducing-the-new-jsx-transform.html)**.
>
> With React 17, you can actually *omit* the line `import React from "react"` in your component files, and they'll still work just fine. You can find out which version of React a project is using by looking at `dependencies` section of the `package.json` file.

The import for `ExampleComponent` is slightly different. Rather than importing code from a node package, here we are importing something from within our app's file tree. By importing `./ExampleComponent`, we make `<ExampleComponent />` available for use in the `App` component's return statement.

OK, then what is happening with `export`? By including the `export` line, we are allowing *other* files to *import* things from the `App.js` file. There are different types of exports, like named exports and default exports, but we will dive deeper into this topic in a later lesson.

For now, we will just focus on default exports. The line, `export default App` denotes that our `App` function is the main thing we want to export from our `App.js` file. You can have only one default export per file. If you take a look at `index.js`, you can see at the top of the file that we are *importing* `App` from `App.js` (the `.js` does not need to be included). This import statement is what we use to import something that is the default export of another file:

Help

```
import App from "./App";
```

This structure of importing and exporting allows for files to create a 'tree' of dependencies. `ExampleComponent.js` has an `export` statement as well (take the time to locate it), and is imported into `App.js`. Additionally, `App.js` is imported into `index.js`.

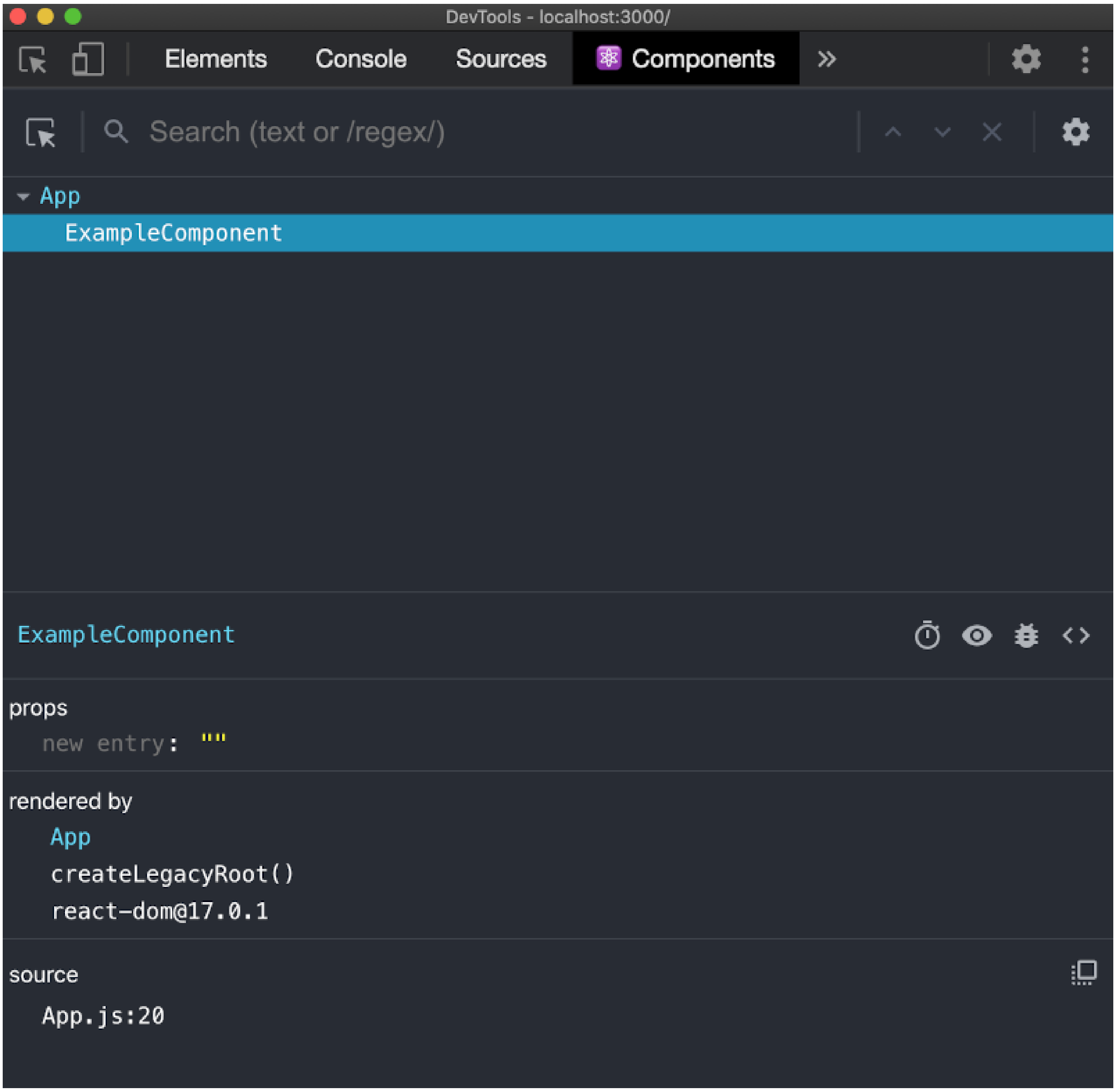
The `index.js` file doesn't have an export. It is the 'top' of this dependency tree.

Debugging Components

React has a great set of developer tools that make it easier to view our components in the browser and debug what's happening under the hood. Install the React Developer Tools **Chrome Extension (https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en)** or **Firefox Extension (https://addons.mozilla.org/en-US/firefox/addon/react-devtools/)**. Here is the **readme (https://www.npmjs.com/package/react-devtools)** for the extension.

After installing, restart your browser, then go to `http://localhost:3000` to view our React application. Open your browser's developer tools, and find the **Components** tab. Here, you'll see the component hierarchy with information about all the components we're using so far in the app!

Help



```
App
  ExampleComponent
```

```
ExampleComponent

props
  new entry: ""

rendered by
  App
  createLegacyRoot()
  react-dom@17.0.1

source
  App.js:20
```

Deliverables

There are three tests to pass in this lesson. They are all related to the content within `src/components/App.js` .

1. Replace the contents of the `h1` element so that, instead of a time, it just says 'Now'
2. If you added a second `<ExampleComponent />` , be sure to remove it
3. Add in a *new* component, `<TestComponent />` , just below `<ExampleComponent />` , in the `App.js` file (remember: you will need to **import** the component before you can use it)

When working on React labs, it's helpful to have two terminal tabs open:

- In the first tab, run `npm start` to run your React app in the browser
- In the second tab, run `learn test` or `npm test` to see the test output

Help

Conclusion

There is a lot we still haven't touched on, and we will go into greater depth on how things work in the following lessons. Almost all of our work, however, will be done within `App.js` and child components of it. This is where all of our creative energy will be spent.

This file structure is used by Facebook, the creators of React, and is what `create-react-app` automatically generates. Using this structure, a lot of set-up is abstracted away. We have all the boilerplate code in place in the `index.html` and `index.js` files so that we can start focusing on writing the core functionality of our app using components, like in `App.js` .

Resources

- **React DevTools - Chrome (https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en)**
- **React DevTools - Firefox (https://addons.mozilla.org/en-US/firefox/addon/react-devtools/)**

Help