React: Modules and CRA

Download Demo Code

Goals

- Understand what Create React App is and how to use it
- Use ES2015 modules to share code across files
- Compare default vs. non-default exports
- Using assets (images and CSS) in components

Create React App

React is a front-end library — you don't need server-side stuff.

You can get ***react.js*** and ***react-dom.js*** from a CDN.

You can transpile JSX in the browser at runtime.

But there's a better way!

Create-React-App is a utility script that:

- Creates a skeleton React project
- Sets it up so that JS files are run through Babel automatically
- Lets us use super-modern JavaScript features/idioms
- Makes testing & deployment much easier

npx

To scaffold a project with Create React App, we'll use ***npx***.

npx will download Create React App and execute it.

You can think of ***npx*** as being an alternative to installing packages globally.

Example

```
$ npx create-react-app my-app-name
```

Skeleton

This provides a nice starter skeleton:

```
├── README.md                   README, can edit or delete
├── package-lock.json           Lock file, don't edit directly
├── package.json                Can edit, as usual
│
├── public                      Rarely need to edit these
│   ├── favicon.ico
│   ├── index.html              Main HTML page of site
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
│
└── src                         Where React stuff goes
    ├── App.css                     CSS for example component
    ├── App.js                      Example component
    ├── App.test.js                 Tests for App component
    ├── index.css                   Site-wide CSS
    ├── index.js                    Start of JS
    ├── logo.svg                    React logo
    ├── serviceWorker.js            (Ignore this for now)
    └── setupTests.js               Starter test configuration
```

Starting Your App

```
$ npm start
```

Webpack

CRA is built on top of Webpack, a JS utility that:

- Enables module importing/exporting
  - Packages up all CSS/images/JS into a single file for browser
  - Dramatically reduces # of HTTP requests for performance
- Hot reloading: when you change a source file, automatically reloads
  - Is very clever and tries to only reload relevant files
- Enables easy testing & deployment

Note: The Webpack Rabbit Hole

Webpack is a powerful tool, and configuring it can be quite complicated. Create React App abstracts away that configuration from you, which is great when you're first learning. It's not worth your time right now to learn too much about webpack other than the high-level bullet points we've outlined. If you're curious, you can always to go to the Webpack website, but be warned: Webpack is a rabbit hole it's easy to go down and isn't terribly important at this stage in your learning.

Modules

- ES2015 introduces the idea of "modules", but browser support is highly limited
- This is a newer, standardized version of Node's ***require()***
- You use this to export/import classes/data/functions between JS files
- You will see these everywhere in modern JS codebases!

How does it work?

Using two keywords, **import** and **export**

- We export out variables (functions, objects, strings etc) so other files can use them
- We import "exported" values into a file so that we can use them in the current file we are in

An example

hello.js

```
function sayHello(){
  return "hello"!
}

export default sayHello;
```

main.js

```
import sayHello from "./hello.js"

sayHello();
```

Importing "Default" Export

demo/import-export/mystuff.js

```
function myFunc() {
  console.log("Hi");
}

export default myFunc;
```

demo/import-export/index.js

```
// Must start with dot --- "mystuff" would be a npm module!

import myFunc from './mystuff';
```

Importing Non-Default Named Things

demo/import-export/mythings.js

```
function otherFunc() {
  console.log("Hey");
}

const luckyNumber = 13;

export { otherFunc, luckyNumber };
```

demo/import-export/index.js

```
import { otherFunc, luckyNumber} from "./mythings";
```

Importing Both

demo/import-export/both.js

```
function mainFunc() {
  console.log("OK");
}

const msg = "Awesome!";

export default mainFunc;
export { msg };
```

demo/import-export/index.js

```
import mainFunc, { msg } from "./both";
```

To Default or Not?

- Conventionally, default exports are used when there's a "most likely" thing to exporting.
- For example, in a React component file, it's common to have the component be the default export.
- You never **need** a default export, but it can be helpful to indicate most important thing in a file.

Resources

Export

Import

Transpiling Using Babel

Babel?

- It is a JavaScript compiler
- You run modern JS that browsers can't understand and you get out JS that all browsers can understand!
- babeljs.io

CRA and Components

Good style:

- Each React component goes in separate file
  - ***src/Car.js*** for ***Car*** component
  - ***src/House.js*** for ***House*** component
- Define your function component, then export it as the default
- Skeleton assumes top object is ***App*** in ***App.js***
  - Best to keep this

Assets and CRA

To include images and CSS, you can import them in JS files!

demo/my-app-name/src/App.js

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

CSS

- Make a CSS file for each React component
  - ***House.css*** for ***House*** component
- Import it at the top of ***House.js***
  - Create-React-App will automatically load that CSS
- Conventional to add `className="House"` onto ***House*** div
  - And use that as prefix for sub-items to style:

```
<div className="House">
  <p className="House-title">{ props.title }</p>
  <p className="House-address">{ props.addr }</p>
</div>
```

Images

- Store images in **src/** folder with the components
- Load them where needed and use imported name where path should go:

```
import puppy from "./puppy.jpg";

function Animal() {
  return (
    <div>
      <img src={puppy} alt="Cute puppy!" />
    </div>
  );
}
```

Building for Deployment

`npm run build` makes ***build/*** folder of static files

You can serve from a web server.