

26 Frontend Tooling and React

Objectives

- Become familiar with Webpack Concepts
- Create a simple React app
- Render data from a component to HTML
- Attach click handlers to our component
- Modify and rerender components after clicks

Install Dependencies

```
npm i react react-dom  
    webpack webpack-dev-server  
    babel-core babel-loader  
    babel-preset-env  
    babel-preset-react  
    style-loader css-loader
```

Four Main Webpack Concepts

- **Entry** - the file where the dependency graph begins.
- **Output** - the folder where everything gets compiled into (like `dist/bundle.js`)
- **Loaders** - things that allow webpack to process more than just JavaScript. There are loaders for CSS, images, raw files, and much more!
- **Plugins** - Fancy modules that accomplish a wide range of tasks, like optimizing bundles, minifying code, and much more!

The .babelrc file

.bachelrc

```
{  
  "presets": ["env", "react"]  
}
```

webpack.config.js

```
const path = require('path');

const config = {
  mode: "development",
  entry: "./app.js",
  output: {
    path: path.resolve(__dirname, './dist'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {test: /\.js$/, loader: "babel-loader"},
      {test: /\.css/, loader: [
        "style-loader", "css-loader"
      ]}
    ]
  }
};

module.exports = config;
```

Loader Order

- Notice: order matters for these loaders!
- Using `css-loader` before `style-loader` won't work.

Correct:

```
loader: ['style-loader', 'css-loader']
```

Incorrect:

```
loader: ['css-loader', 'style-loader']
```

Minimal React App

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './style.css';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      heading: 'My React app',
    }
  }
  render() {
    return <div className="App">
      <h1>{this.heading}</h1>
    </div>
  }
};

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

Minimal index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample React App</title>
  </head>
  <body>
    <div id="root"></div>
    <script src="./dist/bundle.js"></script>
  </body>
</html>
```


Binding Clicks

click-counter.js

```
class ClickCounter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {count: 0};
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => {
      return {count: state.count + 1};
    });
  }

  render() {
    return <p onClick={this.handleClick}>
      Clicked {this.state.count} times.
    </p>
  }
};

module.exports = ClickCounter
```

Nesting Components

```
import ClickCounter from './click-counter';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      heading: 'My React app',
    }
  }
  render() {
    return <div className="App">
      <h1>{this.heading}</h1>
      <ClickCounter></ClickCounter>
      <ClickCounter></ClickCounter>
      <ClickCounter></ClickCounter>
    </div>
  }
};
```

Configuring package.json

- Configure `build` and `watch` commands to run your server
- Use `npm run build` and `npm run watch` to run the commands.

package.json

```
{
  "name": "webpack-simple",
  "main": "index.js",
  "scripts": {
    "build": "webpack",
    "watch": "webpack-dev-server --inline --hot"
  },
  "dependencies": {
    ...
  }
}
```

React Tips

- Accept a `props` parameter in the `constructor`
- Call `super(props)` on the first in the `constructor`
- Bind click handler methods at the end of the `constructor`
 - `this.handleClick = this.handleClick.bind(this);`
- Modify state through `.setState(state => return {foo: state.foo + 1})`
- The `render()` method can only return one element. When in doubt, wrap everything in a `<div>`.

SASS

- SASS offers extensions to CSS that make writing CSS "syntactically awesome!"
- SASS files have the extension `.scss`
- <https://sass-lang.com/guide>

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  
  li { display: inline-block; }  
  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
  }  
}
```

Steps to Create React App

1. `npm init -y`
2. install dependencies
3. configure `.babelrc` file
4. configure `webpack.config.js` file
5. create minimal `index.html` file
6. create minimal `main.js` react file
7. reference `./dist/bundle.js` from `index.html` in `<script>` tag

