27 Forms and Props

Objectives

- Gather and render user input
- React to Form submit events
- Use `if` statements, `for` loops and `.map` constructs in JSX
- Pass props into components
- Lift state from components to the root of the app
- Access data from the web via `fetch`

Minimal webpack.config.js Revisted

Webpack 4 introduced giving `output` a default value. If you omit giving your config an output property it will build your bundle and place it in `./dist/main.js` by default.

```
const config = {
  mode: "development",
  entry: "./src/app.js",
  module: {
    rules: [
      {test: /\.js$/, loader: 'babel-loader'},
      {test: /\.css$/, loader: ['style-loader', 'css-loader']},
    ]
  }
}

module.exports = config;
```

Gather and render user input

- Use a `onChange` property on an `<input>`
- You must `setState` and reset the value of the input before React re-renders it on thep page.

```
onChange(ev) {
  let userInput = ev.target.value;
  console.log('input', userInput);
  this.setState({userInput});
}

render() {
  <input type="text"
    onChange={this.onChange}
    value={this.state.userInput}
    placeholder="enter text here">
  </input>
}
```

React to Form submit events

- Set an `onSubmit` property equal to a function in the Component
- Accept an event parameter in the `onSubmit` function.
- Remember to call `ev.preventDefault()` to prevent the page from navigating somewhere else.

```
onSubmit(ev) {
  ev.preventDefault();
  this.props.submit(this.state.query);
}

render() {
  return <div>
    <form onSubmit={this.onSubmit}>
      <input type="text" value={this.state.query} onChange={this.onChange}>
      </input>
    </form>
  </div>
}
```

Pass Props into Components (1/2)

```
constructor(props) {
  super(props);
  this.handleSubmit = this.handleSubmit.bind(this);
}

handleSubmit(query) {
  console.log('q:', query);
}

render() {
  return <div>
    <h1>{this.state.title}</h1>
    <SearchForm submit={this.handleSubmit} />
    <SearchResults results={this.state.result} />
  </div>
}
```

Pass Props into Components (2/2)

```
class SearchForm extends React.Component {
  constructor(props) {
    super(props);
    this.onSubmit = this.onSubmit.bind(this);
  }

  onSubmit(ev) {
    // prevent the form from submitted, access query and call
    // the submit function in app through props
    ev.preventDefault();
    this.props.submit(this.state.query);
  }

  render() {
    return <form onSubmit={this.onSubmit}>
      <input type="text"
        value={this.state.query}
        onChange={this.onChange}>
      </input>
    </form>
  }
```

Using if-statements in JSX

```
getList() {
  if (this.props.results === 0) {
    return <p>No phrases.</p>
  } else if (this.props.results === 1) {
    return <p>One phrase.</p>
  } else {
    return <p>Many phrases!</p>
  }
}

render() {
  return <div>
    <p>Search results:</p>
    {this.getList()}
  </div>
}
```

Rendering Lists with forEach

```
phrases() {
  // define some array
  let phrases = ["cowabunga", "any array"];

  // map the elemnts in the array to JSX elements
  phrases = phrases.map(phrase => {
    return <li>{phrase}</li>
  });

  // render the list of JSX elements
  return <ol>
    {phrases}
  </ol>
}

render() {
  return <div>
    {this.phrases()}
  </div>
}
```

Lift state from components to the root of the app

Design your app so their is a separation between where the data is stored and how the components render the data. Keep the data at the app level, pass the data into the components in via props.

React has a one-way data flow through props. Data flows from the app down into components. Data flows down from a parent component to a nested component.

```
App
  SearchForm
  SearchResults
```

Lift state from components to the root of the app

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {title: "App", results: []};
    this.onSubmit = this.onSubmit.bind(this);
  };

  onSubmit(params) { console.log(params) }

  render() {
    return <div>
      <h1>{this.state.title}</h1>
      <MyForm submit={this.onSubmit} />
      <MyResult results={this.state.results} />
    </div>
  }
}
```

Access data from the web via `fetch`

Use the native function `fetch()` to make AJAX requests to APIs on the internet, receive data, parse it as JSON and set the state of your application.

```
let url = `http://someurl.com/foo=${myFooVar}`
fetch(url)
.then(response => {
  return response.json()
})
.then(json => {
  let content = json.path.to.your.results.from.api;
  this.setState({results: content})
})
.catch(()=> {
  this.setState({results: []})
});
```

Bonus: Hot Reloading

`Hot Reloading` is a wonderful development tool that automatically reloads a page you're working on whenever you make a change in your file.

It allows you to change a color in a CSS file, save it, then you'll see your webpage automatically reload colors change.

Hot realoding isn't just a CSS thing. It works for any file in your project. You can modify JS, or HTML too. Any time you save you'll see the entire app refresh.

It's great, but at the end of the day it's not necessary. It's possible to change your tail for hours or day trying to configure a project to get hot reloading set up correctly.

If you ever have trouble configuring hot realoding, good luck, and may you get it configured, but know when to quit and work old-school without it!

Bonus: Hot Reloading Configuration

Note: remove the `index.html` you manually configured that had the root div and referenced the `bundle.js` file in a script tag. The `html-webpack-plugin` will automatically generate an HTML file that's served for you.

```
npm install html-webpack-plugin
```

Modify the end of `app.js` so it creates it's own `<div>` and attach it to the page:

./src/app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {...}

const root = document.createElement('div');
document.body.appendChild(root);
ReactDOM.render(<App/>, root);
```

Bonus: Hot Reloading Configuration

webpack.config.js

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

const config = {
  mode: "development",
  entry: "./src/app.js",
  module: {
    rules: [
      {test: /\.js$/, loader: 'babel-loader'},
      {test: /\.css$/, loader: ['style-loader', 'css-loader']},
    ]
  },
  plugins: [
    new HtmlWebpackPlugin()
  ],
}

module.exports = config;
```

Bonus: Hot Reloading Scripts

package.json

```
"scripts": {
  "build": "webpack",
  "watch": "webpack --watch",
  "serve": "webpack-dev-server",
  "hot": "webpack-dev-server --inline --hot"
},
```