React pt. 2: NodeJS, Hooks, and API calls

Kianoosh Abbasi

CSC309 Fall 2022

Some content is from Dr. Sadia Sharmin's slides of CSC309 Winter 2021: www.rainsharmin.com

So far

- HTML, CSS, and Django backend
- JavaScript front-end
  - DOM, jQuery, Ajax
  - Advanced topics: closures, arrow functions, promises
- Single-page applications with React
  - JSX, props, events, state

This week

- React projects
- NodeJS, npm
- Enhanced function components
  - Hooks
- API calls

React so far

- Enabled by importing some scripts to our HTML file
- JSX code must be translated to JS every time
- Very slow

React projects

- A dedicated project for React
  No longer part of backend/html project
- Front-end server that returns appropriate files per request
- A pre-compiled and bundled build for production

node

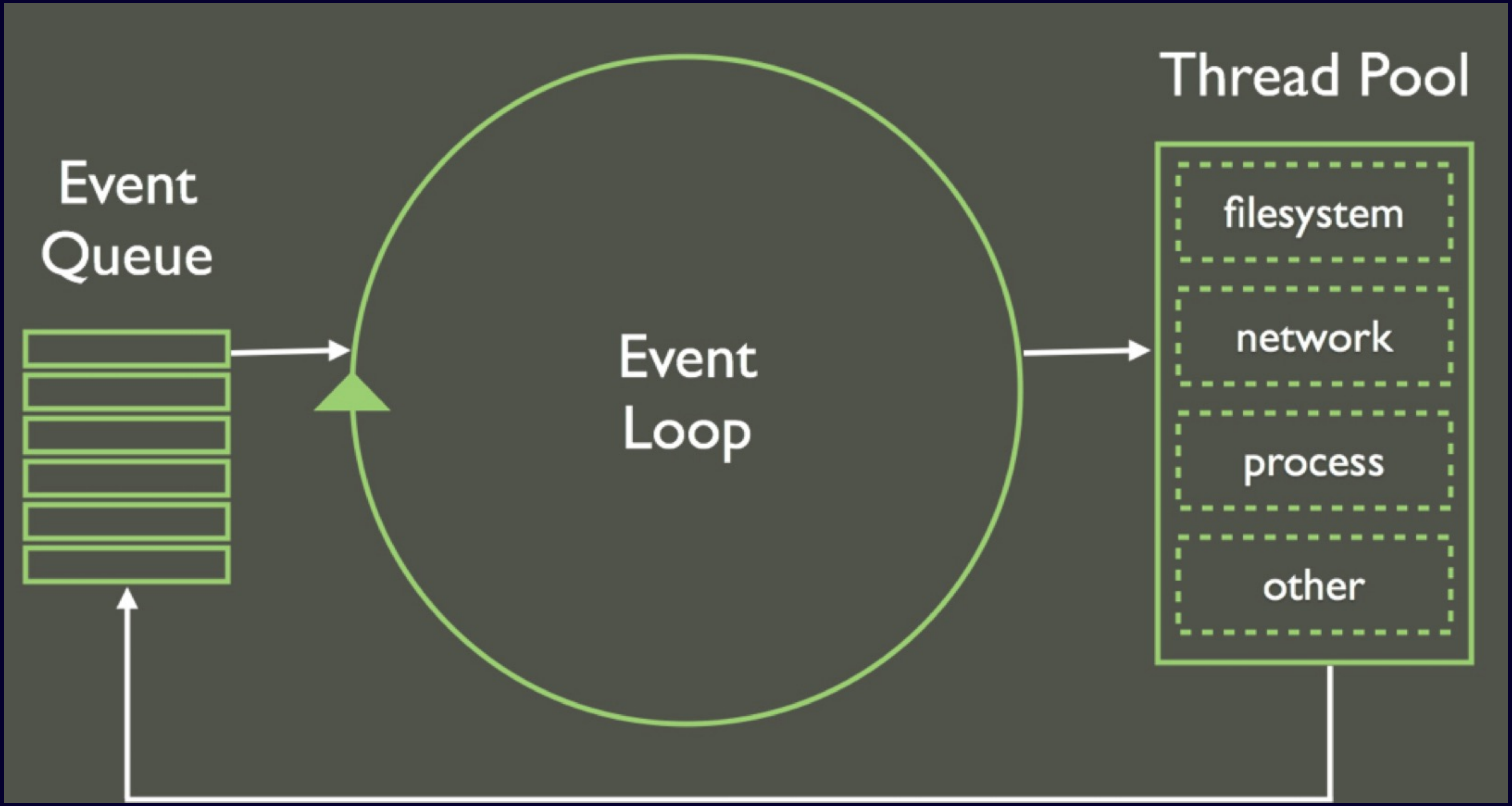
JS®

Front-end server

- JS does not have to be run on the browser!
- NodeJS: a runtime environment to for running JS server-side
- Includes a package manager, console, build tools, etc.

Processing model

Visit https://www.youtube.com/watch?v=zphcsoSJMvM



Node console

- Opens with the node command
- You can execute inline JS code
- No window or DOM object

  We are outside of the browser
- Files can be run as well

  ```
  node <filename>
  ```

Installing modules

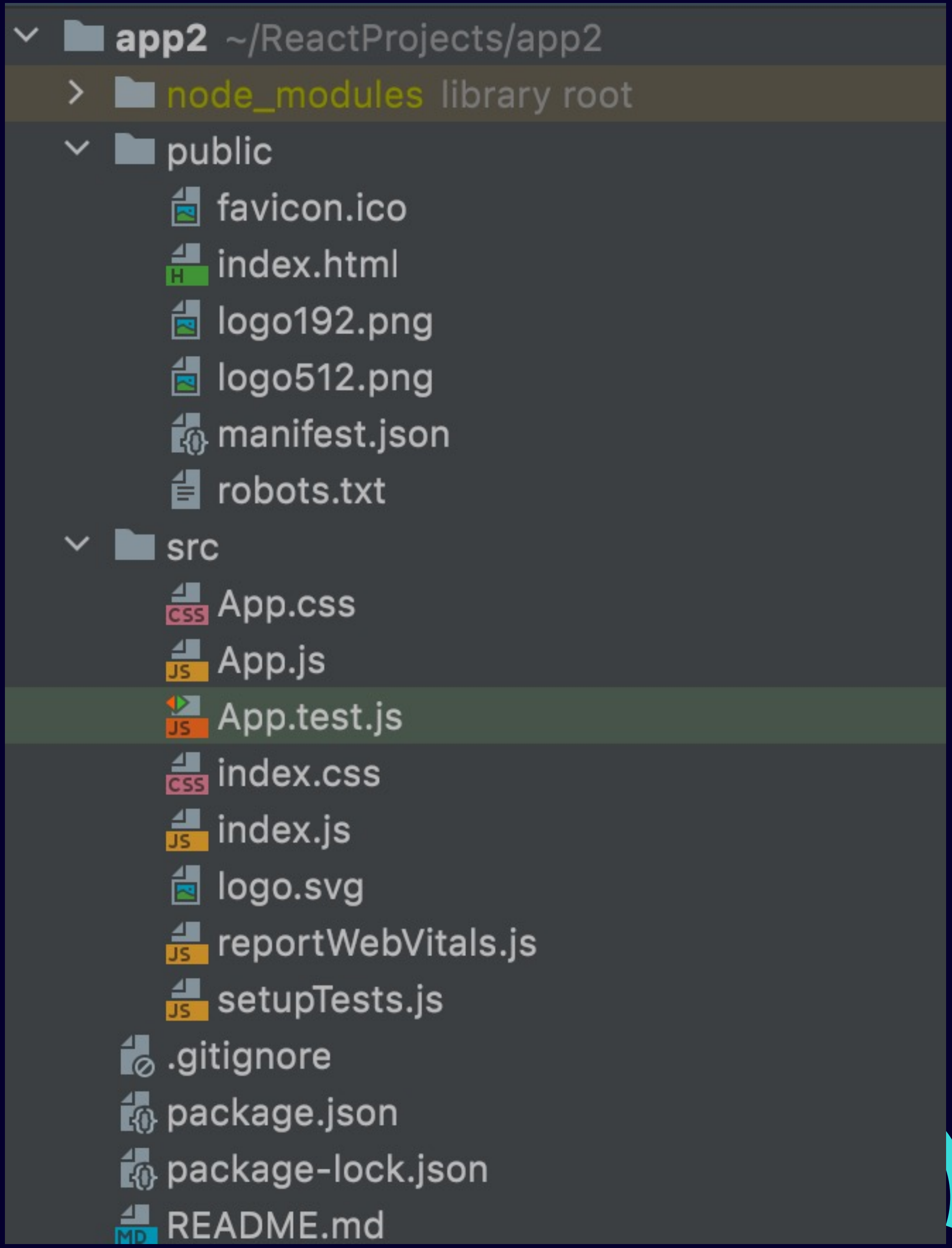
- Node Package Manager (NPM)
  Very similar to pip

- Install packages via `npm install <package_name>`
  Packages are stored in the node_modules directory

- Automatically generates and maintains a file named package.json

Creating a React project

- Via command:
```
npx create-react-app <name>
```
- Run the server:
```
npm start
```
- Make a production build:
```
npm build
```



```
app2 ~/ReactProjects/app2
  node_modules library root
  public
    favicon.ico
    index.html
    logo192.png
    logo512.png
    manifest.json
    robots.txt
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package.json
  package-lock.json
  README.md
```

React project

- The same code but more organized
- index.js contains the invocation of `ReactDOM.render`
- The root component is the default export from App.js

Exports

- Variables, classes, or functions can be exported from a JS module

```
const var1 = 3, var2 = (x) => x + 1;
export { var1, var2 };
```

- Can be reduced to one statement:

```
export const var1 = 3, var2 = (x) => x + 1;
```

- Other modules can import them

```
import { var1 } from './App';
```

Default export

- Each module can have one default export
  Usually the main component
  ```
  export default App;
  ```
- Importing the default export:
  ```
  import App from "./App";
  ```
- This time, the names do not have to match
  Can be imported under any arbitrary name

File structure

- Already creates the HTML in the public folder
- Cherry on the cake: You can import all your css and assets (image, font, etc.) to your JS modules
  - You should NOT import them to your HTML
- Handled and served properly by the server

File structure

- Dedicate a separate file for each component

- Create a components subdirectory
  - Might want to create a subdirectory for each components as well
    - Contains the JS and CSS for that component

- Always have some re-usable base components
  - Inputs, forms, headings, etc.

File structure

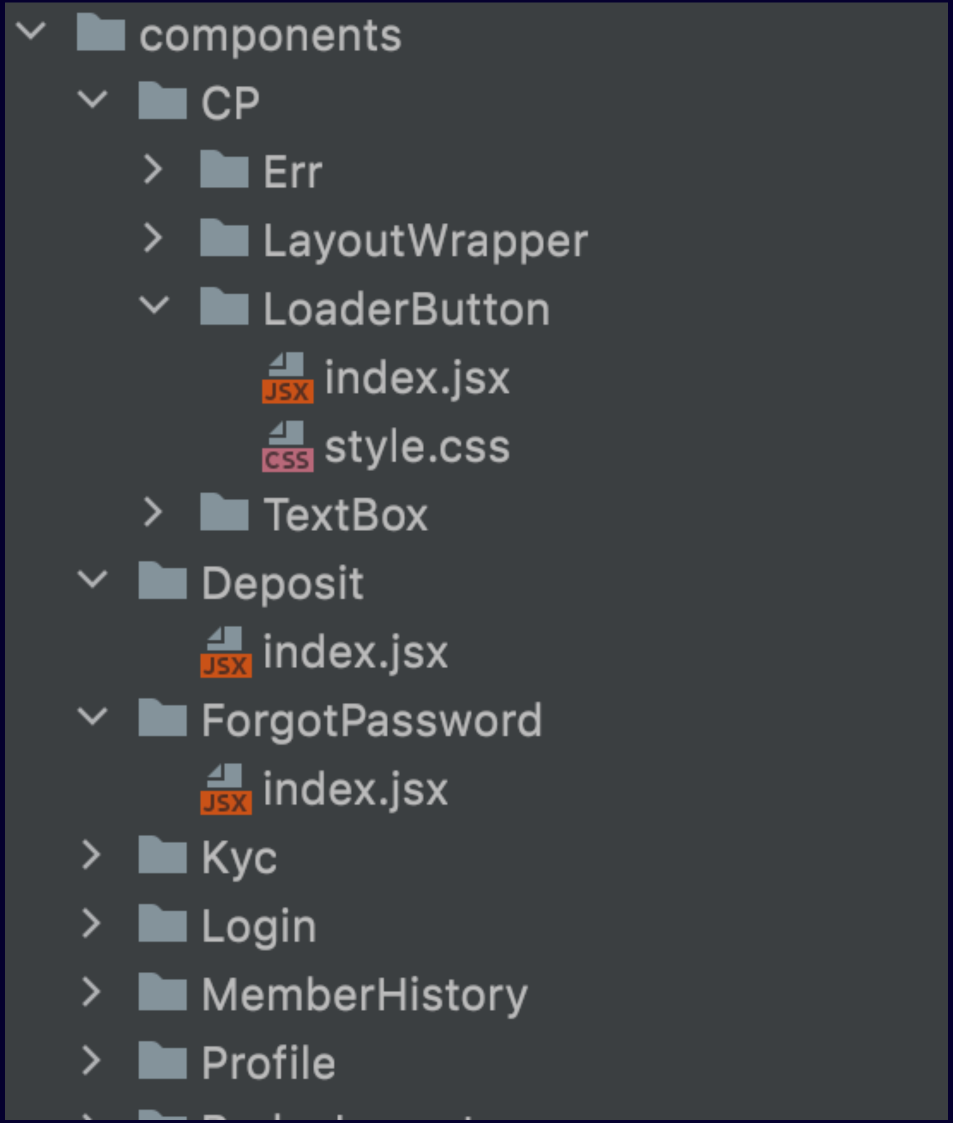
- Import css files:

```
import "./style.css";
```

- Images and other static files can gather under the `assets` directory

- Don't make components too big:

  Have nested, child components



```
⌄ components
  ⌄ CP
    › Err
    › LayoutWrapper
    ⌄ LoaderButton
        index.jsx
        style.css
    › TextBox
  ⌄ Deposit
      index.jsx
  ⌄ ForgotPassword
      index.jsx
  › Kyc
  › Login
  › MemberHistory
  › Profile
```

Hooks

Hooks

- Great syntax sugars introduced in React 16.8
- No need to write verbose classes, constructors, and `setState` anymore
- You can move back to function components

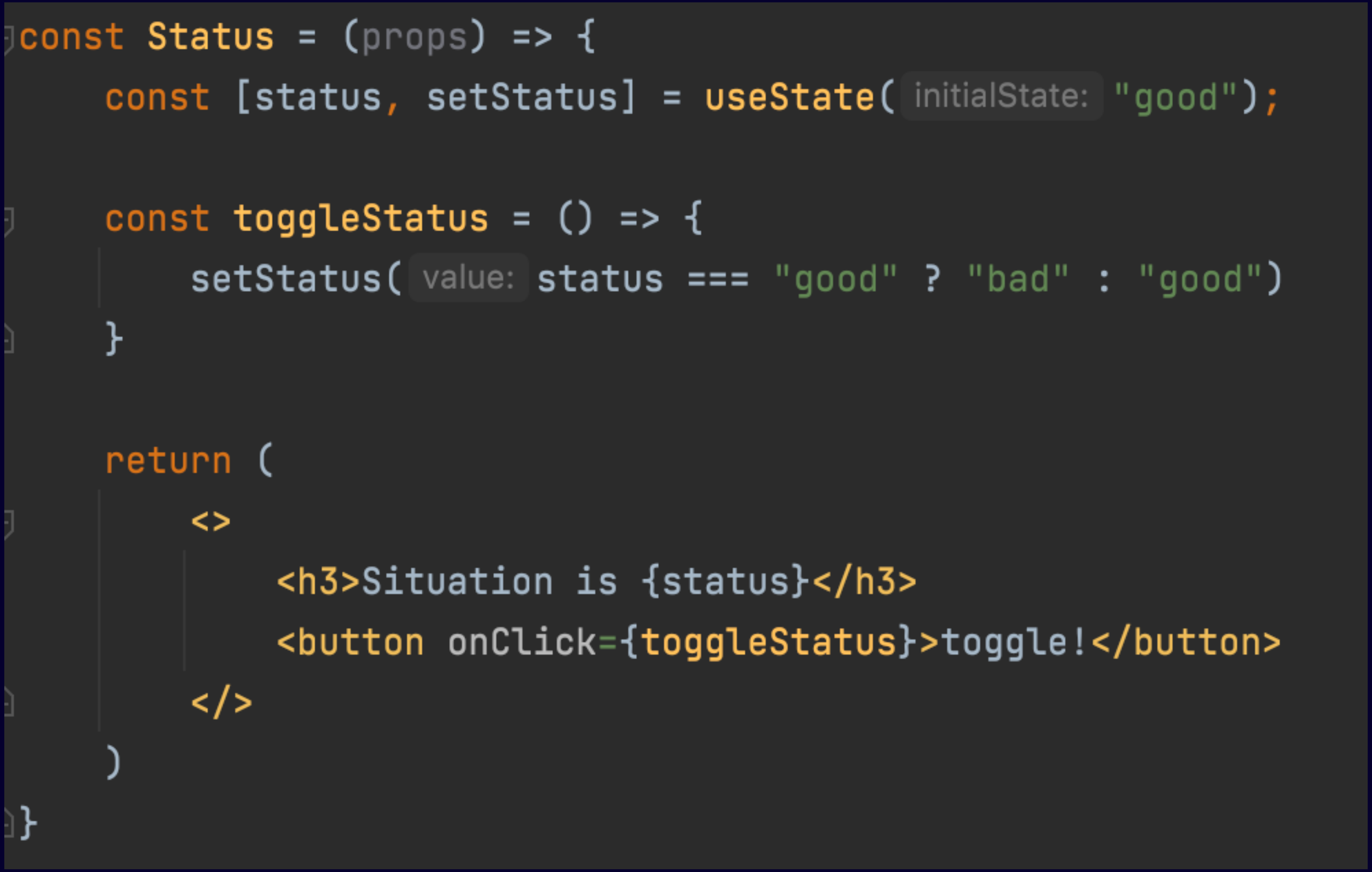
useState

- State does not have to be one object anymore
- Define separate state variables via the useState hook

```
import React, { useState } from 'react';
```

- Returns the variable and update function
- Component gets re-rendered when the value changes

Example



```
const Status = (props) => {
    const [status, setStatus] = useState( initialState: "good");

    const toggleStatus = () => {
        setStatus( value: status === "good" ? "bad" : "good")
    }

    return (
        <>
            <h3>Situation is {status}</h3>
            <button onClick={toggleStatus}>toggle!</button>
        </>
    )
}
```

Benefits

Visit https://blog.bitsrc.io/6-reasons-to-use-react-hooks-instead-of-classes-7e3ee745fe04

- Function components instead of verbose class components
- Enables multiple state variables
- No more this, no more method binding
- Easy to share state with child elements

  Each state variables comes with its own setter

Lifecycle

- So far, we only know to run code when render is called
  - In both class and function components
- You might not want to run code this way
  - Example: Sending a request upon load, accessing state values, etc.
- Adding lifecycle
  - In class components: `componentWillMount()`, `componentDidMount()`, `componentWillUnmount()`, etc.

useEffect

- A powerful hook to replace lifecycle functions
- Called when component mounts
- Also, can be called when something changes

- Import the hook

```
import React, { useState, useEffect } from 'react';
```

- Usage

```
useEffect(() => {
    console.log("This is called when component mounts")
}, [])
```

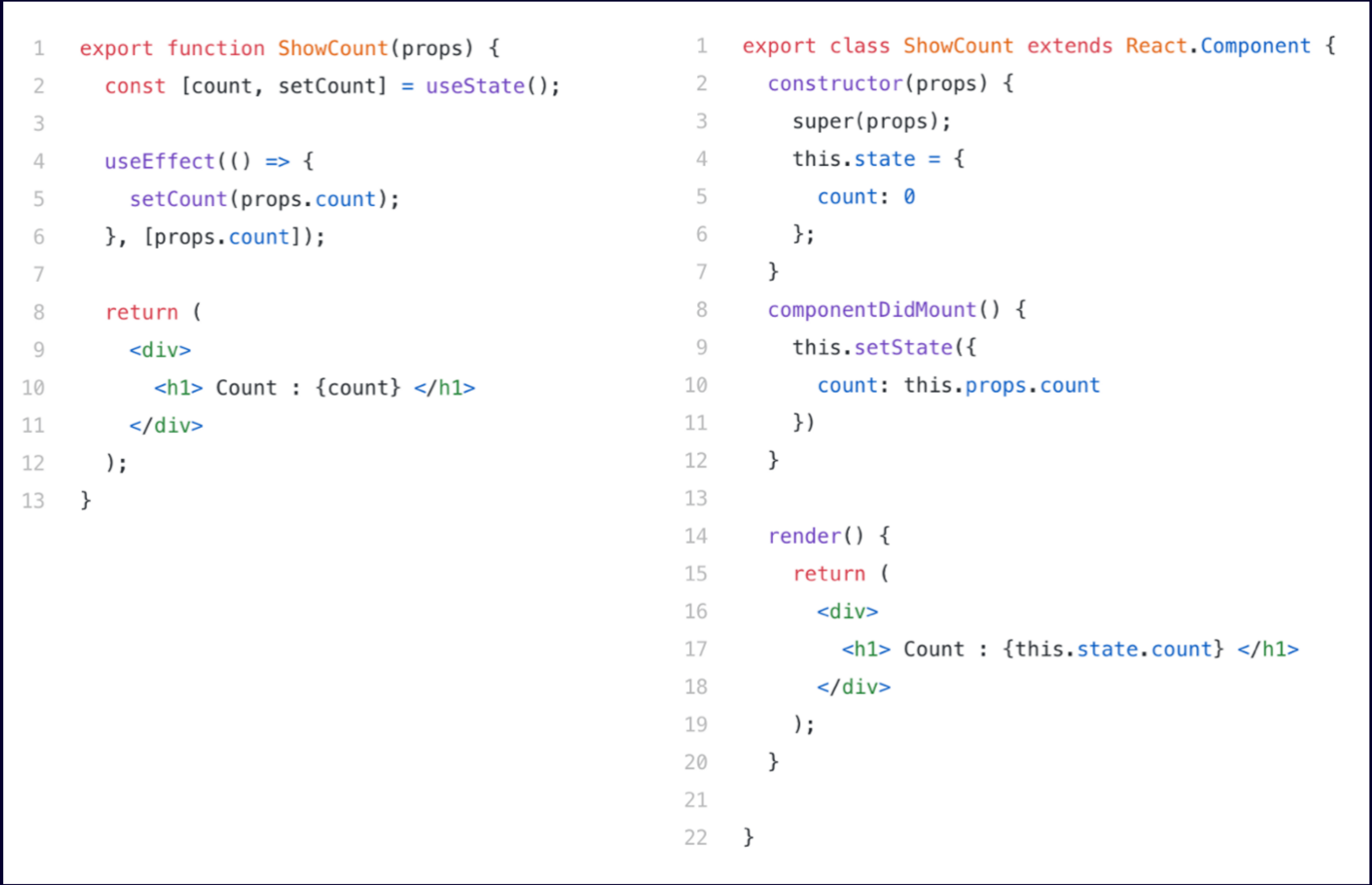
- Subscription

  When any element of the array changes, the effect is invoked

```
useEffect(() => {
    console.log("props size or status has changed")
}, [status, props.length])
```

- Recommended to have a separate useEffect for different concerns

Benefits of hooks



```
export function ShowCount(props) {
  const [count, setCount] = useState();

  useEffect(() => {
    setCount(props.count);
  }, [props.count]);

  return (
    <div>
      <h1> Count : {count} </h1>
    </div>
  );
}
```

```
export class ShowCount extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  componentDidMount() {
    this.setState({
      count: this.props.count
    })
  }

  render() {
    return (
      <div>
        <h1> Count : {this.state.count} </h1>
      </div>
    );
  }

}
```

Benefits of hooks

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

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }
  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Notes

- Do not leave out the second argument
  - The effect would run at every re-render: inefficient
- The array should include all variables that are used in the effect
  - Otherwise, it might use stale values at re-renders

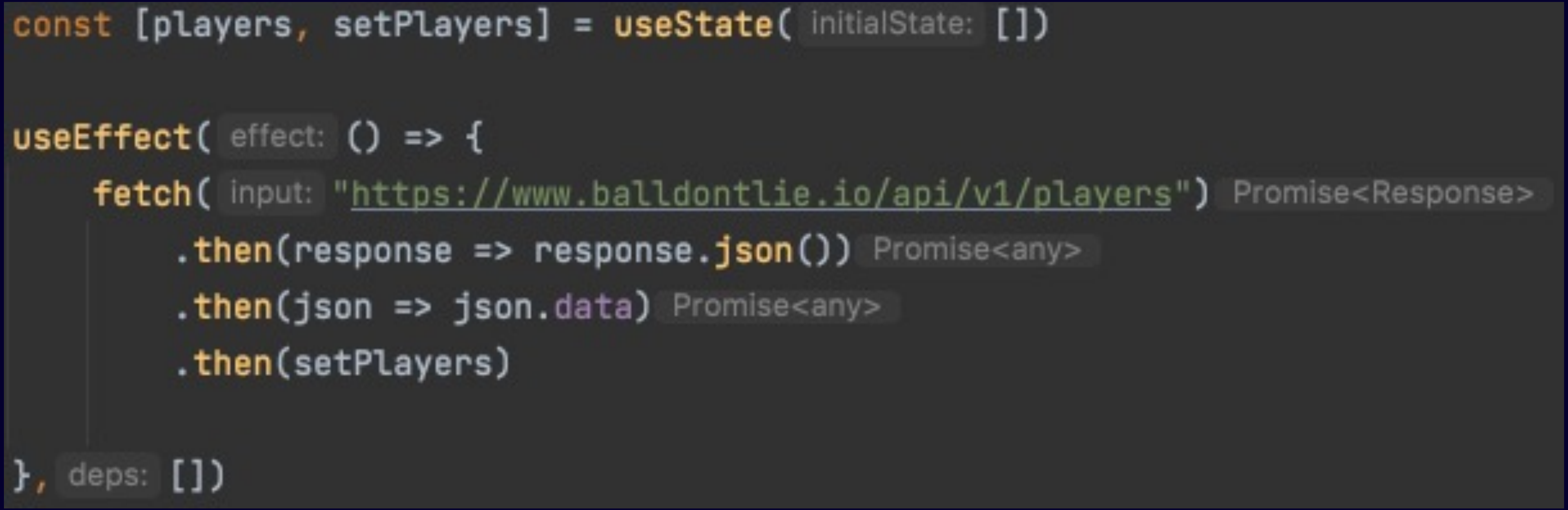
Exercise: a calculator with React

API calls

- Recap from lecture 4: Fetch API
- Returns a promise that could be handled with the then callbacks
- Error handling with the catch callback
- Very straightforward!

API calls and hooks

- Example: fetching data on page load and adding it to state



```
const [players, setPlayers] = useState( initialState: [])

useEffect( effect: () => {
    fetch( input: "https://www.balldontlie.io/api/v1/players") Promise<Response>
        .then(response => response.json()) Promise<any>
        .then(json => json.data) Promise<any>
        .then(setPlayers)

}, deps: [])
```

API calls and hooks

- Can be turned to a search easily
- Can also support pagination
- There are some nuances to support both!

This week

- React projects
- NodeJS, npm
- Enhanced function components
  - Hooks
- API calls

Next week

- Global state and Context
- Multi-page React apps
  - Routers and Links
- Review of concepts: PF prep