Internet Applications Design and Implementation 2017/2018 (Lab 3: React)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Leitão (jc.leitao@fct.unl.pt)
João Costa Seco (joao.seco@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Lab3 Goal:

- Learn the fundamentals of REACT.
- Learn how to setup your work environment for React.
- Learn how to run a React application localy (with node).
- Build your first React application (Yes message board again).

Introduction to React:

- A JavaScript library for building user interfaces.
  - It does not make assumptions about the remaining aspects of your application stack.
- Originally created by Facebook, with the goal of making their application more reactive and interactive for their user base.

Introduction to React:

React

- Component-Based.
  - You build a set of components (that might have and manage their own state) and you combine them to make your application.
  - All component logic is written in javascript, meaning that state can be left out of the DOM (which is good for performance).

- We will be using a XML-like sintax called **JSX**.

Introduction to React: Component

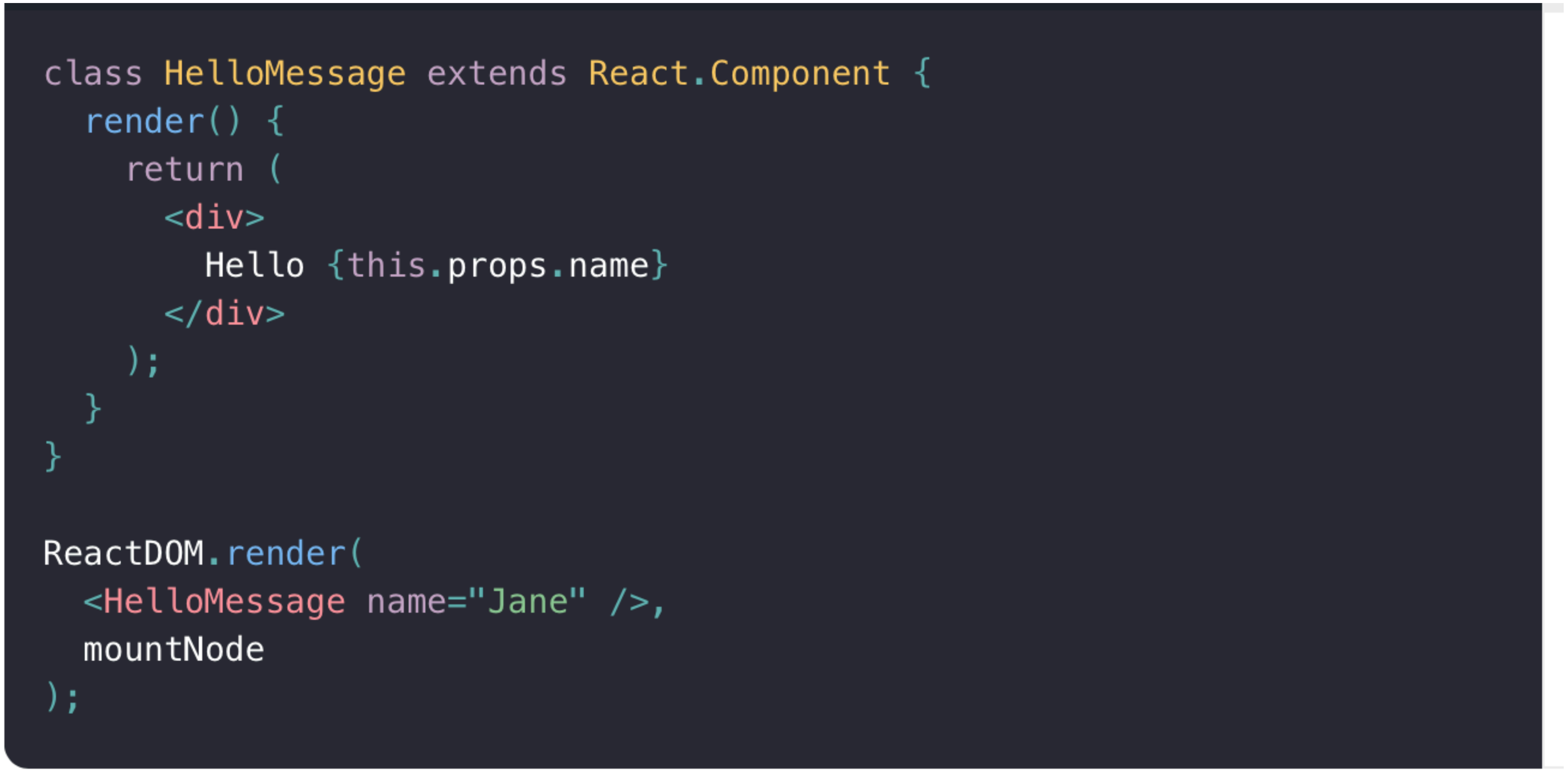
React

- A Component is responsible for displaying some input data (accessible through the this.props varibale).
- At its most simple materialization, a Component will have simply to render some data into the application. This implies that a component will always have a render() method that defines what to be displayed.
  - Note: the render method can only return a single top level entitiy, if you want to render more than one you have to encapsulate it.

Introduction to React: Component

React

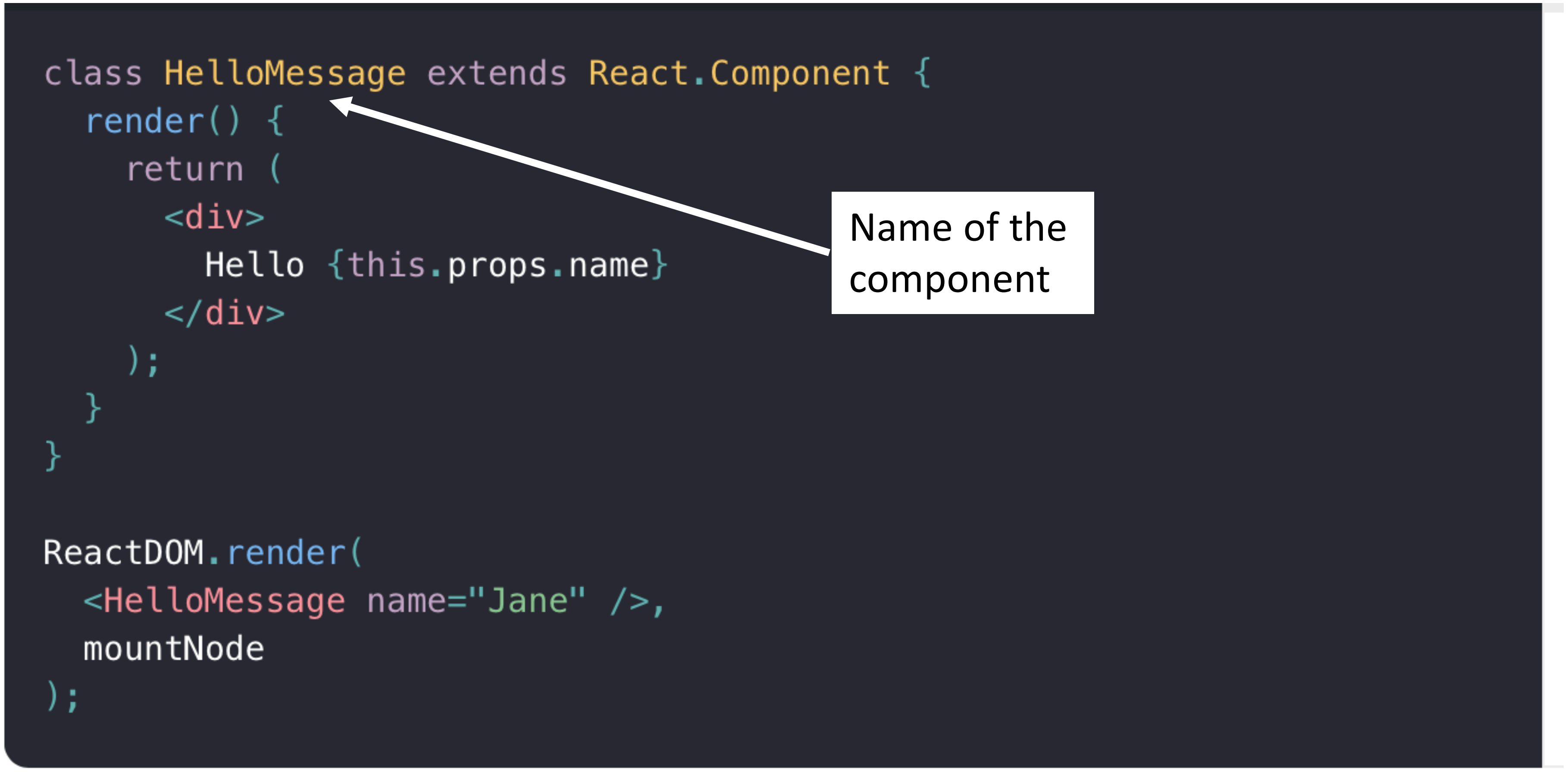
- Example of a simplistic component (and how to instantiate it)



```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

Introduction to React: Component

- Example of a simplistic component (and how to instantiate it)

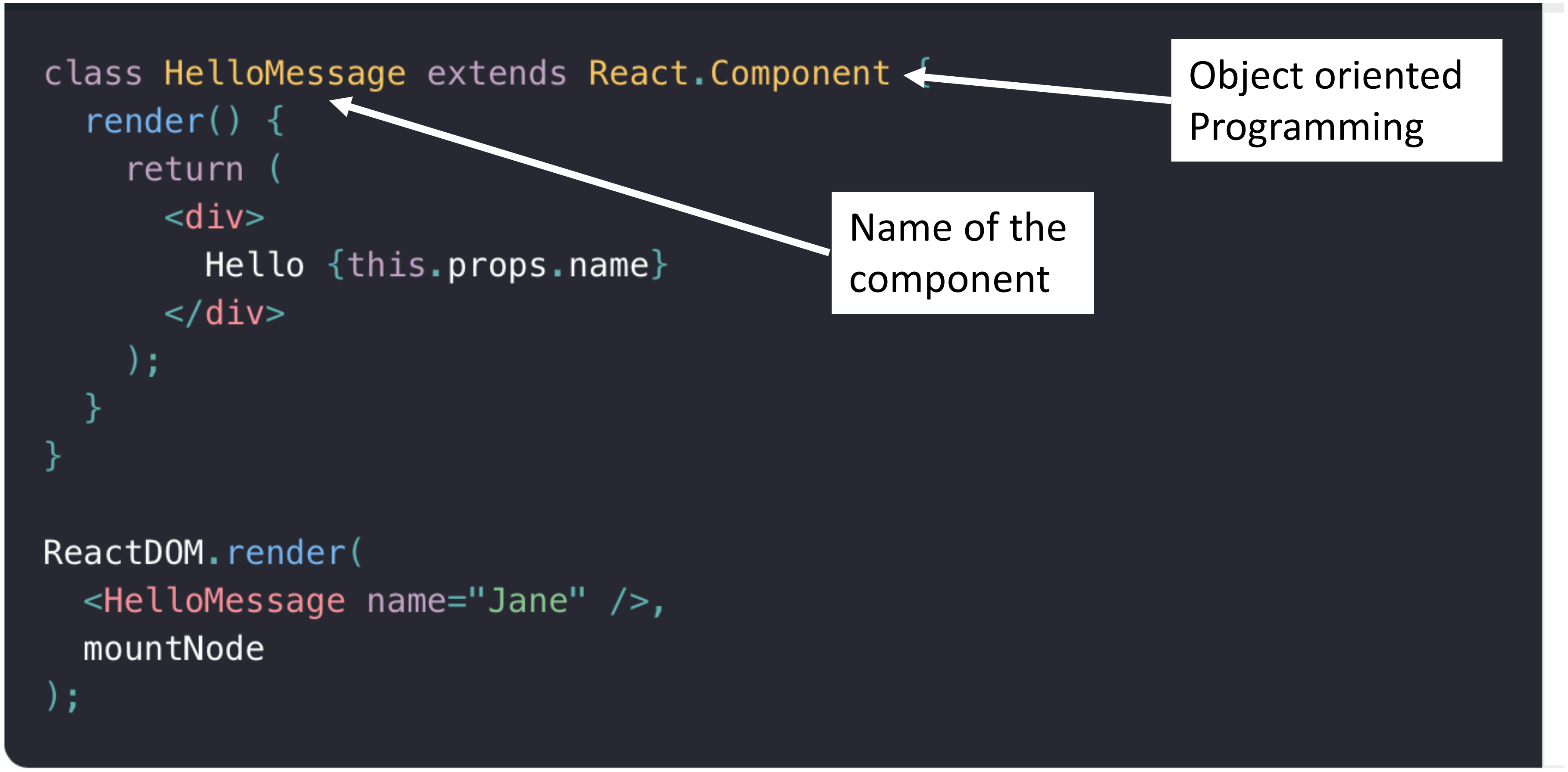


```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

Introduction to React: Component

React

- Example of a simplistic component (and how to instantiate it)

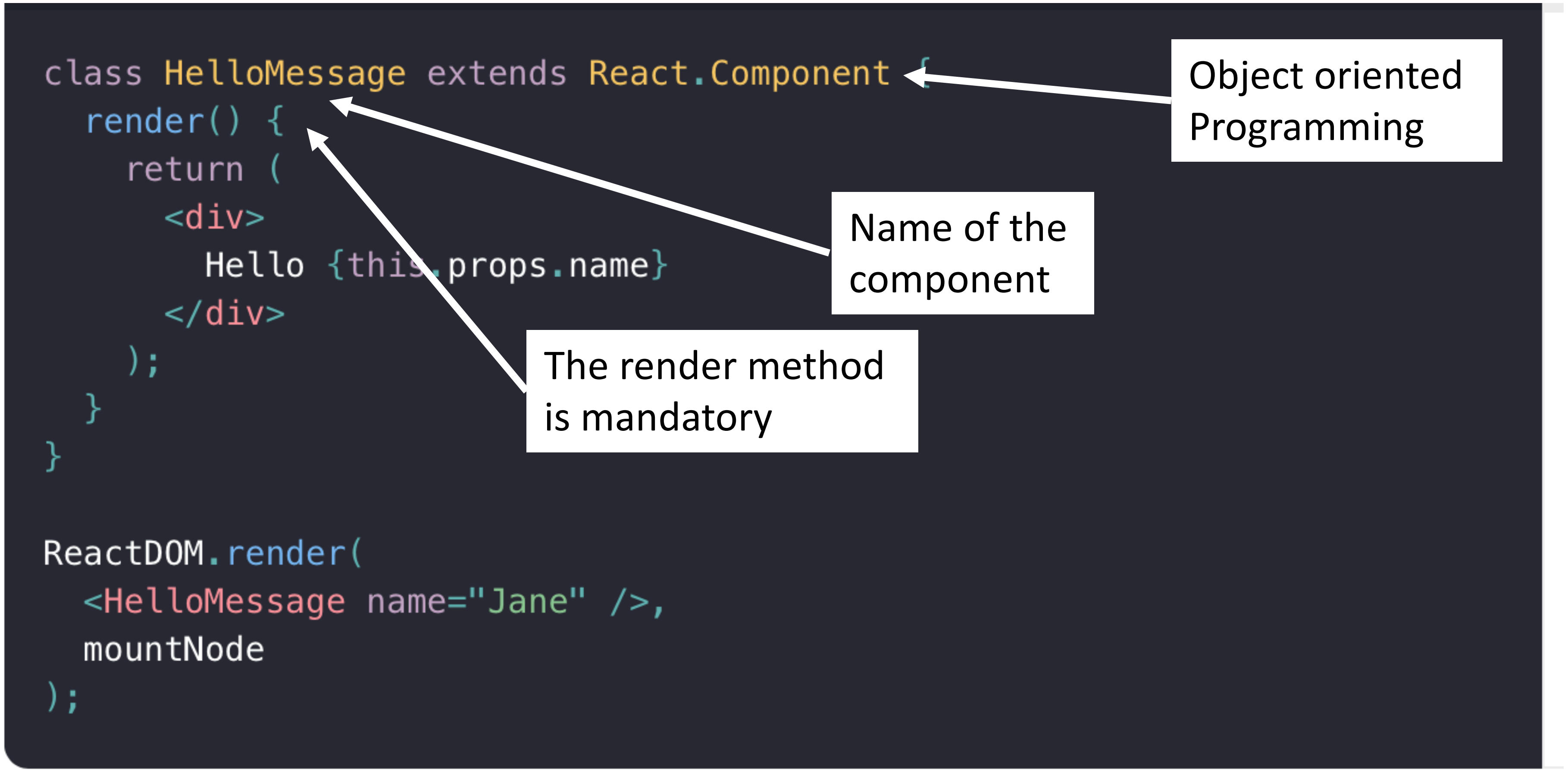


```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

Introduction to React: Component

React

- Example of a simplistic component (and how to instantiate it)

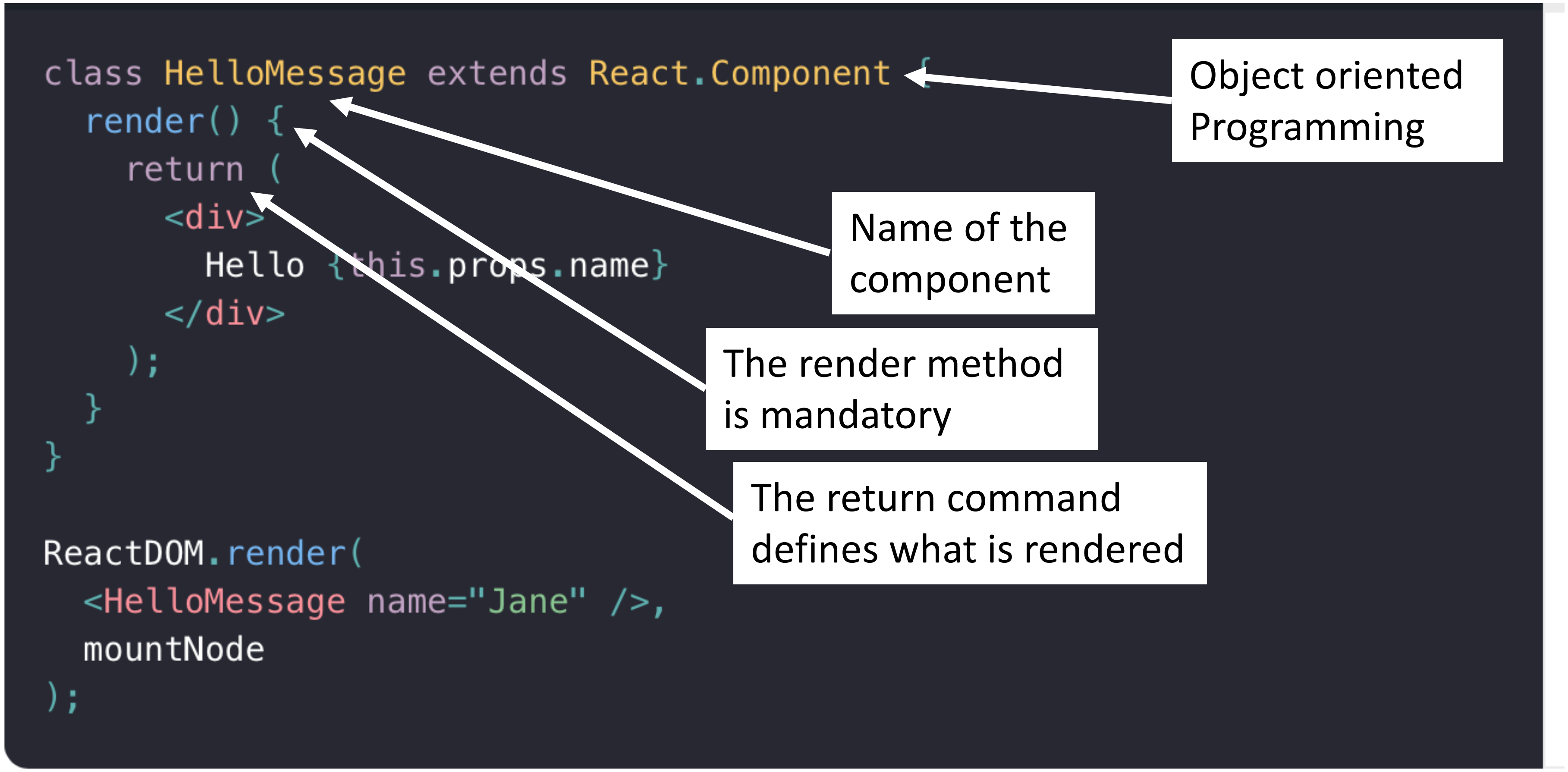


```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

Introduction to React: Component

React

- Example of a simplistic component (and how to instantiate it)



```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

Object oriented Programming

Name of the component

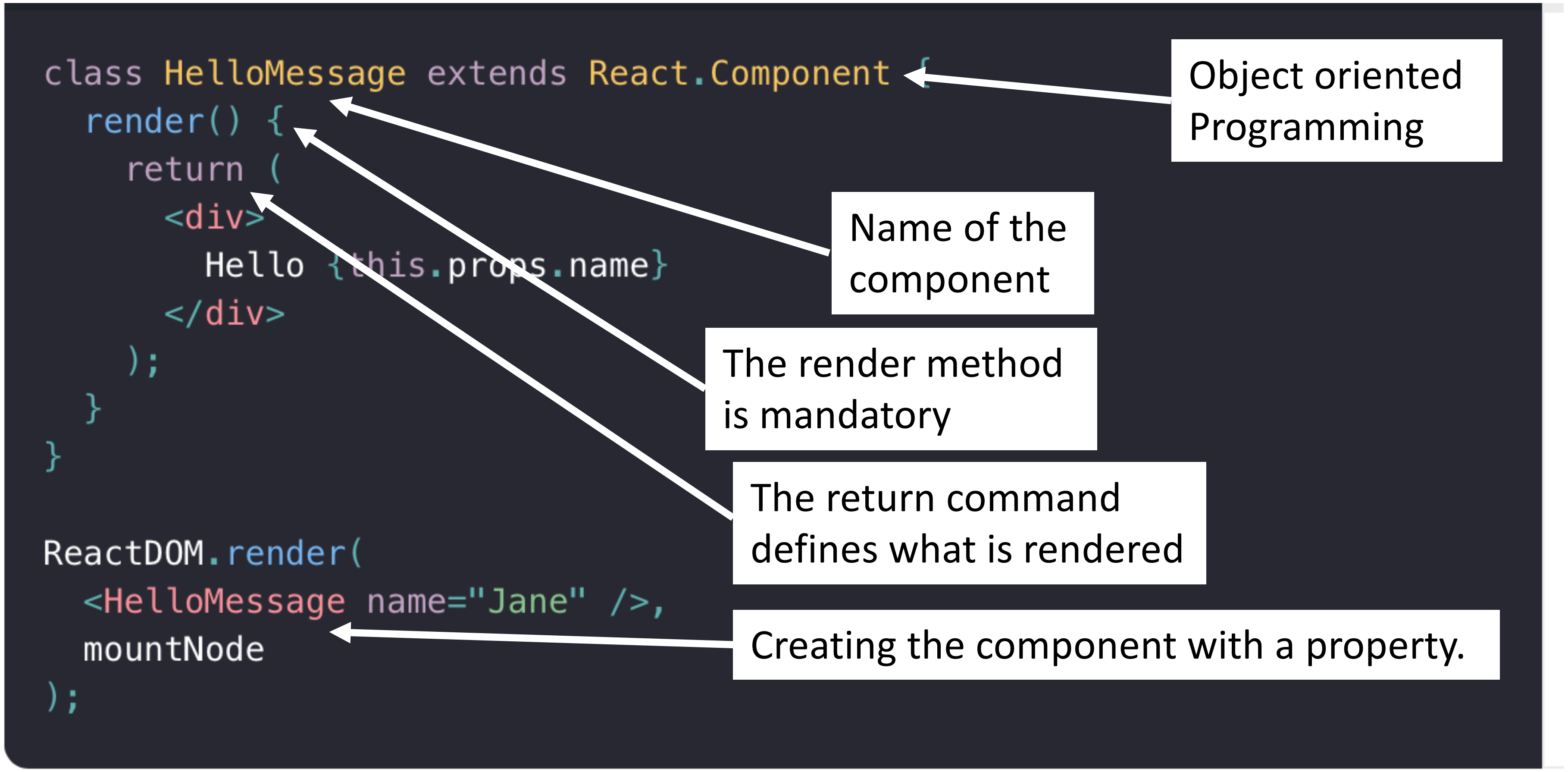
The render method is mandatory

The return command defines what is rendered

Introduction to React: Component

React

- Example of a simplistic component (and how to instantiate it)



```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

Object oriented Programming

Name of the component

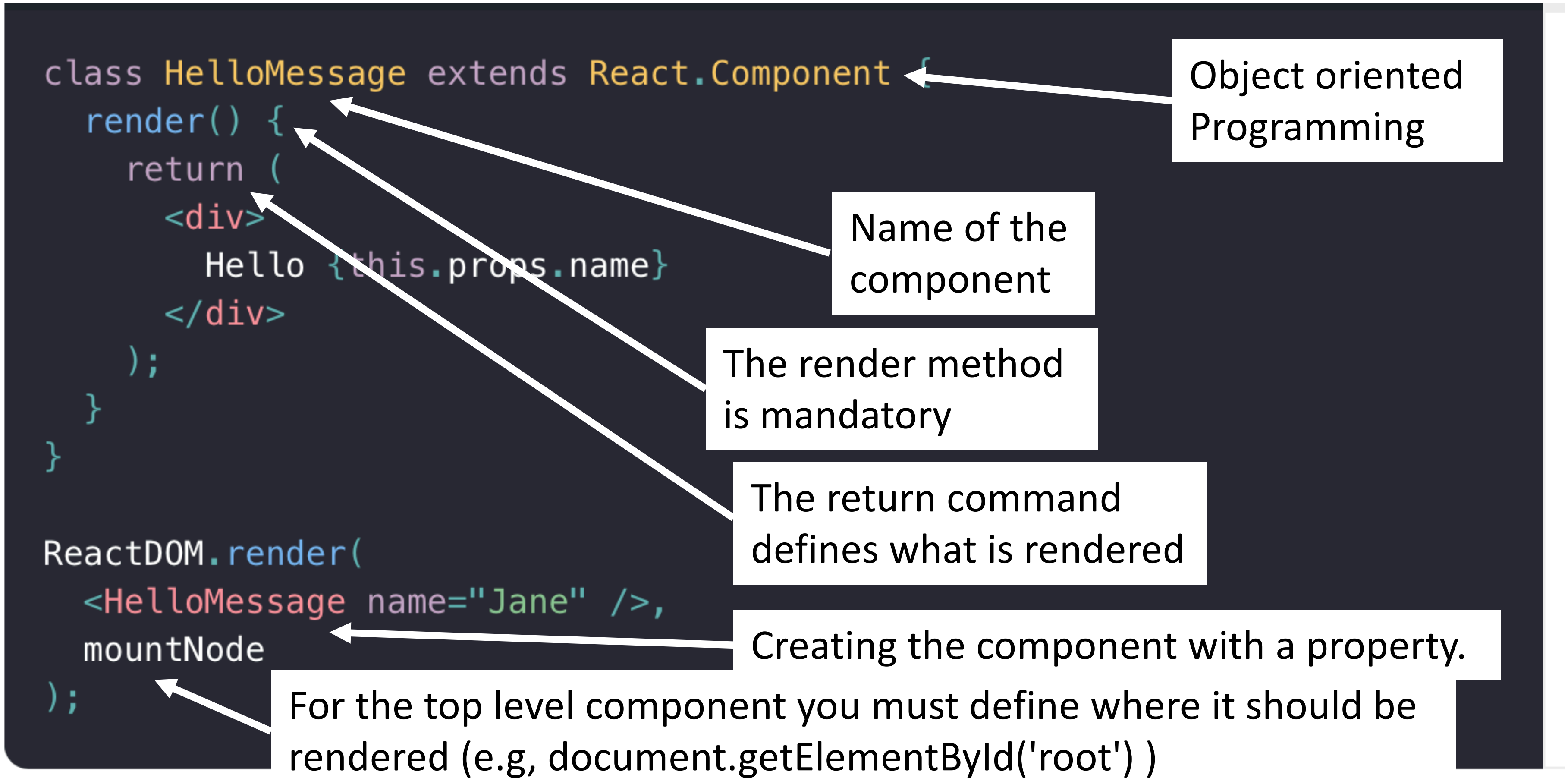
The render method is mandatory

The return command defines what is rendered

Creating the component with a property.

Introduction to React: Component

- Example of a simplistic component (and how to instantiate it)



```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

Object oriented Programming

Name of the component

The render method is mandatory

The return command defines what is rendered

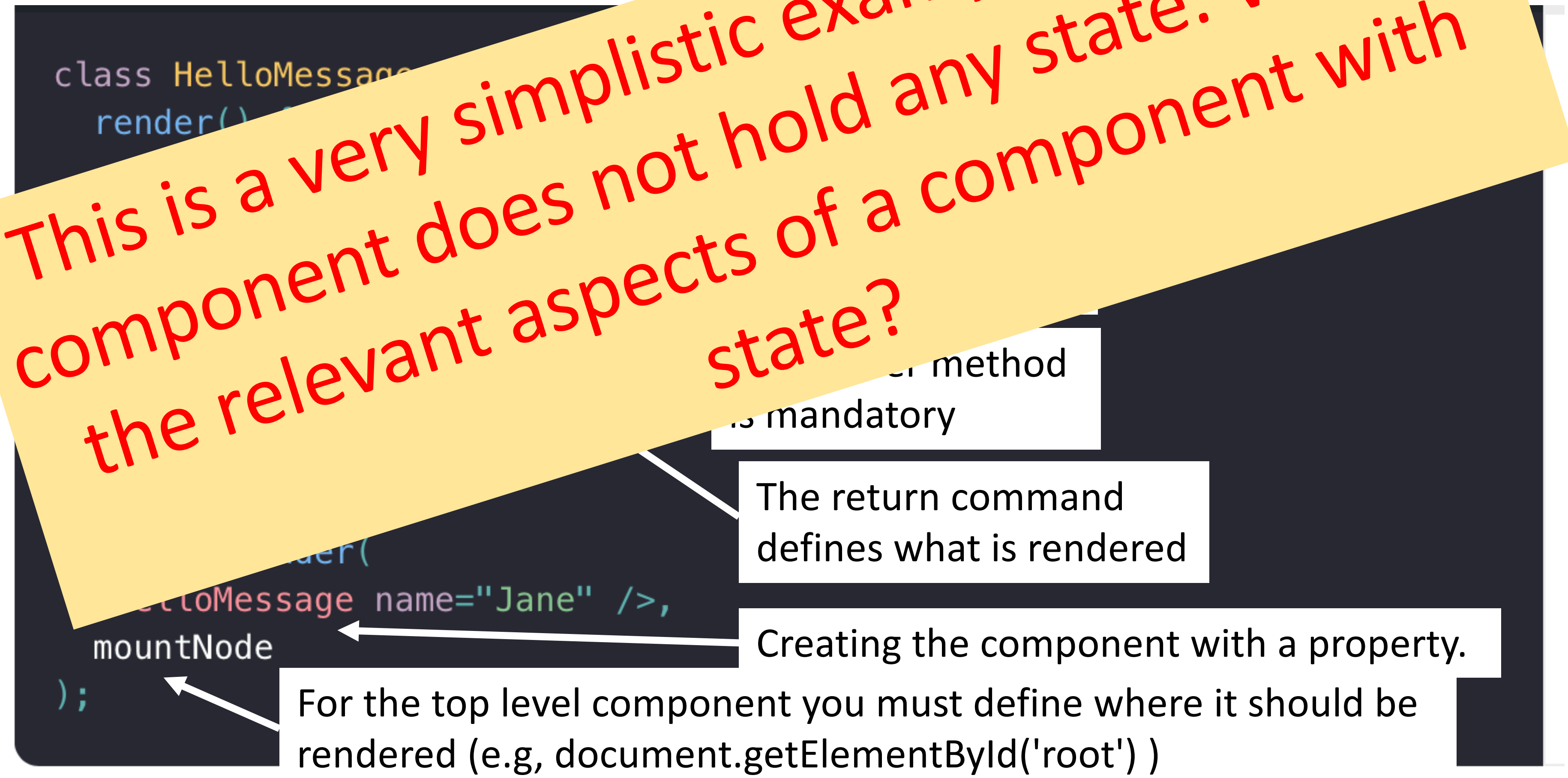
Creating the component with a property.

For the top level component you must define where it should be rendered (e.g, document.getElementById('root'))

Introduction to React: Component

- Example of a simplistic component (
  instantiate it)

This is a very simplistic example where the component does not hold any state. What is the relevant aspects of a component with state?



```
class HelloMessag
  render() {

        der(
  lloMessage name="Jane" />,
  mountNode
);
```

r method
s mandatory

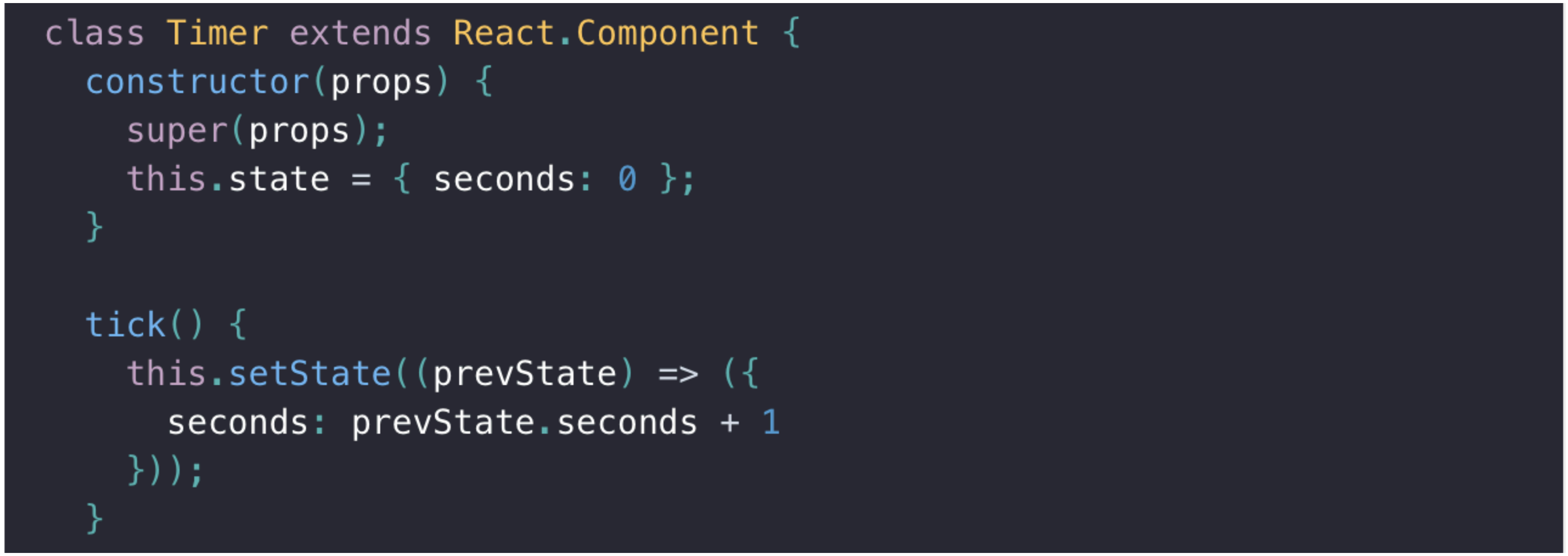
The return command defines what is rendered

Creating the component with a property.

For the top level component you must define where it should be rendered (e.g, document.getElementById('root'))

Introduction to React: Component (+complex)

- Initialization and Functions

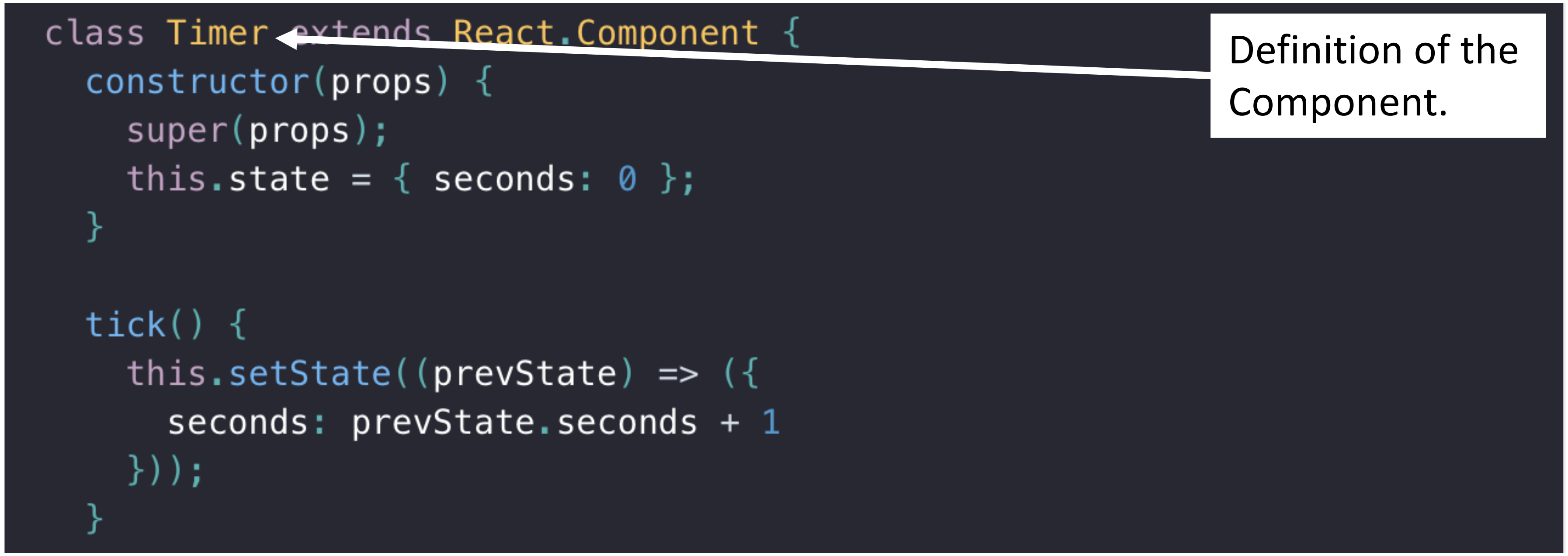


```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }
```

Introduction to React: Component (+complex)

React

- Initialization and Functions

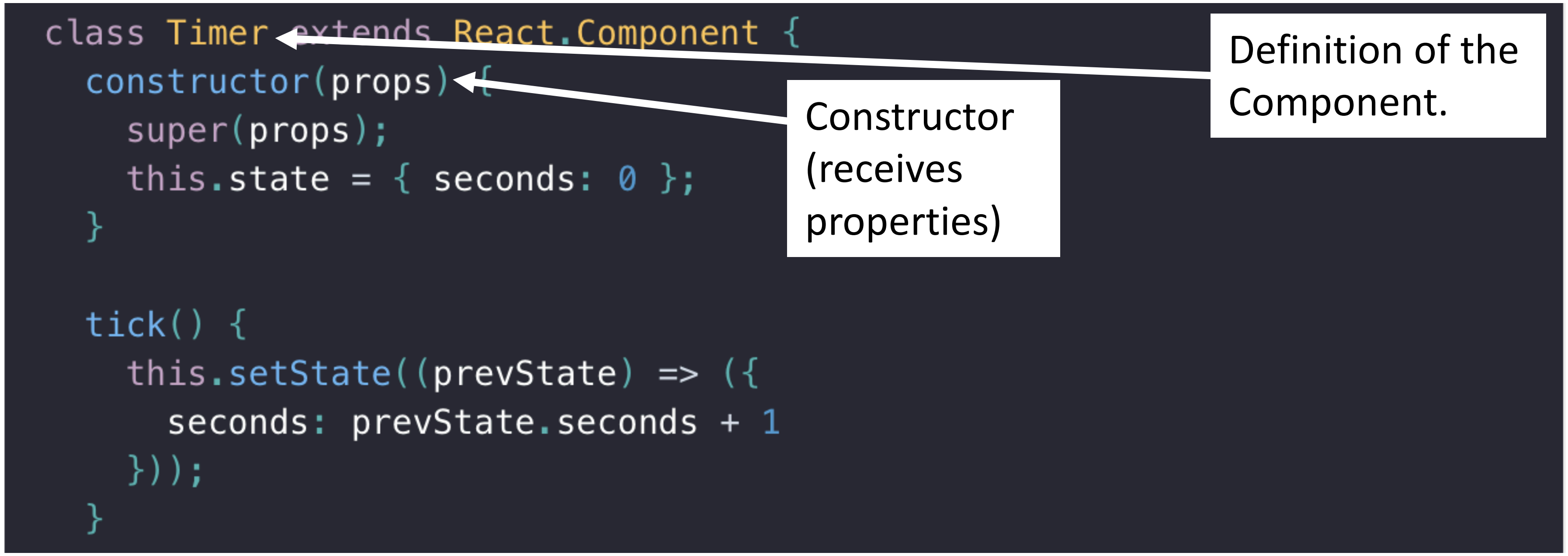


```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }
```

Definition of the Component.

Introduction to React: Component (+complex)

- Initialization and Functions



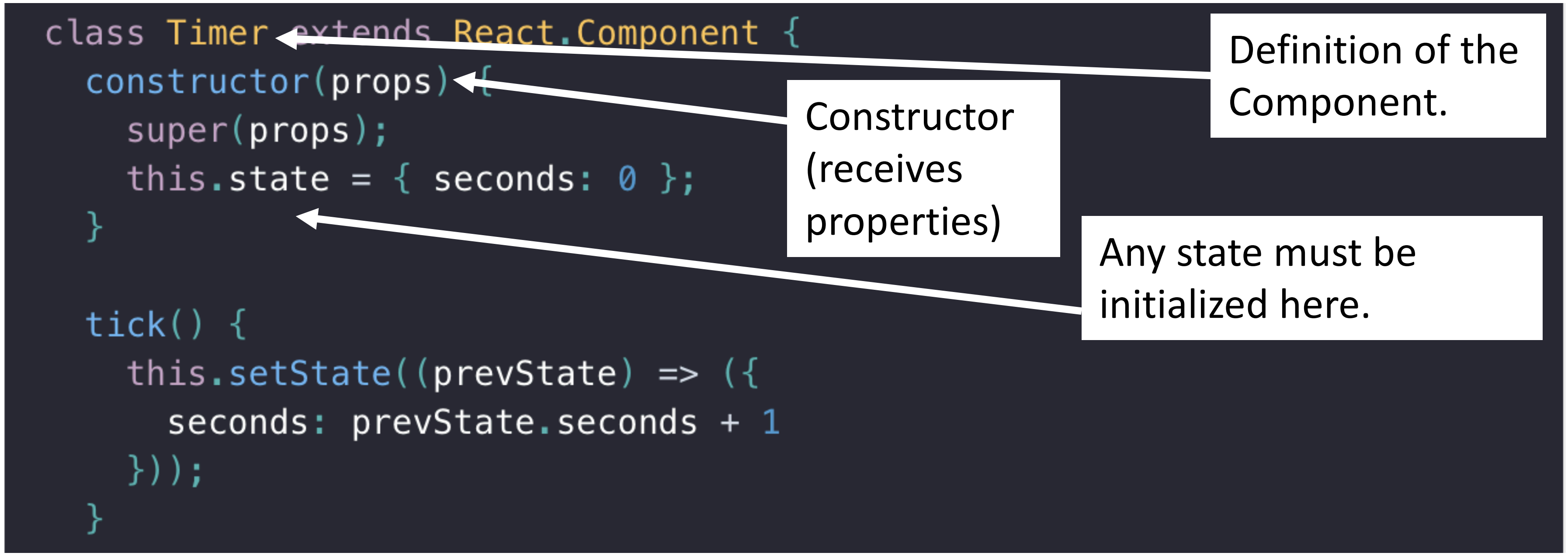
```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }
```

Definition of the Component.
Constructor (receives properties)

Introduction to React: Component (+complex)

React

- Initialization and Functions



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

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }
```

Definition of the Component.

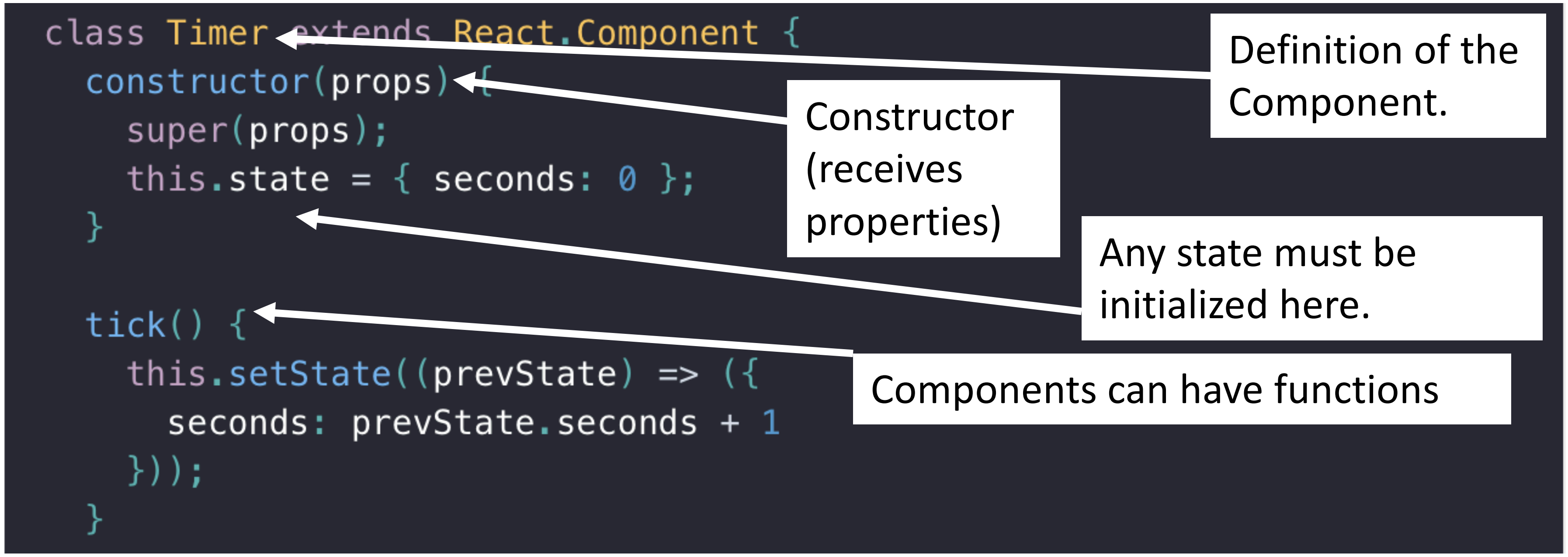
Constructor (receives properties)

Any state must be initialized here.

Introduction to React: Component (+complex)

React

- Initialization and Functions



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

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }
```

Definition of the Component.

Constructor (receives properties)

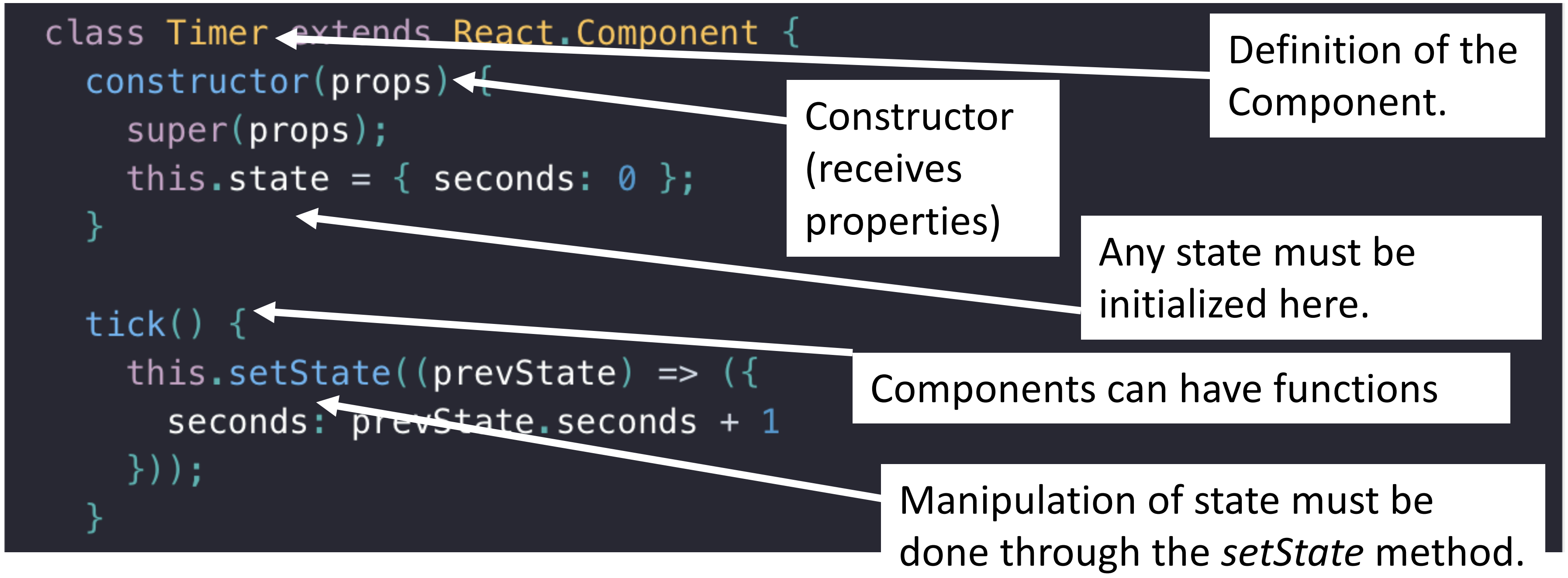
Any state must be initialized here.

Components can have functions

Introduction to React: Component (+complex)

React

- Initialization and Functions



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

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }
```

Definition of the Component.

Constructor (receives properties)

Any state must be initialized here.

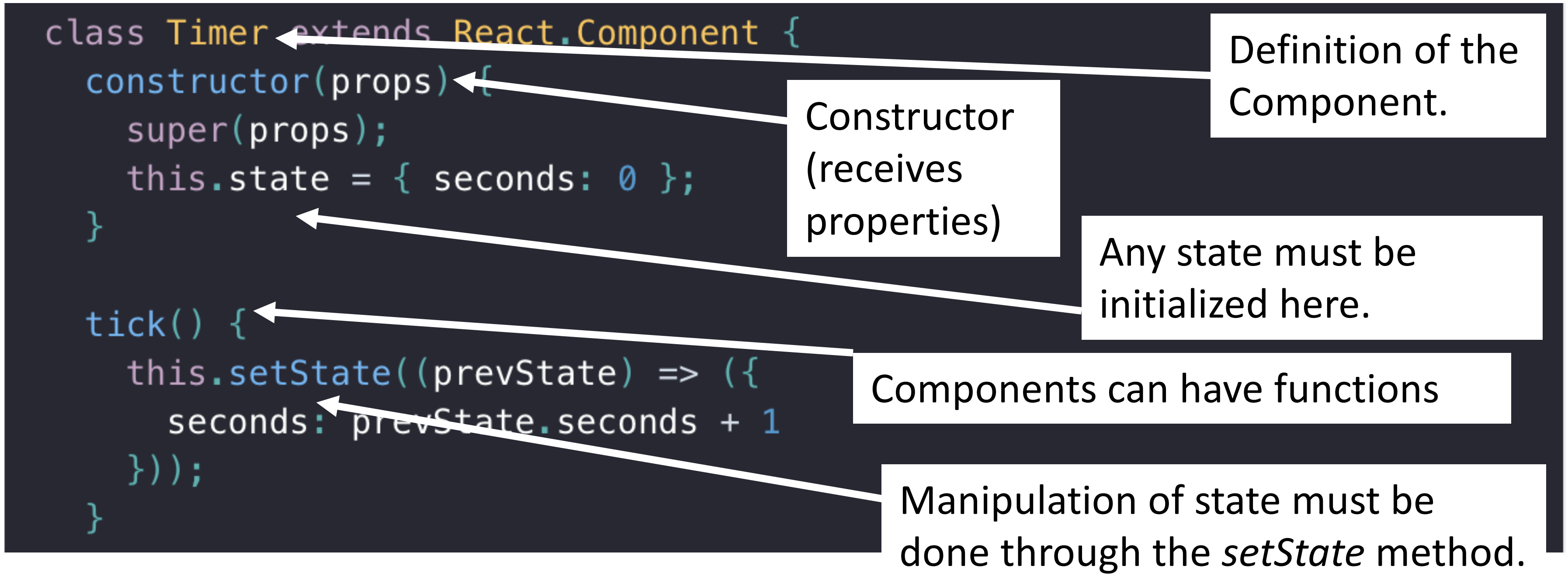
Components can have functions

Manipulation of state must be done through the *setState* method.

Introduction to React: Component (+complex)

React

- Initialization and Functions



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

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }
```

Definition of the Component.

Constructor (receives properties)

Any state must be initialized here.

Components can have functions

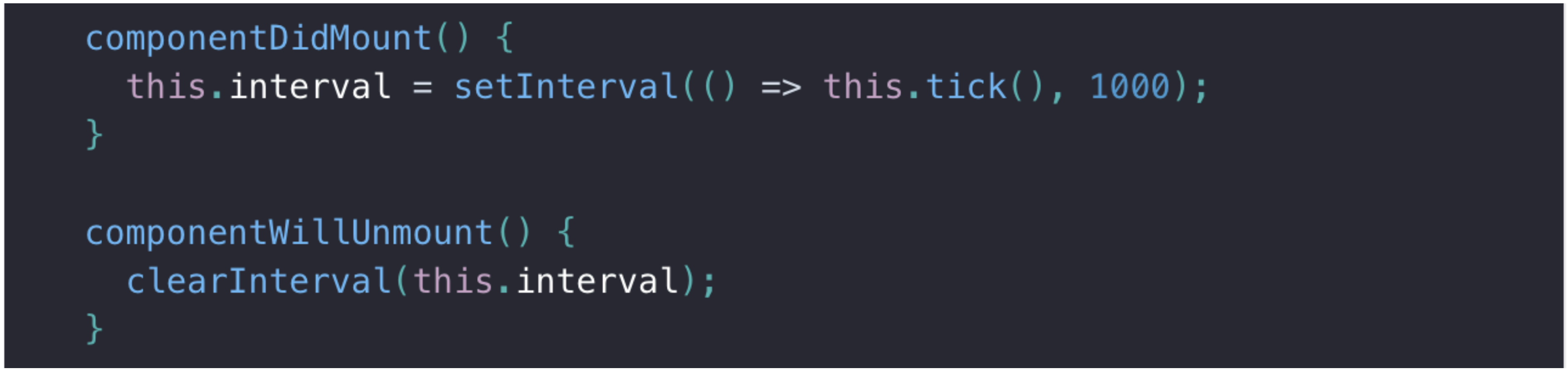
Manipulation of state must be done through the *setState* method.

While you must use the *setState* function to modify state, there are multiple ways to do it. “Reading” the state can be done by direct access (e.g, this.state.seconds)

Introduction to React: Component (+complex)

React

- On Mount and On UnMount

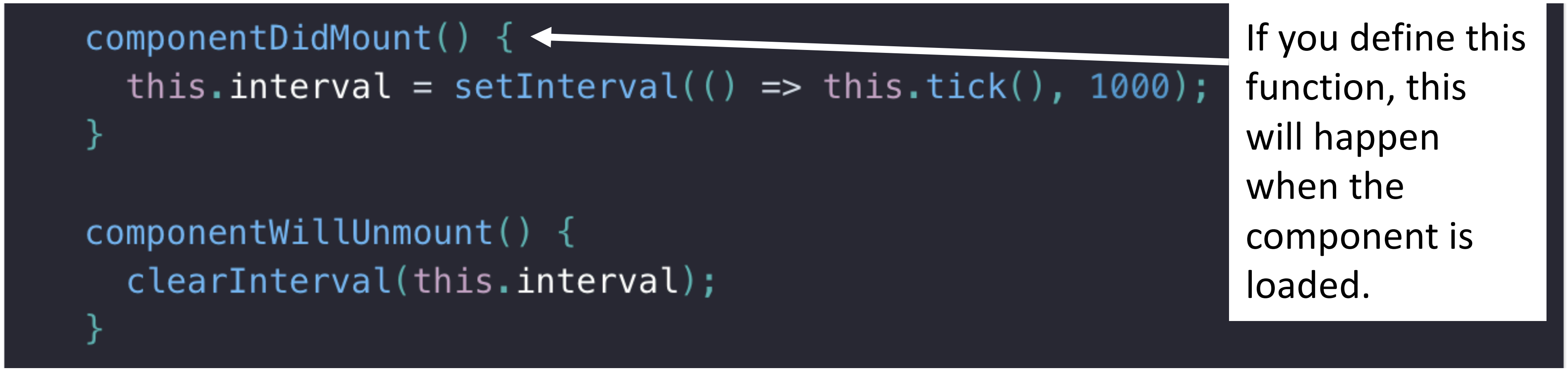


```
componentDidMount() {
  this.interval = setInterval(() => this.tick(), 1000);
}

componentWillUnmount() {
  clearInterval(this.interval);
}
```

Introduction to React: Component (+complex)

React

- On Mount and On UnMount



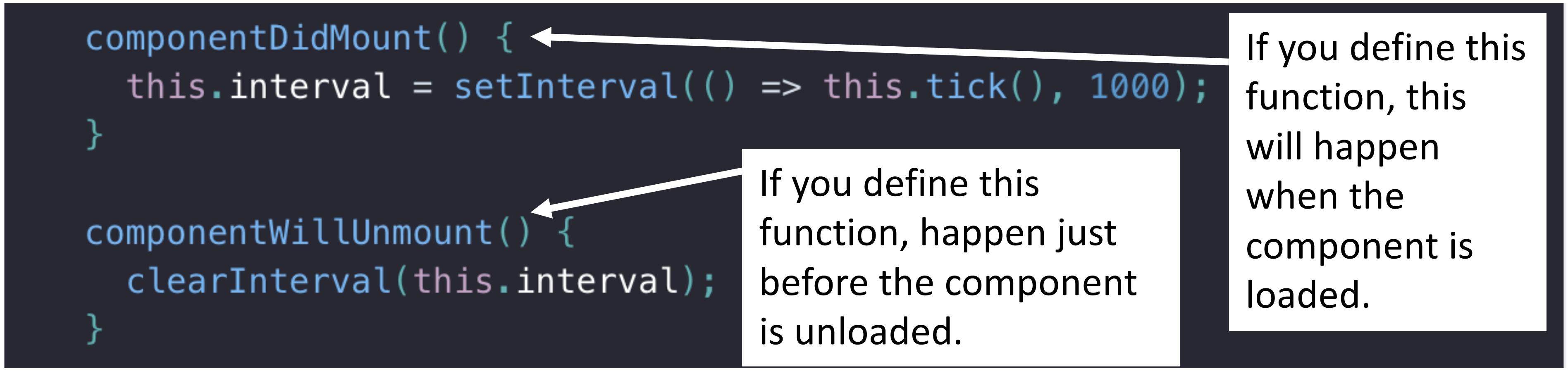
```
componentDidMount() {
  this.interval = setInterval(() => this.tick(), 1000);
}

componentWillUnmount() {
  clearInterval(this.interval);
}
```

If you define this function, this will happen when the component is loaded.

Introduction to React: Component (+complex)

React

- On Mount and On UnMount



```
componentDidMount() {
  this.interval = setInterval(() => this.tick(), 1000);
}

componentWillUnmount() {
  clearInterval(this.interval);
}
```

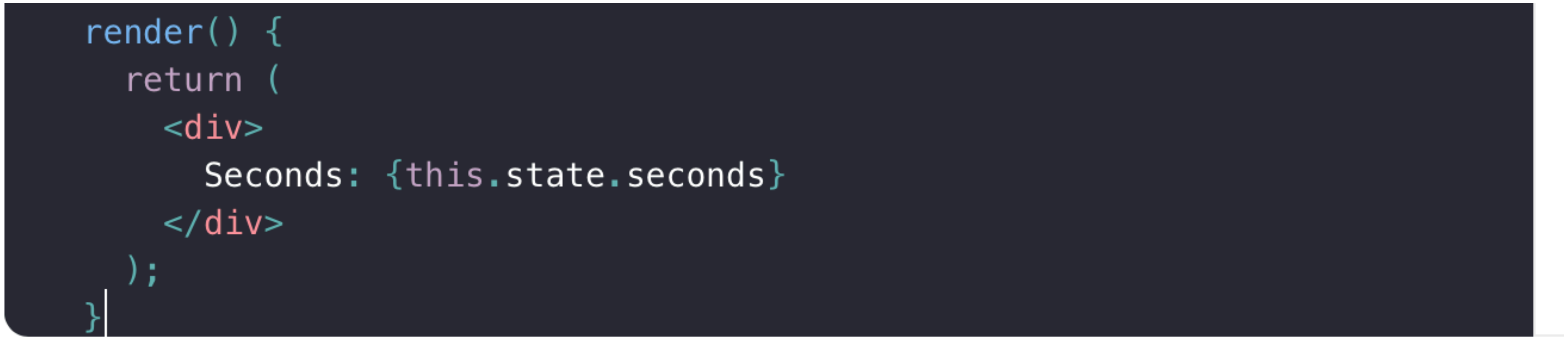
If you define this function, this will happen when the component is loaded.

If you define this function, happen just before the component is unloaded.

Introduction to React: Component (+complex)

React

- Render and Adding other Components

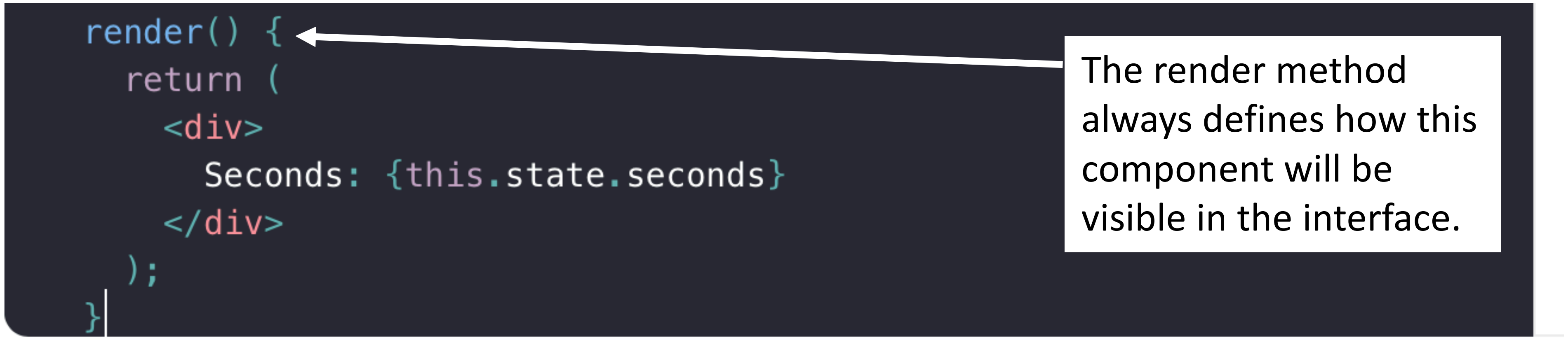


```
render() {
  return (
    <div>
      Seconds: {this.state.seconds}
    </div>
  );
}
```

Introduction to React: Component (+complex)

React

- Render and Adding other Components



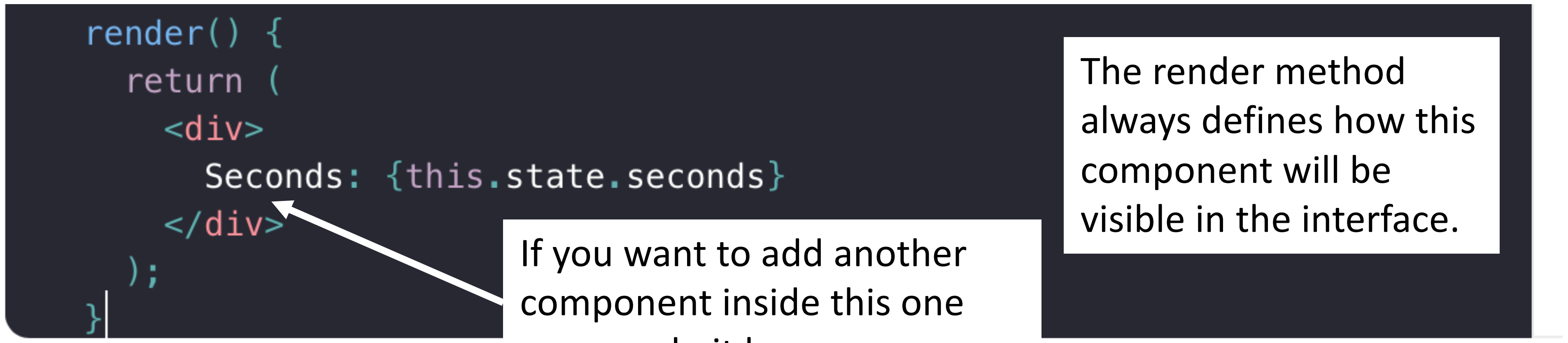
```
render() {
  return (
    <div>
      Seconds: {this.state.seconds}
    </div>
  );
}
```

The render method always defines how this component will be visible in the interface.

Introduction to React: Component (+complex)

React

- Render and Adding other Components



```
render() {
  return (
    <div>
      Seconds: {this.state.seconds}
    </div>
  );
}
```

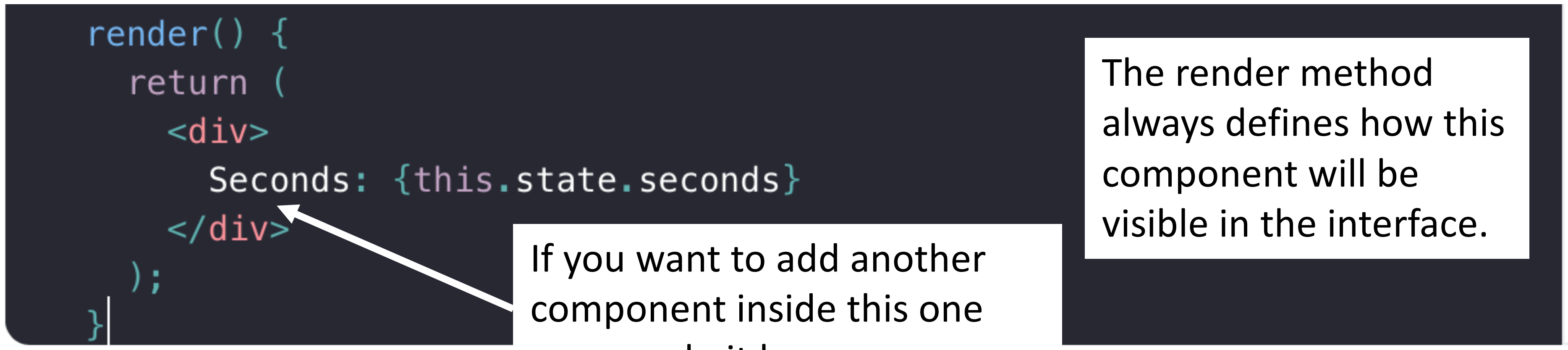
If you want to add another component inside this one you can do it here.

The render method always defines how this component will be visible in the interface.

Introduction to React: Component (+complex)

React

- Render and Adding other Components



```
render() {
  return (
    <div>
      Seconds: {this.state.seconds}
    </div>
  );
}
```

If you want to add another component inside this one you can do it here.

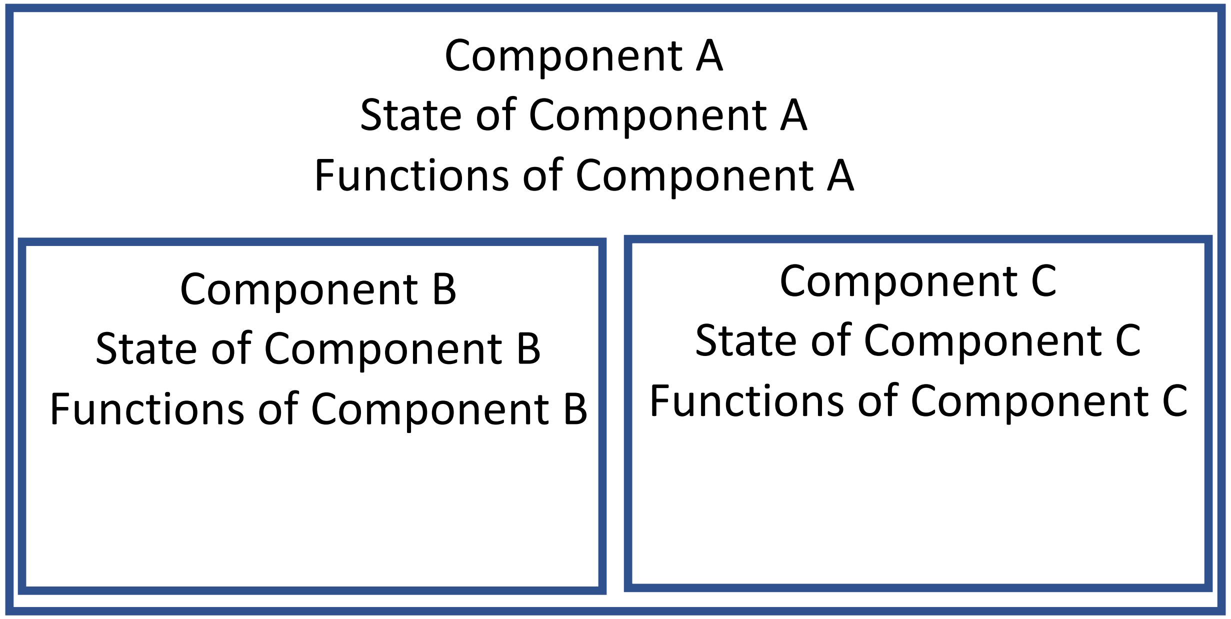
The render method always defines how this component will be visible in the interface.

For instance, imagine you have another component, called ComponentA (that does not receive any state) and you want to add it to this component. You can simply do this by writing here: <ComponentA />. React will handle the rest.

Introduction to React: Component Hiearchy

React

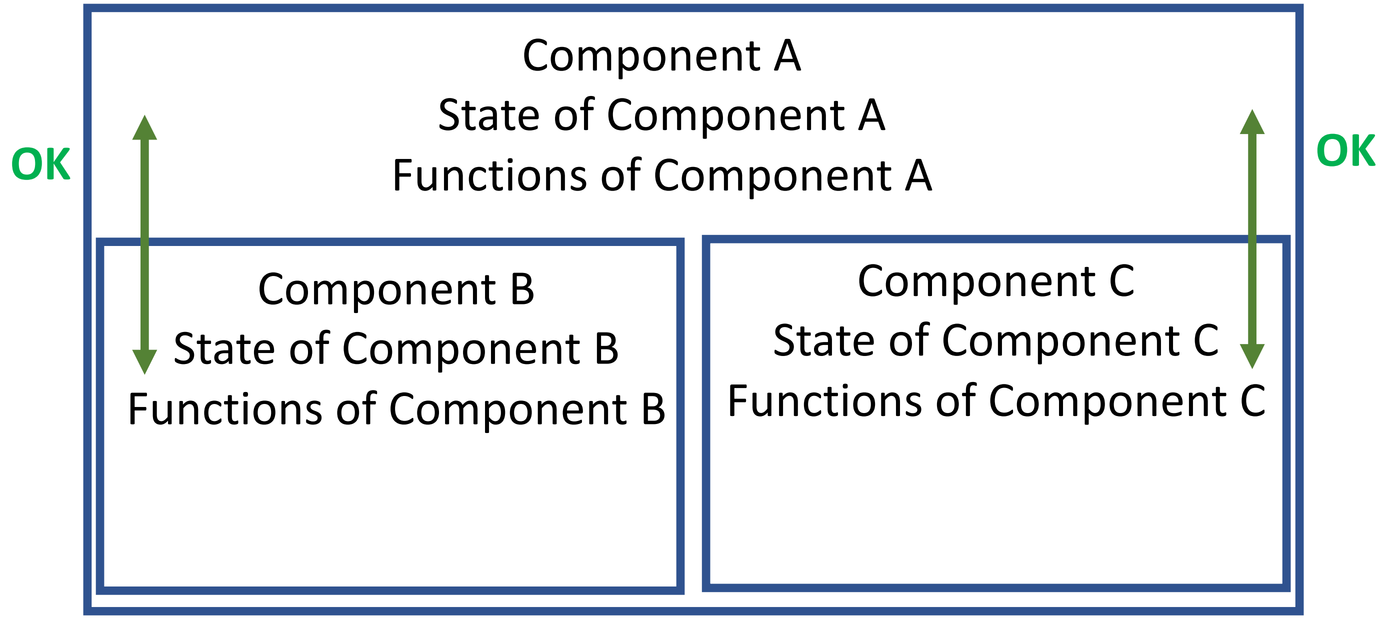
- Components in React applications are usually organized in an hiearchy.



Introduction to React: Component Hiearchy

React

- Components in React applications are usually organized in an hiearchy.

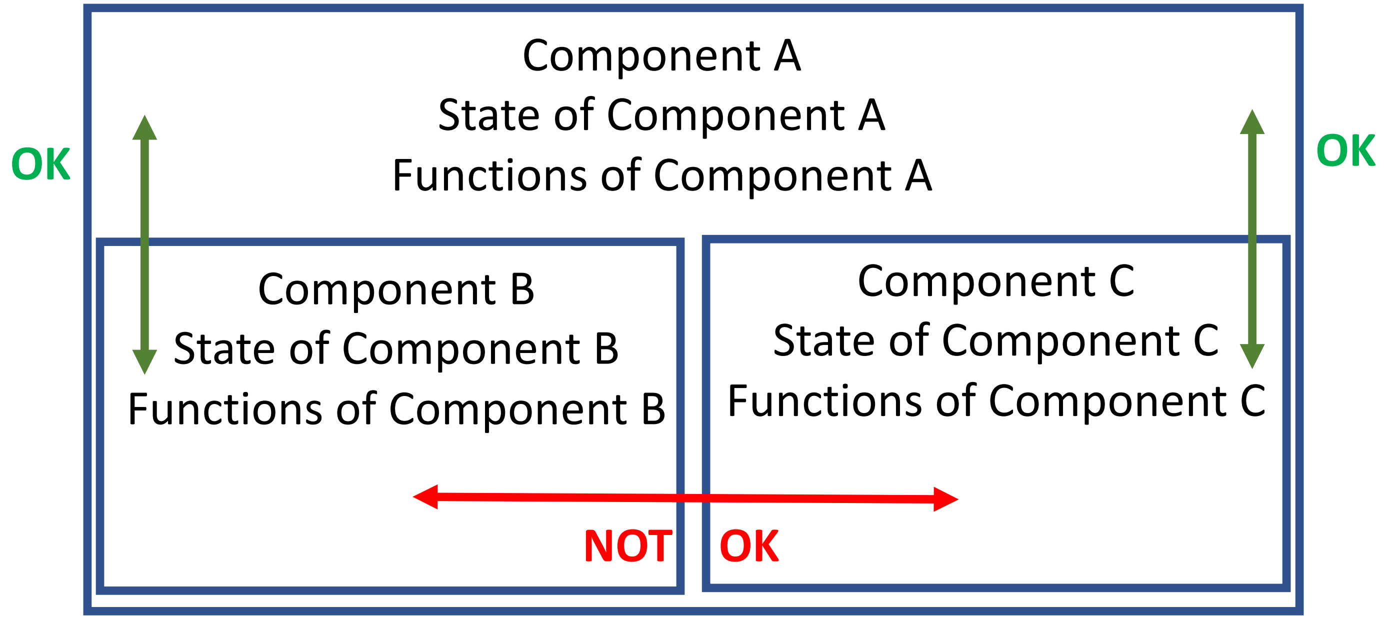


Available Interactions and Data Flows

Introduction to React: Component Hiearchy

React

- Components in React applications are usually organized in an hiearchy.

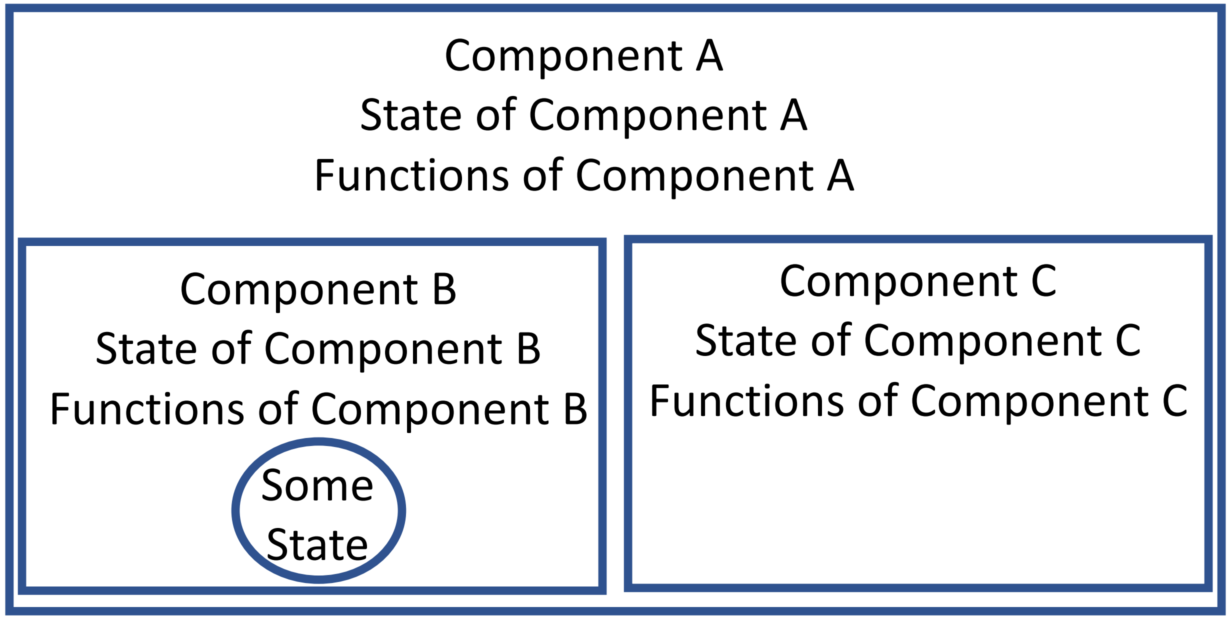


NOT Available Interactions and Data Flows

Introduction to React: Component Hiearchy

React

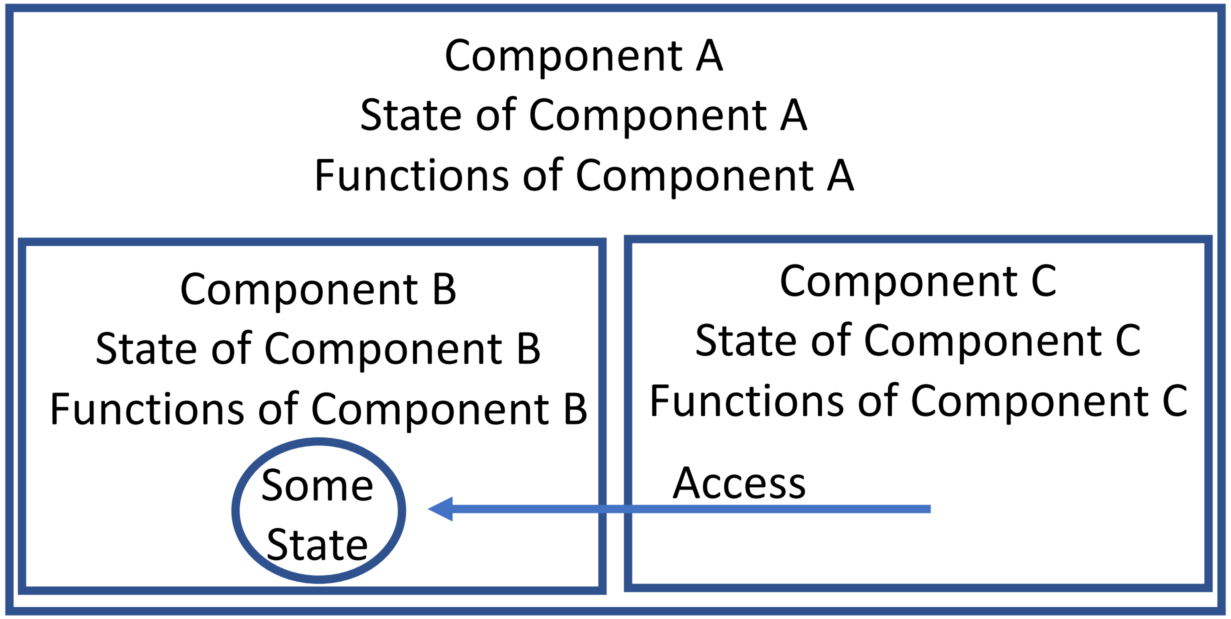
- Components in React applications are usually organized in an hiearchy.



Introduction to React: Component Hiearchy

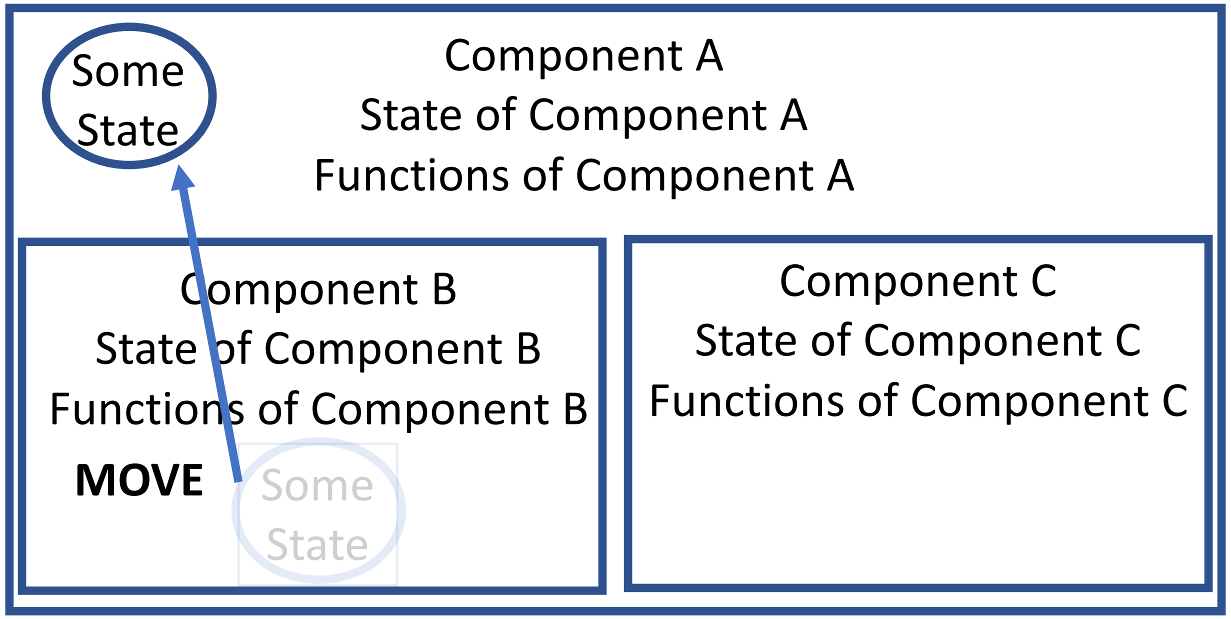
React

- Components in React applications are usually organized in an hiearchy.



Introduction to React: Component Hiearchy

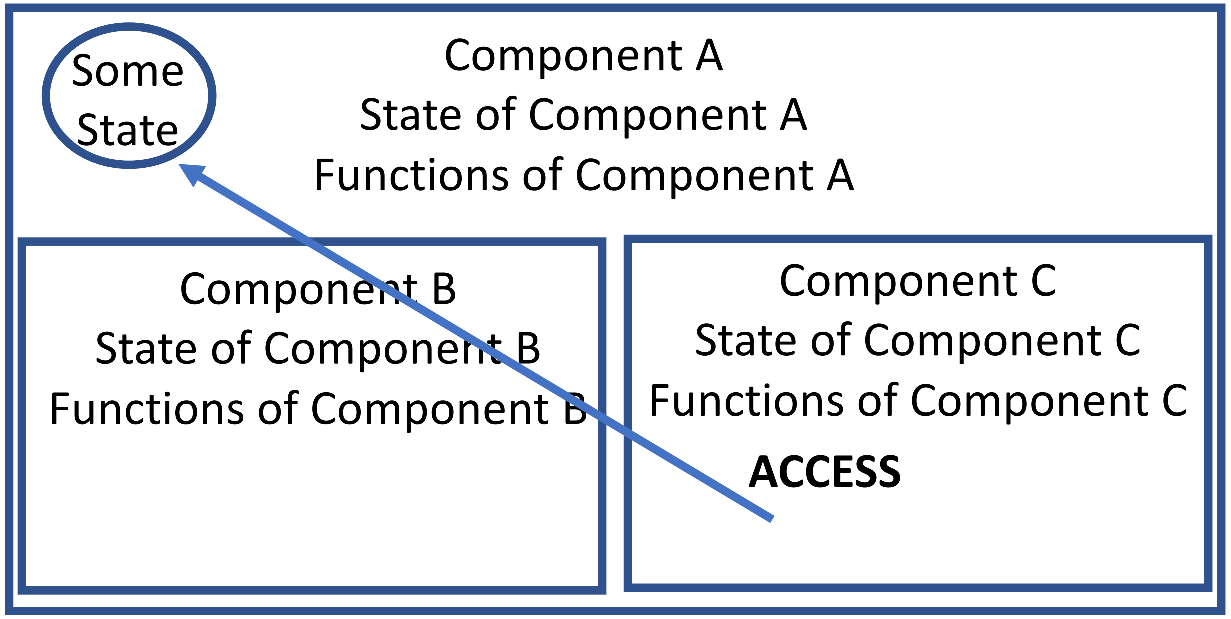
- Components in React applications are usually organized in an hiearchy.



Introduction to React: Component Hiearchy

React

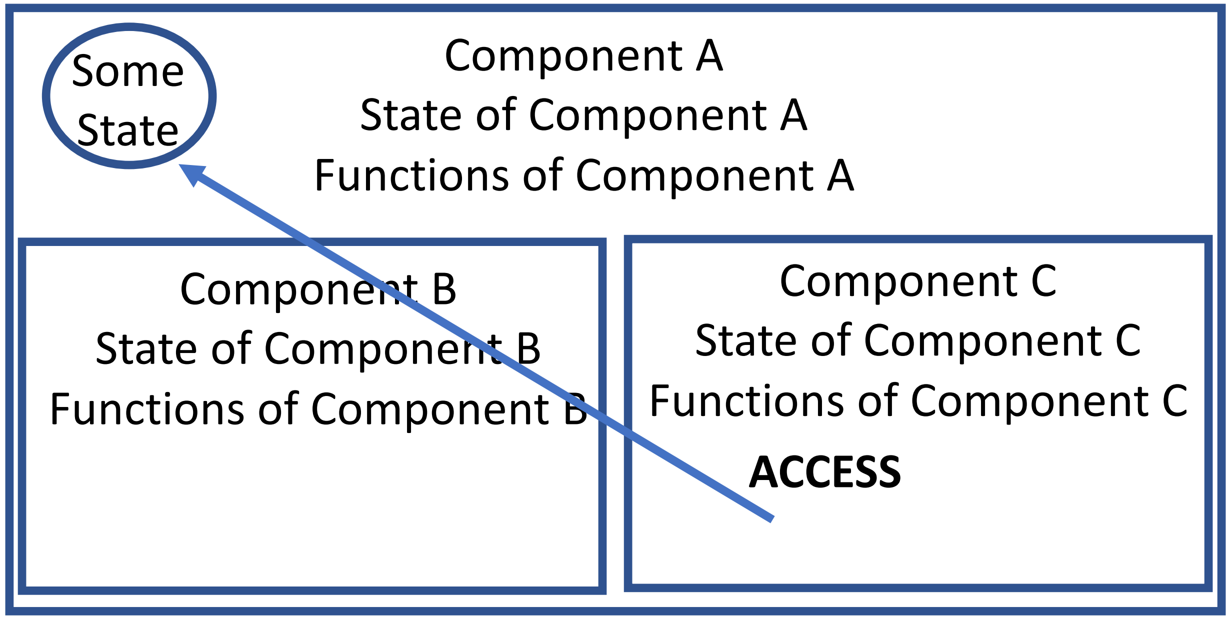
- Components in React applications are usually organized in an hiearchy.



Introduction to React: Component Hiearchy

React

- Components in React applications are usually organized in an hiearchy.



This might require the creation of Components that simply encapsulate other components and hold state to be shared among them.

Introduction to React: Component Hiearchy

React

Want to learn more, and revise these concepts, maybe get some guidance??

Go to the **online tutorial**:

https://reactjs.org/tutorial/tutorial.html

This might require the creation of Components that simply encapsulate other components and hold state to be shared among them.

Todays Exercise…

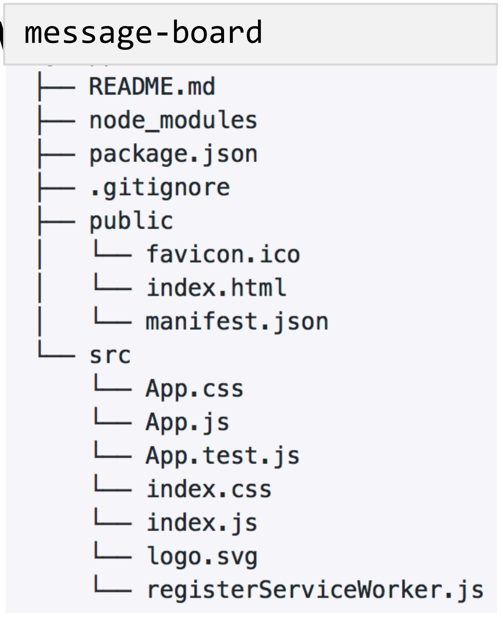
- Lets pick up the example from last weeks, and remake it as a React application:
  - Your HTML file will only have contents in the head and a single (emply) div with id="root" on the body, everything else will be done with React.
  - Let's decompose de application in components and write the code for each component (we will see that next).
  - We will keep the CSS file we did on our first iteration of this application and ensure that the style remains unchanged.

Setting up node and creating your project

- First of all, install node.js:
  - Linux: **sudo apt-get install nodejs**
  - Mac:
    - Install brew: **/usr/bin/ruby -e "$(curl -fsSL [https://raw.githubusercontent.com/Homebrew/install/master/install)](https://raw.githubusercontent.com/Homebrew/install/master/install)”**)
    - brew install node
  - Windows: goto: **https://nodejs.org/en/download/**

Setting up node and creating your project

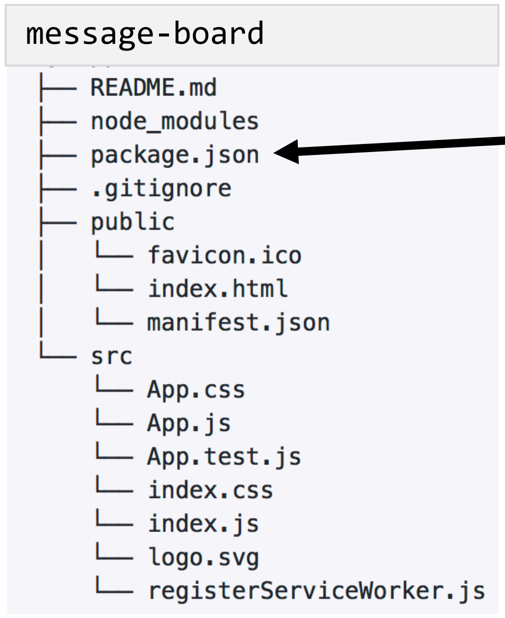
- Lets create your project, on the terminal type:
- **npm install –g create-react-app**
- **create-react-app message-board**
- This will create a directory named message-board with the follo



```
message-board
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

Setting up node and creating your project

- Lets create your project, on the terminal type:
- create-react-app message-board
- This will create a directory named message-board with the following structure:

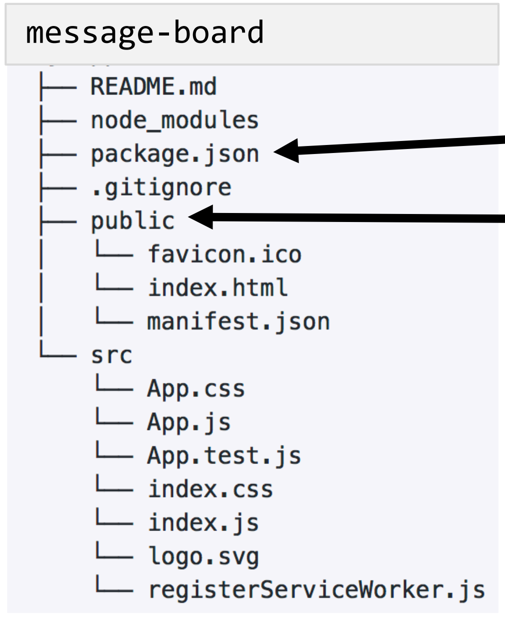


```
message-board
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

This is the file controlling the dependencies of your project.

Setting up node and creating your project

- Lets create your project, on the terminal type:
- create-react-app message-board
- This will create a directory named message-board with the following structure:



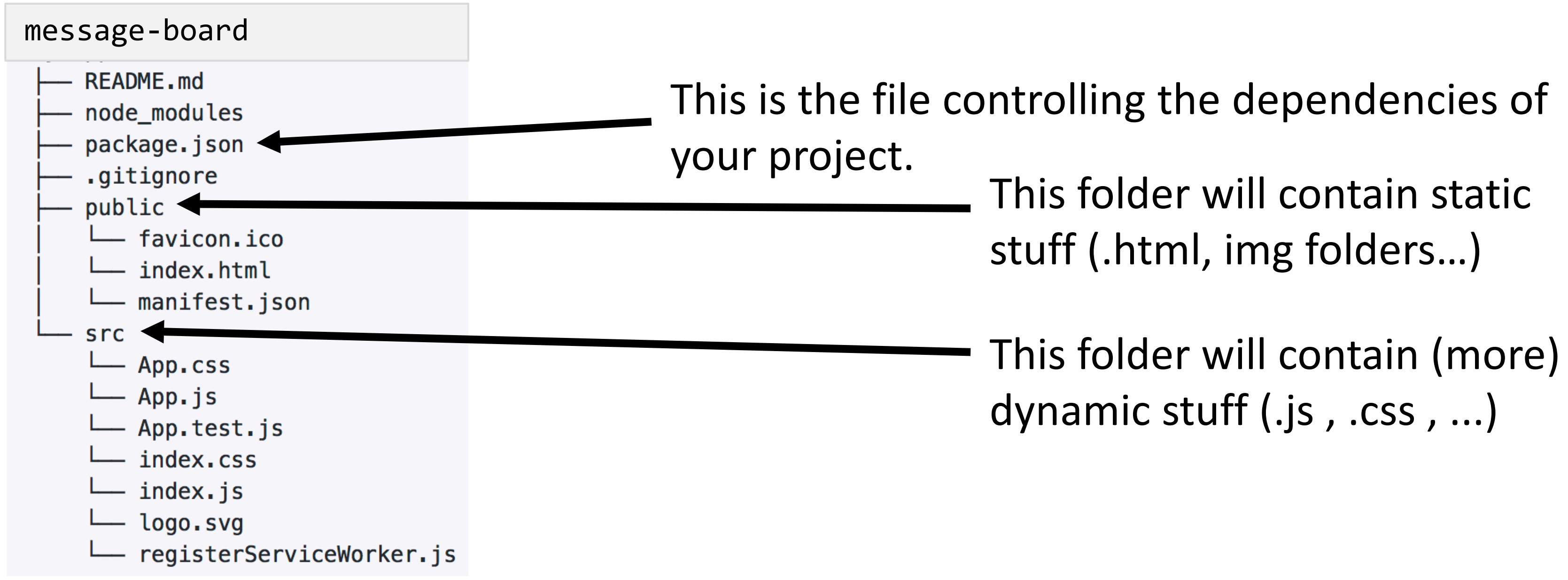
```
message-board
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

This is the file controlling the dependencies of your project.

This folder will contain static stuff (.html, img folders…)

Setting up node and creating your project

- Lets create your project, on the terminal type:
- create-react-app message-board
- This will create a directory named message-board with the following structure:



```
message-board
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

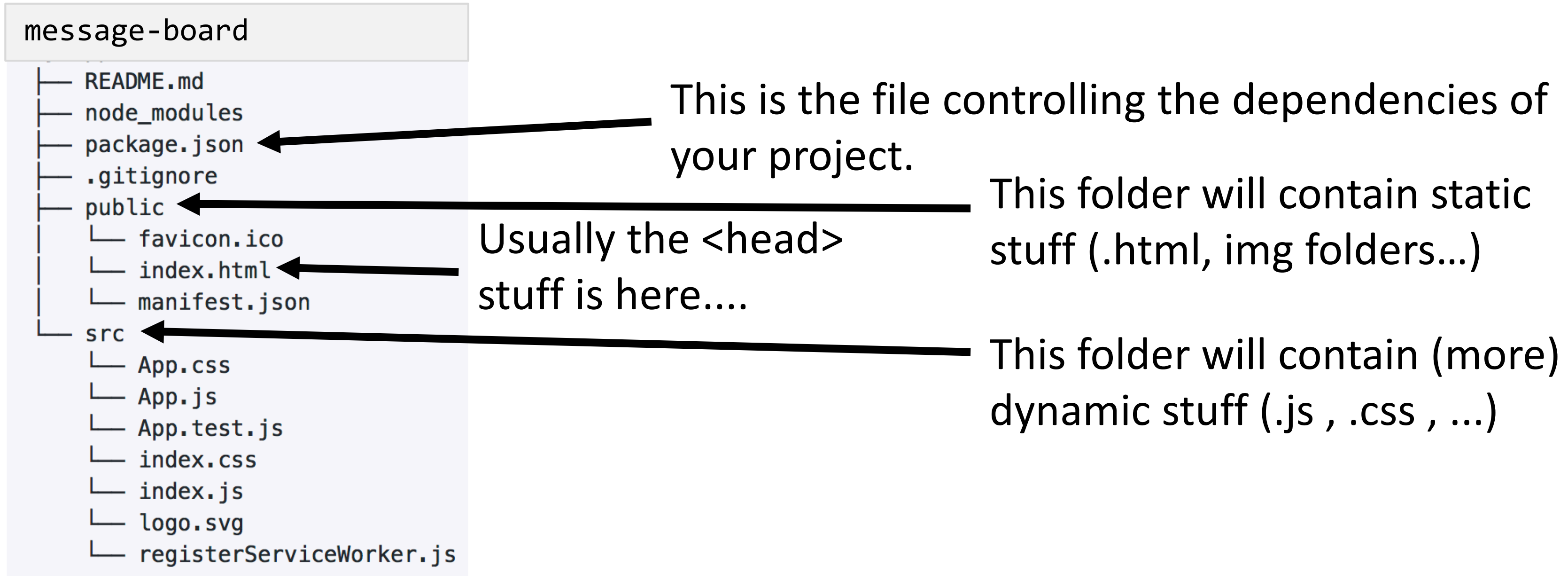
This is the file controlling the dependencies of your project.

This folder will contain static stuff (.html, img folders…)

This folder will contain (more) dynamic stuff (.js , .css , ...)

Setting up node and creating your project

- Lets create your project, on the terminal type:
- create-react-app message-board
- This will create a directory named message-board with the following structure:



```
message-board
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

This is the file controlling the dependencies of your project.

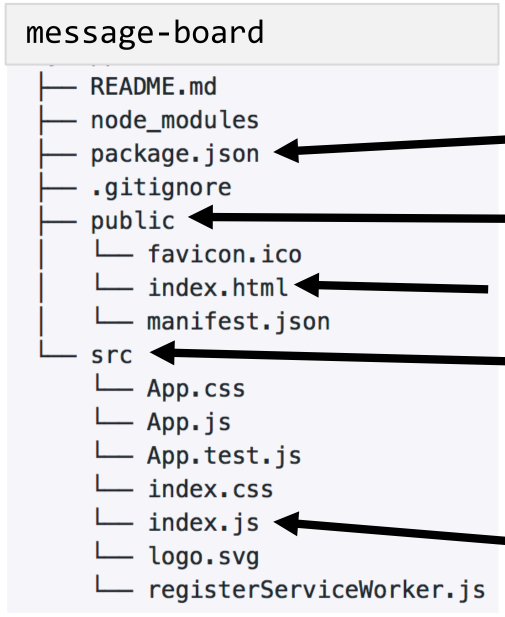
This folder will contain static stuff (.html, img folders…)

Usually the <head> stuff is here....

This folder will contain (more) dynamic stuff (.js , .css , ...)

Setting up node and creating your project

- Lets create your project, on the terminal type:
- create-react-app message-board
- This will create a directory named message-board with the following structure:



```
message-board
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

This is the file controlling the dependencies of your project.

This folder will contain static stuff (.html, img folders…)

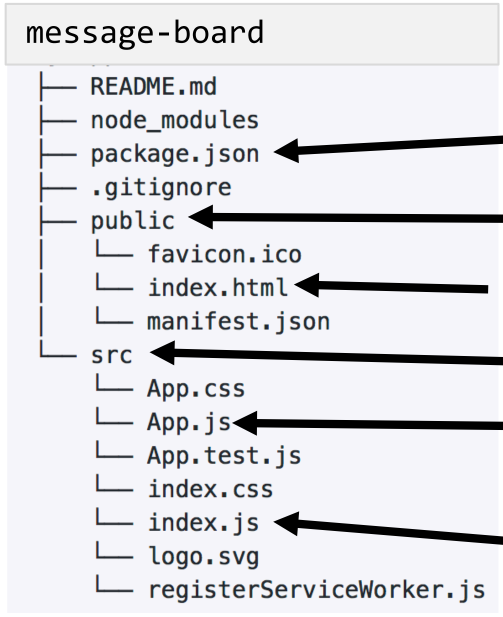
Usually the <head> stuff is here....

This folder will contain (more) dynamic stuff (.js , .css , ...)

This file will load the main component of your application into the page.

Setting up node and creating your project

- Lets create your project, on the terminal type:
- create-react-app message-board
- This will create a directory named message-board with the following structure:



```
message-board
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   └── favicon.ico
│   └── index.html
│   └── manifest.json
└── src
    └── App.css
    └── App.js
    └── App.test.js
    └── index.css
    └── index.js
    └── logo.svg
    └── registerServiceWorker.js
```

This is the file controlling the dependencies of your project.

This folder will contain static stuff (.html, img folders…)

Usually the <head> stuff is here....

This folder will contain (more) dynamic stuff (.js , .css , ...)

Your app logic (and components) is here

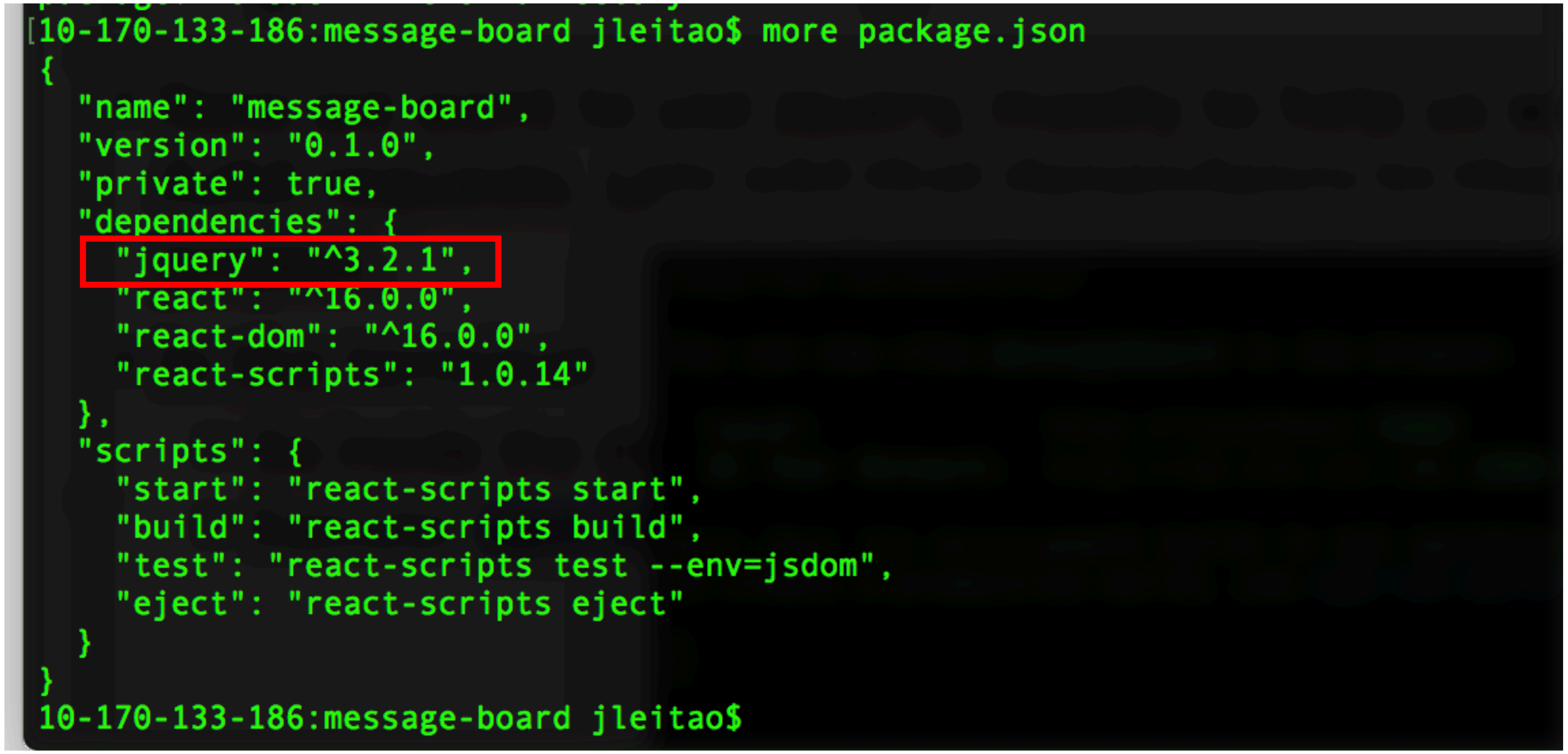
This file will load the main component of your application into the page.

Setting up node and creating your project

- Now we want to use jquery, mostly to help us with AJAX requests. Lets add that dependency to the project.
- In the terminal:
  - Go inside the directory you created: **cd message-board**
  - Execute:
  - **npm install --save react react-dom react-scripts jquery**

Setting up node and creating your project

- To check if the dependecy was added look at the contents of package.json



```
10-170-133-186:message-board jleitao$ more package.json
{
  "name": "message-board",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "jquery": "^3.2.1",
    "react": "^16.0.0",
    "react-dom": "^16.0.0",
    "react-scripts": "1.0.14"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
10-170-133-186:message-board jleitao$
```

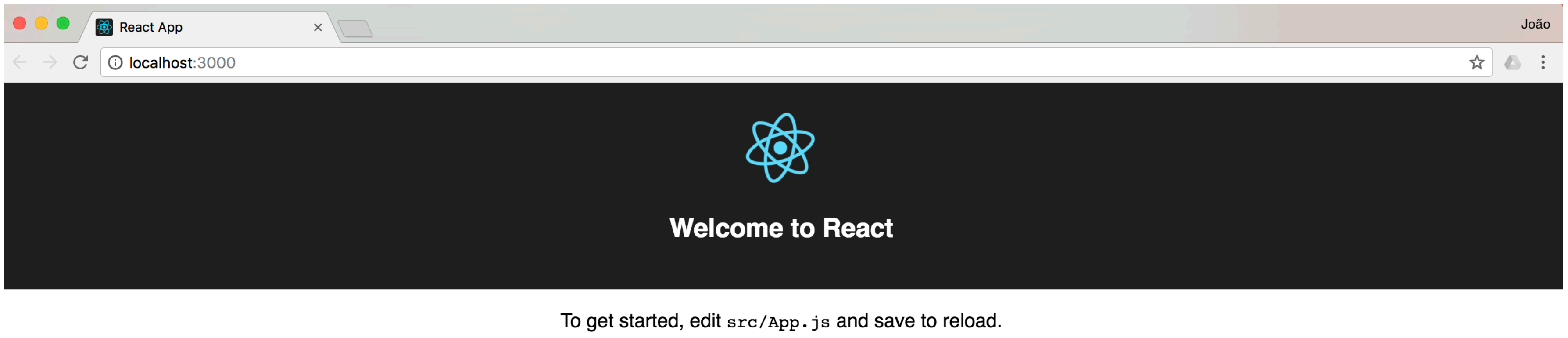
Setting up node and creating your project

- Now lets run your application locally with node.
- In the terminal execute: **npm start**

Setting up node and creating your project

- Now lets run your application locally with node.
- In the terminal execute: **npm start**

Browser: http://localhost:3000

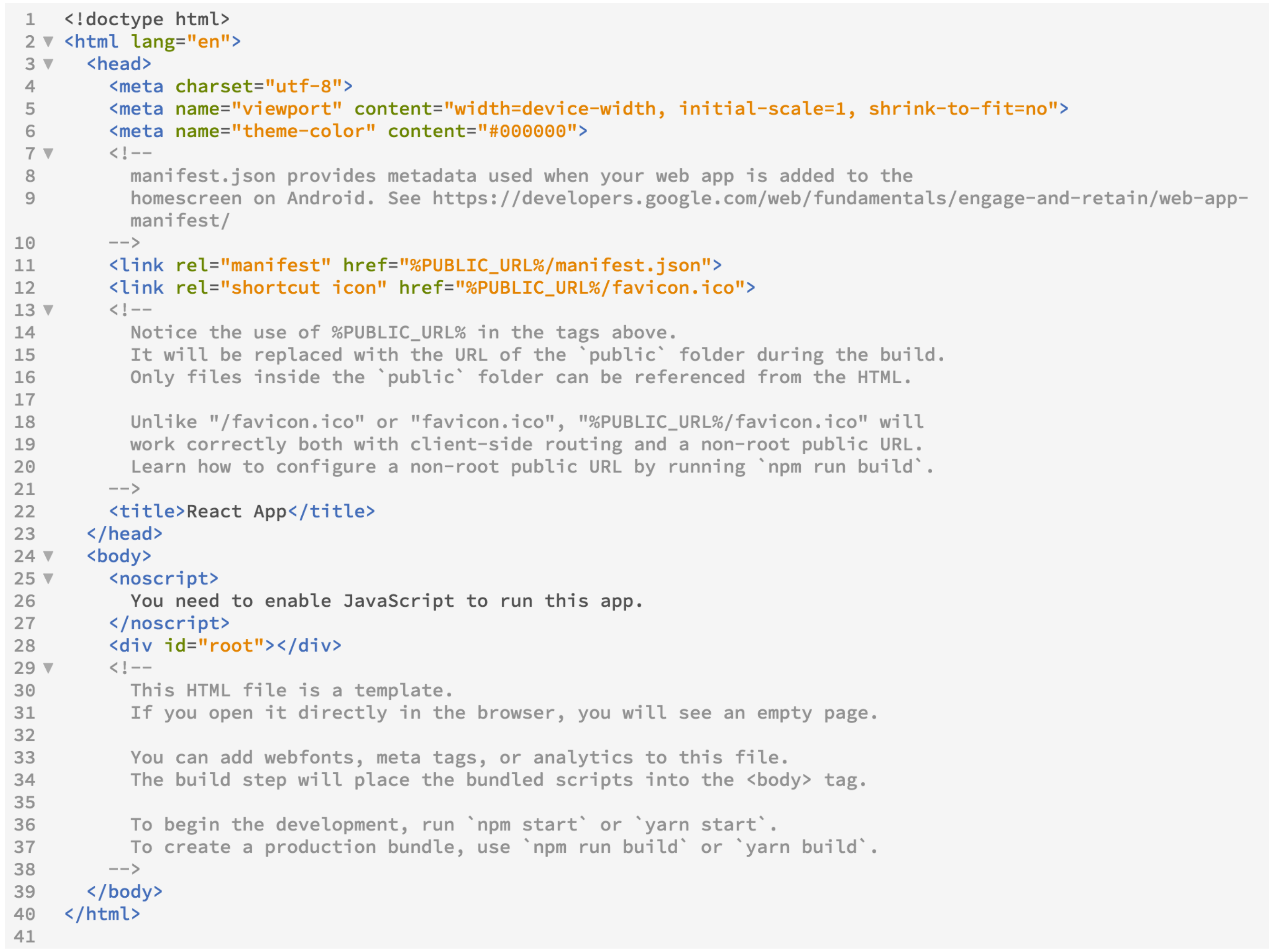


Welcome to React

To get started, edit `src/App.js` and save to reload.

Lets check some of those files and make some adjustments:

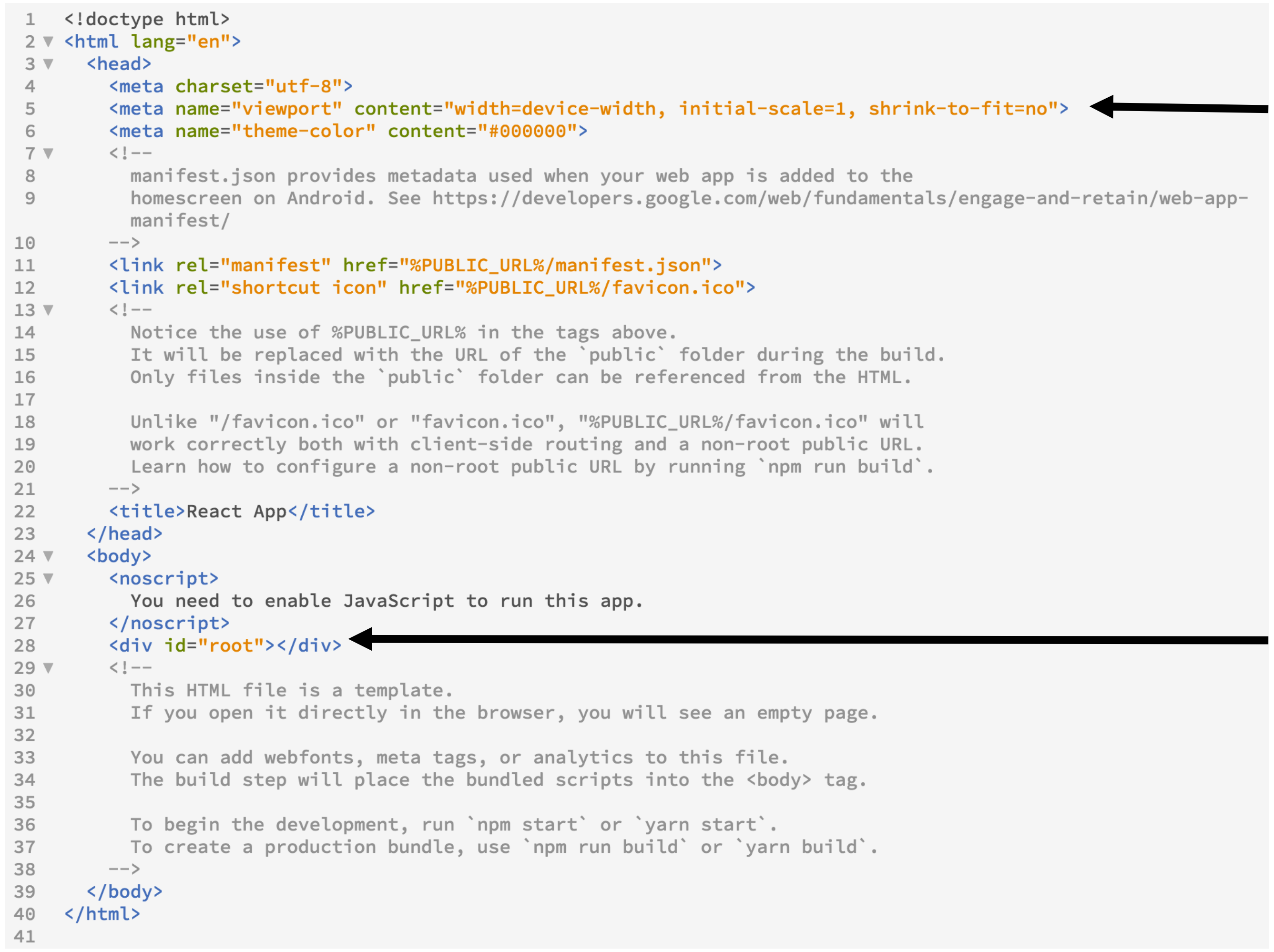
- index.html (in folder public)



```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <!--
      manifest.json provides metadata used when your web app is added to the
      homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>

```

Lets check some of those files and make some adjustments:

- index.html (in folder public)



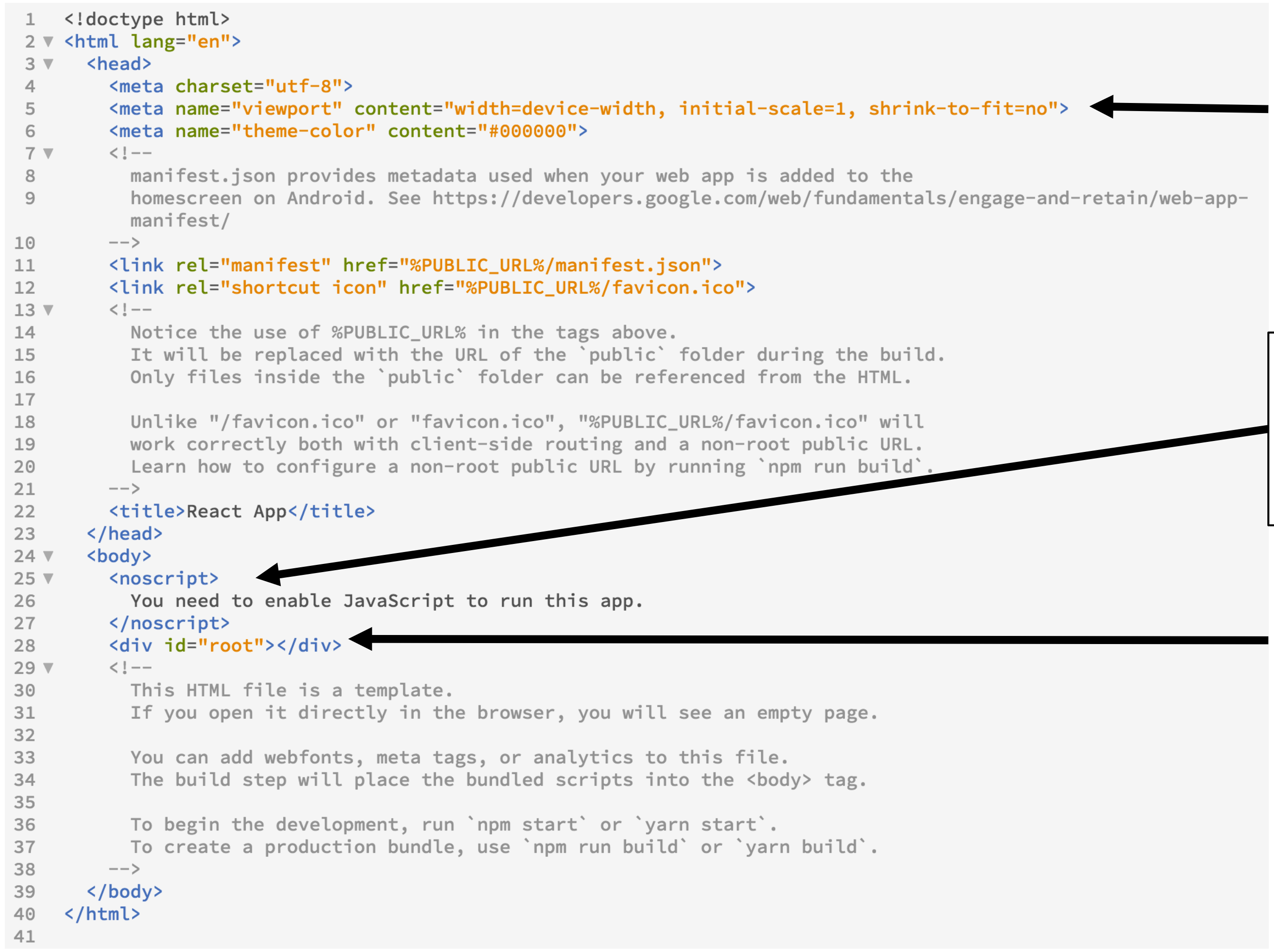
```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <!--
      manifest.json provides metadata used when your web app is added to the
      homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

Add missing <meta> elements to the head (we do not require scripts now, react is handling that)

Only relevant thing in the body, we have a div with id="root"

Lets check some of those files and make some adjustments:

- index.html (in folder public)



```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <!--
      manifest.json provides metadata used when your web app is added to the
      homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-retain/web-app-
      manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

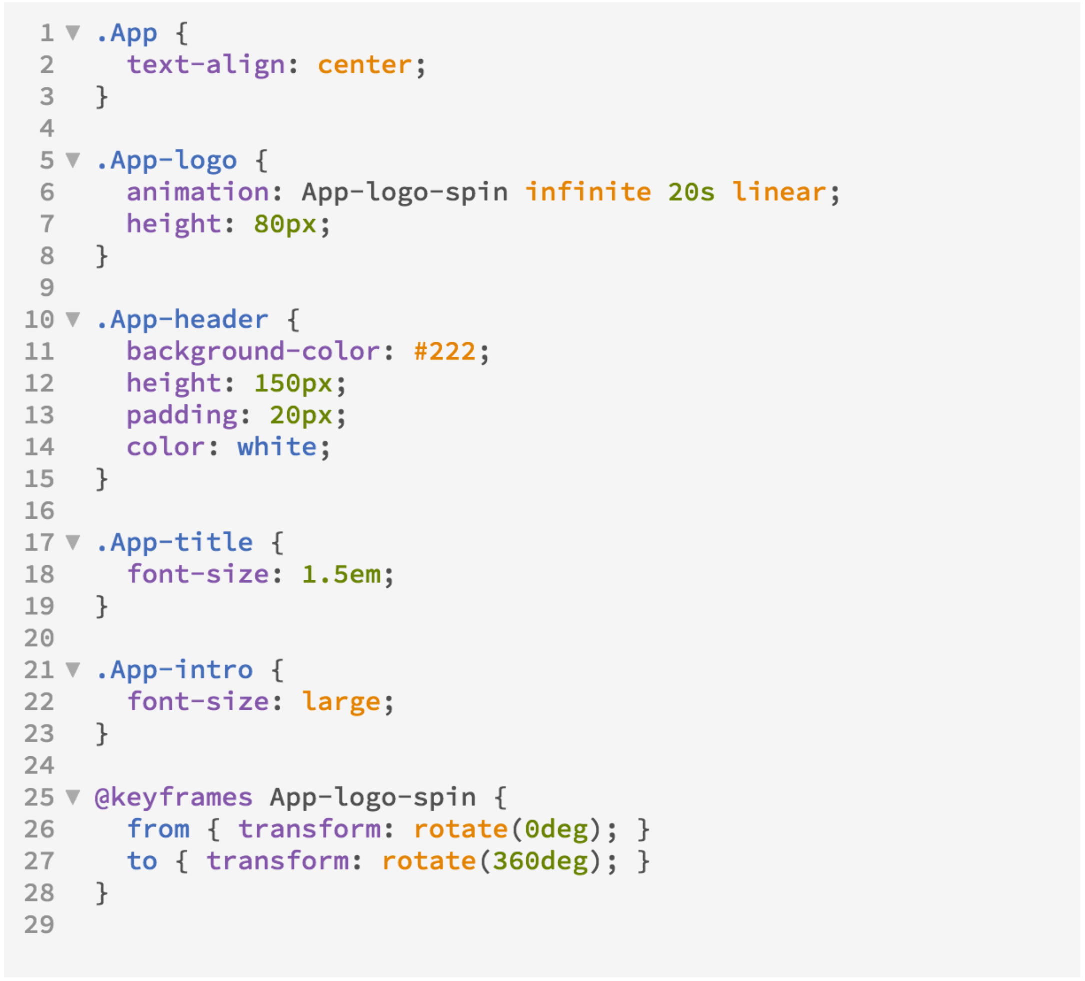
Add missing <meta> elements to the head (we do not require scripts now, react is handling that)

This is just something to deal with old browsers…

Only relevant thing in the body, we have a div with id="root"

Lets check some of those files and make some adjustments:

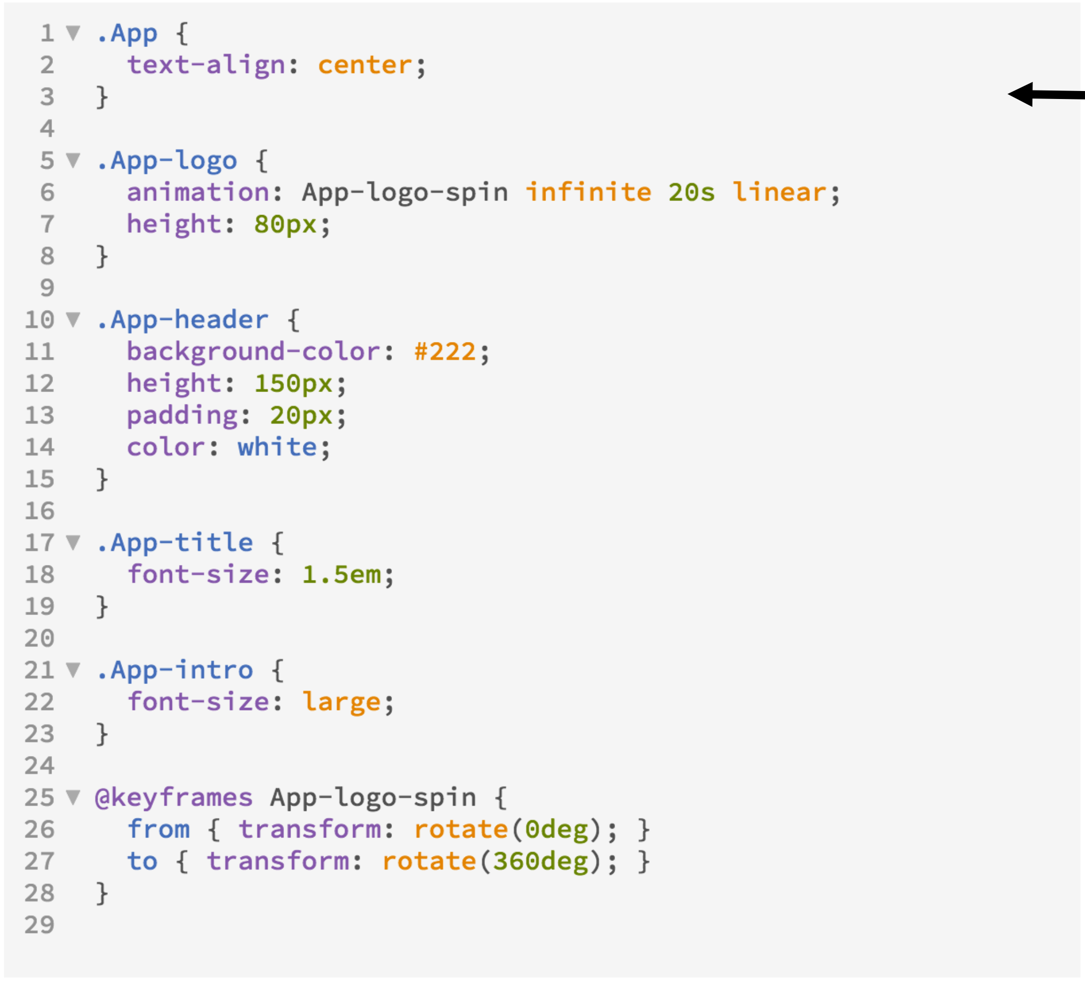
- App.css (in folder src)



```css
.App {
  text-align: center;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 80px;
}

.App-header {
  background-color: #222;
  height: 150px;
  padding: 20px;
  color: white;
}

.App-title {
  font-size: 1.5em;
}

.App-intro {
  font-size: large;
}

@keyframes App-logo-spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}

```

Lets check some of those files and make some adjustments:

- App.css (in folder src)



```
.App {
  text-align: center;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 80px;
}

.App-header {
  background-color: #222;
  height: 150px;
  padding: 20px;
  color: white;
}

.App-title {
  font-size: 1.5em;
}

.App-intro {
  font-size: large;
}

@keyframes App-logo-spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}
```

Replace this by the contents of the CSS we wrote in previous classes.

Lets check some of those files and make some adjustments:

- index.js (in folder src)

```javascript
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

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

```

Lets check some of those files and make some adjustments:

- index.js (in folder src)

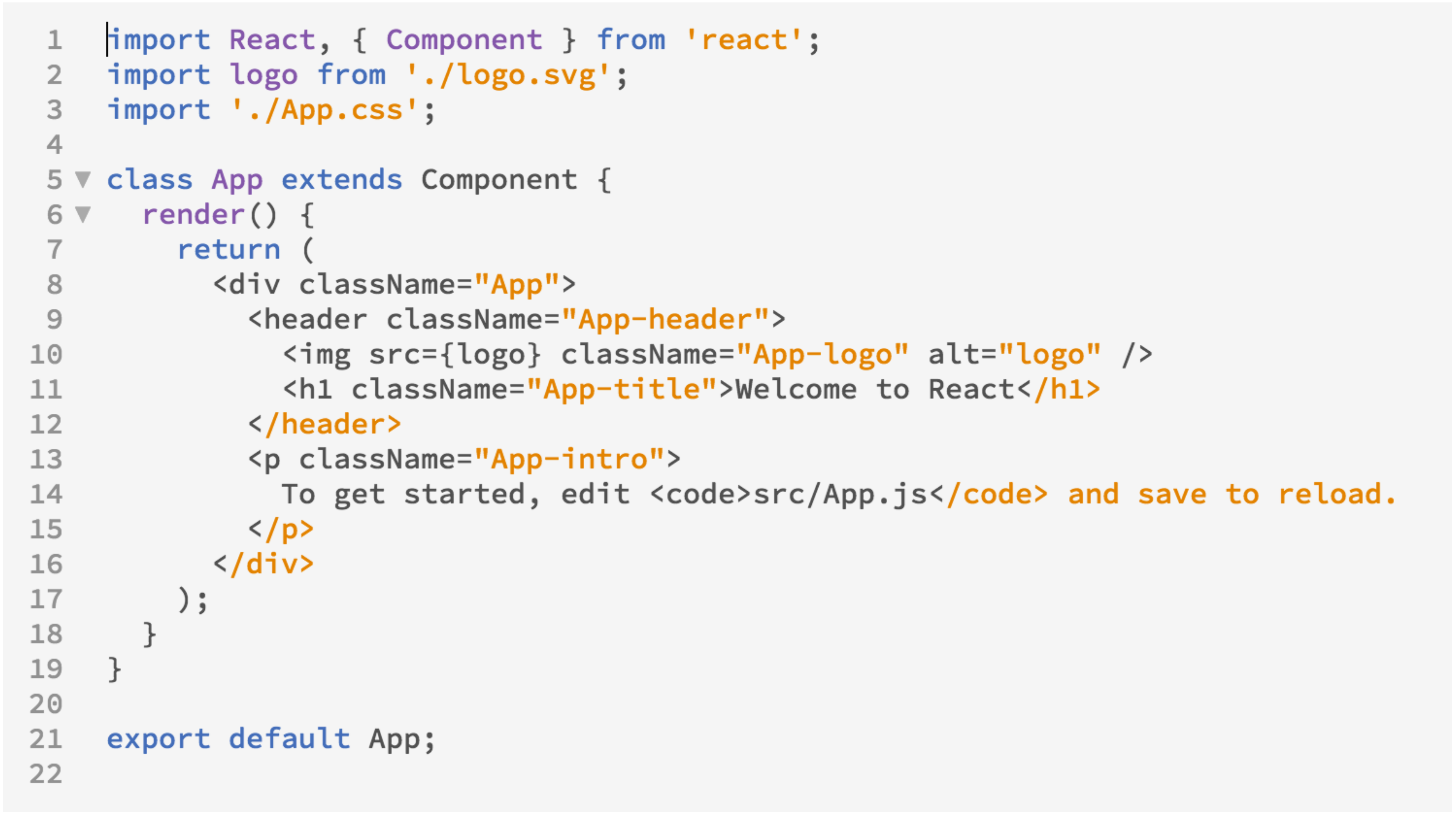
This is how you import:
React components;
Other files in your workspace;

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

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

This is attaching the top level component to your web page, inside de div with the id “root”

Lets check some of those files and make some adjustments:

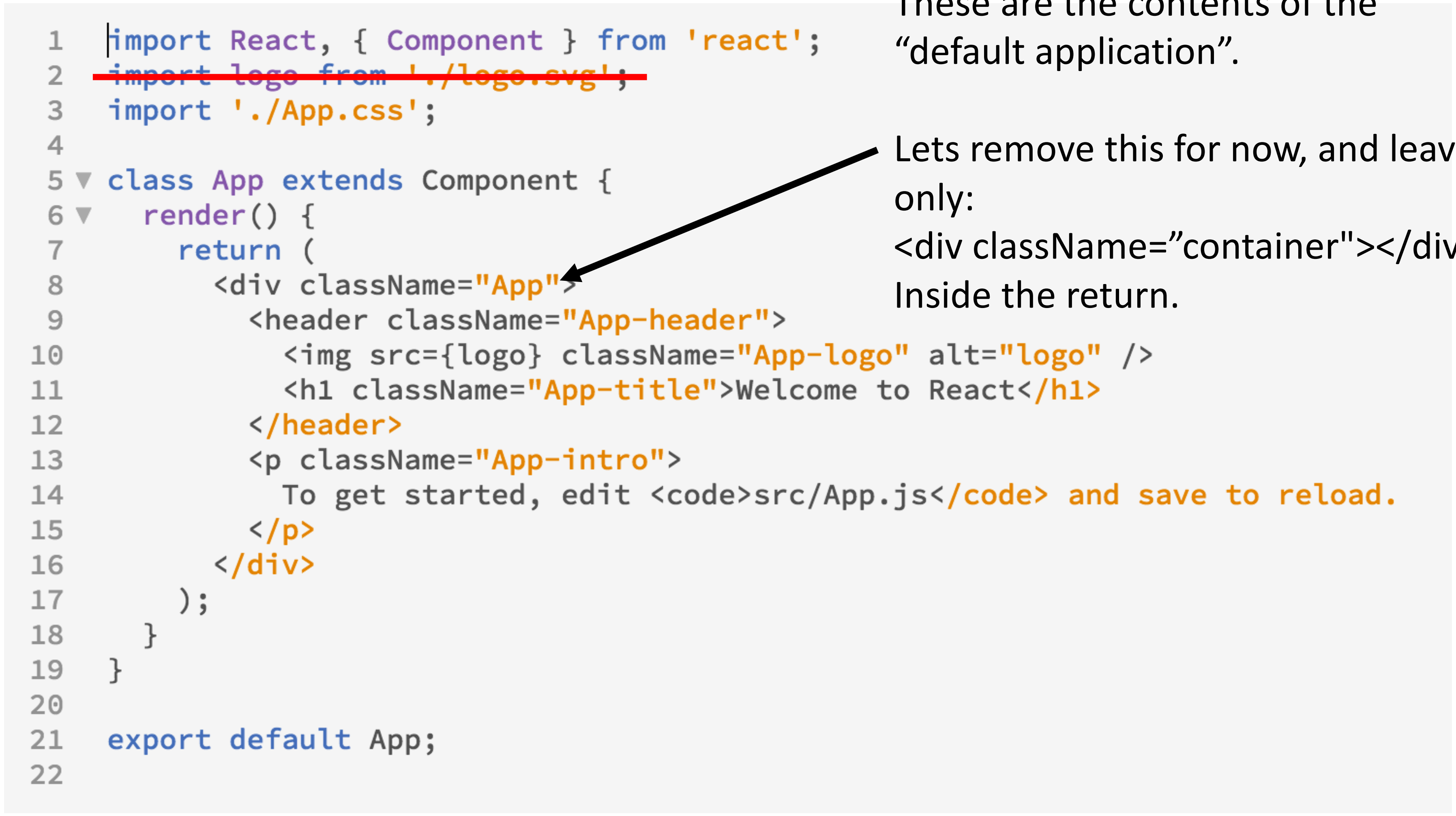
- App.js (in folder src)



```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;

```

Lets check some of those files and make some adjustments:

- App.js (in folder src)



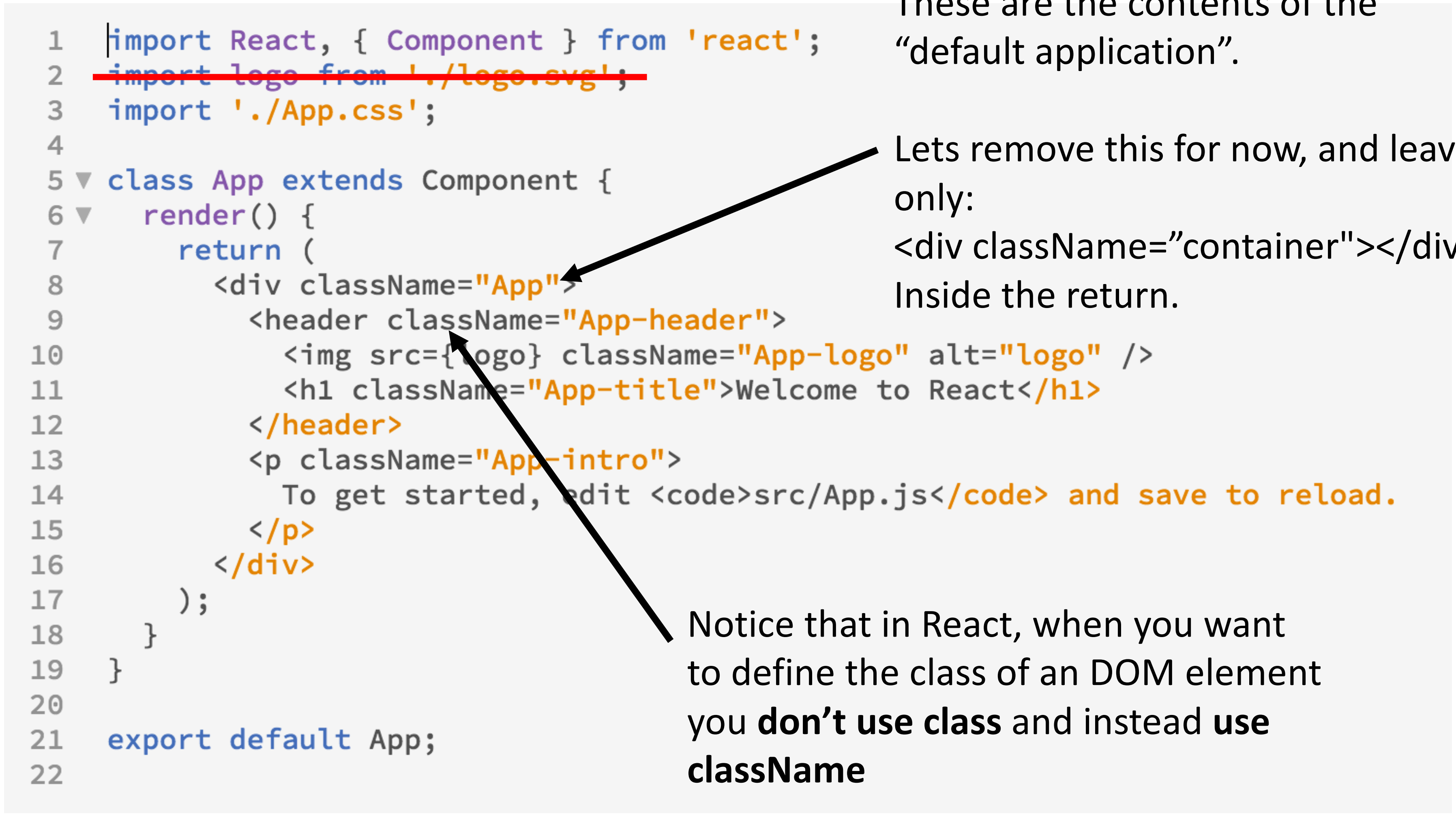
```
import React, { Component } from 'react';
~~import logo from './logo.svg';~~
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

These are the contents of the "default application".

Lets remove this for now, and leave only:
<div className="container"></div>
Inside the return.

Lets check some of those files and make some adjustments:

- App.js (in folder src)



```
import React, { Component } from 'react';
~~import logo from './logo.svg';~~
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

These are the contents of the "default application".

Lets remove this for now, and leave only:
<div className="container"></div>
Inside the return.

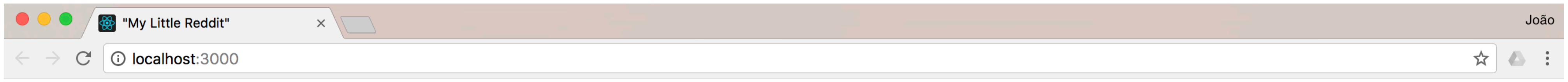
Notice that in React, when you want to define the class of an DOM element you **don't use class** and instead **use className**

Lets check some of those files and make some adjustments:

- Finally, add the folder *img* (the one containing images user in our application) to the public folder of your project (this will allow us to use those images).

After these adjustements…

- Your app should have refreshed in your browser:



Todays Exercise…

- Lets pick up the example from last weeks, and remake it as a React application:
  - Your HTML file will only have contents in the head and a single (emply) div with id="root" on the body, everything else will be done with React.
  - Let's decompose de application in components and write the code for each component (we will see that next).
  - We will keep the CSS file we did on our first iteration of this application and ensure that the style remains unchanged.

Todays Exercise…

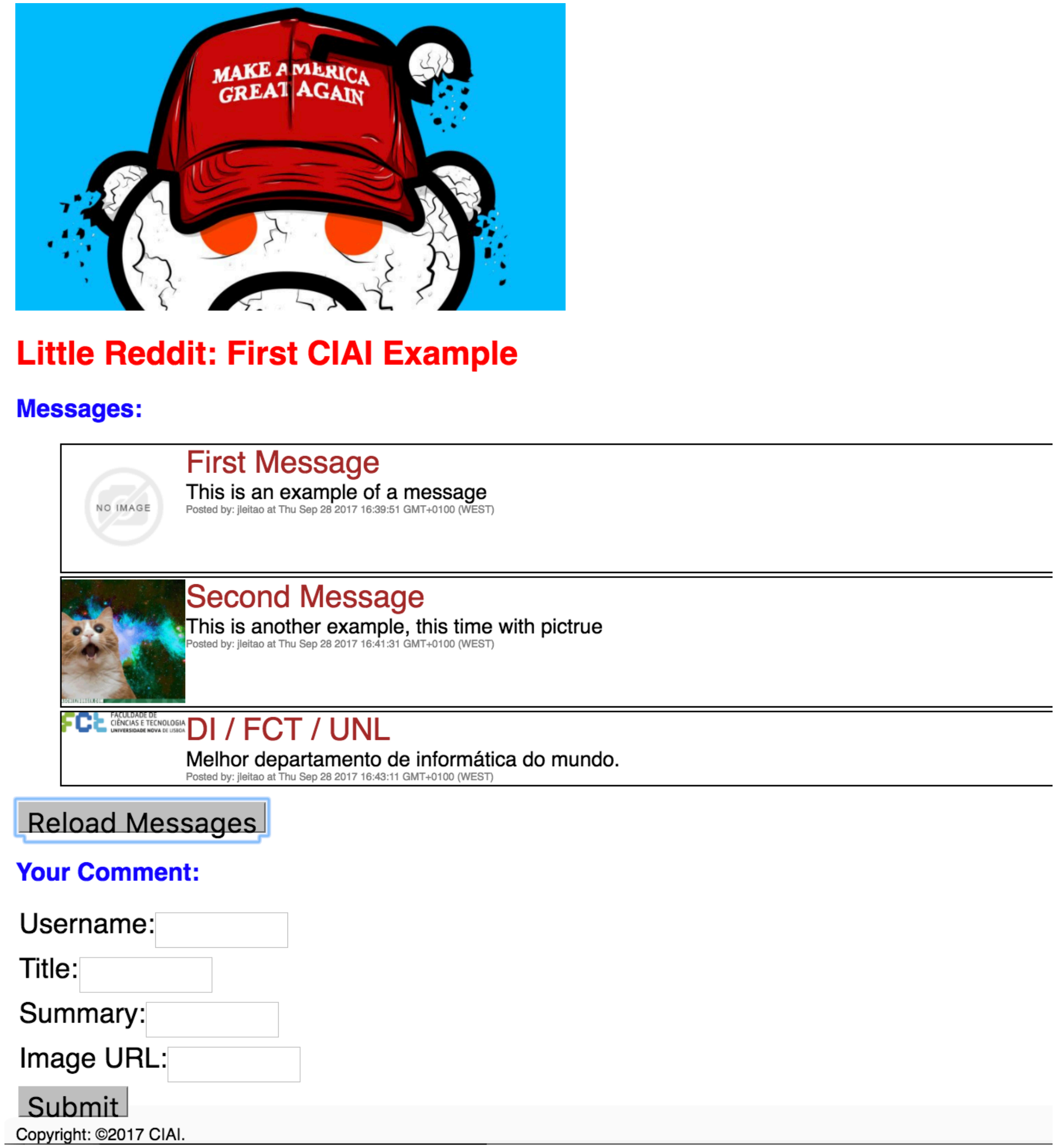
- Again you can use our service to get and post messages:
- Endpoint: http://ciai2017-180918.appspot.com/rest/
- Has two operations:
  - GET Operation: Provides you with a JSON array of message entries.
  - POST Operation: Receives a JSON object with the structure of a message and stores it.

Todays Exercise…

- The message JSON representation remains the same:

```
{ "title":"Second Message",
  "summary":"This is another example, this time with pictrue",
"imageURL":"https://memegenerator.net/img/images/600x600
/14834216/cat-on-drugs.jpg",
 "username":"jleitao",
"timestamp":1506418698824}
```

Starting Point:



Little Reddit: First CIAI Example

Messages:

First Message
This is an example of a message
Posted by: jleitao at Thu Sep 28 2017 16:39:51 GMT+0100 (WEST)

Second Message
This is another example, this time with pictrue
Posted by: jleitao at Thu Sep 28 2017 16:41:31 GMT+0100 (WEST)

DI / FCT / UNL
Melhor departamento de informática do mundo.
Posted by: jleitao at Thu Sep 28 2017 16:43:11 GMT+0100 (WEST)

Reload Messages

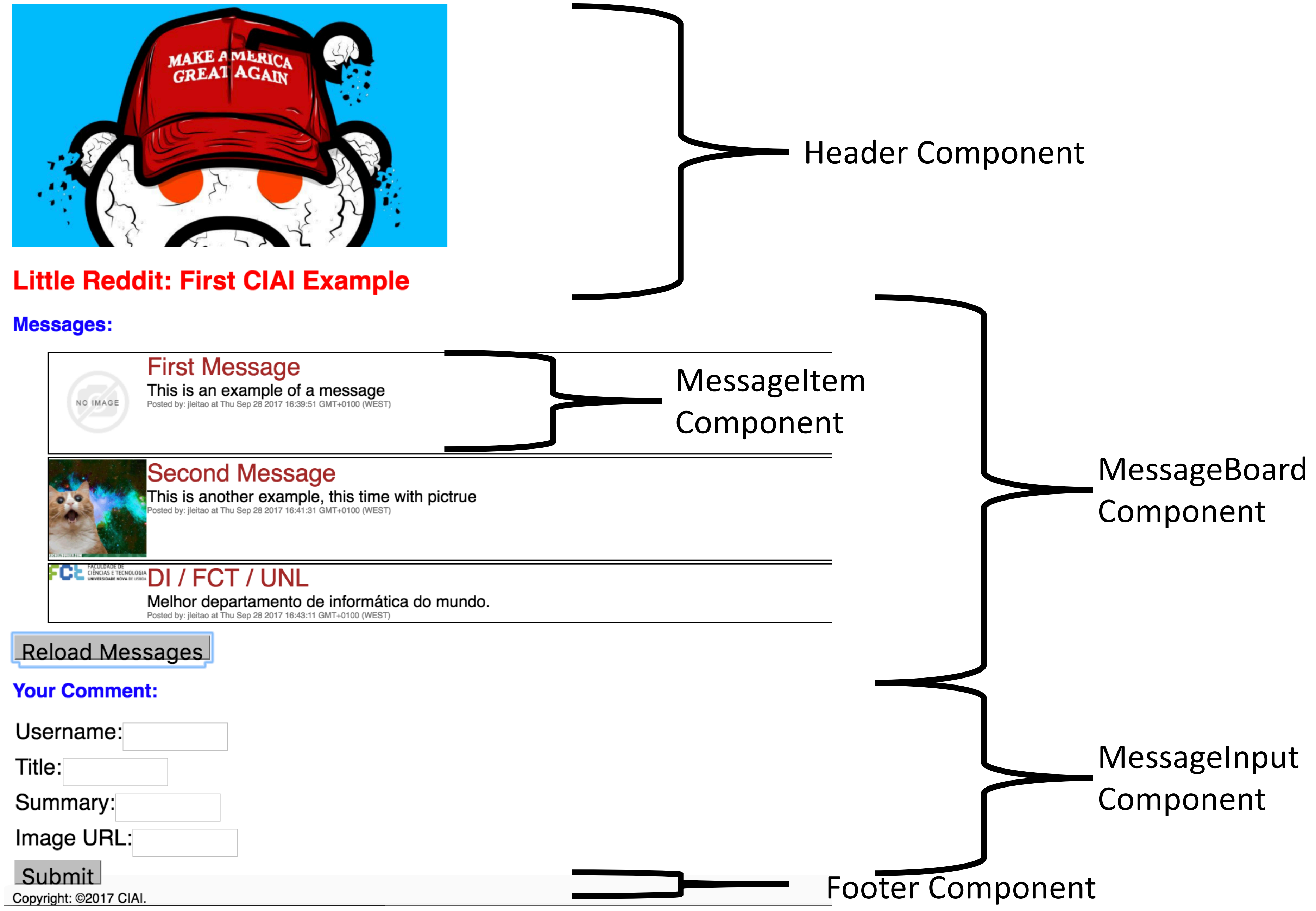
Your Comment:

Username:
Title:
Summary:
Image URL:

Submit

Copyright: ©2017 CIAI.

Starting Point:



Little Reddit: First CIAI Example

Messages:

First Message
This is an example of a message
Posted by: jleitao at Thu Sep 28 2017 16:39:51 GMT+0100 (WEST)

Second Message
This is another example, this time with pictrue
Posted by: jleitao at Thu Sep 28 2017 16:41:31 GMT+0100 (WEST)

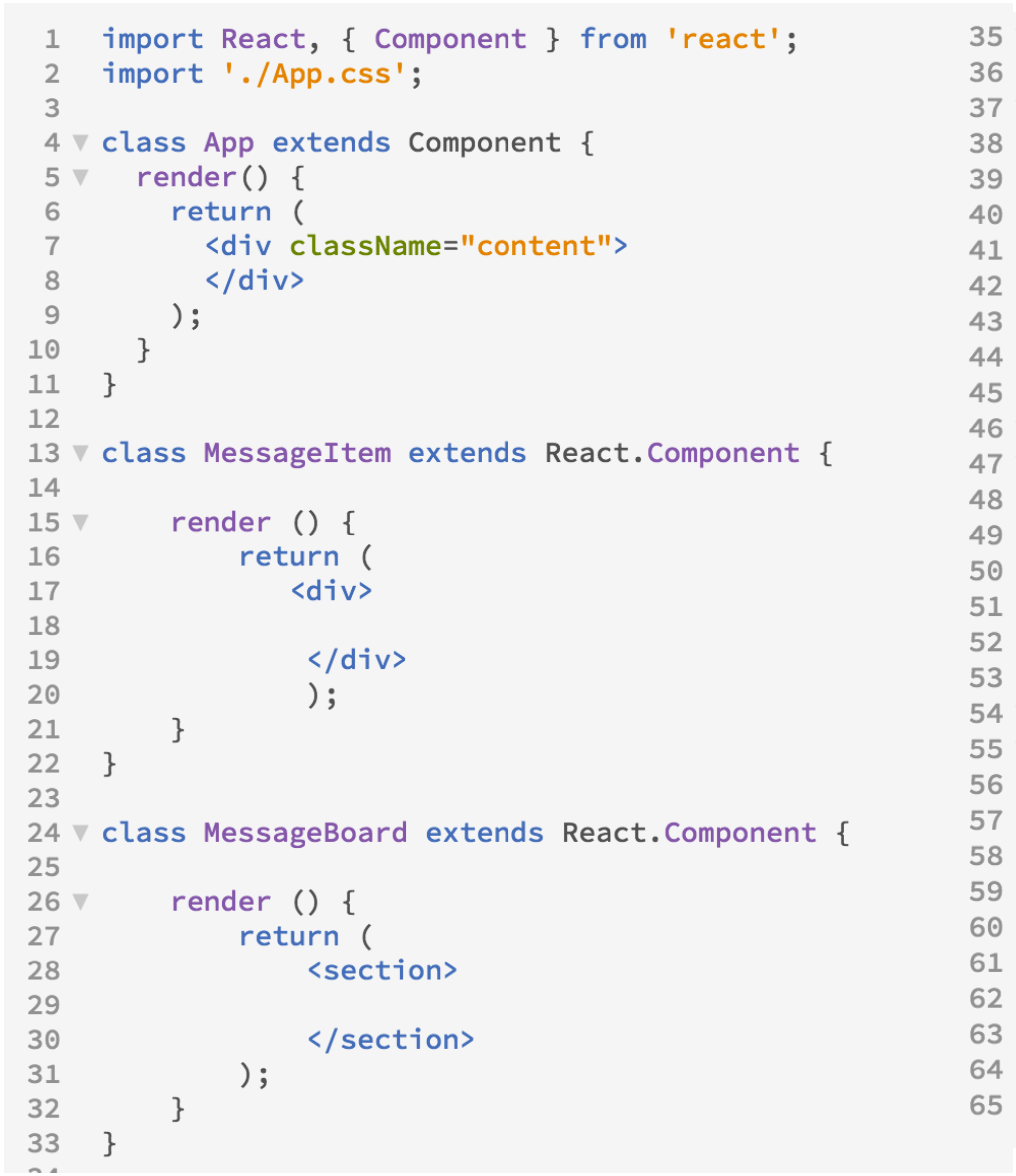
DI / FCT / UNL
Melhor departamento de informática do mundo.
Posted by: jleitao at Thu Sep 28 2017 16:43:11 GMT+0100 (WEST)

Your Comment:

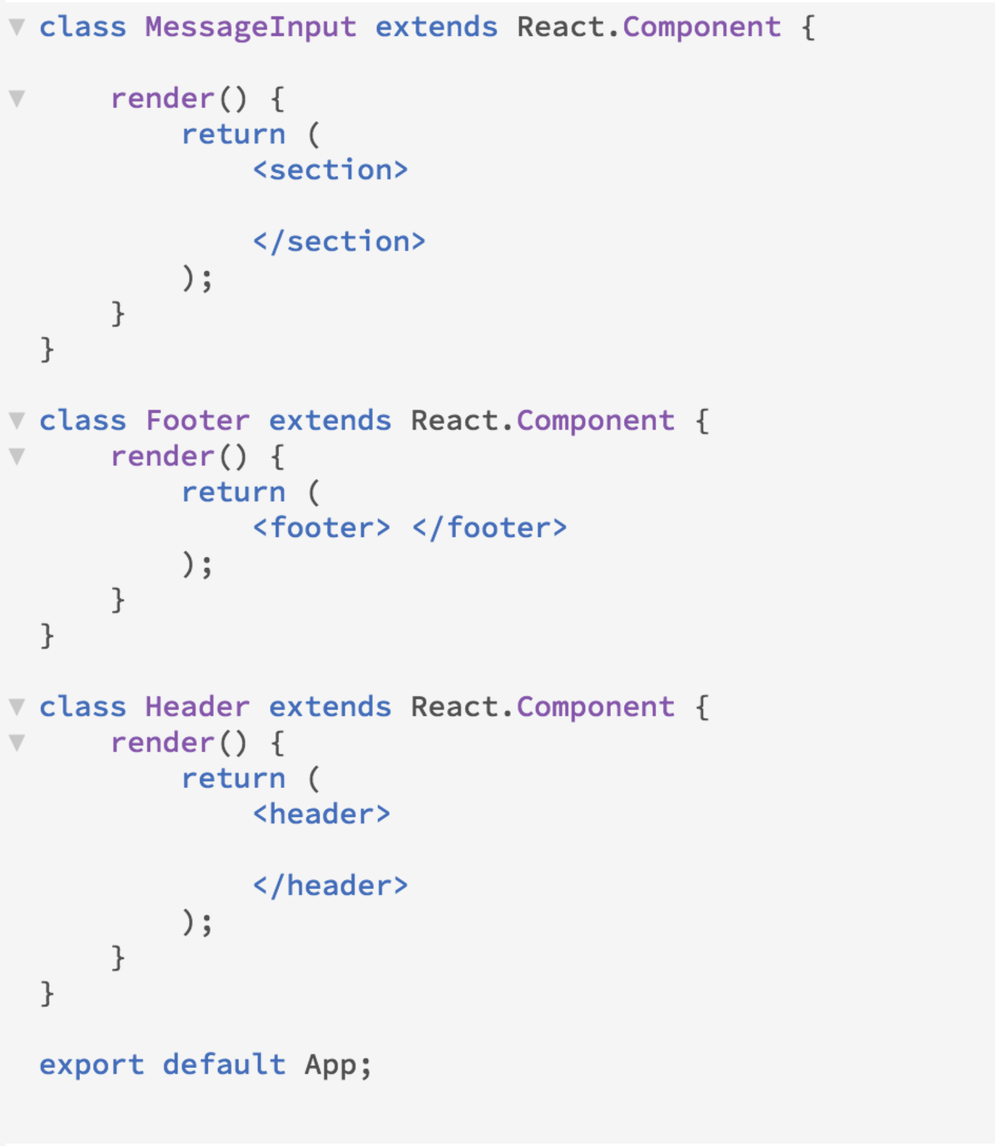
Username:
Title:
Summary:
Image URL:

Copyright: ©2017 CIAI.

Translation of the Starting Point:



```
import React, { Component } from 'react';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="content">
      </div>
    );
  }
}

class MessageItem extends React.Component {

    render () {
        return (
           <div>

            </div>
            );
    }
}

class MessageBoard extends React.Component {

    render () {
        return (
            <section>

            </section>
        );
    }
}
```



```
class MessageInput extends React.Component {

    render() {
        return (
            <section>

            </section>
        );
    }
}

class Footer extends React.Component {
    render() {
        return (
            <footer> </footer>
        );
    }
}

class Header extends React.Component {
    render() {
        return (
            <header>

            </header>
        );
    }
}

export default App;
```

Make the Application a Reality

- Start with simple things:
  - Header.
  - Footer.
  - Make both render in your application.
- Move to more complex things.
  - Make the messages appear (loaded from the server).
  - Make the post action work.
  - Make sure that when the page loads, messages are immediatly fetched.
  - Make sure that after a successful post you reload the messages again.
  - Hint: For AJAX requests, use jquery. Simply import it in your App.js by adding in the top: **import $ from 'jquery';**

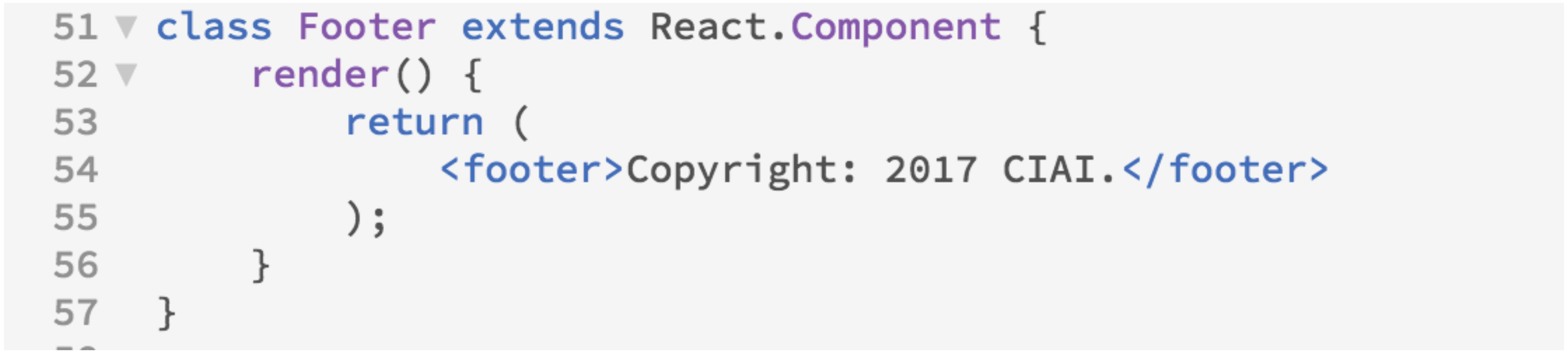
Make the Application a Reality

- **Start with simple things:**
  - **Header.**
  - **Footer.**
  - **Make both render in your application.**
- Move to more complex things.
  - Make the messages appear (loaded from the server).
  - Make the post action work.
  - Make sure that when the page loads, messages are immediatly fetched.
  - Make sure that after a successful post you reload the messages again.
  - Hint: For AJAX requests, use jquery. Simply import it in your App.js by adding in the top: **import $ from 'jquery';**

Solution: Header

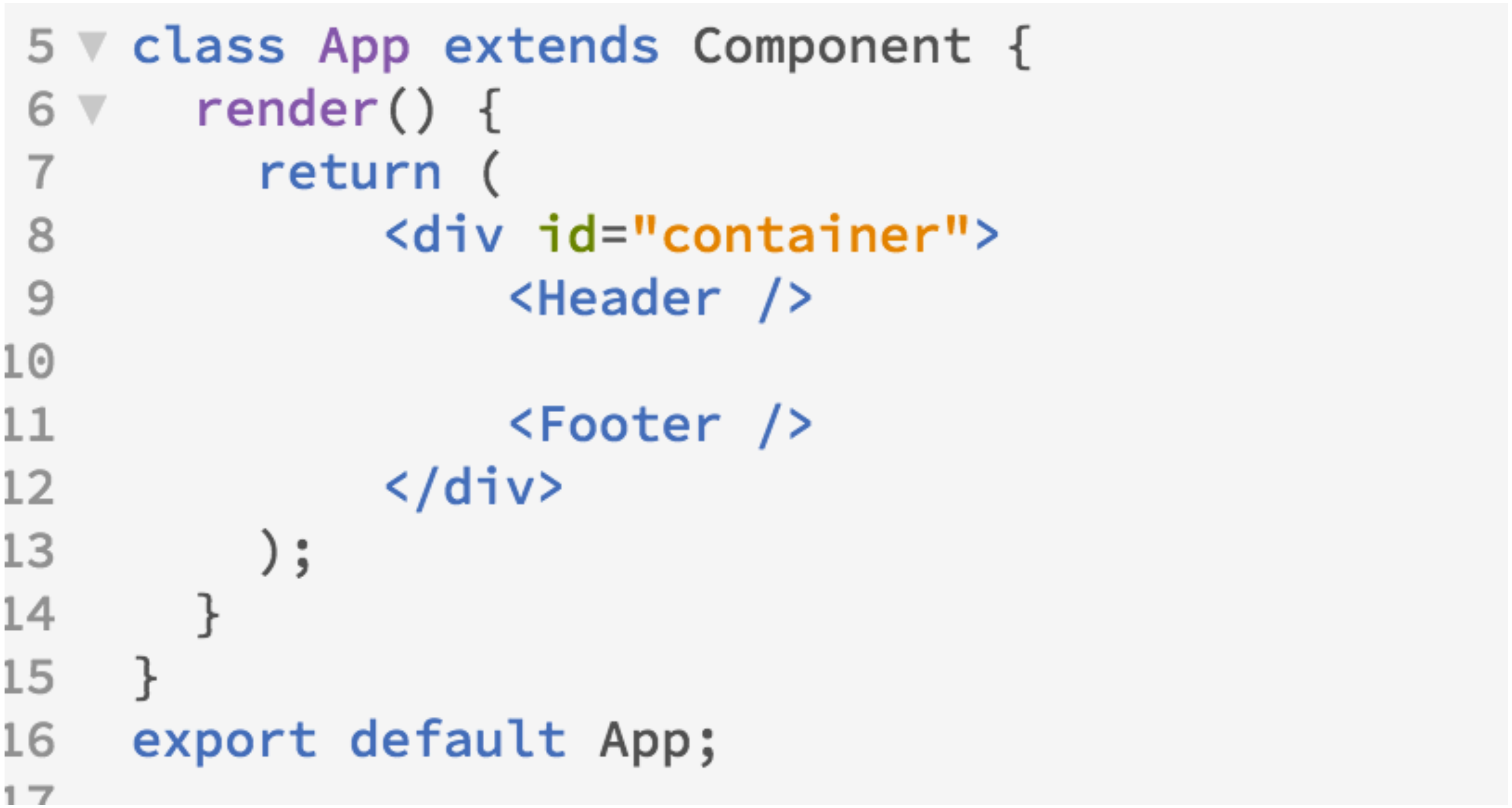
```
class Header extends React.Component {
    render() {
        return (
            <header>
                <div className="banner">
                    <img src="img/little-reddit.jpg" alt="MyLittleReddit" />
                    <h1>Little Reddit: First CIAI Example</h1>
                </div>
            </header>
        );
    }
}
```

Solution: Footer



```
class Footer extends React.Component {
    render() {
        return (
            <footer>Copyright: 2017 CIAI.</footer>
        );
    }
}
```

Solution: Rendering these components.

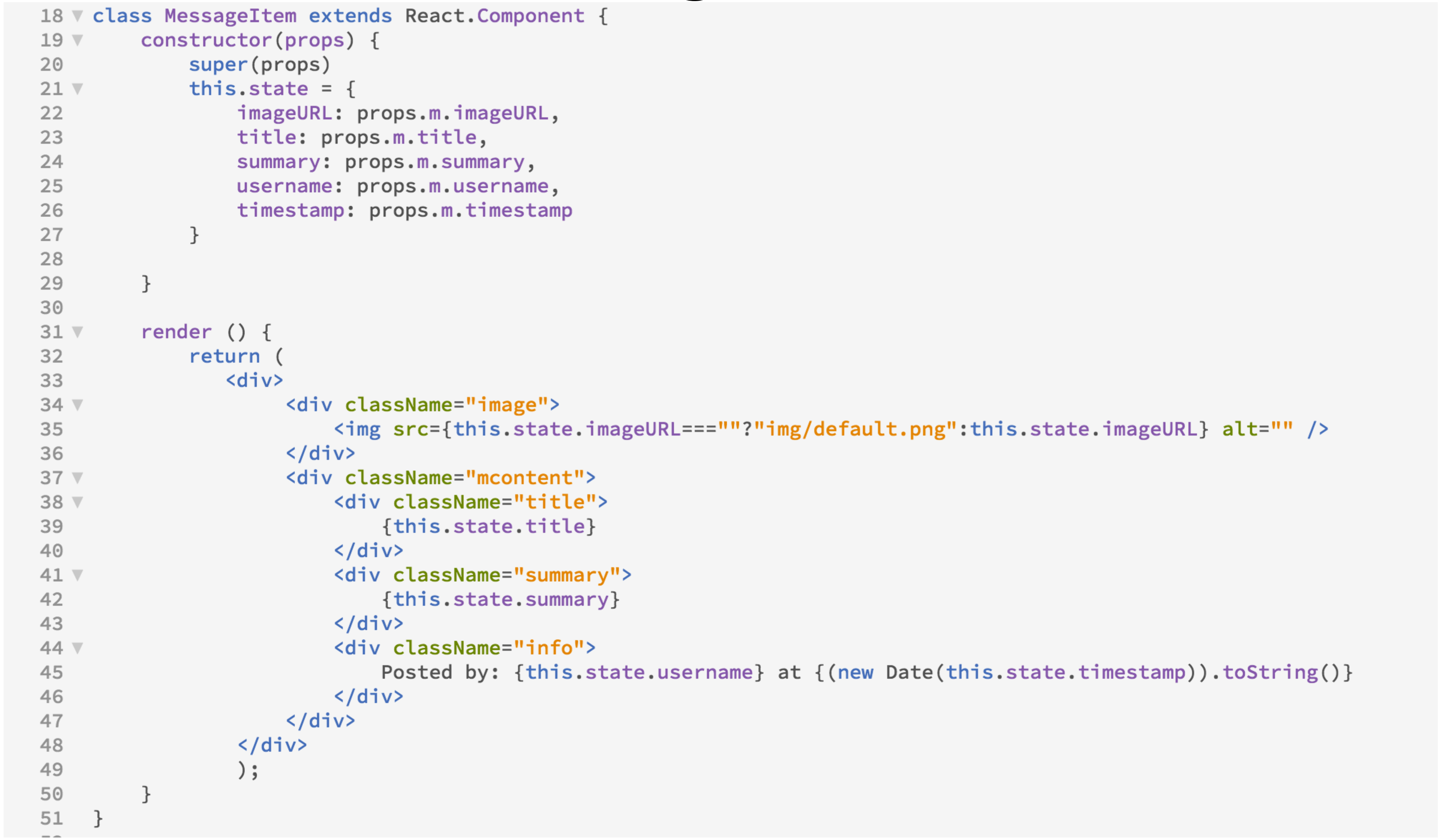


```
class App extends Component {
  render() {
    return (
        <div id="container">
            <Header />

            <Footer />
        </div>
    );
  }
}
export default App;
```

Make the Application a Reality

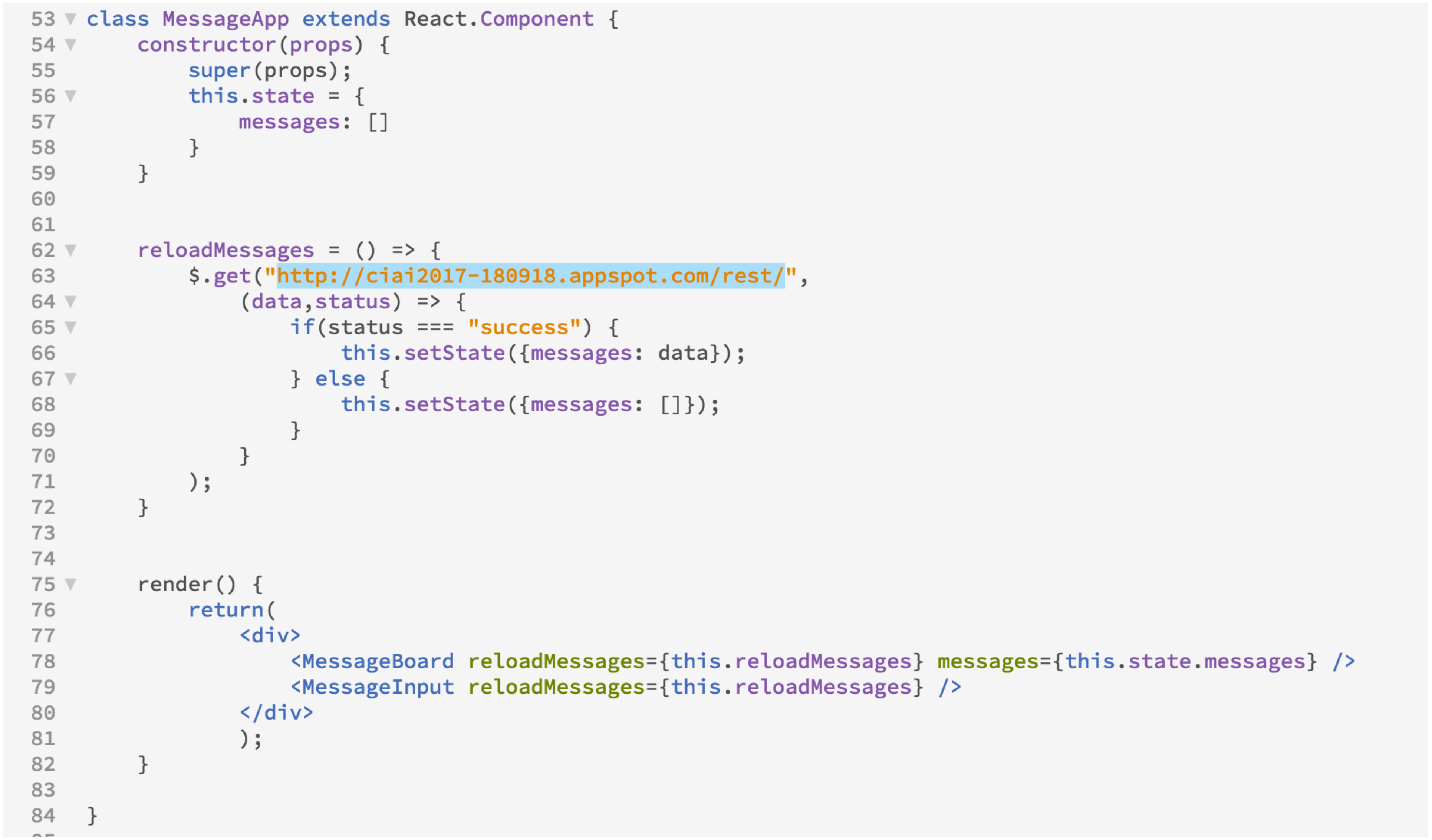
- Start with simple things:
  - Header.
  - Footer.
  - Make both render in your application.
- **Move to more complex things.**
  - **Make the messages appear (loaded from the server).**
  - **Make the post action work.**
  - **Make sure that when the page loads, messages are immediatly fetched.**
  - **Make sure that after a successful post you reload the messages again.**
  - Hint: For AJAX requests, use jquery. Simply import it in your App.js by adding in the top: **import $ from 'jquery';**

Solution: Message Item



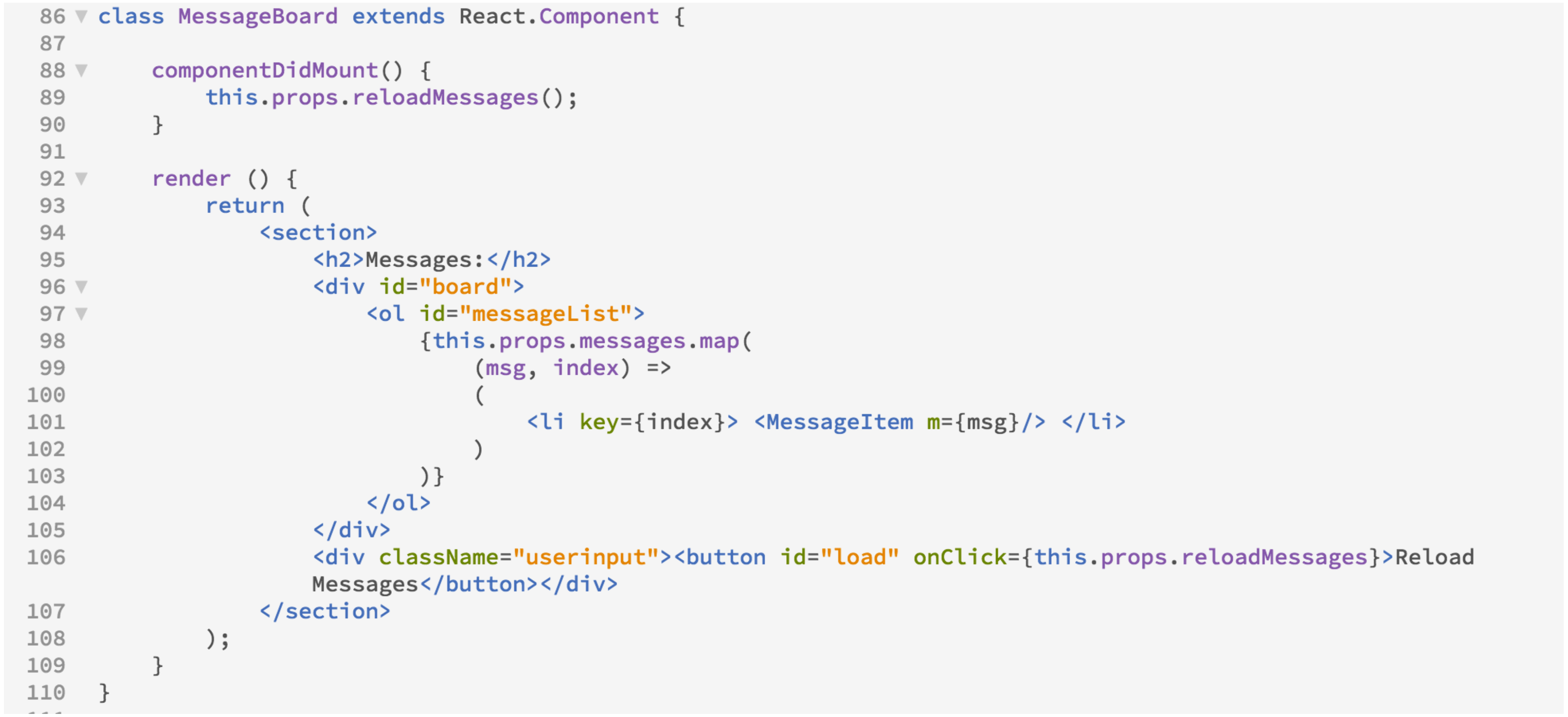
```
class MessageItem extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            imageURL: props.m.imageURL,
            title: props.m.title,
            summary: props.m.summary,
            username: props.m.username,
            timestamp: props.m.timestamp
        }

    }

    render () {
        return (
            <div>
                <div className="image">
                    <img src={this.state.imageURL===""?"img/default.png":this.state.imageURL} alt="" />
                </div>
                <div className="mcontent">
                    <div className="title">
                        {this.state.title}
                    </div>
                    <div className="summary">
                        {this.state.summary}
                    </div>
                    <div className="info">
                        Posted by: {this.state.username} at {(new Date(this.state.timestamp)).toString()}
                    </div>
                </div>
            </div>
            );
    }
}
```

Solution MessageApp (container for MessageBoard and MessageInput)



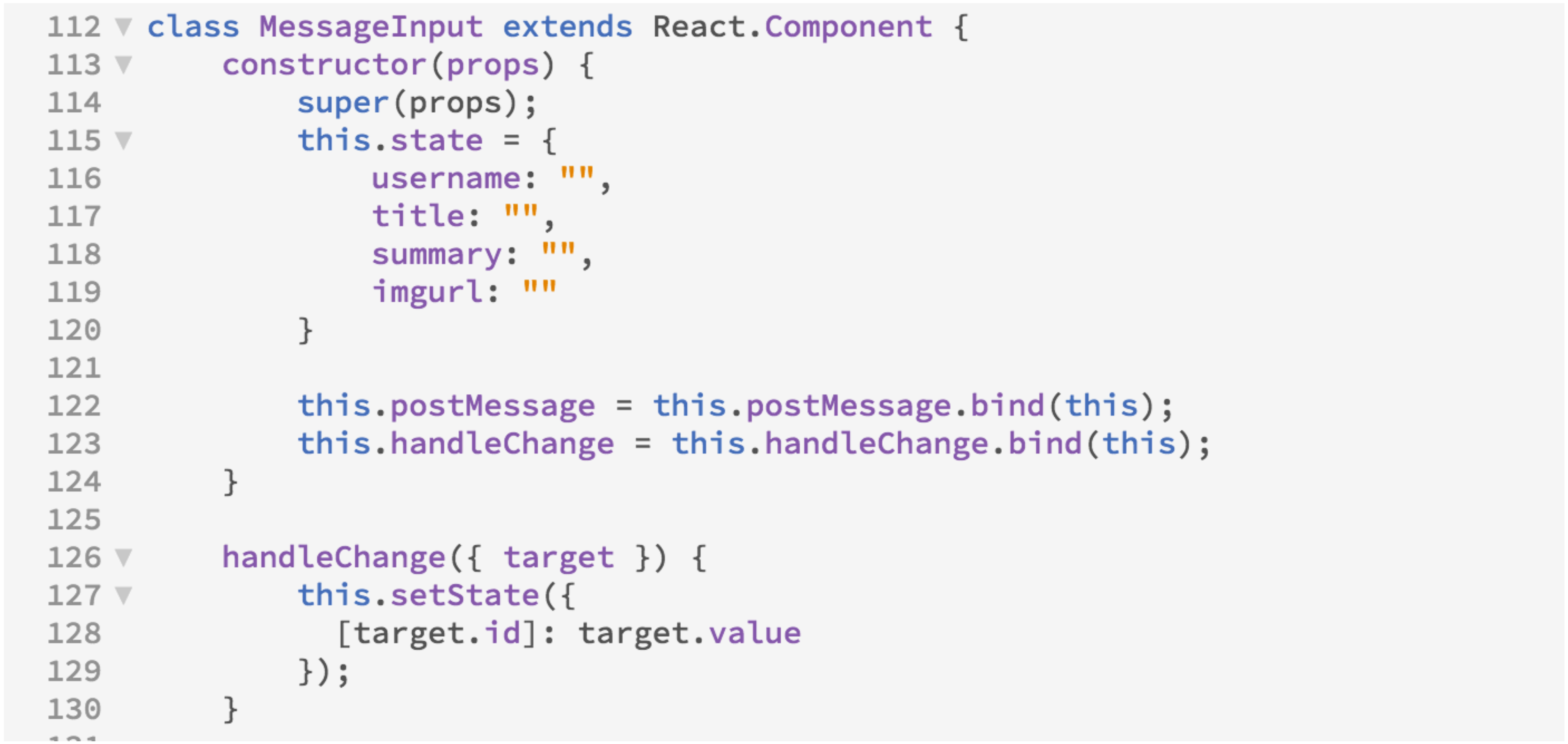
```
class MessageApp extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            messages: []
        }
    }


    reloadMessages = () => {
        $.get("http://ciai2017-180918.appspot.com/rest/",
            (data,status) => {
                if(status === "success") {
                    this.setState({messages: data});
                } else {
                    this.setState({messages: []});
                }
            }
        );
    }


    render() {
        return(
            <div>
                <MessageBoard reloadMessages={this.reloadMessages} messages={this.state.messages} />
                <MessageInput reloadMessages={this.reloadMessages} />
            </div>
            );
    }

}
```

Solution: MessageBoard



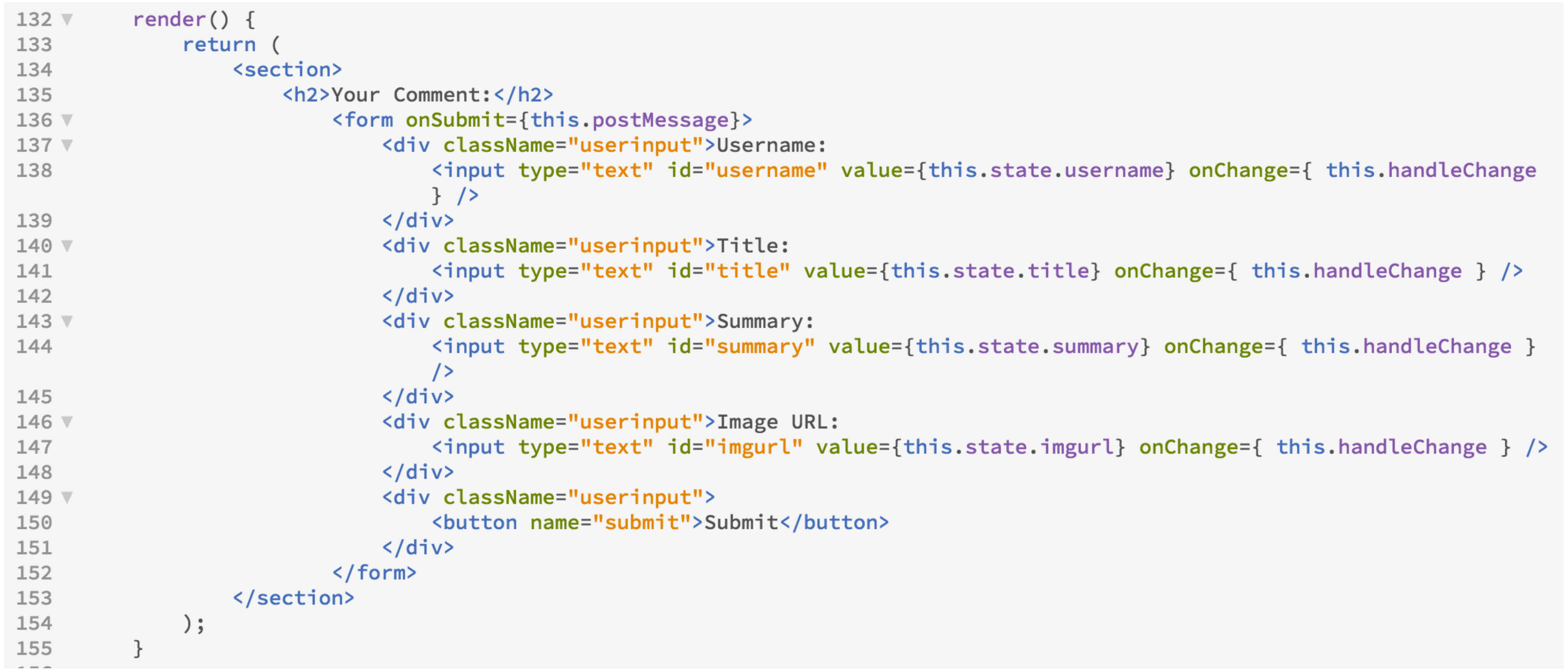
```
class MessageBoard extends React.Component {

    componentDidMount() {
        this.props.reloadMessages();
    }

    render () {
        return (
            <section>
                <h2>Messages:</h2>
                <div id="board">
                    <ol id="messageList">
                        {this.props.messages.map(
                            (msg, index) =>
                            (
                                <li key={index}> <MessageItem m={msg}/> </li>
                            )
                        )}
                    </ol>
                </div>
                <div className="userinput"><button id="load" onClick={this.props.reloadMessages}>Reload Messages</button></div>
            </section>
        );
    }
}
```

Solution Message Input (1/3)



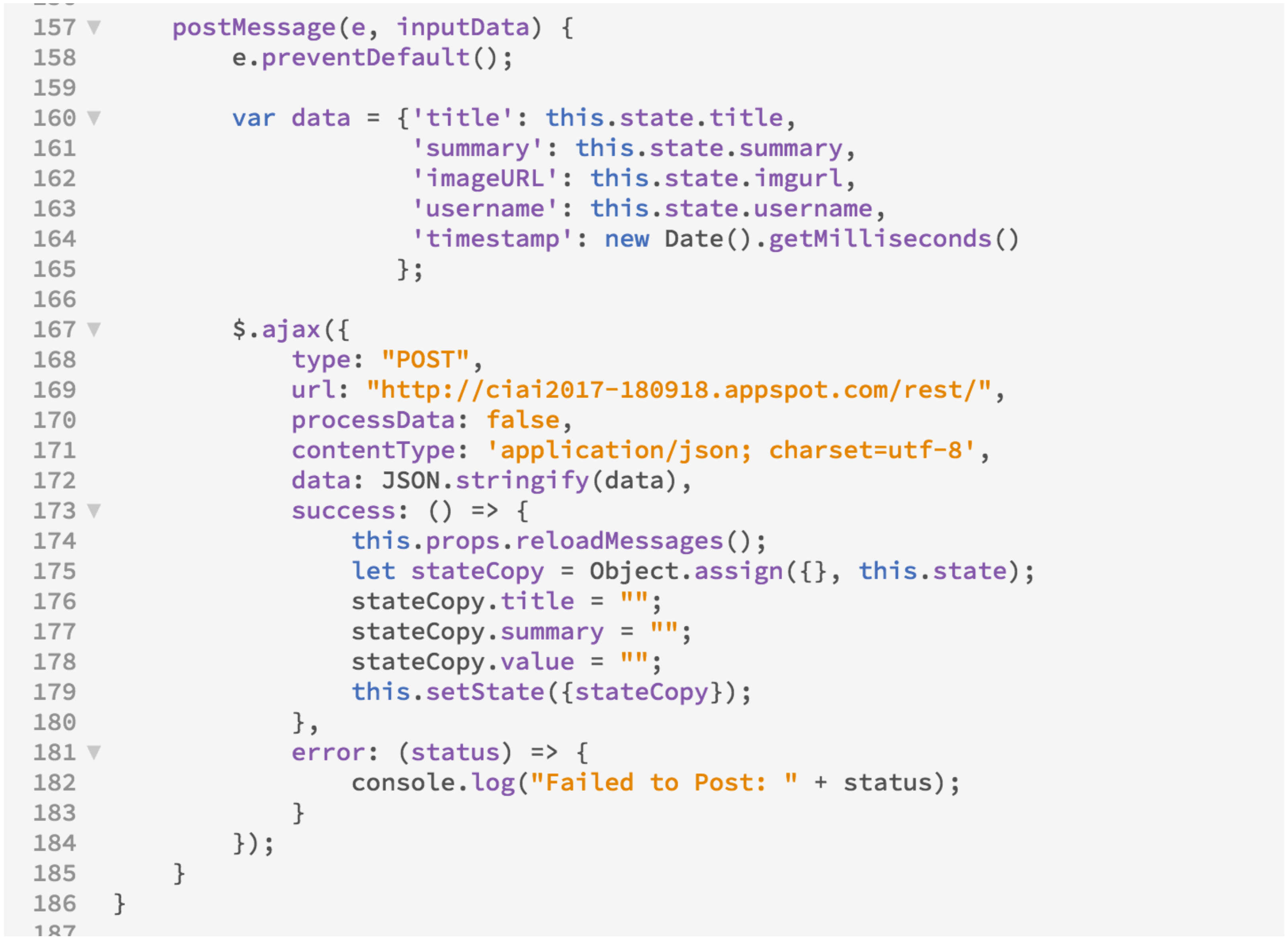
```
class MessageInput extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            username: "",
            title: "",
            summary: "",
            imgurl: ""
        }

        this.postMessage = this.postMessage.bind(this);
        this.handleChange = this.handleChange.bind(this);
    }

    handleChange({ target }) {
        this.setState({
          [target.id]: target.value
        });
    }
```

Solution Message Input (2/3)



```
    render() {
        return (
            <section>
                <h2>Your Comment:</h2>
                <form onSubmit={this.postMessage}>
                    <div className="userinput">Username:
                        <input type="text" id="username" value={this.state.username} onChange={ this.handleChange
                        } />
                    </div>
                    <div className="userinput">Title:
                        <input type="text" id="title" value={this.state.title} onChange={ this.handleChange } />
                    </div>
                    <div className="userinput">Summary:
                        <input type="text" id="summary" value={this.state.summary} onChange={ this.handleChange }
                        />
                    </div>
                    <div className="userinput">Image URL:
                        <input type="text" id="imgurl" value={this.state.imgurl} onChange={ this.handleChange } />
                    </div>
                    <div className="userinput">
                        <button name="submit">Submit</button>
                    </div>
                </form>
            </section>
        );
    }
```

Solution Message Input (3/3)



```
    postMessage(e, inputData) {
        e.preventDefault();

        var data = {'title': this.state.title,
                    'summary': this.state.summary,
                    'imageURL': this.state.imgurl,
                    'username': this.state.username,
                    'timestamp': new Date().getMilliseconds()
                   };

        $.ajax({
            type: "POST",
            url: "http://ciai2017-180918.appspot.com/rest/",
            processData: false,
            contentType: 'application/json; charset=utf-8',
            data: JSON.stringify(data),
            success: () => {
                this.props.reloadMessages();
                let stateCopy = Object.assign({}, this.state);
                stateCopy.title = "";
                stateCopy.summary = "";
                stateCopy.value = "";
                this.setState({stateCopy});
            },
            error: (status) => {
                console.log("Failed to Post: " + status);
            }
        });
    }
}
```