



Professional Certificate in Coding: Full Stack Development with MERN: Week 20

Testing

Install Create React App (1/14)

```
● ● ● ~/MIT/Courses/React — johnwilliams — npm - node /usr/local/bin/npx create-react-app cartsln01 — 93x27
[✓ React % npx create-react-app cartsln01
((.....)) : fetchMetadata: sill resolveWithNewModule ansi-regex@4.1.0 chec
```

Install Create React App (2/14)

```
React % npx create-react-app cartsln01
npx: installed 98 in 5.042s

Creating a new React app in /Users/johnwilliams/MIT/Courses/React/cartsln01.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

(( )) : fetchMetadata: sill resolveWithNewModule @babel/helper-plugin-...
```

Install Create React App (3/14)

```
Success! Created cartsoln01 at /Users/johnwilliams/MIT/Courses/React/cartsoln01
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

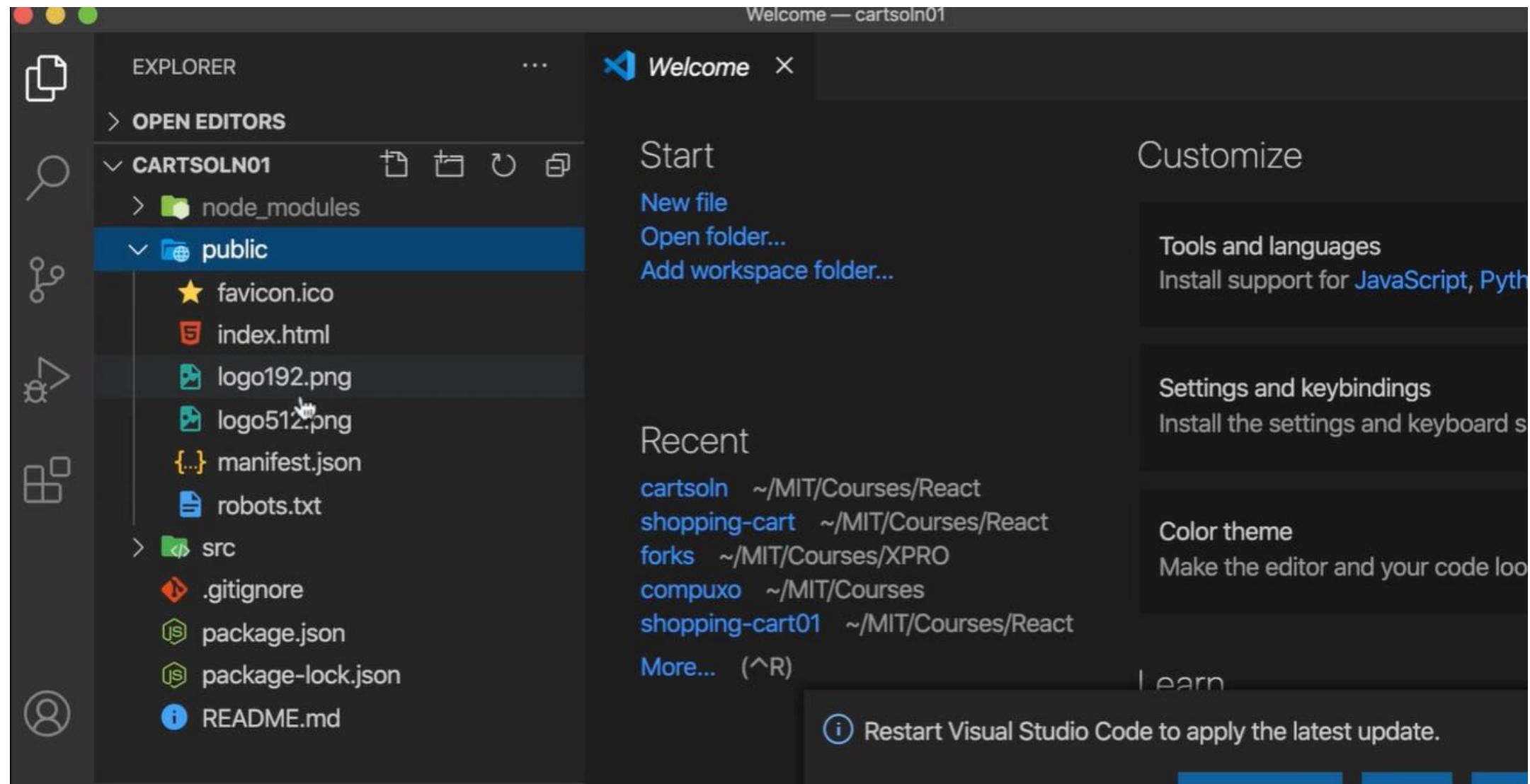
  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

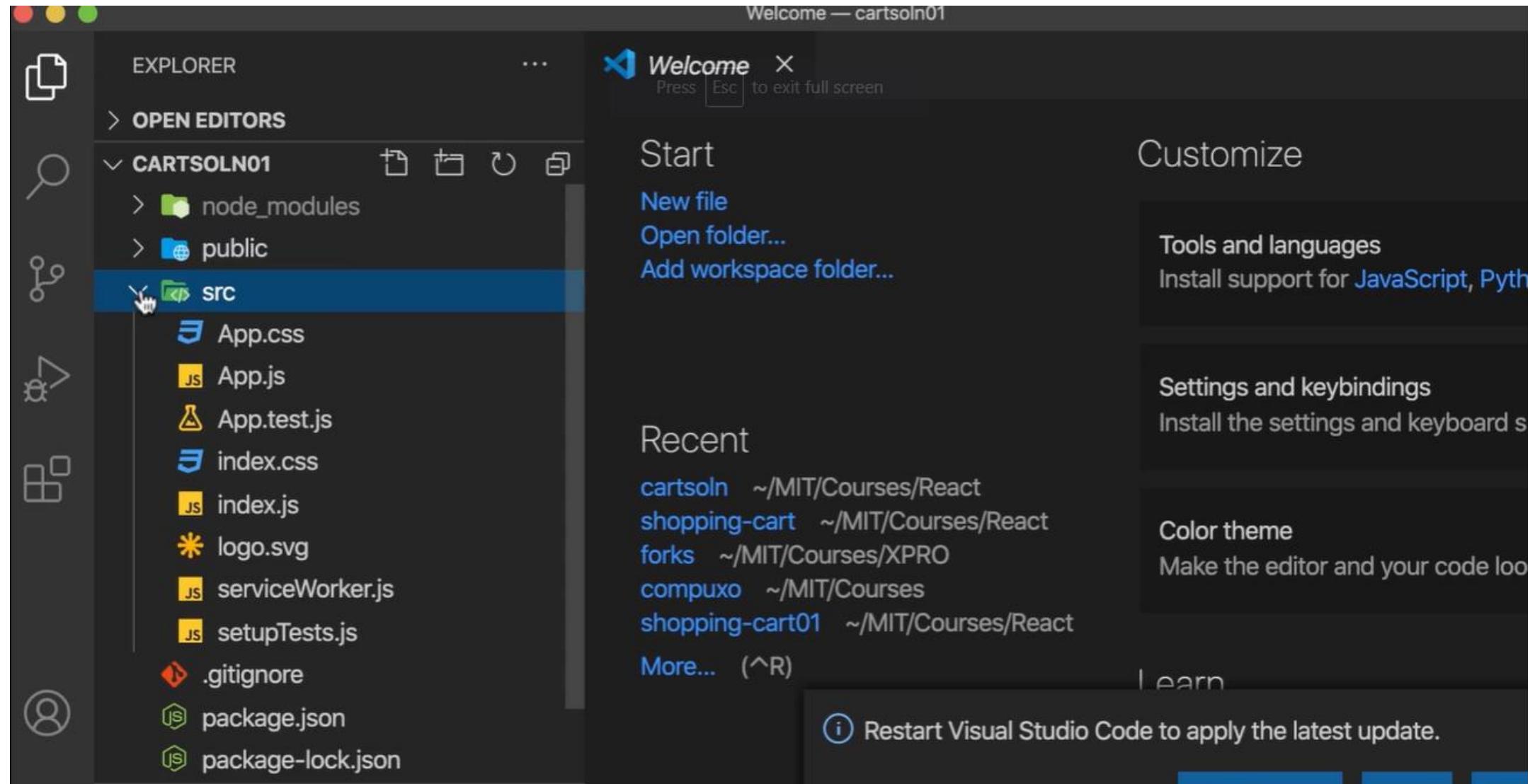
  cd cartsoln01
  npm start

Happy hacking!
✓ React %
```

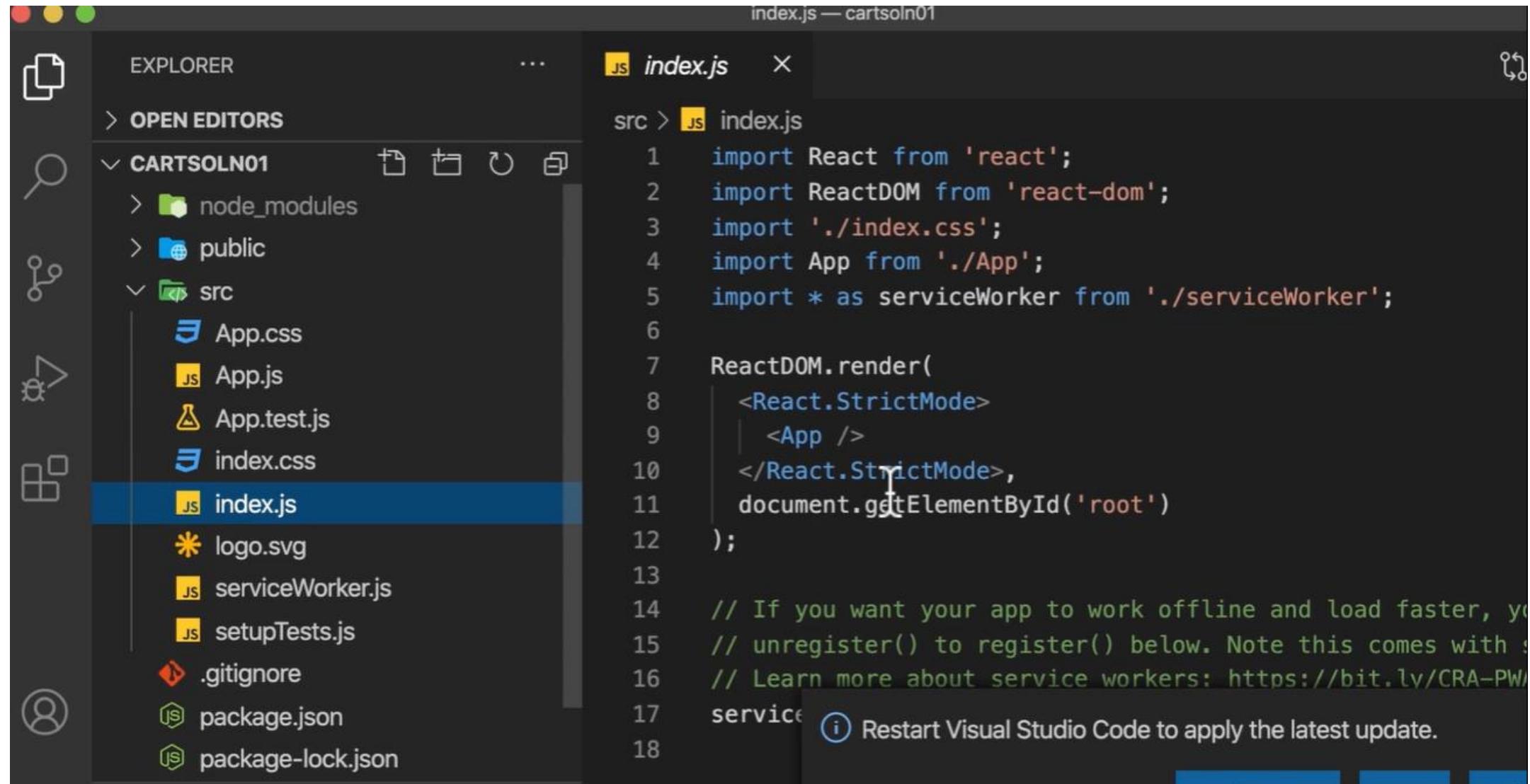
Install Create React App (4/14)



Install Create React App (5/14)



Install Create React App (6/14)



The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER** view: Shows the project structure under "CARTSOLN01". The "src" folder is expanded, displaying files: App.css, App.js, App.test.js, index.css, index.js (which is selected and highlighted in blue), logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, and package-lock.json.
- index.js — cartsln01** editor view: Displays the content of the selected file "index.js". The code is as follows:

```
index.js — cartsln01
src > JS index.js

1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import * as serviceWorker from './serviceWorker';

6
7 ReactDOM.render(
8   <React.StrictMode>
9     |   <App />
10    </React.StrictMode>,
11    document.getElementById('root')
12 );
13
14 // If you want your app to work offline and load faster, you
15 // can register() an ServiceWorker registration (https://bit.ly/CRA-PWA)
16 // Learn more about service workers: https://bit.ly/CRA-PWA
17 serviceWorker.register()
18
  ⓘ Restart Visual Studio Code to apply the latest update.
```

Install Create React App (7/14)

The screenshot shows a dark-themed instance of Visual Studio Code. On the left is the Explorer sidebar, which lists the project structure:

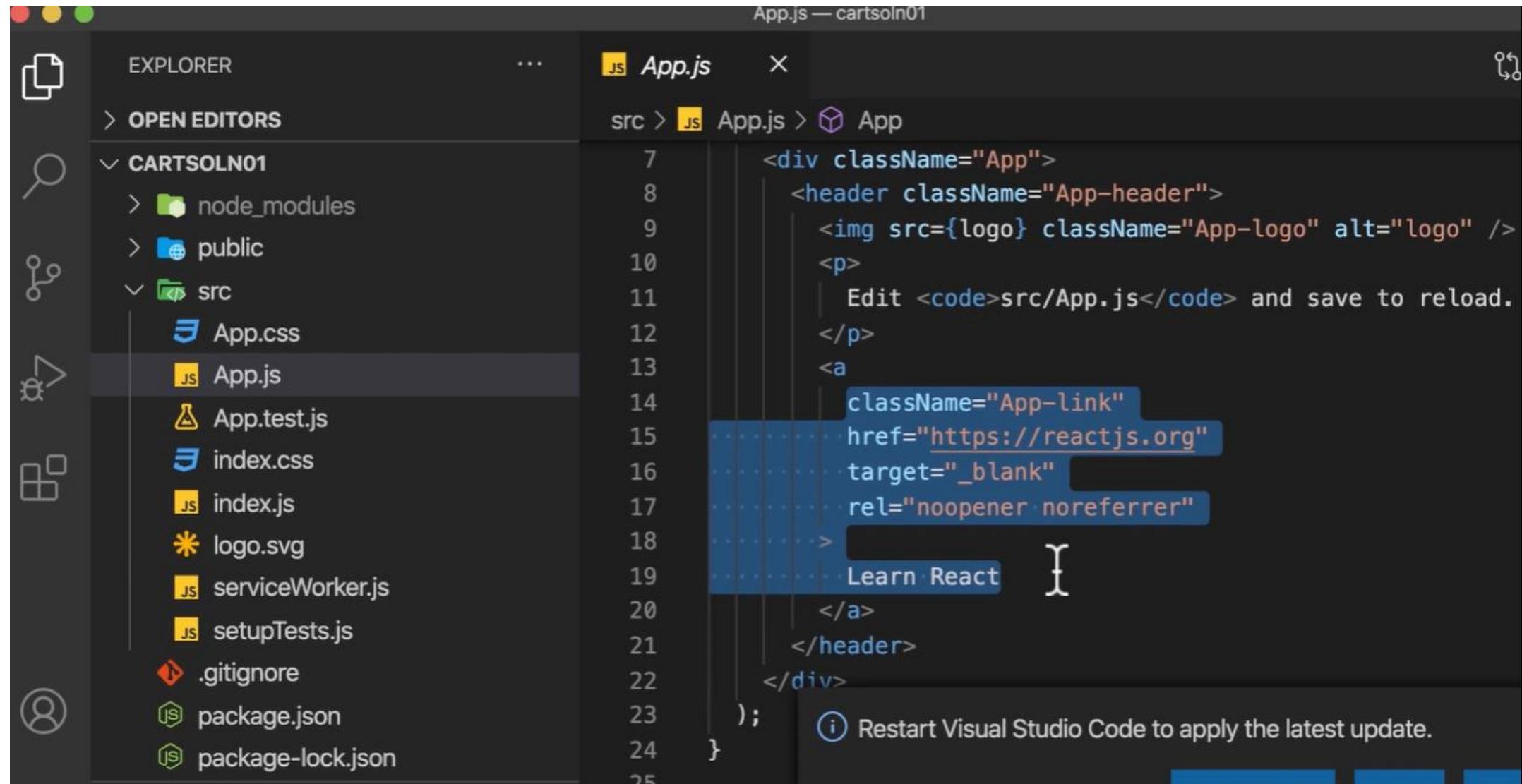
- CARTSOLN01**
 - node_modules
 - public
 - src**
 - App.css
 - App.js**
 - App.test.js
 - index.css
 - index.js
 - logo.svg
 - serviceWorker.js
 - setupTests.js
 - .gitignore
 - package.json
 - package-lock.json

The right side of the interface shows the **App.js** editor tab. The code is as follows:

```
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>
  );
}
```

A tooltip at the bottom right of the editor area says: "Restart Visual Studio Code to apply the latest update."

Install Create React App (8/14)



The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "CARTSOLN01". The "src" folder is expanded, displaying files: App.css, App.js, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, and package-lock.json.
- App.js Editor**: The current file is "App.js" located at "src/App.js". The code is as follows:

```
<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>
```

- A tooltip is displayed over the "Learn React" link, containing the URL <https://reactjs.org>.
- A status bar message at the bottom right says: "Restart Visual Studio Code to apply the latest update."

Install Create React App (9/14)

The screenshot shows a Visual Studio Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "CARTSOLN01". The "src" folder contains "App.css", "App.js", "App.test.js", "index.css", "index.js", "logo.svg", "serviceWorker.js", and "setupTests.js". Other files shown include ".gitignore", "package.json", and "package-lock.json".
- OPEN EDITORS**: 1 UNSAVED
- App.js — cartsoln01** (Active Editor): The code for the main application component.

```
src > JS App.js > App
6  return [
7    <div className="App">
8      <header className="App-header">
9        <img src={logo} className="App-logo" alt="logo" />
10       <p>
11         | Edit <code>src/App.js</code> and save to reload.
12       </p>
13       <a
14         | className="App-link"
15         | href="https://reactjs.org"
16         | target="_blank"
17         | rel="noopener noreferrer"
18       >
19         |   Let's Learn React
20       </a>
21     </header>
22   </div> ⓘ Restart Visual Studio Code to apply the latest update.
23 ];
```

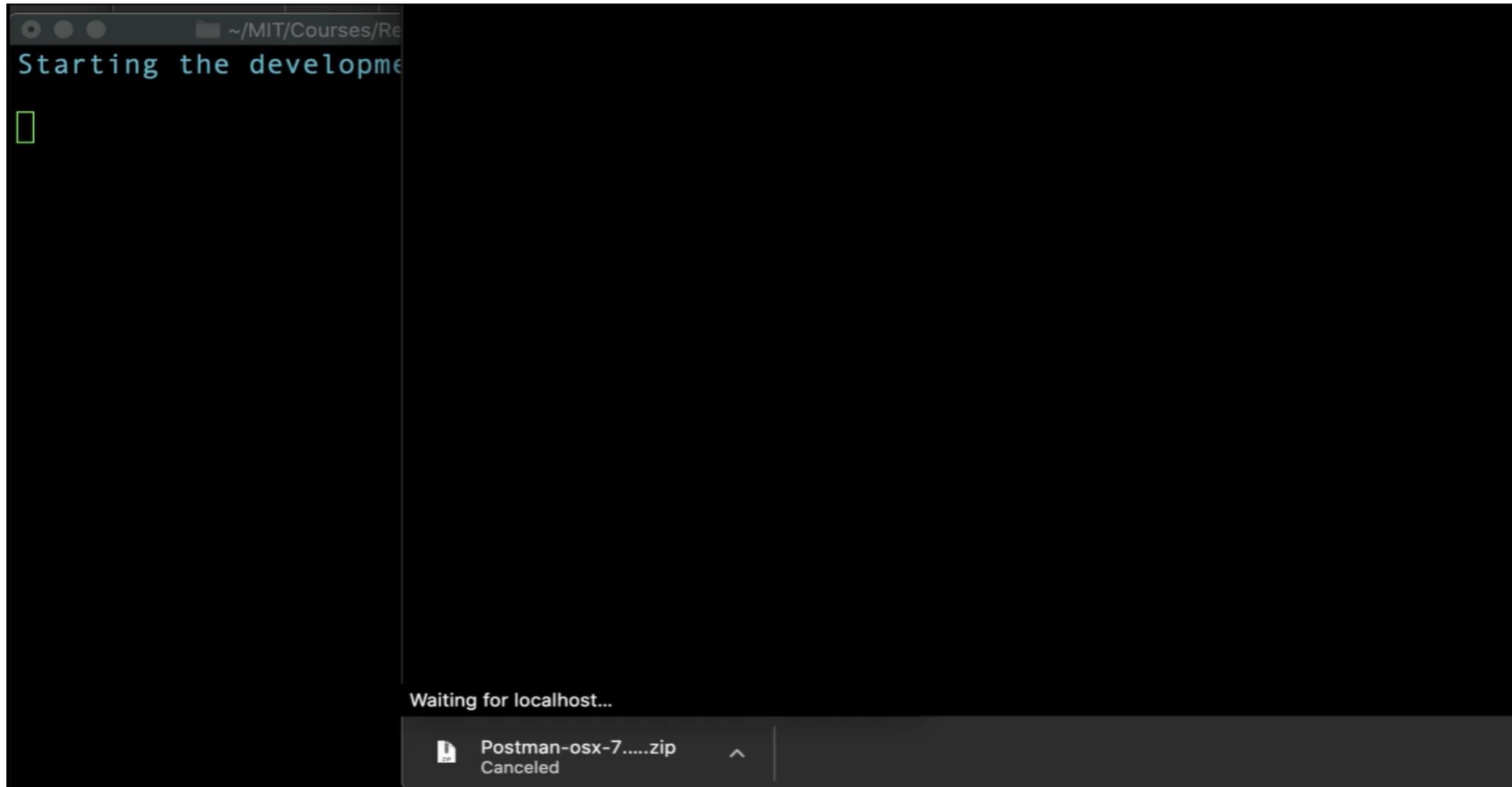
A tooltip message at the bottom right of the editor area says: "Restart Visual Studio Code to apply the latest update."

Install Create React App (10/14)

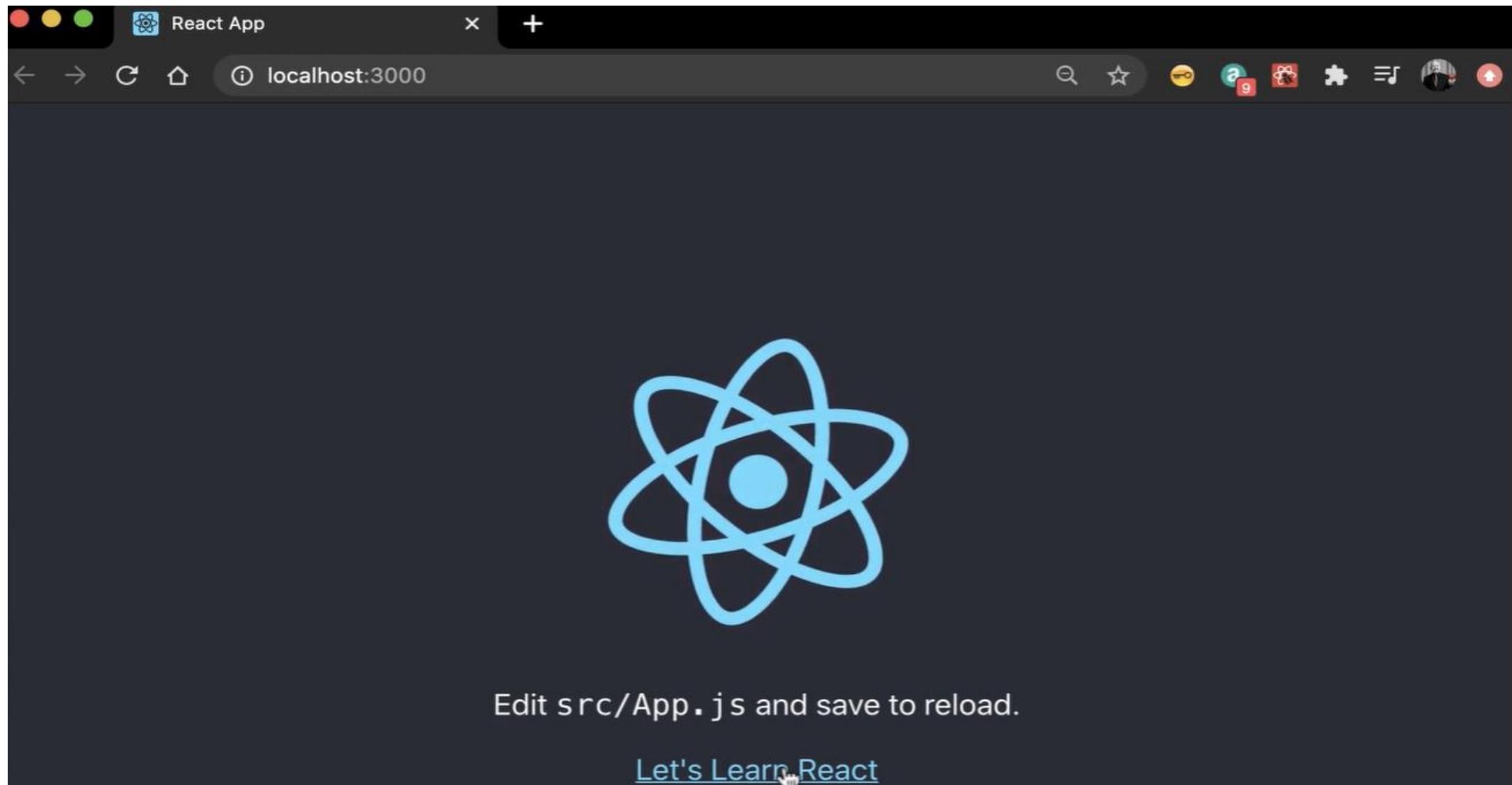
```
[~/React % cd cartsoln01  
[~/cartsoln01 % npm start  
  
> cartsoln01@0.1.0 start /Users/johnwilliams/MIT/Courses/React/cartsoln01  
> react-scripts start
```

m

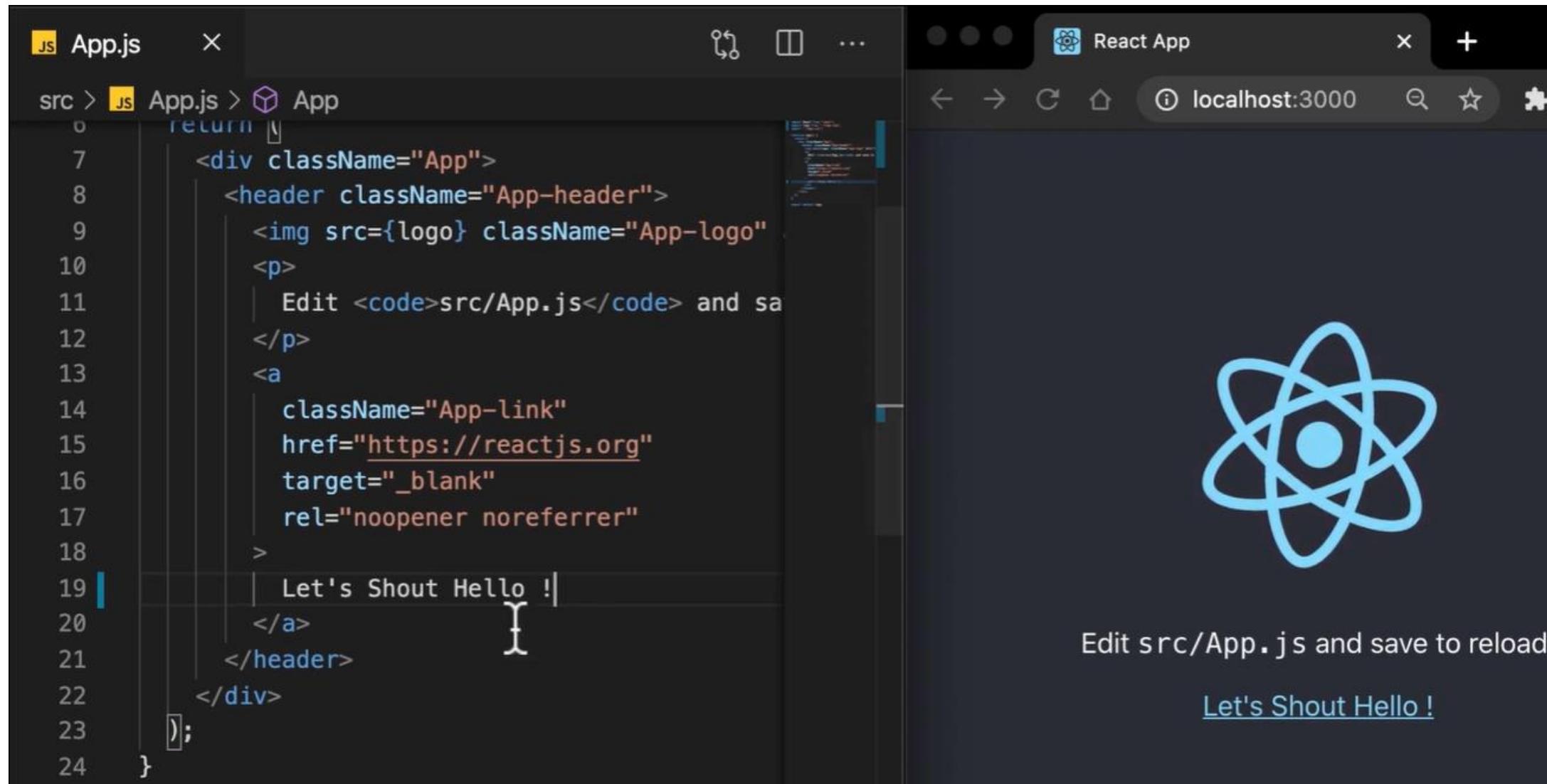
Install Create React App (11/14)



Install Create React App (12/14)



Install Create React App (13/14)



The image shows a development environment with a code editor and a browser window.

Code Editor (left): The file `App.js` is open, displaying the following code:

```
JS App.js    X
src > JS App.js > App
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="React logo" data-bbox="100px 100px 200px 200px"/>
        <p>
          Edit src/App.js and save to reload
        </p>
        <a href="https://reactjs.org" target="_blank" rel="noopener noreferrer">
          Let's Shout Hello !
        </a>
      </header>
    </div>
  );
}
```

Browser (right): A browser window titled "React App" is showing the URL `localhost:3000`. It displays the React logo (a blue atom symbol) and the text "Edit `src/App.js` and save to reload". Below that, there is a blue link labeled "Let's Shout Hello!".

Install Create React App (14/14)

The screenshot shows a development environment with a code editor and a browser window.

Code Editor (left): The file `src/App.js` contains the following code:

```
src > JS App.js > App
  o  LURII \
  7   <div className="App">
  8     <header className="App-header">
  9       <p>
 10         Edit <code>src/App.js</code> and save to
 11         </p>
 12         <a
 13           className="App-link"
 14           href="https://reactjs.org"
 15           target="_blank"
 16           rel="noopener noreferrer"
 17         >
 18           Let's Shout Hello !
 19         </a>
 20       </header>
 21     </div>
 22
 23
 24
```

Browser (right): The browser window titled "React App" shows the rendered output of the code. It displays a header with the text "Let's Shout Hello!" and a link to <https://reactjs.org>.

A tooltip message "Edit src/App.js and save to reload" is visible near the browser window.

Create React App – Shopping Cart Example (1/25)

The screenshot shows the VS Code interface with the following details:

- Title Bar:** App.js — cartsln01
- Explorer:** Shows the project structure under 'CARTSOLN01':
 - public
 - src
 - App.css
 - App.js (highlighted)
 - App.test.js
 - index.css
 - index.js
 - logo.svg
 - serviceWorker.js
 - setupTests.js
 - .gitignore
 - package.json (modified)
 - package-lock.json (modified)
 - README.md

There are 1 unsaved file and 3 changes in the src folder.

- Editor:** The 'App.js' file is open, showing the following code:

```
1 import React from "react";
2 import logo from "./logo.svg";
3 import "./App.css";
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Let's Shout Hello !
19         </a>
20       </header>
21       <main className="App-main">
22         <p>Learn React</p>
23       </main>
24     </div>
25   );
26 }
27
28 <div style={{"background-color: #f0f0f0; padding: 10px; border-radius: 5px; width: fit-content; margin: auto; position: absolute; left: 50%; top: 50%; transform: translate(-50%, -50%);>
29   <h1>Hello, world!</h1>
30   <p>This is a simple React app.</p>
31 </div>
32
33 <script>
34   window.onload = () => {
35     const rootElement = document.getElementById("root");
36     const reactRoot = ReactDOM.createRoot(rootElement);
37     reactRoot.render(<App />);
38   };
39 </script>
```
- Bottom Status Bar:** Shows the status bar with file paths and line numbers.

Create React App – Shopping Cart Example (2/25)

The screenshot shows a dark-themed code editor interface. On the left, there's a sidebar with various icons: a file with a blue circle containing a '1', a magnifying glass, a gear with a '3', a play button, a grid, and a user icon.

The main area displays a file named `src/App.js`. The code contains the following content:

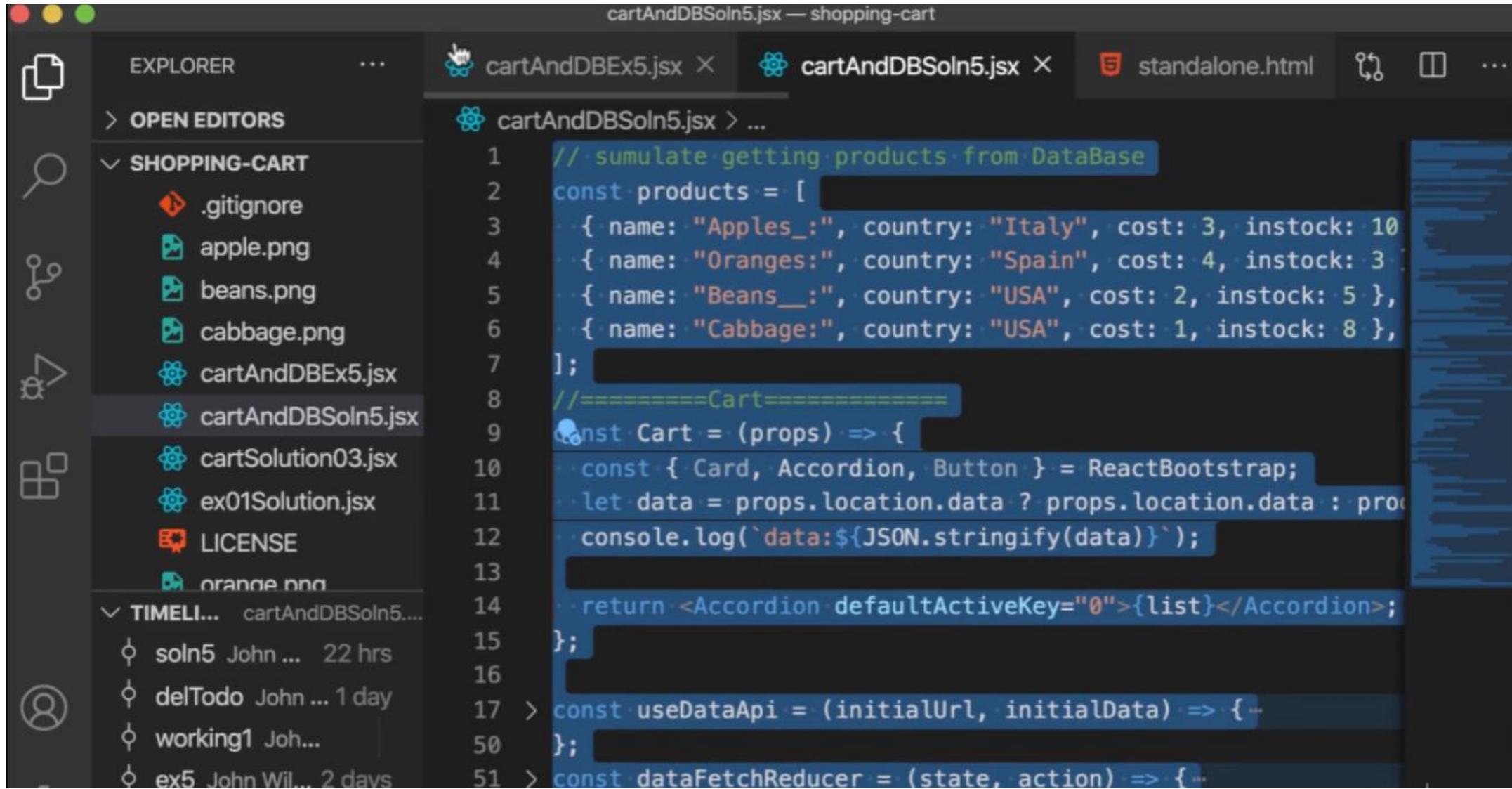
```
10 <p>Edit <code>src/App.js</code> and save to reload.</p>
11 </p>
12 <a href="https://reactjs.org" target="_blank" rel="noopener noreferrer">
13   className="App-link"
14   href="https://reactjs.org"
15   target="_blank"
16   rel="noopener noreferrer"
17 >
18   Let's Shout Hello !
19 </a>
```

A context menu is open at the top of the screen, listing recent files and other options. The recent files section includes:

- ~/MIT/Courses/React/dataFetch
- ~/MIT/Courses/React/es6warmup
- ~/MIT/Courses/React/JohnReact/todos2/todos2
- ~/MIT/Courses/React/JohnReact/bankFunc03/src/components/SetupRestaurants.jsx
- ~/MIT/Courses/XPRO/Course1/wordsnake/randomwalk.html
- ~/.zshrc
- ~/MIT/Courses/OAuth2/SigninWidget/signin.html
- ~/MIT/CEE/Slack/slackapp001.js
- ~/.oh-my-zsh/themes/arrow.zsh-theme
- ~/.oh-my-zsh/oh-my-zsh.sh

At the bottom of the recent files list, there's a "More..." link and a "Clear Recently Opened" option.

Create React App – Shopping Cart Example (3/25)

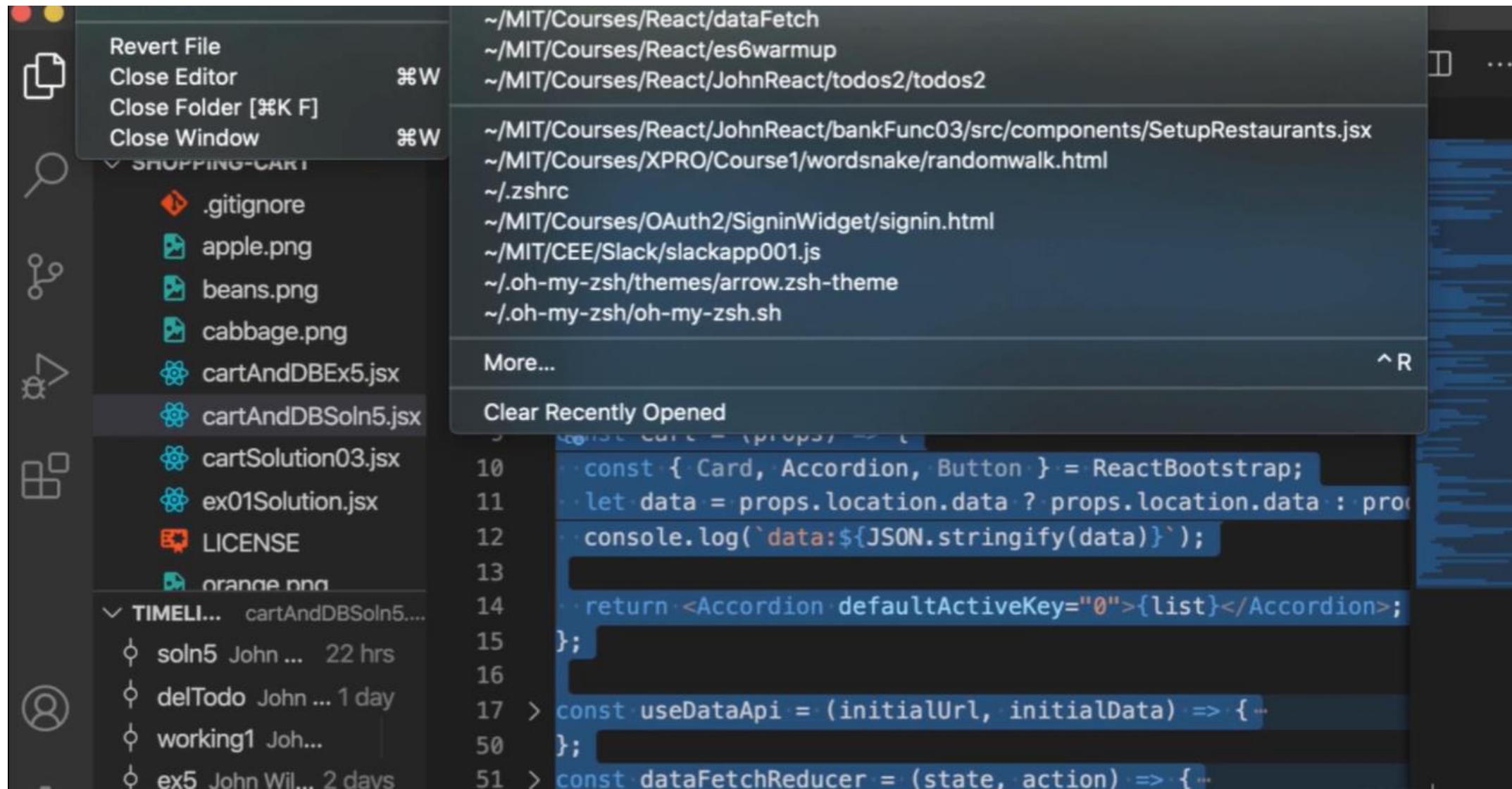
A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows "cartAndDBSoln5.jsx — shopping-cart". The left sidebar has icons for File, Explorer, Open Editors, Shopping-Cart, GitHub, Search, Find, and Help. The "OPEN EDITORS" section shows "SHOPPING-CART" expanded, listing ".gitignore", "apple.png", "beans.png", "cabbage.png", "cartAndDBEx5.jsx", "cartAndDBSoln5.jsx" (which is currently selected), "cartSolution03.jsx", "ex01Solution.jsx", "LICENSE", and "orange.png". Below this, under "TIMELINE", there are four entries: "soln5 John ... 22 hrs", "delTodo John ... 1 day", "working1 Joh...", and "ex5 John Wil... 2 days". The main editor area displays the file "cartAndDBSoln5.jsx" with the following content:

```
// simulate getting products from DataBase
const products = [
  { name: "Apples:", country: "Italy", cost: 3, instock: 10 },
  { name: "Oranges:", country: "Spain", cost: 4, instock: 3 },
  { name: "Beans:", country: "USA", cost: 2, instock: 5 },
  { name: "Cabbage:", country: "USA", cost: 1, instock: 8 },
];
//=====Cart=====
const Cart = (props) => {
  const { Card, Accordion, Button } = ReactBootstrap;
  let data = props.location.data ? props.location.data : products;
  console.log(`data:${JSON.stringify(data)}`);

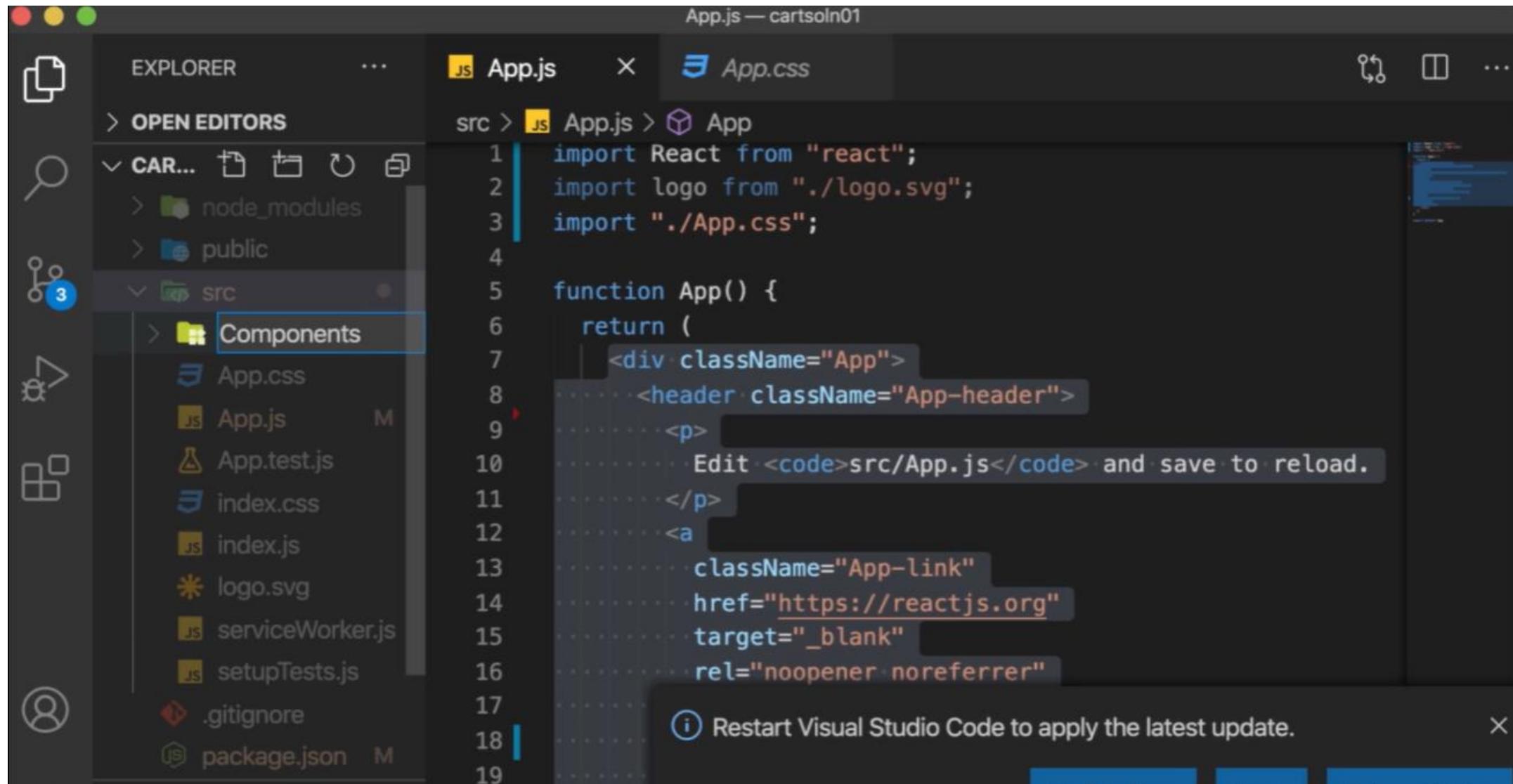
  return <Accordion defaultActiveKey="0">{list}</Accordion>;
};

> const useDataApi = (initialUrl, initialData) => {
50 };
> const dataFetcher = (state, action) => {
```

Create React App – Shopping Cart Example (4/25)



Create React App – Shopping Cart Example (5/25)

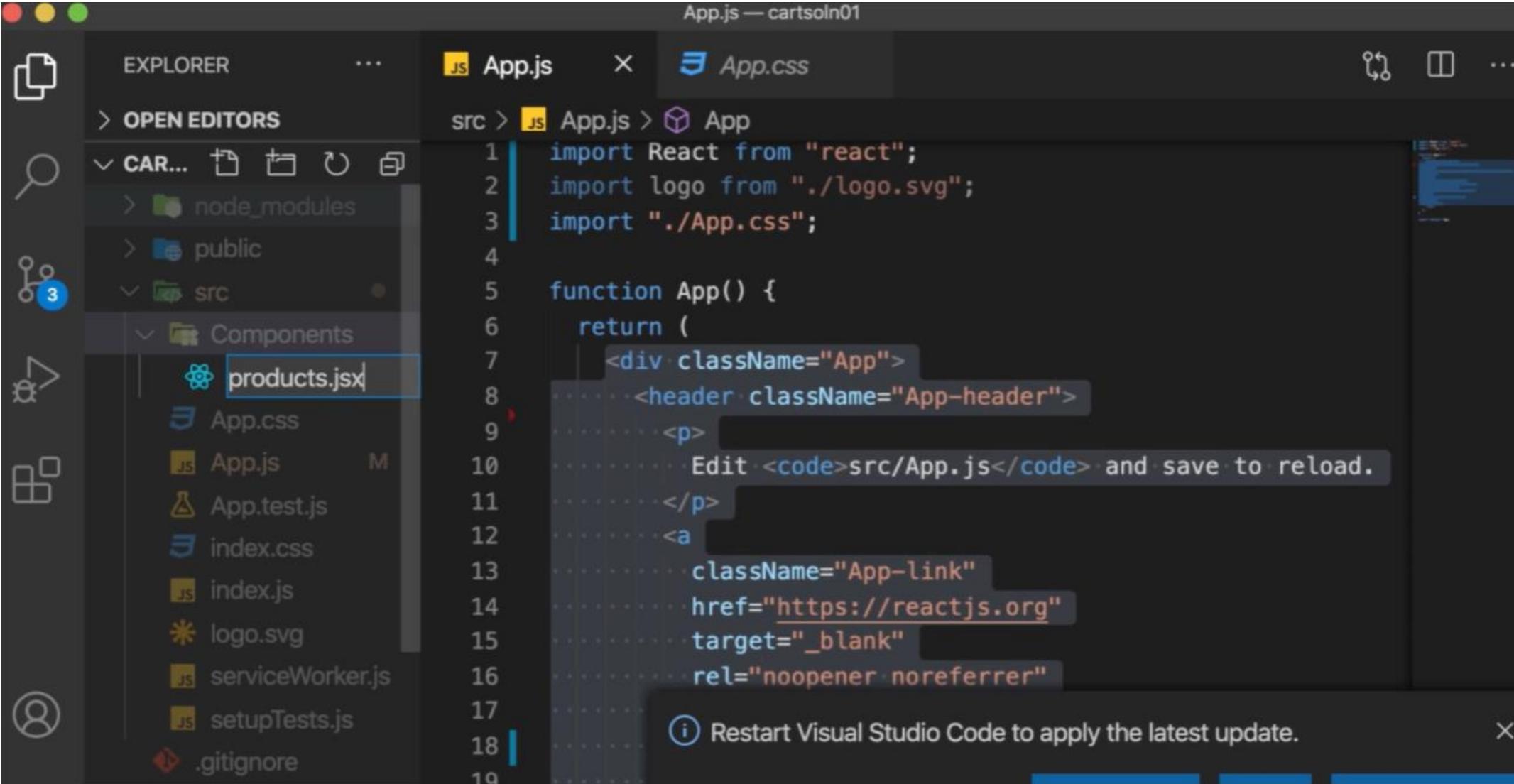


The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure with files like `node_modules`, `public`, and `src`. Inside `src`, there is a `Components` folder which is currently selected.
- Editor View:** Displays the `App.js` file content:

```
1 import React from "react";
2 import logo from "./logo.svg";
3 import "./App.css";
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Get started
19         </a>
20       </header>
21       <main className="App-main">
22         <h1>Welcome to React</h1>
23         <p>This is a simple React app. Edit <code>src/App.js</code> and save to reload.
24         </p>
25         <img alt="React logo" src={logo} />
26       </main>
27     </div>
28   )
29 }
30
31 export default App;
```
- Bottom Status Bar:** A message indicates: **Restart Visual Studio Code to apply the latest update.**

Create React App – Shopping Cart Example (6/25)



App.js — cartsIn01

EXPLORER ...

OPEN EDITORS

CAR... node_modules public src

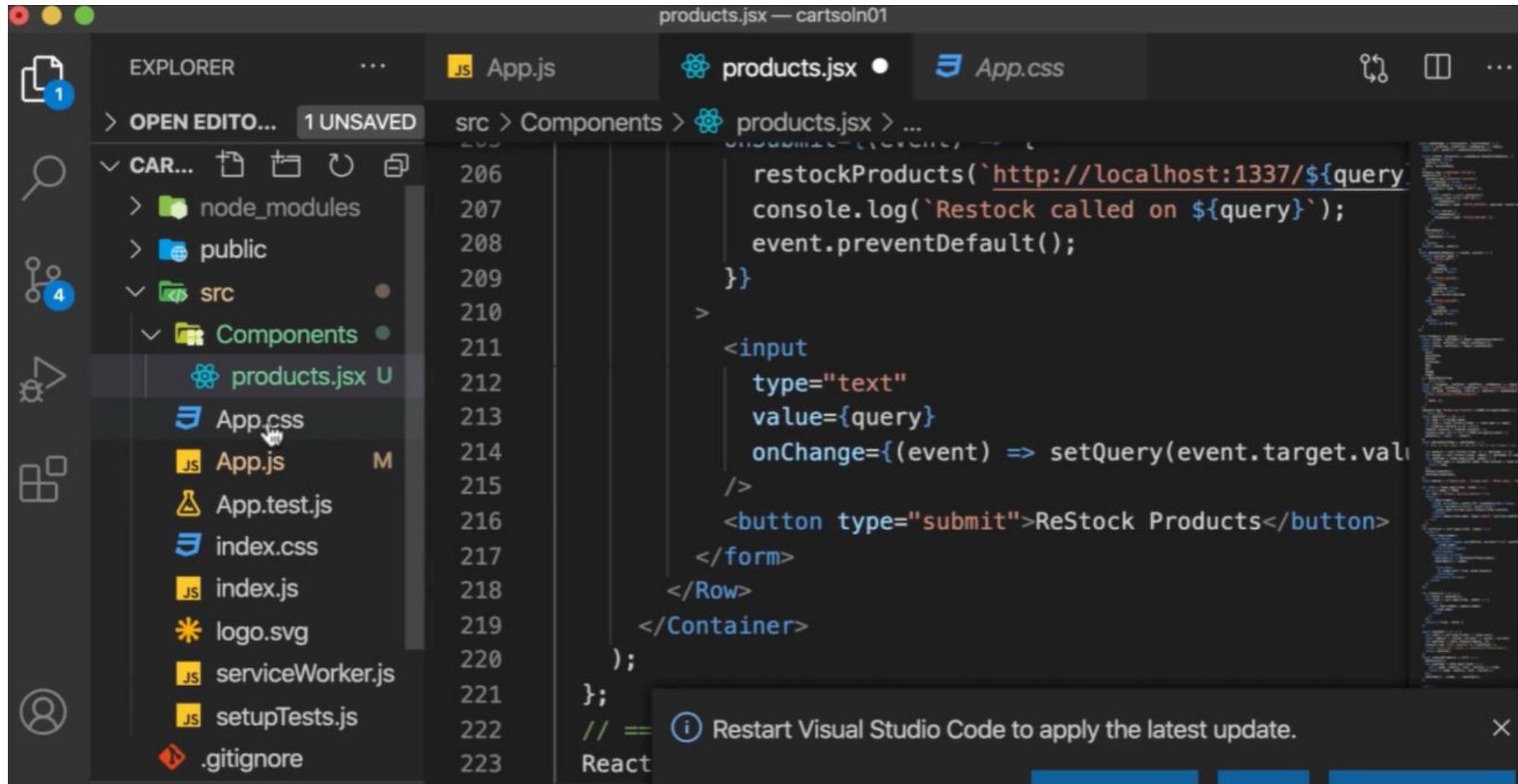
Components products.jsx App.css App.js App.test.js index.css index.js logo.svg serviceWorker.js setupTests.js .gitignore

src > App.js > App

```
1 import React from "react";
2 import logo from "./logo.svg";
3 import "./App.css";
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Get started
19         </a>
10       </header>
11       <main className="App-main">
12         <h1>Welcome to React</h1>
13         <p>This is a simple React app. Edit <code>src/App.js</code> and save to reload.
14         </p>
15       </main>
16     </div>
17   );
18 }
19
20 export default App;
```

Restart Visual Studio Code to apply the latest update.

Create React App – Shopping Cart Example (7/25)



The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** products.jsx — cartsIn01
- Explorer:** Shows the project structure:
 - CAR... (with 1 unsaved file)
 - node_modules
 - public
 - src (with 4 files):
 - Components (with 1 file: products.jsx)
 - App.css
 - App.js
 - App.test.js
 - index.css
 - index.js
 - logo.svg
 - serviceWorker.js
 - setupTests.js
 - .gitignore
- Editor:** The products.jsx file is open, showing code related to restocking products. The code includes a form with an input field and a submit button, and a function to handle the restock query.
- Bottom Status Bar:** A message says "Restart Visual Studio Code to apply the latest update."

```
products.jsx — cartsIn01

EXPLORER ... JS App.js products.jsx • App.css ...
src > Components > products.jsx > ...
206 restockProducts(`http://localhost:1337/${query}`);
207 console.log(`Restock called on ${query}`);
208 event.preventDefault();
209 }
210 >
211 <input
212   type="text"
213   value={query}
214   onChange={(event) => setQuery(event.target.value)}
215   />
216   <button type="submit">ReStock Products</button>
217 </form>
218 </Row>
219 </Container>
220 );
221 };
222 // == ⓘ Restart Visual Studio Code to apply the latest update.
223 React
```

Create React App – Shopping Cart Example (8/25)

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under "CARTSOLN01".
- Editor View:** The file "products.jsx" is open in the center editor tab.
- Code Content:** The code defines a component named "products" which simulates getting products from a database. It includes a list of product objects and a "Cart" component.

```
products.jsx — cartsln01
App.js products.jsx App.css
src > Components > products.jsx > ...
1 import React from "react";
2 import ReactBootstrap from "react-bootstrap";
3 import axios from "axios";
4
5 // simulate getting products from DataBase
6 const products = [
7   { name: "Apples:", country: "Italy", cost: 3, instock: 10 },
8   { name: "Oranges:", country: "Spain", cost: 4, instock: 3 },
9   { name: "Beans:", country: "USA", cost: 2, instock: 5 },
10  { name: "Cabbage:", country: "USA", cost: 1, instock: 8 },
11];
12 //=====Cart=====
13 const Cart = (props) => {
14   const { Card, Accordion, Button } = ReactBootstrap;
15   let data = props.location.data ? props.location.data : products;
16   console.log(`data:${JSON.stringify(data)}`);
17
18   return (
19     <div>
```

- Message Bar:** A message at the bottom right says "Restart Visual Studio Code to apply the latest update."

Create React App – Shopping Cart Example (9/25)

```
Compiled with warnings.
```

```
./src/App.js
  Line 2:8:  'logo' is defined but never used  no-unused-vars
```

```
Search for the keywords to learn more about each warning.
```

```
To ignore, add // eslint-disable-next-line to the line before.
```

```
^C
```

```
[?130 cartsoln01 % npm install react-bootstrap
npm WARN tsutils@3.17.1 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >=
dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev
  3.7.0-beta but none is installed. You must install peer dependencies yourself.
```

```
+ react-bootstrap@1.3.0
```

```
updated 1 package and audited 1645 packages in 10.98s
```

```
67 packages are looking for funding
  run `npm fund` for details
```

```
found 0 vulnerabilities
```

```
✓ cartsoln01 % █
```

Create React App – Shopping Cart Example (10/25)

```
dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev
  3.7.0-beta but none is installed. You must install peer dependencies yourself.

+ react-bootstrap@1.3.0
updated 1 package and audited 1645 packages in 10.98s

67 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

[✓ cartsoln01 % npm install axios
npm WARN tsutils@3.17.1 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >=
  dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev
  3.7.0-beta but none is installed. You must install peer dependencies yourself.

+ axios@0.20.0
updated 1 package and audited 1645 packages in 9.649s

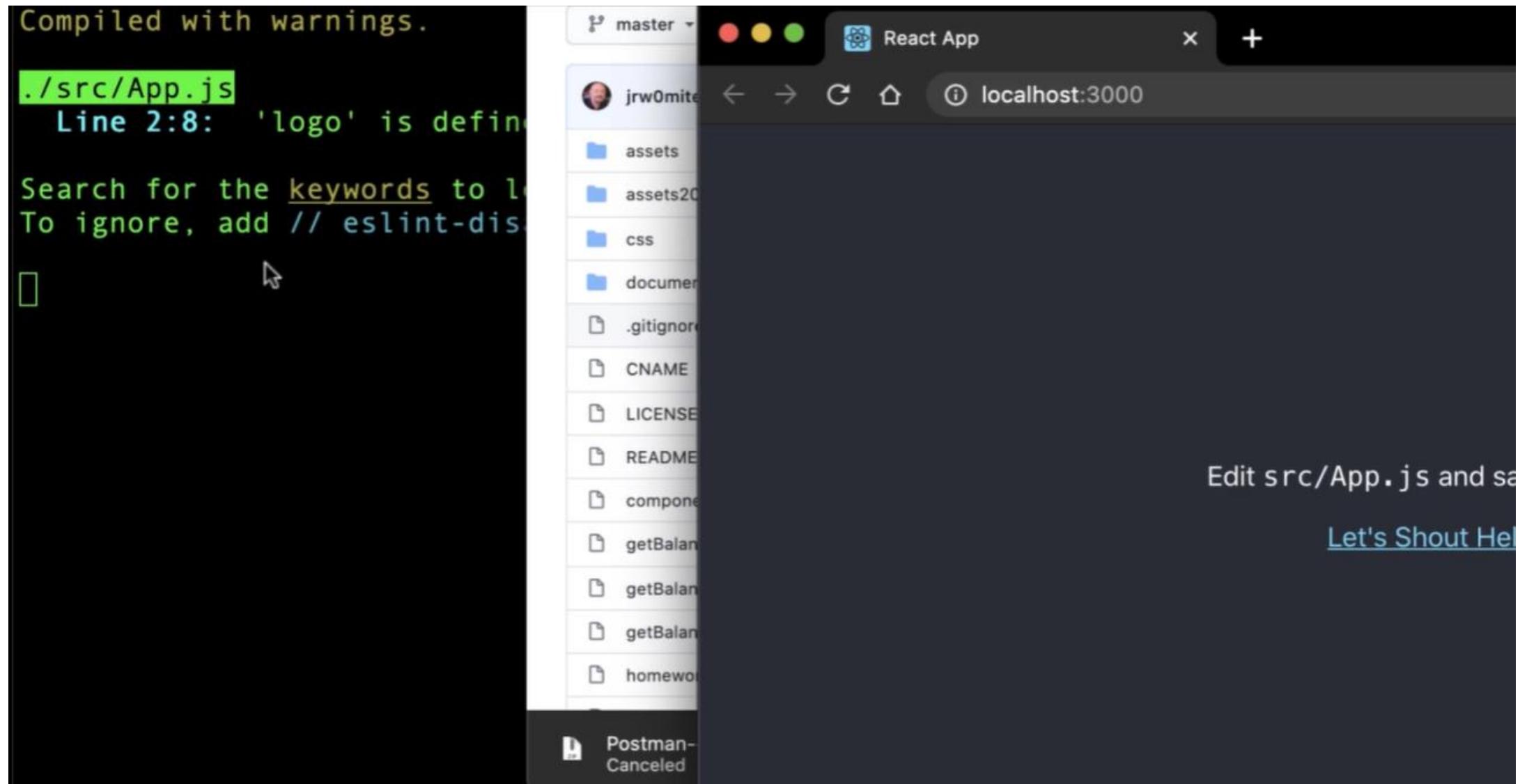
68 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

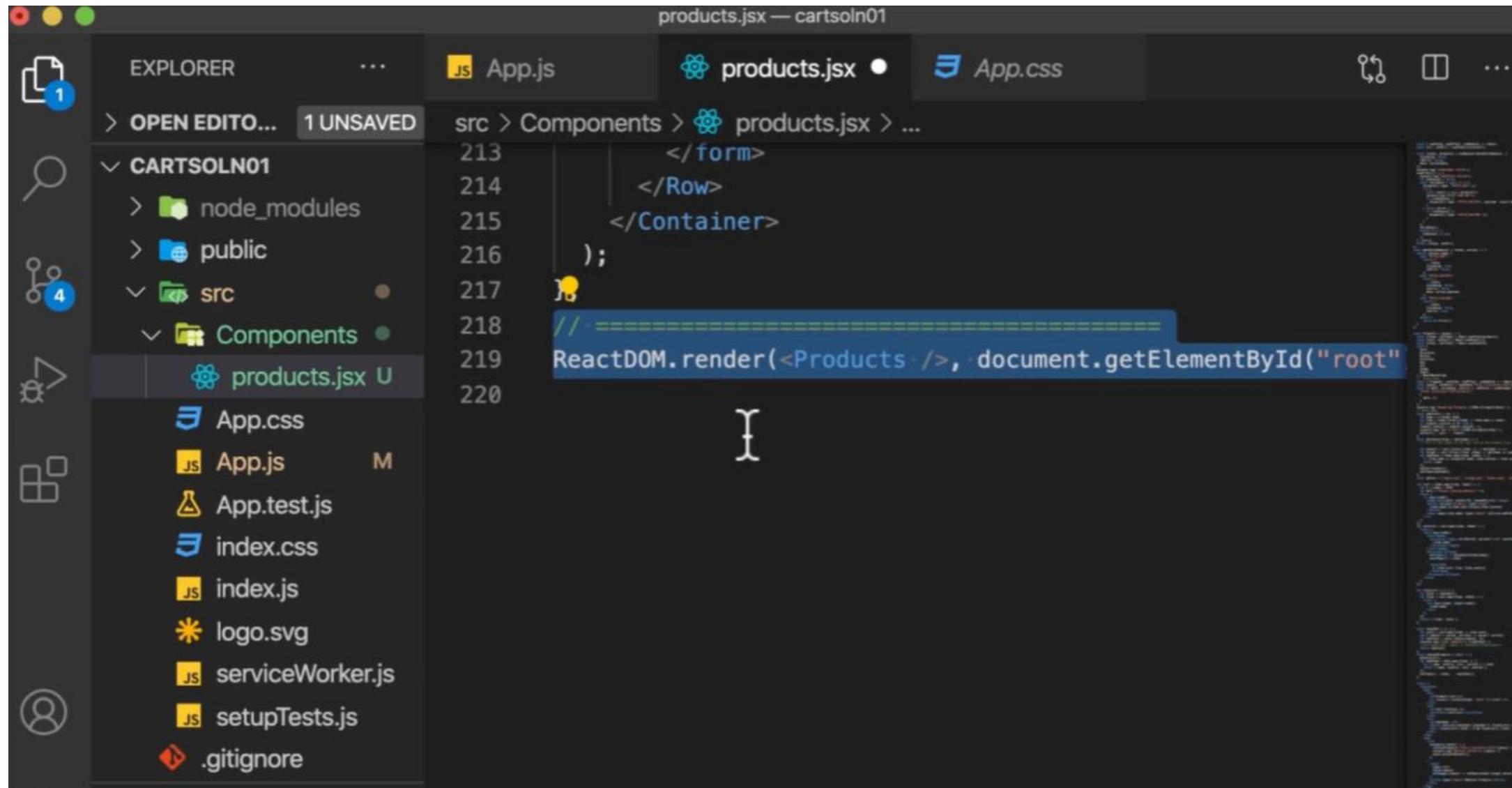
Create React App – Shopping Cart Example (11/25)

```
[✓ cartsoln01 % npm start  
> cartsoln01@0.1.0 start /Users/johnwilliams/MIT/Courses/React/cartsoln01  
> react-scripts start
```

Create React App – Shopping Cart Example (12/25)



Create React App – Shopping Cart Example (13/25)



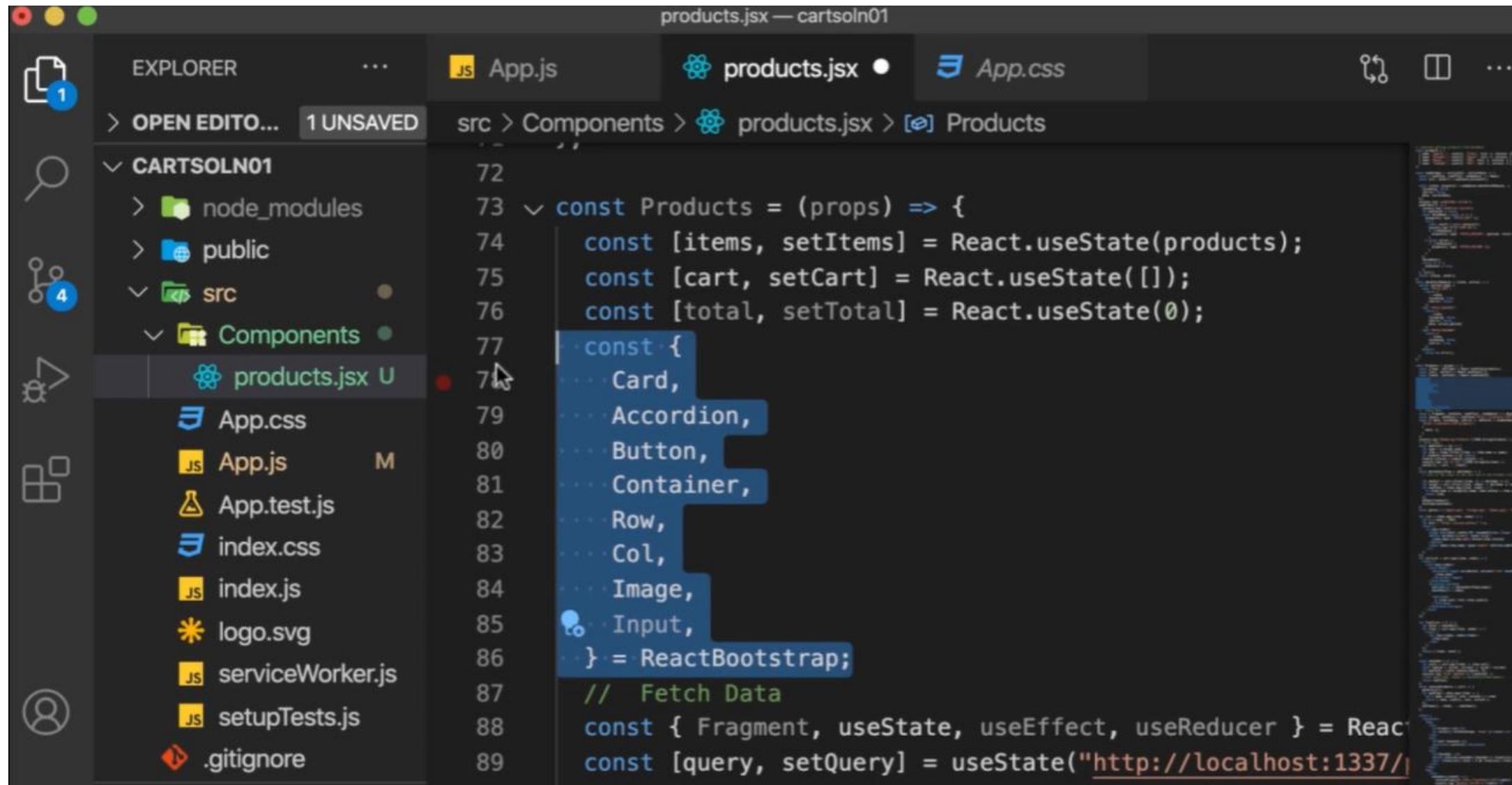
The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows the project structure under "CARTSOLN01".
- File Explorer:** Shows files like App.js, App.css, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, and .gitignore.
- Editor:** The "products.jsx" file is open in the editor.
- Code:** The code is as follows:

```
products.jsx — cartsln01
App.js products.jsx App.css
src > Components > products.jsx > ...
213     </form>
214     </Row>
215   </Container>
216 );
217 // -----
218 ReactDOM.render(<Products />, document.getElementById("root"))
219
220
```

A yellow dot is placed on line 217, and a tooltip shows the code for line 218.

Create React App – Shopping Cart Example (14/25)

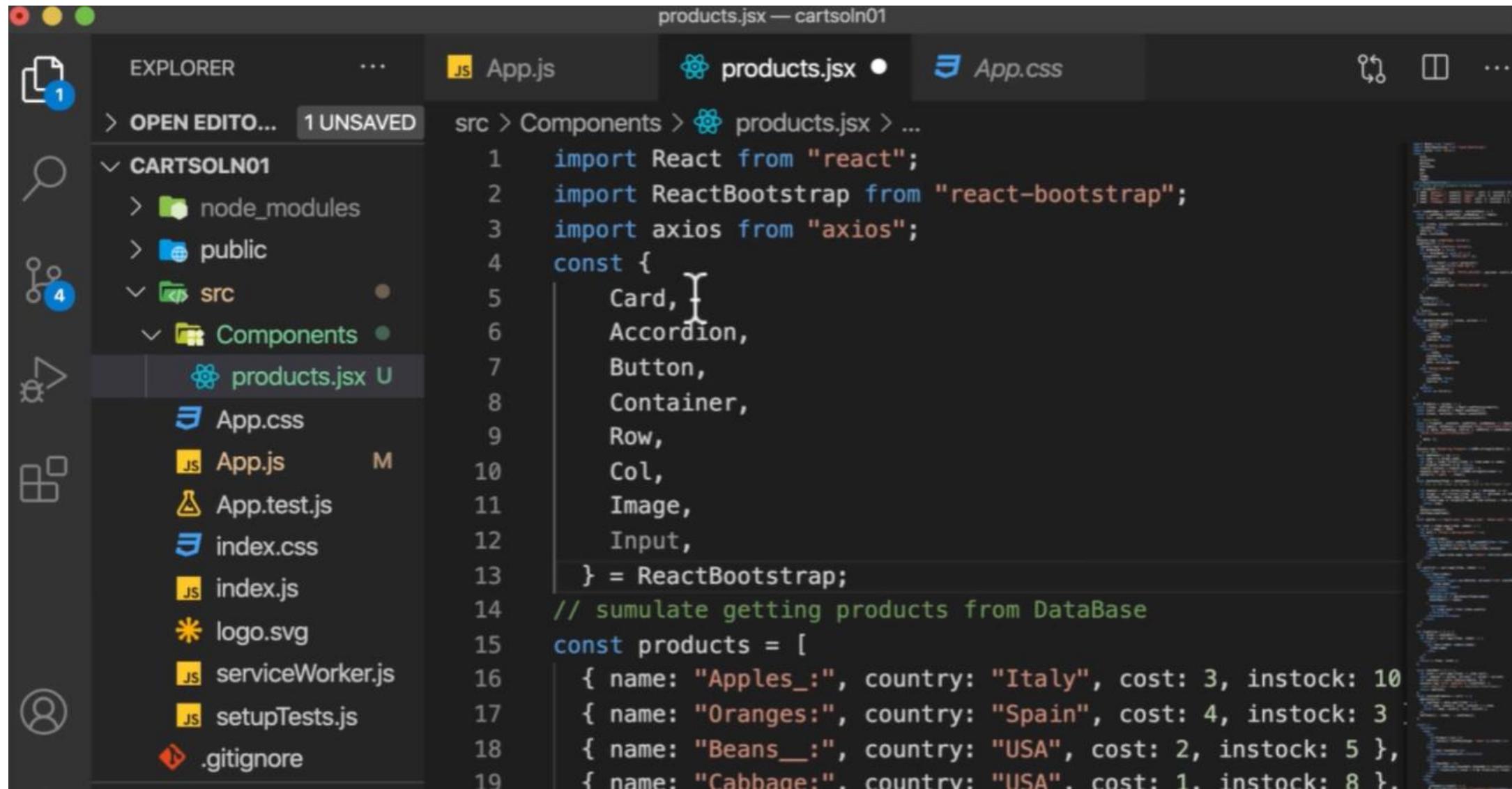


The screenshot shows a dark-themed instance of Visual Studio Code (VS Code) with the title bar "products.jsx — cartsln01". The Explorer sidebar on the left shows a project structure with a file named "products.jsx" currently selected. The main editor area displays the following code:

```
products.jsx — cartsln01
EXPLORER ... App.js products.jsx App.css ...
src > Components > products.jsx > [o] Products
72
73  const Products = (props) => {
74    const [items, setItems] = React.useState(products);
75    const [cart, setCart] = React.useState([]);
76    const [total, setTotal] = React.useState(0);
77    const {
78      Card,
79      Accordion,
80      Button,
81      Container,
82      Row,
83      Col,
84      Image,
85      Input,
86    } = ReactBootstrap;
87    // Fetch Data
88    const { Fragment, useState, useEffect, useReducer } = React;
89    const [query, setQuery] = useState("http://localhost:1337/");

The code uses React hooks like useState and useEffect, and imports components from ReactBootstrap. A tooltip is visible over the "Input" import statement.
```

Create React App – Shopping Cart Example (15/25)



The screenshot shows a dark-themed instance of Visual Studio Code (VS Code) with the title bar "products.jsx — cartsln01". The Explorer sidebar on the left shows a project structure for "CARTSOLN01" with files like "node_modules", "public", "src", "Components", "products.jsx", "App.css", "App.js", "App.test.js", "index.css", "index.js", "logo.svg", "serviceWorker.js", "setupTests.js", and ".gitignore". There are 1 unsaved file and 4 changes in the workspace. The products.jsx file is open in the editor, displaying code that imports React, ReactBootstrap, and axios, and defines a component that uses Card, Accordion, Button, Container, Row, Col, Image, and Input components from ReactBootstrap. It also includes a simulated database of products.

```
products.jsx — cartsln01
EXPLORER    ...
JS App.js    CS products.jsx ●  CSS App.css
src > Components > CS products.jsx > ...
1 import React from "react";
2 import ReactBootstrap from "react-bootstrap";
3 import axios from "axios";
4 const {
5   Card,
6   Accordion,
7   Button,
8   Container,
9   Row,
10  Col,
11  Image,
12  Input,
13 } = ReactBootstrap;
14 // simulate getting products from DataBase
15 const products = [
16   { name: "Apples:", country: "Italy", cost: 3, instock: 10 },
17   { name: "Oranges:", country: "Spain", cost: 4, instock: 3 },
18   { name: "Beans:", country: "USA", cost: 2, instock: 5 },
19   { name: "Cabbage:", country: "USA", cost: 1, instock: 8 }]
```

Create React App – Shopping Cart Example (16/25)

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "CARTSOLN01". The "Components" folder contains "products.jsx" (which is currently open), "App.css", "App.js", "App.test.js", "index.css", "index.js", "logo.svg", "serviceWorker.js", and "setupTests.js". Other files like "node_modules" and "public" are also visible.
- Editor:** The "products.jsx" tab is active, displaying the following code:

```
products.jsx — cartsoln01
src > Components > products.jsx > ...
1 import React from "react";
2 import ReactBootstrap from "react-bootstrap";
3 import axios from "axios";
4 import {
5   Card,
6   Accordion,
7   Button,
8   Container,
9   Row,
10  Col,
11  Image,
12  Input,
13 } from "react-bootstrap";
14 // simulate getting products from DataBase
15 const products = [
16   { name: "Apples:", country: "Italy", cost: 3, instock: 10 },
17   { name: "Oranges:", country: "Spain", cost: 4, instock: 3 },
18   { name: "Beans:", country: "USA", cost: 2, instock: 5 },
19   { name: "Cabbage:", country: "USA", cost: 1, instock: 8 },

```

Create React App – Shopping Cart Example (17/25)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CARTSOLN01". The "src" folder contains "Components", "products.jsx", "App.css", and "App.js" (which is currently selected).
- Editor:** The "App.js" editor tab is active, displaying the following code:

```
4
5  function App() {
6    return (
7      <div className="App">
8        <header className="App-header">
9          <p>
10            Edit <code>src/App.js</code> and save to reload.
11          </p>
12          <a
13            className="App-link"
14            href="https://reactjs.org"
15            target="_blank"
16            rel="noopener noreferrer"
17          >
18            Let's Shout Hello !
19          </a>
20        </header>
21      </div>
22    )
23  }
```
- Status Bar:** Shows "xPRO" at the bottom right.

Create React App – Shopping Cart Example (18/25)

The screenshot shows the VS Code interface with the following details:

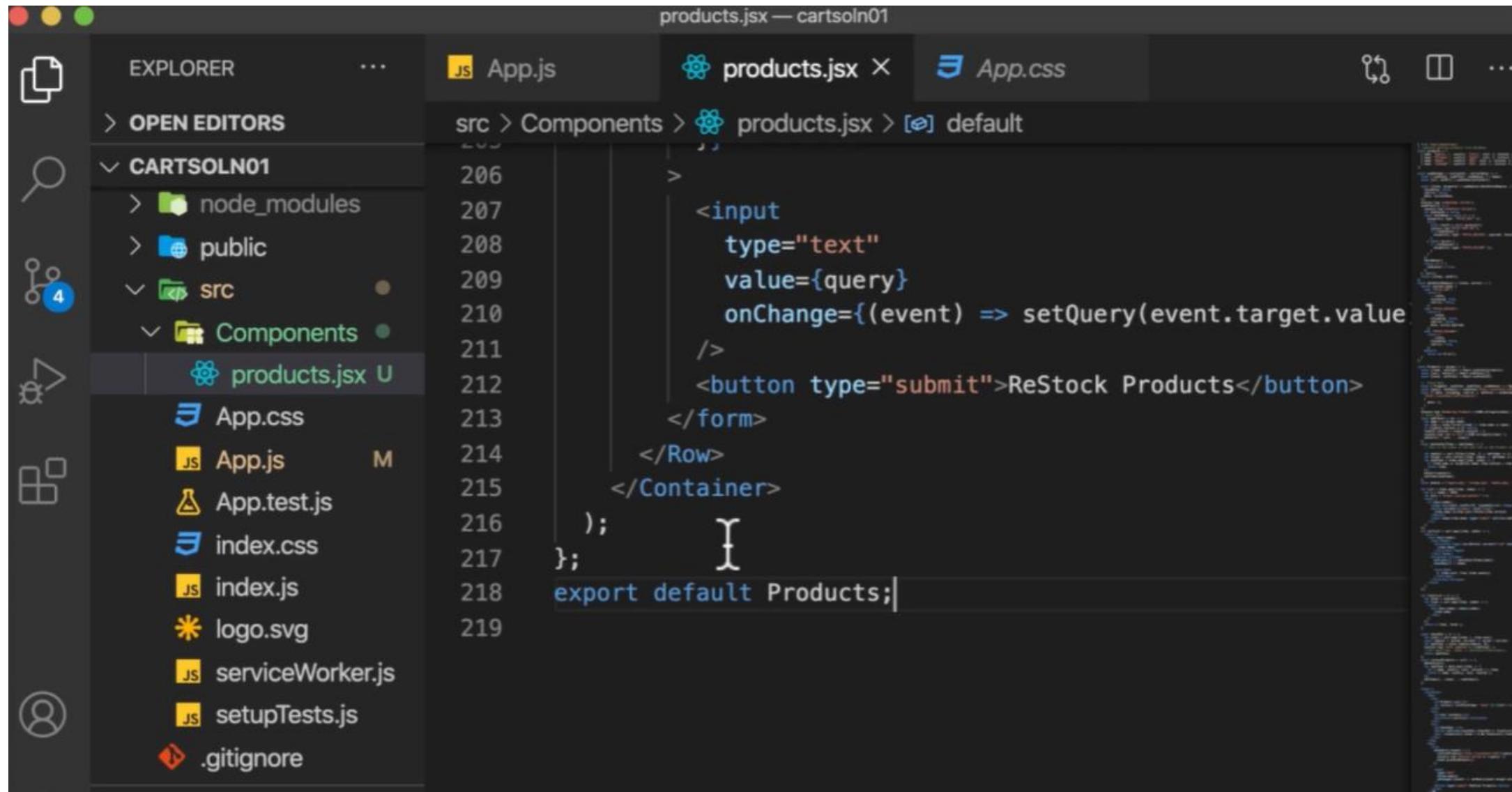
- EXPLORER** view: Shows the project structure. The `src` folder contains `Components`, `products.jsx`, `App.css`, `App.js`, `App.test.js`, `index.css`, `index.js`, `logo.svg`, `serviceWorker.js`, `setupTests.js`, and `.gitignore`. A cursor is over `products.jsx`.
- App.js — cartsoin01** tab: The active editor shows the following code:

```
import React from "react";
import logo from "./logo.svg";
import "./App.css";
import Products from "./Components/products";

function App() {
  return <Products></Products>;
}

export default App;
```

Create React App – Shopping Cart Example (19/25)



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- Title Bar:** products.jsx — cartsoln01
- File Explorer (Left):** Shows the project structure under 'CARTSOLN01': node_modules, public, src (with Components, App.css, App.js, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js), and .gitignore. There are 4 changes pending.
- Editor Area (Center):** The file 'products.jsx' is open, showing the following code:

```
206 >
207 <input
208   type="text"
209   value={query}
210   onChange={(event) => setQuery(event.target.value)}
211   />
212   <button type="submit">ReStock Products</button>
213 </form>
214 </Row>
215 </Container>
216 );
217 };
218 export default Products;
```
- Right Side:** Includes a search bar, a preview pane showing a dark-themed UI, and a sidebar with various icons.

Create React App – Shopping Cart Example (20/25)

The screenshot shows a development environment with a code editor and a browser window.

Code Editor (VS Code):

- EXPLORER:** Shows the project structure:
 - CARTSOLN01
 - node_modules
 - public
 - src
 - Components
 - products.jsx
 - App.css
 - App.js
 - App.test.js
 - index.css
 - index.js
 - logo.svg
 - serviceWorker.js
 - setupTests.js
 - .gitignore
- App.js:** The code is partially visible, showing imports and component definitions.

Browser: A React App running at localhost:3000.

The application displays a **Product List** with four items:

- Apples :: \$3 - Stock=10
- Oranges :: \$4 - Stock=3
- Bananas :: \$2 - Stock=5
- Cabbage :: \$1 - Stock=8

Each item has a "Submit" button. Below the list, there are links for **Cart Contents** and **CheckOut**.

Create React App – Shopping Cart Example (21/25)

The screenshot shows a development environment with the following details:

- File Explorer (Left):** Displays the project structure under "CARTSOLN01". Key files shown include "App.css", "App.js", "App.test.js", "index.css", "index.js", "logo.svg", "serviceWorker.js", "setupTests.js", and ".gitignore".
- Code Editor (Top Right):** Shows the "App.js" file with code related to rendering a product list.
- Browser Preview (Bottom Right):** A "React App" window titled "Product List" displays four items: "Apples :: \$3 Stock=9", "Oranges :: \$4 Stock=2", "Beans :: \$2 Stock=5", and "Cabbage :: \$1 Stock=8". Each item has a "Submit" button.
- Cart Preview (Bottom Right):** A "Cart Contents" section lists "Apples :: \$ 3 from Italy" and "Oranges :: \$ 4 from Spain".

```
prod
App.js
src > Components >
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
Product List
Apples :: $3-Stock=9 Submit
Oranges :: $4-Stock=2 Submit
Beans :: $2-Stock=5 Submit
Cabbage :: $1-Stock=8 Submit
Cart Contents
Apples :: $ 3 from Italy
Oranges :: $ 4 from Spain
```

Create React App – Shopping Cart Example (22/25)

The screenshot shows a development environment for a React application. On the left, the VS Code interface displays the project structure and code for `products.jsx`. The browser window on the right shows the application's interface.

VS Code Explorer:

- CARTSOLN01** folder:
 - `node_modules`
 - `public`
 - src** folder:
 - Components** folder
 - `products.jsx` (selected)
 - `App.css`
 - `App.js`
 - `App.test.js`
 - `index.css`
 - `index.js`
 - `logo.svg`
 - `serviceWorker.js`
 - `setupTests.js`
 - `.gitignore`

Browser Preview:

Product List

 - Apples :: \$3-Stock=9
 - Oranges :: \$4-Stock=3
 - Beans :: \$2-Stock=5
 - Cabbage :: \$1-Stock=8

Cart Contents

Apples :: \$ 3 from Italy

CheckOut

Create React App – Shopping Cart Example (23/25)

```
Line 109:53: Expected '!==' and instead saw '!='          eqeqeq
Line 110:56: Expected '===' and instead saw '=='          eqeqeq
Line 112:21: Expected '===' and instead saw '=='          eqeqeq
Line 118:9:   'photos' is assigned a value but never used no-unused-vars
```

```
./src/App.js
  Line 2:8: 'logo' is defined but never used no-unused-vars
```

Search for the [keywords](#) to learn more about each warning.
To ignore, add `// eslint-disable-next-line` to the line before.

```
^C
?130 cartsoln01 % npm install bootstrap
( ( )) :. rollbackFailedOptional: verb npm-session 1488a0bb3a9dbcbe
npm WARN tsutils@3.17.1 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >
dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev
  3.7.0-beta but none is installed. You must install peer dependencies yourself.
npm WARN bootstrap@4.5.2 requires a peer of jquery@1.9.1 - 3 but none is installed.
  st install peer dependencies yourself.
npm WARN bootstrap@4.5.2 requires a peer of popper.js@^1.16.1 but none is installed.
  ust install peer dependencies yourself.

+ bootstrap@4.5.2
updated 1 package and audited 1646 packages in 9.39s
```

Create React App – Shopping Cart Example (24/25)

```
?130 cartsoln01 % npm install bootstrap
((          )) :. rollbackFailedOptional: verb npm-session 1488a0bb3a9dbcbe
npm WARN tsutils@3.17.1 requires a peer of typescript@>=2.8.0 || >= 3.2.0-dev || >=
dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev
  3.7.0-beta but none is installed. You must install peer dependencies yourself.
npm WARN bootstrap@4.5.2 requires a peer of jquery@1.9.1 - 3 but none is installed.
  st install peer dependencies yourself.
npm WARN bootstrap@4.5.2 requires a peer of popper.js@^1.16.1 but none is installed.
  ust install peer dependencies yourself.

+ bootstrap@4.5.2
updated 1 package and audited 1646 packages in 9.39s

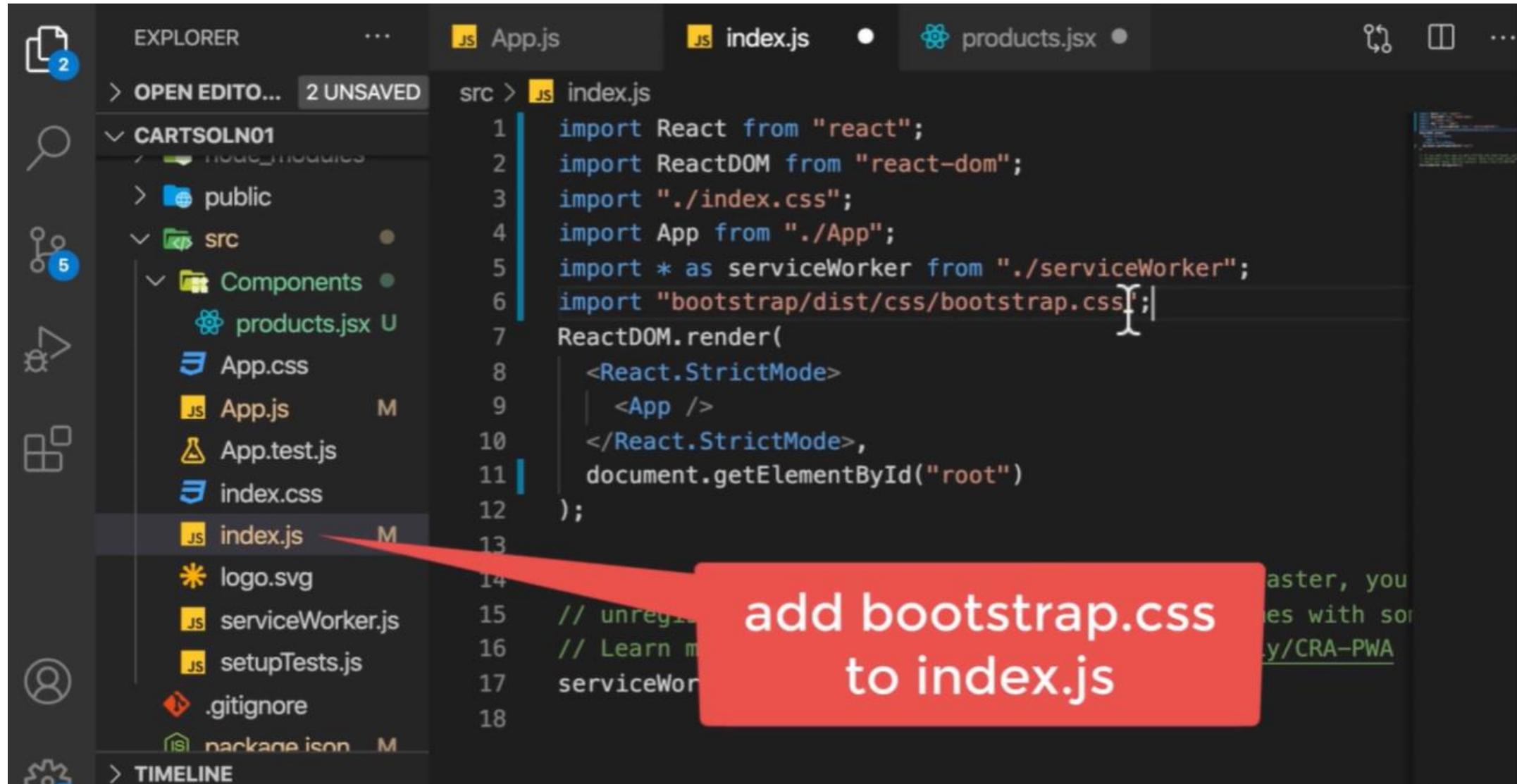
69 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

✓ cartsoln01 %
✓ cartsoln01 % npm start

> cartsoln01@0.1.0 start /Users/johnwilliams/MIT/Courses/React/cartsoln01
> react-scripts start
```

Create React App – Shopping Cart Example (25/25)



The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure with files like `index.js`, `App.css`, `App.test.js`, `index.css`, `index.js` (the current file), `logo.svg`, `serviceWorker.js`, `setupTests.js`, `.gitignore`, and `package.json`.
- tabs**: `App.js`, `index.js`, `products.jsx` (active tab).
- Content Area**:

```
src > index.js
1 import React from "react";
2 import ReactDOM from "react-dom";
3 import "./index.css";
4 import App from "./App";
5 import * as serviceWorker from "./serviceWorker";
6 import "bootstrap/dist/css/bootstrap.css";
7 ReactDOM.render(
8   <React.StrictMode>
9     <App />
10    </React.StrictMode>,
11    document.getElementById("root")
12  );
13
14 // unregis...
15 // Learn m...
16 // serviceWor...
```

A red callout box with the text "add bootstrap.css to index.js" is positioned over the line `import "bootstrap/dist/css/bootstrap.css";`.

Create React App – Shopping Cart Example: Output (1/3)

Product List

Cart Contents

CheckOut

CheckOut \$ 0

Product	Price	Stock
Apples	\$3	10
Oranges	\$4	3
Beans	\$2	5
Cabbage	\$1	8

Create React App – Shopping Cart Example: Output (2/3)

The screenshot shows a web browser window with three tabs:

- React App
- Adding Bootstrap
- localhost:3000

The localhost:3000 tab displays a shopping cart application with the following sections:

Product List

- Apples_:::\$3-Stock=9
Submit
- Oranges:::\$4-Stock=2
Submit
- Beans_:::\$2-Stock=5
Submit
- Cabbage:::\$1-Stock=8

Cart Contents

- Apples_:
- Oranges:
- \$ 4 from Spain

CheckOut

- CheckOut \$ 7
- Apples_:
- Oranges:

Create React App – Shopping Cart Example: Output (3/3)

The screenshot shows a web browser window with three tabs:

- React App**: Shows the React application interface.
- Adding Bootstrap**: Shows the browser's developer tools or a related page.
- localhost:3000**: Shows the running React application.

The application interface consists of three main sections:

- Product List**: A list of products with images and submission buttons.
 - Apples: Image of apples, button text "Apples_:::\$3-Stock=9", "Submit" button.
 - Oranges: Image of oranges, button text "Oranges:::\$4-Stock=3", "Submit" button.
 - Beans: Image of beans, button text "Beans_:::\$2-Stock=5", "Submit" button.
 - Cabbage: Image of a cabbage, button text "Cabbage:::\$1-Stock=8", "Submit" button.
- Cart Contents**: Displays the items in the cart.
 - Apples_: Input field containing "Apples_:".
- CheckOut**: Displays the total and an input field.
 - CheckOut \$ 3
 - Apples_: Input field containing "Apples_:".

Interacting With Static Website (1/8)

The image shows a split-screen view. On the left is a screenshot of a web browser displaying a React-based shopping cart application at `localhost:8080/standalone_15_5`. The page has a header "React Shopping Cart" and navigation links "Product List", "Cart Contents", and "CheckOut". The "CheckOut" section shows a total of \$0. Below it is a "CheckOut \$ 0" button. The main content area lists four products with their details and a "Submit" button:

- Apples:::\$3-Stock=10
- Oranges:::\$4-Stock=3
- Beans:::\$2-Stock=5
- Cabbage:::\$1-Stock=8

At the bottom of the browser window, there are two input fields: "http://localhost:1337/pr" and "ReStock Products".

On the right is a terminal window showing the command `http-server -c1-1` being run in the directory `~/MIT/Courses/React/shopping-cart`. The output shows the server starting and providing two URLs for access: `http://127.0.0.1:8080` and `http://192.168.1.42:8080`. It also indicates that `CTRL-C` can be used to stop the server.

```
~/MIT/Courses/React/shopping-cart — johnwilliams — h
[✓ shopping-cart % http-server -c1-1
Starting up http-server, serving .
Available on:
  http://127.0.0.1:8080
  http://192.168.1.42:8080
Hit CTRL-C to stop the server
[2020-09-01T12:19:06.161Z] "GET /standalone_15_5" AppleWebKit/537.36 (KHTML, like Ge
(node:44358) [DEP0066] DeprecationWarning:
  
```

Interacting With Static Website (2/8)

The screenshot shows a React application titled "React Shopping Cart". The interface is divided into three main sections: "Product List", "Cart Contents", and "CheckOut".

- Product List:** Displays four items with small icons:
 - Apples: \$3 Stock=9
 - Oranges: \$4 Stock=2
 - Beans: \$2 Stock=5
 - Cabbage: \$1 Stock=8
- Cart Contents:** Shows the current items in the cart:
 - Apples: [input field]
 - Oranges: [input field]
- CheckOut:** Displays the total cost and a button:
 - CheckOut \$ 7
 - \$ 4 from Spain

At the bottom, there is a link to "http://localhost:1337/pr" and a "ReStock Products" button.

```
✓ shopping-cart % http-server -c1-1
Starting up http-server, serving .
Available on:
  http://127.0.0.1:8080
  http://192.168.1.42:8080
Hit CTRL-C to stop the server
[2020-09-01T12:19:06.161Z]  "GET /standalo
0_15_5) AppleWebKit/537.36 (KHTML, like Ge
(node:44358) [DEP0066] DeprecationWarning:
  
```

Interacting With Static Website (3/8)

The screenshot illustrates the interaction between a static website (React application) and a local development environment (terminal).

Browser View:

- The browser title bar says "React App".
- The address bar shows "localhost:3000".
- The page content includes:
 - A "Product List" section with four items: Apples, Oranges, Beans, and Cabbage.
 - An "Apples" item is highlighted with a blue box, showing its details:
 - Image: Apples icon.
 - Text: Apples:::\$3-Stock=9
 - Button: Submit
 - An "Oranges" item:
 - Image: Oranges icon.
 - Text: Oranges::\$4-Stock=3
 - Button: Submit
 - An "Beans" item:
 - Image: Beans icon.
 - Text: Beans:::\$2-Stock=5
 - Button: Submit
 - An "Cabbage" item:
 - Image: Cabbage icon.
 - Text: Cabbage::\$1-Stock=8
 - Button: Submit
- A "Cart Contents" section shows:
 - Apples: \$ 3 from Italy
- A "Checkout" section shows:
 - Apples: \$ 3 from Italy

Interacting With Static Website (4/8)

The screenshot shows a React application running at `localhost:3000`. The application has a navigation bar with three items: "Product List", "Cart Contents", and "CheckOut". The "Cart Contents" item is currently selected, indicated by a cursor hovering over it. Below the navigation, there are four product items listed:

- Apples:::\$3-Stock=10
- Oranges:::\$4-Stock=3
- Beans:::\$2-Stock=5
- Cabbage:::\$1-Stock=8

Each item has a small thumbnail image, a description box with price and stock information, and a "Submit" button.

At the bottom of the page, there is a link to `http://localhost:1337/pr` and a button labeled "ReStock Products".

On the right side of the screenshot, the browser's developer tools Network tab is open. The tab shows a list of requests with the following details:

Name	Status	Type
1052	302	
sockjs-node	101	webso...
products	200	xhr
1049.jpg?hmac=L8gliMEgrvLtL...	200	jpeg
1052.jpg?hmac=hfjynpgJ_o9G...	200	jpeg
1050.jpg?hmac=KEPXZjPsolHI...	200	jpeg
1051.jpg?hmac=aWbsQSVkKU...	200	jpeg
favicon.ico	200	xicon

Interacting With Static Website (5/8)

The screenshot illustrates a React application interface and its corresponding network activity in a browser's developer tools.

Product List: Shows four items with images and details:

- Apples:::\$3-Stock=9
- Oranges:::\$4-Stock=2
- Beans:::\$2-Stock=4
- Cabbage:::\$1-Stock=8

Cart Contents: Displays the selected items from the product list:

- Apples_:
- Oranges:
- Beans_:

CheckOut: Shows the total cost and quantity of items in the cart.

Network Tab (DevTools): Shows the following network requests:

Name	Status	Type
sockjs-node	101	websocket
products	200	xhr
1049.jpg?hmac=L8gliMEgrvLtL...	200	jpeg
1052.jpg?hmac=hfjynpgJ_o9G...	200	jpeg
1050.jpg?hmac=KEPXZjPsolHI...	200	jpeg
1051.jpg?hmac=aWbsQSVkKU...	200	jpeg
favicon.ico	200	x-icon
manifest.json	200	manifest

Interacting With Static Website (6/8)

The screenshot displays a web browser window with the title "React App" at the top. The address bar shows "localhost:3000". The main content area contains three sections: "Product List", "Cart Contents", and "CheckOut".

- Product List:** Contains four items with images and descriptions:
 - Apples::\$3-Stock=9
 - Oranges::\$4-Stock=2
 - Beans::\$2-Stock=4
 - Cabbage::\$1-Stock=8
- Cart Contents:** Shows the items added to the cart:
 - Apples_:
 - Oranges:
 - Beans_:
- CheckOut:** Displays a total of \$9 and a "CheckOut" button.

At the bottom left, there is a link to "http://localhost:1337/pr" and a "ReStock Products" button. On the right, the browser's developer tools are open, specifically the "Elements" tab, showing the HTML code for the page structure.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  ... <body> == $0
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      <div class="container">
        <div class="row">
          <div class="col">
            <h1>Product List</h1>
            <ul style="list-style-type: none;">...</ul>
          </div>
          <div class="col">
            <h1>Cart Contents</h1>
            <div class="accordion">
              <div class="card">...</div>
              <div class="card">...</div>
              <div class="card">
                <div class="card-header">...</div>
                <div class="collapse">...</div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Interacting With Static Website (7/8)

The screenshot shows a static website interface with three main sections: Product List, Cart Contents, and CheckOut.

- Product List:** Displays four items with images and details:
 - Apples:::\$3-Stock=9
 - Oranges:::\$4-Stock=2
 - Beans:::\$2-Stock=5
 - Cabbage:::\$1-Stock=8
- Cart Contents:** Shows the current items in the cart:
 - Apples:::
 - Oranges:::
- CheckOut:** A button labeled "CheckOut \$ 7".

The browser's developer tools are open, showing the HTML structure of the page. The code includes components like `<div class="row">`, `<div class="col">`, and `<h1>Product List</h1>`.

```
<div class="row">
  <div class="col">
    <h1>Product List</h1>
    <ul style="list-style-type: none;">...</ul>
  </div>
  <div class="col">
    <h1>Cart Contents</h1>
    <div class="accordion">
      <div class="card">...</div>
      <div class="card">...</div>
    </div>
    <div class="col">...</div>
  </div>
</div>
<!--
  This HTML file is a template.
  If you open it directly in the browser,
  empty page.
-->
```

Interacting With Static Website (8/8)

React App

localhost:3000

Product List

Apples::\$3-Stock=9

Submit

Oranges::\$4-Stock=3

Submit

Beans::\$2-Stock=5

Submit

Cabbage::\$1-Stock=8

Submit

ReStock Products

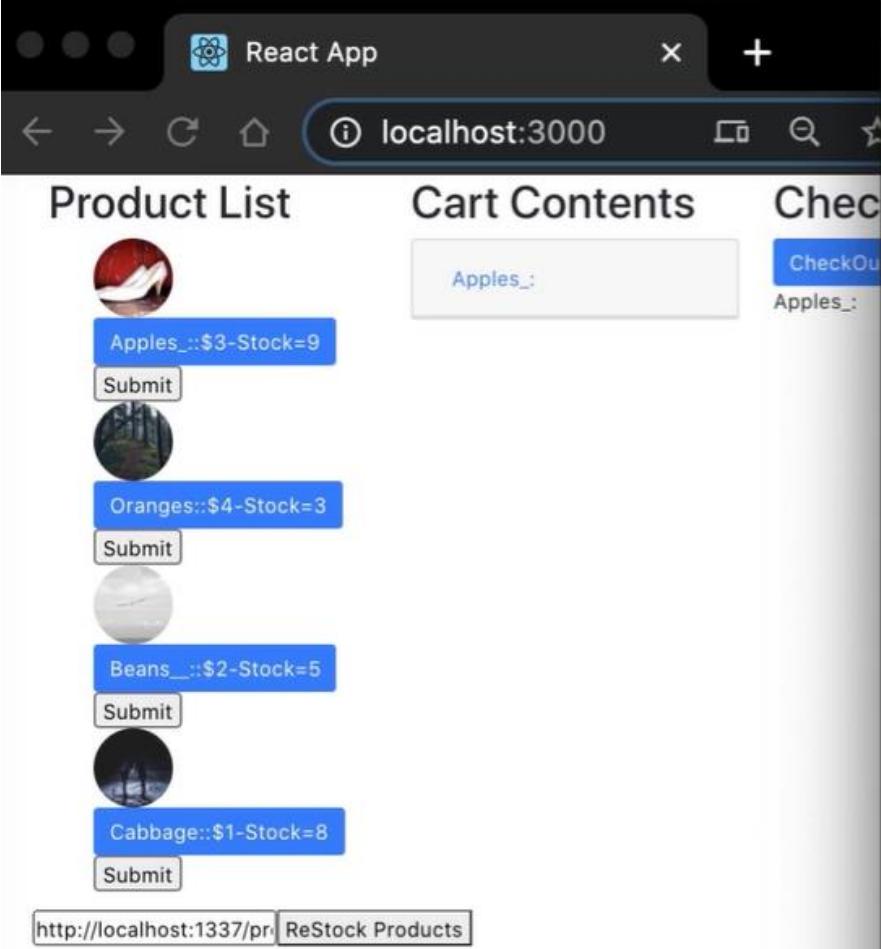
DevToo

Network Console Elements Sources

```
<div class="row">
  <div class="col">
    <h1>Product List</h1>
    <ul style="list-style-type: none;">...</ul>
  </div>
  <div class="col">
    <h1>Cart Contents</h1>
    <div class="accordion">
      <div class="card">...</div>
    </div>
    <div class="col">...</div>
  </div>
</div>
<!--
  This HTML file is a template.
  If you open it directly in the browser,
  empty page.

  You can add webfonts, meta tags, or ana
  file-->
```

Building Static Website (1/7)

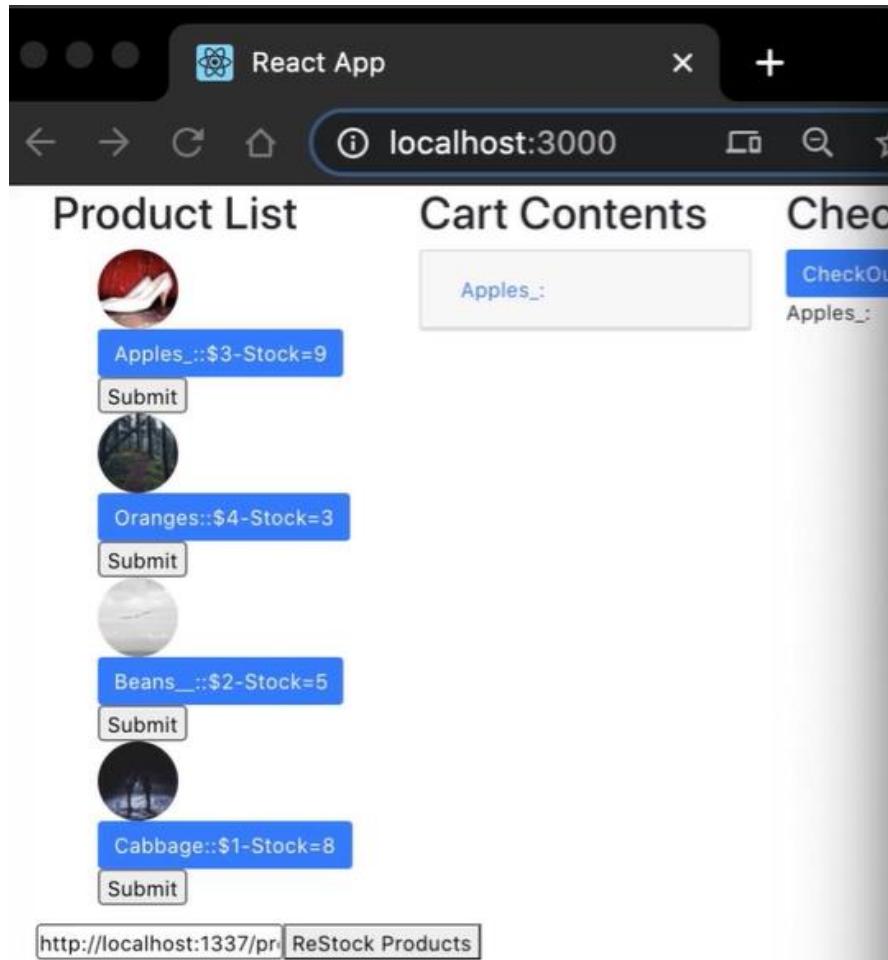


The screenshot shows a React application running at `localhost:3000`. The interface includes a navigation bar with 'Product List', 'Cart Contents', and 'CheckOut' buttons. Below the navigation, there are four product items: Apples (\$3, Stock=9), Oranges (\$4, Stock=3), Beans (\$2, Stock=5), and Cabbage (\$1, Stock=8). Each item has a small image, a name, price, stock count, and a 'Submit' button.

```
Line 90:18:  'isLoading' is assigned a value b
Line 90:29:  'isError' is assigned a value but
Line 100:49: Expected '===' and instead saw '='
Line 101:25: Expected '===' and instead saw '='
Line 109:53: Expected '!==' and instead saw '!'
Line 110:56: Expected '===' and instead saw '='
Line 112:21: Expected '===' and instead saw '='
Line 118:9:  'photos' is assigned a value but
./src/App.js
Line 2:8:  'logo' is defined but never used  no
Search for the keywords to learn more about each
To ignore, add // eslint-disable-next-line to the
^C
?130 cartsoln01 % npm run build
> cartsoln01@0.1.0 build /Users/johnwilliams/MIT/
> react-scripts build
Creating an optimized production build...
```

The terminal window shows ESLint errors for `./src/App.js`, specifically regarding the use of `isLoading` and `isError` without assignment operators, and the use of `!==` instead of `!==`. It also shows the command `npm run build` being run, which is part of the static website build process.

Building Static Website (2/7)



```
~ /MIT/Courses/React/cartsoln01 — johnwi
22.5 KB    build/static/css/2.8aa5a7f8.chunk.css
1.99 KB   build/static/js/main.9c83b749.chunk.j
774 B      build/static/js/runtime-main.83451c3c
547 B      build/static/css/main.5f361e03.chunk.

The project was built assuming it is hosted at /.
You can control this with the homepage field in y

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:
  bit.ly/CRA-deploy

√ cartsoln01 % ls
 README.md          package-lock.json
 build              package.json
 node_modules       public
```

Building Static Website (3/7)

The image shows a split-screen view. On the left, a web browser window titled "React App" displays a product management application at "localhost:3000". The interface includes a "Product List" tab, a "Cart Contents" tab, and a "CheckOut" button. Under "Product List", there are four items: "Apples:::\$3-Stock=9", "Oranges:::\$4-Stock=3", "Beans:::\$2-Stock=5", and "Cabbage:::\$1-Stock=8", each with a "Submit" button. At the bottom, there is a link "http://localhost:1337/pr" and a "ReStock Products" button. On the right, a terminal window titled "~/MIT/Courses/React/cartsoln01/build — john" shows the command "serve -s build" and deployment instructions. It then lists the contents of the "build" directory, which includes "README.md", "build", "node_modules", "package-lock.json", "package.json", "public", "index.html", "asset-manifest.json", "favicon.ico", "logo192.png", "logo512.png", "manifest.json", "precache-manifest.4d3f05a3d0bfea316e595541031a2c2", "robots.txt", "service-worker.js", and "static".

```
serve -s build
Find out more about deployment here:
bit.ly/CRA-deploy
[✓ cartsoln01 % ls
 README.md          package-lock.json
 build              package.json
 node_modules       public
[✓ cartsoln01 % cd build
[✓ build % ls
 asset-manifest.json
 favicon.ico
 index.html
 logo192.png
 logo512.png
 manifest.json
 precache-manifest.4d3f05a3d0bfea316e595541031a2c2
 robots.txt
 service-worker.js
 static
```

Building Static Website (4/7)

The image shows a split-screen view. On the left is a screenshot of a web browser window titled "React App" displaying a product list. The products are listed as follows:

- Apples:::\$3-Stock=9
- Oranges:::\$4-Stock=3
- Beans:::\$2-Stock=5
- Cabbage:::\$1-Stock=8

Each item has a small image, a price, a stock count, and a "Submit" button. At the bottom of the page are two buttons: "http://localhost:1337/pr" and "ReStock Products".

On the right is a terminal window with the following session:

```
~ /MIT/Courses/React/cartsoln01/build — johnwilliams — http-server
[✓ cartsoln01 % ls
 README.md
 build
 node_modules
 package-lock.json
 package.json
 public
 [✓ cartsoln01 % cd build
 [✓ build % ls
 asset-manifest.json
 favicon.ico
 index.html
 logo192.png
 logo512.png
 manifest.json
 precache-manifest.4d3f05a3d0bfea316e595541031a2c2
 robots.txt
 service-worker.js
 static
 [✓ build % http-server
 Starting up http-server, serving .
 Available on:
 http://127.0.0.1:8081
 http://192.168.1.42:8081
 Hit CTRL-C to stop the server
```

Building Static Website (5/7)

The image shows a Mac desktop with three windows:

- A top-level window titled "React App" containing two tabs: "Product List" and "CheckOut".
- The "Product List" tab displays a list of products with images and stock information:
 - Apples:::\$3-Stock=5
 - Oranges:::\$4-Stock=3
 - Beans:::\$2-Stock=5
 - Cabbage:::\$1-Stock=8
- The "CheckOut" tab shows a summary:
 - CheckOut \$ 3
 - Apples:::
- A bottom-level window titled "ReStock Products" with a single input field: "http://localhost:1337/pr".
- A terminal window on the right showing the command-line interface for building and running the application.

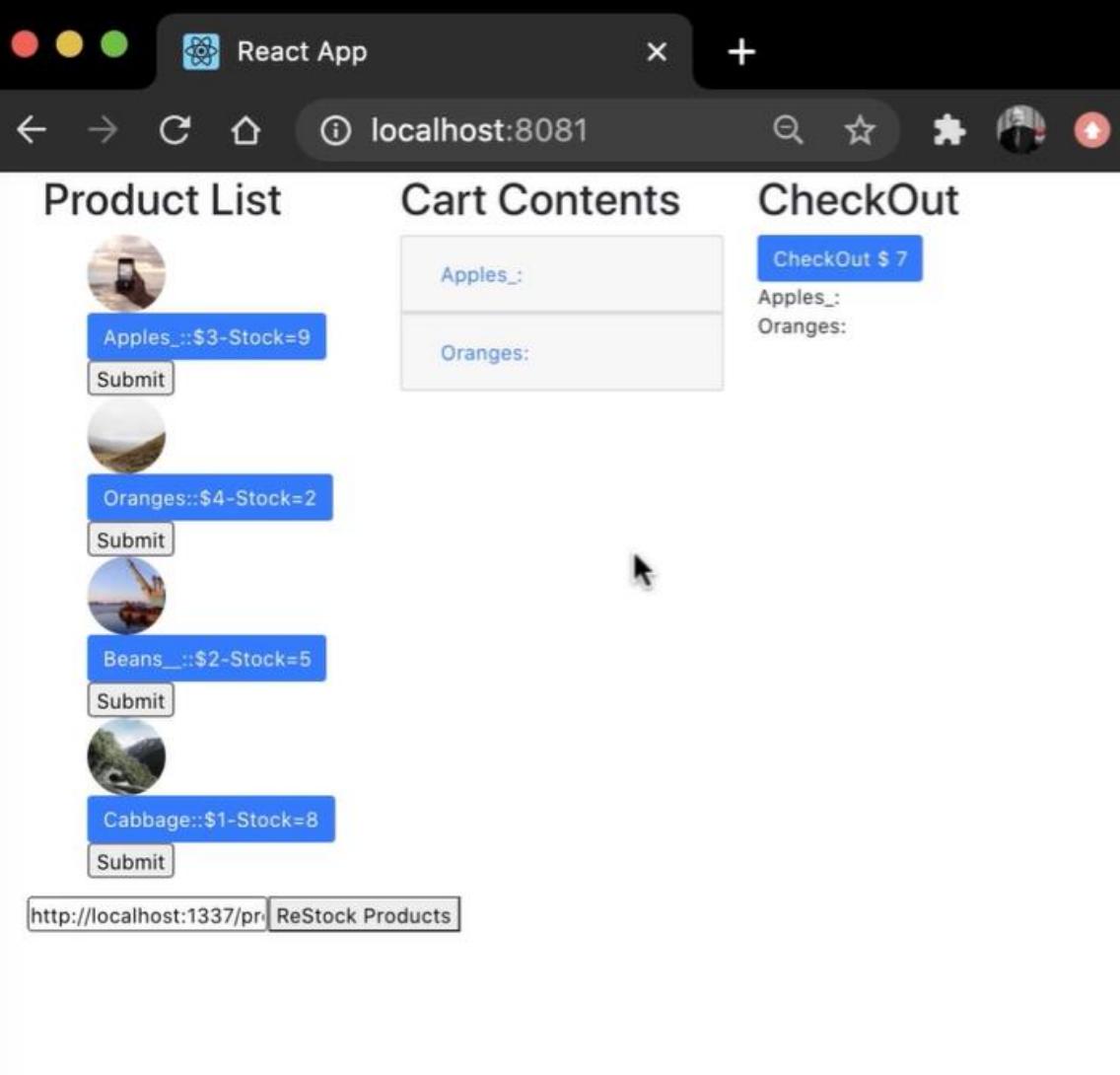
Terminal Command Line (right side):

```
~ /MIT/Courses/React/cartsoIn01/build — johnwilliams — http-server
1 % ls
package-lock.json
package.json
public
s
1 % cd build
s
est.json

on
nifest.4d3f05a3d0bfea316e595541031a2c2
ker.js

http-server
http-server, serving ./
n:
7.0.0.1:8081
2.168.1.42:8081
to stop the server
```

Building Static Website (6/7)



The screenshot shows a React application running on localhost:8081. The interface includes a Product List section with four items: Apples (\$3, Stock=9), Oranges (\$4, Stock=2), Beans (\$2, Stock=5), and Cabbage (\$1, Stock=8). Each item has a small image, a price, stock count, and a 'Submit' button. To the right is a Cart Contents section showing 'Apples: 1' and 'Oranges: 1'. Below that is a CheckOut section with a total of '\$7' and a 'CheckOut' button. A terminal window to the right displays the command-line logs for the application's server, showing various GET requests for static files like CSS and JS, and manifest.json.

```
~/MIT/Courses/React/carts01/build — johnwilliams — http-server
7.36"
T12:24:13.792Z] "GET /static/js/main.js" "Mozilla/5.0 (Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
T12:24:13.914Z] "GET /static/js/2.early.js" "Mozilla/5.0 (Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
T12:24:13.915Z] "GET /static/js/main.js" "Mozilla/5.0 (Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
T12:24:13.937Z] "GET /static/css/2.8a.css" "Mozilla/5.0 (Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
T12:24:13.938Z] "GET /static/css/main.css" "Mozilla/5.0 (Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
T12:24:15.163Z] "GET /favicon.ico" "Mozilla/5.0 (Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
T12:24:15.164Z] "GET /manifest.json" "Mozilla/5.0 (Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
T12:24:15.171Z] "GET /logo192.png" "Mozilla/5.0 (Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.131 Safari/537.36"
```

Building Static Website (7/7)

The image shows a split-screen view. On the left is a screenshot of a web browser displaying a React application. The title bar says "React App" and the address bar says "localhost:8081". The page has three sections: "Product List", "Cart Contents", and "CheckOut". The "Product List" section shows four items with images and stock information: "Apples::\$3-Stock=9", "Oranges::\$4-Stock=2", "Beans::\$2-Stock=5", and "Cabbage::\$1-Stock=8", each with a "Submit" button. The "Cart Contents" section shows "Apples:" and "Oranges:". The "CheckOut" section has a "Checkout" button. At the bottom, there are two links: "http://localhost:1337/pr" and "ReStock Products". On the right is a terminal window with the following text:

```
~ /MIT/Courses/React/cartsoln01/build — john
147.125 Safari/537.36"
[2020-09-01T12:24:13.938Z] "GET /static/css/main
(Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/
0.4147.125 Safari/537.36"
[2020-09-01T12:24:15.163Z] "GET /favicon.ico" "M
10_15_5) AppleWebKit/537.36 (KHTML, like Gecko)
[2020-09-01T12:24:15.164Z] "GET /manifest.json"
X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko)
[2020-09-01T12:24:15.171Z] "GET /logo192.png" "M
10_15_5) AppleWebKit/537.36 (KHTML, like Gecko)
^Chttp-server stopped.
[✓ build % ls
asset-manifest.json
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
precache-manifest.4d3f05a3d0bfea316e595541031a2c2
robots.txt
service-worker.js
static
```

Deploying Static Website On AWS3 (1/28)

The screenshot shows a web browser window with three tabs: "React App", "Settings", and a blank tab. The main content area displays a React application for managing a grocery cart.

Product List

- Apples::\$3-Stock=9
- Oranges::\$4-Stock=2
- Beans::\$2-Stock=4
- Cabbage::\$1-Stock=8

Each item has a "Submit" button below it.

Cart Contents

- Apples:: \$ 3 from Italy
- Oranges:
- Beans::

CheckOut

CheckOut \$ 9

- Apples::
- Oranges:
- Beans::

<http://localhost:1337/prc> ReStock Products

Deploying Static Website On AWS3 (2/28)

```
bit.ly/CRA-deploy

[✓ cartsoln01 % ls
README.md          package-lock.json      src
build              package.json
node_modules       public
[✓ cartsoln01 % ls build
asset-manifest.json
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
precache-manifest.4d3f05a3d0bfea316e595541031a2c2c.js
robots.txt
service-worker.js
static
[✓ cartsoln01 % npm run build
> cartsoln01@0.1.0 build /Users/johnwilliams/MIT/Courses/React/cartsoln01
> react-scripts build

Creating an optimized production build...
```

Deploying Static Website On AWS3 (3/28)

```
You may serve it with a static server:
```

```
npm install -g serve  
serve -s build
```

```
Find out more about deployment here:
```

```
bit.ly/CRA-deploy
```

```
└─ cartsoln01 % ls  
  README.md          package-lock.json      src  
  build              package.json           public  
  node_modules  
└─ cartsoln01 % ls build  
  asset-manifest.json  
  favicon.ico  
  index.html  
  logo192.png  
  logo512.png  
  manifest.json  
  precache-manifest.4d3f05a3d0bfea316e595541031a2c2c.js  
  robots.txt  
  service-worker.js  
  static  
└─ cartsoln01 %
```

Deploying Static Website On AWS3 (4/28)

The screenshot shows the AWS S3 Management Console interface. The top navigation bar includes tabs for 'S3 Management Console', 'React App', and 'Settings'. The main header features the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, user profile 'John R. Williams', and 'Global' and 'Support' dropdowns. A search bar labeled 'Search for buckets' and a filter 'All access types' are present. On the left, a sidebar titled 'Amazon S3' lists 'Buckets', 'Batch operations', 'Access analyzer for S3', 'Block public access (account settings)', and 'Feature spotlight' (with a '2' badge). The central area displays bucket statistics: '10 Buckets' and '1 Regions'. Below this, two buckets are listed:

Bucket Name	Region	Last Modified	Created
sagemaker-us-east-1-048416854...	US East (N. Virginia)	Oct 11, 2019 4:27:21 PM GMT-0400	2019 Mar 28, 2020 10:27:31 AM GMT-0400
sparkdemo001	US East (N. Virginia)		

Deploying Static Website On AWS3 (5/28)

The screenshot shows the AWS Management Console interface. The top navigation bar includes tabs for "S3 Management Console" (selected), "React App", and "Settings". Below the navigation bar is a search bar with the URL "s3.console.aws.amazon.com/s3/home?region=us-east-1". The main menu bar features "Services" (with a dropdown arrow), "Resource Groups" (with a dropdown arrow), and user information for "John R. Williams".

The left sidebar has a "History" section and a "Storage" category under "Services". The "Storage" category includes links for S3, EFS, FSx, S3 Glacier, Storage Gateway, and AWS Backup.

The main content area displays a grid of service icons and names. The services listed are:

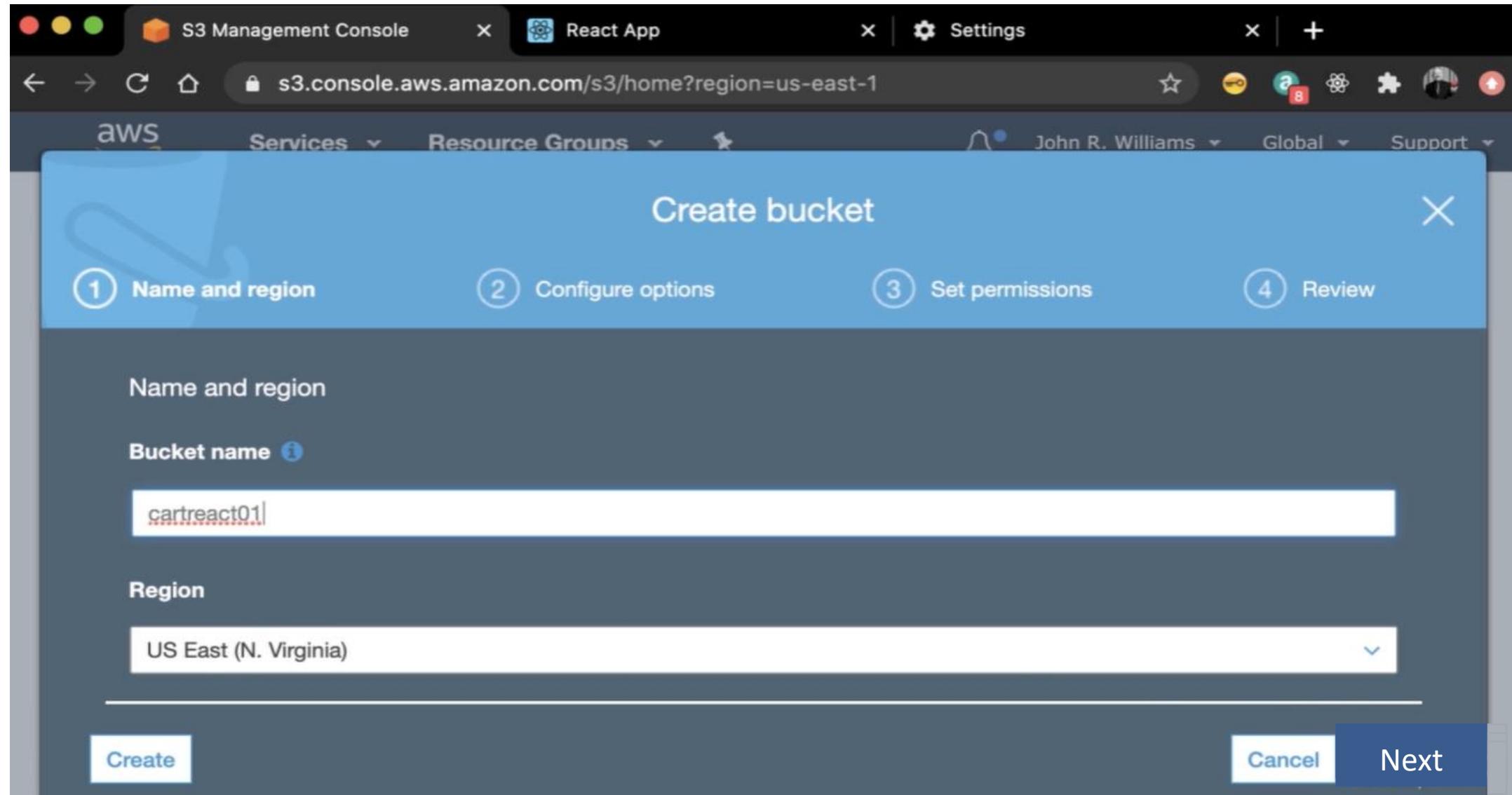
Category	Service	Workload
Storage	AWS Outposts	Workload
	EC2 Image Builder	AppSync
	S3	Amazon Braket
	EFS	AWS Data Exchange
Management & Governance	Quantum Technologies	Workload
	CloudWatch	AWS Glue
	AWS Auto Scaling	AWS Lake Formation
	CloudFormation	MSK
Security, Identity, & Compliance	IAM	Interactions
	CloudTrail	Resource Access Manager
	Config	Cognito
	OpsWorks	Secrets Manager
AWS Data Exchange	GuardDuty	IoT Device
	CloudWatch Metrics	IoT Analytics
	CloudWatch Metrics Insights	IoT Device
	CloudWatch Metrics Insights Insights	IoT Device

Deploying Static Website On AWS3 (6/28)

The screenshot shows the AWS S3 Management Console interface. The left sidebar has 'Buckets' selected. The main area is titled 'S3 buckets' and features a search bar and filters for 'All access types'. A prominent blue button labeled '+ Create bucket' is visible. Below it, a summary shows 10 Buckets and 1 Region. A table lists one bucket: 'aws-logs-048416854569-us-east-1'. The table includes columns for Bucket name, Access, Region, and Date created. The date listed is Mar 28, 2020, at 4:22:00 PM GMT-0400. The 'Access' column indicates that objects can be public.

Bucket name	Access	Region	Date created
aws-logs-048416854569-us-east-1	Objects can be public	US East (N. Virginia)	Mar 28, 2020 4:22:00 PM GMT-0400

Deploying Static Website On AWS3 (7/28)



Deploying Static Website On AWS3 (8/28)

The screenshot shows a web browser window with three tabs: "S3 Management Console", "React App", and "Settings". The address bar displays the URL s3.console.aws.amazon.com/s3/home?region=us-east-1. The main content is a modal dialog titled "Create bucket" with four steps: 1. Name and region (completed), 2. Configure options (current step), 3. Set permissions, 4. Review. The "Configure options" tab is active. The "Properties" section contains three groups: "Versioning", "Server access logging", and "Tags". Under "Versioning", there is a checkbox for "Keep all versions of an object in the same bucket." Under "Server access logging", there is a checkbox for "Log requests for access to your bucket." Under "Tags", there is a note: "You can use tags to track project costs." At the bottom right of the modal are "Previous" and "Next" buttons.

Create bucket

① Name and region ② Configure options ③ Set permissions ④ Review

Properties

Versioning

Keep all versions of an object in the same bucket. [Learn more](#)

Server access logging

Log requests for access to your bucket. [Learn more](#)

Tags

You can use tags to track project costs. [Learn more](#)

Previous Next

Deploying Static Website On AWS3 (9/28)

The screenshot shows a web browser window with three tabs: "S3 Management Console", "React App", and "Settings". The main content area is titled "Create bucket" and is on step 3: Set permissions. The steps are numbered 1 through 4: 1. Name and region, 2. Configure options, 3. Set permissions (which is active), and 4. Review. Step 3 has a sub-section titled "Block all public access" which is currently selected. It contains four checkboxes:

- Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
- Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through new public bucket or access point policies**

At the bottom right are "Previous" and "Next" buttons. The "Next" button is highlighted in blue.

Deploying Static Website On AWS3 (10/28)

The screenshot shows the AWS S3 Management Console interface. The top navigation bar includes tabs for 'S3 Management Console', 'React App', and 'Settings'. The main title is 'Create bucket'. The current step is 'Set permissions', indicated by a blue header bar with the number '3'. There are four steps in total: 'Name and region', 'Configure options', 'Set permissions', and 'Review'. A warning message in an orange box states: '⚠️ Disabling Block all public access may result in this bucket and the objects within becoming public. AWS recommends that you block all public access to your bucket, unless public access is required for specific and verified use cases such as static website hosting.' Below this, there is a checked checkbox: 'I acknowledge that the current settings may result in this bucket and the objects within becoming public'. At the bottom, there is a section titled 'Block all public access' with a note: 'Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.' Navigation buttons at the bottom right include 'Previous' and 'Next'.

Create bucket

Name and region Configure options 3 Set permissions 4 Review

⚠️ Disabling Block all public access may result in this bucket and the objects within becoming public. AWS recommends that you block all public access to your bucket, unless public access is required for specific and verified use cases such as static website hosting.

I acknowledge that the current settings may result in this bucket and the objects within becoming public

Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Previous Next

Deploying Static Website On AWS3 (11/28)

The screenshot shows the AWS S3 Management Console interface. The browser tabs include "S3 Management Console", "React App", and "Settings". The address bar shows the URL s3.console.aws.amazon.com/s3/home?region=us-east-1. The main content area is a "Create bucket" wizard with four steps: "Name and region", "Configure options", "Set permissions", and "Review". The "Review" step is currently active, indicated by a blue header bar with the number "4".

Create bucket

4 Review

Name and region

Bucket name: cartreact01 Region: US East (N. Virginia) [Edit](#)

Options

Versioning: Disabled [Edit](#)

Server access logging: Disabled [Edit](#)

Tagging: 0 Tags [Edit](#)

[Previous](#) [Create bucket](#)

Deploying Static Website On AWS3 (12/28)

The screenshot shows the AWS S3 Management Console interface. The top navigation bar includes tabs for 'S3 Management Console', 'React App', 'Settings', and others. The main header displays the URL 's3.console.aws.amazon.com/s3/home?region=us-east-1' and the user 'John R. Williams'. The left sidebar, titled 'Amazon S3', has a 'Buckets' section selected, along with links for 'Batch operations', 'Access analyzer for S3', 'Block public access (account settings)', and 'Feature spotlight' (with a '2' notification). The central content area shows a search bar and filters for 'All access types'. A prominent blue button labeled '+ Create bucket' is visible. Below it, the statistics '11 Buckets' and '1 Regions' are displayed. A table lists three buckets: 'aws-logs-048416854569-us-east-1', 'cart-react01' (which is currently selected), and 'cdktoolkit-stagingbucket-4qt4ad...'. Each row provides details such as object visibility, region, and creation date.

Bucket	Objects can be public	Region	Last Modified
aws-logs-048416854569-us-east-1	US East (N. Virginia)	2020-08-31 4:22:00 PM GMT-0400	
cart-react01	US East (N. Virginia)	2020-08-31 5:57:50 PM GMT-0400	
cdktoolkit-stagingbucket-4qt4ad...	US East (N. Virginia)	2019-11-08 1:09:23 PM GMT-0500	

Deploying Static Website On AWS3 (13/28)

The screenshot shows the AWS S3 Management Console interface. The top navigation bar includes tabs for 'S3 Management Console' (active), 'React App', and 'Settings'. The URL in the address bar is `s3.console.aws.amazon.com/s3/buckets/cartreact01?region=us-east-1&tab=p...`. The main navigation bar features 'aws', 'Services', 'Resource Groups', and user information for 'John R. Williams'. Below this, the breadcrumb navigation shows 'Amazon S3 > cartreact01'. The main content area displays the 'cartreact01' bucket details. A navigation bar at the top of the bucket page includes 'Overview' (selected), 'Properties', 'Permissions' (active), 'Management', and 'Access points'. Two callout boxes are present: one for 'Versioning' (described as keeping multiple versions of an object in the same bucket) and one for 'Server access logging' (described as setting up access log records). Both boxes include a 'Learn more' link.

Amazon S3 > cartreact01

cartreact01

Overview Properties Permissions Management Access points

Versioning
Keep multiple versions of an object in the same bucket.
[Learn more](#)

Server access logging
Set up access log records that provide details about access requests.
[Learn more](#)

Deploying Static Website On AWS3 (14/28)

The screenshot shows the AWS S3 Management Console interface. At the top, there are tabs for "S3 Management Console", "React App", and "Settings". The main navigation bar includes "Services", "Resource Groups", "John R. Williams", "Global", and "Support". Below the navigation, there are several sections:

- Static website hosting**: Described as "Host a static website, which does not require server-side technologies." A "Learn more" link and a "Disabled" status indicator are present.
- Object-level logging**: Described as "Record object-level API activity using the CloudTrail data events feature (additional cost)." A "Learn more" link and a "Disabled" status indicator are present.

Deploying Static Website On AWS3 (15/28)

The screenshot shows the AWS S3 Management Console interface for static website hosting. The browser tabs include "S3 Management Console", "React App", and "Settings". The URL in the address bar is s3.console.aws.amazon.com/s3/buckets/cartreact01?region=us-east-1&tab=publicWebsite. The main content area is titled "Static website hosting" and displays the following configuration:

- Endpoint : <http://cartreact01.s3-website-us-east-1.amazonaws.com>
- Use this bucket to host a website [Learn more](#)
- Index document [i](#): index.html
- Error document [i](#): error.html
- Redirection rules (optional) [i](#)

Deploying Static Website On AWS3 (16/28)

The screenshot shows the AWS S3 Management Console interface for configuring a static website. The top navigation bar includes tabs for 'S3 Management Console', 'React App', 'Settings', and a '+' icon. The main content area is titled 'Error document' with a help icon. A text input field contains the value 'error.html'. Below this is a section for 'Redirection rules (optional)' with an empty text area. At the bottom, there are three radio button options: 'Redirect requests' (with a 'Learn more' link), 'Disable website hosting', and 'Disabled' (which is selected). Two buttons at the bottom right are 'Cancel' and 'Save', with 'Save' being highlighted.

Deploying Static Website On AWS3 (17/28)

The screenshot shows the AWS S3 Management Console interface. At the top, there are tabs for "S3 Management Console", "React App", and "Settings". The main navigation bar includes "Services", "Resource Groups", and user information for "John R. Williams". Below the navigation, there are two main sections: "Static website hosting" and "Object-level logging".

Static website hosting

Host a static website, which does not require server-side technologies.

[Learn more](#)

Bucket hosting

Object-level logging

Record object-level API activity using the CloudTrail data events feature (additional cost).

[Learn more](#)

Disabled

Deploying Static Website On AWS3 (18/28)

The screenshot shows the AWS S3 Management Console interface. At the top, there are three tabs: 'S3 Management Console', 'React App', and 'Settings'. Below the tabs, the URL bar displays the address: `s3.console.aws.amazon.com/s3/buckets/cartreact01?region=us-east-1&tab=p...`. The main navigation bar includes the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, user profile 'John R. Williams', 'Global' dropdown, and 'Support' dropdown.

In the breadcrumb navigation, it shows 'Amazon S3 > cartreact01'. The bucket name 'cartreact01' is displayed prominently below the breadcrumb.

The top navigation bar has five tabs: 'Overview', 'Properties', 'Permissions', 'Management', and 'Access points'. The 'Management' tab is currently selected, indicated by a blue background and white text.

Below the tabs, there are four buttons: 'Block public access' (highlighted in blue), 'Access Control List', 'Bucket Policy', and 'CORS configuration'. The 'Block public access' button is described as being under 'bucket settings'.

The main content area contains the following text:

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level

Deploying Static Website On AWS3 (19/28)

The screenshot shows the AWS S3 Management Console interface. The top navigation bar includes tabs for 'S3 Management Console', 'React App', and 'Settings'. The main content area has tabs for 'Block public access', 'Access Control List', 'Bucket Policy' (which is highlighted in blue), and 'CORS configuration'. Below these tabs, the title 'Bucket policy editor' is followed by the ARN: arn:aws:s3:::cartreact01. A text input field says 'Type to add a new policy or edit an existing policy in the text area below.' To the right are three buttons: 'Delete', 'Cancel', and 'Save' (in blue). The main area contains a code editor with a numbered JSON policy document:

```
1  "Version": "2012-10-17",
2  "Statement": [
3      {
4          "Sid": "PublicReadGetObject",
5          "Effect": "Allow",
6          "Principal": "*",
7          "Action": [
8              "s3:GetObject"
9          ],
10         "Resource": [
11             "arn:aws:s3:::example.com/*"
12         ]
13     }
14   ]
15 }
16 }
```

Deploying Static Website On AWS3 (20/28)

The screenshot shows the AWS S3 Management Console interface. The top navigation bar includes tabs for 'S3 Management Console', 'React App', and 'Settings'. The main content area is titled 'Bucket policy editor' for the ARN: arn:aws:s3:::cartreact01. It displays a JSON-based policy document:

```
1 "Version": "2012-10-17",
2 "Statement": [
3     {
4         "Sid": "PublicReadGetObject",
5         "Effect": "Allow",
6         "Principal": "*",
7         "Action": [
8             "s3:GetObject"
9         ],
10        "Resource": [
11            "arn:aws:s3:::cartreact01/*"
12        ]
13    }
14 ]
15 }
16 }
```

Below the code editor, there are three buttons: 'Delete', 'Cancel', and 'Save'. At the bottom of the page, there are links for 'Documentation' and 'Policy generator'.

Deploying Static Website On AWS3 (21/28)

This bucket has public access

You have provided public access to this bucket. We highly recommend that you do not provide public access to your S3 bucket.

Bucket policy editor ARN: arn:aws:s3:::cartreact01

Type to add a new policy or edit an existing policy in the text area below.

```
1 "Version": "2012-10-17",
2 "Statement": [
3     {
4         "Sid": "PublicReadGetObject",
5         "Effect": "Allow",
6         "Principal": "*",
7         "Action": [
8             "s3:GetObject"
9         ],
10        "Resource": [
11            "arn:aws:s3:::cartreact01/*"
12        ]
13    }
14 }
```

your bucket name
eg cartreact01

Deploying Static Website On AWS3 (22/28)

The screenshot shows the AWS S3 Management Console interface. At the top, there are three tabs: "S3 Management Console" (active), "React App", and "Settings". Below the tabs, the URL is `s3.console.aws.amazon.com/s3/buckets/cartreact01?region=us-east-1&tab=o...`. The AWS logo and navigation links for "Services", "Resource Groups", and "John R. Williams" are visible. A banner at the bottom of the page reads "This bucket is empty. Upload new objects to get started."

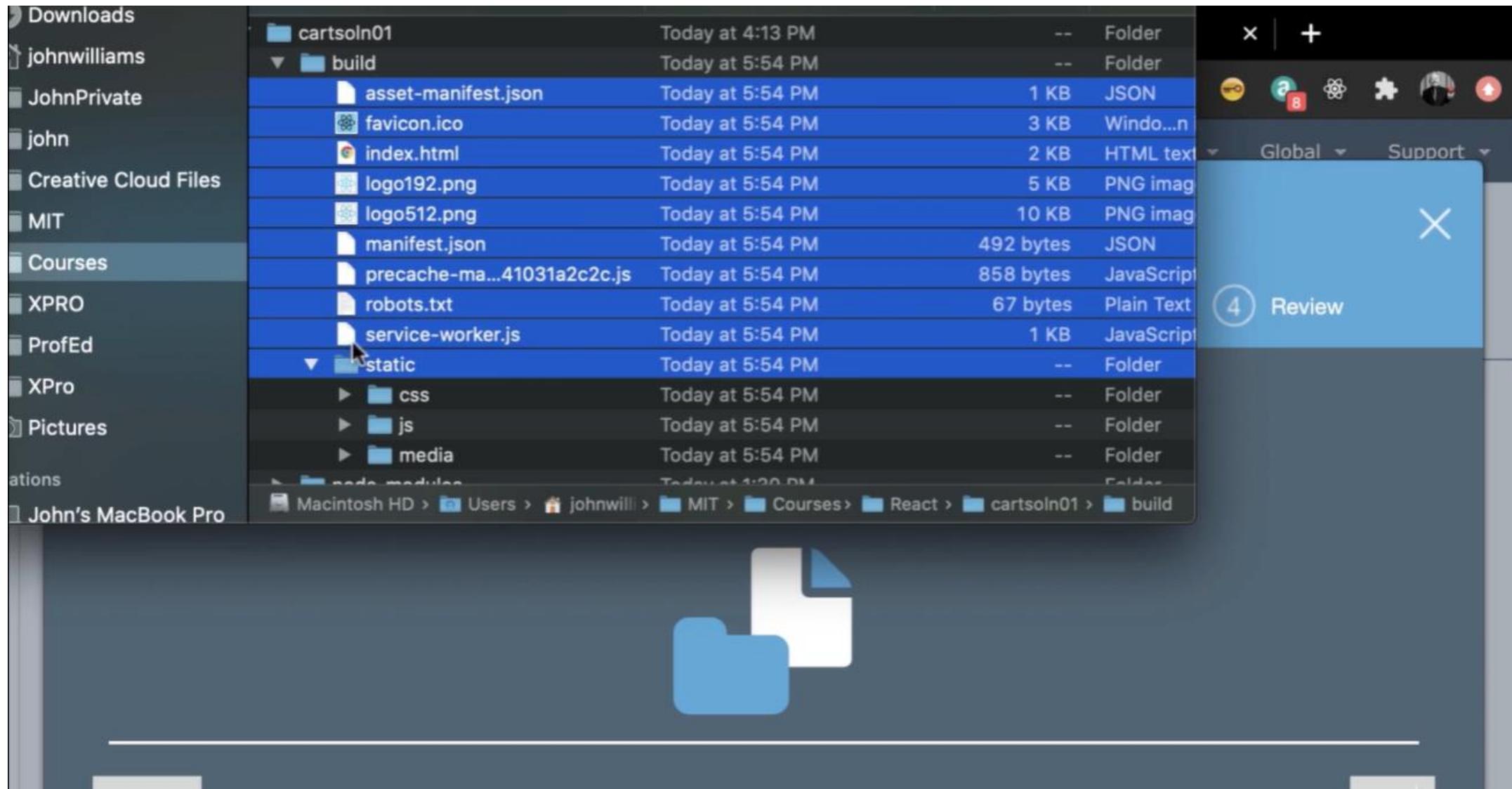
cartreact01

Overview Properties Permissions Management Access points

Upload Create folder Download Actions US East (N. Virginia) ↻

This bucket is empty. Upload new objects to get started.

Deploying Static Website On AWS3 (23/28)



Deploying Static Website On AWS3 (24/28)

The screenshot shows the AWS S3 Management Console interface for uploading files to a bucket named "cartreact01". The top navigation bar includes tabs for "S3 Management Console", "React App", "Settings", and a "+" icon. The main area is titled "Upload" and displays four steps: 1. Select files, 2. Set permissions, 3. Set properties, and 4. Review. Below these steps, it shows "21 Files" with a total "Size: 1.3 MB" and a "Target path: cartreact01". A note indicates that for larger files, the AWS CLI, AWS SDK, or Amazon S3 REST API should be used. There is a "+ Add more files" button. The uploaded files listed are "static" (12 Objects - 1.3 MB) and "asset-manifest.json" (- 1.3 KB). At the bottom are "Upload" and "Next" buttons.

aws Services Resource Groups John R. Williams Global Support

Upload

① Select files ② Set permissions ③ Set properties ④ Review

21 Files Size: 1.3 MB Target path: cartreact01

To upload a file larger than 160 GB, use the AWS CLI, AWS SDK, or Amazon S3 REST API. [Learn more ↗](#)

+ Add more files

static
12 Objects - 1.3 MB

asset-manifest.json
- 1.3 KB

Upload Next

Deploying Static Website On AWS3 (25/28)

The screenshot shows the AWS S3 Management Console interface. At the top, there are tabs for 'S3 Management Console' (active), 'React App', 'Settings', and 'arn:aws:s3:::cartreact01'. Below the tabs, the URL is s3.console.aws.amazon.com/s3/buckets/cartreact01/?region=us-east-1&tab=o... The main header includes the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, user 'John R. Williams', 'Global' dropdown, and 'Support' dropdown. A search bar with placeholder text 'Type a prefix and press Enter to search. Press ESC to clear.' is present. Below the search bar are buttons for 'Upload', '+ Create folder', 'Download', and 'Actions'. The region is set to 'US East (N. Virginia)'.

File	Last Modified	Size	Type
favicon.ico	Aug 31, 2020 6:02:27 PM GMT-0400	3.1 KB	Standard
index.html	Aug 31, 2020 6:02:27 PM GMT-0400	2.2 KB	Standard
logo192.png	Aug 31, 2020 6:02:27 PM GMT-0400	5.2 KB	Standard
logo512.png	Aug 31, 2020 6:02:27 PM GMT-0400	9.4 KB	Standard
manifest.json	Aug 31, 2020 6:02:27 PM GMT-0400	492.0 B	Standard

Deploying Static Website On AWS3 (26/28)

The screenshot shows the AWS S3 Management Console interface. The top navigation bar includes tabs for 'S3 Management Console' (selected), 'React App', 'Settings', and an object detail view for 'arn:aws:s3:::cartreact01/index.html?region=us-east-1'. The main content area displays the following details for the object:

- Etag**: c4d88fc2d3936a0320cb0e4608798d1b
- Storage class**: Standard
- Server-side encryption**: None
- Size**: 2.2 KB
- Key**: index.html
- Object URL**: <https://cartreact01.s3.amazonaws.com/index.html>

The 'Object URL' link is highlighted with a blue selection bar.

Deploying Static Website On AWS3 (27/28)

The screenshot shows a web browser window with four tabs open:

- S3 Management Console
- React App
- Settings
- arn:aws:s3:::cartreact01

The address bar displays the URL <https://cartreact01.s3.amazonaws.com/index.html>. The main content area is a Google search results page for the same URL.

Google search results for <https://cartreact01.s3.amazonaws.com/index.html>:

- <https://cartreact01.s3.amazonaws.com/index.html>
- <https://cartreact01.s3.amazonaws.com/index.html - Google Search>

Search filters: All, Shopping, Videos, Images, Maps, More.

About 8 results (0.43 seconds)

docs.aws.amazon.com › latest › dev › s3-arn-format ▾

[Amazon S3 Resources - AWS Documentation](#)

The ARN format for Amazon S3 resources reduces to the following: arn:aws:s3:::
bucket_name/key_name. For a complete list of Amazon S3 resources, see ...

Missing: eartreact01 | Must include: [cartreact01](#)

People also ask

How do I get AWS s3 Arn?

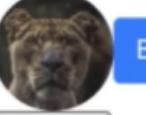
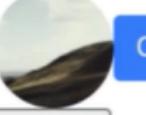
Deploying Static Website On AWS3 (28/28)

The screenshot shows a web browser window with three tabs:

- S3 Management Console
- React App (active tab)
- Settings

The React App tab displays the following content:

Product List

-  Apples_::\$3-Stock=9
-  Oranges::\$4-Stock=3
-  Beans_::\$2-Stock=5
-  Cabbage:\$1-Stock=8

Each item has a "Submit" button below it.

Cart Contents

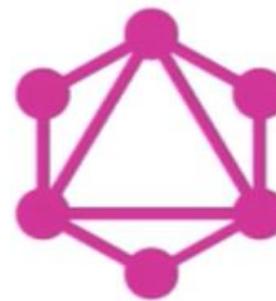
Apples_:

CheckOut

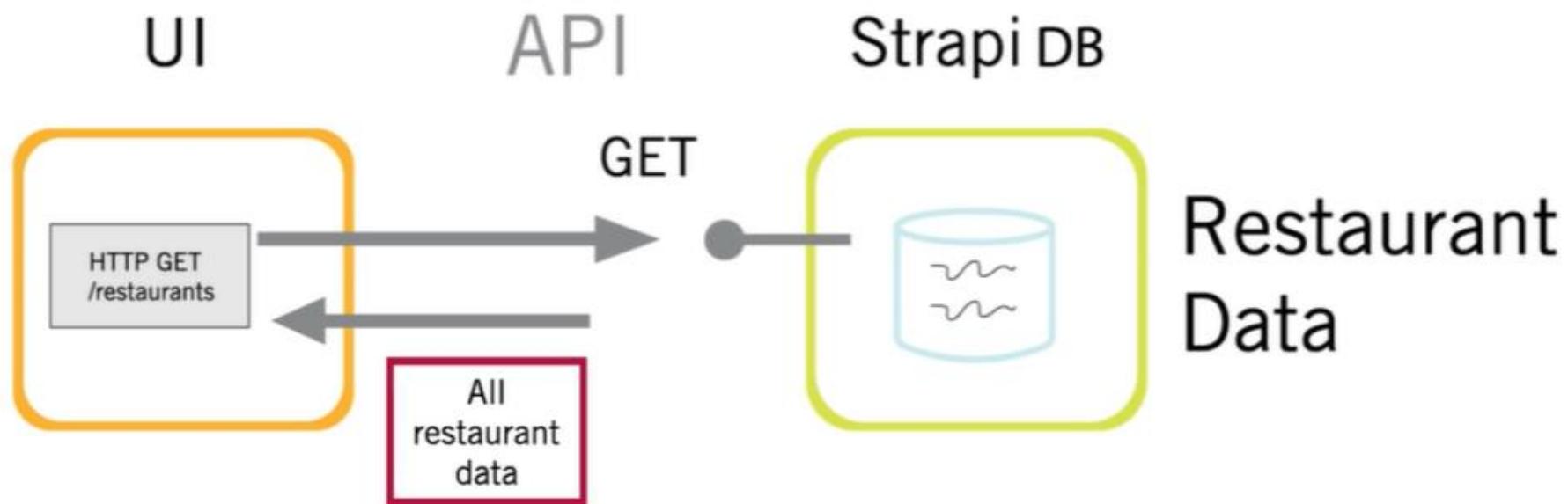
CheckOut \$ 3

Apples_:

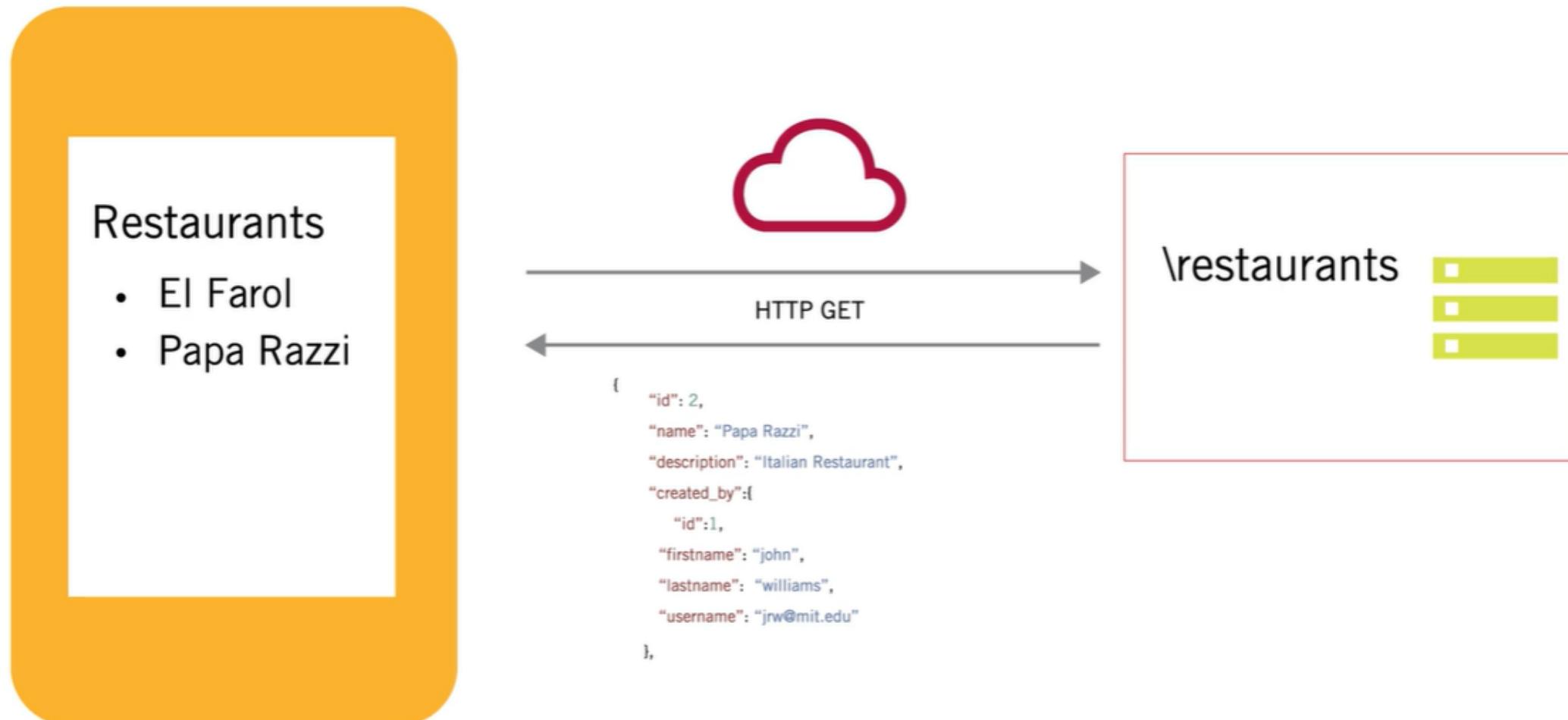
At the bottom of the page, there is a URL bar with the address <http://localhost:1337/p> and a "ReStock Products" button.



REST API Example



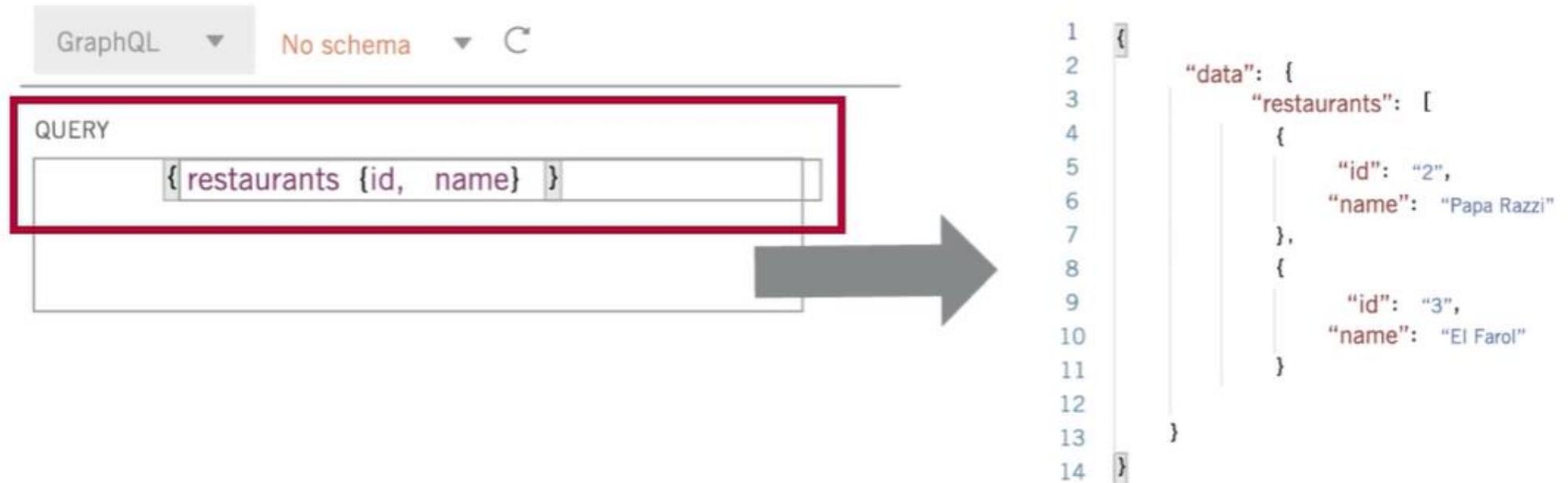
Fetch Data With REST API



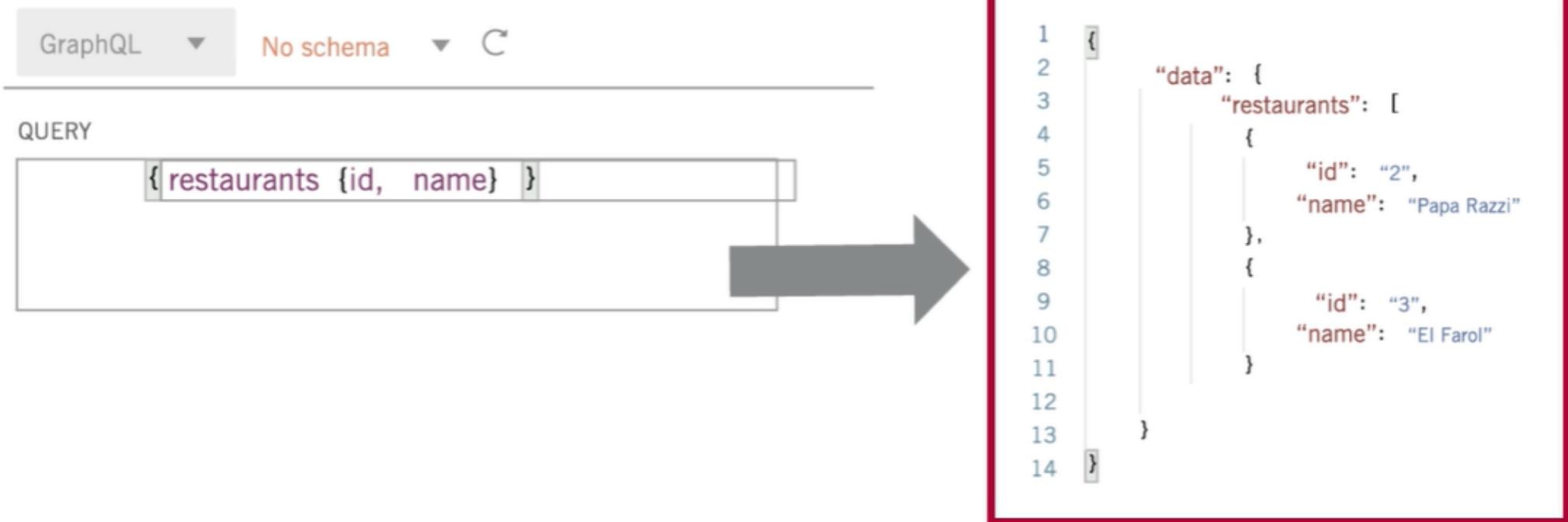
REST API – GET \Restaurants Returns All Data About A Restaurant

```
{  
  "id": 2,  
  "name": "Papa Razzi",  
  "description": "Italian Restaurant",  
  "created_by": {  
    "id": 1,  
    "firstname": "john",  
    "lastname": "williams",  
    "username": "jrw@mit.edu",  
  },  
  "updated_by": {  
    "id": 1,  
    "firstname": "john",  
    "lastname": "williams",  
    "username": "jrw@mit.edu",  
  }  
  "created_at": "2020-08-02T15:54:25.775Z",  
  "updated_at": "2020-08-02T15:54:25.787Z",  
  "photo": [  
    {  
      "id": 1,  
      "name": "Screen Shot 2020-08-02 at 11.44.42 AM.png",  
      "alternativeText": "",  
      "caption": "",  
      "width": 344,  
      "height": 242  
      "formats": {  
        "thumbnail": {  
          "name": "thumbnail_Screen Shot 2020-08-02 at 11.44.42 AM.png",  
          "hash": "Screen_Shot_2020_08_02_at_11_44_42_AM_b623b44eff",  
          "ext": ".png",  
          "mime": "image/png",  
          "width": 222,  
          "height": 156,  
          "size": 82.86,  
          "path": null,  
          "url":  
            "/uploads/thumbnaill_Screen_Shot_2020_08_02_at_11_44_42_AM_b623b44eff.png"  
        },  
        "large": {  
          "name": "large_Screen Shot 2020-08-02 at 11.44.42 AM.png",  
          "hash": "Screen_Shot_2020_08_02_at_11_44_42_AM_b623b44eff",  
          "ext": ".png",  
          "mime": "image/png",  
          "width": 344,  
          "height": 242,  
          "size": 172.2,  
          "path": null,  
          "url":  
            "/uploads/thumbnaill_Screen_Shot_2020_08_02_at_11_44_42_AM_b623b44eff.png"  
        }  
      },  
      "created_by": 1,  
      "updated_by": 1,  
      "created_at": "2020-08-02T15:45:10.604Z",  
      "updated_at": "2020-08-02T15:45:10.627Z"  
    },  
    {  
      "id": 2,  
      "name": "Thumbnail_Screen Shot 2020-08-02 at 11.44.42 AM.png",  
      "alternativeText": "",  
      "caption": "",  
      "width": 222,  
      "height": 156,  
      "size": 82.86,  
      "path": null,  
      "url":  
        "/uploads/thumbnaill_Screen_Shot_2020_08_02_at_11_44_42_AM_b623b44eff.png"  
    }  
  ]  
}
```

Graphql Query – Returns Only The Selected Information (1/2)



Graphql Query – Returns Only The Selected Information (2/2)



Graphql Query Example – Strapi

The screenshot shows the Strapi admin interface for the 'Restaurant' collection. The left sidebar contains navigation links for 'COLLECTION TYPES' (Contacts, Restaurants, Users), 'PLUGINS' (Content-Types Builder, Media Library, Roles & Permissions), and 'GENERAL' (Marketplace, Plugins, Settings). The main content area has a search bar and a 'Restaurant' filter dropdown. A table lists two entries: 'El Farol' (Id: 3, Photo: thumbnail, Created_at: Monday, August 3rd 2020 ..., Name: El Farol) and 'Papa Razzi' (Id: 2, Photo: thumbnail, Created_at: Sunday, August 2nd 2020 ..., Name: Papa Razzi). Below the table are buttons for 'entries per page' (10) and a page number indicator (1). On the right side, there is a 'GET STARTED' section with three video thumbnails: 'Create your first content-type' (1:43), 'Fill your content with data' (1:06), and 'Fetch data through the API' (0:44). The status for these videos is '0% COMPLETED'. The top right corner shows the user email 'jrw@mit.edu' and language 'EN USA'.

Id	Photo	Created_at	Name
3		Monday, August 3rd 2020 ...	El Farol
2		Sunday, August 2nd 2020 ...	Papa Razzi

entries per page: 10 | Page: 1

GET STARTED VIDEOS

- Create your first content-type 1:43
- Fill your content with data 1:06
- Fetch data through the API 0:44

Graphql Query Example – Postman (1/6)

The screenshot shows the Postman application interface. The top navigation bar includes 'New', 'Import', 'Runner', 'My Workspace' (selected), 'Invite', and 'Upgrade'. The left sidebar displays a 'History' tab with several recent requests, including multiple GET requests to 'localhost:1337/restaurants' and various POST requests to 'localhost:1337/graphql' on August 2nd. A 'Collections' and 'APIs' tab are also present. The main workspace shows an 'Untitled Request' with a 'GET' method pointing to 'localhost:1337/restaurants'. The 'Params' tab is selected, showing a table for 'Query Params' with columns for KEY, VALUE, DESCRIPTION, and Bulk Edit. The 'Body' tab is expanded, showing a JSON response with fields like 'updated_at', 'id', 'name', and 'description'. The status bar at the bottom indicates a 200 OK response with 29 ms and 2.3 KB.

Untitled Request

GET localhost:1337/restaurants

Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body

Pretty Raw Preview Visualize JSON

```
52
53
54
55
56
57
58
59
```

```
      "updated_at": "2020-08-02T15:45:10.627Z"
    },
    {
      "id": 3,
      "name": "El Farol",
      "description": "Santa Fe Restaurant",
```

Graphql Query Example – Postman (2/6)

The screenshot shows the Postman application interface. On the left, the 'History' tab is selected, displaying a list of API requests made on August 2nd. The requests include several POSTs to `localhost:1337/graphql` and one GET request to `localhost:1337/restaurants`. On the right, an 'Untitled Request' is being configured. The method is set to GET, and the URL is `localhost:1337/restaurants`. The 'Send' button is highlighted with a cursor. Below the URL, the 'Params' tab is active, showing a table for 'Query Params'. The 'Body' tab is also visible, showing the JSON response received from the API. The response body is as follows:

```
52 |     }           "updated_at": "2020-08-02T15:45:10.627Z"
53 |   ]
54 | }
55 | },
56 | {
57 |   "id": 3,
58 |   "name": "El Farol",
59 |   "description": "Santa Fe Restaurant",
```

Graphql Query Example – Postman (3/6)

The screenshot shows the Postman application interface. On the left, the 'History' tab is selected, displaying a list of previous requests. In the center, an 'Untitled Request' is being edited. The request method is set to 'GET' and the URL is 'localhost:1337/restaurants'. The 'Params' tab is active, showing an empty table for query parameters. The 'Body' tab shows the JSON response received from the server. The response is as follows:

```
1 [  
2 {  
3   "id": 2,  
4   "name": "Papa Razzi",  
5   "description": "Italian Restaurant",  
6   "created_by": {  
7     "id": 1,  
8     "firstname": "john",  
9     "-----"  
10 }  
11 ]
```

Graphql Query Example – Postman (4/6)

The screenshot shows the Postman application interface. The top navigation bar includes 'New', 'Import', 'Runner', 'My Workspace' (selected), 'Invite', and 'Upgrade'. The left sidebar has tabs for 'History' (selected), 'Collections', and 'APIs', with a 'Save Responses' toggle and a 'Clear all' button. The main workspace shows an 'Untitled Request' with a 'POST' method to 'localhost:1337/graphql'. The 'Body' tab is selected, showing a dropdown menu with options: none, form-data, x-www-form-urlencoded, raw, binary, and GraphQL (which is currently highlighted). The 'GraphQL' section contains the query: `query { restaurants { id, name } }`. To the right, the 'GRAPHQL VARIABLES' section shows a variable named '1' with a value of '1'. Below the variables, the response status is '200 OK' with a time of '31 ms' and a size of '384 B'. The response body is displayed in JSON format:

```
2 "data": {  
3   "restaurants": [  
4     {  
5       "id": "2",  
6       "name": "Papa Razzi"  
7     },
```

Graphql Query Example – Postman (5/6)

The screenshot shows the Postman application interface. The top navigation bar includes 'New', 'Import', 'Runner', 'My Workspace' (selected), 'Invite', and 'Upgrade'. The left sidebar displays a 'History' section with several recent requests, mostly 'GET localhost:1337/restaurants'. A 'Collections' and 'APIs' tab are also present. The main workspace is titled 'Untitled Request' and shows a POST request to 'localhost:1337/graphql'. The 'Body' tab is selected, showing a GraphQL query: `1 {restaurants {id, name}}`. The 'GRAPHQL VARIABLES' section contains a single variable '1'. Below the query, the 'Body' section shows the response in JSON format:

```
1 {
2   "data": {
3     "restaurants": [
4       {
5         "id": "2",
6         "name": "Papa Razzi"
7       },

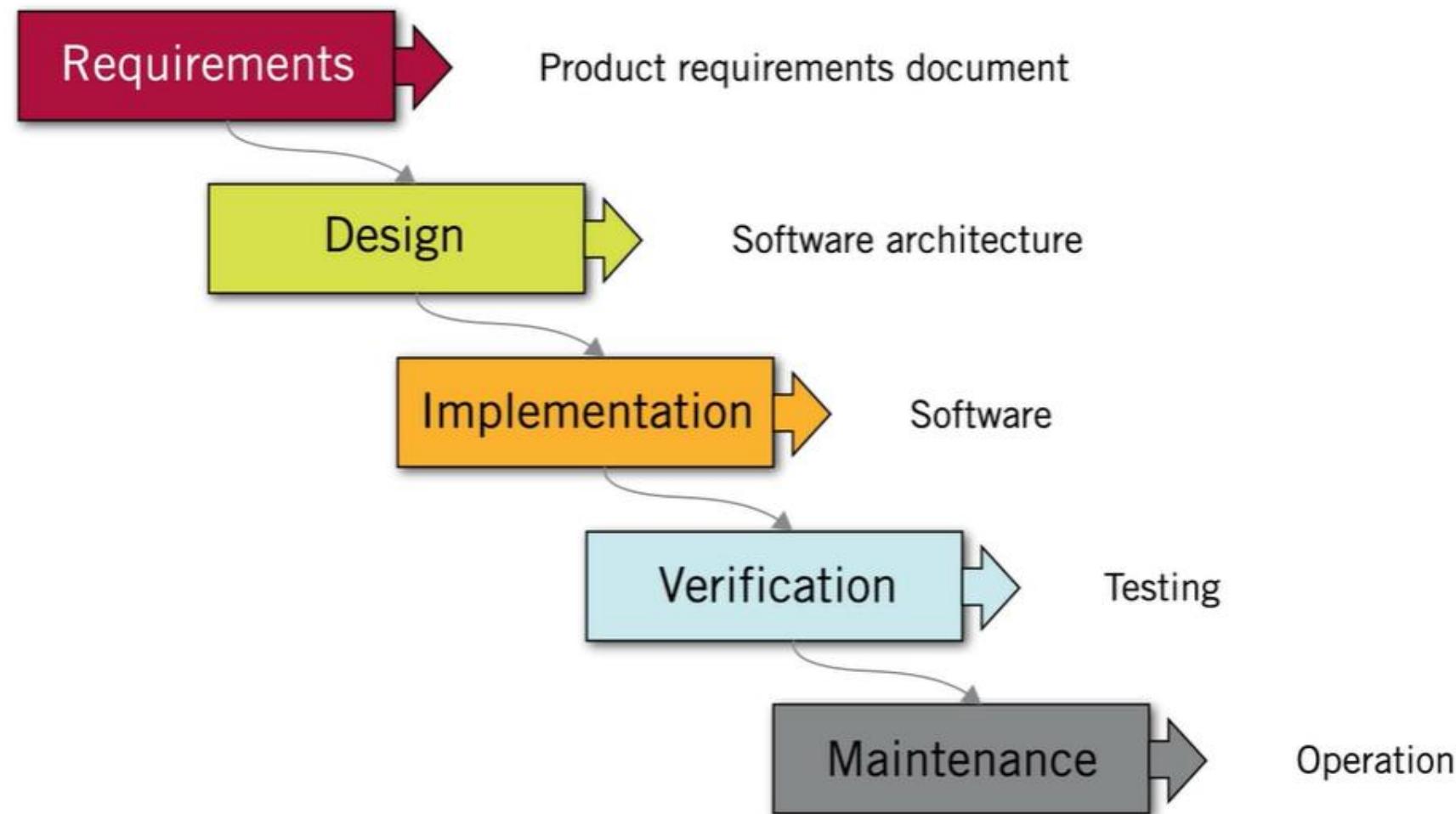
```

Graphql Query Example – Postman (6/6)

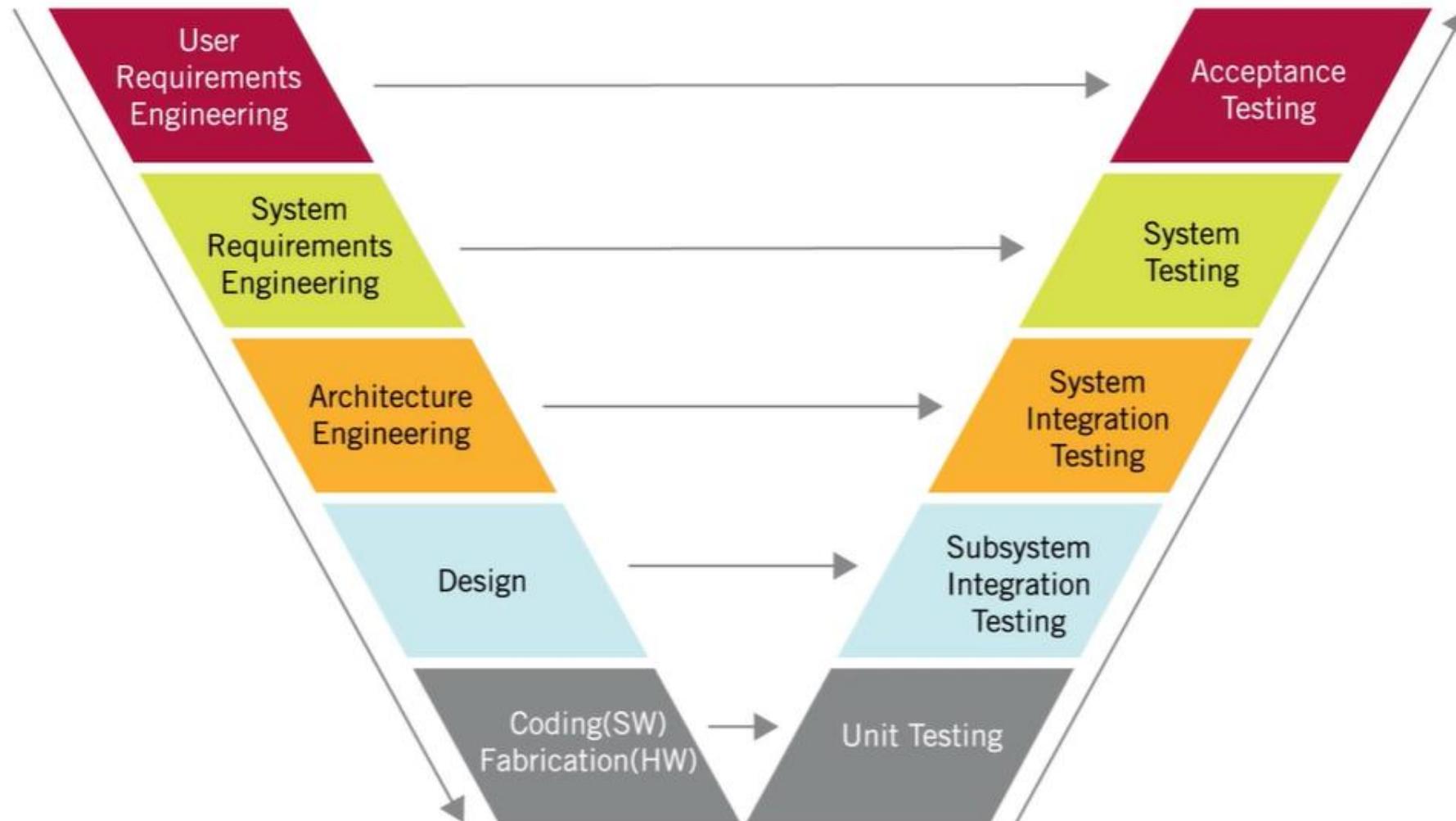
The screenshot shows the Postman application interface. The top navigation bar includes 'New', 'Import', 'Runner', 'My Workspace' (selected), 'Invite', and 'Upgrade'. The left sidebar displays a 'History' section with a 'Save Responses' toggle and a list of requests from 'August 2'. The main workspace shows an 'Untitled Request' with a 'POST' method to 'localhost:1337/graphql'. The 'Body' tab is selected, showing a GraphQL query: `1 {restaurants {id, name}}`. The 'GRAPHQL VARIABLES' section contains variable definitions. Below the query, the 'Body' tab shows the response: `1` (status 200 OK, 33 ms, 384 B). The response JSON is displayed in 'Pretty' format:

```
4
5   {
6     "id": "2",
7     "name": "Papa Razzi"
8   },
9   {
10    "id": "3",
11    "name": "El Farol "
```

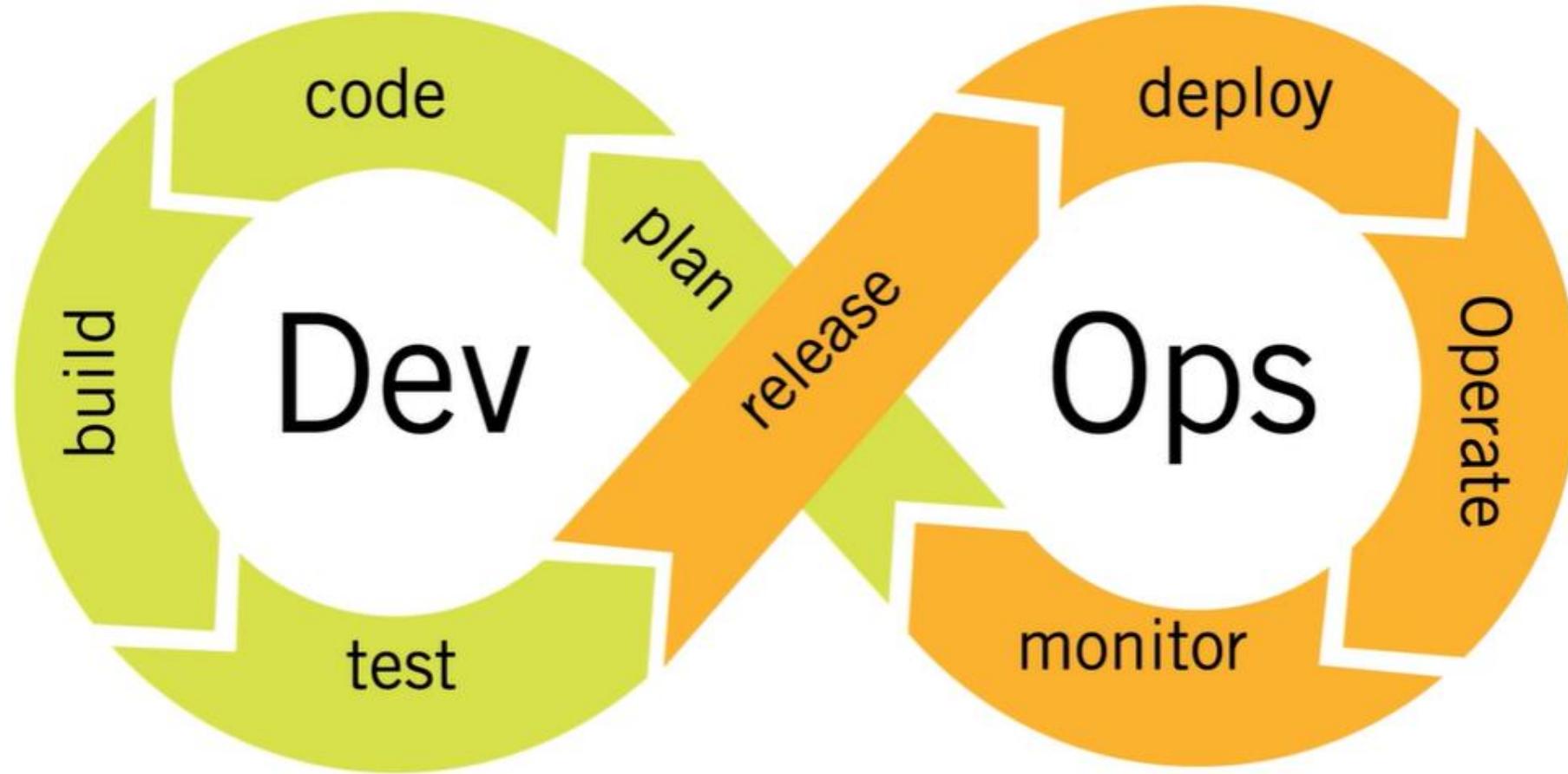
System Development Life Cycle – Old school



V Models For Testing – Old school



https://insights.sei.cmu.edu/sei_blog/2013/11/using-v-models-for-testing.html



Testing → QA → QE

Intelligent Testing

Start with the basics



Jest

<https://jestjs.io/>



<https://nodejs.org/>

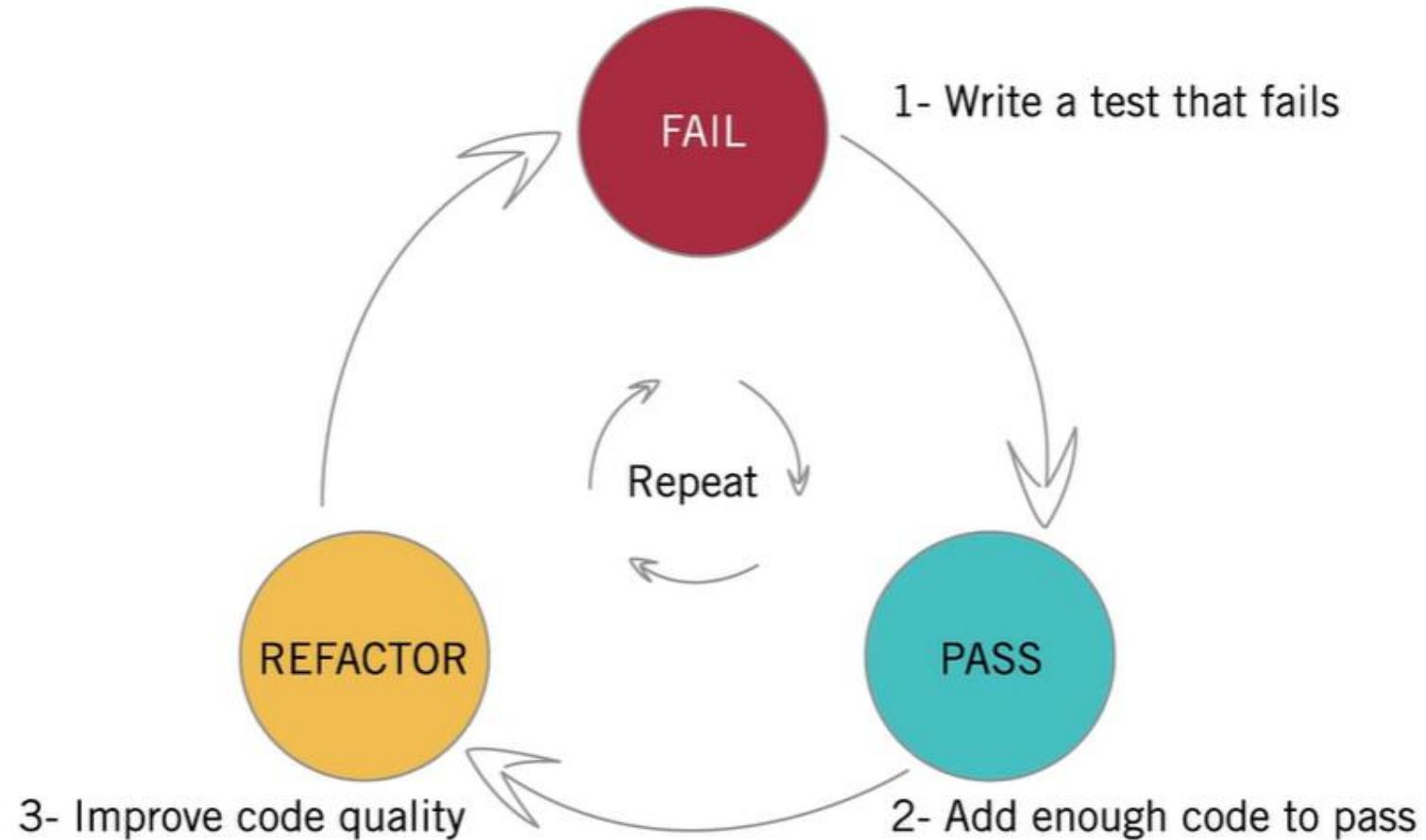
Examining Node.js Version



A screenshot of a macOS Terminal window titled "Terminal". The window shows the command "node -v" being run and its output "v12.18.1". The terminal has a dark background with white text. The window title bar includes the date and time: "Tue 9:56 AM".

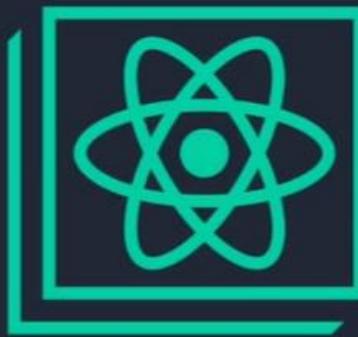
```
Terminal Shell Edit View Window Help
abel@VM ~ % node -v
v12.18.1
abel@VM ~ %
```

Test-Driven Development (TDD)



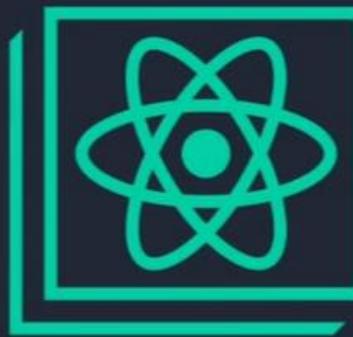
TDD Workflow

- Write a test for a desired software feature
- Watch it fail 
- Write the code
- Watch it fail 
- Fix
- Watch it pass 
- Feel good
- Add, commit, push



Create React App

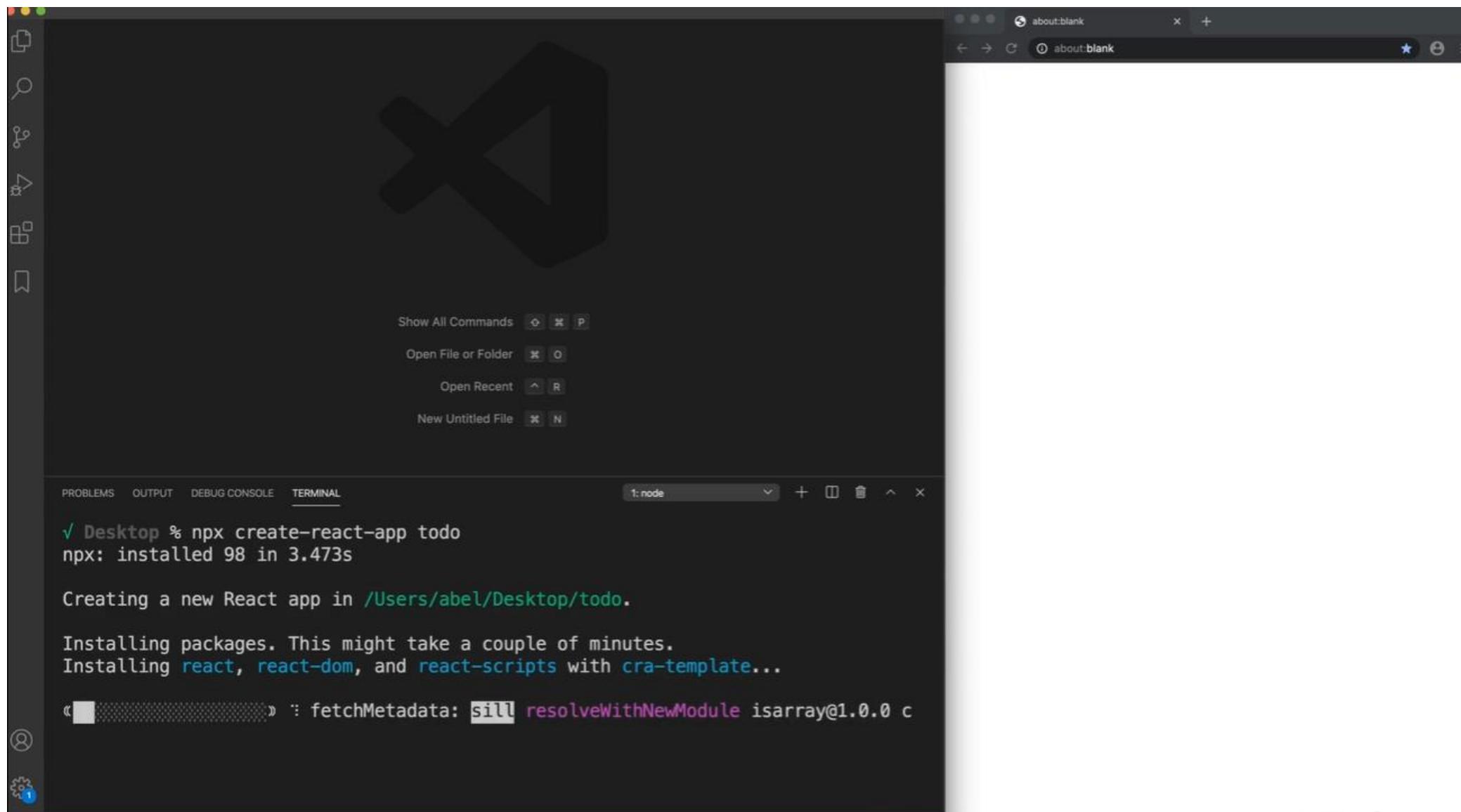
Set up a modern web app by running one command.



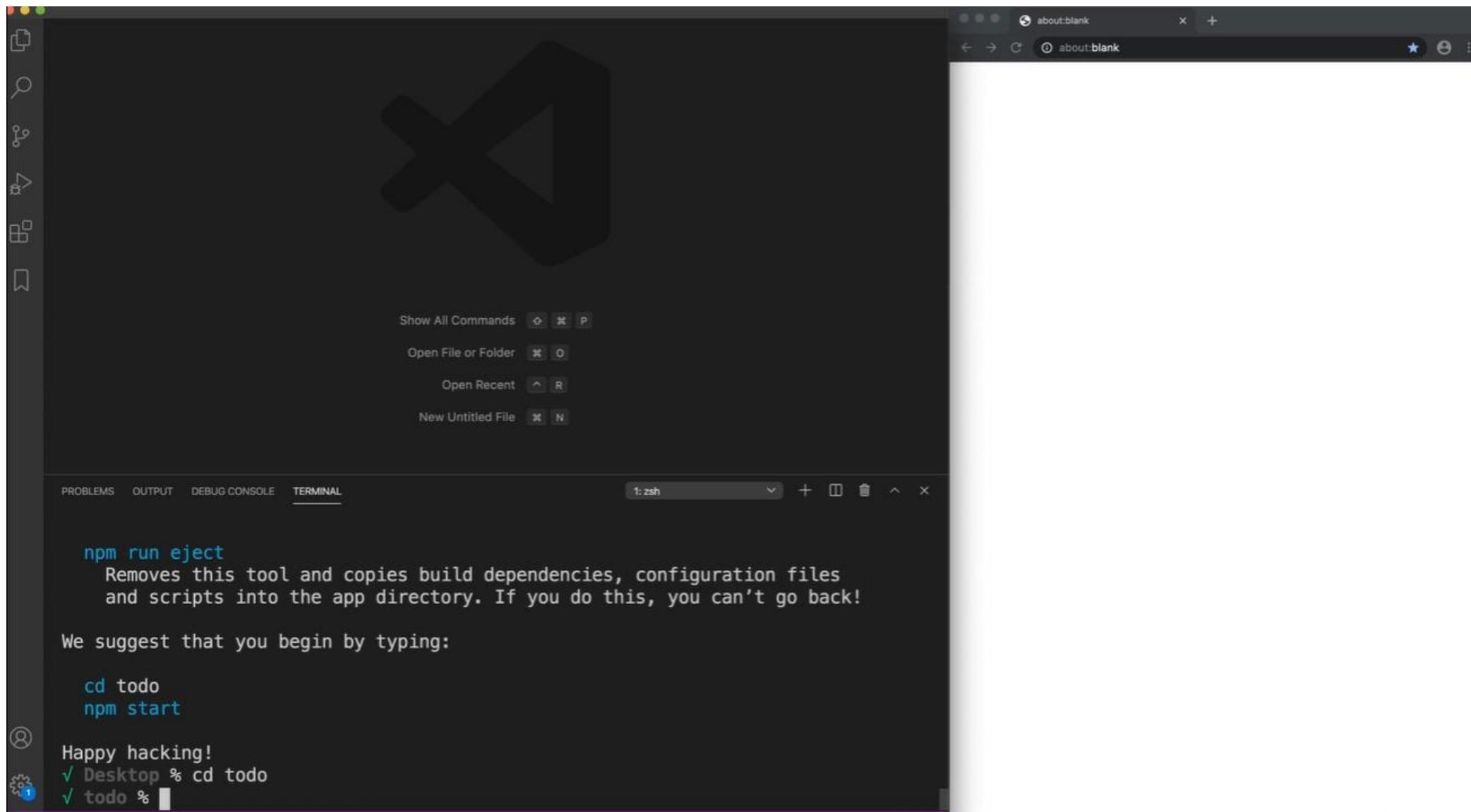
Create React App

<https://create-react-app.dev/>

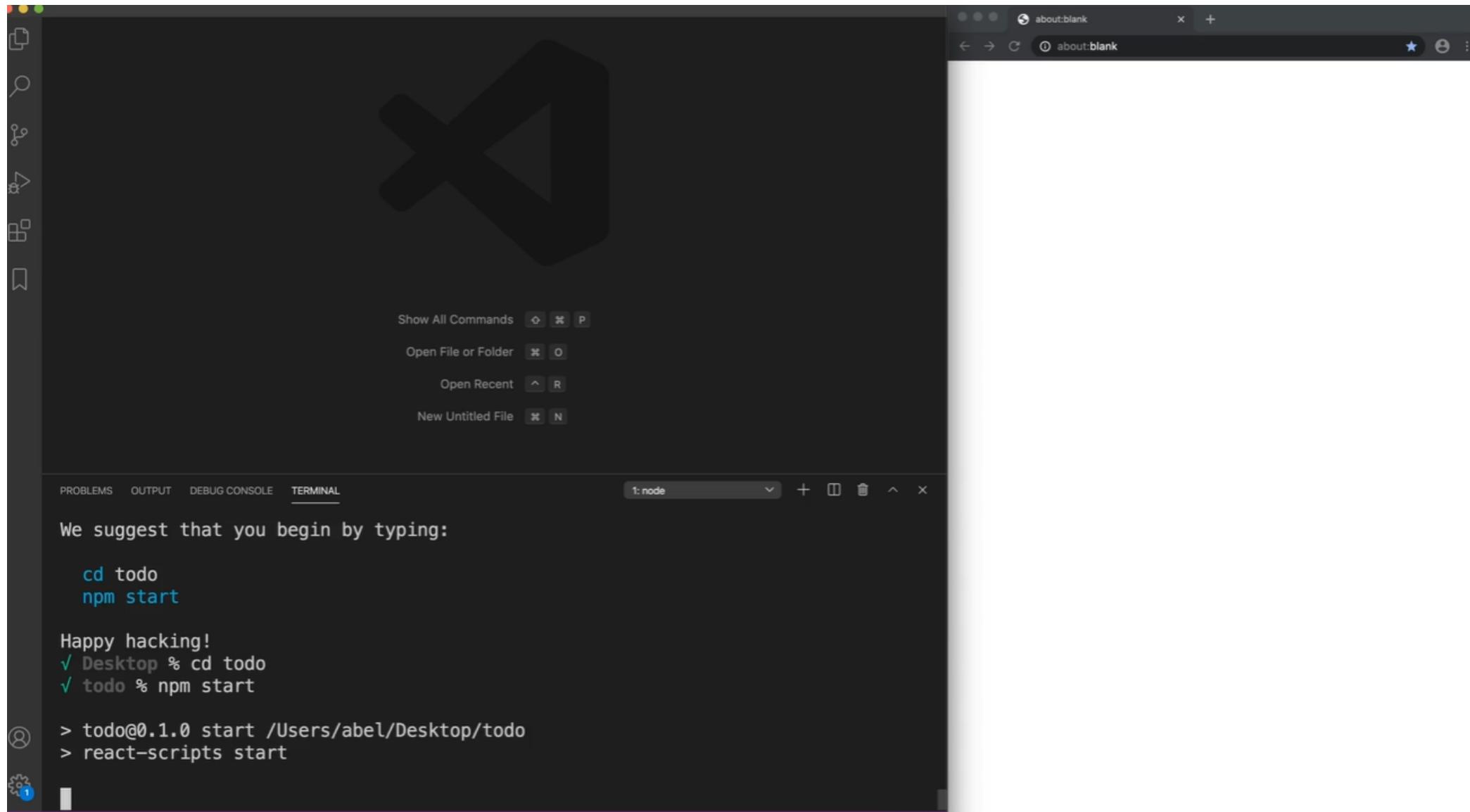
Testing A Simple DOM Application (1/25)



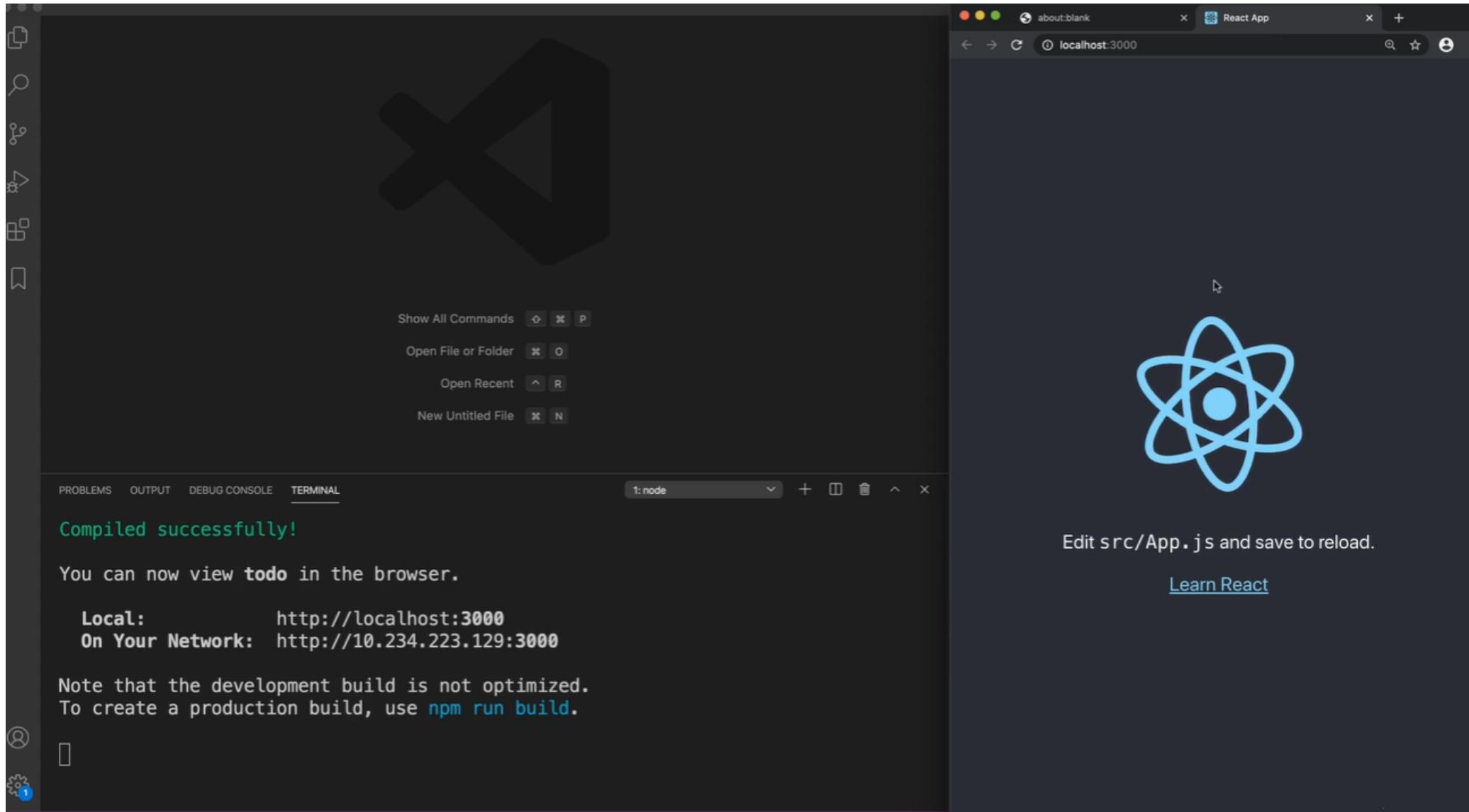
Testing A Simple DOM Application (2/25)



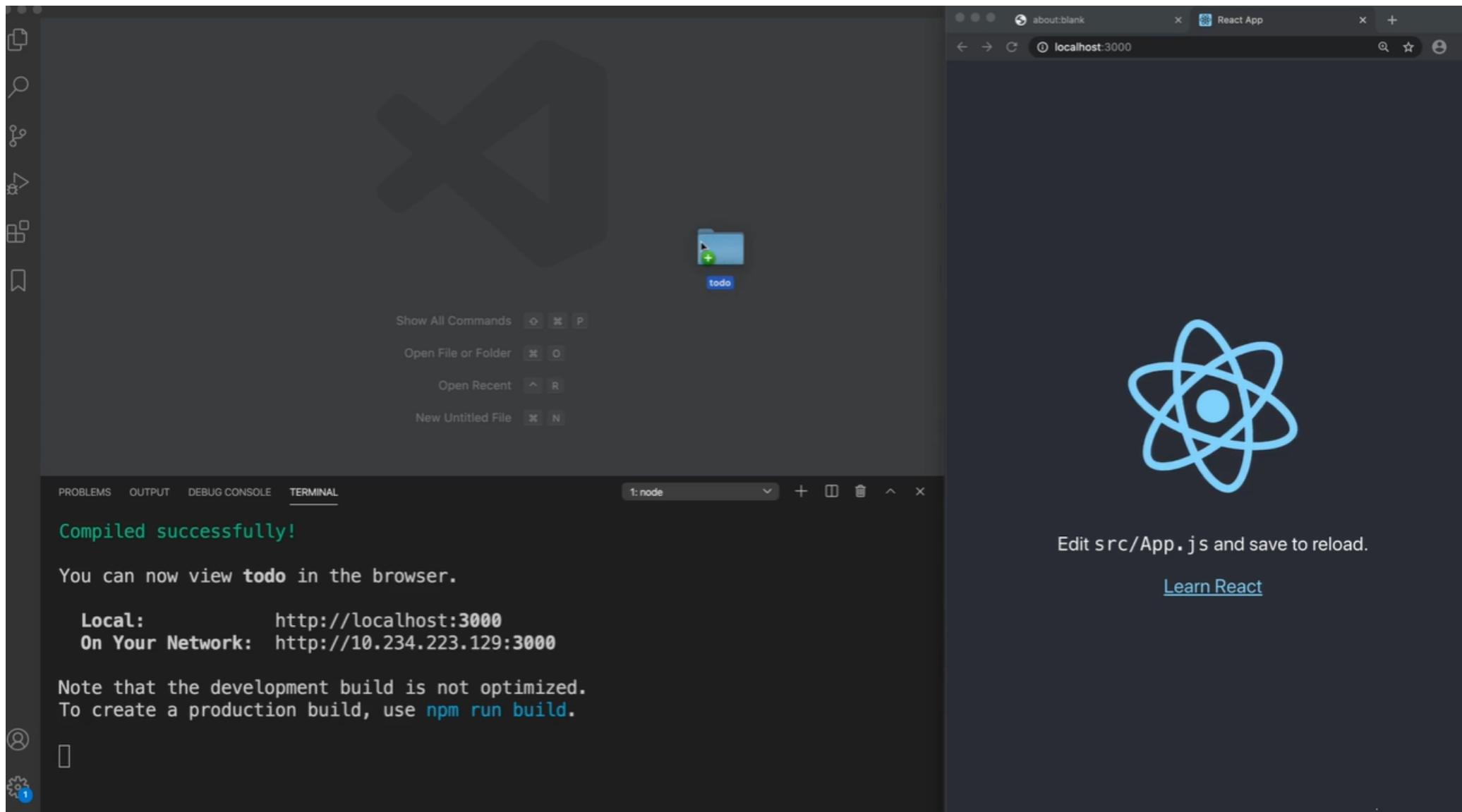
Testing A Simple DOM Application (3/25)



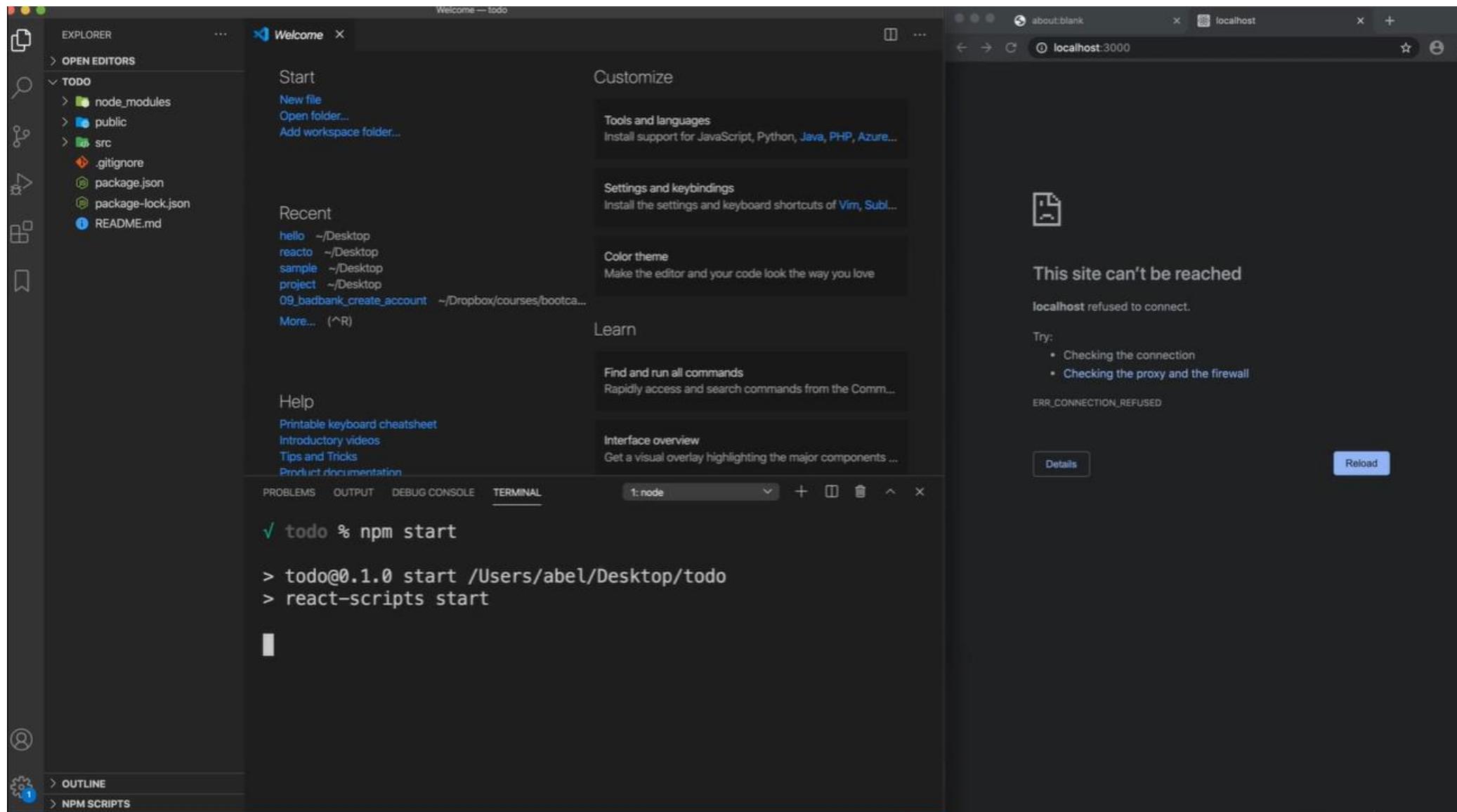
Testing A Simple DOM Application (4/25)



Testing A Simple DOM Application (5/25)



Testing A Simple DOM Application (6/25)



Testing A Simple DOM Application (7/25)

The screenshot shows a development environment for a React application named "todo".

Code Editor: The `App.js` file is open, displaying the following code:

```
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;
```

Terminal: The terminal shows the build process:

```
Compiling...  
Compiled successfully!
```

Browser Preview: The browser window shows the React application running at `localhost:3000`. It features a blue atom-like logo and a "Learn React" link.

Documentation Links:

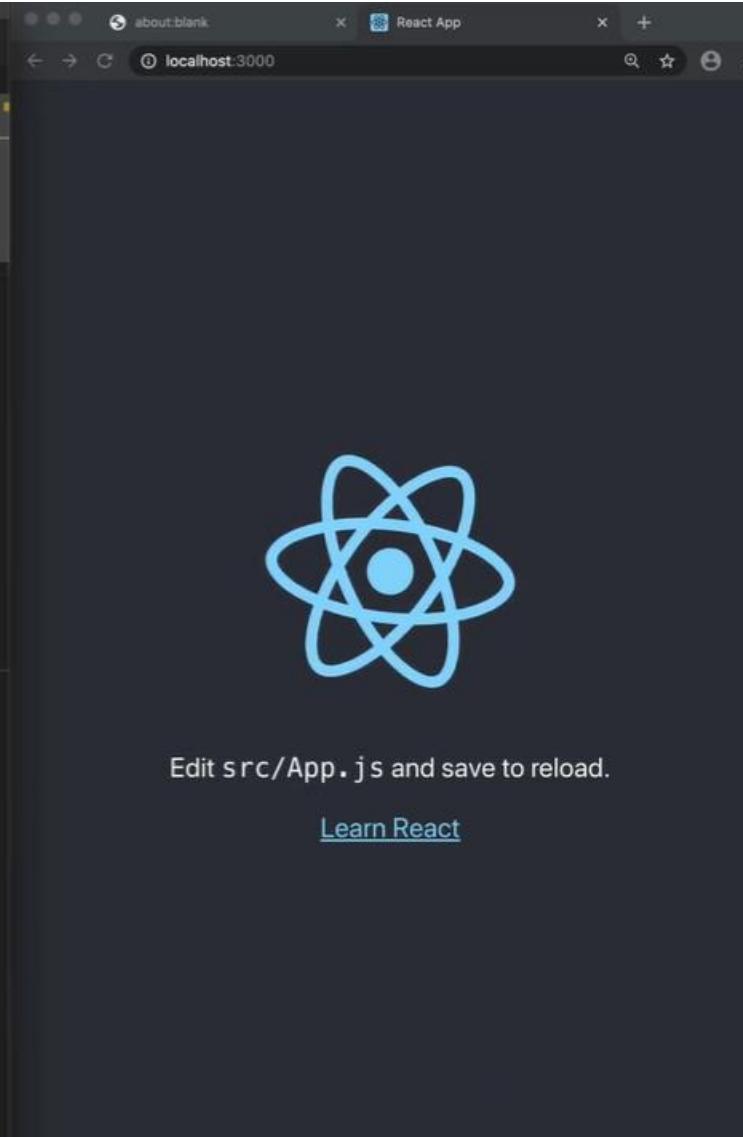
- [Edit src/App.js and save to reload.](#)
- [Learn React](#)

Local and Network URLs:

Local: `http://localhost:3000`
On Your Network: `http://10.234.223.129:3000`

Note that the development build is not optimized.
To create a production build, use `npm run build`.

Testing A Simple DOM Application (8/25)



The screenshot shows a Microsoft Edge browser window displaying a React application. The URL is `localhost:3000`. The browser title bar says "React App". The main content area shows a large blue React logo.

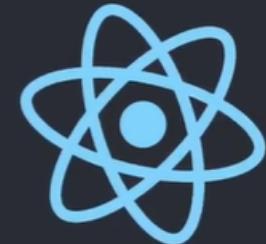
On the left, there is a code editor window for `App.js` in a project named `App`. The code defines a functional component `App` that uses `useState` to manage state and `useEffect` to handle side effects. It includes a form for adding items to a list.

Below the code editor, the terminal output shows:

```
Compiled successfully!
You can now view todo in the browser.
Local: http://localhost:3000
On Your Network: http://10.234.223.129:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
```

At the bottom right of the browser window, there is a link: "Edit src/App.js and save to reload." and "Learn React".

Testing A Simple DOM Application (9/25)



A screenshot of a computer screen showing a development environment for a React application. On the left is a dark-themed code editor with an open file named `App.js`. The code defines a `App` component that returns a `<div>` containing an `<h1>TODO</h1>`, a `<TodoList items={items} />`, and a form with a text input and a button. The input has an `id="new-todo"`, a placeholder "Add Todo...", and an `onChange` event handler that sets the value of the input. The button adds the current length of the `items` array plus one. Below the code editor, a terminal window shows the message "Compiled successfully!" followed by instructions to view the app at `http://localhost:3000` or `http://10.234.223.129:3000`. It also notes that the build is not optimized and suggests using `npm run build` for production. To the right of the code editor is a browser window titled "React App" showing the application running at `localhost:3000`. The browser displays the React logo and the text "Edit src/App.js and save to reload." with a link to "Learn React".

Testing A Simple DOM Application (10/25)

The screenshot shows a development environment for a React application named "todo".

Code Editor: The left side displays the `App.js` file with the following code:

```
App.js — todo
src > App.js > App
24   placeholder="Add Todo..." 
25   onChange={e => setValue(e.target.value)}
26   />
27   <button>
28     Add #{items.length + 1}
29   </button>
30   </form>
31   </div>
32 );
33 }
34
35 function TodoList(props){
36   return (
37     <ul>
38       {props.items.map((item,i) => (
39         <li key={i}>{item}</li>
40       )));
41     </ul>
42 );
```

The terminal at the bottom shows the message: **Compiled successfully!**

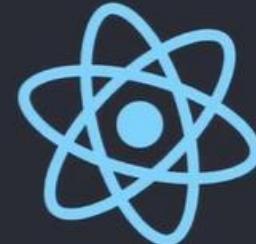
Browser: The right side shows a browser window titled "React App" at `localhost:3000`. The page displays a simple React component with a text input field and a button, and a list of items below it.

UI Elements: A large blue atom-like logo is centered on the page. Below it, there is a message: **Edit `src/App.js` and save to reload.** and a link: [Learn React](#).

Local: `http://localhost:3000`
On Your Network: `http://10.234.223.129:3000`

Note that the development build is not optimized.
To create a production build, use `npm run build`.

Testing A Simple DOM Application (11/25)



A screenshot of a Microsoft Visual Studio Code (VS Code) interface showing a React application development environment. The left side features the Explorer sidebar with project files like 'node_modules', 'public', and 'src' containing 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js', 'logo.svg', 'serviceWorker.js', 'setupTests.js', '.gitignore', 'package.json', 'package-lock.json', and 'README.md'. The center is the main editor area with the code for 'App.js':

```
App.js
1 import React from 'react';
2 import './App.css';
3
4 function App(){
5   const [items, setItems] = React.useState([]);
6   const [value, setValue] = React.useState('');
7
8   function handleSubmit(e){
9     e.preventDefault();
10    setItems([...items, value]);
11    setValue('');
12  }
13
14  return (
15    <div>
16      <h1>TODO</h1>
17      <TodoList items={items} />
18      <form onSubmit={handleSubmit}>
19        <label htmlFor="new-todo">Add todo: </label>
```

The bottom of the editor shows a terminal window output:

```
Compiled successfully!
You can now view todo in the browser.
Local:          http://localhost:3000
On Your Network: http://10.234.223.129:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
```

To the right of the editor, a browser window titled 'React App' shows the application running at 'localhost:3000'. The browser displays a simple 'TODO' application with a single item: 'Edit src/App.js and save to reload.' Below it is a link 'Learn React'.

Testing A Simple DOM Application (12/25)

The screenshot shows a code editor (VS Code) and a web browser side-by-side.

Code Editor (VS Code):

- EXPLORER:** Shows the project structure with files: TODO, node_modules, public, src (containing App.css, App.js, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js), .gitignore, package.json, package-lock.json, and README.md.
- EDITOR:** The file `App.js` is open, displaying the following code:

```
App.js --- todo
App.js
src > App.js > ...
1 import React from 'react';
2 import './App.css';
3
4 function App(){
5   const [items, setItems] = React.useState([]);
6   const [value, setValue] = React.useState('');
7
8   function handleSubmit(e){
9     e.preventDefault();
10    setItems([...items, value]);
11    setValue('');
12  }
13
14  return (
15    <div>
16      <h1>TODO</h1>
17      <TodoList items={items} />
18      <form onSubmit={handleSubmit}>
19        <label htmlFor="new-todo">Add todo: </label>
```

Below the code editor:

- PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL tabs.
- Terminal output:
 - Compiled successfully!
 - You can now view `todo` in the browser.
 - Local: <http://localhost:3000>
 - On Your Network: <http://10.234.223.129:3000>
- Note that the development build is not optimized.
To create a production build, use `npm run build`.

Browser:

A browser window titled "React App" is open at `localhost:3000`. The page displays a "TODO" application with the following interface:

- A header with the word "TODO".
- An input field with placeholder "Add todo..." and a button labeled "Add #1".
- A list area where todos are displayed.

Testing A Simple DOM Application (13/25)

The screenshot shows a development environment with two main windows.

VS Code (Left Window):

- EXPLORER:** Shows the project structure with files like `App.js`, `App.css`, `index.css`, `index.js`, `logo.svg`, `serviceWorker.js`, `setupTests.js`, `.gitignore`, `package.json`, `package-lock.json`, and `README.md`.
- EDITOR:** Displays the `App.js` file content:

```
1 import React from 'react';
2 import './App.css';
3
4 function App(){
5   const [items, setItems] = React.useState([]);
6   const [value, setValue] = React.useState('');
7
8   function handleSubmit(e){
9     e.preventDefault();
10    setItems([...items, value]);
11    setValue('');
12  }
13
14  return (
15    <div>
16      <h1>TODO</h1>
17      <TodoList items={items} />
18      <form onSubmit={handleSubmit}>
19        <label htmlFor="new-todo">Add todo: </label>
```

Below the editor, the terminal shows:

```
Compiled successfully!
You can now view todo in the browser.
Local: http://localhost:3000
On Your Network: http://10.234.223.129:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
```

Browser Preview (Right Window):

A browser window titled "React App" at `localhost:3000` displays a "TODO" application. The page shows a list of items: "one" and "two". There is an input field with the placeholder "Add todo:" containing "three" and a button labeled "Add #3".

Testing A Simple DOM Application (14/25)

The screenshot shows a development environment with a code editor and a browser preview.

Code Editor (VS Code):

- EXPLORER:** Shows the project structure with files: node_modules, public, src (containing App.css, App.js, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js), .gitignore, package.json, package-lock.json, and README.md.
- OPEN EDITORS:** Shows the content of `App.js`.
- TERMINAL:** Shows the output of the build process:
 - Compiled successfully!
 - You can now view `todo` in the browser.
 - Local: <http://localhost:3000>
 - On Your Network: <http://10.234.223.129:3000>

Browser Preview:

The browser window displays a simple "TODO" application. The title bar says "React App" and the address bar says "localhost:3000". The page content includes:

- A heading "TODO" above a list of items: one, two, three, four.
- An input field labeled "Add todo..." and a button labeled "Add #5".

Testing A Simple DOM Application (15/25)

The screenshot displays a development environment with several windows:

- Code Editor (App.test.js):** Shows a test script for a React application. The test checks if a link with the text "learn react" is rendered correctly.
- Browser (localhost:3000):** Shows a simple "TODO" application with four items in the list: "one", "two", "three", and "four". There is also a text input field labeled "Add todo:" with placeholder "Add Todo..." and a button labeled "Add #5".
- Terminal:** Shows the command output for running tests:

```
✓ todo % npm test
> todo@0.1.0 test /Users/abel/Desktop/todo
> react-scripts test
```

Testing A Simple DOM Application (16/25)

The screenshot shows a development environment with three main panes:

- EXPLORER** pane on the left displaying project files: node_modules, public, and src (containing App.css, App.js, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md).
- EDITOR** pane in the center showing `App.test.js` code:import React from 'react';
import { render } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
 const { getByText } = render(<App />);
 const linkElement = getByText(/learn react/i);
 expect(linkElement).toBeInTheDocument();
});
- TERMINAL** pane at the bottom showing test results:

```
Tests: 1 failed, 1 total
Snapshots: 0 total
Time: 1.529s
Ran all test suites related to changed files.

Watch Usage
> Press a to run all tests.
> Press f to run only failed tests.
> Press q to quit watch mode.
> Press p to filter by a filename regex pattern.
> Press t to filter by a test name regex pattern.
> Press Enter to trigger a test run.
```

To the right, a browser window displays the application's interface with the title "TODO" and a list of items: one, two, three, four. There is also an "Add todo:" input field with placeholder "Add Todo..." and a button "Add #5".

Testing A Simple DOM Application (17/25)

The screenshot shows a development environment with the following components:

- EXPLORER**: Shows the project structure with files like App.js, App.test.js, App.css, etc.
- EDITOR**: Displays `App.test.js` containing a Jest test for a `ToDo` component.
- BROWSER**: Shows a React application titled "TODO" with four items: "one", "two", "three", and "four". There is an input field "Add todo:" and a button "Add #5".
- TERMINAL**: Shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (14ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.489s, estimated 2s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```

Testing A Simple DOM Application (18/25)

The screenshot shows a code editor interface with several windows open:

- EXPLORER**: Shows the project structure with files like App.js, App.css, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- App.js**: The main application component.
- App.test.js**: The test file for the application. It contains a single test case for a 'ToDo' component, which renders a list of items ('one', 'two', 'three', 'four') and a text input field for adding new todos.
- TERMINAL**: Shows the output of running the test suite. The terminal window title is "node". The output shows the test passed: "PASS src/App.test.js" and "✓ ToDo (7ms)". It also displays summary statistics: "Test Suites: 1 passed, 1 total", "Tests: 1 passed, 1 total", "Snapshots: 0 total", "Time: 0.439s, estimated 1s", and "Ran all test suites related to changed files".
- about:blank**: A browser tab showing the application at "localhost:3000". The page displays a "TODO" list with the same four items as the test. There is an "Add todo:" input field with placeholder "Add Todo..." and a button labeled "Add #5".

Testing A Simple DOM Application (19/25)

The screenshot shows a developer's workspace with several open windows:

- EXPLORER**: Shows the project structure with files like `node_modules`, `public`, and `src` containing `App.css`, `App.js`, `App.test.js`, `index.css`, `index.js`, `logo.svg`, `serviceWorker.js`, `setupTests.js`, `.gitignore`, `package.json`, `package-lock.json`, and `README.md`.
- EDITOR**: The `App.js` file is open, displaying React component code for a "TODO" application. It includes a form for adding items and a list of existing items.
- TERMINAL**: Shows the test results for `src/App.test.js`, indicating 1 test passed in 7ms. It also displays Jest summary statistics: 1 test suite, 1 test, 0 snapshots, and a total time of 0.439s.
- BROWSER**: A browser window titled "React App" shows the live application at `localhost:3000`. The application displays a "TODO" heading, a list of items ("one", "two", "three", "four"), and an input field with placeholder "Add Todo..." and a button labeled "Add #5".

Testing A Simple DOM Application (20/25)

The screenshot shows a development environment with the following components:

- Code Editor (VS Code):** The left pane displays the file structure and the content of `App.test.js`. The code is a Jest test for a React component named `App`. It uses `ReactDOM.render` to render the component into a div, then uses DOM API assertions to check if the rendered content matches expectations.
- Browser Preview:** The right pane shows a browser window at `localhost:3000` displaying a "TODO" application. The page lists four items: "one", "two", "three", and "four". There is an input field labeled "Add todo:" with placeholder "Add Todo..." and a button labeled "Add #5".
- Terminal:** The bottom pane shows the output of the Jest test run. It indicates 1 failed test suite and 1 failed test. The error message points to line 12 of `src/App.test.js`, which contains an assertion that failed.

```
App.test.js -- todo
App.js App.test.js
src > App > test('ToDo') callback
  1 import React from 'react';
  2 import * as ReactDOM from 'react-dom';
  3 import App from './App';
  4
  5 test('ToDo', () => {
  6   const root = document.createElement('div');
  7   ReactDOM.render(<App/>, root);
  8
  9   // after rendering our component
 10  // use DOM APIs (query selector) to make assertions
 11  expect(root.querySelector('h1').textContent).toBe('TODO');
 12  expect(root.querySelector('label').textContent).toBe('Add todos: []');
 13 });
 14

at Object.<anonymous> (src/App.test.js:12:51)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        0.364s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

Testing A Simple DOM Application (21/25)

The screenshot shows a developer's workspace with the following components:

- Code Editor (VS Code):** The left pane displays the file structure and the content of `App.test.js`. The code is a Jest test for a React application, specifically testing the `ToDo` component.
- Browser:** The right pane shows a web browser window titled "React App" at `localhost:3000`. It displays a "TODO" page with a list of items: "one", "two", "three", and "four". Below the list is an input field labeled "Add todo:" with placeholder "Add Todo..." and a button labeled "Add #5".
- Terminal:** At the bottom, a terminal window shows the output of a Jest test run. It includes the following text:

```
expect(received).toBe(expected) // Object.is equality

Expected: "Add todos: "
Received: "Add todo: "
```

Testing A Simple DOM Application (22/25)

The screenshot shows a developer's workspace with the following components:

- Code Editor (VS Code):** The left pane displays the file structure and the content of `App.test.js`. The code is a Jest test for a React application, specifically for a component named `ToDo`. It uses `ReactDOM.render` to render the component into a div, then uses `expect` to assert that the h1 text content is 'TODO' and the label text content is 'Add todo: []'.

```
import React from 'react';
import * as ReactDOM from 'react-dom';
import App from './App';

test('ToDo', () => {
  const root = document.createElement('div');
  ReactDOM.render(<App/>, root);

  // after rendering our component
  // use DOM APIs (query selector) to make assertions
  expect(root.querySelector('h1').textContent).toBe('TODO');
  expect(root.querySelector('label').textContent).toBe('Add todo: []');
});
```
- Browser:** The right pane shows a browser window at `localhost:3000` displaying a simple 'TODO' application. It has a heading 'TODO' and a list of items: 'one', 'two', 'three', 'four'. Below the list is an input field labeled 'Add todo...' and a button labeled 'Add #5'.
- Terminal:** At the bottom, a terminal window shows the command `RUNS src/App.test.js`.

Testing A Simple DOM Application (23/25)

The screenshot displays a development environment with several windows open:

- EXPLORER**: Shows the project structure with files like App.js, App.css, index.css, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- App.js**: The main application file.
- App.test.js**: The test file being edited, containing a single test case for the 'ToDo' component.
- TERMINAL**: Shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (5ms)

Test Suites: 1 passed, 1 total
Tests:    1 passed, 1 total
Snapshots: 0 total
Time:      0.328s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```
- about:blank**: A browser window showing the application's UI. It has a title 'React App' and a URL 'localhost:3000'. The page displays a 'TODO' heading and a form with an input field 'Add todo:' and a button 'Add #1'.

Testing A Simple DOM Application (24/25)

The screenshot shows a development environment with several windows open:

- Explorer:** Shows the project structure with files like App.js, App.css, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- Editor:** The file `App.test.js` is open, containing a test for the `ToDo` component. The code uses `ReactDOM.render` to render the component into a div, then uses `expect` to assert the text content of the h1, label, and button elements.
- Terminal:** Shows the test results:
 - FAIL** src/App.test.js
 - x ToDo (6ms)
 - ToDo
 - `expect(received).toBe(expected) // Object.is equality`
 - Expected: "Add #5"
 - Received: "Add #1"
- Browser:** A browser window showing the application at `localhost:3000`. The page title is "TODO". It has an input field "Add todo:" with placeholder "Add Todo..." and a button "Add #1".

Testing A Simple DOM Application (25/25)

The screenshot displays a development environment with several open windows:

- Code Editor:** Shows `App.test.js` with test code for a `ToDo` component. The code imports React and ReactDOM, and uses Jest's `querySelector` to assert the rendered DOM structure.
- Terminal:** Shows the output of a test run:

```
PASS  src/App.test.js
  ✓ ToDo (5ms)

Test Suites: 1 passed, 1 total
Tests:    1 passed, 1 total
Snapshots: 0 total
Time:      0.328s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```
- Browser Preview:** A window titled "React App" shows a simple "TODO" application with a header, an input field labeled "Add todo:", and a button labeled "Add #1".

Testing Using The React Library (1/9)

The screenshot shows a development environment with the following components:

- Explorer:** Shows the project structure with files like App.js, App.test.js, index.css, etc.
- Editor:** Displays the `App.test.js` file containing a test suite for a `ToDo` component using the `@testing-library/react` library.
- Terminal:** Shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (4ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.313s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```
- Browser:** Shows the application running at `localhost:3000`, displaying a `TODO` page with an input field and a button.

Testing Using The React Library (2/9)

The screenshot displays a development environment with several windows open:

- EXPLORER**: Shows the project structure with files like App.js, App.test.js, index.css, etc.
- EDITOR**: Shows the code for `App.test.js`:

```
src > App.test.js > test('ToDo') callback
1 import React from 'react';
2 import * as ReactDOM from 'react-dom'
3 import {getQueriesForElement} from '@testing-library/dom';
4 import App from './App';

5
6 test('ToDo', () => {
7   const root = document.createElement('div');
8   ReactDOM.render(<App/>, root);
9   const {getByText, getByLabelText} = getQueriesForElement(root);
10
11   // after rendering our component
12   expect(getByText('TODO')).not.toBeNull();
13 });
14
```

- TERMINAL**: Shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (4ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.313s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```

- BROWSER**: Shows the application running at `localhost:3000`. The page title is "React App". The main content area says "TODO" and has a text input field with placeholder "Add todo..." and a button labeled "Add #1".

Testing Using The React Library (3/9)

The screenshot shows a development environment with two main panes. The left pane is a code editor for a 'todo' application. The 'App.test.js' file is open, containing a Jest test for the 'ToDo' component. The test uses the '@testing-library/react' library to render the component and check for text elements. The right pane shows a web browser window at 'localhost:3000' displaying a 'TODO' page with an input field and a button. Below the browser is a terminal window showing the test results.

App.test.js — todo

```
import React from 'react';
import * as ReactDOM from 'react-dom';
import {getQueriesForElement} from '@testing-library/dom';
import App from './App';

test('ToDo', () => {
  const root = document.createElement('div');
  ReactDOM.render(<App/>, root);
  const {getByText, getByLabelText} = getQueriesForElement(root);

  // after rendering our component
  expect(getByText('TODO')).not.toBeNull();
  expect(getByLabelText('Add todo:')).not.toBeNull();
  expect(getByText('Add #1')).not.toBeNull();
});
```

FAIL src/App.test.js
x ToDo (21ms)

● ToDo

 TypeError: expect(...).not.toBeNull is not a function

```
10 |
11 |
> 12 |   expect(getByText('TODO')).not.toBeNull();  
    |          ^
13 |   expect(getByLabelText('Add todo:')).not.toBeNull
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: node

about:blank × React App localhost:3000

TODO

Add todo: Add Todo... Add #1

Testing Using The React Library (4/9)

The screenshot shows a development environment with three main panes:

- EXPLORER** pane on the left displaying project files: node_modules, public, src (containing App.css, App.js, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, README.md).
- EDITOR** pane in the center showing the content of `App.test.js`:

```
App.test.js — todo
src > App.js > App.test.js
  1 import React from 'react';
  2 import * as ReactDOM from 'react-dom';
  3 import {getQueriesForElement} from '@testing-library/dom';
  4 import App from './App';

  5
  6 test('ToDo', () => {
  7   const root = document.createElement('div');
  8   ReactDOM.render(<App/>, root);
  9   const {getByText, getByLabelText} = getQueriesForElement(root);

 10   // after rendering our component
 11   expect(getByText('TODO')).not.toBeNull();
 12   expect(getByLabelText('Add todo:')).not.toBeNull();
 13   expect(getByText('Add #1')).not.toBeNull();
 14 });
 15
 16
```

- TERMINAL** pane at the bottom showing the test results:

```
PASS  src/App.test.js
  ✓ ToDo (10ms)

  Test Suites: 1 passed, 1 total
  Tests:    1 passed, 1 total
  Snapshots: 0 total
  Time:      0.493s, estimated 1s
  Ran all test suites related to changed files.

  Watch Usage: Press w to show more. ⌂
```

- BROWSER** pane on the right showing the application UI titled "TODO" with an input field "Add todo: Add Todo..." and a button "Add #1".

Testing Using The React Library (5/9)

The screenshot displays a development environment with several windows open:

- EXPLORER**: Shows the project structure with files like App.js, App.css, index.css, etc., and a folder named TODO.
- EDITOR**: Shows the code for `App.test.js`. The code imports React and ReactDOM, and uses the `@testing-library/dom` library to test the `App` component. It includes a `test('ToDo', () => { ... })` block that renders the component and checks for specific text elements.
- BROWSER**: Shows a React application titled "TODO". It has a text input field "Add todo:" with placeholder "Add Todo..." and a button "Add #1".
- TERMINAL**: Shows the test results:
 - A red "FAIL" message: `FAIL src/App.test.js`
 - An error message: `x ToDo (7ms)`
 - A failure detail: `● ToDo`
 - A message: `Unable to find a label with the text of: Add todos:`
 - HTML source code of the rendered component:

```
<div>
  <div>
    <h1>
      TODO
    </h1>
    <ul />
```

Testing Using The React Library (6/9)

The screenshot displays a development environment with three main panes:

- Left Pane (Explorer):** Shows the project structure with files like App.js, App.css, and App.test.js.
- Middle Pane (Code Editor):** Displays the `App.test.js` file containing a test suite for a `ToDo` component. The code uses `react`, `ReactDOM`, and `@testing-library/dom` to render the component and check for specific text elements.
- Right Pane (Browser Preview):** Shows a simple "TODO" application with an input field and a button. The URL is `localhost:3000`.

Below the code editor, the terminal shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (10ms)

  Test Suites: 1 passed, 1 total
  Tests:    1 passed, 1 total
  Snapshots: 0 total
  Time:      0.493s, estimated 1s
  Ran all test suites related to changed files.

  Watch Usage: Press w to show more. ⌂
```

Testing Using The React Library (7/9)

The screenshot displays a development environment with several windows open:

- Code Editor (App.test.js — todo):** Shows a Jest test file for a 'ToDo' component. The test imports React and ReactDOM, and uses @testing-library/dom to query the rendered component.
- Browser (localhost:3000):** Shows a simple 'TODO' application with a header, a text input field labeled 'Add todo...', and a button labeled 'Add #1'.
- Terminal (node):** Shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (6ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.509s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌘
```

Testing Using The React Library (8/9)

The screenshot shows a development environment with two main panes. The left pane is a code editor displaying `App.test.js` with the following content:

```
App.test.js — todo
src > App.test.js > test('ToDo') callback
1 import React from 'react';
2 import * as ReactDOM from 'react-dom';
3 import {getQueriesForElement} from '@testing-library/dom';
4 import App from './App';

5
6 test('ToDo', () => {
7   const root = document.createElement('div');
8   ReactDOM.render(<App/>, root);
9   const {getByText, getByLabelText} = getQueriesForElement(root);

10  // after rendering our component
11  getByText('TODOs');
12  getByLabelText('Add todo:');
13  getByText('Add #1');
14 });
15 );
16 );
```

The right pane shows a browser window at `localhost:3000` displaying a "TODO" application. The page has a header "TODO" and a text input field with placeholder "Add todo..." and a button "Add #1". Below the input field, there is some placeholder text: "Add todo..." and "Add #1".

In the bottom left corner of the code editor, there is a terminal window showing a test failure:

```
FAIL src/App.test.js
  x ToDo (7ms)

● ToDo

  Unable to find an element with the text: TODOs. This could be because the text is broken up by multiple elements. In this case, you can provide a function for your text matcher to make your matcher more flexible.

<div>
  <div>
    <h1>
```

Testing Using The React Library (9/9)

The screenshot displays a development environment with several windows open:

- Code Editor (App.test.js — todo):** Shows a Jest test file for a 'ToDo' component. The test imports React and ReactDOM, and uses @testing-library/dom to query the rendered component.
- Browser (localhost:3000):** Shows a simple 'TODO' application with a header, a text input field, and a button labeled 'Add #1'.
- Terminal:** Shows the output of a Jest test run:

```
PASS  src/App.test.js
  ✓ ToDo (6ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.509s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```

Testing Render (1/3)

The screenshot shows a development environment with three main panes:

- Left Pane (Explorer):** Displays the project structure for a "TODO" application. It includes files like App.js, App.css, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- Middle Pane (Code Editor):** Shows the file `App.test.js` containing a test for the "ToDo" component. The code uses `ReactDOM.render` to render the component and `getQueriesForElement` to check for specific text elements.
- Right Pane (Browser Preview):** A browser window titled "React App" at "localhost:3000" displays a "TODO" page with a header, a text input field, and a button labeled "Add #1".

Below the code editor, the terminal output shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (6ms)

  Test Suites: 1 passed, 1 total
  Tests:    1 passed, 1 total
  Snapshots: 0 total
  Time:      0.439s, estimated 1s
  Ran all test suites related to changed files.

  Watch Usage: Press w to show more. ⌂
```

Testing Render (2/3)

The screenshot shows a development environment with two main panes. The left pane is a code editor for a file named `App.test.js` in a project structure labeled `todo`. The code defines a `render` function and a test suite for a `ToDo` component. The right pane shows a browser window at `localhost:3000` displaying a "TODO" application with an input field and a button. Below the code editor, the terminal output shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (23ms)

Test Suites: 1 passed, 1 total
Tests:    1 passed, 1 total
Snapshots: 0 total
Time:      0.684s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```

Testing Render (3/3)

The screenshot shows a development environment with three main panes:

- EXPLORER** pane on the left displaying the project structure:

```
src > App.test.js > ...
  1 import React from 'react';
  2 import {render} from '@testing-library/react';
  3 import App from './App';
  4
  5 test('ToDo', () => {
  6   const {getByText, getByLabelText} = render(<App/>);
  7
  8   // after rendering our component
  9   getByText('TODO');
 10   getByLabelText('Add todo:');
 11   getByText('Add #1');
 12 });
 13
```
- TERMINAL** pane at the bottom showing the test results:

```
PASS  src/App.test.js
  ✓ ToDo (9ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        0.471s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```
- BROWSER** pane on the right showing the application interface with a "TODO" heading and an input field.

Testing Fire Event (1/7)

The screenshot shows a development environment with the following components:

- Explorer:** Shows the project structure with files like App.js, App.css, index.css, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- Editor:** The file `App.test.js` is open, containing a test for the `ToDo` component using `@testing-library/react`. The code uses `fireEvent` to trigger a click event on a button labeled "Add todo:".
- Browser Preview:** A window titled "React App" shows the application's interface with a "TODO" title and a text input field with placeholder "Add Todo...".
- Terminal:** The terminal output shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (8ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.439s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.□
```

Testing Fire Event (2/7)

The screenshot displays a development environment with several windows open:

- EXPLORER**: Shows the project structure with files like App.js, App.css, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- App.test.js — todo**: An editor window containing Jest test code for the App component. It includes two test cases: one for 'ToDo' and one for 'Todo'. Both tests render the component and check for specific text elements using getByText and getLabelText.
- TERMINAL**: A terminal window showing the test results:

```
PASS  src/App.test.js
  ✓ ToDo (8ms)

Test Suites: 1 passed, 1 total
Tests:    1 passed, 1 total
Snapshots: 0 total
Time:      0.439s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌂
```
- Browser Preview**: A window titled "React App" showing the application's interface. It has a header with the word "TODO" and a text input field with placeholder "Add todo..." and a button "Add #1".

Testing Fire Event (3/7)

The screenshot shows a development environment with the following components:

- Code Editor:** An open file is `App.test.js`, which contains Jest test cases for a component named `App`. The tests include rendering the component and checking for specific text elements like 'TODO' and 'Add todo:'.
- Terminal:** The terminal output shows the results of the test run:

```
PASS  src/App.test.js
  ✓ ToDo (8ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.439s, estimated 1s
Ran all test suites related to changed files.
```
- Browser:** A browser window titled "React App" is open at `localhost:3000`. It displays a "TODO" list with an input field labeled "Add todo:" and a button labeled "Add #1".

Testing Fire Event (4/7)

The screenshot shows a development environment with two main panes. The left pane is a code editor for a React application named 'todo'. It displays the file `App.test.js` with the following content:

```
App.test.js — todo
src > App.test.js > test('add items to list') callback
5  test('ToDo', () => {
6    const {getByText, getByLabelText} = render(<App/>);
7
8    // after rendering our component
9    getByText('TODO');
10   getByLabelText('Add todo:');
11   getByText('Add #1');
12 });
13
14 test('add items to list', () => {
15   const {getByText, getByLabelText} = render(<App/>);
16
17   // after rendering our component
18   getByText('TODO');
19   const input = getByLabelText('Add todo:');
20   fireEvent.change(input, {target:{value:'wash car'}});
21   getByText('Add #1');
22 });
23
```

The right pane shows the application's interface. The browser window title is "React App" and the URL is "localhost:3000". The page displays a "TODO" list with an input field labeled "Add todo:" containing "Add #1". Below the input field is a button labeled "Add #1".

The terminal below the code editor shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (8ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.439s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more. ⌘
```

Testing Fire Event (5/7)

The screenshot shows a development environment with several windows open:

- Left Panel (Explorer):** Shows the project structure with files like App.js, App.css, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- Middle Panel (Editor):** The `App.test.js` file is open, containing a test suite for a component named `App`. The test cases involve interacting with text inputs and buttons to add items to a list.
- Bottom Panel (Terminal):** The terminal output shows the test results:
 - PASS** src/App.test.js
 - ✓ ToDo (6ms)
 - ✓ add items to list (14ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 0.546s, estimated 1s
Ran all test suites related to changed files.
- Right Panel (Browser Preview):** A browser window titled "React App" shows the application's UI. It has a header "TODO" and a form with an input field "Add todo:" and a button "Add #1".

Testing Fire Event (6/7)

The screenshot shows a development environment with two main panes. On the left is a code editor in VS Code displaying `App.test.js`. The code is a Jest test for a component named `App`. It includes a setup for rendering the component and interacting with its UI elements using `fireEvent.change` and `fireEvent.click`. The test fails with the message "FAIL src/App.test.js". The failing test is "add items to list". A detailed error message below the test results states: "Unable to find an element with the text: wash cars. This could be because the text is broken up by multiple elements. In this case, you can provide a function for your text matcher to make your matcher more flexible." On the right, a browser window titled "React App" shows the application's interface with a "TODO" heading and a text input field containing "Add Todo...". Below the input is a button labeled "Add #1".

```
getByText('TODO');
getByLabelText('Add todo:');
getByText('Add #1');

test('add items to list', () => {
  const {getByText, getByLabelText} = render(<App/>);

  // after rendering our component
  getByText('TODO');
  const input = getByLabelText('Add todo:');
  fireEvent.change(input, {target:{value:'wash car'}});
  fireEvent.click(getByText('Add #1'));

  // confirm data
  getByText('Add #2');
  getByText('wash cars');
});
```

FAIL src/App.test.js

- ✓ ToDo (9ms)
- ✗ add items to list (11ms)

● add items to list

Unable to find an element with the text: wash cars. This could be because the text is broken up by multiple elements. In this case, you can provide a function for your text matcher to make your matcher more flexible.

```
<body>
<div>
```

Testing Fire Event (7/7)

The screenshot shows a development environment with the following components:

- Left Panel (Explorer):** Shows the project structure with files like App.js, App.css, App.test.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- Middle Panel (Editor):** The `App.test.js` file is open, containing a test suite for a `ToDo` component. It includes rendering the component, interacting with its inputs (label and text input), and confirming the changes. The code uses `getByText`, `getByLabelText`, `fireEvent.change`, and `fireEvent.click`.
- Bottom Panel (Terminal):** The terminal shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (6ms)
  ✓ add items to list (14ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.546s, estimated 1s
Ran all test suites related to changed files.
```
- Right Panel (Browser Preview):** A browser window titled "React App" shows the application's UI. It has a header "TODO" and a form with an input field "Add todo:" and a button "Add #1".

Testing User Event (1/3)

The screenshot shows a development environment with two main windows. On the left is a code editor (VS Code) displaying `App.test.js` with test cases for a 'ToDo' application. On the right is a web browser window showing the application's interface.

Code Editor (VS Code):

```
App.test.js — todo
src > App.test.js > ...
1 import React from 'react';
2 import {render, fireEvent} from '@testing-library/react';
3 import userEvent from "@testing-library/user-event";
4 import App from './App';

5
6 test('ToDo', () => {
7   const {getByText, getByLabelText} = render(<App/>);

8   // after rendering our component
9   getByText('TODO');
10  getByLabelText('Add todo:');
11  getByText('Add #1');

12 });

13
14
15 test('add items to list', () => {
16   const {getByText, getByLabelText} = render(<App/>);

17   // after rendering our component
18   getByText('TODO');
19   const input = getByLabelText('Add todo:');
20   fireEvent.change(input, {target:{value:'wash car'}});
21   fireEvent.click(getByText('Add #1'));

22
23   // confirm data
24   getByText('Add #2');
25   getByText('wash car');

26 });
27
28
29 // userEvent expresses intent better
30 test("user-events allows users to add...", () => {
31
32 });

33 
```

Browser Window:

The browser window displays a simple 'ToDo' application. The title bar says 'React App' and the address bar shows 'localhost:3000'. The page has a header 'TODO' and a text input field with placeholder 'Add Todo...'. Below it is a button labeled 'Add #1'.

Testing User Event (2/3)

The screenshot shows a development environment with the following components:

- Code Editor:** An open file is `App.test.js`, which contains Jest test code for a React application. The code includes rendering the component, interacting with user interface elements (like input fields), and asserting the state changes.
- Terminal:** The terminal window shows the test results:

```
PASS  src/App.test.js
  ✓ ToDo (18ms)
  ✓ add items to list (8ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.461s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```
- Browser Preview:** A browser window titled "React App" is open at `localhost:3000`. It displays a "TODO" list with one item: "Add #1". There is an input field labeled "Add todo:" with the placeholder "Add Todo..." and a button labeled "Add #1".

Testing User Event (3/3)

The screenshot shows a development environment with three main panes:

- EXPLORER** pane on the left displaying project files like `App.js`, `App.css`, and `src` (containing `App.test.js`).
- EDITOR** pane in the center showing `App.test.js` with code for testing user events.
- TERMINAL** pane at the bottom showing test results:

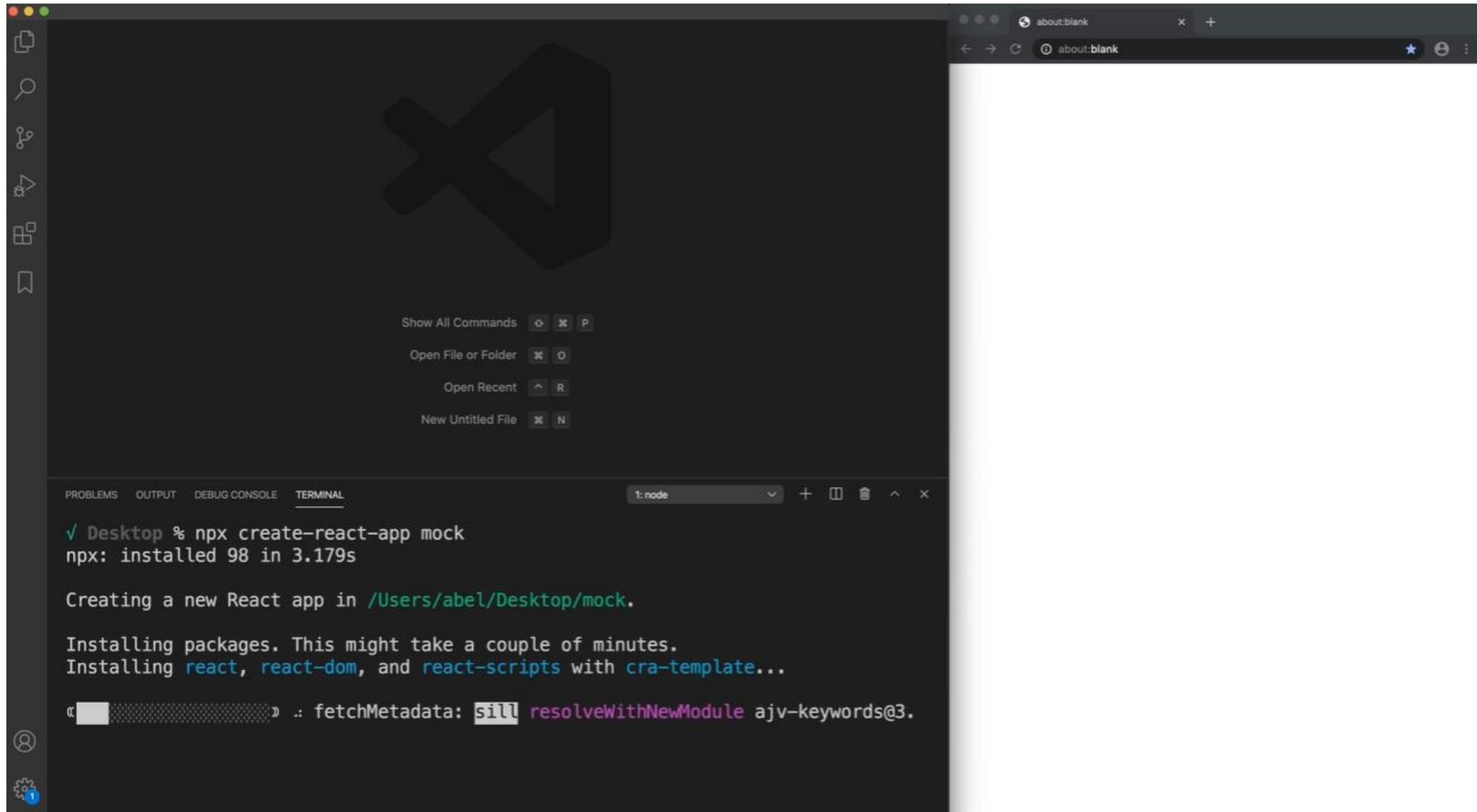
```
PASS  src/App.test.js
  ✓ ToDo (6ms)
  ✓ add items to list (7ms)
  ✓ user-events allows users to add... (28ms)

Test Suites: 1 passed, 1 total
Tests:      3 passed, 3 total
Snapshots:  0 total
Time:       0.543s, estimated 1s
Ran all test suites related to changed files.

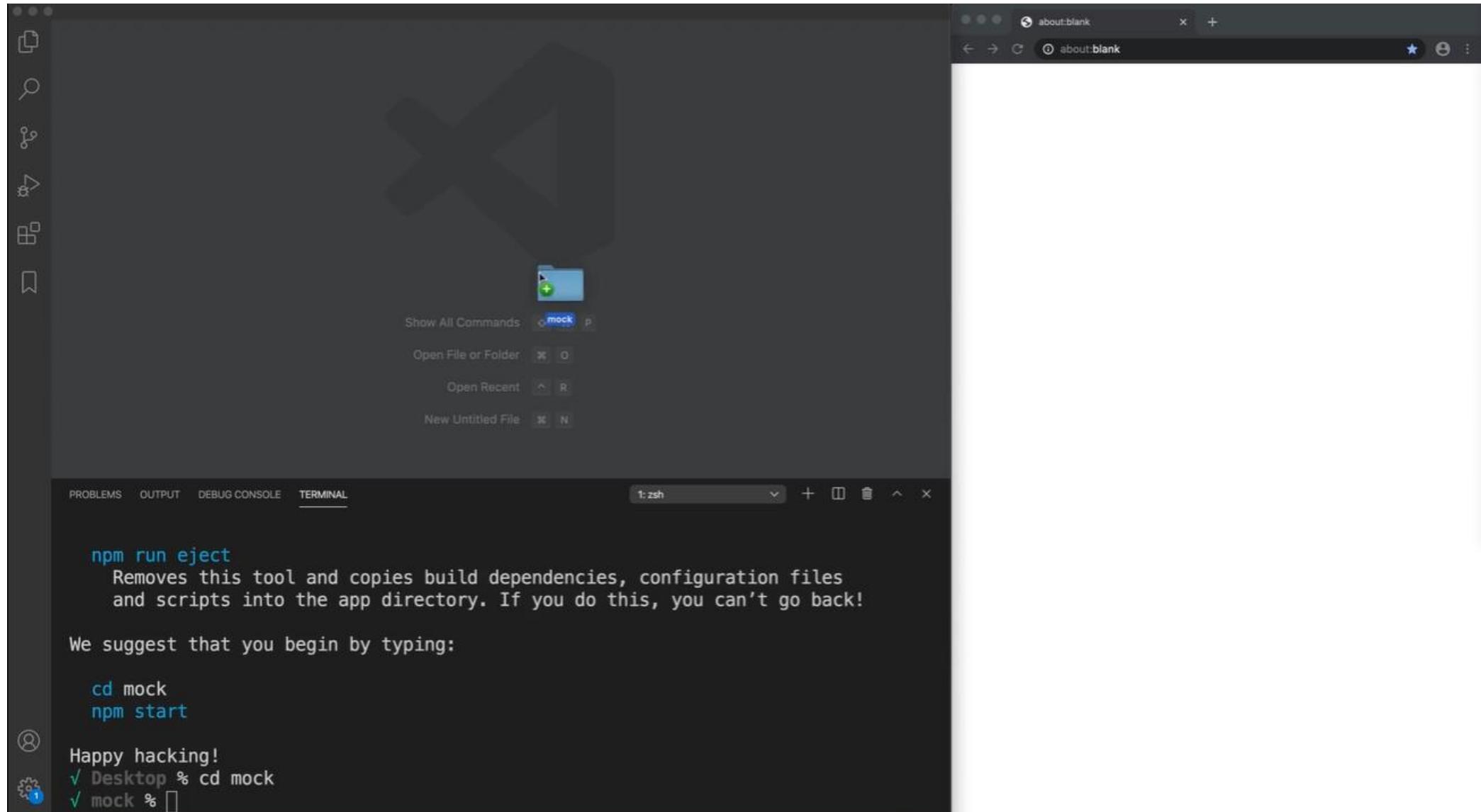
Watch Usage: Press w to show more.
```

To the right, a browser window shows a "TODO" application with an input field containing "Add Todo..." and a button labeled "Add #1".

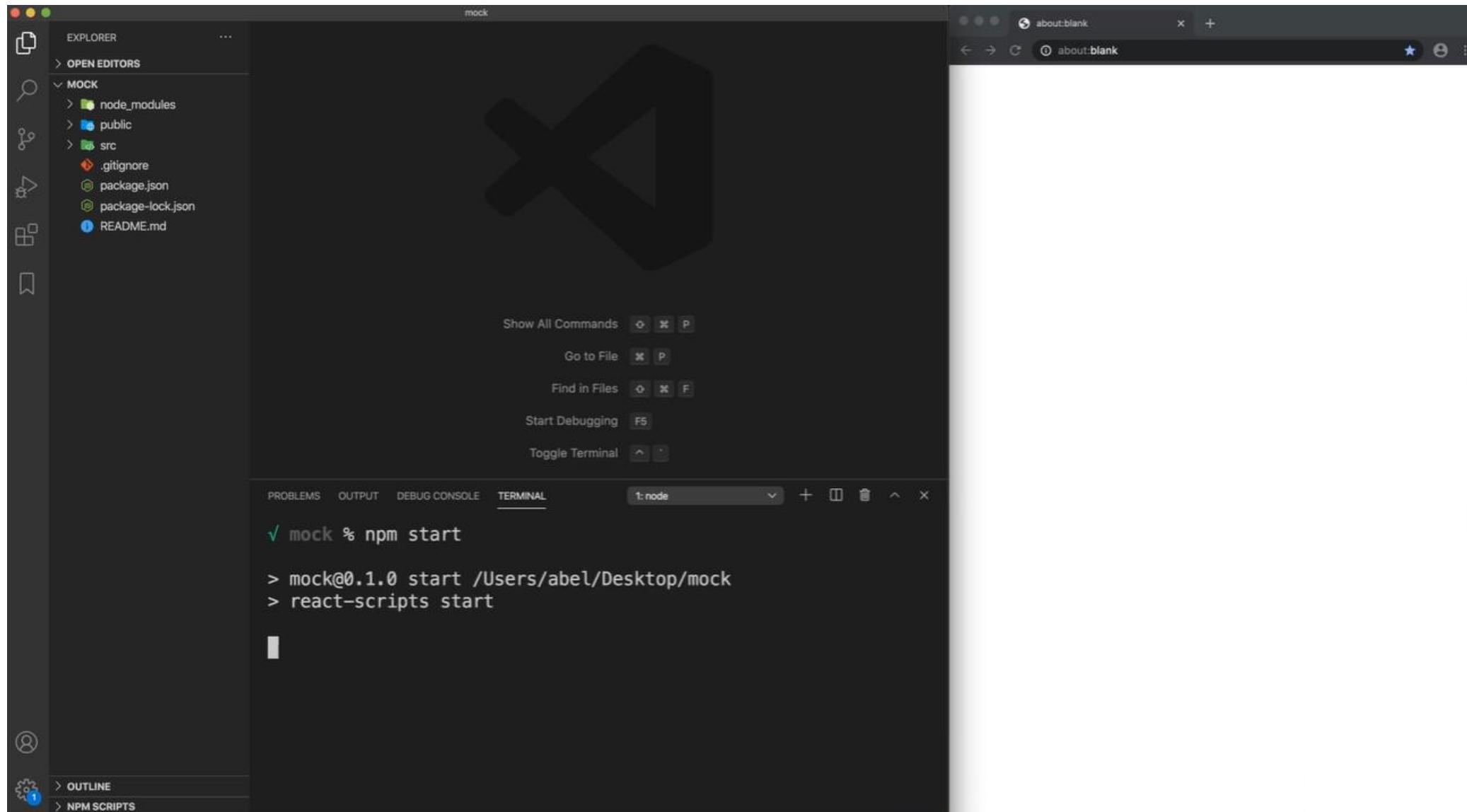
Mock A Dependency (1/11)



Mock A Dependency (2/11)



Mock A Dependency (3/11)



Mock A Dependency (4/11)

The screenshot shows a dark-themed IDE interface, likely Visual Studio Code, with a terminal window displaying the output of a build process for a React application named 'mock'.

Terminal Output:

```
Compiled successfully!
You can now view mock in the browser.
Local: http://localhost:3000
On Your Network: http://10.234.223.129:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
```

Browser Preview:

A browser window titled "React App" is open at "localhost:3000". The page displays the React logo (a blue atom symbol) and the text "Edit src/App.js and save to reload." Below it is a link "Learn React".

Mock A Dependency (5/11)

The screenshot shows a development setup for a React application. On the left, the VS Code interface is visible with the following details:

- EXPLORER:** Shows the project structure with a folder named "MOCK" containing "node_modules" and "public". Inside "src", there are files: App.css, App.js, App.test.js, index.css, index.js, and logo.svg. The file "MyComponent.js" is currently open in the editor.
- Editor Content (MyComponent.js):**

```
import React from "react";
// simulating component
export default function MyComponent(){
  return (
    <div>this is my component</div>
  );
}
```
- TERMINAL:**

```
Compiled successfully!
You can now view mock in the browser.
Local: http://localhost:3000
On Your Network: http://10.234.223.129:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
```

On the right, a browser window titled "React App" is open at "localhost:3000". It displays a blue atomic symbol icon. Below the icon, there is a message: "Edit src/App.js and save to reload." and a link "Learn React".

Mock A Dependency (6/11)

The screenshot displays a developer's environment with three main windows:

- Code Editor (App.js — mock):** Shows the following code:

```
1 import React from 'react';
2 import './App.css';
3
4 export default function App() {
5   return (
6     <div className="App">
7       <MyComponent />
8     </div>
9   );
10 }
11
12
13 export default App;
14
```
- Terminal:** Shows the error message:

```
Failed to compile.

./src/App.js
Line 13:1:  Parsing error: Only one default export allowed
per module.

11 |
12 |
> 13 | export default App;
| ^
14 |
```
- Browser (localhost:3000):** Shows a 404 error page with the message "Failed to compile".

Mock A Dependency (7/11)

The screenshot shows a Visual Studio Code (VS Code) interface with a dark theme. On the left is the Explorer sidebar showing project files: `src`, `node_modules`, `public`, and `MOCK`. The `src` folder contains `App.css`, `App.js`, `App.test.js`, `index.css`, `index.js`, `logo.svg`, `MyComponent.js`, `serviceWorker.js`, and `setupTests.js`. It also includes `.gitignore`, `package.json`, `package-lock.json`, and `README.md`. The `App.js` file is open in the main editor, displaying the following code:

```
import React from 'react';
import './App.css';
import MyComponent from "./MyComponent";

function App() {
  return (
    <div className="App">
      <MyComponent />
    </div>
  );
}

export default App;
```

The terminal at the bottom shows the build process:

```
Compiled successfully!
```

You can now view `mock` in the browser.

Local: <http://localhost:3000>
On Your Network: <http://10.234.223.129:3000>

Note that the development build is not optimized.
To create a production build, use `npm run build`.

To the right, a browser window titled "React App" shows the output: "this is my component".

Mock A Dependency (8/11)

The screenshot shows a developer's workspace with the following components:

- Explorer View:** Shows the project structure with a focus on the `src` folder containing `App.css`, `App.js`, and `App.test.js`.
- Editor View:** Displays the `App.test.js` file, which contains a test for the `App` component. The test checks if it renders a link element with the text "learn react".
- Terminal View:** Shows the build process output:
 - "Compiled successfully!"
 - You can now view `mock` in the browser.
 - Local: `http://localhost:3000`
 - On Your Network: `http://10.234.223.129:3000`
 - Note that the development build is not optimized. To create a production build, use `npm run build`.
- Browser Preview:** A tab labeled "React App" shows the rendered component with the text "this is my component".

Mock A Dependency (9/11)

The screenshot shows a developer's workspace with several open tabs and windows:

- EXPLORER** sidebar: Shows project structure with a tree view. The **MOCK** folder is expanded, showing **node_modules**, **public**, and **src**. Inside **src**, files include **App.css**, **App.js**, **index.css**, **index.js**, **logo.svg**, **MyComponent.js**, **serviceWorker.js**, and **setupTests.js**. Other files like **.gitignore**, **package.json**, **package-lock.json**, and **README.md** are also listed.
- EDITOR**: The **App.test.js — mock** tab is active, displaying Jest test code:

```
import React from 'react';
import { render } from '@testing-library/react';
import App from './App';

jest.mock('./MyComponent', ()=> ()=> (<div>Hello World</div>));

test('mocking test', () => {
  const {container} = render(<App/>);
  expect(container.textContent).toMatch('Hello World');
});
```
- TERMINAL**: Shows the output of a Jest test run:

```
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 1.486s
Ran all test suites related to changed files.

Watch Usage
> Press a to run all tests.
> Press f to run only failed tests.
> Press q to quit watch mode.
> Press p to filter by a filename regex pattern.
> Press t to filter by a test name regex pattern.
> Press Enter to trigger a test run.
```
- BROWSER**: An **about:blank** tab titled **React App** is open at **localhost:3000**, displaying the text **this is my component**.

Mock A Dependency (10/11)

The screenshot shows a development environment with several windows open:

- Explorer:** Shows the project structure with files like `MyComponent.js`, `App.js`, and `App.test.js`.
- Editor:** The `App.test.js` file is open, containing Jest test code. It imports `React` and `@testing-library/react`, and uses `jest.mock` to mock `MyComponent`. The test checks if the container text content matches "Hello World".
- Terminal:** Shows the command `node` running the test. The output indicates 1 failed test.
- Browser:** A tab titled "React App" at `localhost:3000` displays the text "this is my component".

```
App.test.js — mock
src > App.test.js > test('mocking test') callback
1 import React from 'react';
2 import { render } from '@testing-library/react';
3 import App from './App';
4
5 jest.mock('./MyComponent', ()=> ()=> (<div>Hello World</div>));
6
7 test('mocking test', () => {
8   const {container} = render(<App/>);
9   expect(container.textContent).toMatch('Hello Worlds');
10 });
11

at Object.<anonymous> (src/App.test.js:9:33)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        1.735s, estimated 2s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.□
```

Mock A Dependency (11/11)

The screenshot shows a code editor interface with several panes. On the left is the Explorer pane, which lists files and folders including `MyComponent.js`, `App.js`, `App.test.js`, `App.css`, `index.css`, `index.js`, `logo.svg`, `MyComponent.js`, `serviceWorker.js`, `setupTests.js`, `.gitignore`, `package.json`, `package-lock.json`, and `README.md`. The `src` folder is expanded, showing `App.css`, `App.js`, and `App.test.js`. The `App.test.js` file is open in the center editor pane, containing the following Jest test code:

```
import React from 'react';
import { render } from '@testing-library/react';
import App from './App';

jest.mock('./MyComponent', ()=> ()=> (<div>Hello World</div>));

test('mocking test', () => {
  const {container} = render(<App/>);
  expect(container.textContent).toMatch('Hello World');
});
```

The rightmost pane shows a browser window at `localhost:3000` displaying the text "this is my component". Below the code editor is a terminal window showing the test results:

```
PASS src/App.test.js
  ✓ mocking test (13ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.406s, estimated 2s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

Mock ToDo Application API (1/8)

The image shows a development environment with two main windows. On the left is a code editor (VS Code) displaying a test file named `App.test.js`. The file contains a single test case for a component named `App`. The test uses `react`, `@testing-library/react`, and `App` from the local project. It renders the `App` component, interacts with it by clicking an "Add" button and entering text into an input field, and then asserts that the state has been updated correctly. On the right is a web browser window titled "React App" showing the application's interface. The page title is "TODO" and the content area lists three items: "one", "two", and "three". Below the list is a form with a text input field containing "Add Todo..." and a button labeled "Add #4".

```
import React from 'react';
import {render, fireEvent} from '@testing-library/react';
import App from './App';

test('add items to list', () => {
  const {getByText, getByLabelText} = render(<App/>);

  // after rendering our component
  getByText('TODO');
  const input = getByLabelText('Add todo:');
  fireEvent.change(input, {target:{value:'wash car'}});
  fireEvent.click(getByText('Add #1'));

  // confirm data
  getByText('Add #2');
  getByText('wash car');
});
```

Mock ToDo Application API (2/8)

The screenshot shows a development environment with two main windows.

VS Code Editor: The left window displays the file structure and code for a 'todo' application. The 'src' folder contains 'api.js' and other files like 'App.css'. The 'api.js' file contains the following code:

```
// api simulation
export const api = {
  createItem: (newItem) => {
    return Promise.resolve(newItem);
}
```

Browser Window: The right window shows a 'React App' running at 'localhost:3000'. The page title is 'React App' and the URL is 'localhost:3000'. The content area is titled 'TODO' and lists three items: 'one', 'two', and 'three'. Below the list is a form with an input field 'Add todo:' and a button 'Add #4'.

Mock ToDo Application API (3/8)

The image shows a development environment with a code editor and a browser window.

Code Editor (VS Code):

- Explorer:** Shows the project structure with files like `App.test.js`, `api.js`, `App.css`, and `App.js`.
- Editor:** The `App.js` file contains the following code:

```
2 import './App.css';
3 import { api } from './api';
4
5 function App(){
6   const [items, setItems] = React.useState([]);
7   const [value, setValue] = React.useState("");
8
9   function handleSubmit(e){
10     e.preventDefault();
11     api.createItem(value).then((persistedItem) => {
12       setItems([...items, value]);
13       setValue('');
14     })
15   }
16
17   return (
18     <div>
19       <h1>TODO</h1>
20       <TodoList items={items} />
21       <form onSubmit={handleSubmit}>
22         <label htmlFor="new-todo">Add todo:</label>
23         <input
24           id="new-todo"
25           value={value}
26           placeholder="Add Todo..."
27           onChange={e => setValue(e.target.value)}
28         />
29         <button>
30           Add #{items.length + 1}
31         </button>
32       </form>
33     </div>
34   );
35 }
36 
```

Browser: The browser window displays a "React App" at `localhost:3000`. The page has a title "TODO" and a form with an input field and a button labeled "Add #1".

Mock ToDo Application API (4/8)

The screenshot shows a development environment with two main windows. On the left is a code editor (VS Code) displaying `App.test.js` with test code for a 'todo' application. On the right is a web browser window showing the application's interface.

Code Editor (VS Code):

```
App.test.js — todo
src > App.test.js > test('add items to list') callback
1 import React from 'react';
2 import {render, fireEvent, wait} from '@testing-library/react';
3 import App from './App';
4 import { api } from "./api";
5
6 // mock API
7 const mockCreateItem = (api.createItem = jest.fn());
8
9 test('add items to list', async () => {
10   const todoText = "Learn spanish";
11   mockCreateItem.mockResolvedValueOnce(todoText);
12   const { getByText, getByLabelText } = render(<App />);
13
14   // enter content, interact with page
15   const input = getByLabelText('Add todo:');
16   fireEvent.change(input, {target:{value:'wash car'}});
17   fireEvent.click(getByText('Add #1'));
18
19   await wait(() => getByText(todoText));
20
21   expect(mockCreateItem).toBeCalledTimes(1);
22   expect(mockCreateItem).toBeCalledWith(
23     expect.stringContaining('wash car')
24   );
25 });
26 });

// Mock API implementation
const mockCreateItem = jest.fn();
export default mockCreateItem;
```

Browser Window:

The browser window displays a simple 'TODO' application. The title bar says 'React App' and the address bar shows 'localhost:3000'. The page has a header 'TODO' and a text input field with placeholder 'Add todo...'. Below it is a button labeled 'Add #1'.

Mock ToDo Application API (5/8)

The screenshot shows a development environment with the following components:

- Left Panel (Explorer):** Shows the project structure with files like `App.test.js`, `api.js`, `App.css`, `App.js`, and `index.css`.
- Middle Panel (Editor):** Displays the `App.test.js` file containing a Jest test for the application's `todo` feature. The code uses `@testing-library/react` to render the `<App>` component and `jest.fn()` to mock the `api.createItem` function.
- Right Panel (Browser):** Shows a `React App` running at `localhost:3000`. The page has a heading "TODO" and a form with an input field "Add todo:" and a button "Add #1".
- Bottom Panel (Terminal):** Shows the command `npm test` being run, with the output indicating successful tests for the `todo` and `react-scripts` packages.

Mock ToDo Application API (6/8)

The screenshot shows a development environment with two main windows. On the left is a terminal window for Node.js, displaying test results:

```
Tests: 1 failed, 1 total
Snapshots: 0 total
Time: 6.163s
Ran all test suites related to changed files.
```

Below the terminal is a "Watch Usage" section with the following instructions:

- > Press **a** to run all tests.
- > Press **f** to run only failed tests.
- > Press **q** to quit watch mode.
- > Press **p** to filter by a filename regex pattern.
- > Press **t** to filter by a test name regex pattern.
- > Press **Enter** to trigger a test run.

On the right is a browser window titled "React App" showing a "TODO" application. The URL is "localhost:3000". The page has a header "TODO" and a form with "Add todo:" and "Add #1" buttons. A test message "Add todo: Add Todo... Add #1" is displayed below the form.

The VS Code Explorer sidebar shows the project structure:

- TODO
- node_modules
- public
- src
 - api.js
 - App.css
 - App.js
 - App.test.js
 - index.css
 - index.js
 - logo.svg
 - serviceWorker.js
 - setupTests.js
- .gitignore
- package.json
- package-lock.json
- README.md

Mock ToDo Application API (7/8)

The screenshot shows a development environment with the following components:

- Left Panel (Explorer):** Shows the project structure with files like `App.test.js`, `api.js`, and `App.js`.
- Middle Panel (Editor):** Displays the `App.test.js` file containing Jest test code. The test fails at line 19 with the message "FAIL src/App.test.js (5.736s)" and the error "add items to list (4771ms)".
- Bottom Panel (Terminal):** Shows the failing test result and a detailed error message about finding an element with the text "Learn spanish".
- Right Panel (Browser Preview):** Shows the application running at `localhost:3000`. The page title is "React App" and the content is "TODO". There is an input field labeled "Add todo:" with placeholder "Add Todo..." and a button labeled "Add #1".

Mock ToDo Application API (8/8)

The screenshot shows a development environment with the following components:

- Left Panel (Explorer):** Shows the project structure with files like App.test.js, api.js, App.css, App.js, index.css, index.js, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, package-lock.json, and README.md.
- Middle Panel (Editor):** Displays the `App.test.js` file containing Jest test code. The code uses Jest's `mockResolvedValueOnce` to mock the API's `createItem` method. It then renders the `<App>` component, enters "Learn spanish" into the input field, and clicks the "Add #1" button. Finally, it waits for the item to appear in the list and asserts that the `createItem` method was called once.
- Bottom Panel (Terminal):** Shows the test results:
 - PASS** `src/App.test.js`
 - `✓ add items to list (35ms)`
 - Test Suites:** `1 passed`, `1 total`
 - Tests:** `1 passed`, `1 total`
 - Snapshots:** `0 total`
 - Time:** `1.752s`, estimated `6s`

Ran all test suites related to changed files.
- Right Panel (Browser Preview):** Shows the application interface titled "TODO". It has a header with "Add todo:" and "Add #1" buttons. Below is a list area where the "wash car" item is visible.



© MIT xPRO 2021. All rights reserved.