Class Components and Lifecycle Methods

Dr Harshad Prajapati
28 Nov 2023

ES6 Classes

ES6 (ECMAScript 2015) Classes

- ES6 introduced classes.
- Class is an entity that describes blueprint to create instances/objects of the entity.
- A constructor() method is added in class.
- The constructor method is called every time an object of the class is instantiated/created.
- The constructor() is used to initialize the properties.

ES6: Class Inheritance

- A new class (specialized) can be created from existing class (generalized) by using extends keyword.
- The specialized class inherits all the abilities of the generalized class and can add its own specialized abilities.
- The derived class can call constructor of base class using super(). We can also pass parameters through it.

ES6: Static Methods in a Class

- Static methods are defined for a class, not for instances or objects.
- We invoke static methods on the name of the class.

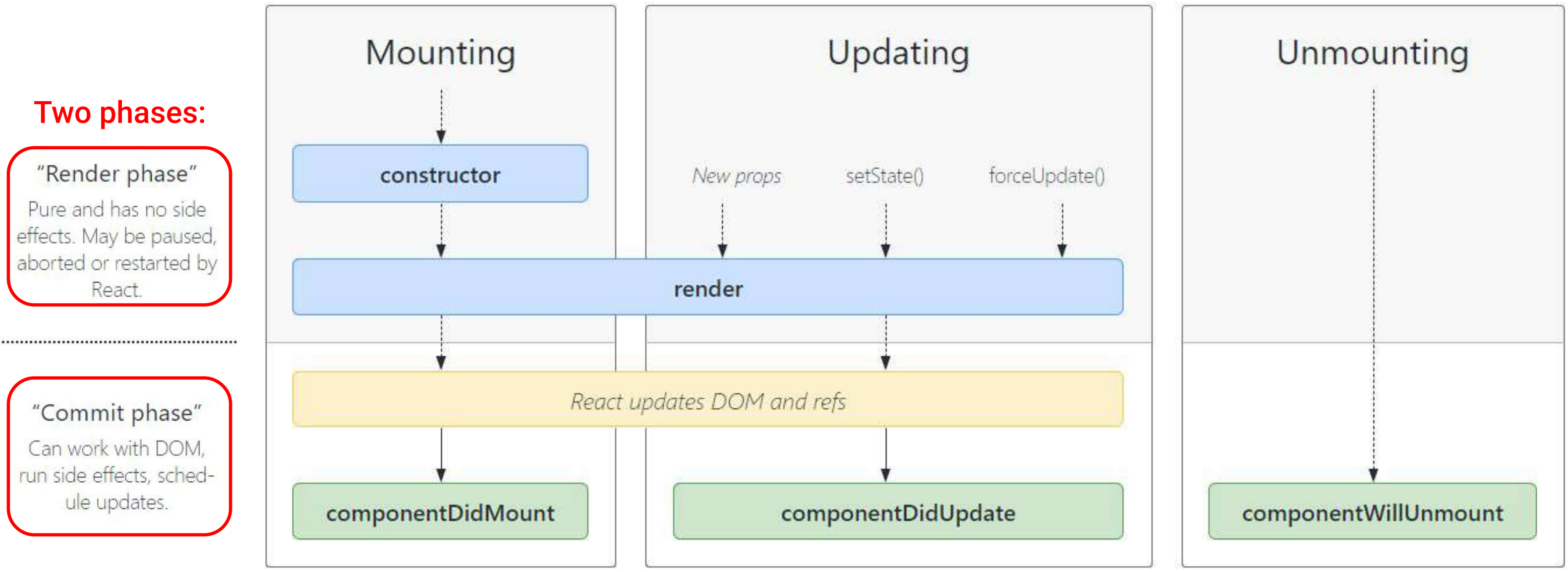
Lifecycle of React Class Components

Phases of React Components

- There are **four phases** through which a react component goes:
  - **Mounting**: When an **instance** of a **component** is being **created** and **inserted** into the **DOM**.
  - **Updating**: When a **component** is being **re-rendered** as a result of **changes** to either its **props** or **state**.
  - **Unmounting**: When a **component** is being **removed** from the **DOM**.
  - **Error Handling**: When there is an **error during rendering**, in a **lifecycle method**, or in the **constructor of any child component**.

Source: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

Common Lifecycle Methods



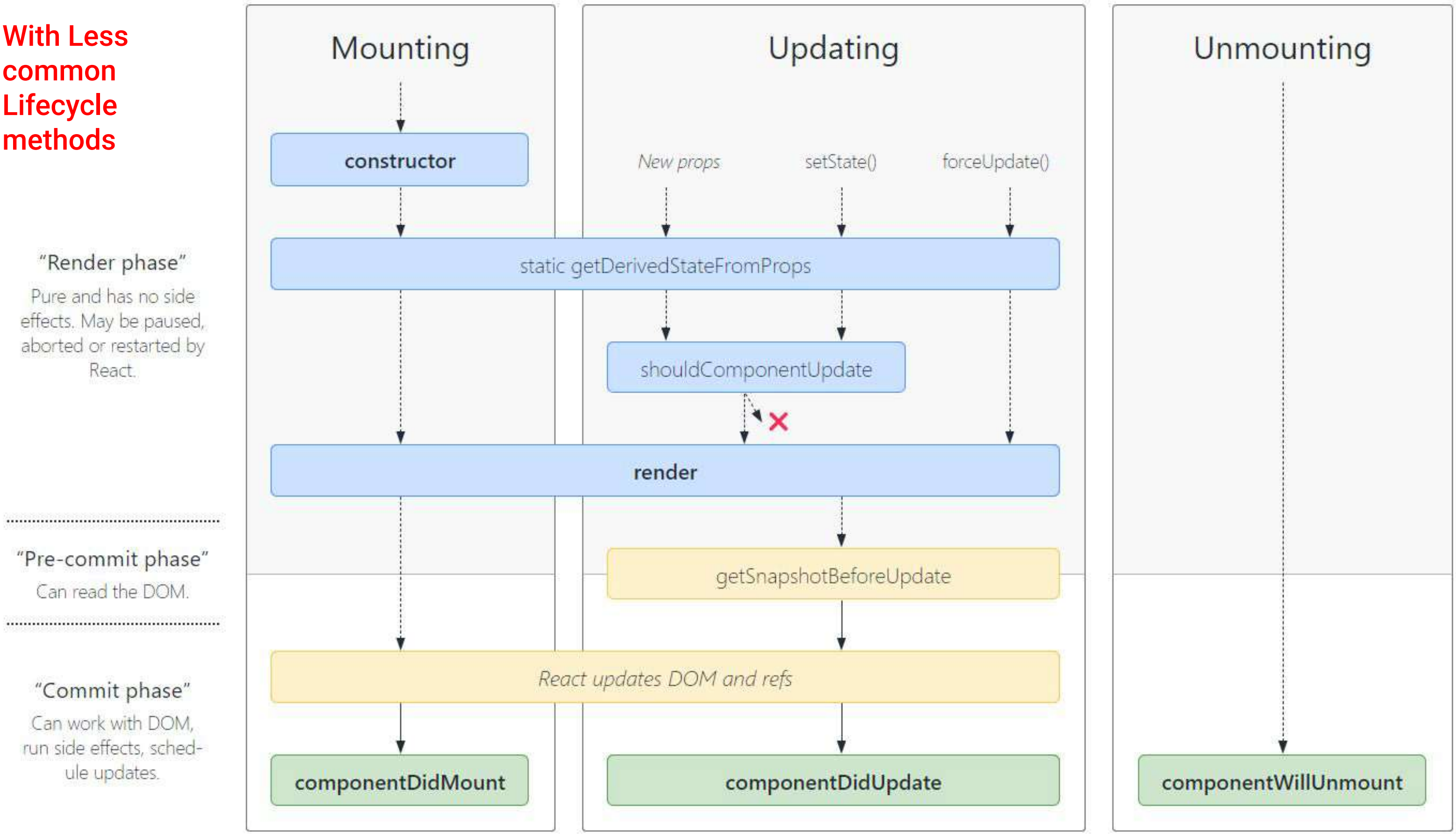
Two phases:

"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React.

"Commit phase"
Can work with DOM, run side effects, schedule updates.

Source: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

With Less common Lifecycle methods



Mounting Phase

- The following **methods** are **called** in the **given order** when an instance of a **component** is being **created** and **inserted** into the **DOM**.
  - constructor()
  - static getDerivedStateFromProps()
  - render()
  - componentDidMount()

Source: https://legacy.reactjs.org/docs/react-component.html

Updating Phase

- An update in props or state of a component triggers re-rendering of the component.
- The following methods are called in the given order when a component is being re-rendered.
  - static getDerivedStateFromProps()
  - shouldComponentUpdate()
  - render()
  - getSnapshotBeforeUpdate()
  - componentDidUpdate()

Source: https://legacy.reactjs.org/docs/react-component.html

Unmounting Phase

- The following method is called when a component is being removed from the DOM:
  - componentWillUnmount()

Source: https://legacy.reactjs.org/docs/react-component.html

Error Handling Phase

- Errors can occur during:
  - rendering.
  - in lifecycle method.
  - in constructor of any child component.
- The following methods are called when error occurs:
  - static getDerivedStateFromError()
  - componentDidCatch()

Lifecycle Methods

render()

- The render() method is the only required method in a class component.
- When called, render() should examine this.props and this.state and return one of the following:
  - React elements: typically created with JSX.
  - Arrays and fragments.
  - String and numbers.
  - Booleans or null or undefined.

render()

- The render() function should be pure.
  - It does not modify component state.
  - It returns the same result each time it is invoked.
  - It does not directly interact with the browser.
- If we need to interact with browser, we need to perform such work in componentDidMount() or other lifecycle methods.
  - For example, bring focus in some element.
- The render() will not be invoked if shouldComponentUpdate() returns false.

constructor()

- The constructor is called before the component is mounted.
- When implementing our constructor, we should call super(props).
- Use of constructor:
  - Initializing local state by assigning an object to this.state.
  - Binding event handler methods to an instance.
- If we do not initialize state and we don't bind methods, we do not need to write constructor.

constructor()

- Dealing with state inside constructor:
  - We should not call setState in the constructor.
  - Instead, we need to assign initial state to this.state directly in the constructor.
- What should be avoided inside constructor?
  - Avoid use of any side-effects or subscriptions in the constructor.
    - For such cases, we should use componentDidMount() method.

componentDidMount()

- The componentDidMount() is invoked immediately after the component is mounted (inserted into the DOM tree).
- This method is a good place for the following:
  - Load data from a remote endpoint. (Using AJAX call)
  - Setup any subscriptions. (Connect to External Web API)
  - Set timers using setTimeout() or setInterval() for periodic or time-based activity.
- If we setup any subscriptions in the componentDidMount(), we should unsubscribe the subscription in componentWillUnmount().
  - Same way cancel timers.

Calling setState in componentDidMount()

- We can call setState() immediately in componentDidMount().
- This will trigger extra rendering, but it will happen before the browser updates the screen.

componentDidUpdate()

- componentDidUpdate(prevProps, prevState, snapshot)
  - The componentDidUpdate() is invoked immediately after DOM updating occurs.
  - It is not called for the initial render.
- We can use this method to operate on the DOM node when the component has been updated.
  - We can use it to do network requests. We can decide to make new requests based on comparison of current props to previous props.
- If our component implements getSnapshotBeforeUpdate(), the value it returns will be passed as third parameter to the componentDidUpdate().

Calling setState in componentDidUpdate()

- We may call setState() immediately in componentDidUpdate(), but it must be done based on some condition.
  - Otherwise, it will cause an infinite loop.
- The componentDidUpdate() will not be invoked if shouldComponentUpdate() returns false.

componentWillUnmount()

- The componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. (Removed from DOM)
- We can perform any necessary cleanup in this method.
  - Invalidating timers.
  - Cancelling network requests.
  - Cleaning up subscriptions.
- We should never call setState() in componentWillUnmount() as the component will never be re-rendered after this method is invoked.

shouldComponentUpdate(nextProps, nextState)

- The shouldComponentUpdate() decides whether render() method should be called or not.
- If shouldComponentUpdate() returns
  - true, then render() method will get called.
  - false, then the component will not update, i.e., render() method will not get called.

getDerivedStateFromProps(nextProps, nextState)

- The getDerivedStateFromProps() method is called before rendering.
- It is called every time before render() method gets called.
- This method takes props and state as arguments.
- This method is used to setup state object based on values of props.
- The method returns an object with changes to the state object.

getSnapshotBeforeUpdate(prevProps, prevState)

- The getSnapshotBeforeUpdate() is invoked right before the most recently rendered output (virtual DOM) is committed to the actual DOM.
- Even after the component is rendered, we can access before update what were previous props and previous state using the method getSnapshotBeforeUpdate().
- It allows our component to capture some information from the DOM (e.g. scroll position) before it is potentially changed.
- Any value returned by this method will be passed as a parameter to componentDidUpdate().
  - If we write getSnapshotBeforeUpdate(), then we also need to write componentDidUpdate().

Required Development Environment

Required Environment for Development in React

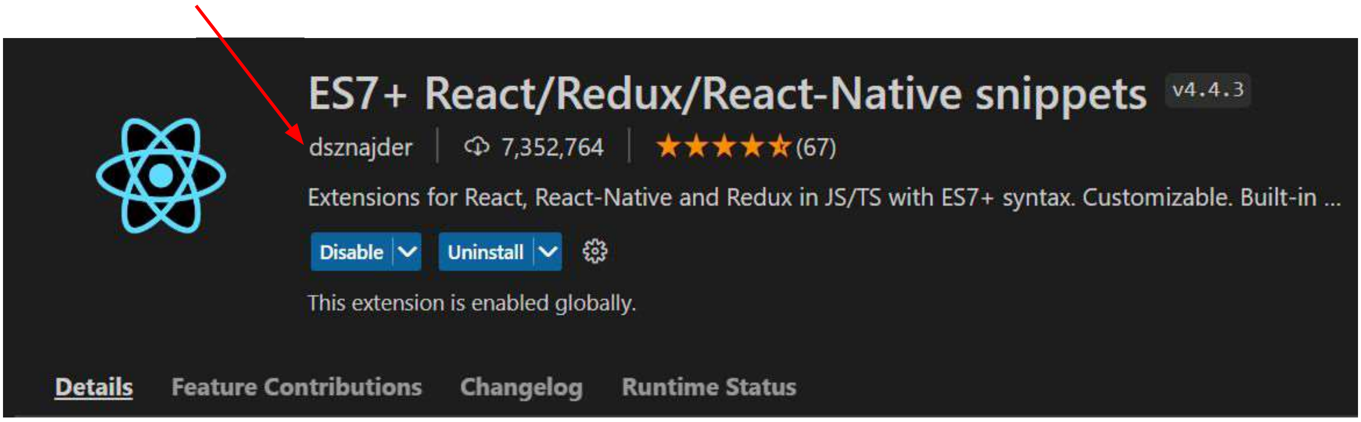
- For **production code**, we need to setup React environment for react application development.
- Download and install latest version of the following:

Visual Studio Code (VS Code) from Microsoft.
(https://code.visualstudio.com/)

Node.js
(https://nodejs.org/en/download)

Node Package Manager
Gets installed along with Node.js

Required Plugins in VS Code



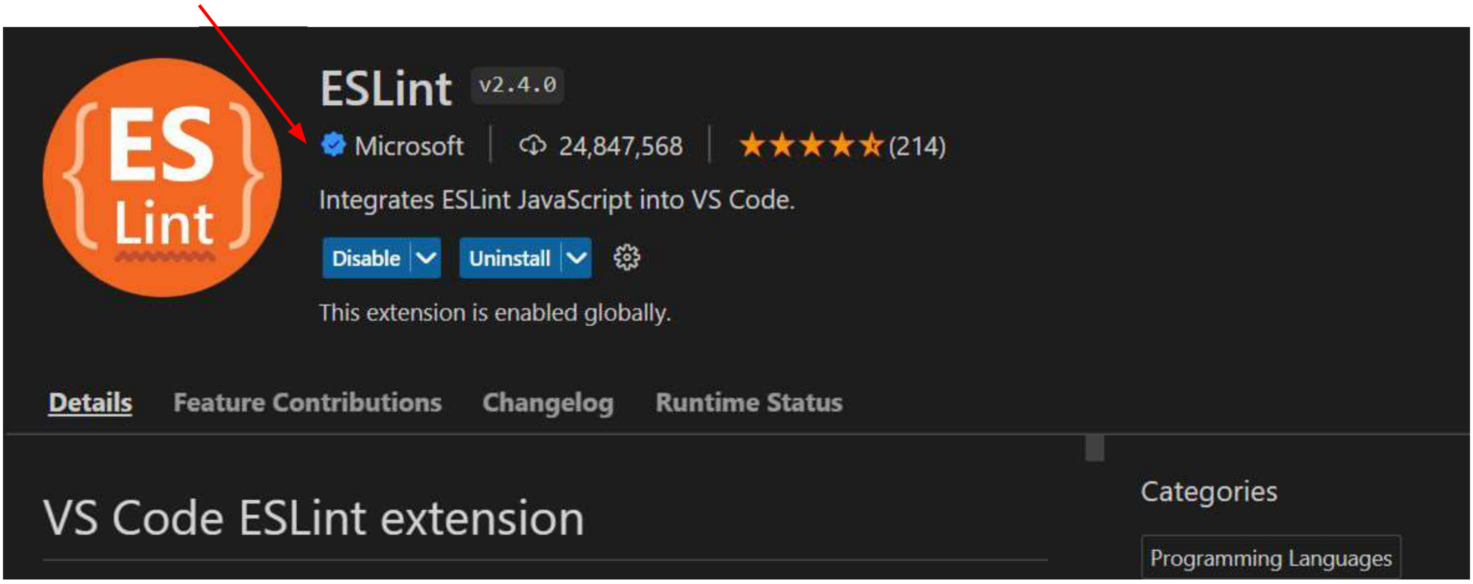
ES7+ React/Redux/React-Native snippets v4.4.3

dsznajder | 7,352,764 | ★★★★☆ (67)

Extensions for React, React-Native and Redux in JS/TS with ES7+ syntax. Customizable. Built-in ...

This extension is enabled globally.

Required Plugins in VS Code



ESLint v2.4.0

Microsoft | 24,847,568 | ★★★★★ (214)

Integrates ESLint JavaScript into VS Code.

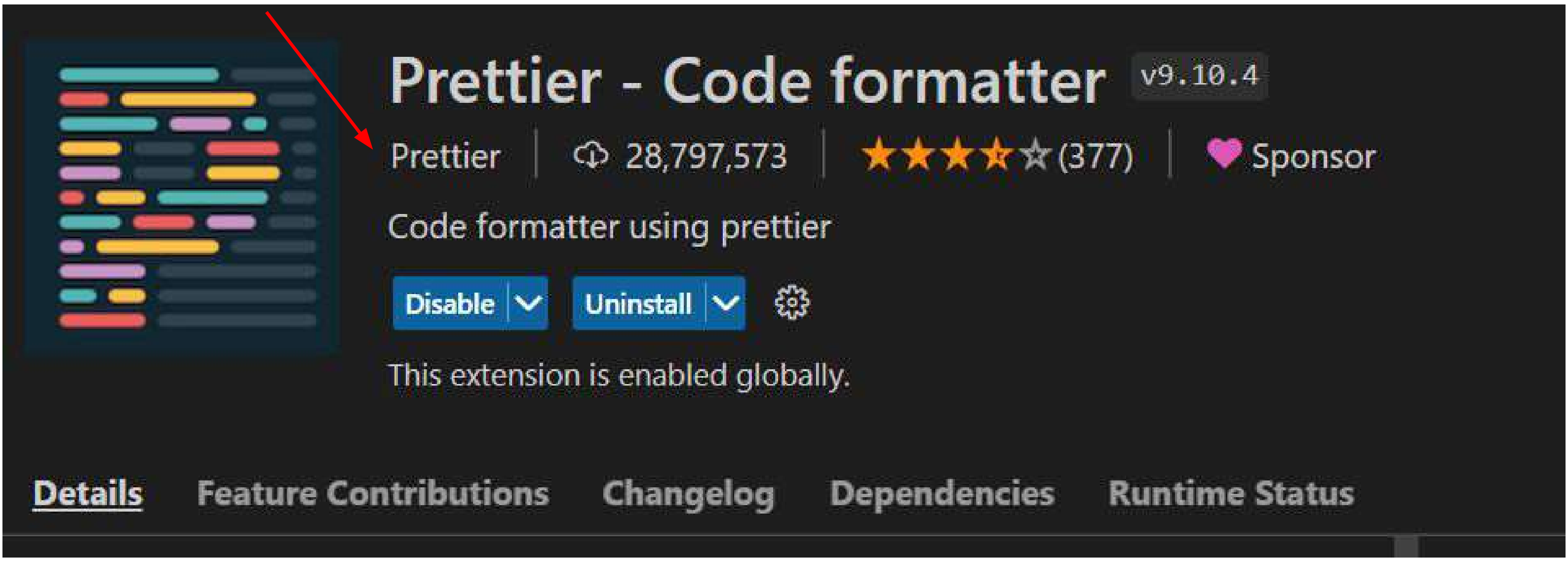
This extension is enabled globally.

VS Code ESLint extension

Categories

Programming Languages

Required Plugins in VS Code



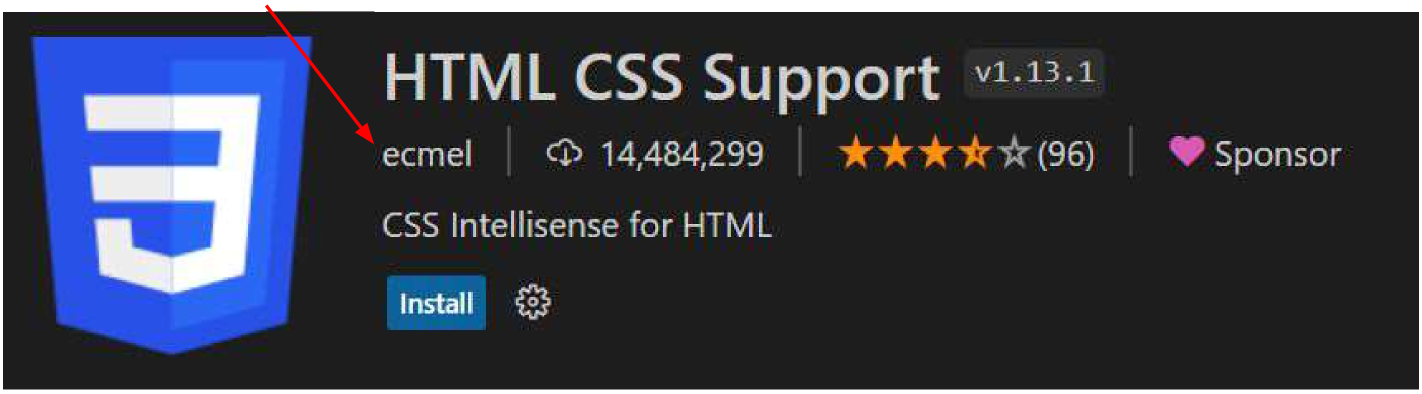
Prettier - Code formatter v9.10.4

Prettier | 28,797,573 | ★★★★☆ (377) | Sponsor

Code formatter using prettier

This extension is enabled globally.

Required Plugins in VS Code

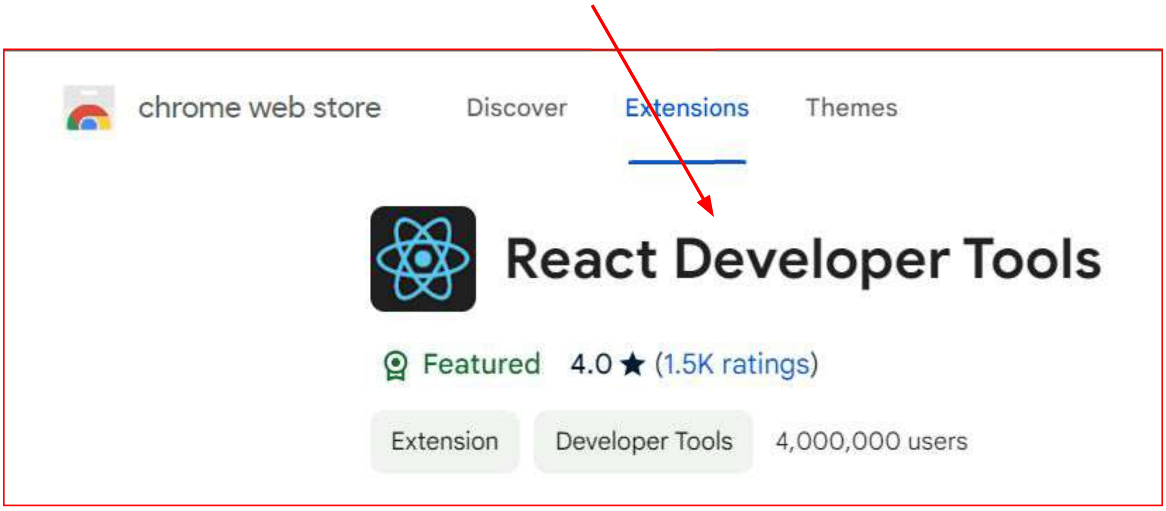


HTML CSS Support v1.13.1

ecmel | 14,484,299 | ★★★★☆ (96) | Sponsor

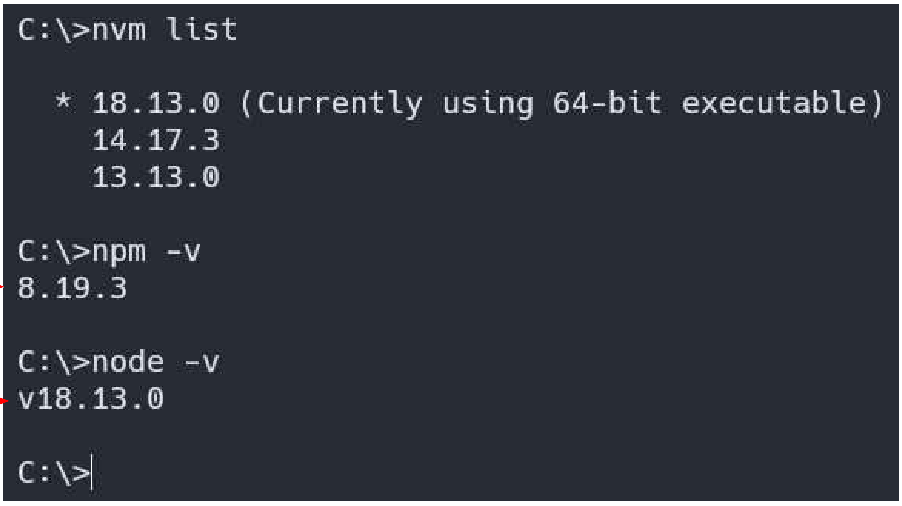
CSS Intellisense for HTML

Required Plugin in Chrome Browser



My Configuration

- I use nvm to manage multiple versions of node.js.
- You need to install latest version of Node.js (As of writing on 29 Nov 2023, it is 20.10.0)



```
C:\>nvm list

  * 18.13.0 (Currently using 64-bit executable)
    14.17.3
    13.13.0

C:\>npm -v
8.19.3

C:\>node -v
v18.13.0

C:\>
```

npm version

node version

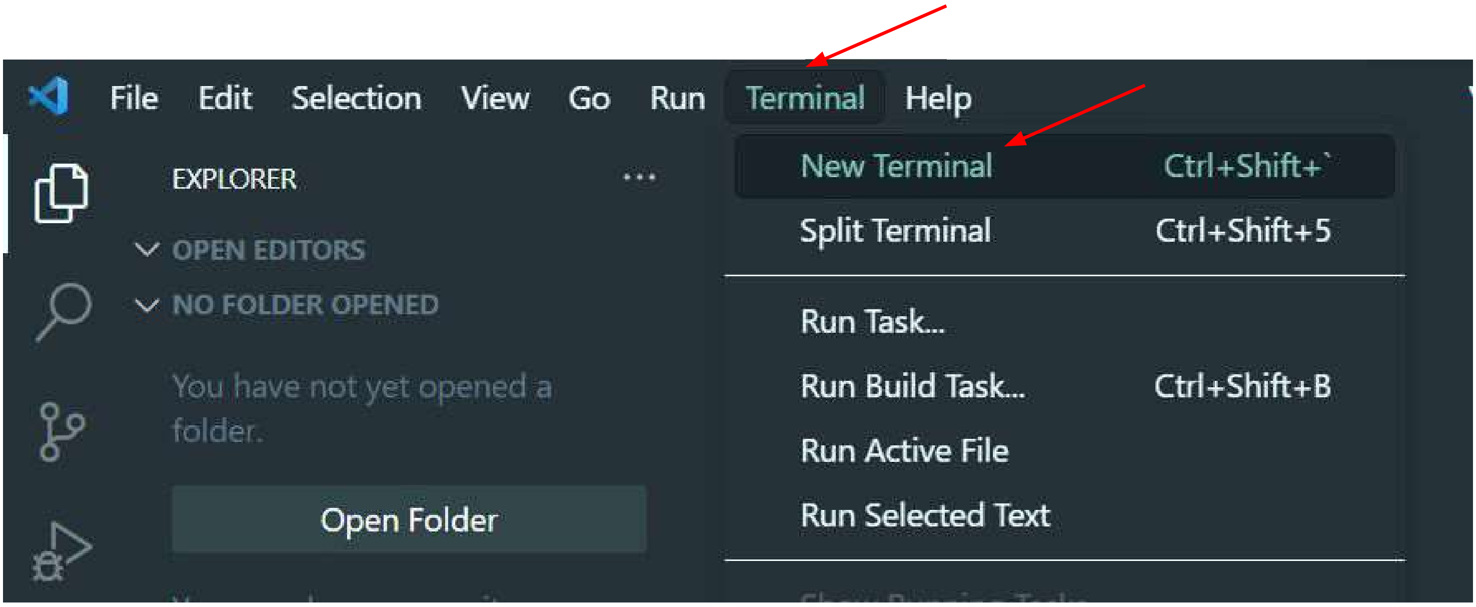
Example: Create and Run React App

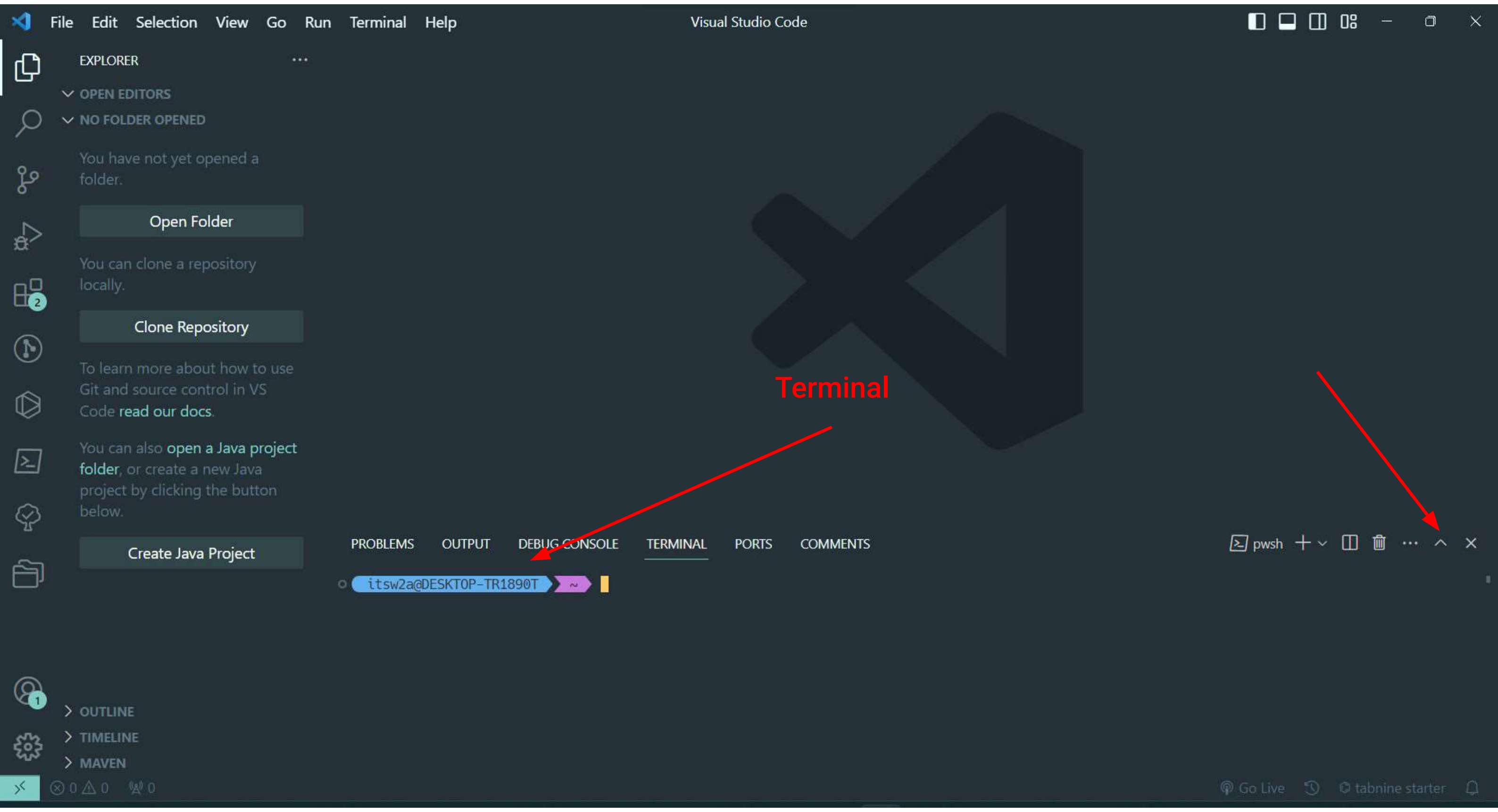
Example: Create React App

- We need to perform the following steps:
- Start VSCode
  - Start a terminal.
  - Create a project named myreactapp using the following command:
    - npx create-react-app myreactapp
  - Go inside the project directory
    - cd myreactapp
  - Run our application in development mode using the following command:
    - npm start

Start VSCode

- Start a new terminal

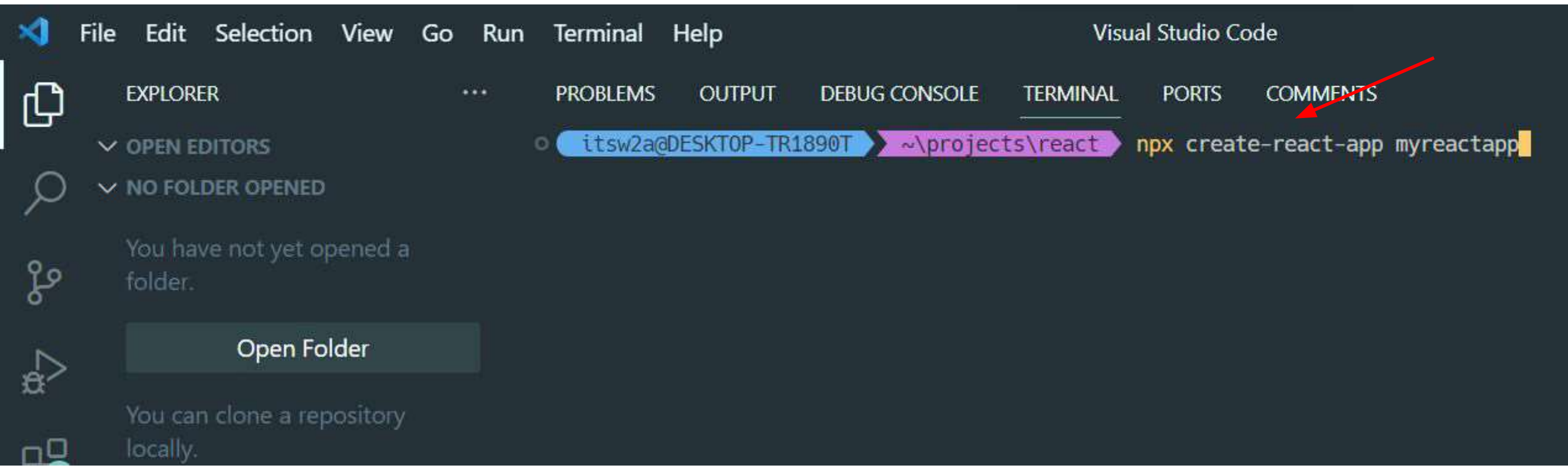



File
Edit
Selection
View
Go
Run
Terminal
Help
Visual Studio Code
EXPLORER
OPEN EDITORS
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
You can clone a repository locally.
Clone Repository
To learn more about how to use Git and source control in VS Code read our docs.
You can also open a Java project folder, or create a new Java project by clicking the button below.
Create Java Project
Terminal
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS
pwsh
OUTLINE
TIMELINE
MAVEN
Go Live
tabnine starter

```
itsw2a@DESKTOP-TR1890T ~
```

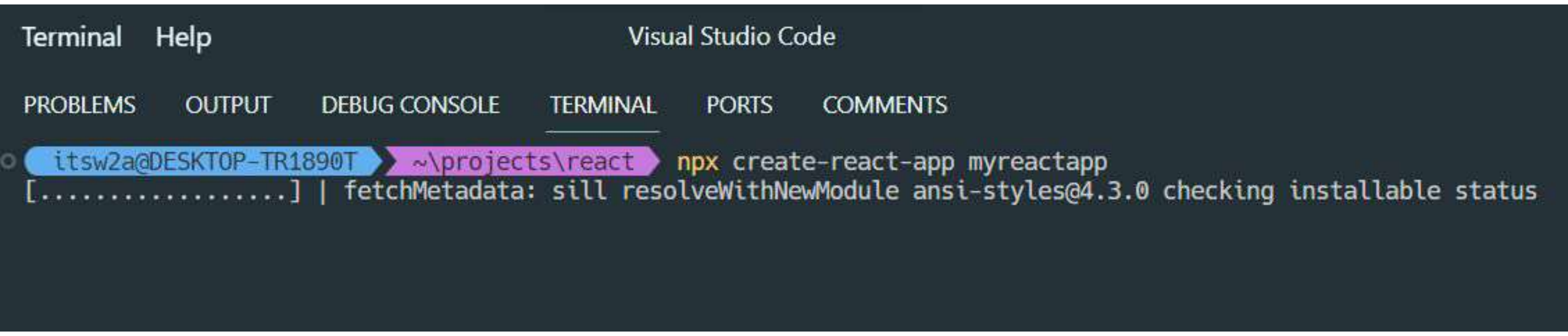
Create a react app

- Create a react app using the following command:
  - npx create-react-app myreactapp



```
itsw2a@DESKTOP-TR1890T  ~\projects\react  npx create-react-app myreactapp
```

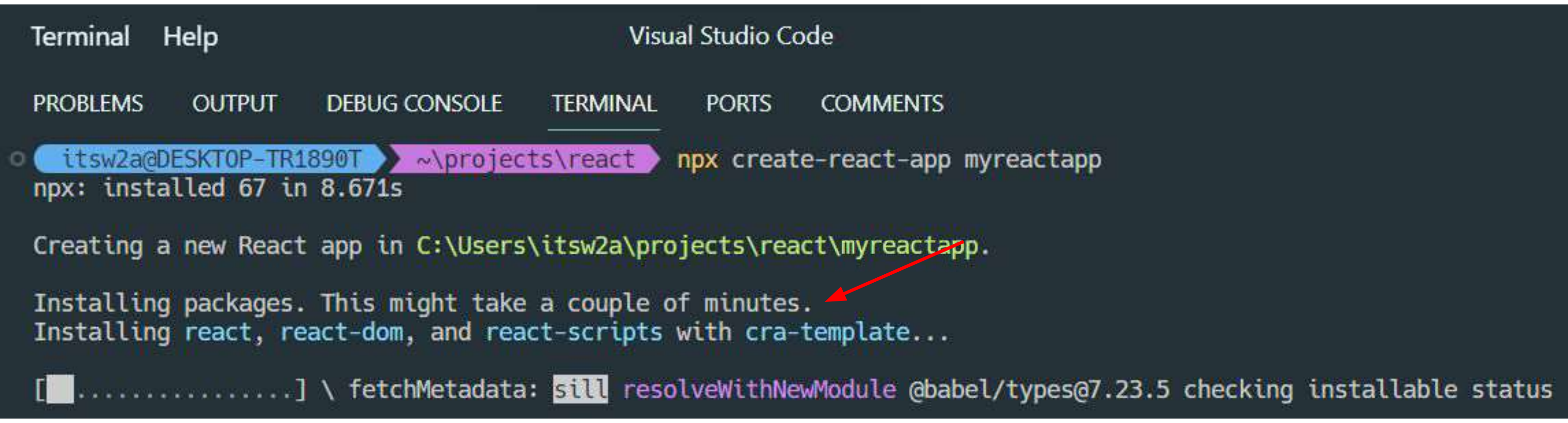
Create a react app



```
itsw2a@DESKTOP-TR1890T  ~\projects\react  npx create-react-app myreactapp
[.................] | fetchMetadata: sill resolveWithNewModule ansi-styles@4.3.0 checking installable status
```

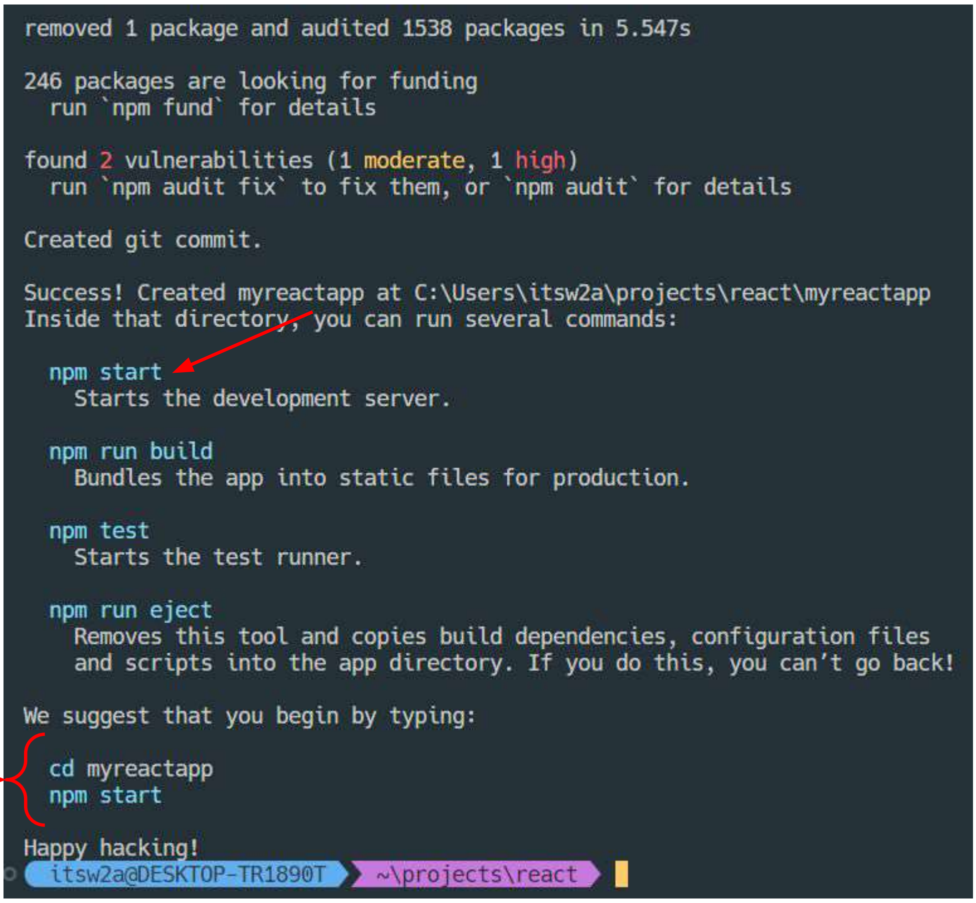
Create a react app

- It will take couple of minutes to create react project.
  - Need good internet connection.



```
itsw2a@DESKTOP-TR1890T  ~\projects\react  npx create-react-app myreactapp
npx: installed 67 in 8.671s

Creating a new React app in C:\Users\itsw2a\projects\react\myreactapp.

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

[█...............] \ fetchMetadata: sill resolveWithNewModule @babel/types@7.23.5 checking installable status
```

The project gets created successfully

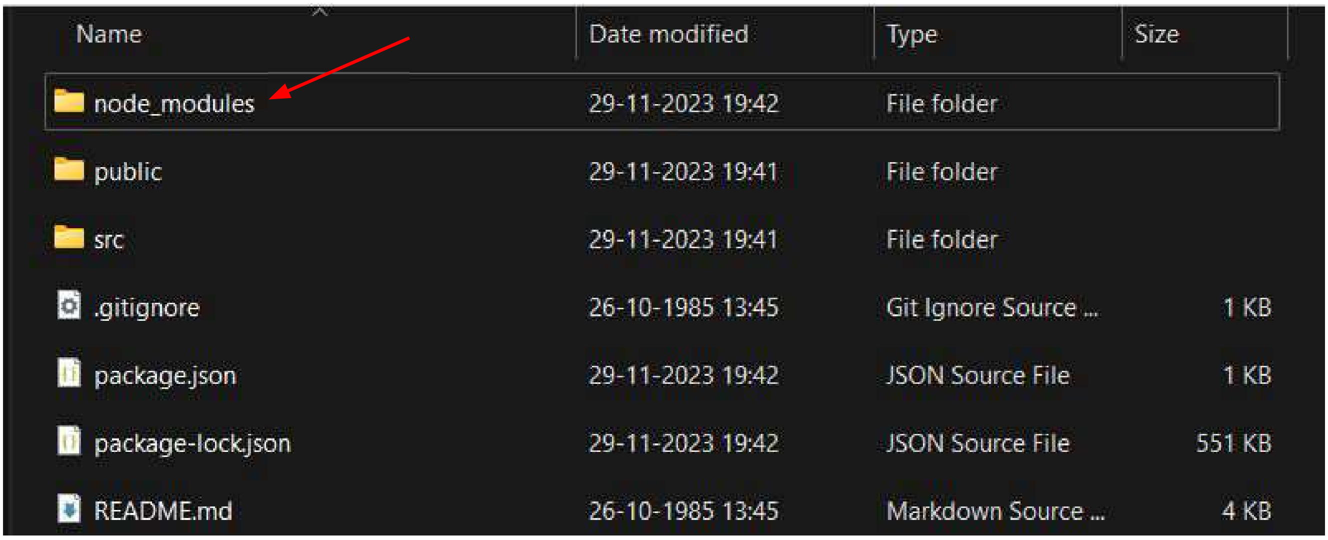


```
removed 1 package and audited 1538 packages in 5.547s

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

found 2 vulnerabilities (1 moderate, 1 high)
  run `npm audit fix` to fix them, or `npm audit` for details

Created git commit.

Success! Created myreactapp at C:\Users\itsw2a\projects\react\myreactapp
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 myreactapp
  npm start

Happy hacking!
itsw2a@DESKTOP-TR1890T  ~\projects\react  █
```

Next, perform these two steps

The content of myreactapp

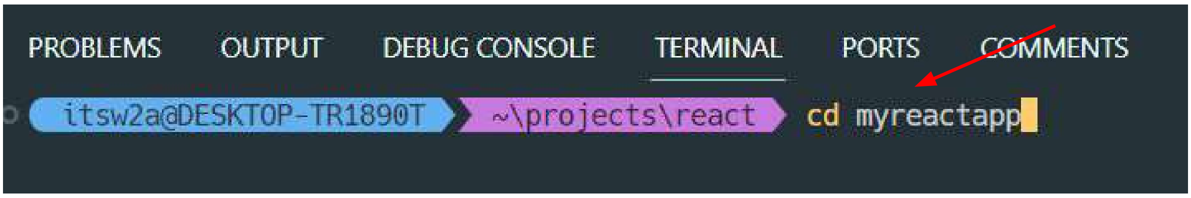
- The project size is around 200+MB.
  - Major space used by node_modules.



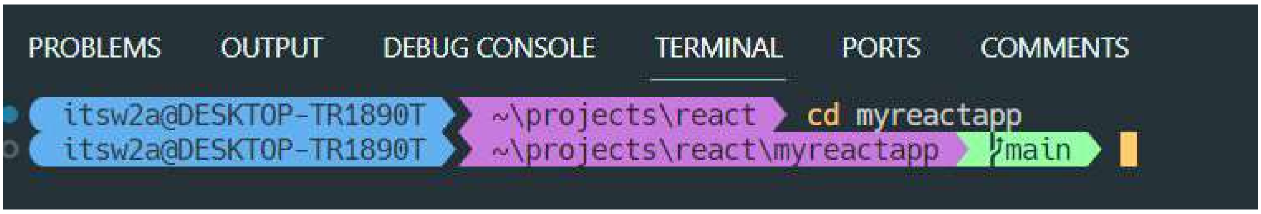
Name	Date modified	Type	Size
node_modules	29-11-2023 19:42	File folder	
public	29-11-2023 19:41	File folder	
src	29-11-2023 19:41	File folder	
.gitignore	26-10-1985 13:45	Git Ignore Source ...	1 KB
package.json	29-11-2023 19:42	JSON Source File	1 KB
package-lock.json	29-11-2023 19:42	JSON Source File	551 KB
README.md	26-10-1985 13:45	Markdown Source ...	4 KB

Change to Project Directory

- Change to the project directory using the following command:
  - cd myreactapp



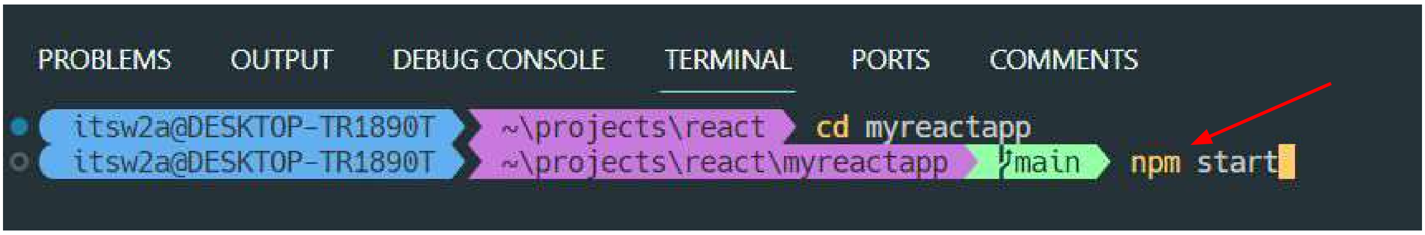
```
itsw2a@DESKTOP-TR1890T  ~\projects\react  cd myreactapp
```



```
itsw2a@DESKTOP-TR1890T  ~\projects\react  cd myreactapp
itsw2a@DESKTOP-TR1890T  ~\projects\react\myreactapp  main
```

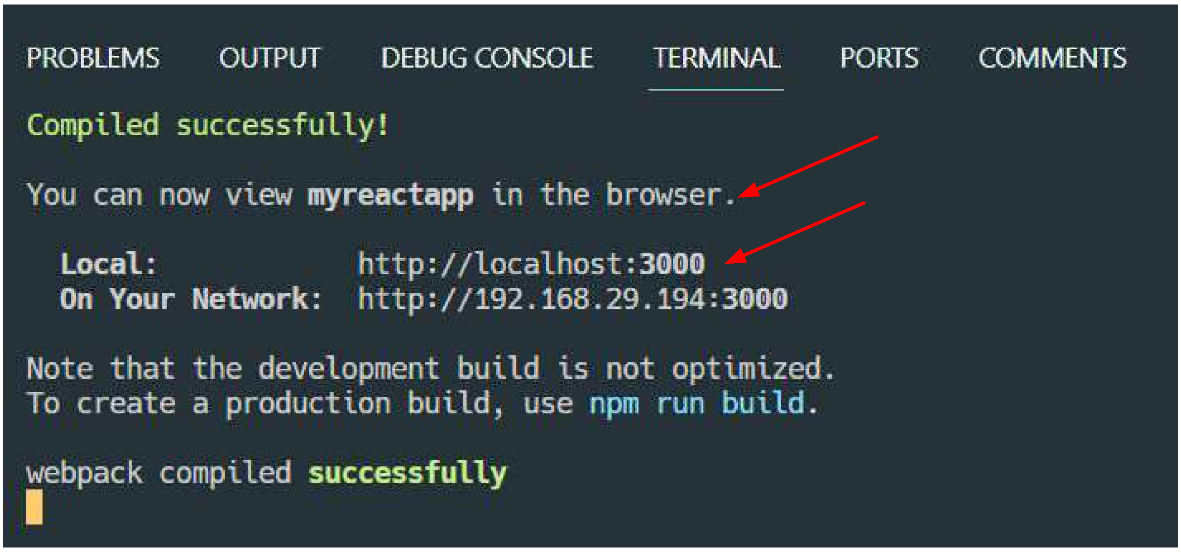
Run Application

- Run Application using the following command:
    - npm start



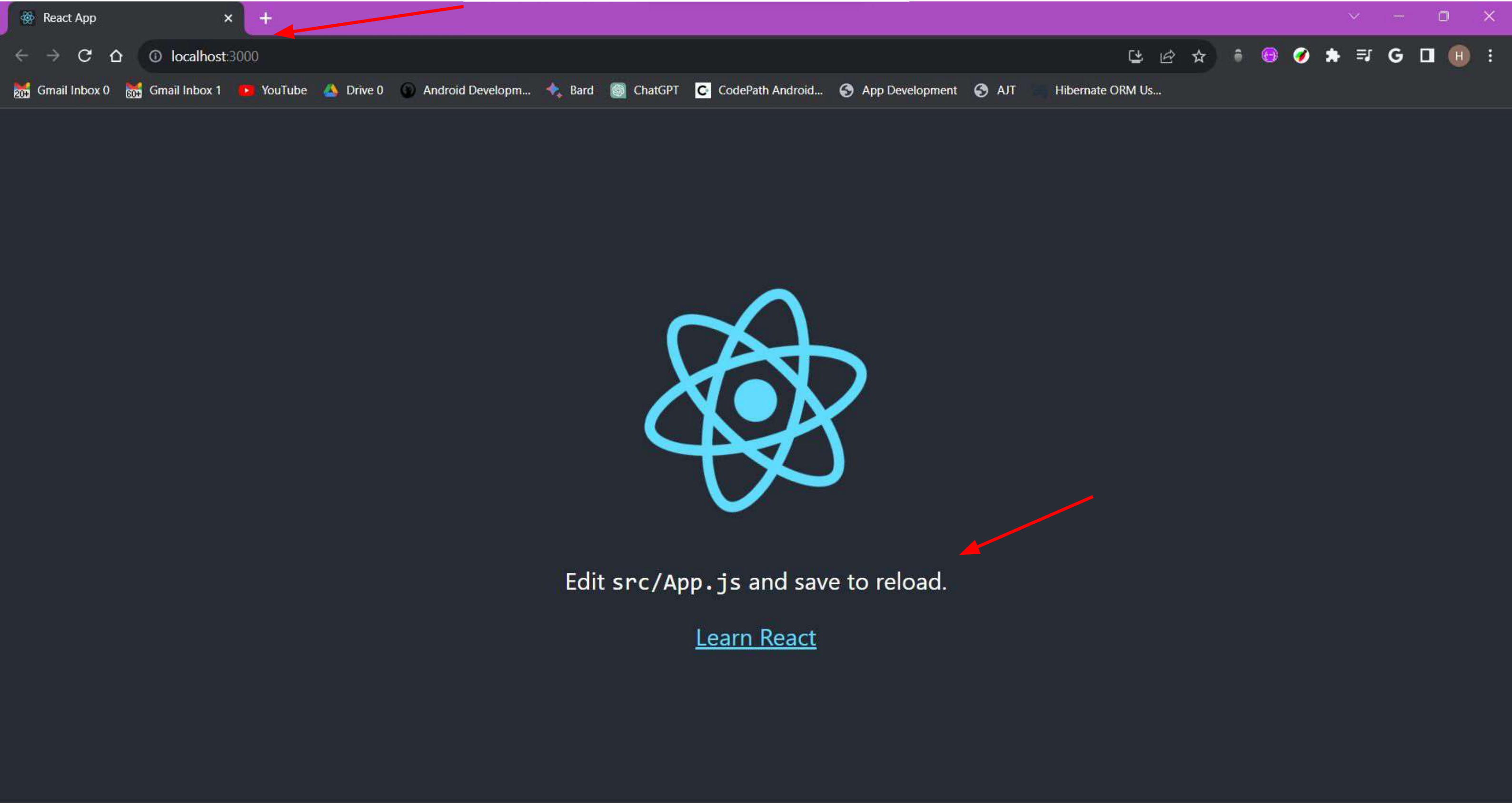
```
itsw2a@DESKTOP-TR1890T  ~\projects\react  cd myreactapp
itsw2a@DESKTOP-TR1890T  ~\projects\react\myreactapp  main  npm start
```

Run Application



```
Compiled successfully!

You can now view myreactapp in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.29.194:3000

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

webpack compiled successfully
```

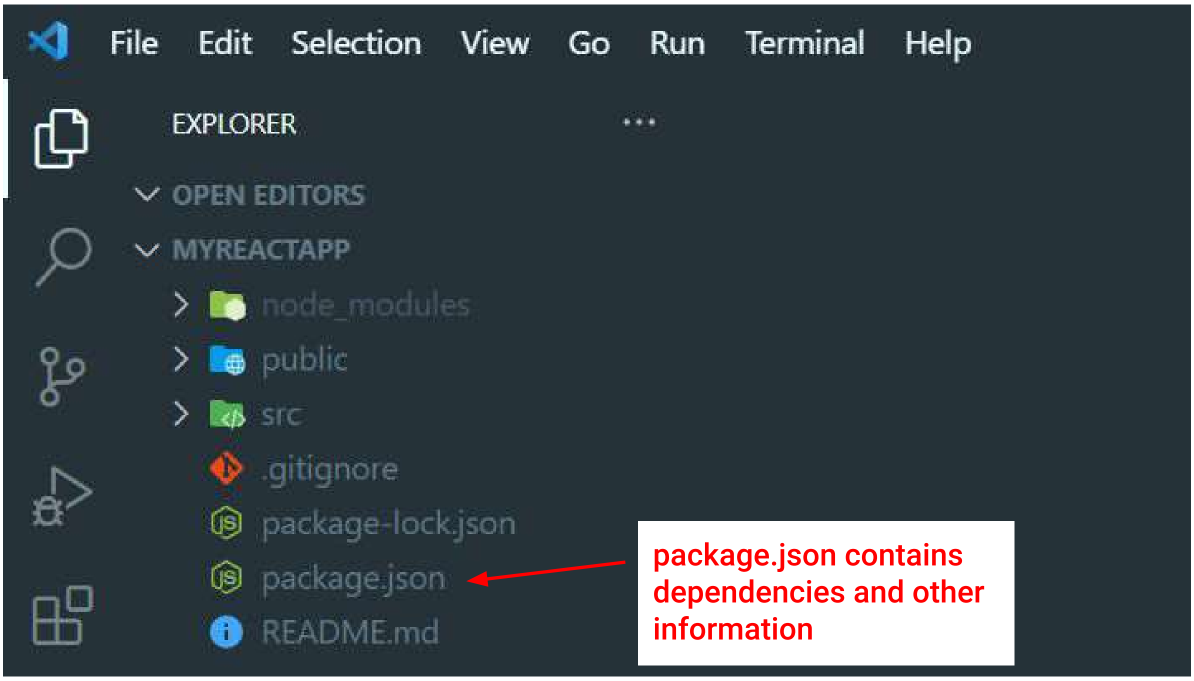
Our app is running in browser



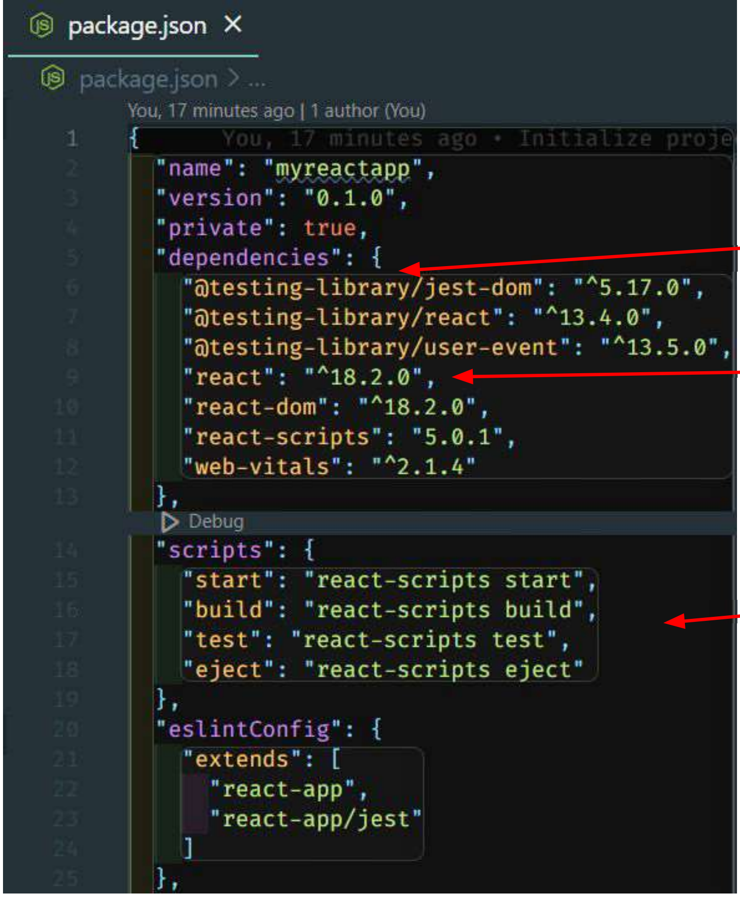
Edit `src/App.js` and save to reload.

Learn React

Explore Files and Folders



package.json contains dependencies and other information package.json


package.json ×
package.json > ...
You, 17 minutes ago | 1 author (You)
You, 17 minutes ago • Initialize proje
Debug

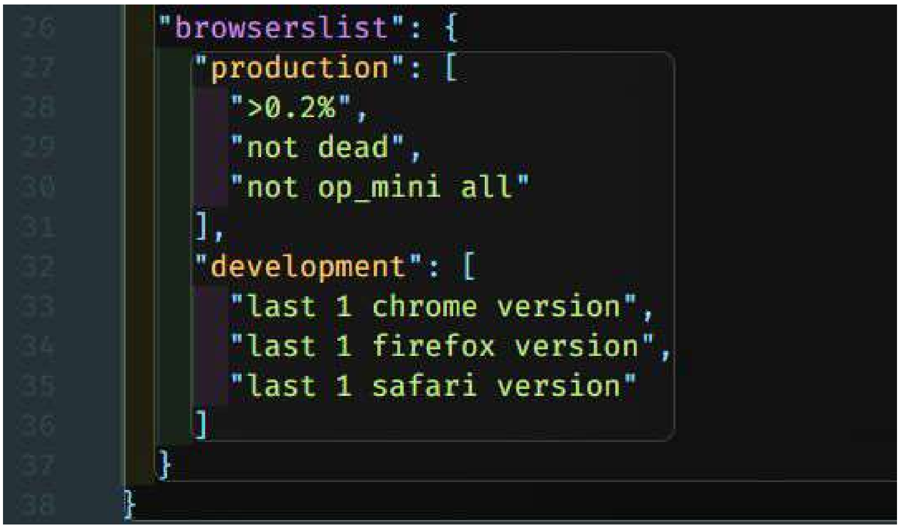
```
{
  "name": "myreactapp",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
```

Dependencies get downloaded when we performed npx create-react-app

Version of React our app is using.

Scripts that we can run

package.json



```
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

What is the use of this file?

- The git repository does not include node_modules folder.
- After cloning repo, we execute the following command npm install
- This npm install command reads this package.json file and downloads all required dependencies.

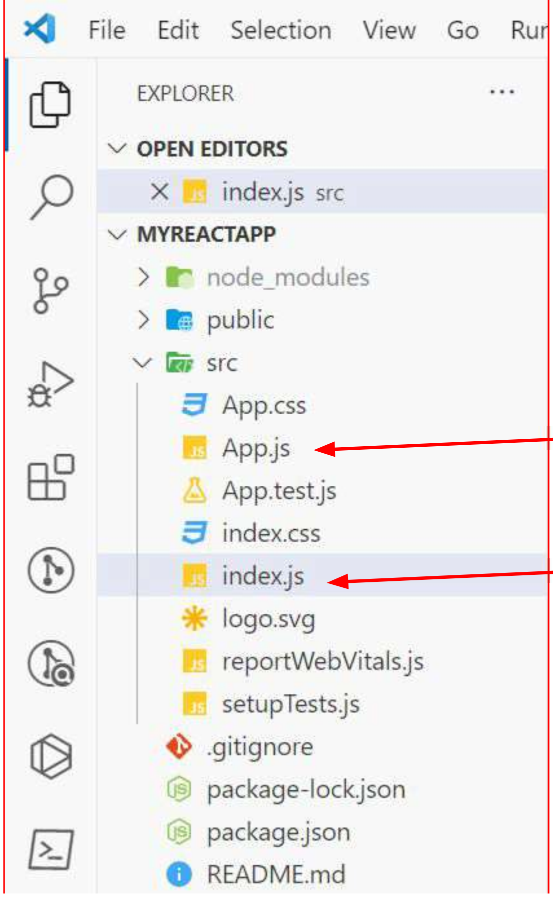
Version Numbering

- In Semantic versioning (SemVer) principles,
  - A version number consists of three parts: MAJOR.MINOR.PATCH.
- Exact Version:
  - Example: "package-name": "1.2.3"
  - This locks your project to a specific version of the package.
- Tilde (~):
  - Example: "package-name": "~1.2.3"
  - This allows updates that do not include breaking changes.
  - Allows patch version to update.

Version Numbering

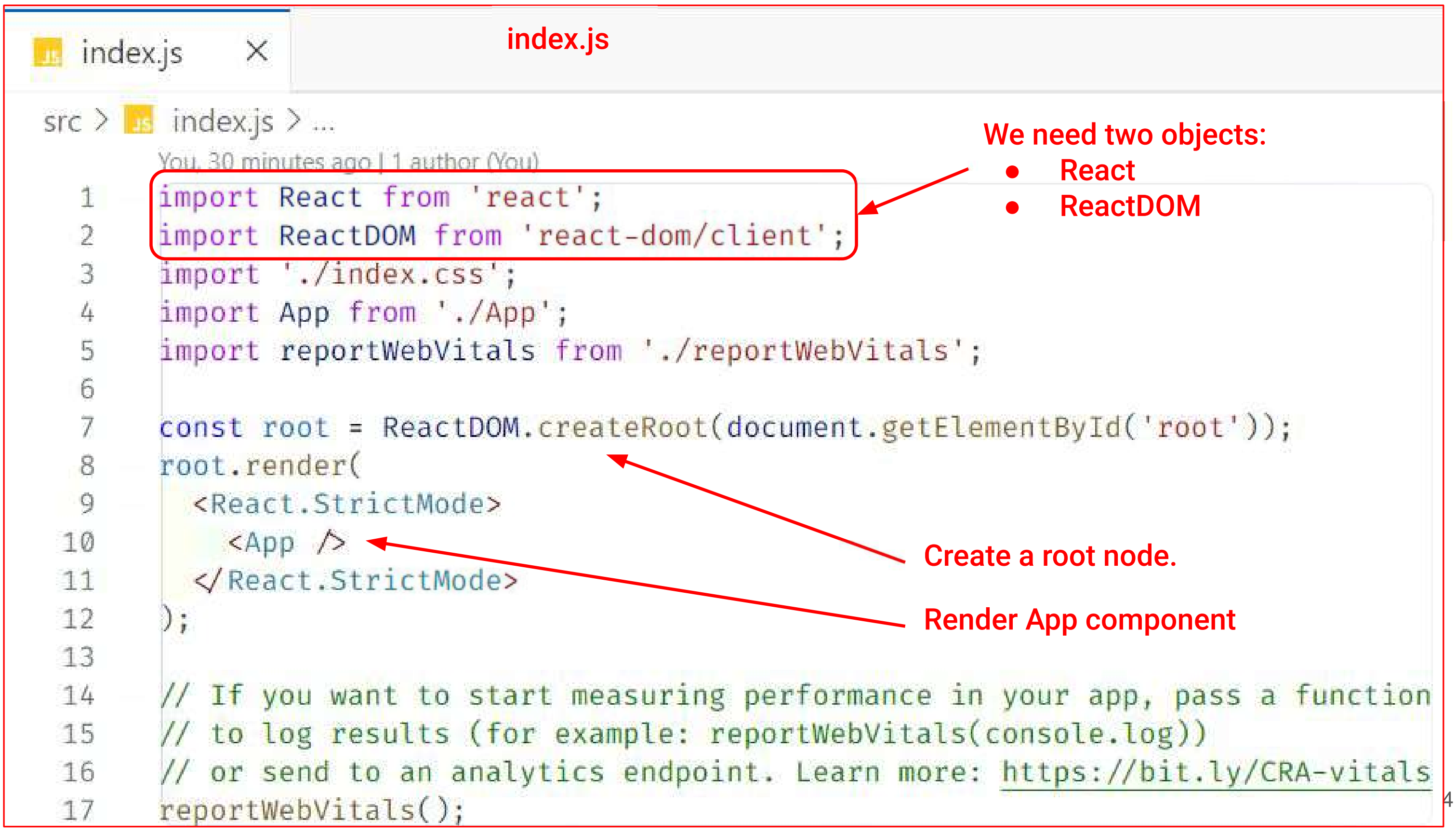
- Caret (^):
  - Example: "package-name": "^1.2.3"
  - Allows updates that do not include breaking changes. Less restrictive than ~. Do not allow major version changes.
- Wildcard (*):
  - Example: "package-name": "*"
  - This allows any version of the package. Not recommended.
- Range:
  - Example: "package-name": ">=1.2.3 <2.0.0"
  - We can specify a range of versions using comparison operators.

Project Structure



We see the content of these two files.

index.js



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

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

We need two objects:

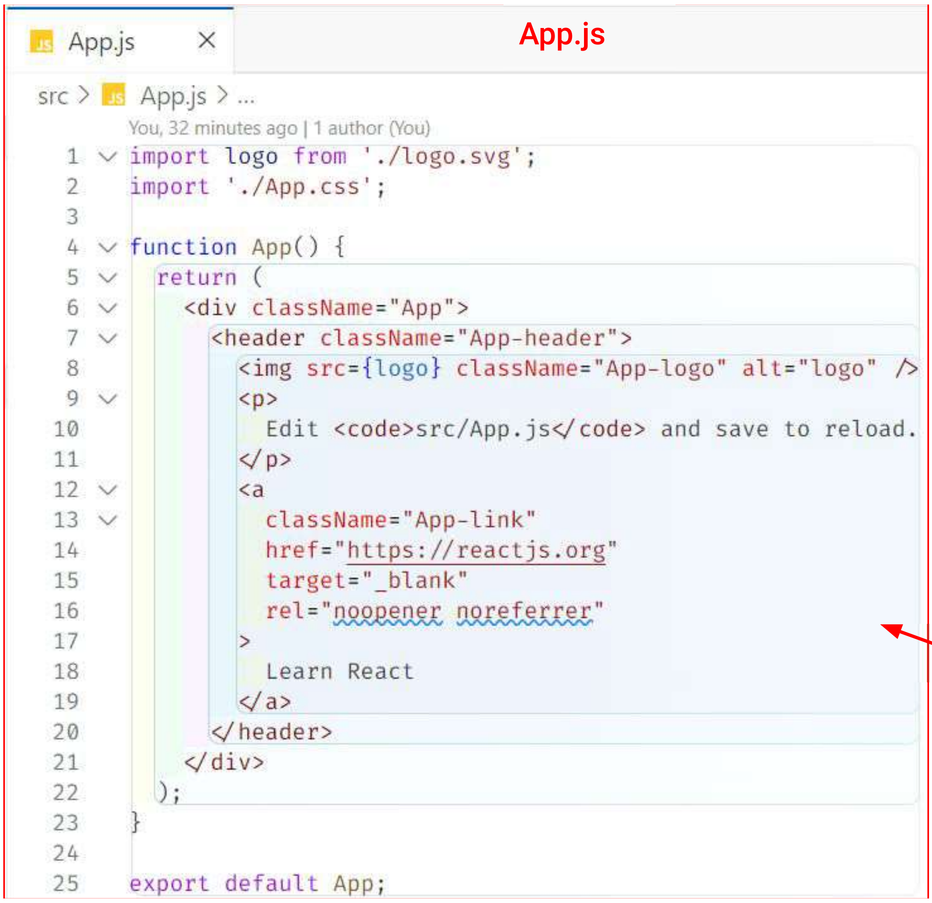
- React
- ReactDOM

Create a root node.

Render App component

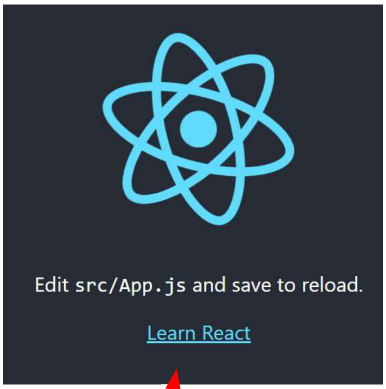
Code in Old Projects

- We may find the following in old projects, prior React 18.x
  - ReactDOM.render(<App/>, document.getElementById('root'))
- New way is to perform the following:
  - const root = ReactDOM.createRoot(document.getElementById('root'));
    root.render(<App/>);



App.js

```
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;
```



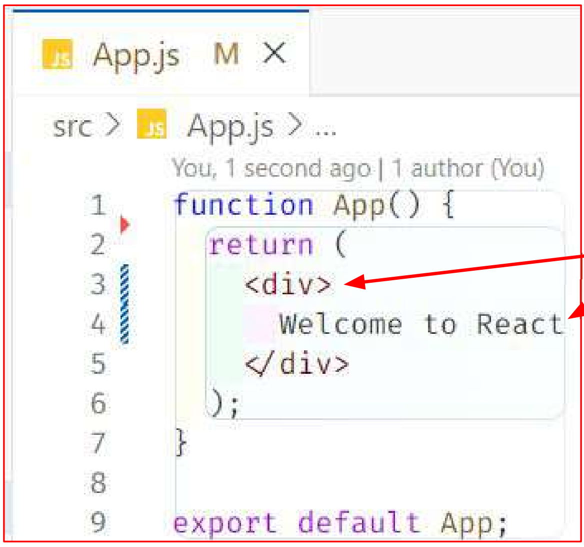
Edit src/App.js and save to reload.

Learn React

This output is coming from this App component

Update Content of App.js

- Remove everything from App.js and make the code look like this:

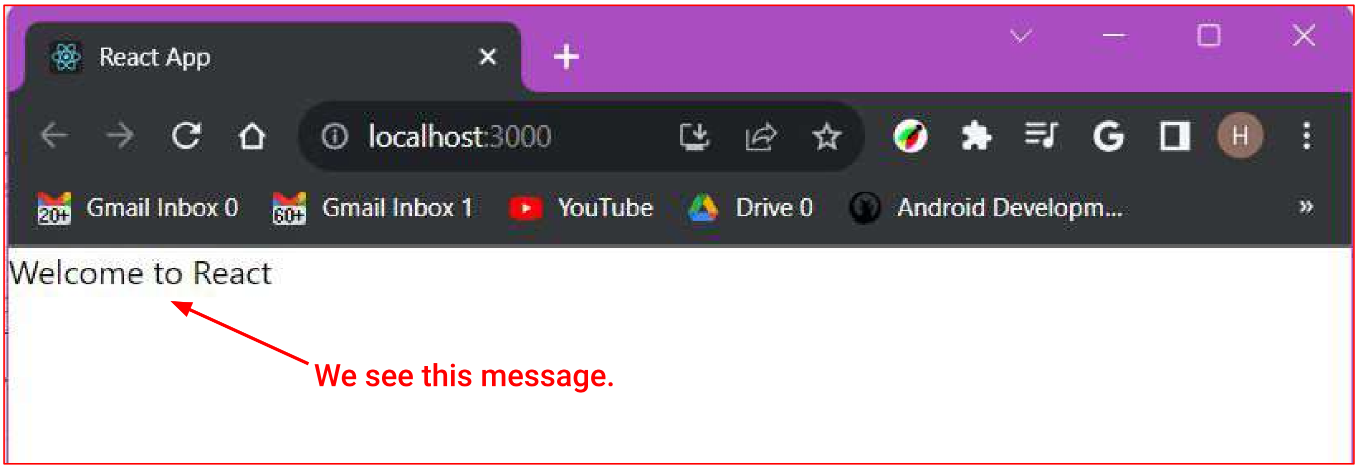


```
function App() {
  return (
    <div>
      Welcome to React
    </div>
  );
}

export default App;
```

Just keep outer div.
Add some text.

Observe Change in Browser

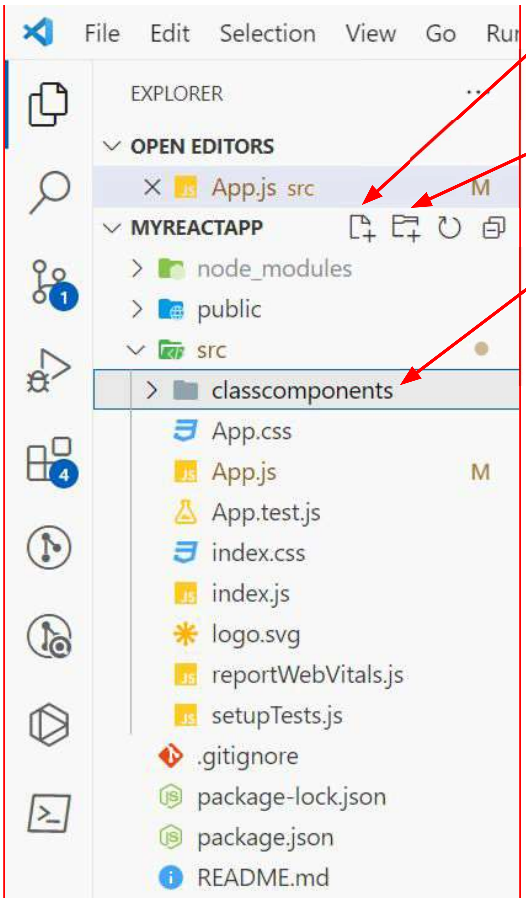
- On saving the file App.js, we see its effect in the browser automatically.
  - That is happening as our app is running in development mode.



```
Welcome to React
```

We see this message.

Class Component Lifecycle
GitHub Repo:
myreactapp/class-component-lifecyle

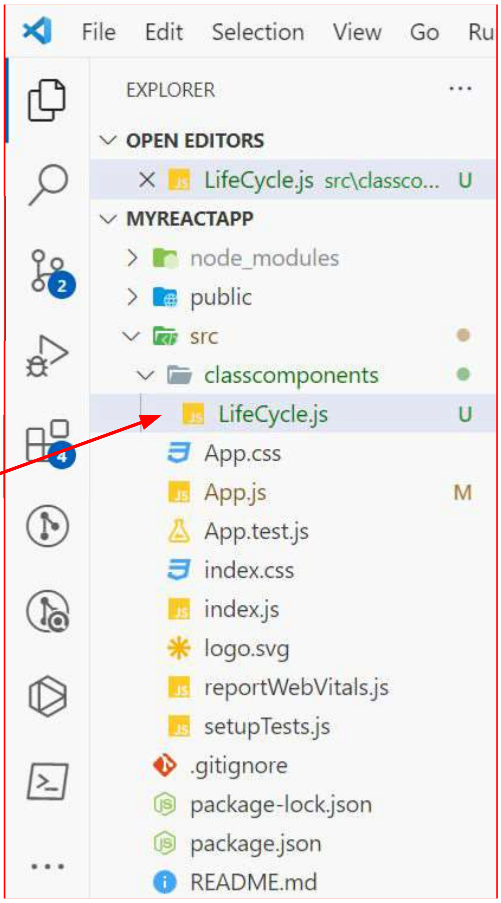


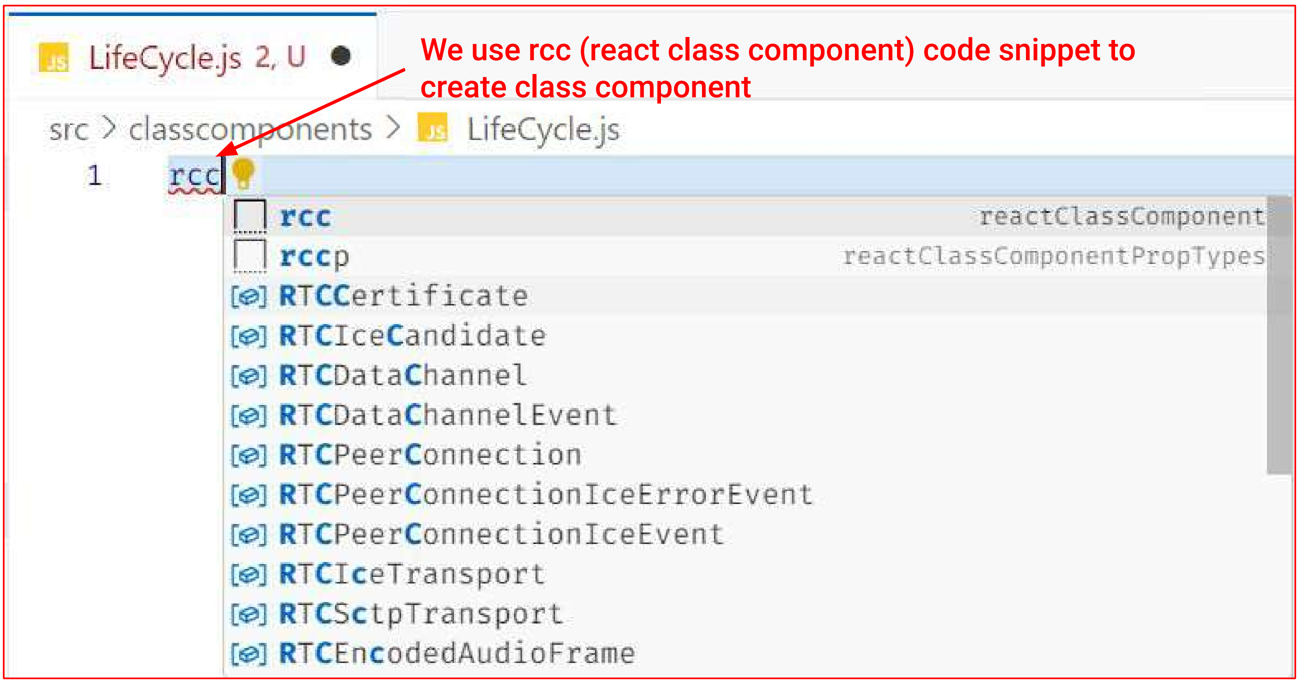
To add new file

To add new folder

Add new folder named classcomponents under src

Add new file named LifeCycle.js under classcomponents

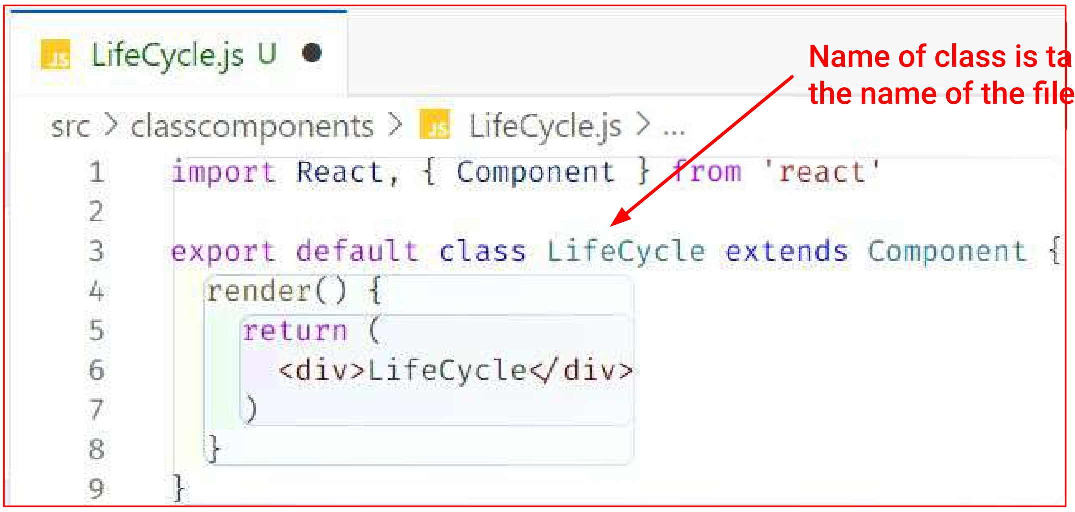




```
rcc
```

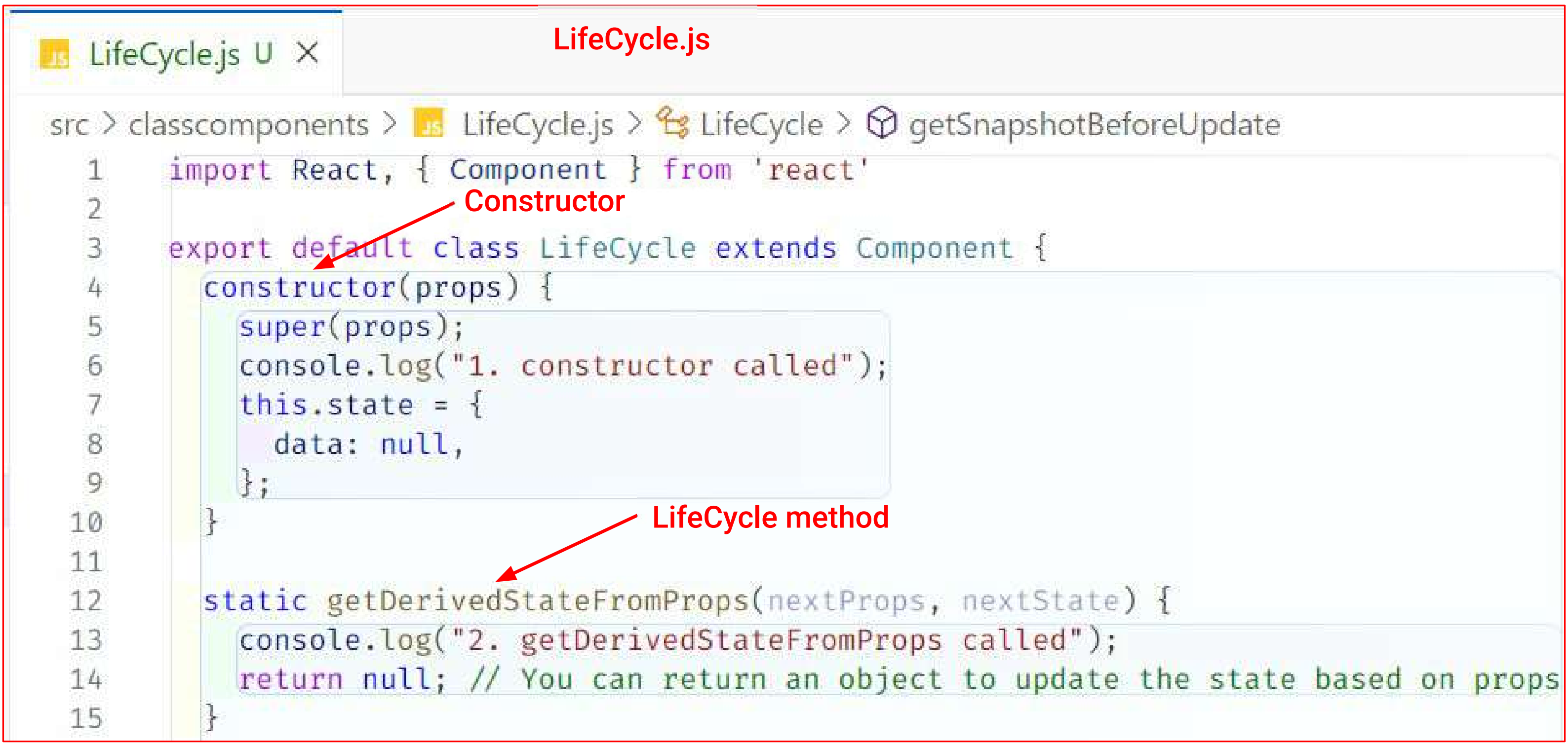
We use rcc (react class component) code snippet to create class component

Default Class Component Created by Code Snippet



```
import React, { Component } from 'react'

export default class LifeCycle extends Component {
  render() {
    return (
      <div>LifeCycle</div>
    )
  }
}
```

Name of class is taken based on the name of the file.


LifeCycle.js U ×
src > classcomponents > LifeCycle.js > LifeCycle > getSnapshotBeforeUpdate

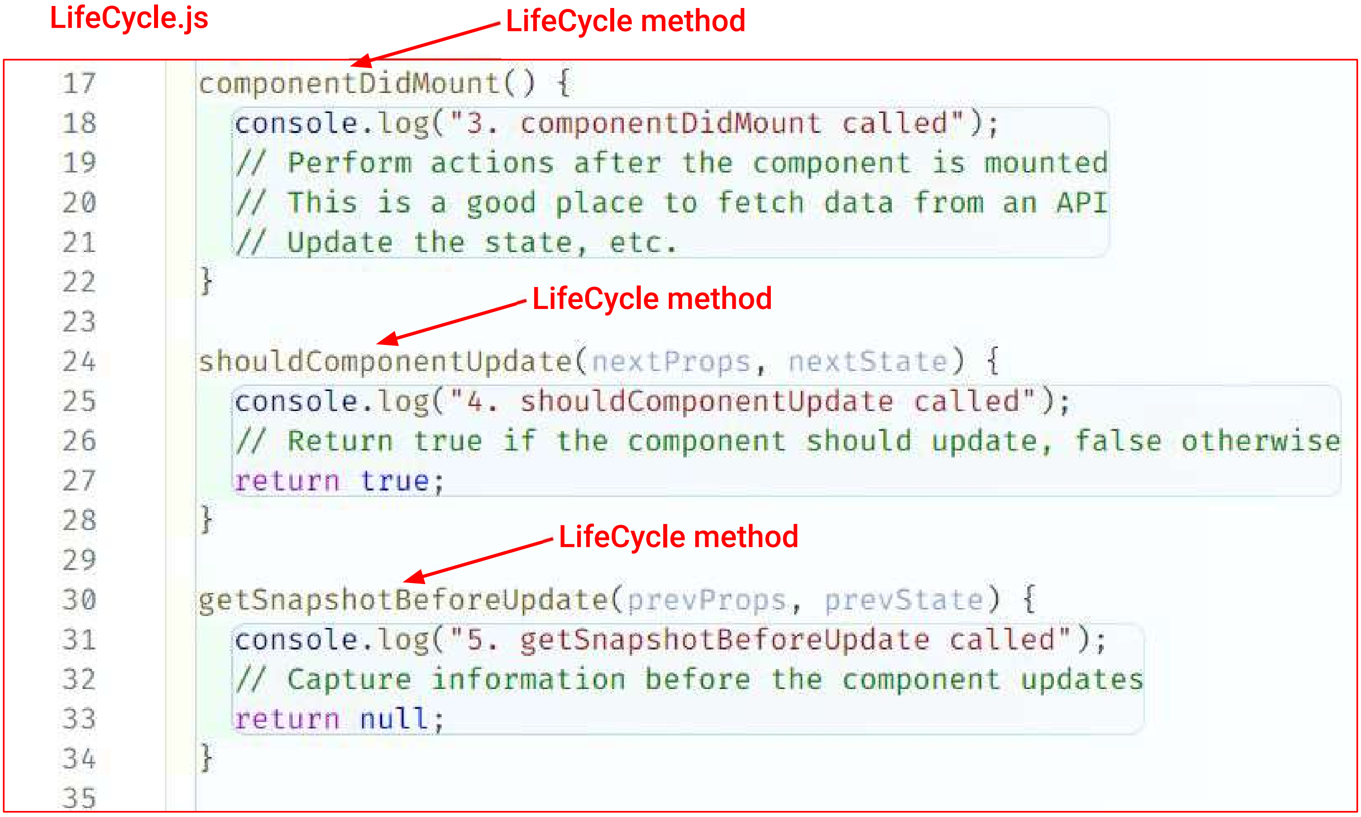
LifeCycle.js

Constructor

LifeCycle method

```
import React, { Component } from 'react'

export default class LifeCycle extends Component {
  constructor(props) {
    super(props);
    console.log("1. constructor called");
    this.state = {
      data: null,
    };
  }

  static getDerivedStateFromProps(nextProps, nextState) {
    console.log("2. getDerivedStateFromProps called");
    return null; // You can return an object to update the state based on props
  }
```

LifeCycle.js



LifeCycle method

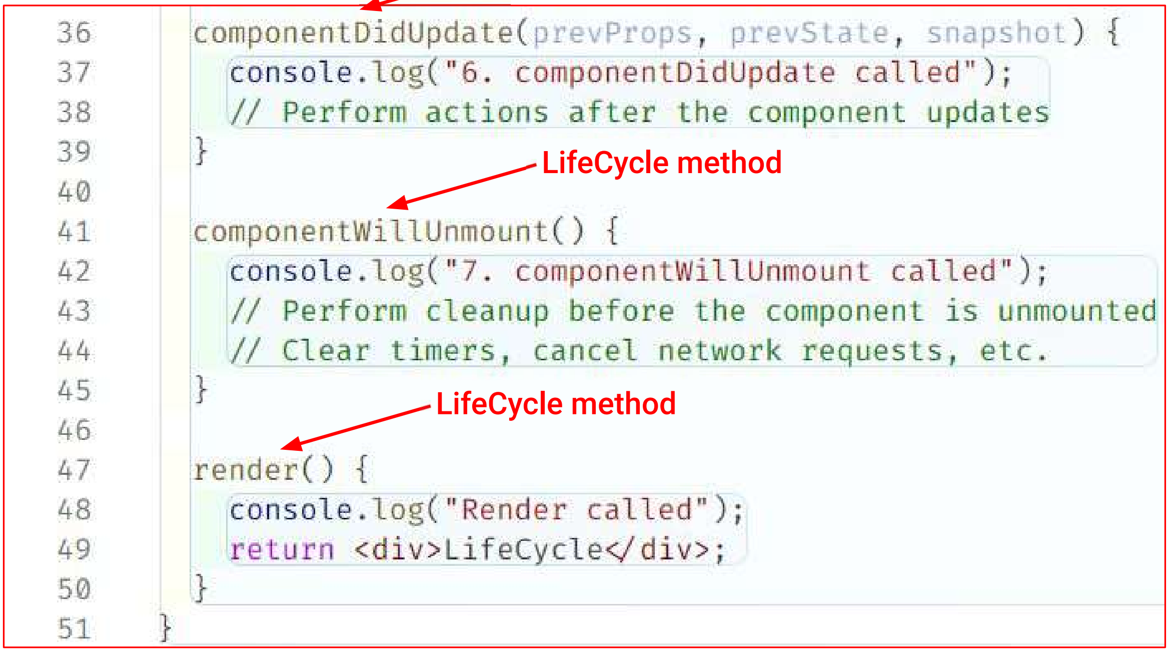
LifeCycle method

LifeCycle method

```
  componentDidMount() {
    console.log("3. componentDidMount called");
    // Perform actions after the component is mounted
    // This is a good place to fetch data from an API
    // Update the state, etc.
  }

  shouldComponentUpdate(nextProps, nextState) {
    console.log("4. shouldComponentUpdate called");
    // Return true if the component should update, false otherwise
    return true;
  }

  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log("5. getSnapshotBeforeUpdate called");
    // Capture information before the component updates
    return null;
  }

```

LifeCycle.js

LifeCycle method

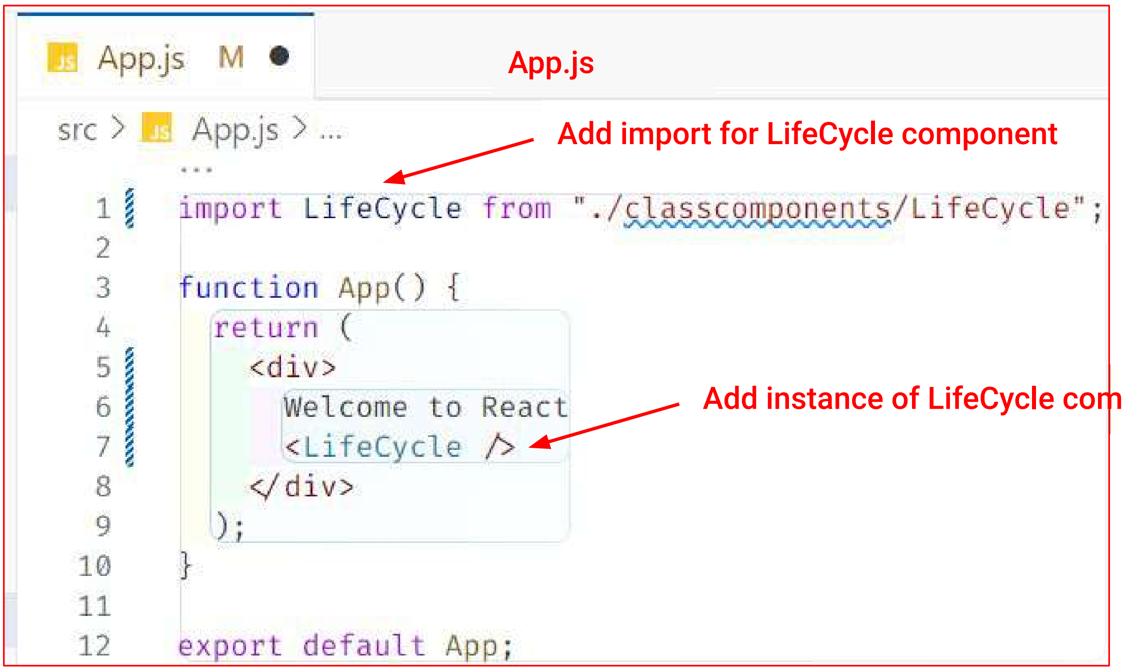


```
  componentDidUpdate(prevProps, prevState, snapshot) {
    console.log("6. componentDidUpdate called");
    // Perform actions after the component updates
  }

  componentWillUnmount() {
    console.log("7. componentWillUnmount called");
    // Perform cleanup before the component is unmounted
    // Clear timers, cancel network requests, etc.
  }

  render() {
    console.log("Render called");
    return <div>LifeCycle</div>;
  }
}
```

LifeCycle method

LifeCycle method

Add LifeCycle Component in App Component

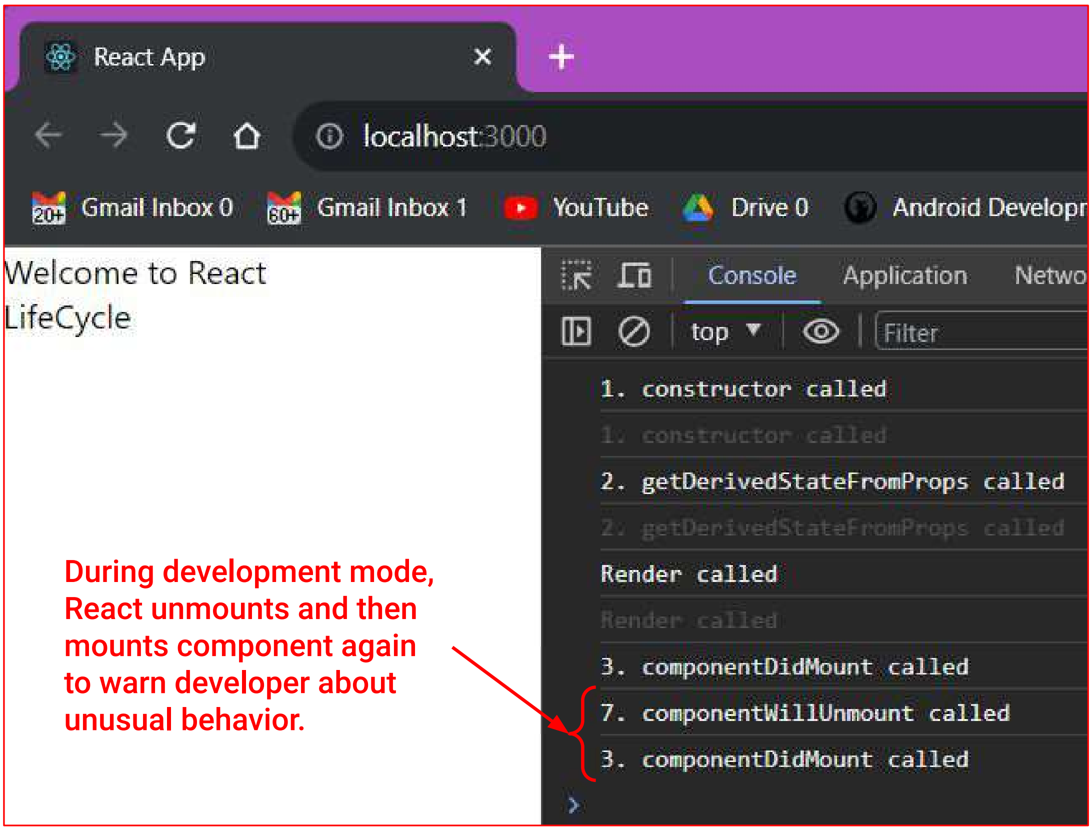


App.js

Add import for LifeCycle component

```
...
import LifeCycle from "./classcomponents/LifeCycle";

function App() {
  return (
    <div>
      Welcome to React
      <LifeCycle />
    </div>
  );
}

export default App;
```

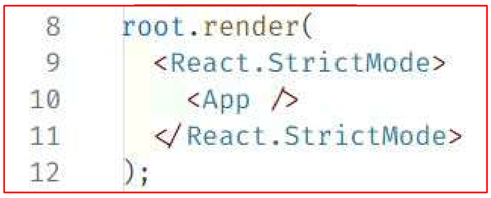
Add instance of LifeCycle component



```
1. constructor called
1. constructor called
2. getDerivedStateFromProps called
2. getDerivedStateFromProps called
Render called
Render called
3. componentDidMount called
7. componentWillUnmount called
3. componentDidMount called
```

During development mode, React unmounts and then mounts component again to warn developer about unusual behavior.

index.js



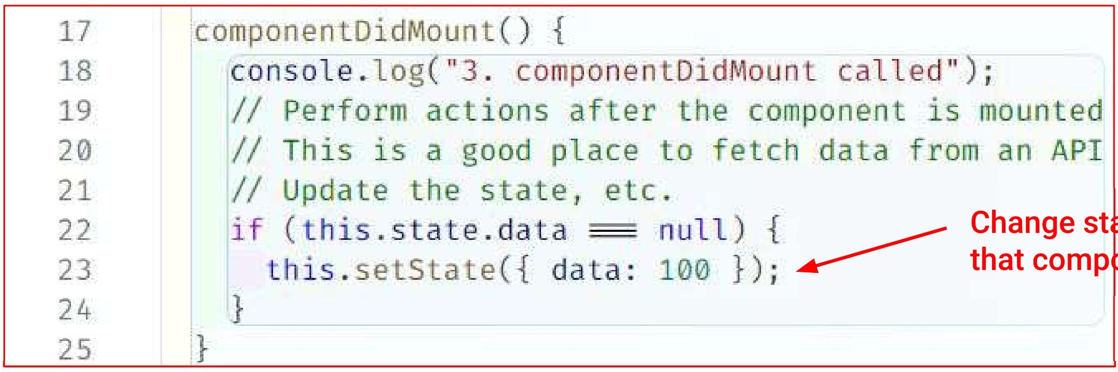
```
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Use of StrictMode:

If there are inconsistencies in the rendering or if there are unexpected side-effects during rendering, extra mounting and rendering can reveal those to the developers.

Add Code to Update State in componentDidMount

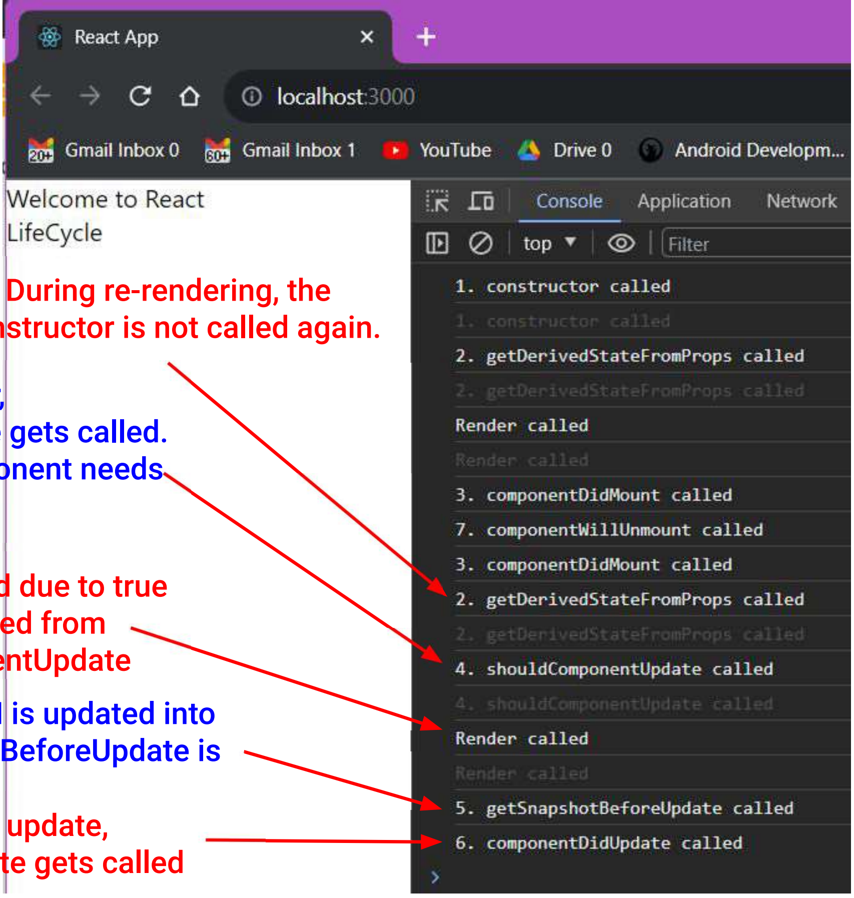
- We add code in componentDidMount such that our component gets re-rendered:
  - We change the state of the component (state change triggers re-render)



```
componentDidMount() {
  console.log("3. componentDidMount called");
  // Perform actions after the component is mounted
  // This is a good place to fetch data from an API
  // Update the state, etc.
  if (this.state.data === null) {
    this.setState({ data: 100 });
  }
}
```

Change state of the component so that component gets updated.

Important note: When we change the state, we have to change in such a way that there is no infinite re-rendering. For example, if we directly write this.setState({...}) inside componentDidUpdate, there will be infinite rendering. We should change state based on some condition.



Welcome to React LifeCycle

```
1. constructor called
1. constructor called
2. getDerivedStateFromProps called
2. getDerivedStateFromProps called
Render called
Render called
3. componentDidMount called
7. componentWillUnmount called
3. componentDidMount called
2. getDerivedStateFromProps called
2. getDerivedStateFromProps called
4. shouldComponentUpdate called
4. shouldComponentUpdate called
Render called
Render called
5. getSnapshotBeforeUpdate called
6. componentDidUpdate called
```

Initial Rendering

Re-Rendering due to change in state

During re-rendering, the constructor is not called again.

Before render, shouldComponentUpdate gets called. It decides whether component needs to re-render.

render() gets called due to true value is returned from shouldComponentUpdate

Before component’s DOM is updated into actual DOM, getSnapShotBeforeUpdate is called

After the DOM update, componentDidUpdate gets called

References

- https://legacy.reactjs.org/docs/react-component.html