



HOW TO CREATE REACT JS PROJECT IN VS CODE

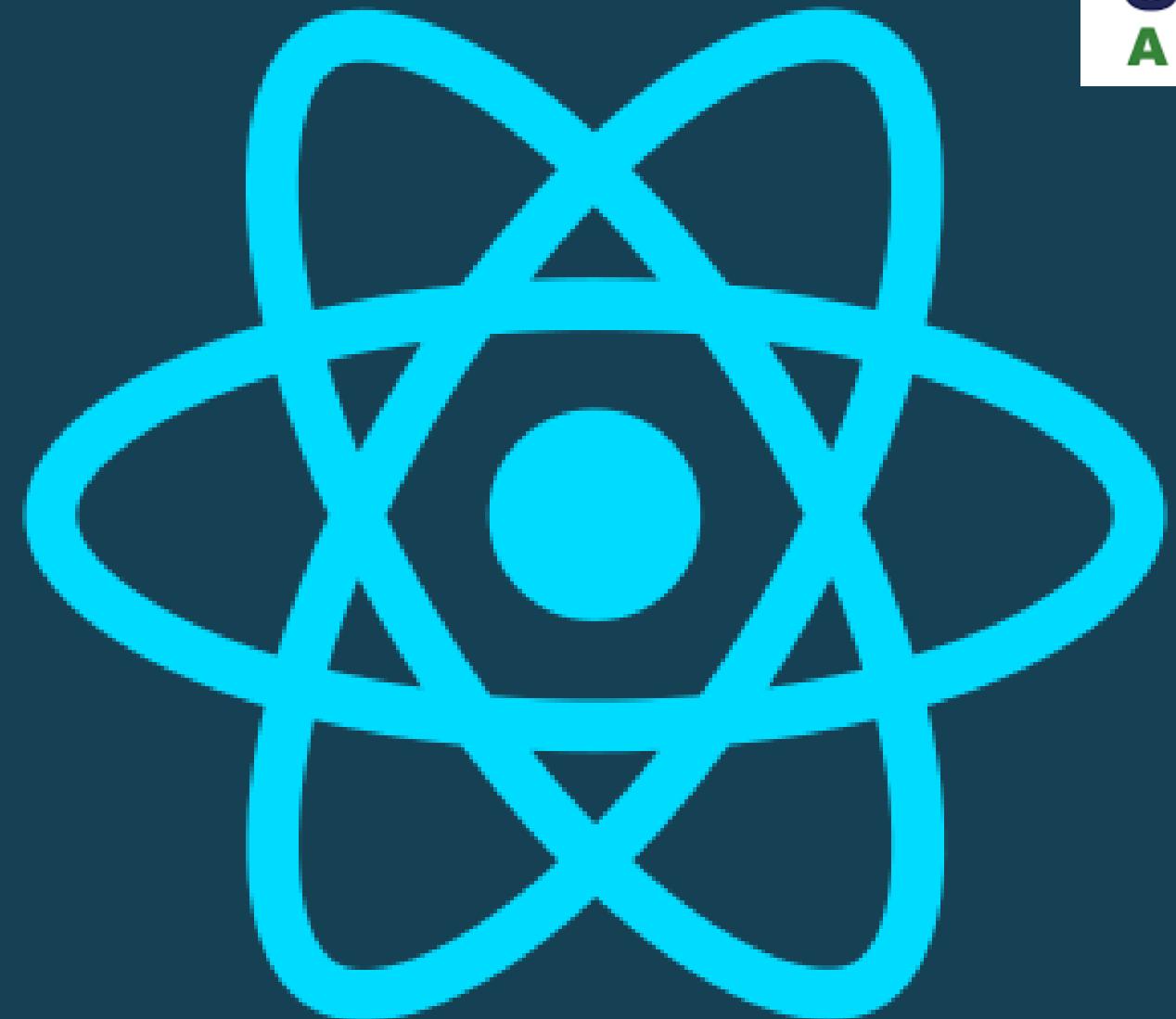
**DAY 4 - WEB DEVELOPMENT
MASTERCLASS**





REACT JS PROJECTS

React is a popular JavaScript library developed by Facebook for building user interfaces. The Visual Studio Code editor supports React.js IntelliSense and code navigation out of the box.

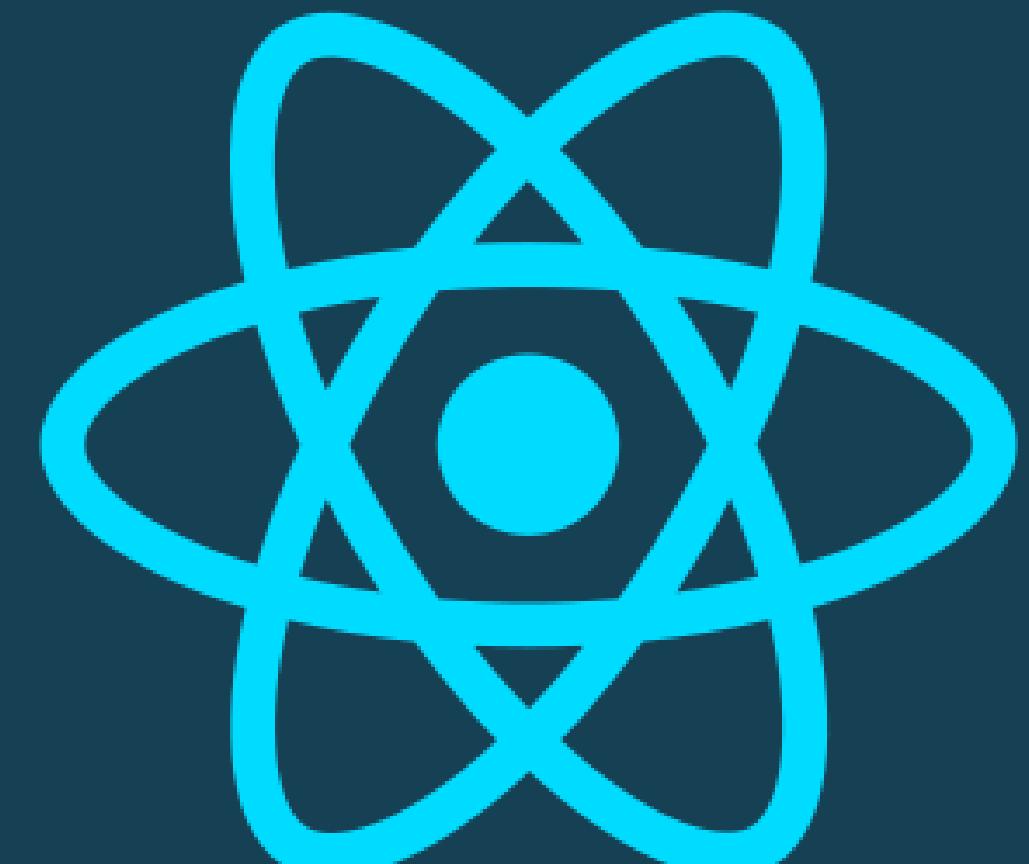


React JS

REACT JS PROJECTS



We'll be using the create-react-app generator for this. To use the generator as well as run the React application server, you'll need Node.js JavaScript runtime and npm (Node.js package manager) installed. NPM is included with Node.js which you can download and install from Node.js downloads.



React JS

REACT JS PROJECTS



Tip: To test that you have Node.js and npm correctly installed on your machine, you can type node --version and npm --version in a terminal or command prompt.



Pantech e Learning
DIGITAL LEARNING SIMPLIFIED

30 DAYS WEB DEVELOPMENT MASTERCLASS

REACT JS PROJECTS

You can now create a new React application by typing:

```
npx create-react-app my-app
```



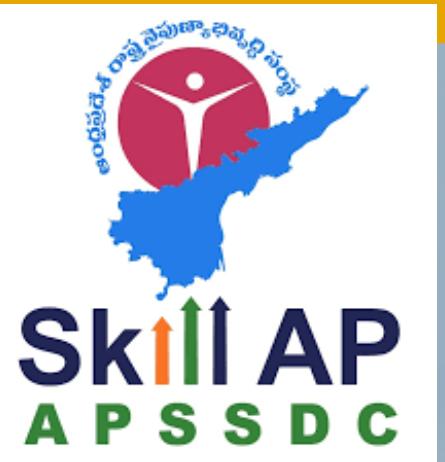
Pantech e Learning
DIGITAL LEARNING SIMPLIFIED

30 DAYS WEB DEVELOPMENT MASTERCLASS

REACT JS PROJECTS

where my-app is the name of the folder for your application. This may take a few minutes to create the React application and install its dependencies.

Note: If you've previously installed create-react-app globally via `npm install -g create-react-app`, we recommend you uninstall the package using `npm uninstall -g create-react-app` to ensure that `npx` always uses the latest version.



REACT JS PROJECTS

Let's quickly run our React application by navigating to the new folder and typing npm start to start the web server and open the application in a browser:

```
Ncd my-app  
npm start
```

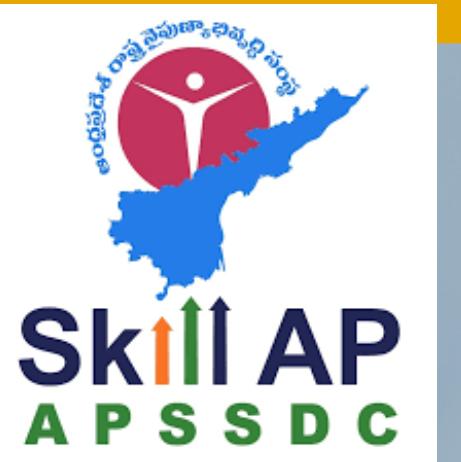


REACT JS PROJECTS

You should see the React logo and a link to "Learn React" on <http://localhost:3000> in your browser. We'll leave the web server running while we look at the application with VS Code.

To open your React application in VS Code, open another terminal or command prompt window, navigate to the my-app folder and type code :

```
cd my-app  
code .
```



REACT JS PROJECTS

1. MARKDOWN PREVIEW
2. Syntax highlighting and bracket matching
3. IntelliSense
4. Go to Definition, Peek definition
5. Hello World
6. Debugging React
7. Set a breakpoint
8. Configure the debugger
9. Live editing and debugging
10. Linting



MARKDOWN PREVIEW

In the File Explorer, one file you'll see is the application README.md Markdown file. This has lots of great information about the application and React in general. A nice way to review the README is by using the VS Code Markdown Preview. You can open the preview in either the current editor group (Markdown: Open Preview Ctrl+Shift+V) or in a new editor group to the side (Markdown: Open Preview to the Side Ctrl+K V). You'll get nice formatting, hyperlink navigation to headers, and syntax highlighting in code blocks.



MARKDOWN PREVIEW

In the File Explorer, one file you'll see is the application README.md Markdown file. This has lots of great information about the application and React in general. A nice way to review the README is by using the VS Code Markdown Preview. You can open the preview in either the current editor group (Markdown: Open Preview Ctrl+Shift+V) or in a new editor group to the side (Markdown: Open Preview to the Side Ctrl+K V). You'll get nice formatting, hyperlink navigation to headers, and syntax highlighting in code blocks.





Skill AP
A P S S D C

EXPLORER

OPEN EDITORS

Preview README.md

MY-APP

- .vscode
- launch.json
- node_modules
- public
- src
- .gitignore
- package.json
- README.md
- yarn.lock

This project was bootstrapped with [Create React App](#).

Available Scripts

In the project directory, you can run:

yarn start

Runs the app in the development mode.
Open <http://localhost:3000> to view it in the browser.

The page will reload if you make edits.
You will also see any lint errors in the console.

yarn test

Launches the test runner in the interactive watch mode.
See the section about [running tests](#) for more information.

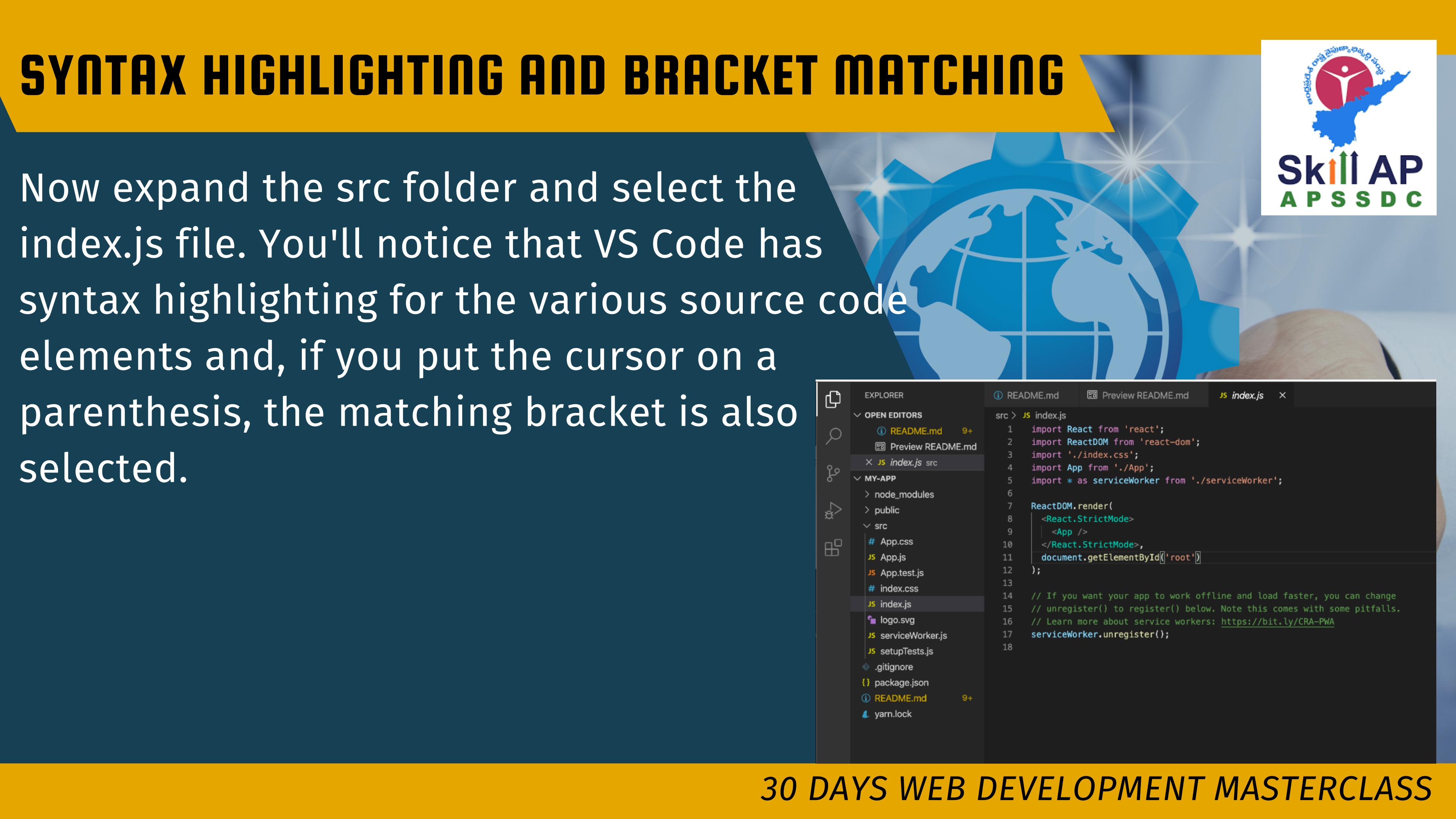
yarn build

Builds the app for production to the **build** folder.
It correctly bundles React in production mode and optimizes the build for the best performance.



SYNTAX HIGHLIGHTING AND BRACKET MATCHING

Now expand the src folder and select the index.js file. You'll notice that VS Code has syntax highlighting for the various source code elements and, if you put the cursor on a parenthesis, the matching bracket is also selected.

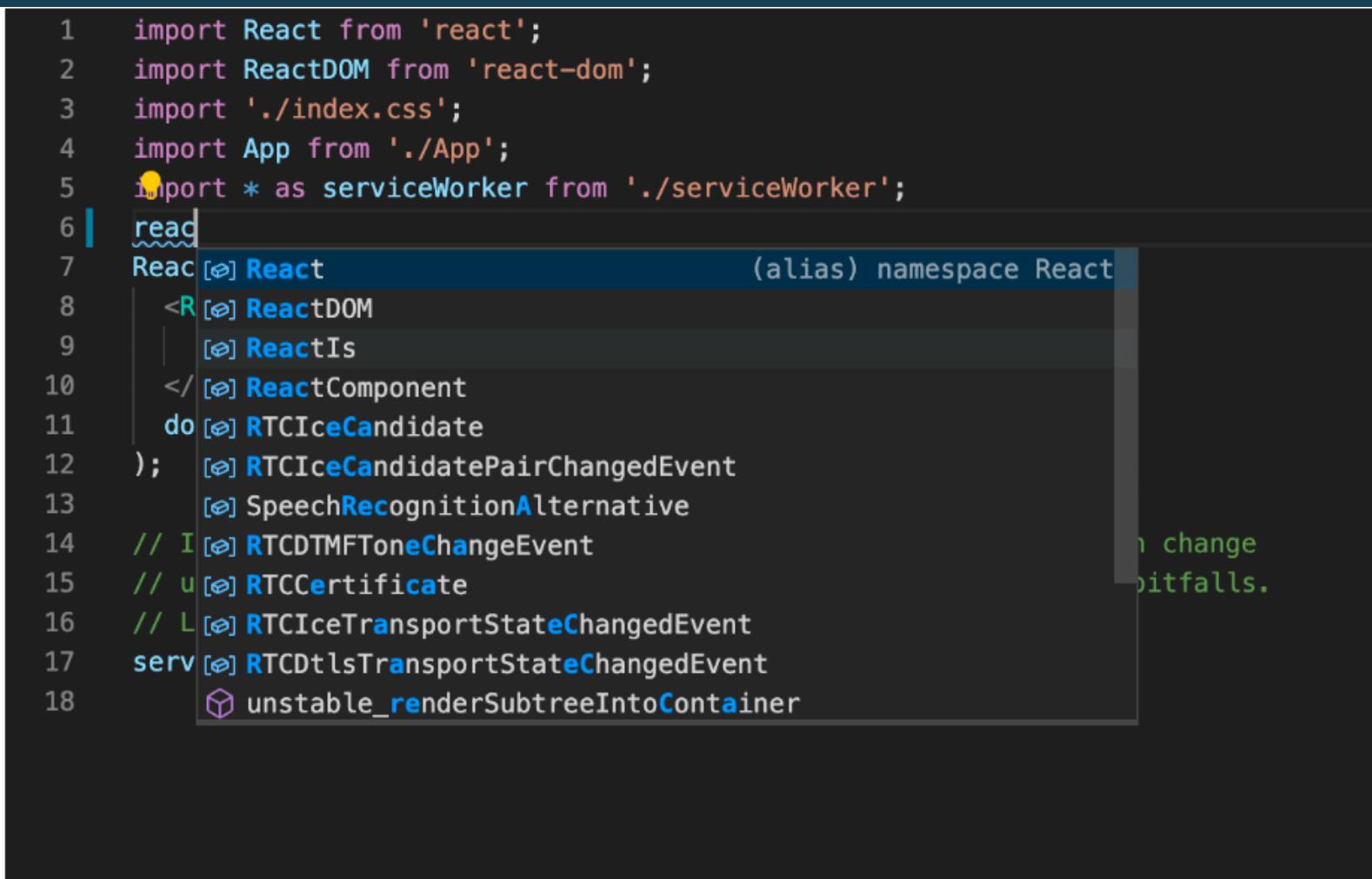


A screenshot of the Visual Studio Code interface. The Explorer sidebar shows a project structure with files like README.md, Preview README.md, index.js (selected), App.css, App.js, App.test.js, index.css, logo.svg, serviceWorker.js, setupTests.js, .gitignore, package.json, README.md, and yarn.lock. The main editor area displays the contents of index.js, which contains React code for rendering an App component. The code is color-coded for syntax: purple for imports, blue for React components, green for strings, and yellow for comments. A cursor is placed on a closing parenthesis in the 11th line, and the matching opening parenthesis in the 8th line is highlighted in yellow, demonstrating the bracket matching feature.

```
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 can change
15 // unregister() to register() below. Note this comes with some pitfalls.
16 // Learn more about service workers: https://bit.ly/CRA-PWA
17 serviceWorker.unregister();
18
```

INTELLISENSE

As you start typing in index.js, you'll see smart suggestions or completions.



A screenshot of a code editor showing Intellisense suggestions for the word 'read'. The code editor has a dark theme. The current line of code is:

```
6 | read
```

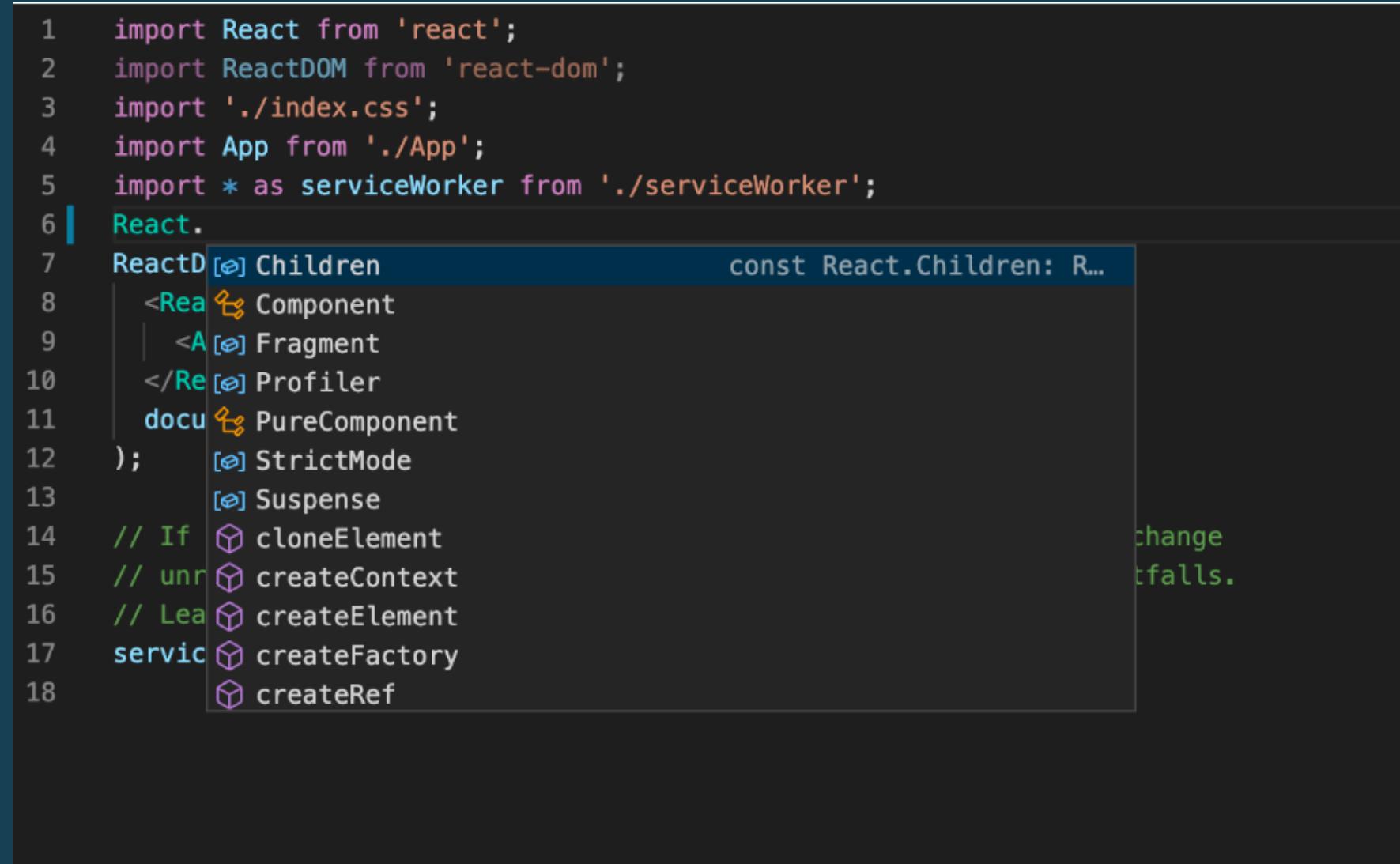
The cursor is at the end of 'read'. A dropdown menu shows suggestions:

- Reac [e] React (alias) namespace React
- <R [e] ReactDOM
- [e] ReactIs
- </ [e] ReactComponent
- do [e] RTCIceCandidate
-); [e] RTCIceCandidatePairChangedEventArgs
- [e] SpeechRecognitionAlternative
- // I [e] RTCDTMFToneChangeEvent
- // u [e] RTCCertificate
- // L [e] RTCIceTransportStateChangedEvent
- serv [e] RTCDtlsTransportStateChangedEvent
- unstable_renderSubtreeIntoContainer

The suggestion 'React' is highlighted with a yellow circle. The status bar at the bottom right of the editor window says 'In change pitfalls.'

After you select a suggestion and type ., you see the .types. a methods on the object through IntelliSense.

-
-
-
-
-
-
-
-
-



A screenshot of the Visual Studio Code interface showing a code editor with the following code:

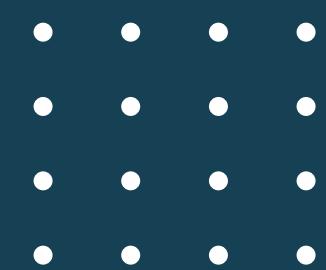
```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import * as serviceWorker from './serviceWorker';
6 | React.
```

The cursor is at the end of the word "React" in line 6. A code completion dropdown menu is open, listing several React modules:

- ReactD [e] Children
- <Rea [e] Component
- | <A [e] Fragment
- 10 </Re [e] Profiler
- 11 docu [e] PureComponent
- 12); [e] StrictMode
- [e] Suspense
- 14 // If [e] createElement
- 15 // unr [e] createContext
- 16 // Lea [e] createElement
- 17 servic [e] createFactory
- [e] createRef

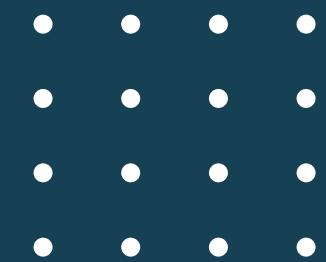
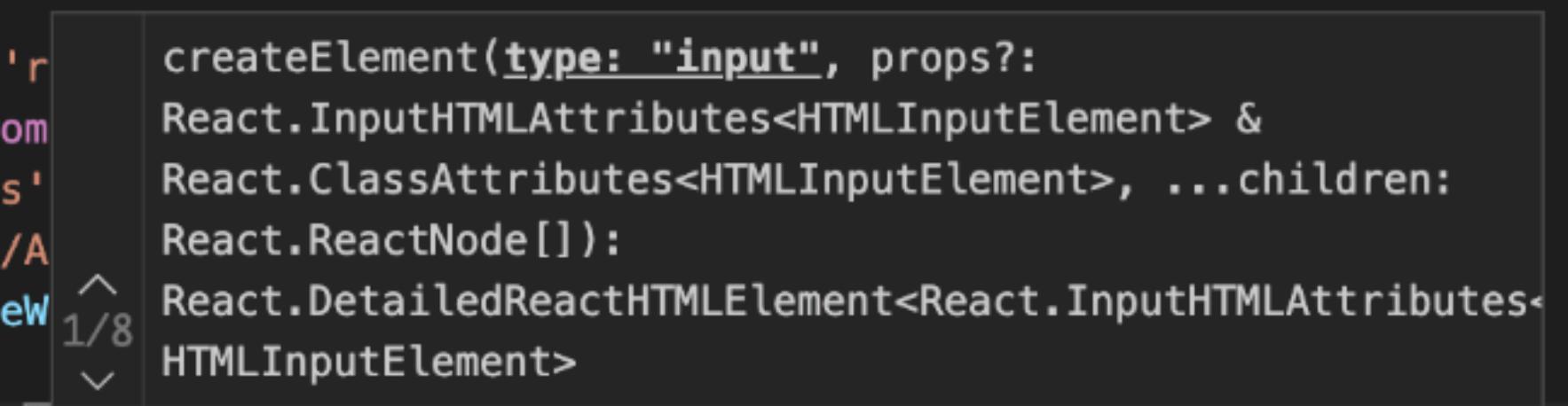
The tooltip "const React.Children: R..." is visible to the right of the dropdown.

VS Code uses the TypeScript language service for its JavaScript code intelligence and it has a feature called Automatic Type Acquisition (ATA). ATA pulls down the npm Type Declaration files (*.d.ts) for the npm modules referenced in the package.json.



If you select a method, you'll also get parameter help:

```
src > JS index.js
1 import React from 'r
2 import ReactDOM from
3 import './index.css'
4 import App from './A
5 import * as serviceW
6
7 | React.createElement()
8 ReactDOM.render(
9   <React.StrictMode>
10  |   <App />
11  </React.StrictMode>,
12  document.getElementById('root')
13 );
14
15 // If you want your app to work offline and load faster, you can change
16 // unregister() to register() below. Note this comes with some pitfalls.
17 // Learn more about service workers: https://bit.ly/CRA-PWA
18 serviceWorker.unregister();
19
```



GO TO , PEEK

Go to Definition, Peek definition

Through the TypeScript language service, VS Code can also provide type definition information in the editor through Go to Definition (F12) or Peek Definition (Alt+F12). Put the cursor over the App, right click and select Peek Definition. A Peek window will open showing the App definition from App.js.

The screenshot shows a code editor with a dark theme. In the top-left, there is a file named 'App.js' with the following code:

```
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 />
```

The cursor is positioned over the word 'App' in the line '8 <App />'. A small yellow lightbulb icon appears above the cursor, indicating a tooltip or peek definition is available. A tooltip window titled 'App.js ~/my-app/src - Definitions (1)' opens below the cursor, showing the definition of the 'App' function:

```
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         <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        >
```

The tooltip also includes a note: 'Edit <code>src/App.js</code> and save to reload.' To the right of the tooltip, a preview pane shows the beginning of the 'App' function definition:

```
function App()
```

At the bottom right of the image, there is a vertical column of four rows, each containing four small circular icons, likely representing additional navigation or context options.

HELLO WORLD !

Hello World Let's update the sample application to "Hello World!". Create a new H1 header with "Hello, world!" and replace the <App /> tag in ReactDOM.render with element.

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

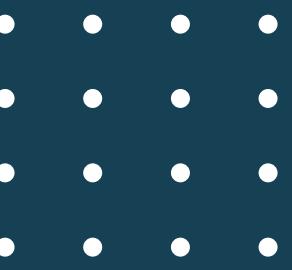
var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');
ReactDOM.render(element, document.getElementById('root'));

reportWebVitals();
```

• • •
• • •
• • •
• • •

HELLO WORLD !

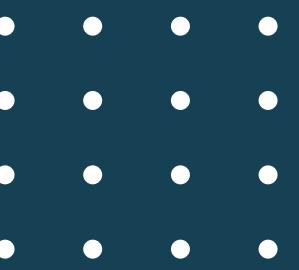
Once you save the index.js file, the running instance of the server will update the web page and you'll see "Hello World!" when you refresh your browser.



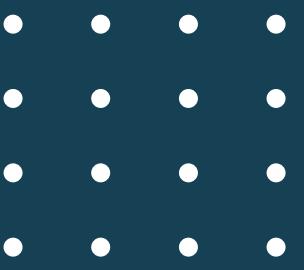
HELLO WORLD !

Tip: VS Code supports Auto Save, which by default saves your files after a delay.

Check the Auto Save option in the File menu to turn on Auto Save or directly configure the files.autoSave user setting.



To debug the client side React code,
we'll use the built-in JavaScript debugger.



SET A BREAKPOINT

To set a breakpoint in index.js, click on the gutter to the left of the line numbers. This will set a breakpoint which will be visible as a red circle.

```
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 var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');
8 ReactDOM.render(element, document.getElementById('root'));
9
10 // If you want your app to work offline and load faster, you can change
11 // unregister() to register() below. Note this comes with some pitfalls.
12 // Learn more about service workers: https://bit.ly/CRA-PWA
13 serviceWorker.unregister();
```

• • •
• • •
• • •
• • •

CONFIGURE THE DEBUGGER

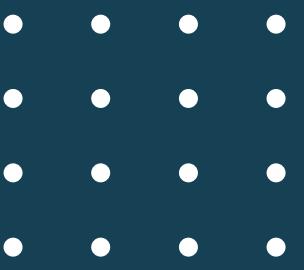
We need to initially configure the debugger. To do so, go to the Run and Debug view (Ctrl+Shift+D) and select the create a launch.json file link to create a launch.json debugger configuration file. Choose Web App (Edge) from the Select debugger dropdown list. This will create a launch.json file in a new .vscode folder in your project which includes a configuration to launch the website.

We need to make one change for our example: change the port of the url from 8080 to 3000. Your launch.json should look like this:

• • •
• • •
• • •
• • •

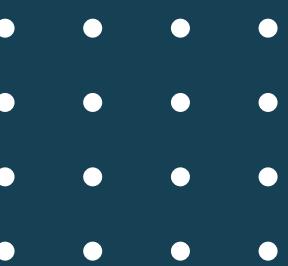
CONFIGURE THE DEBUGGER

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "type": "msedge",  
      "request": "launch",  
      "name": "Launch Edge against localhost",  
      "url": "http://localhost:3000",  
      "webRoot": "${workspaceFolder}"  
    }  
  ]  
}
```



CONFIGURE THE DEBUGGER

Ensure that your development server is running (`npm start`). Then press F5 or the green arrow to launch the debugger and open a new browser instance. The source code where the breakpoint is set runs on startup before the debugger was attached, so we won't hit the breakpoint until we refresh the web page. Refresh the page and you should hit your breakpoint.



The screenshot shows the VS Code interface with the debugger extension active. The main editor window displays the `index.js` file:

```
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 var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');
8 ReactDOM.render(element, document.getElementById('root'));
9
10 // If you want your app to work offline and load faster, you can change
11 // unregister() to register() below. Note this comes with some pitfalls.
12 // Learn more about service workers: https://bit.ly/CRA-PWA
13 serviceWorker.unregister();
14
```

The line `var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');` is highlighted with a red circle and a yellow background, indicating it is the current line of execution.

The left sidebar contains the following sections:

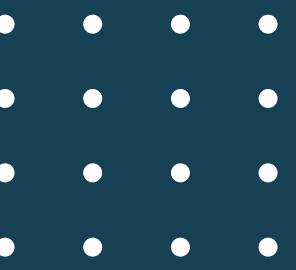
- VARIABLES**: Shows a tree structure with `Local` variables, including `this`, `__webpack_exports__`, `__webpack_require__`, `_App__WEBPACK_IMPORTED_MODULE_0__`, `__index_css__WEBPACK_IMPORTED_MODULE_0__`, and `__index_css__WEBPACK_IMPORTED_MODULE_0__`.
- WATCH**: An empty section.
- CALL STACK PAUSED ON BREAK**: Shows the call stack:
 - `./src/index.js inde...`
 - `__webpack_require__ |`
 - `fn bootstrap 150:1`
 - `1 main.chunk.js`
 - `__webpack_require__ |`
- LOADED SCRIPTS**: Shows loaded scripts:
 - `/static/js`
 - `<eval>`
- BREAKPOINTS**: Shows breakpoints for `All Exceptions`, `Uncaught Exceptions`, and `index.js src`. The `index.js src` breakpoint is checked and highlighted with a red dot.

The bottom status bar shows the following information:

- Line 14, Column 1
- Spaces: 2
- UTF-8
- LF

CONFIGURE THE DEBUGGER

You can step through your source code (F10), inspect variables such as element, and see the call stack of the client side React application.



The screenshot shows the Chrome DevTools Variables panel. At the top, there's a RUN button and a dropdown menu set to "Launch Chrome against localhost". Below that is a search bar and a gear icon. The main area is titled "VARIABLES" and contains a tree view of object properties. One property, "element", is highlighted with a blue background. The "element" node has the following properties:

- _owner: null
- _self: null
- _source: null

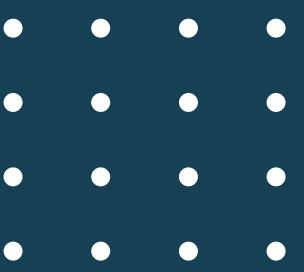
Underneath "element" are several other nodes:

- _store: Object {validated: false}
 \$\$typeof: Symbol(react.element)
 key: null
- props: Object {className: "greeting", children: "Hello, world..."
 ref: null
 type: "h1"}
 __proto__: Object {constructor: , __defineGetter__: , __defin...
- module: Object {i: "./src/index.js", l: false, exports: Module, ...}
- react__WEBPACK_IMPORTED_MODULE_0__: Object {Children: Object, C...
- react__WEBPACK_IMPORTED_MODULE_0___default: function getModuleE...
- react_dom__WEBPACK_IMPORTED_MODULE_1__: Object {__SECRET_INTERN...
- react_dom__WEBPACK_IMPORTED_MODULE_1___default: function getMod...

At the bottom of the panel, there are two sections: "Global" and "WATCH".

Live editing and debugging

If you are using webpack together with your React app, you can have a more efficient workflow by taking advantage of webpack's HMR mechanism which enables you to have live editing and debugging directly from VS Code



Linting

Linters analyze your source code and can warn you about potential problems before you run your application. The JavaScript language services included with VS Code has syntax error checking support by default, which you can see in action in the Problems panel (View > Problems Ctrl+Shift+M).



Try making a small error in your React source code and you'll see a red squiggle and an error in the Problems panel.

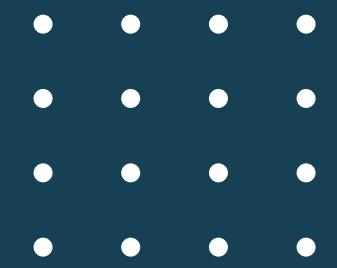
The screenshot shows a code editor with a dark theme. An open file is named `index.js`. The code is a standard React application setup, including imports for `React`, `ReactDOM`, and `App`, and a render call. At line 13, there is a line of code: `serviceWorker.unregister();`. A red squiggle underline is present under the closing brace of this line, indicating a syntax error. The bottom right corner of the code editor has a yellow status bar with four small circular icons.

```
JS index.js  ×
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 var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');
8 ReactDOM.render(element, document.getElementById('root'));
9
10 // If you want your app to work offline and load faster, you can change
11 // unregister() to register() below. Note this comes with some pitfalls.
12 // Learn more about service workers: https://bit.ly/CRA-PWA
13 serviceWorker.unregister();
14
```

PROBLEMS 1 OUTPUT ... Filter: E.g.: text, **/*.ts, !**/node_modules/**

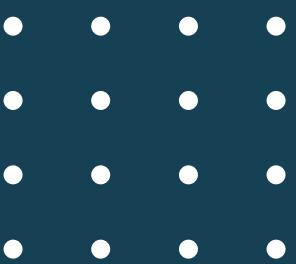
JS index.js src 1

④ ')' expected. ts(1005) [13, 26]



Linters

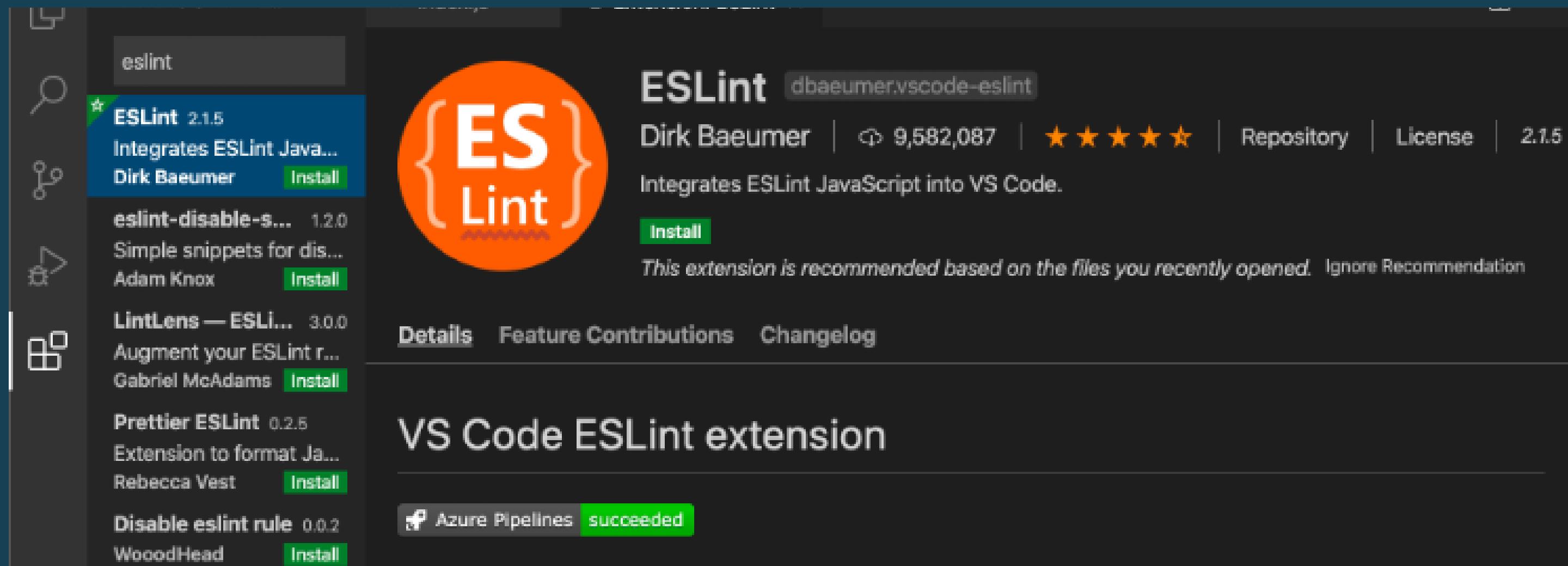
Linters can provide more sophisticated analysis, enforcing coding conventions and detecting anti-patterns. A popular JavaScript linter is ESLint. ESLint, when combined with the ESLint VS Code extension, provides a great in-product linting experience.



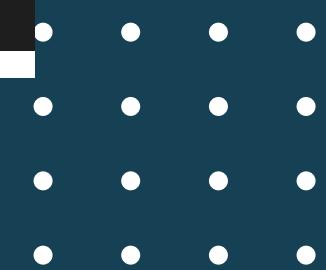
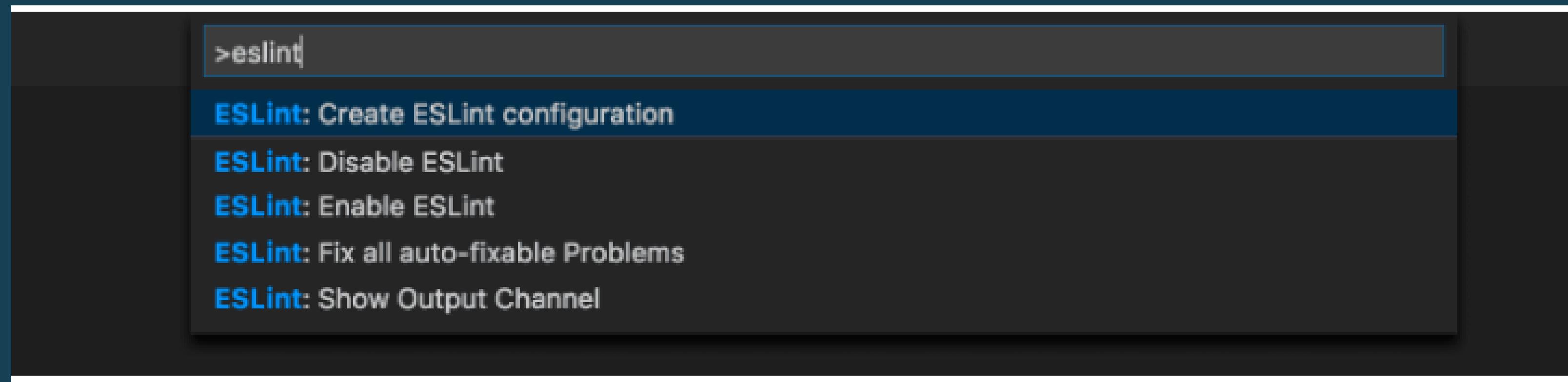
First, install the ESLint command-line tool:

```
npm install -g eslint
```

Then install the ESLint extension by going to the Extensions view and typing 'eslint'.



Once the ESLint extension is installed and VS Code reloaded, you'll want to create an ESLint configuration file, `.eslintrc.js`. You can create one using the extension's ESLint: Create ESLint configuration command from the Command Palette (Ctrl+Shift+P).



The command will prompt you to answer a series of questions in the Terminal panel. Take the defaults and it will create a `.eslintrc.js` file in your project root that looks something like this:

```
module.exports = {  
    env: {  
        browser: true,  
        es2020: true  
    },  
    extends: ['eslint:recommended', 'plugin:react/recommended'],  
    parserOptions: {  
        ecmaFeatures: {  
            jsx: true  
        },  
        ecmaVersion: 11,  
        sourceType: 'module'  
    },  
    plugins: ['react'],  
    rules: {}  
};  
• • • •  
• • • •  
• • • •  
• • • •  
• • • •
```

ESLint will now analyze open files and shows a warning in index.js about 'App' being defined but never used.

The screenshot shows a code editor window with the file 'index.js' open. The code is a simple React application. ESLint has identified a warning in the 4th line, where it detects that the variable 'App' is defined but not used. This warning is highlighted in red in the code editor's interface. The ESLint status bar at the bottom indicates 1 problem found.

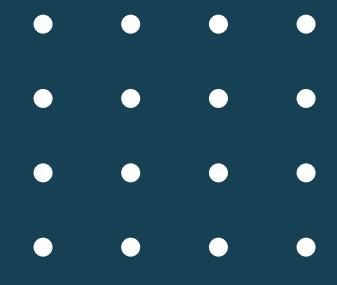
```
JS index.js  ×

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 var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');
8 ReactDOM.render(element, document.getElementById('root'));
9
10 // If you want your app to work offline and load faster, you can change
11 // unregister() to register() below. Note this comes with some pitfalls.
12 // Learn more about service workers: https://bit.ly/CRA-PWA
13 serviceWorker.unregister();
14

PROBLEMS 1 ... Filter. E.g.: text, **/*.ts, !**/node_modules/**
```

JS index.js src 1

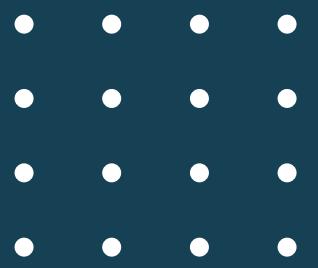
✖ 'App' is defined but never used. eslint([no-unused-vars](#)) [4, 8]



You can modify the ESLint rules in the `.eslintrc.js` file.

Let's add an error rule for extra semi-colons:

```
"rules": {  
  "no-extra-semi": "error"  
}
```



Now when you mistakenly have multiple semicolons on a line, you'll see an error (red squiggle) in the editor and error entry in the Problems panel.

The screenshot shows a code editor with a dark theme. At the top, there are tabs for 'JS index.js' and '.eslintrc.js'. The main editor area contains the following code:

```
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 var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');
8 ReactDOM.render(element, document.getElementById('root'));

9
10 // If you want your app to work offline and load faster, you can change
11 // unregister() to register() below. Note this comes with some pitfalls.
12 // Learn more about service workers: https://bit.ly/CRA-PWA
13 serviceWorker.unregister();
14
15
```

Line 13 has a red squiggle under the closing brace of the serviceWorker.unregister() call, indicating an error. Below the editor, the 'PROBLEMS' panel shows two errors:

- JS index.js src 2
 - ✖ 'App' is defined but never used. eslint(no-unused-vars) [4, 8]
 - ✖ Unnecessary semicolon. eslint(no-extra-semi) [13, 28]

On the right side of the slide, there is a decorative yellow triangle graphic.



ANY QUERIES ?





THANK YOU

