Microsoft / **vscode-docs**

Branch: **master** ▾ **vscode-docs** / docs / nodejs / **reactjs-tutorial.md** **Find file** **Copy path**

gregvanl Update DateApproved 2e38add on Dec 13, 2017

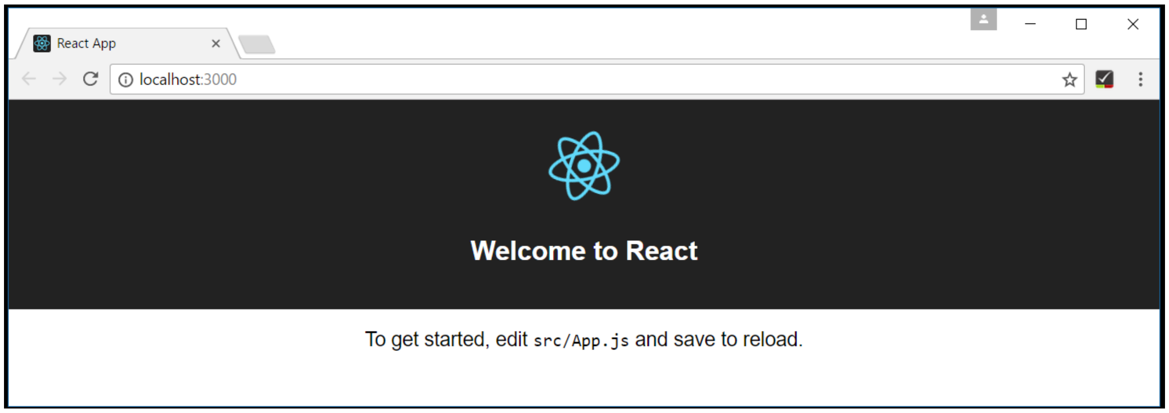
8 contributors

279 lines (182 sloc) 14.4 KB

Order	Area	TOCTitle	ContentId	PageTitle	DateApproved	MetaDescription	MetaSocialImage
5	nodejs	React Tutorial	2dd2eeff-2eb3-4a0c-a59d-ea9a0b10c468	React JavaScript Tutorial in VS Code	12/14/2017	React JavaScript tutorial showing IntelliSense, debugging, and code navigation support in the Visual Studio Code editor.	/assets/images/nodejs_javascript_vscode.png

Using React in VS Code

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



Welcome to React

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

> **Tip**: To test that you have Node.js and npm correctly install on your machine, you can type `node --version` and `npm --version`.

To install the `create-react-app` generator, in a terminal or command prompt type:

```
npm install -g create-react-app
```

This may take a few minutes to install. You can now create a new React application by typing:

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

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 it's dependencies.

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:

```
cd my-app
npm start
```

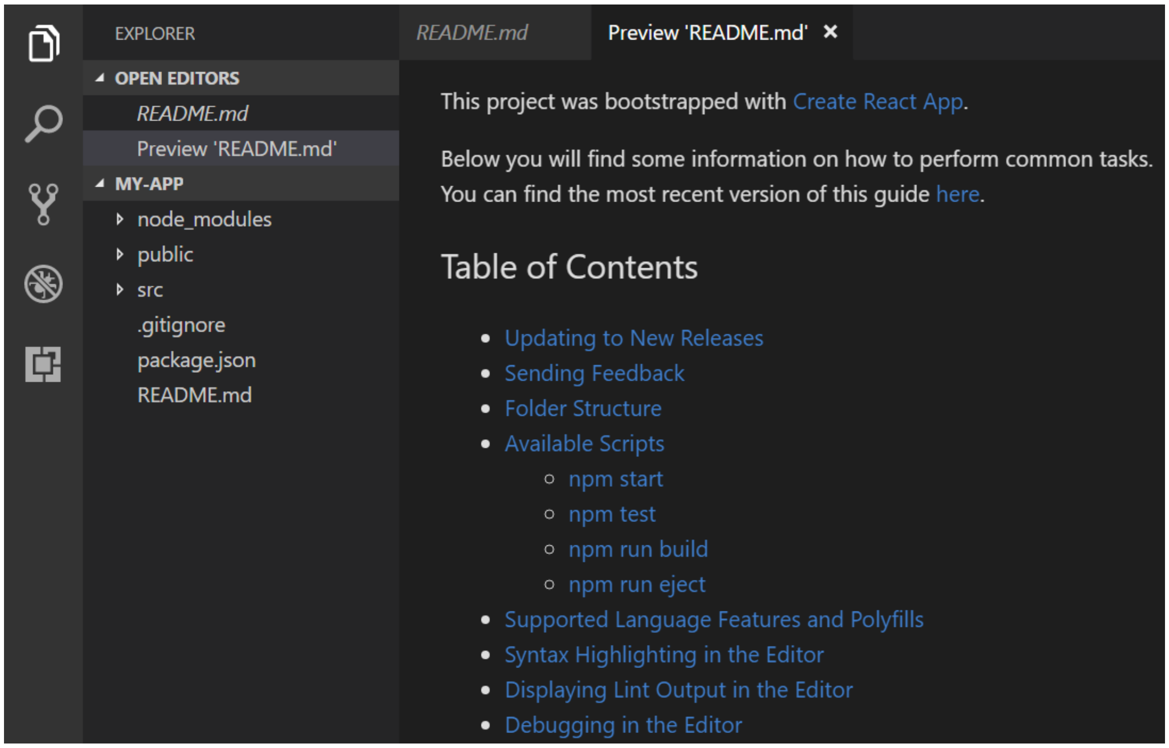
You should see "Welcome to React" on `http://localhost:3000` in your browser. We'll leave the web server running while we look at the application with VS Code.

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

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

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** `kb(markdown.showPreview)`) or in a new editor group to the side (**Markdown: Open Preview to the Side** `kb(markdown.showPreviewToSide)`). You'll get nice formatting, hyperlink navigation to headers, and syntax highlighting in code blocks.



This project was bootstrapped with Create React App.

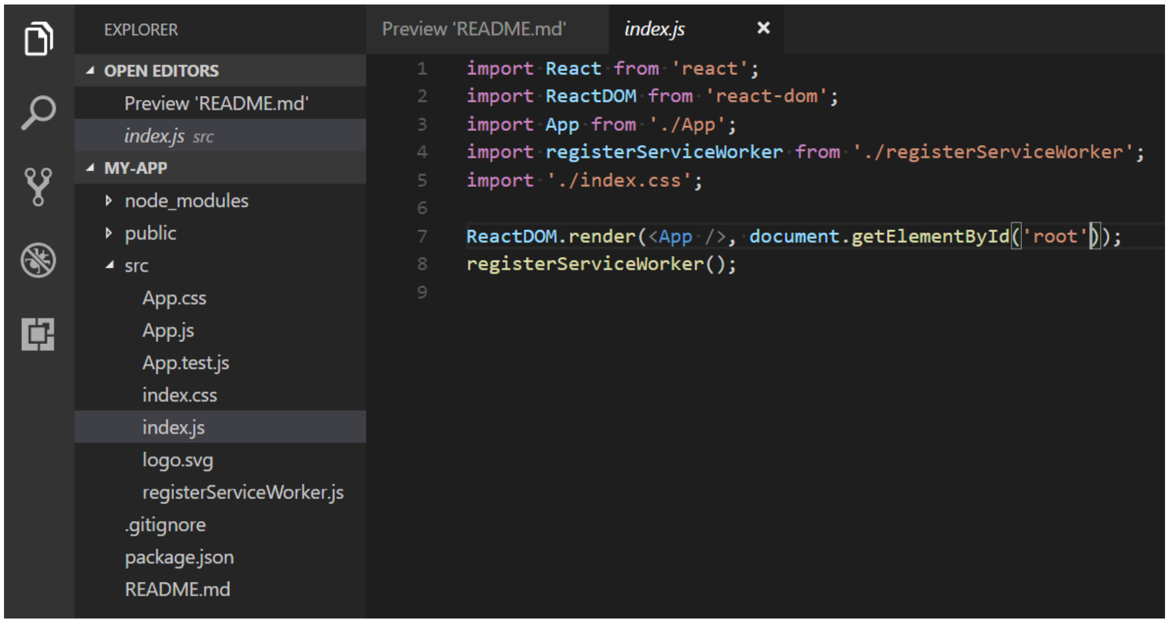
Below you will find some information on how to perform common tasks. You can find the most recent version of this guide here.

Table of Contents

- Updating to New Releases
- Sending Feedback
- Folder Structure
- Available Scripts
  - npm start
  - npm test
  - npm run build
  - npm run eject
- Supported Language Features and Polyfills
- Syntax Highlighting in the Editor
- Displaying Lint Output in the Editor
- Debugging in the Editor

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 parentheses, the matching bracket is also selected.

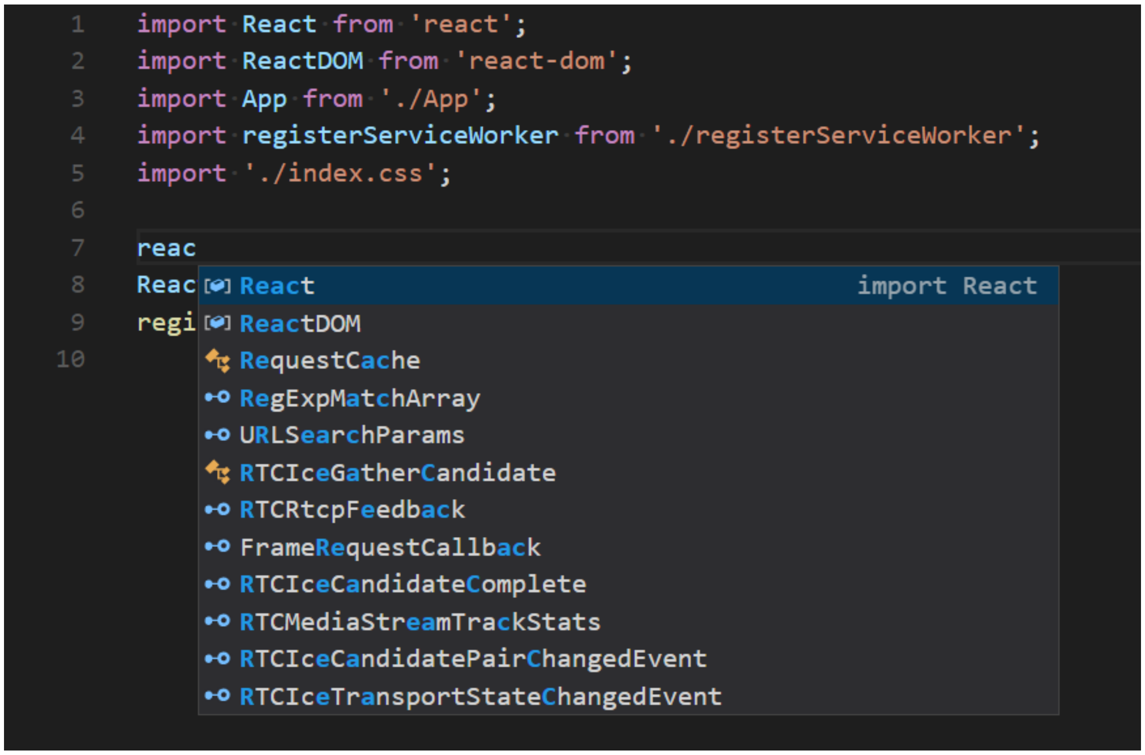


```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import './index.css';

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

IntelliSense

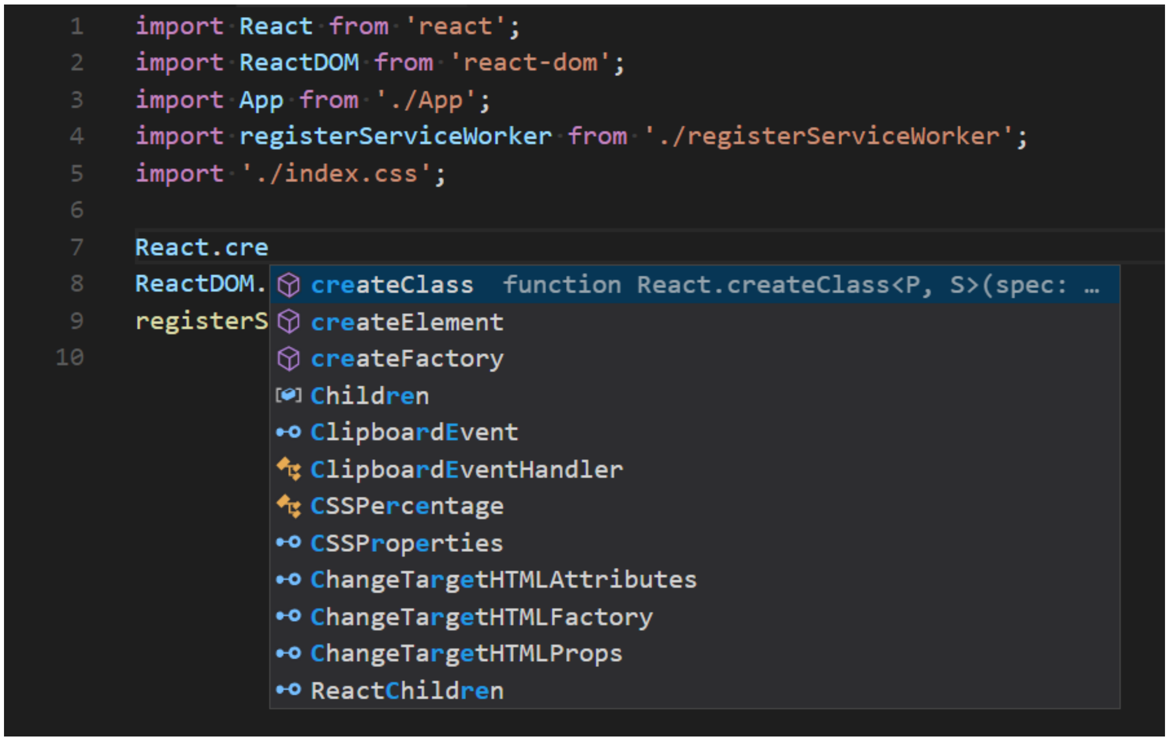
As you start typing in `index.js` , you'll see smart suggestions or completions.



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

reac
Reac
regi
```

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

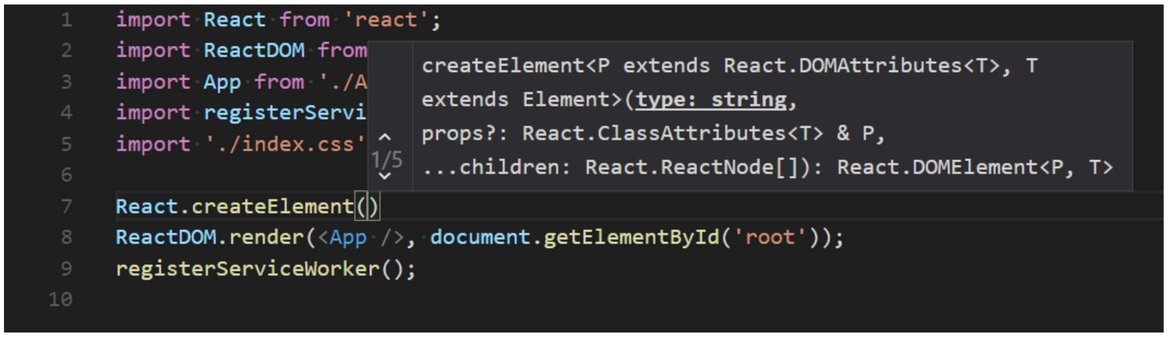


```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import './index.css';

React.cre
ReactDOM.
registerS
```

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:



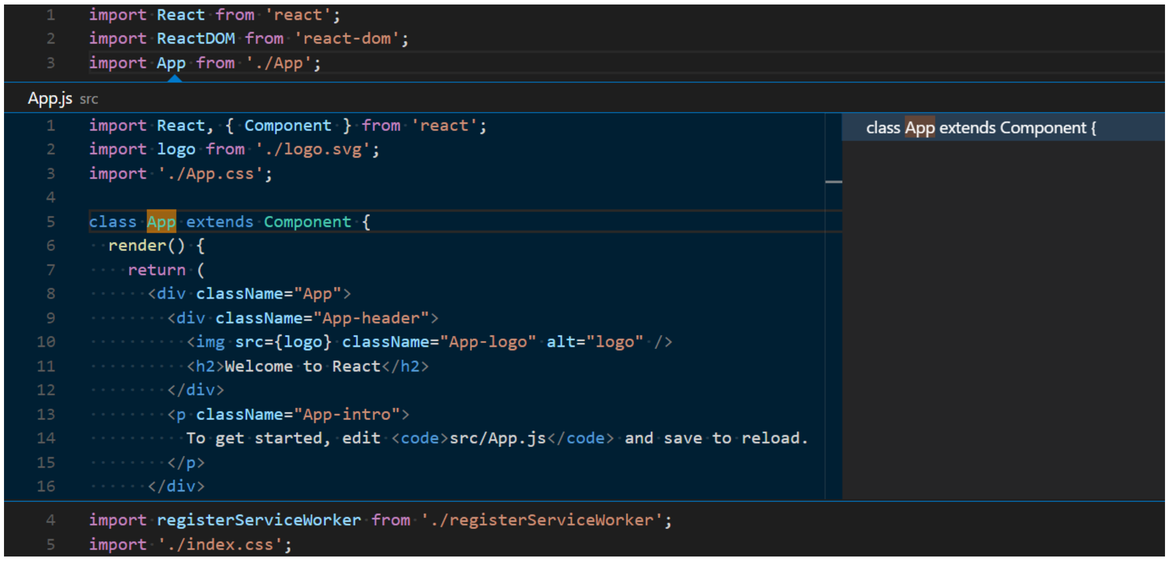
```
import React from 'react';
import ReactDOM from
import App from './A
import registerServi
import './index.css'

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

```
createElement<P extends React.DOMAttributes<T>, T
extends Element>(type: string,
props?: React.ClassAttributes<T> & P,
...children: React.ReactNode[]): React.DOMElement<P, T>
```

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** (`kb(editor.action.gotodeclaration)`) or **Peek Definition** (`kb(editor.action.peekImplementation)`). Put the cursor over the `App` , right click and select **Peek Definition**. A Peek window will open showing the `App` definition from `App.js` .



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

App.js src

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

class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
```

class App extends Component {

```
import registerServiceWorker from './registerServiceWorker';
import './index.css';
```

Press `kbstyle(Escape)` to close the Peek window.

Hello World!

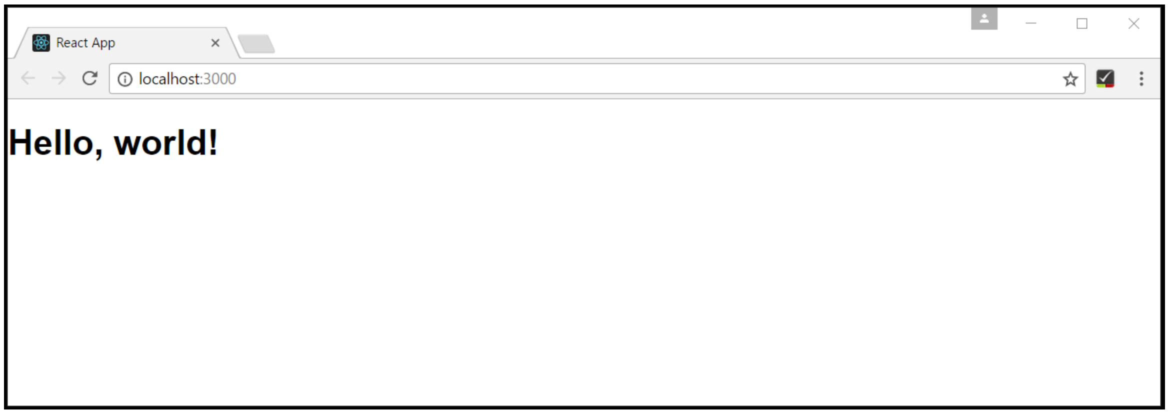
Let's update the sample application to "Hello World!". Add the link to declare a new H1 header and replace the `<App />` tag in `ReactDOM.render` with `element`.

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

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

Once you save the `index.js` file, the running instance of the server will update the web page and you'll see "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.



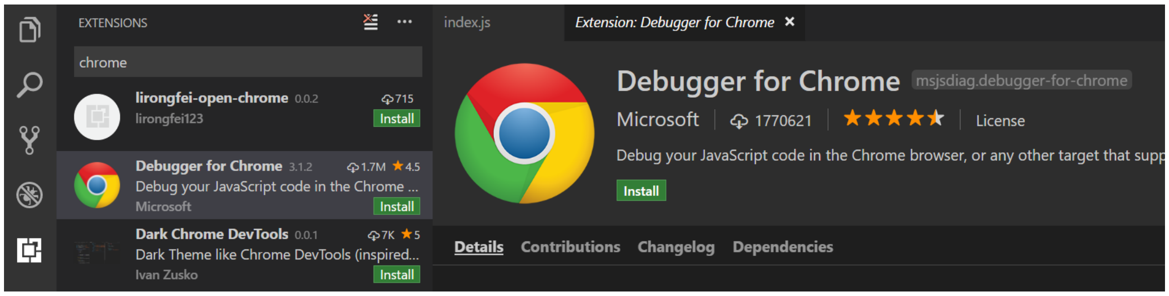
Hello, world!

Debugging React

To debug the client side React code, we'll need to install the Debugger for Chrome extension.

> Note: This tutorial assumes you have the Chrome browser installed. The builders of the Debugger for Chrome extension also have versions for the Safari on iOS and Edge browsers.

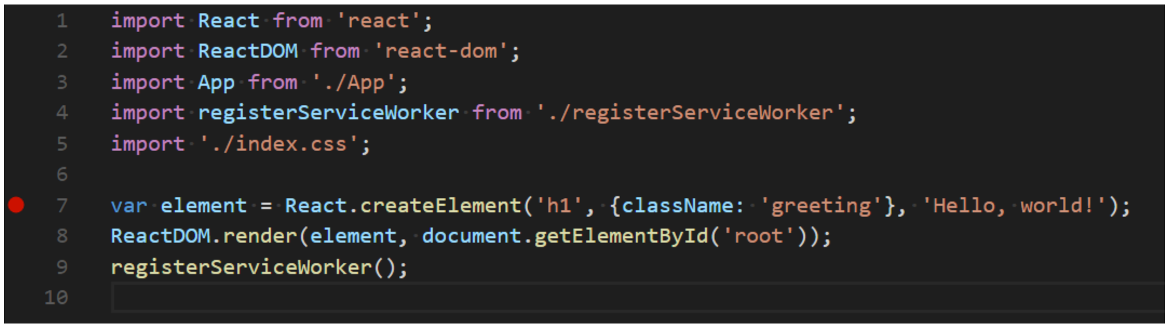
Open the Extensions view (`kb(workbench.view.extensions)`) and type 'chrome` in the search box. You'll see several extensions which reference Chrome.


EXTENSIONS
chrome
lirongfei-open-chrome 0.0.2
715
lirongfei123
Install
Debugger for Chrome 3.1.2
1.7M ★4.5
Debug your JavaScript code in the Chrome ...
Microsoft
Install
Dark Chrome DevTools 0.0.1
7K ★5
Dark Theme like Chrome DevTools (inspired...
Ivan Zusko
Install
index.js
Extension: Debugger for Chrome
Debugger for Chrome
msjsdiag.debugger-for-chrome
Microsoft
1770621
License
Debug your JavaScript code in the Chrome browser, or any other target that supp
Install
Details
Contributions
Changelog
Dependencies

Press the **Install** button for **Debugger for Chrome**. The button will change to **Installing** then, after completing the installation, it will change to **Reload**. Press **Reload** to restart VS Code and activate the extension.

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.



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

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

```

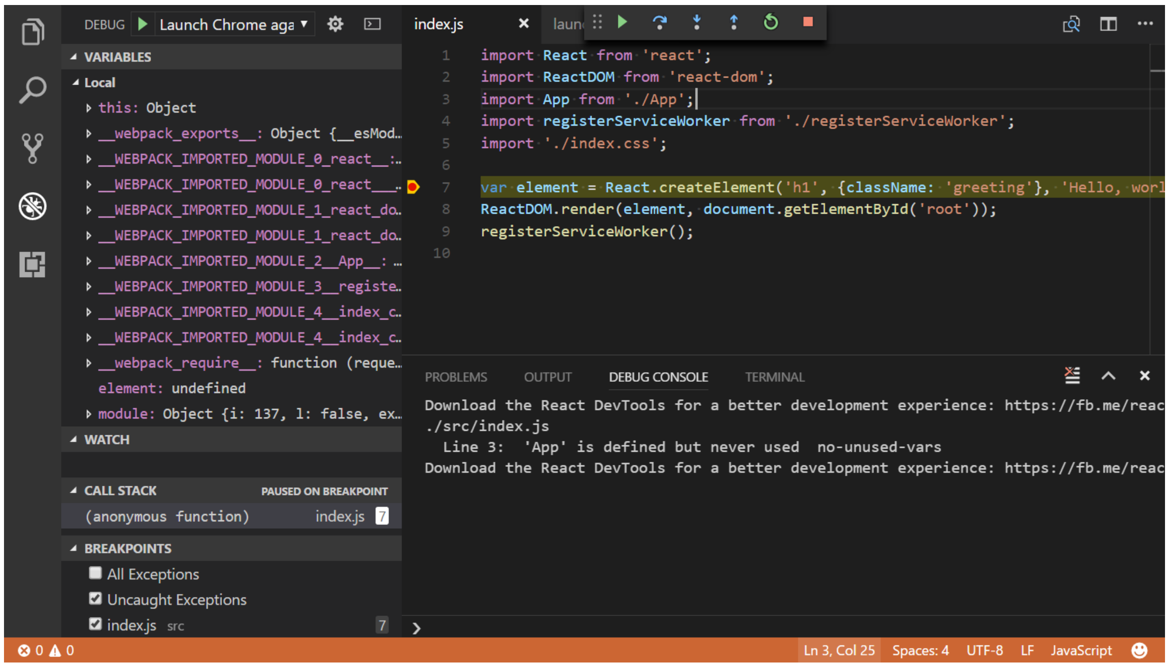
Configure the Chrome debugger

We need to initially configure the debugger. To do so, go to the Debug view (`kb(workbench.view.debug)`) and click on gear button to create a `launch.json` debugger configuration file. Choose **Chrome** from the **Select Environment** dropdown. This will create a `launch.json` file in a new `.vscode` folder in your project which includes configuration to both launch the website or attach to a running instance.

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

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "chrome",
            "request": "launch",
            "name": "Launch Chrome against localhost",
            "url": "http://localhost:3000",
            "webRoot": "${workspaceFolder}"
        },
        {
            "type": "chrome",
            "request": "attach",
            "name": "Attach to Chrome",
            "port": 9222,
            "webRoot": "${workspaceFolder}"
        }
    ]
}
```

Ensure that your development server is running ("npm start"). Then press `kb(workbench.action.debug.start)` 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.



```
this: Object
__webpack_exports__: Object {__esMod…
__WEBPACK_IMPORTED_MODULE_0_react__:…
__WEBPACK_IMPORTED_MODULE_0_react___…
__WEBPACK_IMPORTED_MODULE_1_react_do…
__WEBPACK_IMPORTED_MODULE_1_react_do…
__WEBPACK_IMPORTED_MODULE_2__App__: …
__WEBPACK_IMPORTED_MODULE_3__registe…
__WEBPACK_IMPORTED_MODULE_4__index_c…
__WEBPACK_IMPORTED_MODULE_4__index_c…
__webpack_require__: function (reque…
element: undefined
module: Object {i: 137, l: false, ex…
```

```
(anonymous function)    index.js 7
```

```
All Exceptions
Uncaught Exceptions
index.js src    7
```

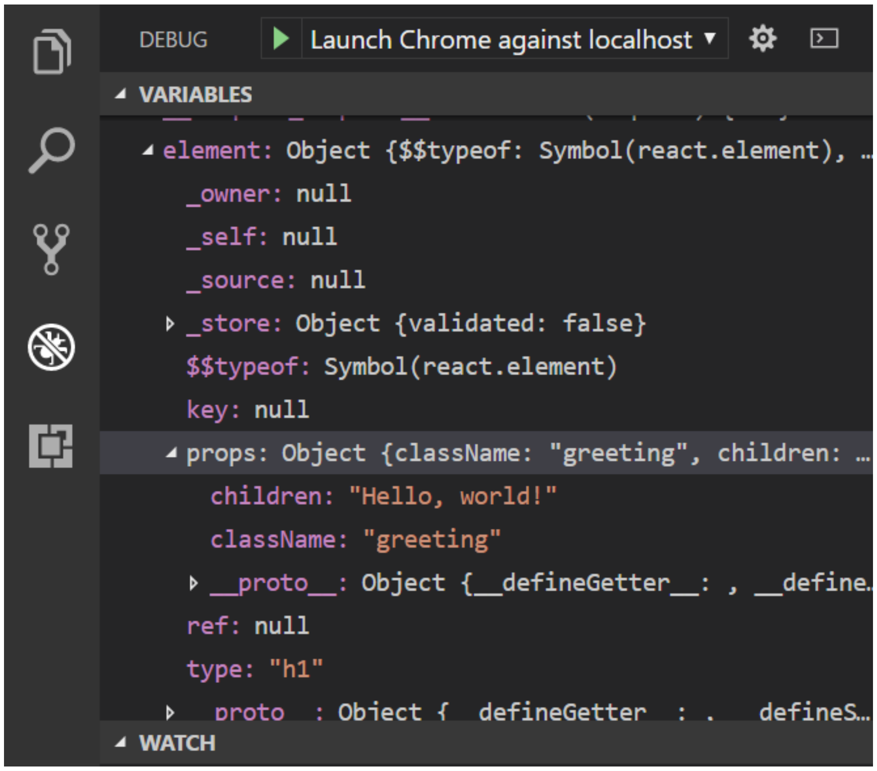
```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import './index.css';

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

```
Download the React DevTools for a better development experience: https://fb.me/reac
./src/index.js
  Line 3:  'App' is defined but never used  no-unused-vars
Download the React DevTools for a better development experience: https://fb.me/reac
```

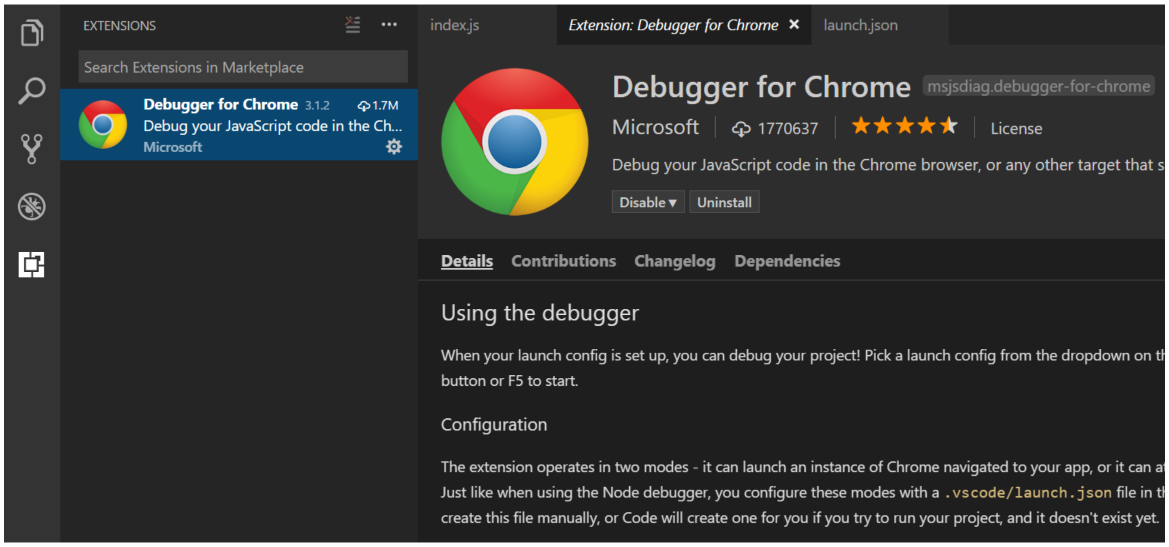
```
⊗ 0 ⚠ 0    Ln 3, Col 25   Spaces: 4   UTF-8   LF   JavaScript
```

You can step through your source code (`kb(workbench.action.debug.stepOver)`), inspect variables such as `element` , and see the call stack of the client side React application.



```
element: Object {$$typeof: Symbol(react.element), …
  _owner: null
  _self: null
  _source: null
  _store: Object {validated: false}
  $$typeof: Symbol(react.element)
  key: null
  props: Object {className: "greeting", children: …
    children: "Hello, world!"
    className: "greeting"
    __proto__: Object {__defineGetter__: , __define…
  ref: null
  type: "h1"
  __proto__: Object {__defineGetter__: , __defineS…
```

The **Debugger for Chrome** extension README has lots of information on other configurations, working with sourcemaps, and troubleshooting. You can review it directly within VS Code from the **Extensions** view by clicking on the extension item and opening the **Details** view.


EXTENSIONS
Search Extensions in Marketplace
Debugger for Chrome 3.1.2 1.7M
Debug your JavaScript code in the Ch...
Microsoft
index.js
Extension: Debugger for Chrome
launch.json
Debugger for Chrome msjsdiag.debugger-for-chrome
Microsoft 1770637 License
Disable
Uninstall
Details
Contributions
Changelog
Dependencies

Debug your JavaScript code in the Chrome browser, or any other target that s

Using the debugger

When your launch config is set up, you can debug your project! Pick a launch config from the dropdown on th
button or F5 to start.

Configuration

The extension operates in two modes - it can launch an instance of Chrome navigated to your app, or it can a
Just like when using the Node debugger, you configure these modes with a `.vscode/launch.json` file in th
create this file manually, or Code will create one for you if you try to run your project, and it doesn't exist yet.

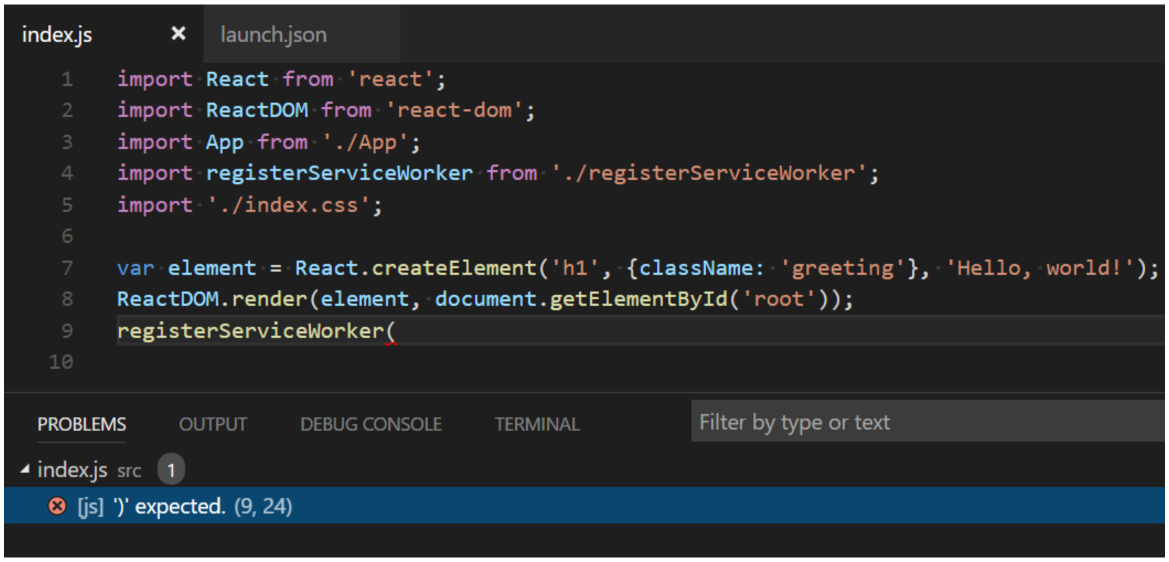
Live editing and debugging

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. You can learn more in this Live edit and debug your React apps directly from VS Code blog post.

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** `kb(workbench.actions.view.problems)`).

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



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

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

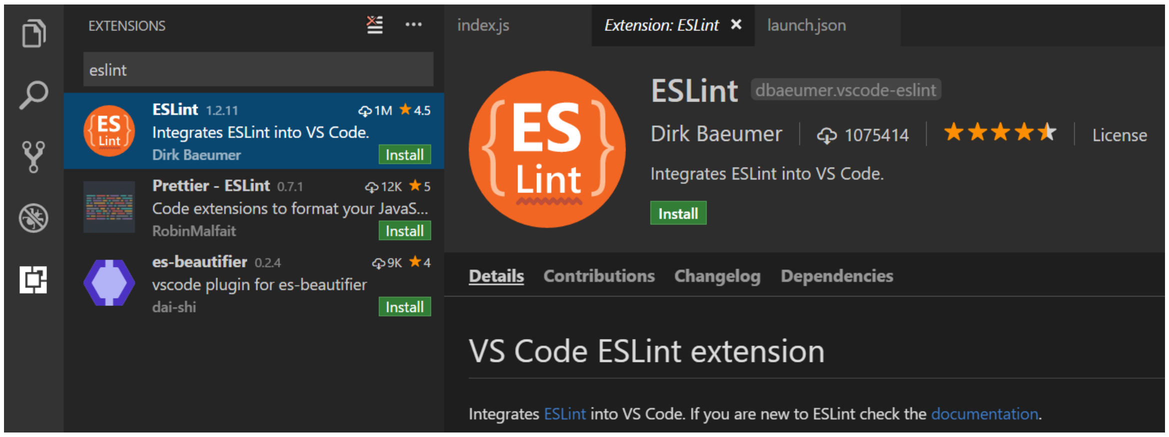
```
[js] ')' expected. (9, 24)
```

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
```

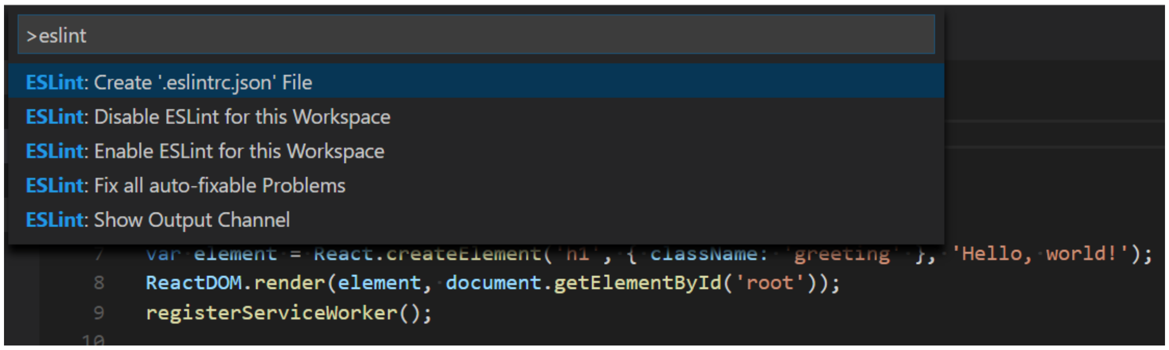
Then install the ESLint extension by going to the **Extensions** view and typing 'eslint'.


EXTENSIONS
eslint
ESLint 1.2.11 1M 4.5
Integrates ESLint into VS Code.
Dirk Baeumer
Install
Prettier - ESLint 0.7.1 12K 5
Code extensions to format your JavaS...
RobinMalfait
Install
es-beautifier 0.2.4 9K 4
vscode plugin for es-beautifier
dai-shi
Install
index.js
Extension: ESLint
launch.json
ESLint dbaeumer.vscode-eslint
Dirk Baeumer 1075414 License
Integrates ESLint into VS Code.
Install
Details
Contributions
Changelog
Dependencies

VS Code ESLint extension

Integrates ESLint into VS Code. If you are new to ESLint check the documentation.

Once the ESLint extension is installed and VS Code reloaded, you'll want to create an ESLint configuration file `eslintrc.json` . You can create one using the extension's **ESLint: Create 'eslintrc.json' File** command from the **Command Palette** (`kb(workbench.action.showCommands)`).


>eslint
ESLint: Create '.eslintrc.json' File
ESLint: Disable ESLint for this Workspace
ESLint: Enable ESLint for this Workspace
ESLint: Fix all auto-fixable Problems
ESLint: Show Output Channel

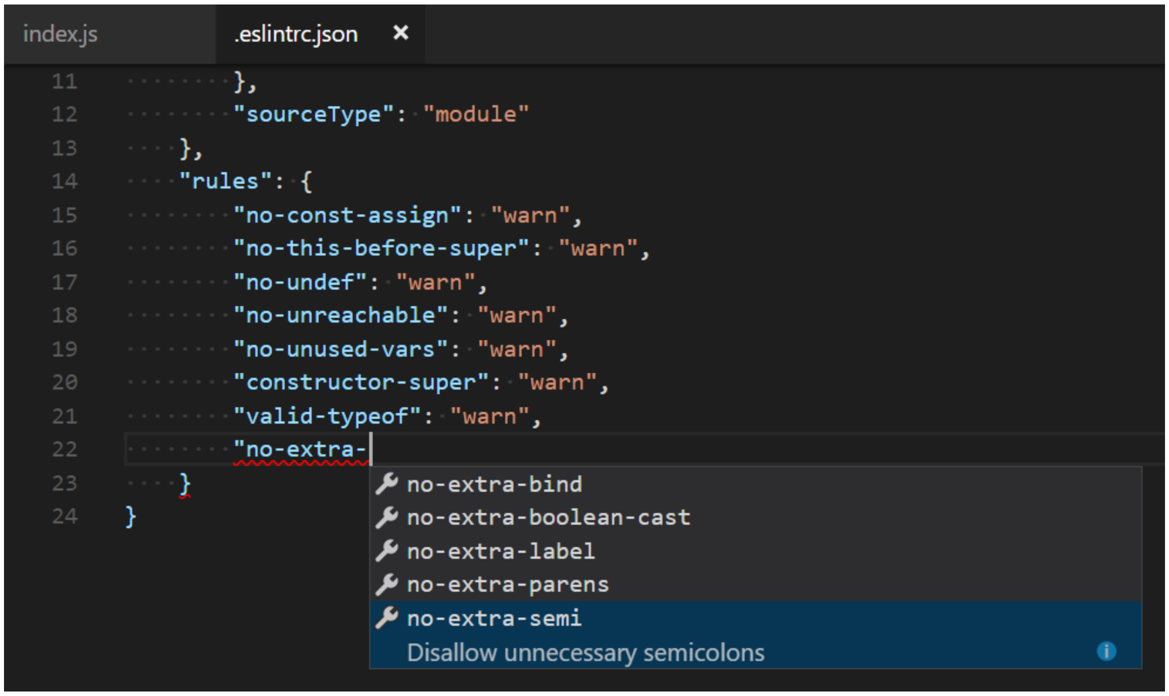
```
var element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');
ReactDOM.render(element, document.getElementById('root'));
registerServiceWorker();
```

The command will create a `.eslintrc.json` file in your project root:

```
{
    "env": {
        "browser": true,
        "commonjs": true,
        "es6": true,
        "node": true
    },
    "parserOptions": {
        "ecmaFeatures": {
            "jsx": true
        },
        "sourceType": "module"
    },
    "rules": {
        "no-const-assign": "warn",
        "no-this-before-super": "warn",
        "no-undef": "warn",
        "no-unreachable": "warn",
        "no-unused-vars": "warn",
        "constructor-super": "warn",
        "valid-typeof": "warn"
    }
}
```

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

You can modify the ESLint rules and the ESLint extension provides IntelliSense in `eslintrc.json` .

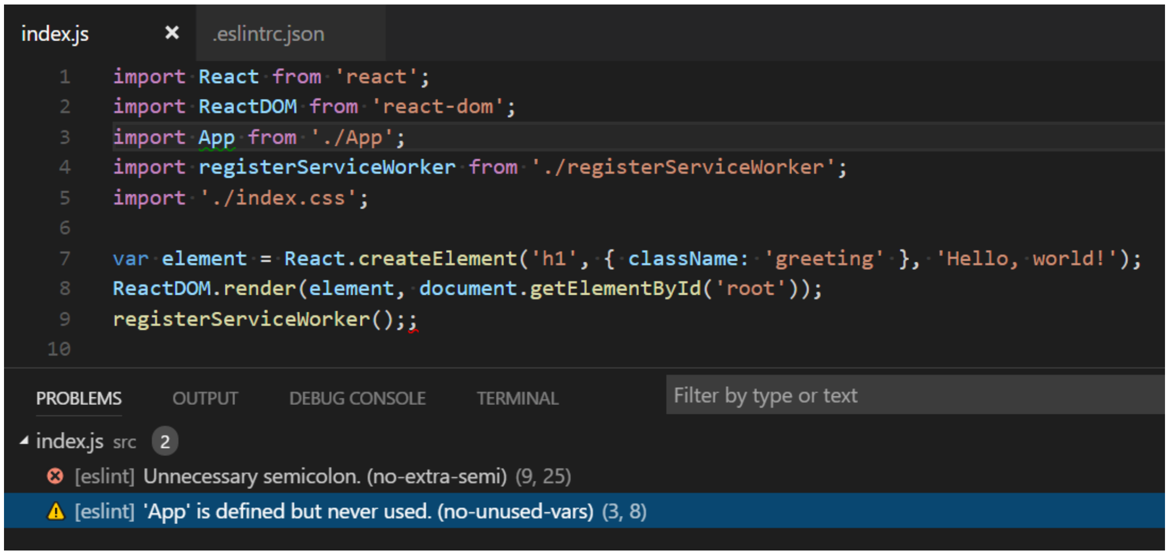


```
        },
        "sourceType": "module"
    },
    "rules": {
        "no-const-assign": "warn",
        "no-this-before-super": "warn",
        "no-undef": "warn",
        "no-unreachable": "warn",
        "no-unused-vars": "warn",
        "constructor-super": "warn",
        "valid-typeof": "warn",
        "no-extra-
    }
}
```

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

```
"rules": {
       "no-const-assign": "warn",
       "no-this-before-super": "warn",
       "no-undef": "warn",
       "no-unreachable": "warn",
       "no-unused-vars": "warn",
       "constructor-super": "warn",
       "valid-typeof": "warn",
       "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.



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

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

```

```
index.js src 2
[eslint] Unnecessary semicolon. (no-extra-semi) (9, 25)
[eslint] 'App' is defined but never used. (no-unused-vars) (3, 8)
```

Popular Starter Kits

In this tutorial, we used the `create-react-app` generator to create a simple React application. There are lots of great samples and starter kits available to help build your first React application.

VS Code React Sample

This is a sample React application used for a demo at this year's //Build conference. The sample creates a simple TODO application and includes the source code for a Node.js Express server. It also shows how to use the Babel ES6 transpiler and then use webpack to bundle the site assets.

MERN Starter

If you'd like to see a full MERN (MongoDB, Express, React, Node.js) stack example, look at the MERN Starter. You'll need to install and start MongoDB but you'll quickly have a MERN application running. There is helpful VS Code-specific documentation at vscode-recipes which details setting up Node.js server debugging.

TypeScript React

If you're curious about TypeScript and React, you can also create a TypeScript version of the `create-react-app` application. See the details at TypeScript-React-Starter on the TypeScript Quick Start site.

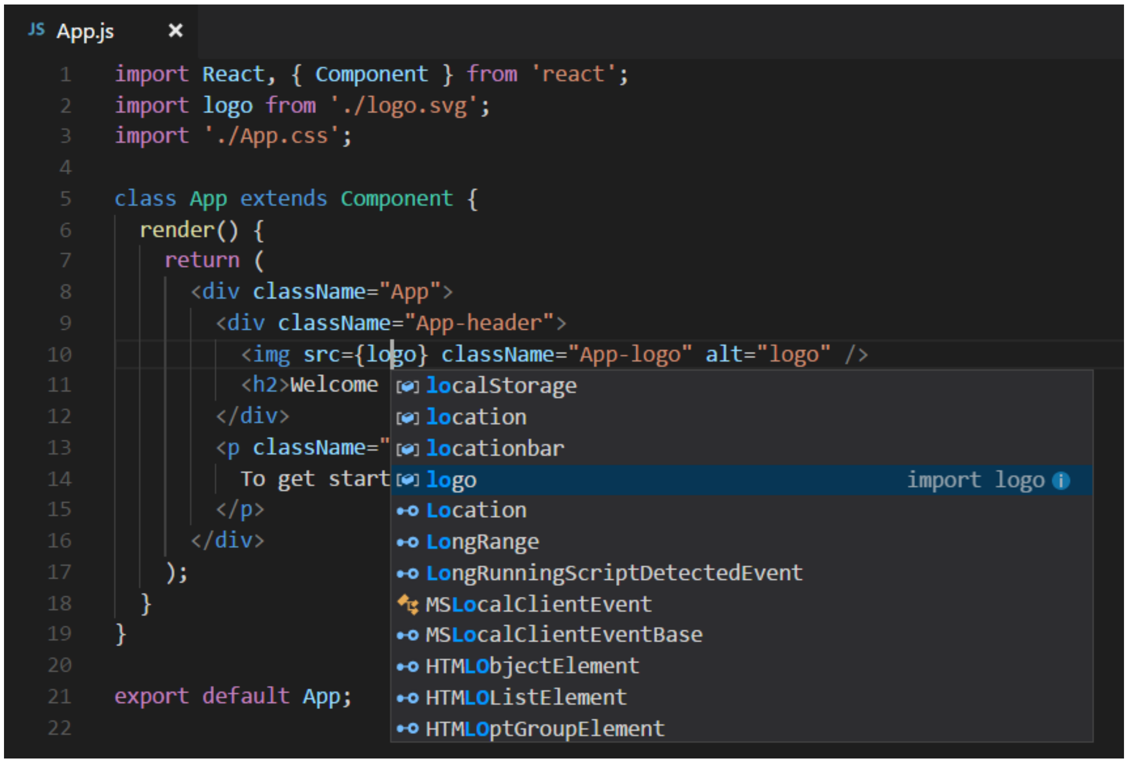
Angular

Angular is another popular web framework. If you'd like to see an example of Angular working with VS Code, check out the Chrome Debugging with Angular CLI recipe. It will walk you through creating an Angular application and configuring the `launch.json` file for the Debugger for Chrome extension.

Common Questions

Q: Can I get IntelliSense within declarative JSX?

A: Yes. For example, if you open the `create-react-app` project's `app.js` file, you can see IntelliSense within the React JSX in the `render()` method.


App.js

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

class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome
        </div>
        <p className="
          To get start
        </p>
      </div>
    );
  }
}

export default App;
```

```
localStorage
location
locationbar
logo                                   import logo
Location
LongRange
LongRunningScriptDetectedEvent
MSLocalClientEvent
MSLocalClientEventBase
HTMLObjectElement
HTMLOListElement
HTMLOptGroupElement
```