Hyperiondev

TASK

Setting up Visual Studio Editor for Node and ReactJS (FAQ)

Visit our website

TABLE OF CONTENTS

Introduction

There are various editors that can be used effectively for creating applications using React and Express. You are already familiar with Sublime Text Editor. Visual Studio Code is another editor that supports full stack web development well. Many find Visual Studio Code (VS Code) easier to use for building, debugging and refactoring React apps. We recommend that you try it out. You are, however, welcome to use whichever editor you prefer for development. This FAQ doc will help you install and configure VS code for full stack web development with JavaScript. Notice that there is help for configuring Sublime for React development in this guide too.

HOW DO YOU CONFIGURE VS CODE FOR NODE DEVELOPMENT?

Installation

To run a Node.js application, you will need to install the Node.js runtime on your machine.

Configure your system environment variables

This process has already been described in the Debugging JavaScript FAQ created for Sublime.

Tip: To test that you've got node.js correctly installed on your computer, open a new terminal and type `node --help` and you should see the usage documentation.

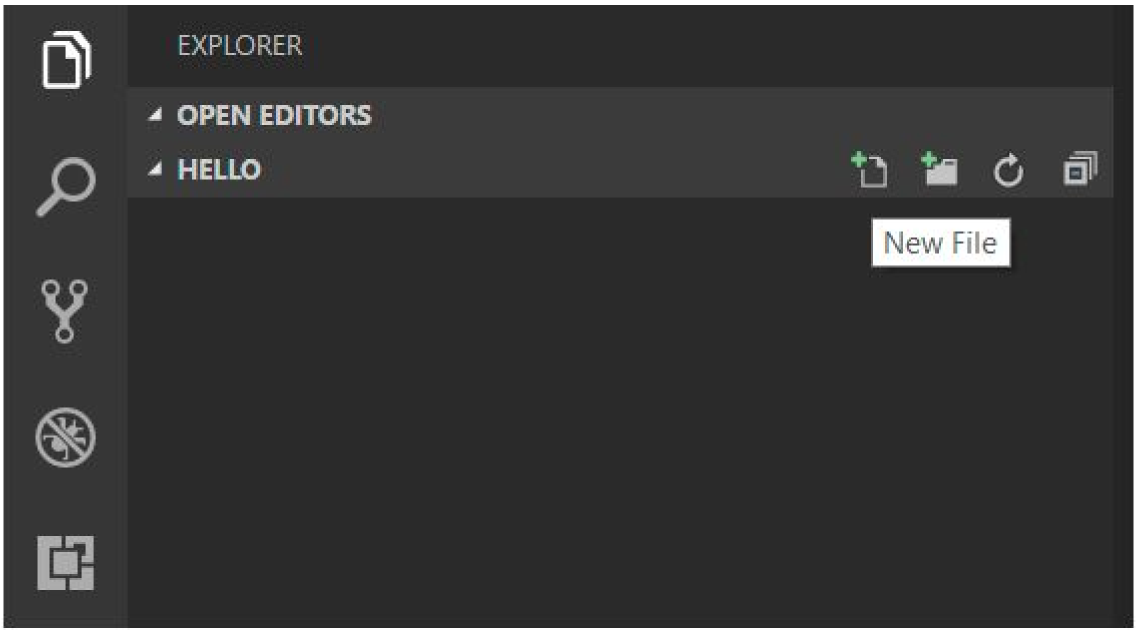
Configure your editor: Visual Studio Code (VS Code)

- Follow this link to install VS Code on your machine.

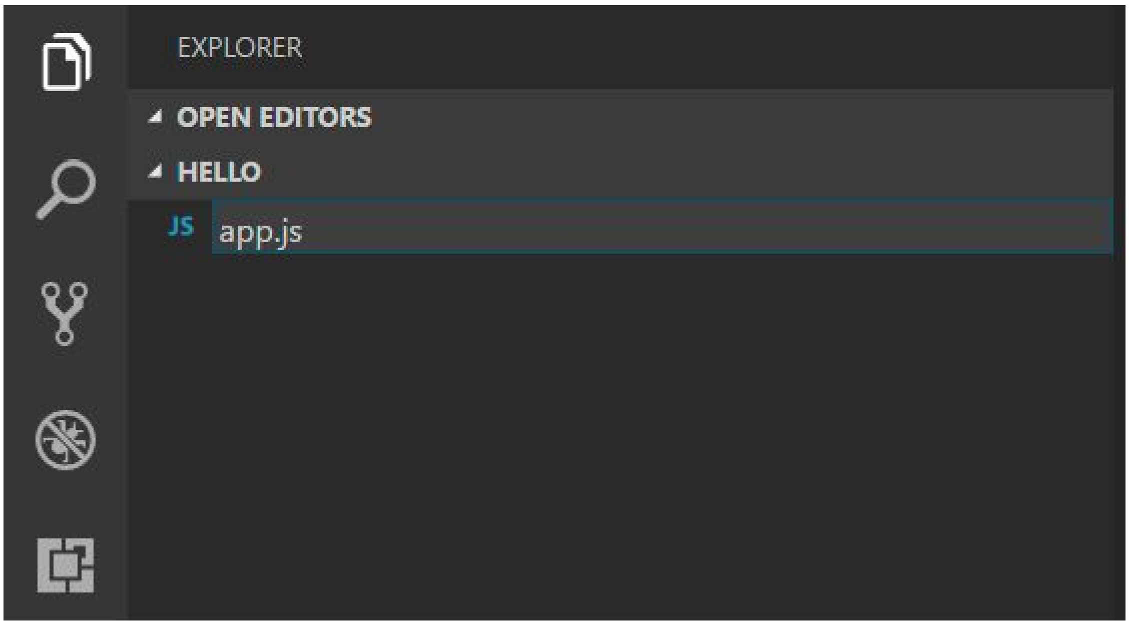
To start running JavaScript files:

- Create a new folder via the Windows Explorer
- Open the folder via VS Code and add JavaScript files via the File Explorer. Assuming that we have a folder called **Hello**, we want to create a new file called **app.js** which we're going to run using Node.

From the File Explorer toolbar, press the New File button:

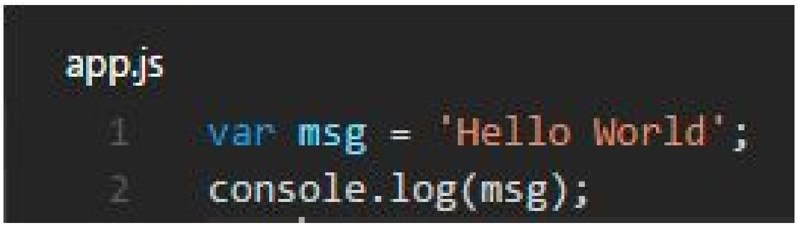


Create a new file called **app.js:**



By using the **.js** file extension, VS Code interprets this file as JavaScript and will evaluate the contents with the JavaScript language service.

Write a simple program in this file:



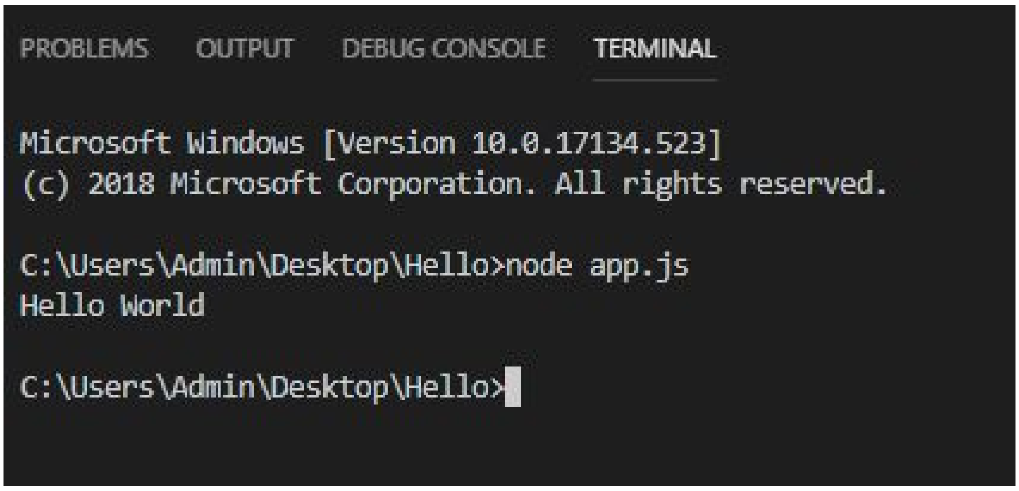
```
app.js
var msg = 'Hello World';
console.log(msg);
```

- To run this file, you could open the command line terminal (used to run shell commands) and type: `node app.js`

- You should see "Hello World" output to the terminal and then Node.js returns.

However, VS Code has an integrated terminal which you can use to run shell commands. You can run Node.js directly from there and avoid switching out of VS Code while running command line tools.

Navigate to **View** > **Terminal** (Ctrl+` with the backtick character) on the VS Code menu bar to open the integrated terminal and you can run `node app.js` there:



```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Admin\Desktop\Hello>node app.js
Hello World

C:\Users\Admin\Desktop\Hello>
```

Note: The integrated terminal opens the file path to the where the Hello folder is located. This ensures that the terminal has access to the app.js we created. Always ensure that the file path points to your files. You can read more about how to do this here.

HOW DO YOU CONFIGURE VS CODE FOR REACT DEVELOPMENT?

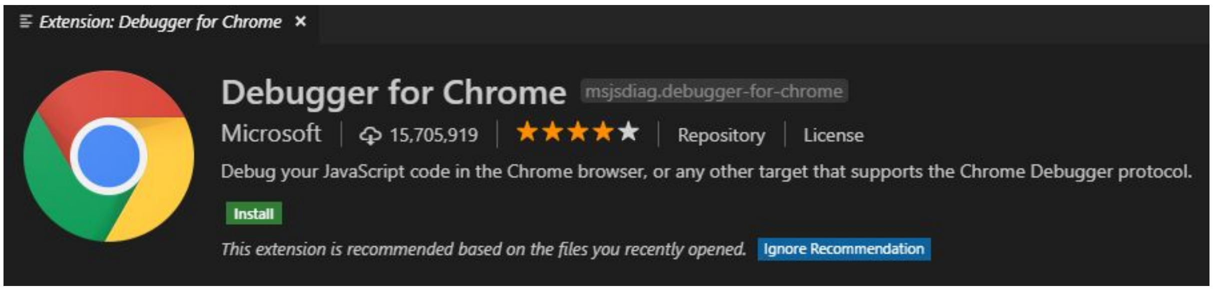
Create React App comes with a bunch of tools that improve the editing experience and maximize your productivity- if configured correctly. This enables you as a developer to write and debug your React code without leaving the editor, and most importantly it enables you to have a continuous development workflow, where context switching is minimal, as you don't have to switch between tools.

Running React in VS Code

You would need to have the latest version of VS Code and VS Code Chrome Debugger Extension installed.

You could also download the Chrome Debugger Extension via VS Code by clicking on the Extensions icon(highlighted on the left vertical toolbar) and searching for the debugger:

Install the extension by clicking on the Install button:



Extension: Debugger for Chrome ×

Debugger for Chrome msjsdiag.debugger-for-chrome

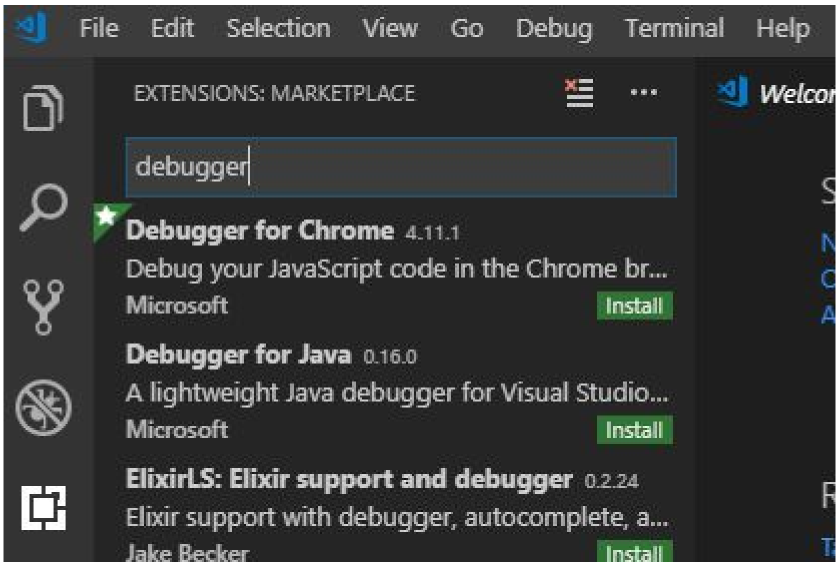
Microsoft | 15,705,919 | ★★★★★ | Repository | License

Debug your JavaScript code in the Chrome browser, or any other target that supports the Chrome Debugger protocol.

Install

This extension is recommended based on the files you recently opened. Ignore Recommendation

After completing the installation, the Install button will change to Reload. Press Reload to restart VS Code and activate the extension.



debugger

Debugger for Chrome 4.11.1
Debug your JavaScript code in the Chrome br...
Microsoft Install

Debugger for Java 0.16.0
A lightweight Java debugger for Visual Studio...
Microsoft Install

ElixirLS: Elixir support and debugger 0.2.24
Elixir support with debugger, autocomplete, a...
Jake Becker Install

To start running React apps:

- Create a new folder via the Windows Explorer
- Open the folder via VS Code and
- Open the integrated terminal ensuring that the file path points to your app folder

- You can create a React app using the create-react-app generator tool via VS Code by typing: `npm install -g create-react-app` and following the instructions in the task document to create your first React app!
- Start your app by running `npm start`

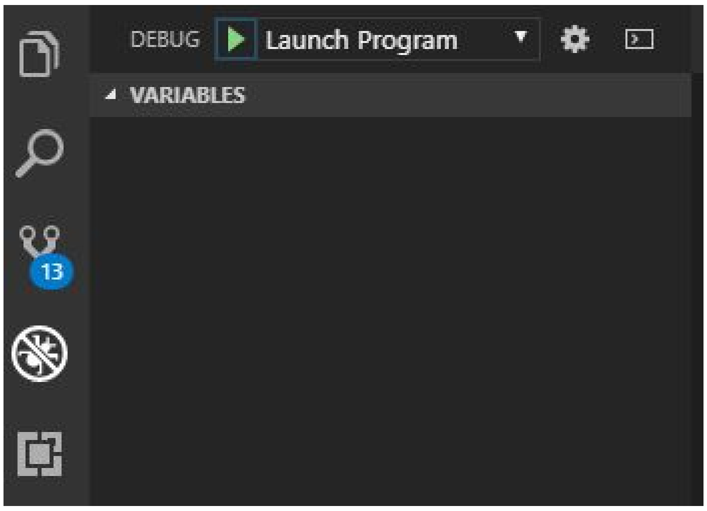
Note: npm is included with Node.js which you can download and install from here.

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.

HOW DO YOU DEBUG REACT APPS IN VS CODE?

To debug our React app, we need to initially configure the Chrome debugger:

- Go to the Debug view (Ctrl+Shift+D or click on the icon with the crossed out bug on the left vertical toolbar) and
- Click on the gear button to create a launch.json debugger configuration file.



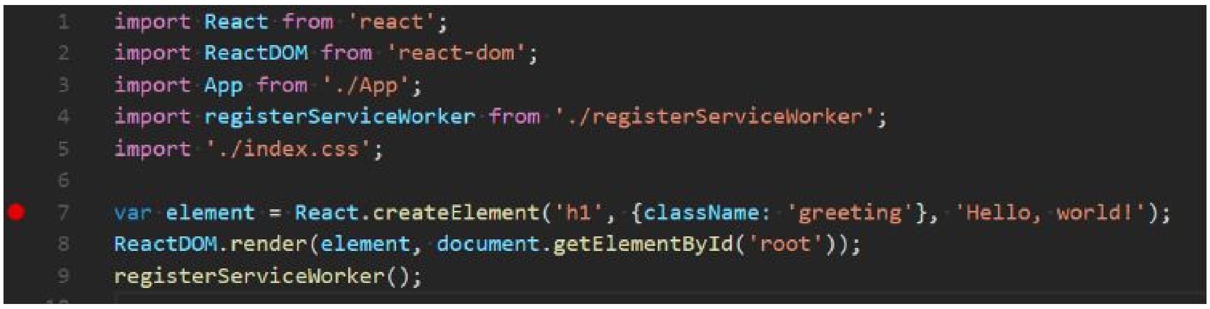
- Choose Chrome from the Select Environment drop-down list. This will create a launch.json file in a new .vscode folder in your project which includes a configuration to launch the website.

We need to make one change for our default app: change the port of the url 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}"
        }
    ]
}
```

We can test the debugging feature by setting a breakpoint in our index.js file:

- 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.

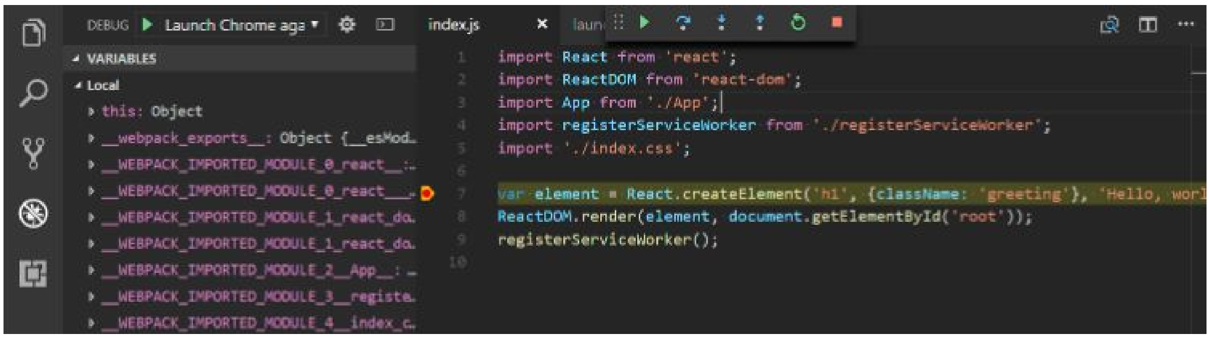


```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import './index.css';

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

- 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.

Note: 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.

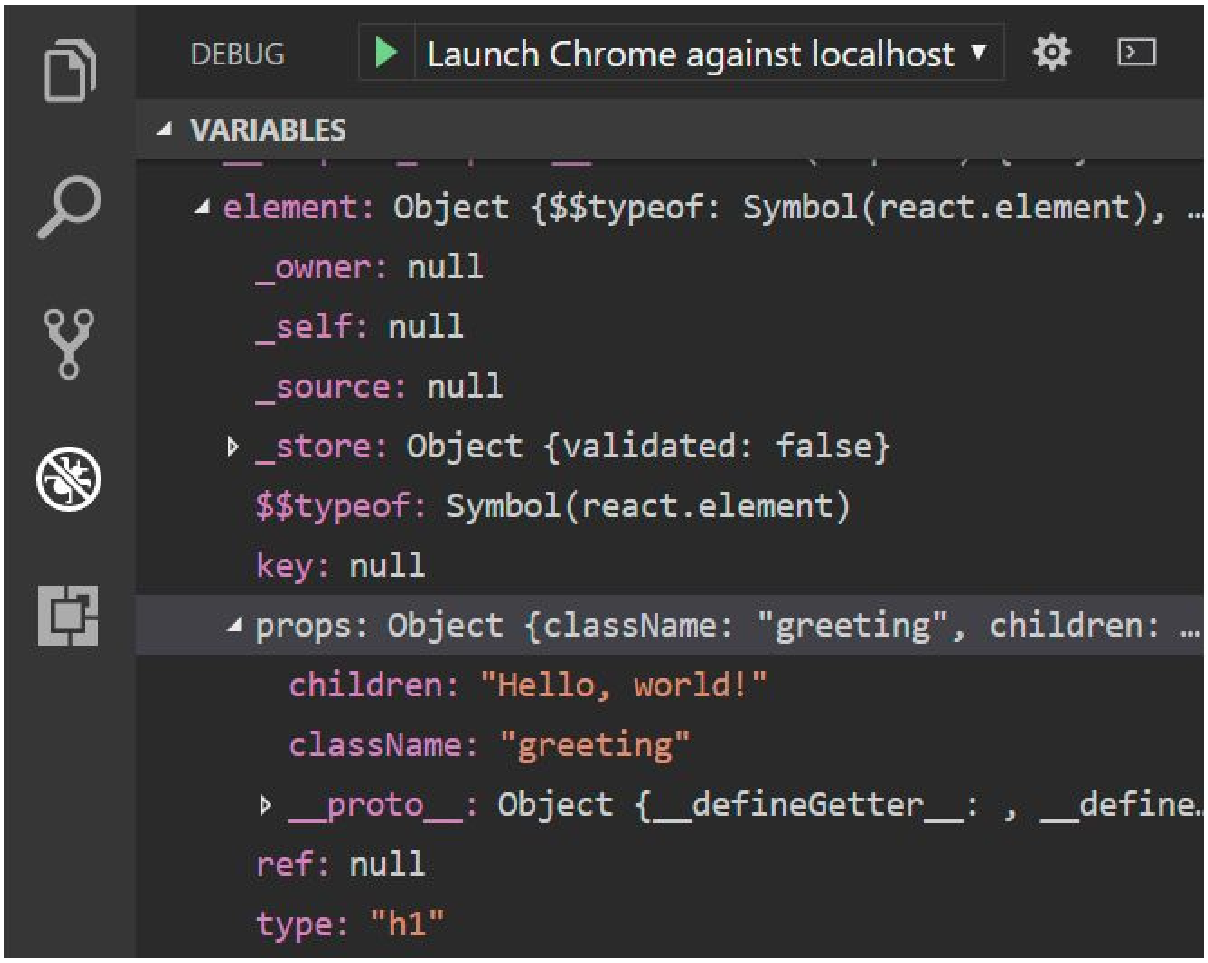


```
VARIABLES
  Local
    this: Object
    __webpack_exports__: Object {__esMod…
    __WEBPACK_IMPORTED_MODULE_0_react__:…
    __WEBPACK_IMPORTED_MODULE_0_react___…
    __WEBPACK_IMPORTED_MODULE_1_react_do…
    __WEBPACK_IMPORTED_MODULE_1_react_do…
    __WEBPACK_IMPORTED_MODULE_2__App__: …
    __WEBPACK_IMPORTED_MODULE_3__registe…
    __WEBPACK_IMPORTED_MODULE_4__index_c…
```

```
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();
```

- You can step through your source code (F10), 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"
```

You can now write code, set breakpoints, make changes to the code, and debug your newly modified code—all from your editor.

CAN I USE SUBLIME FOR REACT DEVELOPMENT?

Yes. You can set up Sublime to format your React code more appropriately. You can check out this guide for more information.