# Learning Outcome

# After completing this module, the student should be Able to develop the real time scenarios based on Node JS applications.

# To meet the learning outcome, a student has to complete the following activities

1. Create a basic Program with Node

# Activity 1

## Aim: Create a basic Program with Node

**Learning outcome:** Able to develop the real time scenarios based on Node JS applications.

**Duration:** 5 hour

**List of Hardware/Software requirements:**

1. Git
2. Node.js and npm
3. Angular CLI
4. IDE (e.g. Visual Studio Code

**Code/Program/Procedure (with comments):**

Let's get started by creating the simplest Node.js application, "Hello World".

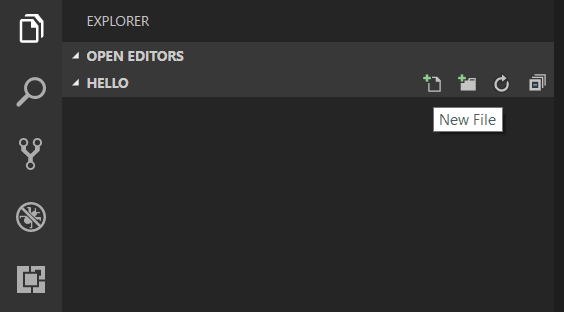
Create an empty folder called "hello", navigate into and open VS Code:

mkdir hello

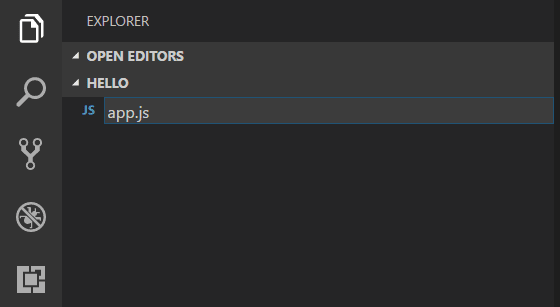
cd hello

code .

From the File Explorer toolbar, press the New File button



and name the file app.js:



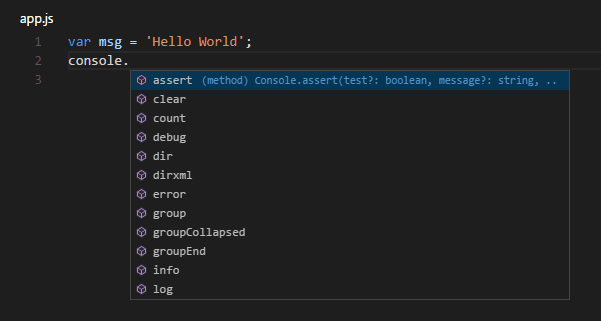
By using the .js file extension, VS Code interprets this file as JavaScript and will evaluate the contents with the JavaScript language service. Refer to the VS Code JavaScript language topic to learn more about JavaScript support.

Create a simple string variable in app.js and send the contents of the string to the console:

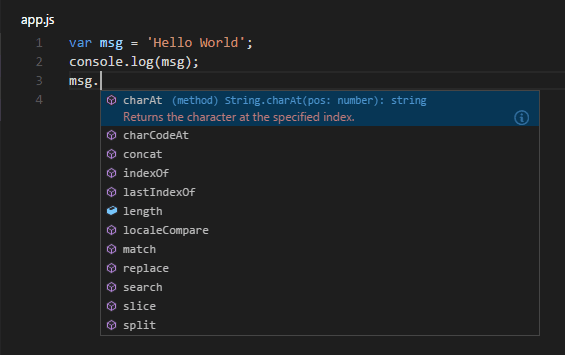
var msg = 'Hello World';

console.log(msg);

Note that when you typed console. IntelliSense on the console object was automatically presented to you.



Also notice that VS Code knows that msg is a string based on the initialization to 'Hello World'. If you type msg. you'll see IntelliSense showing all of the string functions available on msg.



After experimenting with IntelliSense, revert any extra changes from the source code example above and save the file (Ctrl+S).

**Running Hello World#**

It's simple to run app.js with Node.js. From a terminal, just type:

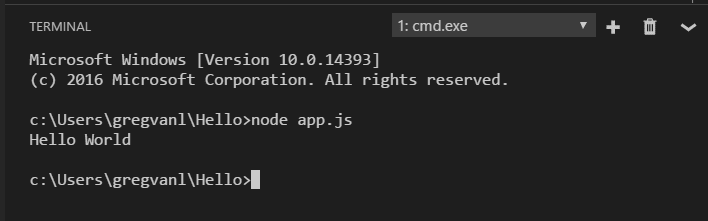
node app.js

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

**Integrated Terminal#**

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.

View > Terminal (Ctrl+` with the backtick character) will open the integrated terminal and you can run node app.js there:

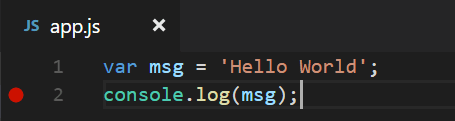


For this walkthrough, you can use either an external terminal or the VS Code integrated terminal for running the command-line tools.

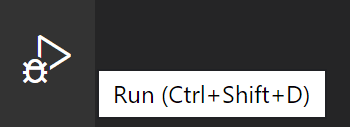
**Debugging Hello World#**

As mentioned in the introduction, VS Code ships with a debugger for Node.js applications. Let's try debugging our simple Hello World application.

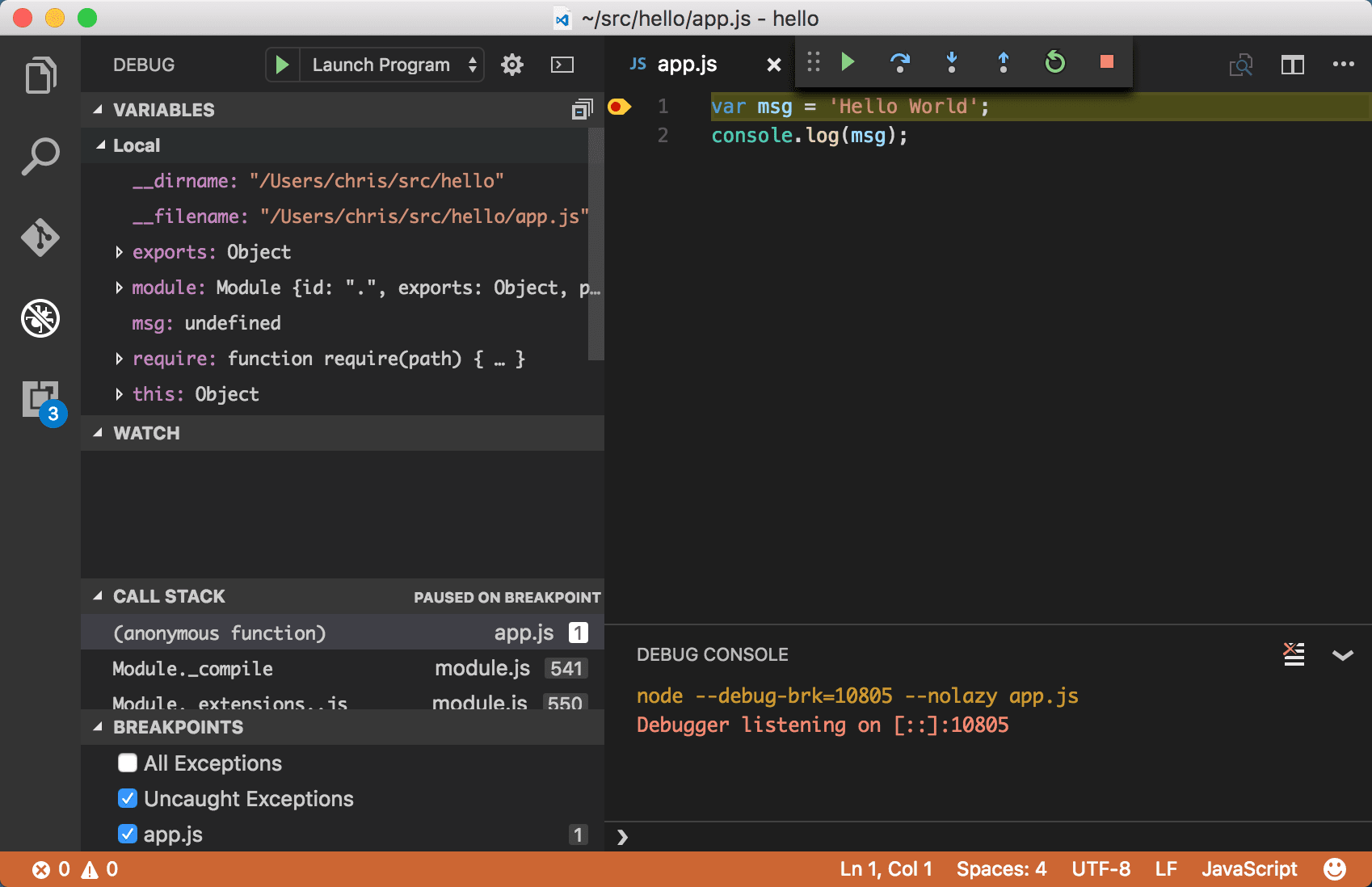
To set a breakpoint in app.js, put the editor cursor on the first line and press F9 or click in the editor left gutter next to the line numbers. A red circle will appear in the gutter.



To start debugging, select the Run View in the Activity Bar:



You can now click Debug toolbar green arrow or press F5 to launch and debug "Hello World". Your breakpoint will be hit and you can view and step through the simple application. Notice that VS Code displays a different colored Status Bar to indicate it is in Debug mode and the DEBUG CONSOLE is displayed.



Now that you've seen VS Code in action with "Hello World", the next section shows using VS Code with a full-stack Node.js web app.

**An Express application#**

Express is a very popular application framework for building and running Node.js applications. You can scaffold (create) a new Express application using the Express Generator tool. The Express Generator is shipped as an npm module and installed by using the npm command-line tool npm.

Install the Express Generator by running the following from a terminal:

npm install -g express-generator

The -g switch installs the Express Generator globally on your machine so you can run it from anywhere.

We can now scaffold a new Express application called myExpressApp by running:

express myExpressApp --view pug

This creates a new folder called myExpressApp with the contents of your application. The --view pug parameters tell the generator to use the pug template engine.

To install all of the application's dependencies (again shipped as npm modules), go to the new folder and execute npm install:

cd myExpressApp

npm install

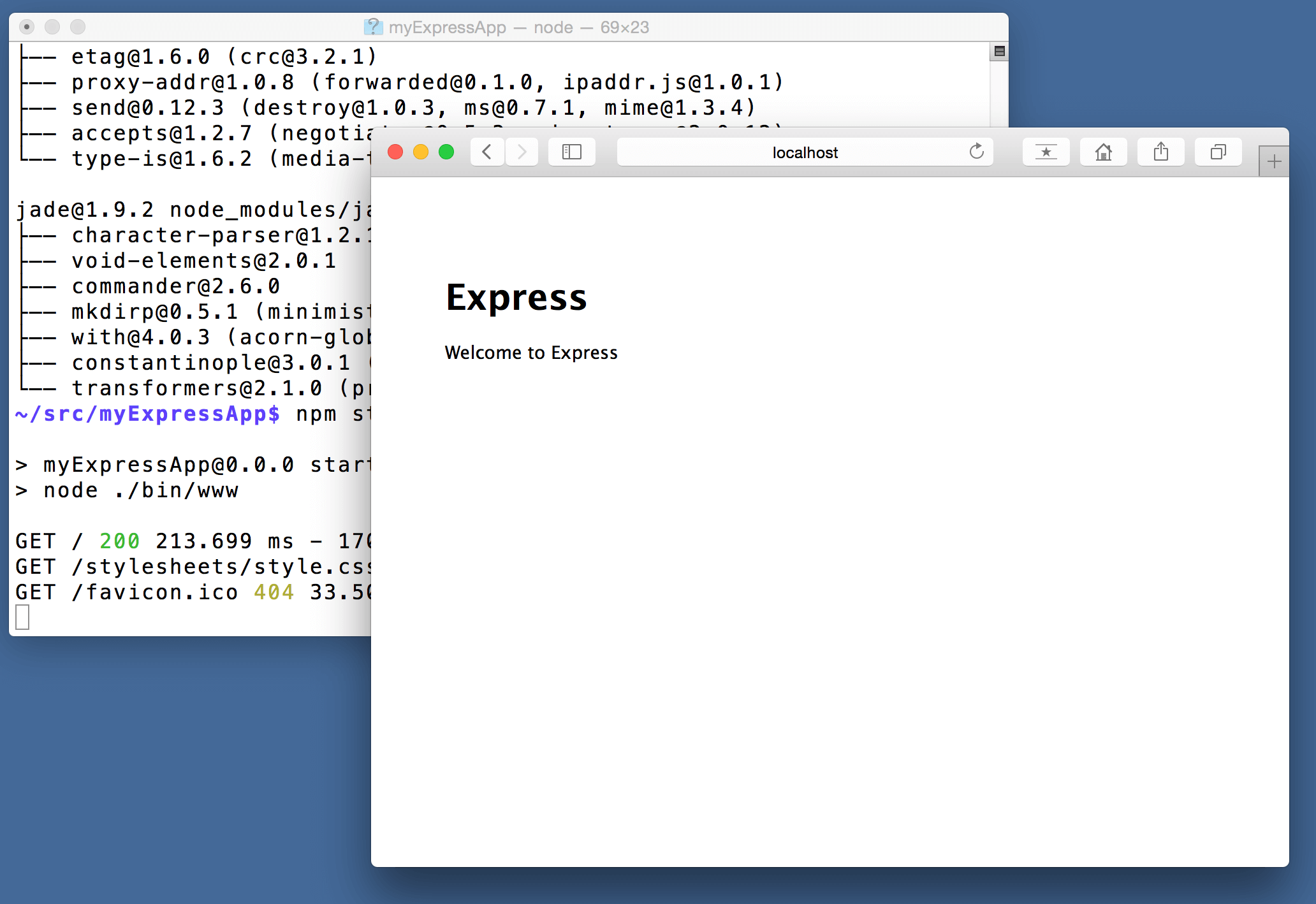
At this point, we should test that our application runs. The generated Express application has a package.json file which includes a start script to run node ./bin/www. This will start the Node.js application running.

From a terminal in the Express application folder, run:

npm start

The Node.js web server will start and you can browse to http://localhost:3000 to see the running application.

Your first Node Express App



**Great code editing#**

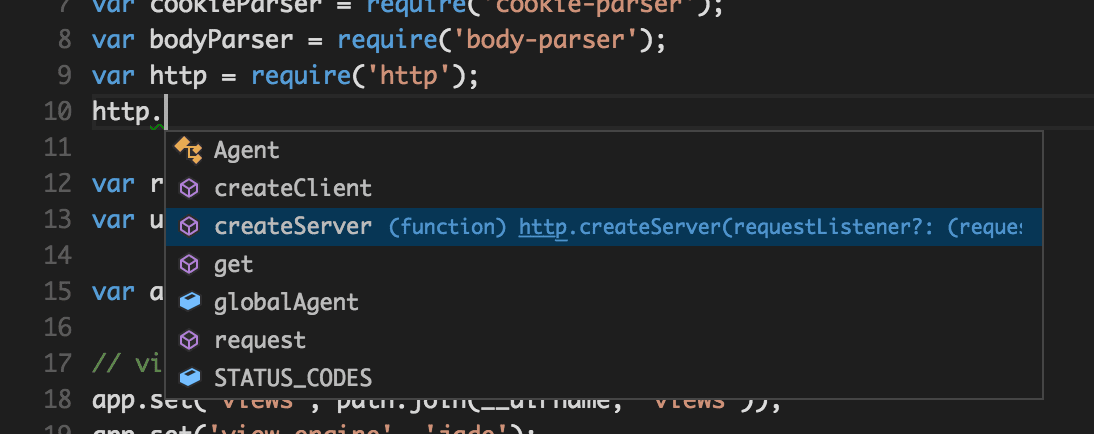
Close the browser and from a terminal in the myExpressApp folder, stop the Node.js server by pressing CTRL+C.

Now launch VS Code:

code .

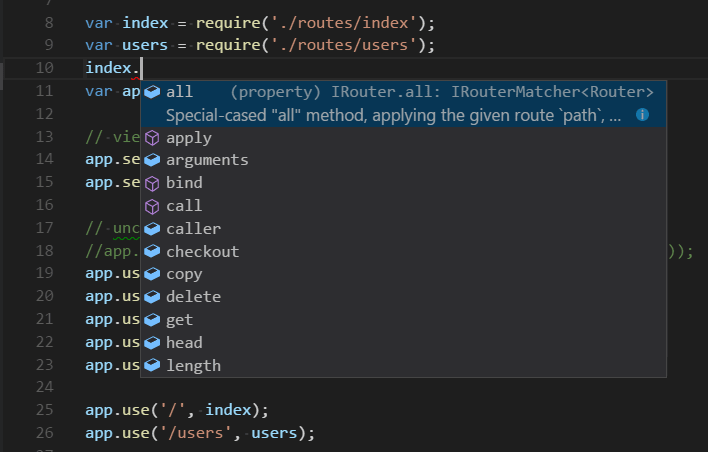
The Node.js and Express documentation does a great job explaining how to build rich applications using the platform and framework. Visual Studio Code will make you more productive in developing these types of applications by providing great code editing and navigation experiences.

Open the file app.js and hover over the Node.js global object \_\_dirname. Notice how VS Code understands that \_\_dirname is a string. Even more interesting, you can get full IntelliSense against the Node.js framework. For example, you can require http and get full IntelliSense against the http class as you type in Visual Studio Code.



VS Code uses TypeScript type declaration (typings) files (for example node.d.ts) to provide metadata to VS Code about the JavaScript based frameworks you are consuming in your application. Type declaration files are written in TypeScript so they can express the data types of parameters and functions, allowing VS Code to provide a rich IntelliSense experience. Thanks to a feature called Automatic Type Acquisition, you do not have to worry about downloading these type declaration files, VS Code will install them automatically for you.

You can also write code that references modules in other files. For example, in app.js we require the ./routes/index module, which exports an Express.Router class. If you bring up IntelliSense on index, you can see the shape of the Router class.



Debug your Express app#

You will need to create a debugger configuration file launch.json for your Express application. Click on the Run icon in the Activity Bar and then the Configure gear icon at the top of the Run view to create a default launch.json file. Select the Node.js environment by ensuring that the type property in configurations is set to "node". When the file is first created, VS Code will look in package.json for a start script and will use that value as the program (which in this case is "${workspaceFolder}\\bin\\www) for the Launch Program configuration.

{

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

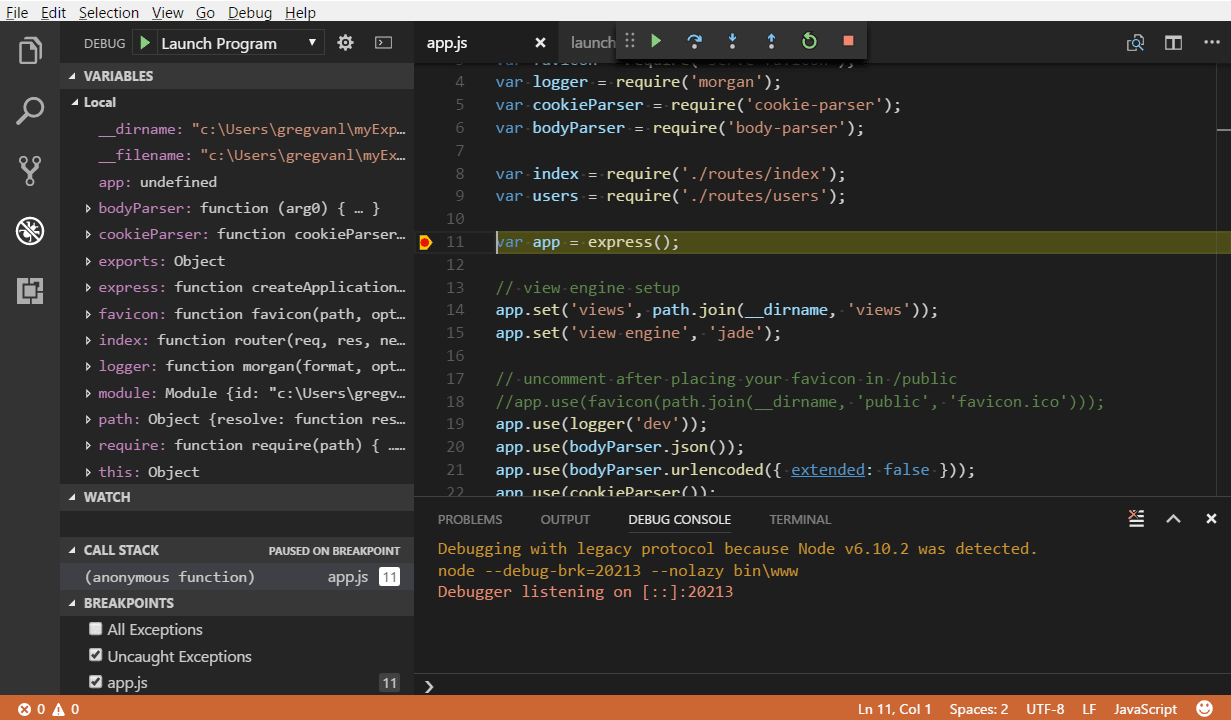
"program": "${workspaceFolder}\\bin\\www"

}

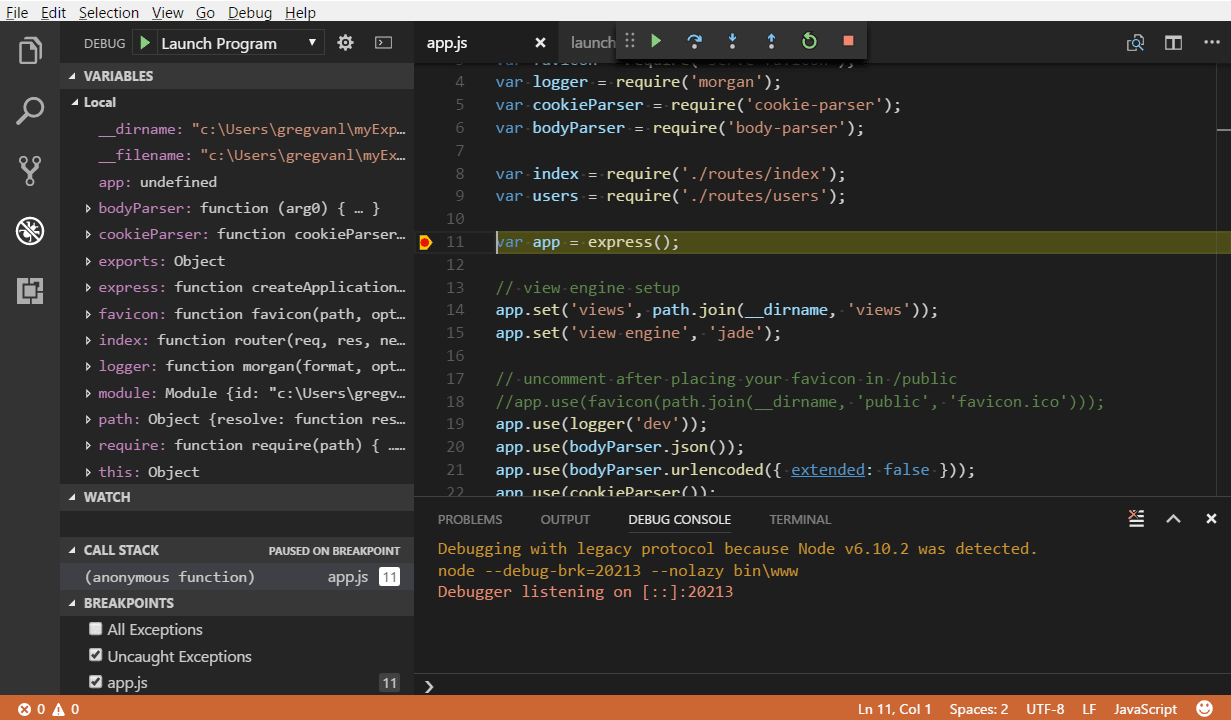
]

}

Save the new file and make sure Launch Program is selected in the configuration dropdown at the top of the Run view. Open app.js and set a breakpoint near the top of the file where the Express app object is created by clicking in the gutter to the left of the line number. Press F5 to start debugging the application. VS Code will start the server in a new terminal and hit the breakpoint we set. From there you can inspect variables, create watches, and step through your code.



**Output/Results snippet:**



**References:**

* + - 1. <https://code.visualstudio.com/docs/nodejs/nodejs-tutorial#_great-code-editing>