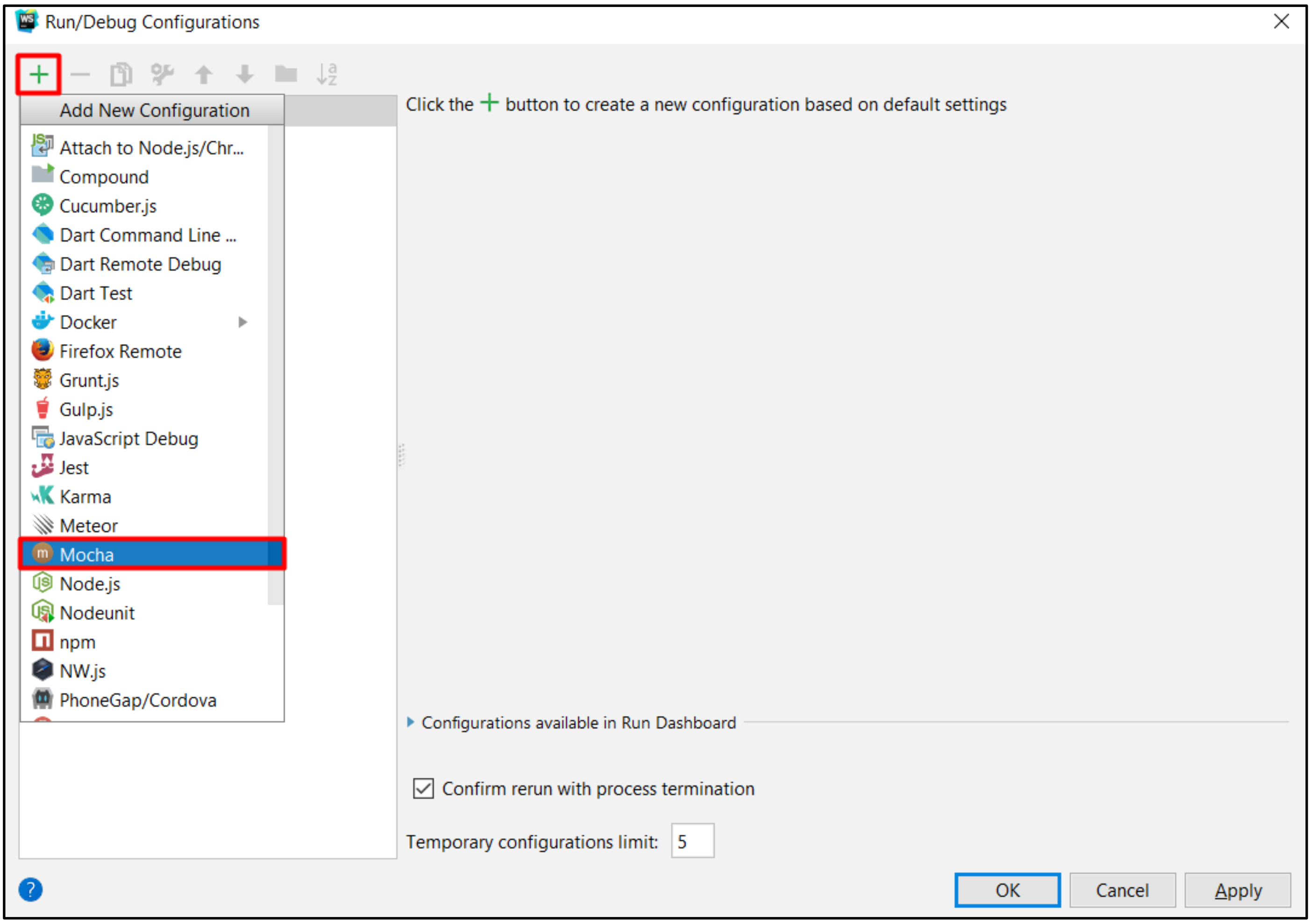
NpmUnit Testing Configuration

WebStorm Configuration

Open the **terminal** in **WebStorm** and type in the following command:

```
>npm -g install mocha
```

From the menu above, select **Run** and **Edit Configurations**. In order to **add new configuration**, click on the green plus and select **Mocha** from the dropdown menu.



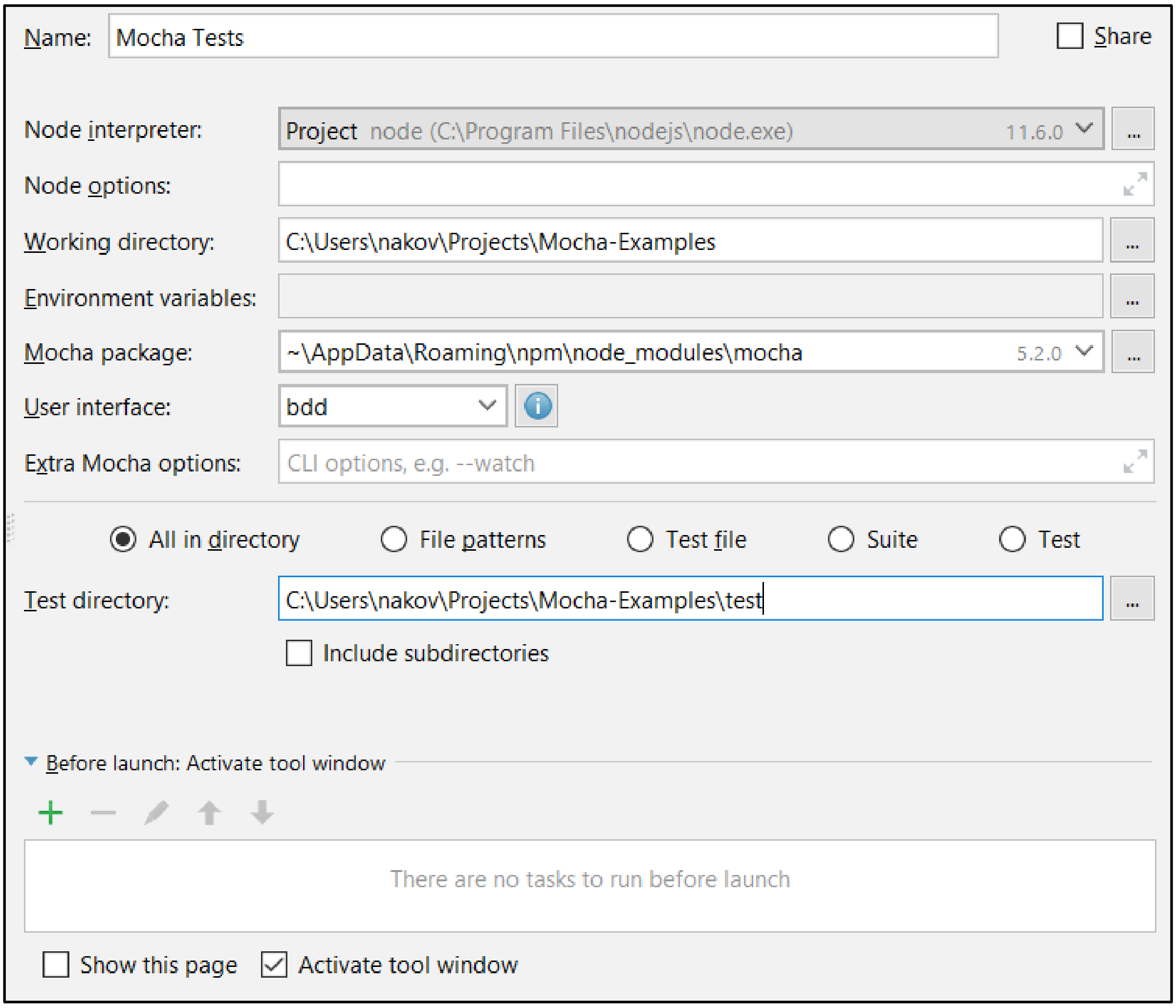
Click the + button to create a new configuration based on default settings

Configurations available in Run Dashboard

☑ Confirm rerun with process termination

Temporary configurations limit: 5

After that, you will have to choose a **name** for your configuration. A **good example** of one would be "**Mocha Tests**". For **Node interpreter**, choose the **directory** in which **node** has been **installed**. The **Working directory** should be your **current** working directory and for **Mocha package** you should select the directory in which Mocha has been installed.



Name: Mocha Tests □ Share

Node interpreter: Project node (C:\Program Files\nodejs\node.exe) 11.6.0

Node options:

Working directory: C:\Users\nakov\Projects\Mocha-Examples

Environment variables:

Mocha package: ~\AppData\Roaming\npm\node_modules\mocha 5.2.0

User interface: bdd

Extra Mocha options: CLI options, e.g. --watch

● All in directory ○ File patterns ○ Test file ○ Suite ○ Test

Test directory: C:\Users\nakov\Projects\Mocha-Examples\test

□ Include subdirectories

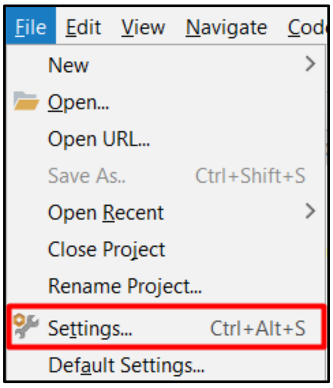
Before launch: Activate tool window

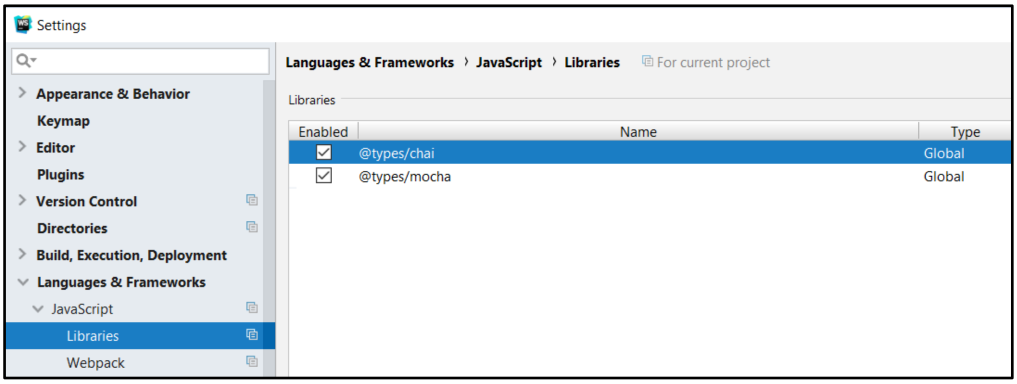
There are no tasks to run before launch

□ Show this page ☑ Activate tool window

Now that you have set up the new configuration, it is time to get the "**auto complete**" and "**syntax checks**" working for Mocha and Chai. To do that, you must add them as **libraries** in WebStorm.

Go to **File** and select **Settings**. Then click on **Languages & Frameworks**, go to the **JavaScript** menu and select **Libraries**.



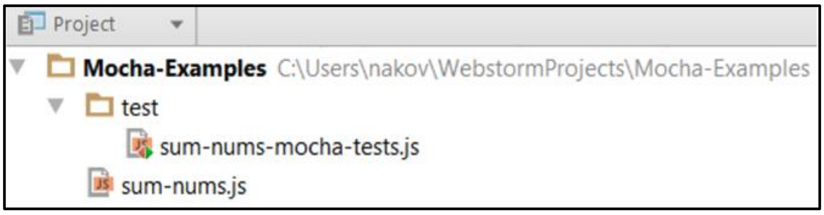


Languages & Frameworks › JavaScript › Libraries For current project

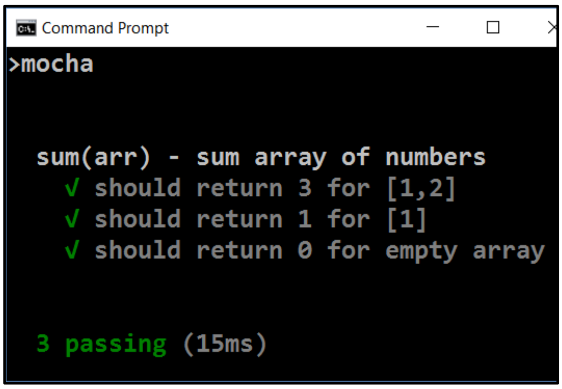
Libraries

Enabled	Name	Type
☑	@types/chai	Global
☑	@types/mocha	Global

In order to run Mocha, you must create a **new folder** (for example, it could be called *test*) in your JS project. After that, you must **put your test code** in the folder you have just created. Provided you have named your folder *test*, your directory should look like this: *test/{test-group-name}.js*. Then run **Mocha** from the **console**.

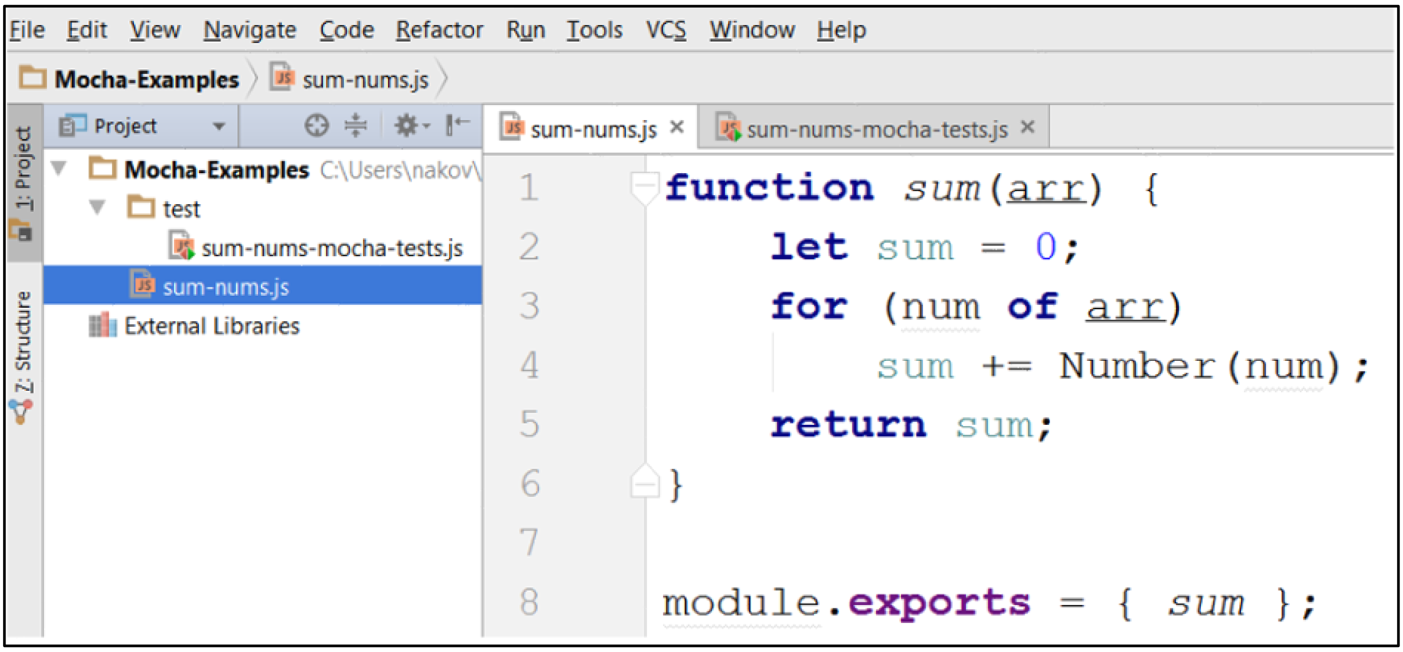


```
Mocha-Examples C:\Users\nakov\WebstormProjects\Mocha-Examples
  test
    sum-nums-mocha-tests.js
  sum-nums.js
```



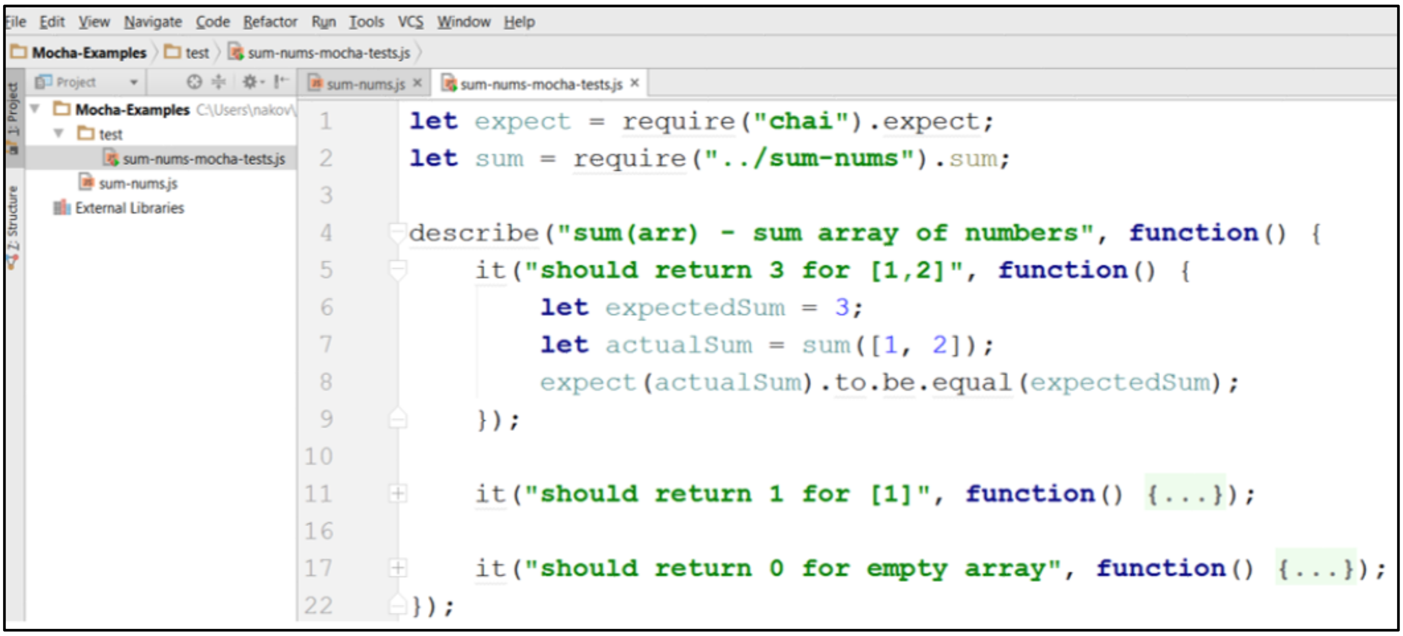
```
>mocha


  sum(arr) - sum array of numbers
    √ should return 3 for [1,2]
    √ should return 1 for [1]
    √ should return 0 for empty array


  3 passing (15ms)
```

Here is what a sample project should look like with a source code to be tested:



```js
function sum(arr) {
    let sum = 0;
    for (num of arr)
        sum += Number(num);
    return sum;
}

module.exports = { sum };
```

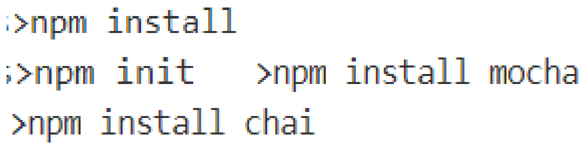
And here is a simple example of how testing of the function above should be done:



```js
let expect = require("chai").expect;
let sum = require("../sum-nums").sum;

describe("sum(arr) - sum array of numbers", function() {
    it("should return 3 for [1,2]", function() {
        let expectedSum = 3;
        let actualSum = sum([1, 2]);
        expect(actualSum).to.be.equal(expectedSum);
    });

    it("should return 1 for [1]", function() {...});

    it("should return 0 for empty array", function() {...});
});
```

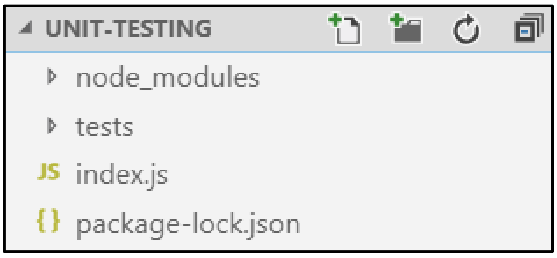
Visual Studio Code Configuration

Open the **terminal** in Visual Studio Code and write the following **commands** to **install Mocha** and **Chai**:

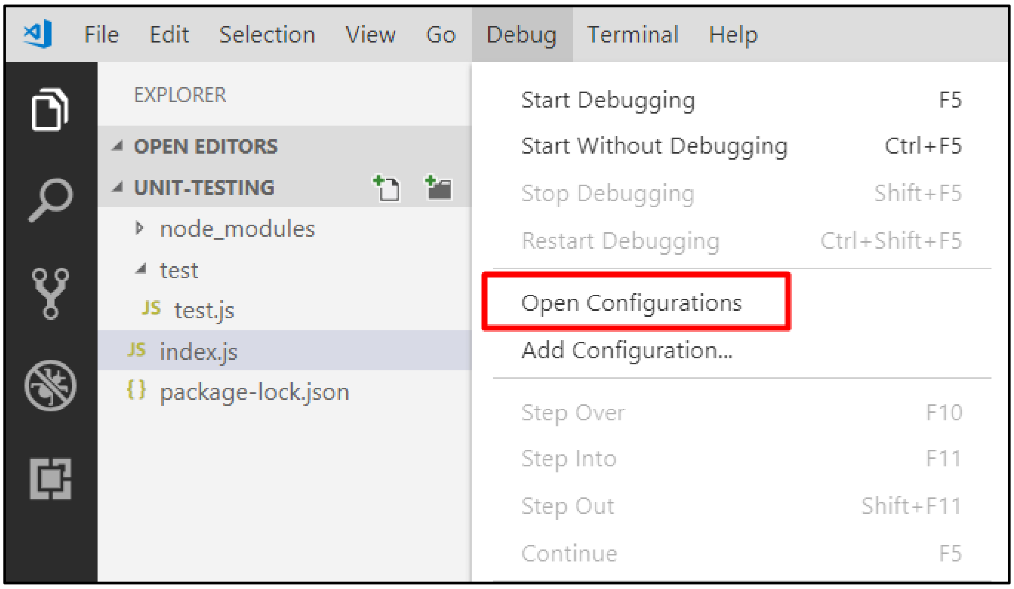


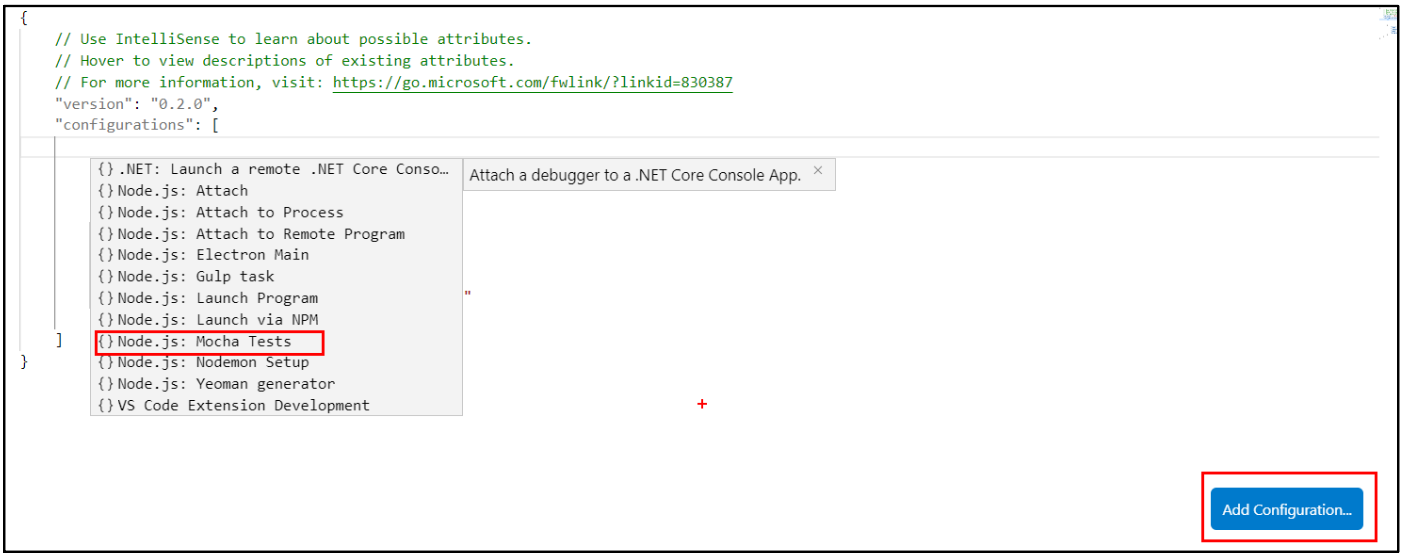
```
>npm install
>npm init    >npm install mocha
>npm install chai
```

The structure of your *Unit Testing* folder should look something like this:



In order to be able to **run unit testing**, you have to do some **configurations**. Start by clicking on **Debug** in the menu above and choose **Open Configurations**.

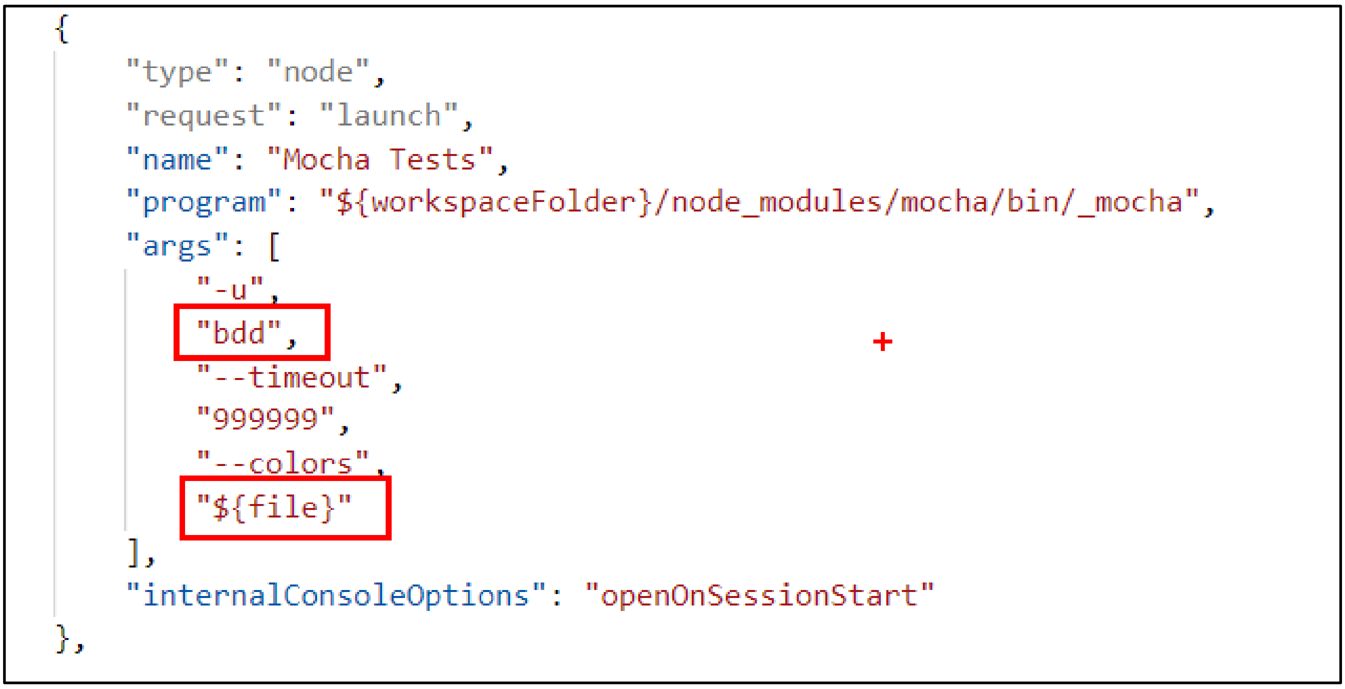




```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
    ]
}
```

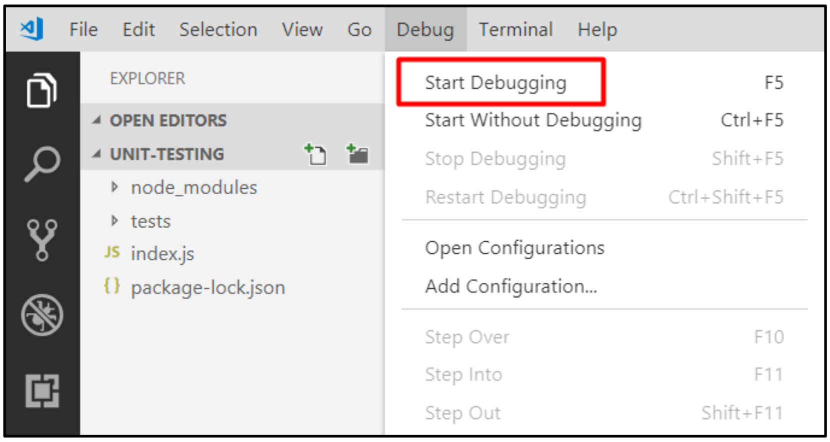
{} .NET: Launch a remote .NET Core Conso...
{} Node.js: Attach
{} Node.js: Attach to Process
{} Node.js: Attach to Remote Program
{} Node.js: Electron Main
{} Node.js: Gulp task
{} Node.js: Launch Program
{} Node.js: Launch via NPM
{} Node.js: Mocha Tests
{} Node.js: Nodemon Setup
{} Node.js: Yeoman generator
{} VS Code Extension Development
Attach a debugger to a .NET Core Console App.
Add Configuration...

This will add a mocha configuration in **launch.json** and you should change the code to match the following:

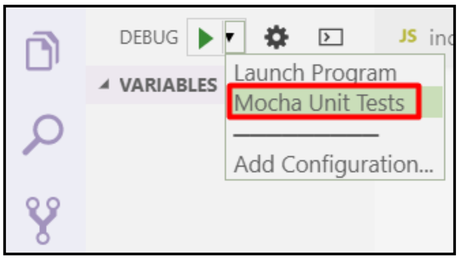


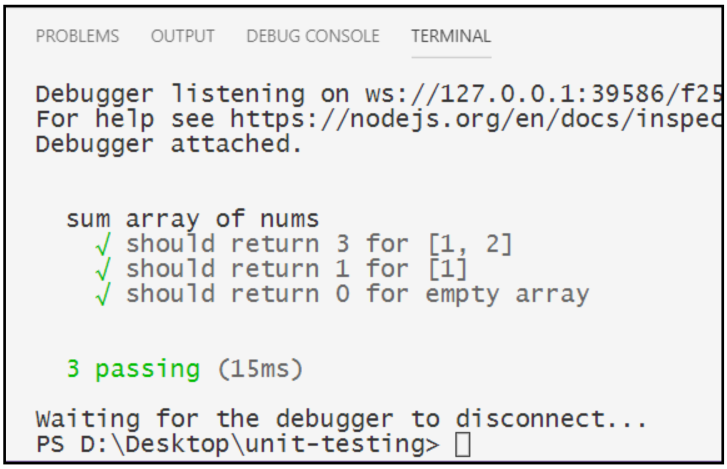
```
{
    "type": "node",
    "request": "launch",
    "name": "Mocha Tests",
    "program": "${workspaceFolder}/node_modules/mocha/bin/_mocha",
    "args": [
        "-u",
        "bdd",
        "--timeout",
        "999999",
        "--colors",
        "${file}"
    ],
    "internalConsoleOptions": "openOnSessionStart"
},
```

Then click on **Debug** and choose **Start Debugging**.



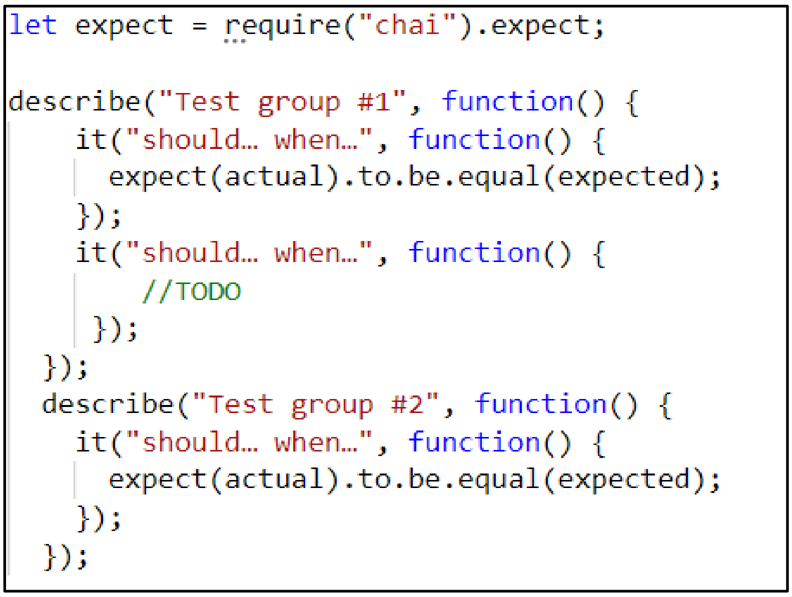
The following window will show up. Select **Mocha Unit Tests** for **configuration**. Now you should be able to make unit tests in VS Code. ☺





```
Debugger listening on ws://127.0.0.1:39586/f25
For help see https://nodejs.org/en/docs/inspec
Debugger attached.


  sum array of nums
    √ should return 3 for [1, 2]
    √ should return 1 for [1]
    √ should return 0 for empty array


  3 passing (15ms)

Waiting for the debugger to disconnect...
PS D:\Desktop\unit-testing> 
```

This is a **sample structure** of what **testing groups** and **classes** should look like. Don't forget to **require chai** in the JavaScript file.



```
let expect = require("chai").expect;

describe("Test group #1", function() {
    it("should… when…", function() {
      expect(actual).to.be.equal(expected);
    });
    it("should… when…", function() {
        //TODO
     });
  });
  describe("Test group #2", function() {
    it("should… when…", function() {
      expect(actual).to.be.equal(expected);
    });
  });
```