# Unit Testing Configuration

Guide for installing and setting up the needed libraries and frameworks to allow local testing of JavaScript code with Visual Studio Code for the ["Back-End Technologies Basics"](https://softuni.bg/trainings/4726/back-end-technologies-basics-september-2024) course @ SoftUni.

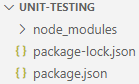
## Visual Studio Code Configuration

It's **generally** **recommended** to install testing libraries **locally** **within each project**. This ensures that each project has the **correct** **version** of the library it needs and that all project **dependencies** are clearly **documented** in the **package.json** file.

Open the **terminal** in Visual Studio Code, navigate to the directory of your project and execute the following commands:

**npm init  
npm install  
npm install chai  
npm install mocha**

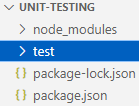
The structure of your **UNIT-TESTING**folder should look something like this:



## Running Our First Mocha Test

In order to be able to **run Mocha tests**, you have to make some **additional steps**.

Start by creating a new folder, named "**test**", in the directory where you ran the commands from above. Your **UNIT TESTING** folder should look like this:



For the purposes of this session, it **doesn't matter** which standard you'd prefer to use – **ESM** or **CommonJS**. Below you will find instructions on how to run your tests in both ways. Remember that the Judge system expects only the testing code, without the **imports**/**exports** / **require()**/**module.exports** code.

### ESM

If you want to use the **ESM** **standard**, follow the steps below.

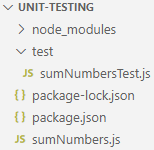
Add a **new** **file** to the **UNIT-TESTING** folder and name it **sumNumbers.js**.

Use the code bellow:

|  |
| --- |
| sumNumbers.js |
| export function sum(arr) {  let sum = 0;  for (let num of arr){  sum += Number(num);  }  return sum;  } |

Now, add a new file to the **test** folder and name it **sumNumbersTest.js**. Use the code below:

|  |
| --- |
| sumNumbersTest.js |
| import { expect } from 'chai';  import { sum } from '../sumNumbers.js';  describe('Sum numbers', () => {      it('sums single number', () => {          expect(sum([1])).to.equal(1);      });      // Test overloading      it('sums multiple numbers', () => {          expect(sum([1,1])).to.equal(2);      });      it('sums different numbers', () => {          expect(sum([2,3,4])).to.equal(9);      });  }); |

Your **UNIT-TESTING** folder should look like this:  


If we want to use the standard '**.js' extension** for the **test** files, when using the **ESM Standard**, we have two options:

* We can rename the test files and give them the '**.mjs**' extension
* We can keep the extension '**.js**', but we have to add **"type": "modules"** in our **package.json** file. This specifies that the '**.js**' files are **ES** **modules**.

In this tutorial, we will choose the second option.

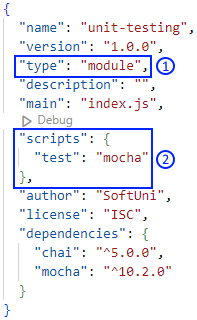
There are two ways to run Mocha tests. The first one is using the command **mocha**, which by default **will look** for **test files** in the **"test" directory** and **run all of them.** If you want to **run specific test files,** use thecommand **mocha path/to/test.js.**

For our example, we will use the **mocha** command. This is the **other** change that we have to make in our **package.json** file.

In order to **use** the **mocha** command, we will add it to the **scripts** section in the **package.json** file and indicate to our Node.js project to run this command when we type **npm test command.**

**NOTE:** The **scripts** section in the **package.json** file defines shortcuts for custom shell commands that are related to the lifecycle of a Node.js project. It allows developers to run tasks such as starting the application, running tests, or custom build processes with simple commands like **npm start** or **npm test**.

Finally, your **package.json** file should look like this:



After we have made our changes, we can simply open a terminal and run the command **npm test**. If you have set up everything correctly, you should get this output:

### CommonJS

If you chose the CommonJS standard, you don't have to change the file extension of the test files and you don't need to add **"type": "module"** in the **package.json** file.

The **sumNumbers.js** should look like the code below:

|  |
| --- |
| sumNumbers.js |
| function sum(arr) {  let sum = 0;  for (let num of arr){  sum += Number(num);  }  return sum;  }  module.exports = { sum }; |

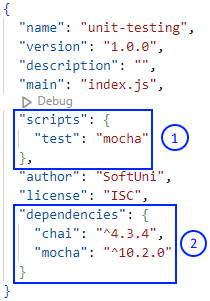
The **sumNumbersTest.js** should look like the code below:

|  |
| --- |
| sumNumbersTest.js |
| const { expect } = require('chai');  const { sum } = require('../sumNumbers');  describe('Sum numbers', () => {      it('sums single number', () => {          expect(sum([1])).to.equal(1);      });      // Test overloading      it('sums multiple numbers', () => {          expect(sum([1,1])).to.equal(2);      });      it('sums different numbers', () => {          expect(sum([2,3,4])).to.equal(9);      });  }); |

We are still going to use the **scripts** section of the **package.json** file to run the test.

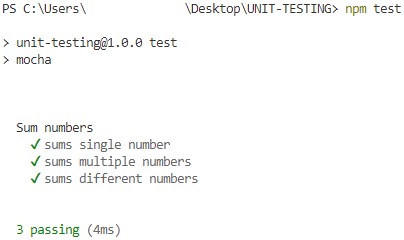
However, the **latest** **versions** of **Chai** are **ES modules**, which can't be loaded, if we want to use the **require()**/**module.exports** syntax. To fix this error, we will have to **downgrade** to a version of Chai, which still **supports** CommonJS.

To downgrade the version of Chai, we will make changes in the **dependencies** section of the **package.json** file. Finally, it should look like this:



After we have made changes to the dependencies section, we have to run the **npm install** command again in order for the changes to the version to take effect.

Our next step is to simply open a terminal and run the command **npm test**. If you have set up everything correctly, you should get this output:

Congratulations! You have run your first Mocha and Chai test! 😊