Lab Manual- 3

Unit Testing

Unit testing is a common practice where developers write test cases together with regular code. Developers develop the products using programming languages such as Java, JavaScript, C# and so on. As JavaScript is familiar to all, we write and test our unit testing using JavaScript. Besides, for every programming languages, there are many testing frameworks. Here, For JavaScript, we use Jest testing framework. Jest is a JavaScript testing framework designed to ensure correctness of any JavaScript codebase. It allows you to write tests with an approachable, familiar and feature-rich API that gives you results quickly. Jest is well-documented, requires little configuration and can be extended to match your requirements.

To understand unit testing, we consider an example of Calculator. Assume, there are two types of calculator such as Basic and Advanced calculator. Basic calculator has the following functionalities-

* Add(a, b): It takes two numbers as input and returns the summation (a+b) of these two numbers.
* Subtract(a, b): It takes two numbers as input and returns the subtraction (a-b) of these two numbers.
* Multiply(a, b): It takes two numbers as input and returns the multiplication value (a\*b) of these two numbers.
* Divide(a, b): It takes two numbers, dividend and divisor as input and returns the quotient (a/b) of these two numbers.

Advanced calculator has the following functionalities-

* Pow(x, n): It takes two numbers as input and returns the powered value (x^n) of these two numbers.
* Modulo(a, b): It takes two numbers as input and returns the modulo value (a%b) of these two numbers.

To test the calculator, at first, we have to implement the Basic and Scientific calculator. As mentioned earlier, we will implement the calculator using JavaScript. Before implanting it, we need to install some libraries.

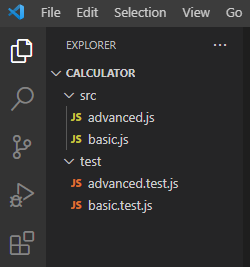
Prerequisite:

* Implementation IDE: For writing code, you can use any IDE. Here, we will use VSCode IDE. Download it from this [link](https://code.visualstudio.com/) and install the .exe.
* Node JS: Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Download it from this [link](https://nodejs.org/en/) and install the .msi. Check the version of Node JS using ***node –v*** command. Also, you can check the version of npm using ***npm –v*** command.
* JEST: Jest is a testing framework. Install it using the following command from the terminal.

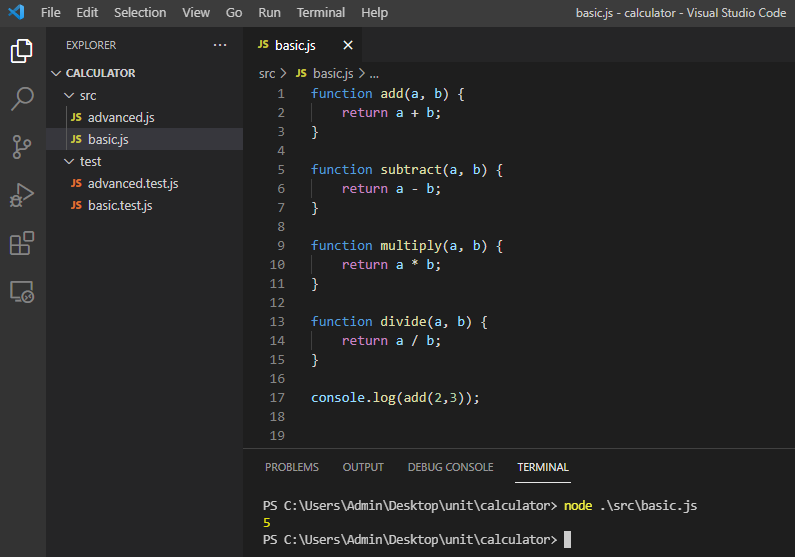
npm install --save-dev jest

Environment setup:

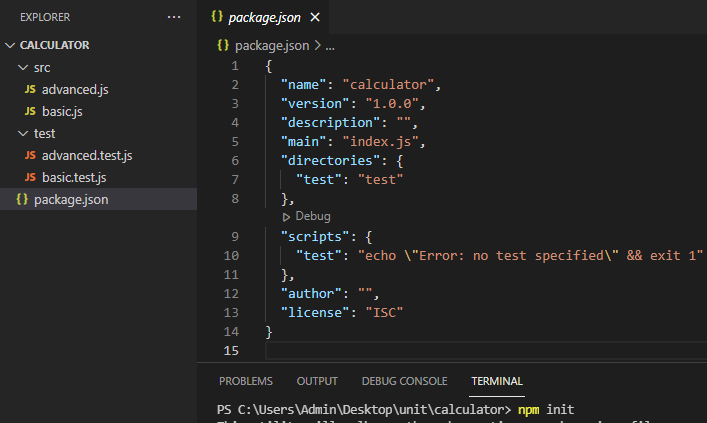
Now, environment is ready to implement calculator. At first, a folder named calculator is created to organize the whole project. Then, inside the calculator folder, two folders are created such as src and test. Now, two .js files are created for implementation the functional requirements inside the src folder. In the same way, two .test.js files are created for testing purpose inside the test folder.



At first, we implement the basic calculator functionalities.



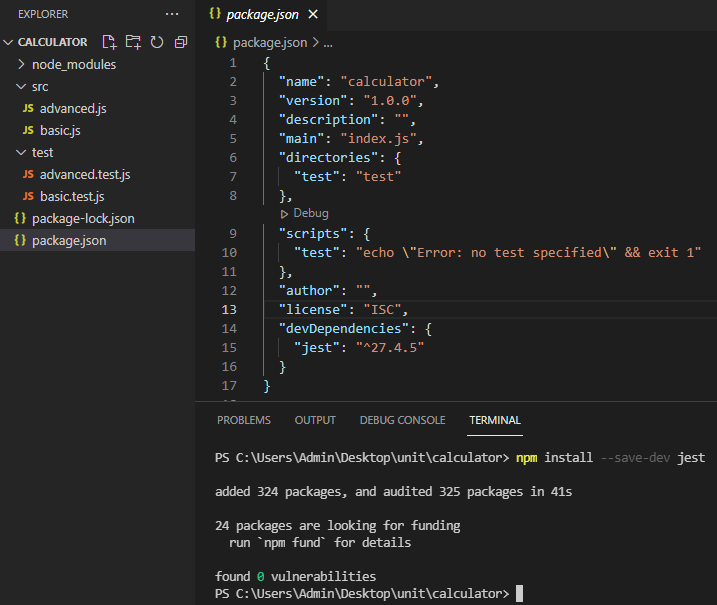
Now, from the calculator folder, node modules are needed to be initiated typing npm init from the command line and click ENTER with default value. After that, a package.json file is created which looks like the following.



Now, we will install the JEST using the following command line.

npm install --save-dev jest

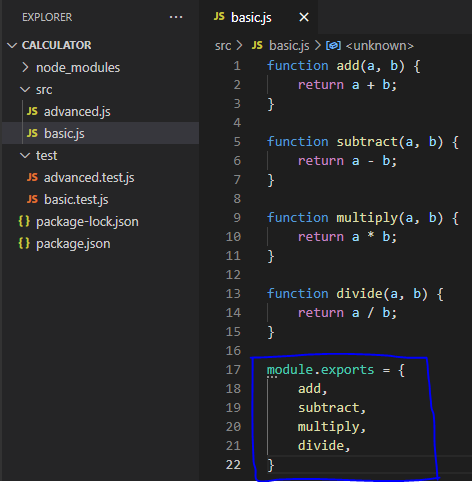
After that, the folder structure looks like the following-



Here, the package.json file is also updated by adding JEST dependency. Also, package-lock.json file is added where all the information about the node-modules folder is recorded.

Test the basic.js file:

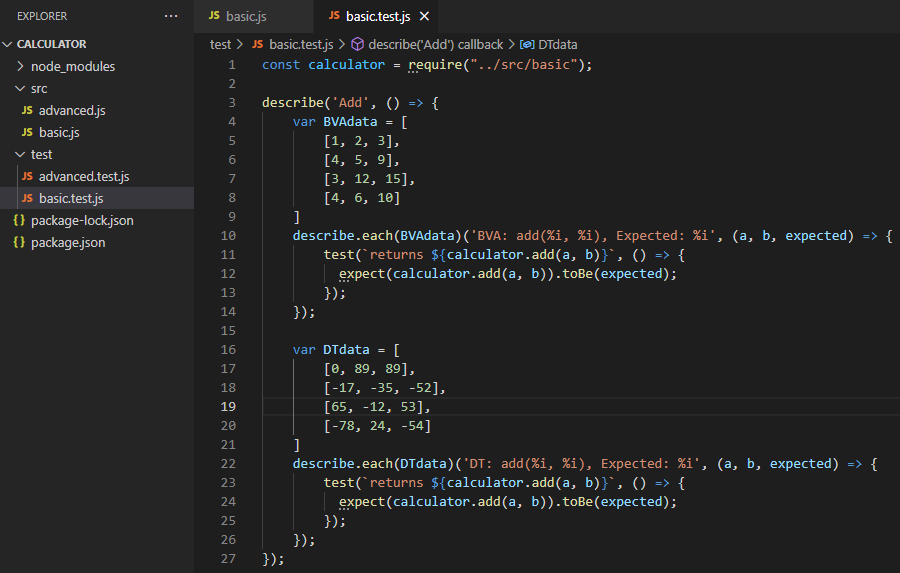
To test the basic calculator, we implement the basic.test.js file. Here, we follow JEST terminology to test individual functions having different types of test cases. Before implementing the test file, first of all, we have to export the methods of basic calculator.



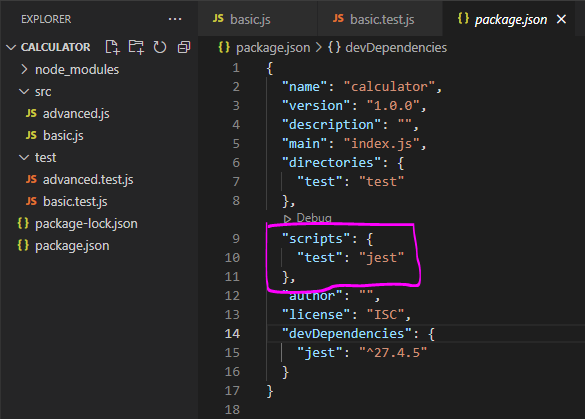
As mentioned in the theory class, there are different types of testing such as boundary value analysis (BVA), decision table (DT) based testing and so on. Every type of testing has several test cases that need to be tested. For example, considering the add function, a sample test cases can be the following-

|  |  |  |  |
| --- | --- | --- | --- |
| Add | | | |
| Method Name | a | b | Expected |
| BVA | 1 | 2 | 3 |
| 4 | 5 | 9 |
| 3 | 12 | 15 |
| 4 | 6 | 10 |
| DT | 0 | 89 | 89 |
| -17 | -35 | -52 |
| 65 | -12 | 53 |
| -78 | 24 | -54 |

Here is the implementation details to test the add function of basic calculator.



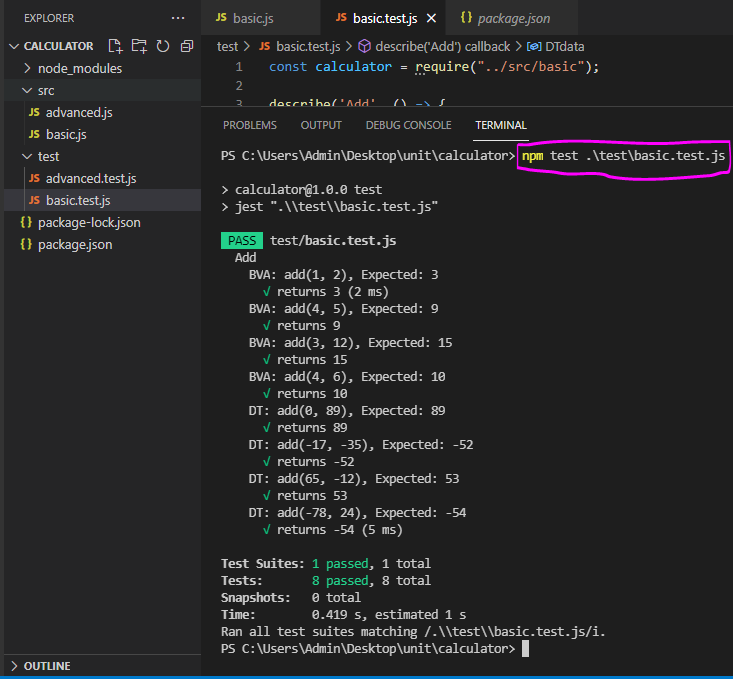
Now, it’s time to test our first add function with two testing methods having several test cases. Before run the test, we have to add the JEST in the package.json file by updating in the scripts member.



As we want to test the only basic.test.js file, the following command is needed to be executed.



The output of the test cases are following-

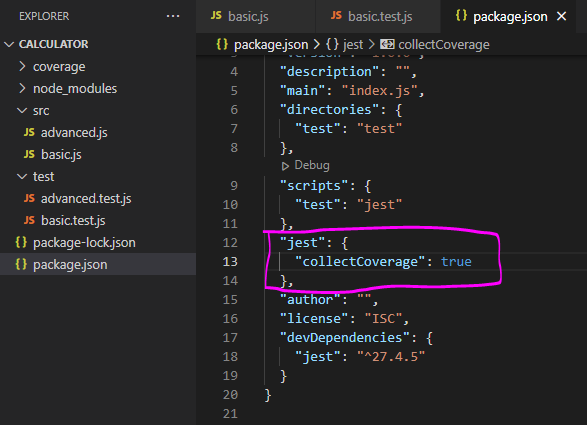


Here, there are 8 test cases and every test cases is passed.

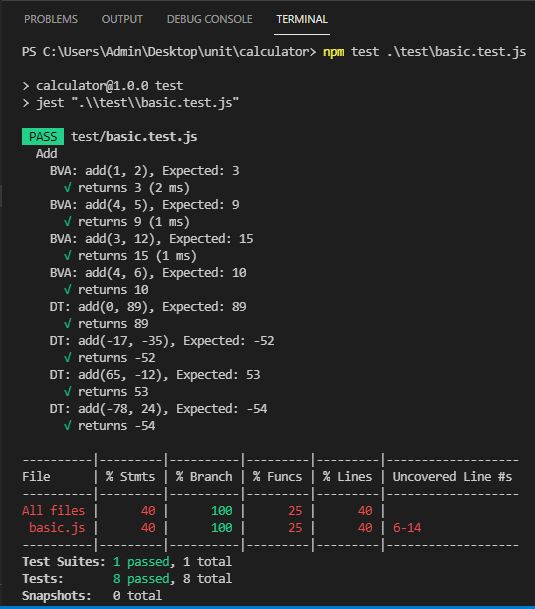
However, if the test cases are huge, it will be difficult for us to comprehend all the results from terminal (console). So, it is needed to generate a report considering all the test cases. Here, we will generate two types of reports-

**Test Coverage Report:**

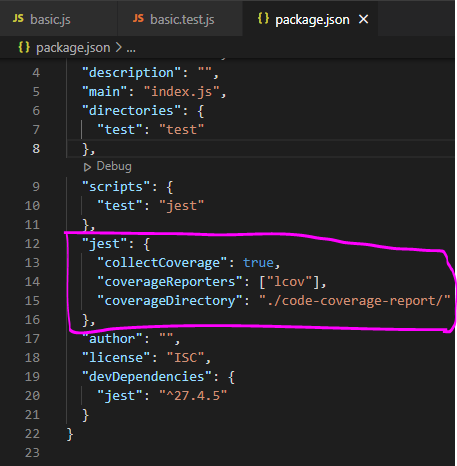
Test coverage report is generated using lcov and text reporters. To do this, we need to add the following code snippet in the package.json file.



After that, run the previous npm test command again.



Here, in addition to the previous output, we have found something interesting stats such as how many statements are covered by executing this test file. However, the previous problem (result is shown in console) is exists. We want to generate a report separately. To do so, we need to add two more lines on the package.json file.



Now, we run the same command (npm test) again. Now, a folder named code-coverage-report is created where all the information related to code coverage are reported in an HTML file (index.html).

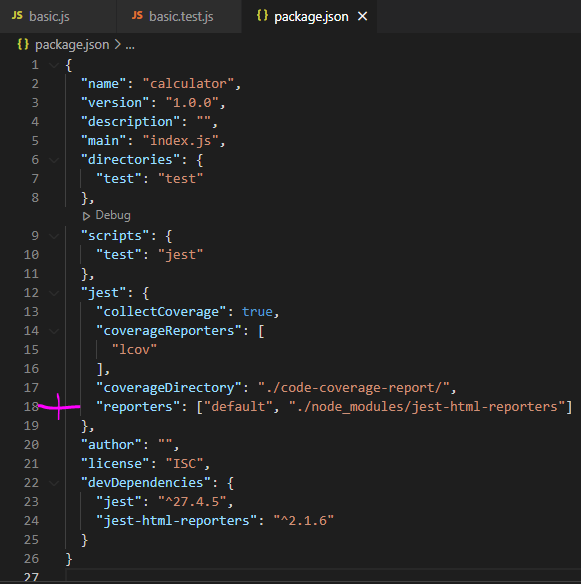
However, we want to generate a report considering the statistics of test cases such as how many test cases are passed or fails. To do this, we need jest-html-reported which is described bellow section.

**HTML Report:**

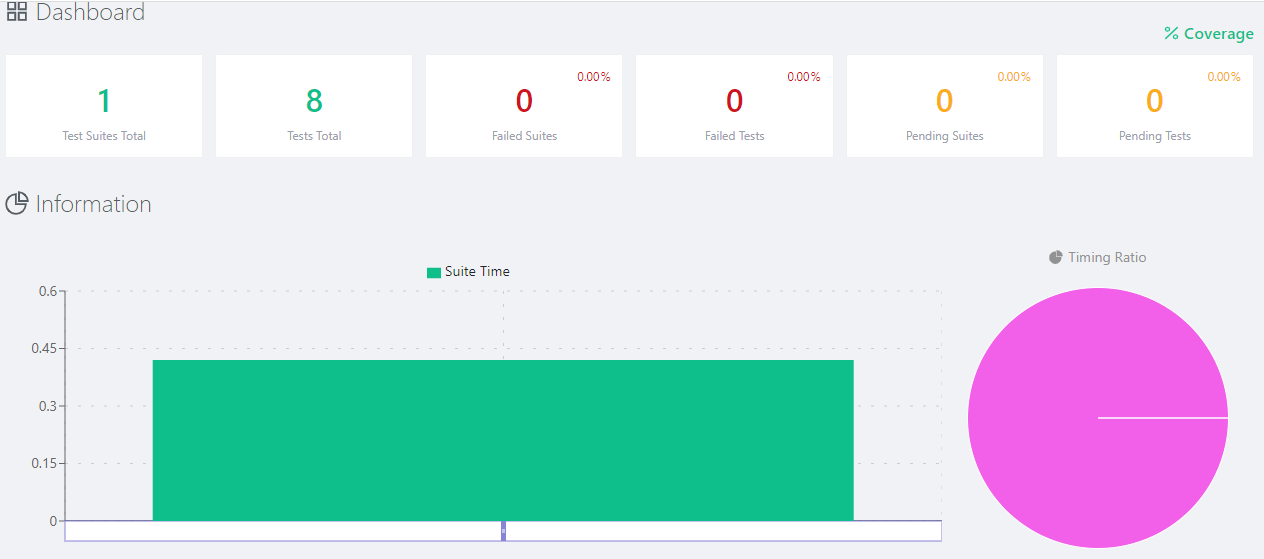
HTML report is generated using jest-html-reporter package. So, at first, we have to install it by the following command.

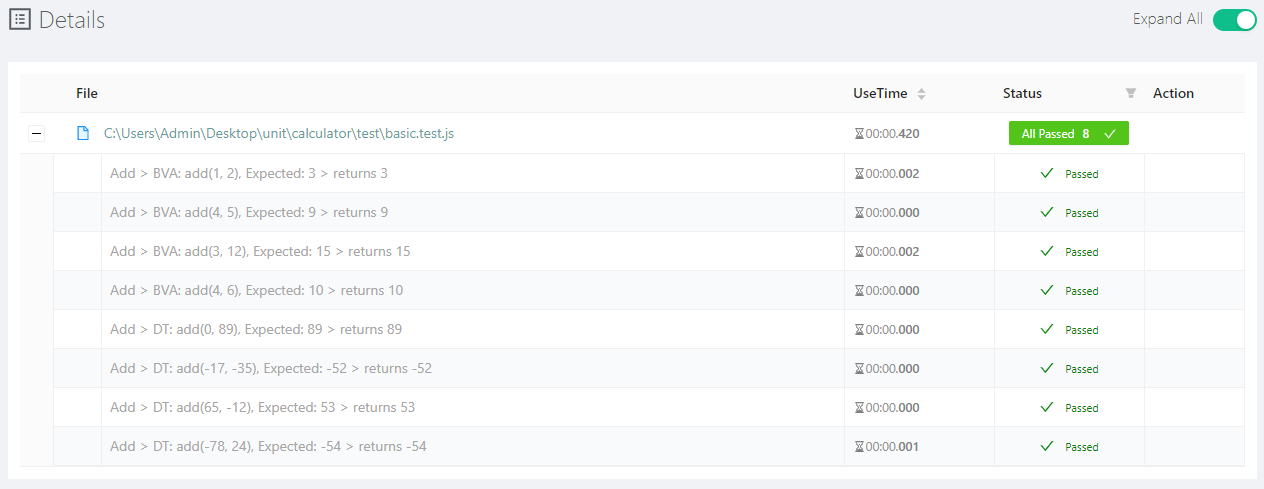
npm install --save-dev jest-html-reporters

Now, we need to add this into package.json file.

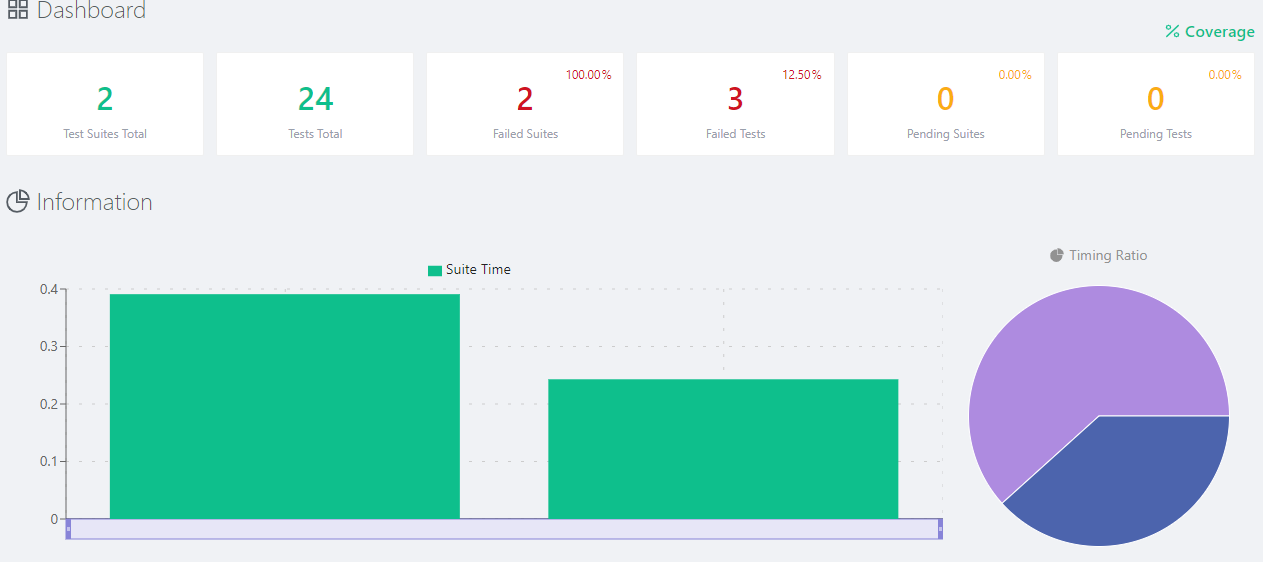


After that, run the previous npm test command again. Now, we will see an HTML file (jest\_html\_reporters.html) where a dashboard will be seen. Here, all the information related to test cases are shown.





That’s it. Now, we can easily add another testing method of basic calculator into the test file. After implementing the advanced calculator and corresponding file, the output will be the following.





By clicking the info button, you will find the error message in detail.