

How To Test A Function JavaScript Developer Tools

TechRounder PDF Edition

Live article: <https://www.techrounder.com/javascript/how-to-test-a-function-javascript-developer-tools/>

By Vipin PG | Published January 16, 2021 | Updated January 23, 2021 | Format: Guide | 5 min read

Quick answer

Welcome again to the world of development! I think no wonder to anyone that function testing in the JavaScript development process exists for the ease of checking large projects for bugs.

Welcome again to the world of development! I think no wonder to anyone that function testing in the JavaScript development process exists for the ease of checking large projects for bugs. Sometimes it happens that after refactoring a component in one place it began to work better, but in another, it just fell off.

Tests will show this without manually clicking all the logic after each code change. Tests simulate user actions automatically, and they also allow you to write high-quality code. In this article which is called 'How To Test A Function JavaScript Developer Tools' you will read a lot about the functions in JavaScript and how you can test the function quickly and easily. Let's start from the beginning.

A JavaScript function is a piece of code that can be described once, and then called for execution in different parts of the program any number of times. This is the classic purpose of a function. In this scenario, we put the code, which we repeat several times on the page, into a function, and then use it in those places in the program where we need to execute it. Another traditional use of functions is when we use them as the main building blocks of an application.

In this case, the task is divided into subtasks, and then we solve each of them by creating separate functions. After that, we usually move on to developing the central code using all the previously developed functions in it. As a result, such a program becomes more structured. It is easier to make various changes and add new features to it. There are several ways to create a function in JavaScript. These are: Function Declaration; Function Expression; Arrow Function. Function operations in JavaScript can be divided into 2 steps:

- Function declaration (creation). Writing a function using a Function Declaration begins by writing the function keyword. After that, the name of the function is indicated, in which parentheses are listed, if necessary, separated by commas, and the function code enclosed in curly braces. A function can have as many parameters as you like or not at all. Parentheses are indicated in any case. If there are several parameters, then they must be separated by a comma. They allow you to more conveniently (by name) get the passed arguments to a function when it is called. A function code (also called a function body) is a set of instructions enclosed in curly braces that must be executed when it is called.
- Calling (executing) this function. The declared function is not executed by itself. In order for a function to run, it must be called. The function is called by specifying its name and two parentheses. Inside the brackets, if necessary, you can pass arguments (additional data) to it, separating them from each other using a comma.

After writing the JavaScript code (function), the responsible developer wants to check its serviceability and operability. Let's move on to the main question of this article - How to check the debugged function - quickly, simply, clearly, and intelligibly? Now we will take a look at such a wonderful testing framework as Jest. As stated on the homepage of this technology: Jest is a delightful JavaScript testing framework with a focus on simplicity.

And it's true, Jest is very simple. It does not require additional settings, is easy to understand and use, and also has pretty good documentation. Great for projects which are using Node, React, Angular, Vue, Babel, TypeScript, and more. It also has an open-source code and is maintained by Facebook. The initial stage of working with Jest is installation. To install Jest into your project, run:

```
npm install -save-dev jest. After installation, you can update the scripts section of your package.json:  
"scripts" : { "test" : "jest" }
```

With such a simple call, you can already run your tests (in fact, Jest will require at least one test to exist). By calling the 'jest -init' command at the root of the project and answering a few questions, you will get the jest.config.js configuration file. Or you can add the configuration directly to your package.json. To do this, add the "jest" key to the JSON root, and in the corresponding object, you can add the settings you need.

We'll discuss the options themselves later. At this stage, this is not necessary, since jest can be used "outright", without additional configurations. The test function is used to create a new test. It takes three arguments. The first one is a line with the name of the test, which Jest will display in the report. The second is a function that contains the logic of your test. You can also use the 3rd argument - timeout.

It is optional and the default is 5 seconds. It is set in milliseconds. This parameter is required when you are working with asynchronous code and returning a promise from the test function. It specifies how long Jest should wait for the promise to resolve. After this time, if the promise was not resolved, Jest will consider the test to be failed. You can also use it() instead of test(). There is no difference between such calls: it() is just an alias for the test() function.

Let's look at the main testing methods and why they are needed, which Jest provides us with:

- toBe() - suitable if we need to compare primitive values or whether the passed value is a reference to the same object that is specified as the expected value. Values are compared using Object.is(). In contrast to ===, this makes it possible to distinguish 0 from -0, check if NaN is equal to NaN.
- toEqual() - suitable if we need to compare the structure of more complex types. It will compare all fields of the passed object with the expected one. It will validate every element of the array and do it recursively throughout the nesting.
- toContain() - check if the array or iterable contains the value. The === operator is used for comparison.
- toContainEqual() - it checks or contains an array element with the expected structure.
- toHaveLength() - it checks whether the object's length property is as expected.
- toBeNull() - it checks for equality with null.
- toBeUndefined() - it checks for equality with undefined.
- toBeCloseTo() - such a method is convenient for floating-point numbers when you don't care about precision and don't want the test to depend on a slight difference in fraction. The second argument can be passed to what decimal place the precision is required for the comparison.

These were some of the basic techniques when working with Jest, which helps to test the JavaScript function. But there are many more of them, and I am sure that you will easily get to know them during the development and testing of the function.

Many of the newbies forget how to check for the existence of a function in JavaScript, so I leave here a little prompt on how to do it:

```
'if (typeof functionName == 'function') { console.log ('Yes! Function exists!'); } else { console.log ('Alas, the function is missing'); }'
```

Conclusion

In the article 'How To Test A Function JavaScript Developer Tools'. Above, I have described some simple tests using Jest's validation functions. I hope you will use the received information with benefit and save your time during the development process. Good luck in testing JavaScript functions!

References

1. tutorialspoint.com - What-are-different-ways-of-defining-functions-in-JavaScript - <https://www.tutorialspoint.com/What-are-different-ways-of-defining-functions-in-JavaScript>