

# Jest

## What is Jest?

Jest is a testing framework for JavaScript that includes both a test-runner and assertion functions in one package.

## Installing Jest

Jest is included by default when initializing a React app using `create-react-app`. However, when manually installing Jest with `npm`, use the provided command to install it as a developer dependency.

```
npm install jest --save-dev
```

## Configuring Jest

To configure Jest to run tests with the `npm test` command, the **package.json** file must have the "test" script defined.

- The provided configuration will run the `jest` command on all files in the `__test__` directory with the extension `.test.js` or `.spec.js`.
- The `--coverage` flag will produce a coverage report in the output and in the generated file `coverage/index.html`.

```
{
  "scripts": {
    "test": "jest __tests__/ --coverage"
  }
}
```

## The test() function

Every Jest test begins with the `test()` function, which accepts two required arguments and one optional argument:

- A string describing the functionality being tested
- A callback function containing the testing logic to execute
- An optional timeout value in milliseconds. The Jest test must wait for this timeout to complete before completing the test

Each `test()` function call will produce a separate line in the testing report. In order for a given test to pass, the test callback must run without throwing errors or failed `expect()` assertions.

The `it()` function is an alias for `test()`.

```
test('test description', () => {
  // testing logic and assertions go
  here...
}, timeout)
```

## Detecting false positives

Jest will automatically pass a test that it perceives to have no `expect()` assertions or errors. As a result, false positives are likely to occur when naively testing code with asynchronous functionality.

To ensure that Jest waits for asynchronous assertions to be made before marking a test as complete, there are two asynchronous patterns that Jest supports, each with its own syntax for testing:

1. Asynchronous callback execution can be tested with the `done()` parameter function.
2. Promise values can be used in tests with the `async/await` keywords.

## Testing async code: callbacks

When testing asynchronous code that uses a callback to deliver a response, the `test()` function argument should accept the `done()` callback function as a parameter. Jest will wait for `done()` to be called before marking a test as complete.

You should execute `done()` immediately after `expect()` assertions have been made within a `try` block and then again within a `catch` block to display any thrown error messages in the output log.

```
test('testing async code: callbacks',
  (done)=>{
    //act
    asyncFunc(input, response => {
      //assertions
      try {
        expect(response).toBeDefined();
        done();
      } catch (error) {
        done(error);
      }
    });
  });
```

## Testing async code: Promises

When testing asynchronous code that returns a Promise, you must `await` the Promise and the callback passed to `test()` function must be marked as `async`.

```
test('testing promises', async () => {
  //arrange
  const expectedValue = 'data';
  //act
  const actualValue = await
  asyncFunc(input);
  //assertions

  expect(actualValue).toBe(expectedValue);
});
```

## Mocking functions with `jest.fn()`

The Jest library provides the `jest.fn()` function for creating a “mock” function.

- An optional implementation function may be passed to `jest.fn()` to define the mock function’s behavior and return value.
- The mock function’s behavior may be further specified using various methods provided to the mock function such as `.mockReturnValueOnce()`.
- The mock function’s usage (how it was called, what it returned, etc...) may be validated using the [expect\(\) API](#).

```
const mockFunction = jest.fn(() => {
  return 'hello';
});

expect(mockFunction()).toBe('hello');

mockFunction.mockReturnValueOnce('goodbye');

expect(mockFunction()).toBe('goodbye');
expect(mockFunction()).toBe('hello');
expect(mockFunction).toHaveBeenCalledTimes(3);
```

## Mocking modules with `jest.mock()`

When mocking entire modules, mock implementations of the module should be created in a `__mocks__` folder adjacent to the file being mocked.

In the test files, the `jest.mock()` method may be used. It accepts a path to the file where the module to be mocked is defined and replaces the actual module with the version defined in the `__mocks__` folder.

The file to be mocked must be imported before it can be mocked with `jest.mock()`.

```
// ../utils/utilities.js
export const someUtil = () => 'hello';

// ../utils/__mocks__/utilities.js
export const someUtil = jest.fn(() =>
'goodbye');

// myTest.test.js
import { someUtil } from
'../utils/utilities';
jest.mock('../utils/utilities');

test('using a mock function', () => {
  expect(someUtil()).toBe('goodbye');
});
```