

Testing with Jest



Goals

Download Demo Code

- Write unit tests using Jest

An Introduction to Jest

Jest

- Jest is an open-source testing platform written by Facebook
- It's built on top of Jasmine!
- Easy to test in environments that aren't browser-based
- Also very popular for testing React apps
- More information: [Jestjs.io](https://jestjs.io)

Installing Jest

```
$ npm i --global jest
```

This installs jest globally so you can use it anywhere

Organizing Tests

- Test files should be named **NAME_OF_FILE.test.js**
 - You can place in the same directory as the JS file it tests
 - Or, you can organize all tests into a folder called **__tests__**
- If you have a **package.json**, you don't need additional configuration.
 - If not, create **jest.config.js** file. It can be empty, you just need one.
- Run all tests using the command **jest**
 - You can run an individual test using **jest NAME_OF_FILE**

Test

Write tests inside of **test** function callbacks:

```
demo/add.js

function add(x, y) {
  return x + y;
}

module.exports = {add};
```

```
demo/add.test.js

const { add } = require("./add");

test('add should return sum', function () {
  let sum = add(2, 3);
  expect(sum).toEqual(5);
});
```

Describe

To group together related tests, wrap these in **describe** callback:

```
demo/add.js

function add(x, y) {
  return x + y;
}

module.exports = {add};
```

```
demo/add.test.js

describe("add function", function () {

  test('return sum', function () {
    let sum = add(2, 3);
    expect(sum).toEqual(5);
  });

  test('return sum with neg numbers', function () {
    let sum = add(-2, 3);
    expect(sum).toEqual(1);
  });

});
```

Expectations

```
expect(1 + 1).toEqual(2);
```

Expectations should go inside of **test** function callbacks

A function can have several expectations — but be thoughtful about keeping tests small and simple.

Matchers

- .toEqual(obj)**
Has the same value (eg, different objects with same values match)
- .toBe(obj)**
Is the same object (eg, different objects with same values do not)
- .toContain(sought)**
Does object/array contain this item?
- .not.**
Add before matcher to invert (eg `expect("hi").not.toEqual("bye")`)

<https://jestjs.io/docs/en/using-matchers>

Matchers in action

Let's compare **toBe** and **toEqual**

```
matchers.test.js

describe("matchers", function(){
  test("toBe and toEqual are different", function () {
    let nums = [1,2,3];
    let newNums = nums.slice();

    expect(nums).not.toBe(newNums); // not the same reference!
    expect(nums).toEqual(newNums);  // same values so we use toEqual
  });
});
```

Expecting Anything

Sometimes, you're not sure what part of an object will be.

Use **expect.any(type)** and it will match any of that type.

```
demo/any.js

const TOYS = ["doll", "top", "iPad"];

function getRandomToy() {
  let idx = Math.floor(
    Math.random() * TOYS.length + 1);
  return {
    toy: {
      name: TOYS[idx],
      price: 34.99
    }
  };
}

module.exports = {getRandomToy};
```

```
demo/any.test.js

const { getRandomToy } = require("./any");

test("random toy", function () {
  let toy = getRandomToy();
  expect(toy).toEqual({
    toy: {
      name: expect.any(String),
      price: 34.99
    }
  });
});
```

Before / After

Demo: Shopping Cart Totals

We have a shopping cart system with function to test:

getCartTotal(cart, discount=0)

```
demo/cart.test.js

describe("getCartTotal", function () {
  test("get total w/o discount", function () {
    const cart = [
      { item: "le croix", price: 4.99,
        qty: 3 },

      { item: "pretzels", price: 8.99,
        qty: 10 },
    ];

    const total = getCartTotal(cart);
    expect(total).toBe(104.87);
  });

  test("gets total w/discount", function () {
    const cart = [
      { item: "le croix", price: 4.99,
        qty: 3 },
      { item: "pretzels", price: 8.99,
        qty: 10 },
    ];

    const total = getCartTotal(cart, 0.5);
    expect(total).toBe(52.44);
  });
});
```

We can make data in tests:

```
demo/cart.test.js

describe("getCartTotal", function () {
  test("get total w/o discount", function () {
    const cart = [
      { item: "le croix", price: 4.99,
        qty: 3 },

      { item: "pretzels", price: 8.99,
        qty: 10 },
    ];

    const total = getCartTotal(cart);
    expect(total).toBe(104.87);
  });

  test("gets total w/discount", function () {
    const cart = [
      { item: "le croix", price: 4.99,
        qty: 3 },
      { item: "pretzels", price: 8.99,
        qty: 10 },
    ];

    const total = getCartTotal(cart, 0.5);
    expect(total).toBe(52.44);
  });
});
```

Can factor out common setup:

```
demo/cart.test.js

describe("getCartTotal", function () {
  // will hold the cart for the tests
  let cart;

  beforeEach(function () {
    cart = [
      { item: "le croix",
        price: 4.99, qty: 3 },
      { item: "pretzels",
        price: 8.99, qty: 10 }
    ];
  });

  test("gets total w/o discount", function () {
    const total = getCartTotal(cart);
    expect(total).toBe(104.87);
  });

  test("gets total w/discount", function () {
    const total = getCartTotal(cart, 0.5);
    expect(total).toBe(52.44);
  });
});
```

Before / After

Jest gives us *hooks* we can tap into so to not repeat common setup/teardown:

- For one-time setup/teardown:
 - **beforeAll**: run before all tests start
 - **afterAll**: run after all tests finish
- For frequent setup/teardown:
 - **beforeEach**: run before each test starts
 - **afterEach**: run before teach test finish

```
describe("my set of tests", function () {
  beforeAll(function(){
    console.log("Run before all tests")
  })

  beforeEach(function(){
    console.log("Run before each it")
  })

  afterEach(function(){
    console.log("Run after each it")
  })

  afterAll(function(){
    console.log("Run after all tests")
  })
});
```

Can put these directly in file (outside of functions) to run before/after any/all tests in entire file.