Google | 1000 طويق

Jest

أكاديمية طويق
TUWAIQ ACADEMY

وزارة الاتصالات وتقنية المعلومات
MINISTRY OF COMMUNICATIONS AND INFORMATION TECHNOLOGY

الاتحاد السعودي للأمن السيبراني والبرمجة والدرونز
SAUDI FEDERATION FOR CYBERSECURITY, PROGRAMMING & DRONES

مهارات المستقبل

stc

Objective

- Learn to write tests for your React application using Jest.

Type of Tests

- Manual testing
- Documented manual testing
- End-to-end testing
- Unit tests
- Integration testing
- Snapshot testing

Why we need Testing?

- Quality Of Code & Design
- Find Issues & Bugs Early
- Functionality Documentation

To improve quality:

- Performance testing
- Usability testing
- Security testing

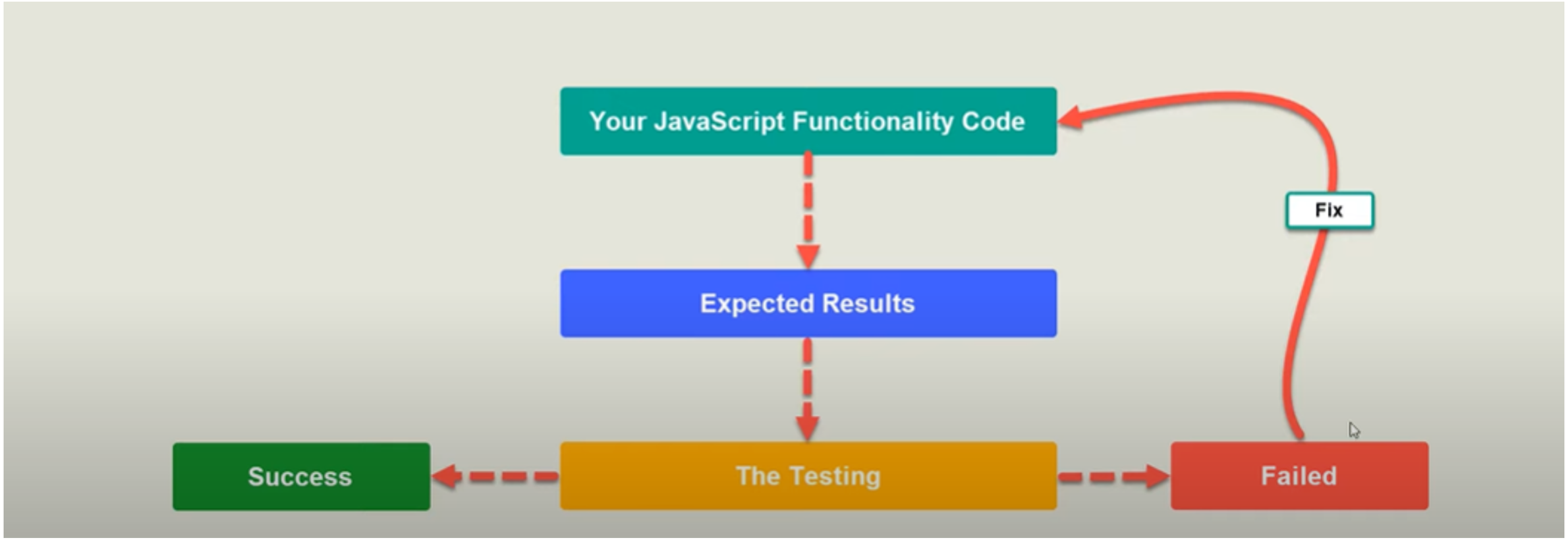
Testing Frameworks for React

- The official React documentation lists a number of tools that can be used for testing React apps
- Jest is automatically installed when you create a React app using the Create React App starter kit that we have been using throughout this level of the Bootcamp.
- We will use Jest to test React.

What Is Unit Testing ?

- Software Testing Method That Break Your Software To Pieces
- Tests Written And Run By The Software Developers .
- It Test a Section (Unit) In Your App To Ensure It Work And Behave Like Expected
- Unit Test Can Be For One Method
- Unit Test Can Be For The Whole Class in OOP
- Unit Test Can Be The Whole Module
- Unit Test Can Be Done For The Whole Procedure
- A Unit is The Smallest Testable Part of Any Software

Testing cycle



Testing Rules

1. Get The Function To Test.
2. Give Input To The Function.
3. Define What Is The Output
4. Check For The Output

TDD => Test Driven Development

1. Think About What Your Code Will Do
2. Write The Code
3. Test The Code

Jest Unit Testing

- The Jest test() module and expect() object can be used to write any number of unit tests.
- Below we create a test that tests a function that adds two numbers together.

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

- Step 1: In the test directory, create a file called ‘sum.test.js’
- Step 2: Add the code shown to ‘sum.test.js’ and save the file.
- Step 3: Run the test. From the command line interface, type **npm test** to run the test.

Using Matchers

- Jest uses "matchers" to let you test values in different ways
- The simplest way to test a value is with exact equality.
- In this code, .toBe(4) is the matcher. When Jest runs, it tracks all the failing matchers so that it can print out nice error messages for you.

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4);
});
```

Common Matchers

- If you want to check the value of an object, use toEqual instead:

```
test('object assignment', () => {
const data = {one: 1};
data['two'] = 2;
expect(data).toEqual({one: 1, two: 2});
});
```

Truthiness

- **`toBeNull`** matches only null
- **`toBeUndefined`** matches only undefined
- **`toBeDefined`** is the opposite of toBeUndefined
- **`toBeTruthy`** matches anything that an if statement treats as true
- **`toBeFalsy`** matches anything that an if statement treats as false

Numbers

- Most ways of comparing numbers have matcher equivalents.

```
test('two plus two', () => {
const value = 2 + 2;
expect(value).toBeGreaterThan(3);
expect(value).toBeGreaterThanOrEqual(3.5);
expect(value).toBeLessThan(5);
expect(value).toBeLessThanOrEqual(4.5);
// toBe and toEqual are equivalent for numbers
expect(value).toBe(4);
expect(value).toEqual(4);
});
```

Resources

- Elzero Web School
- https://github.com/sapegin/jest-cheat-sheet