Unit Testing

Using Jest

Approach to Testing

Testing Framework

- Test Runner
  - Allows developers to describe and organize test cases
  - Run test cases, collect results
  - Analysis and Test Coverage
- Matchers
  - Assertions to verify result of calculation, result of function calls,
  - Utility predicates for writing boolean expressions
- Test Doubles (“Stunt” in movie making)
  - Developer-managed **replacement** for external dependencies
  - Function spies: to monitor function calls
  - Mocks, Stubs: to substitute a real function with a “fake” one
  - Fakes: fake timer, fake canvas, fake network, …

Scaffold for wing, Sensors, computers….

Electronic transducers, Analog/Digital converters

Cockpit mockup, Pilot dashboard mockup, warning lights, etc..

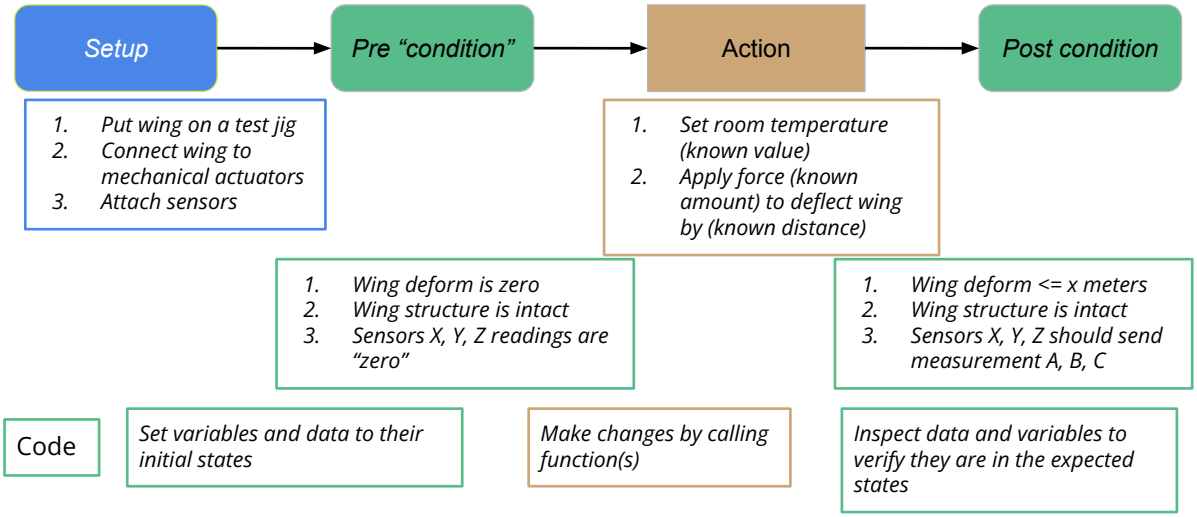
JS Test Libraries

	Test Runner	Assertions / Matchers	Test Doubles	Coverage
Chai		✅		
Istanbul				✅
Jasmine	✅	✅	✅	
Jest	✅	✅	✅	✅
Karma (browser)	✅			
Mocha	✅			
Sinon			✅	

Which One?

How to write tests? (in Jest)

General Approach



Jest Scoping

```
describe("Description of your test suite", function() {
  test("Description of your test case", function() {
    // function body of test case
  });

});
```

```
describe("Description of your test suite", () => {
  test("Description of your test case", () => {
    // function body of test case
  });

});
```

You can use either

`test(____, ___)`

or

`it(____, ___)`

Jest "Hello World"

```
// in tests/hello.spec.js
describe("Hello Suite", function() {

  test("says hello", function() {
    expect ("Hello World").toContain("or");
  });

});
```

JUnit vs. Jest

```
// in TestSuiteSample.java
public class TestSuiteSample {
  private Calculator calc;

  @Before
  public void setup() {
    calc = new Calculator();
  }

  @Test public void addTwoIntegers() {
    assertEquals (14, calc.add(6, 8));
  }

  @Test public void addTwoFloats() {
    assertEquals (14.0, calc.add(6.0, 8.0), 1E-3);
  }

  @After public void cleanup() {
    // runs after each test case
  }
}
```

```
// In calctest.spec.js
describe("Sample Test Suite", () => {
  var calc;

  beforeEach(() => {
    calc = new Calculator();
  }

  test("adds two integers", () => {
    expect (calc.add(6, 8)).toEqual(14);
  }

  test("adds two floats", () => {
    expect(calc.add(6.0, 8.0))
      .toBeCloseTo(14.0, 3);
  }

  afterEach( () => {
    // runs after each test case
  }
});
```


Jest Matchers

```
expect(___).to____();
expect(___).toBe____();
expect(___).toHave____();
```

Complete Ref: https://jestjs.io/docs/en/expect

Jest Matchers

- Boolean Matchers
  - toBe(true), toBe(false). toBeTruthy(), toBeFalsy()
- Numeric Matchers
  - toBe(value), toEqual(value), toBeNaN(), toBeCloseTo(number, numDigits), toBeGreaterThan(value), toBeGreaterThanOrEqual(value), toBeLessThan(value), ...
- String Matchers
  - toContain(item), toMatch(regex), ...
- Object Matchers
  - toBeDefined(), toBeUndefined(), toBeNull(), toContain(), toHaveProperty(), ...
- Array Matchers
  - toHaveLength(value), toContain(value), ...

Matchers vs. Spies

- Compare **actual** values against expected values
- Wide range of data types to compare
  - Number, string, boolean
  - Arrays, Objects (equality vs identity)
  - DOM elements
  - Functions
  - ...
- These **actual** values are usually output of a **function**

- Instead of evaluating function output, spies **monitor function activities**
  - Are functions invoked at all?
  - Are they invoked with the right args?
  - Do they trigger exceptions?
  - How many times they are invoked?
  - What are the input arguments on the most recent call?
- Peek into function call graph

Sample Tests?


```
describe("User login by email/password", () => {
  beforeEach(() => {
    // Reset the userid/password to "none", and reset other variables
    // Setup a fake authentication server (fake timer)
  });
  it("shows the user/password input field, () => {
    // Test code #1 here
    expect(____).toBeVisible();
  });
  it("enables the login button after user enters id & password", () => {
    // Test code #2 here
  });
  it("disables login button when id & password are empty", () => {
    // Test code #3 here, connect to fake auth server here
  });
  it("authenticates user when id & password are correct", () => {
    // Test code #4 here, connect to fake auth server here
  });
  it("shows error when user cannot be authenticated", () => {
    // Test code #5 here, connect to fake auth server here
    expect(message).toContain(".....");
  });
})
```

Jest Mock Functions (Spies)

Jest: Function Spies / Mock Functions

Mock without implementation	`const spy1 = jest.fn();`
	`myObj.verifyLogin = spy1;`
	`// spy on all calls to myObj.verifyLogin`

Mock with implementation	`const myStub = function(usr,pwd) {`
	`  // write your code here`
	`}`
	`const spy2 = jest.fn(myStub)`
	`myObj.verifyLogin = spy2;`
	`// all calls to verifyLogin will be replaced with myStub`

Spy On Methods of a Global Object

```
// In MyComponent.vue
<template>
</template>

<script>
export default {
  mounted() {
    firebase.initializeApp(____);
  },
}
</script>
```

```
// mycomponent.test.js
import {shallowMount} from '@vue/test-utils';
import MyComponent from '@/MyComponent.vue';
describe("Sample", () => {

  it("correctly initialized a DB instance", () => {
    const initSpy = jest.fn();
    const firebase = {
      initializeApp: initSpy
    }

    const w = shallowMount(MyComponent);
    expect(initSpy).toBeCalled();
  });
});
```


Spy On Global Methods

```
// In MyComponent.vue
<template>
</template>

<script>
export default {
  getWeather() {
    fetch("https://darksky.net?zip=12345")
      .then(____)
  },
}
</script>
```

```
// mycomponent.test.js
import {shallowMount} from '@vue/test-utils';
import MyComponent from '@/MyComponent.vue';

describe("Sample", () => {

  it("correctly initialized a DB instance", () => {
    global.fetch = jest.fn();
    const w = shallowMount(MyComponent);
    // more code here

    expect(global.fetch).toBeCalled();
  });
});
```

Jest Spies: Assertions and Matchers

Actual Function Call	Spy Assertion

`funOne("Carbon", 12);`	`const funOne = jest.fn();`
	`// ...`
	`expect(funOne).toBeCalledWith("Carbon", 12);`

`funOne("Carbon", 12);`	`const funOne = jest.fn();`
	`// ...`
	`expect(funOne.mock.calls[0][0]).toContain("Carbon");`
	`expect(funOne.mock.calls[0][1]).toBe(12);`

`funTwo({`
`  atom: "Carbon",`
`  weight: 12}`
`);`
	`const funcTwo = jest.fn();`
	`// …`
	`expect(funTwo).toBeCalledWith(expect.objectContaining({atom: "Carbon"});`
	`expect(funTwo).toBeCalledWith(expect.objectContaining({weight:12});`

Testing Asynchronous Code

VueJS + Jest

VueJS Test Guidelines

Additional config in package.json

```
// package.json
“dependencies”: {
  “core-js”, “versionNum”
},
“devDevendencies”: {
  “@vue/cli-plugin-babel”: “versionNum”,
  “@vue/cli-plugin-unit-jest”: “versionNum”,
  “@vue/test-utils”: “versionNum”,
  “babel-core”, “versionNum”,
  “babel-jest”, “versionNum”
}
```

```
// package.json
“jest”: {
  “moduleFileExtension: [“json”, “vue”],
  “transform”: {
    “^.+\\.vue$”: “vue-jest”,
  },
  “moduleNameMapper”: {
    “^@/(.*)$”: “<rootDir>/src/$1”
  }
}
```


Setup for Vue projects

- To an existing project

```
# From the directory of package.json
yarn add jest @vue/test-utils vue-jest
```

- For a new project, vue-cli will prompts for Unit Testing Option (Mocha+Chai or Jest)

@vue/test-utils

@vue/test-utils

- Component inflator
  - mount(), shallowMount()
- DOM query
  - *wrapper*.find() similar to document.querySelector()
  - *wraper*.findAll() similar to document.querySelectorAll()
- Wrapper functions
  - Non-mutating: attributes(), exists(), html(), text(), isVisible()
  - Mutating: setChecked(), setData(), setPropse(), setSelected(), setValue()
  - Event: trigger()

Example: mount()/shallowMount()

- shallowMount(): inflate the component **without** its children
  - Faster
  - Recommended for Unit Testing
- mount(): inflate the component **and** its children

```
import {shallowMount} from '@vue/test-utils';
import YourFile from '@/src/components/YourFile.vue';

describe("Sample", () => {
  test("My first test", () => {
    const w = shallowMount(YourFile);

  });
});
```


Typical Skeleton

```
// In mytest.spec.js

import {shallowMount} from '@vue/test-utils';
import YourFile from '@/src/components/YourFile.vue';

describe("Sample", () => {
  test("My first test", () => {
    const w = shallowMount(YourFile);

    expect(___).toBe____();
    expect(___).toBe____();
    expect(___).toBe____();
  });
});
```

Example: find()/findAll()

```
import {shallowMount} from '@vue/test-utils';
import YourFile from '@/src/components/YourFile.vue';

describe("Sample", () => {
  it("has a level-2 heading", () => {
    const w = shallowMount(YourFile);
    const heading = w.find("h2");
    expect(heading.exists()).toBe(true);
    expect(heading.text()).toContain("Sample Title Text");
  });

  it("has level-3 headings", () => {
    const w = shallowMount(YourFile);
    const headings = w.findAll("h3");
    expect(headings.length).toBe(5);
  });

});
```


Example: at(), text(), isVisible()

```
import {shallowMount} from '@vue/test-utils';
import YourFile from '@/src/components/YourFile.vue';

describe("Sample", () => {
  it("show 3 buttons with proper label", () => {
    const w = shallowMount(YourFile);
    const btns = w.findAll("#top button");
    expect(btns.length).toBe(3);
    expect(btns.at(0).text()).toContain("Action 1");
    expect(btns.at(1).text()).toContain("Action 2");
    // the third button must be initially hidden
    expect(btns.at(2).text()).toContain("Action 3");
    expect(btns.at(2).isVisible()).toBe(false);
  });
});
```

Example: setData()

```
<template>
  <div>
    <ul>
      <li v-for="x in items">{{x}}</li>
    </ul>
  </div>
</template>

<script>
export default {
  data() {
    return {
      items: []
    }
  }
}
</script>
```

```
import {shallowMount} from '@vue/test-utils';
import YourFile from '@/src/components/YourFile.vue';

describe("Sample", () => {

  it("show 3 buttons with proper label", () => {
    const w = shallowMount(YourFile);
    w.setData({items: ["Neon", "Fluor", "Carbon"]});
    const atoms = w.findAll("ul li");
    expect(atoms.length).toBe(3);
    expect(atoms.at(0).text()).toBe("Neon");
    expect(atoms.at(1).text()).toBe("Fluor");
    expect(atoms.at(2).text()).toBe("Carbon");
  });

});
```


Example: shallowMount with propsData

```
// in App.vue
<template>
  <div>
    <WorldClock
      timeZone="UTC+2" lang="fr" />
  </div>
</template>
```

```
// in WorldClockApp.vue
<script>
export default {
  props: ["timeZone", "lang"],
  methods: {
    // more code here
  }
}
</script>
```

```
import {shallowMount} from '@vue/test-utils';
import WorldClock from '@/src/components/WorldClock.vue';

describe("Sample", () => {

  it("show 3 buttons with proper label", () => {
    const w = shallowMount(WorldClock, {
       propsData: {
         timeZone: "UTC+2",
         lang: "fr"
       }
     });
     // more test code
  });

});
```

Spy On Vue Instance Methods

```
// In MyComponent.vue
<template>
  <div>
    <button onClick=”register”>SignUp</button>
  </div>
</template>
<script>
export default {
  methods: {
    register() {
      // your code here
    },
    otherFunction() {
    }
  }
}
</script>
```

```
// mycomponent.test.js
import {shallowMount} from ‘@vue/test-utils’;
import MyComponent from ‘@/MyComponent.vue’;

describe(“Sample”, () => {

  it(“calls register”, () => {
    const regSpy = jest.fn();
    const w = shallowMount(MyComponent, {
      methods: {
        register: regSpy
      }
    });
    w.find(“button”).trigger(‘click’);
    expect(regSpy).toBeCalled();
  });
});
```


Spy On Vue Router Methods

```
// In MyComponent.vue
<template>
  <div>
  </div>
</template>
<script>
export default {
  methods: {
    toPurchaseDetails() {
      this.$router.push({
        path: “/pdetails”
      })
    },
  }
}
</script>
```

```
// mycomponent.test.js
import {shallowMount, createLocalVue } from ‘@vue/test-utils’;
Import VueRouter from ‘vue-router’;
import MyComponent from ‘@/MyComponent.vue’;

describe(“Sample”, () => {
  it(“moves to purchase details”, () => {
    const myVue = createLocalVue();
    const myRouter = new VueRouter();
    myVue.use(VueRouter);
    const pushSpy = jest.fn();
    const w = shallowMount(MyComponent, {localVue: myVue, router: myRouter});
    w.vm.$router.push = pushSpy;
    // more code
    expect(regSpy).toBeCalled();
  });
});
```

React + Jest

Additional packages

- Option #1 for projects created using CRA
  - `yarn add -D react-test-renderer`
- Option #1 for non-CRA projects
  - `jest babel-jest @babel/preset-env @babel/preset-react react-test-renderer`