Jest

TODO

- Copy some sides from training/React/keynote/09-jest.key to here

Jest Overview

- “a delightful JavaScript Testing Framework with a focus on simplicity”
  - https://jestjs.io/
- Supports unit tests
- Uses jsdom to simulate browser environment
  - “A JavaScript implementation of the WHATWG DOM and HTML standards, for use with node.js”
  - https://github.com/jsdom/jsdom
- Can install Jest plugin through Vue CLI
  - also installs **`@vue/test-utils`** and **`@vue/cli-plugin-unit-jest`** (depends on **`jest`** and more)
  - creates **`tests/unit/example.spec.js`** containing an example test
  - creates **`jest.config.js`** that configures the Jest plugin
- To run tests
  - **`npm run test:unit`**

Test Files

- Tests can be written in any **`.spec.js`** file under **`tests/unit`**
  - file extensions can also be **`.jsx`**, **`.ts`**, or **`.tsx`**
- To colocate tests with source files they test,
  modify **`testMatch`** property in **`jest`** section of **`package.json`**

```
"testMatch": [
  "**/src/**/*.spec.js"
],
```

Watch Mode

- To enable, add **`--watch`** to npm script

```
"test:unit": "vue-cli-service test:unit --watch"
```

Same Tests

```
describe('some component', () => {
  test('does something', () => {
    const someResult = someAction();
    expect(someResult).toBe(someExpectedValue);
  });
});
```

Test Functions

- Tests use familiar functions found in other test libraries
- `describe`
  - takes a test suite name and a function that defines it
- `before`
  - takes a function that runs at beginning of test suite
- `after`
  - takes a function that runs at end of test suite
- `beforeEach`
  - takes a function that runs before each test in the suite
- `afterEach`
  - takes a function that runs after each test in the suite
- `test` or `it`
  - takes a test name and a function that defines it
  - can be called inside a test suite or at file scope

tests are not required
to be in test suites

These are for setup and teardown code.
When called outside a test suite,
they apply to all tests in the source file.
When called inside a test suite,
they only apply to tests in that suite.

Skip and Only

- Can add `.skip` after `describe` or `it`
  - to temporarily skip running them
- Can add `.only` after `describe` or `it`
  - to temporarily skip running other test suites or tests

Matchers ...

- “Test values in different ways”
    - https://jestjs.io/docs/en/using-matchers
- Uses “matchers” for assertions
- Can add **`.not`** before any matcher

… Matchers

- `toBe(`***`value`***`)` - compares references
- `toEqual(`***`value`***`)` - compares deeply with ===
- `toBeNull()`, `toBeUndefined()`, `toBeDefined()`
- `toBeTruthy()`, `toBeFalsy()` - compares with type coercions
- `toBeLessThan(`***`number`***`)`, `toBeLessThanOrEqual(`***`number`***`)`
- `toBeGreaterThan(`***`number`***`)`, `toBeGreaterThanOrEqual(`***`number`***`)`
- `toBeCloseTo(`***`number, digits`***`)` - for floating point; ***`digits`*** defaults to 2
- `toMatch(`***`regExp`***`)` - for strings
- `toContain(`***`element`***`)` - for arrays
- `expect(function).toThrow()`
  - takes nothing for throwing anything, a `String` message, a `RegExp`, or an error class
- and more!

Asynchronous Tests

- Four approaches
- 1) Function passed to `test` or `it` has `done` parameter that is a function
  - call `done` when test is finished
  - test fails if `done` is never called

```
test('some name', done => {
  const callback = result => {
    expect(result).toBe(expectedValue);
    done();
  };
  someAsyncFn(callback)
});
```

can also call `done.fail(error)` to explicitly cause test to fail

- 2) Return a `Promise`
  - test passes if `Promise` resolves and fails if it rejects

```
test('some name', () => {
  const args = ...;
  return someAsyncFn(args);
});
```

- 3) Use `Promise` matchers

  -
    ```
    expect(someAsyncFn()).resolves.toBe(goodValue);
    expect(someAsyncFn()).rejects.toBe(badValue);
    ```
- 4) Use `async/await`
  - test fails if a `Promise` throws

```
test('some name', async () => {
  const result = await someAsyncFn();
  expect(result).toBe(expectedValue);
});
```

My favorite!

Snapshots

- Saves a “snapshot” of the DOM
- Compares against it in subsequent runs
- Can update snapshots when components change

Mocks

- Can easily create mock versions of functions