



Jest



# 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
  - to add to an existing application: `vue add @vue/unit-jest`
  - adds several `devDependencies`, including `@vue/test-utils` <https://vue-test-utils.vuejs.org/>
  - creates `jest.config.js` that configures the Jest plugin
  - creates `tests/unit/example.spec.js` containing an example test



# 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, change `testMatch` property in `jest.config.js`

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

- Can delete `tests` directory
  - if this is done, modify `package.json` to add `"jest": true`, under `"eslintConfig"` `"env"`



# Jest API Highlights

- **describe**(*name*, *fn*)
  - describes a "test suite"
- **describe.only**(*name*, *fn*)
  - alias **fdescribe**
- **describe.skip**(*name*, *fn*)
  - alias **xdescribe**
- **beforeAll**(*fn*)
  - run once before all tests in suite begin
- **beforeEach**(*fn*)
  - run before each test in suite begins
- **afterEach**(*fn*)
  - run after each test in suite begins
- **afterAll**(*fn*)
  - run once after all tests in suite finish

test calls are not required  
to be inside a **describe**

- **test**(*name*, *fn*)
  - alias **it**
- **test.only**(*name*, *fn*)
  - alias **fit**
- **test.skip**(*name*, *fn*)
  - alias **xit**
- **expect**(*value*)
  - chain a matcher call onto this
  - see next two slides



# Jest Matchers ...

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

- **.not** - can be prepended to any matcher
- **.toBe(value)** - uses ===
- **.toEqual(value)**
  - deep object comparison
- **.toBeTruthy()**
- **.toBeFalsy()**
- **.toBeDefined()**
- **.toBeUndefined()**
- **.toBeNull()**
- **.toMatch(regex)**
- **.toMatchObject(object)**
  - all properties in *object* match those in receiver
- **.toBeCloseTo(number, digits)**
- **.toBeGreaterThan(number)**
- **.toBeGreaterThanOrEqual(number)**
- **.toBeLessThan(number)**
- **.toBeLessThanOrEqual(number)**
- **.toBeInstanceOf(Class)**
- **.toContain(item)** for arrays
- **.toContainEqual(item)**
  - deep object comparison
- **.toHaveLength(number)**



# ... Jest Matchers

- `.toBeCalled()`
  - alias is `.toBeCalled`
- `.toBeCalledTimes(number)`
- `.toBeCalledWith(arg1, arg2, ...)`
  - alias is `.toBeCalledWith`
- `.toBeCalledWith(arg1, arg2, ...)`
  - alias is `.lastCalledWith`
- `.toThrow()`
- `.toThrowError(error)`
- `.toMatchSnapshot()`
- `.toThrowErrorMatchingSnapshot()`

# To Run Tests

- `npm run test:unit`
- For verbose output
  - shows result of each test, not just each suite
  - add "`verbose: true,`" in `jest.config.js`
  - `console.log` output is suppressed when `verbose` is `true`!



# Watch Mode

- Can watch source and test files and automatically reruns tests when they change
- Can run all tests or only ones that failed in last run
- To enable, add `--watch` to npm script

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



# Example Tests

- The following tests are for the app shown in “Todo App” section

# Todo.spec.js ...

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

describe('Todo', () => {
  const text = 'buy milk';
  const todo = {text, done: false};

  test('should render', () => {
    const wrapper = shallowMount(Todo, {
      propsData: {onDeleteTodo() {}, onToggleDone() {}, todo}
    });
    const checkbox = wrapper.find('input[type="checkbox"]');
    expect(checkbox).not.toBeNull();
    const html = wrapper.html();
    expect(html).toContain(text); // the todo text
    expect(html).toContain('Delete'); // the button
  });
});
```



# ... Todo.spec.js

```
test('should handle Delete button', () => {
  const onDeleteTodo = jest.fn();
  const wrapper = shallowMount(Todo, {
    propsData: {
      onDeleteTodo,
      onToggleDone() {},
      todo
    }
  });
  const deleteBtn = wrapper.find('button');
  deleteBtn.trigger('click');
  expect(onDeleteTodo).toHaveBeenCalled();
});

test('should toggle done', () => {
  const onToggleDone = jest.fn();
  const wrapper = shallowMount(Todo, {
    propsData: {
      onDeleteTodo() {},
      onToggleDone,
      todo
    }
  });
  const checkbox = wrapper.find('input[type="checkbox"]');
  checkbox.trigger('click');
  expect(onToggleDone).toHaveBeenCalled();
});
```

# TodoList.spec.js ...

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

describe('TodoList', () => {
  const PREDEFINED_TODOS = 2;

  function expectTodoCount(wrapper, count) {
    // Each todo has an <li> root element.
    const lis = wrapper.findAll('li');
    expect(lis.length).toBe(count);
  }

  test('should render', () => {
    const wrapper = mount(TodoList);
    const html = wrapper.html();
    expect(html).toContain('To Do List');
    expect(html).toContain('1 of 2 remaining');
    expect(html).toContain('Archive Completed');
    expectTodoCount(wrapper, PREDEFINED_TODOS);
  });
});
```

searching for descendant elements deeper than direct children requires using `mount` instead of `shallowMount`



# ... TodoList.spec.js ...

```
test('should add a todo', () => {
  const wrapper = mount(TodoList);
  const input = wrapper.find('.todo-input');
  const text = 'buy milk';
  input.element.value = text;
  input.trigger('input');

  const addBtn = wrapper.find('.add-btn');
  addBtn.trigger('click');

  expectTodoCount(wrapper, PREDEFINED_TODOS + 1);
  const html = wrapper.html();
  expect(html).toContain(text);
});

test('should archive completed', () => {
  const wrapper = mount(TodoList);
  const archiveBtn = wrapper.find('.archive-btn');
  archiveBtn.trigger('click');
  expectTodoCount(wrapper, PREDEFINED_TODOS - 1);
  const html = wrapper.html();
  expect(html).toContain('1 of 1 remaining');
});
```

# ... TodoList.spec.js

```
test('should delete a todo', () => {
  const wrapper = mount(TodoList);
  const deleteBtn = wrapper.find('.delete-btn'); // for first todo
  deleteBtn.trigger('click');
  expectTodoCount(wrapper, PREDEFINED_TODOS - 1);
});

test('should toggle a todo', () => {
  const wrapper = mount(TodoList);
  const checkboxes = wrapper.findAll('input[type="checkbox"]');
  expect(checkboxes.length).toBe(2);

  checkboxes.at(1).trigger('click'); // second todo
  let html = wrapper.html();
  expect(html).toContain('0 of 2 remaining');

  checkboxes.at(0).trigger('click'); // first todo
  html = wrapper.html();
  expect(html).toContain('1 of 2 remaining');
});
});
```

checkboxes  
is not an array



# Snapshot Testing

- Snapshot tests assert that ...
  - a component will render same content as last successful test
- In first run ...
  - **toMatchSnapshot** matchers save a representation of rendered output in subdirectory of test file named `__snapshots__`
    - mostly, but not exactly HTML
- In subsequent runs ...
  - same representation is generated again and compared to what was saved in last successful run
- When snapshot tests fail ...
  - scroll back to review differences in rendered output
  - if changes are correct, press “u” to accept them
    - overwrites previous snapshot files with new ones
  - if changes are incorrect, fix code and run tests again

`__snapshot__` directories should be checked into version control!

# Snapshot Test Example

```
import {mount} from '@vue/test-utils';

...

test('should match snapshot', () => {
  const wrapper = mount(Todo, {
    propsData: {
      onDeleteTodo() {},
      onToggleDone() {},
      todo
    }
  });
  expect(wrapper.element).toMatchSnapshot();
});
```



# 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!