tests/unit/Todo.spec.js

javascript

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

import Todo from '@/components/Todo.vue';

describe('Todo.vue Unit Tests', () => {

let wrapper;

// Setup: mount the component before each test

beforeEach(() => {

wrapper = shallowMount(Todo, {

// If needed, you could inject mocks or stub child-components here.

});

});

// Cleanup: destroy the wrapper after each test to avoid side effects

afterEach(() => {

wrapper.destroy();

});

// Test 1: Computed Property (todoCount)

it('should correctly compute the number of todos via todoCount computed property', async () => {

// Assert initial state: no todos means todoCount should be 0

expect(wrapper.vm.todoCount).toBe(0);

// Directly update the data to simulate adding a todo

wrapper.setData({ todos: ['Learn Vue Testing'] });

await wrapper.vm.$nextTick();

// Check that the computed property reflects the change

expect(wrapper.vm.todoCount).toBe(1);

});

// Test 2: User Interaction (Adding a Todo)

it('should add a new todo when the user types in an input and clicks the add button', async () => {

// Locate the input field and simulate user input

const input = wrapper.find('input');

input.setValue('New Todo Item');

// Locate the button and simulate a user click

const button = wrapper.find('button');

await button.trigger('click');

await wrapper.vm.$nextTick();

// Verify that the new todo has been added to the todos array

expect(wrapper.vm.todos).toContain('New Todo Item');

});

// Test 3: Edge Case (Submitting Empty Input)

it('should not add an empty todo and should display an error message', async () => {

// Set the input field to an empty string

const input = wrapper.find('input');

input.setValue('');

// Simulate clicking the add button

const button = wrapper.find('button');

await button.trigger('click');

await wrapper.vm.$nextTick();

// Verify that the error message is set in the component

expect(wrapper.vm.errorMsg).toBe('Input cannot be empty');

// Also check if the error message is actually rendered in the DOM

const errorEl = wrapper.find('.error');

expect(errorEl.exists()).toBe(true);

expect(errorEl.text()).toBe('Input cannot be empty');

});

});

**Testing Notes & Screencast Explanation**

1. **Mocking:** In these tests, **no external dependencies required mocking** because the Todo.vue component is self-contained. We used shallowMount from vue-test-utils to isolate the component. If the component had external API calls or child components with heavy logic, we would stub or mock those dependencies. In this case, direct state manipulation (using setData) and standard DOM event triggers were sufficient.
2. **A Failing Test and Its Fix:**
   * **Issue:** Initially, the test for the edge case (submitting an empty input) failed because the component did not update the errorMsg when the input was empty. The failing test revealed that despite clicking the add button, no validation was performed.
   * **Resolution:** After noticing the failure, the implementation in the addTodo method was updated to include a simple check. For instance:

javascript

addTodo() {

if (!this.newTodo.trim()) {

this.errorMsg = 'Input cannot be empty';

return;

}

// Proceed with adding the todo

this.todos.push(this.newTodo);

this.newTodo = '';

this.errorMsg = '';

}

This change ensured that the component would now properly handle the empty input case. Once adjusted, the failing test passed, serving as an effective demonstration of test-driven troubleshooting.

These unit tests validate critical aspects of the Todo component, from computed logic to user actions and edge-case handling. By following this approach, you ensure that the component behaves as expected, and any future changes that cause regressions will be quickly identified through automated testing.