Angular Component Testing with Jest

Khalil Themri

Introduction to Component Testing

In this chapter, we'll look at how to test Angular components using Jest . Below are examples of test setups and explanations for each part of the code.

Basic Component Test Setup

```
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { DebugElement } from '@angular/core';
import { DashboardComponent } from
    '../../src/app/components/dashboard/dashboard.component';

describe('DashboardComponent', () => {
  let fixture: ComponentFixture<DashboardComponent>;
  // fixture :Stores an instance of the ComponentFixture,
  // which contains methods that help you debug and test a component

  let component: DashboardComponent;
  // component:Stores an instance of the DashboardComponent

  let rootElement: DebugElement
  // rootElementStores the DebugElement for your component,
  // which is how youll access its children

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        DashboardComponent,
      ],
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(DashboardComponent);
    component = fixture.componentInstance;
    rootElement = fixture.debugElement;
    fixture.detectChanges();
  });

  describe("init test", () => {
    it('component testing ', () => {
      expect(component).toBeTruthy();
    });
  });
});
```

Listing 1: Basic Test Setup for DashboardComponent

Explanation:

- **TestBed:** Angular's primary API for setting up and configuring unit tests.
- **ComponentFixture:** A test harness that provides methods to interact with the component.
- **DebugElement:** Facilitates DOM manipulation and interaction in tests.
- **beforeEach:** Prepares the testing environment by configuring and creating the component instance.
- **it:** Defines an individual test case. In this example, it checks if the 'DashboardComponent' is created successfully.

Advanced Test Setup with Additional Modules

```
import { TestBed, ComponentFixture, fakeAsync, tick } from
    '@angular/core/testing';
import { of } from 'rxjs';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { MatButtonModule } from '@angular/material/button';
import { MatSnackBarModule } from '@angular/material/snack-bar';
import { DebugElement } from '@angular/core';
import { By } from '@angular/platform-browser';
import { DashboardComponent } from
    '../../src/app/components/dashboard/dashboard.component';
import { AddressService } from '../../src/app/services/address.service';
import { CustomerService } from
    '../../src/app/services/customer.service';

describe('DashboardComponent', () => {
  let component: DashboardComponent;
  let fixture: ComponentFixture<DashboardComponent>;
  let rootElement: DebugElement;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        DashboardComponent,
        CommonModule,
        RouterModule.forRoot([]),
        MatButtonModule,
        MatSnackBarModule,
      ],
      // Adding required modules for the component and its dependencies
    }).compileComponents();
  });
```

```
  beforeEach(() => {
    fixture = TestBed.createComponent(DashboardComponent);
    component = fixture.componentInstance;
    rootElement = fixture.debugElement;
    fixture.detectChanges();
  });

  describe("init test", () => {
    it('component testing ', () => {
      expect(component).toBeTruthy();
    });
  });
});
```

Listing 2: Advanced Test Setup with Additional Modules

Explanation:

- **Additional Modules:** In this setup, various Angular Material modules (e.g., 'MatButtonModule', 'MatSnackBarModule') and 'RouterModule' are imported to support component testing. This ensures that all dependencies of 'DashboardComponent' are available during testing.
- **NoopAnimationsModule:** Often used to disable animations during testing to avoid timing issues.
- **fakeAsync/tick:** Utility functions used for testing asynchronous code by simulating the passage of time.

Chapter 1

Testing Angular Components with Spies and Mocks

This chapter focuses on advanced Angular component testing techniques using Jest, with a particular emphasis on mocking dependencies and using spies to simulate and verify interactions with those dependencies.

Mocking Dependencies and Using Spies

```
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, ReactiveFormsModule } from '@angular/forms';
import { of } from 'rxjs';
import { DebugElement } from '@angular/core';
import { NoopAnimationsModule } from
    '@angular/platform-browser/animations';
import { CustomerFormComponent } from
    '../../src/app/components/customer-form/customer-form.component';
import { CustomerService } from
    '../../src/app/services/customer.service';

describe('CustomerFormComponent', () => {
  let component: CustomerFormComponent;
  let fixture: ComponentFixture<CustomerFormComponent>;
  let rootElement: DebugElement;

  let customerServiceSpy: any;
  let snackBarSpy: any;
  let routerSpy: any;
```

```
let activatedRouteSpy: any;

beforeEach(async () => {
  customerServiceSpy = {
    getCustomerById: jest.fn(),
    updateCustomer: jest.fn().mockReturnValue(of({})),
    createCustomer: jest.fn().mockReturnValue(of({})),
  };

  snackBarSpy = {
    open: jest.fn(),
  };

  routerSpy = {
    navigate: jest.fn(),
  };

  activatedRouteSpy = {
    snapshot: {
      params: {
        id: null,
      },
    },
  };

  await TestBed.configureTestingModule({
    imports: [
      ReactiveFormsModule,
      CustomerFormComponent,
      NoopAnimationsModule
    ],
    providers: [
      FormBuilder,
      { provide: CustomerService, useValue: customerServiceSpy },
      { provide: MatSnackBar, useValue: snackBarSpy },
      { provide: Router, useValue: routerSpy },
      { provide: ActivatedRoute, useValue: activatedRouteSpy },
    ],
  }).compileComponents();
});

beforeEach(() => {
  fixture = TestBed.createComponent(CustomerFormComponent);
  component = fixture.componentInstance;
  rootElement = fixture.debugElement;
  fixture.detectChanges();
});

describe('init', () => {
  it('should create the component', () => {
```

```
    expect(component).toBeTruthy();
  });
});
});
```

Listing 1.1: Testing CustomerFormComponent with Mocked Dependencies

Explanation:

- **Spies and Mocks:** The test uses Jest to create spies (‘jest.fn()‘) for the services used by the component, such as ‘CustomerService‘, ‘MatSnackBar‘, and ‘Router‘. This allows us to simulate the behavior of these services and verify interactions with them.
- **TestBed Configuration:** The ‘TestBed.configureTestingModule‘ method is used to set up the testing environment, including importing required modules and providing mocked services.
- **ReactiveFormsModule:** This is imported to support form functionalities within the ‘CustomerFormComponent‘.
- **NoopAnimationsModule:** This is used to disable animations during testing to prevent timing issues.
- **Component Initialization:** The test checks if the ‘CustomerFormComponent‘ is created successfully.

Testing Component Initialization and Dependency Interaction

```
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder } from '@angular/forms';
import { of } from 'rxjs';
import { DebugElement } from '@angular/core';
import { NoopAnimationsModule } from
    '@angular/platform-browser/animations';
import { CustomerFormComponent } from
    '../../src/app/components/customer-form/customer-form.component';
import { CustomerService } from
    '../../src/app/services/customer.service';

describe('CustomerFormComponent', () => {
    let component: CustomerFormComponent;
    let fixture: ComponentFixture<CustomerFormComponent>;

    let customerServiceSpy: any;
```

```
let snackBarSpy: any;
let routerSpy: any;
let activatedRouteSpy: any;

beforeEach(async () => {
    customerServiceSpy = {
        getCustomerById: jest.fn(),
        updateCustomer: jest.fn().mockReturnValue(of({})),
        createCustomer: jest.fn().mockReturnValue(of({})),
    };

    snackBarSpy = {
        open: jest.fn(),
    };

    routerSpy = {
        navigate: jest.fn(),
    };

    activatedRouteSpy = {
        snapshot: {
            params: {
                id: null,
            },
        },
    };

    await TestBed.configureTestingModule({
        imports: [
            CustomerFormComponent,
            NoopAnimationsModule
        ],
        providers: [
            FormBuilder,
            { provide: CustomerService, useValue: customerServiceSpy
                },
            { provide: MatSnackBar, useValue: snackBarSpy },
            { provide: Router, useValue: routerSpy },
            { provide: ActivatedRoute, useValue: activatedRouteSpy },
        ],
    }).compileComponents();
});

beforeEach(() => {
    fixture = TestBed.createComponent(CustomerFormComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
});

describe('init', () => {
```

```
        it('should create the component', () => {
            expect(component).toBeTruthy();
        });
    });

    describe('ngOnInit', () => {
        it('should initialize the form and add the data if id is
             present', () => {
            const mockCustomer = {
                id: 1,
                firstName: 'kt',
                lastName: 'the',
                email: 'kt.the@barthauer.com',
                addressId: 123,
                storeId: 456,
            };

            activatedRouteSpy.snapshot.params.id = mockCustomer.id;
            customerServiceSpy.getCustomerById.mockReturnValue(of(mockCustomer));

            component.ngOnInit();
            fixture.detectChanges();

            expect(customerServiceSpy.getCustomerById).toHaveBeenCalledWith(mockCustomer.id);
            expect(component.customerForm.value).toEqual({
                firstName: 'kt',
                lastName: 'the',
                email: 'kt.the@barthauer.com',
                addressId: 123,
                storeId: 456,
            });
        });
    });
});
```

Listing 1.2: Testing ngOnInit and Data Initialization in CustomerFormComponent

Explanation:

- **Testing ngOnInit:** This test verifies that when ‘ngOnInit‘ is called, the component correctly initializes the form with data if an ‘id‘ is provided through ‘ActivatedRoute‘.
- **Mocking ActivatedRoute:** The ‘ActivatedRoute‘ spy is used to simulate the presence of an ‘id‘ in the route parameters.
- **Mocking Service Calls:** ‘CustomerService.getCustomerById‘ is mocked to return the expected data for the provided ‘id‘.

- **Form Initialization:** The test checks that the form is populated correctly with the data retrieved from the service.

Testing Component Methods and Form Submissions

```
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder } from '@angular/forms';
import { of } from 'rxjs';
import { DebugElement } from '@angular/core';
import { NoopAnimationsModule } from
    '@angular/platform-browser/animations';
import { CustomerFormComponent } from
    '../../src/app/components/customer-form/customer-form.component';
import { CustomerDTO } from '../../src/app/models/customer-dto';
import { CustomerService } from
    '../../src/app/services/customer.service';

describe('CustomerFormComponent', () => {
    let component: CustomerFormComponent;
    let fixture: ComponentFixture<CustomerFormComponent>;

    let customerServiceSpy: any;
    let snackBarSpy: any;
    let routerSpy: any;
    let activatedRouteSpy: any;

    beforeEach(async () => {
        customerServiceSpy = {
            getCustomerById: jest.fn(),
            updateCustomer: jest.fn().mockReturnValue(of({})),
            createCustomer: jest.fn().mockReturnValue(of({})),
        };

        snackBarSpy = {
            open: jest.fn(),
        };

        routerSpy = {
            navigate: jest.fn(),
        };

        activatedRouteSpy = {
            snapshot: {
                params: {
                    id: null,
```

```
            },
        },
    };

    await TestBed.configureTestingModule({
        imports: [
            CustomerFormComponent,
            NoopAnimationsModule
        ],
        providers: [
            FormBuilder,
            { provide: CustomerService, useValue: customerServiceSpy
                },
            { provide: MatSnackBar, useValue: snackBarSpy },
            { provide: Router, useValue: routerSpy },
            { provide: ActivatedRoute, useValue: activatedRouteSpy },
        ],
    }).compileComponents();
});

beforeEach(() => {
    fixture = TestBed.createComponent(CustomerFormComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
});

describe('init', () => {
    it('should create the component', () => {
        expect(component).toBeTruthy();
    });
});

describe('ngOnInit', () => {
    it('should initialize the form and add the data if id is
         present', () => {
        const mockCustomer = {
            id: 1,
            firstName: 'kt',
            lastName: 'the',
            email: 'kt.the@barthauer.com',
            addressId: '123',
            storeId: '456',
        };

        activatedRouteSpy.snapshot.params.id = mockCustomer.id;
        customerServiceSpy.getCustomerById.mockReturnValue(of(mockCustomer));

        component.ngOnInit();
        fixture.detectChanges();
```

```
        expect(customerServiceSpy.getCustomerById).toHaveBeenCalledWith(mockCustomer.id);
        expect(component.customerForm.value).toEqual({
            firstName: 'kt',
            lastName: 'the',
            email: 'kt.the@barthauer.com',
            addressId: '123',
            storeId: '456',
        });
    });
});

describe('onSubmit', () => {
    it('should call createCustomer if customerId is not present', ()
        => {
        const mockCustomer: Partial<CustomerDTO> = {
            firstName: 'kt',
            lastName: 'the',
            email: 'kt.the@barthauer.com',
            addressId: 123,
            storeId: 456,
        };

        customerServiceSpy.createCustomer.mockReturnValue(of('Customer
            created successfully'));
        component.customerForm.patchValue(mockCustomer);
        component.onSubmit();

        expect(customerServiceSpy.createCustomer).toHaveBeenCalledWith(mockCustomer,
            123, 456);
        expect(routerSpy.navigate).toHaveBeenCalledWith(['/customers']);
    });

    it('should call createCustomer if customerId is not present', ()
        => {
        const createCustomerSpy = jest.spyOn((component as any),
            'createCustomer');
        component.customerId = null;

        component.customerForm.setValue({
            firstName: 'kt',
            lastName: 'the',
            email: 'kt.the@barthauer.com',
            addressId: 123,
            storeId: 456,
        });

        component.onSubmit();

        expect(createCustomerSpy).toHaveBeenCalled();
        expect(createCustomerSpy).toHaveBeenCalledWith(
```

```
            {
                firstName: 'kt',
                lastName: 'the',
                email: 'kt.the@barthauer.com',
                addressId: 123,
                storeId: 456,
            },
            123,
            456
        );
    });
});
});
```

Listing 1.3: Testing Form Submission in CustomerFormComponent

Explanation:

- **Testing onSubmit:** This test verifies the behavior of the 'onSubmit' method when creating a new customer. It ensures that the 'createCustomer' method of the 'CustomerService' is called with the correct parameters and that the user is navigated to the correct route upon successful creation.
- **Mocking CustomerService:** The 'createCustomer' method is mocked to simulate the customer creation process.
- **Spying on createCustomer:** A Jest spy is used to verify that the 'createCustomer' method within the component is called correctly.
- **Router Navigation:** The test also checks that the router navigates to the correct route after the form submission.