Angular Service Testing with Jest

Khalil Themri

Unit Testing in Angular with Jest

In this document, we will demonstrate how to unit test an Angular service using Jest. The test example uses Angular's 'TestBed' and 'HttpTestingController' to mock HTTP requests.

Jest Test for CustomerService

This code is a test suite for the `CustomerService` class, written in **Angular** using **jest** and Angular's `HttpClientTestingModule`. Here's an explanation of each part:

Structure:

1. `describe('CustomerService', ...):`
   - The `describe` function defines a test suite for the `CustomerService`. This is a grouping of related test cases, and the suite is identified by the name `'CustomerService'`.
2. `let service: CustomerService; let httpMock: HttpTestingController;:`
   - Declares two variables:
     - `service`: This holds the instance of `CustomerService` that is being tested.
     - `httpMock`: This mocks and intercepts HTTP requests so that actual network calls aren't made.
3. `beforeEach(() => ...):`
   - A setup function that runs before each test in the suite. It ensures that each test starts with a fresh instance of the `CustomerService` and mock HTTP service.

     Inside `beforeEach`:
     - `TestBed.configureTestingModule({ ... }):`
       - `TestBed` is an Angular utility that helps create a test environment. It configures the testing module, registering the necessary dependencies:
         - `CustomerService`: The service being tested.
         - `provideHttpClient()`: Provides the `HttpClient` service to enable HTTP communication.
         - `provideHttpClientTesting()`: Provides the `HttpTestingController` to mock HTTP requests for testing without real network communication.
     - `service = TestBed.inject(CustomerService):`

        * Injects the `CustomerService` into the `service` variable, allowing tests to interact with it.
    - `httpMock = TestBed.inject(HttpTestingController):`
        * Injects the `HttpTestingController` into the `httpMock` variable, allowing the test to mock and verify HTTP requests.

4. `afterEach(() => { httpMock.verify(); }):`

    - A cleanup function that runs after each test in the suite.
    - `httpMock.verify()` ensures that all mock HTTP requests have been correctly handled and no unexpected requests are left pending.

Key Concepts:

- `HttpTestingController`**:** Used in Angular tests to mock HTTP requests. Instead of making actual network calls, `HttpTestingController` allows us to simulate and control HTTP requests and responses.
- `TestBed.configureTestingModule()`**:** Configures and initializes the Angular testing environment. It allows you to declare providers and dependencies needed for your service.
- **Lifecycle Hooks (**`beforeEach`**,** `afterEach`**):**
    - `beforeEach`**:** Prepares a fresh test setup.
    - `afterEach`**:** Ensures no HTTP calls are left pending, which is crucial to prevent tests from affecting each other.

Purpose:

This test suite creates a controlled environment for testing the `CustomerService` class, specifically focusing on its interaction with the `HttpClient`. The `HttpTestingController` is used to mock and verify HTTP requests, making the tests more reliable and isolated from actual backend systems.

Example Test

Here's an example of what a test might look like inside the suite:

Listing 1: Example Test for CustomerService

```
it('should make a GET request to fetch customers', () => {
  const mockCustomers = [{ id: 1, name: 'kt' }];

  service.getCustomers().subscribe(customers => {
    expect(customers).toEqual(mockCustomers);
  });
```

```
  const req = httpMock.expectOne('/api/customers');
  expect(req.request.method).toBe('GET');
  req.flush(mockCustomers); // Simulates the server response.
});
```

This test ensures that when `getCustomers` is called, it makes a `GET` request to `/api/customers` and the response matches the mocked data.