SPFx 1.20.0 Unit Testing Guide (with Jest)

This guide helps you set up and write unit test cases for your SPFx 1.20.0 solution using Jest, covering:

- Pure Functions

- API Calls

- React Components (UI)

# 1. Install Dependencies

In your SPFx project root directory, run:

npm install --save-dev jest @types/jest ts-jest identity-obj-proxy  
npm install --save-dev @testing-library/react @testing-library/jest-dom

# 2. Configure Jest

Create a file named jest.config.js in the project root:

module.exports = {  
 preset: 'ts-jest',  
 testEnvironment: 'jsdom',  
 moduleNameMapper: {  
 "\\.(css|less|scss|sass)$": "identity-obj-proxy"  
 },  
 transform: {  
 "^.+\\.tsx?$": "ts-jest"  
 },  
 testMatch: [  
 "\*\*/\_\_tests\_\_/\*\*/\*.test.ts?(x)",  
 "\*\*/?(\*.)+(spec|test).ts?(x)"  
 ],  
 moduleFileExtensions: ['ts', 'tsx', 'js', 'jsx', 'json', 'node'],  
 setupFilesAfterEnv: ['<rootDir>/jest.setup.js']  
};

# 3. Create Jest Setup File

Create a file named jest.setup.js and add:

import '@testing-library/jest-dom';

# 4. Add Script to package.json

Add the following script:

"scripts": {  
 "test": "jest"  
}

# 5. Example Tests

## 5.1 Pure Function

File: utils/mathUtils.ts

export const add = (a: number, b: number): number => a + b;

Test: \_\_tests\_\_/mathUtils.test.ts

import { add } from '../utils/mathUtils';  
  
describe('mathUtils', () => {  
 test('adds two numbers correctly', () => {  
 expect(add(2, 3)).toBe(5);  
 });  
});

## 5.2 API Call

File: services/UserService.ts

export const getUser = async (id: string): Promise<any> => {  
 const response = await fetch(`https://jsonplaceholder.typicode.com/users/${id}`);  
 return response.json();  
};

Test: \_\_tests\_\_/UserService.test.ts

import { getUser } from '../services/UserService';  
  
global.fetch = jest.fn(() =>  
 Promise.resolve({  
 json: () => Promise.resolve({ id: '1', name: 'John Doe' })  
 })  
) as jest.Mock;  
  
describe('UserService', () => {  
 test('fetches user data', async () => {  
 const user = await getUser('1');  
 expect(user).toEqual({ id: '1', name: 'John Doe' });  
 expect(fetch).toHaveBeenCalledWith('https://jsonplaceholder.typicode.com/users/1');  
 });  
});

## 5.3 React Component

File: components/Greeting.tsx

import \* as React from 'react';  
  
interface Props {  
 name: string;  
}  
  
const Greeting: React.FC<Props> = ({ name }) => <div>Hello, {name}!</div>;  
  
export default Greeting;

Test: \_\_tests\_\_/Greeting.test.tsx

import \* as React from 'react';  
import { render, screen } from '@testing-library/react';  
import Greeting from '../components/Greeting';  
  
describe('Greeting Component', () => {  
 test('renders with name', () => {  
 render(<Greeting name="Munendra" />);  
 expect(screen.getByText('Hello, Munendra!')).toBeInTheDocument();  
 });  
});

# 6. Tips for SPFx-specific Projects

- For SPFx services using @microsoft/sp-http, mock SPHttpClient using Jest mocks.  
- Use dependency injection or mocking for PageContext and SPFx context-based services.  
- For React components:  
 - Test prop changes, state updates, and rendering.  
 - Use @testing-library/react for UI behavior testing.