**Aim:** Write Unite test cases and carryout functional testing of the software.

**Description:**

In this example, we demonstrated how to write unit tests using Mocha and Chai for a simple utility class, StringUtil. This class contains two methods:

* reverseString(): Reverses a given string, throwing an error if the input is not a string.
* isPalindrome(): Checks if a string is a palindrome (i.e., it reads the same backward and forward), throwing an error if the input is not a string.

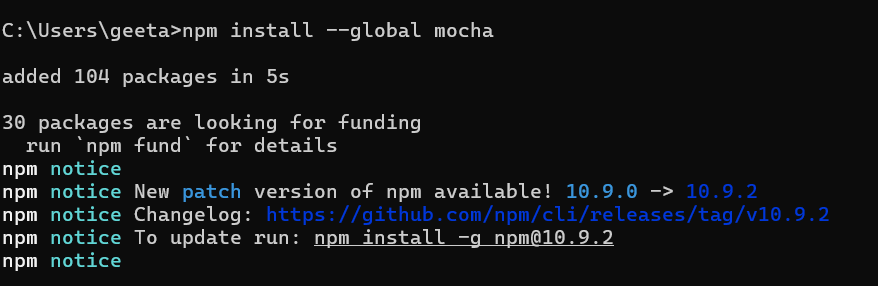
The tests are structured to check the correct functionality of these methods, including edge cases like empty strings and invalid inputs (e.g., numbers or non-string values). We used Mocha as the test framework and Chai for assertions, specifically the expect style.

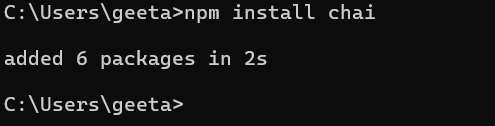
The test file was written as an ES module (.mjs extension) and used the ES import syntax to import both the StringUtil module and the Chai assertion library.

Steps Taken in the Example

1. Writing the StringUtil class:  
   The class StringUtil was implemented with two methods, one for reversing a string and another for checking if a string is a palindrome.
2. Writing the test cases:  
   In the test file, we used Mocha to define test cases for both methods, including valid and invalid input tests.
3. ES Module Syntax:  
   We used ES module imports and exports (import/export) in the test file and the module, ensuring that the import statements were correct and included file extensions.
4. Running the tests:  
   We ran the tests using Mocha and made sure to resolve issues related to module paths, duplicate declarations, and caching errors.

**Implementation:**





**// stringUtil.test.js**

import { expect } from 'chai';

import StringUtil from '../stringUtil.js'; // Add .js extension

describe('StringUtil Tests', () => {

let stringUtil;

beforeEach(() => {

stringUtil = new StringUtil();

});

describe('reverseString()', () => {

it('should reverse a given string', () => {

expect(stringUtil.reverseString('hello')).to.equal('olleh');

expect(stringUtil.reverseString('')).to.equal('');

});

it('should throw an error for non-string input', () => {

expect(() => stringUtil.reverseString(123)).to.throw('Input must be a string');

});

});

describe('isPalindrome()', () => {

it('should return true for a palindrome string', () => {

expect(stringUtil.isPalindrome('madam')).to.be.true;

expect(stringUtil.isPalindrome('racecar')).to.be.true;

});

it('should return false for a non-palindrome string', () => {

expect(stringUtil.isPalindrome('hello')).to.be.false;

});

it('should throw an error for non-string input', () => {

expect(() => stringUtil.isPalindrome(123)).to.throw('Input must be a string');

});

});

});

**// stringUtil.js**

class StringUtil {

reverseString(str) {

if (typeof str !== 'string') {

throw new Error('Input must be a string');

}

return str.split('').reverse().join('');

}

isPalindrome(str) {

if (typeof str !== 'string') {

throw new Error('Input must be a string');

}

const reversed = str.split('').reverse().join('');

return str === reversed;

}

}

module.exports = StringUtil;

**//package.json**

{

"type": "module"

}

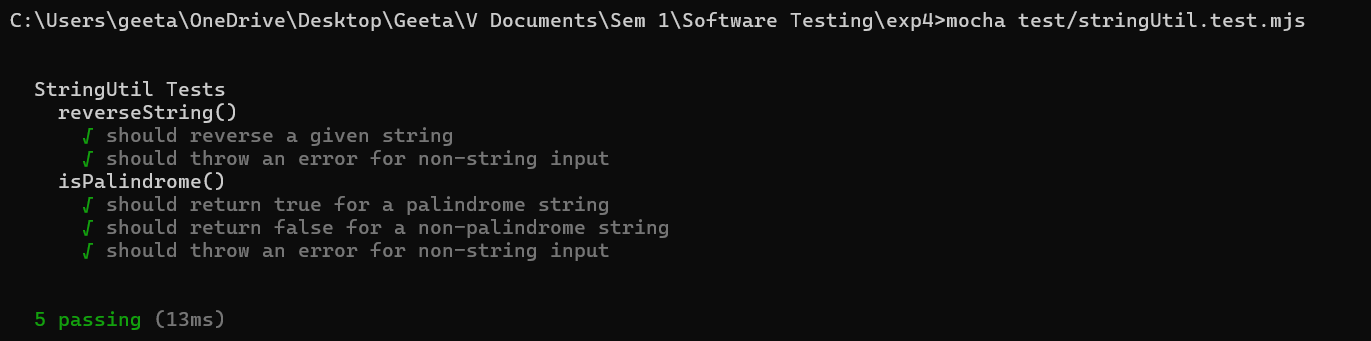
*Your project structure should look like this:*

*project/*

*stringUtil.js*

*test/*

*stringUtil.test.js*



**Conclusion:**

This example demonstrates how to write effective unit tests for a JavaScript class using Mocha and Chai. By:

* Structuring your class and test files correctly,
* Ensuring proper handling of ES module syntax,
* Managing imports and file paths with precision,

You can ensure that your unit tests are organized and functional. The tests provide clear validation for expected behavior, error handling, and edge cases. Furthermore, resolving issues such as duplicate imports and module resolution errors ensures smooth test execution.

Through this process, we showed how to implement, test, and troubleshoot basic JavaScript functionality using popular testing tools like Mocha and Chai.