**Title:** Test Plan for String Manipulation and File Operations in JavaScript

**By whom:** Van Minh Le

**Date:** 23.03.2025

**Version:** 1.0

**1. Introduction**

This test plan defines the strategy, process, and workflow for testing JavaScript string manipulation and file operations. The goal is to verify that various string operations and file read/write functions work correctly using Node.js.

**1.1. Scope**

**1.1.1. In Scope**

The following functionalities will be tested:

* String operations (length, character retrieval, slicing, case change, concatenation, trimming, replacing, splitting).
* File operations (writing and reading text files in JavaScript using Node.js).
* Verification of outputs in Node.js console.

**1.1.2. Out of Scope**

* Performance testing of string operations.
* File operations using other languages (C#).
* UI-based verification.

**1.2. Quality Objective**

The objective of this test is to ensure:

* All string operations function as expected.
* The file write and read functions correctly store and retrieve data.
* All test cases pass before submission.

**1.3. Roles and Responsibilities**

| **Role** | **Responsibility** |
| --- | --- |
| Developer | Implements functions and ensures correctness. |
| Tester | Executes test cases and records results. |
| Reviewer | Reviews test results and verifies code quality. |

**2. Test Methodology**

**2.1. Test Levels**

* **Unit Testing**: Each string operation and file function is tested individually.
* **Integration Testing**: Ensure functions work together correctly.
* **System Testing**: Verify all functionalities meet the requirement.

**2.2. Suspension Criteria and Resumption Requirements**

* Testing is suspended if critical bugs block further execution.
* Testing resumes once the issues are fixed and verified.

**2.3. Test Completeness**

The testing is considered complete when:

* 100% of the required functionalities are tested.
* All test cases pass without critical defects.
* Code meets JavaScript coding guidelines.

**3. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Input** | **Expected Output** | **Status** |
| TC-001 | Create string variables | myString = "This is a string"  anotherString = " Another string"  hello = "Hello there!"  myName = "van minh" | myString: [**This is a string**] anotherString: [ **Another string**] hello: [**Hello there!**] myName: [**van minh**] | Pass |
| TC-002 | Get length of myString | myString.length | Length of myString : **16** | Pass |
| TC-003 | Get first and 11th character of myString | myString[0], myString[10] | First Character of myString : **T**  11th Character of myString : **s** | Pass |
| TC-004 | Slice "is a" from myString | myString.slice(5, 9) | Create sliceString by slice “is a” from myString : **is a** | Pass |
| TC-005 | Get "the" from anotherString | anotherString.substring(6, 9); | Create substringString by substring “the” from anotherString : **the** | Pass |
| TC-006 | Change myName to uppercase and lowercase | myName.toUpperCase(), myName.toLowerCase() | Uppercase of myName : VAN MINH  Lowercase of myName : van minh | Pass |
| TC-007 | Concatenate hello and myName | hello.concat(myName) | Concat hello and myName together : **Hello there! van minh** | Pass |
| TC-008 | Trim spaces from anotherString | anotherString.trim() | Trim the spaces in anotherString : **Another string** | Pass |
| TC-009 | Replace "is a" with empty space in myString | myString.replace("is a", "") | Replace “is a” to an empty space in myString : **This string** | Pas |
| TC-010 | Split myString on spaces | myString.split(" ") | Split myString on the spaces : **This,is,a,string** | Pass |
| TC-011 | Write string to output.txt | fs.writeFile("output.txt", data, callback) | File written successfully | Pass |
| TC-012 | Read string from output.txt | fs.readFile("output.txt", callback) | Reading from file  The file content are:  **This is a string that we are going to write to file.** | Pass |

**4. Resource & Environment Needs**

**4.1. Testing Tools**

* Code Editor: VS Code
* Debugging: Chrome Developer Tools, Node.js console
* Version Control: GitHub

**4.2. Test Environment**

* OS: Windows
* JavaScript: ECMAScript 6+
* Node.js: Latest stable version

**5. Terms/Acronyms**

|  |  |
| --- | --- |
| **TERM/ACRONYM** | **DEFINITION** |
| API | Application Program Interface |
| AUT | Application Under Test |
| JS | JavaScript |
| FS | File System |

This test plan ensures that all string manipulations and file operations work correctly before submission. All test cases will be executed, and failures will be debugged and fixed accordingly.