Test Document for SLAIT

Secure Language Assembly Inspector Tool (SLAIT)

Team Members
Maria Linkins-Nielsen – mlinkinsniel2022@my.fit.edu
Michael Bratcher – mbratcher2021@my.fit.edu

Client & Faculty Advisor
Dr. Marius Silaghi – msilaghi@fit.edu

1. Introduction

This Test Plan describes how SLAIT will be tested to ensure that it meets the requirements defined in SLAIT's Software Requirements Specification (SRS). Testing focuses on functional correctness, error handling, usability, and security within the scope of Phase 1. The test plan covers:

- Code input and inspection management
- Code execution and state capture
- Visualization and analysis
- User interface behaviors
- Performance constraints

2. Test Cases by Requirement

2.1 Code Input and Inspection Management

Requirement 3.1.1.1 – Upload Code

- **Test Case 1 (Normal):** Upload a valid .asm file containing simple MASM code. Expect: code displayed in editor with correct formatting

- **Test Case 2 (Edge):** Upload an empty .asm file. Expect: editor shows empty code with no errors.
- **Test Case 3 (Error):** Upload a non-.asm file (e.g., .txt, .exe). Expect: rejection with descriptive error message.

Requirement 3.1.1.2 – View Code

- **Test Case 4 (Normal):** Verify syntax highlighting and readability when valid MASM code is loaded.
- **Test Case 5 (Edge):** Load a very large MASM file (>1000 lines). Expect: editor performance remains acceptable (<1s render time).

Requirement 3.1.1.3 – Create Inspection

- **Test Case 6 (Normal):** User selects a valid line and chooses `Register` and `Flag`. Expect: inspection marker placed, registers tracked.
- **Test Case 7 (Error):** User selects a non-existent line number. Expect: rejection with clear error.
- **Test Case 8 (Unusual):** User selects all registers/flags at multiple consecutive lines. Expect: inspections created, no performance degradation.

Requirement 3.1.1.4 – Delete Inspection

- **Test Case 9:** Add and then remove an inspection. Expect: marker disappears, execution reflects updated inspections.

2.2 Code Execution and Data Capture

Requirement 3.1.2.1 – Execute Code

- **Test Case 10 (Normal):** Compile and run valid MASM code that halts cleanly. Expect: successful execution with results returned.

- **Test Case 11 (Error):** Compile invalid MASM code. Expect: compiler error displayed in error panel.
- **Test Case 12 (Unusual):** Infinite loop in code. Expect: execution halted by timeout, error message displayed.

Requirement 3.1.2.2 – Capture State

- **Test Case 13 (Normal):** Capture the register EAX at the designated line. Expect: correct value shown.
- **Test Case 14 (Unusual):** Capture all registers/flags at multiple inspections. Expect: full state recorded with no corruption.

Requirement 3.1.2.3 – Return Results

- **Test Case 15:** Verify results are formatted as structured lists (registers as columns, rows for inspection points).

Requirement 3.1.2.4 – Handle Errors

- **Test Case 16 (Normal):** Division by zero runtime error. Expect: descriptive runtime error message.
- **Test Case 17 (Unusual):** Invalid memory access. Expect: descriptive sandbox error without crashing SLAIT.

2.3 Visualization and Analysis

Requirement 3.1.3.1 – Display Results

- **Test Case 18:** Verify captured results shown in timeline view, ordered by execution.

Requirement 3.1.3.2 – Highlight Differences

- **Test Case 19:** Run code with changing EAX. Expect: differences highlighted between snapshots.

Requirement 3.1.3.3 – Support Multiple Runs

- **Test Case 20:** Execute same code twice, compare results. Expect: user can view side-by-side.
- **Test Case 21 (Unusual):** Compare two runs with different input values. Expect: differences displayed clearly.

2.4 Interface Requirements

Requirement 3.2.1 – Graphical User Interface

- **Test Case 22:** Verify syntax highlighting visible.
- **Test Case 23:** Verify inspections visually marked in editor.
- **Test Case 24:** Verify errors shown in a distinct error panel with red markers.

2.5 Performance Requirements

Requirement 3.3.1 – Responsiveness

- **Test Case 25:** Modify code and check "Run" button toggles within 100 ms.

Requirement 3.3.2 – Resource Management

- **Test Case 26 (Unusual):** Submit code with recursive infinite calls. Expect: sandbox timeout kills process without exceeding memory limits.

---

3. Test Strategy

- **Unit Testing:** Backend functions for compilation, execution, and error reporting will be tested independently.
- **Integration Testing:** Frontend-to-backend communication will be tested with real Docker containers.
- **System Testing:** End-to-end user workflows (uploading code, marking inspections, running, viewing results) will be validated.
- **Stress Testing:** Large inputs, infinite loops, and excessive inspection points will be tested to ensure SLAIT remains stable.

---

4. Conclusion

This Test Plan provides a structured approach to verifying SLAIT's required functionality. By testing both expected and unusual cases, we ensure the tool is secure, reliable, and usable for students and faculty.