

CTRL 5

WASM AS OS

TEST PLAN

Date: 10/18/25

Table of Contents

INTRODUCTION.....	3
1.1 OBJECTIVES.....	3
1.2 TEAM MEMBERS.....	3
2 SCOPE.....	3
3 ASSUMPTIONS / RISKS.....	4
3.1 ASSUMPTIONS.....	4
3.2 RISKS.....	4
4 TEST APPROACH.....	4
4.1 TEST AUTOMATION.....	4
5 TEST ENVIRONMENT.....	5
6 MILESTONES / DELIVERABLES.....	5
6.1 TEST SCHEDULE.....	5
6.2 DELIVERABLES.....	5

Introduction

The Test Plan has been created to communicate the test approach to team members. It includes the objectives, scope, schedule, risks and approach. This document will clearly identify what the test deliverables will be and what is deemed in and out of scope.

1.1 Objectives

The goal of our project is to build a webassembly runtime from the ground up tailored to the needs of our client. The runtime shall be accessible via a command line interface and/or a web interface. The runtime will feature scheduling so multiple wasm files may be run simultaneously with resources being allocated at the clients preference. Wasm as Os will also supply the user with runtime reports so they can better understand the behavior of their webassembly modules.

Sprint 1 of the project will deliver skeleton code featuring a basic command line menu, the first iteration of our design document containing class and ER diagrams as well as wire frames of what our web ui will look like, and finally this test document.

1.2 Team Members

Resource Name	Role (<i>examples are given below</i>)
Shivam Pandi	Frontend Developer
Sritan Gangidi	Frontend Developer
Biraj Sharma	Tester
Ololade Awoyemi	Rust Backend Developer
Benjamin Wilson	Rust Backend Developer

2 Scope

The initial sprint will include ‘must have’ requirements. These and any other requirements that get included must all be tested.

The following sections indicate what is tested during each sprint. The scope of testing is determined at the beginning of the current sprint.

At the end of Sprint 1, a user must be able to:

1. Validate that both the CLI and WebUI deliver the stated user stories with acceptable quality
2. Establish a repeatable test process aligned with agile ceremonies
3. Achieve traceability between requirements(user stories),test cases,defects and builds
4. Derisk the release by early automation with integration

At the end of Sprint 2, a user must be able to:

1. Load and run a wasm file
2. Receive runtime metrics

Ensure Wasm as OS doesn't overwrite or manipulate unrelated user files

At the end of Sprint 3, a user must be able to:

1. Load and run multiple wasm files
2. Schedule/prioritize resources to wasm files

Ensure Wasm as OS scheduler doesn't use all of users computer resources when scheduling

Assumptions / Risks

2.1 Assumptions

We have or can attain the knowledge needed to develop a web assembly from scratch.

We have enough time to learn Rust for our project.

It is possible to save a wasm module's state.

Rust has cargo containers adequate for our project.

We have an accurate understanding of the project's requirements.

2.2 Risks

The following risks have been identified and the appropriate action identified to mitigate their impact on the project. The impact (or severity) of the risk is based on how the project would be affected if the risk was triggered. The trigger is what milestone or event would cause the risk to become an issue to be dealt with.

#	Risk	Impact	Trigger	Mitigation Plan
1	Code Loss	Low to High	Code is lost or corrupted before uploading to Github	Commit code to github often.
2	Overly Expanded Scope	Medium	Too many extra features are added leading to failure to release adequate product	Complete required features first and move on to other features later.
3	Project Doesn't Meet Clients Expectations	High	The project functions as intended but the UI or elements of the project don't match what the client expects.	Meet often with the client to demonstrate the project and receive feedback.
4	Disorganized Workflow	Medium	Areas of work are improperly defined leading to wasted time and effort.	Communicating and updating team members on work being done and future work.

3 Test Approach

The project is using an agile approach, with 3-week sprints. *Mention how you will conduct testing during the sprint in terms of the techniques you plan to do and when. Add a new subsection for each sprint.*

Sprint 1 : Functional testing (CLI, module load, ABI calls).

Sprint 2: UI integration testing, scheduling behavior, basic fuzzing.

Sprint 3: Stress testing, performance analysis, security validation.

3.1 Test Automation

Discuss the role of automated testing and how you plan to conduct any (including tools).

*-Command Line Interface (CLI) test harness for loading and running .wasm modules.
Browser automation using Playwright for UI validation (future sprints). Continuous Integration (CI) pipeline for automated test execution after commits*

3.2 Test Cases (Black Box)

3.2.1 Feature 1 (Name this based on the name in your Req doc)

Have a table for the test cases needed to test the User Story

Test Case ID	Description	Requirements Trace	Directions	Expected Output
TC-001	Load WASM module	US-001	1. Load <code>hello.wasm</code>	Successful module execution
TC-002	Run basic ABI call	US-002	Call <code>log("hello")</code> from module	Output "hello" in CLI logs

3.2.2 Feature n (Name this based on the name in your Req doc)

Have a table for the test cases needed to test the User Story. A sample is below

Test Case ID	Description	Requirements Trace	Directions	Expected Output
TC-101	Spawn a single process	US-005	From CLI, <code>spawn test_module.wasm.</code>	Process appears in task list; runs completion: reported exit code
TC-102	Spawn multiple process	US-005	Spawn 3 worker modules concurrently	All 3 processes run concurrently; no crashes; scheduler interleaves tasks
TC-103	Suspend and resume a process	US-005	1. Start process 2. Suspend via CLI 3. Resume	Processes stops when suspended and resumes from same state after resume

3.3 Test Cases (White Box)

3.3.1..n Name by Feature (like above)

Like section 3.2, you will add to the table as you progress through each sprint.

Test Case ID	Description	Directions/Goals	Expected Output
TC-1001	Scheduler fairness check	Instrument scheduler to measure run time distribution	No single process starves; fairness within expected bounds
TC-1002	IPC race condition detection	Rapidly send/receive messages between two processes	No message loss, no deadlocks; ordering preserved if required
TC-1003	Memory bookkeeping integrity	Allocate/free in loop across processes	No leaked memory after repeated allocations/frees
TC-1004	Ensure data validation is working properly.	Follow TCB-006 only with improper input by leaving fields blank. Also attempt to use improperly formatted email and phone numbers, i.e. The last name for both fields.	System should check to make sure that fields are filled in and warned about otherwise. If there is an improper format for the email and phone number, that should be marked as well.
TC-1005	Attempt to use the system when the database is down.	Access any page without the database running. Check to make sure the error seems reasonable.	Although not a requirement, if the database connection cannot be made, a reasonable error message would be nice.
TC-1006	Attempt to use the system when JavaScript is disabled in the user's browser.	Can you still make a reservation with blanks in the field if you disable JavaScript? See how verbose the error handling is.	Since JavaScript does the error handling, not the code, disabling JavaScript should allow bad input.
TC-1007	Attempt to use apostrophes in text fields whose values get used in SQL statements.	Follow TCB-006 but use O'Conner as the last name. Checking to make sure data checking includes escaping characters.	SQL uses the apostrophe character as a special character, so does the system escape the apostrophe character if it appears in a name to prevent a bad SQL call.

TC-1008	Bounds checking on the Erlang implementation.	Use the Erlang page and try using a blank field or negative numbers.	Should fail since there is no error handling to round or check the input to the Erlang implementation.
TC-1009	Add an incentive directly to the database and ensure it is displayed.	<ol style="list-style-type: none"> 1. Open a web browser and go to http://rrs.se.rit.edu/rrs/sql.jsp. 2. Enter the SQL statement "INSERT INTO incentives (restaurant_id, start_time, end_time, description) VALUES (0, sysdate, TO_DATE('MM/DD/YYYY', '12/31/2002'), 'TC-1009 Incentive
<I>This is a new incentive!</I>')" 3. Execute TCB-001. 	The incentive just entered should be displayed on the page.

4 Test Environment

Wasm as OS will primarily be tested on developer personal computers. The tests will occur in an isolated virtual environment such as a Windows sandbox or a virtual machine like VMWare to avoid harm to personal devices.

5 Test Schedule

Task Name (<i>sample is below, focus on sprint 1 to start</i>)	Start	Finish	Effort	Comments
<i>Test Planning</i>	Oct 20	Oct 21	1 day	Align on stories/risk/coverage
<i>Review Requirements documents</i>	Oct 20	Oct 22	2 days	Add ID's and acceptance criteria
<i>Create initial test estimates</i>	Oct 21	Oct 22	1 day	Per story/test type
<i>Learn new test resources</i>	Oct 21	Oct 24	3 days	Setup
<i>First deploy to QA test environment</i>	21	24	3 days	Automated deploy
<i>Functional testing – Sprint 1</i>	22	23	1 day	Rolling as features land
<i>Iteration 2 deploy to QA test environment</i>	22	23	1 day	Cross features path
<i>Functional testing – Sprint 2</i>	22	24	2 days	CI regression pack
<i>System testing</i>	Oct 22	Oct 25	3 days	
<i>Regression testing</i>	Oct 24	Oct 28	4 days	Exit criteria check
<i>Usability Testing</i>	Oct 23	Oct 27	3 days	Release candidate
<i>Resolution of final defects and final build testing</i>	Oct 24	Oct 28	3 days	Select build
<i>Deploy to Staging environment</i>	Oct 24	Oct 26	2 days	Initial risks; review
<i>Performance testing</i>	Oct 28	Oct 29	1 day	Walkthrough plan/cases; capture feedback
<i>Release to Production</i>	Nov 8	Nov 9	1 day	Summarize artifacts; exit criteria for research sprint