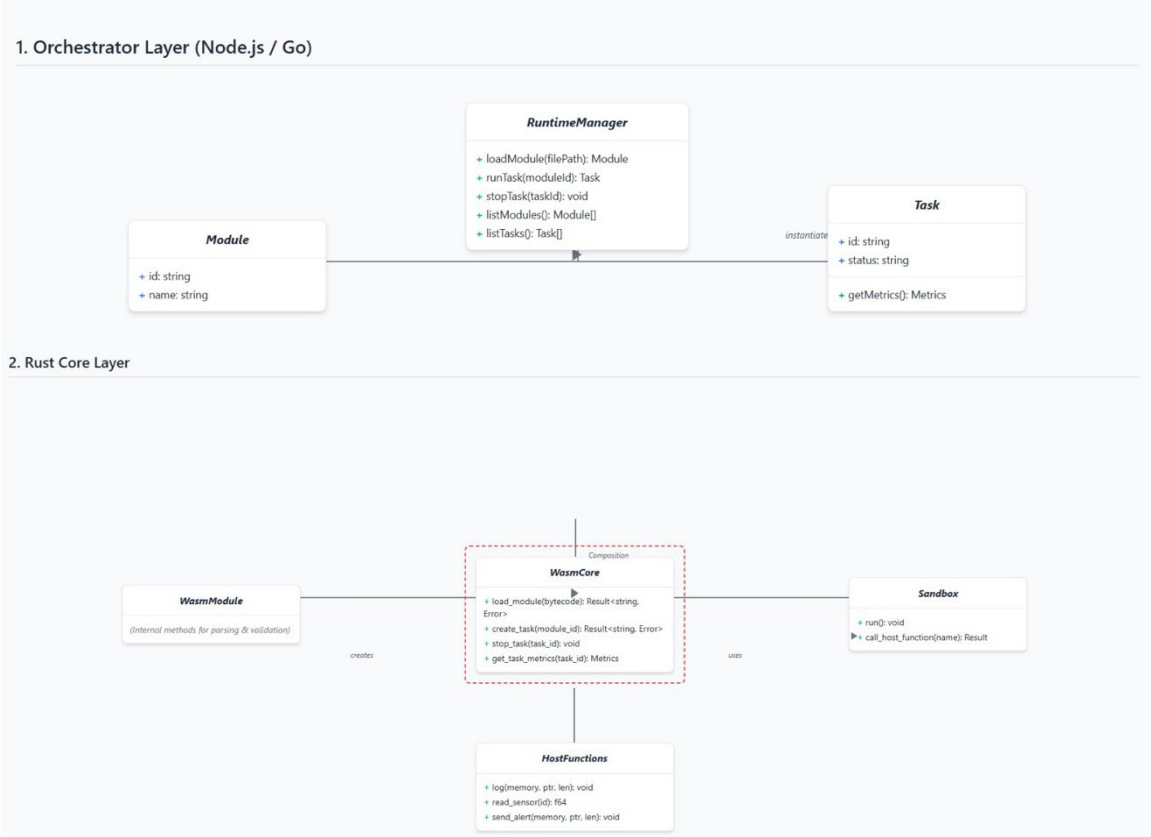
Product Design

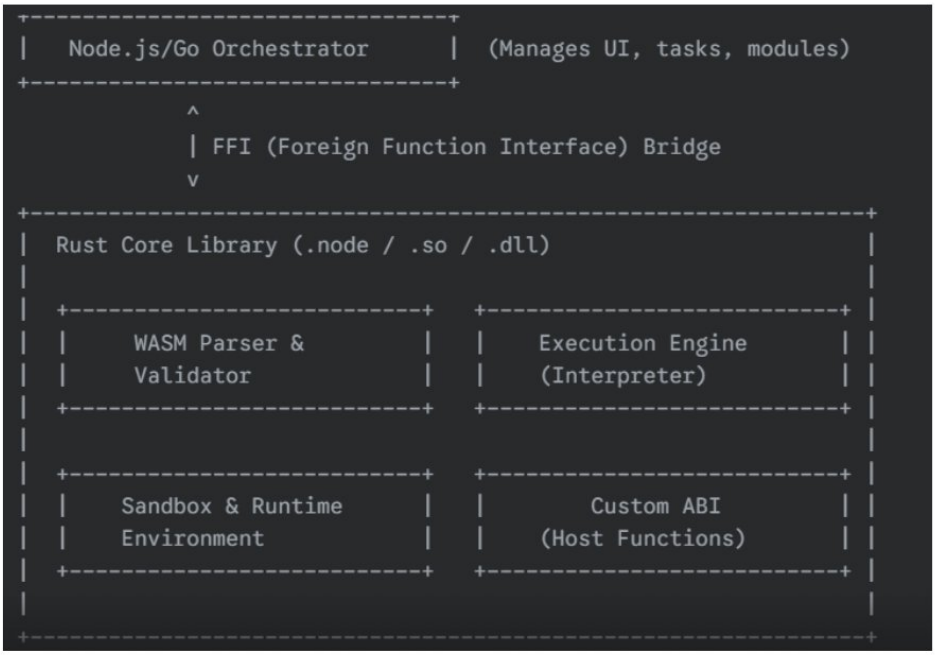
Team Ololade Awoyemi, Shivam Sakthivel Pandi, Biraj Sharma, Benjamin Wilson, Sritan Reddy Gangidi

This document presents the product design for our capstone project being put together by Team CTRL 5. It describes the overall system depiction, design rationale, and technical choices that help in developing our solution. The purpose of this document is to indicate that the designs are well-structured, traceable to the requirements and open to changes as we go through development.
Our project involves the building of a secure and modular WebAssembly runtime with a Rust-based back end and a React-based front-end. The design is concentrated upon performance, scalability, and safety so that untrusted WASM modules may execute themselves in an environment characterized by control. The present introduction lays the groundwork for the class diagrams, ER diagrams, UI wireframes and rationales which follow in this document. As development progresses, this document too will progress in describing the refinements and improvements of our system.

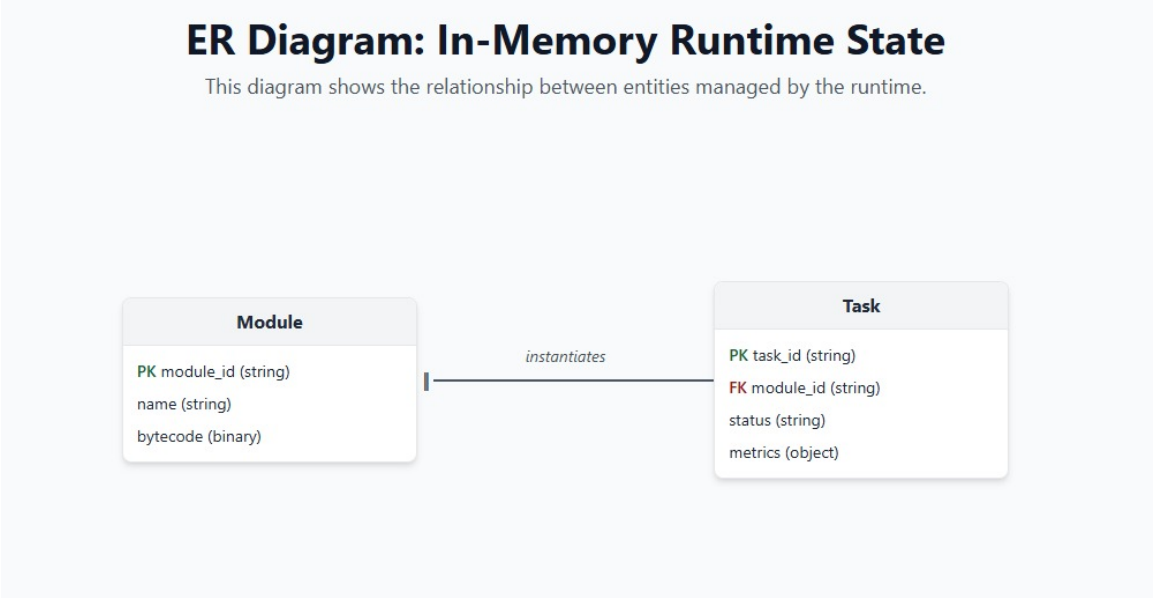
Revision Number	*Revision Date*	*Summary of Changes*	*Author(s)*
0.1	*10/08/2025*	*Initial design and diagrams*	*Shivam Sakthtivel Pandi ,Sritan Reddy Gangidi ,Benjimin wilson, Ololade Awoyemi, Biraj sharma*
0.2	***10/18/2025***	***Updated Class diagram, removed the template text.***	*Shivam Sakthtivel Pandi ,Sritan Reddy Gangidi ,Benjimin wilson, Ololade Awoyemi, Biraj sharma*

Class Diagram(s)

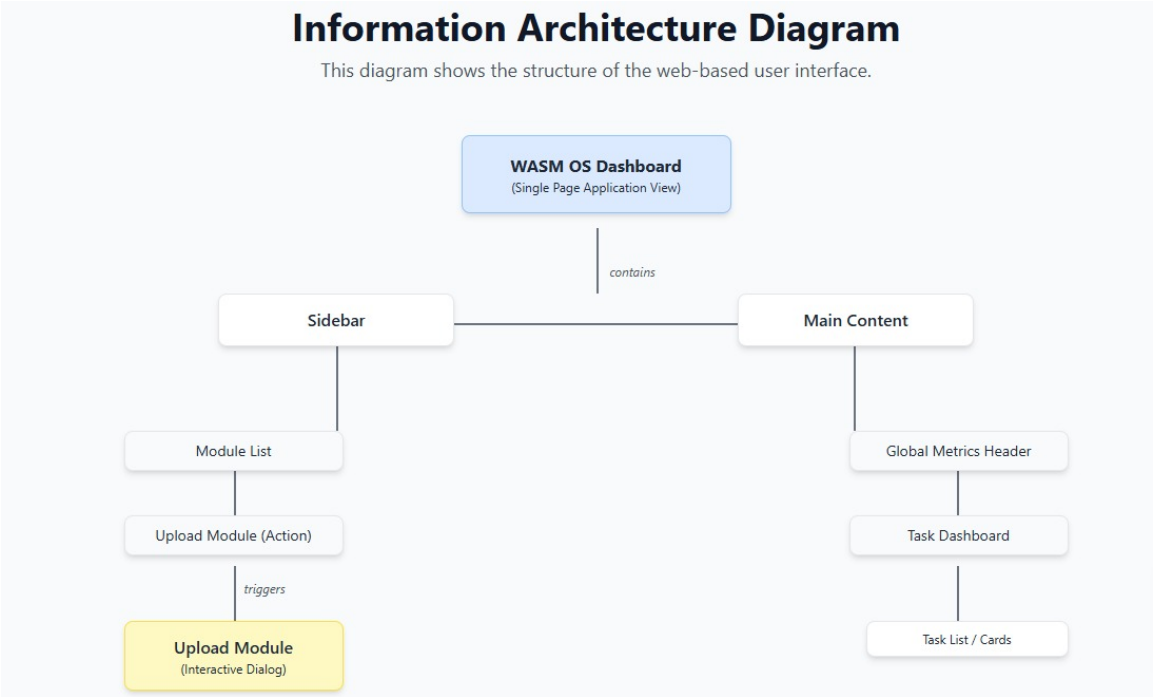




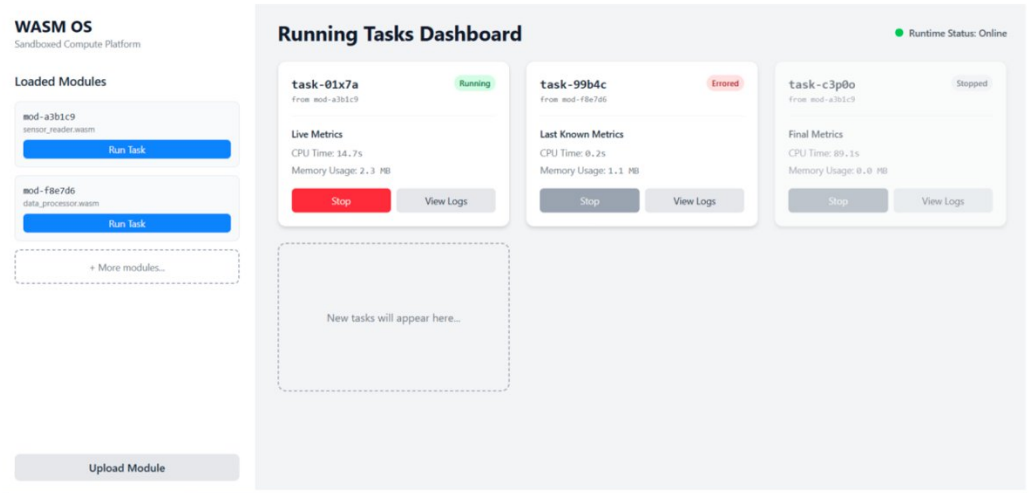
ER Diagram(s)

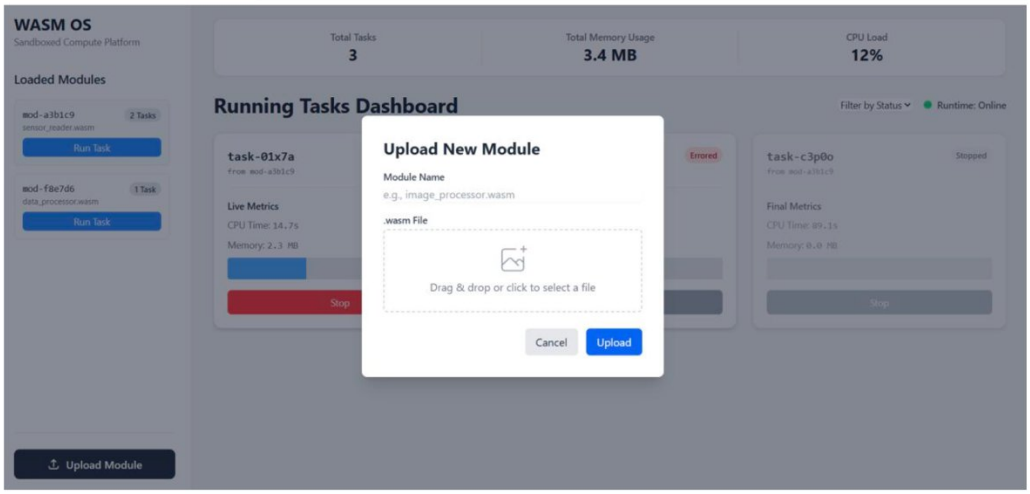


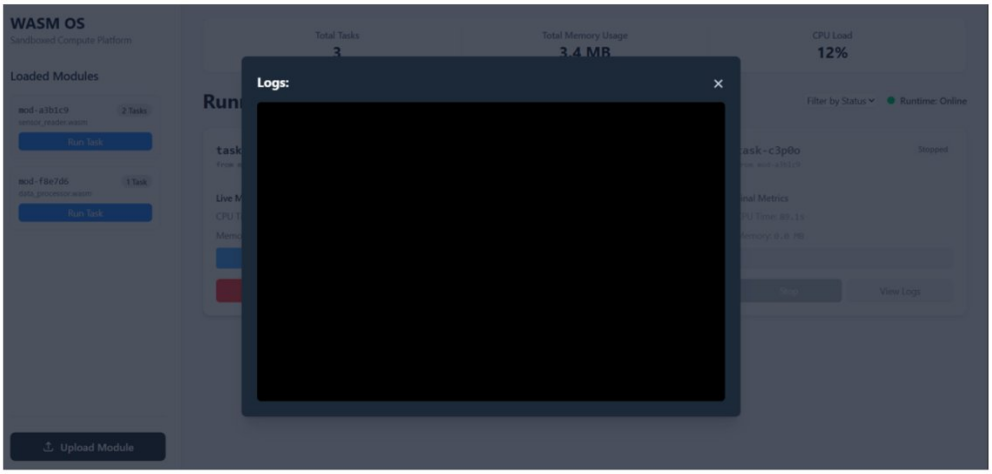
Information Architecture Diagram



User Interface Wireframe(s)/Screenshot(s)







Design Summary

Our design focuses on building a secure and modular WebAssembly runtime using Rust for the backend and React for the web interface. The runtime acts as the core execution environment, responsible for parsing, validating, and executing .wasm modules within strict sandbox limits. It manages tasks, memory, and system calls through a custom ABI to ensure safe and isolated execution.
A lightweight command-line interface is implemented first for simplicity and testing, while the web
UI (developed in React) provides a more user-friendly dashboard. The dashboard allows users to upload, start, pause, and stop modules, and view real-time runtime metrics such as memory usage,
operation speed, and active tasks.
The system's structure includes:
● A Rust core library handling the execution engine, sandbox environment, and task scheduling.
● An optional Node.js/Go orchestrator layer that bridges the Rust core with the web frontend.
● A React-based dashboard that serves as the main point of interaction for users.
Together, these layers form a secure, extensible platform that simulates a lightweight, sandboxed operating system capable of running untrusted WASM code safely and efficiently.

Design Rationale

What language should be used for our backend (WebAssembly Engine): Rust/Go -> Rust
○ Rust: Memory safe and efficient, large number of resources available for learning but difficult to learn, useful cargo containers for our project, existing webassembly engines to draw inspiration from.
○ Go: Easy to learn, less resources to learn from, less libraries for webassembly, built in garbage collector adds overhead, Memory safe.
● Command line or web interface: -> Implement both, cmd interface first Web UI would be nice but our client thinks a command line interface is acceptable.
○ Command line: Simple to implement, would only require us to code in Rust, Not very impressive, our client is ok with this
○ Web UI: Can better display data, more resource consumptive, easier to use, shows extra effort.

● What language should be used for the web interface
○ Rust: Has packages for ui use, would allow us to code the project solely in Rust but may not be the most well suited for our UI since it's more of a backend language and lacks features useful for UI development.
○ Go: Would allow us to study multiple languages but could complicate the development process by increasing the learning load. Also more of a backend language which would make the UI harder to implement.
○ React: Good choice for our web UI, made for developing web UI's. Would work

well as the frontend to our Rust backend.