

C.O.S. Project Documentation

Author: Joseph Wangai Mwaniki

# Introduction

C.O.S. (Customized Operating System) is a visionary modular OS inspired by Linux, designed to adapt dynamically to a user’s hardware and needs. It aims to balance flexibility, performance, and user empowerment through sandboxed applications, AI assistance, and adaptive computing.

# Vision and Goals

- Create an adaptable and modular OS environment.  
- Ensure secure and controlled AI integration for educational and medical uses.  
- Provide a lightweight, fast, and sleek UI.  
- Make all applications sandboxed for better security and file management.  
- Ensure wide compatibility through emulated environments.

# Development Philosophy

The system encourages community and developer contributions while remaining curated to ensure safety, creativity, and control. An AI guide is included to assist users in mastering advanced features without being overwhelmed.

# Technical Summary

- Modular architecture: 64-bit host, adaptable down to 16-bit.  
- Sandboxed app storage: Each app writes to isolated virtual space.  
- Superuser required for USB access to the main file system.  
- Environment emulation (Windows/Linux/macOS) for compatibility without copying proprietary behavior.  
- AI-enhanced UI for learning curves, alerts for heavy resource usage, and graceful hibernation of background apps.  
- Web apps rendered securely via custom Java container (preventing direct link exposure).  
- Uses TLS over SSL for better performance and security.  
- Password recovery through linked cloud account.

# Technologies & Stack

- Front-end: HTML, CSS, JavaScript  
- Back-end: Python, PHP, Node.js  
- Security: TLS, sandboxing, admin/sudo controls  
- Server: Apache  
- AI: Embedded AI guidance within the OS

# Challenges & Mitigation

- Complexity of managing hardware resources: Use of modular design and dynamic scaling.  
- Risk of project overload: Divide into phases, use community support, maintain focused core.  
- Licensing concerns: Use of emulation and not direct API/DLL copying helps avoid infringement.  
- Maintaining performance with sandboxing: Optimization with virtualization limits and resource alerts.

# Future Outlook & Timeline

- Short-term (1–2 years): Develop core modules, basic UI, and sandboxing system.  
- Mid-term (3–5 years): Expand AI integration, app emulation layers, compatibility modes.  
- Long-term (5–10 years): Global rollout, community marketplace, potential hardware integration.