# WALL-E Control System Project Summary

Project Owner: Mike Owen

## Overview

This project involves building a full-scale 1:1 WALL-E animatronic robot with a production-grade control system. The system consists of a PyQt6 front-end running on a Steam Deck, a Raspberry Pi 5 backend with hardware drivers, and a Windows stub backend for UI testing. The Raspberry Pi also acts as a Wi-Fi hotspot (SSID: wall-e, Password: EVEROCKS2025).

## Hardware Components

- Raspberry Pi 5 (main controller)  
- Sabertooth 2x60 motor driver (tracks)  
- NPC2122 motors (track drive)  
- ASME-05B actuators (shoulder up/down, neck up/down)  
- NEMA 32 stepper motor + TB6600 driver (arm gantry)  
- Pololu Maestro 18-channel x2 (arm and neck servos)  
- ESP32-CAM (head camera with MJPEG stream)  
- ACS758LCB-100B current sensor  
- ADS1115 ADC (voltage/current sensing)  
- DFPlayer Mini (audio playback)

## Software Architecture

- Front End: PyQt6 application with seven screens, WebSocket client to Raspberry Pi  
- Back End: Asyncio WebSocket server, serial drivers for Maestro and DFPlayer, scene engine, telemetry loop  
- Stub Backend: Pure Python WebSocket server for UI testing with simulated telemetry and command logging

## Front End Screens

1. Health  
2. ServoConfig  
3. CameraFeed  
4. ControllerConfig  
5. ExpressionEditor  
6. SceneEditor  
7. SceneDashboard

## Health Screen Details

- Resolution: 1280x800  
- Theme: WALL-E inspired dark mode with glowing accents  
- Top Bar: Voltage (left), 'Health' title (center), Wi-Fi strength (right)  
- Graph: Dual-axis plot with voltage (0–16V) and current (0–120A)  
- System Stats: CPU usage, memory usage, Pi temperature  
- MJPEG Stream Stats: FPS, resolution, latency  
- DFPlayer: Serial connection status, SD card file count  
- Maestro 1 & 2: Connection status, channel count  
- Failsafe Button: Stops tracks, neutralizes servos, interrupts scenes  
- Bottom Navigation Bar: Icon-based navigation to all screens

## Back End Features

- Asyncio WebSocket server on port 8765  
- Serial drivers for Maestro and DFPlayer  
- Scene loader with JSON schema validation  
- Telemetry loop every 5 seconds  
- Idle behavior engine and failsafe logic  
- Voltage and current sensing via ADS1115 and ACS758  
- Systemd service for auto-start

## Stub Backend Features

- WebSocket server on localhost:8765  
- Emits fake telemetry every 5 seconds  
- Simulates MJPEG stream stats  
- Simulates DFPlayer and Maestro health metrics  
- Responds to play\_scene and failsafe commands  
- Logs all incoming commands

## Development Progress

- Health screen fully implemented with telemetry and component metrics  
- Front-end and stub backend code generated and tested  
- Ready to proceed with ServoConfig screen and remaining UI components

## Suggested Prompt to Resume in Copilot

I’m continuing my full-scale WALL-E animatronic project. I’ve already built the Health screen in PyQt6 with telemetry, DFPlayer and Maestro metrics, and a stub backend. Please refer to the project summary document and help me build the next screen: ServoConfig. Maintain the same WALL-E inspired dark theme and architecture.