Full Stack Candidate Project: "Access Control Manager & Real-time Dashboard" - 5-Day Challenge

Project Goal (Revised)

Create a robust web application that simulates a real-time Access Control System (ACS) manager. The project must demonstrate proficiency in building a WebSocket server to manage the state and push status of virtual IoT devices (simulated smart doors), an HTTP server to register access logs, and a dynamic frontend for commanding and monitoring these devices.

The candidate must implement **Access Control Logic** on the backend, where the server is the single source of truth for the device status.

Core Features

Frontend (The Dashboard) (Simplified for Backend Focus)

1. **Device Status View (Minimal):** A simple interface, which can be a basic HTML table or list, displaying the current information for **at least two (2)** simulated "Smart Door" devices. Aesthetic complexity is secondary to functionality. For each device, it must clearly show:
    - Device ID (e.g., `DOOR-001`) and Location.
    - Current **Physical Status** (Simulated: `Open` / `Closed`).
    - Current **Lock State** (Simulated: `Locked` / `Unlocked`).
2. **Command Interface:** For each device listing, include basic buttons or toggles to send control commands via the WebSocket connection:
    - `Open Door` / `Close Door` controls.
    - `Lock Device` / `Unlock Device` controls.
3. **Real-time Event Log:** A basic, scrolling text area or list that displays incoming access log events (door openings, closing, failure acces)
4. **WebSocket Client:** The frontend must establish a persistent connection to the WebSocket server to receive and send data.

Simulation (MANDATORY)

1. **Physical Device Integration & Simulation:** The candidate **must** demonstrate the complete logic required to integrate a physical device (e.g., ESP32, Arduino, or Raspberry Pi) for at least one door.
    - **Logic Requirement:** The implementation must show how a command sent from the dashboard via the WebSocket server is received, processed by the server, and then sent to the simulated device (ESP32/Arduino context), causing a state change (e.g., `Open Door` command -> server sends control message -> **simulated device logic changes state to `Open`**).

Backend (The Servers)

1. **Access Control State Manager:** The server must maintain the current, authoritative state of **at least two (2)** simulated devices (Physical Status and Lock State) in memory.
2. **WebSocket Server (Bi-directional):**
    - **Status Broadcast:** Pushes real-time status updates of all devices to all connected clients (e.g., when a door state changes from `Locked` to `Unlocked`).
    - **Command Listener:** Receives control commands from the frontend (e.g., `{"command": "LOCK", "device_id": "DOOR-001"}`). The server must execute the underlying access control logic before confirming the state change.
3. **HTTP Server (Audit & Management Endpoints):** A simple server to handle initial asset serving and core REST endpoints for management and audit purposes:
    - `GET /api/devices/status`: Endpoint to fetch the initial list and current state of all registered devices (connected devices).
    - `GET /api/access_logs`: Endpoint to retrieve the stored access history/log (in memory).
    - `POST /api/access_log`: **MANDATORY SIMULATION.** This endpoint simulates a device (like an ESP32) sending an access attempt. to open or close a door. The payload should include `device_id` and `user_card_id`. The server must: a. Log the attempt (in memory). b. Apply logic (e.g., "If door is locked and the `user_card_id` is *not* 'admin', log a `DENIED` event; otherwise, log `GRANTED`, change the door status to `OPEN`, and broadcast the change via WebSocket").

Technology Stack (Suggested)

Area	Requirement / Suggestion
Frontend	React, Angular, or Vue.js. **Styling:** Any standard CSS approach is acceptable; responsiveness is required, but complex styling is not.

Backend	Node.js (with Express/ws) or Python (with FastAPI/Flask and websockets library).
Data Source/Sim	All data (Device States, User List, Access Logs) must be managed **in-memory** for the duration of the server process.

User Stories

- **As a Security Operator,** I want to see the real-time status (Locked/Unlocked, Open/Closed) of all monitored doors on a single screen.
- **As an Administrator,** I want to be able to immediately send a command from the dashboard to remotely lock or unlock any door.
- **As an Audit System,** I want to see a log of all simulated access attempts (Granted/Denied) as they occur on the dashboard.
- **As a Developer,** I want the server to manage the device states and ensure only valid state transitions occur (e.g., cannot open a door while it is locked).

Bonus Features (Proposal & Explanation Required)

The candidate should propose and briefly explain at least two enhancements to the application, focusing on scalability, security, or robustness (e.g., persistence, advanced access rules, simulated user authentication).

Deployment

The candidate should be able to explain how to deploy the application to a cloud provider (e.g., Vercel, Netlify, GCP or AWS).