# Repport: RFID Access Control System

**GitHub** github.com/Silent0Wings

## Contents



## 0.1 Project Context and Overview

The project is an IoT access control system built around an ESP32 TTGO LoRa32 node with an MFRC522 RFID reader, SSD1306 OLED display, RGB status LED, piezo buzzer, capacitive touch button, and a Node.js backend with Excel-based persistence and a web dashboard. The device authenticates RFID cards against a central server, logs all events, supports user registration and deletion, and exposes a small local web UI with real-time status and admin tools.

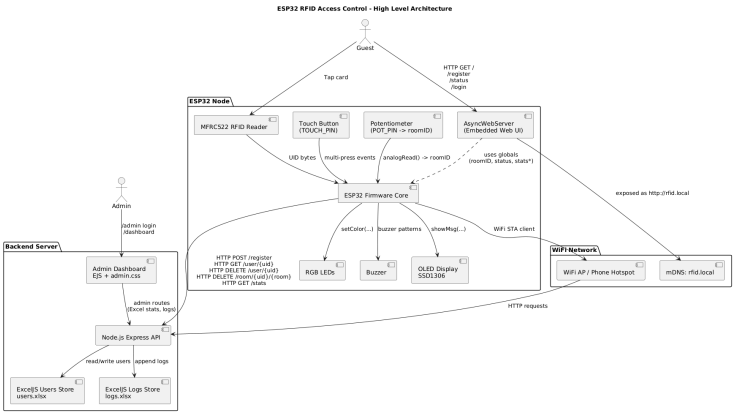



Figure 1: High-level architecture showing ESP32 node, sensors, actuators, WiFi link, and backend.

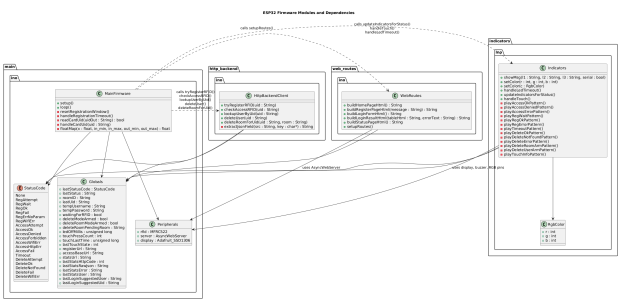



Figure 2: Firmware module structure: core loop, backend HTTP, indicators, and web routes.

## 0.2 System Architecture and Technology Stack

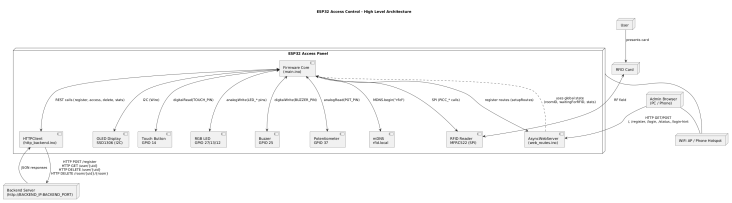


Figure 3: Access control high-level architecture focused on door node, network, and backend roles.

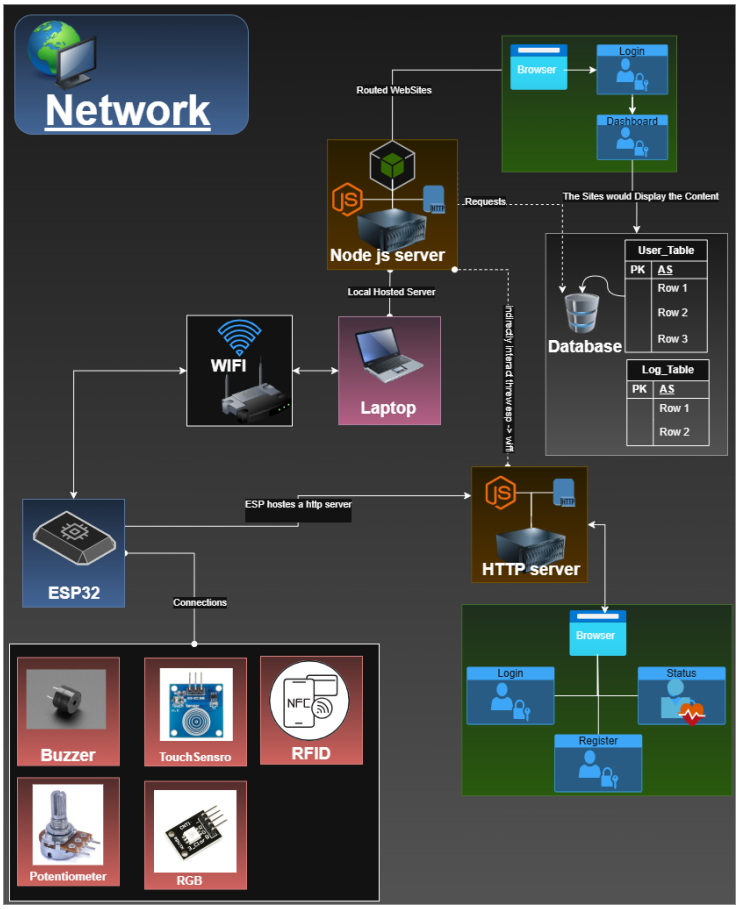



Figure 4: Technology stack: ESP32 toolchain, Node.js backend, Excel storage, and browser UI.

## 0.3 Presentation, Technical Depth, and Organization

The final presentation and video are structured in five short blocks:

- **Problem and Requirements**: Why door access needs auditable logs, per-room policies, and easy updates for staff.
- **System Architecture**: Slides show the ESP32 node, sensors and actuators, WiFi link, backend server, Excel files, and browser dashboard. A layered diagram makes the separation between hardware, firmware, API, and UI explicit.
- **Firmware Design**: The split into modules for RFID handling, indicators, HTTP backend calls, and web routes is explained. The flow from `loop()` to `readCardUid()`, `handleCardUid()`, and HTTP functions is highlighted.
- **Backend and Data**: The Node.js server, Excel-based storage (users and logs), and auto-refresh admin dashboard are shown with screenshots and a short code excerpt to demonstrate how each access attempt is appended to the log.

- **Non-technical Considerations**: Privacy (minimal fields per user), safety (relay isolation, fuse, error feedback on the OLED), maintainability (Excel for non-technical admins), and usability (color and sound patterns, simple web forms) are covered explicitly.

The video follows the same structure and alternates between slides, close-up shots of the hardware, and screen capture of the web dashboard. All files (slides, video, and this reflection) are consistently named and organized by project folder.

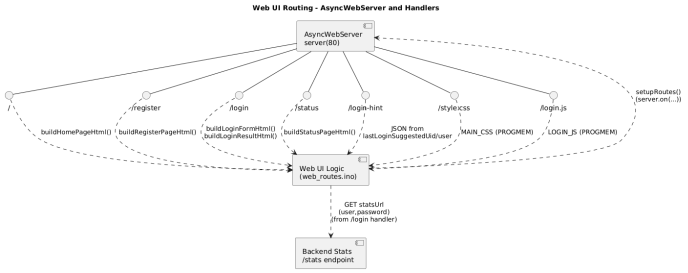



Figure 5: Web UI routing on the ESP32 using AsyncWebServer.

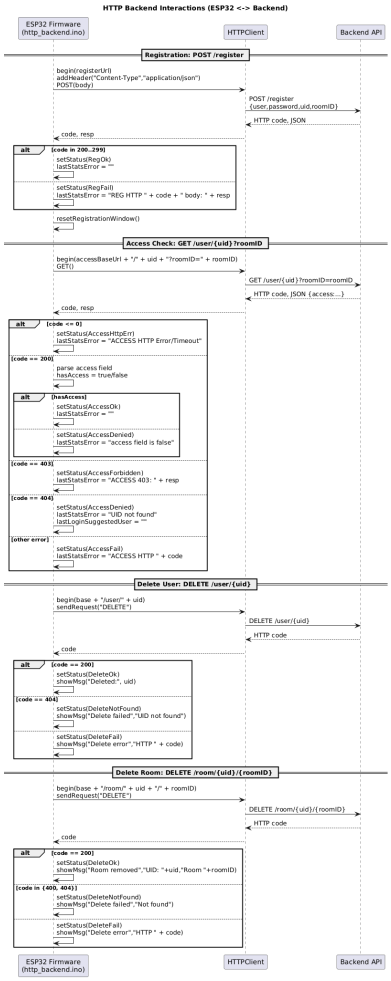



Figure 6: Backend interactions for registration, access control, and delete operations.

## 0.4 Difficulty and Creativity

The project goes beyond a trivial door opener and deliberately pushes several axes of difficulty:

- **Multi-module firmware**: Core loop, indicator module (LED and buzzer patterns), HTTP backend module, and web UI routing on the ESP32 are separated for clarity.
- **Touch multi-press interaction**: A debounced capacitive touch input with timing windows implements one-tap status, double-tap delete-room, and triple-tap delete-user modes, reducing hardware buttons at the cost of more logic.
- **Backend with Excel storage**: The Node.js server manages two Excel files (users and logs) as a simple database. The admin dashboard renders them through EJS templates with periodic refresh so non-technical staff can inspect access attempts.
- **Status-driven indicators**: LED colors and buzzer patterns encode access results, errors, timeouts, and delete modes to create a compact, intuitive interface at the door node.
- **mDNS and local web UI**: The ESP32 exposes a small web panel (home, register, login, status) discoverable as `rfid.local`, combining embedded and web programming.

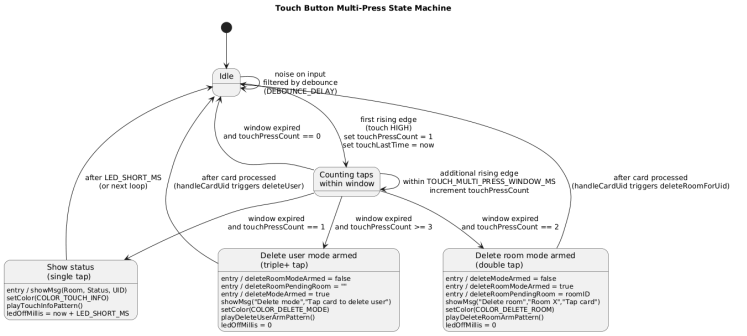



Figure 7: Touch multi-press timing and state transitions for delete-user and delete-room modes.

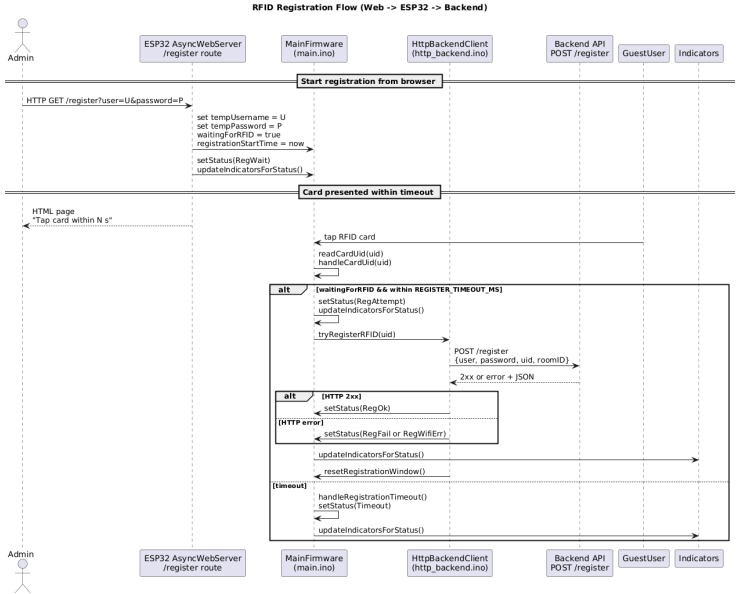



Figure 8: RFID registration flow between ESP32 and backend.

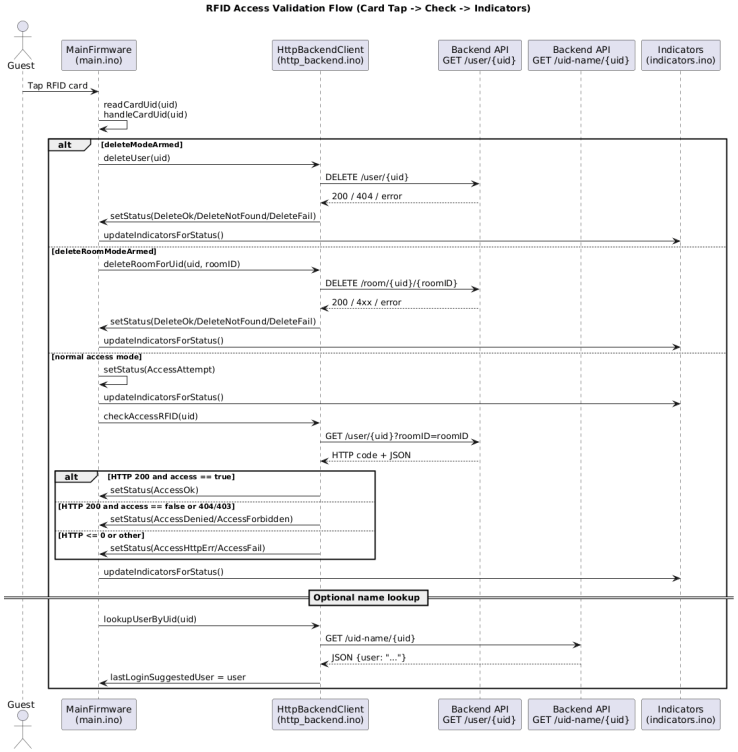



Figure 9: RFID access validation flow and decision stages.

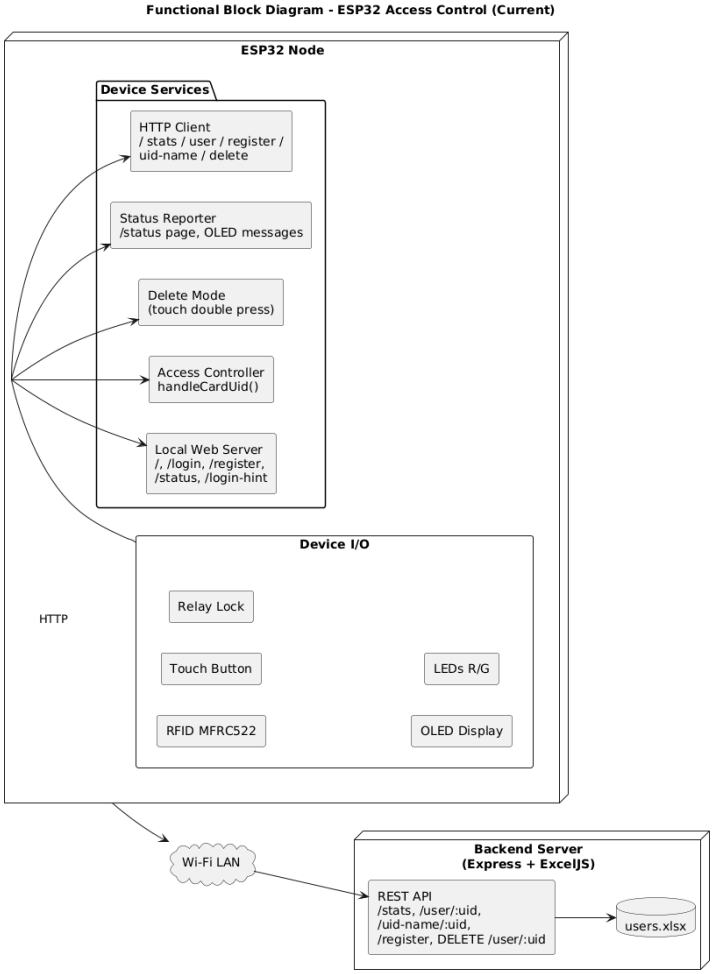



Figure 11: Functional block diagram showing main subsystems.

## 0.5 Technical Elements and Course Guidelines

Technical Elements vs. Project Guidelines

| Guideline Item | How the Project Satisfies It |
|---|---|
| Sensors and Actuators | MFRC522 RFID reader, touch sensor, RGB LED, piezo buzzer, relay output, and OLED display. |
| Microcontroller and IoT Connectivity | ESP32 TTGO LoRa32 running Arduino-based firmware, connected over WiFi to the backend. |
| Distributed Logic | Node handles registration, access decisions, logging, and admin views; ESP32 performs card reading, local state, and actuation. |
| Bidirectional Messaging | Device sends HTTP requests for registration, access, and delete operations. Server responses drive local indicators and OLED messages. |
| User Interface | On-device OLED and LED/buzzer patterns plus admin login page, card registration form, user view, and status page in a browser. |
| Data and Persistence | Users and events are stored in Excel files, used as a lightweight database with clear audit trails for debugging and analysis. |
| Security and Roles | Admin login on the dashboard, basic input validation on the ESP32, clear separation between card owners and administrators. |

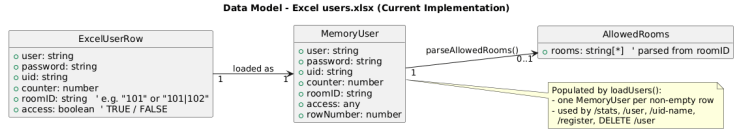



Figure 10: Data model containing users, logs, API responses, and device state.

## 0.6 Hardware Quality and GPIO Mapping

The hardware is selected to match the constraints of a door access scenario: ESP32 provides WiFi, enough GPIOs, and flash; MFRC522 is widely available and cheap; SSD1306 is readable in small spaces; the buzzer and RGB LED give compact feedback.

GPIO Mapping for ESP32 Access Node

| GPIO | Symbol | Function |
|---|---|---|
| 21 | SS_PIN | MFRC522 SDA / chip select |
| 22 | RST_PIN | MFRC522 reset |
| 27 | LED_RED | Status LED red channel |
| 13 | LED_GREEN | Status LED green channel |
| 12 | LED_BLUE | Status LED blue channel |
| 14 | TOUCH_PIN | Capacitive touch button input |
| 37 | POT_PIN | Room selector potentiometer (maps to room ID) |
| 25 | BUZZER_PIN | Piezo buzzer for audio feedback |
| 4 | OLED_SDA | SSD1306 I2C data line |
| 15 | OLED_SCL | SSD1306 I2C clock line |
| 16 | OLED_RST | SSD1306 reset pin |

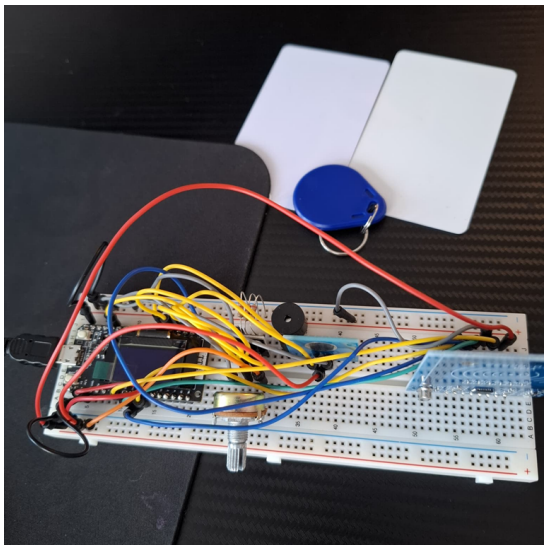
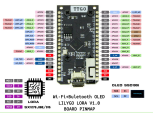
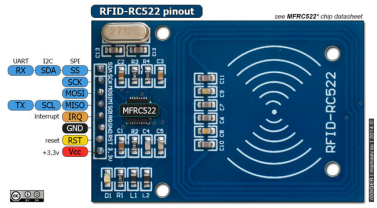


Figure 12: Circuit and wiring for the ESP32 access node: reader, OLED, indicators, and relay.

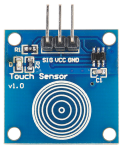
## 0.7 Hardware Components Used (Illustrations)



ESP32 TTGO LoRa32 development board (main controller with WiFi and OLED).



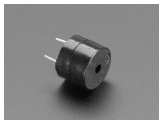

MFRC522 RFID reader module for card detection and UID reading.




Capacitive touch sensor for multi-press delete and status modes.

B10K potentiometer used as room selector mapped to room IDs.


Common-anode RGB LED providing color-coded status feedback.



Piezo buzzer for success, denial, error, and timeout patterns.

Resistors for current limiting and safe interfacing.

Figure 13: Photos of the main hardware components used in the ESP32 RFID access control prototype.

## 0.8 Quality of the Implementation

### Hardware fit and contextual factors

- The ESP32 board provides WiFi and enough flash to host both firmware logic and the async web server, which suits a campus setting where wired network is not always practical at doors.
- The components are powered from a stable 5 V supply with appropriate regulation for the board and peripherals. Relay driving is isolated from logic, in line with safety expectations for controlling a door strike or small lock.
- The potentiometer-based room selector makes it easy to reuse the same firmware for different rooms during the demo and future lab scenarios.

### Firmware and backend integration

- The main loop remains readable: it polls the room selector, handles registration timeouts, reads touch events, performs card reads, and delegates decisions to dedicated functions.
- Status codes encapsulate all states: registration attempts, access results, WiFi and HTTP errors, and delete operations. The indicator module translates these into consistent LED and buzzer patterns.
- The Node.js backend exposes clear endpoints for registration, access checks, per-room deletion, and statistics. It updates Excel files atomically and feeds an auto-refresh dashboard that reads the same data as the device.

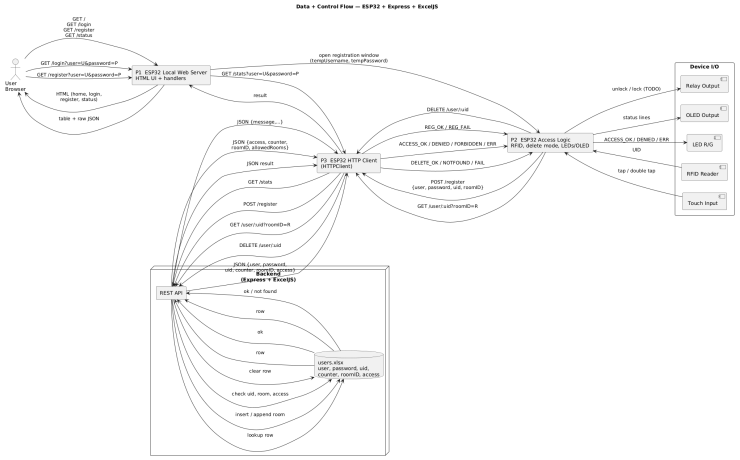



Figure 14: Bidirectional data and control flow across device, backend, logs, and UI.

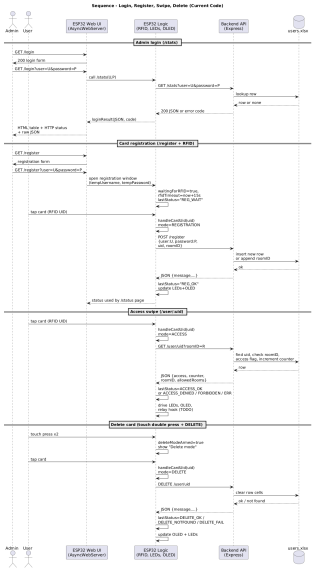



Figure 15: End-to-end access sequence from tap to LED feedback to log storage.

## 0.9 Neatness, Visual Design, and Organization

The system is designed to be visually clean both physically and in software:

- Wiring on the breadboard or prototype board is color-coded by function (power, data, indicators), and pin labels match the GPIO mapping table.
- OLED messages are short and structured in up to three lines (context, status, detail), for example `ESP32 Access`, `Room 105`, `Registration OK`.
- The web pages share a common dark theme, consistent typography, and responsive layout. Tables for users and logs are clearly separated with headers and scrollable wrappers.
- The project folders separate firmware, backend, and documentation so that the presentation assets, video, and code can be located quickly.

## 0.10 Effectiveness, Completion, and Demonstration Flow

The live demo and video are organized around concrete scenarios that exercise the full system:

- **Scenario 1: New user registration** The admin opens the local web registration page, enters a new user and password, then taps a blank RFID card within the registration window. The OLED, LED, and buzzer confirm registration success, while the dashboard shows the new row in `users.xlsx`.
- **Scenario 2: Access granted** The same card is presented at the reader for the configured room. The ESP32 calls the backend, receives a positive decision, lights the LED green, plays the access-ok pattern, and simulates door unlock. The logs table immediately displays the successful event.
- **Scenario 3: Access denied and error handling** An unknown card is tapped. The indicators show access denied in red with a different sound. The dashboard shows the failed access row. A short WiFi disconnect is induced to show specific error feedback and recovery when the connection returns.
- **Scenario 4: Delete room and delete user** The touch button is used for double-tap delete-room mode and triple-tap delete-user mode. After a confirmation tap with a card, the backend removes the corresponding mapping, and the next access attempt reflects the new policy in real time.

These scenarios fit the allocated presentation slot and cover core functionality, error conditions, and admin workflows. They show that the system is usable as a small access control prototype, not just an isolated hardware or software demo.

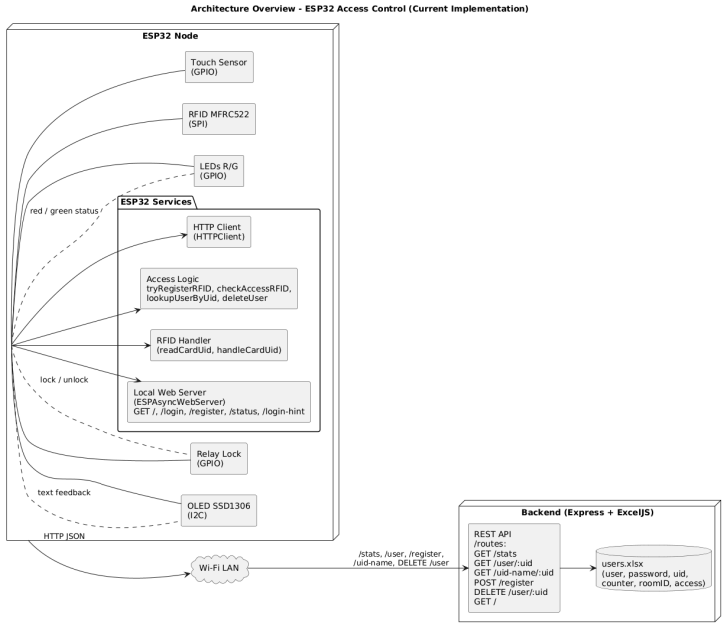



Figure 16: Overall system architecture used during the demonstration.

## 0.11 System Risks and Limitations

### WiFi and local-host dependency

- The system does not require internet access, but it depends entirely on a stable WiFi LAN because all authentication and user checks come from the backend service.
- If the WiFi connection drops, the ESP32 cannot recover automatically. A manual reset is required to re-establish communication.
- The backend IP must be statically assigned. If the server uses DHCP and its IP changes, the ESP32 can no longer reach the backend, and all door operations fail until the address is fixed or the firmware is updated.

### RFID security limitations

- The MFRC522 is used in UID-only mode, with all MIFARE Classic authentication and encryption features disabled.
- UID-only identification is insecure and vulnerable to card cloning.
- No authentication keys or user permissions are stored on the ESP32 for safety, but this prevents local validation and increases reliance on the backend.

### Backend and data storage risks

- The system stores all user and log data in unencrypted CSV/Excel files. This format is simple for demos but fragile and vulnerable to corruption during concurrent writes.

- Anyone with access to the LAN and knowledge of the backend JSON endpoints can read all users, export logs, or inject new entries.
- Because API access is fully open inside the LAN, a malicious user could delete all users, wipe logs, or bypass normal registration processes.
- A corrupted or compromised network can expose the entire database, since there is no encryption of stored data and no fine-grained API access control.
- A straightforward improvement would be encrypting the CSV/Excel files and adding authentication and authorization to the backend API.

**System architecture constraints**

- No local caching is implemented, so each RFID tap requires a live validation request to the backend.
- There is a single backend dependency. If the Node.js server fails, locks the file, or becomes unreachable, the entire system stops functioning.
- If the static backend IP configuration is incorrect or changes, all device-to-server communication fails until corrected manually.

**Hardware and prototype limitations**

- Breadboard wiring limits durability and reliability compared to a PCB.
- MFRC522 read range is short and sensitive to card orientation and positioning.
- Capacitive touch input introduces environmental noise and may miscount very fast or inconsistent multi-press interactions.

**OLED reliability limitation**

- The system fully supports on-device OLED debugging for status messages, errors, and registration prompts, but the SSD1306 module used in the prototype became non-functional during final testing.
- Without the OLED, real-time debugging and visual status feedback are reduced to LED and buzzer indicators only, which limits clarity during error states or backend failures.
- This also reduces usability during registration windows, since text prompts and timing feedback are no longer displayed.
- The issue appears hardware-related rather than firmware-related, but it highlights the fragility of the prototype setup and the dependency on the OLED for full diagnostic visibility.

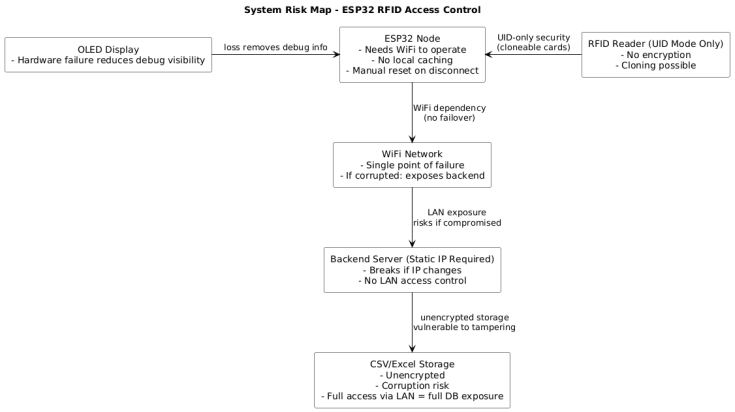



Figure 17: System risk map summarizing main dependencies and failure points.