The proposed fix is to decouple **I/O hardware** from the Raspberry Pi and treat the Pi as a kiosk-style web client, while all physical controls send events to the web server via HTTP. This removes the GPIO bottleneck caused by the touch display and lets the system behave like a custom embedded device with a web-based "control plane".

Context and goal

Earlier iterations already assumed a Raspberry Pi-based player with a touch display, plus tactile controls (buttons/knob) and an NFC reader as key interaction primitives.
This re-architecture keeps those interaction elements, but moves them off the Pi and turns the Pi into a "remote screen + default playback endpoint" controlled by the server.

Components (logical + hardware)

- **Raspberry Pi (Kiosk Client / Default Player)**
  - Runs a locked-down browser in kiosk mode.
  - Opens the web app (same site already partially operational).
  - Identified as the default device using a heuristic (e.g., screen resolution / physical dimensions) and/or a persistent device_id.
- **Arduino #1 (NFC Node)**
  - Connected to the NFC reader.
  - On tag read, sends an HTTP POST event to the server containing: account_id, user_id, track_id (or "music code"), plus metadata like timestamp and reader id.
- **Arduino #2 (Controls Node)**
  - Connected to 3 LED buttons and a rotary encoder (knob).
  - Sends HTTP events for button presses and knob rotation.
  - Receives optional HTTP responses for LED states (or pulls state periodically).
- **Web Server (Control Plane)**
  - Maintains device registry per account (which devices exist, which are online, capabilities).
  - Converts incoming hardware events into playback/control commands.
  - Routes commands to the selected playback device (default: Raspberry Pi).

HTTP API specification (minimal, pragmatic)

All requests should be over HTTPS, with a device-scoped token (e.g., Authorization: Bearer <device_token>).

1) Device registration / presence

- POST /api/v1/devices/register
  - Body: { device_id, account_id, device_type, capabilities, screen: {width, height}, fw_version }
- POST /api/v1/devices/heartbeat
  - Body: { device_id, ts, status }
  - Server uses heartbeats to mark devices online/offline and to count "how many devices are connected to an account".

2) Event ingestion (from Arduinos)

- NFC read:

    - POST /api/v1/events/nfc
    - Body: { device_id, account_id, user_id, track_id, card_uid, ts }
- Buttons:
    - POST /api/v1/events/button
    - Body: { device_id, account_id, control: "prev|play_pause|next", action: "press|release", ts }
- Encoder:
    - POST /api/v1/events/encoder
    - Body: { device_id, account_id, delta: <int>, ts }
    - delta is signed (e.g., +1/-1 per detent) and server maps it to volume/menu actions.

3) Command delivery (to Raspberry Pi web client)

Because you want HTTP-only (GET/POST), use one of these patterns:

- **Polling (simple, robust)**
    - Pi calls: GET /api/v1/devices/{device_id}/commands?since=<cursor>
    - Server returns: { cursor, commands: […] }
    - Pi acknowledges: POST /api/v1/devices/{device_id}/commands/ack
- (Optional later) SSE/WebSocket would be cleaner, but not required for the stated approach.

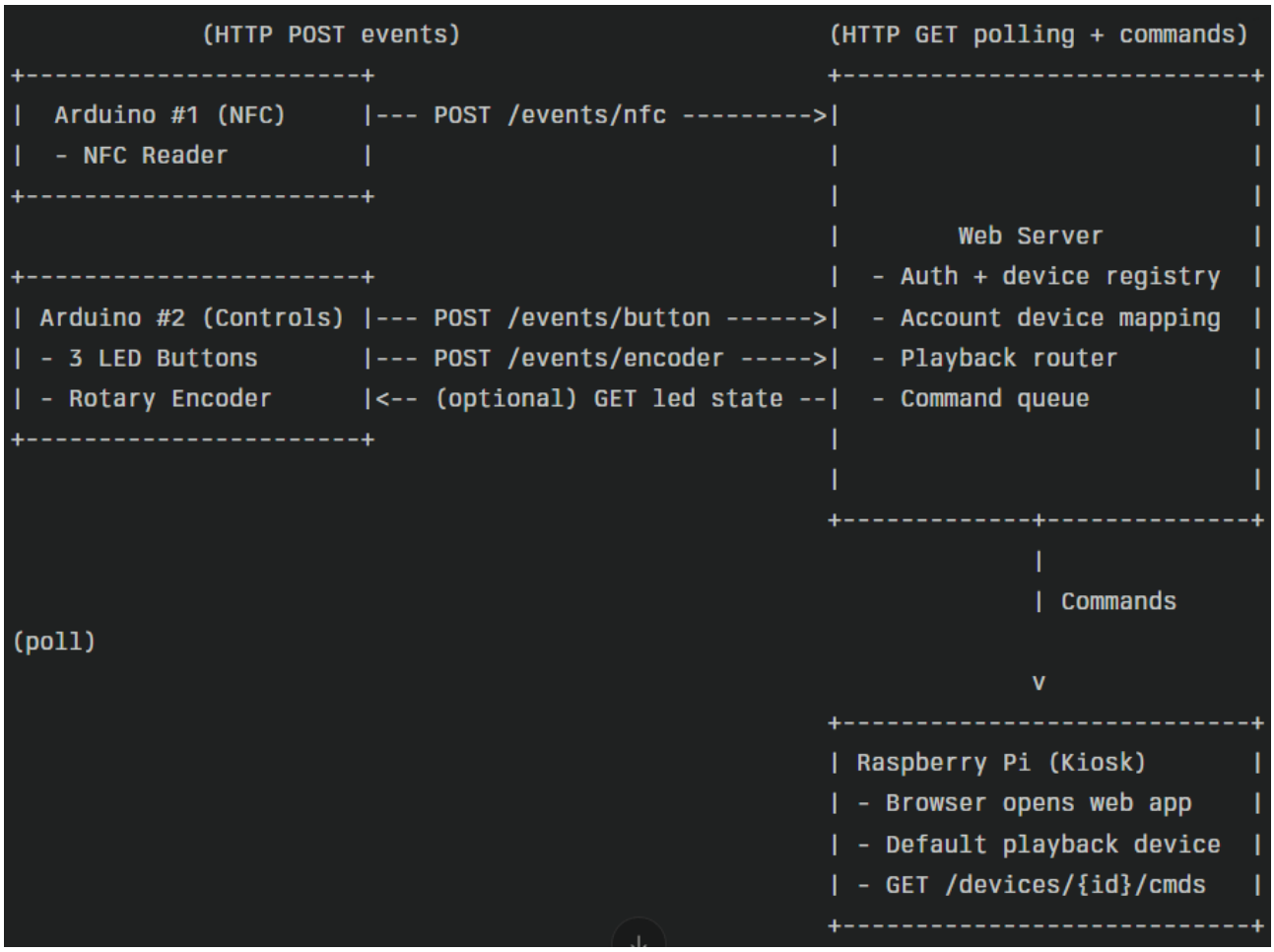
4) Playback routing logic (server-side)

When an event arrives (e.g., NFC):

1. Resolve account_id → list online devices for that account.
2. Select target playback device:
    - If a “preferred player” is set: use it.
    - Else default to the Raspberry Pi by heuristic (e.g., known device_type="raspberry_kiosk" and/or expected screen dimensions).
3. Emit a command to that device, e.g.:
    - { type: "play", track_id, user_id }
    - { type: "pause" }, { type: "next" }, { type: "set_volume", value }
4. Update UI state so the Pi page reflects “Now playing”, user, and track metadata.

Text schematic (components + flows)

text


(HTTP POST events)
(HTTP GET polling + commands)
Arduino #1 (NFC)
- NFC Reader
--- POST /events/nfc --------->
Arduino #2 (Controls)
- 3 LED Buttons
- Rotary Encoder
--- POST /events/button ------>
--- POST /events/encoder ----->
<-- (optional) GET led state --
Web Server
- Auth + device registry
- Account device mapping
- Playback router
- Command queue
Commands
(poll)
Raspberry Pi (Kiosk)
- Browser opens web app
- Default playback device
- GET /devices/{id}/cmds