Project: Real-Time Venue Operating System (Beta)

Absolute constraints

- Output **ONLY bash scripts** runnable as **root**
- VPS is **empty** (Ubuntu 22.04 LTS assumed)
- DNS is **already configured via Cloudflare**
- You must **not ask questions**
- You must **choose sane defaults**
- Focus on **core behavior + automation**
- Ignore UI polish
- Use **simple web pages / APIs** only

---

1. SYSTEM PURPOSE (DO NOT MISINTERPRET)

This system is a **real-time operating system for a venue**, not a POS, not a loyalty app.

It must demonstrate:

1. One app, multiple roles (adaptive behavior)
2. Real-time operational automation
3. Venue-controlled incentives
4. User engagement via rewards
5. Admin has full real-time overview
6. All actions are logged and observable

If these cannot be tested, the build is wrong.

---

2. REQUIRED DOMAINS (ALREADY POINTING TO VPS)

You must configure NGINX virtual hosts for:

- `os.peoplewelike.club` → main app (all roles)
- `admin.peoplewelike.club` → admin interface
- `api.peoplewelike.club` → backend API

Cloudflare proxy is already active (see DNS screenshot)

Use **Let’s Encrypt (certbot)**.

---

3. HIGH-LEVEL ARCHITECTURE (KEEP IT SIMPLE)

Stack

- NGINX (reverse proxy)
- Node.js (Fastify or Express)
- SQLite (or PostgreSQL if faster for you)
- No Docker
- No frontend frameworks
- No auth complexity (simple role tokens)

Services

1. **API service** (`:4000`)
2. **Static app service** (`:4001`)

3. **Admin service** (`:4002`)

All managed by **PM2**.

---

4. CORE CONCEPTS (YOU MUST IMPLEMENT)

4.1 One App, Multiple Modes (CRITICAL)

There is **ONE web app**, but it behaves differently depending on:

- role = `guest | bar | runner | security | admin`
- venue_id
- session_state = inside / outside

This is called **Adaptive Mode Switching**.

Do **NOT** create separate apps.

4.2 Venue Session (CORE LOGIC)

When a user enters:

- A `venue_session` opens
- All actions attach to it
- When they leave, it closes

Fields:

- user_id
- venue_id

- opened_at
- closed_at
- total_spend
- interactions_count

4.3 Wallet & Rewards (BETA VERSION)

Implement a **closed-loop wallet**:

- balance (integer points)
- earn points via:
    - entry
    - quests
- spend points on:
    - drinks (simulated)

No real payments.

---

5. ROLES & FUNCTIONALITY (MANDATORY)

5.1 Guest

Needs:

- See wallet balance
- See active quests
- Receive push-like notifications (polling OK)

- Get rewards for actions

Actions:

- “Check in”
- “Complete quest”
- “Spend points”

5.2 Bar Staff

Needs:

- Simple POS page
- Sell item → reduce stock
- Trigger stock events

Actions:

- Create order
- Deduct inventory

5.3 Runner Staff

Needs:

- Real-time alerts when stock is low

Logic:

- If stock < threshold → notify runner

5.4 Security

Needs:

- Live headcount
- Incident log

5.5 Venue Admin (MOST IMPORTANT)

Admin must have **real-time overview**.

Needs:

- Live dashboard:
    - active sessions
    - stock levels
    - recent events
- Ability to:
    - change prices
    - create quests
    - push notifications
    - set automation rules

Admin **does not** manually operate — configures rules.

---

6. AUTOMATION ENGINE (MINIMUM)

Implement **simple rule engine**:

Examples:

- IF stock < X → notify runner
- IF before 22:00 → price = lower
- IF user completes quest → reward points
- IF venue empty → activate incentive

Rules can be hardcoded JSON.

This is **non-negotiable**.

---

7. REAL-TIME REQUIREMENTS (BETA LEVEL)

- Use polling or WebSockets (your choice)
- Admin dashboard must update within seconds
- Logs must be visible

---

8. DATA MODELS (MINIMAL)

Tables:

- users (id, role, name)
- venues
- venue_sessions
- inventory
- orders
- quests

- quest_completions
- notifications
- logs

Everything must be inspectable.

---

9. WHAT MUST BE TESTABLE AFTER INSTALL

After running the bash script, I must be able to:

1. Open `os.peoplewelike.club`
2. Choose role (guest / staff / admin)
3. Simulate:
    - entry
    - spend
    - low stock
4. See:
    - runner alerted
    - admin dashboard update
5. Create a quest
6. Complete quest as guest
7. Receive reward
8. Change price live

If this fails → build failed.

---

10. BASH SCRIPT RESPONSIBILITIES

Your bash script must:

1. Update system
2. Install:
   - nginx
   - nodejs
   - pm2
   - certbot
3. Configure NGINX (3 domains)
4. Obtain SSL certs
5. Create folders:
   - `/srv/os/api`
   - `/srv/os/app`
   - `/srv/os/admin`
6. Install Node dependencies
7. Write minimal server code
8. Start services via PM2
9. Print URLs at the end

---

11. THINGS YOU MUST NOT DO

❌ No Docker
❌ No Kubernetes
❌ No React/Vue
❌ No over-engineering
❌ No external SaaS
❌ No payment integration

---

12. GUIDING PRINCIPLE (DO NOT FORGET)

This system exists to prove **one thing**:

> **Venues can shape behavior in real time by rewarding participation, and users enjoy it.**

Everything else is secondary.

---

13. FINAL OUTPUT FORMAT

You must output:

- One or multiple **bash scripts**
- Clearly labeled
- Runnable as root
- No explanations
- No commentary

END OF INSTRUCTIONS