PROJECT: World Mall (The Human Square) — Phase 2

Goal (today):

1. Enable **Guest Mode** for web users (no World ID required) with **strict limits**, while **Verified** users get full features and perks.
2. Implement **real World ID server verification** (Cloud/MiniKit) and **Permit2 signature verification** foundations.
3. Add **DB integrity constraints** and small UX fixes (after flows, “return to previous page”).
4. Keep production deploy stable (single port / no dev HMR in prod).

---

0) Guardrails & conventions

- Never weaken server-side gates for verified-only actions; Guest Mode is **additive** with strict limits.
- Everything feature-flagged via `.env`:
  - `ALLOW_GUEST_CHAT=true` (default true for dev, false allowed for prod if needed)
  - `GUEST_MAX_CHARS=10` (short “hello” style)
  - `GUEST_COOLDOWN_SEC=600`
  - `GUEST_MAX_PER_DAY=3`
  - `WORLDID_VERIFY_MODE=cloud|minikit` (start with `cloud`)
  - `WORLDID_APP_ID=...`, `WORLDID_ACTION=humans-square-post`
  - `PERMIT2_CHAIN_ID=8453` (or the chain you target), `PERMIT2_CONTRACT=` (official address for the chain), `RPC_URL=` (HTTPS RPC)
- Keep **prod** `.replit` as single port 80; do not reintroduce dev servers there.
- Code style: TypeScript strict; small pure functions; don’t mix view and policy.

---

1) Guest Mode (web) + Verified Perks

What to build

- Add a **role** concept: `guest | verified | admin`.
- Guests (no World ID) **can** send one-liners with **hard cap** and **heavy rate limits**.
- Guests **cannot**: star, report, post links, post in Work room, send DMs, or trigger payouts.
- Verified users keep full chat (240 chars), stars, work posts, connect, report, ledger, etc.
- UI clearly nudges: “Verify to unlock longer posts, stars ⭐, Work Mode, and perks.”
- **Redirect after actions**: when a user verifies or closes the verification sheet, route them back to the last page (Global or Work) and focus the composer.

Implementation details

- **Server**
  - In `shared/schema.ts` (or equivalent):
    - Add `UserRole = 'guest' | 'verified' | 'admin'`.
    - Extend message model with `authorRole: UserRole`, `isGuest:boolean`.
  - In `server/storage.ts` (DB layer):
    - Ensure message insert writes `authorRole`.
  - In `server/index.ts` / routes:
    - **Session**: For guests, assign a signed, httpOnly cookie `gsid` (uuid v4) if absent. Persist in DB `guest_sessions(id, ip_hash, user_agent_hash, created_at, last_seen)`.
    - **Limiter**: Add per-**gsid** window + per-IP fallback. Enforce `GUEST_COOLDOWN_SEC`, `GUEST_MAX_PER_DAY`. Return `GUEST_LIMIT` on violation.
    - **Whitelist**: Guests can only post text matching **safe one-liner** rules:
      - max length `GUEST_MAX_CHARS`
      - must not contain URL/mention/emojis flood/casing spam
      - optional whitelist of greetings: `hello|hi|hey|gm|`👋 (normalize and check).
    - **Room policy**: guests only to `/room/global`. Reject `/room/work` for guests.
    - Ensure all privileged endpoints (`/api/star`, `/api/report`, `/api/work/*`, `/api/payouts/*`) check `role==='verified' || 'admin'`.
- **Client**
  - Add `useAuthRole()` hook reading role from `/api/me` (new tiny endpoint returning `{role, verified, handle}`).
  - **Composer** behavior:
    - If `role==='guest'`:
      - placeholder: “Say hello 👋 (verify to unlock full chat)”
      - input `maxLength = GUEST_MAX_CHARS` (fetched from `/api/policy`)
      - disable star/report buttons; show `Tooltip: Verify to use`.
    - If verified: restore 240 char limit, stars, Work Mode toggle.
  - **Redirect after verify**:
    - Save `lastRoute` in `localStorage` before showing verify UI.
    - On verify success or close, `router.navigate(lastRoute ?? '/room/global')` and `scrollToComposer()`.

Acceptance tests (Agent can run)

- **Guest post OK**:
  - `curl -s -X POST http://localhost:5000/api/messages -H "Content-Type: application/json" -d '{"text":"hello","room":"global"}'` → 200 with `authorRole:'guest'`.
- **Guest blocked on long text**: `"hello everyone"` → 400 `GUEST_FORBIDDEN`.
- **Guest rate-limit**: 4th post same day → 429 `GUEST_LIMIT`.
- **Guest cannot star**: POST `/api/star` → 403 `VERIFICATION_REQUIRED`.
- **Verified can star/post long**: simulate by sending `x-dev-verified: 1` header in dev mode (guarded by `NODE_ENV!=='production'`) → 200.

---

2) World ID — real server verification

What to build

- Add real verification on server for posted proofs (Cloud first, MiniKit optional).
- Store **hashed nullifier** + `verified_at`.
- Expose `/api/verify/worldid` expecting `{proofPayload}`; server validates and sets session role to `verified`.

Implementation details

- Add `server/worldid.ts` with:
  - **Cloud** mode: POST to World ID Cloud verify endpoint with `WORLDID_APP_ID`, `WORLDID_ACTION`; on `valid:true`, trust nullifier.
  - **MiniKit** mode: verify proof locally with SDK and app keys.
- Hash nullifier with `scrypt`/`argon2id` + salt; save `{hashedNullifier, appId, action, verified_at}`.
- Update `/api/me` to read role from session (`verified` if session tied to a verified nullifier).

Acceptance tests

- Successful verify returns `{role:'verified'}`; posting long text after verify succeeds.
- Attempts with invalid proof → 400 with `INVALID_PROOF`.

---

3) Permit2 signature verification (foundation)

What to build

- Build **server-side EIP-712 Permit2 verification** utilities so that when you later enable on-chain transfers, signatures are validated **before** any transaction attempt.
- No real transfers yet; just a `/api/permit2/verify` endpoint returning validity and parsed fields.

Implementation details

- Add `server/permit2.ts`:
  - Use canonical Permit2 domain for target chain (`name: "Permit2"`, `chainId: PERMIT2_CHAIN_ID`, `verifyingContract: PERMIT2_CONTRACT`).
  - Validate `signature` against `PermitTransferFrom` struct using `@ethersproject/*` or `viem`.
  - If `RPC_URL` provided, optional `eth_call` static to check token allowance/nonce.
- Add admin toggle flag in `/api/policy` to expose whether transfers are enabled in this environment.

Acceptance tests

- POST `/api/permit2/verify` with a known bad signature → `{valid:false}`.
- With dev fixture (generate a signature in tests) → `{valid:true}`.

---

4) Database integrity & indexes

What to build

- Harden schema: unique constraints and helpful indexes.

Implementation details (Drizzle/SQL)

- `users`: unique `handle`, unique `hashed_nullifier`.
- `messages`: index `(room, created_at DESC)`, FK to `users`.
- `stars`: composite unique `(message_id, user_id)`.
- `rate_limits`: index by `(key, window_start)`.
- Add migration file and run migrations on start.

Acceptance tests

- Double-star same message → 409.
- Duplicate nullifier insert → error handled with 409.

---

5) UX polish & navigation

What to build

- **Back to previous page** after verify, connect, or admin modal close.
- Small copy update on Global:
    - Banner when guest: “You’re in **Guest Mode** — verify with World ID to unlock full chat, ⭐stars, and Work Mode.”

Implementation details

- Modal close handlers call `popToLastRoute()`; keep `lastRoute` in `localStorage`.

Visual tests

- Verify in Global → returns to Global with composer focused.
- Verify in Work (as dev) → returns to Work.

---

6) Admin & policy endpoints

- `/api/policy` returns public policy knobs: `{guest:{enabled,maxChars,cooldown,maxPerDay}, message:{maxCharsVerified:240}, features:{workMode:true, stars:true, permit2:false}}`.
- Client reads to render limits and tooltips dynamically.

---

7) Non-regressions (must keep working)

- Health & presence endpoints unchanged.
- Production deploy stays **single port 80** (no vite HMR), run script `NODE_ENV=production node dist/index.js`.
- Logs remain clean; no secrets in client bundles.

---

8) Step plan (execute in this order)

1. **Scaffold**: feature flags, `/api/policy`, `/api/me`, guest session cookie.
2. **Server policy**: guest validators + rate-limiting + room policy.

3. **Client**: role-aware composer; UI tooltips; banners; redirect handling.
4. **World ID**: implement `/api/verify/worldid` (Cloud first); wire to client verify flow; set role=verified.
5. **Permit2 utils**: add `/api/permit2/verify` and tests (no real transfers).
6. **DB constraints & migrations**.
7. **Tests** (curl script set below).
8. **Docs**: update README with env vars & how to toggle Guest Mode.
9. **Deploy** to production (unchanged `.replit` deploy section).

---

9) Test script (Agent can run these)

```
# 1) Policy

curl -s http://localhost:5000/api/policy


# 2) Guest post OK, one-liner

curl -s -X POST http://localhost:5000/api/messages \

  -H "Content-Type: application/json" \

  -d '{"text":"hello","room":"global"}'


# 3) Guest blocked on long

curl -s -X POST http://localhost:5000/api/messages \

  -H "Content-Type: application/json" \

  -d '{"text":"hello everyone","room":"global"}'


# 4) Guest cannot star

curl -s -X POST http://localhost:5000/api/star \

  -H "Content-Type: application/json" \
```

```
  -d '{"messageId":"test"}'


# 5) Dev-verified (only in dev): emulate verified for tests

curl -s -X POST http://localhost:5000/api/messages \

  -H "Content-Type: application/json" -H "x-dev-verified: 1" \

  -d '{"text":"This is a full length verified
message.","room":"global"}'
```

---

10) Deliverables

- Code changes with clear commits:
  - `feat(guest): limited guest mode + policy endpoints`
  - `feat(worldid): server verification & role upgrade`
  - `feat(permit2): eip-712 verify utilities`
  - `chore(db): add constraints & indexes`
  - `feat(ui): role-aware composer, redirect after verify`
- README section: **Guest Mode & Verified Perks**, **Environment Variables**, **Security Notes**.
- Short loom/log note summarizing tests passed.
- Re-deploy to production. Confirm `https://human-square-<…>.replit.app` shows Guest Mode and verified perks when run in World App.

If anything fails, stop, print the failing route/log, and fix before proceeding.

Notes / rationale

- Guest Mode lets you demo from web safely while keeping spam low.
- Verified users get obvious perks → strong incentive to verify in World App.
- All sensitive actions remain server-gated.
- Permit2 and World ID server checks move you from “works” to “review-safe”.

End of prompt.