# Owner Service Development Plan

## 1. Define the service boundary (what Owner Service owns)

**Owner Service is the source of truth for:**

- Owner identity in this system (not Keycloak user)
- Owner business profile + address
- Owner status lifecycle (PENDING/ACTIVE/SUSPENDED/CLOSED)
- KYC-lite verification records
- Owner settings (language/timezone/notifications)
- Tenant isolation by `ownerId`

**Owner Service is NOT responsible for:**

- Authentication login sessions (use Keycloak later)
- Room listing itself (Room Service owns rooms)
- Payments (Payment service owns subscription)

## 2. Core domain model (entities + lifecycle)

### Owner (Aggregate Root)

Minimum fields we'll need:

- `id` (ownerId)
- `email?`, `phone?` (at least one required)
- `status` = `PENDING | ACTIVE | SUSPENDED | CLOSED`
- `profile` (name, contact person, etc.)
- `businessInfo` (businessName, taxId? optional, type)
- `address` (province/district/commune/village + detail)
- `settings` (language, timezone, notifications)
- `verification` (kycLite status + submittedAt + reviewedAt)
- `createdAt`, `updatedAt`

### OwnerVerification (KYC-lite)

- `ownerId`
- `status` = `NOT_SUBMITTED` | `SUBMITTED` | `APPROVED` | `REJECTED`
- `documentType` (ID_CARD / PASSPORT / BUSINESS_LICENSE)
- `documentUrl` / `objectKey` (if using MinIO)
- `selfieUrl` optional
- `remark` / `rejectReason`
- audit fields

### OwnerSettings

- `language` (km/en)
- `timezone` (Asia/Phnom_Penh default)
- `notificationPrefs` (email/sms/push toggles)

## 3. Status rules (this is what makes it "enterprise")

Define allowed transitions clearly:

**OwnerStatus**

- `PENDING → ACTIVE` (only after verification approved OR admin manual approve)
- `PENDING → CLOSED` (owner cancels)
- `ACTIVE → SUSPENDED` (admin action, policy issue, payment issue)
- `SUSPENDED → ACTIVE` (admin restore)
- `ACTIVE/SUSPENDED → CLOSED` (permanent close)

**VerificationStatus**

- `NOT_SUBMITTED → SUBMITTED`
- `SUBMITTED → APPROVED | REJECTED`
- `REJECTED → SUBMITTED` (resubmit)

*__business rules live in service/domain layer__, not controller.

# 4. API design

### Auth/registration flow (MVP)

1. `POST /owners/register`
   - accept phone/email + password later (or OTP later)
   - create Owner with `PENDING`
2. `GET /owners/{ownerId}`
3. `PUT /owners/{ownerId}/profile`
4. `PUT /owners/{ownerId}/business`
5. `PUT /owners/{ownerId}/address`
6. `PUT /owners/{ownerId}/settings`

### Verification flow

7. `POST /owners/{ownerId}/verification/submit`
8. `POST /owners/{ownerId}/verification/approve` (admin)
9. `POST /owners/{ownerId}/verification/reject` (admin)

### Status control (admin)

10. `POST /owners/{ownerId}/activate`
11. `POST /owners/{ownerId}/suspend`
12. `POST /owners/{ownerId}/close`

# 5. Multi-tenant readiness (owner isolation)

Our rule: **every record in this service is always tied to `ownerId`**.

Practical ways:

- Every query includes `ownerId`
- Every sub-collection uses `ownerId` as partition key (Mongo) or tenant key (SQL)
- Later when we add Keycloak, our gateway can inject `ownerId` into headers/claims

# 6. Data & validation rules (make it strict)

- Must provide **phone OR email**
- Phone format validation (Cambodia rules for now)
- Status cannot be changed directly via update profile endpoint
- Verification submit requires at least 1 document
- When `status = CLOSED`, block updates (read-only)

# 7. Events we should publish (so other services react)

When Owner changes, publish events to Kafka (outbox pattern):

- `OWNER_REGISTERED`
- `OWNER_PROFILE_UPDATED`
- `OWNER_VERIFICATION_SUBMITTED`
- `OWNER_VERIFICATION_APPROVED`
- `OWNER_STATUS_CHANGED`

Consumers:

- Room Service: allow posting rooms only when `ACTIVE`
- Notification Service: send messages
- Payment Service: enable subscription

*this is where our “Reactive System” story becomes real.

# 8. Implementation plan (best sequence)

## Sprint 1 (MVP)

- Owner entity + repository
- Register owner (PENDING)
- Get owner profile
- Update profile/business/address/settings

## Sprint 2

- Verification submit + admin approve/reject

- Status transitions + validations

## Sprint 3

- Kafka outbox events
- Audit log + history table/collection

## Sprint 4

- Integration with Keycloak