# PAYIR THEKKAM — Product Design Doc

> *Below is a single, actionable design doc you can hand to engineers (backend + frontend/KMM mobile) to build the product. It assumes a Kotlin-stack everywhere.*

**Mobile:** Kotlin Multiplatform Mobile (KMM) + Jetpack Compose (Android) / Compose Multiplatform (iOS fallback) — SQLDelight for local DB, Ktor client, Kotlinx.serialization.

**Server:** Kotlin (Ktor), Postgres, Exposed or jdbi (or use Ktorm), Kotlinx.serialization, Basic username/password auth.

**Dev infra:** Flyway (migrations), Postgres.

I'll break into two major parts: **BACKEND** then **FRONTEND** (mobile/KMM). Each contains models, APIs, DB design, services, sync, and implementation notes.

---

## 1 — Backend (Kotlin server-side)

### 1.1 Goals & constraints

- Provide mobile app a compact, reliable REST/JSON API.
- Minimal features for v1: account (register/login with username/password), facility catalog (by area), availability & booking, storage records, seed/offline data package, basic admin endpoints to update facility availability.
- **Secure:** Basic username/password auth, role-based (farmer, admin, agri_dept).
- **Scale:** PostgreSQL, horizontal Ktor instances behind load balancer.

### 1.2 Tech stack

- **Ktor** (server) + Kotlinx.serialization (JSON)
- **Database:** PostgreSQL
- **ORM/DB access:** Exposed or SQLDelight on server, or plain SQL via jdbi
- **DB migrations:** Flyway
- **Auth:** Basic username/password authentication (no JWT)

### 1.3 High-level services

- **AuthService** — register/login with username/password
- **FarmerService** — farmer profile, crop plantings, document uploads
- **AdminService** — add/update facilities, manage capacities, bookings
- **AgriDeptService** — approve/disapprove farmer requests, verify documents
- **FacilityService** — list facilities by district/taluk/village, capacities, pricing, contact info
- **BookingService** — check availability, reserve space, cancel (with history)
- **StorageRecordService** — farmer's stored crops metadata & history (with audit trail)
- **SyncService** — offline sync endpoints & delta sync (seed + diffs)

### 1.4 Database Schema Tables

#### `users` table

*Common table for all user types (farmers, admin, agri dept personnel)*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique user identifier |
| username | text | UNIQUE, NOT NULL | - | Username for login |
| password_hash | text | NOT NULL | - | Hashed password (bcrypt) |
| role | text | NOT NULL | - | User role: FARMER, ADMIN, AGRI_DEPT |
| email | text | - | - | Email address |
| phone | text | - | - | Phone number |
| is_active | boolean | NOT NULL | true | Account active status |
| created_at | timestamptz | - | now() | Account creation timestamp |
| updated_at | timestamptz | - | now() | Last update timestamp |
| created_by | uuid | FOREIGN KEY → users.id | - | User who created this account |
| updated_by | uuid | FOREIGN KEY → users.id | - | User who last updated this account |

**`farmers` table**

*Extended details for farmers - can plant multiple crops per season in different areas*

| Column | Type | Constraints | Default |
|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() |
| user_id | uuid | FOREIGN KEY → users.id, UNIQUE | - |
| name | text | NOT NULL | - |
| aadhaar_number | text | UNIQUE | - |
| district | text | NOT NULL | - |
| taluk | text | NOT NULL | - |

| | | | |
|---|---|---|---|
| village | text | NOT NULL | - |
| address | text | - | - |
| land_registration_number | text | - | - |
| verification_status | text | NOT NULL | 'PENDING' |
| verified_by | uuid | FOREIGN KEY → agri_dept_personnel.id | - |
| verified_at | timestamptz | - | - |
| rejection_reason | text | - | - |
| created_at | timestamptz | - | now() |
| updated_at | timestamptz | - | now() |
| created_by | uuid | FOREIGN KEY → users.id | - |
| updated_by | uuid | FOREIGN KEY → users.id | - |

**`crop_plantings` table**

*Tracks multiple crop plantings per farmer per season in different areas*

| Column | Type | Constraints | Default | Descriptio |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique planting identifier |
| farmer_id | uuid | FOREIGN KEY → farmers.id | - | Farmer who planted |
| crop_type | text | NOT NULL | - | Type of cro (e.g., PADD |

| | | | | |
|---|---|---|---|---|
| | | | | WHEAT, CORN |
| season | text | NOT NULL | - | Season: KHAR RABI, ZAID |
| year | int | NOT NULL | - | Year of planting |
| area_measurement | numeric(10,2) | NOT NULL | - | Area in acres/hecta |
| measurement_unit | text | NOT NULL | 'ACRES' | Unit: ACRES, HECTARES |
| district | text | NOT NULL | - | District of planting location |
| taluk | text | NOT NULL | - | Taluk of planting location |
| village | text | NOT NULL | - | Village of planting location |
| plot_details | text | - | - | Additional plot/locati details |
| planting_date | date | - | - | Actual planting da |
| expected_harvest_date | date | - | - | Expected harvest dat |
| status | text | NOT NULL | 'PLANTED' | Status: PLANTED, GROWING, HARVESTED, CANCELLED |
| created_at | timestamptz | - | now() | Record creation timestamp |
| updated_at | timestamptz | - | now() | Last update timestamp |
| created_by | uuid | FOREIGN KEY → users.id | - | User who created thi record |
| updated_by | uuid | FOREIGN KEY → users.id | - | User who la updated thi record |

**`farmer_documents` table**

*Uploaded government documents for farmer verification*

| Column | Type | Constraints | Default | De |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Uniq<br>iden |
| farmer_id | uuid | FOREIGN KEY → farmers.id | - | Farm<br>uplo |
| document_type | text | NOT NULL | - | Type<br>LAND_<br>BANK_<br>GOVT_ |
| document_name | text | NOT NULL | - | Docu<br>name |
| file_path | text | NOT NULL | - | Serv<br>or U |
| file_size_bytes | bigint | - | - | File<br>byte |
| mime_type | text | - | - | File |
| verification_status | text | NOT NULL | 'PENDING' | Stat<br>VERIF |
| verified_by | uuid | FOREIGN KEY → agri_dept_personnel.id | - | Agri<br>who |
| verified_at | timestamptz | - | - | Veri<br>time |
| rejection_reason | text | - | - | Reas<br>reje<br>reje |
| created_at | timestamptz | - | now() | Uplo |
| updated_at | timestamptz | - | now() | Last<br>time |
| created_by | uuid | FOREIGN KEY → users.id | - | User<br>uplo |
| updated_by | uuid | FOREIGN KEY → users.id | - | User<br>upda |

**`admin` table**

*Admin users who manage facilities, bookings, etc.*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique admin |

| | | | | |
|---|---|---|---|---|
| | | | | identifier |
| user_id | uuid | FOREIGN KEY → users.id, UNIQUE | - | Reference to users table |
| name | text | NOT NULL | - | Admin's full name |
| email | text | - | - | Admin email |
| phone | text | - | - | Admin phone |
| designation | text | - | - | Admin designation/role |
| department | text | - | - | Department name |
| created_at | timestamptz | - | now() | Record creation timestamp |
| updated_at | timestamptz | - | now() | Last update timestamp |
| created_by | uuid | FOREIGN KEY → users.id | - | User who created this record |
| updated_by | uuid | FOREIGN KEY → users.id | - | User who last updated this record |

**`agri_dept_personnel` table**

*Agricultural department personnel who approve/disapprove farmer requests*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique personnel identifier |
| user_id | uuid | FOREIGN KEY → users.id, UNIQUE | - | Reference to users table |
| name | text | NOT NULL | - | Personnel's full name |
| employee_id | text | UNIQUE | - | Government employee ID |
| email | text | - | - | Email address |
| phone | text | - | - | Phone number |
| designation | text | NOT NULL | - | Job designation |
| district | text | - | - | Assigned district |
| | | | | |

| | | | | |
|---|---|---|---|---|
| department | text | - | - | Department name |
| created_at | timestamptz | - | now() | Record creation timestamp |
| updated_at | timestamptz | - | now() | Last update timestamp |
| created_by | uuid | FOREIGN KEY → users.id | - | User who created this record |
| updated_by | uuid | FOREIGN KEY → users.id | - | User who last updated this record |

**`facilities` table**

*Storage facilities with audit trail*

| Column | Type | Constraints | Default | Descript |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique facility identifi |
| name | text | NOT NULL | - | Facility name |
| type | text | NOT NULL | - | Type: GOVERNMENT PRIVATE_MI COMMUNITY |
| district | text | NOT NULL | - | District location |
| taluk | text | NOT NULL | - | Taluk location |
| village | text | NOT NULL | - | Village location |
| address | text | - | - | Full address |
| total_capacity_sacks | int | NOT NULL | - | Total storage capacity sacks |
| available_capacity_sacks | int | NOT NULL | - | Currentl availabl capacity sacks |

| | | | | |
|---|---|---|---|---|
| price_per_sack | numeric(10,2) | - | - | Price pe sack (if applicab |
| contact_name | text | - | - | Contact person na |
| contact_phone | text | - | - | Contact phone number |
| owner_type | text | - | - | Owner typ descripti |
| is_active | boolean | NOT NULL | true | Facility active status |
| created_at | timestamptz | - | now() | Facility creation timestam |
| updated_at | timestamptz | - | now() | Last upda timestam |
| created_by | uuid | FOREIGN KEY → users.id | - | User who created this facility |
| updated_by | uuid | FOREIGN KEY → users.id | - | User who last updated this facility |

**`facilities_history` table**

*Audit trail for facilities table*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique history record identifier |
| facility_id | uuid | FOREIGN KEY → facilities.id | - | Reference to facility |
| action | text | NOT NULL | - | Action: CREATED, UPDATED, DELETED |
| changed_fields | jsonb | - | - | JSON object of changed |

| | | | | |
|---|---|---|---|---|
| | | | | fields |
| old_values | jsonb | - | - | JSON object of old values |
| new_values | jsonb | - | - | JSON object of new values |
| changed_at | timestamptz | - | now() | Change timestamp |
| changed_by | uuid | FOREIGN KEY → users.id | - | User who made the change |

**`bookings` table**

*Storage bookings with audit trail*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique booking identifier |
| farmer_id | uuid | FOREIGN KEY → farmers.id | - | Farmer who made the booking |
| facility_id | uuid | FOREIGN KEY → facilities.id | - | Facility being booked |
| crop_type | text | NOT NULL | - | Type of crop to store |
| quantity_sacks | int | NOT NULL | - | Number of sacks to store |
| status | text | NOT NULL | 'PENDING' | Status: PENDING, CONFIRMED, CANCELLED, COMPLETED |
| price_per_sack | numeric(10,2) | - | - | Price per sack at booking time |
| price_total | numeric(10,2) | - | - | Total price for booking |
| start_date | date | NOT NULL | - | Booking start date |
| end_date | date | - | - | Booking end date |
| notes | text | - | - | Additional |

| | | | | |
|---|---|---|---|---|
| | | | | notes |
| created_at | timestamptz | - | now() | Booking creation timestamp |
| updated_at | timestamptz | - | now() | Last update timestamp |
| created_by | uuid | FOREIGN KEY → users.id | - | User who created booking |
| updated_by | uuid | FOREIGN KEY → users.id | - | User who last updated booking |

**`bookings_history` table**

*Audit trail for bookings table*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique history record identifier |
| booking_id | uuid | FOREIGN KEY → bookings.id | - | Reference to booking |
| action | text | NOT NULL | - | Action: CREATED, UPDATED, STATUS_CHANGED, CANCELLED |
| old_status | text | - | - | Previous status (if status changed) |
| new_status | text | - | - | New status (if status changed) |
| changed_fields | jsonb | - | - | JSON object of changed fields |
| changed_at | timestamptz | - | now() | Change timestamp |
| changed_by | uuid | FOREIGN KEY → users.id | - | User who made the change |

**`storage_records` table**

*Storage records with audit trail*

| Column | Type | Constraints | Default | Descriptio |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  |  |  | record identifier |
| farmer_id | uuid | FOREIGN KEY → farmers.id | - | Farmer who owns the record |
| booking_id | uuid | FOREIGN KEY → bookings.id | - | Related booking (i applicable |
| crop_type | text | NOT NULL | - | Type of crop store |
| quantity_sacks | int | NOT NULL | - | Quantity stored in sacks |
| storage_type | text | - | - | Type of storage (e.g., HDPE_BAGS, SILO) |
| moisture_percent | numeric(5,2) | - | - | Moisture percentage |
| storage_location_desc | text | - | - | Descriptio of storage location |
| facility_id | uuid | FOREIGN KEY → facilities.id | - | Facility where stored (if applicable |
| stored_date | date | - | - | Date when crop was stored |
| created_at | timestamptz | - | now() | Record creation timestamp |
| updated_at | timestamptz | - | now() | Last updat timestamp |
| created_by | uuid | FOREIGN KEY → users.id | - | User who created this recor |
| updated_by | uuid | FOREIGN KEY → users.id | - | User who last updated this recor |

**`storage_records_history` table**

*Audit trail for storage_records table*

| Column | Type | Constraints | Default | Descriptio |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique history record identifier |
| storage_record_id | uuid | FOREIGN KEY → storage_records.id | - | Reference to storage record |
| action | text | NOT NULL | - | Action: CREATED, UPDATED, DELETED |
| changed_fields | jsonb | - | - | JSON objec of changed fields |
| old_values | jsonb | - | - | JSON objec of old values |
| new_values | jsonb | - | - | JSON objec of new values |
| changed_at | timestamptz | - | now() | Change timestamp |
| changed_by | uuid | FOREIGN KEY → users.id | - | User who made the change |

**`seed_metadata` table**

*Helps clients know seed package version*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique metadata identifier |
| version | int | UNIQUE, NOT NULL | - | Seed package version number |
| created_at | timestamptz | - | now() | Version creation timestamp |

**`device_sync_state` table**

*Tracks device sync status for offline mobile apps*

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| user_id | uuid | FOREIGN KEY → users.id | - | User identifier |
| device_id | text | NOT NULL | - | Unique device identifier |
| last_sync_at | timestamptz | - | - | Last successful sync timestamp |
| last_seed_version | int | - | - | Last synced seed version |

**`districts` table (Master lookup)**

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique district identifier |
| name | text | UNIQUE, NOT NULL | - | District name |
| state | text | - | - | State name |
| code | text | UNIQUE | - | District code |

**`taluks` table (Master lookup)**

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique taluk identifier |
| district_id | uuid | FOREIGN KEY → districts.id | - | Parent district |
| name | text | NOT NULL | - | Taluk name |
| code | text | UNIQUE | - | Taluk code |

**`villages` table (Master lookup)**

| Column | Type | Constraints | Default | Description |
|---|---|---|---|---|
| id | uuid | PRIMARY KEY | gen_random_uuid() | Unique village identifier |
| taluk_id | uuid | FOREIGN KEY → taluks.id | - | Parent taluk |
| name | text | NOT NULL | - | Village name |
| code | text | UNIQUE | - | Village code |

### 1.5 REST API (examples)

All payloads JSON, versioned: `/api/v1/...`

Use Basic Authentication (username/password) for protected endpoints.

**Auth**

- **`POST /api/v1/auth/register`**
    - Body: `{ "username", "password", "role", "email", "phone" }`
    - Response: `{ "userId", "username", "role" }`
- **`POST /api/v1/auth/login`**
    - Body: `{ "username", "password" }`
    - Response: `{ "userId", "username", "role", "token" }` (session token)
- **`POST /api/v1/auth/logout`**
    - Headers: `Authorization: Basic <credentials>`
    - Response: `{ "message": "Logged out successfully" }`

**Farmers**

- **`POST /api/v1/farmers/register`**
    - Body: `{ "username", "password", "name", "aadhaar_number", "district", "taluk", "village", "address" }`
    - Response: `{ "farmerId", "userId", "verification_status" }`
- **`GET /api/v1/farmers/{id}`** (protected)
    - Response: Farmer details with crop plantings
- **`GET /api/v1/farmers/{id}/crop-plantings`** (protected)
    - Response: List of crop plantings for farmer
- **`POST /api/v1/farmers/{id}/crop-plantings`** (protected)
    - Body: `{ "crop_type", "season", "year", "area_measurement", "measurement_unit", "district", "taluk", "village", "plot_details" }`
    - Response: `{ "plantingId" }`
- **`POST /api/v1/farmers/{id}/documents/upload`** (protected)
    - Body: Multipart form data with file
    - Response: `{ "documentId", "file_path" }`

**Agri Dept**

- **`GET /api/v1/agri/farmers/pending`** (protected, agri_dept only)
    - Response: List of farmers pending verification
- **`POST /api/v1/agri/farmers/{id}/approve`** (protected, agri_dept only)
    - Response: `{ "status": "APPROVED" }`
- **`POST /api/v1/agri/farmers/{id}/reject`** (protected, agri_dept only)
    - Body: `{ "rejection_reason" }`
    - Response: `{ "status": "REJECTED" }`
- **`GET /api/v1/agri/documents/pending`** (protected, agri_dept only)
    - Response: List of documents pending verification

- **`POST /api/v1/agri/documents/{id}/verify`** (protected, agri_dept only)
  - Body: `{ "verification_status", "rejection_reason?" }`
  - Response: `{ "status": "VERIFIED" }`

**Facilities & lookup**

- **`GET /api/v1/locations/districts`** → list districts
- **`GET /api/v1/locations/{district}/taluks`** → taluks
- **`GET /api/v1/locations/{taluk}/villages`** → villages
- **`GET /api/v1/facilities?district=&taluk=&village=&type=`**
  - Returns list of facilities with `available_capacity_sacks` and `price_per_sack`
- **`GET /api/v1/facilities/{id}`** → details
- **`POST /api/v1/facilities`** (protected, admin only) → create facility
- **`PUT /api/v1/facilities/{id}`** (protected, admin only) → update facility
- **`GET /api/v1/facilities/{id}/history`** (protected, admin only) → audit trail

**Availability & Booking**

- **`POST /api/v1/bookings/check`**
  - Body: `{ "facilityId", "quantitySacks" }` → returns `{ available: bool, availCount }`
- **`POST /api/v1/bookings`** (protected)
  - Body: `{ "facilityId", "cropType", "quantitySacks", "startDate", "endDate" }` → returns booking id & status
- **`GET /api/v1/bookings?farmerId=`** (protected) → list of bookings for farmer
- **`GET /api/v1/bookings/{id}`** (protected) → booking details
- **`GET /api/v1/bookings/{id}/history`** (protected) → booking audit trail
- **`POST /api/v1/bookings/{id}/cancel`** (protected)

**Storage Records**

- **`POST /api/v1/storage-records`** (protected)
  - Body: `{ "cropType", "quantitySacks", "moisturePercent", "storageType", "facilityId?", "bookingId?" }` → returns record id
- **`GET /api/v1/storage-records?farmerId=`** (protected) → list storage records
- **`GET /api/v1/storage-records/{id}/history`** (protected) → audit trail

**Sync & seed**

- **`GET /api/v1/seed?version=<localVersion>`**
  - → returns seed_package (facilities lookup minimal fields, locations) and seed_version. If client version matches, server returns delta or 204.
- **`POST /api/v1/sync/changes`**
  - Body: `{ deviceId, lastSyncAt, changes: { bookings[], storageRecords[] } }`
  - Server validates and returns conflicts/resolution suggestions.

### 1.6 Booking & consistency

Availability updates must be transactional: on booking confirm, decrement `available_capacity_sacks` using SQL transactions to avoid overbooking. Use `SELECT FOR UPDATE` or `UPDATE ... WHERE available_capacity_sacks >= :qty RETURNING ...`.

### 1.7 Admin endpoints

- Admin API to add/update facilities (protected endpoints).
- Admin can view all bookings, storage records.
- Admin can view audit trails for facilities, bookings, storage records.

### 1.8 Security

- Basic username/password authentication (bcrypt hashing).
- Passwords validated/sanitized. Rate-limit login endpoints.
- Use HTTPS + CORS policies.
- Validate payload server-side.
- Role-based access control (RBAC) for endpoints.

### 1.9 Migrations & seed data

Flyway migrations keep schema versioned. Seed initial facility data, districts/taluks/villages as JSON import.

### 1.10 Audit Trail

All CRUD operations on `facilities`, `bookings`, and `storage_records` are tracked in their respective `*_history` tables. This includes:

- Who made the change (`changed_by`)
- When the change was made (`changed_at`)
- What changed (`old_values`, `new_values`, `changed_fields`)
- Type of action (`CREATED`, `UPDATED`, `DELETED`, `STATUS_CHANGED`)

---

# 2 — Frontend / Mobile (Kotlin KMM)

### 2.1 Goals & constraints

- KMM single codebase for business logic, networking, database (SQLDelight), serialization.
- Native UI on Android (Compose) and basic on iOS (Compose Multiplatform or SwiftUI wrapper) — for hackathon focus on Android Compose.
- App must work fully offline: local seed data + local bookings/storage records, and sync when online.

### 2.2 Tech stack

- **KMM shared module:** Kotlin, Kotlinx.serialization, Ktor client, SQLDelight, Coroutines, Koin or Kodein DI.
- **Android app:** Jetpack Compose, Navigation Compose, Material3 tokens.

### 2.3 High-level architecture

```
Presentation (platform-specific UI)
  ← ViewModels (shared or platform-adapted)
```

```
    ← Repositories (KMM shared)
      ← Data sources (local SQLDelight + remote Ktor)
```

- Repositories expose suspend flows and result wrappers.
- Sync manager runs when online and pushes local changes to server; also pulls seed updates.

### 2.4 Shared domain models (Kotlin data classes)

(Use Kotlinx.serialization)

```kotlin
@Serializable
data class User(
    val id: String,
    val username: String,
    val role: String,
    val email: String?,
    val phone: String?
)

@Serializable
data class Farmer(
    val id: String,
    val userId: String,
    val name: String,
    val aadhaarNumber: String?,
    val district: String,
    val taluk: String,
    val village: String,
    val verificationStatus: String
)

@Serializable
data class CropPlanting(
    val id: String,
    val farmerId: String,
    val cropType: String,
    val season: String,
    val year: Int,
    val areaMeasurement: Double,
    val measurementUnit: String,
    val district: String,
    val taluk: String,
    val village: String
)

@Serializable
data class Facility(
    val id: String,
    val name: String,
    val type: FacilityType,
    val district: String,
    val taluk: String,
    val village: String,
    val totalCapacity: Int,
    val available: Int,
    val pricePerSack: Double?
)

@Serializable
data class Booking(
    val id: String,
    val farmerId: String,
    val facilityId: String,
    val cropType: String,
    val quantity: Int,
    val status: String,
    val startDate: String,
    val endDate: String?
)
```

### 2.5 Local DB design (SQLDelight)

Tables mirror server: `users`, `farmers`, `crop_plantings`, `facilities`, `bookings`, `storage_records`, `seed_metadata`. SQLDelight gives typed Kotlin models and multiplatform support.

**Important:** keep `seed_version` in metadata to detect out-of-date.

### 2.6 Networking (Ktor client)

- Ktor client configured with Json (Kotlinx), Logging, Basic Auth plugin.
- Endpoints: same as server REST. Implement retry/backoff and offline fallback.

### 2.7 Offline-first strategy

- On first run, app loads bundled `seed_package.json` (facilities + locations) into SQLDelight. This allows usage without network.
- On network availability, call `/api/v1/seed?version=localVersion` to fetch deltas.
- Any booking or storage-record created while offline is stored locally with `synced = false`. Sync manager uploads changes when online.

### 2.8 Sync behaviour

**Background sync triggered:**

- On app open when network available
- Periodic WorkManager (Android) if background allowed
- After user action (Confirm Booking) when online

**Sync algorithm:**

1. Upload local unsynced bookings/storageRecords → server returns serverIds or conflict.
2. Pull updated facilities seed_version → update local DB.
3. Reconcile availability (server wins for availability).

### 2.9 UI Pages & Components (flat minimal UI)

Implement pages as Compose screens. Each screen broken into components.

**Pages**

**Login Screen**

- Components: `TextInput(username)`, `TextInput(password)`, `PrimaryButton(Login)`, `Link(Register)`

**Register Screen**

- Components: `TextInput(username)`, `TextInput(password)`, `TextInput(name)`, `RoleSelector`, `PrimaryButton(Register)`

**Farmer Registration Screen**

- Components: `TextInput(aadhaar)`, `LocationDropdown(district->taluk->village)`, `FileUpload(document)`, `PrimaryButton(Submit)`

**Dashboard Screen**

- Components: `LargeActionCard("My Crops")`, `LargeActionCard("Find Storage")`, `LargeActionCard("My Bookings")`, `LargeActionCard("My Storage")`

**Crop Plantings Screen**

- Components: `CropPlantingList`, `FloatingActionButton(Add Planting)`, `CropPlantingCard`

**Add Crop Planting Screen**

- Components: `CropDropdown`, `SeasonSelector`, `NumberInput(Area)`, `UnitToggle(acres/hectares)`, `LocationDropdown`, `PrimaryButton(Save)`

**Facility List & Availability Screen**

- Components: `FacilityListItem` (name, available, price), `FilterBar(district/taluk)`, `Search`

**Confirm Booking / Storage Screen**

- Components: summary, confirm button, contact info, local receipt card.

**My Bookings / Storage History**

- Components: `BookingCard` with status and action buttons (Cancel, View History)

**Shared Compose components**

`PrimaryButton`, `InfoCard`, `SimpleList`, `IconButtonLarge`, `StatusBadge`, `SmallField` etc.

## 2.10 Component & ViewModel mapping (example)

- `LoginScreen` ↔ `AuthViewModel` (shared KMM ViewModel)
- `DashboardScreen` ↔ `DashboardViewModel` (exposes quick stats & actions)
- `CropPlantingsScreen` ↔ `CropPlantingViewModel` (manages crop plantings)
- `FacilityListScreen` ↔ `FacilityViewModel` (exposes facilities `Flow<PagedList>`)
- `BookingScreen` ↔ `BookingViewModel` (creates booking)

ViewModels use `StateFlow` for UI state.

### 2.11 Synchronization examples (end-to-end)

1. User confirms 20 sacks booking offline → local booking created status `PENDING_OFFLINE`.
2. On next online sync:
   - Upload booking to `/api/v1/bookings` → server validates and returns booking id and reduces `available_capacity_sacks`. Update local booking to `CONFIRMED` and store server id.
   - If server doesn't have capacity → return `CONFLICT` and app shows user message "Not enough space — choose alternate."

### 2.12 Localization & UX

- Strings in `strings.xml` for Android and shared strings in KMM for logic. Provide Tamil translations.
- Large fonts, high contrast, big buttons as specified.

### 2.13 Error handling & UX flows

- Clear, plain-language error messages (both EN and TA).
- If booking fails during sync, show explicit conflict resolution screen: options "Try another facility" / "Queue for auto-retry" / "Contact admin."

### 2.14 Testing

- Unit tests for Repositories (mock Ktor), DB migrations tests.
- UI tests for Compose screens (Espresso/Compose testing).
- Integration tests for sync flows using local test server.

---

# 3 — Data contracts & examples

### Sample API request/response (booking)

**`POST /api/v1/bookings`**

Request:

```json
{
  "facilityId": "a1b2-...",
  "cropType": "PADDY",
  "quantitySacks": 20,
  "startDate": "2025-11-05",
  "endDate": "2025-12-05"
}
```

Response (success):

```json
{
  "bookingId": "b-987",
  "status": "CONFIRMED",
  "availableAfterBooking": 430
}
```

Response (conflict):

```json
{
  "error": "INSUFFICIENT_CAPACITY",
  "available": 10,
  "message": "Only 10 sacks are available at selected facility."
}
```

### Sample API request/response (farmer registration)

`POST /api/v1/farmers/register`

Request:

```json
{
  "username": "farmer123",
  "password": "securepass",
  "name": "Raj Kumar",
  "aadhaarNumber": "1234-5678-9012",
  "district": "Coimbatore",
  "taluk": "Pollachi",
  "village": "Kinathukadavu",
  "address": "123 Main St, Kinathukadavu"
}
```

Response:

```json
{
  "farmerId": "f-123",
  "userId": "u-456",
  "verificationStatus": "PENDING",
  "message": "Registration successful. Please upload documents for verification."
}
```

---

## 4 — Operational concerns & scaling notes

- **Avoid overbooking:** always use DB-level transaction & decrement available capacity while locking facility row.
- **Seed updates:** ensure small size (only minimal fields) — do not ship heavy images in seed.
- **Data privacy:** do not store unnecessary PII; phone number stored but minimize sharing. Provide simple privacy statement in-app.
- **Offline capacity consistency:** server is source-of-truth for availability. Mobile only suggests availability; final confirmation happens when booking synced and server returns success.
- **Audit trails:** All critical operations are logged in history tables for compliance and debugging.

---

## 5 — Roadmap & MVP scope (engineering plan)

**MVP (one-week realistic, one-day hackathon minimal):**

- **Day-1 hackathon MVP:** KMM app with bundled seed JSON + offline booking capability (no server) + Compose UI (5 screens). Local sample booking with

simulated confirmation. (This is what you can present.)
- **Post-hack:** Implement Ktor backend, Postgres, real bookings, transactional capacity updates, seed delta APIs, admin endpoints, document upload/verification workflow.

**Stretch:**

- OTP-based verification
- Payment integration (if charging storage)
- Push notifications for booking status
- Sensor integration (moisture sensors) for automatic moisture updates

---

## 6 — Dev tasks / deliverables (to hand to team)

### Backend

- Ktor project skeleton with basic auth & migrations
- DB schema + Flyway migrations + initial seed loader
- Farmers API + document upload endpoint
- Agri dept approval/rejection endpoints
- Facilities API + availability transaction logic
- Booking endpoints + tests
- Storage records API
- Audit trail endpoints
- Seed API & admin endpoints

### Mobile (KMM + Android)

- KMM shared module skeleton (models, repo interfaces)
- SQLDelight schema + seed loader
- Ktor client + Basic Auth store
- Compose screens for registration, crop plantings, facilities, bookings
- Sync manager + offline booking queue
- Unit tests & instrumentation tests

---