# Unified App Blueprint – Grower & Agri Supply Chain (Replit)

Single, consolidated blueprint that merges the original slot‑booking MVP with the extensible addendum. Designed for Replit, Supabase (Postgres + Auth), and FastAPI (or Node as an alt). This file is the **source of truth** for how the app should work and evolve.

## 1) Product Overview

### 1.1 Problem

Packhouses must coordinate inbound deliveries from many growers without congestion or manual chaos (calls, WhatsApp, spreadsheets). They also need the option to expand into logistics, quality, compliance, and integrations.

### 1.2 Users & Roles

* **Admin / Grower Liaison (packhouse)** – Creates/edits slots, sets rules, blackouts, overrides; monitors schedule and inbound loads.
* **Grower (external)** – Views availability, books a time slot with quantity (resource unit), manages own bookings.
* **Ops (read‑only)** – Views upcoming schedule, exports data.
* **Future roles** – **Transporter, Buyer, Warehouse** via generic **Party** model.

### 1.3 MVP Scope (must‑have)

* Mobile‑first web app (PWA‑friendly) with **Day/Week** availability.
* **Transactional booking** with capacity controls and restrictions (by grower/cultivar/variant).
* Admin: **bulk slot generation**, **blackouts**, **notes**, **per‑slot restrictions**.
* Notifications v1: **email** confirmations; SMS/WhatsApp later.
* Multi‑tenant and RBAC foundation.

### 1.4 Near‑term Extensions (nice‑to‑have)

* **Consignments & Checkpoints** (turn bookings into tracked loads; gate/weigh/QC events).
* **Quality Inspections**, **Compliance docs** (GLOBALG.A.P, organic, phytosanitary).
* **Events & Outbox** for audit + integrations (ERP/WMS/weighbridge/ANPR).
* **Rules & Workflows** (quotas, state machines) as JSON configs.

## 2) Architecture

* **Frontend**: React (Vite) + TanStack Query + Tailwind. Feature flags for modules.
* **Backend**: FastAPI (Python) with asyncpg. (Node/Express alt OK; keep REST contracts.)
* **DB**: Supabase Postgres. Migrations in /infra. Optionally enable RLS later.
* **Auth**: Supabase Auth (magic link/OTP). JWT carries tenant\_id & roles.
* **Dev Hosting**: Replit (FE+BE). **Prod**: FE on Vercel, BE on Replit/Render/Fly, DB on Supabase.
* **Observability**: JSON logs, /health, UptimeRobot; Sentry optional.

Diagram:

React (PWA) ⇄ FastAPI (REST) ⇄ Supabase (Postgres + Auth)  
 ↳ Email (Resend/SendGrid)  
 ↳ Webhooks (ERP/WMS)

## 3) Monorepo Layout (Replit)

/app  
 /frontend  
 index.html  
 src/  
 main.tsx  
 App.tsx  
 api/client.ts # axios client w/ auth header  
 api/endpoints.ts # typed API calls  
 features/  
 booking/  
 logistics/  
 quality/  
 compliance/  
 reports/  
 core/ # auth, rbac, flags  
 styles/  
 /backend  
 main.py  
 db.py  
 security.py  
 schemas.py  
 routers/  
 auth.py  
 slots.py  
 bookings.py  
 restrictions.py  
 logistics.py # consignments + checkpoints  
 services/  
 events.py # domain\_events/outbox helpers  
 notifications.py  
 /infra  
 001\_init.sql  
 002\_seed.sql  
 101\_parties\_products.sql  
 102\_logistics.sql  
 103\_events\_rules.sql  
 .env  
 README.md

## 4) Data Model (SQL)

### 4.1 Core (MVP)

-- Tenants  
CREATE TABLE IF NOT EXISTS tenants (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 name text NOT NULL,  
 timezone text DEFAULT 'Africa/Johannesburg',  
 created\_at timestamptz DEFAULT now()  
);  
  
-- Growers (kept for MVP simplicity; can be replaced by parties view later)  
CREATE TABLE IF NOT EXISTS growers (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 name text NOT NULL,  
 contact text,  
 UNIQUE(tenant\_id, name)  
);  
  
-- Cultivars (useful now; can map to product\_variants later)  
CREATE TABLE IF NOT EXISTS cultivars (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 name text NOT NULL  
);  
  
-- Slots (capacity windows)  
CREATE TABLE IF NOT EXISTS slots (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 date date NOT NULL,  
 start\_time time NOT NULL,  
 end\_time time NOT NULL,  
 capacity numeric(10,2) NOT NULL,  
 resource\_unit text DEFAULT 'tons',  
 blackout boolean DEFAULT false,  
 notes text,  
 created\_by uuid,  
 CONSTRAINT slots\_time\_chk CHECK (end\_time > start\_time)  
);  
CREATE INDEX IF NOT EXISTS idx\_slots\_tenant\_date ON slots(tenant\_id, date);  
  
-- Per-slot restrictions  
CREATE TABLE IF NOT EXISTS slot\_restrictions (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 slot\_id uuid NOT NULL REFERENCES slots(id) ON DELETE CASCADE,  
 allowed\_grower\_id uuid REFERENCES growers(id),  
 allowed\_cultivar\_id uuid REFERENCES cultivars(id)  
);  
CREATE INDEX IF NOT EXISTS idx\_restr\_slot ON slot\_restrictions(slot\_id);  
  
-- Bookings  
CREATE TABLE IF NOT EXISTS bookings (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 slot\_id uuid NOT NULL REFERENCES slots(id) ON DELETE CASCADE,  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 grower\_id uuid NOT NULL REFERENCES growers(id),  
 cultivar\_id uuid REFERENCES cultivars(id),  
 quantity numeric(10,2) NOT NULL,  
 status text NOT NULL DEFAULT 'confirmed', -- confirmed/cancelled  
 created\_at timestamptz DEFAULT now()  
);  
CREATE INDEX IF NOT EXISTS idx\_bookings\_slot ON bookings(slot\_id);  
CREATE INDEX IF NOT EXISTS idx\_bookings\_tenant ON bookings(tenant\_id);  
  
-- Usage view  
CREATE OR REPLACE VIEW slot\_usage AS  
SELECT s.id AS slot\_id,  
 s.capacity,  
 COALESCE(SUM(CASE WHEN b.status='confirmed' THEN b.quantity END),0) AS booked,  
 s.capacity - COALESCE(SUM(CASE WHEN b.status='confirmed' THEN b.quantity END),0) AS remaining  
FROM slots s  
LEFT JOIN bookings b ON b.slot\_id = s.id  
GROUP BY s.id, s.capacity;

### 4.2 Extensibility (add‑on tables)

-- Parties (generic stakeholders)  
CREATE TABLE parties (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 name text NOT NULL,  
 type text NOT NULL, -- 'grower','transporter','buyer','warehouse'  
 contact jsonb DEFAULT '{}'  
);  
  
-- Products and variants (map cultivars later)  
CREATE TABLE products (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 sku text,  
 name text NOT NULL,  
 attributes jsonb DEFAULT '{}'  
);  
  
CREATE TABLE product\_variants (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 product\_id uuid NOT NULL REFERENCES products(id) ON DELETE CASCADE,  
 name text NOT NULL  
);  
  
-- Consignments and checkpoints (logistics module)  
CREATE TABLE consignments (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 booking\_id uuid REFERENCES bookings(id) ON DELETE SET NULL,  
 supplier\_party\_id uuid REFERENCES parties(id),  
 transporter\_party\_id uuid REFERENCES parties(id),  
 product\_variant\_id uuid REFERENCES product\_variants(id),  
 expected\_qty numeric(10,2),  
 received\_qty numeric(10,2),  
 unit text DEFAULT 'tons',  
 status text NOT NULL DEFAULT 'scheduled', -- scheduled|in\_transit|arrived|weighed|received|closed|rejected  
 metadata jsonb DEFAULT '{}',  
 created\_at timestamptz DEFAULT now()  
);  
  
CREATE TABLE checkpoints (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 consignment\_id uuid NOT NULL REFERENCES consignments(id) ON DELETE CASCADE,  
 type text NOT NULL, -- 'gate\_in','weigh\_in','weigh\_out','dock','qc'  
 ts timestamptz NOT NULL DEFAULT now(),  
 payload jsonb DEFAULT '{}'  
);  
  
-- Events & outbox (audit + integrations)  
CREATE TABLE domain\_events (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 aggregate\_type text NOT NULL, -- 'slot','booking','consignment','quality'  
 aggregate\_id uuid NOT NULL,  
 event\_type text NOT NULL, -- 'BOOKING\_CREATED','BOOKING\_CANCELLED', etc.  
 occurred\_at timestamptz DEFAULT now(),  
 payload jsonb NOT NULL  
);  
  
CREATE TABLE outbox (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 event\_id uuid NOT NULL REFERENCES domain\_events(id) ON DELETE CASCADE,  
 target text NOT NULL, -- 'webhook:erp','queue:analytics'  
 status text NOT NULL DEFAULT 'pending',  
 attempts int DEFAULT 0,  
 last\_error text  
);  
  
-- Rules & Workflows (JSON-configurable)  
CREATE TABLE rules (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 name text NOT NULL,  
 scope text NOT NULL, -- 'booking','consignment','quality'  
 definition jsonb NOT NULL,  
 enabled boolean DEFAULT true  
);  
  
CREATE TABLE workflows (  
 id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tenant\_id uuid NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,  
 name text NOT NULL, -- 'InboundDelivery'  
 states jsonb NOT NULL, -- state graph + transitions  
 enabled boolean DEFAULT true  
);

Compatibility: you may create a view growers\_v from parties where type='grower' later to unify UI without breaking MVP code.

## 5) Booking Concurrency (Required)

**SQL pattern:**

BEGIN;  
SELECT capacity,  
 COALESCE((SELECT SUM(quantity) FROM bookings WHERE slot\_id=$1 AND status='confirmed'),0) AS booked  
FROM slots  
WHERE id=$1  
FOR UPDATE; -- lock row  
  
-- if $qty <= (capacity - booked) INSERT booking; else ROLLBACK  
COMMIT;

**Event emission (after commit):** insert into domain\_events (BOOKING\_CREATED) and one outbox row for later webhook delivery.

## 6) API Contracts (v1)

### 6.1 Auth

POST /auth/login -> { token }  
GET /auth/me -> { user, roles, tenant\_id }

### 6.2 Slots

GET /v1/slots?date=YYYY-MM-DD  
POST /v1/slots/bulk # create slots for date/time window  
PATCH /v1/slots/{id} # blackout/capacity/notes  
GET /v1/slots/{id}/usage # { capacity, booked, remaining }

**Slot (response)**

{  
 "id":"uuid","tenant\_id":"uuid","date":"2025-08-13",  
 "start\_time":"08:00:00","end\_time":"09:00:00",  
 "capacity":20.0,"resource\_unit":"tons","blackout":false,  
 "notes":"Maintenance at 12:00",  
 "restrictions":{"growers":["uuid"],"cultivars":["uuid"]}  
}

### 6.3 Bookings

POST /v1/bookings  
DELETE /v1/bookings/{id}  
GET /v1/bookings?date=&grower\_id=

**Create request**

{ "slot\_id":"uuid", "grower\_id":"uuid", "cultivar\_id":"uuid", "quantity":5.0 }

**Errors**: 409 (capacity exceeded), 403 (slot restricted), 404 (slot not found).

### 6.4 Restrictions

POST /v1/restrictions/apply

{ "date":"2025-08-13", "slot\_id":null, "grower\_ids":["uuid"], "cultivar\_ids":["uuid"], "note":"A4 only PM" }

### 6.5 Logistics (extensible)

POST /v1/logistics/consignments # create from booking\_id  
GET /v1/logistics/consignments?date=YYYY-MM-DD # list + last checkpoint  
POST /v1/logistics/consignments/{id}/checkpoints

**Checkpoint payload**

{ "type":"gate\_in", "payload":{ "plate":"ABC123" } }

## 7) Frontend Plan

* Replace wireframe’s local state with API calls (TanStack Query cache + invalidation).
* Pages:
  + **Dashboard (Day/Week)** – availability grid with status dots/badges.
  + **Book** – modal flow; quantity + cultivar/variant.
  + **Admin Slots** – bulk generation, blackout, restrictions, notes.
  + **Inbound (feature‑flagged)** – read‑only consignments list with latest checkpoint.
* RBAC guards: show Admin tools only if role==='admin'.
* **Feature flags** via import.meta.env.VITE\_FEATURE\_\*.

## 8) Security & Tenancy

* All queries filter by tenant\_id.
* JWT claims: sub, tenant\_id, roles[], optional party\_id.
* Growers only see bookings/consignments where they are the supplier.
* Input validation: Pydantic (FastAPI) or Zod (Node).
* Rate limiting, CORS allowlist.
* Future: enable RLS; map UI growers to parties(type='grower').

## 9) Notifications

* Start with email (Resend/SendGrid) on booking create/cancel.
* Template variables: tenant, slot time, quantity, unit, notes.
* Later: WhatsApp/SMS, push (PWA), webhook to ERP.

## 10) DevOps

* **Env/Secrets**: DATABASE\_URL, JWT\_SECRET, RESEND\_API\_KEY, SUPABASE\_URL, SUPABASE\_ANON\_KEY.
* **Migrations**:
  + 001\_init.sql, 002\_seed.sql (core MVP tables & dev data)
  + 101\_parties\_products.sql, 102\_logistics.sql, 103\_events\_rules.sql (extensibility)
* **Health**: /health returns { status: 'ok' }.
* **Backups**: Supabase PITR if available.

## 11) Testing

* **Unit**: validators, capacity math.
* **Integration**: two concurrent booking requests on same slot → one must fail with 409.
* **E2E**: login → view day → book → cancel → admin blackout → verify UI.

## 12) Rollout

**Phase 0** – Core up & running (slots, bookings, admin, email).  
**Phase 1** – Restrictions, week view, CSV export, quotas.  
**Phase 2** – Logistics: consignments + checkpoints (read‑only UI), domain events/outbox.  
**Phase 3** – Quality & compliance; integrations; analytics dashboards.

Acceptance (MVP):

* Atomic booking (no overbooking), admin slot tools, grower self‑service, email confirmations, tenant scoping.

## 13) Replit Assistant Prompt (use this exact block)

**Goal:** Implement this unified blueprint in a Replit monorepo with React (Vite) frontend and FastAPI backend wired to Supabase. Keep v1 slot‑booking MVP intact, and add the extensibility tables + minimal logistics endpoints and event emission.

**Tasks:**

1. Create folders exactly as specified in Section 3.
2. Initialize Vite React + Tailwind in /frontend.
3. Initialize FastAPI in /backend with routers: auth.py, slots.py, bookings.py, restrictions.py, logistics.py.
4. Connect to Supabase Postgres via asyncpg (DATABASE\_URL env).
5. Implement endpoints per Section 6, ensuring booking uses SELECT ... FOR UPDATE and returns 409 on capacity exceeded.
6. After /v1/bookings success, emit a BOOKING\_CREATED row into domain\_events and one into outbox.
7. Add migrations in /infra using the SQL in Section 4 (both MVP and extensibility files). Provide a script/README snippet to run them.
8. Frontend: replace local state with real API calls; build **Inbound** page (read‑only list of consignments + last checkpoint) behind VITE\_FEATURE\_LOGISTICS=true.
9. Provide .env.example with all required env vars.
10. Add run scripts: npm run dev for FE; uvicorn backend.main:app --reload --port 8000 for BE.

## 14) Node/Express Alternative (optional)

* Use pg, express, zod, express-rate-limit.
* Keep identical REST shape and SQL concurrency pattern.

## 15) Future Modules (non‑blocking)

* **Quality Inspections**: templates + pass/fail gates before receiving.
* **Compliance Docs**: expiry alerts & booking/receiving checks.
* **Inventory/Traceability**: link consignments → lots/batches → SKUs.
* **Optimization**: suggest slots based on throughput & historical data.
* **Internationalization**: en/af; time‑zone aware scheduling.

**This file replaces separate blueprints and should be saved at repo root as “Blueprint.md”**