Phase 0 Status

Phase 0 Status — Foundation Complete

MC: #104223 | **Validation MC:** #104225 | **Date:** 2026-06-22 | **Proveo Verdict:** PASS (7/7 tests green)

Status: COMPLETE

Phase 0 scaffold and foundation delivered and independently validated by Proveo (Angie Jones) with real executed evidence.

Exit Criteria — All Met

- ✓ CI green (lint + compileKotlin + test)
- ✓ `docker-compose up` boots API+DB+Unleash
- ✓ `/health` endpoint returns 200 with RLS self-check
- ✓ Fail-closed startup: app refuses to start if `qody_app` has BYPASSRLS
- ✓ Two-venue RLS isolation test PASS (reads isolated, cross-tenant INSERT rejected)
- ✓ 3 MFE shells deployable independently

Deliverables

Repo Scaffold

- Gradle Kotlin/Ktor project structure (per `~/system/blueprints/types/kotlin-ktor.json`)
- `.gitignore`, `.env.example`, `BUILD-BLUEPRINT.md`
- CI config: GitHub Actions (lint, compile, test)
- `docker-compose.yml`: Postgres 16 + Unleash + app service

Database Foundation

- Flyway V1 baseline migration: `organization`, `venue`, `restaurant_table`, `staff`, `role`
- RLS ENABLED + FORCED on `restaurant_table` and `staff`
- Two DB roles:
  - `qody_flyway`: DDL/migration owner (BYPASSRLS allowed, NOT used at runtime)
  - `qody_app`: Runtime role (NOBYPASSRLS, NOT table owner)
- RLS policies:
  - PERMISSIVE ALL policy: `venue_id = current_setting('app.current_venue_id', true)::uuid`
  - RESTRICTIVE INSERT policy: prevents cross-tenant writes

API Foundation

- Ktor app with `/health` endpoint
- HikariCP connection pool (Phase 1: wire `SET ROLE qody_app` in `connectionInitSql`)
- Fail-closed RLS role verification on boot:

```
fun verifyRlsRoleFailClosed() {
    val result = transaction {
        exec("SELECT rolname, rolbypassrls FROM pg_roles WHERE rolname = 'qody_app'")
{ rs ->
            if (rs.next()) {
                val bypassRls = rs.getBoolean("rolbypassrls")
                if (bypassRls) {
                    throw IllegalStateException(
                        "SECURITY VIOLATION: qody_app has BYPASSRLS. App refuses to
start."
                    )
                }
            }
        }
    }
    logger.info("RLS self-check PASS: qody_app has BYPASSRLS=false")
}
```

Frontend Foundation

- 3 MFE shells (Vite + React):
  - `guest-mfe/`: Public QR menu (port 5173)
  - `staff-mfe/`: Kitchen/staff board (port 5174)
  - `admin-mfe/`: Venue dashboard (port 5175)
- Each MFE independently deployable (separate build/deploy)

Validation Evidence (Proveo)

Test 1: /health Check — RLS Role Self-Check (PASS)

```
curl -s -i http://localhost:8088/health

HTTP/1.1 200 OK
{
  "status":"ok",
  "version":"0.1.0",
  "db":{
    "connected":true,
    "rlsRoleCheck":{
      "role":"qody_app",
      "bypassRls":false,
      "status":"PASS"
    }
  }
}
```

Verdict: PASS. HTTP 200. `rlsRoleCheck.bypassRls=false`, `status="PASS"`. qody_app confirmed NOBYPASSRLS at runtime.

Test 2: RLS ENABLED + FORCED on Tenant Tables (PASS)

```
SELECT relname AS table_name, relrowsecurity AS rls_enabled, relforcerowsecurity AS rls_forced
FROM pg_class WHERE relname IN ('restaurant_table', 'staff') ORDER BY relname;

    table_name    | rls_enabled | rls_forced
------------------+-------------+------------
 restaurant_table | t           | t
 staff            | t           | t
(2 rows)
```

Verdict: PASS. Both tenant tables have RLS ENABLED (t) and FORCED (t).

Test 3: RLS Policies — PERMISSIVE USING + RESTRICTIVE INSERT (PASS)

```
SELECT tablename, policyname, permissive, cmd
FROM pg_policies WHERE tablename IN ('restaurant_table', 'staff') ORDER BY tablename,
policyname;

    tablename     |            policyname             | permissive  |  cmd
------------------+-----------------------------------+-------------+--------
 restaurant_table | tenant_insert_restaurant_table    | RESTRICTIVE | INSERT
 restaurant_table | tenant_isolation_restaurant_table | PERMISSIVE  | ALL
 staff            | tenant_insert_staff               | RESTRICTIVE | INSERT
 staff            | tenant_isolation_staff            | PERMISSIVE  | ALL
(4 rows)
```

Verdict: PASS. Both tables have PERMISSIVE USING policy (filters reads) and RESTRICTIVE INSERT policy (rejects cross-tenant writes).

Test 4: Two-Venue RLS Isolation (Core Tenant Isolation Test)

Setup (as qody_flyway / table owner):

```
venue A: id=6d1b9c47-c088-4808-8473-e8b1672c7acc  name="Alpha Bistro"
venue B: id=fcf66a03-ef67-41bd-9d6b-348b0ee9908a  name="Beta Grill"

restaurant_table rows seeded:
  Table A1 -> venue A
  Table A2 -> venue A
  Table B1 -> venue B
  Table B2 -> venue B
```

Test 4a: Context = venue A — venue B rows INVISIBLE (as qody_app)

```
BEGIN;
SET LOCAL app.current_venue_id = '6d1b9c47-c088-4808-8473-e8b1672c7acc';
SELECT label, venue_id FROM restaurant_table ORDER BY label;
ROLLBACK;
```

```
  label   |              venue_id
----------+--------------------------------------
 Table A1 | 6d1b9c47-c088-4808-8473-e8b1672c7acc
 Table A2 | 6d1b9c47-c088-4808-8473-e8b1672c7acc
(2 rows)
```

Verdict: PASS. Only 2 venue-A rows returned. Venue B rows (Table B1, Table B2) are invisible.

Test 4b: Context = venue A — INSERT with venue_id=B REJECTED (as qody_app)

```
BEGIN;
SET LOCAL app.current_venue_id = '6d1b9c47-c088-4808-8473-e8b1672c7acc';
INSERT INTO restaurant_table (venue_id, label, qr_token_id, capacity)
  VALUES ('fcf66a03-ef67-41bd-9d6b-348b0ee9908a', 'Smuggled B3', 'qr-smuggled', 2);
ROLLBACK;

ERROR:  new row violates row-level security policy for table "restaurant_table"
```

Verdict: PASS. Cross-tenant INSERT correctly rejected by RESTRICTIVE insert policy.

Test 4c: Context = venue B — venue A rows INVISIBLE (symmetric isolation)

```
BEGIN;
SET LOCAL app.current_venue_id = 'fcf66a03-ef67-41bd-9d6b-348b0ee9908a';
SELECT label, venue_id FROM restaurant_table ORDER BY label;
ROLLBACK;

  label   |              venue_id
----------+--------------------------------------
 Table B1 | fcf66a03-ef67-41bd-9d6b-348b0ee9908a
 Table B2 | fcf66a03-ef67-41bd-9d6b-348b0ee9908a
(2 rows)
```

Verdict: PASS. Only 2 venue-B rows returned. Venue A rows (Table A1, Table A2) invisible.

Test 5: No Context Set — Zero Rows Returned (PASS)

```
-- As qody_app, no SET of app.current_venue_id
SELECT label, venue_id FROM restaurant_table ORDER BY label;
```

```
 label | venue_id
-------+----------
(0 rows)
```

Verdict: PASS. Fail-safe: no context = no rows returned. No cross-tenant data leakage.

Test 6: Fail-Closed Negative — BYPASSRLS Simulation (PASS)

Step 1: Grant BYPASSRLS to qody_app (as qody_flyway)

```
ALTER ROLE qody_app BYPASSRLS;

SELECT rolname, rolbypassrls FROM pg_roles WHERE rolname = 'qody_app';

 rolname  | rolbypassrls
----------+--------------
 qody_app | t
(1 row)
```

Step 2: Prove /health returns HTTP 500 with BYPASSRLS active (live app)

```
curl -s -i http://localhost:8088/health

HTTP/1.1 500 Internal Server Error
{
  "status":"degraded",
  "version":"0.1.0",
  "db":{
    "connected":true,
    "rlsRoleCheck":{
      "role":"qody_app",
      "bypassRls":true,
      "status":"FAIL"
    }
  }
}
```

Verdict: PASS. /health correctly returns HTTP 500 + `status:"FAIL"` when BYPASSRLS is active.

Step 3: Prove the Bilko breach — BYPASSRLS silently exposes all tenant data

```
-- As qody_flyway with SET ROLE qody_app (who now has BYPASSRLS)
SET ROLE qody_app;
SET LOCAL app.current_venue_id = '6d1b9c47-c088-4808-8473-e8b1672c7acc'; -- context = venue A
SELECT label, venue_id FROM restaurant_table ORDER BY label;

  label   |               venue_id
----------+--------------------------------------
 Table A1 | 6d1b9c47-c088-4808-8473-e8b1672c7acc
 Table A2 | 6d1b9c47-c088-4808-8473-e8b1672c7acc
 Table B1 | fcf66a03-ef67-41bd-9d6b-348b0ee9908a
 Table B2 | fcf66a03-ef67-41bd-9d6b-348b0ee9908a
(4 rows)
```

Evidence: With BYPASSRLS, even with `app.current_venue_id` scoped to venue A, ALL 4 rows across both venues are returned. This is the exact Bilko breach reproduced. The fail-closed /health check is not cosmetic — it is the guard against this silent breach.

Step 4: Restore safe state

```
ALTER ROLE qody_app NOBYPASSRLS;

curl -s -i http://localhost:8088/health
-> HTTP/1.1 200 OK ... "bypassRls":false,"status":"PASS"
```

Verdict: PASS. Reverted cleanly. /health confirms restored to safe state.

Summary of Non-Negotiables (All Verified)

#	Requirement	Verified	Evidence
1	qody_app NOBYPASSRLS + not table owner + fail-closed startup	PASS	Test 1 + startup log
1	fail-closed at boot (before Netty)	PASS	startup log lines 12-13
1	/health 500 if BYPASSRLS active	PASS	Test 6 step 2

#	Requirement	Verified	Evidence
2	RLS ENABLED+FORCED on restaurant_table, staff	PASS	Test 2
2	PERMISSIVE USING + RESTRICTIVE INSERT policies	PASS	Test 3
2	Two-venue isolation: B invisible when context=A	PASS	Test 4a
2	Cross-tenant INSERT rejected	PASS	Test 4b
2	Symmetric: A invisible when context=B	PASS	Test 4c
2	No context = zero rows (fail-safe)	PASS	Test 5
–	Bilko breach reproduced + guarded against	PROVEN	Test 6 step 3

Gaps / Phase 1 Actions

1. **Runtime role switch not yet wired:** The app currently connects to Postgres as `qody_flyway` (the owner/DDL role) for both Flyway migrations AND runtime queries. Phase 1 must wire `connectionInitSql = "SET ROLE qody_app"` in HikariCP config before any data-carrying endpoint is live.
2. **Flyway baseline note:** The V1 migration was applied manually (no Flyway schema history table initially). For production/CI this must be handled via `flyway.baselineOnMigrate=true` in initial deploy or by ensuring Flyway runs against a clean DB.

Evidence Files

- `/tmp/evidence-104222/proveo-rls-validation-phase0.md` — Full Proveo validation report (360 lines, real executed evidence)
- `/tmp/evidence-104222/petter-architecture.md` — Full architecture spec (435 lines)
- `/tmp/evidence-104222/QODY-MASTER-PLAN.md` — Synthesis doc (71 lines)

Next Phase

Phase 1 — MVP Vertical Slice (MC #104224): QR → menu → order → pay → kitchen → served (the demo).

Exit Criteria: Live Proveo E2E (browser, real evidence — not dry-run) of full flow; RLS isolation E2E green; QA-19 ≥ 17.

Revision #1
Created 2026-06-22 15:50:56 UTC by John
Updated 2026-06-22 15:50:56 UTC by John