QODY Architecture & Operations

QODY Architecture & Operations

Product: QODY — AI-native QR-based restaurant ordering and payment platform
Operator: SnowIT d.o.o. Sarajevo (powered by ALAI Holding AS)
Market: Bosnia and Herzegovina (BiH) primary, Norway secondary
Status: LIVE — Demo environment (rg-qody-demo), Clean production (rg-qody-prod)
Updated: 2026-06-26

What is QODY?

QODY is a **sit / order & pay** web application for hospitality venues (restaurants, cafes, bars). Guests scan a QR code at their table, browse a digital menu, place an order, and pay — all without installing an app or creating an account. Orders appear in real-time on the kitchen display system (KDS), and staff manage the order lifecycle through to delivery.

Core Flow

1. Guest scans QR code on table → loads venue menu (anonymous, no login)
2. Guest builds cart → submits order → pays via Stripe (test) or Monri (BiH payment gateway, when live)
3. Order instantly appears on kitchen display (WebSocket + SSE realtime)
4. Staff accepts → preps → marks ready → delivers
5. Guest receives receipt, can track status live

Architecture Overview

Tech Stack

- **Backend:** Kotlin 2.1.0 + Ktor 3.0.3 + PostgreSQL 16 + Flyway + Exposed ORM + JWT
- **Frontend:** 3 Vite 6 + React 19 + TypeScript micro-frontends (MFEs) + Astro 5 landing
- **Database:** PostgreSQL 16 with Row-Level Security (RLS) enforced, qody_app role NOBYPASSRLS
- **Realtime:** WebSocket + SSE (Server-Sent Events), transactional outbox pattern
- **Payments:** Stripe Connect (live test mode), Monri architecture ready (Model B)
- **Infrastructure:** Azure Container Apps (ACA), swedencentral region, managed Postgres Flexible Server
- **CI/CD:** Azure Pipelines (local Docker fallback when ACR build stalls)

The 4 Apps + Landing

App	Purpose	DNS (demo)	Port (local)
qody-api	Backend: orders, payments, auth, realtime hub	api.qody.alai.no	8080
qody-guest	Guest ordering (anonymous)	qody.alai.no	5173
qody-admin	Venue manager dashboard (menu CRUD, QR gen, reports, settings)	admin.qody.alai.no	5175
qody-staff-kitchen	Kitchen display system (KDS) — live order board	kuhinja.qody.alai.no	5174
landing	Marketing site (premium "Concept A" design)	(future qody.ba)	N/A

Deployment Environments

Demo/Stage (LIVE)

- **Resource Group:** rg-qody-demo
- **Region:** swedencentral
- **Subscription:** 5b0b4d9b-e677-464e-abf0-5170cbce3b8e (Azure subscription 1)
- **Container Registry:** qodydemoacr.azurecr.io
- **Database:** qody-demo-db.postgres.database.azure.com (Postgres 16, Standard_B1ms)
- **URLs:** qody.alai.no (guest), api.qody.alai.no (API), admin.qody.alai.no (admin), kuhinja.qody.alai.no (kitchen)

Production/Clean (DEPLOYED, no custom DNS yet)

- **Resource Group:** rg-qody-prod
- **Container Registry:** qodyprodacr.azurecr.io
- **Database:** qody-prod-db.postgres.database.azure.com
- **No demo seed** — clean production environment

- **Custom domain pending:** Waiting on qody.ba domain registration (Asmir/SnowIT)

Note: rg-qody-demo is the live demo CEO/Asmir use. Always deploy there for active testing. rg-qody-prod exists but nothing points to it yet.

Feature Catalog (Talas 0?6)

Built over 6 waves (2026-06-22 to 2026-06-24):

Phase 0 (Foundation)

- Gradle Kotlin/Ktor scaffold
- Postgres RLS enforced (qody_app NOBYPASSRLS)
- /health endpoint with RLS self-check (fail-closed)
- 3 MFE shells (Vite) deployable independently
- CI: lint + compileKotlin + test

Phase 1 (Vertical Slice)

- Guest order flow (QR resolve → menu → cart → submit → pay)
- Stripe test payments
- KDS realtime (WebSocket + SSE)
- Admin menu CRUD

Phase 2 (AI + Payments)

- Groq AI chat integration
- Stripe Connect per-venue (destination charge + 0.5% application fee)
- Subscription billing (39 KM Pro / 49 KM Enterprise per month)

Talas 1 (Foundation++)

- Audit log (V11: audit_log table with RLS)
- Address field on venue (JSONB)
- Area/zone hierarchy (venue → area → table)
- 10 payment statuses (PRD-compliant: pending_payment, authorized, paid, failed, cancelled, expired, refund_requested, partially_refunded, refunded, voided)
- Order numbering (order_number TEXT with trigger, e.g., Q-QODYDE-000001)

Talas 2A (Super-Admin Backend)

- Super-admin backend services (SuperAdminService, RefundService, EnhancedSalesReportService)
- Refund request workflow (staff → admin approve/reject → gateway call)
- CSV export with all PRD §12.2 fields (order_number, merchant, location, table, datetime, subtotal, tip, total, payment_status, gateway_reference, order_status, refund_amount)
- Enhanced receipt generation

Talas 2B (Super-Admin UI)

- Super-admin panel (5 tabs: merchants, transactions, billing, refunds, audit)
- Sold-out toggle endpoint (PATCH /admin/menus/items/{id}/availability)
- Staff audio alerts (Web Audio API synthetic beep on new orders)
- Multilang BS-default (Bosnian default for admin + kitchen apps)

Talas 3 (qLub Parity)

- Google Review CTA (post-order, configurable URL per venue)
- Pay-at-table option (pending_cash payment status, order sent to kitchen before payment)
- Configurable tip presets (venue.tip_presets JSONB, default [0,5,10,15])
- Waiter-call button (guest → staff WS push notification)

Talas 4 (Handover + Geofence)

- Per-order handover QR (TOTP-like delivery confirmation, HandoverService, staff scans guest QR to mark delivered)
- Geofence soft opt-in (default OFF, graceful fallback, never hard-block — CEO directive)
- i18n gap-fill (19 missing T3 keys + admin.settings.* keys)

Talas 5 (Gap-Fill)

- CSV table column fix (table_session → restaurant_table join, populates table label in export)
- Email receipt graceful stub (SMTP not configured → logs request, returns 200 with note)
- QR token regeneration endpoint (PATCH /admin/tables/{id}/regenerate-qr, deactivates old token)
- Staff print button (window.print popup with monospace order ticket)
- KDS payment gate (no pending_payment orders visible to staff)

Talas 6 (M-07 Operating Hours)

- Operating hours / ordering schedule
- venue.ordering_enabled (master toggle)

- venue_operating_hours table (day_of_week, open_time, close_time, is_closed, RLS enforced)
- V15 migration
- Demo venue seeded 24/7 (never blocked during demos)

Additional Features (Security, Payments, I18n)

- **Password hashing:** argon2id (replaced SHA-256 in commit 40071db)
- **Step-up 2FA:** TOTP (RFC 6238) for sensitive mutations (fee changes, refunds, staff delete)
- **RLS isolation:** Postgres RLS policies enforce tenant_id boundary, fail-closed
- **JWT auth:** HS256 signing, distinct secrets for JWT vs QR tokens
- **Payments:** Stripe Connect (live test mode), Monri architecture ready (Model B: per-venue merchant accounts)
- **Internationalization:** BS (Bosnian), HR (Croatian), SR (Serbian), EN (English) — full coverage
- **Easy Mode:** Mobile-optimized admin dashboard (touch-friendly for phone/tablet)
- **Market-aware payments:** BiH-simple flow, EU-advanced flow (configurable per venue)

Database Schema

Migrations

V1–V19 in apps/api/src/main/resources/db/migration/

- **V1:** Baseline (organization, venue, table, staff, role, RLS policies)
- **V2:** Domain (menu, order, payment, modifier, translation)
- **V10:** PRD fee model (platform_fee_pct, monthly_fee on venue)
- **V11:** Talas 1 (audit log, address, area/zone, payment statuses, order numbering)
- **V12:** Talas 2A (super-admin backend, refunds, enhanced sales)
- **V13:** Talas 3 (waiter-call, tip presets, pay-at-table)
- **V14:** Talas 4 (handover QR, geofence)
- **V15:** Talas 6 (operating hours, venue.ordering_enabled, venue_operating_hours table)
- **V16:** Fix wave A (audit log field mismatch, websocket reconnect)
- **V17:** Market field (BiH-simple vs EU-advanced payment flows)
- **V18:** Step-up 2FA (TOTP, totp_secret on staff table)
- **V19:** Subscription billing (Stripe subscription tracking)

Key Tables

- **organization, venue** (with platform_fee_pct, monthly_fee, ordering_enabled, address JSONB, branding JSONB)

- **area** (zones within venue, V11)
- **restaurant_table** (with area_id nullable, qr_token_id FK)
- **qr_token** (signed HMAC tokens, nonce for uniqueness)
- **menu, category, menu_item, menu_item_translation** (multilang BS/HR/SR/EN)
- **modifier_group, modifier**
- **order** (with order_number TEXT, payment_status, handover_code, delivered_at)
- **order_line, order_line_modifier** (price snapshots)
- **payment** (unique index on provider + provider_payment_id)
- **table_session** (guest → table → order linkage)
- **staff** (argon2id password_hash, totp_secret for 2FA)
- **role** (OWNER, MANAGER, WAITER, KITCHEN, SUPERADMIN)
- **audit_log** (actor_user_id, action, entity_type, entity_id, metadata JSONB, RLS enforced)
- **waiter_call** (V13: guest call-waiter feature)
- **refund_request** (V12: staff → admin approval workflow)
- **venue_operating_hours** (V15: day_of_week, open_time, close_time, is_closed, RLS)
- **subscription** (V19: Stripe subscription tracking for billing)

Demo Seed

V7__demo_seed_idempotent.sql creates "QODY Demo Bistro" with menu, 2 tables, QR tokens, staff accounts. Idempotent (ON CONFLICT UPSERT). Loads only when ENV=demo.

Payment Architecture

Stripe Connect (Live Test Mode)

- **Model:** Per-venue connected accounts (destination charge + 0.5% application fee)
- **Fee:** 0.5% platform fee (configurable per venue, max 0.7%), guest pays menu total, restaurant nets total−fee, QODY keeps fee
- **Subscription:** 39 KM Pro / 49 KM Enterprise per month (Stripe BAM billing)
- **Onboarding:** Express hosted onboarding (Custom account for demo)
- **Prod hardening pending:** Use Express account type, production keys

Monri (Architecture Ready, Model B)

- **Model B:** Per-venue Monri merchant accounts (bank-agnostic)
- **Banks:** Raiffeisen, Atos, UniCredit, Intesa Sanpaolo (restaurant chooses their own bank)
- **Settlement:** Money flows restaurant-bank → restaurant directly (QODY does NOT hold funds)
- **QODY fee:** Invoiced separately (monthly: 29-49 KM + 0.5% of venue's Qody-channel revenue)

- **Credentials:** Per-venue Monri merchant_id + auth_token encrypted in DB (Azure Key Vault future)
- **Frontend:** Monri.js integration pending (currently Stripe Elements)
- **Test credentials:** Form sent to Monri AM Mahir Mulaomerović (+387 66 284 773)
- **Production:** Each restaurant signs own Monri + bank merchant agreement (7-10 days onboarding)

Recommendation: Start with Model B manual invoicing (certain, no dependency). Explore Monri split-payment API if they confirm support.

Regulatory Compliance (BiH): QODY is a software service provider, NOT a payment intermediary. No e-money/PI license needed under Model B. PCI-DSS scope = SAQ-A (Monri.js hosted checkout). BiH payment lawyer consult recommended (~500 EUR, confirm no PI license needed).

Security

- **Password hashing:** argon2id (Argon2id variant, replaced SHA-256)
- **Step-up 2FA:** TOTP (RFC 6238) for sensitive mutations (fee changes, refunds, staff delete, branding edits)
- **RLS isolation:** Postgres RLS policies enforce venue_id boundary, qody_app role NOBYPASSRLS, /health verifies bypassRls=false on startup (fail-closed)
- **JWT auth:** HS256 signing, 64 hex secrets (JWT_SECRET, QR_TOKEN_SECRET distinct)
- **CORS:** Configured per environment (demo origins vs prod origins)
- **Webhook verification:** Signed webhooks (HMAC), idempotency via unique index on (provider, provider_payment_id)

Internationalization

- **Languages:** BS (Bosnian), HR (Croatian), SR (Serbian), EN (English)
- **Scope:** All guest UI, all staff/kitchen UI, all admin UI (full coverage post-Talas 2B/3/4)
- **Default:** BS for BiH market, EN for Norway/international
- **Implementation:** i18n/bs.json, hr.json, sr.json, en.json; menu_item_translation table (V5); LangSwitcher component

Deployment Procedure

Build Images (Local Docker, amd64 Required for ACA)

```
cd apps/api
docker buildx build --platform linux/amd64 -t qodydemoacr.azurecr.io/qody-api:$(git rev-parse
--short HEAD) .

cd ../guest
docker buildx build --platform linux/amd64 \
  --build-arg VITE_API_BASE_URL=https://api.qody.alai.no \
  --build-arg VITE_STRIPE_PUBLISHABLE_KEY=pk_test_... \
  -t qodydemoacr.azurecr.io/qody-guest:$(git rev-parse --short HEAD) .

# Repeat for admin and staff-kitchen with VITE_API_BASE_URL
```

Push to ACR

```
az acr login --name qodydemoacr
docker push qodydemoacr.azurecr.io/qody-api:$(git rev-parse --short HEAD)
docker push qodydemoacr.azurecr.io/qody-guest:$(git rev-parse --short HEAD)
# ... push admin and staff-kitchen
```

Update Container Apps

```
TAG=$(git rev-parse --short HEAD)

# API (creates new revision, verify before promoting)
az containerapp update \
  --name qody-api \
  --resource-group rg-qody-demo \
  --image qodydemoacr.azurecr.io/qody-api:$TAG

# Verify /health returns 200 + bypassRls=false
curl https://api.qody.alai.no/health

# Promote to 100% traffic (or use traffic split for canary)
az containerapp ingress traffic set \
  --name qody-api \
  --resource-group rg-qody-demo \
  --revision-weight qody-api--0000XXX=100
```

```
# Guest/Admin/Kitchen (single-revision mode, auto-100%)
az containerapp update --name qody-guest --resource-group rg-qody-demo \
  --image qodydemoacr.azurecr.io/qody-guest:$TAG
az containerapp update --name qody-admin --resource-group rg-qody-demo \
  --image qodydemoacr.azurecr.io/qody-admin:$TAG
az containerapp update --name qody-staff-kitchen --resource-group rg-qody-demo \
  --image qodydemoacr.azurecr.io/qody-staff-kitchen:$TAG
```

Post-Deploy Verification (ZAKON PI2)

```
curl https://api.qody.alai.no/health  # Must return
{"status":"ok","db":{"rlsRoleCheck":{"bypassRls":false,"status":"PASS"}}}
curl -I https://qody.alai.no          # 200 + HTML
curl -I https://admin.qody.alai.no    # 200 + HTML
curl -I https://kuhinja.qody.alai.no  # 200 + HTML

# Real browser UAT: load guest menu, add item to cart, verify API calls work (not localhost)
```

?? CRITICAL: MFE Build Args

All three MFEs (guest, admin, staff-kitchen) **MUST** receive `VITE_API_BASE_URL` at build time. Vite bakes env vars into the bundle. If missing, the MFE defaults to `localhost:8080` → deploy succeeds, page loads, but API calls fail. **Always verify** build logs show:

```
===== QODY <MFE> BUILD: API base resolves to https://api.qody.alai.no =====
```

Recurring Lessons (Do NOT Re-Break)

1. **Build-arg or localhost bakes in:** MFE Dockerfiles default VITE_API_BASE_URL=http://localhost:8080 if not passed. Deploy looks fine (200), but guest can't reach API. Always pass --build-arg VITE_API_BASE_URL=https://api.qody.alai.no.
2. **Webhook / system DB calls MUST bypass RLS:** Any background job or webhook handler that queries across venues (e.g., SuperAdminService, billing export, webhook sync) must use the admin datasource (BYPASSRLS connection), NOT the per-request tenant-scoped datasource. Symptom: webhook succeeds but order not updated (RLS blocked the write).
3. **i18n single top-level key:** All i18n keys must be unique at root level (e.g., guest.menu.addToCart, not nested menu: { addToCart: ... }). The i18n library flattens keys. Duplicate keys across modules = one overwrites the other.

4. **Idempotent seed migrations:** Seed migrations (demo venue, tables, menu) must use ON CONFLICT ... DO UPDATE or existence guards that run AFTER dependent rows exist. Existence guards that check BEFORE the row is created silently skip INSERTs → "deploy OK" but data missing.
5. **Landing scroll-reveal fail-safe:** Intersection Observer animations must have a fallback timeout (fade-in after 3s) or IntersectionObserver polyfill. Safari sometimes doesn't fire isIntersecting on initial load.
6. **Verify by live outcome, not green build:** curl 200 + green CI ≠ "works". Always run real-browser UAT (Playwright or manual) that exercises the feature end-to-end (e.g., guest menu → add item → checkout → order reaches KDS).

Secrets & Credentials

Storage: Bitwarden items (alem@alai.no vault)

- "QODY Demo Secrets" (demo environment)
- "QODY Prod Secrets" (production environment)

Azure Container Apps secrets (encrypted at rest, per-app): db-password, jwt-secret, qr-token-secret, stripe-secret-key, stripe-webhook-secret, monri-api-key (when live)

NEVER commit secrets. Always reference Bitwarden item names in documentation, not actual keys.

External Dependencies & Blocked Items

Waiting on Asmir/SnowIT

- **qody.ba domain** registration and DNS delegation (emailed asmirmc@gmail.com)
- **Monri test credentials** (form sent to Monri AM Mahir Mulaomerović)
- **Colors/branding inputs** for final qody.ba site polish
- **Actual restaurant photos** for Specijal (separate SnowIT client project)

Pending BiH Compliance (Pre-Commercial Launch)

- Monri production merchant agreements (per-venue, Model B architecture)

- BiH payment lawyer consult (~500 EUR, confirm no PI license needed)
- Fiscalization integration (eFiskalizacija.ba or fiscal printer)

Operational Continuity Decisions (OCD Register)

ID	Date	Owner	Decision	Rationale
OCD-QODY-001	2026-06-22	FlowForge	Azure ACA (not GCP Cloud Run)	Tenant isolation: QODY needs own RG, not Bilko's GCP project
OCD-QODY-002	2026-06-22	Alem	DNS: qody.alai.no (not qody-demo.alai.no)	Short, brandable URL for demo/pilot
OCD-QODY-003	2026-06-22	FlowForge	Unleash deferred (not deployed)	Feature flags optional for demo; reduces cost/complexity
OCD-QODY-004	2026-06-22	FlowForge	Stripe TEST keys (no live payments)	Demo only; payment flow tested with test cards
OCD-QODY-005	2026-06-22	FlowForge	PostgreSQL RLS enforced (NOBYPASSRLS)	Fail-closed security; multi-tenant isolation at DB layer
OCD-QODY-006	2026-06-24	CEO	Pricing: 0.5% platform fee (max 0.7%), 39-49 KM/month	PRD §2.3 model (NOT 5% — corrected from earlier prototype)
OCD-QODY-007	2026-06-24	Finverge	Monri Model B (per-venue merchant accounts)	Bank-agnostic, no PI license needed, QODY bills separately
OCD-QODY-008	2026-06-24	CodeCraft	MFE build-args NON-OPTIONAL	VITE_API_BASE_URL must be passed at docker build; missing = localhost bake-in
OCD-QODY-009	2026-06-24	CodeCraft	Subscription billing: Stripe BAM (not EUR)	39 KM Pro / 49 KM Enterprise per month (BiH market)
OCD-QODY-010	2026-06-26	Skillforge	BookStack canonical docs	https://docs.alai.no/books/qody — architecture/decisions/features source of truth

Support & Escalation

Owner: FlowForge (DevOps company, ALAI Holding AS)
Escalation: John (AI Director) → Alem (CEO, alem@alai.no, +47 404 74 251)

Runbooks: See repo RUNBOOK.md for deploy, rollback, health checks, secrets rotation, DB ops.

Incidents: Follow ~/system/runbooks/azure-aca-incident.md

Monitoring: Azure Monitor (6 alerts → alem@alai.no). Future: Sentinel ops-watchdog integration.

Support Model: See /tmp/qody-prd/support-model.md for full issue catalog, SLA tiers, resolution playbooks, and proactive monitoring design.

Account Aliases (for MC / Task Management)

- **qody** (product)
- **snowit** (operator)
- **asmir** (partner, tier-1 referral)

Related Documentation

- **Repo:** /Users/makinja/business/ALAI-Holding-AS/products/qody
- **CLAUDE.md:** Project working context
- **BUILD-BLUEPRINT.md:** Build plan (Phase 0→1→2, Talas 0→6)
- **DEPLOY-MAP.md:** Azure ACA deployment authority
- **RUNBOOK.md:** Operational procedures
- **PRD:** /tmp/qody-prd/PRD.txt (Asmir's technical spec v0.1)
- **Gap Analysis:** /tmp/qody-prd/prd-acceptance-audit.md (Talas 0-6 vs PRD audit)
- **Payment Architecture Decision:** /tmp/qody-prd/monri-architecture-decision.md (Finverge, Model B recommendation)
- **Support Model:** /tmp/qody-prd/support-model.md (45 realistic tickets catalogued, SLA, playbooks)

Last Updated: 2026-06-26
Status: LIVE demo (rg-qody-demo), prod clean (rg-qody-prod), Talas 0–6 shipped, V19 migrations applied
Co-Authored-By: Claude Opus 4.8 (1M context) <noreply@anthropic.com>

Revision #1
Created 2026-06-26 07:35:55 UTC by John
Updated 2026-06-26 07:35:55 UTC by John