Payment Layer

QODY Payment Layer

Author: Finverge (Markos Zachariadis) | **Date:** 2026-06-22

Payment Provider Strategy per Market

Bosnia & Herzegovina / Balkans (Primary Market)

Provider	Use Case	Coverage	Integration Complexity
Stripe	Card payments (Visa/Mastercard)	Global, BiH-supported	Low (REST API, Kotlin SDK)
MonriPay	Local Balkan PSP	Regional card acquiring	Medium (API docs available)
Corvus Pay	Regional card processor	Croatia + BiH	Medium (REST API)

Recommendation:

1. **Start with Stripe** — best developer experience, supports BiH merchants (USD/EUR settlement), card tokenization, PCI-compliant
2. **Add Monri** as Phase 2 — local brand recognition, BAM settlement option, lower interchange for Balkan cards

Norway (Secondary Market)

Provider	Use Case	Coverage	Integration Complexity
Vipps MobilePay	Dominant Norwegian wallet	Norway only	Medium (OAuth, polling)
Stripe	Card payments + Apple Pay	Global	Low

Recommendation: Vipps MobilePay (90%+ Norwegian adoption) + Stripe as fallback for international cards.

Provider Abstraction Layer

CRITICAL: QODY must NOT be locked into one provider. Payment Gateway Abstraction pattern:

```
interface PaymentGateway {
    suspend fun createPaymentIntent(request: PaymentIntentRequest): PaymentIntentResponse
    suspend fun confirmPayment(intentId: String): PaymentConfirmationResponse
    suspend fun refund(paymentId: String, amount: Money): RefundResponse
    suspend fun handleWebhook(payload: String, signature: String): WebhookEvent
}

// Implementations:
class StripeGateway : PaymentGateway { /* Stripe-specific */ }
class VippsGateway : PaymentGateway { /* Vipps-specific */ }
class MonriGateway : PaymentGateway { /* Monri-specific */ }

// Factory for per-venue routing:
class PaymentGatewayFactory(private val config: PaymentConfig) {
    fun forVenue(venueId: UUID): PaymentGateway {
        return when (config.getProviderForVenue(venueId)) {
            PaymentProvider.STRIPE -> StripeGateway(config.stripe)
            PaymentProvider.VIPPS -> VippsGateway(config.vipps)
            PaymentProvider.MONRI -> MonriGateway(config.monri)
        }
    }
}
```

Checkout Flows

Pay-Now (Per Order)

Flow:

1. Guest adds items to cart
2. Guest taps "Pay Now"
3. Backend creates `PaymentIntent` (provider-agnostic)
4. Frontend redirects to payment provider (Stripe Checkout, Vipps landing page, or Monri hosted form)
5. Provider webhooks `payment.succeeded` → backend confirms order → notifies kitchen

Pay-at-End (Open Tab)

Flow:

1. Guest orders multiple rounds (drinks, appetizers, mains)
2. Each order appends to the same `session_id` (table session)
3. When guest requests bill, backend aggregates all unpaid orders for that session
4. Guest sees total → pays once

Split Bill

Three Modes:

Mode	Description	Backend Logic
By Item	Guest A pays for items 1, 3; Guest B pays for items 2, 4	Create separate orders per guest
Evenly	Total divided by N guests	Single order, N payment intents of `total / N`
By Amount	Guest A pays 30 BAM, Guest B pays 20 BAM	Validate `sum(amounts) == order_total`

Tipping

Implementation:

1. After payment intent created, frontend shows tip options (10%, 15%, 20%, custom)
2. Tip is added to `payment.amount` before provider confirmation
3. Backend splits tip revenue in settlement

Feature Flag: Tipping may be disabled for some markets. Use Unleash flag `qody.tipping.enabled` (venue-level).

Money Model

Amount Storage

RULE: Always store monetary amounts in **minor units** (cents, øre, feninga).

```
data class Money(
    val amountMinor: Int,  // e.g., 1250 = 12.50 BAM
```

```
    val currency: Currency
) {
    val amountMajor: BigDecimal
        get() = BigDecimal(amountMinor).divide(BigDecimal(100), 2, RoundingMode.HALF_UP)
}

enum class Currency(val code: String, val symbol: String, val minorUnits: Int) {
    BAM("BAM", "KM", 2),
    NOK("NOK", "kr", 2),
    EUR("EUR", "€", 2)
}
```

Tax / VAT Calculation

Market	Category	Rate
Bosnia & Herzegovina	All items (food, alcohol, general)	17%
Norway	Food	15%
Norway	Alcohol	25%
Norway	General	25%

```
val TAX_RULES = mapOf(
    "BA" to mapOf(
        "food" to BigDecimal("0.17"),
        "alcohol" to BigDecimal("0.17"),
        "general" to BigDecimal("0.17")
    ),
    "NO" to mapOf(
        "food" to BigDecimal("0.15"),
        "alcohol" to BigDecimal("0.25"),
        "general" to BigDecimal("0.25")
    )
)

fun calculateTax(item: MenuItem, quantity: Int, country: String): Int {
    val rate = TAX_RULES[country]?.get(item.taxCategory) ?: BigDecimal("0.25")
    val subtotal = item.priceMinor * quantity
    return (subtotal.toBigDecimal() * rate).toInt()
}
```

Currency & Rounding

Multi-Currency Note: QODY must support BAM (BiH), NOK (Norway), EUR (potential expansion). Venue sets its default currency in `venues.default_currency`. Prices in `menu_items.price_minor` are always in that venue's currency.

Reconciliation

Daily Reconciliation Flow:

1. Batch job runs nightly (cron or Ktor scheduled task)
2. For each venue, query all `payments.status = 'succeeded'` from yesterday
3. Compare with provider settlement reports (Stripe Payouts API, Vipps reports)
4. Flag discrepancies (missing payments, refunds not recorded)

Settlement & Payouts to Venues

Marketplace Model vs Venue-Direct PSP

Model	Description	Pros	Cons
Marketplace (Stripe Connect)	QODY holds master Stripe account; venues are Connected Accounts	Centralized control, auto platform fee	QODY responsible for payouts, regulatory complexity
Venue-Direct PSP	Each venue has own Stripe/Vipps account	No payment license needed, venue owns relationship	Cannot auto-deduct SaaS fees

Recommendation:

- **Phase 1 (MVP):** Marketplace model (Stripe Connect) — simpler for pilot venues, faster onboarding
- **Phase 2:** Offer venue-direct option for large chains with existing PSP contracts

Stripe Connect Implementation (Marketplace Model)

```
val paymentIntent = stripe.paymentIntents.create(
    PaymentIntentCreateParams.builder()
        .setAmount(order.totalMinor.toLong())
        .setCurrency(order.currency.code.lowercase())
```

```
        .setApplicationFeeAmount((order.totalMinor * 0.05).toLong()) // 5% QODY fee
        .setTransferData(
            PaymentIntentCreateParams.TransferData.builder()
                .setDestination(venue.stripeConnectedAccountId)
                .build()
        )
        .build()
)
```

Payout Cadence: Stripe automatically pays out to venue bank account (default: daily for Standard accounts, weekly for Express).

Fiscalization / Receipts

Bosnia & Herzegovina

Fiscal Device Requirement: Cash sales require **ESET fiscal devices**. Card/online payments: Current regulation unclear whether ESET required for cashless-only venues.

QODY Implementation:

- **Phase 1:** Generate PDF receipt (not fiscalized). Mark as "Proforma" or "Non-Fiscal Receipt"
- **Phase 2:** Integrate CPF API for B2B invoices (when CPF specs published)
- **ESET Integration:** Requires hardware device. Send order data to ESET device via REST API (if device supports)

Recommendation: Launch QODY in BiH with **non-fiscal receipts** (PDF) for pilot phase. Add ESET integration when regulatory clarity is confirmed.

Norway

Fiscal Requirement: Norway requires **sales records** for VAT reporting, but no real-time fiscal device. Receipts must include:

- Venue name, address, org.nr
- Date, time
- Itemized list with VAT breakdown
- Payment method
- Receipt number (sequential or unique)

QODY Implementation: Generate receipt with VAT breakdown (25% vs 15% for food). Store receipt PDF in cloud storage. Email receipt to guest (optional).

Webhooks & Idempotency

Webhook Handling

Providers send webhooks for:

- `payment.succeeded` (confirm order, notify kitchen)
- `payment.failed` (mark order as failed, notify guest)
- `refund.created` (update order status to refunded)

```
post("/webhooks/stripe") {
    val payload = call.receiveText()
    val signature = call.request.header("Stripe-Signature") ?: throw
BadRequestException("Missing signature")

    val event = stripeGateway.handleWebhook(payload, signature)

    when (event.type) {
        "payment_intent.succeeded" -> {
            val paymentIntentId = event.data["id"] as String
            paymentService.confirmPayment(paymentIntentId)
        }
        "payment_intent.payment_failed" -> {
            val paymentIntentId = event.data["id"] as String
            paymentService.markFailed(paymentIntentId)
        }
    }

    call.respond(HttpStatusCode.OK)
}
```

Security: Verify webhook signature (Stripe uses HMAC SHA256, Vipps uses HMAC SHA512). Store webhook secret in environment variable.

Idempotency

RULE: Payment confirmations must be idempotent. A webhook may arrive multiple times.

```
suspend fun confirmPayment(paymentIntentId: String) {
    val payment = paymentRepository.findByProviderPaymentId(paymentIntentId)
        ?: throw NotFoundException("Payment not found")

    if (payment.status == PaymentStatus.SUCCEEDED) {
        // Already processed; idempotent return
        return
    }

    transaction {
        paymentRepository.updateStatus(payment.id, PaymentStatus.SUCCEEDED, Instant.now())
        orderRepository.updateTotalPaid(payment.orderId, payment.amountMinor)
        // Notify kitchen, send receipt, etc.
    }
}
```

Database Constraint:

```
CREATE UNIQUE INDEX idx_payments_provider_id ON payments(provider, provider_payment_id);
```

This ensures `(provider, provider_payment_id)` is unique → prevents duplicate payment records.

Feature-Flag Gating

Feature	Unleash Flag	Default	Gating Reason
Split Bill	`qody.payment.split_bill`	OFF	Premium plan only
Tipping	`qody.payment.tipping`	ON (BiH), OFF (NO)	Cultural preference
Partial Payments	`qody.payment.partial_payments`	OFF	Premium plan only
Service Charge	`qody.payment.service_charge`	OFF	Per-venue opt-in

Implementation Roadmap

Phase 1 (MVP — 4-6 weeks)

- Stripe integration (card payments)
- Pay-now per order

- Pay-at-end (open tab)
- Basic receipt generation (PDF, non-fiscal)
- Marketplace model (Stripe Connect)
- Payment webhook handling + idempotency
- Unleash feature flags for tipping, split bill

Phase 2 (Expansion — 8-10 weeks)

- Split bill (by item, evenly, by amount)
- Tipping with configurable rates
- Vipps integration (Norway)
- Monri integration (BiH)
- Partial payments
- ESET fiscal device integration (BiH)
- Reconciliation reports

Phase 3 (Advanced — 12+ weeks)

- Venue-direct PSP option
- Multi-currency support (BAM, NOK, EUR)
- CPF e-invoice integration (BiH B2B)
- Refund self-service for venues
- Payment analytics dashboard

Summary — Key Decisions

1. **Stripe-first** for BiH/Balkans (card), Vipps for Norway (wallet), Monri as Phase 2 local option
2. **Provider abstraction layer** (`PaymentGateway` interface) to avoid lock-in
3. **Marketplace model (Stripe Connect)** for Phase 1 — QODY takes 3-5% platform fee, venues auto-paid out
4. **Money in minor units** (Int, never Float) — strict double-entry discipline
5. **Split bill, tipping, partial payments** — all gated by Unleash flags (plan-tier and market-specific)
6. **Non-fiscal receipts Phase 1** — add ESET/CPF when regulatory clarity achieved
7. **Idempotent webhook handling** — `(provider, provider_payment_id)` unique constraint
8. **Reconciliation nightly** — compare QODY ledger vs provider settlement reports

---

Revision #1
Created 2026-06-22 15:47:00 UTC by John
Updated 2026-06-22 15:47:01 UTC by John