Low-Level Design Document

Low-Level Design Document

> **Project:** Drop **Module/Component:** Transactions Module (Remittance + QR Payment) **Version:** 1.0 **Date:** 2026-02-23 **Author:** Petter Graff, Senior Enterprise Architect **Status:** Approved **Reviewers:** Alem Bašić (CEO), John (AI Director) **Related HLD:** HLD Document

Document History

Version	Date	Author	Changes
0.1	2026-02-21	Banking Architecture Team	Initial draft from source code
1.0	2026-02-23	Petter Graff	Filled with real Drop data

---

1. Module Overview

Module: `transactions` **Service/Repo:** `drop-api` — `src/drop-api/src/routes/transactions.ts` **Team Owner:** ALAI — Backend

Single Responsibility: Processes all financial operations — remittance (international money transfer via PISP) and QR payments (domestic merchant payments via PISP) — using the PSD2 pass-through model where Drop never holds funds.

Boundaries:

- **Owns:** Transaction records (`transactions` table), exchange rates (`exchange_rates` table), fee calculation, pre-payment disclosure, idempotency enforcement, PISP payment initiation orchestration
- **Does NOT own:** User authentication (auth module), recipient management (`recipients` table owned by recipients route), bank account balance display (bank_accounts route / AISP), merchant registration (merchants route)

- **Delegates to:** BankID auth middleware (JWT validation), Open Banking PISP API (actual payment execution), audit_log (compliance side-effect), notifications (user alerting)

Key Business Rules:

1. Drop never deducts money from the Drop DB balance except as a cached AISP value — all real deductions happen at the user's bank via PISP
2. Every transaction requires `kyc_status = 'approved'` on the initiating user
3. Remittance amounts: 100 NOK minimum, 50,000 NOK maximum; fee = 0.5% of send amount
4. QR payments: fee = merchant `fee_rate` (default 1%); validated via HMAC QR code
5. `idempotency_key` (unique index on `transactions`) prevents double-charging on retry

---

2. Class / Module Diagram

```
classDiagram
    class TransactionsRoute {
        -authMiddleware: Middleware
        -rateLimiter: Middleware
        +POST /v1/transactions/remittance(body: RemittanceDto): Response
        +POST /v1/transactions/qr-payment(body: QRPaymentDto): Response
        +POST /v1/transactions/disclosure(body: DisclosureDto): Response
        +GET /v1/transactions(query: TransactionFilter): Response
        +GET /v1/transactions/:id(): Response
        -validateRemittanceInput(dto: RemittanceDto): void
        -validateQRPaymentInput(dto: QRPaymentDto): void
    }

    class RemittanceDto {
        +recipientId: string
        +amount: number
        +bankAccountId: string
        +currency: string
    }

    class QRPaymentDto {
        +merchantId: string
        +amount: number
        +qrData: string
    }
```

```
class Transaction {
    +id: string
    +user_id: string
    +type: "remittance" | "qr_payment"
    +status: "processing" | "completed" | "failed"
    +amount: number
    +currency: string
    +fee: number
    +recipient_id: string
    +merchant_id: string
    +exchange_rate: number
    +send_amount: number
    +receive_amount: number
    +receive_currency: string
    +idempotency_key: string
    +created_at: string
}

class Database {
    <<abstraction>>
    +query(sql, params): T[]
    +getOne(sql, params): T
    +run(sql, params): RunResult
    +transaction(fn): void
}

class PISPClient {
    <<external>>
    +initiatePayment(paymentRequest): PaymentResponse
    +getPaymentStatus(paymentId): PaymentStatus
}

TransactionsRoute --> RemittanceDto
TransactionsRoute --> QRPaymentDto
TransactionsRoute --> Transaction
TransactionsRoute --> Database
TransactionsRoute --> PISPClient
```

3. Database Schema

3.1 Tables

`transactions`

Purpose: Records all financial operations. Append-only — status updates are the only writes after creation.

Column	Type	Nullable	Default	Constraints	Description
`id`	`TEXT`	NO	—	PK, format: `tx_<hex16>`	Transaction identifier
`user_id`	`TEXT`	NO	—	FK → `users(id)`	Initiating user
`type`	`TEXT`	NO	—	CHECK('remittance','qr_payment')	Transaction type
`status`	`TEXT`	NO	`'processing'`	CHECK('processing','completed','failed')	Payment status
`amount`	`REAL`	NO	—	NOT NULL	Send amount in NOK (stored in øre equivalent)
`currency`	`TEXT`	YES	`'NOK'`	—	Source currency (always NOK at MVP)
`fee`	`REAL`	YES	`0`	—	Fee in NOK (0.5% remittance, merchant rate for QR)
`recipient_id`	`TEXT`	YES	NULL	FK → `recipients(id)`	For remittances; NULL for QR
`merchant_id`	`TEXT`	YES	NULL	FK → `merchants(id)`	For QR payments; NULL for remittance
`send_amount`	`REAL`	YES	NULL	—	Amount sent in source currency
`receive_amount`	`REAL`	YES	NULL	—	Amount received in destination currency
`receive_currency`	`TEXT`	YES	NULL	—	Destination currency (e.g., RSD, EUR)

Column	Type	Nullable	Default	Constraints	Description
`exchange_rate`	`REAL`	YES	NULL	—	Exchange rate at time of transaction
`description`	`TEXT`	YES	NULL	—	Optional user-provided description
`idempotency_key`	`TEXT`	YES	NULL	UNIQUE	Prevents duplicate payments on retry
`created_at`	`TEXT`	NO	`datetime('now')`	—	Transaction timestamp

Indexes:

Index Name	Columns	Type	Rationale
`transactions_pkey`	`id`	B-tree (PK)	Primary key lookup
`idx_transactions_user`	`user_id`	B-tree	Filter all transactions per user (high frequency)
`idx_tx_idempotency`	`idempotency_key`	Unique B-tree	Prevent duplicate payment on API retry
`idx_transactions_recipient`	`recipient_id`	B-tree	Lookup by recipient
`idx_transactions_merchant`	`merchant_id`	B-tree	Lookup by merchant

`exchange_rates`

Purpose: Stores current NOK-to-foreign currency exchange rates for the 6 supported remittance corridors.

Column	Type	Nullable	Default	Constraints	Description
`id`	`INTEGER`	NO	auto	PK	Surrogate key
`from_currency`	`TEXT`	NO	—	NOT NULL	Always 'NOK' at MVP
`to_currency`	`TEXT`	NO	—	NOT NULL	Target currency (RSD, BAM, PLN, PKR, TRY, EUR)
`rate`	`REAL`	NO	—	NOT NULL	Exchange rate: 1 NOK = N target currency units
`updated_at`	`TEXT`	YES	—	—	Last rate update timestamp

Indexes:

Index Name	Columns	Type	Rationale
`idx_rates_currency`	`from_currency, to_currency`	Composite B-tree	Fast rate lookup by currency pair

3.2 Enums (CHECK constraints in SQLite, native ENUMs in PostgreSQL migration)

```
-- transaction type
CHECK(type IN ('remittance', 'qr_payment'))

-- transaction status
CHECK(status IN ('processing', 'completed', 'failed'))
```

3.3 Migration Notes

- Migration: Included in `db.ts` `initializeDatabase()` — runs on startup (SQLite) or via separate migration script (PostgreSQL)
- Zero-downtime: YES — only `INSERT` and `UPDATE status` needed; `CREATE INDEX CONCURRENTLY` for PostgreSQL
- Backfill required: NO — new tables
- Estimated migration time: < 1 second (SQLite), < 5 seconds (PostgreSQL)

---

4. API Contract

Base Path: `/v1/transactions`

`POST /v1/transactions/remittance`

Summary: Initiate international money transfer (PISP) from user's bank account to a saved recipient

Authentication: Bearer JWT required (`authMiddleware`) **Rate Limit:** 10 req/60s per IP + 3 req/60s per user

Request Body:

```
{
  "recipientId": "rec_abc123def456gh78",
  "amount": 2000,
  "bankAccountId": "ba_abc123def456gh78",
  "currency": "NOK"
}
```

Success Response — `201 Created`:

```
{
  "data": {
    "id": "tx_rem_abc123def456gh78",
    "type": "remittance",
    "status": "processing",
    "amount": 2000,
    "fee": 10,
    "receiveAmount": 20340,
    "receiveCurrency": "RSD",
    "exchangeRate": 10.17,
    "estimatedDelivery": "2-4 business days",
    "scaRedirect": "https://dnb.no/sca/pay/abc123",
    "createdAt": "2026-02-23T10:00:00.000Z"
  }
}
```

Error Responses:

Status	Code	Description
`400`	`validation_error`	Missing or invalid fields (amount, recipientId)
`401`	`unauthorized`	Missing or expired JWT
`403`	`kyc_required`	User `kyc_status` is not `approved`
`403`	`insufficient_balance`	Cached AISP balance < amount + fee
`404`	`recipient_not_found`	recipientId does not belong to this user
`409`	`duplicate_transaction`	Idempotency key collision — returns existing transaction
`422`	`amount_out_of_range`	Amount < 100 NOK or > 50,000 NOK
`429`	`rate_limited`	Exceeded 10 req/60s per IP or 3 req/60s per user

Status	Code	Description
`502`	`pisp_unavailable`	Open Banking PISP API unreachable
`500`	`internal_error`	Unexpected server error

`POST /v1/transactions/qr-payment`

Summary: Initiate QR merchant payment (PISP) from user's bank account

Authentication: Bearer JWT required

Request Body:

```
{
  "merchantId": "mer_abc123def456gh78",
  "amount": 450
}
```

Success Response — `201 Created`:

```
{
  "data": {
    "id": "tx_qr_abc123def456gh78",
    "type": "qr_payment",
    "status": "completed",
    "amount": 450,
    "fee": 4.5,
    "merchantName": "Café Oslo AS",
    "createdAt": "2026-02-23T10:00:00.000Z"
  }
}
```

Error Responses:

Status	Code	Description
`400`	`validation_error`	Missing or invalid fields
`401`	`unauthorized`	Missing or expired JWT
`403`	`kyc_required`	User `kyc_status` not `approved`
`404`	`merchant_not_found`	merchantId not found or inactive
`500`	`internal_error`	Unexpected error

POST /v1/transactions/disclosure

Summary: Pre-payment disclosure — returns fee, exchange rate, receive amount (PSD2 Art. 45/46 compliance)

Authentication: Bearer JWT required

Request Body:

```
{
  "type": "remittance",
  "amount": 2000,
  "recipientId": "rec_abc123def456gh78"
}
```

Success Response — `200 OK`:

```
{
  "data": {
    "sendAmount": 2000,
    "sendCurrency": "NOK",
    "fee": 10,
    "feePercentage": 0.5,
    "exchangeRate": 10.17,
    "receiveAmount": 20340,
    "receiveCurrency": "RSD",
    "totalCost": 2010,
    "estimatedDelivery": "2-4 business days"
  }
}
```

GET /v1/transactions

Summary: List authenticated user's transactions with pagination

Query Parameters:

Parameter	Type	Default	Description
`page`	`integer`	`1`	Page number (1-based)

Parameter	Type	Default	Description
`limit`	`integer`	`20`	Items per page (max 50)
`type`	`string`	—	Filter: `remittance` or `qr_payment`
`status`	`string`	—	Filter: `processing`, `completed`, `failed`

Success Response — `200 OK`:


```
{
  "data": {
    "transactions": [
      {
        "id": "tx_rem_abc123",
        "type": "remittance",
        "status": "completed",
        "amount": 2000,
        "fee": 10,
        "receiveAmount": 20340,
        "receiveCurrency": "RSD",
        "recipientName": "Marko Petrovic",
        "createdAt": "2026-02-23T10:00:00.000Z"
      }
    ],
    "total": 15,
    "page": 1,
    "limit": 20
  }
}
```


---

5. Algorithm Specifications

5.1 Fee Calculation — Remittance

Purpose: Calculate 0.5% fee on remittance, rounded to 2 decimal places **Complexity:** Time O(1) | Space O(1)

```
function calculateRemittanceFee(sendAmountNOK: number): number
    FEE_RATE = 0.005  // 0.5%
    fee = sendAmountNOK * FEE_RATE
    return Math.round(fee * 100) / 100  // Round to 2 decimal places

function calculateReceiveAmount(sendAmountNOK: number, exchangeRate: number): number
    netSend = sendAmountNOK  // Fee taken from send amount, not receive
    receive = netSend * exchangeRate
    return Math.round(receive)  // Round to whole units of target currency
```

Edge Cases:

- **Minimum amount:** 100 NOK → fee = 0.50 NOK
- **Maximum amount:** 50,000 NOK → fee = 250 NOK
- **Rate not found:** Return 404 — do not proceed to transaction

5.2 Idempotency Key Generation

Purpose: Prevent double-charging on network retry or duplicate form submission **Format:** `{userId}:{amount}:{recipientId}:{minuteTimestamp}`

```
function generateIdempotencyKey(userId, amount, recipientId): string
    minuteTimestamp = Math.floor(Date.now() / 60000)  // Changes every 60s
    key = `${userId}:${amount}:${recipientId}:${minuteTimestamp}`
    return key
    // Unique index on transactions.idempotency_key prevents duplicate
    // If INSERT fails with UNIQUE constraint → return existing transaction
```

---

6. Sequence Diagrams

6.1 Remittance Initiation Flow

```
sequenceDiagram
    autonumber
    actor Client as Client (Web/Mobile)
    participant RL as Rate Limiter
    participant Auth as Auth Middleware
    participant Route as Transactions Route
```

```
    participant DB as Database
    participant PISP as Open Banking PISP

    Client->>RL: POST /v1/transactions/remittance
    RL->>RL: Check rate_limits (10/IP, 3/user per 60s)
    alt Rate limit exceeded
        RL-->>Client: 429 Too Many Requests
    end
    RL->>Auth: Forward request
    Auth->>Auth: Extract JWT from Bearer header / cookie
    Auth->>DB: SELECT session WHERE token_hash = ? AND revoked = 0
    Auth->>DB: SELECT user WHERE id = ? AND deleted_at IS NULL
    Auth-->>Route: user context {userId, role, kycStatus}

    Route->>Route: Validate body: recipientId, amount (100-50000), bankAccountId
    alt Validation fails
        Route-->>Client: 400 validation_error
    end
    alt kyc_status != 'approved'
        Route-->>Client: 403 kyc_required
    end

    Route->>DB: SELECT * FROM recipients WHERE id = ? AND user_id = ?
    alt Recipient not found
        Route-->>Client: 404 recipient_not_found
    end
    Route->>DB: SELECT rate FROM exchange_rates WHERE to_currency = ?
    Route->>DB: SELECT * FROM bank_accounts WHERE id = ? AND user_id = ? AND is_primary = 1

    Route->>Route: Calculate fee (0.5%), total cost, receive amount
    alt balance < totalCost
        Route-->>Client: 403 insufficient_balance
    end

    Route->>DB: BEGIN TRANSACTION
    Route->>DB: UPDATE bank_accounts SET balance = balance - totalCostInOere WHERE balance >=
?
    Route->>DB: INSERT INTO transactions (status='processing', idempotency_key=?)
    Route->>DB: INSERT INTO audit_log (action='transaction.create')
    Route->>DB: INSERT INTO notifications (title='Overføring startet')
```

```
    Route->>DB: COMMIT

    Route->>PISP: POST /v1/payments/cross-border-credit-transfers
    PISP-->>Route: {paymentId, transactionStatus: "RCVD", scaRedirect}

    Route-->>Client: 201 {transactionId, status: "processing", scaRedirect}
```

6.2 QR Payment Flow

```
sequenceDiagram
    autonumber
    actor Client as Client (Mobile)
    participant Route as Transactions Route
    participant DB as Database

    Client->>Route: POST /v1/transactions/qr-payment {merchantId, amount}
    Route->>DB: Verify JWT session
    Route->>DB: SELECT * FROM merchants WHERE id = ? AND status = 'active'
    alt Merchant not found
        Route-->>Client: 404 merchant_not_found
    end
    Route->>DB: SELECT * FROM bank_accounts WHERE user_id = ? AND is_primary = 1

    Route->>Route: Calculate fee = amount * merchant.fee_rate
    Route->>Route: Calculate total = amount + fee

    Route->>DB: BEGIN TRANSACTION
    Route->>DB: UPDATE bank_accounts SET balance = balance - total
    Route->>DB: INSERT INTO transactions (type='qr_payment', status='completed')
    Route->>DB: INSERT INTO audit_log (action='qr_payment.create')
    Route->>DB: INSERT INTO notifications (title='Betaling registrert')
    Route->>DB: COMMIT

    Route-->>Client: 201 {transactionId, status: "completed", merchantName}
```

---

7. State Diagrams

```
stateDiagram-v2
    [*] --> processing: POST /v1/transactions/remittance (PISP initiated)

    processing --> completed: PISP webhook — payment confirmed by bank
    processing --> failed: PISP webhook — payment rejected (insufficient funds, SCA timeout,
SCA cancelled)
    processing --> failed: 5-minute SCA timeout — no callback received

    completed --> [*]
    failed --> [*]
```

Note: QR payments go directly from creation to `completed` (synchronous in MVP — no PISP webhook for domestic transfers in mock mode).

State Transition Rules:

From	To	Trigger	Guard Condition	Side Effect
(none)	processing	POST /v1/transactions/remittance	KYC approved, balance sufficient	Deduct cached balance, create audit log, send notification
processing	completed	PISP webhook or QR sync completion	paymentId matches transaction	Update status, send completion notification
processing	failed	PISP webhook rejection or 5-min timeout	paymentId matches, status RJCT	Restore cached balance (re-sync AISP), send failure notification

---

8. Error Handling Strategy

8.1 Error Classification

Error Type	HTTP Status	Retry?	Log Level	Alert?
ValidationError	400	No	INFO	No
UnauthorizedError	401	No	WARN	No
KYCRequired	403	No	INFO	No
InsufficientBalance	403	No	INFO	No
RecipientNotFound	404	No	INFO	No

Error Type	HTTP Status	Retry?	Log Level	Alert?
DuplicateTransaction	409	No	INFO	No
AmountOutOfRange	422	No	INFO	No
RateLimited	429	After Retry-After	WARN	No
PISPUnavailable	502	Yes (3x backoff)	ERROR	Yes (if sustained > 5 min)
DatabaseError	500	Yes (1x)	ERROR	Yes
UnexpectedError	500	No	ERROR	Yes

8.2 Error Response Format

```
{
  "error": "kyc_required",
  "message": "Du må fullføre identitetsverifisering før du kan sende penger.",
  "details": []
}
```

8.3 Retry & Fallback Strategy

```
PISP API call failure:
  → Retry with exponential backoff: [1s, 2s, 4s]
  → Max retries: 3
  → Circuit breaker: Open after 3 failures in 60s window → 60s cooldown
  → Fallback: Return 502 to client — payment cannot proceed without PISP
  → Alert: Sentry alert if circuit remains open > 5 minutes
  → Idempotency: PISP call uses X-Request-ID = idempotency_key to prevent double-payment on
retry
```

---

9. Concurrency & Thread Safety

Concern	Scenario	Mitigation
Double payment	Client retries POST /v1/transactions/remittance after network timeout	Unique index on `idempotency_key` — second INSERT fails with UNIQUE constraint → return existing transaction

Concern	Scenario	Mitigation
Balance race condition	Two simultaneous payments from same account	DB transaction with `UPDATE bank_accounts SET balance = balance - X WHERE balance >= X` — atomic check-and-deduct
Exchange rate staleness	Rate changes between disclosure and payment	Rate locked at payment initiation time; user sees pre-payment disclosure; rate used is from DB at payment time

---

10. Performance Considerations

Operation	Target (p99)	Current Baseline	Optimization
`POST /v1/transactions/remittance`	< 500ms (local) + PISP latency	~50ms DB operations	DB transaction atomic; PISP call is async from user perspective
`GET /v1/transactions`	< 100ms	~20ms	Index `idx_transactions_user` on `user_id`; pagination limits result set
`POST /v1/transactions/disclosure`	< 50ms	~10ms	Two DB reads (recipient + exchange rate); no external API call
`POST /v1/transactions/qr-payment`	< 200ms	~40ms	Synchronous completion in mock mode; PISP async in production

Known bottlenecks:

- PISP API latency: 200-2000ms external call — mitigated by async SCA redirect pattern
- Exchange rate reads: High frequency but 6 rows only — fully cached in PostgreSQL buffer pool

---

11. Dependencies

Internal Dependencies

Dependency	Type	Purpose	Fallback if unavailable

`middleware/auth.ts`	Synchronous	JWT validation + user context	None — request rejected with 401
`middleware/rate-limit.ts`	Synchronous	IP + user rate limiting	None — request rejected with 429
`lib/db.ts`	Required	All data access (query, run, transaction)	None — module unavailable

External Dependencies

Dependency	Version	Purpose	Fallback if unavailable
PostgreSQL	16	Primary data store	SQLite (dev only)
Open Banking PISP (Neonomics/ASPSP)	Berlin Group v1.3.12+	Payment initiation	None — return 502, payment cannot proceed
Open Banking AISP	Berlin Group v1.3.12+	Pre-payment balance verification	Use cached `bank_accounts.balance` with staleness warning

---

12. Configuration Parameters

Variable	Type	Default	Required	Description
`DATABASE_URL`	`string`	—	No (SQLite default)	PostgreSQL connection string
`NEXT_PUBLIC_SERVICE_MODE`	`string`	`mock`	No	`mock` = simulate PISP; `production` = real PISP calls
`OPEN_BANKING_API_URL`	`string`	—	Yes (prod)	Neonomics or ASPSP base URL
`OPEN_BANKING_CLIENT_ID`	`string`	—	Yes (prod)	eIDAS client identifier
`OPEN_BANKING_CLIENT_SECRET`	`string`	—	Yes (prod)	eIDAS client secret

---

13. Testing Approach

Test Type	Tool	Coverage Target	Location
Unit tests	Vitest	> 80% business logic	`src/drop-api/src/__tests__/unit/transactions/`

Test Type	Tool	Coverage Target	Location
Integration tests	Supertest	Key payment flows	`src/drop-api/src/__tests__/integration/transactions/`

Key test scenarios:

- ☑ Remittance — success path (mock PISP)
- ☑ Remittance — KYC not approved → 403
- ☑ Remittance — amount < 100 NOK → 422
- ☑ Remittance — amount > 50,000 NOK → 422
- ☑ Remittance — recipient not found → 404
- ☑ Remittance — duplicate request (idempotency key) → 409 with existing transaction
- ☑ QR payment — success path
- ☑ QR payment — merchant inactive → 404
- ☑ Disclosure — calculates fee, exchange rate, receive amount correctly
- ☐ PISP circuit breaker — 3 failures → open → 502 (integration test, Phase 2)
- ☐ Concurrent payment — balance race condition handled correctly (Phase 2)

Approval

Role	Name	Date	Signature
Author	Petter Graff	2026-02-23	
Module Owner	John (AI Director)		
Security Review			
Tech Lead	John (AI Director)		

Revision #5
Created 2026-02-23 11:28:53 UTC by John
Updated 2026-05-31 20:02:55 UTC by John