Service Design Document — Payment Service

Service Design Document — Payment Service

> **Project:** Drop **Service:** Payment Service (Remittance + QR Payments)
> **Version:** 0.1.0 **Date:** 2026-02-23 **Author:** Platform Architect (AI) **Status:** In Review **Reviewers:** Alem Bašić (CEO)

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Platform Architect (AI)	Initial draft from source code analysis

---

1. Service Overview

The Payment Service is Drop's core business logic module, responsible for:

1. **Remittance** — international money transfers from Norway to 5 corridors (Serbia, Bosnia, Poland, Pakistan, Turkey)
2. **QR Payments** — instant in-store payments to registered merchants

Drop uses a **PSD2 pass-through model** — it never holds customer money. Payments are PISP-initiated directly from the user's bank account. The service orchestrates: balance verification → fee calculation → atomic debit + transaction record creation.

Source files:

- `src/drop-app/src/app/api/transactions/remittance/route.ts`
- `src/drop-app/src/app/api/transactions/qr-payment/route.ts`
- `src/drop-app/src/app/api/transactions/disclosure/route.ts`
- `src/drop-app/src/lib/db.ts` (transaction + bank account operations)

---

2. Domain Model

2.1 Core Entities

```
User
  ├── has many BankAccount (AISP-read from user's real bank)
  ├── has many Recipient (saved international recipients)
  ├── has many Transaction
  └── has one Merchant (optional — if registered as merchant)

Transaction
  ├── type: "remittance" | "qr_payment"
  ├── status: "processing" | "completed" | "failed"
  ├── sendAmount + sendCurrency (NOK)
  ├── receiveAmount + receiveCurrency (destination)
  ├── exchangeRate
  ├── fee (NOK)
  └── links to: Recipient (remittance) OR Merchant (QR)

BankAccount
  ├── balance (AISP-read cache — NOT Drop-held funds)
  ├── isPrimary
  └── bankName, accountNumber, currency
```

2.2 Supported Currency Corridors

Destination	Currency	Exchange Rate (illustrative)	Fee
Serbia	RSD	11.7 NOK/RSD	0.5%
Bosnia	BAM	1.04 NOK/BAM	0.5%
Poland	PLN	0.41 NOK/PLN	0.5%
Pakistan	PKR	26.8 NOK/PKR	0.5%

Destination	Currency	Exchange Rate (illustrative)	Fee
Turkey	TRY	3.45 NOK/TRY	0.5%

QR Payments: NOK only, fee 1%, instant settlement.

---

3. Remittance Service Design

3.1 Flow

```
POST /api/transactions/remittance
│
├── 1. requireAuth() — verify JWT cookie + session not revoked
├── 2. Rate limit check (10/min per IP)
├── 3. KYC gate — verify user.kyc_status === 'approved'
├── 4. Validate request body
│      ├── amount: 100–50,000 NOK, max 2 decimal places
│      └── recipientId: must belong to current user
├── 5. Load recipient → extract currency (e.g., RSD)
├── 6. Look up exchange rate for currency
├── 7. Calculate fee (0.5% of amount, rounded to 2 decimals)
├── 8. Load bank account (bankAccountId or primary)
├── 9. Verify balance >= (amount + fee)
├── 10. ATOMIC DATABASE TRANSACTION:
│      ├── Debit bank_accounts.balance by (amount + fee)
│      └── INSERT transaction record (status: 'processing')
└── 11. Return 201 with transaction details
```

3.2 Fee Calculation

```
const fee = Math.round(amount * 0.005 * 100) / 100;  // 0.5%, 2 decimal places
const total = amount + fee;
const receiveAmount = Math.round(amount * exchangeRate * 100) / 100;
```

3.3 ETA Logic

Recipient country	ETA

EEA countries	"1-2 business days"
Non-EEA countries	"2-4 business days"

Note: Serbia, Bosnia are non-EEA. Poland is EEA. Pakistan, Turkey are non-EEA.

3.4 Transaction Status Flow

```
processing → completed (when PISP provider confirms settlement)
processing → failed (on PISP rejection or bank rejection)
```

Current implementation: Status starts as `processing`. Settlement tracking (webhooks from PISP provider) is pending — requires Open Banking provider integration.

---

4. QR Payment Service Design

4.1 Flow

```
POST /api/transactions/qr-payment
│
├── 1. requireAuth() — verify JWT + session
├── 2. Rate limit check (10/min per IP)
├── 3. Validate request body
│      ├── merchantId: must exist
│      └── amount: 1–100,000 NOK, max 2 decimal places
├── 4. Load merchant
├── 5. Get user's primary bank account
├── 6. Calculate fee (1% of amount)
├── 7. Verify balance >= (amount + fee)
├── 8. ATOMIC DATABASE TRANSACTION:
│      ├── Debit bank_accounts.balance by (amount + fee)
│      └── INSERT transaction record (status: 'completed')
└── 9. Return 201 with transaction details
```

4.2 Fee Calculation

```
const fee = Math.round(amount * 0.01 * 100) / 100;  // 1%, 2 decimal places
```

4.3 QR Code Format

Merchant QR codes encode: `drop://pay/{merchantId}`

The mobile app scans this URI, extracts `merchantId`, and pre-fills the QR payment form.

---

5. Pre-Payment Disclosure

Endpoint: `POST /api/transactions/disclosure`

The disclosure endpoint provides full fee transparency BEFORE a payment is initiated, complying with Finansavtaleloven requirements (users must see costs before confirming).

Response includes:

- `amount` — send amount
- `fee` — Drop fee (0.5% remittance / 1.0% QR)
- `feePercentage` — percentage
- `exchangeRate` — NOK to destination currency
- `receiveAmount` — amount recipient receives
- `receiveCurrency` — destination currency
- `estimatedDelivery` — ETA string
- `totalCost` — amount + fee

---

6. Database Operations

6.1 Atomic Transaction Pattern

All payment operations use database transactions to ensure atomicity:

```
BEGIN;
  UPDATE bank_accounts
    SET balance = balance - $1
    WHERE id = $2 AND user_id = $3 AND balance >= $1;

  INSERT INTO transactions (id, user_id, type, status, send_amount, ...)
    VALUES ($1, $2, 'remittance', 'processing', ...);
COMMIT;
```

If either operation fails, the entire transaction rolls back — preventing partial state (debit without record, or record without debit).

6.2 Balance Check

Balance is checked atomically in the UPDATE statement (`WHERE balance >= required_amount`). If the UPDATE affects 0 rows, the transaction fails with `insufficient_balance` error.

6.3 Key Queries

```
-- Get transaction with exchange rate detail
SELECT t.*, r.name as recipient_name, r.country as recipient_country,
       er.rate as exchange_rate
FROM transactions t
  LEFT JOIN recipients r ON t.recipient_id = r.id
  LEFT JOIN exchange_rates er ON er.currency = r.currency
WHERE t.id = $1 AND t.user_id = $2;
```

---

7. Validation Rules

Field	Validation	Rule
`amount` (remittance)	`validateAmount()`	100–50,000 NOK, max 2 decimal places
`amount` (QR payment)	`validateAmount()`	1–100,000 NOK, max 2 decimal places
`recipientId`	ownership check	Must exist in `recipients` table for current user
`merchantId`	existence check	Must exist in `merchants` table
`bankAccountId`	ownership check	Must exist in `bank_accounts` for current user

---

8. Error Handling

Error	HTTP Status	Code	Trigger
Missing required fields	400	`bad_request`	null/undefined required field

Error	HTTP Status	Code	Trigger
No bank account	400	`no_bank_account`	User has no linked bank account
Insufficient balance	402	`insufficient_balance`	`balance < (amount + fee)`
KYC not approved	403	`kyc_required`	`kyc_status !== 'approved'`
Recipient not found	404	`not_found`	Recipient doesn't belong to user
Unsupported currency	422	`validation_error`	No exchange rate for currency
Rate limited	429	`rate_limited`	> 10 req/min per IP

---

9. Audit Trail

Every transaction creates an audit log entry:

```
INSERT INTO audit_log (action, user_id, resource_type, resource_id, details)
VALUES ('transaction_created', $userId, 'transaction', $txId, $detailsJson);
```

AML monitoring: `aml_alerts` table is checked for high-value transactions (> NOK 100,000 equivalent per day, per regulatory requirements).

---

10. Future: Open Banking Integration (PISP)

Current implementation: balance is tracked in Drop's own database (`bank_accounts.balance`), debited atomically.

Target architecture (requires Open Banking provider):

1. Initiate PISP payment at provider API
2. User's actual bank account is debited (not Drop's DB record)
3. Provider webhook confirms settlement
4. Drop updates transaction status from `processing` → `completed`/`failed`

AISP balance refresh: balance in `bank_accounts` should be refreshed via AISP API on each login or dashboard load.

Related Documents

- API Reference
- Backend Architecture
- External Services Integration
- Source: SERVICES.md

Approval

Role	Name	Date	Signature
Author	Platform Architect (AI)	2026-02-23	
Reviewer			
Approver	Alem Bašić		

Revision #5
Created 2026-02-23 12:05:26 UTC by John
Updated 2026-06-07 20:00:29 UTC by John