Module Design Document

Module Design Document

> **Project:** Drop **Module:** Payments Module (`transactions` + `recipients` + `exchange_rates`) **Service:** drop-api — `src/drop-api/src/routes/` **Version:** 1.0 **Date:** 2026-02-23 **Author:** Petter Graff, Senior Enterprise Architect **Status:** Approved **Reviewers:** Alem Bašić (CEO), John (AI Director)

Document History

Version	Date	Author	Changes
0.1	2026-02-23	Petter Graff	Initial draft from source code analysis

---

1. Module Overview & Responsibility

Module: `payments` **Layer:** Application (routes) + Domain (business logic) + Infrastructure (DB access) **Repository:** `src/drop-api/src/routes/transactions.ts`, `src/drop-api/src/routes/recipients.ts`, `src/drop-api/src/routes/rates.ts` **Team Owner:** ALAI — Backend

Single Responsibility Statement:

> The Payments module orchestrates all financial operations — remittance (international money transfer) and QR merchant payments — using the PSD2 pass-through model, meaning Drop never holds funds but initiates payments from the user's bank via PISP.

This module owns:

- Transaction lifecycle (create, status update, list, fetch)

- Remittance business rules (fee calculation, FX rate application, PSD2 disclosure)
- QR payment business rules (merchant fee calculation, HMAC QR validation)
- Recipient management (CRUD for saved remittance contacts)
- Exchange rate lookup and caching (6 corridors)
- Idempotency enforcement for payment operations
- Pre-payment disclosure (PSD2 Art. 45/46)

This module does NOT own:

- User authentication and session management — owned by `auth` module (`routes/auth.ts`, `lib/bankid.ts`)
- KYC/AML status — owned by `compliance` module (`routes/user.ts` + Sumsub integration)
- Bank account linking (AISP consent) — owned by `banking` module (`routes/bank-accounts.ts`)
- Merchant registration — owned by `merchants` module (`routes/merchants.ts`)
- Notifications delivery — owned by `notifications` module (`routes/notifications.ts`)

Why this is a separate module: Payments is the core revenue-generating bounded context for Drop. It has distinct business rules (PSD2 compliance, FX calculation, PISP orchestration), its own data domain (`transactions`, `recipients`, `exchange_rates`), and its own compliance requirements (PSD2 Art. 45/46 disclosure, idempotency, AML monitoring). Separating it enables independent testing, focused security review, and future extraction to a dedicated service if transaction volume demands it (per ADR-005 extraction triggers).

---

2. Interface Definition (Public API)

2.1 Exported Service Interface

```
// Remittance
interface IRemittanceService {
  /**
   * Calculate pre-payment disclosure (fee, FX rate, receive amount)
   * Required by PSD2 Art. 45/46 before every payment
   * @throws {NotFoundError} if recipientId not found or not owned by user
   * @throws {NotFoundError} if exchange rate for currency not found
   */
  calculateDisclosure(dto: DisclosureDto, userId: string): Promise<DisclosureResult>;

  /**
   * Initiate international remittance via PISP
```

```
   * @throws {ForbiddenError} if KYC not approved
   * @throws {ForbiddenError} if insufficient balance
   * @throws {NotFoundError} if recipient not found
   * @throws {ConflictError} if idempotency key collision (returns existing tx)
   * @throws {ValidationError} if amount outside 100-50000 NOK range
   */
  initiateRemittance(dto: RemittanceDto, userId: string): Promise<Transaction>;

  /**
   * Initiate QR merchant payment via PISP (domestic)
   * @throws {ForbiddenError} if KYC not approved
   * @throws {NotFoundError} if merchant not found or inactive
   */
  initiateQRPayment(dto: QRPaymentDto, userId: string): Promise<Transaction>;

  /**
   * List user's transactions with pagination and optional filters
   */
  listTransactions(filter: TransactionFilter, userId: string): Promise<PaginatedTransactions>;

  /**
   * Get a single transaction by ID (must belong to requesting user)
   * @throws {NotFoundError} if not found or access denied
   */
  getTransaction(txId: string, userId: string): Promise<Transaction>;
}

// Recipients
interface IRecipientsService {
  createRecipient(dto: CreateRecipientDto, userId: string): Promise<Recipient>;
  listRecipients(userId: string): Promise<Recipient[]>;
  getRecipient(recipientId: string, userId: string): Promise<Recipient>;
  deleteRecipient(recipientId: string, userId: string): Promise<void>;
}

// Types
export type RemittanceDto = {
  recipientId: string;
  amount: number; // NOK, 100-50000
  bankAccountId: string;
```

```
  currency?: string; // Default: 'NOK'
};

export type QRPaymentDto = {
  merchantId: string;
  amount: number; // NOK, positive
};

export type DisclosureDto = {
  type: 'remittance';
  amount: number;
  recipientId: string;
};

export type Transaction = {
  id: string; // tx_<hex16>
  userId: string;
  type: 'remittance' | 'qr_payment';
  status: 'processing' | 'completed' | 'failed';
  amount: number;
  fee: number;
  receiveAmount?: number;
  receiveCurrency?: string;
  exchangeRate?: number;
  recipientId?: string;
  merchantId?: string;
  idempotencyKey?: string;
  createdAt: string;
};
```

2.2 HTTP Endpoints

Method	Path	Auth	Rate Limit	Description
`POST`	`/v1/transactions/remittance`	JWT	10/IP + 3/user per 60s	Initiate remittance
`POST`	`/v1/transactions/qr-payment`	JWT	10/IP + 3/user per 60s	Initiate QR payment
`POST`	`/v1/transactions/disclosure`	JWT	None specific	Pre-payment fee disclosure

Method	Path	Auth	Rate Limit	Description
`GET`	`/v1/transactions`	JWT	None specific	List user transactions
`GET`	`/v1/transactions/:id`	JWT	None specific	Get transaction by ID
`GET`	`/v1/recipients`	JWT	None specific	List saved recipients
`POST`	`/v1/recipients`	JWT	None specific	Add recipient
`GET`	`/v1/recipients/:id`	JWT	None specific	Get recipient
`DELETE`	`/v1/recipients/:id`	JWT	None specific	Delete recipient
`GET`	`/v1/rates/:currency`	JWT	120 req/60s per IP	Get exchange rate

2.3 Events Published

The Payments module does not use an async event bus (monolith-first architecture per ADR-005). Side effects (notifications, audit log) are written synchronously within the same DB transaction.

Side Effect	Table Written	Trigger
Audit trail	`audit_log`	Every transaction creation
User notification	`notifications`	Transaction created, completed, failed
AML monitoring	`aml_alerts`	Triggered by AML rules engine (high amount, high-risk corridor)

---

3. Internal Structure

```
routes/
├── transactions.ts         # HTTP request handling, rate limiting, routes
├── recipients.ts           # Recipient CRUD routes
└── rates.ts                # Exchange rate lookup

lib/
├── db.ts                   # Dual-driver DB abstraction (query, run, transaction)
├── middleware/
│   ├── auth.ts             # JWT verification, session validation
│   └── rate-limit.ts       # DB-backed IP + user rate limiting
└── openbanking/
    └── neonomics.ts        # PISP client (Phase 2 — currently mock)
```

Layer rules:

- Routes only call `db.ts` functions (no raw SQL strings outside db.ts)
- All SQL uses parameterized queries — no string interpolation
- Business logic (fee calculation, validation) lives in route handlers or dedicated helper functions — not in db.ts
- External API calls (Neonomics PISP) called after DB transaction commits to ensure idempotency key is stored before external call

---

4. Database Schema

Primary Table: `transactions`

Column	Type	Nullable	Default	Constraints	Description
`id`	`TEXT`	NO	—	PK, `tx_<hex16>`	Transaction ID
`user_id`	`TEXT`	NO	—	FK → `users(id)`	Initiating user
`type`	`TEXT`	NO	—	CHECK('remittance','qr_payment')	Payment type
`status`	`TEXT`	NO	`'processing'`	CHECK('processing','completed','failed')	Payment status
`amount`	`REAL`	NO	—	NOT NULL	Amount in NOK
`currency`	`TEXT`	YES	`'NOK'`	—	Source currency
`fee`	`REAL`	YES	`0`	—	Fee in NOK
`recipient_id`	`TEXT`	YES	NULL	FK → `recipients(id)`	Remittance target
`merchant_id`	`TEXT`	YES	NULL	FK → `merchants(id)`	QR payment target
`send_amount`	`REAL`	YES	NULL	—	Amount in source currency
`receive_amount`	`REAL`	YES	NULL	—	Amount in target currency
`receive_currency`	`TEXT`	YES	NULL	—	Target currency
`exchange_rate`	`REAL`	YES	NULL	—	Rate at payment time
`description`	`TEXT`	YES	NULL	—	User description
`idempotency_key`	`TEXT`	YES	NULL	UNIQUE	Duplicate prevention
`created_at`	`TEXT`	NO	`datetime('now')`	—	Creation timestamp

Indexes:

```
-- PostgreSQL
CREATE INDEX CONCURRENTLY idx_transactions_user_id ON transactions(user_id);
-- Rationale: Every list query filters by user_id

CREATE UNIQUE INDEX idx_tx_idempotency ON transactions(idempotency_key)
    WHERE idempotency_key IS NOT NULL;
-- Rationale: Prevent duplicate payments on retry

CREATE INDEX CONCURRENTLY idx_transactions_recipient ON transactions(recipient_id)
    WHERE recipient_id IS NOT NULL;

CREATE INDEX CONCURRENTLY idx_transactions_merchant ON transactions(merchant_id)
    WHERE merchant_id IS NOT NULL;
```

Secondary Table: `recipients`

Column	Type	Nullable	Default	Constraints	Description
`id`	`TEXT`	NO	—	PK, `rec_<hex16>`	Recipient ID
`user_id`	`TEXT`	NO	—	FK → `users(id)`	Owner
`name`	`TEXT`	NO	—	NOT NULL	Recipient full name
`country`	`TEXT`	NO	—	NOT NULL	Country code (RS, BA, PL, PK, TR, EU)
`currency`	`TEXT`	NO	—	NOT NULL	Target currency
`bank_account`	`TEXT`	NO	—	NOT NULL	IBAN or local account number
`bank_name`	`TEXT`	YES	NULL	—	Bank name (optional)
`created_at`	`TEXT`	NO	`datetime('now')`	—	Created timestamp

Secondary Table: `exchange_rates`

Column	Type	Nullable	Default	Constraints	Description
`id`	`INTEGER`	NO	auto	PK	Surrogate key

Column	Type	Nullable	Default	Constraints	Description
`from_currency`	`TEXT`	NO	—	NOT NULL	Always 'NOK' at MVP
`to_currency`	`TEXT`	NO	—	NOT NULL	RSD, BAM, PLN, PKR, TRY, EUR
`rate`	`REAL`	NO	—	NOT NULL	1 NOK = N target units
`updated_at`	`TEXT`	YES	—	—	Last update timestamp

---

5. API Endpoints (Detailed)

`POST /v1/transactions/remittance`

Request:

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

Validation:

- `recipientId`: required, string, must exist in `recipients` WHERE `user_id = currentUser`
- `amount`: required, number, 100 ≤ amount ≤ 50000
- `bankAccountId`: required, string, must exist in `bank_accounts` WHERE `user_id = currentUser`

Success `201`:

```
{
  "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://bank.no/sca/pay/...",
    "createdAt": "2026-02-23T10:00:00.000Z"
  }
}
```

`POST /v1/transactions/disclosure`

Request:

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

Success `200`:

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

---

6. Business Logic Specifications

6.1 Business Rules

Rule ID	Rule	Enforced In	Error
BR-001	User must have `kyc_status = 'approved'` before initiating any payment	Route handler	`kyc_required` (403)
BR-002	Remittance amount must be 100–50,000 NOK	Route validation	`amount_out_of_range` (422)
BR-003	Fee = 0.5% of send amount for remittance	Route calculation	N/A (business calculation)
BR-004	QR payment fee = `merchants.fee_rate` (default 1%)	Route calculation	N/A
BR-005	Recipient must belong to the authenticated user	DB query WHERE user_id	`recipient_not_found` (404)
BR-006	Cached bank balance must cover amount + fee	Route check	`insufficient_balance` (403)
BR-007	Idempotency key uniqueness prevents duplicate payment on retry	UNIQUE DB constraint	Return existing transaction (409 or 200)
BR-008	Pre-payment disclosure must be shown before every remittance (PSD2 Art. 45/46)	Frontend enforces; API provides via `/disclosure`	N/A
BR-009	Drop never initiates PISP without recording transaction in DB first	Atomic transaction: INSERT tx → PISP call	Rollback on PISP error

6.2 Validation Rules

Field	Type	Required	Validation	Error Message
`recipientId`	`string`	Yes	Exists in `recipients` WHERE user_id	"Mottaker ikke funnet"
`amount`	`number`	Yes	100 ≤ amount ≤ 50000, positive	"Beløp må være mellom 100 og 50 000 kr"
`bankAccountId`	`string`	Yes	Exists in `bank_accounts` WHERE user_id	"Bankkonto ikke funnet"
`merchantId`	`string`	Yes (QR)	Exists in `merchants` WHERE status='active'	"Butikk ikke funnet"

6.3 Authorization Rules

Operation	Required Role	Additional Conditions
Initiate remittance	`user`	`kyc_status = 'approved'`
Initiate QR payment	`user`	`kyc_status = 'approved'`
Get disclosure	`user`	Must own recipient
List transactions	`user`	Only own transactions (`user_id = currentUser`)
Add recipient	`user`	Any authenticated user
Delete recipient	`user`	Must own recipient
View exchange rates	`user`	Any authenticated user

---

7. Event Publishing / Consuming

The Payments module operates in a monolith (ADR-005) — no async message bus. All side effects are synchronous within DB transactions:

7.1 Side Effects on Payment Creation

Side Effect	Table Written	Method	Notes
Audit trail	`audit_log`	INSERT within transaction	action='transaction.create', resource_type='transaction'
User notification	`notifications`	INSERT within transaction	title='Overføring startet', body='Din overføring på {amount} kr er under behandling'
AML monitoring	`aml_alerts`	INSERT if rule triggered	Checked post-commit by AML rules engine

7.2 Events Consumed

The module receives PISP payment status updates via:

- **HTTP webhooks** from Open Banking provider (Neonomics in production) → `POST /v1/webhooks/openbanking`
- **Polling** in mock mode — transaction status checked by frontend polling `GET /v1/transactions/:id`

---

8. Dependencies

8.1 Upstream (what this module depends on)

Dependency	Type	Coupling	Reason
`middleware/auth.ts`	Internal module	Hard (required for every route)	JWT validation, user identity
`middleware/rate-limit.ts`	Internal module	Hard (required for payment routes)	IP + user rate limiting
`lib/db.ts`	Shared library	Hard (required)	All data access
PostgreSQL / SQLite	Infrastructure	Hard	Primary storage
Open Banking PISP (Neonomics)	External API	Hard (prod)	Payment initiation — module unavailable if PISP down

8.2 Downstream (what depends on this module)

Consumer	What they use	Notes
`drop-web` (Next.js)	REST API `/v1/transactions/*`, `/v1/recipients/*`	Via fetch with cookie auth
`drop-mobile` (Expo)	REST API `/v1/transactions/*`, `/v1/recipients/*`	Via fetch with Bearer token
`compliance/aml` module	`transactions` table reads	AML rules engine monitors transaction patterns

---

9. Error Handling & Recovery

Error Scenario	Handling	User Impact	Recovery
DB connection lost	Retry 1x, then 503	Request fails — user retries	Auto-recover when DB reconnects
PISP API timeout	Return 502, transaction stays `processing`	Payment may or may not have gone through	PISP idempotency key prevents double charge; poll status endpoint
Duplicate submission	Detect via UNIQUE constraint on idempotency_key	Return existing transaction	No user action needed
KYC not approved	Return 403 immediately	Clear error message with KYC link	User completes KYC verification

Error Scenario	Handling	User Impact	Recovery
Exchange rate missing	Return 404 — payment blocked	Clear error: corridor not supported	Admin updates exchange_rates table
Balance insufficient	Return 403	Clear error with balance shown	User reduces amount or top-up bank account

---

10. Configuration & Feature Flags

Environment Variables

Variable	Type	Default	Description
`NEXT_PUBLIC_SERVICE_MODE`	`string`	`mock`	`mock` = simulated PISP; `production` = real Neonomics calls
`OPEN_BANKING_API_URL`	`string`	—	Neonomics base URL (production)
`OPEN_BANKING_CLIENT_ID`	`string`	—	eIDAS client ID for Neonomics
`OPEN_BANKING_CLIENT_SECRET`	`string`	—	eIDAS client secret (from Secrets Manager)

Feature Flags (environment variables)

Flag	Type	Default	Description
`FEATURE_QR_ENABLED`	`boolean`	`true`	Toggle QR payment feature
`FEATURE_WITHDRAW_ENABLED`	`boolean`	`false`	Toggle withdrawal feature (Phase 3)

---

11. Monitoring & Health Checks

Health Check Endpoint

`GET /v1/health` (owned by health route, includes payment module indicators)

```
{
  "status": "ok",
  "version": "0.1.0",
  "uptime": 3600,
  "db": "connected",
  "dbLatencyMs": 1,
  "timestamp": "2026-02-23T10:00:00.000Z"
}
```

Key Metrics (via Sentry + CloudWatch)

Metric	Type	Alert Threshold	Dashboard
Remittance `201` rate	Counter	Drop > 50% over 5m	Sentry Issues
Remittance `502` (PISP down)	Counter	Any occurrence	Sentry Alert
Transaction processing time	Histogram	p99 > 2000ms	CloudWatch
`kyc_required` 403 rate	Counter	> 20% of remittance attempts	Sentry
`insufficient_balance` 403 rate	Counter	> 30% of remittance attempts	Sentry

---

12. Primary Flow — Sequence Diagram

```
sequenceDiagram
    autonumber
    participant C as Client (Web/Mobile)
    participant RL as Rate Limiter
    participant Auth as Auth Middleware
    participant Route as Transactions Route
    participant DB as PostgreSQL
    participant PISP as Open Banking PISP (Neonomics)

    C->>RL: POST /v1/transactions/remittance
    RL->>RL: Check rate_limits (10/IP, 3/user per 60s)
    RL->>Auth: Forward if within limits
    Auth->>DB: Verify JWT + session + user
```

```
    Auth-->>Route: {userId, role, kycStatus}

    Route->>Route: Validate: kycStatus='approved', amount 100-50000
    alt Validation fails
        Route-->>C: 400/403/422
    end

    Route->>DB: SELECT recipient WHERE id=? AND user_id=?
    Route->>DB: SELECT exchange_rate WHERE to_currency=?
    Route->>DB: SELECT bank_account WHERE id=? AND user_id=?
    Route->>Route: Calculate fee, totalCost, receiveAmount
    alt balance < totalCost
        Route-->>C: 403 insufficient_balance
    end

    Route->>DB: BEGIN TRANSACTION
    Route->>DB: UPDATE bank_accounts SET balance = balance - totalCostInOere
    Route->>DB: INSERT INTO transactions (status='processing', idempotency_key=?)
    Route->>DB: INSERT INTO audit_log
    Route->>DB: INSERT INTO notifications
    Route->>DB: COMMIT

    Route->>PISP: POST /v1/payments/sepa-credit-transfers (with idempotency_key)
    PISP-->>Route: {paymentId, scaRedirect, transactionStatus: "RCVD"}

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

Approval

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

Revision #5
Created 2026-02-23 12:05:00 UTC by John
Updated 2026-05-31 20:03:09 UTC by John