Open Banking (AISP/PISP)

Open Banking Integration: AISP & PISP

Version: 1.0 **Date:** 2026-02-21 **Author:** Banking Architecture Team **Status:** Approved **Applies to:** Drop — PSD2 Pass-Through Model

---

1. Overview

Drop operates as a **Third Party Provider (TPP)** under PSD2, using two regulated services:

- **AISP (Account Information Service Provider)** — reads bank account balances and transaction history from the user's bank
- **PISP (Payment Initiation Service Provider)** — initiates payments directly from the user's bank account

Drop **never holds customer funds**. All money stays in the user's bank account. Drop initiates payments on behalf of the user via PISP and reads balances via AISP, both requiring explicit user consent and Strong Customer Authentication (SCA).

TPP Registration

Before operating as AISP/PISP, Drop must:

1. **Register with Finanstilsynet** (Norwegian FSA) as a payment institution or authorized agent
2. **Obtain eIDAS certificates** (QWAC for TLS, QSeal for signing) from a qualified TSP
3. **Register in the EBA TPP Register** for EU-wide passporting
4. **Onboard with ASPSPs** (banks) via their Open Banking developer portals or through an aggregator (e.g., Neonomics, Tink, Enable Banking)

Registration Item	Authority	Status
Finanstilsynet PISP/AISP license	Finanstilsynet	Not applied (Phase 2 blocker)

Registration Item	Authority	Status
eIDAS QWAC certificate	Qualified TSP (e.g., Buypass, Commfides)	Not obtained
eIDAS QSeal certificate	Qualified TSP	Not obtained
EBA TPP Register entry	EBA	Pending license
ASPSP onboarding (DNB, SpareBank 1, Nordea)	Per bank	Pending license

Interim approach: Drop can operate as an **agent** under a licensed PSP's umbrella (1-3 months to set up) while preparing its own license application (6-12 months).

---

2. Berlin Group NextGenPSD2 API Standard

Drop targets the **Berlin Group NextGenPSD2** specification (v1.3.12+), which is the dominant Open Banking standard in Scandinavia and the EEA. Norwegian banks (DNB, SpareBank 1, Nordea) expose APIs conforming to this standard, often through the **BITS** (Banking Industry's Technical Secretariat) coordination framework.

2.1 API Endpoint Mapping

Berlin Group Endpoint	Method	Drop Usage	Drop Feature
`/v1/consents`	POST	Create AISP consent	Bank Account Linking (`/accounts`)
`/v1/consents/{consentId}`	GET	Check consent status	Consent lifecycle management
`/v1/consents/{consentId}`	DELETE	Revoke consent	Settings / Account unlinking
`/v1/consents/{consentId}/authorisations`	POST	Start SCA for consent	Bank linking SCA redirect
`/v1/consents/{consentId}/status`	GET	Poll consent status	Post-SCA consent verification
`/v1/accounts`	GET	List user's bank accounts	Dashboard account selector
`/v1/accounts/{accountId}`	GET	Get account details	Bank Account detail view
`/v1/accounts/{accountId}/balances`	GET	Read balance	Dashboard balance display (`bank_accounts.balance`)
`/v1/accounts/{accountId}/transactions`	GET	Read transaction history	Transaction History (future)

Berlin Group Endpoint	Method	Drop Usage	Drop Feature
`/v1/payments/sepa-credit-transfers`	POST	Initiate SEPA payment	Remittance (EEA corridors)
`/v1/payments/cross-border-credit-transfers`	POST	Initiate cross-border payment	Remittance (non-EEA corridors)
`/v1/payments/domestic-credit-transfers`	POST	Initiate Norwegian payment	QR Payment to merchant
`/v1/payments/{paymentId}`	GET	Check payment status	Transaction status tracking
`/v1/payments/{paymentId}/authorisations`	POST	Start SCA for payment	Payment SCA redirect
`/v1/payments/{paymentId}/status`	GET	Poll payment status	Post-SCA payment verification

2.2 Base URLs (Norwegian Banks)

Bank (ASPSP)	Sandbox URL	Production URL
DNB	`https://developer.dnb.no/sandbox/psd2/`	`https://api.dnb.no/psd2/`
SpareBank 1	`https://developer.sparebank1.no/sandbox/`	`https://api.sparebank1.no/open-banking/`
Nordea	`https://developer.nordeaopenbanking.com/`	`https://api.nordeaopenbanking.com/`

---

3. AISP — Account Information Service

3.1 Consent Flow

AISP access requires explicit user consent. The consent specifies which accounts and data types (balances, transactions) the TPP can access, and has a maximum validity of 90 days (per PSD2 RTS Art. 10).

```
sequenceDiagram
    participant U as User (Browser/App)
    participant D as Drop API
    participant A as ASPSP (User's Bank)

    U->>D: Link bank account
    D->>A: POST /v1/consents<br/>{access: {balances: [iban], transactions:
[iban]},<br/>recurringIndicator: true, validUntil: "2026-08-21",<br/>frequencyPerDay: 4,
```

```
combinedServiceIndicator: false}
    A-->>D: 201 {consentId, consentStatus: "received",<br/>_links: {scaRedirect:
"https://bank.no/sca/..."}}
    D-->>U: Redirect to bank SCA page
    U->>A: Authenticate with BankID (SCA)
    A-->>U: Redirect to Drop callback
    U->>D: GET /api/accounts/callback?consentId=xxx
    D->>A: GET /v1/consents/{consentId}/status
    A-->>D: {consentStatus: "valid"}
    D->>A: GET /v1/accounts?consentId={consentId}
    A-->>D: {accounts: [{iban, currency, name, ...}]}
    D->>A: GET /v1/accounts/{accountId}/balances
    A-->>D: {balances: [{balanceType: "expected", balanceAmount: {currency: "NOK", amount:
"45230.00"}}]}
    D->>D: Store in bank_accounts table<br/>(balance cached, balance_synced_at = now)
    D-->>U: Bank account linked successfully
```

3.2 Balance Retrieval

After consent is granted, Drop reads balances to display on the Dashboard. The `bank_accounts.balance` column stores the **cached AISP-read value** — it is never a Drop-held balance.

Refresh strategy:

Trigger	Frequency	Method
User opens Dashboard	On demand	GET `/v1/accounts/{id}/balances`
Background sync	Every 6 hours (max 4/day per PSD2 RTS)	Scheduled job per linked account
Pre-payment check	Before PISP initiation	GET `/v1/accounts/{id}/balances`
User manual refresh	Pull-to-refresh gesture	GET `/v1/accounts/{id}/balances`

PSD2 RTS constraint: A TPP may access the ASPSP's account data **a maximum of 4 times per day** without the PSU's active request (Art. 36(6) RTS). User-initiated requests are unlimited.

Drop API mapping:

- `GET /api/auth/me` returns `totalBalance` and `bankAccounts[].balance` — these are cached AISP reads from the `bank_accounts` table
- `bank_accounts.balance_synced_at` tracks when the balance was last refreshed from the ASPSP

3.3 Data Storage

AISP Data	Drop DB Column	Table	Notes
Account IBAN	`bank_accounts.iban`	`bank_accounts`	Stored on linking
Account name	`bank_accounts.bank_name`	`bank_accounts`	ASPSP name (e.g., "DNB")
Account number	`bank_accounts.account_number`	`bank_accounts`	Domestic format
Balance amount	`bank_accounts.balance`	`bank_accounts`	Cached AISP read (stored in oere)
Balance timestamp	`bank_accounts.balance_synced_at`	`bank_accounts`	Last refresh time
Consent ID	(new column needed)	`bank_accounts` or `consents`	Links to ASPSP consent
Consent expiry	(new column needed)	`consents`	Max 90 days from grant

---

4. PISP — Payment Initiation Service

4.1 Payment Initiation Flow

PISP initiates payments from the user's bank account. Every PISP transaction requires SCA with **dynamic linking** — the authentication must be tied to the specific amount and payee (PSD2 RTS Art. 97(2)).

```
sequenceDiagram
    participant U as User (Browser/App)
    participant D as Drop API
    participant A as ASPSP (User's Bank)
    participant R as Recipient Bank

    U->>D: POST /api/transactions/remittance<br/>{recipientId, amount, bankAccountId}
    D->>D: Validate: KYC approved, balance sufficient,<br/>amount 100-50000 NOK, exchange rate lookup
    D->>D: POST /api/transactions/disclosure<br/>Calculate fee (0.5%), FX rate, total
    D-->>U: Show pre-payment disclosure<br/>(PSD2 Art. 45/46 compliance)
    U->>D: Confirm payment
    D->>A: POST /v1/payments/sepa-credit-transfers<br/>{debtorAccount: {iban},<br/>instructedAmount: {currency, amount},<br/>creditorAccount: {iban}, creditorName,<br/>remittanceInformationUnstructured}
```

```
    A-->>D: 201 {paymentId, transactionStatus: "RCVD",<br/>_links: {scaRedirect:
"https://bank.no/sca/pay/..."}}
    D->>D: Create transaction record<br/>(status: "processing", idempotency_key set)
    D-->>U: Redirect to bank SCA page
    U->>A: Authenticate payment with BankID<br/>(dynamic linking: amount + payee shown)
    A-->>U: Redirect to Drop callback
    U->>D: GET /api/payments/callback?paymentId=xxx
    D->>A: GET /v1/payments/{paymentId}/status
    A-->>D: {transactionStatus: "ACCP"}
    D->>D: Update transaction status to "completed"
    D-->>U: Payment confirmed
    Note over A,R: Bank executes SEPA/SWIFT transfer<br/>Settlement via interbank rails
```

4.2 Payment Types

Drop Transaction Type	Berlin Group Payment Product	Use Case	Settlement Time
QR Payment (domestic)	`domestic-credit-transfers`	Pay merchant in Norway	Instant (SEPA Inst) or T+1
Remittance (EEA)	`sepa-credit-transfers`	Send to EU/EEA countries	1-2 business days
Remittance (non-EEA)	`cross-border-credit-transfers`	Send to Serbia, Pakistan, Turkey, etc.	2-4 business days

4.3 Dynamic Linking (PSD2 RTS Art. 97(2))

For every PISP payment, the SCA procedure must incorporate **dynamic linking**:

- The payer must be made aware of the **amount** and **payee** during authentication
- The authentication code must be **specific** to that amount and payee
- Any change to amount or payee invalidates the authentication

Implementation: Drop passes `instructedAmount` and `creditorName` in the PISP request. The ASPSP displays these on the BankID SCA screen. The user confirms by authenticating, creating a cryptographic link between the consent and the specific transaction parameters.

4.4 Idempotency

Drop uses the `idempotency_key` column in the `transactions` table (with a unique index `idx_tx_idempotency`) to prevent duplicate payments:

1. Generate `idempotency_key` from: `{userId}:{recipientId}:{amount}:{timestamp_minute}`
2. Include as `X-Request-ID` header in PISP API calls

3. If ASPSP returns a duplicate error, look up existing transaction by `idempotency_key`
4. Return the existing transaction to the user (no double-charge)

---

5. Consent Lifecycle

5.1 State Diagram

```
stateDiagram-v2
    [*] --> Requested: User initiates bank linking
    Requested --> ScaPending: ASPSP returns scaRedirect
    ScaPending --> Valid: User completes SCA
    ScaPending --> Failed: SCA timeout / user cancels
    ScaPending --> Failed: SCA rejected by bank
    Valid --> Valid: Balance refresh (within 90 days)
    Valid --> Expired: 90-day validity exceeded
    Valid --> RevokedByUser: User unlinks bank account
    Valid --> RevokedByASPSP: Bank revokes access
    Valid --> RenewalPending: 90 days before expiry, prompt renewal
    RenewalPending --> ScaPending: User re-authenticates
    RenewalPending --> Expired: User ignores renewal
    Expired --> Requested: User re-links account
    Failed --> Requested: User retries
    RevokedByUser --> [*]
    RevokedByASPSP --> Requested: User re-links
    Expired --> [*]
```

5.2 Consent Properties

Property	Value	PSD2 Reference
Maximum validity	90 days	RTS Art. 10(1)
Renewal SCA required	Yes, every 90 days	RTS Art. 10(2)
Access frequency (TPP-initiated)	Max 4x/day per account	RTS Art. 36(6)
Access frequency (PSU-initiated)	Unlimited	RTS Art. 36(6)
Revocation	User can revoke at any time	PSD2 Art. 94
Scope	Per-account (balances + transactions)	Berlin Group consent model

Property	Value	PSD2 Reference
Combined service	`false` (AISP separate from PISP)	Berlin Group `combinedServiceIndicator`

5.3 Consent Storage

Drop tracks consents in two places:

1. **`consents` table** — GDPR consent records (consent_type: `psd2_aisp`, `psd2_pisp`)
2. **`bank_accounts` table** — Links to ASPSP consent ID for each linked account

When a consent expires or is revoked:

- `bank_accounts.balance` is zeroed (stale data removed)
- `bank_accounts.balance_synced_at` is nulled
- User is prompted to re-link via notification

---

6. SCA Requirements

6.1 When SCA Is Required

Operation	SCA Required	SCA Type
AISP consent creation	Yes	Redirect to bank (BankID)
AISP consent renewal (90 days)	Yes	Redirect to bank (BankID)
AISP balance read (after initial consent)	No	Access token sufficient
PISP payment initiation	Yes, always	Redirect to bank with dynamic linking
PISP payment > threshold	Yes (no exemption)	Drop does not apply SCA exemptions

6.2 SCA Methods

Drop does not perform SCA directly. SCA is delegated to the ASPSP (user's bank), which uses BankID as the authentication mechanism. The flow is:

1. Drop sends AISP consent or PISP payment request to ASPSP
2. ASPSP returns an `scaRedirect` URL
3. Drop redirects the user to the bank's SCA page
4. User authenticates with BankID (knowledge + possession factors)

5. Bank redirects back to Drop with the result

6.3 SCA Exemptions

Drop does **not** apply SCA exemptions for PISP transactions. All payments require full SCA regardless of amount. This is a conservative approach that:

- Simplifies implementation
- Reduces fraud risk
- Avoids complex exemption logic (low value, trusted beneficiary, recurring)
- Aligns with Norwegian banks' SCA enforcement

---

7. Fallback Mechanisms

7.1 ASPSP API Unavailability

If the ASPSP's dedicated PSD2 API is unavailable:

Scenario	Fallback	PSD2 Reference
API down (AISP)	Show last cached balance with timestamp	RTS Art. 33(4)
API down (PISP)	Display error, suggest retry later	No fallback — payment requires live API
API degraded (slow)	30s timeout, retry once	Standard HTTP retry
API returns 5xx	Circuit breaker (3 failures → 60s cooldown)	Operational resilience
Consent expired	Prompt user to re-authenticate	Renewal flow

7.2 Fallback Access (Screen Scraping)

Under PSD2 RTS Art. 33(4), if an ASPSP's dedicated API does not meet availability and performance standards, the TPP may fall back to the ASPSP's customer-facing online banking interface. Drop does **not** plan to implement screen scraping — instead relying on aggregators (Neonomics, Tink) who handle multi-bank connectivity and fallback.

7.3 Multi-Bank Strategy

Norwegian users may have accounts at multiple banks. Drop supports multiple linked accounts via the `bank_accounts` table (one marked `is_primary`). Each linked account has its own AISP consent with its own 90-day lifecycle.

---

8. Error Handling

8.1 ASPSP Error Codes (Berlin Group)

HTTP Status	Berlin Group Code	Drop Handling	User Message
400	`FORMAT_ERROR`	Log + show validation error	"Kunne ikke koble til banken. Sjekk kontonummeret."
401	`CERTIFICATE_INVALID`	Alert ops, block requests	"Teknisk feil. Prøv igjen senere."
403	`CONSENT_INVALID`	Delete consent, prompt re-link	"Tilgangen til banken din er utløpt. Koble til på nytt."
403	`CONSENT_EXPIRED`	Delete consent, prompt re-link	"Tilgangen til banken din er utløpt. Koble til på nytt."
404	`RESOURCE_UNKNOWN`	Log + remove stale data	"Kontoen ble ikke funnet i banken."
429	`ACCESS_EXCEEDED`	Back off, use cached data	"For mange forespørsler. Viser sist kjente saldo."
500+	Server error	Circuit breaker, use cached data	"Banken svarer ikke. Prøv igjen senere."

8.2 Payment-Specific Errors

Scenario	ASPSP Response	Drop Handling
Insufficient funds	`RJCT` (rejected)	Update transaction status to `failed`, notify user
Invalid IBAN	`FORMAT_ERROR`	Reject before sending to ASPSP (validate locally)
SCA timeout	No callback within 5 min	Mark transaction as `failed`, release hold
SCA cancelled	User cancels at bank	Mark transaction as `failed`
Duplicate payment	`DUPLICATE` or HTTP 409	Look up by `idempotency_key`, return existing

---

9. Aggregator Strategy

Rather than integrating directly with each ASPSP, Drop plans to use an **Open Banking aggregator** for production:

Aggregator	Coverage	Strengths	Consideration
Neonomics (Norwegian)	Nordics + EEA	Strong Nordic bank coverage, Norwegian company	Primary candidate
Tink (Visa)	EU-wide (6000+ banks)	Largest coverage, Visa backing	Broader coverage
Enable Banking	Nordics	Direct PSD2, no screen scraping	Privacy-focused

Benefits of aggregator approach:

- Single API integration (vs. per-bank integration)
- Aggregator handles eIDAS certificates, bank onboarding, fallback
- Faster time-to-market
- Aggregator maintains bank API compatibility as banks update

Trade-offs:

- Additional per-transaction cost
- Data passes through third party (GDPR implications)
- Dependency on aggregator uptime

---

10. Implementation Roadmap

Phase	Milestone	Dependencies
Current (MVP)	Mock AISP/PISP (local DB balances, simulated payments)	None
Phase 2a	Aggregator sandbox integration (Neonomics or Tink)	Aggregator contract signed
Phase 2b	BankID OIDC for Drop auth + ASPSP SCA for consents	BankID client credentials
Phase 2c	AISP live (read real balances from DNB, SpareBank 1)	eIDAS cert + consent flow
Phase 2d	PISP live (initiate real payments)	Finanstilsynet license or agent arrangement

Phase	Milestone	Dependencies
Phase 3	Multi-bank support, consent renewal automation, SEPA Instant	Production scaling

---

11. Cross-References

- **BankID OIDC Integration:** bankid-oidc-integration.md — Drop's authentication layer (separate from ASPSP SCA)
- **Payment Processing:** payment-processing.md — SEPA/SWIFT settlement, FX, fees
- **Security Architecture:** ../hld/security-architecture.md — Threat model, SCA controls
- **Bank Account Linking Flow:** ../lld/flow-bank-account-linking.md — Detailed AISP consent UX
- **Remittance Flow:** ../lld/flow-remittance.md — Detailed PISP payment UX
- **Database Schema:** ../../backend/DATABASE-SCHEMA.md — `bank_accounts`, `transactions`, `consents` tables
- **API Reference:** ../../backend/API-REFERENCE.md — Drop API endpoints
- **Compliance Status:** ../../security/COMPLIANCE.md — PSD2 readiness assessment

---

Revision #5
Created 2026-02-21 05:59:03 UTC by John
Updated 2026-05-23 10:57:02 UTC by John