Runbook: PISP Payment Failure

Runbook: PISP Payment Failure (Remittance & QR)

Service: Payment Initiation (PISP via Open Banking) **Severity:** HIGH (blocks money transfers)
MTTR Target: <30 minutes **Owner:** John (AI Director)

Overview

PISP (Payment Initiation Service Provider) enables Drop to initiate payments directly from users' bank accounts. Failures in PISP prevent both **remittance** (send money abroad) and **QR payments** (in-store merchant payments).

Symptoms

Users report they cannot complete payments:

- Payment initiation fails with error message
- Payment status stuck at "pending" indefinitely
- Bank redirect loop (never returns to Drop)
- Error: "Payment service unavailable"

User impact: Cannot send money or pay merchants.

Diagnosis

1. Identify Payment Type

Determine which payment flow is affected:

- **Remittance:** User sends money to recipient abroad (`POST /api/transactions/remittance`)
- **QR Payment:** User pays merchant by scanning QR code (`POST /api/transactions/qr-payment`)

Check recent transactions:

```
# CloudWatch Logs
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "payment_initiation" \
  --start-time $(date -u -d '30 minutes ago' +%s)000 \
  --region eu-west-1 \
  | jq '.events[].message' \
  | grep -E "remittance|qr_payment|pisp_error"
```

2. Check Open Banking Provider Status

Provider: Neonomics (Norway), Swan BaaS (cross-border)

Neonomics Status:

```
# No official status page — check via test API call
curl -X POST https://sandbox.neonomics.io/payments/v1/payment-initiation \
  -H "Authorization: Bearer <sandbox-token>" \
  -H "Content-Type: application/json" \
  -d '{"amount":100,"currency":"NOK"}' \
  -v

# Expected: HTTP 200 or 400 (validation error)
# If 500/503: Neonomics outage
```

Swan API Status:

```
# Check Swan status page
open https://status.swan.io

# Or test API
curl https://api.swan.io/graphql \
  -H "Authorization: Bearer <api-key>" \
  -d '{"query": "{viewer{id}}"}' \
```

```
  -v

# Expected: HTTP 200
# If 500/503: Swan outage
```

3. Check Drop Logs for Error Codes

Common PISP error codes:

Code	Meaning	Cause
`INSUFFICIENT_FUNDS`	User's bank account balance too low	User error
`ACCOUNT_NOT_ACCESSIBLE`	Bank account locked or closed	Bank issue
`CONSENT_EXPIRED`	Open Banking consent needs renewal	User must re-authenticate
`PAYMENT_REJECTED`	Bank declined payment	Fraud detection, limits
`TIMEOUT`	Bank API took too long to respond	Network/bank issue
`INVALID_IBAN`	Recipient bank account number invalid	User error
`LIMIT_EXCEEDED`	Payment exceeds daily limit	User or bank limit

Search logs for error codes:

```
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "PISP_ERROR" \
  --start-time $(date -u -d '1 hour ago' +%s)000 \
  | jq -r '.events[].message' \
  | jq '.metadata.errorCode'
```

4. Test Payment Flow

Manual test (staging environment):

```
# 1. Login
TOKEN=$(curl -X POST https://drop-staging.fly.dev/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","password":"test1234"}' \
  | jq -r '.data.token')
```

```
# 2. Initiate test payment (small amount)
curl -X POST https://drop-staging.fly.dev/api/transactions/remittance \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "recipientId": "rec_test123",
    "amount": 100,
    "currency": "NOK",
    "sendCurrency": "NOK",
    "receiveCurrency": "EUR"
  }' \
  -v

# Expected: HTTP 200, transaction created
# If 500: PISP integration broken
```

Common Causes & Solutions

Cause 1: Open Banking Provider Outage

Probability: 10% (Neonomics/Swan service disruption)

Symptoms:

- All payments fail with timeout or 503 error
- Provider status page reports incident
- Test API call fails

Solution:

1. **Verify outage:**
   - Check Neonomics/Swan status pages
   - Contact provider support if no public status
2. **Communicate to users:**

```
Subject: Payment processing temporarily unavailable
Body: Our payment provider is experiencing issues.
      We're monitoring the situation and expect service
      to resume within <X> minutes.
```

3. **Monitor provider status:**
   - Subscribe to provider status updates
   - Check every 15 minutes for resolution
4. **Queue failed payments (if applicable):**
   - Store payment requests in `pending_payments` table
   - Retry automatically when provider is back online

ETA: Depends on provider (typically <2 hours)

Cause 2: Expired Open Banking Consent

Probability: 30% (user consent expires after 90 days)

Symptoms:

- Error code: `CONSENT_EXPIRED` or `ACCOUNT_NOT_ACCESSIBLE`
- Payments fail for specific users only (not all)
- Logs show: "Open Banking consent invalid"

Solution:

1. **Identify affected users:**

   ```
   SELECT user_id, bank_account_id, consent_expires_at
   FROM bank_accounts
   WHERE consent_expires_at < datetime('now');
   ```

2. **Notify users to re-authenticate:**
   - Send push notification: "Please reconnect your bank account"
   - In-app banner: "Bank connection expired, tap to reconnect"
3. **Guide user through re-consent flow:**
   - User taps "Reconnect bank account"
   - Redirect to AISP consent flow (BankID + bank approval)
   - Update `consent_expires_at` in database (90 days from now)
4. **Retry payment after re-consent:**
   - Original payment request should be retryable
   - Or user initiates new payment

ETA: Immediate (user action required)

Cause 3: Insufficient Funds in User's Bank Account

Probability: 25% (user error)

Symptoms:

- Error code: `INSUFFICIENT_FUNDS`
- Payment fails for specific transaction only
- Logs show: "Account balance too low"

Solution:

1. **Show clear error message to user:**

   ```
   Payment failed: Insufficient funds
   Your bank account balance is too low to complete this payment.
   Please add funds or choose a different payment method.
   ```

2. **Suggest alternatives:**
   - Link different bank account (if multi-account supported)
   - Reduce payment amount
   - Try again later
3. **No action needed on Drop side** (user must resolve)

ETA: N/A (user-side issue)

Cause 4: Bank Fraud Detection / Payment Rejection

Probability: 15% (bank security systems)

Symptoms:

- Error code: `PAYMENT_REJECTED` or `SECURITY_BLOCK`
- Payment fails after bank redirect
- Logs show: "Bank declined transaction"

Solution:

1. **Advise user to contact their bank:**

   ```
   Payment failed: Your bank declined this transaction.
   This may be due to fraud protection or payment limits.
   Please contact your bank to authorize the payment.
   ```

2. **Check if payment is unusual for user:**

- First international transfer?
- Amount significantly higher than usual?
- High-risk destination country?

3. **User should:**
   - Call their bank's fraud department
   - Confirm the payment is legitimate
   - Ask bank to whitelist Drop payments
   - Retry after bank approval
4. **Document pattern:**
   - If many users from same bank report this, investigate bank compatibility
   - May need to add bank-specific messaging

ETA: Depends on user's bank (minutes to hours)

Cause 5: PISP API Rate Limiting

Probability: 5% (during high-traffic periods)

Symptoms:

- Error code: `RATE_LIMIT_EXCEEDED` or HTTP 429
- Intermittent failures (some payments succeed, others fail)
- Logs show: "Too many requests"

Solution:

1. **Check rate limit headers:**

```
# Find rate limit status in logs
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "X-RateLimit" \
  --start-time $(date -u -d '10 minutes ago' +%s)000
```

2. **Implement request queuing:**

```
// src/lib/pisp-client.ts
const queue = new PQueue({ concurrency: 5, interval: 1000 });

async function initiatePayment(params) {
  return queue.add(() => pisService.createPayment(params));
}
```

3. **Exponential backoff on retry:**

```
async function retryPayment(id, attempt = 1) {
  if (attempt > 3) throw new Error('Max retries exceeded');
  try {
    return await initiatePayment(id);
  } catch (error) {
    if (error.status === 429) {
      await sleep(1000 * Math.pow(2, attempt)); // 2s, 4s, 8s
      return retryPayment(id, attempt + 1);
    }
    throw error;
  }
}
```

4. **Contact provider to increase limits (if persistent):**
   - Email Neonomics support with usage stats
   - Request higher API quota for production

ETA: 5 minutes (automatic retry), 1-2 days (if quota increase needed)

Cause 6: Invalid Recipient Bank Account (IBAN/SWIFT)

Probability: 20% (user input error)

Symptoms:

- Error code: `INVALID_IBAN` or `ACCOUNT_NOT_FOUND`
- Payment fails immediately (no bank redirect)
- Logs show: "Recipient account validation failed"

Solution:

1. **Show clear validation error:**

```
Payment failed: Invalid recipient bank account
The IBAN you entered is not valid. Please check and try again.
IBAN: DE89 3704 0044 0532 0130 00 (example format)
```

2. **Improve frontend validation:**
   - Add real-time IBAN validation (checksum algorithm)
   - Use IBAN validation library (e.g., `ibantools`)

- Show format hints per country

3. **Ask user to verify recipient details:**
   - Double-check IBAN/SWIFT code
   - Confirm with recipient
   - Try alternative payment method if IBAN is correct but still rejected

ETA: Immediate (user correction)

Emergency Workarounds

Option 1: Manual Payment Processing

Use case: PISP provider down >2 hours, urgent payments needed

Steps:

1. Collect payment requests manually:

```
SELECT id, user_id, amount, currency, recipient_iban
FROM transactions
WHERE status = 'pending' AND created_at > datetime('now', '-2 hours');
```

2. **Alem initiates payments manually** via Drop's business bank account:
   - Log into business banking portal
   - Enter recipient details manually
   - Process payment one by one
3. Update Drop transaction status:

```
UPDATE transactions SET status = 'completed', completed_at = datetime('now')
WHERE id = '<transaction-id>';
```

4. Notify users:

```
Subject: Your payment has been processed
Body: Your payment of <amount> to <recipient> has been completed manually
      due to a temporary service issue. Thank you for your patience.
```

Risk: Manual work, prone to errors. Only use for critical/urgent payments.

Option 2: Redirect to Alternative Payment Method

Use case: PISP down, no ETA, users need alternative

Steps:

1. Show modal in app:

   ```
   Payment Initiation Unavailable
   Our payment service is temporarily down.
   Alternative options:
   - Bank transfer (manual IBAN entry)
   - Try again later (we'll notify you when service is restored)
   ```

2. Provide manual bank transfer instructions:

   ```
   Transfer to:
   Account holder: Drop AS
   IBAN: NO93 8601 1117 947
   Amount: <calculated-amount>
   Reference: <unique-ref>
   ```

3. Monitor for manual transfers:
   - Check business bank account for incoming payments
   - Match reference code to pending Drop transactions
   - Mark as completed when received

ETA: Immediate (user can pay via manual transfer)

Monitoring & Alerts

Metrics to Track

- **Payment success rate:** Should be >95%
- **Payment latency:** p50 <5s, p95 <15s, p99 <30s
- **Error rate by code:** Track `INSUFFICIENT_FUNDS`, `CONSENT_EXPIRED`, `TIMEOUT` separately

Alert Rules

```
// src/lib/payment-monitor.ts
export async function trackPaymentFailure(errorCode: string, transactionId: string) {
  const failureRate = await calculateFailureRate('last_5_minutes');

  if (failureRate > 0.1) { // 10% failure rate
    await sendAlert({
      severity: 'critical',
      title: 'High payment failure rate',
      message: `${(failureRate * 100).toFixed(1)}% of payments failing in last 5 min`,
    });
  }
}
```

Dashboard Queries

```
-- Payment success rate (last 24h)
SELECT
  COUNT(*) FILTER (WHERE status = 'completed') * 100.0 / COUNT(*) as success_rate,
  COUNT(*) as total_payments
FROM transactions
WHERE created_at > datetime('now', '-24 hours');

-- Top error codes (last hour)
SELECT error_code, COUNT(*) as count
FROM transactions
WHERE status = 'failed' AND created_at > datetime('now', '-1 hour')
GROUP BY error_code
ORDER BY count DESC;
```

Post-Incident Actions

1. **Update transaction status:**

   ```
   -- Mark timed-out payments as failed (after 1 hour)
   UPDATE transactions
   SET status = 'failed', error_code = 'TIMEOUT', error_message = 'Payment timed out'
   WHERE status = 'pending' AND created_at < datetime('now', '-1 hour');
   ```

2. **Notify affected users:**
   - Send email/push notification about failed payment
   - Offer to retry or refund
3. **Document incident:**
   - Create post-mortem in `comms/incidents/`
   - Track downtime duration
   - Calculate financial impact (lost transactions)
4. **Review provider SLA:**
   - Check if outage violates SLA
   - Request compensation/credits if applicable
5. **Improve resilience:**
   - Add payment retry queue
   - Implement circuit breaker for provider API
   - Consider multi-provider failover (backup PISP)

Escalation

Time	Action
0 min	John starts diagnosis
10 min	If provider outage confirmed, notify Alem
30 min	If not resolved, assess manual processing need
1 hour	If critical payments pending, start manual workaround (Alem approval)
2 hours	Public communication to all users

Contacts

- **Neonomics Support:** support@neonomics.io, Slack: #neonomics-support
- **Swan Support:** support@swan.io (email), Swan Slack (if available)
- **Internal:** Alem (CEO, manual payment approval)

Related Documentation

- `docs/architecture/payments.md` — PISP flow diagrams
- `src/app/api/transactions/remittance/route.ts` — Remittance implementation
- `src/app/api/transactions/qr-payment/route.ts` — QR payment implementation

- `docs/compliance/psd2-requirements.md` — Regulatory requirements

Last Updated: 2026-02-22 **Next Review:** Before Phase 2 (Banking Integration) **Test Status:** Pending (Phase 2 live payments)

Revision #6
Created 2026-02-23 11:29:21 UTC by John
Updated 2026-05-25 07:27:43 UTC by John