Runbooks

Operational runbooks for failure scenarios

Runbook: AISP Balance Failure

Runbook: AISP Balance Fetch Failure

Service: AISP (Account Information Service Provider) **Severity:** MEDIUM (users can't see bank balance) **MTTR Target:** <20 minutes **Owner:** John (AI Director)

Symptoms

Users report they cannot see their bank account balance in Drop. Symptoms include:

- Dashboard shows "Balance unavailable" or stale balance
- Error message: "Could not fetch account information"
- Infinite loading spinner on balance widget
- Balance shows "0 kr" or "—" instead of actual amount

User impact: Cannot verify available funds before making payments (may lead to insufficient funds errors).

Diagnosis

1. Check Neonomics AISP Status

External status:

```
# Neonomics has no public status page — test via API
curl -X GET https://api.neonomics.io/health \
  -H "Authorization: Bearer <api-key>" \
  -v

# Expected: HTTP 200
```

```
# If 500/503: Neonomics outage
```

Check specific bank connectivity:

```
# List supported banks and their status
curl -X GET https://api.neonomics.io/banks \
  -H "Authorization: Bearer <api-key>" \
  | jq '.[] | select(.country == "NO") | {name, status}'

# Look for: "status": "degraded" or "offline"
```

2. Check Drop Logs

```
# CloudWatch Logs (production)
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "aisp" \
  --start-time $(date -u -d '15 minutes ago' +%s)000 \
  --region eu-west-1

# Look for:
# - "AISP consent expired"
# - "AISP API timeout"
# - "AISP 401 Unauthorized"
# - "Bank API unavailable: DNB"
```

3. Check User Consent Status

```
# Verify Open Banking consent hasn't expired
# Consent is valid for 90 days from last authorization

# Check database for expired consents (PostgreSQL 16)
psql "$DATABASE_URL" <<EOF
SELECT
  user_id,
  bank_name,
  consent_expires_at,
  EXTRACT(EPOCH FROM (consent_expires_at - NOW())) / 86400 AS days_remaining
```

```
FROM bank_accounts
WHERE consent_expires_at < NOW() + INTERVAL '7 days'
ORDER BY consent_expires_at ASC
LIMIT 10;
EOF

# If days_remaining < 0: consent expired
# If days_remaining < 7: warn user to renew soon
```

4. Test AISP Flow

Manual test (staging):

```
# 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. Fetch balance
curl -X GET https://drop-staging.fly.dev/api/accounts/balance \
  -H "Authorization: Bearer $TOKEN" \
  -v

# Expected: HTTP 200, { "balance": 15000.50, "currency": "NOK" }
# If 401: Consent expired
# If 500: AISP integration broken
```

5. Check Rate Limiting

```
# Check if Neonomics API rate limit exceeded
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "rate_limit" \
  --start-time $(date -u -d '10 minutes ago' +%s)000 \
  | jq '.events[].message' \
  | grep -E "429|X-RateLimit"

# If many 429 errors: rate limiting issue
```

Common Causes & Solutions

Cause 1: Expired Open Banking Consent

Probability: 40% (PSD2 consent expires after 90 days)

Symptoms:

- Error code: `CONSENT_EXPIRED` or `CONSENT_INVALID`
- Logs show: "AISP consent no longer valid"
- Specific users affected (not all users)

Solution:

1. **Identify affected users:**

```
-- PostgreSQL 16
SELECT user_id, email, bank_name, consent_expires_at
FROM bank_accounts
JOIN users ON users.id = bank_accounts.user_id
WHERE consent_expires_at < NOW();
```

2. **Notify users to re-authorize:**
   Push notification (Norwegian):

```
Banktilkobling utløpt
Godkjenningen for å hente saldo fra [Bank] har utløpt.
Trykk her for å fornye tilkoblingen.
```

   Email (Norwegian):

```
Emne: Godkjenn tilgang til bankkonto på nytt

Hei,

Din godkjenning for å vise saldo fra [Bank] har utløpt etter 90 dager.
Dette er et PSD2-sikkerhetskrav.

Logg inn i Drop og koble til bankkontoen på nytt for å fortsette å se saldoen din.

Mvh,
```

```
Drop
```

3. **Guide user through re-consent:**
   - User taps notification → redirect to "Reconnect Bank Account" screen
   - Initiate new AISP consent flow (BankID + bank authorization)
   - Update `consent_expires_at` = NOW() + INTERVAL '90 days'
4. **Automatic consent renewal reminder:**

```
# Cron job to warn users 7 days before expiry
# Send reminder: "Your bank connection expires in 7 days, renew now"
```

ETA: Immediate (user action required)

Cause 2: Bank API Outage or Maintenance

Probability: 15% (specific bank temporarily unavailable)

Symptoms:

- All users of specific bank (e.g., DNB, Nordea) cannot fetch balance
- Other banks work fine
- Logs show: "Bank API timeout" or "502 Bad Gateway"

Solution:

1. **Identify affected bank:**

```
# Check which bank is failing
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "Bank API" \
  --start-time $(date -u -d '30 minutes ago' +%s)000 \
  | jq '.events[].message' \
  | grep -o '"bank":"[^"]*"' \
  | sort | uniq -c | sort -rn

# Example output: "bank":"DNB" appears 50 times
```

2. **Check bank status:**
   - Visit bank's website: check for maintenance announcements
   - Norwegian banks often schedule maintenance 02:00-06:00 CET
   - DNB status: https://www.dnb.no/drift
   - Nordea status: https://www.nordea.no/info/driftsmeldinger

3. **Notify affected users (Norwegian):**

```
Emne: Saldo midlertidig utilgjengelig for [Bank]

Hei,

Vi opplever for øyeblikket problemer med å hente saldo fra [Bank].
Dette skyldes tekniske problemer hos banken.

Du kan fortsatt gjøre betalinger, men saldoen vises ikke akkurat nå.
Vi jobber med å gjenopprette tjenesten.

Estimert løsning: [X minutter/timer]

Mvh,
Drop
```

4. **Implement graceful degradation:**

```
// src/app/api/accounts/balance/route.ts
async function fetchBalance(userId: string) {
  try {
    return await neonomicsClient.getBalance(userId);
  } catch (error) {
    if (error.code === 'BANK_API_TIMEOUT') {
      // Return cached balance with warning
      const cached = await getCachedBalance(userId);
      return {
        balance: cached?.balance || null,
        currency: 'NOK',
        lastUpdated: cached?.timestamp,
        warning: 'Balance may be outdated due to bank API issues'
      };
    }
    throw error;
  }
}
```

ETA: Depends on bank (typically <2 hours for maintenance, <1 hour for incidents)

Cause 3: Neonomics API Outage

Probability: 10% (Neonomics service disruption)

Symptoms:

- ALL users cannot fetch balance regardless of bank
- Logs show: "Neonomics API unreachable" or HTTP 503
- Test API call to Neonomics fails

Solution:

1. **Verify Neonomics outage:**

```
# Test Neonomics health endpoint
curl -X GET https://api.neonomics.io/health \
  -H "Authorization: Bearer <api-key>" \
  -v

# If timeout or 503: confirmed outage
```

2. **Contact Neonomics support:**
   - Email: support@neonomics.io
   - Slack: #neonomics-support (if available)
   - Check Neonomics Slack for incident updates
3. **Enable fallback mode:**

```
# Show cached balances to all users
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    AISP_FALLBACK_MODE=cached,
    AISP_FALLBACK_CACHE_TTL=3600
  }"
```

4. **Communicate to users (Norwegian):**

```
Emne: Saldo vises med forsinkelse

Hei,

Vår leverandør for bankdata opplever tekniske problemer.
Saldoen du ser kan være opptil 1 time gammel.

Du kan fortsatt gjøre betalinger som normalt.
```

```
Vi forventer at tjenesten er tilbake innen [X minutter].

Mvh,
Drop
```

5. **Monitor Neonomics status:**
   - Check every 10 minutes for resolution
   - When API is back: disable fallback mode

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    AISP_FALLBACK_MODE=live
  }"
```

ETA: Depends on Neonomics (typically <2 hours)

Cause 4: Invalid or Revoked API Credentials

Probability: 5% (after credential rotation or account issue)

Symptoms:

- Logs show: "401 Unauthorized" or "invalid_api_key"
- All AISP requests fail immediately
- Other Drop services work fine (auth, database, etc.)

Solution:

1. **Verify Neonomics API credentials:**

```
bw get item "Neonomics API" --session $BW_SESSION

# Check:
# - API key is not expired
# - API key has AISP permissions
# - Correct environment (production vs sandbox)
```

2. **Update App Runner environment variables:**

```
aws apprunner update-service --service-arn <ARN> \
  --source-configuration "ImageRepository={...}" \
  --instance-configuration "EnvironmentVariables={
```

```
    NEONOMICS_API_KEY=<correct-key>,
    NEONOMICS_ENVIRONMENT=production
  }"
```

3. **Trigger deployment:**

```
aws apprunner start-deployment --service-arn <ARN> --region eu-west-1

# Wait 3-5 minutes for deployment to complete
```

4. **Test after deployment:**

```
# Verify AISP working
curl -X GET https://getdrop.no/api/accounts/balance \
  -H "Authorization: Bearer <test-user-token>" \
  -v

# Expected: HTTP 200 with balance data
```

ETA: 10 minutes

Cause 5: Network or Firewall Issues

Probability: 5% (AWS security group misconfiguration)

Symptoms:

- Logs show: "Connection timeout" or "ECONNREFUSED"
- AISP API requests never reach Neonomics
- Other external APIs may also fail

Solution:

1. **Check outbound connectivity:**

```
# App Runner egress is unrestricted by default
# If using VPC connector, check security group
aws ec2 describe-security-groups \
  --group-ids <vpc-connector-sg> \
  --region eu-west-1 \
  | jq '.SecurityGroups[].IpPermissionsEgress'
```

2. **Test DNS resolution:**

```
# From your local machine or bastion host
nslookup api.neonomics.io

# Should resolve to Neonomics IP
# If NXDOMAIN: DNS issue
```

3. **Check AWS service health:**

```
# Check App Runner service events
aws apprunner list-operations \
  --service-arn <ARN> \
  --region eu-west-1 \
  | jq '.OperationSummaryList[] | select(.Type == "CREATE_SERVICE" or .Type ==
"UPDATE_SERVICE")'

# Look for recent errors
```

4. **Whitelist Neonomics IPs (if using strict firewall):**
   - Contact Neonomics for IP ranges
   - Add to security group outbound rules
   - Allow HTTPS (443) to Neonomics endpoints

ETA: 15 minutes (if quick fix), 1 hour (if requires networking changes)

Cause 6: Rate Limiting (High Traffic)

Probability: 10% (during peak hours or viral event)

Symptoms:

- Logs show: HTTP 429 "Too Many Requests"
- Intermittent failures (some users see balance, others don't)
- Rate limit headers in logs

Solution:

1. **Check rate limit headers:**

```
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "X-RateLimit" \
```

```
  --start-time $(date -u -d '5 minutes ago' +%s)000 \
  | jq -r '.events[].message' \
  | grep -E "X-RateLimit-(Limit|Remaining|Reset)"
```

2. **Implement request throttling:**

```
// src/lib/aisp-client.ts
import PQueue from 'p-queue';

const queue = new PQueue({
  concurrency: 10,      // Max 10 concurrent requests
  interval: 1000,        // Per second
  intervalCap: 50        // Max 50 requests per second
});

export async function fetchBalance(userId: string) {
  return queue.add(() => neonomicsClient.getBalance(userId));
}
```

3. **Cache balance aggressively during rate limit:**

```
// src/lib/balance-cache.ts
const CACHE_TTL_NORMAL = 60;      // 60 seconds
const CACHE_TTL_RATE_LIMIT = 300; // 5 minutes during rate limit

export async function getBalanceWithCache(userId: string) {
  const cached = await redis.get(`balance:${userId}`);
  if (cached) return JSON.parse(cached);

  try {
    const balance = await fetchBalance(userId);
    await redis.setex(`balance:${userId}`, CACHE_TTL_NORMAL,
JSON.stringify(balance));
    return balance;
  } catch (error) {
    if (error.status === 429) {
      // Extend cache TTL during rate limit
      await redis.expire(`balance:${userId}`, CACHE_TTL_RATE_LIMIT);
    }
    throw error;
  }
```

```
}
```

4. **Contact Neonomics to increase rate limit:**
   - Email support with traffic stats
   - Request higher API quota for production
   - Provide justification (user growth, peak times)

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

Emergency Workarounds

Option 1: Cached Balance Mode

Use case: AISP provider down >30 minutes, users need to see approximate balance

Steps:

1. Enable cached balance fallback:

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    AISP_MODE=cached,
    AISP_CACHE_TTL=3600
  }"
```

2. Show warning banner in app:

```
⚠ Saldo vises med forsinkelse
Vi viser din sist kjente saldo fra [timestamp].
Tjenesten er tilbake til normal snart.
```

3. Allow payments to proceed:
   - Users can still initiate payments (PISP)
   - Balance check uses cached value
   - Risk: Insufficient funds errors if balance changed
4. **Revert when AISP is back:**

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    AISP_MODE=live
  }"
```

Risk: Cached balance may be stale (up to 1 hour old). Users may attempt payments with insufficient funds.

Option 2: Hide Balance, Allow Payments

Use case: AISP down, no reliable cache, but PISP still works

Steps:

1. Show "Balance unavailable" message:

```
Saldo midlertidig utilgjengelig
Du kan fortsatt gjøre betalinger som normalt.
Banken vil avvise betalingen hvis du ikke har nok midler.
```

2. Allow payments without balance check:
   - User enters payment amount
   - Drop initiates payment via PISP
   - Bank performs real-time balance check
   - If insufficient funds: bank rejects, user gets clear error
3. Communicate ETA to users:

```
Vi jobber med å gjenopprette saldovisning.
Estimert tid: [X minutter]
```

Risk: User experience degraded. May attempt failed payments.

Post-Incident Actions

1. **Refresh all expired consents proactively:**

```
-- PostgreSQL 16: send renewal reminders 7 days before expiry
SELECT user_id, email, consent_expires_at
FROM bank_accounts
JOIN users ON users.id = bank_accounts.user_id
WHERE consent_expires_at < NOW() + INTERVAL '7 days'
AND consent_renewal_reminder_sent = FALSE;
```

2. **Document incident:**

```
touch ~/ALAI/products/Drop/comms/incidents/$(date +%Y-%m-%d)-aisp-failure.md
```

3. **Review caching strategy:**
   - Is cache TTL appropriate?
   - Should we cache balance longer during incidents?
   - Add metrics: cache hit rate, staleness
4. **Update monitoring:**
   - Add synthetic AISP test (fetch balance every 5 min)
   - Alert on AISP failure rate >10%
   - Track consent expiry dates
5. **Improve user communication:**
   - Auto-notify users when AISP is degraded
   - Show balance age: "Updated 5 minutes ago"

Escalation

Time	Action
0 min	John starts diagnosis
10 min	If Neonomics outage confirmed, notify Alem
20 min	If not resolved, enable cached balance mode
1 hour	Public communication to users (Norwegian email/push)
2 hours	Contact Neonomics support via phone if no response

Contacts

- **Neonomics Support:** support@neonomics.io
- **Neonomics Slack:** #neonomics-support (if available)
- **Internal:** Alem (CEO, final decision on fallback modes)

Related Documentation

- `docs/architecture/open-banking.md` — AISP flow diagrams
- `src/app/api/accounts/balance/route.ts` — Balance fetch implementation
- `docs/compliance/psd2-requirements.md` — PSD2 consent rules (90-day expiry)
- Vaultwarden item: "Neonomics API" — Credentials

Last Updated: 2026-02-22 **Next Review:** Before Phase 2 (Banking Integration)

Runbook: BankID Failure

Runbook: BankID Integration Failure

Service: BankID OAuth Authentication **Severity:** CRITICAL (blocks all logins) **MTTR Target:** <15 minutes **Owner:** John (AI Director)

Symptoms

Users report they cannot log in. Symptoms include:

- Login button doesn't redirect to BankID
- BankID redirect returns error page
- OAuth callback fails with 401/403
- Error message: "Authentication service unavailable"

Diagnosis

1. Check BankID Service Status

External status page:

```
# Check BankID status (no official status page, monitor Twitter)
open https://twitter.com/search?q=BankID%20Norge

# Or check community forums
open https://www.reddit.com/r/Norge/search?q=BankID
```

Quick test:

```
# Try BankID login from another service (e.g., tax portal)
open https://www.skatteetaten.no/person/
# If BankID works there but not in Drop → problem is our integration
```

2. Check Drop Logs

```
# CloudWatch Logs (production)
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "bankid" \
  --start-time $(date -u -d '10 minutes ago' +%s)000 \
  --region eu-west-1

# Look for:
# - "BankID OAuth error: invalid_client"
# - "BankID callback failed: invalid_state"
# - "BankID API timeout"
```

3. Check Environment Variables

```
# Verify BankID credentials are set
aws apprunner describe-service \
  --service-arn <ARN> \
  --region eu-west-1 \
  | jq
'.Service.SourceConfiguration.ImageRepository.ImageConfiguration.RuntimeEnvironmentVariables'
\
  | grep BANKID

# Expected:
# BANKID_CLIENT_ID: <client-id>
# BANKID_CLIENT_SECRET: <exists> (value hidden)
# BANKID_CALLBACK_URL: https://getdrop.no/api/auth/bankid/callback
```

4. Check OAuth Flow

Test OAuth initiation:

```
# Start OAuth flow
curl -X POST https://getdrop.no/api/auth/bankid/initiate \
  -H "Content-Type: application/json" \
  -d '{"redirectUrl": "/dashboard"}' \
  -v

# Expected: HTTP 302 redirect to BankID with state parameter
# If 500: Check BANKID_CLIENT_ID and BANKID_CALLBACK_URL
```

Test OAuth callback:

```
# Simulate callback (replace <code> and <state> with real values from BankID redirect)
curl -X GET "https://getdrop.no/api/auth/bankid/callback?code=<code>&state=<state>" \
  -v

# Expected: HTTP 302 redirect to /dashboard with auth cookie
# If 401: Check BANKID_CLIENT_SECRET
# If 400: Check state validation logic
```

Common Causes & Solutions

Cause 1: BankID Service Outage (External)

Probability: 5% (BankID is highly reliable)

Symptoms:

- All BankID logins fail across all services
- BankID status page reports incident
- Social media mentions BankID outage

Solution:

1. **Communicate:** Post status update to users

   ```
   Subject: Login temporarily unavailable
   Body: BankID authentication is experiencing issues.
         We're monitoring the situation and will restore service
         as soon as BankID is back online. Estimated: <X> minutes.
   ```

2. **Monitor:** Watch BankID Twitter/status for updates
3. **Fallback (if available):** If demo mode exists, consider temporary activation:

```
# Enable demo mode (ONLY in emergency, requires Alem approval)
aws apprunner update-service --service-arn <ARN> \
  --source-configuration "ImageRepository={...}" \
  --instance-configuration "EnvironmentVariables={NEXT_PUBLIC_SERVICE_MODE=demo}"
```

4. **Post-incident:** Document outage duration, user impact

ETA: Depends on BankID (typically <2 hours)

Cause 2: Invalid OAuth Credentials

Probability: 20% (after credential rotation or environment change)

Symptoms:

- Logs show: "invalid_client" or "unauthorized_client"
- OAuth flow fails immediately (no redirect to BankID)

Solution:

1. **Verify credentials in Vaultwarden:**

```
bw get item "BankID OAuth" --session $BW_SESSION
```

2. **Update App Runner environment variables:**

```
aws apprunner update-service --service-arn <ARN> \
  --source-configuration "ImageRepository={...}" \
  --instance-configuration "EnvironmentVariables={
    BANKID_CLIENT_ID=<correct-client-id>,
    BANKID_CLIENT_SECRET=<correct-secret>
  }"
```

3. **Trigger deployment:**

```
aws apprunner start-deployment --service-arn <ARN> --region eu-west-1
```

4. **Test:** Attempt login after deployment completes (3-5 minutes)

ETA: 10 minutes

Cause 3: Callback URL Mismatch

Probability: 15% (after domain change or deployment error)

Symptoms:

- Logs show: "redirect_uri_mismatch"
- BankID redirects to wrong URL (404 or CORS error)

Solution:

1. **Check registered callback URL in BankID portal:**
   - Login to BankID integration portal
   - Navigate to OAuth settings
   - Verify callback URL: `https://getdrop.no/api/auth/bankid/callback`
2. **If mismatch, update BankID portal:**
   - Change redirect URI to match current domain
   - Save changes (may require approval, 1-2 hours)
3. **Update App Runner env var:**

   ```
   aws apprunner update-service --service-arn <ARN> \
     --source-configuration "ImageRepository={...}" \
     --instance-configuration "EnvironmentVariables={
       BANKID_CALLBACK_URL=https://getdrop.no/api/auth/bankid/callback
     }"
   ```

4. **Test:** Login flow should work after both changes

ETA: 15 minutes (if no BankID approval required), 2 hours (if approval needed)

Cause 4: State Parameter Validation Failure

Probability: 10% (race condition or session timeout)

Symptoms:

- Logs show: "Invalid state parameter"
- User completes BankID flow but callback rejects

Solution:

1. **Check session storage:**
   - BankID state is stored in server session
   - If session expires before callback (>10 min), state is lost

2. **Increase session timeout (if needed):**

```
// src/lib/auth.ts
const SESSION_TIMEOUT = 15 * 60 * 1000; // 15 minutes (was 10)
```

3. **Clear stale sessions:**

```
# If using Redis for sessions
redis-cli FLUSHDB

# If using database sessions
sqlite3 drop.db "DELETE FROM sessions WHERE expires_at < datetime('now');"
```

4. **Ask user to retry:** State timeout is usually one-time issue

ETA: 5 minutes

Cause 5: BankID API Rate Limiting

Probability: 5% (during high-traffic events)

Symptoms:

- Logs show: "rate_limit_exceeded" or HTTP 429
- Intermittent failures (some users succeed, others fail)

Solution:

1. **Check rate limit headers in logs:**

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 0
X-RateLimit-Reset: 1640000000
```

2. **Wait for rate limit reset:** Typically resets every 60 seconds
3. **Implement exponential backoff (if not present):**

```
// src/lib/bankid-client.ts
async function callBankIDAPI(retries = 3) {
  try {
    return await fetch(url);
  } catch (error) {
    if (error.status === 429 && retries > 0) {
      await sleep(1000 * (4 - retries)); // 1s, 2s, 3s
```

```
      return callBankIDAPI(retries - 1);
    }
    throw error;
  }
}
```

4. **Contact BankID support:** If rate limits are too low for production traffic

ETA: 5 minutes (automatic), 1-2 days (if support ticket needed)

Cause 6: Network/Firewall Issues

Probability: 5% (AWS security group misconfiguration)

Symptoms:

- Logs show: "Connection timeout" or "ECONNREFUSED"
- BankID API requests never reach destination

Solution:

1. **Check outbound rules (App Runner → BankID):**

```
# App Runner egress is unrestricted by default
# Check VPC connector security group (if using VPC)
aws ec2 describe-security-groups --group-ids <vpc-connector-sg> --region eu-west-1
```

2. **Test connectivity from container:**

```
# Exec into running container (if possible)
curl -v https://oidc.bankid.no/.well-known/openid-configuration

# Expected: HTTP 200 with JSON response
# If timeout: Network/firewall issue
```

3. **Check DNS resolution:**

```
nslookup oidc.bankid.no
# Should resolve to BankID IP addresses
```

4. **Whitelist BankID IPs (if using strict firewall):**
   - Contact BankID for IP ranges
   - Add to AWS security group outbound rules

ETA: 15 minutes (if quick fix), 1 hour (if requires networking changes)

Emergency Workarounds

Option 1: Fallback to Demo Mode (Temporary)

Use case: BankID outage affects all users, estimated >1 hour downtime

Steps:

1. Enable demo mode:

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={NEXT_PUBLIC_SERVICE_MODE=demo}"
```

2. Communicate to users:

```
Subject: Temporary login method available
Body: Due to BankID outage, we've enabled demo login.
      Use email/password to access your account.
      BankID will be restored as soon as possible.
```

3. Monitor BankID status
4. **Revert to BankID when available:**

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={NEXT_PUBLIC_SERVICE_MODE=live}"
```

Risk: Demo mode may bypass KYC checks. Only use with Alem approval.

Option 2: Redirect to Status Page

Use case: BankID outage, no ETA, no fallback available

Steps:

1. Deploy maintenance page:

```
# Update health endpoint to return 503
# This triggers BetterStack alert + status page update
```

2. Show user-friendly message:

```
<h1>Login Temporarily Unavailable</h1>
<p>Our authentication provider (BankID) is experiencing issues.</p>
<p>We expect service to resume within <strong>X minutes</strong>.</p>
<p>Status updates: <a href="https://status.drop.no">status.drop.no</a></p>
```

3. Monitor and communicate updates every 30 minutes

Post-Incident Actions

1. **Document incident:**

```
# Create incident report
touch ~/ALAI/products/Drop/comms/incidents/$(date +%Y-%m-%d)-bankid-failure.md
```

2. **Root cause analysis:**
   - What triggered the failure?
   - Why didn't monitoring detect it sooner?
   - What prevented faster recovery?
3. **Update monitoring:**
   - Add synthetic BankID login test (every 5 min)
   - Alert on OAuth callback failures >5/min
4. **Update runbook:**
   - Add new failure mode if discovered
   - Improve diagnosis steps based on what worked
5. **Team debrief (if >30 min outage):**
   - Review timeline
   - Identify improvements
   - Update on-call procedures

Escalation

Time	Action
0 min	John starts diagnosis
5 min	If not resolved, alert Alem via Slack + SMS
15 min	If BankID outage confirmed, enable fallback (Alem approval)
30 min	If still unresolved, schedule team call

Time	Action
1 hour	If major outage, public communication via email/social media

---

Contacts

- **BankID Support:** support@bankid.no
- **BankID Phone:** +47 XXXX XXXX (24/7 for critical issues)
- **Internal:** Alem (CEO, final decision on fallback modes)

Related Documentation

- `docs/architecture/authentication.md` — BankID OAuth flow
- `src/app/api/auth/bankid/route.ts` — BankID integration code
- `docs/dr-runbook.md` — Infrastructure disaster recovery
- Vaultwarden item: "BankID OAuth" — Credentials

Last Updated: 2026-02-22 **Next Review:** Before Phase 2 (Banking Integration)

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)

Runbook: Sumsub KYC Failure

Runbook: Sumsub KYC/AML Verification Failure

Service: Sumsub Identity Verification (KYC/AML) **Severity:** HIGH (blocks new user registrations) **MTTR Target:** <30 minutes **Owner:** John (AI Director)

Overview

Sumsub provides automated identity verification (KYC - Know Your Customer) and AML (Anti-Money Laundering) checks for Drop. Required for regulatory compliance before users can make payments.

KYC Process:

1. User uploads ID document (passport, driver's license, national ID)
2. User takes selfie (liveness check)
3. Sumsub verifies document authenticity
4. Sumsub performs AML sanctions screening
5. Result: APPROVED, REJECTED, or MANUAL_REVIEW

Impact: If Sumsub fails, new users cannot complete registration. Existing users are unaffected.

Symptoms

Users report they cannot complete identity verification:

- ID upload fails with error
- Verification stuck at "Processing..." indefinitely
- Error message: "Verification service unavailable"
- Webhook never receives result from Sumsub
- User status stuck at "pending_kyc"

User impact: Cannot complete registration, cannot make payments.

Diagnosis

1. Check Sumsub Service Status

External status:

```
# Sumsub does not have a public status page
# Test via API health check
curl https://api.sumsub.com/resources/healthcheck \
  -H "X-App-Token: <app-token>" \
  -v

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

2. Check Drop Logs

```
# CloudWatch Logs (production)
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "sumsub" \
  --start-time $(date -u -d '30 minutes ago' +%s)000 \
  --region eu-west-1

# Look for:
# - "Sumsub API timeout"
# - "Sumsub webhook failed"
# - "KYC verification failed: document_expired"
# - "AML sanctions match: [name]"
```

3. Check Sumsub Dashboard

```
# Login to Sumsub Dashboard
open https://cockpit.sumsub.com

# Check:
```

```
# - Recent applicants (last 1 hour)
# - Failed verifications
# - Manual review queue length
# - Webhook delivery status
```

4. Check Webhook Delivery

Verify webhook endpoint is reachable:

```
# Sumsub sends webhooks to: https://getdrop.no/api/webhooks/sumsub
# Test endpoint manually
curl -X POST https://getdrop.no/api/webhooks/sumsub \
  -H "Content-Type: application/json" \
  -H "X-Sumsub-Signature: test" \
  -d '{"type":"applicantReviewed","reviewResult":{"reviewAnswer":"GREEN"}}' \
  -v

# Expected: HTTP 200
# If 404: Webhook endpoint not deployed
# If 401: Signature validation issue
```

5. Test KYC Flow

Manual test (staging):

```
# 1. Create test applicant
curl -X POST https://api.sumsub.com/resources/applicants \
  -H "X-App-Token: <sandbox-app-token>" \
  -H "Content-Type: application/json" \
  -d '{
    "externalUserId": "test-user-123",
    "levelName": "basic-kyc-level",
    "email": "test@example.com"
  }' \
  -v

# Expected: HTTP 201, applicant created
# If 400: Invalid request
# If 500: Sumsub API issue
```

Common Causes & Solutions

Cause 1: Sumsub API Outage (External)

Probability: 5% (Sumsub service disruption)

Symptoms:

- All KYC verifications fail
- Sumsub API health check returns 503
- Dashboard shows no recent applicants
- Logs show API timeouts

Solution:

1. **Verify outage:**

```
# Test Sumsub API from different networks
curl https://api.sumsub.com/resources/healthcheck \
  -H "X-App-Token: <app-token>" \
  -v

# If consistent failure: confirmed outage
```

2. **Contact Sumsub support:**
   - Email: support@sumsub.com
   - Live chat: https://cockpit.sumsub.com (bottom-right)
   - Phone: Check Sumsub Dashboard for support number
3. **Communicate to users (Norwegian):**

```
Emne: Identitetsverifisering midlertidig utilgjengelig

Hei,

Vi opplever for øyeblikket tekniske problemer med identitetsverifisering.
Du kan fortsette registreringen senere.

Vi forventer at tjenesten er tilbake innen [X minutter/timer].

Mvh,
```

```
Drop
```

4. **Queue pending verifications:**

```
-- Mark users as pending KYC retry
UPDATE users
SET kyc_status = 'pending_retry',
    kyc_retry_at = datetime('now', '+1 hour')
WHERE kyc_status = 'pending_kyc'
AND created_at > datetime('now', '-2 hours');
```

5. **Retry when Sumsub is back:**

```
# Cron job to retry pending KYC
node ~/ALAI/products/Drop/scripts/retry-kyc.js
```

ETA: Depends on Sumsub (typically <2 hours)

Cause 2: Document Verification Failure (User Error)

Probability: 40% (user uploads poor quality or invalid document)

Symptoms:

- Specific users fail KYC (not all users)
- Logs show: "document_not_readable", "document_expired", "document_type_mismatch"
- Sumsub dashboard shows rejection reason

Common rejection reasons:

- Blurry photo (document not readable)
- Expired document (passport/ID expired)
- Wrong document type (e.g., bank statement instead of ID)
- Photo cropped (missing corners/edges)
- Underage (user < 18 years old)

Solution:

1. **Identify rejection reason:**

```
SELECT user_id, kyc_rejection_reason, kyc_rejected_at
FROM users
WHERE kyc_status = 'rejected'
ORDER BY kyc_rejected_at DESC
LIMIT 10;
```

2. **Show clear error to user (Norwegian):**
   Blurry document:

```
Dokumentet er ikke leselig
Ta et nytt bilde i godt lys.
Sørg for at all tekst er skarp og leselig.
```

   Expired document:

```
Dokumentet er utløpt
Vennligst last opp et gyldig pass eller førerkort.
Dokumentet må være gyldig i minst 1 måned.
```

   Wrong document type:

```
Feil dokumenttype
Vi godtar kun: Pass, Nasjonalt ID-kort, Førerkort.
Bankkort og regninger godtas ikke.
```

   Underage:

```
Du må være 18 år eller eldre
Drop er kun tilgjengelig for brukere over 18 år.
```

3. **Allow user to retry:**
   - Show "Try Again" button in app
   - Provide tips for better photo quality
   - Link to FAQ: "How to take a good ID photo"
4. **Track retry success rate:**

```
-- How many users succeed on 2nd attempt?
SELECT
  COUNT(*) FILTER (WHERE kyc_attempt = 1 AND kyc_status = 'approved') as
first_attempt_success,
  COUNT(*) FILTER (WHERE kyc_attempt = 2 AND kyc_status = 'approved') as
second_attempt_success,
  COUNT(*) FILTER (WHERE kyc_attempt >= 3) as multiple_retries
FROM users;
```

ETA: Immediate (user must retry with better document)

Cause 3: AML Sanctions Match (Compliance Issue)

Probability: 3% (user flagged by sanctions screening)

Symptoms:

- Specific user's KYC fails with: "AML_SANCTIONS_MATCH"
- Sumsub dashboard shows "Red flag" or "Manual review required"
- User name matches sanctions list (OFAC, EU, UN, etc.)

Solution:

1. **Identify flagged users:**

```
SELECT user_id, email, full_name, kyc_rejection_reason
FROM users
WHERE kyc_rejection_reason LIKE '%sanctions%'
OR kyc_status = 'manual_review_aml';
```

2. **Review Sumsub dashboard:**
   - Login: https://cockpit.sumsub.com
   - Navigate to applicant
   - Check AML screening results
   - Review sanctions list match details
3. **False positive (common names):**
   - Example: "Ali Hassan" may match many sanctioned individuals
   - Sumsub shows match details (date of birth, nationality)
   - If clearly different person: manually approve in Sumsub
4. **True positive (actual sanctions match):**
   - **DO NOT approve.** This is a legal/regulatory issue.
   - Reject user registration immediately
   - Document incident for compliance records
5. **Notify user (if false positive, manually approved):**

```
Din identitetsverifisering er godkjent
Takk for tålmodigheten. Du kan nå bruke Drop.
```

6. **Notify user (if true positive, rejected):**

```
Vi kan dessverre ikke godkjenne din registrering
På grunn av regulatoriske krav kan vi ikke tilby tjenester til deg.
Ta kontakt med support@getdrop.no hvis du mener dette er en feil.
```

7. **Escalate to Alem if uncertain:**
   - AML compliance is critical
   - False rejection = bad UX, but false approval = legal risk
   - Alem makes final call on borderline cases

ETA: 10 minutes (false positive), N/A (true positive - reject)

Cause 4: Webhook Delivery Failure

Probability: 15% (Drop webhook endpoint down or unreachable)

Symptoms:

- Sumsub completes verification, but Drop never updates user status
- Logs show: "Webhook not received"
- Sumsub dashboard shows "Webhook delivery failed"
- User stuck at "pending_kyc" despite Sumsub showing "approved"

Solution:

1. **Check webhook endpoint health:**

```
# Test webhook endpoint
curl -X POST https://getdrop.no/api/webhooks/sumsub \
  -H "Content-Type: application/json" \
  -d '{"type":"ping"}' \
  -v

# Expected: HTTP 200
# If 404/500: Drop webhook endpoint broken
```

2. **Check Sumsub webhook delivery logs:**
   - Login: https://cockpit.sumsub.com
   - Navigate to Settings → Webhooks
   - Check recent delivery attempts
   - Look for: 404, 500, timeout errors
3. **Manually retry failed webhooks:**
   - Sumsub Dashboard → Applicant → "Resend Webhook"
   - This triggers new webhook delivery to Drop

- Verify Drop receives and processes it

4. **Fetch verification results via API (if webhook lost):**

```
# Manually fetch applicant status from Sumsub
curl -X GET https://api.sumsub.com/resources/applicants/<applicant-id>/status \
  -H "X-App-Token: <app-token>" \
  -v

# Parse result and update Drop database
```

5. **Update Drop database manually:**

```
UPDATE users
SET kyc_status = 'approved',
    kyc_approved_at = datetime('now')
WHERE sumsub_applicant_id = '<applicant-id>';
```

6. **Fix webhook endpoint (if broken):**
   - Check App Runner deployment status
   - Verify webhook route exists: `src/app/api/webhooks/sumsub/route.ts`
   - Check signature validation (Sumsub signs webhooks with HMAC)

ETA: 10 minutes (manual retry), 30 minutes (if endpoint fix needed)

Cause 5: Invalid or Expired API Credentials

Probability: 5% (after credential rotation)

Symptoms:

- Logs show: "401 Unauthorized" or "403 Forbidden"
- All Sumsub API calls fail
- Webhook signature validation fails

Solution:

1. **Verify Sumsub API credentials:**

```
bw get item "Sumsub API" --session $BW_SESSION

# Check:
# - App Token is correct
# - Secret Key is correct (for webhook signature)
```

```
# - Environment: production vs sandbox
```

2. **Regenerate API credentials (if needed):**
   - Login: https://cockpit.sumsub.com
   - Navigate to Settings → API
   - Generate new App Token + Secret Key
   - Copy to Vaultwarden
3. **Update App Runner environment variables:**

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    SUMSUB_APP_TOKEN=<new-app-token>,
    SUMSUB_SECRET_KEY=<new-secret-key>,
    SUMSUB_ENVIRONMENT=production
  }"
```

4. **Trigger deployment:**

```
aws apprunner start-deployment --service-arn <ARN> --region eu-west-1
```

5. **Test after deployment:**

```
# Try creating test applicant
curl -X POST https://getdrop.no/api/kyc/initiate \
  -H "Authorization: Bearer <test-user-token>" \
  -v

# Expected: HTTP 200, Sumsub applicant created
```

ETA: 10 minutes

Cause 6: Liveness Check Failure (Selfie)

Probability: 20% (user fails selfie/liveness verification)

Symptoms:

- Specific users fail at selfie stage
- Logs show: "liveness_check_failed", "face_mismatch"
- Sumsub dashboard shows "Selfie does not match ID photo"

Common reasons:

- Poor lighting (too dark, too bright)
- User wears sunglasses/hat
- Multiple people in frame
- Photo of a photo (not live person)
- Face does not match ID document

Solution:

1. **Show clear instructions before selfie (Norwegian):**

```
Slik tar du et godt selfie-bilde:
✓ God belysning (dagslys er best)
✓ Fjern briller/solbriller
✓ Se rett i kameraet
✓ Kun ditt ansikt i bildet
✗ Ikke bruk foto av foto
```

2. **Allow retry with better instructions:**

```
Selfie-verifisering mislyktes
Prøv igjen med bedre belysning.
Sørg for at ansiktet ditt er tydelig synlig.
```

3. **Improve liveness detection settings (if too strict):**
   - Login: https://cockpit.sumsub.com
   - Navigate to Settings → Verification Levels
   - Adjust liveness sensitivity (low/medium/high)
   - Balance: security vs user friction
4. **Manual review (if automated fails repeatedly):**
   - Some users may need manual review
   - Sumsub team reviews video/photos manually
   - ETA: 1-24 hours depending on Sumsub queue

ETA: Immediate (user retry), 1-24 hours (manual review)

Emergency Workarounds

Option 1: Manual KYC Review (Temporary)

Use case: Sumsub down >1 hour, urgent user needs verification

Steps:

1. Collect KYC documents manually:
   - Ask user to email ID photo + selfie to support@getdrop.no
   - Subject: "KYC Manual Review - [User ID]"
2. **Alem or John reviews manually:**
   - Verify ID document authenticity (check security features)
   - Compare selfie to ID photo
   - Check ID expiry date
   - Verify age >= 18
3. **Manual AML check:**
   - Search user name on: https://sanctionssearch.ofac.treas.gov
   - Check EU sanctions list: https://eeas.europa.eu/topics/sanctions-policy
   - Document findings
4. **Approve in database (if passes checks):**

   ```
   UPDATE users
   SET kyc_status = 'approved_manual',
       kyc_approved_at = datetime('now'),
       kyc_approved_by = 'john',
       kyc_notes = 'Manual review during Sumsub outage'
   WHERE user_id = '<user-id>';
   ```

5. **Notify user:**

   ```
   Din identitet er verifisert
   Velkommen til Drop! Du kan nå gjøre betalinger.
   ```

Risk: Manual review is slow, error-prone, not scalable. Only for critical cases.

Option 2: Delay Registration, Notify When Ready

Use case: Sumsub down, no ETA, non-urgent registrations

Steps:

1. Show maintenance message:

   ```
   Identitetsverifisering midlertidig utilgjengelig
   Vi jobber med å løse problemet.
   Du vil motta en e-post når du kan fortsette registreringen.
   ```

2. Collect user email:

```
// src/app/api/auth/register/route.ts
if (sumsubUnavailable) {
  await db.insert('pending_registrations', {
    email: userEmail,
    status: 'waiting_kyc',
    created_at: new Date(),
  });

  return {
    success: true,
    message: 'We will notify you when registration is available',
  };
}
```

3. **When Sumsub is back, notify users:**

   ```
   SELECT email FROM pending_registrations WHERE status = 'waiting_kyc';
   ```

   Email (Norwegian):

   ```
   Emne: Du kan nå fullføre registreringen i Drop

   Hei,

   Identitetsverifisering er tilbake.
   Klikk her for å fortsette registreringen: [Link]

   Mvh,
   Drop
   ```

ETA: Delayed registration (hours to days)

Monitoring & Alerts

Metrics to Track

- **KYC success rate:** Should be >85% (accounting for user errors)
- **KYC processing time:** p50 <5min, p95 <30min, p99 <2h (includes manual review)

- **Rejection reasons:** Track document_not_readable, expired, underage, sanctions separately

Alert Rules

```
// src/lib/kyc-monitor.ts
export async function trackKYCFailure(userId: string, reason: string) {
  const failureRate = await calculateKYCFailureRate('last_hour');

  if (failureRate > 0.3) { // 30% failure rate
    await sendAlert({
      severity: 'high',
      title: 'KYC failure rate high',
      message: `${(failureRate * 100).toFixed(1)}% of KYC attempts failing`,
      reason,
    });
  }
}
```

Post-Incident Actions

1. **Retry failed KYC verifications:**

   ```
   UPDATE users
   SET kyc_status = 'pending_retry',
       kyc_retry_at = datetime('now')
   WHERE kyc_status IN ('failed', 'pending_kyc')
   AND created_at > datetime('now', '-24 hours');
   ```

2. **Document incident:**

   ```
   touch ~/ALAI/products/Drop/comms/incidents/$(date +%Y-%m-%d)-sumsub-kyc-failure.md
   ```

3. **Review rejection reasons:**
   - High document_not_readable rate? Improve photo instructions
   - High liveness_check_failed rate? Adjust Sumsub settings
   - Track improvements in next month's KYC metrics
4. **Update user onboarding:**
   - Add better photo guides
   - Show example of good vs bad ID photos
   - Pre-flight check: "Is your ID expired?"

Escalation

Time	Action
0 min	John starts diagnosis
15 min	If Sumsub outage confirmed, notify Alem
30 min	If urgent user needs KYC, consider manual review (Alem approval)
1 hour	Public communication to users
2 hours	Contact Sumsub support via phone if no response

Contacts

- **Sumsub Support:** support@sumsub.com
- **Sumsub Live Chat:** https://cockpit.sumsub.com (bottom-right)
- **Sumsub Phone:** Check Sumsub Dashboard for support number
- **Internal:** Alem (CEO, manual KYC approval authority)

Related Documentation

- `docs/architecture/kyc-aml.md` — KYC/AML flow diagrams
- `src/app/api/kyc/initiate/route.ts` — Sumsub integration code
- `docs/compliance/kyc-requirements.md` — Regulatory requirements (age, ID types)
- Vaultwarden item: "Sumsub API" — Credentials

Last Updated: 2026-02-22 **Next Review:** Before Phase 2 (Banking Integration)

Runbook: Swan API Outage

Runbook: Swan BaaS API Outage

Service: Swan Banking-as-a-Service **Severity:** CRITICAL (blocks accounts, cards, payments if Swan is primary provider) **MTTR Target:** <15 minutes **Owner:** John (AI Director)

Overview

Swan provides core banking infrastructure for Drop. Depending on Drop's architecture phase, Swan may handle:

- **Account creation** (virtual IBAN accounts for users)
- **Card issuance** (virtual/physical debit cards)
- **Payment processing** (domestic/international transfers)
- **Balance management** (wallet balances, not Open Banking)

Impact: If Swan is the primary BaaS provider, an outage affects ALL core banking operations.

Symptoms

Users report critical failures:

- Cannot create new account
- Cannot view wallet balance (if using Swan wallets)
- Card payments fail or decline
- Error: "Banking service unavailable"
- Dashboard shows "System error" for account-related features

User impact: Complete inability to use banking features (depending on Drop's reliance on Swan).

Diagnosis

1. Check Swan Status Page

External status:

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

# Check for:
# - Incident reported
# - Degraded performance
# - Scheduled maintenance
```

2. Test Swan API

Health check:

```
# GraphQL health query
curl https://api.swan.io/graphql \
  -H "Authorization: Bearer <api-key>" \
  -H "Content-Type: application/json" \
  -d '{"query": "{viewer{id}}"}' \
  -v

# Expected: HTTP 200, {"data": {"viewer": {"id": "..."}}}
# If 500/503: Swan API down
# If 401: Credential issue
# If timeout: Network or Swan connectivity issue
```

Test account creation:

```
# Attempt to create test account
curl https://api.swan.io/graphql \
  -H "Authorization: Bearer <api-key>" \
  -H "Content-Type: application/json" \
  -d '{
    "query": "mutation { createAccount(input: {name: \"Test Account\"}) { id } }"
```

```
  }' \
  -v

# Expected: HTTP 200 with account ID
# If error: Check response for Swan error codes
```

3. Check Drop Logs

```
# CloudWatch Logs (production)
aws logs filter-log-events \
  --log-group-name /aws/apprunner/drop-production \
  --filter-pattern "swan" \
  --start-time $(date -u -d '15 minutes ago' +%s)000 \
  --region eu-west-1

# Look for:
# - "Swan API timeout"
# - "Swan GraphQL error: INTERNAL_SERVER_ERROR"
# - "Swan 503 Service Unavailable"
# - "Swan rate limit exceeded"
```

4. Check Swan API Credentials

```
# Verify Swan API key is valid
bw get item "Swan API" --session $BW_SESSION

# Check App Runner environment variables
aws apprunner describe-service \
  --service-arn <ARN> \
  --region eu-west-1 \
  | jq
'.Service.SourceConfiguration.ImageRepository.ImageConfiguration.RuntimeEnvironmentVariables'
\
  | grep SWAN

# Expected:
# SWAN_API_KEY: <exists>
# SWAN_ENVIRONMENT: production (or sandbox)
```

```
# SWAN_PARTNER_ID: <partner-id>
```

5. Check Recent Swan API Changes

Review Swan changelog:

```
# Swan may deprecate API endpoints or change schemas
# Check Swan developer portal for breaking changes
open https://docs.swan.io/changelog

# Review recent GraphQL schema changes
# Verify Drop uses supported API versions
```

Common Causes & Solutions

Cause 1: Swan Service Outage (External)

Probability: 5% (Swan is highly reliable, but incidents happen)

Symptoms:

- Swan status page reports incident
- All Swan API calls fail with 500/503
- No error in Drop code/config
- Social media mentions Swan issues

Solution:

1. **Verify outage scope:**
   - Check Swan status page
   - Test API from different networks (rule out local network issue)
   - Contact Swan support for ETA
2. **Communicate to users (Norwegian):**

```
Emne: Bankfunksjoner midlertidig utilgjengelig

Hei,

Vår bankinfrastruktur-leverandør (Swan) opplever tekniske problemer.
```

```
Dette påvirker:
- Kontoopprettelse
- Korttransaksjoner
- Overføringer

Vi overvåker situasjonen og forventer at tjenesten er tilbake innen [X
minutter/timer].

Mvh,
Drop
```

3. **Enable degraded mode:**

```
# Disable features that depend on Swan
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    FEATURE_ACCOUNTS=disabled,
    FEATURE_CARDS=disabled,
    SWAN_MODE=degraded
  }"

# Show maintenance banner in app
```

4. **Monitor Swan status:**
   - Subscribe to Swan status updates (RSS/email)
   - Check every 10 minutes for resolution
   - Test API as soon as Swan reports "Resolved"
5. **Re-enable features when Swan is back:**

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    FEATURE_ACCOUNTS=enabled,
    FEATURE_CARDS=enabled,
    SWAN_MODE=live
  }"
```

ETA: Depends on Swan (typically <2 hours for major incidents)

Cause 2: Invalid or Expired API Credentials

Probability: 15% (after credential rotation or Swan account changes)

Symptoms:

- Logs show: "401 Unauthorized" or "Forbidden"
- All Swan API requests fail immediately
- Swan API test returns authentication error

Solution:

1. **Verify Swan API credentials:**

```
bw get item "Swan API" --session $BW_SESSION


# Check:
# - API key is not expired
# - API key has correct permissions (accounts, cards, payments)
# - Partner ID is correct
```

2. **Regenerate API key (if needed):**
   - Login to Swan Dashboard: https://dashboard.swan.io
   - Navigate to Settings → API Keys
   - Revoke old key, generate new key
   - Copy new key to Vaultwarden
3. **Update App Runner environment variables:**

```
aws apprunner update-service --service-arn <ARN> \
  --source-configuration "ImageRepository={...}" \
  --instance-configuration "EnvironmentVariables={
    SWAN_API_KEY=<new-key>,
    SWAN_PARTNER_ID=<partner-id>
  }"
```

4. **Trigger deployment:**

```
aws apprunner start-deployment --service-arn <ARN> --region eu-west-1
```

5. **Test after deployment (3-5 min):**

```
curl https://getdrop.no/api/accounts/create \
  -H "Authorization: Bearer <test-user-token>" \
  -H "Content-Type: application/json" \
  -d '{"accountType": "personal"}' \
  -v


# Expected: HTTP 200, account created
```

ETA: 10 minutes

Cause 3: Swan API Rate Limiting

Probability: 10% (during high-traffic events or viral growth)

Symptoms:

- Logs show: HTTP 429 "Too Many Requests"
- Intermittent failures (some requests succeed, others fail)
- Rate limit headers in response

Solution:

1. **Check rate limit headers:**

```
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 \
  | jq -r '.events[].message' \
  | grep Swan
```

2. **Implement request queuing:**

```
// src/lib/swan-client.ts
import PQueue from 'p-queue';

const queue = new PQueue({
  concurrency: 5,     // Max 5 concurrent Swan requests
  interval: 1000,      // Per second
  intervalCap: 20      // Max 20 requests per second
});

export async function swanGraphQL(query: string, variables?: any) {
  return queue.add(() =>
    fetch('https://api.swan.io/graphql', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${process.env.SWAN_API_KEY}`,
        'Content-Type': 'application/json',
```

```
      },
      body: JSON.stringify({ query, variables }),
    })
  );
}
```

3. **Exponential backoff on retry:**

```
async function retrySwan(operation: () => Promise<any>, attempt = 1) {
  try {
    return await operation();
  } catch (error) {
    if (error.status === 429 && attempt <= 3) {
      const delay = 1000 * Math.pow(2, attempt); // 2s, 4s, 8s
      await sleep(delay);
      return retrySwan(operation, attempt + 1);
    }
    throw error;
  }
}
```

4. **Contact Swan to increase rate limit:**
   - Email Swan support with traffic stats
   - Provide justification: user growth, peak times
   - Request higher API quota

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

Cause 4: Swan GraphQL Schema Change (Breaking)

Probability: 5% (Swan updates API, breaks Drop integration)

Symptoms:

- Logs show: "GraphQL validation error"
- Specific queries fail: "Field 'X' doesn't exist on type 'Y'"
- Swan API works for some operations, fails for others

Solution:

1. **Check Swan changelog:**

```
# Review recent API changes
open https://docs.swan.io/changelog

# Look for:
# - Deprecated fields
# - Required fields added
# - Type changes
```

2. **Identify breaking changes:**

```
# Compare current Drop queries to Swan schema
# Example: account creation query
grep -r "createAccount" src/lib/swan-client.ts

# Cross-reference with Swan GraphQL schema
# https://api.swan.io/graphql (GraphQL Playground)
```

3. **Update Drop GraphQL queries:**

```
// Before (deprecated)
mutation {
  createAccount(input: { name: "User Account" }) {
    id
    balance  // □ Deprecated field
  }
}

// After (updated)
mutation {
  createAccount(input: { name: "User Account" }) {
    id
    balances {  // □ New field structure
      available
      currency
    }
  }
}
```

4. **Test updated queries:**

```
# Test in Swan GraphQL Playground first
# Then deploy to staging
# Verify all Swan-dependent features work
```

5. **Deploy fix:**

```
git add src/lib/swan-client.ts
git commit -m "Fix: Update Swan GraphQL queries to match latest schema"
git push origin main

# CI/CD triggers deployment
```

ETA: 30 minutes (if simple field change), 2 hours (if major refactor needed)

Cause 5: Network or Firewall Issues

Probability: 5% (AWS security group misconfiguration)

Symptoms:

- Logs show: "Connection timeout" or "ECONNREFUSED"
- Swan API requests never reach destination
- Works locally but fails in production

Solution:

1. **Check outbound connectivity:**

```
# App Runner egress is unrestricted by default
# If using VPC connector, check security group
aws ec2 describe-security-groups \
  --group-ids <vpc-connector-sg> \
  --region eu-west-1 \
  | jq '.SecurityGroups[].IpPermissionsEgress'
```

2. **Test DNS resolution:**

```
nslookup api.swan.io

# Should resolve to Swan IPs
# If NXDOMAIN: DNS issue
```

3. **Check AWS service health:**

```
# Check App Runner service events
aws apprunner list-operations \
  --service-arn <ARN> \
  --region eu-west-1 \
  | jq '.OperationSummaryList[0]'
```

4. **Whitelist Swan IPs (if strict firewall):**
   - Contact Swan for IP ranges
   - Add to security group outbound rules (port 443)

ETA: 15 minutes (if quick fix), 1 hour (if requires networking changes)

Cause 6: Swan Account Suspended or Payment Overdue

Probability: 2% (billing issue or compliance violation)

Symptoms:

- All Swan API calls fail with "Account suspended"
- Swan Dashboard shows billing alert
- Email from Swan about overdue payment or compliance issue

Solution:

1. **Check Swan Dashboard:**
   - Login: https://dashboard.swan.io
   - Look for alerts: billing, compliance, KYC
2. **Resolve billing issue:**
   - If overdue payment: pay immediately via Swan Dashboard
   - If billing method expired: update payment method
   - Contact Swan billing: billing@swan.io
3. **Resolve compliance issue:**
   - Swan requires KYC for partner accounts
   - Upload missing documents (company registration, director ID, etc.)
   - Respond to Swan compliance team ASAP
4. **Request urgent reactivation:**
   - Email Swan support: support@swan.io
   - Subject: "URGENT: Account reactivation needed - [Partner ID]"
   - Explain impact (users affected)
   - Provide evidence of issue resolution

ETA: 15 minutes (if billing), 24 hours (if compliance review needed)

Emergency Workarounds

Option 1: Degraded Mode (Disable Swan Features)

Use case: Swan down >30 minutes, no ETA, users need core app functionality

Steps:

1. Disable Swan-dependent features:

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    FEATURE_ACCOUNTS=disabled,
    FEATURE_CARDS=disabled,
    FEATURE_SWAN_WALLETS=disabled
  }"
```

2. Show banner in app:

```
⚠ Noen funksjoner er midlertidig utilgjengelige
Kontoopprettelse og korttransaksjoner er ikke tilgjengelig for øyeblikket.
Andre funksjoner virker som normalt.
```

3. Allow core features to work:
   - BankID login: (not Swan-dependent)
   - Open Banking balance: (uses Neonomics, not Swan)
   - PISP payments: (uses Neonomics, not Swan)
   - Swan accounts: (disabled)
4. **Re-enable when Swan is back:**

```
aws apprunner update-service --service-arn <ARN> \
  --instance-configuration "EnvironmentVariables={
    FEATURE_ACCOUNTS=enabled,
    FEATURE_CARDS=enabled,
    FEATURE_SWAN_WALLETS=enabled
  }"
```

Risk: Users cannot create accounts or use cards during outage.

Option 2: Queue Swan Operations for Later

Use case: Swan down, users need to create accounts but can wait

Steps:

1. Queue account creation requests:

```
// src/app/api/accounts/create/route.ts
export async function POST(request: Request) {
  const { accountType } = await request.json();

  try {
    return await swanClient.createAccount(accountType);
  } catch (error) {
    if (error.code === 'SWAN_UNAVAILABLE') {
      // Queue for later processing
      await db.insert('pending_accounts', {
        user_id: userId,
        account_type: accountType,
        status: 'queued',
        created_at: new Date(),
      });

      return {
        success: true,
        message: 'Account creation queued, will complete within 1 hour',
      };
    }
    throw error;
  }
}
```

2. Process queue when Swan is back:

```
# Run cron job to process pending accounts
node ~/ALAI/products/Drop/scripts/process-pending-accounts.js
```

3. Notify users when account is ready:

```
Din konto er klar!
Takk for tålmodigheten. Du kan nå bruke alle funksjoner i Drop.
```

Risk: Delayed user experience. Users may expect instant account creation.

Monitoring & Alerts

Metrics to Track

- **Swan API success rate:** Should be >99%
- **Swan API latency:** p50 <500ms, p95 <2s, p99 <5s
- **Swan error rate by operation:** Track createAccount, issueCard, makePayment separately

Alert Rules

```
// src/lib/swan-monitor.ts
export async function trackSwanFailure(operation: string, error: any) {
  const failureRate = await calculateSwanFailureRate('last_5_minutes');

  if (failureRate > 0.05) { // 5% failure rate
    await sendAlert({
      severity: 'critical',
      title: 'Swan API failure rate high',
      message: `${(failureRate * 100).toFixed(1)}% of Swan calls failing`,
      operation,
    });
  }
}
```

Post-Incident Actions

1. **Process queued operations:**

```
SELECT * FROM pending_accounts WHERE status = 'queued';
-- Retry all pending account creations
```

2. **Document incident:**

```
touch ~/ALAI/products/Drop/comms/incidents/$(date +%Y-%m-%d)-swan-outage.md
```

3. **Review SLA with Swan:**
   - Check if outage violated SLA
   - Request compensation/credits
   - Discuss failover options
4. **Improve resilience:**
   - Add Swan health check (every 5 min)
   - Implement circuit breaker for Swan API
   - Consider multi-provider strategy (backup BaaS)

Escalation

Time	Action
0 min	John starts diagnosis
5 min	If Swan status page shows incident, notify Alem
15 min	If not resolved, enable degraded mode
30 min	Contact Swan support via phone if no ETA
1 hour	Public communication to users

Contacts

- **Swan Support:** support@swan.io
- **Swan Phone:** +33 X XXXX XXXX (check Swan Dashboard for number)
- **Swan Status:** https://status.swan.io
- **Internal:** Alem (CEO, final decision on feature disabling)

Related Documentation

- `docs/architecture/banking.md` — Swan BaaS integration
- `src/lib/swan-client.ts` — Swan GraphQL client

- `docs/compliance/swan-requirements.md` — Swan partner KYC/compliance
- Vaultwarden item: "Swan API" — Credentials

Last Updated: 2026-02-22 **Next Review:** Before Phase 2 (Banking Integration)