Decision Log

Decision Log

Purpose

This document records all significant architectural, technical, and regulatory decisions made during Drop Srbija development. Each entry includes context, decision, rationale, and alternatives considered.

Format

Each decision follows this structure:

- **Date:** When decision was made
- **Decision ID:** Unique identifier (task number if applicable)
- **Title:** Brief description
- **Context:** Why this decision was needed
- **Decision:** What was decided
- **Rationale:** Why this option was chosen
- **Alternatives Considered:** What other options were evaluated
- **Consequences:** Trade-offs and implications
- **Status:** Active | Superseded | Deprecated

Decisions

D1: Bank Partner Adapter Pattern (Task 6)

Date: 2026-04-16
Decision ID: T6
Status: Active

Context:

Drop Srbija needs to integrate with NBS IPS (Narodna Banka Srbije Instant Payment System) to facilitate instant RSD transfers. NBS IPS uses ISO 20022 messaging over mTLS, not a REST API. Direct integration with NBS IPS requires a banking license or Payment Institution (PI) license from NBS.

Decision:

Implement a **bank partner adapter pattern** where Drop Srbija partners with a licensed Serbian bank that provides IPS gateway access. Drop operates as a **registered agent** under Article 24 of the Law on Payment Services.

Rationale:

1. **Faster time-to-market:** Agent registration takes 2-3 months vs 9-14 months for own PI license
2. **Lower upfront capital:** No EUR 125,000 capital requirement
3. **Reduced regulatory burden:** Bank partner handles NBS compliance, Drop focuses on customer experience
4. **Market validation:** Prove product-market fit before committing to full PI license

Alternatives Considered:

1. **Apply for PI license immediately:**
   - **Pros:** Full control, higher margins, regulatory independence
   - **Cons:** 9-14 month timeline, EUR 125,000 capital, extensive regulatory overhead
   - **Why rejected:** Too slow and capital-intensive for MVP phase
2. **Use third-party payment aggregator:**
   - **Pros:** Fastest integration (1-2 months)
   - **Cons:** High transaction fees (2-3%), less control over user experience, aggregator can change terms
   - **Why rejected:** Unsustainable economics, loss of differentiation
3. **Build without IPS (bank transfers only):**
   - **Pros:** Simpler integration, no licensing requirements
   - **Cons:** Transfers take 1-2 business days, poor user experience vs competitors
   - **Why rejected:** Instant payments are core value proposition

Consequences:

- ✅ Enables Year 1 launch with minimal regulatory overhead
- ✅ Drop can focus on product development, not licensing bureaucracy
- ❌ Bank partner takes 0.3-0.5% per transaction (reduces Drop's margin)
- ❌ Dependency on bank's IPS uptime and support quality
- ⚠ Transition to own PI license in Year 2 requires customer migration (account linking, re-KYC)

Related Documents:

- serbian-bank-partnership-pitch.md
- recommendation-year1-vs-year2.md

D2: Agent Model Year 1 ? Own PI License Year 2

Date: 2026-04-16
Decision ID: Finverge Recommendation
Status: Active

Context:

After evaluating regulatory pathways, Drop Srbija must choose between:

1. Permanent agent model under bank partnership
2. Temporary agent model, transition to own PI license after market validation

Decision:

Operate as **registered agent Year 1**, then apply for **own Payment Institution license Year 2** once product-market fit is proven (target: 10,000+ active users, RSD 100M+ monthly transaction volume).

Rationale:

1. **De-risk upfront investment:** Avoid EUR 125,000 capital requirement until revenue is proven
2. **Regulatory learning:** Understand NBS compliance requirements while under bank's supervision
3. **Flexibility:** Can pivot business model or market positioning in Year 1 without sunk regulatory costs
4. **Stronger PI application:** NBS PI license approval more likely if applicant has proven track record

Alternatives Considered:

1. **Permanent agent model:**
   - **Pros:** No licensing costs ever, bank handles compliance
   - **Cons:** Permanent 0.3-0.5% fee to bank, limited negotiation leverage, bank can terminate partnership
   - **Why rejected:** Long-term economics don't work; Drop needs own license for sustainability
2. **PI license from Day 1:**

- **Pros:** Full control, no bank dependency, better margins
- **Cons:** 9-14 month delay to launch, EUR 125,000 upfront, high regulatory burden before proving PMF
- **Why rejected:** Too risky to commit capital and time before validating demand

Consequences:

- □ Minimize upfront capital and regulatory risk
- □ Learn NBS compliance requirements in Year 1
- □ Must re-KYC customers during transition (unless bank agreement allows customer transfer)
- □ Must maintain dual integrations during migration period
- ⚠ Year 2 timeline depends on NBS application backlog (9-14 months from submission)

Trigger for Year 2 Transition:

Start PI license application when Drop Srbija achieves:

- 10,000+ monthly active users, AND
- RSD 100M+ monthly transaction volume, AND
- Profitability at agent model margins (revenue > bank fees + operating costs)

Related Documents:

- recommendation-year1-vs-year2.md
- nbs-pisp-license-requirements.md

D3: MTS Banka ? Banka Poštanska Štedionica (2021 Rebrand)

Date: 2026-04-16
Decision ID: Research Finding
Status: Historical Context

Context:

Early Drop Srbija research identified **MTS Banka** as a potential partner due to Telekom Srbija ownership and phone-to-IBAN synergy. However, MTS Banka ceased to exist as a standalone brand in 2021.

Decision:

Target **Banka Poštanska Štedionica** for partnership pitch, noting its legacy as MTS Banka and continued Telekom Srbija subsidiary status.

Rationale:

1. **Telekom Srbija ownership intact:** BPS is still a subsidiary of Telekom Srbija
2. **Phone-to-IBAN synergy preserved:** BPS has access to Telekom customer phone numbers for IBAN lookup
3. **Digital transformation appetite:** BPS is investing in digital banking (mobile app launched 2022)
4. **NBS IPS participant:** BPS is an active IPS participant

Alternatives Considered:

1. **Raiffeisen Banka:**
   - **Pros:** Largest bank by assets, strong digital banking, proven fintech partnerships
   - **Cons:** May have high partnership fees, slow decision-making (Austrian parent bureaucracy)
2. **Mobi Banka (digital-first bank):**
   - **Pros:** Agile, tech-forward, faster partnership negotiation
   - **Cons:** Smaller balance sheet, less brand trust, unknown IPS reliability

Consequences:

- □ Leverages existing Telekom Srbija phone database for phone-to-IBAN lookup
- □ Aligns with BPS digital transformation strategy
- □ BPS may lack fintech partnership experience compared to larger banks
- ⚠ BPS may require exclusivity clause (no partnerships with competing banks)

Related Documents:

- serbian-banks-api-landscape.md

D4: Idempotency via SHA-256 Hash Only (No Raw Key Storage)

Date: 2026-04-16
Decision ID: T4
Status: Active

Context:

To prevent duplicate transaction submissions (e.g., user double-clicks "Pay" button), Drop Srbija must implement idempotency. Standard approach is to store a unique `idempotency_key` (UUID or user-generated string) in the database with a unique constraint.

Decision:

Store only the **SHA-256 hash of the idempotency key**, not the raw key itself. Add `idempotency_key_hash` column to `transactions` table with unique constraint.

Rationale:

1. **Privacy by design:** Raw idempotency key might contain PII (e.g., client-side key format: `user123-2026-04-16-transfer`)
2. **Minimize PII surface area:** Hash is irreversible; even if database is breached, attacker cannot recover original key
3. **ZZPL compliance:** Storing hashes instead of raw data aligns with data minimization principle (Article 5)
4. **Functional equivalence:** Idempotency only requires uniqueness check, not key retrieval

Alternatives Considered:

1. **Store raw idempotency key:**
   - **Pros:** Easier debugging (can see original key in logs)
   - **Cons:** PII risk, ZZPL non-compliance if key contains user data
   - **Why rejected:** Privacy risk outweighs debugging convenience
2. **No idempotency (rely on client-side debouncing):**
   - **Pros:** Simpler backend
   - **Cons:** Client-side debouncing unreliable (network issues, multiple devices)
   - **Why rejected:** High risk of duplicate charges, poor user experience

Consequences:

- ✅ ZZPL-compliant, minimal PII storage
- ✅ Prevents duplicate transactions
- ❌ Cannot retrieve original idempotency key for debugging (hash is one-way)
- ⚠️ Client must send same idempotency key for retries (backend cannot generate it)

Implementation:

```
// In TransactionService.kt
val idempotencyKeyHash = MessageDigest.getInstance("SHA-256")
    .digest(request.idempotencyKey.toByteArray())
    .joinToString("") { "%02x".format(it) }

// Check for duplicate
val existingTx = Transactions.select {
    Transactions.idempotencyKeyHash eq idempotencyKeyHash
}.singleOrNull()

if (existingTx != null) {
```

```
    return existingTx  // Return existing transaction, do not create new one
}
```

Related Documents:

- Migration: `V4__transaction_idempotency.sql`

D5: OTP Lock Threshold: 5 Failed Attempts (Not 6)

Date: 2026-04-16
Decision ID: T2
Status: Active

Context:

Drop Srbija uses phone OTP for authentication. To prevent brute-force attacks, failed attempts must be limited. Some specs reference 6 attempts, industry standard is 5.

Decision:

Lock account (or require CAPTCHA) after **5 failed OTP attempts**, not 6.

Rationale:

1. **Industry standard:** Most banks and fintech apps use 5 attempts (Google, PayPal, Venmo, Cash App)
2. **Security vs UX balance:** 5 attempts is generous for legitimate users (allows for 4 typos), while preventing brute-force (6-digit OTP = 1M combinations, 5 attempts = 0.0005% success rate)
3. **Consistency:** ALAI's other products (Drop Norway, Bilko) use 5 attempts

Alternatives Considered:

1. **6 failed attempts:**
   - **Pros:** Slightly more user-friendly (one extra chance)
   - **Cons:** Marginal UX benefit, inconsistent with industry standard
   - **Why rejected:** 5 is already generous, no strong reason to deviate
2. **3 failed attempts:**
   - **Pros:** More secure (even harder to brute-force)
   - **Cons:** Too strict, frustrates legitimate users (e.g., mistyping OTP on small keyboard)
   - **Why rejected:** Poor UX, excessive security for low-risk authentication

Consequences:

- ✅ Aligns with industry best practices
- ✅ Prevents brute-force while allowing genuine user errors
- ✅ Users who fail 5 times must request new OTP (adds friction)
- ⚠ Support team must handle "locked out" users (reset mechanism needed)

Implementation:

```
// In PhoneOtpService.kt
if (verification.attempts >= 5) {
    throw IllegalStateException("Too many failed attempts. Request a new OTP.")
}
```

Related Documents:

- Test coverage: `PhoneOtpServiceTest.kt` (Task 20)

D6: No Separate "Pending" Table for Transactions

Date: 2026-04-16
Decision ID: Architecture Decision
Status: Active

Context:

Some payment systems use a two-table approach:

1. `pending_transactions` (temporary, deleted after settlement)
2. `completed_transactions` (permanent, for historical records)

Decision:

Use a **single `transactions` table** with a `status` column (`processing | completed | failed`).

Rationale:

1. **Simplicity:** One source of truth for all transactions
2. **Auditability:** Full transaction lifecycle in one table (easier for compliance audits)
3. **Query performance:** PostgreSQL handles 1M+ rows efficiently with indexes; no need for table partitioning at MVP scale
4. **Retry logic:** Easier to implement (update status, don't move between tables)

Alternatives Considered:

1. **Two-table approach (pending + completed):**
   - **Pros:** Smaller "hot" table (faster queries on active transactions)
   - **Cons:** Complexity (must move records between tables), harder to audit full transaction history
   - **Why rejected:** Premature optimization; PostgreSQL can handle Drop's Year 1 volume in single table
2. **Event sourcing (immutable transaction events):**
   - **Pros:** Full audit trail, supports complex state machines
   - **Cons:** Overkill for MVP, steep learning curve, slower reads (must replay events)
   - **Why rejected:** Too complex for current requirements; revisit if/when Drop builds advanced fraud detection

Consequences:

- □ Simpler codebase, easier debugging
- □ Full transaction history in one query
- □ Table grows indefinitely (must implement archival policy in future)
- ⚠ If volume exceeds 10M+ transactions, may need table partitioning (by month)

Related Documents:

- Schema: `V1__init.sql` (transactions table)

D7: Serbian Language UI (sr-RS Locale)

Date: 2026-04-16
Decision ID: Product Requirement
Status: Active

Context:

Drop Srbija targets Serbian market. UI language must be Serbian (sr-RS), but codebase must support internationalization (i18n) for potential future expansion (Bosnia, Croatia, Montenegro).

Decision:

- **Primary UI language:** Serbian (sr-RS)
- **Fallback language:** English (en-US) for error messages and developer logs
- **i18n library:** next-intl (Next.js 15 recommended)
- **Translation keys location:** `frontend/src/locales/sr.json`

Rationale:

1. **User trust:** Financial apps must be in local language (research shows 87% of Balkan users abandon apps not in their language)

2. **Regulatory requirement:** ZZPL and NBS mandate that user-facing documents (privacy policy, terms of service) are in Serbian
3. **Future-proof:** i18n structure allows adding Bosnian/Croatian/Montenegrin later (Serbo-Croatian mutual intelligibility)

Alternatives Considered:

1. **English-only:**
   - **Pros:** Easier development (no translation management)
   - **Cons:** Severely limits user adoption, non-compliant with ZZPL
   - **Why rejected:** Not viable for consumer fintech in Serbia
2. **Bilingual (Serbian + English):**
   - **Pros:** Appeals to expats and international users
   - **Cons:** Adds complexity, doubles translation effort, most users don't need it
   - **Why rejected:** Year 1 focus is local Serbian users; add English in Year 2 if data shows demand

Consequences:

- ✅ Maximizes user adoption in Serbian market
- ✅ ZZPL and NBS compliant
- ✅ All developers must work with Serbian text (use translation keys, not hardcoded strings)
- ⚠ Translation workflow needed (developer writes key → translator provides Serbian text)

Implementation:

```
// In app/layout.tsx
import { NextIntlClientProvider } from 'next-intl';
import sr from '@/locales/sr.json';

export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="sr">
      <body>
        <NextIntlClientProvider locale="sr" messages={sr}>
          {children}
        </NextIntlClientProvider>
      </body>
    </html>
  );
}
```

Translation key example:

```
// frontend/src/locales/sr.json
{
  "auth.otp.title": "Unesite OTP kod",
  "auth.otp.description": "Poslali smo vam 6-cifreni kod na {phone}",
  "auth.otp.submit": "Potvrdi",
  "auth.otp.resend": "Pošalji ponovo"
}
```

Related Documents:

- UI mockups: `frontend/src/app/(app)/page.tsx`

D8: PostgreSQL 16 Only (No SQLite)

Date: 2026-04-16
Decision ID: ALAI Standard (2026-03-17)
Status: Active

Context:

Some developers prefer SQLite for local development (simpler setup, no Docker required). However, PostgreSQL and SQLite have different SQL dialects and feature sets.

Decision:

Use **PostgreSQL 16 for both local development and production**. No SQLite.

Rationale:

1. **ALAI mandate:** All ALAI products use PostgreSQL (CEO decision 2026-03-17)
2. **Dev-prod parity:** Eliminates "works on my machine" issues due to SQL dialect differences
3. **Feature requirements:** Drop Srbija uses PostgreSQL-specific features (JSONB operators, `TIMESTAMP WITH TIME ZONE`, full-text search)
4. **Test reliability:** Integration tests use Testcontainers (real PostgreSQL in Docker), ensuring 100% parity

Alternatives Considered:

1. **SQLite for dev, PostgreSQL for production:**
   - **Pros:** Faster local setup (no Docker)
   - **Cons:** SQL dialect differences cause bugs, harder to debug production issues locally
   - **Why rejected:** ALAI standard prohibits SQLite; dev-prod parity is critical
2. **MySQL/MariaDB:**

- **Pros:** Widely supported, familiar to many developers
- **Cons:** Weaker JSON support, timezone handling issues, not ALAI standard
- **Why rejected:** PostgreSQL is ALAI standard; no reason to deviate

Consequences:

- □ 100% dev-prod parity
- □ Leverages PostgreSQL advanced features (JSONB, RLS, full-text search)
- □ Requires Docker for local development (adds setup step)
- ⚠ Developers must learn PostgreSQL-specific SQL (not transferable to SQLite/MySQL)

Related Documents:

- ALAI Tech Stack Standard
- Docker setup: `docker-compose.yml`

Future Decisions (Pending)

FD1: Merge Strategy for Linear Feature Branches

Status: Pending
Context: Drop Srbija uses linear feature branch model (feat/A → feat/B → feat/C). Need to decide merge strategy when first feature is production-ready.

Options:

1. **Squash and merge:** Collapse all commits into one
2. **Rebase and merge:** Keep commits, rewrite history to be linear
3. **Merge commit:** Preserve branch history

Recommendation: TBD (pending first production merge)

FD2: Multi-Bank Redundancy Strategy

Status: Pending
Context: Year 1 relies on single bank partner for IPS access. If bank's IPS gateway goes down, Drop is offline.

Options:

1. **Partner with 2+ banks:** Automatic failover if primary bank's IPS is down
2. **Pre-queue transactions:** Buffer locally during outages, process when bank is back
3. **Accept single point of failure Year 1:** Add redundancy in Year 2 after own PI license

Recommendation: TBD (pending first bank partnership negotiations)

Change Log

Date	Decision ID	Change
2026-04-16	D1-D8	Initial decision log created

Last Updated: 2026-04-16
Next Review: After first major architectural change or regulatory update

---

Revision #2
Created 2026-04-16 22:35:16 UTC by John
Updated 2026-05-31 20:06:07 UTC by John