ADR-009: Feature Flag System

ADR-009: Custom Feature Flag System

Status: Accepted **Date:** 2026-02-21 **Deciders:** John (AI Director) **Category:** Backend

---

Context

Drop needs feature flags for several reasons:

1. **Gradual rollout:** Cards feature requires a card issuing partner before activation -- must be gated
2. **Kill switches:** Ability to disable features instantly in production if compliance or operational issues arise
3. **Development:** Feature-in-progress code can be merged to main without being user-visible
4. **A/B testing:** Future capability for comparing payment flows

Feature flag approaches considered:

Approach	Cost	Complexity	Server+Client	Targeting	Audit
Custom (env vars)	Free	Low	Yes (NEXT_PUBLIC_)	No	Via deploy history
LaunchDarkly	$10/seat/mo	Medium	Yes	Yes (per-user)	Yes
Unleash (self-hosted)	Free (OSS)	High (infra)	Yes	Yes	Yes
ConfigCat	Free tier	Low	Yes	Yes	Yes

At Drop's current stage (pre-launch, no production users), a custom system based on environment variables provides exactly what is needed: server+client flag availability, zero operational overhead, and type-safe TypeScript integration. User-level targeting is not needed until there are users to target.

Decision

Implement a custom feature flag system using `NEXT_PUBLIC_FF_*` environment variables, with type-safe TypeScript wrappers for server and client access.

Architecture:

```
graph TB
    subgraph env["Environment Variables"]

vars["NEXT_PUBLIC_FF_VIRTUAL_CARDS=false<br/>NEXT_PUBLIC_FF_PHYSICAL_CARDS=false<br/>NEXT_PUBLIC_FF_NOTIFICATIONS=true<br/>..."]
    end

    subgraph server["Server-Side (API Routes)"]
        isEnabled["isEnabled('virtualCards')"]
        featureGate["featureGate('physicalCards')"]
        getAllFlags["getAllFlags()"]
    end

    subgraph client["Client-Side (React)"]
        useFlag["useFeatureFlag('notifications')"]
        useFlags["useFeatureFlags()"]
    end

    vars -->|"Build-time inline<br/>(NEXT_PUBLIC_ prefix)"| server
    vars -->|"Build-time inline<br/>(NEXT_PUBLIC_ prefix)"| client

    featureGate -->|"Returns 404<br/>if disabled"| api_route["API Route<br/>(e.g., POST /api/cards/[id]/physical)"]
```

Flag registry (`feature-flags.ts:27-36`):

Flag	Env Var	Default	Purpose
`virtualCards`	`NEXT_PUBLIC_FF_VIRTUAL_CARDS`	`false`	Virtual card issuance
`physicalCards`	`NEXT_PUBLIC_FF_PHYSICAL_CARDS`	`false`	Physical card ordering
`cardDetails`	`NEXT_PUBLIC_FF_CARD_DETAILS`	`false`	Card detail view
`cardFreeze`	`NEXT_PUBLIC_FF_CARD_FREEZE`	`false`	Card freeze/unfreeze

Flag	Env Var	Default	Purpose
`cardPin`	`NEXT_PUBLIC_FF_CARD_PIN`	`false`	Card PIN management
`spendingLimits`	`NEXT_PUBLIC_FF_SPENDING_LIMITS`	`false`	Spending limit controls
`notifications`	`NEXT_PUBLIC_FF_NOTIFICATIONS`	`true`	Push notifications
`merchantDashboard`	`NEXT_PUBLIC_FF_MERCHANT_DASHBOARD`	`true`	Merchant dashboard

Server-side API route protection via `featureGate()`:

```
const gate = featureGate("physicalCards");
if (gate) return gate;  // Returns 404: "Feature not available"
```

Client-side conditional rendering via `useFeatureFlag()`:

```
const cardsEnabled = useFeatureFlag("virtualCards");
if (!cardsEnabled) return null;
```

Consequences

Positive

- Zero infrastructure cost and operational overhead
- Type-safe TypeScript API prevents flag name typos at compile time
- Works on both server (API routes) and client (React hooks) via `NEXT_PUBLIC_` prefix
- `featureGate()` provides consistent 404 behavior for disabled API endpoints
- Flags are immutable per deployment (changed via environment variable update + redeploy)
- All card-related features safely gated while awaiting card issuing partner

Negative

- No per-user targeting (all users see the same flags)
- Flag changes require redeployment (not runtime-configurable)
- No built-in audit trail of flag changes (relies on deployment history)
- No gradual percentage-based rollout capability
- `NEXT_PUBLIC_` prefix exposes flag names to client (but values are public anyway)

Risks

- **Stale flags:** Flags left enabled/disabled long after they should be changed. Mitigation: feature tracking system (`features.ts`) monitors implementation status; quarterly flag cleanup reviews.
- **Build-time lock-in:** Flags are inlined at build time, so the same build cannot have different flag values. Mitigation: acceptable for current deployment model (one build per environment).

References

- Feature Flags Documentation -- Full API reference and flag listing
- API Reference -- Routes using `featureGate()`
- Security Architecture -- Feature flags section
- ADR-005: Monolith First -- Single deployment model

Revision #4
Created 2026-02-21 05:59:00 UTC by John
Updated 2026-05-23 10:56:49 UTC by John