State Management Architecture

State Management Architecture

> **Project:** Drop — Fintech Payment App **Version:** 0.1.0 **Date:** 2026-02-23
> **Author:** John (AI Director, ALAI) **Status:** In Review **Reviewers:** Alem Bašić (CEO)

Document History

Version	Date	Author	Changes
0.1	2026-02-23	John	Initial draft from source code analysis

---

1. State Architecture Overview

Drop uses a deliberately simple, lightweight state management approach. There are no global state libraries (Redux, Zustand, Jotai, etc.). All state is local component state.

Layer	Mechanism	Scope
Auth / user state	`useAuth()` custom hook	Per-component (fetches `/api/auth/me` on each mount)
Feature flags	`useFeatureFlag()` hook	Per-component (reads `NEXT_PUBLIC_FF_*` env vars)
Page / API data	`useState` + `useEffect` fetch	Component-local
Form state	`useState` per field	Component-local
UI state (modals, tabs, steps)	`useState`	Component-local
Navigation	Next.js `useRouter` / `usePathname`	Framework-provided
Auth token (web)	httpOnly cookie	Server-set, never in JS

Layer	Mechanism	Scope
Auth token (mobile)	Bearer token in-memory (`let token`) + `AsyncStorage`	Module-level in `lib/api.js`

Guiding principle: Keep it simple. No global state = no accidental shared state bugs. API data is fetched fresh on each page mount. User authentication is the only cross-cutting concern, handled by the `useAuth()` hook.

---

2. Data Flow Diagram

```
flowchart TD
    User["User Interaction"] --> Component["Component"]

    Component -->|"Auth check (every mount)"| AuthHook["useAuth() Hook\nGET /api/auth/me"]
    Component -->|"Data fetch (useEffect)"| LocalState["useState + useEffect\nLocal component state"]
    Component -->|"Form input"| FormState["useState per field\nComponent-local"]
    Component -->|"Navigate"| Router["Next.js Router\nuseRouter / usePathname"]
    Component -->|"Feature check"| FlagHook["useFeatureFlag()\nReads NEXT_PUBLIC_FF_*"]

    AuthHook -->|"cookie auth"| BFF["BFF API Routes\n/api/auth/me"]
    LocalState -->|"fetch with credentials"| BFF
    BFF -->|"User object + data"| Component

    FlagHook -->|"env var read"| EnvVars["process.env\nNEXT_PUBLIC_FF_*"]
```

---

3. State Categories

3.1 Auth State — `useAuth()` Hook

File: `src/lib/use-auth.ts`

Interface:

```
function useAuth(redirectIfUnauthenticated?: boolean): {
  user: User | null;
```

```
  loading: boolean;
  logout: () => Promise<void>;
  refreshUser: () => Promise<void>;
}
// Default: redirectIfUnauthenticated = true
```

User Model:

```
interface User {
  id: string;
  email: string;
  firstName: string;
  lastName: string;
  totalBalance: number;
  bankAccounts: BankAccount[];
  kycStatus: string;
}

interface BankAccount {
  id: string;
  bankName: string;
  accountNumber: string;
  balance: number;
  currency: string;
  isPrimary: boolean;
}
```

Behavior:

1. On mount: fetches `GET /api/auth/me` with `credentials: "include"` (httpOnly cookie auth)
2. If 401 and `redirectIfUnauthenticated` is true: redirects to `/login`
3. `logout()`: calls `POST /api/auth/logout`, clears cookie server-side, redirects to `/login`
4. `refreshUser()`: re-fetches `/api/auth/me` to update user state

Auth flow:

```
Login page → POST /api/auth/login → httpOnly cookie set → router.push("/dashboard")
Dashboard → useAuth() → GET /api/auth/me → User object
Logout → POST /api/auth/logout → cookie cleared → redirect /login
```

Usage patterns:

```
// Standard protected page (redirects if unauthenticated)
const { user, loading } = useAuth();
if (loading) return <Skeleton />;
// user is guaranteed non-null after loading

// Page that checks auth without redirect
const { user } = useAuth(false);
```

Pages using this hook (data source):

- `accounts/page.tsx` — reads `user.bankAccounts` (no separate fetch)
- `profile/page.tsx` — reads `user.firstName`, `user.lastName`, `user.email`

3.2 Feature Flags — `useFeatureFlag()` Hook

File: `src/lib/feature-flags.ts`

Available Flags:

Flag Name	Default	Used In
`virtualCards`	`false`	cards page (gate — shows "not available" if false)
`physicalCards`	`false`	cards page (order physical card option)
`cardDetails`	`false`	cards page (show full card details)
`cardFreeze`	`false`	cards page (freeze/unfreeze)
`cardPin`	`false`	cards page (change PIN)
`spendingLimits`	`false`	cards page (spending limits)
`notifications`	`true`	notification features
`merchantDashboard`	`true`	merchant page (gate)

Environment Variable Pattern:

```
NEXT_PUBLIC_FF_VIRTUAL_CARDS=true
NEXT_PUBLIC_FF_PHYSICAL_CARDS=false
NEXT_PUBLIC_FF_NOTIFICATIONS=true
NEXT_PUBLIC_FF_MERCHANT_DASHBOARD=true
```

Convention: `NEXT_PUBLIC_FF_` + `SCREAMING_SNAKE_CASE` version of flag name.

API:

```
// Server-side (API routes / middleware)
isEnabled(flagName: string): boolean
getAllFlags(): Record<string, boolean>
featureGate(flagName: string): middleware  // Returns 404 if flag disabled

// Client-side (React hooks)
useFeatureFlag(flagName: string): boolean
useFeatureFlags(): Record<string, boolean>
```

Usage pattern:

```
// Page-level gate
const cardsEnabled = useFeatureFlag("virtualCards");
if (!cardsEnabled) return <div>Feature ikke tilgjengelig</div>;

// Conditional rendering
const physicalEnabled = useFeatureFlag("physicalCards");
{physicalEnabled && <OrderPhysicalCard />}
```

3.3 Page Data — `useState` + `useEffect` Fetch (Pattern 1)

Most pages fetch data in a `useEffect` on mount. No SWR, React Query, or other caching library is used.

```
const [data, setData] = useState([]);
const [loading, setLoading] = useState(true);

useEffect(() => {
  fetch("/api/endpoint", { credentials: "include" })
    .then(res => res.json())
    .then(json => setData(json.data))
    .catch(() => {})  // Silent error — empty state
    .finally(() => setLoading(false));
}, []);
```

Pages using this pattern:

- `dashboard/page.tsx` — fetches `GET /api/transactions?limit=10`
- `transactions/page.tsx` — fetches `GET /api/transactions?type={filter}&limit=50`
- `send/page.tsx` — fetches `GET /api/recipients` and `GET /api/rates`
- `notifications/page.tsx` — fetches `GET /api/notifications`
- `profile/notifications/page.tsx` — fetches `GET /api/settings`
- `profile/language/page.tsx` — fetches `GET /api/settings`
- `merchant/page.tsx` — fetches `/api/merchants/dashboard`, `/api/merchants/transactions`, `/api/merchants/qr`
- `cards/page.tsx` — fetches `GET /api/cards` (feature-flagged)

3.4 Form State — `useState` Per Field (Pattern 3)

Form pages use async handlers that POST data and handle success/error states. No form library.

```
const [email, setEmail] = useState("");
const [password, setPassword] = useState("");
const [error, setError] = useState("");
const [loading, setLoading] = useState(false);

const handleSubmit = async () => {
  setLoading(true);
  setError("");
  const res = await fetch("/api/auth/login", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    credentials: "include",
    body: JSON.stringify({ email, password }),
  });
  if (res.ok) {
    router.push("/dashboard");
  } else {
    const data = await res.json();
    setError(data.message || "Noe gikk galt");
  }
  setLoading(false);
};
```

Pages using this pattern:

- `login/page.tsx` — POST `/api/auth/login`
- `register/page.tsx` — POST `/api/auth/register` (4-step: info, OTP, PIN, success)
- `send/page.tsx` — POST `/api/transactions/remittance`
- `scan/page.tsx` — POST `/api/transactions/qr-payment`
- `complaints/page.tsx` — POST `/api/complaints`
- `withdrawal/page.tsx` — POST (withdrawal request)
- `cards/page.tsx` — POST/PATCH/DELETE `/api/cards/*` (feature-flagged)

3.5 Filter-Driven Refetch (Pattern 4)

Transactions and notification pages refetch when filter changes via `useEffect` dependency.

```
const [filter, setFilter] = useState("all");

useEffect(() => {
  fetch(`/api/transactions?type=${filter}&limit=50`, { credentials: "include" })
    .then(res => res.json())
    .then(json => setData(json.transactions))
    .catch(() => {})
    .finally(() => setLoading(false));
}, [filter]);
```

Pages using this pattern:

- `transactions/page.tsx` — filters: "all", "remittance", "qr_payment"
- `notifications/page.tsx` — auto-marks read via PATCH on mount

---

4. API Routes (BFF Layer)

All API routes are under `src/app/api/`. The frontend calls these endpoints via the Next.js BFF (which translates httpOnly cookie auth to Bearer token for the Hono backend).

Endpoint	Method	Called From
`/api/auth/login`	POST	login page
`/api/auth/logout`	POST	useAuth hook, profile page
`/api/auth/me`	GET	useAuth hook (every protected page mount)
`/api/auth/register`	POST	register page

Endpoint	Method	Called From
`/api/transactions`	GET	dashboard, transactions pages
`/api/transactions/remittance`	POST	send page
`/api/transactions/qr-payment`	POST	scan page
`/api/recipients`	GET	send page
`/api/rates`	GET	send page
`/api/notifications`	GET	notifications page
`/api/notifications`	PATCH	notifications page (mark read)
`/api/settings`	GET	profile/notifications, profile/language
`/api/settings`	PATCH	profile/notifications, profile/language
`/api/complaints`	POST	complaints page
`/api/consents`	POST	cookie-consent component
`/api/cards`	GET/POST	cards page (feature-flagged)
`/api/cards/{id}`	PATCH	cards page (freeze/unfreeze, feature-flagged)
`/api/cards/{id}`	DELETE	cards page (delete card, feature-flagged)
`/api/merchants/dashboard`	GET	merchant page
`/api/merchants/transactions`	GET	merchant page
`/api/merchants/qr`	GET	merchant page

---

5. Mobile State Management

File: `src/drop-mobile/lib/api.js`

The mobile app uses a different auth mechanism — Bearer token instead of httpOnly cookie.

```
// Token stored at module level (in-memory)
let token = null;

export const setToken = (t) => { token = t; };
export const getToken = () => token;

// All requests include auth header
const request = async (endpoint, options = {}) => {
```

```
  const headers = {
    "Content-Type": "application/json",
    ...(token && { Authorization: `Bearer ${token}` }),
    ...options.headers,
  };
  // ...
};
```

Token persistence: On login, token stored in `AsyncStorage` for 7-day lifetime. On app launch, token loaded from `AsyncStorage` and set via `setToken()`.

Mobile state patterns:

- All screens use `useState` (same as web)
- Auth data read from `api.getMe()` on dashboard mount
- Pull-to-refresh: `RefreshControl` on `FlatList` components
- No global state library — same philosophy as web

---

6. Persistent State

Data	Storage	Notes
Auth session (web)	httpOnly cookie	Server-set, 7-day lifetime, never in JS
Auth token (mobile)	Bearer token, `AsyncStorage`	7-day lifetime, device-bound
Cookie preferences	`localStorage`	CookieConsent component
Language preference	Server-side (`/api/settings`)	Synced on login
Push notification prefs	Server-side (`/api/settings`)	Synced on page load

RULE: Auth tokens NEVER in localStorage. httpOnly cookies on web, `AsyncStorage` (module-level) on mobile.

---

7. State Debugging

Tool	Usage	Enabled In
React DevTools	Component state tree inspection	Dev only
`__DEV__` flag	Enables demo credentials on login	Dev only
Network tab	Inspect `/api/auth/me` response	Dev only

Tool	Usage	Enabled In
Console logs	API client logs (mobile)	Dev only

No state management devtools — not needed without a global store.

---

8. Performance Considerations

No Unnecessary Re-renders

- Each `useEffect` has explicit dependency arrays
- `useState` updates are batched by React 19
- No derived state that needs memoization in current implementation

Fresh Data on Navigation

- `useAuth()` re-fetches `/api/auth/me` on every page mount — ensures fresh user data
- Page data re-fetches on every mount — no stale cache issues
- Trade-off: more network requests, but always fresh data for financial app accuracy

Future Optimization Path (if needed)

1. Add `SWR` or `TanStack Query` for caching with stale-while-revalidate
2. Persist query cache across navigation (currently re-fetches on each visit)
3. Optimistic updates for settings toggles (partially implemented in notification settings)

Approval

Role	Name	Date	Signature
Author	John (AI Director)	2026-02-23	
Frontend Lead			
Tech Lead			

Revision #6
Created 2026-02-18 08:44:24 UTC by John
Updated 2026-05-31 20:02:11 UTC by John