# ADR-011: Expo Mobile Framework

# ADR-011: Expo SDK 54 for Mobile App

**Status:** Accepted **Date:** 2026-02-21 **Deciders:** John (AI Director), Alem (CEO) **Category:** Mobile

---

## Context

Drop requires a mobile app for iOS and Android. The mobile app is the primary interface for remittance and QR payments -- users scan QR codes with their phone camera and approve payments via BankID on the same device.

Mobile framework options considered:

| Framework | Cross-Platform | Code Sharing (Web) | OTA Updates | Camera/QR | BankID Integration | Dev Experience |
|---|---|---|---|---|---|---|
| **Expo SDK 54** | iOS + Android | High (React shared) | Yes (EAS Update) | expo-camera | expo-web-browser | Excellent |
| **React Native (bare)** | iOS + Android | High (React shared) | Manual | react-native-camera | Custom deep links | Good |
| **Flutter** | iOS + Android | None (Dart vs TS) | No native OTA | camera plugin | Custom deep links | Good |
| **Native (Swift/Kotlin)** | Separate codebases | None | App Store only | Native APIs | Native SDKs | Platform-specific |

Key factors in the decision:

1. **Code sharing:** Drop's web app uses React 19. Expo enables sharing React components, hooks, types, and business logic between web and mobile.

2. **BankID flow:** Mobile BankID authentication requires opening a secure browser (`expo-web-browser`) and handling deep link callbacks (`drop://auth/callback`). Expo provides both natively.
3. **QR scanning:** Core feature requires camera access. `expo-camera` provides this with barcode scanning built in.
4. **OTA updates:** Financial apps need rapid hotfix deployment. Expo Application Services (EAS) provides over-the-air JavaScript bundle updates without App Store review.
5. **Team capacity:** AI-driven development team benefits from a single language (TypeScript) across all platforms.

# Decision

**Use Expo SDK 54 with managed workflow for the Drop mobile app.**

```
graph TB
    subgraph mobile["Mobile App (Expo SDK 54)"]
        screens["Screens<br/>(10 screens matching web)"]
        hooks["Shared Hooks<br/>(useAuth, useTransactions)"]
        types["Shared TypeScript Types"]

        screens --> camera["expo-camera<br/>(QR scanning)"]
        screens --> browser["expo-web-browser<br/>(BankID auth)"]
        screens --> notif["expo-notifications<br/>(push alerts)"]
        screens --> storage["AsyncStorage<br/>(Bearer token)"]
        screens --> linking["expo-linking<br/>(deep links: drop://)"]
    end

    subgraph web["Web App (Next.js 15)"]
        web_screens["Screens<br/>(10 screens)"]
        web_hooks["Shared Hooks"]
        web_types["Shared TypeScript Types"]
    end

    subgraph backend["Backend"]
        hono["Hono v4 API<br/>(/v1/* - Bearer auth)"]
        nextjs["Next.js BFF<br/>(/api/* - Cookie auth)"]
    end

    hooks -.->|"Shared React code"| web_hooks
    types -.->|"Shared types"| web_types
```

```
    mobile --> hono
    web --> nextjs

    classDef expo fill:#E3F2FD,stroke:#1565C0
    classDef web_style fill:#C8E6C9,stroke:#2E7D32
    classDef backend_style fill:#FFF3E0,stroke:#E65100

    class screens,hooks,types,camera,browser,notif,storage,linking expo
    class web_screens,web_hooks,web_types web_style
    class hono,nextjs backend_style
```

# Key Expo Modules Used

| Module | Purpose | Drop Feature |
|---|---|---|
| `expo-camera` | Camera access + barcode scanning | QR payment scanning |
| `expo-web-browser` | Secure in-app browser | BankID OIDC authentication |
| `expo-notifications` | Push notification handling | Transaction alerts, payment receipts |
| `expo-linking` | Deep link handling (`drop://`) | BankID callback, notification deep links |
| `@react-native-async-storage` | Persistent key-value store | Bearer token storage |
| `expo-secure-store` | Encrypted storage | Sensitive data (future biometric) |
| `expo-local-authentication` | Biometric auth | App unlock (Phase 2) |

# Mobile-Specific Auth Flow

The mobile BankID flow differs from web:

1. `GET /v1/auth/bankid/initiate?platform=mobile` returns `{ redirectUrl, state }`
2. Open BankID in `expo-web-browser` (secure, isolated browser)
3. BankID redirects to `drop://auth/callback?code=&state=`
4. `expo-linking` catches the deep link
5. `POST /v1/auth/bankid/callback` exchanges code for Bearer token
6. Token stored in `AsyncStorage` (7-day lifetime)

# Consequences

## Positive

- Single language (TypeScript) across web, mobile, and backend
- High code reuse: shared types, hooks, and validation logic with web app
- OTA updates via EAS enable rapid hotfixes without App Store review cycle
- Managed workflow eliminates native build complexity
- `expo-camera` provides built-in barcode scanning for QR payments
- `expo-web-browser` provides secure BankID integration
- Large React Native ecosystem for additional modules

# Negative

- Expo managed workflow limits access to some native APIs (can eject if needed)
- App size larger than pure native (~25MB vs ~5MB)
- JavaScript bridge performance for heavy computation (not a concern for Drop's use case)
- Must use Expo-compatible packages (some React Native packages require ejection)
- EAS build service adds to CI costs

# Risks

- **Expo SDK upgrade breakage:** Major Expo SDK upgrades can break packages. Mitigation: managed workflow handles most upgrades; test thoroughly before upgrading.
- **App Store rejection:** Financial apps face stricter App Store review. Mitigation: ensure compliance with App Store Review Guidelines section 3.1 (payments) and 5.1 (privacy).
- **Performance on low-end devices:** React Native may lag on older Android devices. Mitigation: minimal animations, lazy loading, optimized list rendering.

# References

- ADR-008: Hono API Framework -- Mobile API backend
- Authentication System -- Mobile BankID flow
- Component Overview (C4 Level 3) -- Mobile app components
- ADR-007: BankID OIDC Auth -- Authentication provider
- Expo documentation: docs.expo.dev

---

Revision #5
Created 2026-02-21 05:59:01 UTC by John
Updated 2026-05-23 10:56:54 UTC by John