

Mobile App

Mobile app architecture and strategy

- [Mobile Architecture](#)
 - [Mobile App Architecture](#)
 - [Mobile Strategy](#)

Mobile Architecture

Mobile App Architecture

Drop Mobile App

“ React Native / Expo Router app in `src/drop-mobile/` .

Tech Stack

Technology	Version/Details
Framework	React Native (Expo)
Routing	Expo Router (file-based)
Fonts	DM Sans (expo-google-fonts), Fraunces (expo-google-fonts)
Navigation	Stack (root) + Tabs (main app)
State	React <code>useState</code> (no global state library)
API	Custom fetch wrapper with Bearer token auth
Storage	AsyncStorage (for auth token)

Directory Structure

```
src/drop-mobile/  
├─ App.js           # Expo entry (unused – Expo Router takes over)  
├─ app/  
│  ├─ _layout.js   # Root Stack layout + font loading  
│  ├─ index.js     # Welcome screen  
│  ├─ login.js     # Login screen  
│  ├─ register.js  # Registration (2-step)  
│  └─ history.js   # Transaction history
```

```

|   └─ (tabs)/
|     └─ _layout.js      # Tab navigator (4 tabs)
|     └─ index.js       # Dashboard / Home
|     └─ send.js        # Send money
|     └─ scan.js        # QR scanner
|     └─ profile.js     # Profile & settings
└─ lib/
  └─ api.js             # API client
  └─ theme.js          # Theme constants
└─ assets/             # Static assets

```

Navigation

Root Stack (`app/_layout.js`)

```

Stack.Screen: index      (Welcome – no header)
Stack.Screen: login      (Login – no header)
Stack.Screen: register   (Register – no header)
Stack.Screen: (tabs)     (Main app – no header)
Stack.Screen: send       (Send money – modal presentation)
Stack.Screen: scan       (QR scan – modal presentation)

```

Font loading happens here: `Fraunces_600SemiBold`, `Fraunces_700Bold`, `DMSans_400Regular`, `DMSans_500Medium`, `DMSans_700Bold`. App shows nothing until fonts are loaded (`SplashScreen.preventAutoHideAsync`).

Tab Navigator (`app/(tabs)/_layout.js`)

Tab	Label	Icon	Screen
index	Hjem	Unicode house	Dashboard
send	Send	Unicode arrow	Send money
scan	QR	Unicode QR	QR scanner
profile	Profil	Unicode person	Profile

Tab bar style: `backgroundColor: colors.white`, `borderTopColor: colors.border`, `height: 60`. Active tint: `colors.green` (`#0B6E35`), inactive: `colors.textLight` (`#9CA3AF`).

Screens

Welcome (`app/index.js`)

- Green background (`colors.green`)
- "drop." wordmark in white Fraunces font
- Headline: "Enklere betalinger. Lavere gebyrer."
- Two buttons: "Logg inn" (outline) → `/login`, "Opprett konto" (solid white) → `/register`

Login (`app/login.js`)

- White background
- Email + password fields
- "Logg inn" green button → calls `api.login(email, password)`
- On success: stores token, navigates to `/(tabs)`
- "Opprett konto" link → `/register`
- Pre-filled demo credentials in dev

Register (`app/register.js`)

- 2-step flow with progress dots indicator
- **Step 1:** firstName, lastName, email, phone
- **Step 2:** password, confirmPassword
- Calls `api.register(data)`, on success navigates to `/(tabs)`

Dashboard (`app/(tabs)/index.js`)

- Greeting: "Hei, {firstName}" with hand wave emoji
- Green balance card: "Total saldo" with formatted NOK amount
- Quick stats row: Sendt, Mottatt, Ventende (sent, received, pending)
- "Siste transaksjoner" section with FlatList
- Each transaction: icon (arrow up=sent, arrow down=received), name, date, amount with color coding
- "Se alle" link → `/history`
- Pull-to-refresh via `RefreshControl`

Send Money (`app/(tabs)/send.js`)

- 2-step flow:
- **Step 1:** Recipient name input + currency picker (5 currencies with flags)
 - Currencies: BAM (Bosnia), RSD (Serbia), PKR (Pakistan), TRY (Turkey), PLN (Poland)
- **Step 2:** Amount input (NOK) + conversion card showing:
 - Exchange rate (fetched from `api.getRate()`)
 - Fee (0.5%)
 - "Mottaker får" (recipient gets) calculated amount
- "Bekreft og send" button → `api.sendRemittance(data)`
- Success: shows confirmation, "Tilbake til hjem" button

QR Scanner (`app/(tabs)/scan.js`)

- Camera placeholder (gray box with QR icon)
- "Skann QR-kode" instruction text
- "Simuler skanning" button for demo
- Nearby merchants list (hardcoded: Ahmetov Kebab, Kafe Oslo, Narvesen)
- Payment flow: merchant info → amount input → confirm → success
- Calls `api.payQR({ merchantId, amount })`

Transaction History (`app/history.js`)

- Filter tabs: Alle, Sender (remittance), QR (qr_payment)
- FlatList with transaction items
- Each item: direction icon, recipient/merchant name, date, amount
- Pull-to-refresh
- Filter changes trigger re-fetch via `api.getTransactions({ type })`

Profile (`app/(tabs)/profile.js`)

- User info section: initials avatar, name, email
- "Mine mottakere" section with recipients list (fetched from `api.getRecipients()`)
- Each recipient: name, country, account number
- Settings menu: Språk, Varsler, Personvern, Vilkar
- Logout button → clears token, navigates to /index

API Client

File: `lib/api.js`

Configuration

```
const API_URL = __DEV__
  ? "http://localhost:3000/api"
  : "https://drop-app.vercel.app/api";
```

Auth Pattern

- Token stored in module-level variable `let token = null`
- All requests include `Authorization: Bearer ${token}` header
- `setToken(t)` / `getToken()` exported for auth management

Available Methods

Method	HTTP	Endpoint	Returns
<code>api.login(email, password)</code>	POST	<code>/auth/login</code>	<code>{ token, user }</code>
<code>api.register(data)</code>	POST	<code>/auth/register</code>	<code>{ token, user }</code>
<code>api.logout()</code>	POST	<code>/auth/logout</code>	—
<code>api.getMe()</code>	GET	<code>/auth/me</code>	<code>{ user }</code>
<code>api.getTransactions(params)</code>	GET	<code>/transactions</code>	<code>{ transactions }</code>
<code>api.sendRemittance(data)</code>	POST	<code>/transactions/remittance</code>	<code>{ transaction }</code>
<code>api.payQR(data)</code>	POST	<code>/transactions/qr-payment</code>	<code>{ transaction }</code>
<code>api.getRecipients()</code>	GET	<code>/recipients</code>	<code>{ recipients }</code>
<code>api.addRecipient(data)</code>	POST	<code>/recipients</code>	<code>{ recipient }</code>
<code>api.getMerchants()</code>	GET	<code>/merchants</code>	<code>{ merchants }</code>
<code>api.getRates()</code>	GET	<code>/rates</code>	<code>{ rates }</code>
<code>api.getRate(currency)</code>	GET	<code>/rates/{currency}</code>	<code>{ rate }</code>
<code>api.health()</code>	GET	<code>/health</code>	<code>{ status }</code>

Error Handling

All methods use a shared `request(endpoint, options)` function that:

- Adds auth header if token exists
 - Parses JSON response
 - Throws on non-OK status with `error.status` and `error.data`
-

Theme

File: `lib/theme.js`

Colors

```
colors: {
  green: '#0B6E35',      // Primary – matches web app
  greenDark: '#095C2C', // Hover/pressed state
  greenLight: '#E8F5E9', // Light green bg
  gold: '#D4A017',      // Accent
  white: '#FFFFFF',
  bg: '#FAFCF8',        // Off-white background
  card: '#FFFFFF',
  text: '#1A1A1A',      // Primary text
  textSecondary: '#6B7280', // Secondary text
  textLight: '#9CA3AF', // Muted text
  border: '#E5E7EB',    // Borders
  error: '#EF4444',     // Errors
  success: '#10B981',   // Success/positive
}
```

Fonts

```
fonts: {
  display: 'Fraunces_700Bold',
  heading: 'Fraunces_600SemiBold',
  body: 'DMSans_400Regular',
  bodyMedium: 'DMSans_500Medium',
  bodyBold: 'DMSans_700Bold',
}
```

Spacing

```
spacing: { xs: 4, sm: 8, md: 16, lg: 24, xl: 32, xxl: 48 }
```

Border Radius

```
radius: { sm: 8, md: 12, lg: 16, full: 9999 }
```

Web App vs Mobile App Comparison

Feature	Web (drop-app)	Mobile (drop-mobile)
Auth storage	httpOnly cookie	Bearer token (in-memory)
Navigation	Next.js App Router	Expo Router (Stack + Tabs)
Send flow	4 steps	2 steps
Onboarding	4 steps (info, OTP, PIN, success)	2 steps (info, password)
QR scanner	Simulated camera UI	Simulated camera UI
Bottom nav	5 tabs (Hjem, Aktivitet, Skann, Kontoer, Profil)	4 tabs (Hjem, Send, QR, Profil)
Cards page	Yes (feature-flagged)	No
Merchant page	Yes (feature-flagged)	No
Accounts page	Yes (dedicated)	No (balance on dashboard)
History	Dedicated page with filters	Dedicated screen with filters
Feature flags	Environment variables	Not implemented
UI components	shadcn/ui (Radix)	React Native StyleSheet
State	useState + useAuth hook	useState + api module

Development Notes

- **API compatibility:** Mobile app calls the same API endpoints as the web app (same backend)
- **Dev API URL:** `http://localhost:3000/api` — requires the Next.js dev server running
- **Prod API URL:** `https://drop-app.vercel.app/api`
- **No deep linking** configured yet
- **No push notifications** implemented yet
- **No offline support** — requires network for all operations
- **No biometric auth** — login is email/password only

Mobile Strategy

Drop Mobile App Strategy

Date: 2026-02-11 **Compiled by:** John (AI Director) **Status:** For Review **Task:** MC #580

Executive Summary

Drop is a fintech app (remittance + QR payments) currently built with Next.js 16, React 19, and Tailwind CSS v4. This document evaluates 5 approaches to building native iPhone and Android apps, with a recommendation based on development time, cost, fintech requirements, and code reuse.

RECOMMENDATION: Progressive Web App (PWA) → Native (Capacitor) hybrid approach

Start with PWA for fastest time-to-market (2-3 weeks), then wrap in Capacitor for app store distribution while maintaining 95%+ code reuse with existing Next.js codebase.

Context

Current Tech Stack

- **Frontend:** Next.js 16 (App Router)
- **UI Framework:** React 19
- **Styling:** Tailwind CSS v4 + shadcn/ui
- **Backend:** Next.js API Routes
- **Database:** PostgreSQL 16 via Drizzle ORM (ADR-014; replaced SQLite 2026-03-03)
- **Auth:** JWT via jose (httpOnly cookie)
- **Deployment:** Local/Vercel-ready

Core Fintech Features Needed on Mobile

1. **QR Code Scanning** — camera access for merchant QR codes

2. **Biometric Authentication** — Face ID, Touch ID, fingerprint
 3. **Push Notifications** — transaction alerts, remittance confirmations
 4. **Offline Support** — queue payments when offline, sync when online
 5. **Secure Storage** — JWT tokens, PINs, sensitive data
 6. **Camera Access** — QR scanning for payments
 7. **Geolocation** — merchant proximity (future)
 8. **BankID/Vipps Integration** — Norwegian strong customer authentication (SCA)
-

Option 1: Progressive Web App (PWA)

Overview

Convert existing Next.js app to installable PWA. Users add to home screen, works like native app.

Pros

- **Fastest path:** 2-3 weeks implementation
- **100% code reuse:** No new codebase, enhance existing app
- **Cross-platform:** iOS, Android, desktop from one codebase
- **No app store approval:** Deploy instantly via web
- **Lower cost:** ~10,000-20,000 NOK total
- **Offline support:** Service workers enable offline transactions
- **Push notifications:** Supported on Android and iOS 16.4+
- **Installable:** Add to home screen, app icon, splash screen

Cons

- **iOS limitations:**
 - Safari doesn't support Barcode Detection API (requires polyfill or library like html5-qrcode)
 - Push notifications only on iOS 16.4+ (limited reach in Norway)
 - Some native features restricted
- **No App Store presence:** Harder discovery, lower perceived trust
- **Performance:** Slightly slower than fully native
- **No biometric API:** Cannot access Face ID/Touch ID directly (workaround: use PIN + device unlock)
- **Limited background processing:** Payments must be initiated when app is open

Technical Implementation

```
// next.config.ts - add PWA support
import withPWA from 'next-pwa';

export default withPWA({
  dest: 'public',
  register: true,
  skipWaiting: true,
  disable: process.env.NODE_ENV === 'development'
});
```

Add `manifest.json` for installability, service worker for offline, Web Push API for notifications.

Fintech Compliance

- **PSD2 SCA:** Can implement 2FA with PIN + SMS/email OTP (no native biometrics)
- **GDPR:** Full compliance possible (HTTPS, encrypted storage)
- **App Store rules:** N/A (web distribution)

Development Timeline

- Week 1: Add PWA manifest, service worker, offline support
- Week 2: Implement push notifications, optimize caching
- Week 3: QR scanning polyfill (html5-qrcode), testing
- **Total: 2-3 weeks**

Cost Estimate

- Development: 10,000-15,000 NOK (AI-assisted)
- Hosting: Included in existing Vercel plan
- Push service: Free tier (OneSignal or Firebase)
- **Total: ~10,000-20,000 NOK**

Sources

- [PWA QR Code Detection Demo](#)
 - [PWA vs Native App Comparison 2026](#)
 - [PWA Push Notifications Guide](#)
-

Option 2: Capacitor (PWA ? Native Wrapper)

Overview

Wrap existing Next.js PWA in native shell using Capacitor. Same codebase runs on web, iOS, Android.

Pros

- **95%+ code reuse:** Existing React/Next.js code works as-is
- **App Store distribution:** Full presence on Apple App Store + Google Play
- **Native features:** Camera, biometrics, push, geolocation via plugins
- **Fast development:** 4-6 weeks (PWA base + native plugins)
- **Easy updates:** OTA updates for web content, only rebuild for native changes
- **Cost-effective:** ~50,000-80,000 NOK total
- **Fintech-proven:** Used by banks and fintech apps in production

Cons

- **WebView performance:** Slightly slower than pure native (acceptable for fintech UI)
- **Plugin dependencies:** Reliance on community plugins for advanced features
- **App Store approval:** Must pass Apple/Google fintech reviews
- **Build complexity:** Need Xcode (iOS) and Android Studio for builds
- **Larger app size:** ~20-30MB vs pure native ~10MB

Technical Implementation

```
# Add Capacitor to existing Next.js app
npm install @capacitor/core @capacitor/cli
npx cap init
npm install @capacitor/ios @capacitor/android

# Add native plugins
npm install @capacitor/camera          # QR scanning
npm install @capacitor/push-notifications
npm install @capacitor/biometric-auth # Face ID, Touch ID
npm install @capacitor/secure-storage # Keychain, Keystore
```

Fintech Compliance

- **PSD2 SCA:** Native biometric auth plugin supports Face ID, Touch ID, fingerprint
- **BankID integration:** Via WebView redirect or Capacitor custom plugin
- **Secure storage:** Native keychain (iOS) and keystore (Android)
- **App Store:** Meets fintech requirements (encryption, secure auth)

Development Timeline

- Week 1-2: PWA foundation (from Option 1)
- Week 3: Capacitor setup, iOS/Android projects
- Week 4: Native plugins (camera, biometrics, push)
- Week 5: BankID integration, testing
- Week 6: App Store submission, review
- **Total: 4-6 weeks**

Cost Estimate

- Development: 50,000-70,000 NOK (AI + specialist for native plugins)
- Apple Developer: 1,200 NOK/year
- Google Play: 325 NOK one-time
- Code signing certificates: 3,000 NOK/year
- **Total: ~55,000-80,000 NOK first year**

Sources

- [Capacitor vs React Native Comparison](#)
 - [Capacitor Official Documentation](#)
 - [PWA to Native App Guide for Banking/Fintech](#)
-

Option 3: React Native + Expo

Overview

Rebuild app in React Native using Expo framework. Share some React component logic, but rewrite UI.

Pros

- **True native performance:** 100% native UI components, no WebView
- **Shared React knowledge:** Team already knows React
- **Expo ecosystem:** Fast setup, OTA updates, managed builds
- **Native features:** Full access to camera, biometrics, push, etc.
- **Hot reload:** Fast development iteration
- **Community:** Huge React Native ecosystem, well-supported

Cons

- **Complete rewrite:** Cannot reuse Next.js pages, API routes, or Tailwind styles
- **Longer timeline:** 12-16 weeks (rebuild all 7+ pages)
- **Higher cost:** 150,000-250,000 NOK
- **Two codebases:** Separate web (Next.js) and mobile (React Native)
- **Maintenance burden:** Update features in two places
- **Learning curve:** React Native APIs differ from web React

Technical Implementation

Start from scratch with Expo:

```
npx create-expo-app drop-mobile
cd drop-mobile
expo install react-native-camera react-native-biometrics
```

Rebuild all pages (login, register, dashboard, send money, cards, transactions) using React Native components.

Fintech Compliance

- **PSD2 SCA:** Full biometric support via react-native-biometrics
- **BankID:** Native SDK integration possible
- **Performance:** Best for complex animations, high-frequency updates
- **App Store:** Native apps have easiest approval path

Development Timeline

- Week 1-3: Project setup, authentication flow
- Week 4-7: Core features (dashboard, send money, QR scanning)
- Week 8-10: Cards management, transaction history
- Week 11-12: BankID integration, biometrics
- Week 13-14: Testing, bug fixes
- Week 15-16: App Store submission, review

- **Total: 12-16 weeks**

Cost Estimate

- Development: 120,000-200,000 NOK (rebuild from scratch)
- Expo EAS: 19 USD/month (~220 NOK/month)
- App Store fees: 1,525 NOK/year
- **Total: ~150,000-250,000 NOK first year**

Sources

- [React Native Cost Breakdown 2026](#)
 - [Expo Development Time Benefits](#)
 - [React Native 2026 Guide](#)
-

Option 4: Flutter

Overview

Rebuild app in Flutter (Dart language). Google's cross-platform framework, used by fintech apps like Nubank.

Pros

- **Excellent performance:** Compiled to native ARM code
- **Beautiful UI:** Material Design + Cupertino widgets out of the box
- **Single codebase:** iOS, Android, web from one codebase
- **Hot reload:** Fast development
- **Strong fintech adoption:** Used by major banks (Nubank, Revolut)

Cons

- **Total rewrite:** No code reuse from Next.js/React
- **New language:** Team must learn Dart
- **Longest timeline:** 16-20 weeks (new stack + rebuild)
- **Highest cost:** 200,000-300,000 NOK
- **Two tech stacks:** Flutter mobile, Next.js web
- **Smaller ecosystem:** Less mature than React Native

Fintech Compliance

- **PSD2 SCA:** Full biometric support via local_auth package
- **BankID:** Custom platform channels needed
- **Performance:** Best for high-performance animations
- **App Store:** Native compilation, smooth approval

Development Timeline

- Week 1-4: Learn Dart, Flutter setup, authentication
- Week 5-10: Core features (dashboard, remittance, QR)
- Week 11-14: Cards, transactions, BankID
- Week 15-18: Testing, optimization
- Week 19-20: App Store submission
- **Total: 16-20 weeks**

Cost Estimate

- Development: 180,000-280,000 NOK (new stack learning curve)
 - App Store fees: 1,525 NOK/year
 - **Total: ~200,000-300,000 NOK first year**
-

Option 5: Native (Swift + Kotlin)

Overview

Build fully native apps: Swift for iOS, Kotlin for Android. Separate codebases.

Pros

- **Best performance:** Platform-optimized, zero overhead
- **Full platform access:** Every iOS/Android API available
- **Apple/Google preferred:** Easiest App Store approval
- **Best UX:** Platform-native UI patterns

Cons

- **Two separate codebases:** iOS and Android teams needed
- **Zero code reuse:** Cannot leverage existing Next.js app

- **Longest timeline:** 20-24 weeks (two apps)
- **Highest cost:** 300,000-500,000 NOK
- **Highest maintenance:** Update three codebases (web + iOS + Android)
- **Requires specialists:** Need Swift and Kotlin developers

Development Timeline

- **iOS:** 10-12 weeks (Swift/UIKit or SwiftUI)
- **Android:** 10-12 weeks (Kotlin/Jetpack Compose)
- Parallel development possible but expensive
- **Total: 20-24 weeks**

Cost Estimate

- iOS development: 150,000-250,000 NOK
 - Android development: 150,000-250,000 NOK
 - App Store fees: 1,525 NOK/year
 - **Total: ~300,000-500,000 NOK first year**
-

Fintech Regulatory Requirements (Norway/EU)

PSD2 Strong Customer Authentication (SCA)

Implemented into Norwegian law in September 2019. Requires **two-factor authentication** for online payments:

1. **Something you know:** PIN, password
2. **Something you have:** Phone, hardware token
3. **Something you are:** Biometric (fingerprint, Face ID)

Mobile app requirement: Must support at least 2 of 3 factors. Best practice: PIN + biometrics.

BankID Integration

- **Required for:** Payments >2,000 NOK (Vipps model)
- **Integration:** WebView redirect (easiest) or native SDK (complex)
- **User requirement:** Norwegian social security number or D-number, Norwegian bank account

GDPR Compliance

- **Data encryption:** HTTPS for transit, encrypted storage at rest
- **User consent:** Explicit opt-in for push notifications, data sharing
- **Right to erasure:** Account deletion functionality required

Apple App Store Requirements (2026)

- **External payments:** Allowed in US, but 27% fee on external purchases
- **Age verification:** Declared Age Range API required (Texas law Jan 2026)
- **Privacy labels:** Disclose all data collection in App Store listing
- **Encryption:** Apps handling financial data require encryption declaration

Google Play Requirements (2026)

- **Target SDK:** Android 15 (API level 35) by August 2025
- **Billing Library:** v7+ for in-app purchases (if selling digital goods)
- **Third-party payments:** Allowed, but Google charges fee
- **Data safety:** Declare data collection and security practices

Sources

- [Fintech Laws Norway 2025-2026](#)
- [PSD2 Overview Nordea](#)
- [App Store Requirements 2026](#)
- [BankID Integration Norway](#)

Comparison Matrix

Criteria	PWA	Capacitor	React Native	Flutter	Native
Development Time	2-3 weeks	4-6 weeks	12-16 weeks	16-20 weeks	20-24 weeks
Cost (First Year)	10-20K NOK	55-80K NOK	150-250K NOK	200-300K NOK	300-500K NOK
Code Reuse	100%	95%	20-30%	0%	0%
Performance	Good	Good	Excellent	Excellent	Best
QR Scanning	Polyfill needed	Native plugin	Native	Native	Native

Criteria	PWA	Capacitor	React Native	Flutter	Native
Biometrics	No (PIN only)	Native plugin	Native	Native	Native
Push Notifications	Yes (limited iOS)	Native plugin	Native	Native	Native
Offline Support	Excellent	Excellent	Good	Good	Excellent
App Store Presence	No	Yes	Yes	Yes	Yes
BankID Integration	WebView	WebView/Plugin	Native SDK	Platform channel	Native SDK
PSD2 Compliance	Partial (no biometrics)	Full	Full	Full	Full
Maintenance Burden	Low	Low	Medium	Medium	High
Team Learning Curve	None	Low	Medium	High	Very High

Recommendation: Phased Approach

Phase 1: PWA (Launch in 2-3 weeks)

Goal: Get to market fast, validate demand, iterate quickly.

Implementation:

1. Add PWA manifest to existing Next.js app
2. Implement service worker for offline support
3. Add push notifications (Web Push API)
4. Use html5-qrcode library for QR scanning (works on iOS/Android)
5. Deploy to production

Cost: 10,000-20,000 NOK **Timeline:** 2-3 weeks

Limitations:

- No App Store presence (web distribution only)
- No native biometrics (use PIN + SMS OTP for SCA)
- Push notifications limited on older iOS devices

Result: Functional mobile app users can install to home screen. Test market fit, gather feedback.

Phase 2: Capacitor Wrapper (Launch in App Stores)

Goal: App Store distribution, native features, maintain code reuse.

Implementation:

1. Wrap existing PWA in Capacitor shell
2. Add native plugins:
 - @capacitor/camera for QR scanning
 - @capacitor/biometric-auth for Face ID/Touch ID
 - @capacitor/push-notifications for native push
 - @capacitor/secure-storage for keychain/keystore
3. Integrate BankID via WebView redirect
4. Submit to Apple App Store and Google Play

Cost: 45,000-60,000 NOK (incremental, after PWA) **Timeline:** 3-4 weeks (after PWA is done)

Result: Native apps in App Stores with 95% code shared with web app. Full PSD2 compliance with biometric SCA.

Phase 3: Optimize (If Needed)

Goal: If performance becomes bottleneck, consider React Native rewrite.

Trigger: User feedback indicates poor performance, or complex features require native optimization.

Decision point: 6-12 months after launch, based on data.

Why This Approach Wins

Speed to Market

- **PWA in 2-3 weeks** — fastest path to validate product-market fit
- **Capacitor in 4-6 weeks** — App Store presence without rebuilding

Cost Efficiency

- **55,000-80,000 NOK total** (PWA + Capacitor) vs 150,000-500,000 NOK (React Native/Flutter/Native)
- **95% code reuse** — maintain one codebase for web + mobile
- **AI-assisted development** — existing Next.js stack already built, just enhance

Fintech Compliance

- **PSD2 SCA:** Capacitor supports biometric auth (Face ID, Touch ID, fingerprint)
- **BankID integration:** WebView redirect works (same as Vipps does it)
- **Secure storage:** Native keychain/keystore via Capacitor plugins
- **App Store approval:** Capacitor apps pass fintech reviews (proven track record)

Risk Mitigation

- **Start simple (PWA)** — if market doesn't respond, minimal loss
- **Upgrade when needed (Capacitor)** — low cost to add native shell
- **Optionally rewrite later** — if performance critical, React Native is still an option

Real-World Validation

- **Fintech PWAs:** Revolut started with PWA, added native later
 - **Capacitor adoption:** Used by banks and fintech apps globally
 - **Next.js compatibility:** Capacitor officially supports Next.js
-

Technical Risks & Mitigations

Risk 1: iOS Safari Limitations

Issue: Safari doesn't support Barcode Detection API. **Mitigation:** Use html5-qrcode library (works on all browsers, 50K+ downloads/week).

Risk 2: BankID Integration Complexity

Issue: BankID may require native SDK instead of WebView. **Mitigation:** Start with WebView (how Vipps does it). If needed, build custom Capacitor plugin (1-2 weeks).

Risk 3: App Store Rejection

Issue: Fintech apps face stricter review. **Mitigation:** Capacitor apps pass regularly. Ensure encryption declaration, privacy policy, age verification API (iOS).

Risk 4: Performance on Low-End Devices

Issue: WebView slower than native on old Android phones. **Mitigation:** Optimize bundle size, lazy load routes, use Next.js performance best practices. If still slow, React Native rewrite is exit strategy.

Risk 5: Push Notification Reliability

Issue: Web Push less reliable than native push on iOS. **Mitigation:** Capacitor uses native push APIs, solves this problem.

Team & Skillset Needed

Phase 1: PWA (2-3 weeks)

- **John (AI Director):** Coordinate development
- **Builder agent:** Implement PWA manifest, service worker
- **Dev agent:** Add html5-qrcode library, test QR scanning
- **QA:** Test on real iOS/Android devices

No new hires needed. Existing AI-assisted workflow handles this.

Phase 2: Capacitor (3-4 weeks)

- **John:** Coordinate Capacitor setup
- **Builder agent:** Add Capacitor, configure iOS/Android projects
- **External specialist (optional):** 1-2 days for Xcode/Android Studio setup if issues arise
- **QA:** Test on physical devices

Possible external cost: 5,000-10,000 NOK for specialist troubleshooting (Xcode signing, Android builds).

Go/No-Go Decision

For GO (PWA ? Capacitor):

- **Fastest time-to-market:** 2-3 weeks (PWA) → validate demand
- **Lowest cost:** 55,000-80,000 NOK total vs 150,000-500,000 NOK alternatives
- **Highest code reuse:** 95% shared with web app
- **Fintech compliant:** PSD2 SCA, GDPR, App Store requirements met
- **Proven technology:** Capacitor used by banks, fintech apps in production
- **Flexible:** Can upgrade to React Native later if needed

Risks:

- iOS Safari limitations (mitigated with polyfill)
- BankID might need custom plugin (mitigated with WebView first)
- WebView performance on old devices (acceptable for fintech UI, validated by other apps)

Recommendation: **GO** with PWA ? Capacitor phased approach

This is the optimal path for Drop:

1. **Validate fast** with PWA (2-3 weeks)
 2. **Scale smart** with Capacitor (3-4 weeks)
 3. **Rewrite only if needed** with React Native (future decision)
-

Next Steps

1. **Week 1:** Implement PWA manifest, service worker, offline support
2. **Week 2:** Add Web Push API, test push notifications
3. **Week 3:** Integrate html5-qrcode, test QR scanning on iOS/Android
4. **Deploy PWA:** Launch to users, gather feedback
5. **Week 4-5:** Add Capacitor wrapper, native plugins (camera, biometrics, push)
6. **Week 6:** BankID integration via WebView
7. **Week 7:** Submit to Apple App Store and Google Play
8. **Week 8:** App Store approval, public launch

Total timeline: 6-8 weeks from start to App Store launch. **Total cost:** 55,000-80,000 NOK.

Sources Summary

PWA Research

- [PWA QR Code Detection](#)
- [PWA vs Native Comparison 2026](#)
- [PWA Push Notifications Guide](#)
- [PWA Offline Support](#)

Capacitor Research

- [Capacitor vs React Native](#)
- [Capacitor Documentation](#)
- [PWA to Native App for Banking/Fintech](#)
- [ABN AMRO Mobile Development Debate](#)

React Native/Expo Research

- [React Native Cost Breakdown](#)
- [Expo Development Time](#)
- [React Native 2026 Guide](#)
- [Expo Pricing](#)

Fintech Compliance Research

- [Fintech Laws Norway 2025-2026](#)
- [PSD2 Overview](#)
- [Biometric Authentication in Fintech](#)
- [App Store Requirements 2026](#)
- [BankID Norway](#)
- [Vipps Integration](#)

App Store Policy Research

- [Apple External Payment Rules 2025](#)

- [App Store Accountability Acts 2026](#)
 - [Google Play Third-Party Payments](#)
-

Compiled: 2026-02-11 by John (AI Director) Status: Awaiting Alem review and GO/NO-GO decision