Cryptrapay React Native Mobile Apps — Full Build Specification (Merchant + User) with NFC

Audience: Coding AI + engineering team
Codebase: React Native (Expo EAS Managed w/ custom dev clients)
Platforms: iOS 16+ (CoreNFC NDEF Reader only), Android 9+ (NFC Reader + HCE)
Chains/Infra: Solana (on-chain value), Cryptrapay backend (REST/WebSocket), third-party billers & card issuer

---

0) Product Scope

User App (Consumers)

- Wallet (USDC/SOL), send/receive, P2P.
- Bills & airtime/data purchase.
- USD Virtual Card: apply, load, freeze, transactions.
- **Payments**: Scan QR **or Tap via NFC** to pay merchants; deep link handoff.

Merchant App (Sellers/Outlets)

- POS keypad → generate payment intent.
- **Accept payments by QR or NFC Tap** (reader mode; optional Android Tap to Pay/HCE pairing with User app).
- Invoices, catalog, refunds, settlements/reports.

Auth: Privy Wallet (primary) + passkey fallback.
Region: NG first; multi-currency aware.
Compliance: KYC/AML; PCI scope avoided (no PAN in-app).

---

1) Monorepo Layout

```
cryptrapay/
  apps/
    user-app/           # Expo RN app (consumer)
    merchant-app/       # Expo RN app (merchant)
  packages/
    ui/                 # shared components
    core/               # domain models, zod schemas, utils
    api-client/         # OpenAPI TS client (REST)
    wallets/            # Privy bridge, Solana helpers
    nfc/                # NFC cross-platform layer (RN + native bridges)
    localization/       # i18n bundles
    config/             # env/flags
```

Tooling: TS strict; ESLint+Prettier; React Query + Zustand; Zod; Jest/Detox; Sentry; EAS Build/Submit.

---

2) Architecture Overview

RN App → API Client (REST/WS) → Cryptrapay Backend → Solana RPC/Websockets → Providers (billers, issuer).
Realtime via WS/SSE + push (FCM/APNs).

Key SDKs/Libraries - Solana: `@solana/web3.js`, `@solana/spl-token`, optional `@solana-mobile/mobile-wallet-adapter`. - Auth: Privy RN (WebView + deep link), fallback passkeys (`react-native-passkeys`). - NFC: `react-native-nfc-manager`; Android HCE (custom native service); optional Google Tap to Pay SDK; iOS CoreNFC (NDEF reader only). - Device: `expo-camera` (QR), `expo-secure-store` (secrets), `react-native-mmkv` (fast storage), `react-native-device-info`. - Networking: `expo-network`, TLS pinning via `react-native-cert-pinner`.

Security - Access/refresh tokens; SecureStore only for refresh. - Signed payment payloads (Ed25519).
- Device integrity (Play Integrity / DeviceCheck) for NFC sessions.
- Jailbreak/root detection → degrade features (no HCE).

---

3) Domain Models (packages/core)

```
export type Currency = 'USDC' | 'SOL' | 'NGN' | 'USD';

export interface UserProfile { id: string; walletAddress: string; kycLevel: 0|1|
2; email?: string; phone?: string; displayName?: string; country: string;
createdAt: string; }
export interface Session { accessToken: string; refreshToken: string;
expiresAt: number; provider: 'privy'|'passkey'; }

export interface PaymentIntent {
  id: string; merchantId?: string; amount: string; currency: Currency;
description?: string;
  status: 'created'|'pending'|'confirmed'|'failed'|'expired'|'refunded';
  expiresAt?: string; createdAt: string;
  transport: { qrPayload: string; ndefUri: string; }; // unified URI (see §7)
  onChain?: { mint?: string; recipient: string; reference?: string; signature?:
string };
}

export interface MerchantProfile { id: string; displayName: string; legalName:
```

```
string; settlementCurrency: Currency; walletAddress: string; posMode:
'simple'|'catalog'; }

export interface CardSummary { id: string; last4: string; brand:
'VISA'|'MASTERCARD'; state:
'active'|'inactive'|'frozen'|'pending'|'terminated'; balanceMinor: string;
currency: 'USD'; }
```

---

4) API Contracts (REST)

Headers: `Authorization: Bearer <token>`; `X-App: user|merchant`; `X-Platform: ios|android`

Auth

- `POST /v1/auth/privy/start` → `{ redirectUrl }`
- `POST /v1/auth/privy/callback` → `{ accessToken, refreshToken, expiresAt, user }`
- `POST /v1/auth/refresh` | `POST /v1/auth/logout`

Users

- `GET /v1/users/me` | `PATCH /v1/users/me`
- `POST /v1/users/kyc/start` | `GET /v1/users/kyc/status`

Wallet & Payments

- `POST /v1/wallet/intents` `{ amountMinor, currency, recipient, metadata? }` → `PaymentIntent`
- `GET /v1/wallet/intents/{id}` → `PaymentIntent`
- `POST /v1/wallet/submit` `{ intentId, signature }` → `{ status }`

NFC Sessions (NEW)

- `POST /v1/nfc/sessions` `{ intentId }` → `{ sessionId, apduProfile?: { aid, selectResp }, expiresAt }`
- `POST /v1/nfc/sessions/{sessionId}/confirm` `{ deviceNonce, proof }` → `{ status: 'bound' }`
- `POST /v1/nfc/sessions/{sessionId}/complete` `{ signature }` → `{ status: 'paid'|'pending' }`
- `DELETE /v1/nfc/sessions/{sessionId}` → 204 (cancel)

Bills & Airtime

- `GET /v1/billers?category=` → `Biller[]`

- `POST /v1/bills/pay` `{ billerId, accountRef, amountMinor, currency }` → `BillPayment`
- `GET /v1/bills/{id}`

Cards

- `POST /v1/cards/apply` → `CardSummary`
- `GET /v1/cards` → `CardSummary[]`
- `POST /v1/cards/{id}/load` `{ amountMinor, source }`
- `POST /v1/cards/{id}/freeze` | `/unfreeze`
- `GET /v1/cards/{id}/txns?cursor=`
- `GET /v1/cards/{id}/details` → hosted details URL (never PAN in-app)

Merchant

- `GET /v1/merchant/me` | `PATCH /v1/merchant/me`
- `POST /v1/merchant/payment-intents` `{ amountMinor, currency, description?, metadata? }` → `PaymentIntent`
- `GET /v1/merchant/payment-intents/{id}` | `POST /v1/merchant/refunds`
- `GET /v1/merchant/payouts?from=&to=` | `POST /v1/merchant/catalog` (CRUD)

Notifications

- `POST /v1/devices/register` `{ token, platform, app }`
  Events: `payment_confirmed`, `invoice_paid`, `kyc_approved`, `card_txn_posted`, `nfc_session_timeout`

Error shape

```
{ "error": { "code": "string", "message": "human readable", "retryable": false,
"details": {} }}
```

---

5) Navigation & Screens

Shared Onboarding

1) Splash → version/maintenance check
2) Legal gates (ToS/Privacy)
3) Auth: Privy (WebView) → deep link
4) KYC prompt (tiered)
5) Setup PIN/biometric

User App

- **Home**: balances, quick actions (Pay Bill, Airtime, Load Card, **Tap/Scan to Pay**), activity feed.

- **Wallet**: tokens, send/receive (QR, address), history.
- **Bills/Airtime**: category → form → receipt.
- **USD Card**: apply, load, freeze/unfreeze, transactions, limits.
- **Pay**: **NFC Tap-to-Transfer** (Android HCE/iOS NDEF) or QR Scan; confirm → on-chain submit.
- **P2P**: contacts/tags, request & send, QR.
- **Profile**: KYC status, limits, devices, security, notifications, support.

Merchant App

- **POS**: keypad → `Create Payment` → modal offering **NFC** (reader) or **QR** with countdown.
- **Incoming**: realtime list; detail/receipt.
- **Invoices**: build, send, track, reminders.
- **Catalog**: CRUD items; quick add.
- **Refunds**: locate intent; partial/full; PIN.
- **Payouts/Reports**: schedule, export.
- **Settings**: outlets, staff roles, taxes, receipt branding, webhooks.

---

6) Unified Payment URI (QR & NFC)

```
cryptrapay://pay?
pid=<intentId>&ref=<reference>&a=<amountMinor>&cur=<currency>&m=<merchantId>&r=<recipient>&exp=<i
```

- **QR**: URI encoded to bitmap (L-level error correction).
- **NFC (NDEF)**: URI placed in an NDEF record (TNF Well-Known, RTD URI).
- **Signature**: payload signature retrievable via GET intent; client verifies before submit.

---

7) NFC Design

Capability Matrix

Capability	Android	iOS
Reader (merchant scans user/app/tag)	✅Foreground reader mode	✅Foreground NDEF reader
Host Card Emulation (user emulates)	✅HCE (APDUs)	❌Not allowed
NDEF URI exchange	✅	✅
Tap to Pay first-party APIs	✅(Google Tap to Pay; partner)	✅(Apple Tap to Pay; partner)

Flows

A) Android User (HCE) → Merchant (Reader)

1. Merchant creates intent → shows **NFC/QR** modal; opens reader mode.
2. User taps phone; **HCE** service responds with short-lived session token bound to `intentId`.
3. Merchant app posts `/v1/nfc/sessions/{sessionId}/confirm` with device proof.
4. User app prompts biometric → constructs & signs Solana tx; submits via `/v1/wallet/submit`.
5. Merchant WS sees `confirmed` → prints receipt.

B) iOS or Android User (NDEF URI) → Merchant (Reader)

1. Merchant opens reader; user taps and transmits NDEF **URI** (signed).
2. Merchant resolves `intentId` via GET to verify.
3. User app finalizes payment on-chain (prompt) and backend notifies merchant.

Client State Machine (both apps)

```
Idle → NFC Enabled? → StartSession(intentId) → WaitingForTag → TagDetected →
  (Android) APDU/HCE Handshake → Bound(sessionId)
  (iOS) NDEF URI Read → Intent Verified
→ User Confirms → OnChain Submit → Confirmed|Failed|Timeout
```

Libraries & Native Bridges

- `react-native-nfc-manager`: NDEF read/write, tag detection.
- Android HCE: native `HostApduService` exposed via TurboModule.

Permissions & Manifests

- **Android**: `android.permission.NFC`; `<uses-feature android:name="android.hardware.nfc" android:required="false"/>`.
  HCE: declare `<service ... android:permission="android.permission.BIND_NFC_SERVICE">` with `<host-apdu-service>` and AID group. Foreground service during active emulation.
- **iOS**: NFC entitlement for NDEF; `NFCReaderUsageDescription` in Info.plist. No background reads; sessions are user-initiated.

Security Controls

- Ephemeral session TTL ≤ 60s; single-use.
- Device attestation (Play Integrity / DeviceCheck) on `/confirm`.
- Block HCE on rooted/jailbroken devices; require biometric before emulation.
- Strict signature/nonce checks; pin API TLS.
- Always provide QR fallback.

---

8) UI/UX for NFC

- **NFC Sheet**: states `Ready → Detecting → Securing Session → Confirm on Device → Done` with progress and haptics on tag detect.
- **Fallbacks** at every step: Switch to QR, Copy Link.
- **Error toasts** mapped to error codes (see §13).
- Accessibility: VoiceOver/TalkBack announcements for state changes; minimum touch targets 44×44.

---

9) State, Caching, Realtime

- React Query for server data; keys like `wallet.intent.$id`.
- Zustand for UI (NFC modal visibility, keypad amount, sheet state).
- WS channel `wss:///v1/realtime?intentId=` → `pending`, `onchain_submitted{signature}`, `confirmed`, `expired`.
- Push notifications for out-of-band `payment_confirmed` and `nfc_session_timeout`.

---

10) QR & Scanner

- `expo-barcode-scanner` for QR, throttled scans; pre-confirmation summary.
- Manual code entry as fallback.

---

11) Security & Privacy

- PIN/biometric lock for sensitive actions.
- No PAN/seed phrases ever stored or logged.
- Token redaction in logs; crash reports scrubbed.
- Clipboard protections (explicit copy actions only).
- Rate limiting and exponential backoff on retries.

---

12) Feature Flags

- `cardsEnabled`, `nfcEnabled`, `androidHceEnabled`, `p2pEnabled`, `darkMode`, `airtimeCategories`, `referrals`.
- Server sends per-platform capability to disable HCE on non-compliant devices.

---

13) Validation & Error Codes

- Auth: `AUTH/UNAUTHORIZED`, `AUTH/SESSION_EXPIRED`

- Payments: `PAYMENT/EXPIRED`, `PAYMENT/AMOUNT_TOO_LOW`, `PAYMENT/CURRENCY_UNSUPPORTED`
- Bills/Cards: `BILL/INVALID_REFERENCE`, `CARD/INSUFFICIENT_FUNDS`
- **NFC**: `NFC/NOT_SUPPORTED`, `NFC/DISABLED`, `NFC/USER_CANCELLED`, `NFC/SESSION_TIMEOUT`, `NFC/SECURITY_BLOCKED`, `NFC/HCE_NOT_ALLOWED_IOS`

---

14) Component Inventory (packages/ui)

- **Atoms**: Button, Text, TextField, AmountInput, Badge, Toggle, Icon, Loader, **NfcStatusBadge**
- **Molecules**: Keypad, QRCodeView, TokenSelector, BillerCard, CardSummaryTile, TxnListItem, **NfcTapArea**, **NfcScanSheet**
- **Organisms**: POSKeypadCard, PaymentQRModal, **PaymentNFCModal**, InvoiceEditor, BillPayForm, CardLoadSheet

---

15) Pseudocode & Native Stubs

Merchant → Create Payment

```
const amountMinor = toMinor(amount, currency);
const { data: intent } = await api.post('/v1/merchant/payment-intents', {
amountMinor, currency });
openModal(<PaymentNFCModal ndefUri={intent.transport.ndefUri}
qr={intent.transport.qrPayload} expiresAt={intent.expiresAt} />);
```

Android HCE (Kotlin)

```
class CryptraHceService: HostApduService() {
  override fun processCommandApdu(apdu: ByteArray, extras: Bundle?): ByteArray {
    // INS_SELECT → return SELECT_OK
    // INS_GET_SESSION → return ephemeral session from /v1/nfc/sessions
    return handleApdu(apdu)
  }
  override fun onDeactivated(reason: Int) { /* cleanup */ }
}
```

iOS NDEF Reader (Merchant)

```
await NfcManager.requestTechnology(NfcTech.Ndef);
const tag = await NfcManager.ndefHandler.read();
const uri = parseNdefUri(tag);
const intent = await api.get(`/v1/wallet/intents/${uri.pid}`);
```

Client NFC Session

```
const startNfc = async (intentId: string) => {
  if (!isNfcAvailable()) throw new Error('NFC/NOT_SUPPORTED');
  const { sessionId } = await api.post('/v1/nfc/sessions', { intentId });
  await nfc.openReader(); // or enable HCE
  const proof = await attestDevice();
  await api.post(`/v1/nfc/sessions/${sessionId}/confirm`, { deviceNonce,
proof });
};
```

---

16) Testing Strategy

- **Unit**: Jest + RTL.
- **Contract**: MSW mocks for REST/WS.
- **E2E**: Detox on devices; NFC tests require physical devices (emulators lack NFC).
- **Android NFC Tests**: HCE handshake, timeout, foreground service behavior, device integrity blocks.
- **iOS NFC Tests**: NDEF read flow, permission prompts, session lifecycle.

---

17) Analytics & Telemetry

Events: `auth_login`, `kyc_started`, `kyc_approved`, `intent_created`, `intent_paid`, `intent_failed`, `bill_paid`, `card_loaded`, `refund_issued`, `payout_requested`, **NFC**: `nfc_sheet_open`, `nfc_tag_detected`, `nfc_session_bound`, `nfc_biometric_ok`, `nfc_timeout`, `nfc_fallback_qr`.

---

18) DevOps & Provisioning

- EAS build profiles: `dev`, `staging`, `prod`; runtime versioning; OTA updates with critical feature flag guards.
- Android manifest & HCE service declarations; iOS NFC entitlement; store listing privacy items explaining NFC use.

---

19) Accessibility & Internationalization

- Screen reader labels; focus order; dynamic type; high-contrast themes.

- i18n via i18next; start `en-NG`; modular namespaces (home, wallet, pos, nfc, bills, cards, settings).

---

20) Offline & Resilience

- Offline banner; read-only cached balances; retry with capped backoff.
- **NFC fallbacks**: if unsupported/disabled/timeout → show QR and deep link.

---

21) Risk & Mitigations (NFC)

- iOS HCE unavailable → use NDEF URI + in-app confirmation.
- Session hijack → signed payloads, short TTL, device attestation, bind session to user+device.
- POS interference → UI guidance + haptics + retries.
- Regulatory changes → gate via feature flags, remote config.

---

22) Acceptance Criteria (Happy Paths)

1. Merchant creates intent → User pays via **NFC** on Android (HCE) → Merchant receipt shows **confirmed** within TTL.
2. Merchant creates intent → User (iOS) taps → NDEF URI verified → User confirms and pays → Merchant notified.
3. Any NFC failure cleanly falls back to QR.
4. Logs contain no secrets; Sentry events redacted.

---

23) Roadmap (Post-MVP)

- Partner integrations for **Tap to Pay** (Apple/Google first-party) in select regions.
- NFC loyalty stamp (separate AID/URI).
- Offline intents with delayed on-chain submission.