Technical Assessment & Requirements Documentation: Hydra Mobile SDK for Android & iOS

1. Functional Requirements

These represent the required capabilities for the Hydra Mobile SDK to deliver a complete and idiomatic developer experience.

1.1 Hydra Node Connectivity Layer

The SDK must provide:

Core Transport

- WebSocket client with:
  - Connection management
  - Automatic reconnection
  - Snapshot-based resync
  - Event stream parsing (TxValid, TxInvalid, Snapshot, Head events)
- HTTP client for:
  - /commit
  - /open
  - /close
  - Status endpoints

1.2 Typed Message Models

SDK must define typed models for:

- Hydra command messages
- Hydra event messages
- Heads, snapshots, transactions
- Errors and rejection reasons

This guarantees type-safety and developer usability.

1.3 Head Lifecycle Management

SDK must implement:

- **createConnection()**
- **openHead()**
- **joinHead()**
- **commit()**
- **closeHead()**
- **fanout() monitoring**

With callback/event emitters such as:

- onHeadOpened
- onHeadClosed
- onSnapshot
- onTxValid
- onTxInvalid
- onError

1.4 Transaction Handling

The SDK must support:

- L2 transaction creation (payment, custom scripts)
- Signing through mobile-secure keys
- Submission through Hydra WS
- Verification via event stream

1.5 Wallet Bridge Requirements

To support Hydra transaction signing:

- Hydra SK must be stored in **Secure Enclave / Keystore**
- SDK must expose signer interfaces for:
    - Hydra SK
    - L1 signing via external wallets (CIP-30 or custom)

Wallet bridge patterns may include:

- Deep links
- WebView injection
- Embedded wallet mode

1.6 Persistence

The SDK must persist:

- Hydra keys
- Last known snapshot
- Pending operation queue
- Connection state

This supports resume-after-crash scenarios.

1.7 Sample Applications

SDK must include reference implementations:

- **Flutter Example App**
    - fast-micropayments
- **Android Kotlin sample**
- **iOS Swift sample**

These must demonstrate an end-to-end Hydra flow:

1. Connect
2. Open/join Head
3. Commit
4. L2 micropayment
5. Close Head
6. Fanout

2. Non-Functional Requirements

2.1 Performance

- WS latency target: **<150 ms** typical mobile conditions
- Signing time: **<50 ms** using secure enclave
- Reconnection time: **<3 sec**

Hydra L2 throughput is extremely high (1M+ TPS) and SDK must not bottleneck it.

2.2 Reliability

- Automatic retry handling
- Crash-safe persistence
- Event-ordering guarantees
- Snapshot-based state restoration

2.3 Security Requirements

- Hardware-backed key storage
- TLS + cert pinning
- Replay protection
- No private keys transmitted externally
- Secure JSON parsing & validation

2.4 Compatibility

- Flutter 3.x
- Android API 23+
- iOS 13+
- hydra-node latest stable

3. Technical Hurdles & Mitigations

3.1 Hydra's Online Participant Requirement

Issue: Mobile clients disconnect frequently.

Mitigation:

- Synthesized session model
- Snapshot-based resume
- Provider-hosted Head architecture

3.2 State Synchronization Challenges

Hydra emits rapid events → risk of missed or out-of-order messages.

Mitigation:

- Buffered event pipeline
- WS reconnection with last-seen snapshot

- Local message sequencing

3.3 Signing Flows Across Platforms

Issue: Hydra L2 signing is distinct from L1 CIP-30 flows.

Mitigation:

- Unified signer interface
- Key derivation stored in secure enclave
- Platform abstractions for cardano-sk vs hydra-sk

4. Project Boundaries

Included

- SDK implementation (Flutter, Android, iOS)
- Wallet bridge adapters
- Sample applications
- Documentation & architecture diagrams
- Tests and CI setup

Not Included

- Running hydra-node on mobile devices
- Smart contract development (outside examples)
- L1 wallet implementation

6. Risks & Mitigation Summary

Risk	Mitigation
Network instability	robust reconnection & snapshot sync
Complex signing logic	abstract signer interfaces
Hydra topology restrictions	provider-managed Heads
Performance variability	optimized WS pipeline
Developer onboarding	comprehensive docs & examples