CASE STUDY

Astra Chat

Decentralized · End-to-End Encrypted · React Native

Muhammad Hassan · Solo Lead Engineer · 2025

hassanprodeveloper.com

4 Weeks	Zero	iOS + Android	E2EE
FULL DELIVERY	THIRD-PARTY BACKENDS	CROSS-PLATFORM	FULL ENCRYPTION

React Native **Matrix Protocol** **End-to-End Encryption** **LiveKit** **TypeScript** **Zustand** **4 Weeks** **Solo Lead**

PROJECT OVERVIEW

Astra Chat is a production-grade, Matrix-protocol mobile messaging app built for iOS and Android in React Native — with **end-to-end encryption**, LiveKit voice/video calling, encrypted push notifications (without Firebase), device verification, and a zero-backend-dependency architecture. Delivered as **sole lead engineer in a 4-week sprint.**

The Client and Their Vision

Astra is building a compliance-first digital trust platform — a decentralized ecosystem where businesses and partners communicate securely without relying on centralized services like Firebase or big-tech authentication providers. The vision was a messaging super-app that could stand alongside Telegram and Slack in terms of UX, while running on infrastructure the client actually owned and controlled.

The client had an existing iOS app (ChitchatX) and needed it fully rebuilt as a cross-platform React Native product. The core protocol was already chosen: **Matrix** — an open, federated, decentralized communication standard with built-in end-to-end encryption. My job was to make it work on mobile.

CLIENT CONSTRAINTS

React Native CLI · **No Firebase** · **No Google/Facebook auth** · **Matrix-native flows only** · 4-week delivery window · iOS + Android parity · White-label ready architecture · Enterprise-grade security throughout

FULL FEATURE SCOPE

Module	Features Required
Authentication	Login/Registration (Matrix UIA), SSO (OIDC/SAML), CAPTCHA, QR-code device login, session recovery
Messaging	1:1 & group chat with E2EE, threads, replies, reactions, read receipts, typing indicators
Calls	1:1 VoIP, group video conferencing (LiveKit), native CallKit/Android call UI, call logs
Media	Encrypted file/image/video upload & download, shared media browsing
Notifications	Push via APNs without Firebase, encrypted notification handling, background event decryption
Security	E2EE, cross-signing, device verification (QR/SAS), SSSS key backup & recovery
Settings	Profile management, device/session management, identity server configuration
Search	Global search across rooms, messages, and files; in-room message search

The Matrix SDK Wall

Matrix provides an official JavaScript SDK (`matrix-js-sdk`). The obvious path would be to install it and start building. But there was a fundamental incompatibility that stopped this cold.

❌ THE BLOCKER

The official matrix-js-sdk depends on browser-only APIs — WebAssembly, the Web Crypto API, and IndexedDB — for its E2EE encryption layer. React Native runs in a JavaScript runtime without any of these browser features. Using the SDK directly breaks encryption entirely on mobile.

→ WHAT THIS MEANT

The primary SDK Matrix officially supports was completely off the table. Every alternative — native bridging, polyfills, or building from scratch — carried painful trade-offs around timeline, team expertise, and long-term maintainability.

THREE OPTIONS EVALUATED

Option A — Native Bridge	Use Matrix Rust/Kotlin/Swift SDKs natively and bridge to React Native via NativeModules. Full E2EE support — but requires building everything twice (iOS + Android separately), massively expanding scope and timeline beyond a 4-week window.
Option B — Polyfill js-sdk	Polyfill the missing browser APIs in React Native. Fragile by nature — crypto polyfills are frequently incomplete, IndexedDB shims are unreliable under load, and encryption correctness becomes impossible to guarantee in production.
Option C — @unomed/react-native-matrix-sdk ✓ CHOSEN	Third-party SDK using the official Matrix Rust SDK under the hood, generating React Native Turbo Modules via UniFFI. Not officially supported by Matrix — but architecturally sound. Zero documentation: implementation had to be reverse-engineered from generated FFI interface files.

DECISION RATIONALE

Option C was chosen because using the official Matrix Rust SDK as the underlying foundation — even via an unofficial bridge — provides the highest long-term reliability. A polyfill approach risks silent encryption failures. A full native bridge doubles development scope on a tight 4-week timeline. The documentation gap in Option C was a **solvable engineering problem**, not a dealbreaker.

Architecture Decisions Under Uncertainty

1

Using AI to generate SDK documentation

The SDK generates binding files through UniFFI — large, structured FFI interface files that describe every method, type, and event emitter. These files were fed into Google Gemini with a prompt to produce structured API documentation including method signatures, expected inputs/outputs, and event names. This produced a working reference in hours rather than days, and enabled full implementation planning before writing a single line of app code.

2

Separating the Matrix client from the React layer

Never let the Matrix client bleed into UI components. The SDK is event-driven and stateful — treating it as a React state source causes cascading re-renders, race conditions, and impossible debugging. Solution: TypeScript singleton classes own all Matrix operations. Zustand acts as a one-way mirror: SDK events update the store, components read from the store. The SDK is always the source of truth — Zustand is purely a snapshot for rendering.

3

Securing sessions without any backend service

With no Firebase and no custom server, session persistence lives entirely on-device. react-native-keychain stores Matrix session credentials in the OS-level secure enclave (Keychain on iOS, Keystore on Android). The Matrix SDK handles key synchronization with the homeserver automatically — so encryption continuity is maintained across logins with zero server-side session management.

4

E2EE push notifications without Firebase

Matrix's push model sends an event ID (not message content) to APNs — this preserves E2EE because the message never leaves the encrypted channel. A background fetch handler using @notifee/react-native receives the push, fetches the event from the homeserver using the stored session, decrypts it locally, and displays the notification. The push infrastructure sees only an opaque identifier.

ARCHITECTURE PATTERN — MATRIX CLIENT CLASS

```
// Matrix client is a singleton — never imported into UI components
class MatrixChatClient {
  private client: MatrixClient | null = null

  async initialize(session: StoredSession) {
    this.client = await MatrixClient.create({
      homeserverUrl: session.homeserver,
      userId: session.userId,
      deviceId: session.deviceId,
    })
    // SDK is source of truth — Zustand only mirrors it
    this.client.on('room.timeline', (event) => {
      useChatStore.getState().addMessage(event)
    })
  }
}
```

Component Relationships & Data Flow

The diagram below shows the full system: the React Native mobile client, the self-hosted Matrix homeserver, and external services. **All messaging data flows through the Matrix protocol with end-to-end encryption maintained throughout.** LiveKit is used for voice/video with Matrix as the signalling layer. Firebase is explicitly excluded.

DIAGRAM LEGEND

■ E2EE / Matrix events	■ Matrix protocol	■ WebRTC / signalling	■ Push notification flow

What Was Built

I led the project as **sole senior engineer** — responsible for all architecture decisions, the full Matrix integration, calling infrastructure, and encryption flows. A junior developer joined mid-project for UI component building, which I then wired into the Matrix data layer.

FEATURES DELIVERED

Module	Delivered
Authentication	✓ Matrix UIA login, registration, session recovery ✓ SSO via OIDC/SAML (in-app browser flow) ✓ QR-code device login ✓ Soft logout & Keychain session persistence
Messaging	✓ Real-time 1:1 and group chat with E2EE ✓ Message threads, replies, reactions ✓ Typing indicators & read receipts ✓ File, image, and video sharing (encrypted)
Calls & Conferencing	✓ 1:1 VoIP calling via LiveKit ✓ Group video conferencing ✓ Native CallKit (iOS) / Android IncomingCall UI ✓ Call logs and missed call notifications
Security	✓ End-to-end encryption (Rust SDK managed) ✓ Device verification via QR and SAS ✓ Cross-signing and SSSS key backup ✓ Key recovery and session continuity
Infrastructure	✓ E2EE push notifications via APNs (no Firebase) ✓ SDK-managed SQLite cache (encrypted at rest) ✓ OS Keychain for zero-backend session storage ✓ White-label ready component architecture

06 — DELIVERY TIMELINE

4-Week Sprint Breakdown

Week	Focus	Deliverables
Week 1	**Authentication & SDK Foundation**	Matrix SDK integration, login/registration flows, SSO via OIDC, QR device login, session storage with Keychain, Zustand architecture, singleton MatrixClient class.
Week 2	**Core Messaging System**	Real-time chat rooms, E2EE message flow, threads, replies, reactions, typing indicators, read receipts, contact management, user directory search.
Week 3	**Calls, Media & Notifications**	LiveKit integration, 1:1 and group calling, native CallKit/Android UI, encrypted media upload/download, E2EE push notification pipeline without Firebase.
Week 4	**Security, Polish & Delivery**	Device verification (QR + SAS), cross-signing, SSSS key backup, settings screens, global search, bug fixes, performance profiling, production build.

07 — TECHNOLOGY STACK

Key Dependencies & Rationale

Package	Version	Rationale
`react-native (CLI)`	0.82	Cross-platform iOS + Android without Expo overhead or limitations
`@unomed/react-native-matrix-sdk`	0.9	Matrix Rust SDK via UniFFI — only viable E2EE path in React Native
`typescript`	5.8	Strict typing essential for complex async event-driven flows
`zustand`	5.0	Lightweight state; SDK remains source of truth, Zustand mirrors it
`@livekit/react-native`	2.9	Self-hosted WebRTC voice/video — no third-party calling service
`react-native-keychain`	10.0	OS-level secure session storage without any custom backend
`@notifee/react-native`	9.1	E2EE-safe push notifications — Firebase explicitly excluded
`react-native-callkeep`	4.3	Native call UI (CallKit on iOS, IncomingCall on Android)
`react-native-mmkv`	3.3	Synchronous key-value storage for fast app state reads
`react-native-fs`	2.20	File system paths required by the Matrix SDK for cache & media

What This Project Proved

Decision-making under uncertainty

The hardest part was not the code — it was making the right call when every option carried real risk. Choosing an undocumented, community-maintained SDK over the "official" path required weighing technical risk against timeline risk and trusting that a sound architectural foundation (the Rust SDK) would hold. It did. The key insight: treat documentation absence as a solvable engineering problem, not a blocker.

AI-assisted exploration of undocumented systems

Feeding raw UniFFI-generated FFI files into an AI model to produce structured API documentation is a workflow I would use again without hesitation. It compressed a week of trial-and-error into a few hours and produced documentation accurate enough to plan the full implementation upfront — significantly reducing dead-end risk throughout the build.

What I would do differently

I would invest more time in automated integration tests against the Matrix homeserver in Week 1. Several late-week debugging sessions were harder to trace than necessary because the event-driven SDK surface had not been exercised with regression tests. Earlier test coverage of the message flow and key exchange paths would have saved meaningful time in Week 4.

Fast learning as a force multiplier

This project required learning the Matrix protocol, UniFFI bindings, LiveKit signalling, and iOS CallKit simultaneously — all within a 4-week window. Leaning on AI agents for rapid concept orientation (not code generation) while verifying results against official Rust SDK documentation proved to be the highest-leverage approach for accelerated onboarding to an unfamiliar stack.

> *"This project taught me that the most valuable skill is not knowing the answer — it is building a fast, reliable path to the answer when no one has documented it yet."*
>
> — Muhammad Hassan · hassanprodeveloper.com

Let's build something together

I specialize in React Native, mobile architecture, and solving hard infrastructure problems. Available for new projects and collaborations.

hassanprodeveloper.com
hassanprodeveloper@gmail.com
linkedin.com/in/hassanprodeveloper
github.com/hassanprodeveloper
+92 339 3700010