CASE STUDY · REACT NATIVE · iOS & ANDROID

Quran AI

Building a Spiritually Intelligent Quran Companion — From Figma to Production

Role	Lead React Native Developer (Frontend, Sole Engineer)
Platform	iOS & Android — React Native Expo
Timeline	MVP — ~3 Months
Status	Shipped · In Production
Tech Stack	React Native, Expo SDK 54, TypeScript, Zustand, SQLite, Supabase, RevenueCat

Muhammad Hassan
Senior React & React Native Developer
hassanprodeveloper.com · github.com/hassanprodeveloper · linkedin.com/in/hassanprodeveloper

■ OVERVIEW

Project Summary

Quran AI is a full-featured Islamic super-app for iOS and Android built with React Native Expo. The concept was delivered as an MVP by translating a complete Figma design into a production-ready application. As the sole React Native engineer, I owned 100% of the frontend implementation — from architecture and component system to native integrations, AI streaming, and App Store submission.

The app serves Muslim users with a unified spiritual companion: AI-powered Q&A from Quran and Hadith, a full Quran reader with offline support, real-time prayer times, daily streak habit tracking, curated reading series, and a unique panic button for instant spiritual support in moments of distress.

11+	2	100%	1
Core Features	**Platforms**	**Offline Quran**	**RN Engineer**
Shipped in MVP	iOS & Android	No internet needed	Sole frontend owner

■ P — PROBLEM

What Was Broken

Existing Quran apps are either static read-only digital copies or generic chatbots with no grounding in Islamic scripture. There was no product that combined authoritative Quran/Hadith access, contextual AI answers with source verification, daily habit-building tools, and emotional support — all in a single polished mobile experience.

The client's vision: a spiritually intelligent companion that strengthens daily practice and provides immediate Quranic guidance when users need it most. The challenge wasn't simply building features — it was doing so in a way the Muslim community would trust. Theological accuracy was non-negotiable.

Core Problems to Solve

- No single app combined Quran reading, AI-powered guidance, and habit tracking
- AI answers about Islamic scripture required verified Quranic references per response
- Target markets have unreliable connectivity — offline access was mandatory
- App Store and Play Store billing compliance for subscription products is strict
- Sole RN engineer with MVP timeline pressure and a parallel admin dashboard

■ S — SITUATION

Real-World Constraints

Solo Ownership	I was the only React Native engineer. Every screen, component, native integration, animation, and performance concern was mine to design and implement.
Theological Trust	Religious content demanded verified citations on every AI answer. Users needed to cross-check each response directly from the Quran reader.
Dual Platform	iOS and Android simultaneously via Expo, with native modules for location, audio, and push notifications, each with platform-specific edge cases.
Offline-First	Full Quran dataset with translation had to be persisted locally in SQLite, accessible with zero network dependency after first launch.
Subscription Billing	App Store and Play Store have strict rules on in-app purchases. RevenueCat abstracted billing, but receipt validation and paywall UX had to be airtight.
MVP Timeline	Features shipped iteratively under deadline pressure with a parallel admin dashboard (Next.js) being built simultaneously by the backend team.

■ T — THINKING

Decisions & Trade-offs

Every architectural decision was a deliberate trade-off. Below are the five most consequential choices made during development and the reasoning behind each.

01

Atomic Design System (Atoms → Molecules → Organisms → Templates)

Rationale: Structured components following atomic design principles from day one. With a full Figma design system and an aggressive MVP timeline, this gave a shared vocabulary with the designer and prevented the component sprawl that kills React Native projects.

Trade-off: *Upfront investment in folder structure and conventions, but paid dividends with every new screen added — adding screens after the first three took hours, not days.*

02

Zustand Over Redux or Context API

Rationale: The app has multiple independent state slices: Quran progress, prayer times, streaks, subscription status, and AI conversation history. Redux would have been verbose overkill. Context API causes unnecessary re-renders across unrelated trees. Zustand gave minimal boilerplate, excellent TypeScript support, and fine-grained subscriptions.

Trade-off: *Less ecosystem tooling than Redux, but with a team of one, that was entirely irrelevant. Zustand's simplicity was the right fit for the project scale.*

03

Streaming AI Responses to the UI

Rationale: Consumed the backend streaming API and rendered tokens in real-time using a custom hook. Streaming eliminates the perception of latency for long AI answers — critical when users are emotionally engaged, especially on the panic button feature.

Trade-off: *More complex streaming state management versus a simple fetch-and-display pattern. The UX improvement justified the extra engineering effort.*

04

expo-sqlite for Offline Quran Persistence

Rationale: The full Quran dataset (Arabic + translation) is too large for AsyncStorage or MMKV. expo-sqlite provides a structured, queryable local database. On first launch, data is downloaded and persisted — subsequent launches read from SQLite with no network dependency.

Trade-off: *Longer first-launch time offset against instant subsequent access. Mitigated with a smooth onboarding progress indicator.*

05

adhan npm Package for Prayer Times (Not a Backend API)

Rationale: Prayer time calculation is a well-defined mathematical problem based on solar position and GPS coordinates. The adhan library provides instant, offline-capable times with zero API dependency. Device location via expo-location feeds the calculation — no round-trip required.

Trade-off: *No server-side tracking of prayer data, but perfect reliability, zero latency, and zero cost. A clear win for a calculation-based problem.*

■ A — ACTION

What I Built

Core Features Delivered

AI Quran & Hadith Assistant	Streaming chat interface powered by a RAG backend (OpenAI + Quran corpus). Every AI response renders embedded Quranic verse references as tappable citations for user verification.
Panic Button	User describes their feelings; the AI agent returns a personalized Dua, Quranic reassurance, and relevant Hadith — creating immediate spiritual connection in moments of distress. Most loved feature by the client.
Quran Reader + Audio Player	Full Quran with Lateef Arabic font, translation overlay, and verse-level audio playback via expo-av. Fully offline via SQLite.
Daily Streak System	Four-step daily task loop (read, listen, reflect, track prayer) that must be fully completed before a streak day is counted. Built to resist habit-gaming — partial completion does not count.
Prayer Tracker	Live prayer countdowns using adhan + device GPS. Updates with location changes. Users can log completed prayers and view weekly adherence.
RevenueCat Subscription	Yearly subscription with 7-day free trial. Full SDK integration: entitlement checks, paywall rendering, receipt validation, and graceful free-tier fallback.
Daily Reminder Notifications	Scheduled local notifications via expo-notifications. Users configure preferred reminder times for daily task completion.
Series Reading	Curated multi-day reading programs (7-day Ramadan Reflections, 10-day Stories of the Prophets). Content managed from the admin dashboard.

Project Architecture

Folder Structure (Molecular Design)

```
src/
  components/atoms/ — molecules/ — organisms/ — templates/
  navigation/ — screens/ — hooks/ — services/ — stores/ — utils/
```

Technology Stack

Core	React Native · Expo SDK 54 · TypeScript
State	Zustand v5
Data & Offline	expo-sqlite · MMKV · AsyncStorage
Backend	Supabase · Axios · REST API (Streaming SSE)

Monetisation	RevenueCat (iOS + Android)
Islamic Tools	adhan (prayer times)
Media	expo-av (Quran audio)
Native	expo-location · expo-notifications
Navigation	React Navigation (native stack + bottom tabs)
Animation	React Native Reanimated 4
Performance	Shopify FlashList
Typography	Lateef (Arabic) · Inter (UI)

■ R — RESULTS

Outcomes & Impact

11+	**2**	**100%**
Features Shipped	**Platforms**	**Offline Quran**
Single MVP release	iOS & Android	Full corpus locally

Key Qualitative Outcomes

- Sole React Native engineer — owned architecture through App Store and Play Store submission
- Streaming AI response UX eliminated perceived latency on all AI-powered features
- Atomic component system allowed new screens to be added in hours, not days
- RevenueCat integration passed App Store and Play Store review on the first submission
- Panic button feature identified as the highest-impact differentiator by the client
- SQLite offline persistence made the app viable in low-connectivity markets
- Quranic verse citation on every AI answer built user trust in a sensitive domain

■ REFLECTION

What I Learned & What I Would Improve

Key Learnings

Streaming UX is worth the engineering complexity. SSE streaming for AI responses required extra effort, but the improvement in perceived performance and emotional UX — especially on the panic button — was significant. The same choice would be made again.

Atomic design pays off under time pressure. The upfront investment in component hierarchy meant every screen after the third was dramatically faster to build and simpler to maintain.

Faith-based apps require trust-by-design. Showing Quranic verse references on every AI answer was essential for user trust in a sensitive domain. Feature decisions in religious apps carry more weight than typical consumer apps.

What I Would Improve

- Add E2E tests (Detox) for the subscription flow and AI chat — the highest-risk user journeys
- Implement error boundary strategy per feature module rather than a single global boundary
- Abstract the AI streaming hook into a standalone utility package for reuse across projects
- Move prayer time notification scheduling to a background task instead of recalculating on app open