IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, May 2026
Impact Factor: 8.2

Taxi Manager: Firebase-Powered Real-Time Backend Architecture for Cross-Platform Fleet Management

Nitish Kaushik[1], Rajendra Singh[2]
[1] Department of Computer Science and Engineering
[2] Dean, Department of Computer Science and Engineering,
Raffles University, Neemrana, Rajasthan, India
nitish2786@gmail.com[1],rajendra.singh@rafflesuniversity.edu.in[2]

Abstract: *Efficient real-time data synchronization and secure backend architecture are critical requirements for mobile fleet management applications operating in low-connectivity environments. This paper presents the backend architecture and cloud infrastructure design of "Taxi Manager," a cross-platform mobile fleet management application for small taxi operators in India. The proposed system leverages Firebase as a complete Backend-as-a-Service (BaaS) platform, integrating Firestore for real-time NoSQL data synchronization, Firebase Authentication for phone-number OTP-based passwordless login, Firebase Storage for driver document management, and Firebase Cloud Functions for serverless push notifications and automated nightly financial report aggregation. A dual-layer Role-Based Access Control (RBAC) architecture is implemented, combining Expo Router client-side routing with Firestore server-side security rules to enforce strict data isolation between Owner and Driver user roles. A hybrid data fetching strategy — using real-time Firestore snapshot listeners for high-frequency data and one-time fetch operations for reference data — minimizes Firestore read costs while maintaining sub-500 millisecond update propagation. The complete system operates within Firebase's free Spark tier, demonstrating zero-cost backend operability for small fleet operators. The application is actively deployed at GM Taxi Service, Behror, Rajasthan. Evaluation across 15 test cases yields a 100% pass rate with booking creation latency under one second and push notification delivery within 2–3 seconds.*

Keywords: Firebase, Firestore, Cloud Functions, Real-Time Synchronization, Role-Based Access Control, Backend-as-a-Service, React Native, Fleet Management, Push Notifications, Serverless Architecture

I. INTRODUCTION

The small taxi fleet management segment in India represents a significantly underserved market for digital technology. While large-scale ride-hailing platforms such as Ola and Uber dominate the aggregator market, they do not address the operational needs of independent fleet owners who maintain direct relationships with their drivers and customers, manage bookings manually, and track finances through physical ledgers. According to the Ministry of Road Transport and Highways, India had over 5 million registered commercial vehicles as of 2023, a substantial proportion of which operate under small independent owners [1].

The backend architecture of a mobile fleet management application must satisfy several competing requirements: real-time data propagation to ensure booking status changes are visible to both owners and drivers instantly; strict data isolation to prevent drivers from accessing other drivers' data or owner-only financial records; low operational cost to remain accessible to small operators with limited capital; and resilience to variable network connectivity common in Tier 2 and Tier 3 Indian cities.

DOI: 10.48175/IJARSCT-35767

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 10, May 2026 Impact Factor: 8.2

Traditional server-based backend architectures require dedicated infrastructure, DevOps expertise, and recurring hosting costs, all of which are barriers for small operators and student developers building solutions for this market. Firebase, Google's Backend-as-a-Service platform, eliminates these barriers by providing managed, scalable infrastructure accessible through a JavaScript SDK with a generous free tier [2].

This paper focuses specifically on the backend architecture and cloud infrastructure design of Taxi Manager, a cross-platform mobile fleet management application. The frontend application is built using Expo and React Native with

TypeScript. The primary contributions of this paper are:

- A Firestore data model designed to minimize read operations through strategic denormalization
- A dual-layer RBAC architecture combining client-side Expo Router grouped layouts with server-side Firestore security rules
- A hybrid real-time and one-time fetch strategy that balances responsiveness with cost efficiency
- A serverless Cloud Functions architecture for push notifications and automated financial aggregation
- Demonstrated zero-cost production deployment on Firebase's Spark free tier

II. LITERATURE REVIEW

A. Backend-as-a-Service Platforms

Backend-as-a-Service platforms have transformed mobile application development by providing managed cloud infrastructure that eliminates server provisioning and maintenance overhead. Garg and Verma (2021) evaluated Firebase against traditional REST server backends for real-time mobile applications, finding that Firestore's listener model eliminated polling complexity and reduced average data staleness from 30 seconds to under 500 milliseconds [3]. AWS Amplify offers comparable features but requires more complex configuration and SQL knowledge for Supabase, making Firebase the preferred choice for React Native developers [4].

B. Real-Time Data Synchronization

Gupta et al. (2023) compared polling-based and event-driven synchronization architectures in mobile fleet management, finding that event-driven architectures reduced bandwidth consumption by 40% while delivering near-instant data updates [5]. Firestore's persistent WebSocket connection model pushes document changes to all subscribed clients without requiring clients to poll at intervals. This is critical for fleet management where booking status, driver availability, and payment records can change at any moment.

C. Role-Based Access Control in Cloud Applications

Johnson and Williams (2022) established a three-tier taxonomy for RBAC in cloud-native mobile applications: presentation-layer only, data-layer only, and dual-layer enforcement [6]. Their study concluded that dual-layer RBAC — enforcing access control at both the client UI and the server data layer — is the only approach that remains secure against both accidental navigation errors and deliberate API manipulation. Firestore security rules provide a declarative, server-side enforcement mechanism that operates independently of the client application.

D. Serverless Computing

Serverless computing models, as reviewed by Baldini et al. (2017), eliminate the need to provision or manage servers by executing code in response to events and scaling automatically [7]. Firebase Cloud Functions implement this model for mobile backends, triggering server-side logic in response to Firestore document writes, HTTP requests, or scheduled Pub/Sub events. This is particularly suitable for infrequent, event-driven operations such as push notifications and nightly report aggregation.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, May 2026
Impact Factor: 8.2

E. NoSQL Data Modeling

Fowler and Sadalage (2012) established that NoSQL document databases benefit from strategic denormalization — storing redundant data in documents to avoid cross-collection joins — to optimize read performance [8]. In Firestore, where each document read incurs a cost against the free tier quota, minimizing the number of reads required per screen render is a critical design objective.

III. SYSTEM ARCHITECTURE AND BACKEND DESIGN

A. Overall Architecture

Taxi Manager follows a three-tier architecture. The Presentation Layer is an Expo/React Native application communicating with Firebase exclusively through the Firebase JavaScript SDK. The Application Layer comprises Firebase Authentication, Firestore, Firebase Storage, and Cloud Functions. The Data Layer consists of Firestore document collections and Firebase Storage buckets.

This architecture eliminates the need for a custom API server entirely. All business logic that must execute server-side is implemented as Cloud Functions triggered by Firestore events or Pub/Sub schedules.

B. Firestore Data Model

The Firestore data model is designed around two principles: minimizing read counts per screen render and enforcing clean ownership boundaries for RBAC. Each document collection stores an ownerId field that serves as the primary access control anchor in Firestore security rules.

The data model consists of six primary collections as described in Table I.

Table I: Firestore Collections and Key Fields

Collection	Key Fields	Purpose
users	uid, name, phone, role, ownerId, fcmToken	User profiles and role assignment
drivers	uid, ownerId, name, salary, paymentDay	Driver profiles and salary config
taxis	id, ownerId, regNumber, make, model	Fleet vehicle records
bookings	id, ownerId, driverId, pickup, dropoff, stops[], totalFare, receivedAmount, balance, paymentStatus	Complete booking lifecycle
payments	id, bookingId, driverId, ownerId, amount, type, category, timestamp	Individual payment and expense entries
reports	ownerId, date, totalIncome, totalExpenses, netProfit, expenseBreakdown{}	Pre-aggregated daily financial data

Strategic denormalization is applied in the bookings collection, which stores driverName and regNumber as redundant fields to avoid additional reads to the drivers and taxis collections when rendering booking cards.

C. Dual-Layer RBAC Architecture

Security enforcement operates at two independent layers. The first layer uses Expo Router's grouped layout system. Owner screens reside under the (owner)/ route group and driver screens under the (driver)/ route group. An AuthContext provider reads the authenticated user's role from Firestore on every app launch and enforces routing to the correct group, preventing cross-role UI access.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, May 2026
Impact Factor: 8.2

The second layer uses Firestore server-side security rules. These rules are evaluated by Firebase's servers on every read and write operation, independently of the client application. The core rules enforce that an owner can only read or write documents where the ownerId field equals their authenticated UID, and a driver can only read bookings where the driverId field equals their UID and can only update the payment-related fields of those bookings.

A key security principle is that these two layers are independent. Even if a malicious actor modifies the client application or constructs direct Firestore API calls, the server-side rules reject any unauthorized operation. This dual-layer approach corresponds to the highest security tier identified by Johnson and Williams (2022) [6].

D. Hybrid Fetch Strategy

A hybrid data fetching strategy is employed to balance real-time responsiveness with Firestore read cost efficiency. Real-time Firestore snapshot listeners (onSnapshot) are used only for high-frequency, user-facing data: active booking status on the owner's bookings screen, and assigned trips on the driver's dashboard. These listeners maintain a persistent WebSocket connection and propagate updates to the UI in under 500 milliseconds.

One-time fetch operations (getDocs) are used for reference data that changes infrequently: the fleet vehicle list, driver roster, and historical financial reports. This hybrid approach reduces daily Firestore read counts by an estimated 60–70% compared to using snapshot listeners universally, which is critical for staying within the 50,000 free daily reads on Firebase's Spark tier.

E. Cloud Functions Design

Two Cloud Functions implement server-side logic that cannot safely execute on client devices.

The first function is a Firestore onCreate trigger on the bookings collection. When a new booking document is created, the function reads the assigned driver's FCM token from their user profile document and sends a push notification via Firebase Cloud Messaging containing the booking details. End-to-end notification delivery from booking creation to device receipt averages 2–3 seconds.

The second function is a scheduled Pub/Sub trigger configured to execute at 23:59 IST daily using a cron expression. It queries all payment documents for the current date, aggregates totals by ownerId into income, expense, and category breakdowns, and writes pre-computed report documents to the reports collection. This pre-aggregation approach ensures the Reports screen loads financial data in under 800 milliseconds through a single document read, avoiding expensive on-demand aggregation queries across potentially hundreds of payment documents.

F. Authentication Architecture

Firebase Phone Authentication is used exclusively, eliminating usernames and passwords. The OTP flow requires users to enter their phone number, receive a 6-digit OTP via SMS, and verify within 60 seconds. OTPs expire after a single use, preventing replay attacks. Firebase Auth sessions are managed with automatic token refresh using JSON Web Tokens, ensuring long-lived sessions remain secure without requiring re-authentication.

All communication between the mobile client and Firebase services is encrypted using TLS 1.3. Firebase Storage enforces authentication on all file operations, preventing unauthorized access to driver documents and booking receipts.

G. Firebase Configuration and Environment Security

Firebase configuration values including API keys and project identifiers are stored as Expo environment variables and injected at build time, ensuring they are never hardcoded in source files committed to version control. The Firebase initialization module uses a singleton pattern to prevent re-initialization during Expo's hot-reload development cycle.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, May 2026
Impact Factor: 8.2

IV. RESULTS AND EVALUATION

A. Test Cases

Fifteen test cases were designed to cover all critical backend flows including authentication, booking creation with notification delivery, real-time payment updates, financial report aggregation, RBAC enforcement, and bilingual caching. All 15 test cases passed, yielding a 100% pass rate. Key backend-focused results are shown in Table II.

Table II: Selected Backend Test Case Results

TC	Scenario	Expected Result	Result
TC-01	Owner OTP Login	Role resolved; routed to owner dashboard	PASS
TC-03	Booking Creation	Firestore write + FCM notification in 2s	PASS
TC-04	Multi-Stop Booking	All stops persisted in bookings document	PASS
TC-05	Partial Payment Recording	Atomic batch: payment + booking balance updated	PASS
TC-06	Full Payment Recording	Balance = 0; paymentStatus set to Paid	PASS
TC-07	Financial Reports	Pre-aggregated report document read correctly	PASS
TC-13	Date Filter on Bookings	Firestore compound query returns correct subset	PASS

B. Performance Metrics

Backend performance metrics collected during testing on a 4G LTE connection are presented in Table III.

Table III: Backend Performance Metrics

Metric	Measured Value
OTP delivery time	< 5 seconds
Booking creation (Firestore write)	< 1 second
Push notification delivery	2–3 seconds
Real-time status update (snapshot)	< 500 ms
Reports screen load (pre-aggregated)	< 800 ms
Payment batch write (atomic)	< 600 ms
Translation cache hit (AsyncStorage)	< 10 ms

C. Firebase Free Tier Usage Analysis

Table IV presents estimated daily Firestore operation counts against the Firebase Spark free tier limits for the production deployment at GM Taxi Service.

Table IV: Firestore Usage vs. Free Tier Limits

Operation	Daily Count (Est.)	Free Tier Limit	Usage %
Document reads	~2,000	50,000/day	4%
Document writes	~200	20,000/day	1%
Document deletes	~10	20,000/day	<1%
Storage used	~50 MB	1 GB total	5%
Cloud Function calls	~50	2,000,000/month	<1%

The analysis confirms that the complete backend infrastructure operates at zero monthly cost within Firebase's Spark free tier, making it financially accessible to small fleet operators with limited capital.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, May 2026
Impact Factor: 8.2

D. Security Validation

RBAC enforcement was validated by attempting direct Firestore API calls from a driver-authenticated session targeting owner-only collections. All such attempts were rejected by Firestore security rules with PERMISSION_DENIED errors, confirming that server-side enforcement operates independently of the client application. OTP replay attack prevention was validated by attempting to reuse an expired OTP, which was correctly rejected by Firebase Authentication.

V. CONCLUSION

This paper presented the Firebase-powered backend architecture of Taxi Manager, a cross-platform mobile fleet management application targeting small taxi operators in India. The key architectural contributions include a denormalized Firestore data model that minimizes read operations per screen render, a dual-layer RBAC system combining Expo Router client-side routing with Firestore server-side security rules, a hybrid snapshot listener and one-time fetch strategy that reduces daily read counts by an estimated 60–70%, and a serverless Cloud Functions architecture delivering push notifications within 2–3 seconds and pre-aggregating financial reports nightly for sub-800 millisecond screen load times.

The complete backend operates at zero monthly cost on Firebase's Spark free tier, with production usage estimated at under 5% of free tier limits for the current deployment scale. The application is actively deployed at GM Taxi Service, Behror, Rajasthan, providing real-world validation of the architecture's reliability and performance.

Future backend enhancements include real-time GPS location tracking using Firestore geospatial queries, in-app payment gateway integration with Razorpay webhooks processed by Cloud Functions, an offline-first architecture using Firestore's native offline persistence for low-connectivity environments, and automated document expiry alerts via weekly scheduled Cloud Functions.

ACKNOWLEDGMENT

I would like to sincerely thank Rajendra Singh, Dean, Department of Computer Science and Engineering, Raffles University, for his valuable guidance, continuous support, and encouragement throughout this project.

I am also grateful to the Department of Computer Science and Engineering, Raffles University, for providing the academic support and necessary environment to complete this research work.

REFERENCES

1. Ministry of Road Transport and Highways, "Road Transport Yearbook 2022–23," Government of India, 2023.
2. Google Firebase, "Firebase Documentation: Firestore, Authentication, Storage, Cloud Functions," 2024. [Online]. Available: https://firebase.google.com/docs
3. R. Garg and S. Verma, "Firebase as Backend-as-a-Service for Real-Time Mobile Applications: A Performance Analysis," International Journal of Engineering and Technology, vol. 9, no. 4, pp. 112–118, 2021.
4. Expo Inc., "Expo SDK 52 Documentation," 2024. [Online]. Available: https://docs.expo.dev
5. R. Gupta, A. Sharma, and N. Patel, "Event-Driven vs. Polling Architectures in Mobile Fleet Management Systems," Journal of Mobile Computing and Applications, vol. 15, no. 2, pp. 88–101, 2023.
6. A. Johnson and T. Williams, "Role-Based Access Control Patterns in Cloud-Native Mobile Applications," IEEE Access, vol. 10, pp. 45612–45624, 2022.
7. I. Baldini et al., "Serverless Computing: Current Trends and Open Problems," in Research Advances in Cloud Computing, Springer, pp. 1–20, 2017.
8. M. Fowler and P. Sadalage, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, 2012.
9. P. Nawrocki, K. Wrona, M. Marczak, and M. Jarosz, "A Comparison of Native and Cross-Platform Frameworks for Mobile Applications," Computer, vol. 54, no. 3, pp. 18–27, 2021.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 10, May 2026
Impact Factor: 8.2

10. V. Bidve, P. Sarkar, and R. K. Tripathy, "A Comparative Study of Cross-Platform Mobile Development Frameworks," International Journal of Computer Applications, vol. 184, no. 12, pp. 1–7, 2022.
11. Meta Platforms Inc., "React Native: Build Mobile Apps with JavaScript," 2023. [Online]. Available: https://reactnative.dev
12. NativeWind, "NativeWind: Tailwind CSS for React Native," 2024. [Online]. Available: https://www.nativewind.dev
13. A. Jain and B. Soni, "Technology Adoption Barriers among Small Taxi Fleet Operators in Tier 2 Indian Cities," Journal of Transport and Technology Policy, vol. 12, no. 3, pp. 45–58, 2020.
14. Google Firebase Cloud Messaging, "Firebase Cloud Messaging Documentation," 2024. [Online]. Available: https://firebase.google.com/docs/cloud-messaging