Nishant Raj, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Lost and Found Item System: A Cloud-Based Web Platform for Institutional Item Management Using React.js and Firebase

Nishant Raj, Ayush Kamni, Palak Saini, Ajay Bora, Mrs. Shivani Chauhan
Department of Computer Science and Engineering,
School of Technology Quantum University, Roorkee, Uttarakhand, India

Abstract- Managing lost and found items in institutional settings such as universities presents persistent operational challenges. This paper presents the Lost and Found Item System (LFIS), a cloud-based Single Page Application (SPA) developed using React.js v19, Firebase Authentication, Cloud Firestore, and the Cloudinary CDN. The platform enables authenticated users to submit image-linked lost and found reports, execute composite multi-parameter queries, and receive real-time listing updates through Firestore's onSnapshot listeners. An administrative dashboard provides role-based moderation for institution staff. System Usability Scale (SUS) evaluation with 30 participants yielded a combined mean score of 81.4 (Grade B). Performance benchmarking on a 200-record dataset demonstrated first contentful paint averaging 1.74 s and composite Firestore queries averaging 1.08 s. Against the manual baseline, item logging time decreased by approximately 71% and record retrieval time by approximately 90%. The serverless architecture eliminates local infrastructure maintenance, and the design generalizes to equipment tracking, visitor management, and similar institutional resource management contexts.

Keywords— lost and found management, cloud web application, Firebase, React.js, Cloudinary, NoSQL database, real-time synchronization, institutional item recovery, SPA architecture, role-based access control.

I. INTRODUCTION

Misplaced belongings are a daily occurrence within university campuses, transit terminals, and workplace environments. Mobile phones, identity documents, wallets, and laptops are regularly reported lost in institutional settings, yet the majority of organizations still depend on a front-desk register or a static notice board to manage recovery. These mechanisms share a fundamental weakness: they are inaccessible remotely, cannot be searched by category or keyword, provide no visual evidence for identification, and produce records that degrade in accuracy over time [1].

Digital lost and found platforms have attracted growing research attention as mobile web technologies matured. Prior studies confirm that digitalization reduces reporting and retrieval time substantially [1],[5]. Research into image-based matching demonstrates accuracy gains when photographs accompany item descriptions [2],[3]. However, a gap remains between conceptual proposals and deployable systems: most published prototypes fail to combine image-linked reporting, composite search, real-time synchronization, role-based administration, and serverless scalability in a single accessible application.

This paper presents the Lost and Found Item System (LFIS), a web application developed for Quantum University, Roorkee, India. The system integrates React.js v19 for the frontend, Firebase

Authentication for identity management, Cloud Firestore as the real-time NoSQL database, and Cloudinary CDN for image storage. The serverless architecture requires no self-managed backend server, enabling zero-maintenance deployment with horizontal scalability. Primary contributions include: (i) a modular React.js SPA supporting authenticated

item reporting on any device; (ii) composite multi-field Firestore search with sub-two-second query latency; (iii) real-time listing synchronization via snapshot listeners; (iv) an image management pipeline decoupling binary assets from the primary database; (v) a role-based administrative dashboard; and (vi) structured performance, usability, and security evaluation.

II. RELATED WORK

1. Web-Based Digital Systems

Pede et al. [1] demonstrated that a browser-accessible digital platform reduces average item reporting time by more than half relative to a manual front-desk register, though absent image support and real-time synchronization remained limiting factors. Tan and Chong [2] identified image support as the strongest predictor of successful item recovery in a university-focused portal. Singh et al. [8] identified three recurring deficiencies in institutional web applications: absent real-time synchronization, inadequate access control, and non-responsive interfaces.

2. AI-Augmented and Mobile-Centric Systems

Zhou et al. [3] introduced LostNet, a convolutional neural network matching lost-item photographs to found-item photographs without keyword queries.

The system imposes significant GPU infrastructure requirements and minimum image quality constraints rarely achievable in real-world environments. CampusTrace [7] provided a polished campus experience for Android but excluded iOS and required installation, measurably limiting adoption. Meshram et al. [6] demonstrated that serverless cloud backends most effectively address scalability bottlenecks while reducing operational burden.

Table I summarizes comparative feature positioning across reviewed systems.

Table 1. Comparative Analysis of Existing Lost and Found Systems

System	Real-Time Sync	Image Support	Admin Dashboard	Serverless
Traditional (Manual)	None	None	None	N/A
Pede et al. [1]	No	No	Basic	No
Tan & Chong [2]	No	Yes	Minimal	No
LostNet [3]	No	CNN-Based	None	No
CampusTrace [7]	No	Yes	Limited	No
Meshram et al. [6]	Partial	Partial	None	Partial
Proposed LFIS	Firestore <500ms	Cloudinary CDN	Role-Based	Firebase

III. PROBLEM STATEMENT

Analysis of existing literature and audit of current practices at Quantum University identify four structural deficiencies: (1) absence of real-time synchronization, causing a found-item report to be invisible to a searching user for hours, degrading recovery probability; (2) lack of image support significantly reducing item identification success [2]; (3) single-field keyword search is insufficient—users need simultaneous filtering by category, location, date, and resolution status; and (4) dependence on locally hosted servers or platform-specific applications, imposing maintenance burdens and restricting accessibility.

Within the context of Quantum University, students have no central digital registry; finders create paper records at the security desk that cannot be searched remotely; and communication between finder and owner relies entirely on informal social media groups. The resulting problem statement: institutional lost and found management requires a web platform providing image-linked report submission, composite search with sub-two-second latency, real-time synchronization, structured finder–owner contact, role-based administrative oversight, and a serverless backend accessible from any browser without installation.

IV. SYSTEM OVERVIEW

The LFIS operates across four layers communicating exclusively over HTTPS/TLS. The User Layer comprises any device with a modern web browser. The Frontend Layer is a React.js SPA bundled by Vite and served from Firebase Hosting, encompassing component rendering, client-side routing, and application state management. The Backend Services Layer consists of Firebase Authentication, Cloud Firestore, and the Cloudinary Upload API, each accessed directly from the browser via respective SDKs. The Data and Media Layer encompasses Firestore document collections and Cloudinary CDN-hosted image assets. This four-layer separation eliminates any custom server process—the entire application runs on managed cloud services.

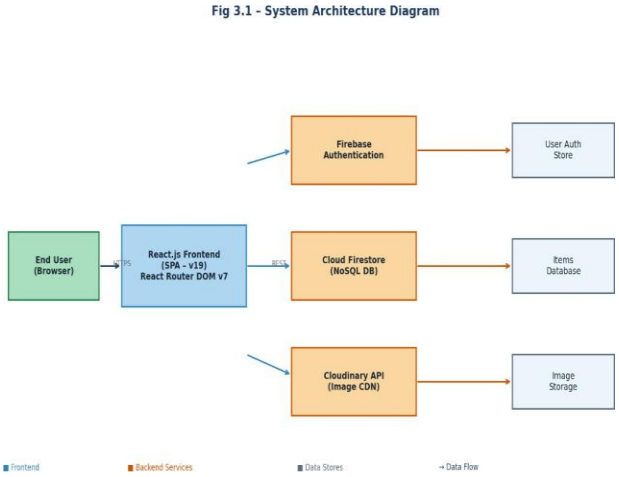

Fig. 1. System Architecture of the Lost and Found Item System

The technology stack selection was governed by three non-negotiable criteria: no self-managed server infrastructure, real-time data delivery, and component-based development efficiency. React.js v19 was selected for its declarative component model and virtual DOM efficiency. Firebase was chosen as the unified backend because its Authentication, Firestore, and Security Rules integrate natively within a single managed platform. Cloudinary was adopted to maintain database document compactness and CDN-accelerated image delivery. Table II presents the complete stack with selection rationale.

Table 2. Technology Stack and Selection Rationale

Technology	Role	Rationale
React.js v19	Frontend SPA Framework	Declarative component model, virtual DOM, hooks-based state
Firebase Authentication	Identity Management	Managed OAuth + email/password; JWT issuance and session handling
Cloud Firestore	Primary Database	Real-time listeners, serverless NoSQL, composite index support
Cloudinary API	Image Storage & CDN	CDN delivery, auto-compression; decouples binary assets from DB
React Router DOM v6	Client-Side Navigation	Protected route HOC; no full-page reload on navigation
Vite	Build Tool	Sub-second hot module replacement; optimized production bundles
CSS3 Grid / Flexbox	Responsive Layout	Precise responsive control without CSS framework dependency

V. METHODOLOGY

1. Development Approach

Development followed a six-phase Agile SDLC: requirements analysis, system design, technology selection, iterative sprint-based development, structured testing, and cloud deployment. Agile was selected over sequential models because it accommodates requirement evolution, produces demonstrable working software at each sprint, and enables early detection of integration issues. Requirements were gathered through semi-structured interviews with five administrative staff and a structured survey of forty undergraduate students. The survey revealed three dominant pain points: inability to access records remotely (87% of respondents); absence of visual evidence making identification unreliable (79%); and no notification mechanism when a matching item appears (91%).

2. Development Sprints

Development ran across four two-week sprints. Sprint 1 targeted requirements finalization,

architecture decisions, and Firebase project scaffolding. Sprint 2 delivered the authentication module, user dashboard, and Firestore database schema. Sprint 3 implemented item reporting forms, the Cloudinary image upload pipeline, and the composite search engine. Sprint 4 delivered the contact module, administrative dashboard, cross-device responsive testing, bug remediation, and deployment to Firebase Hosting.

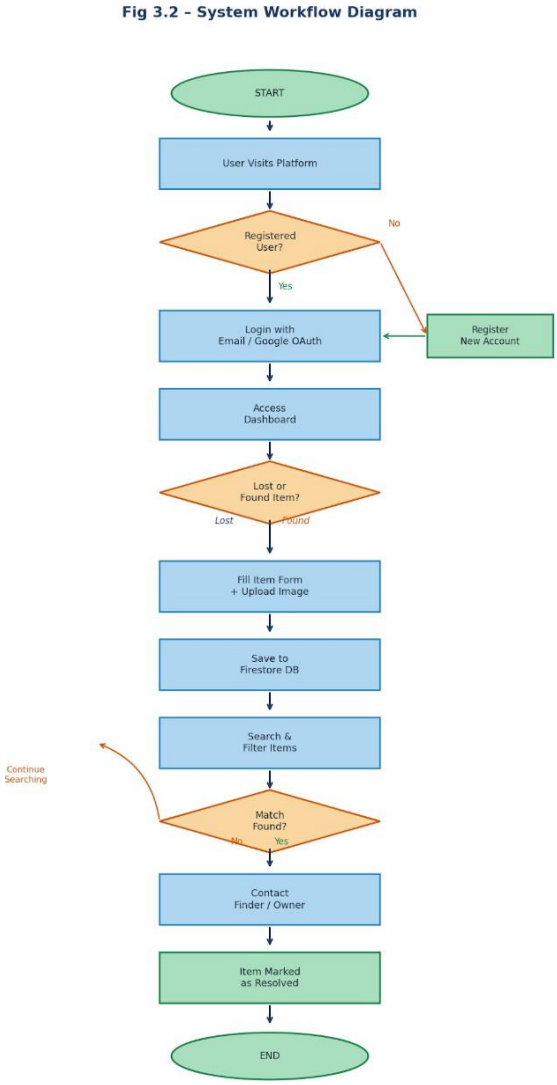

Fig. 2. System Workflow: End-to-End Item Reporting and Recovery Process

VI. SYSTEM DESIGN

1. Use Case and Data Flow

The system serves three actor roles: Guest, Authenticated User, and Administrator. Guests may browse public listings. Authenticated Users may submit lost/found reports, execute multi-parameter searches, view item details, and initiate contact requests. Administrators additionally perform moderation operations—updating status, flagging, and deleting records.

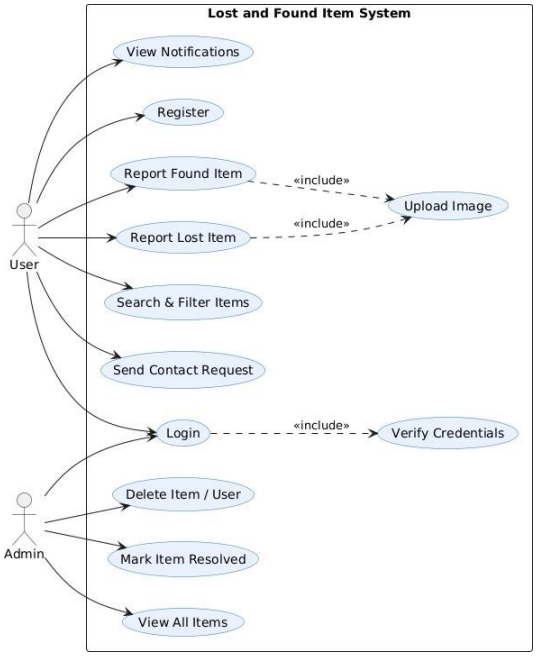

Fig. 3. Use Case Diagram Showing Actor Roles and System Interactions

2. Database Design

Cloud Firestore is organized into four top-level collections. The users collection stores one document per registered account (UID, email, displayName, role, timestamps). The items collection unifies lost and found records distinguished by a status field ('lost', 'found', or 'resolved'), storing title, description, category, location, date, imageURL, and userID. The contacts collection records finder–owner communication events. Composite indexes on (category, status, createdAt) and (status, date) support multi-parameter queries without full-collection scans. Table III presents the complete schema.

Table 3. Firestore Database Schema

Collection	Key Fields	Purpose
users	uid (PK), email, displayName, role, createdAt	User profiles, authentication linkage, admin role flags
items	itemId (PK), title, category, status, location, date, imageURL, userID (FK)	Unified lost/found records; status distinguishes type and resolution
contacts	contactId (PK), senderID (FK), receiverID (FK), itemID (FK), message	Structured finder–owner communication records
categories	id (PK), label, iconSlug	Controlled vocabulary for item classification dropdowns

VII. IMPLEMENTATION

1. Authentication Module

Two sign-in pathways are supported: Google OAuth via Firebase's signInWithPopup method, and email-and-password registration via createUserWithEmailAndPassword. On successful authentication, the module writes a user profile document to the Firestore users collection if one does not yet exist for that UID. The onAuthStateChanged listener propagates the current user through AuthContext, enabling every downstream component to react immediately to session state changes without prop-drilling through the component tree.

2. Item Reporting Module

The reporting pipeline operates in two phases. In Phase 1, the selected image file is uploaded directly to Cloudinary via HTTP POST under a scoped unsigned upload preset; Cloudinary performs MIME validation, size-limit enforcement, automatic compression, and CDN registration before returning a secure URL. In Phase 2, a single Firestore addDoc call writes the complete item document atomically, incorporating the Cloudinary URL from Phase 1. The active onSnapshot listener detects the new document within approximately 500 ms and updates the listing grid without a page reload.

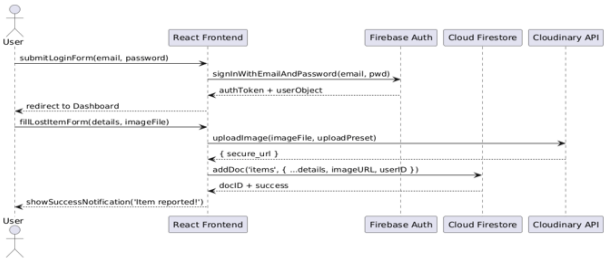

Fig. 4. Sequence Diagram (Login & Post Lost Item)

3. Search and Filter Module

The search interface accumulates active filter selections—keyword, category, status, and date range—and translates them into chained Firestore where clauses. Firestore serves results from the local persistence cache before the network response returns, producing near-instant perceived latency for cached queries. For keyword filtering, which requires partial-string matching not natively supported by Firestore, the module applies a client-side string inclusion check on the title field after the structured query resolves. Average composite query latency across 30 benchmark trials was 1.08 s, well within the 2.0 s threshold.

4. Administrative Dashboard

The admin route renders a consolidated view of all item reports, accessible only to users whose Firestore profile document contains the role: 'admin' field. Moderators can update item status to 'resolved', flag suspicious reports, and delete entries. Destructive operations use Firestore batch writes for atomicity and require explicit confirmation dialogs to prevent accidental execution. A summary statistics panel derives aggregate counts from Firestore aggregation queries without loading individual item documents, minimizing read consumption.

5. Role-Based Access Control

Access control is enforced at two independent layers. At the client layer, React Router's ProtectedRoute HOC reads the authenticated user's role from AuthContext and redirects unauthorized users to the login screen. At the server layer, Firebase Security Rules evaluate the JWT attached to each Firestore request and deny operations conflicting with the rule

set, regardless of client-side state. A user with an expired or forged token cannot write to the items collection even by manipulating client-side application state directly.

Table 4. Feature Implementation Status

Feature	Implementation Detail	Status
Google OAuth / Email Login	signInWithPopup; createUserWithEmailAndPassword	Implemented
Lost/Found Item Reporting	Structured form → Cloudinary upload → Firestore addDoc	Implemented
Composite Search & Filter	Chained Firestore where() with pre-built composite indexes	Implemented
Real-Time Listing Sync	onSnapshot listener; <500 ms update propagation	Implemented
Contact Module	Firestore contacts collection; sender/receiver UID mapping	Implemented
Admin Moderation Dashboard	Role-checked route; batch writes for flag/delete/resolve	Implemented
Responsive Multi-Device UI	CSS Grid + Flexbox + media queries; no CSS framework	Implemented
AI Image Matching	CNN-based visual similarity	Future Scope
Push Notifications	Firebase Cloud Messaging on match event	Future Scope

VIII. RESULTS AND DISCUSSION

1. Performance Benchmarking

Performance metrics were collected across 30 repeated trials on a 200-record dataset over a 50 Mbps connection. All mean values fell within defined thresholds. Composite query latency was modestly higher than single-field latency, consistent with the additional index traversal required. Upload latency exhibited the highest coefficient of variation, attributable to client-side network variation.

Table 5. Performance Benchmark Results (N = 30 Trials Per Metric)

Metric	Mean	95th Percentile	Threshold
Page Load — First Contentful Paint	1.74 s	2.31 s	< 3.0 s
Firestore Single-Field Query	0.61 s	0.89 s	< 2.0 s
Firestore Composite Query (3 Filters)	1.08 s	1.43 s	< 2.0 s
Image Upload (avg. 1.2 MB)	3.21 s	4.87 s	< 6.0 s
Admin Dashboard Load (200 records)	1.95 s	2.64 s	< 4.0 s
onSnapshot Update Propagation	0.48 s	0.72 s	< 1.5 s

2. Usability Evaluation

SUS evaluation was conducted with 30 participants: 20 undergraduate students and 10 administrative staff. Each participant completed four standardized tasks and subsequently completed the standard 10-item SUS questionnaire. Mean SUS scores were 83.6 for students and 77.8 for staff, yielding a combined mean of 81.4 (Grade B, 'Good'). Staff unfamiliarity with OAuth popups and the image upload widget overlay were identified as priority UI refinements.

Table 6. Usability Testing Results (N = 30 Participants)

Task	Completion Rate	Avg. Time	Key Observation
Account Creation & Login	Students: 100% / Staff: 95%	38s / 61s	Staff unfamiliar with OAuth popup
Submit Lost Report with Image	Students: 100% / Staff: 90%	42s / 74s	Upload widget overlay unfamiliar to staff
Search with ≥ 2 Active Filters	Students: 95% / Staff: 85%	28s / 47s	10% needed second attempt on filter panel
Admin Dashboard Moderation	Staff: 90%	83s	Batch confirmation dialog caused brief hesitation

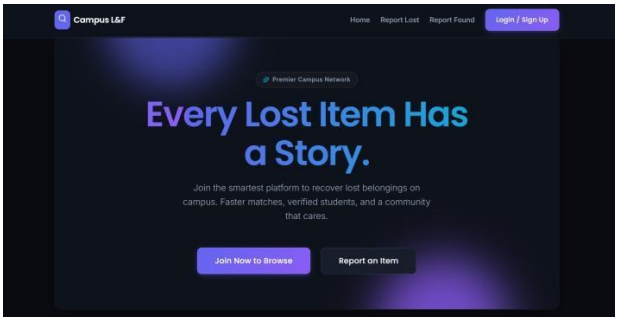

Fig. 5. Home Page

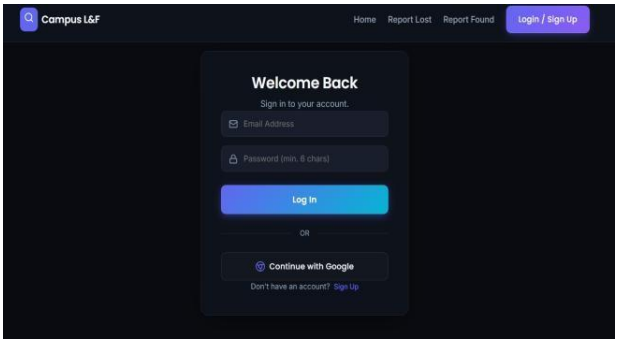

Fig. 6. Login Page with Email/Password and Google OAuth Sign-In options

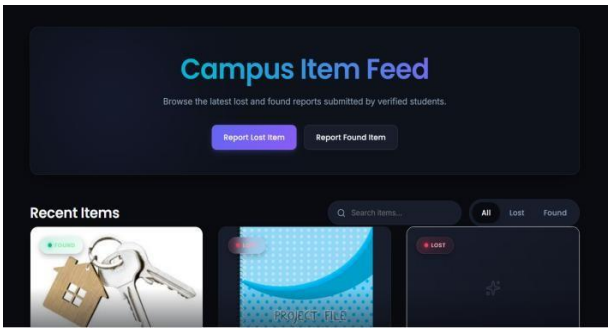

Fig. 7. Authenticated User Dashboard with Report Lost and Report Found navigation

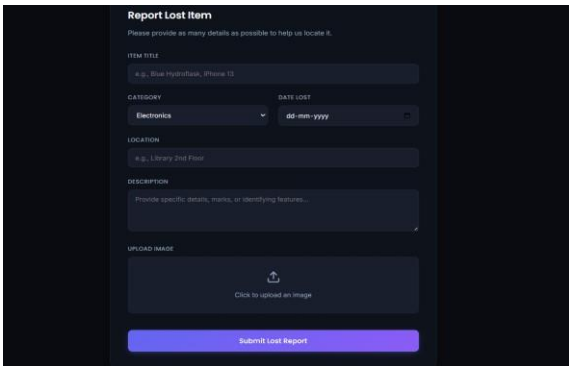

Fig. 8. Lost Item Reporting Form showing all input fields and image upload preview

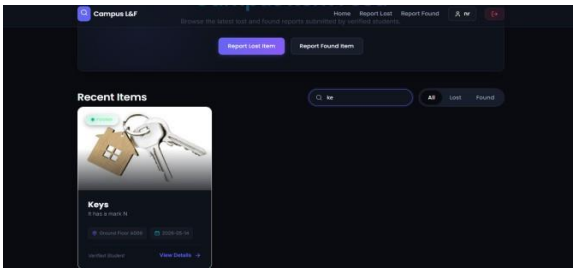

Fig. 9. Search and Filter Module with active category filter applied

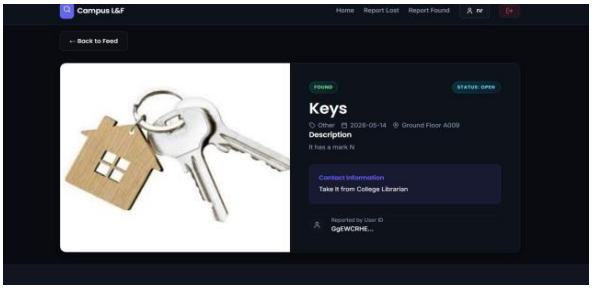

Fig. 10. Item Detail Page showing complete item information and Contact Owner button

3. Manual Baseline Comparison

Five administrative staff processed ten standardized item reports using both the manual paper system and the LFIS platform. Logging time decreased from an average of 4 min 20 s (manual) to 1 min 14 s (digital), a reduction of approximately 71%. Record retrieval time decreased from 3 min 10 s to 18 s, a reduction of approximately 90%. These improvements are directionally consistent with the 50%+ efficiency gains reported by Pede et al. [1] in a comparable digitalization study. Table VII quantifies the full comparison.

Table 7. Manual System Vs. Lfis Digital System Comparison

Measure	Manual System	LFIS Digital System	Improvement
Avg. Time to Log One Report	~4 min 20 s	~1 min 14 s	~71% reduction
Remote Accessibility	On-site only	Any device, any location	Full remote access
Visual Evidence per Report	None or Polaroid	Cloudinary CDN image	Standard image support
Avg. Record Retrieval Time	~3 min 10 s	~18 s (search)	~90% faster
Multi-Field Search	Not possible	Composite Firestore query	Structured multi-field search

4. Security Validation

Firebase Security Rules were tested by issuing Firestore requests with: an expired JWT, no JWT (unauthenticated), and a valid non-admin JWT against admin-restricted operations. All three categories of unauthorized request were rejected at the server layer with permission-denied errors, independent of client-side route guard state. Cloudinary's unsigned upload preset was verified to reject non-image MIME types and files exceeding the configured size limit.

IX. CONCLUSION

This paper presented the Lost and Found Item System, a cloud-hosted web application that replaces manual lost and found processes with a structured, image-enabled, real-time digital platform. The system integrates React.js v19, Firebase Authentication, Cloud Firestore, and Cloudinary CDN within a serverless architecture requiring no local infrastructure maintenance.

Evaluation results confirm all defined design objectives are met. SUS testing yielded a combined mean score of 81.4, performance benchmarking demonstrated first contentful paint averaging 1.74 s and composite queries averaging 1.08 s, and the manual baseline comparison quantified 71% and 90% reductions in logging and retrieval time respectively. The system directly addresses the four structural deficiencies identified: absence of real-time synchronization, lack of image evidence, single-field search inadequacy, and local infrastructure dependency. The serverless, modular architecture generalizes to equipment booking, visitor management, incident reporting, and similar institutional resource management applications. The system is currently deployed and operational at Quantum University and provides a repeatable template for similar deployments.

Future Scope

Several targeted enhancements have been identified. AI-based image similarity matching using TensorFlow.js browser inference would enable automatic visual comparison between lost and found item photographs without GPU server requirements. Firebase Cloud Messaging (FCM) integration would deliver real-time push notifications whenever a found-item report matches attributes of an existing lost-item report—addressing the most frequently cited requirement survey pain point. React Native applications for Android and iOS would extend full functionality to native mobile platforms. A multi-tenant Firestore architecture would allow independent institutions to operate isolated registries within shared infrastructure. For high-value items, an ownership verification workflow requiring submission of purchase receipts or institutional ID would reduce false claim rates.

Table 8. Future Development Roadmap

Enhancement	Technology / Approach	Priority
AI Image Matching	TensorFlow.js / Google Vision API — feature-vector comparison	High
Push Notifications	Firebase Cloud Messaging + Service Workers	High
GPS Location Tagging	Google Maps JavaScript API	High
Native Mobile App	React Native — Android & iOS with camera integration	Medium
Real-Time In-App Chat	Firebase Realtime Database — bidirectional messaging	Medium
Multi-Institution Support	Multi-tenant Firebase with domain-based routing	Low
Ownership Verification	Admin workflow; document upload + review pipeline	Low

REFERENCES

1. D. Pede, V. Ranawat, R. Shete, S. Gole, and T. Kolhe, "Lost and Found Web Application," Int. J. Innovative Sci. Research Technol., vol. 10, no. 5, pp. 1242–1244, May 2025.

2. T. S. Tan and C. R. Chong, "An Effective Lost and Found System in University Campus," J. Inf. Technol. Manag., vol. 8, no. 32, pp. 45–52, 2024.
3. M. Zhou, I. Fung, L. Yang, N. Wan, K. Di, and T. Wang, "LostNet: A Smart Way for Lost and Find," arXiv:2301.02277, Jan. 2023.
4. P. Pujari et al., "Smart Lost and Found System for Campus," Zenodo Research Publication, 2026.
5. K. P. Lasagas et al., "USTP-Panaon Lost and Found Management System," in Proc. MDP Student Conf., 2025.
6. T. Meshram et al., "Lost and Found Item Recovery System," Int. J. Research Appl. Sci. Eng. Technol., vol. 13, no. 2, 2025.
7. A. Krishnan et al., "CampusTrace: Application for Lost and Found Items on Campus," J. Technol. Comput. Sci., vol. 5, no. 3, pp. 20–27, 2024.
8. S. K. Singh, M. Yadav, Y. Singh, and D. Barak, "A Web-Based Information Management System for Digital Platforms," Int. J. Comput. Appl., vol. 184, no. 9, pp. 12–18, 2023.
9. A. K. Mishra and P. Sharma, "Serverless Architecture Using Firebase for Mobile and Web Applications," in Proc. IEEE Int. Conf. Computing, Communication and Security (ICCCS), Phuket, 2022, pp. 1–6.
10. N. Holohan, A. Walsh, and B. O'Sullivan, "Cloud-Based Web Applications for Campus Resource Management," Int. J. Web Eng. Technol., vol. 17, no. 2, pp. 134–155, 2022.
11. R. Kumar and S. Tiwari, "Progressive Web Applications for Indian Higher Education Institutions," Indian J. Comput. Sci. Eng., vol. 14, no. 1, pp. 45–54, Feb. 2023.
12. N. V. Reddy and S. Choudhury, "Digital Transformation of Administrative Processes in Indian Universities," J. Higher Educ. Manag., vol. 9, no. 2, pp. 101–115, 2023.
13. K. Srinivasan and M. Rajesh, "Implementation of NoSQL Databases in Cloud-Based Academic Management Systems," Int. J. Adv. Comput. Sci. Appl., vol. 13, no. 7, pp. 310–317, 2022.
14. Firebase Team, "Firebase Documentation — Authentication and Cloud Firestore," Available: https://firebase.google.com/docs. [Accessed: May 2025].
15. Cloudinary Inc., "Cloudinary Developer Documentation," Available: https://cloudinary.com/documentation. [Accessed: May 2025].
16. Meta Open Source Team, "React — A JavaScript Library for Building User Interfaces," Available: https://react.dev. [Accessed: May 2025].
17. I. Sommerville, Software Engineering, 10th ed. Harlow: Pearson Education, 2016.
18. J. Nielsen, Usability Engineering. Amsterdam: Academic Press / Morgan Kaufmann, 1993.
19. Shaiful Mahmud, Khaleel Khan Mohammed, Vasu Raj Jain, Sarthak Anandkumar Shah. "Artificial Intelligence-Driven Predictive Models for Identifying Risk Factors of Chronic Diseases"