International Journal of Science, Strategic Management and Technology
Volume **02** Issue **11** November-2026 | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Lost and Found Portal: A Web-Based Lost and Found Management System

Dr. S. Prathiba
Assistant Professor
Department of Advanced Computing and Analytics
VISTAS
prathiba.scs@vistas.ac.in

Ms. R. Jeyashree
Assistant Professor
Department of Advanced Computing and Analytics
VISTAS
rjeyashree.scs@vistas.ac.in

Dr. T. R. Nisha Dayana
Associate Professor
Department of Computer Science and Information Technology
VISTAS
nisha.dayana1984@gmail.com

https://doi.org/10.55041/ijsmt.v2i11.040

Cite this Article: Prathiba, S., Jeyashree, R. & Dayana, T. R. N. (2026). Lost and Found Portal: A Web-Based Lost and Found Management System. International Journal of Science, Strategic Management and Technology, *02*(11), 1-9. https://doi.org/10.55041/ijsmt.v2i11.040

License: This article is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting use, distribution, and reproduction in any medium, provided the original author(s) and source are properly credited.

Abstract

Lost and found management is a common challenge in educational institutions, offices, public spaces, and other shared environments. Items reported as lost or found are often communicated through notice boards, phone calls, social media groups, or informal conversations. These methods can make it difficult to maintain accurate records, connect owners with recovered items, and track the status of claims. This paper presents the design of a Lost and Found Portal, a web-based application intended to centralize the reporting and retrieval of lost and found belongings. The proposed system provides separate forms for reporting lost and found items, a searchable listing, item details, claim requests, and an administrative review process. A typical implementation can use HTML, CSS, and JavaScript for the front-end, Python with FastAPI for the back-end, and a relational database for persistent records. The paper describes the problem, objectives, scope, architecture, workflow, implementation plan, preliminary test cases, limitations, and future enhancements. The portal is presented as an academic project design; deployment, real-user testing, and operational results should be added only after they have been completed and verified.

Keywords

Lost and found portal, web application, item recovery, reporting system, FastAPI, database management, REST API.

1. Introduction

People frequently misplace personal belongings such as identity cards, keys, wallets, books, electronic devices, water bottles, and other everyday items. In colleges and other shared environments, a person who finds an item may not know how to contact its owner. Similarly, someone who loses an item may have to ask several departments or search multiple communication channels. A centralized digital portal can make this process more organized by allowing users to submit reports and search existing records.

The Lost and Found Portal is proposed as a web application that connects people who have lost items with people who have found them. Users can enter item descriptions, categories, locations, dates, and relevant images. The portal can display searchable records while protecting sensitive information and requiring verification before an item is handed over. An administrator can review reports, moderate suspicious entries, and manage claim requests.

The project focuses on a clear user interface, structured data storage, reliable front-end and back-end communication, and a traceable claim workflow. It is intended as a foundation that can be extended with notifications, image-based matching, and institutional authentication.

2. Problem Statement

Lost and found information is often scattered across notice boards, reception desks, messaging groups, and verbal announcements. Such fragmented reporting can result in duplicate entries, incomplete descriptions, delayed communication, and difficulty identifying the rightful owner. There may also be no consistent record of whether an item is available, claimed, returned, or still awaiting verification.

A centralized portal is therefore needed to provide a consistent way to report lost and found items, search records, submit claims, and monitor the status of each report. The system should also include basic moderation and privacy controls so that sensitive identifying details are not exposed unnecessarily.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

3. Objectives

- Develop a simple browser-based interface for reporting lost and found items.
- Allow users to search and filter reports by category, date, and location.
- Store item reports and claim requests in a structured database.
- Provide a claim-request process to help verify ownership before handover.
- Develop back-end API endpoints for report creation, search, status updates, and claim handling.
- Include validation, role-based administrative controls, and basic error feedback.
- Establish a modular foundation for future notification and matching features.

4. Scope of the Project

The proposed scope covers a web-based portal for an institution or shared public environment. The main functions are user registration or identification, lost-item reporting, found-item reporting, browsing and searching listings, viewing item details, submitting a claim, and administrator review. A report may contain an item name, category, description, approximate date, location, image, contact preference, and current status.

The initial version can be developed as a local prototype. It does not automatically prove ownership, guarantee that an item will be recovered, or replace an institution's physical lost-property process. Advanced image recognition, GPS tracking, payment handling, large-scale deployment, and integration with external identity systems are outside the initial scope and may be considered future enhancements.

5. Related Work and Technical Background

Digital classified systems and service portals demonstrate how structured forms, search tools, and status tracking can organize user-submitted information. A lost and found application applies these familiar web patterns to the specific task of item recovery. Unlike a general public listing, the portal should include controlled claim handling because an item description alone is not sufficient proof of ownership.

A browser-based front-end can be built using HTML, CSS, and JavaScript. JavaScript fetch() can send HTTP requests to a REST-style API, while JSON provides a consistent request and response format. Python with FastAPI can handle validation, routing, and business logic. A relational database such as SQLite during development or PostgreSQL for a deployed version can store user, item-report, and claim records. These technologies provide a modular educational foundation that can be extended as requirements grow.

6. Proposed System

The proposed system consists of a user interface, a back-end API, a database, and an administrative module. A user selects whether to report a lost item or a found item and enters the available details. The front-end validates required fields and sends the request to the API. The back-end validates the data, stores the report, and returns a JSON response. Other users can search the listings and submit a claim request for a potentially matching item. The administrator reviews reports and claims, updates statuses, and coordinates safe handover according to institutional procedures.

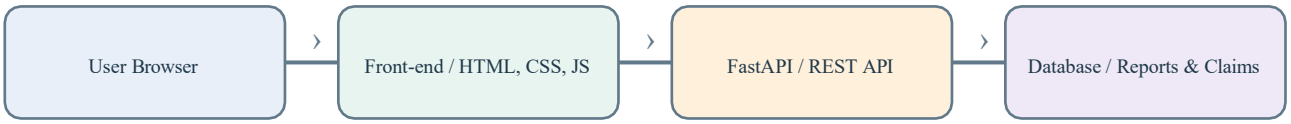


Figure 1. Conceptual architecture of the Lost and Found Portal.

6.1 Main Modules

- User Interface: provides navigation, report forms, search filters, item details, and status display.
- User and Access Module: manages user accounts or institutional identity and applies role-based access.
- Lost Item Module: accepts and stores reports of belongings that users have lost.
- Found Item Module: accepts reports of items that have been found and records where they were submitted.
- Search and Matching Module: filters records using keywords, categories, dates, and locations.
- Claim Management Module: records ownership claims and supports verification by an authorized person.
- Admin Module: reviews reports, moderates content, handles disputes, and updates item status.
- Notification Module: may send email or in-app updates when a claim or report changes status.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

6.2 Design Considerations

The system should collect only the information required to support item recovery. Public listings should avoid exposing phone numbers, identity-card numbers, serial numbers, or other sensitive details. Users should be encouraged to keep one-of-a-kind identifying details private so they can be used during ownership verification. Uploaded images should be checked for file type and size. Administrative actions should be restricted to authorized accounts and recorded for accountability.

A production deployment should use HTTPS, secure password storage, input validation, rate limiting, access controls, safe file-upload handling, and database backups. The application should not be used to publish sensitive personal information or to disclose the exact contents of a wallet, bag, or other private item.

7. Methodology and Workflow

The project can follow an incremental development approach. First, the requirements and user roles are identified. Next, the front-end pages and database structure are designed. The back-end API is then developed to accept reports, retrieve listings, create claims, and update statuses. The front-end communicates with the API using HTTP requests and displays the returned data. Finally, test cases are used to check form validation, search, report creation, claim submission, and administrative actions.



Figure 2. High-level report and claim workflow.

7.1 Execution Steps

1. Open the Lost and Found Portal in a web browser.
2. Register or sign in, if authentication is enabled.
3. Choose “Report Lost Item” or “Report Found Item”.
4. Enter the item name, category, description, date, and location.
5. Add an image if supported and submit the report.
6. The back-end validates the request and stores the report in the database.
7. Users browse or search listings for a possible match.
8. A user submits a claim request with details that can help verify ownership.
9. An administrator reviews the claim and updates its status.
10. Once handover is confirmed, the report is marked as returned or closed.

7.2 High-Level Algorithm

BEGIN
Open portal and identify user role IF user submits a report:
Validate required fields and uploaded file
IF input is valid: save report and return confirmation ELSE: display validation message
ELSE IF user searches:
Apply selected filters and return matching reports ELSE IF user submits a claim:

Save claim as pending verification Notify or route claim to administrator ELSE IF administrator reviews a record:
Verify details and update report or claim status Display result or error message
END

7.3 Tools and Technologies

Front-end: HTML, CSS, JavaScript. Back-end: Python and FastAPI. Server: Uvicorn. Database: SQLite for a local prototype or PostgreSQL for deployment. Communication: HTTP and JSON. Development tools: Visual Studio Code and a web browser. Optional future services: email notifications and secure image storage.

8. Implementation Details

A modular project structure can separate the front-end, API routes, database models, and uploaded files. The front-end may contain index.html, style.css, and script.js. The back-end may contain the FastAPI application, database configuration, schemas, and route modules for reports, search, claims, and administration. The front-end reads form values, validates required fields, sends requests using fetch(), parses the JSON response, and displays confirmation or error messages.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

A report record may include a unique report ID, report type (lost or found), item name, category, description, date, location, image reference, submitter ID, creation timestamp, and status. A claim record may include a claim ID, related report ID, claimant ID, verification response, review status, and timestamps. Exact fields should be finalized during implementation based on the institution's requirements.



Figure 3. Front-end and back-end request exchange.

8.1 API Design

The following REST-style endpoints describe a possible API design. They are proposed endpoints and should be adjusted to match the implemented application.

Endpoint	Purpose	Expected behaviour
POST /reports	Create a lost or found report	Validate and store report
GET /reports	List/search reports	Return filtered report records
GET /reports/{report_id}	View one report	Return report details
POST /reports/{report_id}/claims	Submit a claim	Create pending claim
GET /claims	Review claim requests	Authorized admin access
PATCH /claims/{claim_id}	Update claim status	Record admin decision
PATCH /reports/{report_id}/status	Update item status	Mark returned/closed

9. API Communication and Testing

The front-end sends an HTTP request to the FastAPI back-end. The API validates the request, performs the relevant database operation, and returns a JSON response. The front-end reads the response and updates the page. During development, the API documentation can be made available through FastAPI's /docs route.

Testing should verify both successful and unsuccessful scenarios. For example, an empty report should be rejected, a valid report should be stored, a search should return matching records, and a claim should remain pending until reviewed. The test observations

below are expected outcomes for a planned prototype; actual status should be updated after execution.

9.1 Results and Discussion

The proposed design brings lost-item reports, found-item reports, search, and claim handling into one interface. Its modular structure allows the report, search, and claim functions to be developed and tested independently. A database-backed approach can provide more consistent records than informal announcements and can make it easier to track whether an item is open, under review, returned, or closed. The design does not establish that an item has been recovered or that ownership has been verified. Those outcomes depend on implementation, administrator review, and the institution's handover process. Performance, usability, security, and matching effectiveness should be evaluated using real test data before making claims about operational results.

Test case	Expected result	Verification note
Empty report form	Validation message	To be tested
Valid lost-item report	Report saved	To be tested
Valid found-item report	Report saved	To be tested
Search by category/location	Relevant reports displayed	To be tested
Claim submission	Claim stored as pending	To be tested

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

Unauthorized admin action	Access denied	To be tested
Backend unavailable	Connection error displayed	To be tested

10. Limitations and Future Enhancements

Current / anticipated limitations:

- The portal depends on users submitting accurate and sufficiently detailed reports.
- Similar item descriptions may produce false matches.
- Ownership cannot be confirmed reliably from a public description alone.
- A local prototype may not include production authentication, notifications, or deployment.
- Uploaded images and personal data require careful storage and access control.

Future enhancements:

- Add email or in-app notifications for new matches and claim updates.
- Introduce image-based similarity suggestions with human review.
- Add institution-based login and role-based permissions.
- Provide a dashboard showing open, pending, returned, and closed reports.
- Add QR-code labels or an authorized collection-desk workflow.
- Improve accessibility, mobile responsiveness, audit logs, and deployment security.

11. Security and Privacy Considerations

A production version should use HTTPS, secure authentication, role-based authorization, and strong password hashing. Input fields and uploaded files must be validated on the server. File uploads should be restricted by type and size and stored in a controlled location. Public listings should hide personal contact details and sensitive item identifiers. Claim verification should be handled privately by an authorized person, and administrative changes should be logged. Data retention rules should define when old reports and personal information are archived or deleted. The institution should review applicable privacy and information-security requirements before deployment.

12. Conclusion

This paper presented the design of a Lost and Found Portal, a web-based system intended to organize the reporting and recovery of misplaced belongings. The proposed application connects lost-item reports, found-item reports, searchable listings, claim requests, and administrative review through a common interface. HTML, CSS, and JavaScript can provide the front-end, while Python with FastAPI can handle API requests and a relational database can maintain records.

The portal offers a structured approach to a process that is often managed through scattered announcements and informal communication. Its effectiveness will depend on accurate reporting, secure handling of user information, and a reliable ownership-verification process. Future work can focus on implementation, testing with real users, notifications, image-based matching, and secure deployment.

References

[1] FastAPI, “FastAPI Documentation.” https://fastapi.tiangolo.com/
[2] Python Software Foundation, “Python Documentation.” https://docs.python.org/3/
[3] MDN Web Docs, “Using the Fetch API.” https://developer.mozilla.org/docs/Web/API/Fetch_API
[4] MDN Web Docs, “Working with JSON.” https://developer.mozilla.org/docs/Learn/JavaScript/Objects/JSON
[5] Uvicorn, “Uvicorn Documentation.” https://www.uvicorn.org/
[6] PostgreSQL Global Development Group, “PostgreSQL Documentation.” https://www.postgresql.org/docs/
[7] SQLite, “SQLite Documentation.” https://www.sqlite.org/docs.html
[8] OWASP Foundation, “OWASP Top 10.” https://owasp.org/www-project-top-ten/

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.