Campus Lost & Found System

Section A — Project Report

Academic Year: 2025–2026

Submitted By

RITWICK RANJAN DAS
MOHD ASIF
ASHISH PAWADE

Under the Guidance of

Professor Gayathri Ananthanarayanan

Department of Computer Science
[IIT DHARWAD]
[Dharwad, Karnataka]

Contents

1 Introduction

The **Campus Lost & Found** system is a web-based application developed as *Section A* of the larger Campus Connect project. The goal of this module is to provide a central platform where students and staff can report lost items, register found items, and manage claims in a structured and secure way.

The system follows a modern full-stack architecture: a **Next.js frontend** communicates with **Next.js API routes** that use **Prisma ORM** to talk to a **SQLite** database. Authentication and authorization are handled using **JSON Web Tokens (JWT)**.

2 System Overview

2.1 Objectives

- Provide a unified portal for reporting lost and found items on campus.
- Ensure secure user authentication and role-based access.
- Track ownership and claim status of each item.
- Expose well-documented REST APIs that can integrate with other modules of Campus Connect.

2.2 High-Level Architecture Diagram

The following text-based diagram summarizes how the components interact:

```
+-------------+        HTTP/JSON        +---------------------+       Prisma        +------
|  Browser    | <---------------------> |  Next.js Frontend   | <-----------------> |  SQL
| (React UI)  |                         |  (Pages & Components)                       |  DB
+-------------+                         +---------------------+                      +-----
                                              |
                                              | API calls (/api/...)
                                              v
                                      +---------------------+
                                      |  Next.js API Routes |
                                      |  (/api/auth, /api/items,
                                      |   /api/claims, /api/health) |
                                      +---------------------+
```

The same Next.js project hosts both the user interface and the backend API endpoints. This simplifies deployment and makes local development easier.

3 API Design and Backend Logic

3.1 Technology Stack

- **Backend Framework:** Next.js 16 API Routes (TypeScript).
- **Database:** SQLite, accessed via Prisma ORM.
- **Authentication:** JSON Web Tokens (JWT).
- **Documentation:** OpenAPI 3.0 specification consumed by Swagger UI.

3.2 Core API Endpoints

Table 1 summarizes the most important endpoints defined in the `openapi.yaml` file.

Method	Path	Description
GET	/api/health	Health check endpoint for monitoring.
POST	/api/auth/register	Register a new user (stores hashed password).
POST	/api/auth/login	Login and return a signed JWT token.
GET	/api/items	List all items (public).
POST	/api/items	Create a new item (JWT required).
POST	/api/items/{id}/claim	Create a claim for an item (JWT required).
GET	/api/items/{id}/claims	List all claims for an item (owner-only).
GET	/api/claims/mine	List claims made by the current user.
POST	/api/claims/{id}/approve	Approve a claim (item owner).
POST	/api/claims/{id}/reject	Reject a claim (item owner).

Table 1: Core API endpoints of the Campus Lost & Found backend

3.3 Request Flow (Sequence Diagram — Text Form)

A typical flow for logging in and creating a lost item is described below as a textual sequence diagram.

Login Sequence

```
User         Frontend (Next.js)     Backend (/api/auth/login)      DB (SQLite)
 |                    |                          |                     |
 | 1. Enter creds     |                          |                     |
 |------------------->| 2. POST /api/auth/login  |                     |
 |                    |------------------------->| 3. Find user       |
 |                    |                          |---> Prisma query ->|
 |                    |                          |<--- user row ------|
 |                    |                          | 4. bcrypt.compare  |
 |                    |                          | 5. sign JWT        |
 |                    |<-------------------------| 6. { token } JSON  |
 | 7. Store token     |                          |                     |
```

Create Item Sequence

```
User   Frontend        Backend (/api/items)          DB (SQLite)
 |        |                      |                       |
 | 1. Fill item form            |                       |
 |--------->| 2. POST /api/items with Bearer token      |
 |          |------------------>| 3. Verify JWT         |
 |          |                   | 4. prisma.item.create |
 |          |                   |------> INSERT ------->|
 |          |                   |<----- New row --------|
 |          |<------------------| 5. Return item JSON   |
 | 6. Show success              |                              |
```

4 Frontend Integration

The frontend is built using React components inside the Next.js project. The pages consume the above APIs using standard HTTP calls.

4.1 Connection Between Frontend and Backend

- The frontend displays forms for registration, login, item creation, and claims.
- On form submit, the UI uses `fetch` (or a similar library) to call the corresponding `/api/...` endpoint.
- Responses are parsed as JSON and the UI is updated (for example, redirecting the user or showing success/error messages).
- Because both UI and API are served from the same origin (`http://localhost:3000` for the Next.js app), CORS issues are minimized.

5 Security: JWT-Based Authentication

JSON Web Tokens (JWT) are used to secure protected routes.

- During login, the backend signs a token with a secret key (`JWT_SECRET`) including user ID and role.
- The token is returned to the client in the response body.
- The client sends this token in the `Authorization` header as `Bearer <token>` for subsequent requests.
- Backend middleware validates the token on each request and rejects unauthorized access with status code `401`.

6 Testing and Documentation

6.1 Swagger UI

The OpenAPI specification is stored in files such as `openapi.yaml` and `openapi.3002.yaml`. Swagger UI is run in a Docker container and mounts the YAML file so that all endpoints can be tried out interactively through a web interface.

This allows the development team, as well as the project supervisor, to visually inspect the APIs, send sample requests, and verify responses.

6.2 Integration and Coverage

The project also defines integration tests (for example using Vitest) and can be extended with coverage tools so that more than 60% of the code is automatically tested on each commit through GitHub Actions. This helps to maintain code quality and catch regressions early.

7 Test Coverage Report

To quantitatively evaluate the reliability of the *Campus Lost & Found* backend, we enabled line coverage and executed the full unit and integration test suite using the following command:

```
npm run coverage
```

This internally invokes Vitest with V8 coverage and multiple reporters (text, HTML, and LCOV). The coverage summary generated by the tool indicates that the project satisfies the requirement of at least 60% line coverage.

7.1 Summary of Results

Table 2 summarizes the most important coverage metrics extracted from the coverage report.

Metric	**Value**
Number of test files	5 (all passed)
Total tests	5 (all passed)
Overall statements coverage	70.76%
Overall branches coverage	50.94%
Overall functions coverage	85.71%
Overall lines coverage	80.40%
Configured minimum line threshold	60.00%

Table 2: Coverage summary for the Campus Lost & Found backend

All five test files (one unit test for authentication and four integration tests for profile, claims, lost-and-found flow, and items) passed successfully, and the overall line coverage of approximately 80% is well above the 60% threshold.

7.2 Per-Module Coverage (Selected Files)

Table 3 reports coverage for a subset of important API route modules and helper libraries.

File / Module	**% Lines**	**Notes**
`app/api/auth/login/route.ts`	68.75%	Unit-tested login flow.
`app/api/auth/me/route.ts`	100%	Covered via login–me integration.
`app/api/auth/register/route.ts`	80.00%	Register and conflict cases.
`app/api/items/route.ts`	70.00%	Create and list items tested.
`app/api/claims/route.ts`	73.91%	Claims CRUD covered.
`app/api/profile/route.ts`	87.50%	Profile update and fetch logic.
`lib/auth.ts`	87.50%	JWT helpers exercised by tests.
`lib/prisma.ts`	100%	Prisma client wrapper fully covered.

Table 3: Selected per-module coverage results

These values show that not only is the overall coverage high, but critical security- and data-related modules (authentication, items, claims, and database access) are all well tested.

7.3 Raw Coverage Output (Excerpt)

For completeness, an excerpt of the text coverage report is provided below. The full report (including HTML and LCOV output) is available in the project repository.

```
> campus-lost-found@0.1.0 coverage
> cross-env NODE_ENV=test vitest run --coverage --coverage.provider=v8 ...

Test Files  5 passed (5)
Tests       5 passed (5)

% Coverage report from v8
-------------------|---------|----------|---------|---------
File               | % Stmts | % Branch | % Funcs | % Lines
-------------------|---------|----------|---------|---------
All files          |   70.76 |    50.94 |   85.71 |   80.40
...
```

This confirms that the automated tests exercise a substantial portion of the backend logic and that the Campus Lost & Found module is supported by a robust test suite.

8 Conclusion

The **Campus Lost & Found** module provides a clean, modern, and secure backend for managing lost and found items on campus. The architecture is modular and can be integrated with other sections of the Campus Connect project, such as the main authentication dashboard and hostel or forum modules.

Future improvements include richer search and filtering, notification systems, and tighter integration with other campus services.