WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

HOSTEL REGISTER WEB-BASED HOSTEL COMPLAINT & MANAGEMENT SYSTEM

MUKESH K[1], KRISHNAKUMAR M [2],PRAVEENKUMAR E[3], SARANYA M[4]

[4] Project Guide:
Department of Information Technology
Sri Shakthi Institute of Engineering and Technology
Coimbatore, Tamil Nadu, India
Email Id: kk795896108@gmail.com

ABSTRACT :

Managing student complaints in college hostels is often delayed by paper registers, informal messaging, and lack of a clear audit trail. This project presents Hostel Register, a lightweight web-based complaint management system that serves both students and wardens. The application is built with a pure Node.js backend (no external npm dependencies beyond the runtime) that uses the built-in node:sqlite module for persistent storage, together with a static HTML/CSS/JavaScript frontend. Students can register, raise complaints with optional photo attachments, track status, reply to wardens, and manage their accounts. Wardens can view and filter all complaints, update status, leave public or internal notes, export data to CSV, and administer user accounts. Security features include scrypt password hashing, signed HttpOnly session cookies, rate limiting, content-security policy, and safe file-upload handling. Notifications are delivered in-app and optionally by email when SMTP is configured. Hostel Register is intended as a practical campus tool that improves transparency and response time while remaining simple to deploy on any host that supports Node.js 22.13 or newer.

Keywords: Hostel Management, Complaint System, Node.js, SQLite, Student Portal, Warden Dashboard, Session Authentication, File Upload.

1. INTRODUCTION

Hostel administration in educational institutions involves continuous handling of maintenance issues, facility problems, and student grievances. Traditional paper registers or informal chat groups make it difficult to track the life-cycle of a complaint, measure response times, or produce reports for management. Students frequently do not know the current status of their requests, and wardens lack a single consolidated view of open issues across blocks.

A dedicated web application can address these gaps by providing role-based access, structured complaint categories, status workflows, photo evidence, and notification mechanisms. The present work develops such a system under the name Hostel Register. The design emphasises minimal dependencies, security of credentials and uploads, and ease of deployment on modest hardware.

1.1 Existing challenges

Paper-based complaint registers are slow to update, hard to search, and offer no automatic notification. Messaging apps scatter information across private chats and provide no formal status history. Spreadsheet trackers require manual coordination and are prone to concurrent-edit conflicts. Any digital solution must also protect student data, prevent unauthorised access, and handle photo uploads safely without becoming a vector for malware or storage exhaustion.

1.2 Role of modern web technologies

Recent Node.js releases include a synchronous SQLite driver (node:sqlite), removing the need for an external database server for moderate workloads. Static HTML pages with vanilla JavaScript can deliver a responsive interface without a heavy frontend framework. Signed cookies provide session state without a separate session store. These choices keep the dependency surface small and the deployment footprint light.

1.3 Problem statement and objectives

The project addresses the need for a convenient, role-based hostel complaint portal that records issues with optional photographs, supports status

transitions, enables two-way communication between students and wardens, and produces exportable reports. The system is designed for institutional use rather than public commercial deployment.

The objectives of Hostel Register are:

- To provide a self-service registration and login portal for students and wardens.
- To allow students to submit complaints with category, description, room details, and up to three photo attachments.
- To let students track status, edit pending complaints, cancel open complaints, and reply to warden comments.
- To give wardens a filterable dashboard of all complaints with status updates, public and internal notes, and deletion of spam.
- To support CSV export of filtered complaint lists for administrative reporting.
- To implement secure authentication (scrypt hashing, signed sessions) and safe file handling.
- To deliver in-app notifications and optional email alerts when SMTP is configured.
- To keep the technology stack free of external npm packages so that the application remains easy to audit and deploy.

2. MATERIALS AND METHODS

2.1 Development approach

The project was developed through requirement gathering, database schema design, backend API implementation, frontend page construction, security hardening, and iterative testing. The backend is a single Node.js process that serves both the REST API and the static frontend assets. The frontend consists of separate HTML pages for login, registration, student dashboard, warden dashboard, account settings, and password-reset flows, all sharing a common stylesheet and utility scripts.

2.2 System architecture

Hostel Register follows a classic client–server model. The browser loads static pages from the Node server. Authenticated API calls use a signed HttpOnly cookie. The server validates the session, executes SQLite queries, and returns JSON. Photo uploads are stored on disk under randomly generated names and referenced from the attachments table.

Layer / component	Responsibility
Frontend – HTML / CSS / JS	Login, registration, student & warden dashboards, account pages, complaint forms, notifications UI.
API routes (server.js)	Authentication, complaint CRUD, comments, attachments, notifications, admin user management, CSV export.
SQLite database	Persists users, complaints, comments, attachments metadata, password-reset tokens, and notifications.
File storage (uploads/)	Stores complaint photographs under random filenames; served only to authorised users.

2.3 Technology stack

Technology	Use in the project
Node.js ≥ 22.13	Runtime; provides built-in node:sqlite, crypto, http modules.
node:sqlite (DatabaseSync)	Synchronous SQLite access for users, complaints, comments, attachments, notifications.
Vanilla HTML / CSS / JS	All user-facing pages; no frontend framework or build step.
scrypt (Node crypto)	Password hashing with per-user salt.
HMAC-SHA256 cookies	Signed session tokens (HttpOnly, SameSite=Lax).
Custom mailer.js	Lightweight SMTP client with zero external dependencies.

Table 2. Technology stack used in Hostel Register.

2.4 Database schema

The SQLite schema comprises six primary tables created on first launch (with automatic migrations for later columns):

Table	Purpose & key columns
users	role (student/admin), login_id, name, email, block, room, password_hash, must_change, session version (sv).
complaints	user_id, category, title, description, block, room, status (pending / progress / resolved / rejected / cancelled).
comments	complaint_id, user_id, body, internal flag (warden-only notes).
attachments	complaint_id, file (random name), mime, size.
password_resets	user_id, token_has, expires_at (1-hour single-use tokens).
notifications	user_id, message, link, read flag.

Table 3. Core database tables.

2.5 Authentication and session management

Passwords are hashed with script using a random 16-byte salt stored together with the hash. Login produces a signed JWT-style cookie containing user id, session version, and expiry (default 8 hours). Changing or resetting a password increments the session version, invalidating all existing cookies for that user. Rate limiting is applied to login, registration, and password-reset endpoints. Cross-origin write requests are rejected.

2.6 Complaint workflow

Student side:

- Register with name, roll/login id, email, block, room and password (minimum 8 characters).
- Create a complaint specifying category, title, description and room; optionally attach up to three JPEG/PNG/WebP images (≤ 3 MB each).
- Edit fields while the complaint remains in pending status; cancel an open complaint.
- View status changes and reply to warden comments.
- Change email or password; request a password-reset link by email.

Warden side:

- View all complaints with filters for status, block and free-text search.
- Update complaint status (pending → in progress → resolved / rejected).
- Post public replies visible to the student or internal notes hidden from students.
- Delete spam complaints; export the current filtered list as CSV.
- Add or remove other wardens; force-reset any user's password (one-time temporary password).

2.7 File upload handling

Uploads are validated by file signature (magic bytes) rather than the client-supplied filename. Accepted types are JPEG, PNG and WebP; maximum size is 3 MB and a maximum of three attachments per complaint is enforced. Files are stored under randomly generated names inside the uploads directory and are served only to the complaint owner or a warden.

2.8 Backend API design

Endpoint	Method	Purpose
/api/register	POST	Student self-registration.
/api/login	POST	Authenticate student or warden.
/api/logout	POST	Clear session cookie.
/api/me	GET/PATCH	Retrieve or update own profile (email).
/api/change-password	POST	Change password (requires current password).
/api/forgot / /api/reset	POST	Request and complete password reset.
/api/complaints	GET/POST	List (role-aware) or create complaints.
/api/complaints/:id	PATCH/DELETE	Update status/fields or delete (warden).
/api/complaints/:id/comments	POST	Add public or internal comment.
/api/complaints/:id/attachments	POST	Upload photo (student, max 3).
/api/attachments/:id	GET/DELETE	Download or remove attachment.
/api/notifications	GET/POST	List or mark notifications read.
/api/admin/export.csv	GET	CSV export with optional filters.
/api/admin/users	GET	List all users (warden).
/api/admin/wardens	POST	Create additional warden account.

Table 4. Principal API endpoints.

2.9 Security measures

- Passwords hashed with scrypt + unique salt; timing-safe comparison.
- Signed HttpOnly session cookies; password change invalidates all sessions.
- Rate limiting on authentication endpoints.
- Upload validation by content signature; random storage names; size and count limits.
- Strict Content-Security-Policy headers; all user-supplied text is escaped before display.
- CSV export protected against formula injection.

3. RESULTS AND DISCUSSION

3.1 Implemented features

The delivered application contains fully functional student and warden interfaces, complete complaint life-cycle management, photo attachments, comment threads (public and internal), in-app notifications, optional email delivery, password-reset flow, CSV export, and warden-user administration. The first launch automatically seeds a default warden account from environment variables (or sensible defaults) and creates the SQLite schema together with the uploads directory.

4. ACKNOWLEDGMENTS

The authors express their sincere gratitude to Sri Shakthi Institute of Engineering and Technology for providing the academic environment and facilities required for this project. The authors thank the project guide, SARANYA M, for guidance, encouragement, and support throughout the development of Hostel Register. The authors also acknowledge the faculty members and contributors who supported the learning and implementation process.

4.1 Functional test scenarios

ID	Module	Scenario	Expected handling
F1	Registration	Valid student data	Account created; can log in.
F2	Login	Correct credentials	Session cookie set; redirected to role dashboard.
F3	Complaint	Submit with photos	Complaint stored; attachments saved under random names.
F4	Validation	Unsupported file type	Upload rejected with clear error.
F5	Status	Warden changes status	Student sees update; notification created.
F6	Comments	Internal note by warden	Visible only to wardens.
F7	Export	CSV download	Filtered rows returned; formula-safe.
F8	Password	Forgot / reset flow	One-time token; old sessions invalidated.
F9	Admin	Add / remove warden	New warden can log in; self-deletion blocked.

Table 5. Representative functional test scenarios derived from the implemented routes.

4.2 Discussion and limitations

Hostel Register demonstrates that a complete multi-role complaint system can be realised with a very small dependency surface. The use of built-in SQLite simplifies deployment and backup (a single file plus the uploads folder). Security practices such as scrypt hashing, signed cookies, upload signature checks and CSP headers address the most common web-application risks for an internal campus tool.

Current limitations include the absence of fine-grained audit logging, no mobile-native application, and reliance on a single SQLite file (suitable for hundreds to low thousands of concurrent users but not for very large multi-campus deployments). Email delivery depends on correct SMTP configuration; without it, reset links appear only in the server console. The interface is functional but could benefit from further accessibility and localisation improvements.

5. CONCLUSION

Hostel Register is a web-based hostel complaint and management system developed with Node.js, the built-in SQLite driver, and a static HTML/CSS/JavaScript frontend. Students can register, submit and track complaints (including photographs), and communicate with wardens. Wardens obtain a consolidated, filterable view of all issues, can update status, leave notes, export data, and administer accounts. Security features include strong password hashing, signed sessions, rate limiting, and safe file handling.

The application is ready for campus pilot use once environment variables (session secret, base URL, SMTP settings) are configured and the default warden password is changed. Future work may include:

- Mobile-responsive refinements and progressive-web-app packaging.
- Richer analytics dashboards and response-time metrics.
- Optional multi-hostel / multi-campus support with role hierarchies.
- Integration with institutional single-sign-on (SSO) systems.
- Automated backup and monitoring of the SQLite database and uploads directory.

6. REFERENCES

[1] Node.js Documentation, “node:sqlite module,” Available: https://nodejs.org/api/sqlite.html

[2] Node.js Documentation, “Crypto – scrypt,” Available: https://nodejs.org/api/crypto.html#cryptoscryptpassword-salt-keylen-options-callback

[3] OWASP Foundation, “Password Storage Cheat Sheet,” Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[4] OWASP Foundation, “Session Management Cheat Sheet,” Available: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[5] MDN Web Docs, “Content Security Policy (CSP),” Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[6] SQLite Documentation, “SQLite Home Page,” Available: https://www.sqlite.org/

[7] Project source code, “Hostel Register: Web-based Hostel Complaint & Management System,” supplied project archive hostel-register.zip. Includes server.js, mailer.js, public/ frontend pages, package.json, and README.