iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** XI **Month of publication:** November 2026

DOI: https://doi.org/10.22214/ijraset.2026.85646

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

HostelDesk: A Web-Based Hostel Complaint and Maintenance Management System

Vinothini T[1], Maheshwaran M[2], Manoj R[3], Manjunath S[4]
Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur, India

Abstract: ***Hostel maintenance complaints are often communicated through informal channels, fragmenting records and making assignment, progress tracking, and resolution verification difficult. This paper presents HostelDesk, a web-based Hostel Complaint and Maintenance Management System designed to manage the complaint lifecycle in a hostel environment. The platform provides role-specific access for students, administrators, and maintenance staff; a hierarchical representation of hostels, blocks, floors, rooms, and beds; category- and subcategory-based issue registration; image and short-video evidence; and support for existing complaints. Complaint urgency is determined using category severity and affected-student information, with four priority levels and configurable service-level deadlines. Administrators can assign approved issues to internal personnel or record external maintenance work, while workers update progress and resolution details through a dedicated portal. Students subsequently verify the outcome, reopen or escalate unresolved issues, and submit feedback. The implementation uses React 18 and Vite for the responsive frontend, Node.js and Express.js for REST APIs, PostgreSQL for relational persistence, JWT-based authentication, bcrypt password hashing, and Multer for media uploads. The documented prototype integrates operational dashboards, monthly reporting, technician information, hostel/block heat maps, notifications, and audit history. The source documentation reports functional capabilities but does not provide quantitative performance or user-outcome measurements.***

Keywords: ***Hostel maintenance, complaint management, work-order tracking, role-based access control, service-level agreement, student verification, facility analytics, web application.***

I. INTRODUCTION

Hostel accommodation requires continuous upkeep of shared and room-level facilities, including electrical systems, plumbing, furniture, cleanliness, water supply, internet connectivity, and civil infrastructure. Maintenance issues in educational institutions may be communicated through verbal requests, telephone calls, notebooks, messaging applications, or direct contact with hostel authorities. Although these channels can report an individual fault, they do not inherently provide a consistent record of the complaint, its location, urgency, responsible worker, current status, or verified outcome. A fragmented reporting process also makes it difficult to recognize when several students are affected by the same physical issue. Similar requests may be handled as separate complaints, while administrators may have to reconstruct the maintenance history from different communication channels. The absence of structured classification, prioritization, assignment, escalation, and student confirmation can reduce visibility across the complete maintenance lifecycle. HostelDesk is designed as a centralized web application that connects students, administrators, and maintenance personnel through a common workflow. Its administrator role also includes hostel-warden responsibilities; a separate warden role is therefore not defined in the documented design. Students submit categorized complaints with location details and optional media, administrators review and assign work, and maintenance personnel update assigned tasks through a dedicated worker portal. Following maintenance completion, the student can confirm the resolution or indicate that the problem persists.

The main contribution is the integration of hostel-specific infrastructure management with complaint and maintenance operations. The system combines (i) role-based access for three user groups, (ii) a hostel/block/floor/room/bed hierarchy, (iii) evidence-supported complaints and collaborative issue support, (iv) severity- and affected-student-informed priority with configurable service-level targets, (v) assignment and student verification, and (vi) reporting, heat-map analysis, notifications, and audit history. The remainder of this paper describes the design, implementation, and documented functional outcomes of the system.

II. RELATED WORK AND DESIGN MOTIVATION

Facility-management and work-request platforms provide an established context for organizing maintenance activities as structured requests rather than informal messages. MaintainX describes facility-management and work-request capabilities [1], [2], while ServiceNow documents workplace maintenance management [3].

©IJRASET: All Rights are Reserved

These sources are used here as examples of operational maintenance systems and not as peer-reviewed experimental evidence.

The project documentation identifies several concepts relevant to a hostel-specific implementation: separating reporting, administration, and worker responsibilities; classifying issues before assignment; considering severity and the number of affected users when prioritizing requests; attaching digital evidence; allowing students to support an existing issue; and retaining status, assignment, and verification history. A further requirement is to connect complaints to student allocations and a structured hostel location, which enables hostel- and block-level views of recurring problems.

The design motivation is therefore not simply to digitize a complaint form. It is to connect complaint creation to subsequent review, priority setting, assignment, maintenance execution, student verification, feedback, and analytics. The supplied bibliography contains vendor and software documentation rather than identifiable peer-reviewed studies; accordingly, the present paper uses those references to identify the system and technology context and does not claim a literature-based experimental comparison.

III. SYSTEM DESIGN AND METHODOLOGY

A. *Problem Context and Requirements*

The existing process described in the project report is characterized by informal complaint submission, fragmented history, repeated reports for the same issue, unclear assignment responsibility, limited prioritization and escalation, and the absence of a formal student verification stage. HostelDesk addresses these gaps by maintaining complaints as structured records and linking them to users, categories, locations, assignments, status transitions, verification actions, feedback, and audit events.

The principal users are students, administrators, and maintenance staff. Students authenticate using their register number and date of birth. Administrators and maintenance staff use their respective credentials. Administrators manage students, hostel infrastructure, categories, complaints, priorities, and work assignments. Maintenance staff view and update only their assigned work. Authorization is enforced by backend role checks rather than relying only on the visible frontend screens.

B. *Proposed System and User Responsibilities*

The system organizes functionality around the responsibilities summarized in Table I. The administrator role includes the duties assigned to hostel wardens in the project context. Students can submit complaints, view their requests, support existing issues, comment, and verify completed work. Maintenance staff receive assigned work through a separate portal and provide progress and resolution details.

TABLE I. ROLE-BASED ACCESS AND PRIMARY RESPONSIBILITIES

Role	Primary operations documented in the system
Student	Authenticate; submit and track complaints; attach images or short videos; support existing issues; comment; verify maintenance; reopen unresolved complaints; submit feedback.
Administrator	Manage hostel hierarchy and student records; import students by CSV; configure categories and SLA values; review, approve or reject complaints; adjust priority with a reason; assign internal staff or record external work; view reports, heat maps, and audit information.
Maintenance Staff	Access assigned complaints; inspect location, description, priority, and evidence; update work status; add resolution details; mark maintenance work as completed.

C. *System Architecture*

HostelDesk follows a client-server architecture with a React-based client layer, an Express.js application layer, a PostgreSQL data layer, and a media-handling component. The client layer contains student, administrator, and maintenance portals. Requests are exchanged with the backend through REST-style API operations, represented as HTTPS/JSON in the architecture diagram. The backend performs authentication, authorization, request validation, business processing, and database access. Multer handles multipart media uploads, which are stored in the server-side upload area in the current implementation.

The backend exposes services for authentication and user management, complaint management, hostel infrastructure, priority and SLA management, maintenance assignment and tracking, verification and feedback, reports and analytics, notifications, and audit logs. PostgreSQL stores user and role records, hostel structures, student allocations, complaints, media metadata, support and comment records, assignments, status and priority histories, verification and feedback, notifications, audit logs, and system settings. Figure 1 summarizes these layers and their relationships.

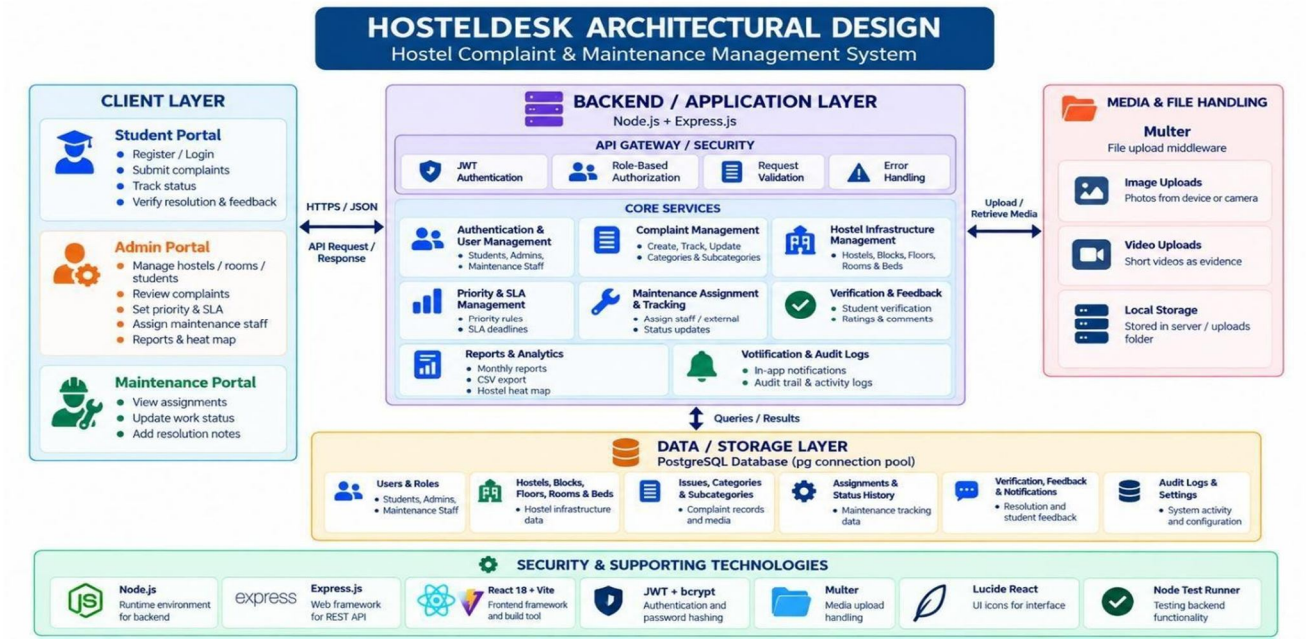


Fig. 1. HostelDesk Architecture Showing Client, Application, Media-Handling, and Data Layers

D. Complaint Lifecycle and Data Flow

A student begins by authenticating and selecting the relevant complaint category and subcategory. The submission includes a description and the associated hostel location; optional images and short video evidence can also be attached. The backend validates the submitted information, creates a unique complaint reference, and stores the complaint and related media metadata. Administrators review the complaint, approve or reject it, determine or override its priority, and assign maintenance work to an internal staff member or record an external maintenance source.

The worker portal presents assigned complaints and allows the responsible person to update progress and provide resolution details. When the maintenance task is marked completed, the complaint enters verification. The student either confirms that the issue has been resolved or reports that it remains unresolved. An unresolved issue can be reopened or escalated; a verified resolution can be closed and rated. The flow retains complaint, status, priority, assignment, verification, feedback, and audit information for later reporting. Figure 2 presents the end-to-end flow.

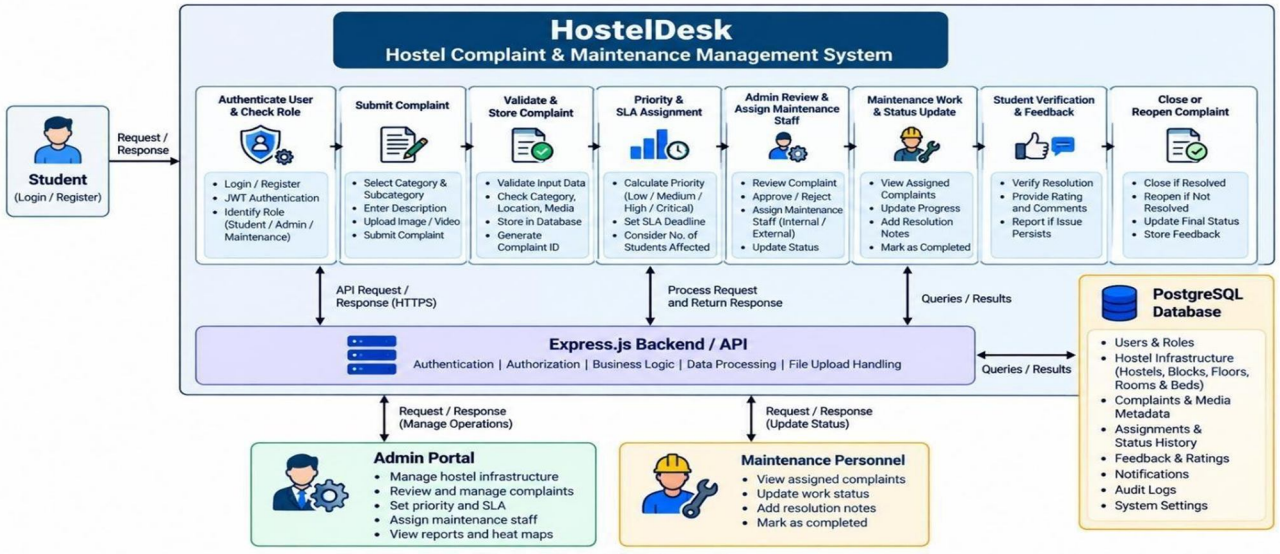


Fig. 2. Data Flow from Student Complaint Submission through Maintenance and Student Verification

E. Priority, SLA, and Escalation Model

Priority is determined using factors including the severity associated with the selected complaint category and the number of students affected. The affected count may include the original reporter, students supporting the issue, and related reports. The system supports LOW, MEDIUM, HIGH, and CRITICAL classifications. It uses configured thresholds to determine a priority when sufficient hostel population information is available; administrators may override the calculated value and must record a reason for the override. The project report specifies the default SLA periods shown in Table II. These values are configuration defaults documented for the prototype, not measured average resolution times or guaranteed service outcomes. The module also supports identification of overdue and escalated complaints. The exact priority scoring equation and numerical thresholds are not specified in the source documentation and are therefore not inferred in this paper.

TABLE II. DOCUMENTED DEFAULT SLA PERIODS BY COMPLAINT PRIORITY

Priority	Configured default SLA
LOW	72 hours
MEDIUM	48 hours
HIGH	24 hours
CRITICAL	12 hours

F. Hostel Infrastructure and Student Data

The system represents the physical hostel structure through the hierarchy Hostel → Block → Floor → Room → Bed. Relationships are maintained between adjacent levels: blocks belong to hostels, floors to blocks, rooms to floors, and beds to rooms. Room numbers are scoped to their block rather than treated as globally unique. This hierarchy supports student allocation, validation of selected locations during complaint registration, and hostel/block-based reporting and heat-map analysis.

Administrators manage student records containing register number, name, gender, date of birth, email, department, and hostel allocation information. Bulk onboarding is supported through CSV import. The import process validates required values and hostel/block/room/bed relationships before insertion. PostgreSQL transactions are used so that failed bulk imports do not leave partially inserted student records in an inconsistent state.

IV. IMPLEMENTATION

A. Application Stack

The documented implementation uses React 18 for the frontend and Vite as its build tool. Node.js provides the server-side JavaScript runtime, while Express.js implements the REST API and application logic. PostgreSQL is the primary relational database and is accessed through the PostgreSQL pg driver. JWT supports authenticated API requests, bcrypt is used for password hashing for administrator and maintenance accounts, Multer processes complaint-media uploads, and Lucide React supplies interface icons. The principal technologies are summarized in Table III [4]-[10].

Table III. Technologies Used In The Documented Implementation

Layer / purpose	Technology	Documented responsibility
Frontend	React 18, Vite, responsive CSS, Lucide React	Role-specific responsive screens, forms, dashboards, status indicators, and navigation.
Backend	Node.js, Express.js, JavaScript ES Modules	REST APIs, authentication middleware, validation, business logic, and workflow operations.
Database	PostgreSQL, pg driver	Relational persistence for users, hostel structures, complaints, assignments, media metadata, history, feedback, notifications, and audit records.
Authentication	JWT, bcrypt	Authenticated API requests, server-side role authorization, and password hashing for administrator and maintenance accounts.
Media handling	Multer	Multipart image/video upload processing and validation.
Development and testing tools	npm, Git/GitHub, Node.js native test runner, browser testing, Postman or equivalent	Dependency management, version control, backend/browser checks, and API testing.

B. Authentication and Role Authorization

Student login uses a register number and date of birth, whereas administrator and maintenance personnel authenticate with their respective credentials. Following successful authentication, the backend issues a JSON Web Token. Protected API requests include the token, which is validated by authentication middleware. A separate role-authorization middleware checks the authenticated user role before allowing access to role-specific routes. Thus, a student cannot perform administrative or worker operations merely by navigating to a protected frontend screen.

The implementation documentation includes an Express.js initialization example and middleware that reads a Bearer token, verifies it with the configured JWT secret, and rejects missing, invalid, or expired tokens with an unauthorized response. The role-checking function returns a forbidden response when the authenticated role is not included in the permitted role list. Passwords for administrator and maintenance accounts are described as bcrypt hashes rather than plaintext values. The report also specifies server-side validation, parameterized PostgreSQL queries, and controlled database access as security requirements.

C. Complaint Registration and Media Processing

The complaint form records the selected category, subcategory, description, and location. The backend checks the subcategory against the selected category before creating the record and generating a unique complaint reference. Students can attach multiple images and short video evidence. Where the device and browser support it, the interface offers camera capture; otherwise, users can select media from the gallery or file picker.

Multer processes multipart form-data, and uploaded files are validated for file type, extension, size, and video duration before association with a complaint. The current implementation stores uploaded media in a server-side upload directory. Cloud object storage is not part of the documented implementation and is identified as a possible future enhancement. Students can also support an existing complaint and add comments, enabling a shared issue to reflect multiple affected users rather than treating every report as an unrelated maintenance request.

D. Maintenance Assignment, Status, and Verification

After administrator review and approval, an issue can be assigned to internal maintenance personnel or recorded as work handled by an external maintenance source. Internal workers access a dedicated portal containing their assigned complaints and relevant location, description, priority, and evidence. They can update progress and enter resolution details. Assignment and status changes are retained in the complaint history. The documented primary lifecycle is Reported, Under Review, Approved, Assigned, In Progress, Maintenance Completed, Verification, and Closed. Alternative workflow states include Reopened, Escalated, Overdue, Rejected, and Cancelled. Completion by the worker does not itself establish final closure: the student is asked to verify the reported outcome. If the issue persists, the complaint may be reopened or escalated. After successful verification, the student can submit a rating and an optional comment.

E. Reporting, Notifications, and Audit History

Administrative reporting includes dashboards, monthly complaint reports, CSV export, technician performance information, and hostel/block heat-map analysis. Reports can be viewed by complaint category, priority, status, time period, maintenance personnel, and location. The heat-map view is intended to highlight hostel areas with a higher concentration of reported issues. In-application notifications communicate relevant workflow updates, while audit logs record important activities such as the responsible user, affected entity, timestamp, and relevant changes.

F. Development Environment and Deployment Requirements

The project documentation describes a Windows-based development environment and states that the application architecture can also be deployed on Linux or other supported server environments. The documented software environment includes Node.js, Express.js, PostgreSQL, React 18, Vite, npm, Git/GitHub, browser testing, and API testing. The stated baseline server requirement is a 1.4 GHz dual-core processor or better, at least 2 GB RAM, and 20 GB available storage. Client devices are specified as modern desktop, laptop, tablet, or smartphone devices with a modern browser, a minimum 1 GHz processor, and 2 GB RAM; a camera-enabled device is recommended for direct image capture. These are stated system requirements, not a measured experimental test configuration.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

V. RESULTS AND DISCUSSION

A. Functional Outcomes

The project report describes HostelDesk as a functional web-based prototype integrating authentication, hostel infrastructure management, CSV-based student onboarding, complaint registration, issue classification, media uploads, priority and SLA handling, staff assignment, status tracking, verification, feedback, administrative reporting, notifications, and audit logging. The principal functional outcome is a connected workflow in which a complaint can be followed from initial student submission to maintenance work and student-confirmed resolution.

The documented interface captures provide visual evidence of the main role-specific screens. The login view presents Student, Admin, and Maintenance Staff role selection. The administrator dashboard displays operational summary cards and complaint-category counts, while a separate heat-map view filters by hostel, category, and status and summarizes affected hostels and blocks. The student view lists reported issues and provides complaint support and navigation to issue details. The maintenance worker view displays active assignments and offers work-status actions, including starting repair and marking work completed. These screens are consolidated in Fig. 3.

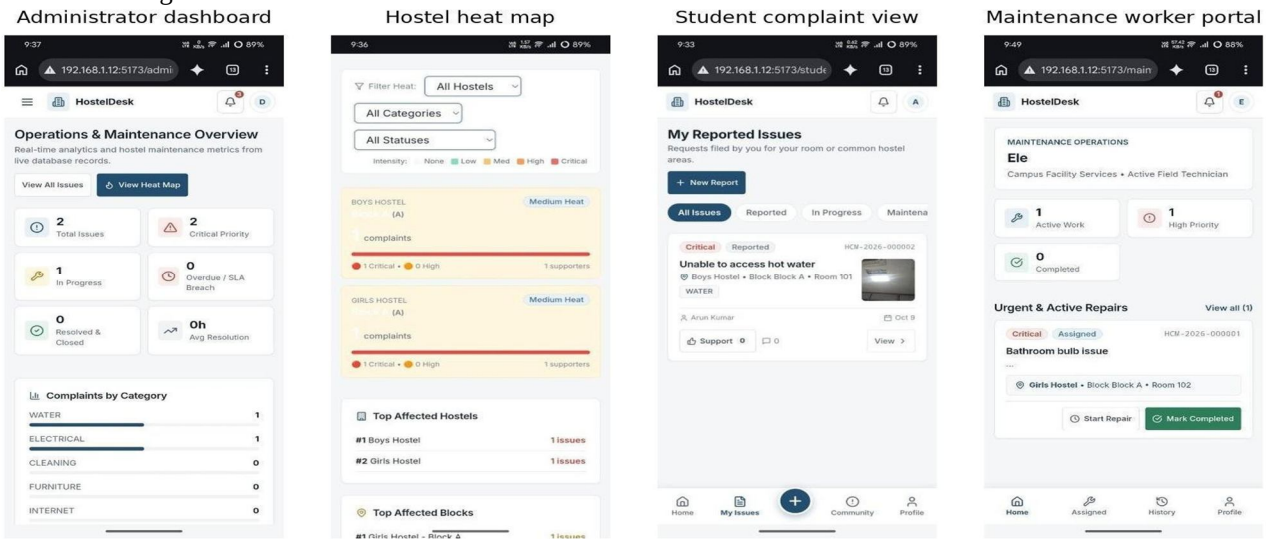


Fig. 3. Selected HostelDesk Interface Evidence: Administrator Dashboard, Hostel Heat Map, Student Complaint View, and Maintenance Worker Portal

B. Discussion of System Characteristics

The integration of complaint support and affected-student counting provides administrators with information about the reach of a shared issue. The combination of category severity and affected-student information provides the basis for priority determination, while administrator override with a recorded reason retains human control over exceptional cases. The explicit verification stage is another distinguishing workflow characteristic: maintenance completion is followed by student confirmation, and unresolved issues can re-enter the workflow through reopening or escalation. The structured hostel hierarchy connects maintenance activity to physical locations and makes hostel/block heat-map analysis possible. Historical status, priority, assignment, verification, feedback, and audit records can also support retrospective administrative review. However, the project report does not provide an experimental comparison against the previous manual process, a user study, complaint-resolution-time measurements, SLA compliance rates, response-latency measurements, load-test results, or statistical evidence of improved satisfaction. Consequently, this paper reports the implemented functionality and documented interface evidence without asserting quantified efficiency gains.

C. Evaluation Scope and Limitations

The project documentation lists the Node.js native test runner, browser-based testing, and API testing using Postman or an equivalent tool as development/testing resources. It does not include a test-case inventory, coverage percentage, pass/fail summary, sample size, or measured system performance. In addition, the exact priority formula and population thresholds are not stated. The current media-storage approach is server-local rather than cloud-based, and notification channels are limited to in-application notifications in the documented implementation. These omissions define the evidence boundary for the present results and identify areas that require further validation.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

VI. CONCLUSION AND FUTURE WORK

This paper presented HostelDesk, a web-based Hostel Complaint and Maintenance Management System designed around the needs of a student hostel environment. The system combines role-based access, structured hostel and room allocation, CSV student onboarding, categorized complaint registration, image and short-video evidence, collaborative complaint support, severity- and affected-student-informed priority, configurable SLA deadlines, internal or external maintenance assignment, worker progress updates, student verification, feedback, reporting, heat maps, notifications, and audit history. The implementation uses React 18 and Vite, Node.js and Express.js, PostgreSQL, JWT, bcrypt, and Multer.

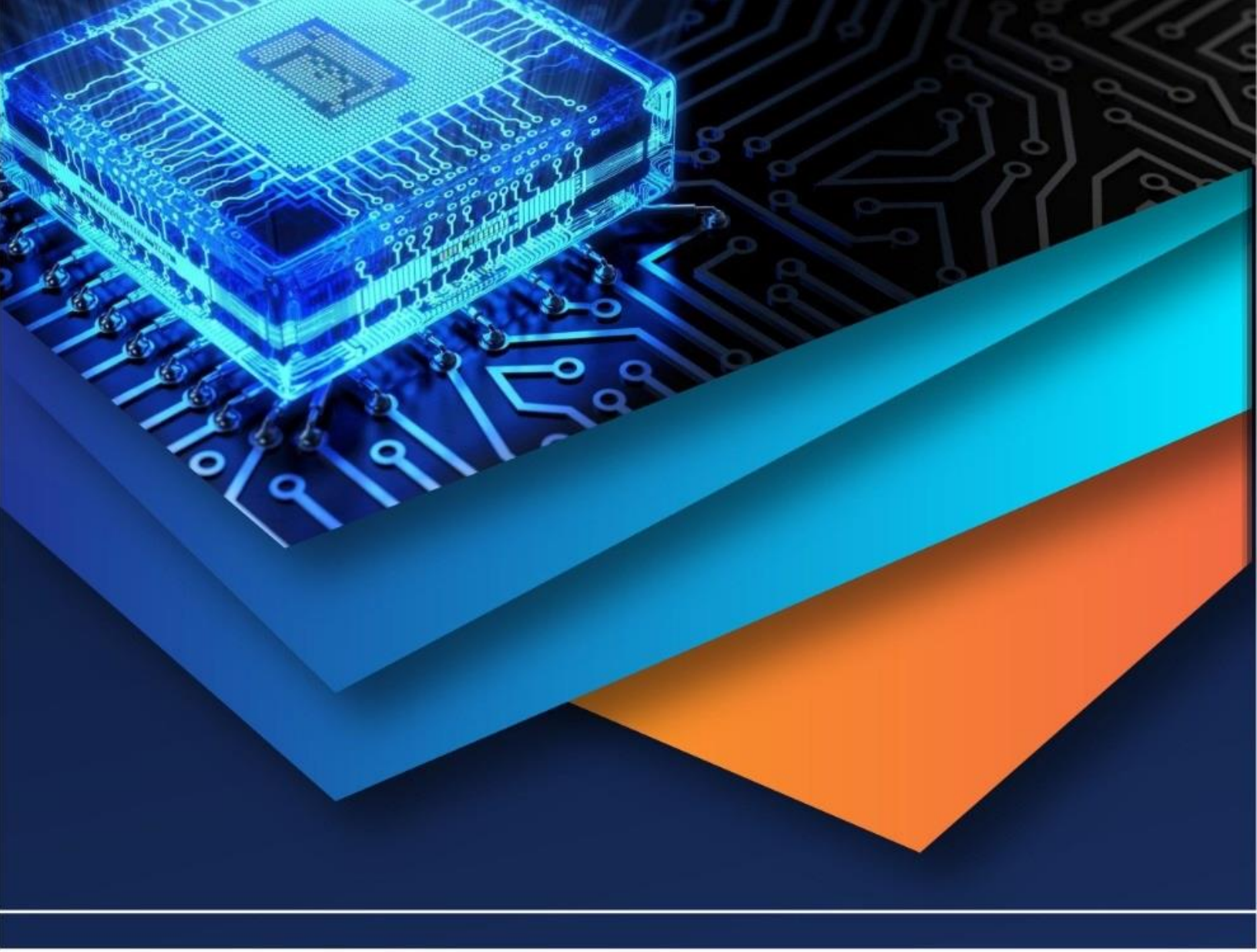
The documented prototype establishes an integrated path from complaint submission to maintenance completion and student verification. Its operational contribution lies in maintaining consistent records and connecting users, locations, work assignments, and outcome confirmation. Since quantitative evaluation results are not supplied, claims about reduction in resolution time, improved service levels, or user satisfaction remain unverified and should be assessed through a future controlled or field-based evaluation.

Future enhancements identified in the project report include email, SMS, and push notifications; a dedicated mobile application; cloud-based media storage; inventory management for maintenance materials and spare parts; QR-code-based room identification; automated escalation for overdue complaints; advanced analytics for recurring issues; intelligent complaint classification; and duplicate complaint detection. Future evaluation should also define the priority formula and thresholds, document test coverage and deployment conditions, and measure response time, SLA compliance, recurrence, and user acceptance.

REFERENCES

[1] MaintainX, “Facility Management,” MaintainX. [Online]. Available: https://www.getmaintainx.com/use-cases/facility-management. (Accessed: 2023).
[2] MaintainX, “Work Request Management,” MaintainX. [Online]. Available: https://www.getmaintainx.com/use-cases/work-request-management. (Accessed: 2024).
[3] ServiceNow, “Workplace Maintenance Management,” ServiceNow. [Online]. Available: https://www.servicenow.com/in/products/workplace-maintenance-management.html. (Accessed: 2025).
[4] PostgreSQL Global Development Group, “PostgreSQL Documentation.” [Online]. Available: https://www.postgresql.org/docs/. (Accessed: 2023).
[5] OpenJS Foundation, “Node.js Documentation.” [Online]. Available: https://nodejs.org/docs/latest/api/. (Accessed: 2024).
[6] Express.js, “Express.js Documentation.” [Online]. Available: https://expressjs.com/. (Accessed: 2025).
[7] Meta Platforms, Inc., “React Documentation.” [Online]. Available: https://react.dev/. (Accessed: 2023).
[8] Vite Contributors, “Vite Documentation.” [Online]. Available: https://vite.dev/guide/. (Accessed: 2024).
[9] Auth0, “JSON Web Tokens (JWT).” [Online]. Available: https://jwt.io/introduction. (Accessed: 2025).
[10] Express.js, “Multer: Node.js Middleware for Handling Multipart/Form-Data.” [Online]. Available: https://github.com/expressjs/multer. (Accessed: 2026).

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)