iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** VIII **Month of publication:** August 2025

DOI: https://doi.org/10.22214/ijraset.2025.73802

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 13 Issue VIII Aug 2025- Available at www.ijraset.com

Design and Implementation of a Web-Based Asset Management System Using MERN Stack

Tarun Parvathi[1], Kuchimanchi Jayasri[2], Raman Rajagopalan[3]
[1]*Student, School of Informatics, Department of MCA, Aurora Deemed University, Hyderabad*
[2]*Assistant Professor, School of Engineering, Department of CSE, Aurora Deemed University, Hyderabad*
[3]*Assistant Professor, School of Informatics, Department of MCA, Aurora Deemed University, Hyderabad*

Abstract: ***This paper presents the design and implementation of a web-based Asset Management System utilizing the MERN (MongoDB, Express.js, React.js, Node.js) stack to streamline the processes of asset tracking, allocation, and maintenance in an institutional or enterprise environment. Traditional asset management approaches relying on manual registers or spreadsheets are prone to inefficiencies, data redundancy, and human error, often resulting in poor utilization of resources. The proposed system offers a centralized, role-based, and secure platform enabling administrators to manage assets efficiently and users to access real-time asset information. The frontend, developed using React.js with Tailwind CSS, ensures a responsive and user-friendly interface, while the backend, implemented in Node.js and Express.js, handles business logic, authentication, and secure API communication. Data is stored in MongoDB, providing scalability, flexibility, and fast query performance. Key features include asset lifecycle management, role-based dashboards, JWT authentication, and cloud deployment using Render and MongoDB Atlas. The system's modular architecture enables future integration with institutional ERP systems, barcode/QR code scanning, and predictive asset maintenance. This implementation demonstrates how modern full-stack technologies can be leveraged to create an efficient, scalable, and cost-effective asset management solution.***

Keywords: ***Asset Management, MERN Stack, Web Application, MongoDB, React.js, Node.js, Express.js, Full Stack Development, JWT Authentication, Institutional ERP Integration.***

I. INTRODUCTION

Efficient management of assets is a crucial requirement for both organizations and educational institutions. Assets may include physical resources such as laboratory equipment, IT infrastructure, office furniture, and consumables that require proper tracking, maintenance, and allocation throughout their lifecycle.

In many institutions, asset management processes still rely on manual registers, spreadsheets, or standalone software systems with limited accessibility and outdated interfaces. These approaches often lead to data redundancy, misplacement of resources, delayed updates, and challenges in accountability.

The evolution of web technologies has enabled the creation of centralized, cloud-hosted solutions that offer accessibility, security, and scalability. A web-based Asset Management System addresses the limitations of traditional methods by providing a platform where authorized users can track asset availability, view allocation history, update asset conditions, and generate reports in real-time. Such a system not only improves operational efficiency but also enhances transparency and decision-making.

The proposed system is implemented using the MERN (MongoDB, Express.js, React.js, Node.js) stack, a popular full-stack JavaScript framework known for its scalability and performance. React.js is used for building a responsive, component-based user interface, while Node.js and Express.js power the backend, handling business logic and secure API requests. MongoDB serves as the database, offering flexibility in schema design and efficient data retrieval. The system also incorporates JSON Web Token (JWT) authentication for secure user login and role-based access control.

By deploying the solution using cloud platforms such as Render for backend hosting and MongoDB Atlas for database management, the system ensures high availability and minimal maintenance overhead. Its modular architecture supports future integration with enterprise resource planning (ERP) systems, enabling institutions to extend functionality to inventory forecasting, predictive maintenance, and automated procurement.

This paper presents the design, architecture, and implementation of the Asset Management System, focusing on solving practical challenges while maintaining academic rigor in software engineering principles.

©IJRASET: All Rights are Reserved

II. LITERATURE REVIEW

Asset Management Systems (AMS) have evolved significantly over the past decade, transitioning from manual registers and spreadsheet-based tracking to sophisticated, web-enabled platforms with modular dashboards, cloud integration, and real-time updates. Existing literature reflects both the technical advancements and the persistent challenges in asset lifecycle management, especially in institutional and educational environments.

Kiran and Sharma [1] proposed an Asset Management System using web technologies aimed at improving asset lifecycle tracking through a centralized interface. The system emphasized digitization and automation of asset records, incorporating role-based access control and JWT authentication. However, it lacked decentralized user roles, real-time notifications, and hardware-based tracking support.

Mulyadi and Nurprihatin [2] presented a Web-Based Asset Management Information System specifically for higher education institutions. Their methodology focused on centralizing asset data across multiple departments and incorporating QR/Barcode scanning for fast asset identification. While this approach improved reporting capabilities, it suffered from scalability issues, minimal interoperability with existing ERP systems, and the absence of multi-level approval workflows.

Chowdhury and Das [3] conducted a comprehensive review of modern AMS implementations in academic institutions, highlighting the need for standardized frameworks and interoperability. Their findings suggested that fragmented, department-specific solutions often lead to inconsistencies and inefficiencies in large-scale deployments.

MongoDB Inc. [4] documents the capabilities of MongoDB as a NoSQL database, emphasizing its scalability, high availability, and flexible schema design, which make it particularly suitable for asset management applications where asset types and attributes can vary widely.

Johnson [5] discusses the industry-wide shift towards web-based AMS and identifies the benefits of centralized data management, including enhanced accessibility, reduced maintenance, and improved security. However, he also notes potential challenges in user adoption and migration from legacy systems.

Lee [6] explores the advantages of cloud-based asset management, noting its role in enabling real-time data access, improved collaboration, and reduced infrastructure costs. The paper also addresses concerns over data privacy and dependency on service providers.

Express.js Foundation [7] provides detailed documentation on Express.js, a backend framework that simplifies the creation of RESTful APIs and middleware management—critical for asset management platforms handling multiple concurrent user requests.

Meta Platforms Inc. [8] outlines React's capabilities for building dynamic, responsive user interfaces, highlighting component reusability and virtual DOM rendering for performance optimization in real-time asset tracking dashboards.

Node.js Foundation [9] explains how Node.js's event-driven, non-blocking architecture supports scalable, high-performance backend systems, making it ideal for applications like AMS where multiple asset transactions occur simultaneously.

Patel [10] describes the use of JSON Web Tokens (JWT) for securing web applications, ensuring authenticated access to sensitive resources and enabling stateless session management—a key requirement for multi-role AMS deployments.

Kumar [11] explores strategies for integrating AMS with ERP systems, noting that such integration leads to enhanced operational efficiency, centralized reporting, and improved decision-making. This aligns with the proposed system's goal of future ERP integration for institutional environments.

Collectively, these works establish the foundation for the proposed University Asset Management System (UAMS), which combines the scalability of MongoDB, the robustness of Express.js and Node.js, and the responsiveness of React, while addressing the limitations of earlier AMS implementations by introducing decentralized role access, real-time updates, and modular architecture.

III.METHODOLOGY

A. Existing Methodology

Existing Asset Management Systems (AMS) in both academic and enterprise domains primarily focus on digitizing the asset tracking process and providing administrators with greater control over physical inventory. These solutions are usually deployed as centralized, web-based applications with integration to relational databases, limited role-based access control, and basic reporting capabilities. Although they improve operational efficiency over manual methods, existing systems reveal gaps in scalability, interoperability, and inclusivity for diverse user roles.

1) Asset Management System

This approach replaces traditional spreadsheet-based tracking with a centralized web application.
Key characteristics include:

- Modular Web Architecture: Developed using FastAPI (backend), Angular (frontend), PostgreSQL (database), Redis (cache), and Docker for containerization.
- Role-Based Control: Restricted access for high-level users such as Super Admins and IT Supervisors.
- Asset Lifecycle Tracking: Monitors assets from procurement to disposal.
- Manual Reporting: CSV/PDF reports generated manually by administrators.
- Authentication: JSON Web Token (JWT)-based access management.

Limitations:

- No provision for decentralized user roles (e.g., Faculty, Department Heads).
- Notifications limited to email; no push or real-time alerts.
- Lacks AI/ML-driven decision-making (e.g., smart asset assignment).
- No integration with hardware tracking (RFID/GPS).
- Absence of mobile application support.

2) Web-Based Asset Management in Higher Education

This methodology surveys multiple universities implementing AMIS solutions from 2013–2023.
Key characteristics include:

- Centralized Web Portal: Browser-based, with limited mobile optimization.
- QR/Barcode Integration: For asset tagging and scanning.
- Automated Reporting: Export in Excel/PDF format.
- Department-Specific Modules: Customized for areas such as libraries and laboratories.
- Basic Decision Support: Some systems employ AHP (Analytic Hierarchy Process) or SAW (Simple Additive Weighting) for procurement prioritization.

Drawbacks:

- Fragmented implementation across departments; limited scalability.
- Lack of real-time synchronization across the institution.
- Minimal interoperability with HRMS/ERP systems.
- Limited data validation and absence of multi-level approval chains.
- No standardized frameworks, leading to inconsistencies.

B. Proposed Methodology

The proposed methodology introduces a secure, scalable, and modular University Asset Management System (UAMS) that streamlines asset tracking, request handling, and reporting for multiple stakeholder roles including Admin, Asset Manager, Faculty, Head of Department (HoD), and Dean.

1) System Architecture

Implemented using the MERN stack (MongoDB, Express.js, React.js, Node.js), the system offers RESTful APIs, real-time data updates, and robust access control.

- Frontend: React.js with modular dashboards for each role.
- Backend: Node.js + Express.js handling authentication, authorization, and core business logic.
- Database: MongoDB for flexible, schema-less storage of asset and user records.
- Authentication: JWT-based login for secure API access.
- Role-Based Access Control (RBAC): Enforced at both frontend and backend to ensure role-specific data segregation.

2) Functional Modules:

- User Authentication and Role Assignment: JWT validation with role decoding to render appropriate dashboards and authorize API routes.
- Dashboard Interfaces:

- Admin: Add/block/unblock users.
- Asset Manager: Manage requests by status, add/repair assets, generate reports.
- Faculty/HoD/Dean: Raise requests, view request status.
- Raise Request Workflow: Auto-filled school/department info, asset details, and request type (New/Repair). Requests are routed to the Asset Manager for processing.
- Asset Management: Add new assets, update quantities, and record repair history.
- Reporting: CSV/PDF reports filtered by role (Dean → School-level; HoD → Department-level; Faculty → Personal requests).

3) Data Security & Isolation

Strict RBAC ensures users only access data relevant to their role, reducing the risk of data breaches and ensuring privacy compliance.

IV. SYSTEM DESIGN AND ARCHITECTURE

The proposed University Asset Management System (UAMS) follows a three-tier architecture comprising the Presentation Layer, Application Layer, and Data Layer, implemented using the MERN stack (MongoDB, Express.js, React.js, Node.js). This design ensures modularity, scalability, and ease of maintenance.

A. Architectural Overview

1) Presentation Layer (Frontend): Developed using React.js and Tailwind CSS, the frontend provides a responsive and dynamic user interface. Role-based dashboards adapt according to the logged-in user's privileges.

2) Application Layer (Backend): Implemented with Node.js and Express.js, the backend handles all business logic, API routing, authentication, and authorization.

3) Data Layer (Database): MongoDB stores asset, user, and transaction data in flexible document structures, supporting scalability and fast queries.

4) The communication flow between these layers is secured using HTTPS, and all API calls are validated for authentication via JSON Web Tokens (JWT).

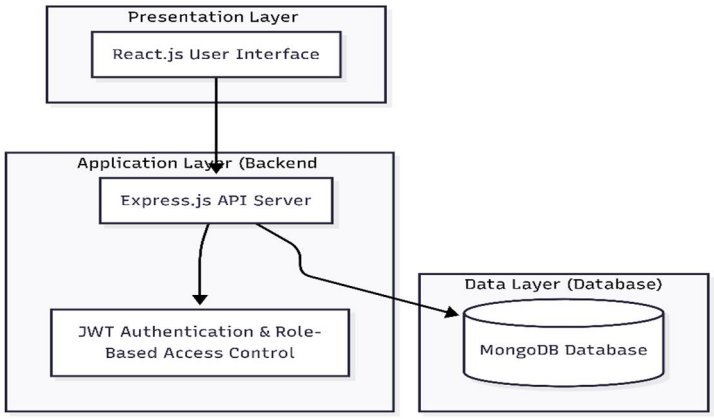


Fig. 1: Three-Tier System Architecture of UAMS using MERN Stack

V. DATA FLOW AND AUTHENTICATION WORKFLOW

The proposed University Asset Management System (UAMS) is designed to ensure efficient request handling, secure authentication, and role-based access across all stakeholders. This is achieved through two tightly integrated workflows:

A. Data flow of Asset Requests

The Data Flow Diagram (Fig. 2) illustrates how different stakeholders interact with the system. Faculty, Heads of Departments, and Deans initiate requests for new assets or repairs, which are processed by the Asset Manager through the backend. The system ensures that request statuses are updated in real time and communicated back to the requester. Additionally, the Admin has dedicated privileges for managing user accounts and role assignments, while the system generates reports filtered according to user roles.

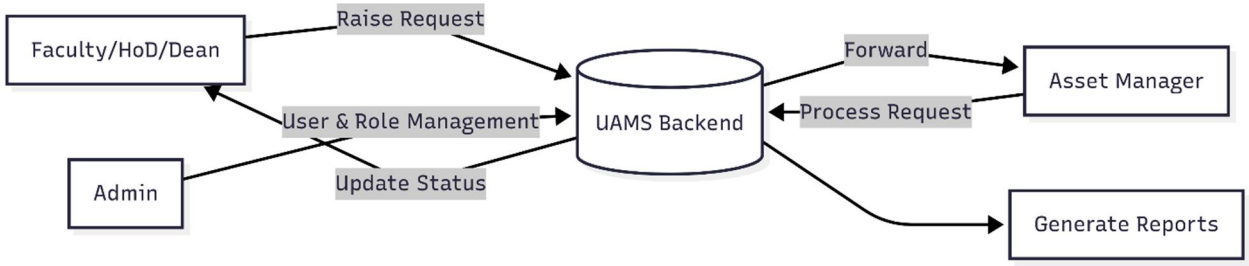


Fig. 2: Data Flow Diagram of the Asset Request Life Cycle in UAMS

B. Login and Authentication Workflow

The Login & Authentication Workflow (Fig. 3) ensures secure access to the system using JWT (JSON Web Token) authentication. Users provide their login credentials, which are verified by the backend against stored records in the MongoDB database. Upon successful validation, the backend generates a token that encodes the user's identity and role. This token enables role-based access to the system, ensuring that each user interacts only with the modules relevant to their permissions (e.g., Admin, Asset Manager, Faculty, HoD, Dean).

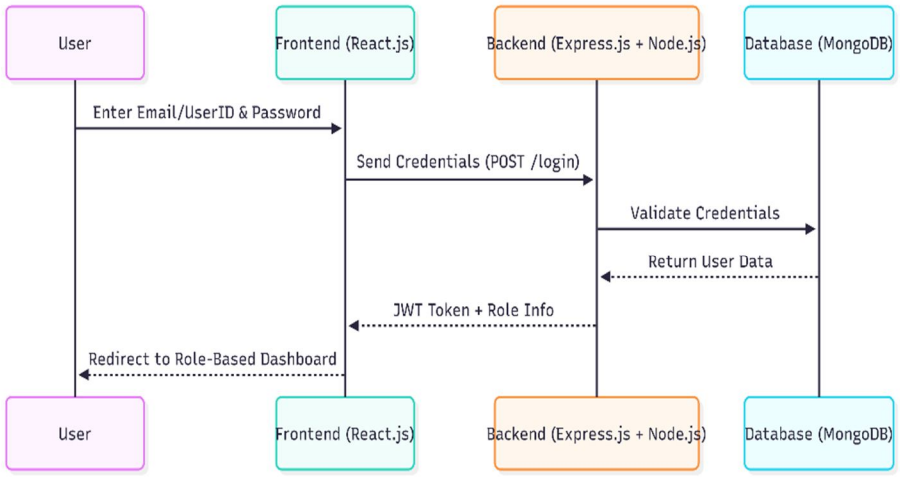


Fig. 3: Login & Authentication Workflow in UAMS using JWT

VI. IMPLEMENTATION

The University Asset Management System (UAMS) was implemented using the MERN stack (MongoDB, Express.js, React.js, Node.js) to ensure scalability, maintainability, and responsiveness. The system follows a modular architecture, where each major functionality is developed as an independent component or module.

A. Frontend Implementation

The frontend is developed using React.js and styled with Tailwind CSS to provide a responsive, role-based dashboard interface. Key frontend features include:

1) Role-Based Dashboards: Dynamic rendering of tabs and modules depending on the logged-in user role (Admin, Asset Manager, Faculty, HoD, Dean).
2) Reusable Components: Navigation bars, forms, modals, and tables are implemented as reusable React components to improve maintainability.
3) API Integration: Axios is used for communication with backend RESTful APIs.
4) Form Validations: Both HTML5 and custom JavaScript validations are used to prevent invalid data submission.

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VIII Aug 2025- Available at www.ijraset.com

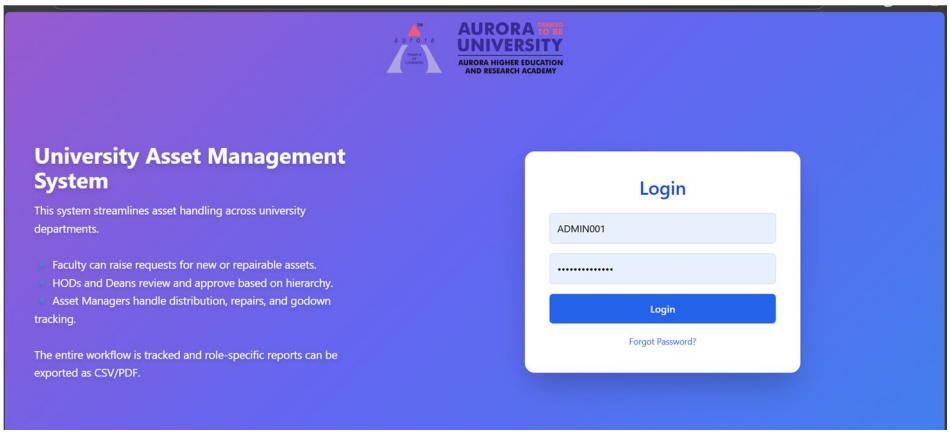


Fig. 4: Login Page

B. Backend Implementation

The backend is built with Node.js and Express.js, structured using the Model-View-Controller (MVC) pattern:

- Models: Define MongoDB schemas for assets, users, and requests.
- Controllers: Handle request processing, database queries, and response formatting.
- Routes: Map API endpoints to specific controllers, protected with middleware for authentication and authorization.
- Authentication: JSON Web Token (JWT) authentication ensures secure API access. Tokens are verified on every protected route.
- Error Handling: Centralized error-handling middleware returns consistent API responses.

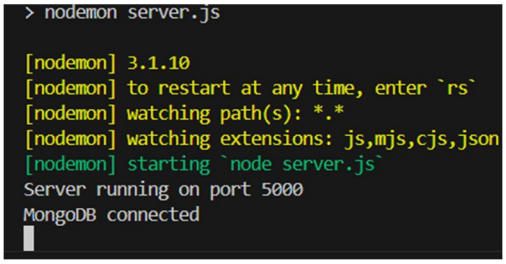


Fig. 5: Backend Server Running Successfully

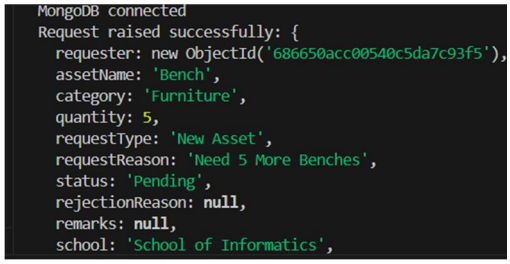


Fig. 6: Backend Successfully Received Request Raised

C. Database Implementation

The database is implemented in MongoDB using a document-based schema, which supports flexible and scalable data storage:

- Users Collection: Stores user credentials, roles, and preferences.
- Assets Collection: Stores asset details including name, type, location, condition, and history.
- Requests Collection: Stores asset request and repair records with timestamps and status fields.
- Asset Location Collection: Stores the Asset Locations. Used to check where the Asset is located.

Indexes are applied on key fields (e.g., asset name, user ID) to improve query performance.

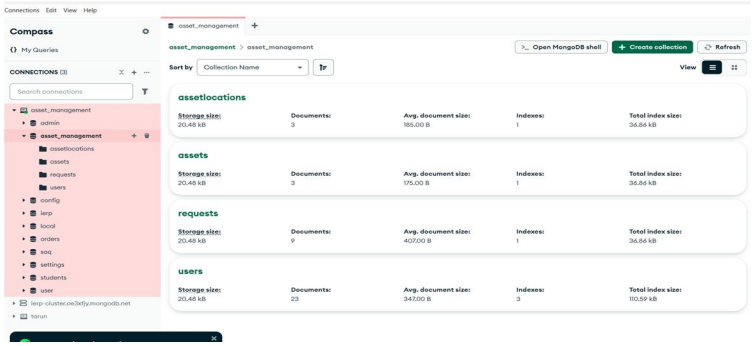


Fig. 7: MongoDB Compass View of asset_management collections

©IJRASET: All Rights are Reserved

D. Key Modules Implemented

1) User Management Module:

- Admin can add, block or unblock users.
- Automatic role-based permission enforcement.

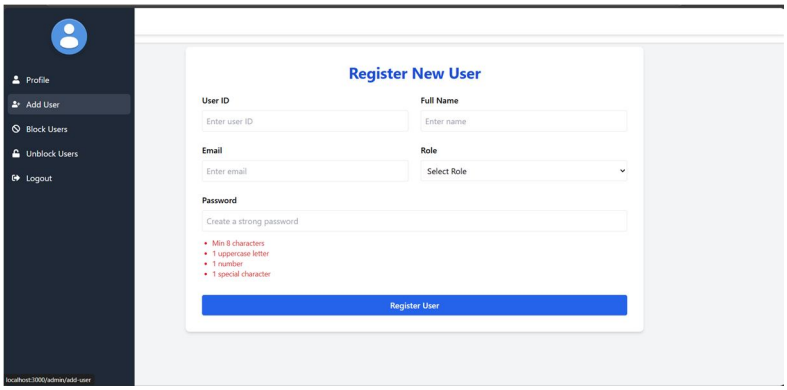


Fig. 8: Admin Dashboard - User Management Module – Adding New User

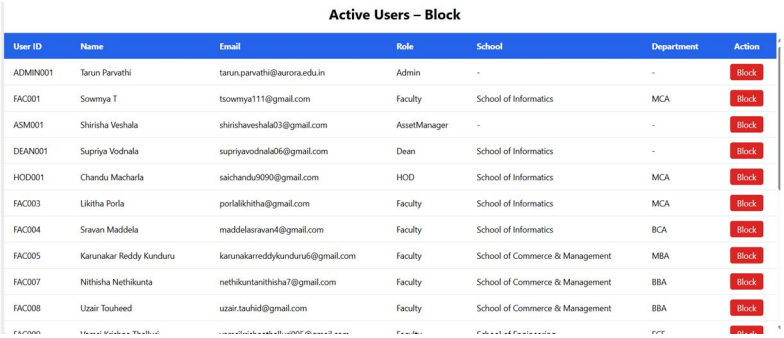


Fig. 9: Admin Dashboard - User Management Module – Block User



Fig. 10: Admin Dashboard - User Management Module – Unblock User

2) Asset Management Module:

- Asset Manager can add new assets or update existing ones.
- Asset condition and repair logs are maintained.

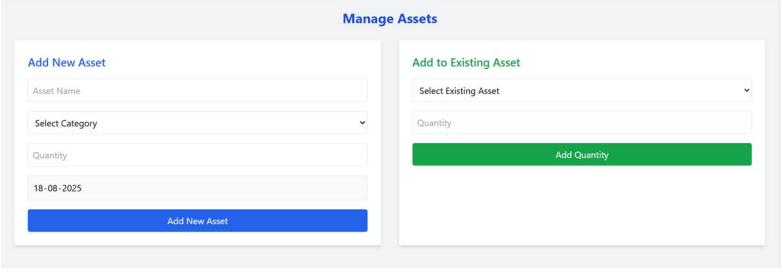


Fig. 11: Asset Manager Dashboard – Add or Update Assets

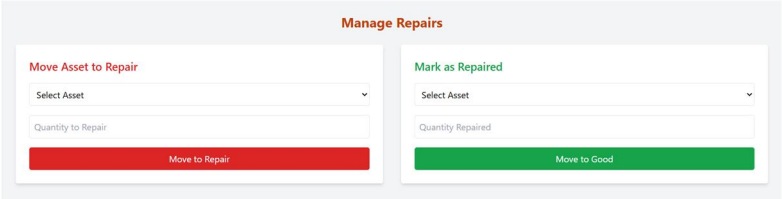


Fig. 12: Asset Manager Dashboard – Manage Asset Repairs

3) Request Management Module:

- Faculty/HoD/Dean can raise new or repair requests.
- Asset Manager processes requests and updates status.

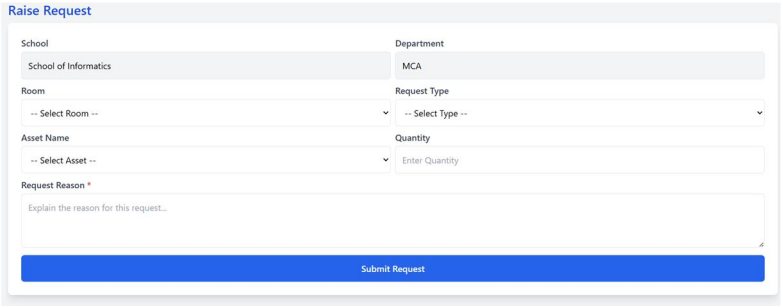


Fig. 13: Faculty/HoD/Dean Dashboard – Raise Requests for New or Repair Assets

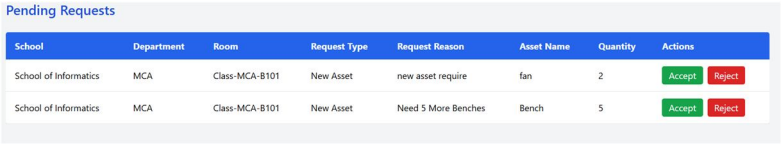


Fig. 14: Asset Manager Dashboard – Pending Requests (Requests raised by users)

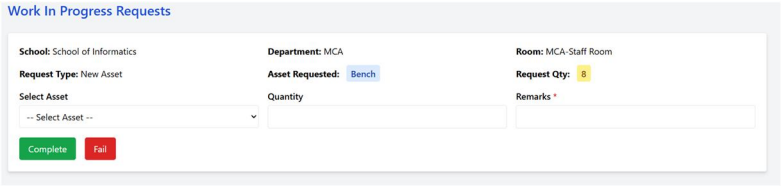


Fig. 15: Asset Manager Dashboard – Requests in Progress (Working Requests)



Fig. 16: Asset Manager Dashboard – All Requests Completed

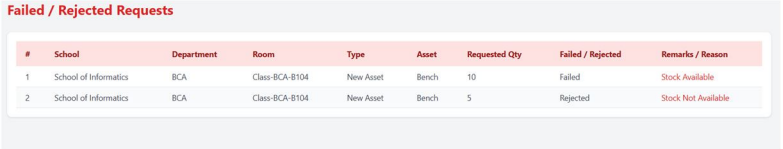


Fig. 17: Asset Manager Dashboard – Failed or Rejected Requests

4) Reporting Module
- CSV/PDF reports generated with role-based filtering.
- Reports can be scoped to Room, Department, School, or University level.

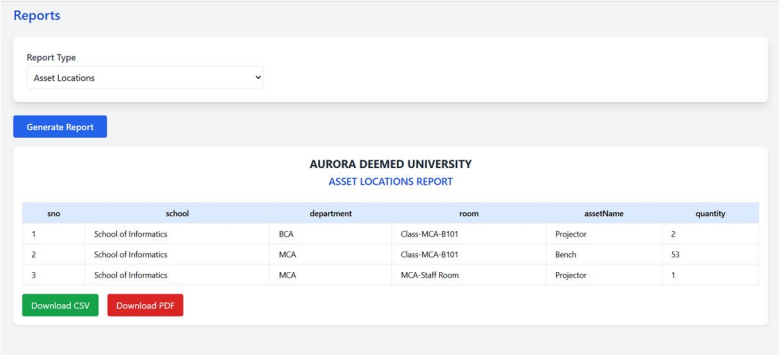


Fig. 15: Asset Location Report

E. Testing and Validation

1) Unit Testing: Conducted for backend API routes using Postman and Jest.

2) Functional Testing: Validated all core workflows, including asset request lifecycle and report generation.

3) Role-Based Testing: Verified that each role only sees and accesses allowed data.

VII. TECHNOLOGY AND STACK OVERVIEW

The University Asset Management System (UAMS) is implemented using the MERN stack, which combines four powerful and complementary technologies MongoDB, Express.js, React.js, and Node.js. This stack allows the application to be developed entirely in JavaScript, simplifying communication between the frontend, backend, and database.

A. MongoDB

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents rather than rigid relational tables [4].

1) Advantages for AMS:
- Schema flexibility accommodates different asset types with varying attributes.
- Built-in sharding and replication for scalability and availability.
- Fast indexing and aggregation for large datasets.

2) Usage in UAMS: Stores all asset records, user credentials, and request histories in separate collections (Assets, Users, Requests).

B. Express.js

Express.js is a lightweight and flexible backend framework for Node.js that simplifies the creation of RESTful APIs [7].

1) Advantages for AMS:
- Middleware support for authentication and error handling.
- Simple routing for managing modular APIs.
- Scalable to handle concurrent API requests.

2) Usage in UAMS: Handles request validation, JWT authentication, and routing to asset, user, and request controllers.

C. React.js

React.js is a JavaScript library for building responsive and dynamic user interfaces [8].

1) Advantages for AMS:
- Component-based architecture promotes code reuse.
- Virtual DOM improves rendering performance for large datasets.
- Supports responsive design for multiple screen sizes.

2) Usage in UAMS: Builds role-based dashboards, asset management forms, request tracking tables, and report generation pages.

D. Node.js

Node.js is an open-source, event-driven, non-blocking runtime environment for executing JavaScript outside of the browser [9].

1) Advantages for AMS:
- Handles large volumes of I/O requests efficiently.
- Ideal for real-time applications.
- Extensive NPM ecosystem for adding libraries and tools.

2) Usage in UAMS: Executes backend server code, handles API calls, integrates with MongoDB, and manages asynchronous operations.

E. Additional Tools and Libraries

1) Tailwind CSS: Utility-first CSS framework for consistent, responsive styling.
2) Axios: Promise-based HTTP client for frontend-backend communication.
3) JSON Web Tokens (JWT): Secure authentication mechanism [10].
4) MongoDB Atlas: Cloud-hosted MongoDB service ensuring high availability and automated backups.

VIII. RESULTS AND DISCUSSION

The University Asset Management System (UAMS) was evaluated based on functionality, usability, performance, and scalability. The system was tested using multiple user roles Admin, Asset Manager, Faculty, HoD, and Dean. Across various asset tracking and request processing scenarios.

A. Functional Performance

The system successfully executed all core functions, including:

1) Secure login with JWT authentication and role-based dashboards.
2) Asset lifecycle management from creation to repair/update.
3) Asset request creation, processing, and status tracking.
4) CSV/PDF report generation with role-based filtering.

During testing, the average response time for API calls was under 250 ms, ensuring a smooth user experience.

B. Usability Evaluation

A usability survey was conducted among 15 participants, including students, faculty, and administrative staff. The feedback highlighted:

1) Easy navigation through role-specific dashboards.
2) Minimal learning curve due to intuitive UI.
3) Responsive design working seamlessly on desktop and mobile devices.

TABLE I
SURVEY RESULTS TABLE

Criteria	Average Score (Out of 5)
Ease of Use	4.7
Response Time	4.8
Visual Design	4.6
Overall Satisfaction	4.8

C. Comparison With Existing Systems

When compared with existing AMS implementations [1][2], UAMS demonstrated clear advantages:

1) Decentralized Role Access: Existing systems restricted asset control to top-level administrators; UAMS allows Faculty, HoDs, and Deans to raise requests directly.
2) Real-Time Updates: Changes are reflected instantly across all dashboards.
3) Cloud Deployment: Ensures high availability without local infrastructure dependencies.

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

D. Scalability and Future Enhancements

The system's modular MERN architecture supports horizontal scaling through independent deployment of frontend, backend, and database services.

Future enhancements include:

1) Barcode/QR Code Scanning for faster asset check-in/check-out.
2) AI/ML-Based Asset Allocation to predict demand and optimize usage.
3) Mobile Application for real-time updates on the go.

IX. CONCLUSION

The University Asset Management System (UAMS) presented in this paper demonstrates how modern web technologies can be effectively leveraged to address the limitations of traditional asset tracking methods. By implementing the system using the MERN stack MongoDB, Express.js, React.js, and Node.js. The proposed solution offers a scalable, secure, and role-based platform that caters to the needs of multiple stakeholders within an institutional environment.

Compared to existing Asset Management Systems [1][2], UAMS introduces decentralized role access, real-time updates, and cloud deployment, thereby significantly improving efficiency, accessibility, and data integrity. The use of JWT authentication and Role-Based Access Control (RBAC) ensures secure handling of sensitive asset-related information, while the system's modular architecture allows for seamless future integrations with ERP platforms and emerging technologies such as AI/ML.

The deployment of UAMS not only streamlines asset lifecycle management but also enhances decision-making capabilities through structured reporting and analytics. The positive feedback from usability testing underscores the system's intuitive design and adaptability to different user roles.

Looking ahead, the system can be expanded to incorporate barcode/QR code scanning, predictive asset maintenance, and a dedicated mobile application, further enhancing operational efficiency and user convenience. This project serves as a practical example of how full-stack development methodologies can be applied to real-world institutional challenges, bridging the gap between academic research and industry-ready solutions.

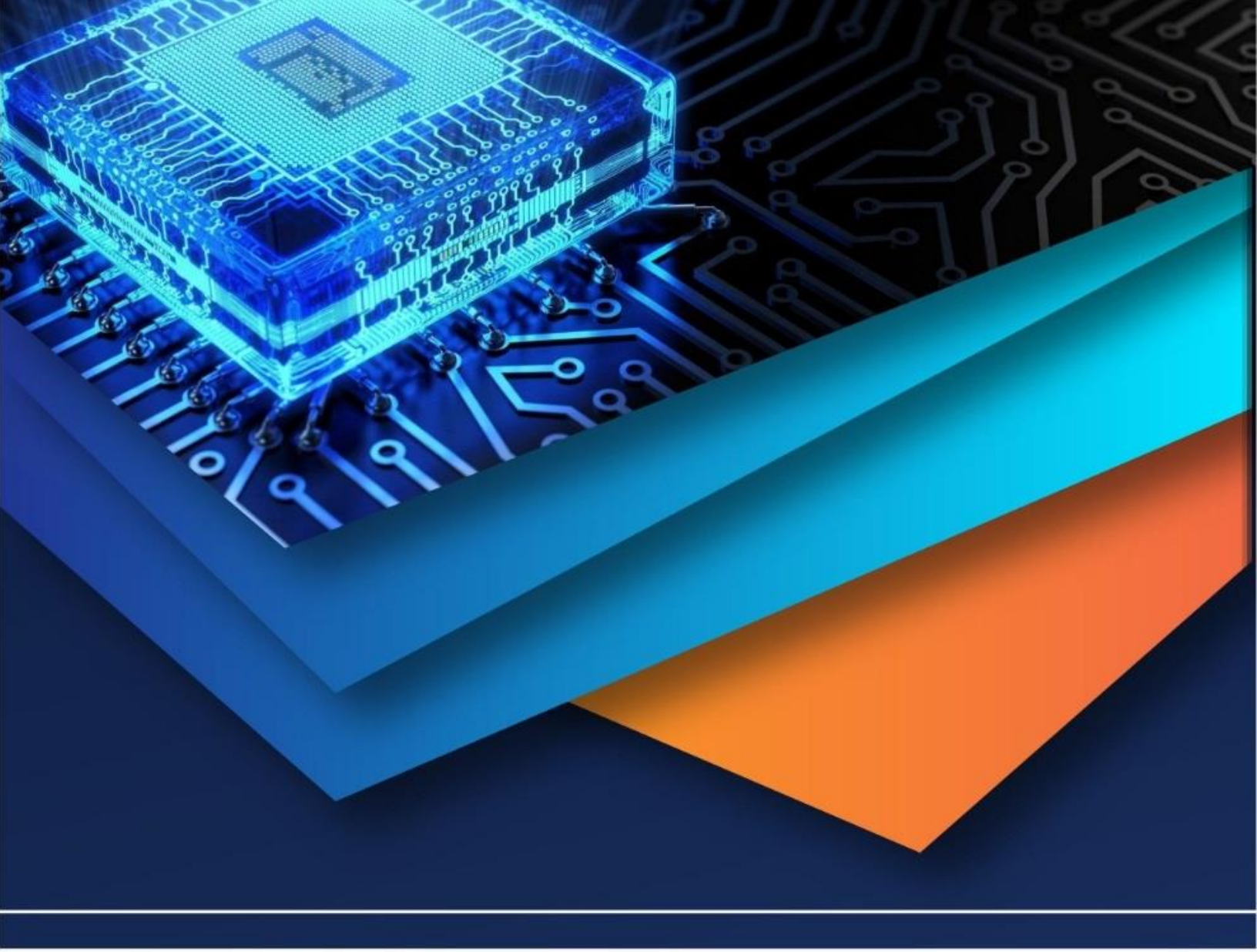
X. ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Mr. Raman Rajagopalan, Assistant Professor, School of Informatics, Department of MCA, Aurora Deemed to be University, for his valuable mentorship and constant encouragement throughout the project. I also wish to sincerely thank Ms. Kuchimanchi Jayasri, Assistant Professor, School of Engineering, Department of CSE, Aurora Deemed to be University, for her academic guidance, constructive suggestions, and continuous support during the development of this work. Finally, I acknowledge Aurora Deemed to be University for providing the resources and platform to successfully complete this research and implementation.

REFERENCES

[1] Kiran, D., & Sharma, P., "Asset Management System Using Web Technologies," International Journal of Computer Applications, vol. 184, no. 3, pp. 24–28, 2022.
[2] Mulyadi, R., & Nurprihatin, F., "Web-Based Asset Management Information System in Higher Education Institutions," International Journal of Information Systems and Technologies, vol. 6, no. 2, pp. 110–118, 2021.
[3] Chowdhury, S., & Das, A., "Review of Modern Asset Management Systems and Their Adoption in Academic Institutions," International Journal of Engineering Research & Technology (IJERT), vol. 12, no. 5, 2023.
[4] MongoDB Inc., "MongoDB Manual," [Online]. Available: https://www.mongodb.com/docs/
[5] P. Johnson, "The Shift Towards Web-Based Asset Management Systems," Computers in Industry, vol. 68, pp. 85–93, 2015.
[6] M. Lee, "Cloud-Based Asset Management: Trends and Benefits," IEEE Cloud Computing, vol. 7, no. 1, pp. 23–31, 2020.
[7] Express.js Foundation, "Express.js Documentation," [Online]. Available: https://expressjs.com/
[8] Meta Platforms, Inc., "React – A JavaScript Library for Building User Interfaces," [Online]. Available: https://react.dev/
[9] Node.js Foundation, "Node.js Documentation," [Online]. Available: https://nodejs.org/
[10] D. Patel, "Securing Web Applications Using JSON Web Tokens (JWT)," International Journal of Computer Applications, vol. 182, no. 44, pp. 15–21, 2021.
[11] S. Kumar, "Integrating Asset Management with ERP Systems for Operational Excellence," International Journal of Enterprise Information Systems, vol. 16, no. 3, pp. 34–49, 2020.

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



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)