iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

DESIGN AND IMPLEMENTATION OF A WEB-BASED FINANCE MANAGEMENT SYSTEM FOR INTEGRATED BUDGET, EXPENDITURE AND PROCUREMENT TRACKING

Jenisha V.
UG Student, Department of CSE (Cyber Security), St. Joseph's Institute of Technology
jenishaagnes@gmail.com
Dr. J. Jebathangam
Professor, Department of Computer Science, VISTAS
jthangam.scs@vistas.ac.in

ABSTRACT
Financial administration in large organisations involves interdependent activities such as budget allocation, expenditure recording, employee reimbursement and procurement monitoring. When these activities are maintained through disconnected spreadsheets and paper-based approvals, consolidating information and tracking the progress of requests can require substantial manual effort. This paper presents the design and implementation of a web-based finance management system that brings these activities into a unified application. The prototype adopts a three-tier architecture comprising a React.js presentation layer, a Node.js and Express.js application layer, and a MySQL relational database. Its functional modules include budget-head management, expenditure recording and approval, reimbursement processing, procurement status tracking, and a dashboard for financial summaries. Authentication is implemented using JSON Web Tokens (JWT), with role checks applied to administrative operations. The prototype was exercised in a local Windows development environment using demonstration records. Functional checks covered authentication, role-restricted actions, database consistency, status transitions and dashboard calculations. In the demonstration dataset, the dashboard represented a total allocation of ₹95 crore and approved expenditure of ₹12 crore; these figures are illustrative test data and not actual organisational financial disclosures. The implementation indicates that a unified web application can provide consolidated visibility and structured workflow support. Further evaluation with representative users, larger datasets, formal security testing and authorised deployment is required before operational adoption.

Keywords:
financial management system, budget monitoring, expenditure tracking, procurement workflow, role-based access control, React.js, Node.js, MySQL, dashboard analytics

1. INTRODUCTION
Financial management in manufacturing and public-sector organisations requires coordination between allocation of funds, recording of expenditure, reimbursement of eligible claims and procurement of goods or services. These activities generate related records but are often handled by different personnel and administrative processes. If the records are maintained in separate spreadsheets or circulated through paper forms, managers may need to reconcile information manually to understand the current financial position or identify requests awaiting action.
A browser-based application can provide a common interface for entering, reviewing and retrieving financial records. A relational database can maintain links between budget heads and expenditure entries, while a server-side application can apply validation and authorisation rules. A dashboard can summarise stored records through key performance indicators (KPIs), trend charts and category-wise views. Such a system does not, by itself, guarantee financial control; its usefulness depends on correct business rules, reliable data, access restrictions and institutional procedures.
This paper describes a prototype of a web-based finance management application intended for use in a large organisation. The work focuses on the software design and implementation of a portal for budget tracking, expenditure management, reimbursement processing and procurement tracking. The contribution is an integrated implementation that connects these functions through a common database and role-aware REST API, supported by dashboard visualisation.
The remainder of the paper is organised as follows. Section 2 discusses the technical background. Section 3 describes the proposed architecture and modules. Section 4 explains the implementation and evaluation approach.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Section 5 presents the observed demonstration results and limitations. Section 6 concludes the paper and identifies further work.

2. TECHNICAL BACKGROUND AND RELATED WORK

Web applications commonly separate the user interface, application logic and persistent data into distinct layers. This separation allows the presentation components to evolve independently from server-side validation and database operations. In the proposed prototype, React.js is used to construct reusable interface components, while Node.js and Express.js provide HTTP endpoints for application operations. MySQL stores structured entities and their relationships. These technologies are documented in their respective official developer resources [1]–[4].

REST-style interfaces expose resources through HTTP methods and exchange structured representations, commonly JSON. This approach allows a browser-based client to request records and submit changes without embedding database operations in the user interface. The prototype uses separate route groups for authentication, budget heads, expenditures, reimbursements, procurement and dashboard summaries.

Authentication and authorisation are distinct concerns. Authentication establishes the identity of a user, whereas authorisation determines which operations that user may perform. JWT is a compact claims representation format defined by the Internet Engineering Task Force [5]. A signed JWT can support integrity verification, but it should not be treated as encrypted or as a replacement for transport security and secure token handling. OWASP guidance highlights implementation considerations and common JWT risks [6]. Password storage should use an adaptive password-hashing scheme rather than plaintext storage; bcrypt is among the recognised approaches, although production configuration and migration choices should be reviewed against current security guidance [7].

Data visualisation can help users interpret aggregated financial records. The prototype uses Recharts to render monthly expenditure trends and category distributions in the React interface [8]. The charts are intended as decision-support views over database records; their accuracy depends on the correctness and completeness of the underlying transactions.

3. PROPOSED SYSTEM

3.1 Architecture

The application follows a three-tier architecture. The presentation tier is a React single-page application that provides navigation, forms, tables and dashboard charts. The application tier is implemented with Node.js and Express.js and exposes REST API routes. It validates requests, verifies authentication tokens, applies role restrictions and performs database operations. The data tier uses MySQL to persist user, budget, expenditure, reimbursement and procurement records.

The interaction sequence begins when a user accesses the application through a browser. After submitting login credentials, the server verifies the stored password hash and issues a JWT for an authenticated session. Subsequent API requests include the token in the HTTP Authorization header. Middleware verifies the token before protected operations are processed. The relevant route executes SQL queries and returns a JSON response, which the React client uses to update the interface.

3.2 Data Model

The prototype uses five principal relational tables. The users table stores identity, password-hash and role information. The budget_heads table records budget codes, descriptions, financial years and allocated amounts. The expenditures table stores expense entries, including a reference to a budget head, amount, category and approval status. The reimbursements table stores employee claim details, purpose, amount and processing status. The procurements table records requested items, quantities, unit prices, supplier information, priority and workflow status. The expenditure-to-budget-head relationship supports calculation of utilisation from approved expenditure records.

3.3 Functional Modules

Dashboard: The dashboard consolidates financial indicators, including total allocation, approved expenditure, remaining balance and counts of pending actions. It also presents a monthly expenditure area chart, a category-wise donut chart and budget utilisation bars. The summary values are calculated from database records rather than being maintained as independent dashboard entries.

Budget-head management: Authorised users can create and review budget heads for a financial year. The interface displays allocation, approved spending, remaining balance and a utilisation percentage. Search and filtering functions support locating a budget code or name.

Expenditure management: Officers can submit expenditure records associated with a budget head. Administrative users can approve or reject pending entries. The approval state is reflected in the expenditure table and is used in the dashboard and budget utilisation calculations.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Reimbursement management: Employee reimbursement claims are recorded with employee details, purpose and amount. The demonstrated lifecycle includes pending, approved, rejected and paid states. The administrative interface provides actions appropriate to the current state.
Procurement management: Purchase requests contain item, quantity, unit price, supplier, priority and required-by information. The interface tracks requests through draft, submitted, approved, ordered and received states. The source report refers to a six-stage pipeline in one section but enumerates five named states; this manuscript therefore describes only the five states explicitly listed in the implementation narrative. The exact workflow should be confirmed against the application before submission.

3.4 Access Control

The prototype distinguishes administrative and officer roles. The server verifies JWTs on protected routes, while administrative middleware restricts approval and status-change operations. The client interface also varies the available actions by role; however, client-side hiding of controls is not a security boundary. The server-side authorisation check is the decisive control. A production deployment should additionally verify object-level permissions, token expiry and revocation strategy, secure transport, input validation, logging and secret management.

4. IMPLEMENTATION AND EVALUATION METHOD

4.1 Development Environment

The application was implemented and demonstrated on a Windows workstation. The frontend uses React.js 18 and React Router. The backend uses Node.js and Express.js, with MySQL 8.0 as the relational database. Axios is used for HTTP communication, Recharts for visualisation, and bcryptjs and JWT-related functionality for authentication. The reported execution arrangement starts the backend on port 5000 and the frontend on port 3000 in a local environment.

4.2 API Design

The REST API is organised around functional resources. Authentication endpoints support login and retrieval of the current user. Budget-head endpoints provide listing and creation operations. Expenditure, reimbursement and procurement endpoints support record creation and status updates. A dashboard summary endpoint returns aggregated KPI and chart data. This organisation keeps the frontend independent of direct database access and provides a central location for validation and access checks.

4.3 Evaluation Procedure

The source project report describes demonstration-based functional checks rather than a controlled performance study. The reported checks included login and protected-route access, role restrictions on approval actions, creation and retrieval of records, status transitions, dashboard aggregation and comparison of dashboard totals with direct SQL calculations. This paper retains that scope and does not interpret the demonstration as evidence of production readiness or statistically established performance.
Table 1 summarises the checks described in the project report. The outcomes are reported qualitatively because the source material does not provide a complete test-case log, test environment specification, repeated-run measurements or a formal security assessment.

Test area	Procedure described	Reported outcome
Authentication	Attempt access with missing, expired and valid tokens	Missing/invalid access was rejected; valid authenticated access succeeded
Role enforcement	Attempt approval operations using officer and admin roles	Approval controls were restricted to the admin role in the demonstration
Budget and expenditure	Create budget heads and expenditure entries; compare totals	Displayed totals were reported to match SQL aggregation checks
Reimbursement	Move claims through pending, approved and paid states	The described lifecycle was demonstrated
Procurement	Advance requests through the listed status sequence	Status updates were reflected in the database and refreshed table
Dashboard	Compare KPI and chart values with demonstration records	Reported sample totals and visual summaries were consistent with test data

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

5. RESULTS AND DISCUSSION

The prototype integrated the five functional data areas into a single browser-based application. During the reported demonstration, users could access the dashboard after authentication and navigate to the budget, expenditure, reimbursement and procurement modules. The role-aware interface displayed administrative actions for the administrator role, while the server-side middleware was reported to block officer-role users from approval endpoints.

The dashboard was exercised with demonstration records comprising three budget heads, a total allocation of ₹95 crore and approved expenditure of ₹12 crore. The resulting balance was ₹83 crore and the reported utilisation was approximately 12.6%. These amounts are sample values used to demonstrate calculations; they must not be interpreted as actual the organisation budget or expenditure figures. The source report also describes five expenditure entries, three reimbursement requests and two procurement requests as part of the demonstration.

The reported dashboard included a monthly expenditure trend, category distribution and budget utilisation indicators. Budget utilisation bars changed visual state at configured thresholds, while the pending-approval area provided links to the corresponding modules. Such presentation can help users identify records requiring attention, provided that the underlying status and amount data are kept accurate.

The source report states that four dashboard API calls completed in approximately 200–400 milliseconds in the local environment. Since the report does not specify hardware configuration, dataset size, network conditions, measurement method or repeated trials, this value should be treated only as an observation from the original demonstration and not as a benchmark or general performance claim. A publishable performance evaluation would require a defined workload, repeated measurements, latency percentiles, concurrency levels and resource utilization.

The implementation demonstrates functional integration, but several aspects remain outside the evidence available in the project report. No formal usability study, penetration test, vulnerability assessment, database stress test, multi-user concurrency test or production deployment evaluation is documented. In addition, the source report's procurement workflow description contains an inconsistency in the number of stages. These limitations are important when interpreting the results.

6. SECURITY, DEPLOYMENT AND GOVERNANCE CONSIDERATIONS

Financial applications process records that may be sensitive even when the prototype uses demonstration data. Before deployment beyond a controlled local environment, the system should be reviewed for secure configuration and organisational compliance. JWT signing keys must be generated and stored securely, token lifetimes and validation rules should be explicit, and tokens should be transmitted only over protected connections. A signed JWT does not conceal its claims [5], [6]. Password hashes should use an appropriately configured adaptive algorithm and unique salts [7].

Authorisation should be enforced on every protected server operation, including checks that a user is permitted to access or modify the specific record requested. Input validation, parameterised database queries, error handling, rate limiting, secure session practices, audit logging, backup and recovery procedures should be assessed. Production deployment would also require approval from the relevant organisation, a security review, user acceptance testing and documented responsibility for maintaining financial data.

The project was demonstrated in a localhost environment. Consequently, the paper does not claim that the prototype has been deployed in a live organisational environment or used for actual financial transactions. The manuscript is presented as a generalised technical study, and all sample data are illustrative.

7. LIMITATIONS AND FUTURE WORK

The principal limitation is the scale and nature of the evaluation: it is a functional demonstration using sample records, not a longitudinal deployment study. The implementation report does not provide a reproducible benchmark dataset, formal usability scores or independent security-test results. The current manuscript also does not include source-code inspection or independent verification of all implementation claims.

Future work can extend the prototype with multi-year financial comparisons, exportable management reports, notification services, a timestamped audit-log module, department-wise expenditure analysis and vendor performance summaries. WebSocket-based updates could support near-real-time dashboard refreshes. Forecasting methods may be investigated for budget utilisation estimation, but such models would require historical, authorised and appropriately governed data. A production deployment should follow a staged process involving security testing, user acceptance, backup validation and monitored pilot operation.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

8. CONCLUSION

This paper presented the design and implementation of a web-based finance management prototype that integrates budget-head administration, expenditure approval, reimbursement processing, procurement tracking and dashboard analytics. The system combines a React.js frontend, a Node.js and Express.js REST API and a MySQL relational database. JWT-based authentication and role checks provide a foundation for separating officer and administrator operations. Demonstration tests described in the project report indicate that the principal workflows and dashboard calculations operated as expected with sample data. The evidence supports the feasibility of integrating these functions in a single web application, but does not establish production readiness, organisation-wide impact or performance under operational workloads. Formal security assessment, broader testing, user evaluation and institutional approval are necessary next steps.

REFERENCES

1) [1] React Team. “React Documentation.” https://react.dev/ (accessed November 24, 2026)
2) OpenJS Foundation. “Node.js Documentation.” https://nodejs.org/en/docs/ (accessed November 24, 2026).
3) OpenJS Foundation. “Express – Node.js Web Application Framework.” https://expressjs.com/ (accessed November 24, 2026).
4) Oracle Corporation. “MySQL 8.0 Reference Manual.” https://dev.mysql.com/doc/refman/8.0/en/ (accessed November 24, 2026).
5) M. Jones, J. Bradley, and N. Sakimura. “JSON Web Token (JWT),” RFC 7519, Internet Engineering Task Force, May 2015. https://www.rfc-editor.org/rfc/rfc7519
6) OWASP Foundation. “JSON Web Token Cheat Sheet.” OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_Cheat_Sheet.html (accessed November 24, 2026).
7) OWASP Foundation. “Password Storage Cheat Sheet.” OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accessed November 24, 2026).
8) Recharts Contributors. “Recharts Documentation.” https://recharts.org/en-US/ (accessed November 24, 2026).
9) M. Fowler. *Patterns of Enterprise Application Architecture*. Boston, MA: Addison-Wesley, 2002.

 © Author(s) | CC BY 4.0 International License.