A Secure Web-Based System for Personal Financial Recording, Budget Management and Expense Analysis

Ektha Ramesha[1], Abhinav R. Adiga[2], Adithya V[3], Yeswanth Raj[4], Chinmay Shivarama Reddy[5]
Dhruv Patel[6], Dr. Jayashree Inchal[7]
[1 2,3,4,5,6]*UG Students, Department. of Computer Science and Engineering (AIML), JAIN (Deemed-to-be University), Bengaluru, Karnataka, India*
[7]*Project Mentor Associate Professor and Faculty Guide, Department. of Computer Science and Engineering, JAIN (Deemed-to-be University), Bengaluru, Karnataka, India*

***Abstract*—Individuals who track personal finances manually often end up with scattered records, limited visibility of category-wise spending and no easy way to compare expenses with a planned limit, while a shared web application must additionally ensure that each account can reach only its own data. This paper presents a secure, multi-user, browser-based personal expense tracker built with Node.js, Express.js and MongoDB. The system supports account registration and login, income and expense recording, searchable and filterable transaction history with pagination, CSV export, monthly budgets and a monthly dashboard. Passwords are hashed with bcrypt, sessions use signed JSON Web Tokens in HttpOnly cookies, and every transaction and budget query is scoped to the authenticated user's identifier, so accounts stay logically separated within a shared database. Server-side validation rejects malformed input, and dashboard aggregation computes income, expenses, balance and category totals in integer paise to limit floating-point error. A walkthrough with a sample month of 18 transactions shows recorded income of ₹61,500.00, expenses of ₹31,357.00 and a balance of ₹30,143.00, with three expense categories accounting for about 81% of spending. A 17-case test plan covering authentication, ownership isolation, validation, pagination, export and budgeting is defined; automated test execution, performance benchmarking and a user study are left for future work. The result is a practical, security-conscious foundation for personal financial record keeping.**

***Index Terms*—budget management, expense tracker, JSON Web Token, MongoDB, multi-user web application, personal finance**

I. INTRODUCTION

Money management is a recurring activity for students, employees, families and independent professionals. Money arrives from several sources and is spent on food, transport, education, bills, health, shopping and other needs. Without a consistent recording method, it is hard to determine how much was spent in a month, which categories account for the largest share, or whether spending has exceeded a planned limit. Personal financial management apps have been reported among the fastest-growing categories of finance apps [1], [2], yet a functionality review of 45 top-rated budgeting apps found limited support for budgeting compared with tracking, and none of the apps reported an evaluation through user studies [2].

A. Background

Traditional record keeping relies on notebooks, calculator entries or spreadsheets. These are flexible, but searching and aggregation become time-consuming as the number of records grows. A web application centralises records, validates entries at the time of input and computes summaries on request, and a browser-based interface avoids installing a dedicated client.

B. Problem Statement

The problem addressed here is the lack of a single, convenient application for maintaining personal financial entries while keeping the records of different registered users logically separated. A useful solution must validate input, store and retrieve records efficiently, support corrections and present totals consistently. In particular, it must address scattered records, limited visibility of monthly income, expenditure and balance, weak categorisation, manual budget checking, multi-account privacy (one user must not be able to view, edit or delete another user's

records) and the need for a portable CSV copy of the data. We therefore design and implement a secure, browser-accessible, multi-user personal expense tracker that supports account management, income and expense recording, searchable transaction history, monthly budgeting, financial summaries and CSV export, while enforcing user ownership at the API and database-query level.

C. Objectives and Scope

The objectives are to (i) implement registration, login and logout with hashed passwords and signed-token route protection; (ii) associate every transaction and monthly budget with the authenticated user; (iii) provide create, view, update and delete operations with validation of type, category, amount, date, payment method and notes; (iv) support text search, type, category and date-range filters, pagination and CSV export; (v) allow one budget per user per month; (vi) calculate monthly income, expenses, balance and category totals; (vii) provide profile viewing, name updating and password changing; and (viii) offer a responsive browser interface.

The scope covers manually entered income and expense records, user accounts, transaction history, monthly budgets and monthly dashboard summaries. Categories and payment methods are predefined in the transaction route, and amounts are INR-oriented (the CSV amount column is labelled INR). Bank-account synchronisation, automatic transaction detection, receipt OCR, recurring payments, investment tracking and tax filing are out of scope. The system is a multi-account personal tracker, not a shared family ledger or group wallet.

D. Contributions and Organization

The main contributions are: (1) a layered Node.js/Express/MongoDB design in which the authenticated owner identifier scopes every transaction and budget operation; (2) a validated transaction module with search, filtering, pagination and spreadsheet-safe CSV export; (3) a monthly dashboard that aggregates income, expenses, balance, category totals and budget status using integer-paise arithmetic; and (4) a functional demonstration, a 17-case test plan and an explicit statement of limitations. Section II surveys related work, Section III presents the methodology, Section IV describes the implementation, Section V reports results and discussion, and Section VI concludes.

II. LITERATURE SURVEY

A. Personal Finance Practices and Digital Tools

Exploratory studies of how people track and manage personal finance [3], [4] indicate that digital tools are used only to a limited extent, and spreadsheets are commonly used for tracking transactions rather than for budgeting [2]. In a study of household budgeting practices, Snow and Vyas [5] found that banking apps were the more common digital tool; as summarised in [2], only about one third of their 15 participants had used budgeting apps, which they tended to abandon after a short time because of the effort of entering expenses and the lack of adequate control over spending.

B. Budgeting Apps and Mental Accounting

Thaler's theory of mental accounting [6] describes how people partition money into labelled categories to organise, track and monitor spending. Alenazi and Sas [2] reviewed 45 top-rated budgeting apps selected from 1,335 candidates in the Google Play and Apple app stores. All 45 apps supported entering expense transactions manually; 33 also monitored expenses against an allocated budget, 26 supported separate budgets for expense categories, only seven could link to online banking (as a premium feature), and none reported an evaluation through user studies [2].

C. Implementation Approaches and Security Practice

Web-based expense trackers are frequently built on the MERN stack (MongoDB, Express.js, React and Node.js). For example, a 2023 thesis [7] describes the design and implementation of an expense-tracking web application created to master full-stack development and to monitor the author's own spending. For authentication, JSON Web Tokens are standardised in RFC 7519 [8]; bcrypt [9] is an adaptive password-hashing scheme whose cost parameter can be increased as hardware improves; and the OWASP password-storage guidance [10] discusses the choice of adaptive hashing algorithms and work factors.

D. Research Gap

Table I relates the gaps reported in the literature to the response taken in this work.

TABLE I GAPS IN THE LITERATURE AND THE RESPONSE OF THIS WORK

Gap reported in the literature	Response in this work
Tracking is widely supported, but support for budgeting is weaker [2].	A monthly budget per user, with remaining amount, over-budget amount, percentage used and a status on the dashboard.
Effort of manual entry leads users to abandon budgeting apps [5].	Validated entry forms, searchable and filterable history and CSV export, intended to make correction and review easier (effect on effort not yet measured).
Few apps report evaluation through user studies [2].	A defined 17-case test plan and metrics (Section V-B); a user study is future work.
A shared web application must keep each user's data separate.	Owner-scoped queries, hashed passwords and signed token cookies (Section IV).

III. METHODOLOGY

A. Functional Requirements

Table II maps each functional requirement to the implemented workflow.

TABLE II FUNCTIONAL REQUIREMENTS

ID	Requirement	Description
FR-01	Account registration	Create a user with name, email and password; reject duplicate email.
FR-02	Login and logout	Verify credentials, issue token cookie and clear it on logout.
FR-03	Profile management	View profile, update display name and change password.
FR-04	Add transaction	Create income or expense records with validated fields.
FR-05	Transaction history	Return the current user's records in descending date order.
FR-06	Search and filter	Filter by title, type, category and date interval.
FR-07	Edit/delete	Modify or remove only a record owned by the logged-in user.
FR-08	Pagination	Return history in pages of ten records with page metadata.
FR-09	CSV export	Export all records matching the supported filters.
FR-10	Budget management	Create or update one monthly budget per user-month.
FR-11	Dashboard	Calculate month totals, category totals and budget comparison.

B. Non-Functional Requirements

Security: passwords are hashed, protected APIs require verified authentication, and owner-scoped queries are used for personal records. Reliability: invalid input is rejected with a client error rather than stored as an inconsistent record. Usability: common tasks are reachable through browser navigation and forms. Maintainability: route handlers, middleware, models and database configuration are separated into modules. Portability: the interface is served by the Node.js application and opens in a compatible browser. Data consistency: Mongoose schema validation and database indexes support consistent records and uniqueness constraints. Performance: pagination limits the records returned per request and indexes support common owner/date and unique lookups; actual performance requires measurement.

C. Hardware and Software Requirements

The application runs as a Node.js web server [11] with a MongoDB database and is accessed through a modern browser. Table III lists the software environment needed to run the project locally.

TABLE III SOFTWARE ENVIRONMENT

Category	Requirement
Runtime	Node.js-compatible runtime [11]
Server framework	Express.js [12]
Database	MongoDB instance accessible through MONGO_URI [13]
ODM	Mongoose [14]
Password hashing	bcryptjs [15]
Browser	Modern browser with JavaScript and cookie support
Configuration	Environment variables for database URI, JWT secret, port and environment
Development	npm and optional nodemon for automatic server restart

D. System Architecture

The application uses a layered client–server architecture (Fig. 1). The front end presents forms,

navigation, transaction tables and dashboard information. The Express server receives HTTP requests, applies route-level logic and authentication middleware, and interacts with MongoDB through Mongoose models. MongoDB stores the persistent user, transaction and budget documents.

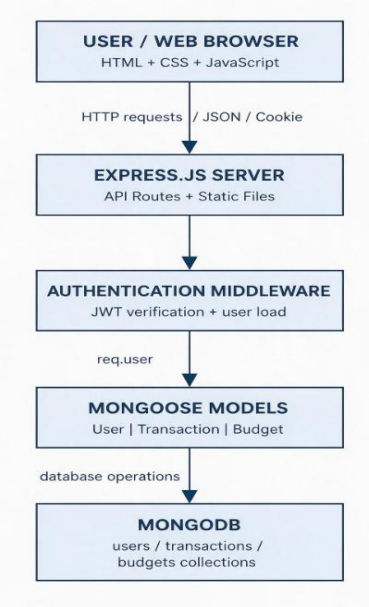


Fig. 1. System architecture of the expense tracker.

E. Data Model and Indexing

Fig. 2 shows the relationships: one user owns many transactions and many budgets. The User collection stores the name, a unique normalised email address and password Hash, which is excluded from ordinary queries and selected explicitly only when a password must be verified; timestamps add created At and updated At. A Transaction document holds user Id (a reference to User), type, title, amount, category, date, payment Method and notes. A Budget document holds use rid, month, year and amount, with month and year stored as numeric values.

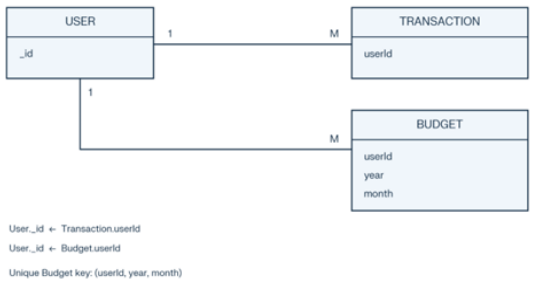


Fig. 2. Entity relationships between User, Transaction and Budget.

Three indexes are defined: a unique index on user email; a compound index on transaction owner and date, which supports chronological retrieval of a user's history; and a unique compound index on budget userId, month and year, which prevents duplicate monthly budgets for the same user. Indexes improve integrity and can support query execution, but their actual effect depends on collection size and query plans.

F. Request Processing Sequence

A request to a protected route is processed as follows.

1) The user acts in the browser, for example saving a transaction or opening the dashboard.
2) Client-side JavaScript sends a request to the corresponding API endpoint.
3) Authentication middleware reads the token cookie and verifies its signature and expiry.
4) The middleware loads the account and attaches it to the request as req.user.
5) The route validates the request values and builds a database query scoped to req.user._id.
6) Mongoose executes the database operation or aggregation.
7) Express returns a JSON response or a CSV file with an appropriate HTTP status.
8) The browser updates the relevant view or displays a validation or error message.

G. Transaction Creation Algorithm

Algorithm 1. Create transaction
1. Receive transaction fields from the client.
2. Verify the authenticated user.
3. Validate the type and the matching category.
4. Validate the title, the amount and its decimal precision.
5. Validate the date and the payment method.
6. Validate the optional notes.
7. Set owner ← authenticated user ID.
8. Insert the transaction document.
9. Return a success response.

H. Financial Computations

The dashboard summarises the entries saved in the application; it does not verify bank balances or cash holdings, which is why the quantities are called "recorded". With recorded income I, recorded

expenses E and monthly budget M, the recorded balance B and the percentage of budget used U are

$$B = I - E \quad (1)$$

$$U = (E \div M) \times 100 \quad (2)$$

The remaining amount R and the over-budget amount O are reported separately so that the dashboard never shows a negative remaining amount:

$$R = \max(M - E, 0), \quad O = \max(E - M, 0) \quad (3)$$

The status is within_budget when E < M, at_budget when E = M and over_budget when E > M. When no budget exists, the budget comparison is null rather than a fabricated value.

I. Monetary Precision

Transaction input is restricted to two decimal places. For n amounts a_i, the dashboard aggregation multiplies each amount by 100, rounds it to integer paise, sums the paise values and converts the result back to rupees:

$$T = (1/100) \sum \text{round}(100 \cdot a_i) \quad (4)$$

This reduces floating-point accumulation effects. The model nevertheless stores the amount field as a numeric value rather than as integer paise or Decimal128.

IV. IMPLEMENTATION

A. Server Initialization

The server entry point loads environment variables with dotenv, creates the Express application, enables JSON and URL-encoded body parsing and configures cookie parsing. It mounts the authentication, profile, transaction, dashboard and budget routes under their API prefixes and serves the public directory as static content. At startup it connects to MongoDB, initialises the User, Transaction and Budget models and their indexes, and only then listens on the configured port (3000 when PORT is not supplied). A startup failure is logged and the process exits.

B. Authentication and Middleware

Registration validates name, email, password and confirmation. The email is normalised to lowercase and duplicate accounts are rejected. Passwords must contain at least eight characters and are limited to 72 UTF-8 bytes for compatibility with bcrypt [9]. The password is hashed with bcryptjs using a work factor of 12; the same factor is used when a password is changed. The login route retrieves the account by normalised email, selects the password hash and compares the submitted password. On success the server signs an HS256 JSON Web Token [8] with a one-hour expiry and sets it in an HttpOnly cookie with SameSite=Strict (and Secure in production). Logout clears the cookie.

The authentication middleware checks that JWT_SECRET is configured, reads the token from the cookie, verifies its signature with HS256 and validates the user identifier, then loads the user from the database. If the token is absent, invalid or expired, or refers to a deleted account, the request is rejected; otherwise, the user object is attached as req.user for downstream routes.

C. Profile Module

All profile endpoints require authentication. Reading the profile returns the account identifier, name and email. A name update trims the value and enforces a length of two to 100 characters. A password change requires the current password, a new password and its confirmation; the new password must satisfy the length and byte limits, match its confirmation and differ from the existing password, and is hashed before it replaces the stored value.

D. Transaction Module

The transaction module is the central record-management component. Creation validates all fields (Table IV) and sets userId from the authenticated account rather than trusting a client-supplied owner. Single-record retrieval, update and deletion include both the requested transaction identifier and the authenticated user identifier in the database query, so a record belonging to another account is never returned as an owned transaction.

TABLE IV TRANSACTION INPUT VALIDATION

Field	Illustrative value	Rule
Type	expense	Allowed values: income or expense
Title	Monthly bus pass	Maximum 150 characters
Amount	850.00	Positive; at most two decimal places
Category	Travel	Must match the selected type
Date	05-11-2026	Strict DD-MM-YYYY input

Field	Illustrative value	Rule
Payment method	UPI	Cash, UPI, Card or Bank Transfer
Notes	November commute	Optional; maximum 1,000 characters

E. Search, Filter and Pagination

The history endpoint supports title search, type, category, start date and end date. Search input is escaped before it is inserted into a case-insensitive regular expression, reducing the chance that user input is interpreted as regular-expression syntax. Dates must use DD-MM-YYYY and are validated by parsing and round-tripping the date; if both are supplied, the start date cannot be later than the end date. The end date is implemented as an exclusive upper boundary at the following UTC day so that the selected end date is included.

History uses a fixed page size of ten. The page parameter must be a positive integer within a safe range. The query sorts by date descending, then by creation time and identifier descending, giving a deterministic order for records with the same date. The response includes page, limit, totalRecords, totalPages, hasPreviousPage and hasNextPage.

F. CSV Export

CSV export reuses the search and filter logic without pagination, selects only the fields needed and sorts in descending order. The columns are Date, Type, Title, Amount (INR), Category, Payment method and Notes. Each cell is quoted and internal quotation marks are doubled. Values that could be interpreted as spreadsheet formulas, including values beginning with formula markers or control characters, are prefixed with an apostrophe. The response carries a UTF-8 byte-order mark, a download filename, no-store cache control and nosniff content-type protection.

G. Budget Module

A user can retrieve or save a budget for a selected month, supplied as a year–month value (YYYY-MM) and converted to month and year. The amount must be positive with at most two decimal places. The operation uses the authenticated user's identifier and an upsert pattern: it creates the monthly budget when none exists and updates it otherwise. The unique compound index is a database-level safeguard against duplicate budgets for the same user-month.

H. Dashboard Module

The dashboard accepts a month (YYYY-MM) and determines its UTC start and end boundaries. It aggregates only the authenticated user's transactions in two branches: one groups transactions by type and calculates totals and counts; the other filters expense records, groups them by category, totals amounts and counts, and sorts categories by total. The route also retrieves the matching monthly budget and calculates recorded income, expenses, balance and transaction count. When a budget exists, it adds the remaining amount, over-budget amount, percentage used and status defined in Section III-H.

I. API Summary and Error Handling

Table V lists the endpoints. JSON responses follow a general pattern of a success indicator, a human-readable message and, where applicable, a data object. Validation failures generally return HTTP 400, unauthenticated requests are rejected by the authentication layer, missing owned transactions return HTTP 404, and unexpected server or database failures return HTTP 500 with a generic message. The server does not return stack traces to the browser and logs error names for diagnosis.

TABLE V API ENDPOINTS

Method and endpoint	Purpose	Access
POST /api/auth/register	Register a new user	Public
POST /api/auth/login	Verify credentials, set cookie	Public
POST /api/auth/logout	Clear authentication cookie	Public
GET /api/profile	Read current profile	Auth.
PATCH /api/profile	Update profile name	Auth.
PATCH /api/profile/password	Change password	Auth.
POST /api/transactions	Create transaction	Auth.
GET /api/transactions	Search/list paginated transactions	Auth.

Method and endpoint	Purpose	Access
GET /api/transactions/export	Export matching transactions to CSV	Auth.
GET /api/transactions/:id	Read one owned transaction	Auth.
PATCH /api/transactions/:id	Update one owned transaction	Auth.
DELETE /api/transactions/:id	Delete one owned transaction	Auth.
GET /api/budgets?month=YYYY-MM	Read monthly budget	Auth.
PUT /api/budgets	Create/update monthly budget	Auth.
GET /api/dashboard?month=YYYY-MM	Read monthly dashboard	Auth.

J. User Interface

The public interface is a single HTML page with embedded styling and client-side JavaScript, served from the public directory. It uses a light background, cards, a hero area, form controls and navigation buttons, with sections for account access, profile, transaction entry and history, budgets and reports. CSS grid and flex layouts organise the page: the transaction area uses two columns on wide screens and one column on narrow screens, and summary cards use four, two or one column on wide, medium and small screens. The history table sits in an overflow container so that it scrolls horizontally when required, income and expense rows use different text classes, and filter controls are arranged in a responsive grid. Interactive controls have focus-visible outlines. The expense-category view uses horizontal bars whose widths show relative category totals; it is a visual summary of the server aggregation, not a forecast or statistical model.

K. Security Controls and Limitations

Three controls protect the multi-user setting. First, only a bcrypt hash is stored, which reduces the impact of a database exposure compared with plain-text storage, although users should still choose strong, unique passwords. Second, the token is signed with a secret held in environment configuration; that secret must be sufficiently random, must never be committed to a public repository, and production traffic should use HTTPS because the Secure cookie flag is enabled there. Third, record ownership is enforced by applying the authenticated user ID to every query: creation assigns the owner on the server, single-record reads, updates and deletes combine the record ID with the current user ID, history and export filters start from the user ID, and budget access likewise includes the owner. A client therefore cannot select a different owner by changing a query parameter.

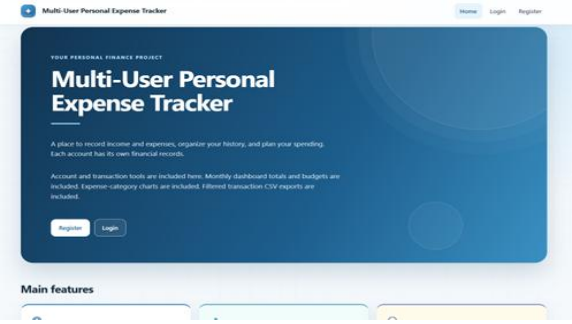
The implementation does not include multi-factor authentication, password reset, account lockout, dedicated rate limiting, centralised security monitoring, a formal threat model or an independent penetration test. Cookie-based authentication should be reviewed for CSRF exposure in the final deployment context. These are recommendations for future work, not features that have been implemented.

V. RESULTS AND DISCUSSION

A. Functional Demonstration

The implementation was exercised through its browser interface using a sample month (November 2026). Fig. 3 shows the home page and primary navigation. Fig. 4 shows the login and registration forms; the registration form states the password rule (at least eight characters and no more than 72 UTF-8 bytes). Fig. 5 shows the monthly dashboard with recorded income, expenses, balance and transaction count, and Fig. 6 shows the monthly budget screen with a saved budget of ₹30,000.00 for 2026-11. Fig. 7 shows transaction entry and the filtered history with search, type, category and date filters, CSV export and edit and delete actions. Fig. 8 shows the monthly report with the expense-category analysis.

Fig. 3. Home page and primary navigation.



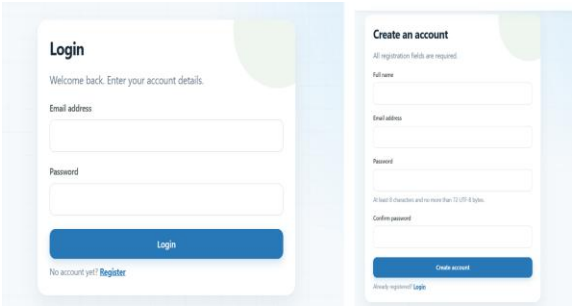


Fig. 4. (a) Login screen and (b) registration screen.

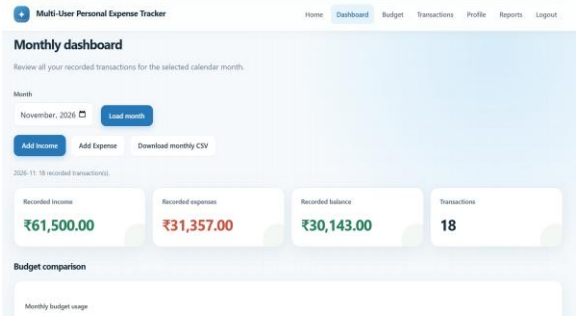


Fig. 5. Monthly dashboard summary.

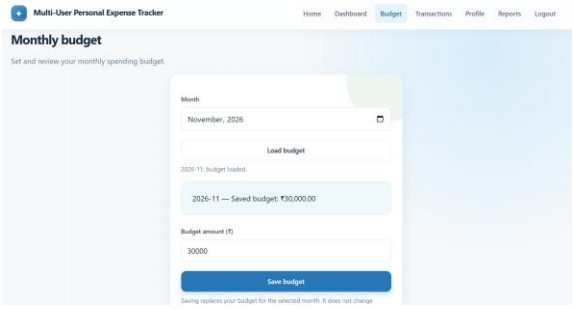


Fig. 6. Monthly budget management.

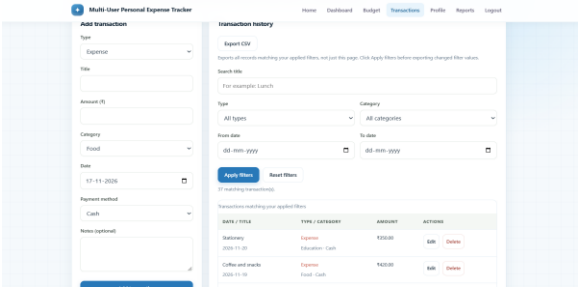


Fig. 7. Transaction entry and filtered history.

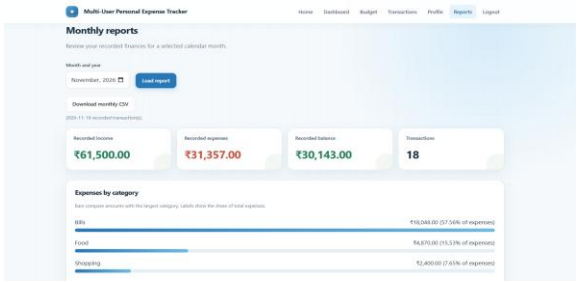


Fig. 8. Monthly report and category analysis.

Table VI summarises the values displayed for the sample month. Applying (1) to the displayed income and expenses gives ₹30,143.00, which matches the recorded balance. Applying (2) and (3) to the saved budget of ₹30,000.00 gives 104.52% of the budget used and an over-budget amount of ₹1,357.00, i.e. the over_budget status. Three categories (Bills, Food and Shopping) together account for 80.74% of the month's expenses.

TABLE VI SAMPLE-MONTH VALUES (NOVEMBER 2026)

Quantity	Value
Recorded income	₹61,500.00
Recorded expenses	₹31,357.00
Recorded balance	₹30,143.00
Recorded transactions	18
Saved monthly budget	₹30,000.00
Budget used, Eq. (2)*	104.52%
Over-budget amount, Eq. (3)*	₹1,357.00
Bills	₹18,048.00 (57.56%)
Food	₹4,870.00 (15.53%)
Shopping	₹2,400.00 (7.65%)

*Computed from the displayed values using the stated equations; not read from the interface.

B. Evaluation Plan

Table VII lists the planned test cases. The expected outcomes describe what should be verified; they are not results already achieved. The planned metrics are functional correctness (proportion of planned cases that pass), authorization isolation (rejection of cross-user access attempts), response time (median and 95th-percentile latency of transaction-list and dashboard requests under defined data volumes), data integrity (no duplicate monthly budgets or invalid transaction values), usability (task completion rate and reported ease of use) and export correctness (CSV row count and field correctness compared with the filtered database result).

TABLE VII PLANNED TEST CASES

ID	Test scenario	Expected outcome
T01	Register with valid details	Account created; password stored as hash.
T02	Register duplicate email	Request rejected; no duplicate account.

ID	Test scenario	Expected outcome
T03	Login with valid password	Authentication cookie issued.
T04	Login with incorrect password	Authentication rejected.
T05	Access protected endpoint without cookie	Request rejected.
T06	Create valid income/expense	Record created under logged-in user.
T07	Submit negative or malformed amount	Validation error; no record created.
T08	Use category incompatible with type	Validation error.
T09	Request another user's transaction ID	No foreign record disclosed or changed.
T10	Filter history by dates/category	Only matching owned records returned.
T11	Navigate transaction pages	Ten records per page; correct metadata.
T12	Export filtered transactions	CSV has all matching records, not only the current page.
T13	Save budget for same month twice	Existing budget updated, not duplicated.
T14	Dashboard with no budget	Budget comparison returned as null.
T15	Dashboard at/below/above budget	Correct status and amounts returned.
T16	Change password with wrong current password	Change rejected.
T17	Logout, then retry protected route	Cookie cleared; request no longer authenticated.

C. Discussion

The strongest design feature is the consistent use of the authenticated account identifier in data operations. This is central to a multi-user personal tracker because the API never relies on a user identifier supplied by the browser. The unique compound budget index enforces the one-budget-per-user-per-month rule at the database level, and aggregating in integer paise gives deterministic totals for two-decimal inputs. Relative to the gaps in Section II, the system adds budget monitoring on top of tracking, but it supports a single monthly budget per user rather than separate budgets per category [2], and it has not yet been evaluated with users. These observations rest on the implemented design and the demonstration in Section V-A; quantifying performance, security assurance and user satisfaction requires the repeatable tests, defined datasets and recorded measurements described in Section V-B.

D. Limitations

Transactions are entered manually, and bank synchronisation is not provided. Categories and payment methods are predefined in the route logic. The system targets personal records; group expense splitting and shared wallets are not supported. The dashboard is monthly and descriptive, with no forecasting. Production deployment, uptime, backups and monitoring have not been established, and no completed automated test suite or performance benchmark exists yet. Amounts are stored as numeric values, whereas a production financial system may prefer integer paise or a decimal type, and the use of UTC month boundaries should be reconsidered if users in several time zones are supported.

VI. CONCLUSION AND FUTURE WORK

This paper presented a web-based multi-user personal expense tracker for recording income and expenses, managing monthly budgets and reviewing financial activity. Its architecture combines a browser interface, an Express.js API, authentication middleware, Mongoose models and MongoDB persistence. Password hashing and signed JSON Web Token cookies protect account access, while user-scoped database queries separate each account's transactions and budgets. Transaction history supports search, filters, pagination and CSV export, and the dashboard summarises monthly income, expenses, balance, transaction count and expense categories and compares spending with a saved monthly budget. The implementation provides a practical foundation for personal financial record keeping.

Future work includes automated unit and integration tests for every route and validation boundary, multi-user authorization tests, load testing with larger datasets and usability testing with representative users; rate limiting, password reset, email verification and optional multi-factor authentication; an audit history of transaction changes; recurring-transaction templates and reminders; user-defined categories, payment methods and per-category budgets; richer

daily, weekly and monthly spending charts; multi-currency support with recorded exchange-rate metadata; and optional bank-data integration through authorised financial data providers.

ACKNOWLEDGMENT

The authors express their sincere gratitude to their project guide, Dr. Jayashree, Department of Computer Science and Engineering, JAIN (Deemed-to-be University), Bengaluru, for her guidance, encouragement, and support throughout the design, implementation, and review of this work. The authors also thank the department and institution for providing computing resources and academic support.

CONFLICT OF INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

FUNDING

This research received no external grant or funding from any public, commercial, or not-for-profit funding agency.

REFERENCES

[1] P. Bitrián, I. Buil, and S. Catalán, "Making finance fun: The gamification of personal financial management apps," Int. J. Bank Marketing, vol. 39, no. 7, pp. 1310–1332, 2021, doi: 10.1108/IJBM-02-2021-0074.

[2] M. Alenazi and C. Sas, "Evaluating budgeting apps: Limited support for budgeting compared to tracking," in Proc. BCS Human-Computer Interaction Conf. (BCS HCI), BCS Learning and Development Ltd., 2023.

[3] J. J. Kaye et al., "Money talks: Tracking personal finances," in Proc. CHI Conf. Human Factors Comput. Syst., 2014, pp. 521–530, doi: 10.1145/2556288.2556975.

[4] M. Lewis and M. Perry, "Follow the money: Managing personal finance digitally," in Proc. CHI Conf. Human Factors Comput. Syst., 2019, pp. 1–14, doi: 10.1145/3290605.3300620.

[5] S. Snow and D. Vyas, "Fixing the alignment: An exploration of budgeting practices in the home," in Proc. CHI Conf. Human Factors Comput. Syst. (Extended Abstracts), 2015, pp. 2271–2276, doi: 10.1145/2702613.2732808.

[6] R. H. Thaler, "Mental accounting matters," J. Behav. Decis. Making, vol. 12, no. 3, pp. 183–206, 1999.

[7] P. Tran, "Expence tracker application using MERN stack," thesis, 2023. [Online]. Available: https://www.theseus.fi/handle/10024/802483

[8] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7519

[9] N. Provos and D. Mazières, "A future-adaptable password scheme," in Proc. USENIX Annu. Tech. Conf., FREENIX Track, Monterey, CA, USA, Jun. 1999, pp. 81–91.

[10] OWASP, "Password storage cheat sheet," OWASP Cheat Sheet Series. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accessed Nov. 21, 2026).

[11] OpenJS Foundation, "Node.js documentation." [Online]. Available: https://nodejs.org/ (accessed Nov. 21, 2026).

[12] Express.js, "Express documentation." [Online]. Available: https://expressjs.com/ (accessed Nov. 21, 2026).

[13] MongoDB, Inc., "MongoDB manual." [Online]. Available: https://www.mongodb.com/docs/ (accessed Nov. 21, 2026).

[14] Mongoose, "Mongoose documentation." [Online]. Available: https://mongoosejs.com/docs/ (accessed Nov. 21, 2026).

[15] dcodeIO, "bcrypt.js: JavaScript bcrypt implementation." [Online]. Available: https://github.com/dcodeIO/bcrypt.js (accessed Nov. 21, 2026).