Personal Finance Tracker

SANJU JOSE J[1], SARATHIRAJA M[2], SACHIN VB[3]
[1,2,3] *Department of Computer Science and Engineering, PSNA College of Engineering and Technology Dindigul, TamilNadu, India*

Abstract: **This paper presents Personal Finance Tracker, a web-based platform designed to help individuals track income, expenses, savings, and investments effectively. Users can add daily transactions manually or via document uploads (like bank statements), categorize spending, set budgets, and view detailed financial analytics through interactive dashboards. The platform ensures secure data management, easy visualization, and promotes financial discipline. It helps users make informed financial decisions, save more, and manage money wisely through an intuitive and user-friendly interface.**

Keywords: **Personal Finance, Expense Tracker, Budgeting, Financial Analytics, Money Management.**

INTRODUCTION

Managing personal finances is crucial for achieving financial stability and goals. However, many individuals lack a structured way to track income and expenses, often leading to overspending or missed savings opportunities.
Personal Finance Tracker is a web-based mini project that addresses this by offering a centralized system where users can log transactions, categorize them, monitor budgets, and generate insightful reports.
It simplifies money management, encourages better financial habits, and provides transparency into spending behavior, empowering users to plan for their future with confidence.

LITERATURE REVIEW

With the increasing focus on data-informed education, many systems have been created to assess and improve student learning outcomes. However, traditional assessments often provide only summative scores, which prevents a deep examination of student outcomes. A particularly useful approach in this regard is the unit-wise (i.e. Course Outcome-wise) analysis of results. Previous research has shown that examining unit-wise performance is a means by which faculty can identify units that students struggle with, and differentiate learning outcomes to enable targeted teaching approaches leading to improved learning results.
Several existing financial applications (like Mint, YNAB, Pocket Guard) offer budget tracking and spending insights. However, many are either too complex for beginner users, lack customizability, or impose subscription fees.
Open-source and lightweight personal finance tools are relatively rare and often limited in features like multi-source transaction import, custom categorization, or predictive analytics.
The proposed system focuses on simplicity, user control, and data visualization, aiming to bridge the gap between manual budgeting and sophisticated financial apps. It emphasizes secure data handling and clear financial insights for individual users.

SYSTEM ARCHITECTURE

A. User Portal

- Add Income/Expense entries manually
- Upload bank statements (optional automated parsing)
- Set monthly budgets and financial goals
- Generate personalized reports

B. Admin Interface

- Manage user accounts
- Monitor platform usage analytics
- Perform CRUD operations on categories, budget templates
- Ensure data privacy and system maintenance
- Generate reports

TRANSACTION PARSING MECHANISM

Input Format and Parsing
The system accepts .csv or .docx bank statements.
We use the Papa Parse library (for .csv) and Mammoth (for .docx) to extract transactional data safely.
Identifying Transaction Tables

The uploaded files are scanned to detect transaction tables with headers like Date, Description, Amount, Type (Credit/Debit).

Extracting and Mapping Transactions

Each transaction is parsed into the database and mapped to categories like Food, Rent, Travel, Salary, Investments, etc.

The user can edit auto-categorized transactions to ensure accuracy.

Validation and Error Handling

The system checks for missing fields, date mismatches, or corrupted uploads and notifies the user to correct errors.

Scalability

Built with performance in mind, the system can process large transaction files without lag, ensuring scalability for frequent users.

User Flow Diagram

To maintain data quality and integrity, the system performs validation checks during parsing. If mandatory columns (e.g., CO, Marks) are missing or if the table structure is inconsistent, the system halts processing and notifies the user, thereby preventing the entry of invalid data into the database.

- Flexibility in Table Design

The platform supports flexible table designs. Column order is not fixed, and additional attributes—such as Bloom’s Taxonomy Level (BL) or Program Indicators (PI)—can be included as needed. The dynamic header recognition logic ensures accurate extraction of each question’s metadata, regardless of layout variations.

- Support for Multiple Tables

For documents containing multiple tables (e.g., different sections or types of assessments), the system iterates through each table independently, evaluating them for structural validity. Only tables containing recognized headers are parsed, while others are ignored to avoid false data capture.

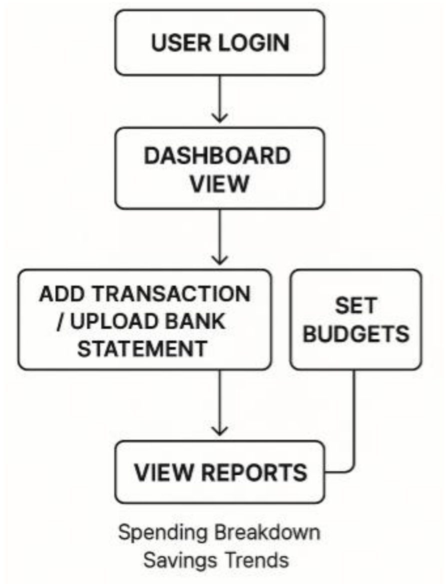
- Scalability and Performance

The combined use of Mammoth and Cheerio ensures efficient, high-performance parsing with minimal memory overhead. This makes the system well-suited for institutional use, capable of handling large documents or batch uploads without performance degradation.

- CO-to-Unit Association Logic

During course configuration, each Course Outcome is mapped to one or more syllabus units. When a question is linked to a CO, the system automatically determines the relevant unit(s) using these predefined mappings. This relationship enables downstream modules—such as report generation—to organize and display performance data unit-wise.

I. USER FLOW DIAGRAM



USES CASES

Users add manual transactions —
Quickly log income and expenses.

Users upload bank statements —
Bulk import transactions automatically.

Budget monitoring —
Set monthly/annual budgets and get alerts.

Financial reports —
Visual graphs showing spending patterns, savings rate, and category-wise analysis.

Admin monitoring —
Manage users and maintain system integrity.

IMPLEMENTATION STRATEGY

Phase 1: Requirements Analysis and Planning
Objective: Define the scope and key features of the Personal Finance Tracker.

Activities: Identify user needs and desired features such as expense tracking, budgeting, savings goals, reports, etc.
Define functional and non-functional requirements (e.g., data security, scalability, user-friendliness). Set up project milestones and deadlines. Choose the tech stack (frontend, backend, database).

Phase 2: System Architecture and Design
Objective: Plan the system's structure, user interface, and data flow.
Activities: Design the architecture (client-server model, database schema, etc.). Create wireframes and mockups for user interface (UI) design Define database models and relationships (tables for transactions, categories, users, budgets).Choose tools for visualization (charts, graphs) to represent financial data.

Phase 3: Frontend Development
Objective: Build the user interface for interacting with the system.
Activities: Develop the UI based on the wireframes, including pages for adding transactions, viewing reports, managing budgets, etc. Ensure responsive design (mobile and desktop).
Integrate financial data visualization tools (e.g., Chart.js, D3.js).Implement client-side validation (e.g., ensuring users input valid amounts for transactions).

Phase 4: Backend Development
Objective: Implement the logic and data processing for the application.
Activities: Set up the server-side code to handle requests from the frontend (Node.js with Express, Python with Flask, etc.).Develop RESTful API endpoints for CRUD operations (Create, Read, Update, Delete) on transactions, budgets, and user data.Implement authentication and authorization for user security (JWT or OAuth).Set up business logic for generating financial reports, calculating budgets, and handling notifications.

Phase 5: Database Integration
Objective: Set up and connect the database to store user data.
Activities: Design and create database schema (tables for users, transactions, categories, budgets). Implement relationships between entities (e.g., a transaction belongs to a user and category). Optimize database queries for performance.

Phase 6: Security Implementation
Objective: Ensure the security of the user's financial data.
Activities: Implement HTTPS for secure communication between frontend and backend.Encrypt sensitive data like user passwords using bcrypt or similar algorithms.Set up input validation to prevent SQL injection and cross-site scripting (XSS).Implement authorization checks to ensure users can only access their data.

Phase 7: Testing
Objective: Ensure the application is bug-free and functions as expected.
Activities: Perform unit testing for individual components (e.g., API endpoints, UI elements). Conduct integration testing to check if different parts of the system work together (e.g., frontend interacting with backend).Perform usability testing with real users to ensure the app is intuitive and easy to use. Test for edge cases (e.g., large amounts of data, empty fields) to ensure robustness.

Phase 8: Deployment
Objective: Make the application live for users.
Activities: Deploy the backend to a cloud service (e.g., AWS, Heroku) and frontend to a hosting platform (e.g., Netlify, Vercel).Set up Continuous Deployment (CD) pipelines for easy updates.Implement monitoring tools to track performance and errors in real-time.Set up a backup mechanism to ensure user data is safe.

Phase 9: Post-Launch Monitoring and Maintenance
Objective: Ensure the app performs well in production and address any issues that arise.
Activities: Monitor the application for bugs, crashes, or performance issues.Collect user feedback to improve the app and prioritize new features.Regularly update the app with security patches, bug fixes, and new features.Optimize the app for scalability as the number of users grows.

Phase 10: Future Enhancements (Optional)
Objective: Add advanced features and improve the application.
Activities: Implement AI-based categorization of expenses or predictive analytics for budgeting.
Integrate with external APIs (e.g., bank integration for automatic transaction importing).
Add features for investment tracking or integration with cryptocurrency portfolios.

Improve the user experience (UX) based on feedback (e.g., dark mode, advanced reporting).

TECHNOLOGY STACK

Frontend: React.js

Backend: Node.js, Express.js

Database: PostgreSQL

ORM: Prisma

Parsing Libraries: PapaParse (CSV), Mammoth (DOCX)

LIMITATIONS AND FUTURE WORK

LIMITATIONS:
Limited support for non-standard bank statement formats
Manual categorization might be needed for uncategorized transactions
Performance optimization needed for very large datasets.

FUTURE WORK:
AI Categorization:
Use Machine Learning to auto-categorize transactions more accurately.
Mobile App:
Build a companion mobile application.
Financial Goals Integration:
Allow users to set and track financial goals like saving for a car, home, etc.
Investment Tracker:
Add support to monitor stocks, mutual funds, and crypto investments.

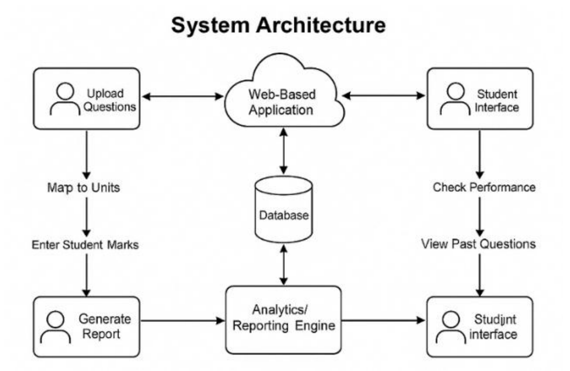
CONCLUSION

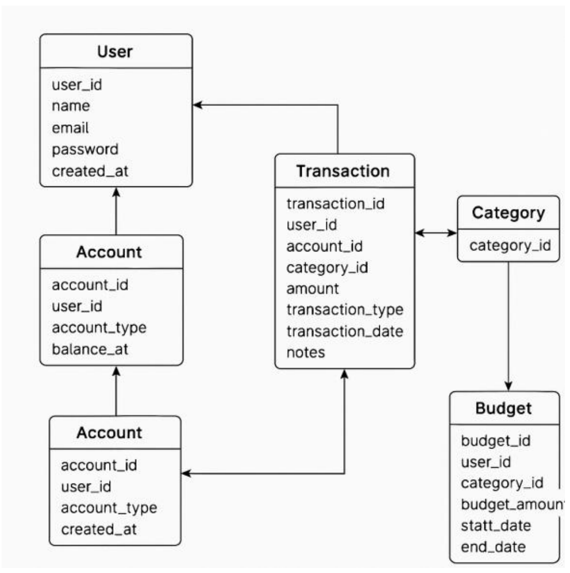
The Personal Finance Tracker offers an efficient way for individuals to manage their finances with clarity and confidence. It simplifies the process of tracking income and expenses, encourages goal-oriented saving, and enables better financial decisions. By offering a secure, intuitive, and insightful platform, the system helps users build healthy financial habits and take control of their financial future.

REFERENCES

[1] Prisma, "Prisma ORM Documentation," [Online]. Available: https://www.prisma.io/docs/.
[2] Facebook, "React: A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org/.
[3] Mamoth.js, "Mamoth: Convert .docx documents into HTML," [Online].Available: https://github.com/mwilliamson/mamoth.

APPENDIX


System Architecture
Upload Questions
Web-Based Application
Student Interface
Map to Units
Enter Student Marks
Database
Check Performance
View Past Questions
Generate Report
Analytics/ Reporting Engine
Studjnt interface


User
user_id
name
email
password
created_at
Transaction
transaction_id
user_id
account_id
category_id
amount
transaction_type
transaction_date
notes
Category
category_id
Account
account_id
user_id
account_type
balance_at
Account
account_id
user_id
account_type
created_at
Budget
budget_id
user_id
category_id
budget_amoun
statt_date
end_date