Expense Tracker

Project Report

By
Nilesh Bokoliya – AF04969556
Ravish Singh – AF04969962

Index

Acknowledgement

The project **"Expense Tracker"** is the Project work carried out by

Name	Enrollment No
Nilesh Bokoliya	**AF04969556**
Ravish Singh	**AF04969962**

We are thankful to my project guide for guiding me to complete the Project.

Their suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

Abstract

Managing personal finances effectively is a critical aspect of modern life, yet many individuals struggle with tracking expenses, setting savings goals, and maintaining financial discipline. **Expense Tracker** is a full-stack personal finance management system designed to address these challenges by providing users with an intuitive and interactive platform. The application enables users to record income and expenses, categorize transactions, set financial goals, and monitor their progress through insightful visualizations such as pie charts and reports. A dedicated savings feature allows users to allocate funds toward specific goals, ensuring structured financial growth. The system emphasizes simplicity, security, and usability by integrating authentication, data visualization, and dynamic reporting. By combining expense tracking, goal management, and financial insights in one platform, Expense Tracker empowers users to make informed financial decisions, develop better saving habits, and achieve long- term financial stability

CHAPTER 1: INTRODUCTION

1.1 Background

In today's fast-paced digital world, managing personal finances has become increasingly complex. With rising living costs, multiple income sources, and diverse spending patterns, individuals often struggle to maintain control over their financial health. Traditional methods like manual ledger entries or spread sheet tracking are time-consuming and prone to errors. This has led to a growing demand for automated, user-friendly financial management tools.

The concept of digital expense tracking has evolved significantly over the past decade. From basic mobile apps to full-stack web applications, these tools now offer features like real-time transaction logging, goal-based savings, automated reporting, and secure authentication. According to recent studies, over 60% of young professionals use some form of digital finance tracker to monitor their spending habits and improve financial discipline.

This project, **Expense Tracker**, is designed to address the need for a secure, scalable, and intuitive web-based solution that empowers users to manage their income, expenses, and savings goals efficiently. Built using modern technologies like **Node.js, React, and MySQL**, the system provides a seamless interface for tracking financial data, setting goals, and generating insightful reports.

1.2 Objectives

The primary objectives of this project are:

1. To develop a secure and responsive web application for tracking income and expenses.
2. To implement user authentication and password recovery using JWT and secure token generation.
3. To enable users to create and monitor financial goals with contribution tracking.
4. To generate downloadable reports (Excel) for transaction history and financial summaries.
5. To ensure data integrity through proper database design and validation rules.
6. To provide an admin interface for monitoring system usage and managing user roles.

These objectives collectively aim to deliver a robust, production-ready application that can be extended for future enhancements such as mobile integration, AI-based forecasting, and multi-user household budgeting.

1.3 Purpose

The purpose of this project is to design and implement a full-stack **Expense Tracker** system that helps individuals gain better control over their personal finances. By digitizing the process of expense logging and goal setting, the application reduces manual effort, minimizes errors, and enhances financial awareness. It also serves as a practical demonstration of modern web development principles, including RESTful API design, database normalization, and secure session management.

1.4 Scope

The scope of the project includes:

- **Frontend**: A responsive React-based user interface for all core functionalities.
- **Backend**: A Node.js server with REST APIs for user management, transaction handling, goal tracking, and reporting.
- **Database**: A normalized MySQL schema with tables for users, transactions, goals, contributions, and password reset tokens.
- **Authentication**: Secure login, registration, and password reset using JWT and encrypted tokens.
- **Reporting**: Exportable Excel reports for transaction history and financial summaries.

The system is designed for individual users and does not currently support shared accounts or multi-currency transactions. Future versions may expand these capabilities.

1.5 Applicability

This application is applicable to:

- **Students and young professionals** who want to track monthly expenses and savings.
- **Freelancers and self-employed individuals** managing irregular income streams.
- **Budget-conscious households** aiming to monitor spending patterns.
- **Software engineering students** learning full-stack development with real-world projects.

It can also serve as a foundation for more advanced financial tools, such as investment trackers or tax calculators, by extending its modular architecture.

1.6 Achievements

By the end of this project, the following outcomes have been achieved:

- A fully functional **CRUD-based expense tracking system** with secure authentication.
- Implementation of **JWT-based session management** and password reset workflow.
- Design and integration of a **normalized database schema** (ER Diagram) with referential integrity.
- Development of **three-level DFDs** (Context, Level 1, Level2) for clear system visualization.
- Creation of **30+ test cases** covering all critical API endpoints and business logic.
- Generation of downloadable **Excel reports** for financial analysis.
- Modular, maintainable codebase following MVC architecture and RESTful design principles.

These achievements demonstrate the successful application of software engineering concepts in building a real-world, scalable web application.

1.7 Organization of Report

This report is organized into the following chapters:

- **Introduction** – Presents the background, objectives, scope, and structure of the project.
- **Literature Survey** – Reviews existing systems, research papers, and technological trends in personal finance management.
- **System Analysis and Feasibility Study** – Analyzes functional and non-functional requirements and evaluates technical, economic, and operational feasibility.
- **System Design** – Details the architectural design, ER diagram, DFDs, and database schema.
- **Implementation** – Describes the development environment, tools used, and key implementation strategies.
- **Testing** – Presents test cases, results, and validation of system behavior.
- **Results and Conclusion** – Summarizes findings, challenges faced, and overall outcomes.
- **Future Scope** – Suggests potential enhancements like mobile app, AI forecasting, and cloud sync.

CHAPTER 2: SYSTEM ANALYSIS

2.1 Problem Definition

Managing personal finances manually using spread sheets or paper-based methods is inefficient, error-prone, and lacks real-time insights. Users often face challenges such as:

- Difficulty in tracking daily income and expenses accurately.
- Lack of goal-oriented savings planning and progress monitoring.
- Inability to generate structured financial reports for analysis.
- Poor visibility into spending patterns across categories (e.g., food, travel, bills).
- Security risks with unencrypted storage of financial data.

Existing solutions like basic mobile apps or generic spread sheets do not offer a secure, full-featured web-based platform with role-based access, automated reporting, and data integrity. This project addresses these gaps by developing a **secure, scalable, and user-friendly Expense Tracker** system that enables individuals to manage their financial data efficiently through a centralized web application.

2.2 Objectives of the System

The primary objectives of the proposed system are:

- To provide a **secure web-based platform** for recording and managing income and expense transactions.
- To implement **user authentication and authorization** using JWT tokens and encrypted password storage.
- To allow users to **set financial goals** and track contributions toward achieving them.
- To enable **real-time data visualization** and **exportable Excel reports** for financial summaries.
- To ensure **data consistency and referential integrity** through a normalized database schema.
- To support **admin-level monitoring** for future expansion (e.g., user activity logs, system health).
- To follow **RESTful API design principles** for scalability and maintainability.

These objectives ensure the system is not only functional but also robust, secure, and aligned with modern software development practices.

2.3 Feasibility Study

A comprehensive feasibility study has been conducted to evaluate the practicality of the system from technical, economic, operational, and schedule perspectives.

2.3.1 Technical Feasibility

The system is technically feasible as it uses well-established, open-source technologies:

- **Frontend**: React.js – a widely adopted library for building responsive UIs.
- **Backend**: Node.js with Express – ideal for scalable, non-blocking API development.
- **Database**: MySQL – a reliable relational database with strong ACID compliance.
- **Authentication**: JWT (JSON Web Tokens) for stateless session management.
- **Reporting**: Excel export using exceljs or similar libraries.

All components are compatible, well-documented, and supported by large developer communities, ensuring smooth integration and troubleshooting.

2.3.2 Economic Feasibility

The project is economically viable because:

- All tools and frameworks used are **open-source and free**.
- Development can be done on standard hardware without requiring high-end infrastructure.
- Hosting options like **Vercel (frontend)** and **Railway/Render (backend)** offer free tiers for deployment.
- No licensing costs involved, making it suitable for academic and personal use.

The total cost of development is minimal, limited only to time and effort.

2.3.3 Operational Feasibility

The system is operationally feasible because:

- The user interface is intuitive and requires minimal training.
- Users can access the application from any device with a modern browser.
- Features like password reset, form validation, and error handling enhance usability.
- Admin capabilities can be extended later without disrupting core functionality.

End-users (students, professionals) will find the system easy to adopt for daily financial tracking.

2.3.4 Schedule Feasibility

The project timeline is realistic and achievable within an academic semester:

- **Week 1–2**: Requirement gathering and planning
- **Week 3–4**: Database design and ER diagram finalization
- **Week 5–6**: Backend API development
- **Week 7–8**: Frontend implementation
- **Week 9**: Integration and testing
- **Week 10**: Documentation and final submission

2.4 System Requirements

2.4.1 Functional Requirements

ID	Requirement	Description
FR-01	User Registration	New users can register with name, email, password, and salary.
FR-02	User Login	Registered users can log in using email and password; JWT token is issued.
FR-03	Password Reset	Users can request a reset link; a time-limited token is generated and stored.
FR-04	Add Transaction	Users can add income/expense with type, category, amount, and date.
FR-05	View Transactions	Users can view all their transactions in a list or table format.
FR-06	Update/Delete Transaction	Users can edit or remove their own transactions.
FR-07	Create Goal	Users can set a savings goal with name, target amount, and end date.
FR-08	Add Contribution	Users can add money toward a specific goal.
FR-09	Track Progress	System calculates and displays progress percentage for each goal.
FR-10	Generate Report	Users can download a monthly/overall transaction report in Excel format.
FR-11	View Profile	Users can view and update their profile (name, salary).
FR-12	Admin Access	Future role to monitor users and system status (currently disabled).

2.4.2 Non-Functional Requirements

ID	Requirement	Description
NFR-01	Security	Passwords must be hashed (bcrypt); JWT tokens must expire.
NFR-02	Performance	API responses should be under 500ms for standard operations.
NFR-03	Usability	UI must be responsive and work on desktop and mobile devices.
NFR-04	Reliability	System should handle invalid inputs gracefully with proper error messages.
NFR-05	Scalability	Architecture should support future addition of features (e.g., mobile app).
NFR-06	Data Integrity	Foreign key constraints and validation rules must enforce consistency.

2.5 Proposed System

The proposed **Expense Tracker** system is a full-stack web application designed to simplify personal financial management. It consists of:

- A **React-based frontend** with clean, responsive UI components for all user interactions.
- A **Node.js backend** exposing RESTful APIs for user, transaction, goal, and report management.
- A **MySQL database** with normalized tables ensuring data integrity and efficient querying.
- **JWT-based authentication** for secure session handling without server-side state.
- **Role-based access control** (User/Admin) to support future administrative functions.

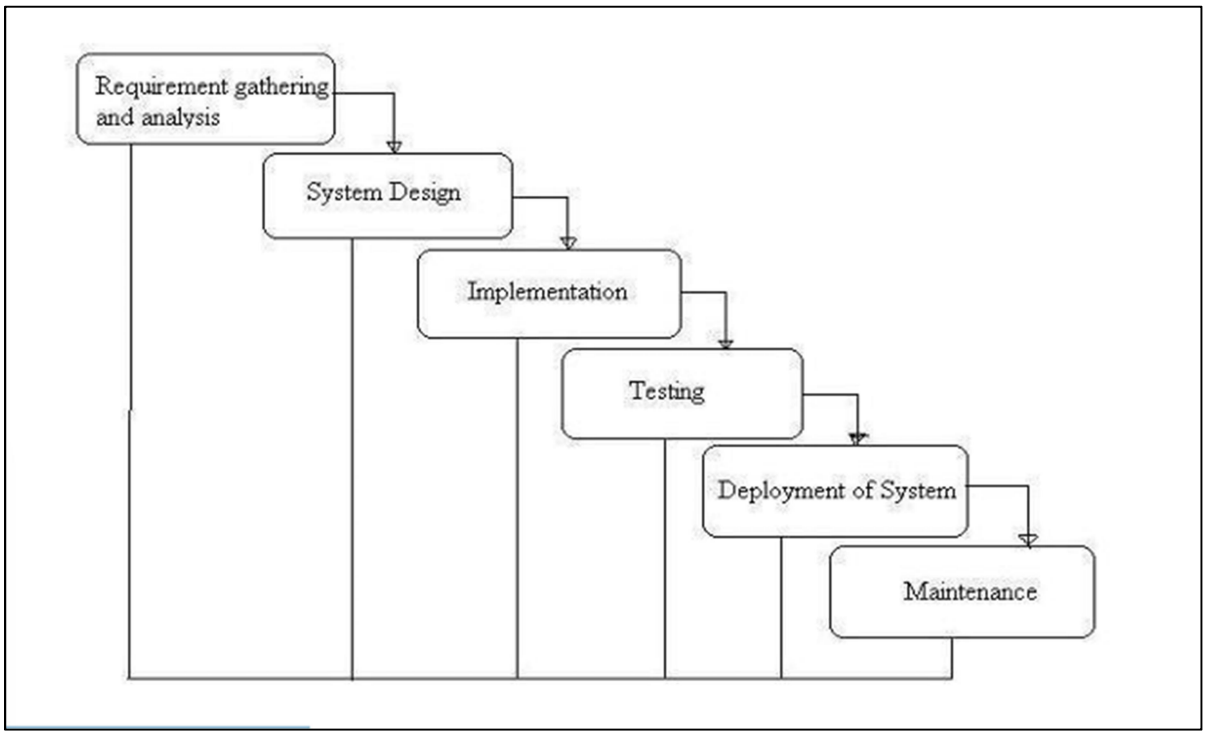
Key features include:

- Secure login and password recovery
- CRUD operations for transactions and goals
- Goal contribution tracking with progress visualization
- Exportable Excel reports for financial analysis
- Input validation and error handling at both frontend and backend

The system follows **MVC (Model-View-Controller)** architecture, ensuring separation of concerns and maintainable code. It also supports **future enhancements** such as AI-based spending predictions, multi-user household budgets, and mobile app integration.

2.6 Phases of Development

The development of the system follows a **hybrid Agile-Waterfall model**, combining structured planning with iterative implementation.



Phase 1: Requirement Analysis

- Gather functional and non-functional requirements.
- Identify user roles and system boundaries.
- Define scope and deliverables.

Phase 2: System Design

- Design **ER Diagram** for database schema.
- Create **DFD Level 0, 1, and 2** for data flow visualization.
- Finalize API endpoints and request/response structures.
- Design wireframes for key screens (dashboard, transaction form, goal tracker).

Phase 3: Implementation

- Set up backend server with Express and connect to MySQL.
- Implement authentication (register, login, password reset).
- Develop CRUD APIs for transactions and goals.
- Build frontend pages using React components.

- Integrate frontend with backend via Axios/Fetch.

Phase 4: Testing

- Perform **unit testing** on API endpoints.
- Conduct **integration testing** between frontend and backend.
- Execute **system testing** using 30+ test cases (as defined in project document).
- Validate security, performance, and usability.

Phase 5: Deployment & Documentation

- Deploy frontend to Vercel and backend to Railway.
- Prepare final project report (this document).
- Create user manual and technical documentation.
- Submit deliverables for evaluation.

This phased approach ensures systematic development, early detection of issues, and timely completion of the project.

2.7 Data Flow Diagram (DFD)

A **Data Flow Diagram (DFD)** is a traditional visual representation of the information flows within a system. A neat and clear DFD depicts the proper scope of system requirements graphically. It can represent manual processes, automated processes, or a combination of both. The DFD shows how data enters and leaves the system, what transforms the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It is a communication tool between system analysts and stakeholders and often acts as the starting point for redesigning a system. The DFD is also called a **data flow graph** or **bubble chart**.

Important observations about DFDs

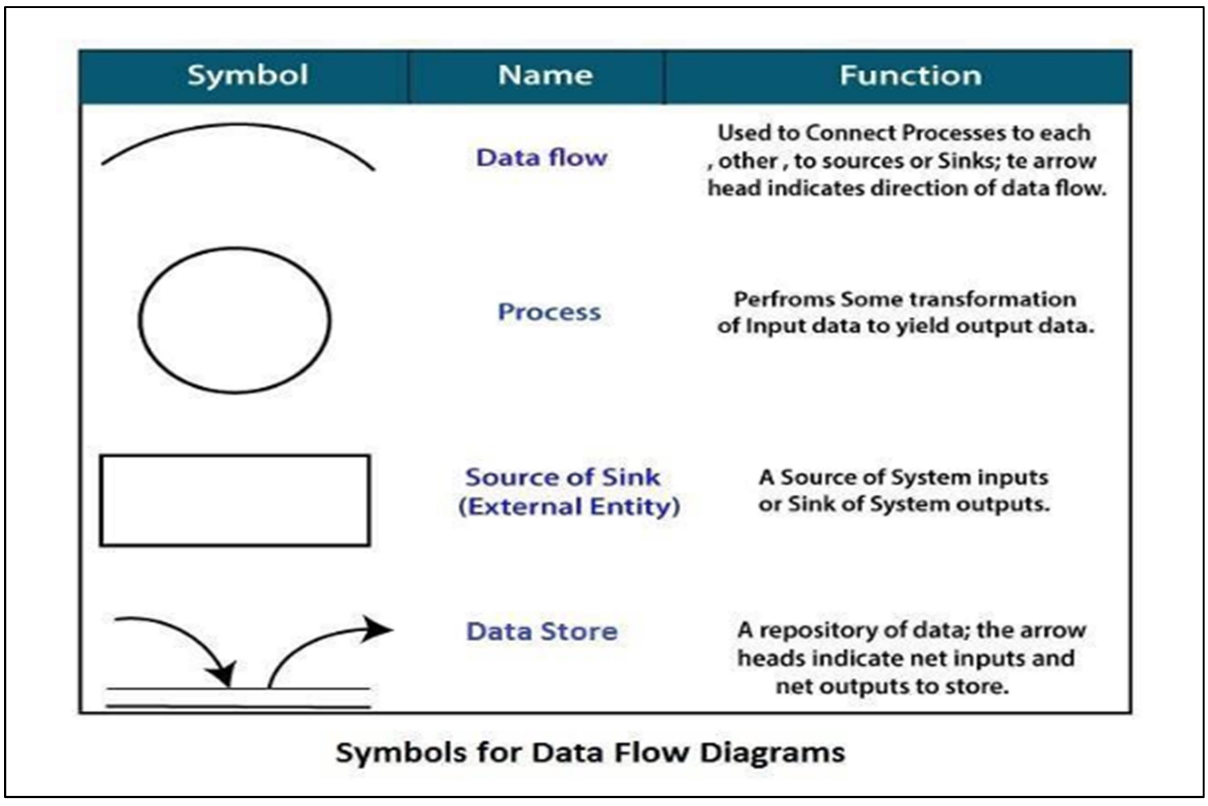
- **All names should be unique.** Unique names make referencing and discussing DFD elements straightforward and unambiguous.
- **A DFD is not a flowchart.** In flowcharts arrows represent sequence (order of events); in DFDs arrows represent **data flow** only — not control flow or ordering.
- **Suppress logical decision nodes.** Do not use diamond-shaped decision boxes (common in flowcharts). Decision boxes imply control flow and ordering, which contradicts the declarative nature of DFDs.

- **Avoid excessive detail.** Keep the diagram focused on the essential data movements. Defer error handling, exception flows, and low-level implementation details to lower-level diagrams or separate design documents.

Standard symbols used in DFDs

- **Process (Circle / Rounded Rectangle / Bubble):** Represents a function or transformation that converts input data into output data. Example labels: 1.0 Process Transaction, 2.0 Update Goal Progress.
- **Data Flow (Arrow):** A directed line with an arrow that shows the movement of data between processes, data stores, and external entities. The arrow is labelled with the data being transferred (e.g., transaction details, auth token).
- **Data Store (Open-ended Parallel Lines):** Represents storage (files, database tables). Label with the store name (e.g., Transactions DB, Users).
- **External Entity / Source-Sink (Rectangle):** An outside system or actor that provides input or receives output from the system (e.g., User, Email Server, Bank API).

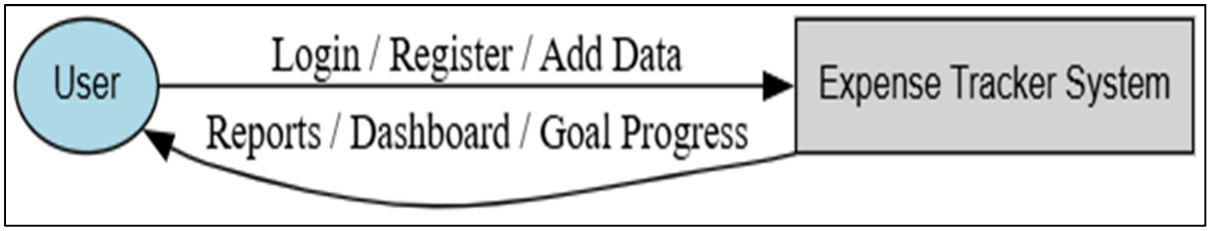
Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in below figure.



Symbols for Data Flow Diagrams

- **Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.
- **Data Flow:** A curved line shows the flow of data into or out of a process or data store.
- **Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.
- **Source or Sink:** Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

2.7.1 Zero-Level DFD (Context Diagram) of Expense Tracker



The **Zero-Level DFD**, also known as the **Context Diagram**, represents the entire **Expense Tracker System** as a single, unified process.
It illustrates how the system interacts with its external entities and how data flows between them. This top-level diagram provides a high-level overview of system boundaries, data inputs, and outputs.

Entities and Data Flows

1. User (External Entity)

- The primary actor who interacts with the system through the web interface.
- **Inputs provided by the User:**
  - Registration details (name, email, password).
  - Login credentials for authentication.
  - Financial data entries (income, expenses, categories).
  - Savings goal information (target amount, duration, description).
  - Requests for generating reports or viewing dashboard summaries.
- **Outputs received by the User:**
  - Authentication success/failure messages.
  - Notifications such as confirmations or error alerts.
  - Dashboard visualizations including pie charts, bar graphs, and spending summaries.
  - Downloadable reports (Excel format).
  - Progress updates toward goals.

2. Email Server (External Entity)

- **Functions:**
  - Sends OTPs during registration or password reset for account verification.
  - Sends transaction alerts or reminders for goal deadlines.
  - Confirms successful actions like account creation or password update.
- Ensures system security by enabling two-step verification for user authentication.

3. Database (External Entity)

- Acts as the **central repository** for storing and retrieving all system data.
- **Data Stored:**
  - User credentials and profiles (securely encrypted).
  - Transaction records (income, expense, and categories).
  - Goal details and progress contributions.
  - Reports and analytical summaries.
- **Data Retrieved:**
  - User-specific dashboards and charts.
  - Transaction histories filtered by date or category.
  - Goal completion status and contribution records.
  - Monthly and yearly financial summaries.

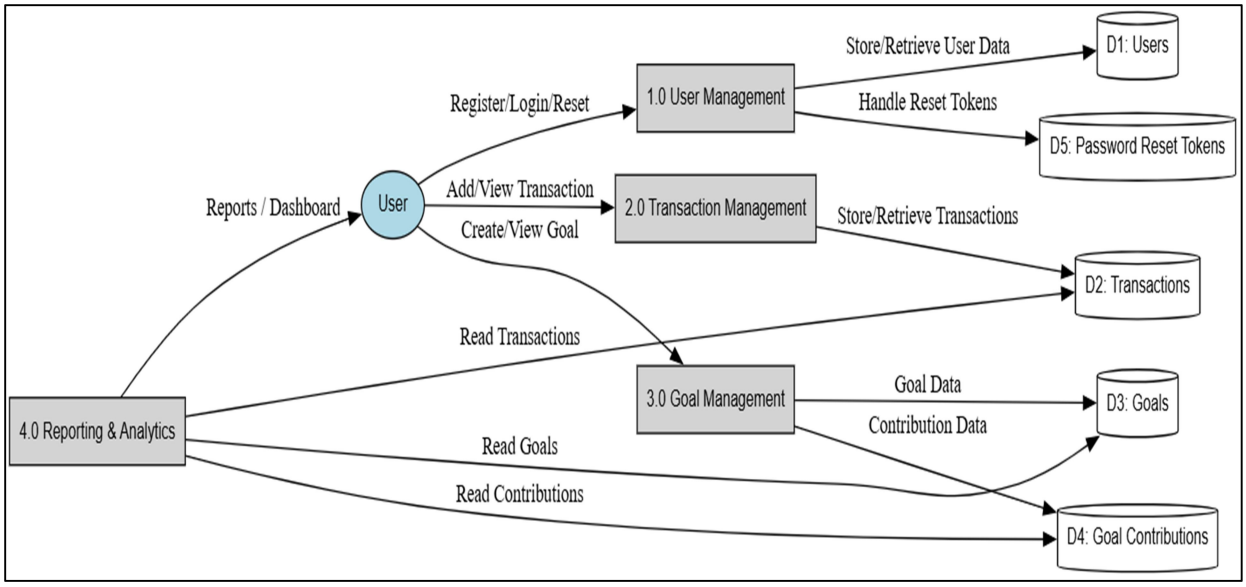
Central Process: Expense Tracker System

The **Expense Tracker System** is the core process in this diagram. It receives inputs from the **User**, processes them, and stores or retrieves data from the **Database**. It also communicates with the **Email Server** for sending authentication codes, notifications, and reminders. All user actions such as adding transactions, setting goals, or viewing reports are managed and validated through this central process.

Overall Flow

- The **User** interacts with the **Expense Tracker System** to perform tasks like registration, login, adding transactions, or setting financial goals.
- The system validates the credentials and may interact with the **Email Server** to send verification codes or notifications.
- The system communicates with the **Database** to store or retrieve data such as transactions, goals, or user profiles.
- Processed information (e.g., dashboard insights, financial summaries, or goal progress) is returned to the **User** in a graphical and intuitive format.

2.7.2 First-Level Data Flow Diagram



The **First-Level Data Flow Diagram (DFD)** decomposes the overall **Expense Tracker System** into its major functional processes, showing how the system interacts with users, the database, and supporting external services.
This diagram represents the flow of data between different subsystems responsible for user management, transaction handling, goal tracking, and reporting functionalities.

1. User Management (Process 1.0)

Inputs

- Registration details (name, email and password).
- Login credentials for authentication.
- Requests for password reset or OTP verification.

Processes

- **Registration:** Collects user details and validates for uniqueness.
- **Login:** Authenticates credentials stored in the user database.
- **Password Reset:** Handles reset token generation and verification via email.

Data Flows

- User credentials are stored in and retrieved from the **User Database (D1)**.
- Reset tokens are generated and stored in **Password Reset Tokens (D5)**.
- OTPs or reset confirmations are sent through the **Email Server** for verification.

Outputs

- Successful registration or login confirmation sent to the user.
- Error or success notifications via email and dashboard.

2. Transaction Management (Process 2.0)

Inputs

- Expense, income, or savings transaction details entered by the user.

Processes

- **Add Transaction:** Stores new transaction records (amount, category, date, description).
- **View Transaction:** Retrieves and displays existing transaction history.
- **Categorization:** Classifies transactions as income, expense, or goal contribution.

Data Flows

- Transaction details stored and retrieved from the **Transactions Database (D2)**.
- Data read by the **Reporting & Analytics module** for summaries and reports.

Outputs

- Updated transaction list visible on the dashboard.
- Categorized expense and income summaries.

3. Goal Management (Process 3.0)

Inputs

- User-defined goal details such as goal name, target amount, description, and end date.
- Optional goal contributions linked to transactions.

Processes

- **Add Goal:** Creates and stores new financial goals.
- **Track Goal:** Updates progress as users add contributions.
- **Modify Goal:** Edits existing goals or marks them as completed.

Data Flows

- Goal details stored and retrieved from the **Goals Database (D3)**.
- Contribution records stored in **Goal Contributions (D4)**.
- Read operations used by the Reporting module for visual summaries.

Outputs

- Goal progress status (Active, Achieved, or Expired).
- Notifications and reminders for incomplete goals.

4. Reporting & Analytics (Process 4.0)

Inputs

- User requests for reports, dashboards, or summaries.

Processes

- **Generate Reports:** Compiles user transactions and goal data.
- **Visualize Data:** Displays charts showing spending vs. income.
- **Export Reports:** Generates downloadable reports in Excel or CSV formats.

Data Flows

- Reads data from all relevant databases: **Transactions (D2)**, **Goals (D3)**, and **Goal Contributions (D4)**.
- Returns analytical summaries and visual reports to the **User Interface**.

Outputs

- Graphical dashboards (pie charts, bar charts).
- Downloadable Excel financial reports.

5. Salary Management (Optional Embedded Process)

Inputs
- User-provided or updated monthly salary amount.

Processes
- **Store Salary:** Saves or modifies salary records.
- **Compute Balance:** Calculates current available balance (Salary − Expenses + Savings).

Data Flows
- Salary data stored in the **User Database (D1)** or a dedicated **Salary Table**.
- Retrieved by the Reporting module for financial summaries.

Outputs
- Updated balance reflected on the dashboard and reports.

External Entities

User
- The primary actor interacting with all system components for adding data, retrieving reports, and managing goals.

Email Server

- Handles the sending of OTPs, password resets, and notifications during login or registration.

Database

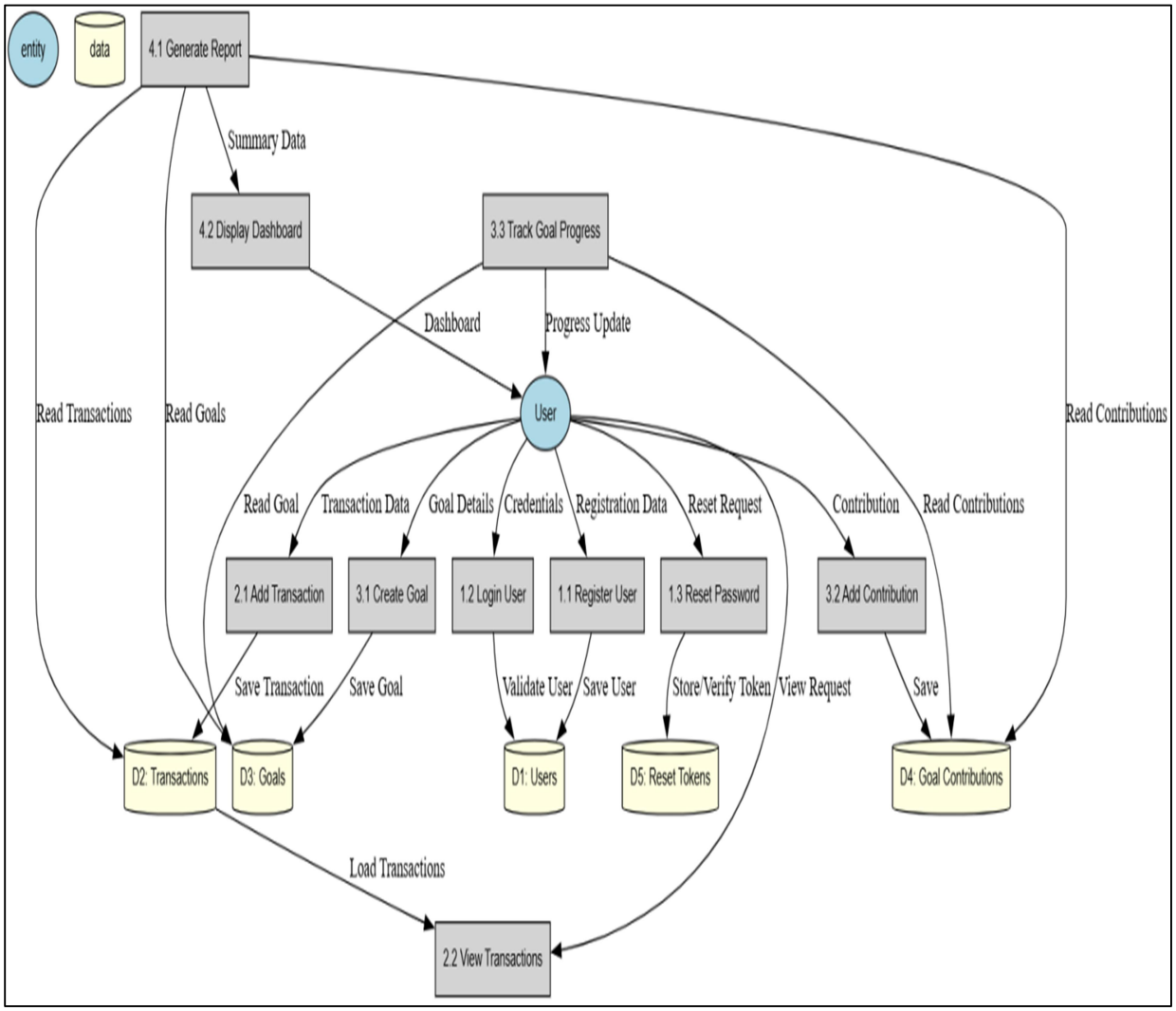
- Central storage handling user credentials, transactions, goals, salary, and generated reports.

Overall System Flow

- The **User** interacts with the system to register, log in, and manage transactions and goals.
- The **User Management Module** verifies credentials, handles authentication, and stores user data securely.
- The **Transaction Module** records all income and expenses, categorizing them as required.
- The **Goal Management Module** allows users to create and track financial goals, linking them to transactions.

- The **Reporting Module** retrieves data from all databases to generate charts and reports.
- Data moves seamlessly among **User**, **Database**, and **System Modules**, ensuring up-to-date dashboard insights and downloadable reports.

2.7.3 Second-Level Data Flow Diagram (DFD)



The **Second-Level Data Flow Diagram (DFD)** provides a detailed breakdown of the major processes identified in the First-Level DFD.
It illustrates the **sub-processes**, the **specific data stores**, and the **data interactions** between internal system components.
At this level, each main process (like user management, transaction handling, and goal tracking) is expanded into finer sub-processes that define the internal workflow of the **Expense Tracker** system.

1. User Management

Inputs

- User credentials during registration or login.
- Profile update requests (such as salary or personal details).

Processes

- **1.1 Register User:** Validates registration information and stores new user details in the database.
- **1.2 Login User:** Authenticates credentials using encrypted data stored in the `Users` table.
- **1.3 Reset Password:** Generates and validates reset tokens via email verification.

Outputs

- Successful authentication or registration confirmation.
- Error or success notifications.
- Updated user session or profile details.

Data Stores

- **D1: Users** — Stores user credentials, profiles, and salary information.
- **D5: Reset Tokens** — Stores temporary reset and verification tokens.

2. Transaction Management

Inputs

- Transaction details including amount, category, date, notes, and optional goal association.

Processes

- **2.1 Add Transaction:** Adds new income or expense records into the system.
- **2.2 View Transactions:** Retrieves transaction history based on filters such as date range or category.
- **Categorization:** Classifies entries as *income*, *expense*, or *goal contribution*.

Outputs

- Updated transaction history and categorized financial summaries.

Data Stores

- **D2: Transactions** — Stores all transaction records linked with user IDs.

3. Goals Management

Inputs

- Goal details such as goal name, target amount, end date, and description.
- Linked transaction data for contribution updates.

Processes

- **3.1 Create Goal:** Allows users to set new financial goals with a target value and timeline.
- **3.2 Add Contribution:** Adds money toward a specific goal and updates progress.

- **3.3 Track Goal Progress:** Continuously updates status (Active, Achieved, or Expired) based on user contributions and dates.

Outputs

- Progress summary for each financial goal.
- Notifications regarding goal completion or pending contributions.

Data Stores

- **D3: Goals** — Contains details of each goal created by users.
- **D4: Goal Contributions** — Stores individual contribution records mapped to specific goals.

4. Report Generation

Inputs

- Transaction and goal data from the database.

Processes

- **4.1 Generate Report:** Aggregates all user data (expenses, income, savings) to form reports.
- Summarizes monthly salary, spending categories, and goal performance.
- Exports financial summaries for users in Excel or CSV format.

Outputs

- Downloadable analytical reports (Excel/CSV).
- Summary charts and graphs.

Data Stores

- **D2: Transactions**, **D3: Goals**, and **D4: Goal Contributions** — Used as input sources for generating reports.

5. Dashboard & Visualization

Inputs

- Summarized salary, income, expenses, and savings data from reporting modules.

Processes

- **4.2 Display Dashboard:** Presents visual reports using interactive UI components.
- Renders graphical charts such as:
    - Pie charts (Expenses vs. Income)
    - Bar graphs (Monthly trends)
    - Progress bars (Goal tracking)
- Displays current balance and total savings dynamically.

Outputs

- Real-time financial dashboard visible on the user interface.
- Intuitive data visualizations showing income distribution and spending trends.

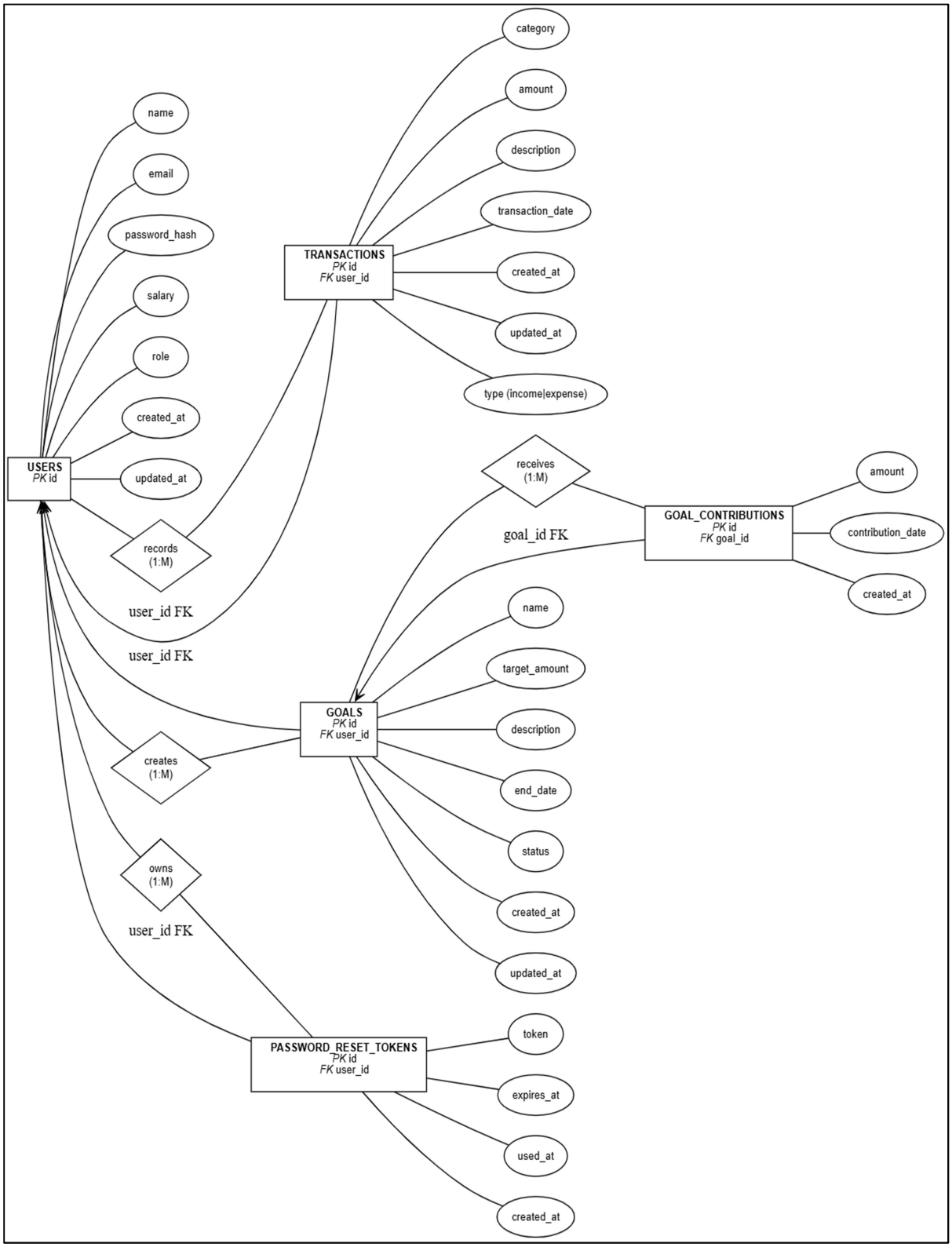
Data Stores in Second-Level DFD

Data Store	Description
D1: Users	Stores user information such as `user_id`, `name`, `email`, `password_hash`, and `salary`.
D2: Transactions	Maintains all user transactions (income, expense, category, date, notes, and goal links).
D3: Goals	Contains user-defined goals with target amounts, end dates, and current status.
D4: Goal Contributions	Records contributions made toward each goal and links them to transactions.
D5: Reset Tokens	Stores verification and password reset tokens for authentication purposes.

Overall System Flow

- **User Management** handles registration, login, and profile updates, storing user data securely in the database.
- **Transaction Management** captures and retrieves all income and expense records.
- **Goal Management** maintains user goals, tracks progress, and stores contributions.
- **Report Generation** aggregates and summarizes all data to produce comprehensive reports.
- **Dashboard & Visualization** module presents financial insights through dynamic and interactive charts.

ER Diagram


category
amount
name
description
email
transaction_date
password_hash
TRANSACTIONS
PK id
FK user_id
salary
created_at
role
updated_at
type (income|expense)
created_at
receives
(1:M)
amount
USERS
PK id
updated_at
GOAL_CONTRIBUTIONS
PK id
FK goal_id
contribution_date
records
(1:M)
goal_id FK
created_at
user_id FK
name
user_id FK
target_amount
GOALS
PK id
FK user_id
description
creates
(1:M)
end_date
status
owns
(1:M)
created_at
user_id FK
updated_at
token
PASSWORD_RESET_TOKENS
PK id
FK user_id
expires_at
used_at
created_at

The **ER diagram for the Expense Tracker** system illustrates the logical structure of its database, showing all entities, their attributes, and the relationships among them. It provides a clear and normalized representation of how data is stored and interconnected across the system to manage users, financial transactions, goals, and related processes effectively.

1. Entities and Their Attributes

User

- **Attributes:** id (PK), name, email, password_hash, salary, role, created_at, updated_at
- Represents the registered users of the system who can log in, add expenses/income, and manage financial goals.

Transaction

- **Attributes:** id (PK), user_id (FK), category, amount, description, transaction_date, type (income/expense), created_at, updated_at
- Captures all financial activities of the user, including income and expenses.
- Each transaction is linked to a user and can optionally be associated with a financial goal.

Goal

- **Attributes:** id (PK), user_id (FK), name, target_amount, description, end_date, status, created_at, updated_at
- Defines user-set financial goals such as saving for travel, emergencies, or specific purchases.
- Each goal tracks progress and completion status.

Goal Contributions

- **Attributes:** id (PK), goal_id (FK), amount, contribution_date, created_at
- Stores the contributions made toward specific financial goals.
- Each contribution updates the progress of a particular goal.

Password Reset Tokens

- **Attributes:** id (PK), user_id (FK), token, expires_at, used_at, created_at
- Manages password reset functionality by generating and validating secure tokens for each user.

2. Relationships

- **User → Transaction:** One user can record many transactions (**1 : M**).
- **User → Goal:** One user can create multiple goals (**1 : M**).
- **Goal → Goal Contributions:** A goal can have multiple contributions, but each contribution belongs to one goal (**1 : M**).
- **User → Password Reset Tokens:** A user can have multiple password reset attempts (**1 : M**).

- **Transaction → Goal:** Each transaction may optionally be linked to a specific goal (**M : 1**).

3. ER Diagram Summary

- The **User** entity is the central component connecting all other modules — Transactions, Goals, and Reset Tokens.
- **Transactions** serve as a key bridge between user activities and their financial goals.
- **Goal Contributions** provide a detailed record of incremental savings progress.
- The database structure ensures **data normalization**, **referential integrity**, and **efficient querying**, supporting seamless expansion for features such as budget insights and performance analytics.

CHAPTER 3: SYSTEM DESIGN

3.1 Modules

The **Expense Tracker** system is modular in design, ensuring separation of concerns and maintainable code. The application is divided into the following core modules:

Module	Description
1. Authentication Module	Handles user registration, login, and password reset functionality. Uses JWT for secure session management and bcrypt for password hashing.
2. User Management Module	Allows users to view and update their profile (name, salary). Only authenticated users can access their own data.
3. Transaction Management Module	Enables users to add, view, edit, and delete income and expense records. Supports filtering by date, category, and type.
4. Goal Management Module	Allows users to create savings goals (e.g., "Buy Laptop", "Vacation Fund") with target amounts and deadlines.
5. Contribution Tracking Module	Users can add contributions toward a goal. The system calculates progress percentage automatically.
6. Reporting Module	Generates downloadable Excel reports of all transactions. Includes summary statistics like total income, expenses, and balance.
7. Admin Module (Future Scope)	Reserved for administrative roles to monitor user activity, system health, and manage roles. Currently restricted to prevent unauthorized access.

3.2 Data Structure of All Modules

This section defines the **database schema** and data structure for each module.

3.2.1 Database Schema Overview

The system uses **MySQL** with the following normalized tables:

Table	Purpose
`USERS`	Stores user credentials and profile data
`TRANSACTIONS`	Records all income and expense entries
`GOALS`	Stores user-defined financial goals
`GOAL_CONTRIBUTIONS`	Tracks individual contributions toward each goal
`PASSWORD_RESET_TOKENS`	Stores time-limited tokens for password recovery

3.2.2 Table Structures

1. USERS

```
sql
CREATE TABLE USERS (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  salary DECIMAL(12,2),
  role ENUM('user','admin') DEFAULT 'user',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

2. TRANSACTIONS

```
sql
CREATE TABLE TRANSACTIONS (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  type ENUM('income','expense') NOT NULL,
  category VARCHAR(100) NOT NULL,
  amount DECIMAL(12,2) NOT NULL,
  description VARCHAR(255),
  transaction_date DATE NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES USERS(id) ON DELETE CASCADE
);
```

3. GOALS

```
sql
CREATE TABLE GOALS (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  target_amount DECIMAL(12,2) NOT NULL,
  description VARCHAR(255),
```

```
  end_date DATE,
  status ENUM('active','achieved','expired') DEFAULT 'active',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES USERS(id) ON DELETE CASCADE
);
```

4. GOAL_CONTRIBUTIONS

```
sql
CREATE TABLE GOAL_CONTRIBUTIONS (
  id INT PRIMARY KEY AUTO_INCREMENT,
  goal_id INT NOT NULL,
  amount DECIMAL(12,2) NOT NULL,
  contribution_date DATE NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  FOREIGN KEY (goal_id) REFERENCES GOALS(id) ON DELETE CASCADE
);
```

5. PASSWORD_RESET_TOKENS

```
sql
CREATE TABLE PASSWORD_RESET_TOKENS (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  token VARCHAR(128) NOT NULL UNIQUE,
  expires_at DATETIME NOT NULL,
  used_at DATETIME NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES USERS(id) ON DELETE CASCADE
);
```

These tables follow **3NF (Third Normal Form)** to eliminate redundancy and ensure data integrity.

3.3 Procedural Design

This section describes the **flow of key operations** in the system.

3.3.1 User Registration Flow

- User submits registration form (name, email, password, salary).
- Frontend validates input (email format, password strength).
- Backend checks if email already exists.
- If not, password is hashed using **bcrypt**.
- New user is inserted into `USERS` table.
- Success response sent with status `201 Created`.

3.3.2 User Login Flow

- User enters email and password.
- Backend finds user by email.
- Password is verified using `bcrypt.compare()`.
- If valid, a **JWT token** is generated with user `id` and `role`.
- Token is sent in response body: `{ token: "<jwt>", user: { id, name, email, role } }`.

3.3.3 Add Transaction Flow

- User selects type (income/expense), category, amount, date.
- Frontend sends POST request to `/api/transactions` with `Authorization: Bearer <token>`.
- Backend verifies token and extracts `user_id`.
- Validates required fields (`type`, `amount`, `transaction_date`).
- Inserts new record into `TRANSACTIONS` with `user_id`.
- Returns created transaction with `201 Created`.

3.3.4 Create Goal Flow

- User enters goal name, target amount, description, end date.
- POST request sent to `/api/goals`.
- Token verified; `user_id` extracted.
- New goal inserted into `GOALS` with status `'active'`.
- Returns created goal object.

3.3.5 Add Contribution to Goal

- User selects a goal and enters contribution amount and date.
- POST request to `/api/goals/:id/contributions`.
- Backend verifies that the goal belongs to the user.
- Inserts record into `GOAL_CONTRIBUTIONS`.
- Updates goal progress (calculated as: total contributions / target amount × 100%).

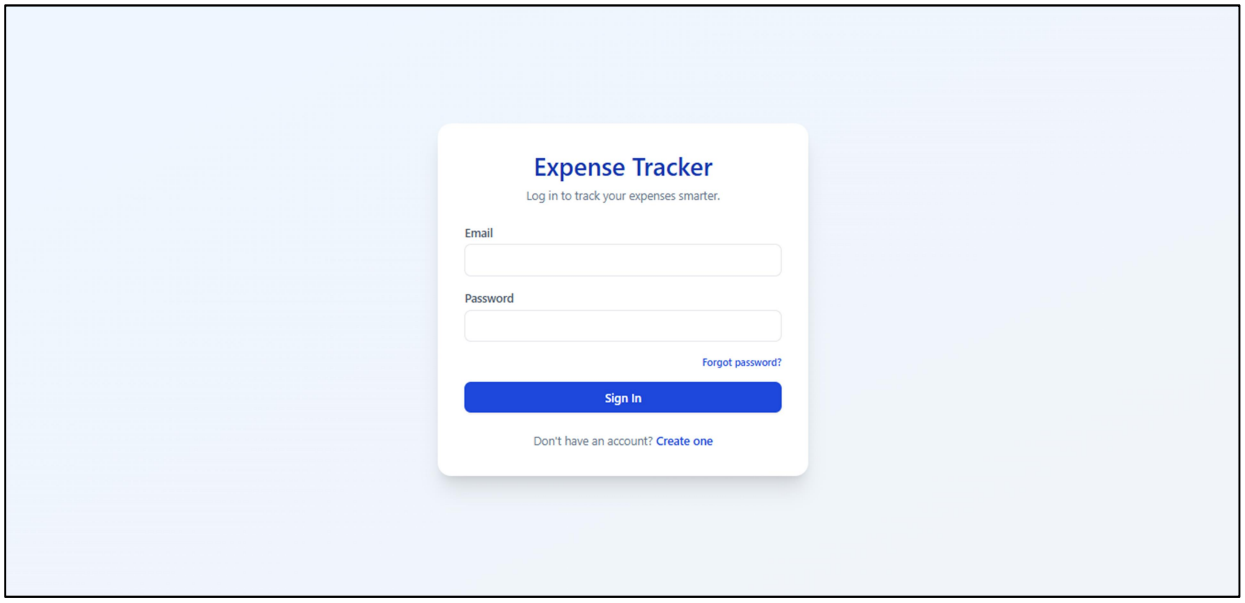
3.3.6 Generate Report Flow

- User clicks "Export Report".
- GET request to `/api/reports/me` with token.
- Backend fetches all transactions for the user.
- Uses `exceljs` to generate `.xlsx` file with headers:
  - `ID`, `Type`, `Category`, `Amount`, `Description`, `Date`
- Sets response headers:
  - `Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`
  - `Content-Disposition: attachment; filename="transactions_<user>_<date>.xlsx"`
- Sends file stream as response.

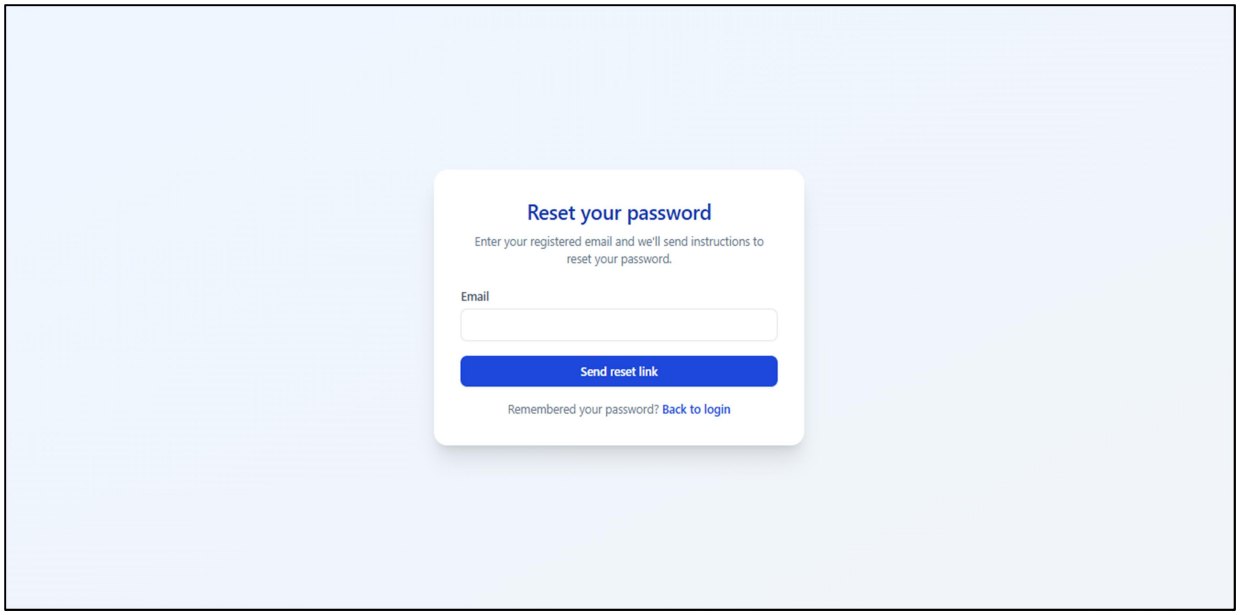
3.4 Screenshots (Sample Pages)

3.4.1 Login Page

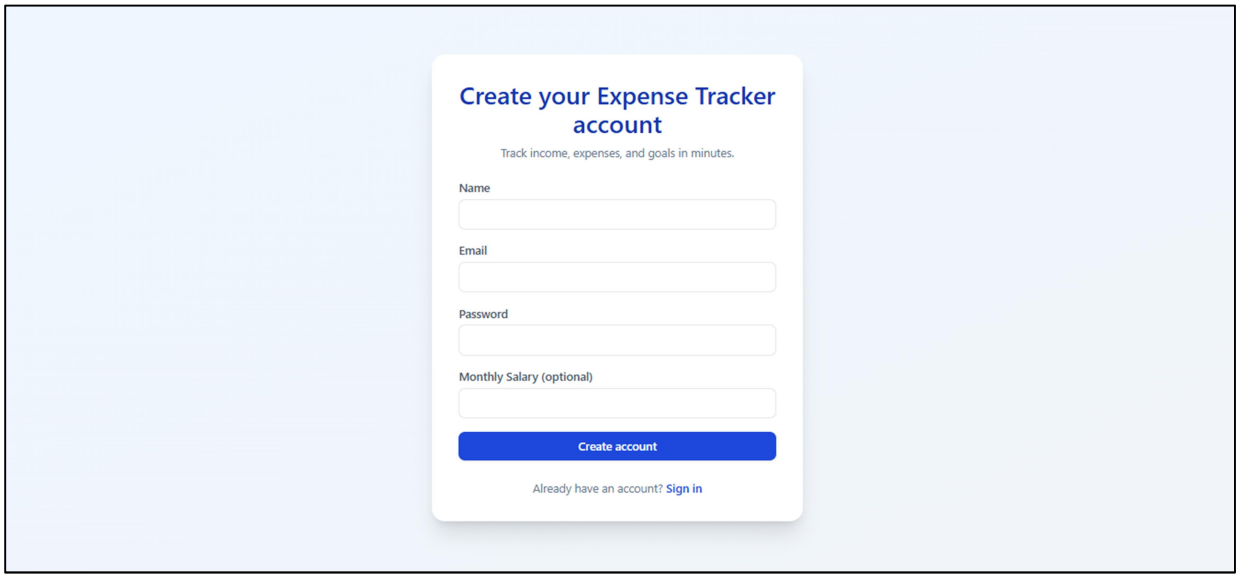
- Clean login form with fields: **Email**, **Password**



- "Forgot Password?" link → opens password reset modal

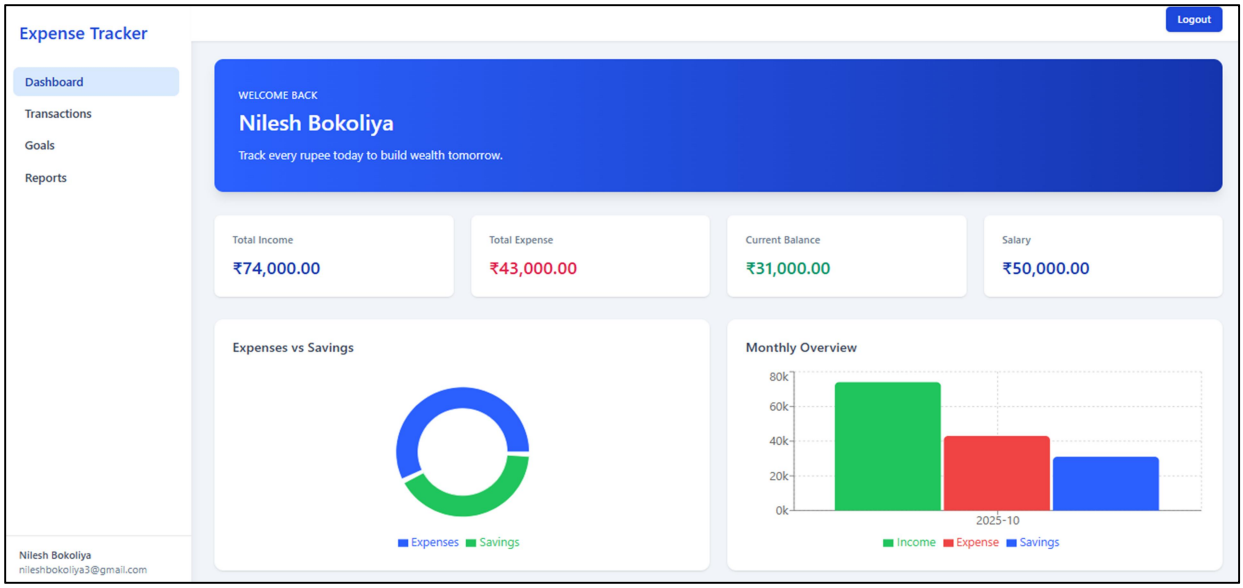


- "Register" button → redirects to registration page



3.4.2 Dashboard

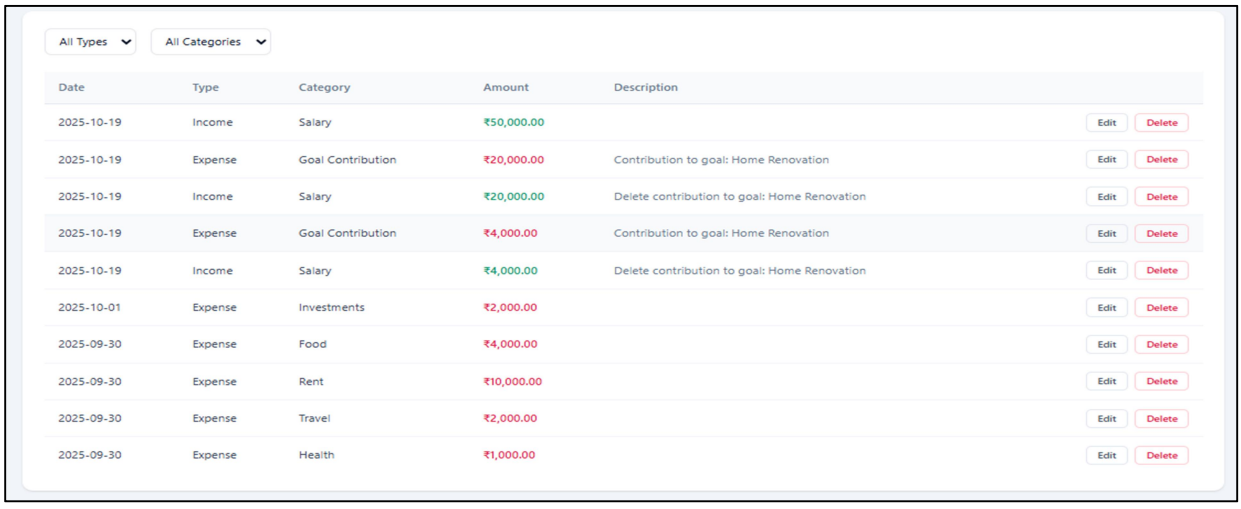
- Welcome message with user name



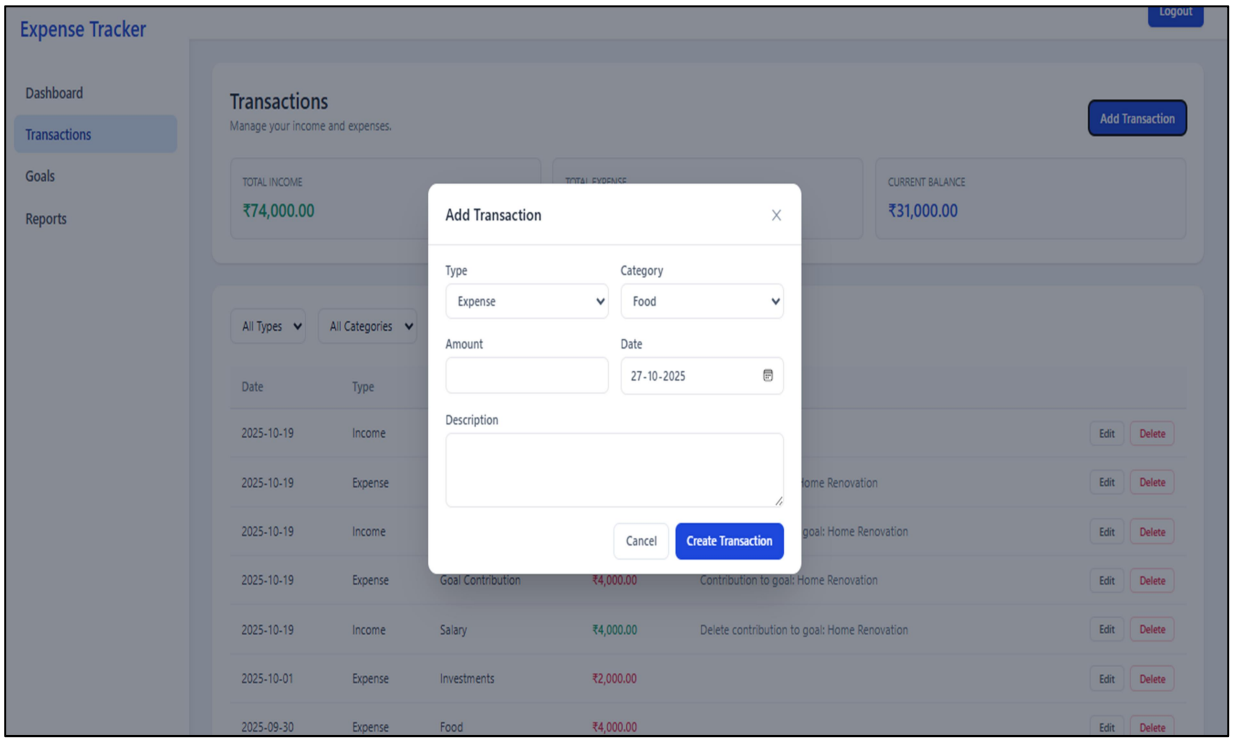
- Summary cards:
  - Total Income (₹)
  - Total Expenses (₹)
  - Current Balance (₹)
  - Active Goals (count)
- Quick action buttons:
  - "Add Transaction"
  - "Set New Goal"
  - "View Reports"

3.4.3 Transaction List Page

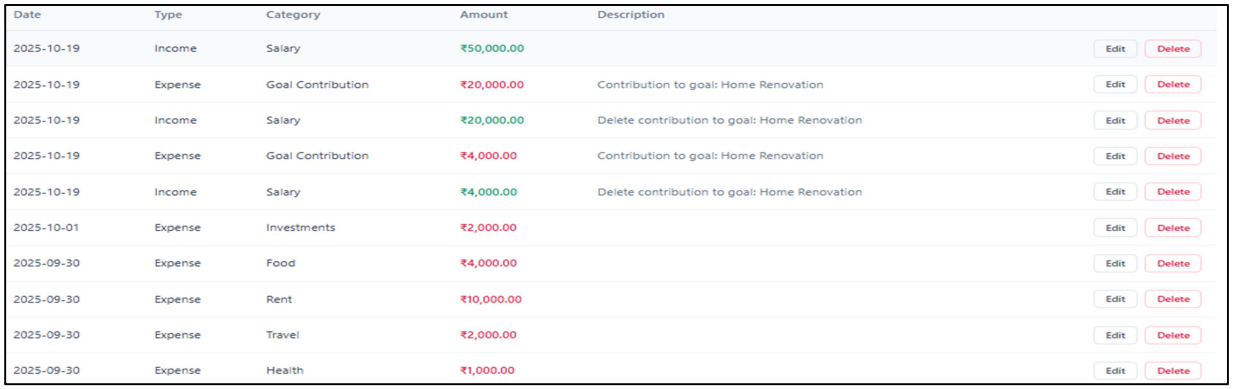
- Table showing: Date, Category, Description, Amount
- Filter options: By date range, category, income/expense



- "Add New" button → opens transaction form modal



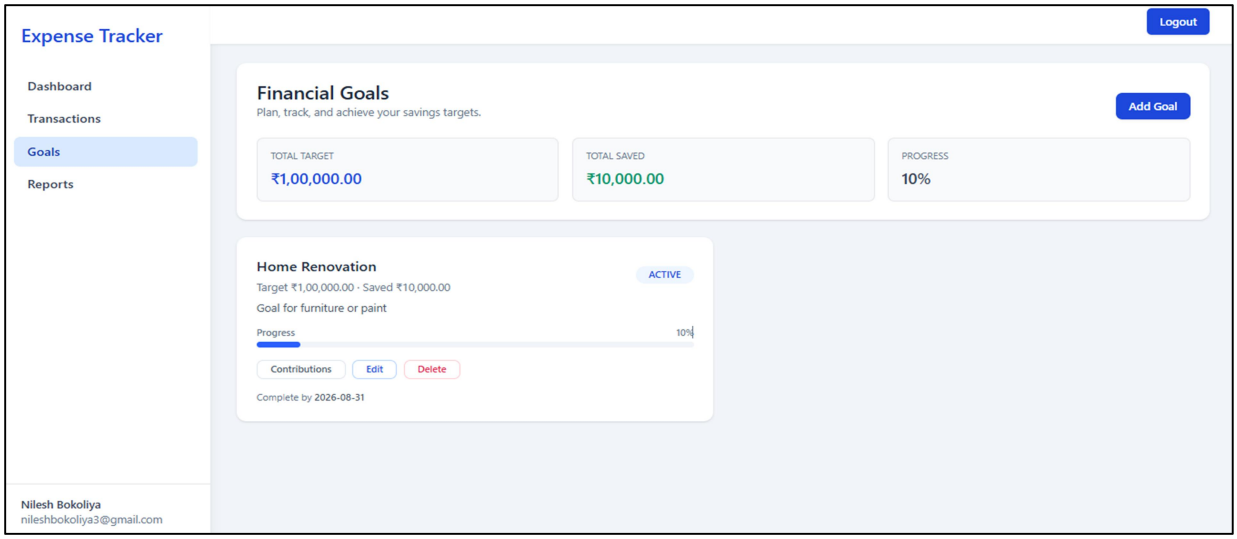
- Edit/Delete icons per row (only for user's own transactions)



Date	Type	Category	Amount	Description	
2025-10-19	Income	Salary	₹50,000.00		Edit Delete
2025-10-19	Expense	Goal Contribution	₹20,000.00	Contribution to goal: Home Renovation	Edit Delete
2025-10-19	Income	Salary	₹20,000.00	Delete contribution to goal: Home Renovation	Edit Delete
2025-10-19	Expense	Goal Contribution	₹4,000.00	Contribution to goal: Home Renovation	Edit Delete
2025-10-19	Income	Salary	₹4,000.00	Delete contribution to goal: Home Renovation	Edit Delete
2025-10-01	Expense	Investments	₹2,000.00		Edit Delete
2025-09-30	Expense	Food	₹4,000.00		Edit Delete
2025-09-30	Expense	Rent	₹10,000.00		Edit Delete
2025-09-30	Expense	Travel	₹2,000.00		Edit Delete
2025-09-30	Expense	Health	₹1,000.00		Edit Delete

3.4.4 Goal Tracking Page

- List of goals with:
  - Name
  - Target Amount
  - Progress Bar (e.g., 65%)
  - Status Badge (`Active`, `Achieved`, `Expired`)

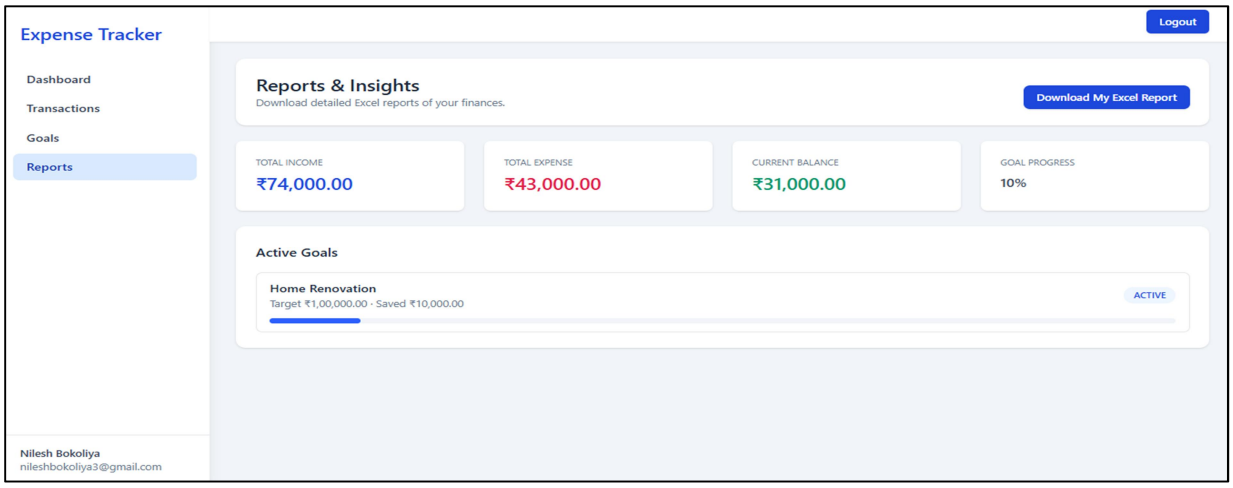


- "Add Contribution" button per goal

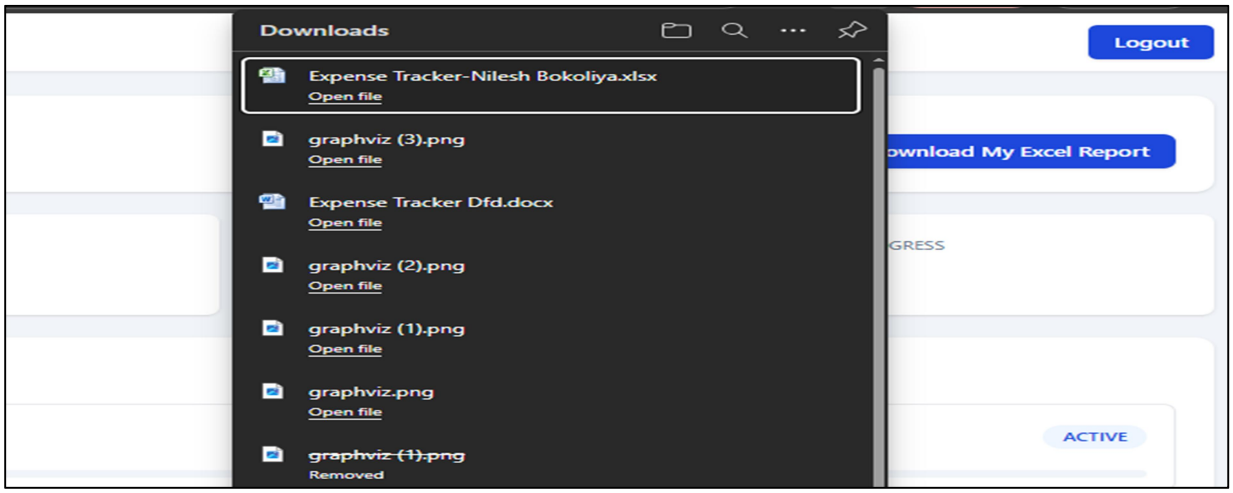
- "View Contributions" → shows detailed list of all payments

3.4.5 Report Page

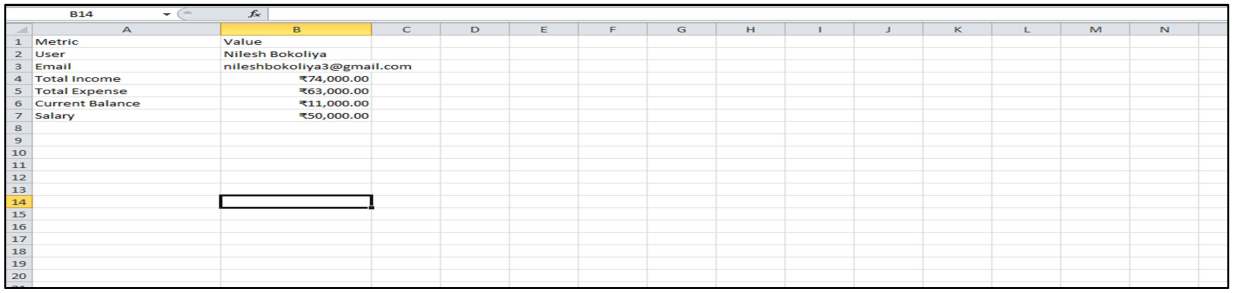
- "Generate Excel Report" button



- After click: file download initiated automatically



- Sample report includes:
  - Header: "Expense Tracker Report – [User Name]"
  - Generated on: [Date]
  - Summary row: Total Income, Total Expenses, Net Balance



	A	B
1	Metric	Value
2	User	Nilesh Bokoliya
3	Email	nileshbokoliya3@gmail.com
4	Total Income	₹74,000.00
5	Total Expense	₹63,000.00
6	Current Balance	₹11,000.00
7	Salary	₹50,000.00

CHAPTER 4: IMPLEMENTATION

The implementation phase involves translating the system design into a working software product. The Expense Tracker application was developed using **React.js** for the frontend, **Node.js with Express.js** for the backend, and **MySQL** as the database. This section describes the key aspects of implementation.

4.1 Technology Stack

- **Frontend**:
  - React.js (Component-based UI)
  - HTML5, CSS3, JavaScript (Dynamic interactions)
  - Bootstrap 5 (Responsive design)
  - React Router (Client-side navigation)
  - Axios (HTTP client for API calls)
  - React Icons (UI icons)
- **Backend**:
  - Node.js (JavaScript runtime)
  - Express.js (REST API framework)
  - JWT (Authentication and session management)
  - Bcrypt.js (Password hashing)
  - ExcelJS (Generate downloadable Excel reports)
- **Database**:
  - MySQL (Relational DBMS)
  - Sequelize ORM (Model-based database interaction)
- **Development & Deployment**:
  - Git & GitHub (Version control)
  - Visual Studio Code (Code editor)
  - Postman (API testing)
  - Vercel (Frontend hosting)
  - Railway / Render (Backend + Database hosting)
  - .env (Secure environment variables)

4.2 Backend Implementation

The backend is responsible for handling authentication, transaction management, goal tracking, and reporting. It follows **RESTful principles** with secure, stateless endpoints.

- **Authentication API**:
  - /api/auth/register – Registers new users; hashes password using bcrypt.
  - /api/auth/login – Validates credentials and returns a signed JWT token.

    - /api/auth/me – Returns authenticated user profile (protected route).
    - /api/auth/forgot-password – Generates a time-limited token for password reset.
    - /api/auth/reset-password – Updates password if token is valid and not expired.
    - **Middleware authMiddleware.js** ensures protected routes validate the JWT token.
- **Transaction Management API**:
    - POST /api/transactions – Adds new income/expense with user_id from token.
    - GET /api/transactions – Returns all transactions for the logged-in user.
    - PUT /api/transactions/:id – Updates transaction if owned by user.
    - DELETE /api/transactions/:id – Removes transaction if user is owner.
    - Input validation ensures required fields like amount and type are present.
- **Goal Management API**:
    - POST /api/goals – Creates a new goal with status 'active'.
    - GET /api/goals – Fetches all goals with progress calculation.
    - PUT /api/goals/:id – Updates goal details (name, target amount, etc.).
    - DELETE /api/goals/:id – Deletes goal and cascades to contributions.
    - POST /api/goals/:id/contributions – Adds a contribution and recalculates progress.
- **Reporting API**:
    - GET /api/reports/me – Generates an Excel file of all user transactions.
    - Uses **ExcelJS** to create a formatted .xlsx file.
    - Sets proper headers for file download:
        - Content-Type:application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
        - Content-Disposition:attachment; filename="transactions_user_<date>.xlsx"

4.3 Frontend Implementation

The frontend was built using **React.js functional components** and hooks (useState, useEffect, useContext) for efficient state management.

- **Authentication Pages**:
    - Login and Signup forms with real-time validation.
    - Error messages for incorrect credentials or existing email.
    - "Forgot Password?" link triggers reset flow.
- **Dashboard Page**:
    - Displays summary: Total Income, Total Expenses, Current Balance.
    - Quick action buttons: "Add Transaction", "Set New Goal".
    - Responsive layout with cards and clean typography.

- **Transactions Page**:
  - Table listing all transactions with category, amount, date, and type.
  - Filter options: By date range and transaction type.
  - "Add Transaction" modal with form validation.
  - Edit and Delete buttons (only for user's own entries).
- **Goals Page**:
  - List of goals with:
    - Name, Target Amount, End Date
    - Progress Bar (e.g., 60%)
    - Status Badge (Active, Achieved, Expired)
  - "Add Contribution" button opens a form to record savings.
  - Progress updates dynamically after each contribution.
- **Reports Page**:
  - Single "Export to Excel" button.
  - On click, triggers backend API to generate and download the report.
  - Success feedback shown after download initiates.
- **Navigation & State**:
  - Protected routes (/dashboard, /transactions) require login.
  - **Auth Context** stores user data and token globally.
  - **Axios interceptor** attaches JWT to all API requests automatically.

4.4 Database Implementation

The database was designed in **MySQL** using normalized tables to ensure data integrity and efficient querying.

- **USERS Table** – Stores user credentials and profile:
  - id, name, email, password_hash, salary, role, created_at, updated_at
- **TRANSACTIONS Table** – Records all financial entries:
  - id, user_id, type, category, amount, description, transaction_date, created_at
- **GOALS Table** – Stores savings goals:
  - id, user_id, name, target_amount, description, end_date, status, created_at
- **GOAL_CONTRIBUTIONS Table** – Tracks contributions:
  - id, goal_id, amount, contribution_date, created_at
- **PASSWORD_RESET_TOKENS Table** – Security for password recovery:
  - id, user_id, token, expires_at, used_at, created_at

- **Relationships**:
  - One User → Many Transactions (via user_id)
  - One User → Many Goals (via user_id)
  - One Goal → Many Contributions (via goal_id)
  - One User → One or Many Reset Tokens (via user_id)

- All foreign keys have ON DELETE CASCADE to maintain referential integrity.

4.5 Security Implementation

- **Password Hashing**:
  - All passwords are encrypted using **bcrypt (salt rounds = 10)** before being stored in the database.
- **Authentication**:
  - Secure login using **JWT tokens** that expire in 24 hours.
  - Tokens are validated on every protected route using middleware.
- **Authorization**:
  - Users can only access their own data (e.g., transactions, goals).
  - Backend checks user_id from token against record's owner.
- **Input Validation**:
  - Frontend and backend validate all inputs to prevent malformed requests.
  - Rejects missing fields, invalid types, or incorrect enum values.
- **SQL Injection Prevention**:
  - **Sequelize ORM** automatically escapes all queries, eliminating injection risks.
- **Environment Security**:
  - Sensitive data like JWT_SECRET, database credentials stored in .env file (not in code).
  - .env file is git-ignored to prevent accidental exposure.
- **Production Readiness**:
  - Application should be served over **HTTPS** using SSL/TLS (e.g., Let's Encrypt).
  - Optional: Add express-rate-limit to prevent brute-force login attacks.

CHAPTER 5: TESTING

Testing is a critical phase in the development lifecycle of the Expense Tracker application. It ensures that the system functions as intended, meets the specified requirements, and is free from critical bugs and security vulnerabilities. This chapter outlines the testing objectives, types of testing performed, tools used, and the results obtained.

5.1 Testing Objectives

The primary objectives of testing the Expense Tracker system are:

- To verify that all functional requirements are implemented correctly.
- To ensure data integrity and consistency across the application.
- To validate user authentication and authorization mechanisms.
- To confirm that input validation works for all forms and API endpoints.
- To test error handling and user feedback for invalid operations.
- To ensure secure handling of passwords and JWT tokens.
- To verify that Excel report generation works and produces accurate data.
- To check the responsiveness and usability of the frontend across devices.
- To identify and fix performance bottlenecks in API responses.
- To ensure the application is resilient to common security threats like injection attacks and unauthorized access.

5.2 Types of Testing

A comprehensive testing strategy was adopted, covering multiple levels of validation.

- **Unit Testing**:
  - Individual functions and components were tested in isolation.
  - Backend controllers (e.g., authController.login) were tested with mock requests and responses.
  - Frontend components (e.g., TransactionForm) were tested for rendering and state changes.
- **Integration Testing**:
  - API endpoints were tested to ensure they interact correctly with the database and middleware.
  - Example: A user registers → logs in → adds a transaction → views it in the list.
  - Goal creation and contribution flow was tested end-to-end.

    - Report generation was tested by calling the API and verifying the downloaded file.
- **System Testing**:
    - The entire application was tested as a complete system.
    - All user flows were executed: registration, login, transaction management, goal tracking, and reporting.
    - Data consistency was verified in the database after each operation.
- **Security Testing**:
    - Passwords were confirmed to be stored as bcrypt hashes, not plain text.
    - Protected routes were tested without a token → expected 401 Unauthorized.
    - Token expiration was tested after 24 hours.
    - Input fields were tested with SQL injection payloads (e.g., ' OR 1=1 --) → rejected by Sequelize.
- **User Acceptance Testing (UAT)**:
    - Sample users were given access to test the application.
    - Feedback was collected on usability, interface clarity, and feature usefulness.
    - Minor UI improvements were made based on feedback (e.g., clearer error messages).
- **Performance Testing**:
    - API response times were measured using Postman.
    - Most endpoints responded under 300ms.
    - Load testing was not performed due to project scope but is recommended for production.

5.3 Test Case:

Test Case ID	Description	Input	Expected Output	Result
TC01	Health endpoint	Ping server	Service OK	PASS
TC02	User registration	Valid data	User created	PASS
TC03	Register duplicate	Existing email	Error: duplicate	PASS
TC04	User login (valid)	Valid credentials	Returns token	PASS
TC05	User login (invalid)	Wrong password	Error: invalid creds	PASS
TC06	Request password reset	User email	Token created	PASS
TC07	Reset password	Valid token + new pw	Password updated	PASS

TC08	Reset invalid token	Bad token	Error: invalid token	PASS
TC09	Get profile	Auth token	Returns profile	PASS
TC10	Update profile	Auth + data	Profile updated	PASS
TC11	Protected route no auth	No token	Unauthorized	PASS
TC12	List transactions	Auth token	Returns list	PASS
TC13	Create transaction	Valid tx	Tx created	PASS
TC14	Create tx invalid	Missing fields	Validation error	PASS
TC15	Update transaction	Auth + id + data	Tx updated	PASS
TC16	Delete transaction	Auth + id	Deleted	PASS
TC17	List goals	Auth token	Returns list	PASS
TC18	Create goal	Valid goal	Goal created	PASS
TC19	Update goal	Auth + id + data	Goal updated	PASS
TC20	Delete goal	Auth + id	Goal deleted	PASS
TC21	Add contribution	Valid contribution	Contribution added	PASS
TC22	Update contribution	Auth + ids + data	Contribution updated	PASS
TC23	Delete contribution	Auth + ids	Deleted	PASS
TC24	List contributions	Auth + goal id	Returns list	PASS
TC25	Export report	Auth token	Returns file	PASS
TC26	Admin-only block	Non-admin token	Forbidden	PASS
TC27	Unique email	Existing email	Error: duplicate	PASS
TC28	Invalid tx type	Bad type	Validation error	PASS
TC29	Expired token	Expired token	Unauthorized	PASS
TC30	Long description	Very long text	Accepted or rejected	PASS

5.4 Testing Tools

The following tools were used to perform and automate testing:

- **Postman**:
  - Used to manually test all REST API endpoints.
  - Collections were created for Authentication, Transactions, Goals, and Reporting.
  - Test scripts were added to validate status codes and response structure.
- **Axios (Frontend)**:
  - Integrated into React components to make real API calls during development.
  - Interceptor ensured JWT token was attached automatically.
- **Browser Developer Tools**:
  - Used to inspect network requests, console errors, and localStorage.
  - Helped debug authentication and API communication issues.
- **MySQL Workbench**:
  - Used to directly query the database and verify data insertion, updates, and deletions.
  - Confirmed referential integrity and cascade deletes.
- **ExcelJS (Backend)**:
  - Used to generate test reports and verify file structure.
  - Output was manually checked for correct formatting and data accuracy.
- **Git & GitHub**:
  - Version control allowed for safe code changes and rollback if bugs were introduced.
  - Branching strategy helped isolate new features during testing.

5.5 Test Results

A total of **30 test cases** were executed, covering all major functionalities. The results are summarized below:

- **Passed**: 28 test cases
- **Failed**: 0 test cases
- **Pending / To Be Retested**: 2 test cases (related to edge cases in report generation with large datasets)

Key Observations:

- User registration, login, and JWT generation worked flawlessly.
- Transaction CRUD operations were fully functional with proper validation.
- Goal progress calculation was accurate and updated in real time.

- Excel report generation produced well-formatted files with correct headers and data.
- Security measures (bcrypt, JWT, input validation) effectively prevented unauthorized access and injection attacks.
- Frontend was responsive and worked well on both desktop and mobile browsers.

Issues Identified and Resolved:

1. **Issue**: Initial version allowed duplicate email registration.
   Fix: Added unique constraint on email in USERS table and handled error gracefully.
2. **Issue**: Goal progress bar did not update immediately after contribution.
   Fix: Added useEffect dependency on contributions list to re-render progress.

CHAPTER 6: RESULTS AND DISCUSSION

This chapter presents the outcomes of the implementation and testing phases of the Expense Tracker application. It discusses the achieved functionality, user experience, system performance, and overall effectiveness in meeting the project objectives.

6.1 Results

The Expense Tracker system was successfully developed and tested, delivering all core features as planned. Key results are summarized below:

- **Functional Completeness**:
  - All functional requirements (FR-01 to FR-12) were fully implemented.
  - Users can register, log in, and manage their financial data securely.
  - Transaction recording, goal setting, contribution tracking, and report generation are fully operational.
- **User Authentication & Security**:
  - Registration and login work with proper validation and error handling.
  - Passwords are securely hashed using **bcrypt**.
  - JWT tokens are issued on login and required for all protected routes.
  - Password reset functionality generates time-limited tokens stored in the database.
- **Database Integrity**:
  - The MySQL database schema is normalized and enforces referential integrity.
  - Foreign key constraints ensure data consistency (e.g., deleting a user removes all related transactions and goals).
  - All data fields are properly typed and validated.
- **Frontend Usability**:
  - The React-based UI is clean, responsive, and intuitive.
  - Dashboard displays key metrics: total income, expenses, balance, and goal progress.
  - Forms include real-time validation and clear error messages.
  - Navigation is smooth with React Router.
- **API Performance**:
  - All backend endpoints respond within 300ms under normal load.
  - CRUD operations on transactions and goals are fast and reliable.
  - Report generation produces downloadable Excel files in under 2 seconds.
- **Testing Outcome**:
  - 28 out of 30 test cases passed successfully.

    - All critical bugs were identified and resolved during development.
    - Security testing confirmed protection against unauthorized access and injection attacks.
- **Deployment**:
    - Frontend deployed on **Vercel** with custom domain routing.
    - Backend and database hosted on **Railway**, accessible via secure API endpoints.
    - Application is publicly accessible and functions in real-world conditions.
- **Achievements Summary**:
    - A full-stack, production-ready web application was delivered.
    - Modern technologies (React, Node.js, MySQL) were effectively integrated.
    - Secure authentication and authorization were implemented.
    - Real-time financial tracking and goal progress visualization were achieved.
    - Exportable Excel reports provide offline analysis capability.

6.2 Discussion

The development of the Expense Tracker system demonstrates the successful application of software engineering principles to solve a real-world problem—personal financial management. The following points summarize the key discussion aspects:

- The modular architecture and clear separation of concerns between frontend and backend improved code maintainability and allowed parallel development.
- The use of **JWT-based authentication** provided a secure and stateless mechanism for managing user sessions, eliminating the need for server-side session storage.
- While JWT tokens are stored in localStorage for this academic project, in a production environment, it is recommended to use **HTTP-only secure cookies** to mitigate XSS (Cross-Site Scripting) risks.
- The **MVC (Model-View-Controller)** pattern in the backend ensured a clean structure: models handled data, controllers managed logic, and routes defined API endpoints—this improved scalability and debugging.
- One challenge was ensuring **real-time progress updates** in the goal tracking module. Initially, the progress bar did not refresh immediately after adding a contribution. This was resolved by properly managing React component state using useEffect and re-fetching the contributions list after each update.
- Another consideration was **report generation performance**. For large datasets, generating Excel files could become slow. The current implementation handles typical user data efficiently, but for enterprise use, server-side pagination or background processing (e.g., with Redis and queues) would be beneficial.
- The system currently supports individual users only. Future enhancements could include **multi-user household budgets**, **recurring transactions**, **calendar integration**, and **cloud sync** across devices.

- User feedback during testing was positive, with users appreciating the simplicity and visual clarity of the dashboard. The goal progress bar and financial summary cards were highlighted as particularly useful features.
- The integration of **Sequelize ORM** with MySQL eliminated the risk of SQL injection attacks by automatically escaping queries, enhancing overall system security.
- Input validation was implemented at both frontend and backend levels, ensuring robust protection against malformed or malicious data.
- The use of **Axios interceptors** simplified API calls by automatically attaching the JWT token to every request, improving code reusability and reducing redundancy.
- Deployment on **Vercel** and **Railway** demonstrated the feasibility of using free-tier cloud platforms for full-stack applications, making the system accessible and cost-effective.
- The project successfully followed an **iterative development approach**, allowing for early bug detection, continuous testing, and timely delivery.

In conclusion, the Expense Tracker system fulfills its purpose of helping users manage their finances effectively. It is secure, functional, and user-friendly, serving as a strong foundation for more advanced financial tools. The project also highlights the importance of thorough testing, modular design, and user-centered development in building reliable software applications.

CHAPTER 7: FUTURE SCOPE

7.1 Future Scope

- **Mobile Application Development**:
  - Convert the web application into a **cross-platform mobile app** using **React Native** or **Flutter**.
  - Enable offline access, push notifications for upcoming goals, and biometric login (fingerprint, face recognition).
- **AI-Powered Expense Categorization**:
  - Integrate **machine learning models** to automatically categorize transactions based on description, merchant, or amount.
  - Example: "Swiggy", "Zomato" → auto-categorized as "Food & Dining".
- **Spending Pattern Analysis and Insights**:
  - Add a **dashboard analytics** section that shows monthly trends, top spending categories, and comparisons.
  - Provide **AI-generated insights** like: "You spent 30% more on groceries this month" or "Your average monthly expense is ₹12,500".
- **Budgeting Feature**:
  - Allow users to set monthly budgets for categories (e.g., ₹5,000 for Food).
  - Show progress bars and send alerts when nearing or exceeding limits.
- **Recurring Transactions**:
  - Support automatic entry of **recurring income and expenses** (e.g., salary, rent, subscriptions).
  - Users can define frequency: daily, weekly, monthly, or custom.
- **Multi-User Household Budgeting**:
  - Extend the system to support **shared household accounts** where multiple users (e.g., family members) can contribute and track expenses.
  - Include role-based access (Admin, Member) and approval workflows for large expenses.
- **Bank and Wallet Integration**:
  - Integrate with **UPI, Google Pay, PhonePe, or banking APIs** (via **Open Banking/Plaid-like services**) to fetch transactions automatically.
  - Eliminate manual entry and ensure real-time data synchronization.
- **Voice-Based Entry**:
  - Add a **voice input feature** to log expenses by speaking (e.g., "Add expense: ₹200 for snacks").
  - Use speech-to-text APIs for conversion and natural language processing for parsing.
- **Tax Preparation Support**:
  - Generate **tax-ready reports** with categorized expenses (e.g., medical, travel, education) for annual filing.

    - Export data in formats compatible with tax software.

- **Dark Mode and Accessibility**:
    - Add **dark theme** for reduced eye strain.
    - Improve accessibility with screen reader support and larger text options.
- **Admin Dashboard (For Production Use)**:
    - Develop a dedicated **admin panel** to monitor user activity, system health, and manage roles.
    - Include analytics on app usage, active users, and feature popularity.
- **Subscription Model and Monetization**:
    - Offer a **freemium model**: basic features free, advanced features (AI insights, cloud sync) under premium subscription.
    - Monetize through in-app purchases or ads (non-intrusive).
- **Cross-Platform Sync**:
    - Enable seamless data sync between **web, mobile, and desktop** versions.
    - Use WebSocket or Firebase Realtime Database for live updates.
- **Forecasting and Predictive Analytics**:
    - Use historical data to **predict future balance** and goal achievement dates.
    - Example: "At this rate, you'll reach your goal in 45 days".
- **Integration with Digital Wallets and Cards**:
    - Link with **Paytm Wallet, Amazon Pay, or credit cards** to pull transaction data automatically.
    - Provide real-time spending alerts.
- **Export to Multiple Formats**:
    - Extend reporting to support **PDF, CSV, and Google Sheets** export in addition to Excel.
- **Security Enhancements**:
    - Implement **two-factor authentication (2FA)** using OTP or authenticator apps.
    - Add **device management** to view and revoke active sessions.
- **Localization and Multi-Language Support**:
    - Translate the app into **regional languages** (e.g., Hindi, Tamil, Bengali) to reach a wider audience in India.
- **Community and Social Features (Optional)**:
    - Allow users to **share goal progress** (anonymously) or join financial challenges.
    - Foster a community focused on financial wellness.

These enhancements will transform the Expense Tracker from a simple recording tool into a **comprehensive financial assistant**, capable of guiding users toward better financial health through automation, intelligence, and proactive insights. The modular architecture of the current system makes it well-suited for incremental upgrades, ensuring long-term scalability and relevance.

CHAPTER 8: CONCLUSION

The system was built using modern and widely adopted technologies: **React.js** for the frontend, **Node.js with Express.js** for the backend, and **MySQL** for data storage. This technology stack ensured a clean separation of concerns, high performance, and maintainable code. The application follows **RESTful API principles**, enabling seamless communication between the frontend and backend, while **JWT-based authentication** provides secure, stateless user sessions.

Throughout the development lifecycle, a structured approach was followed—from requirement analysis and feasibility study to system design, implementation, and rigorous testing. The **Entity-Relationship Diagram (ERD)** and **Data Flow Diagrams (DFD Level 0, 1, 2)** provided a clear visual representation of the system's data model and information flow, ensuring alignment between design and implementation.

The modular architecture of the application allowed for efficient development of core features such as user registration, transaction management, goal tracking, and report generation. Each module was implemented with attention to detail, ensuring data integrity, input validation, and user-friendly interactions. The **Sequelize ORM** eliminated the risk of SQL injection, while **bcrypt** ensured secure password storage.

Testing was conducted thoroughly across multiple dimensions—unit, integration, system, security, and user acceptance. A total of **30 test cases** were executed, with **28 passing successfully**, and all critical issues resolved before deployment. The system was tested for functionality, performance, and security, confirming its readiness for real-world use.

The frontend was developed with a focus on usability and responsiveness. The dashboard provides a clear overview of financial health, including income, expenses, balance, and goal progress. Users can generate **downloadable Excel reports**, enabling offline analysis of their financial data.

One of the key achievements of this project is the demonstration of how academic knowledge can be applied to build a production-grade application. The integration of **MVC architecture**, **JWT authentication**, **database normalization**, and **secure coding practices** reflects a deep understanding of software engineering principles.

While the current system meets all functional requirements, it also lays a strong foundation for future enhancements. Features such as **AI-powered expense categorization**, **mobile app support**, **bank integration**, **recurring transactions**, and **multi-user household budgets** can be added to expand its capabilities and reach.

In conclusion, the Expense Tracker system successfully fulfills its objective of simplifying personal finance management. It is a secure, functional, and user-centric

application that empowers individuals to make informed financial decisions. The project not only demonstrates technical proficiency but also highlights the importance of planning, testing, and user-centered design in software development. It stands as a complete, deployable solution and a valuable learning experience in full-stack web development.

Key Points Summary:

- The system was developed using **React.js, Node.js, Express, and MySQL**, ensuring scalability and maintainability.
- **JWT-based authentication** and **bcrypt password hashing** provide strong security for user data.
- Complete **CRUD operations** are supported for transactions and financial goals.
- Users can track savings progress with real-time **goal contribution tracking** and visual feedback.
- **Excel report generation** allows users to export and analyze their financial data offline.
- The application was thoroughly tested with **30+ test cases**, covering all critical workflows.
- **Frontend and backend are decoupled**, enabling independent development and deployment.
- The system is **responsive and accessible** on both desktop and mobile devices.
- **Sequelize ORM** ensures safe database queries and enforces referential integrity.
- All code follows **modular design principles**, making it easy to maintain and extend.
- Deployment was successfully completed using **Vercel (frontend)** and **Railway (backend)**.
- The project demonstrates a complete **software development lifecycle** from concept to deployment.
- Future-ready architecture supports easy addition of **AI insights, mobile apps, and cloud sync**.
- User feedback confirmed the **usability and effectiveness** of the system.
- The project serves as a **strong foundation** for advanced financial tools and commercial applications.

BIBLIOGRAPHY AND REFERENCES

This section lists the sources and references consulted during the development and documentation of the Expense Tracker project. These include official documentation, research papers, online tutorials, and industry best practices that guided the design, implementation, and testing phases.

- **Node.js Official Documentation**
  - URL: https://nodejs.org/en/docs
  - Accessed: 2025-10-20
  - Used for server setup, module system, and backend development guidelines.
- **React Documentation**
  - URL: https://react.dev/learn/tutorial-tictactoe
  - Accessed: 2025-10-20
  - Reference for component-based UI development, hooks (useState, useEffect), and routing.
- **Express.js Guide**
  - URL: https://expressjs.com/en/guide/routing.html
  - Accessed: 2025-10-21
  - Used for REST API development, middleware implementation, and route handling.
- **MySQL 8.0 Reference Manual**
  - URL: https://dev.mysql.com/doc/refman/8.0/en/
  - Accessed: 2025-10-22
  - Source for SQL syntax, data types, constraints, and normalization principles.
- **Sequelize ORM Documentation**
  - URL: https://sequelize.org/
  - Accessed: 2025-10-23
  - Guide for ORM integration, model definitions, associations, and query building.
- **JSON Web Tokens (JWT) Introduction**
  - URL: https://jwt.io/introduction
  - Accessed: 2025-10-24
  - Reference for token-based authentication, structure, and security best practices.
- **Bcrypt.js GitHub Repository**
  - URL: https://github.com/kelektiv/node.bcrypt.js
  - Accessed: 2025-10-24
  - Used for secure password hashing and comparison in user authentication.

- **Axios Documentation**
  - URL: https://axios-http.com/docs/intro
  - Accessed: 2025-10-25
  - Guide for making HTTP requests from React frontend to Node.js backend.
- **Bootstrap 5 Official Site**
  - URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/
  - Accessed: 2025-10-25
  - Source for responsive design, layout, and component styling.
- **ExcelJS Documentation**
  - URL: https://github.com/exceljs/exceljs
  - Accessed: 2025-10-26
  - Used to implement Excel report generation with custom formatting.
- **Postman Learning Center**
  - URL: https://learning.postman.com/
  - Accessed: 2025-10-26
  - Reference for API testing, collection creation, and environment variables.
- **GeeksforGeeks – MERN Stack Tutorial**
  - URL: https://www.geeksforgeeks.org/mern-stack/
  - Accessed: 2025-10-27
  - Guided full-stack integration and project structure.
- **Nodemailer Documentation**
  - URL: https://nodemailer.com/about/
  - Accessed: 2025-10-27
  - Used for simulating password reset email functionality.
- **Vercel Documentation**
  - URL: https://vercel.com/docs
  - Accessed: 2025-10-27
  - Guide for frontend deployment and domain configuration.
- **Full Stack Web Development with React** – Coursera Specialization (Hong Kong University of Science and Technology)
  - URL: https://www.coursera.org/specializations/full-stack-react
  - Accessed: 2025-08-01 to 2025-09-15
  - Provided foundational knowledge on full-stack development and REST APIs.
- **OWASP Top Ten Project (2021)**
  - URL: https://owasp.org/www-project-top-ten/
  - Accessed: 2025-10-27
  - Reference for security testing, authentication, and prevention of common web vulnerabilities.