# A Project Report On

## Expense_Tracker

**Prepared By:**
Hetshri Ved (24CEOUS916)

B. Tech CE, Semester V
Subject: Advanced Technologies (23CE520)

**Guided By:**

Prof. Siddharth P. Shah
Assistant Professor

Department of Computer Engineering
Faculty of Technology
Dharmsinh Desai University
College Road,
Nadiad-387001 Gujarat,
INDIA

# CERTIFICATE

This is to certify that the practical/term work carried out in the subject of Advanced Technologies (23CE520) and recorded in this report is the bonafide work of

**Hetshri Ved, ID No: 24CEOUS916**

of B.Tech semester V in the branch of Computer Engineering during the academic year 2025-2026.

Prof. Siddharth Shah
Assistant Professor

Dr. C.K.Bhensdadia
Head of Department

# Table of Contents

# Chapter 1: Introduction

## 1.1 Problem Definition

Managing personal finances is a critical skill, yet many individuals struggle to track their monthly expenses effectively. Traditional methods like spreadsheets and paper-based records are time-consuming and lack real-time insights. The challenge addressed by this project is the absence of a simple, secure, and visual expense tracking solution that allows users to record, categorize, delete, and analyze their spending patterns in Indian Rupees. The key missing element is an integrated platform that combines user authentication, expense CRUD operations, and visual analytics through interactive pie charts.

## 1.2 Project Objective(s)

The primary objectives of the Expense_Tracker platform are:

- To create a secure, full-stack application using the MERN (MongoDB, Express, React, Node.js) stack
- To enable users to register, login, and manage their personal accounts securely
- To implement comprehensive expense management including adding, viewing, and deleting expense records
- To display monthly expenses in a visual pie chart format for better financial insights
- To ensure all financial data is handled in Indian Rupees (₹)
- To secure the application using JSON Web Tokens (JWT) for user authentication and authorization
- To provide a multi-page interface for better user experience and navigation

## 1.3 Scope of the Project

The project scope encompasses the complete development of a full-stack expense tracking application with the following features:

**In Scope:**

- User registration and authentication system
- Secure login with JWT-based session management
- Add new expense entries with details (title, amount, category, date)
- Delete existing expense records
- View all expenses for the current month
- Visual representation of expenses through pie charts
- Multi-page navigation using React Router

- Currency handling exclusively in Indian Rupees (₹)
- Initial total expense value set to 0

**Out of Scope:**

- Multi-currency support
- Budget planning and forecasting features
- Expense sharing or multi-user collaboration
- Mobile application development
- Export to PDF or Excel functionality
- Advanced analytics beyond pie charts
- Recurring expense automation

# Chapter 2: System Requirements

## 2.1 Hardware Requirements

The Expense_Tracker platform is designed to be cloud-native and operates on standard consumer hardware.

### 2.1.1 Minimum and Recommended System Configuration (Client Side)

| Component | Minimum | Recommended |
|---|---|---|
| Processor | Dual-Core (2.0 GHz) | Quad-Core or higher |
| RAM | 4 GB | 8 GB or more |
| Storage | 200 MB Free Space | 1 GB Free Space |
| Network | 2 Mbps Down/Up | 10 Mbps Down/Up |

Table 1: Client-Side Hardware Requirements

## 2.2 Software Requirements

### 2.2.1 Core Technologies

- Operating System (OS): Windows, macOS, or Linux (Ubuntu/Debian)
- Backend Runtime: Node.js (Version 18 or higher)
- Database: MongoDB (using MongoDB Atlas cloud service)
- Frontend Library: React (Version 18 or higher)
- Backend Framework: Express (Version 4 or higher)

### 2.2.2 Other Libraries and Tools

- Mongoose: ODM (Object Data Modeling) library for MongoDB and Node.js
- JWT (jsonwebtoken): Library for creating and verifying access tokens
- bcrypt/bcryptjs: Library for hashing user passwords
- React Router DOM: For multi-page navigation in React

- Chart.js & react-chartjs-2: For rendering interactive pie charts
- Axios: HTTP client for making API requests from React to Express
- Postman / Thunder Client: Used for API testing and validation
- VS Code (Visual Studio Code): Primary IDE for development
- Dotenv: For managing environment variables
- CORS: Cross-Origin Resource Sharing middleware for Express

# Chapter 3: System Design

## 3.1 Architecture Diagram (Client-Server Flow)

The Expense_Tracker application follows a decoupled MERN Stack architecture with clear separation of concerns:

1. Client (React): The user interface handles all rendering and user interactions across multiple pages (Signup, Login, Dashboard, Expenses)
2. API Calls: The Client communicates with the Backend via RESTful API calls for authentication and expense operations
3. Authentication Flow:
    - User registers/logs in on the Client
    - Backend verifies credentials and issues a JWT
    - JWT is stored in browser localStorage or cookies
    - All subsequent requests include JWT in Authorization header
4. Database Operations: Express backend communicates with MongoDB Atlas to perform CRUD operations on User and Expense collections
5. Data Visualization: Expense data is fetched from the backend and rendered as pie charts using Chart.js on the frontend

## 3.2 Database Schema

The database utilizes MongoDB Atlas to host two primary collections: User and Expense.

### User Collection

Stores all user account and authentication data.

- _id: ObjectId (Primary Key)
- username: String (Required)
- email: String (Unique, Required)
- password: String (Hashed, Required)
- createdAt: Timestamp
- updatedAt: Timestamp

### Expense Collection

Manages all expense records linked to individual users.

- _id: ObjectId (Primary Key)
- user: User ObjectId (Reference to the user who created the expense)
- title: String (Required - description of expense)
- amount: Number (Required - amount in Rupees)
- category: String (Required - expense category like Food, Transport, etc.)
- date: Date (Required - date of expense)
- createdAt: Timestamp
- updatedAt: Timestamp

# Relationships

- One User can have Many Expenses (One-to-Many relationship)
- Each Expense belongs to exactly One User (enforced by user field reference)

# Chapter 4: Implementation

## 4.1 Frontend Development (React)

The React frontend handles all visual and user-side logic with a multi-page architecture.

### 4.1.1 Components, Routing, and State Management

**Components:**
Functional components are used for modularity. Key components include:

- Signup.js - User registration page
- Login.js - User login page
- Dashboard.js - Main dashboard showing expense summary and pie chart
- Expenses.js - Page for viewing all expenses and managing them
- ExpenseForm.js - Component for adding new expenses
- ExpenseList.js - Component for displaying expense list with delete functionality
- PieChart.js - Component for rendering monthly expense visualization

**Routing:**
react-router-dom is used to manage navigation between pages. Protected routes are implemented to restrict access based on the presence of a valid JWT.

**State Management:**
React hooks (useState, useEffect, useContext) are used to manage global application state, including the authenticated user object, JWT token, and expense data.

## 4.2 Backend Development (Express)

The Express server acts as the central API gateway, handling database interaction and secure token issuance.

### 4.2.1 Express API Routes

Key routes include:

**Auth APIs:**

- POST /api/auth/signup - User Registration
- POST /api/auth/login - User Login
- POST /api/auth/logout - Logout
- GET /api/auth/me - Get logged-in user (protected)

**Expense APIs:**

- POST /api/expenses - Add new expense (protected)
- GET /api/expenses - Get all expenses for current month (protected)
- DELETE /api/expenses/:id - Delete expense by ID (protected)
- GET /api/expenses/total - Get total expenses for current month (protected)

### 4.2.2 Middleware and Authentication

**Authentication Middleware:**
A custom middleware function verifies the JWT sent in the Authorization header for all protected routes.

**CORS Middleware:**
Cross-Origin Resource Sharing is enabled to allow frontend-backend communication.

## 4.3 Database (MongoDB Atlas)

### 4.3.1 MongoDB Schema and Collections

Mongoose is used to enforce schema integrity. The primary function of the database is to maintain user accounts and expense records.

## 4.4 Integration

The core integration involves securing the REST API with JWT and connecting React frontend with Express backend:

1. The React Client calls the Express /api/auth/login endpoint with credentials
2. The Express server validates credentials and returns the JWT token
3. The React Client stores the JWT in localStorage
4. For all subsequent API calls (add/view/delete expenses), the React Client includes the JWT in the Authorization header
5. Express backend verifies JWT using middleware before processing protected routes
6. MongoDB Atlas stores all persistent data and returns query results to Express
7. Express sends formatted JSON responses back to React
8. React renders the data in UI components including pie charts

**Initial State Management:**
The total expense value is initialized to 0 when a user first registers. As expenses are added, the total is calculated dynamically by summing all expense amounts for the current month.

**Currency Handling:**
All amounts are stored as numbers in the database and displayed with the Rupee symbol (₹) in the frontend.

# Chapter 5: Results and Output Screenshots

## 5.1 Screenshots of UI Pages

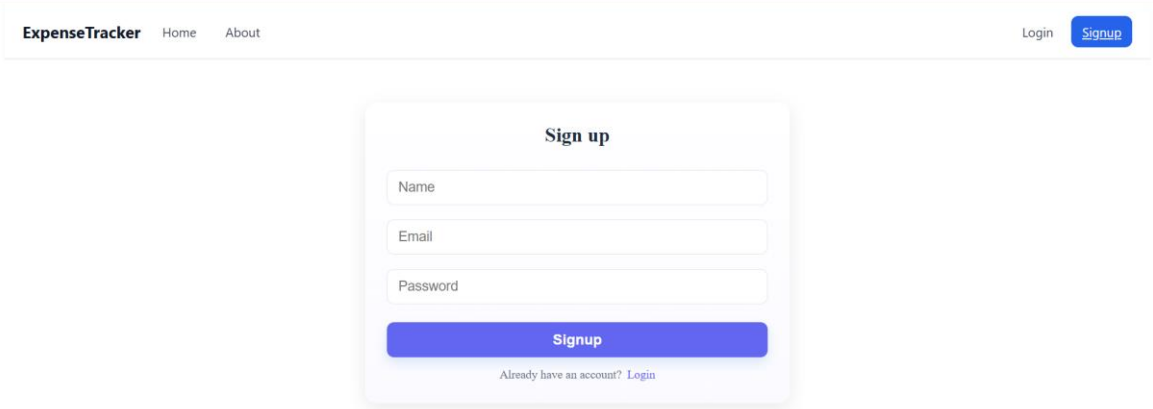
Screenshots demonstrate the key stages of the application workflow:

- Login and Registration Screen: Confirms the user can successfully access the system and receive a JWT
- Dashboard: Shows the monthly expense summary with a pie chart visualization of expenses by category
- Add Expense Page: Demonstrates the form for adding new expenses with title, amount (in Rupees), category, and date
- Expense List: Shows all expenses for the current month with delete functionality
- Empty State: Shows initial state with total expenses at ₹0

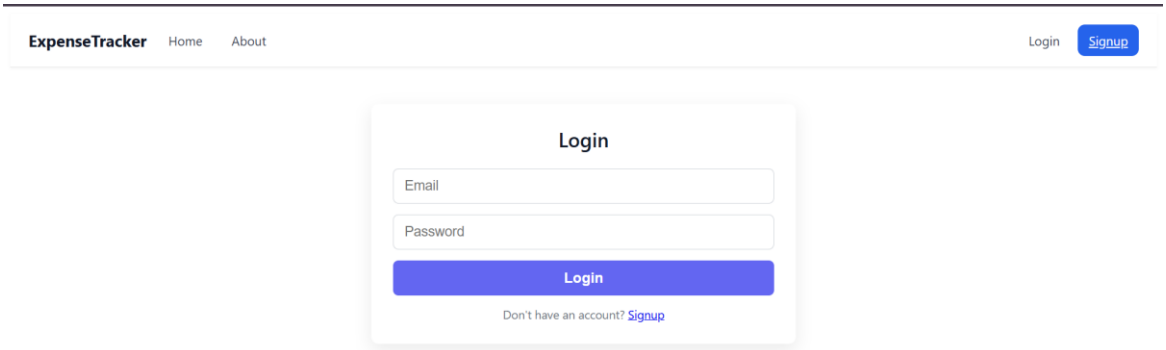
**Note:** Actual screenshots would be inserted here showing the Signup page, Login page, Dashboard with pie chart, Add Expense form, and Expense list with delete buttons. Each expense should clearly show the ₹ symbol for amounts.

**Expected UI Flow:**

**Signup Page:**

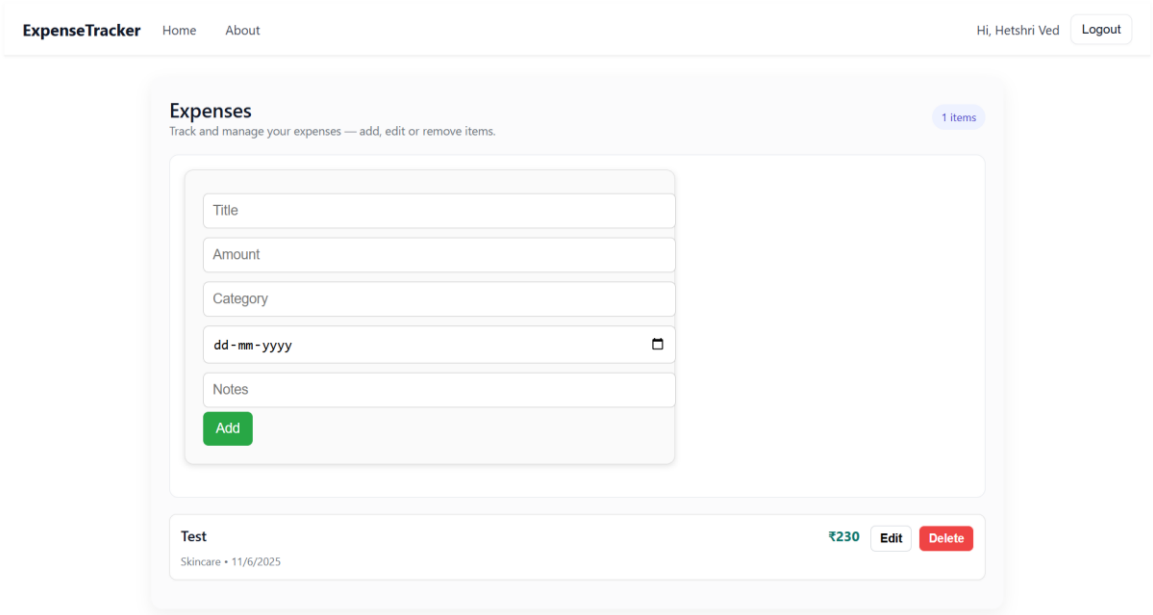



**Login Page:**

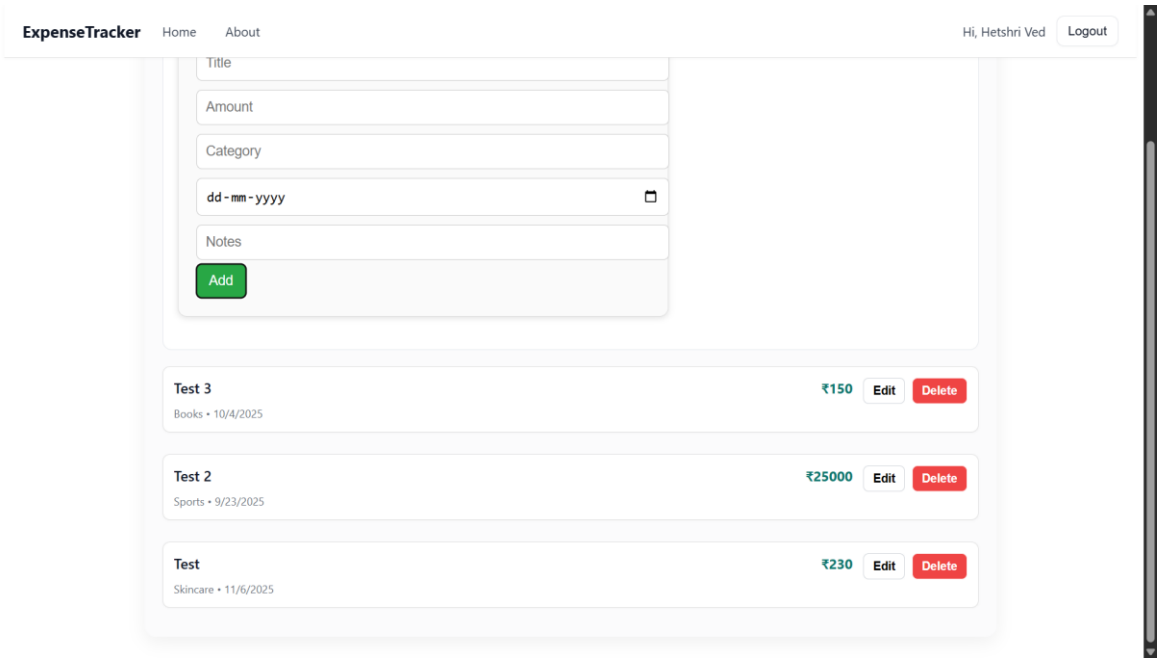



## Dashboard:


## Add Expense Page:




## Expenses List Page:


ExpenseTracker
Home
About
Hi, Hetshri Ved
Logout
Title
Amount
Category
dd-mm-yyyy
Notes
Add
Test 3
₹150
Edit
Delete
Books • 10/4/2025
Test 2
₹25000
Edit
Delete
Sports • 9/23/2025
Test
₹230
Edit
Delete
Skincare • 11/6/2025

# Chapter 6: Testing

## 6.1 Unit Testing (Frontend and Backend)

**Backend Unit Testing:**

Using Jest framework, unit tests were performed on core backend logic, including:

- User registration with password hashing reliability
- Login functionality and JWT generation
- JWT verification middleware for protected routes
- Expense creation with user association
- Expense deletion with ownership validation
- Monthly expense filtering logic

**Frontend Unit Testing:**

React Testing Library was used to verify component rendering and user interaction logic:

- Form submissions (signup, login, add expense)
- Button click events
- State changes after API calls
- Conditional rendering based on authentication status
- Pie chart rendering with expense data

## 6.2 API Testing (Postman and Thunder Client)

Postman was extensively used for functional and integration testing of the REST API:

**Authentication Testing:**

- Verified successful user registration with valid credentials
- Confirmed login returns valid JWT token
- Tested that protected routes reject requests without JWT
- Validated token expiration handling

**Expense CRUD Testing:**

- Tested POST /api/expenses endpoint creates expense correctly

- Verified GET /api/expenses returns only current month's expenses
- Confirmed DELETE /api/expenses/:id removes expense from database
- Tested that users can only delete their own expenses
- Validated amount field accepts decimal values for Rupees

**Edge Case Testing:**

- Invalid email format during registration
- Missing required fields in expense creation
- Negative expense amounts
- Invalid date formats
- Attempting to delete non-existent expenses

# Chapter 7: Limitations

## 7.0.1 Current Shortcomings of the Project

- Single Currency: The platform supports only Indian Rupees (₹) and lacks multi-currency functionality
- Basic Analytics: Only pie charts are provided; no trend analysis, forecasting, or comparative monthly reports
- No Budget Limits: Users cannot set monthly budget limits or receive overspending alerts
- Manual Entry Only: No support for receipt scanning or automatic expense import from bank statements
- No Data Export: Cannot export expense data to CSV, Excel, or PDF formats

## 7.0.2 Technical or Resource Constraints

- Development Time: Due to time constraints, advanced features like recurring expenses and budget planning could not be implemented
- Single User Focus: No multi-user or family expense sharing capabilities
- Browser Storage: JWT tokens stored in localStorage may be vulnerable to XSS attacks
- Limited Validation: Basic input validation exists but could be more comprehensive
- No Mobile Optimization: While responsive, the application is not optimized for mobile devices
- Dependency on Internet: Requires constant internet connection; no offline mode available

# Chapter 8: Future Enhancements

## 8.0.1 Features to be Added in Future Versions

- Multi-Month View: Ability to view and compare expenses across different months and years
- Budget Management: Set monthly budget limits with alerts when approaching or exceeding limits
- Custom Categories: Enable users to create and manage their own expense categories
- Advanced Analytics: Line charts for trends, bar charts for comparisons, monthly reports
- Recurring Expenses: Automatic addition of recurring monthly expenses (rent, subscriptions)
- Data Export: Export expense data to CSV, Excel, or PDF formats
- Receipt Upload: Attach receipt images to expense entries
- Multi-Currency Support: Handle expenses in different currencies with conversion rates
- Search and Filter: Advanced search and filtering options for expense history
- Email Notifications: Monthly expense summary reports sent via email
- Dark Mode: Theme toggle for better user experience

## 8.0.2 Scalability Improvements

- Cloud Deployment: Deploy backend to AWS, Azure, or Heroku for production use
- Database Optimization: Implement indexing on frequently queried fields (user, date)
- Caching Layer: Use Redis for caching frequently accessed expense data
- API Rate Limiting: Implement rate limiting to prevent abuse
- Microservices Architecture: Separate authentication and expense services
- Load Balancing: Distribute traffic across multiple server instances
- CDN Integration: Serve static assets through CDN for faster loading
- Mobile Application: Develop React Native version for iOS and Android
- Progressive Web App: Convert to PWA for offline functionality

- Enhanced Security: Implement refresh tokens, HTTPOnly cookies, and CSRF protection

# Chapter 9: Conclusion

## 9.1 Summary of Learnings

This project provided invaluable experience in developing a full-stack, production-ready financial management application. Key learnings include mastering the MERN stack integration, implementing secure authentication using JWT, designing RESTful APIs, and creating interactive data visualizations using Chart.js. The project successfully demonstrates the capability to create a complete expense tracking system with user authentication, CRUD operations, and visual analytics.

The development process reinforced the importance of proper project structure, separation of concerns between frontend and backend, and the significance of security in handling user data and financial information. Working with MongoDB Atlas highlighted the benefits of cloud-based database solutions, while implementing protected routes emphasized the critical role of middleware in Express applications.

## 9.2 Experience Gained in Working with MERN Stack

The development process significantly enhanced proficiency in:

- Full-stack data flow from React state management through Express routing to MongoDB persistence
- Developing and consuming robust RESTful APIs with proper error handling
- Implementing secure, stateless authentication using JWT tokens
- Managing asynchronous operations in both frontend and backend
- Working with Mongoose ODM for schema validation and database operations
- Creating multi-page applications using React Router with protected routes
- Integrating third-party libraries like Chart.js for data visualization
- Handling currency formatting and date manipulation in JavaScript
- Environment variable management for sensitive configuration
- API testing methodologies using Postman
- Git version control and project organization
- Debugging full-stack applications using browser DevTools and Node.js debugging

The project also provided practical experience in making architectural decisions, such as choosing between different state management approaches, determining optimal database schema design, and deciding how to structure API endpoints for maximum efficiency and security.

# Chapter 10: References


- Node.js Documentation: https://nodejs.org/
- Express.js Documentation: https://expressjs.com/
- React Documentation: https://react.dev/
- MongoDB and Mongoose Documentation: https://mongoosejs.com/
- Chart.js Documentation: https://www.chartjs.org/
- React Chart.js 2 Documentation: https://react-chartjs-2.js.org/
- JWT Implementation Guide: https://jwt.io/
- React Router Documentation: https://reactrouter.com/
- Axios Documentation: https://axios-http.com/
- bcrypt.js Documentation: https://www.npmjs.com/package/bcryptjs
- MDN Web Docs - JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Stack Overflow - MERN Stack Questions: https://stackoverflow.com/