Expense Tracker App Development Plan

This document outlines a step-by-step plan to build the Expense Tracker application. The project will be developed using modern technology stacks and will be built to fulfill the specified requirements for input, storage, and display of transaction data, along with visual reporting features and a simple, interactive UI.

# 1. Technology Stack

- Frontend (UI):   
 - React or Vue.js for building a responsive and interactive single-page application (SPA).  
 - CSS (TailwindCSS or Bootstrap) for a clean, minimalist design.  
 - Chart.js or Recharts for creating pie charts and other visual reports.  
   
- Backend:  
 - Node.js with Express.js for server-side logic.  
 - MongoDB as the database for storing transaction data, dropdown values, and settings.  
  
- Other Tools:  
 - MongoDB Atlas for cloud database hosting, or a local MongoDB instance.  
 - Git/GitHub for version control and collaboration.  
 - Heroku or Netlify for deployment (if web-based).

# 2. Frontend Development

## a. UI Layout and Structure:

- Single-page layout: Design the UI to display all components on one screen without the need for scrolling.  
- Sections for transaction input (Income, Savings, Expenses, Loans/EMI's).  
- Dropdowns, date pickers, and input fields for each transaction type.  
- Light/Dark mode toggle for theme switching.  
- Responsive design to ensure it works well on both mobile and desktop.

## b. Transaction Input Forms:

- For each transaction type (Income, Savings, Expenses, Loans/EMI's), create form components with:  
 - Dynamic dropdowns: Allow the user to select from previously entered values or add new ones.  
 - Date pickers: Easy selection of dates with an option to quickly select "today."  
 - Recurring fields: Dropdowns for Yes/No on recurring transactions and auto-reminders for recurring expenses and EMI payments.

## c. Charts and Summaries:

- Pie charts comparing Income, Expenses, Savings, and Loans.  
- Transaction summaries and total amounts displayed in a dashboard-like view.  
- Use Chart.js to generate charts dynamically based on fetched data.

# 3. Backend Development

## a. API Design:

- Design RESTful API endpoints for CRUD operations (Create, Read, Update, Delete) on transactions.  
- API routes for:  
 - Fetching transaction data: Filter by type (Income, Expenses, Savings, Loans/EMIs) and date range.  
 - Adding and updating transactions: Post new transactions or edit existing ones.  
 - Fetching stored dropdown values: For dynamic sources like income sources, descriptions, etc.  
 - Setting reminders for recurring expenses and EMI payments.

## b. Database Schema:

- Design MongoDB collections to store:  
 - Transactions: Fields for transaction type, amount, date, description, recurring status, etc.  
 - Dropdown values: Store income sources, expense descriptions, and savings goals.  
 - Reminders: Store details of recurring transactions and upcoming EMI due dates.

## c. Business Logic:

- Implement logic for auto-reminders (recurring expenses, loans).  
- Calculate total amounts for each category (Income, Expenses, Savings, Loans) and prepare data for chart visualization.  
- Handle logic for archiving, deleting, and editing transactions.

# 4. Additional Features

- Reminders and Notifications:  
 - Implement a backend scheduler (e.g., using Node-cron) to trigger reminders for recurring expenses and EMI payments.  
 - Store reminders in MongoDB and check them periodically to trigger in-app reminders.  
  
- Data Export:  
 - Create functionality to export data (transactions, summaries, reports) as PDF or Excel files.  
 - Use libraries like jsPDF for PDF export and SheetJS for Excel export.

# 5. Testing

- Unit Testing: Test each component separately to ensure all forms, dropdowns, and features work as expected.  
- Integration Testing: Test the integration between the frontend and backend to ensure smooth data flow and functionality.  
- User Testing: Conduct user testing to check the responsiveness and interactivity of the app across devices.

# 6. Deployment

- Backend Deployment: Host the Node.js backend on Heroku or AWS. Use MongoDB Atlas for the database.  
- Frontend Deployment: Deploy the React/Vue.js frontend on Netlify, Vercel, or other hosting platforms.  
- Continuous Integration/Deployment (CI/CD): Use GitHub Actions or CircleCI for continuous integration and automated deployment.

# 7. Maintenance and Future Enhancements

- Regularly update the dropdown values based on user input (Income sources, Expense descriptions).  
- Implement future user authentication if needed, as well as more advanced reporting and notifications.

# 8. Timeline Estimate

1. Week 1-2:  
 - Set up the project environment.  
 - Create UI layout and integrate forms for each transaction type.  
  
2. Week 3-4:  
 - Develop backend API with MongoDB and Express.  
 - Set up CRUD operations for transactions and dropdown values.  
  
3. Week 5-6:  
 - Implement charts and data export functionality.  
 - Add reminder logic for recurring expenses and loans/EMIs.  
  
4. Week 7:  
 - Final testing and bug fixes.  
 - User testing and performance optimization.  
  
5. Week 8:  
 - Deploy the app and monitor post-deployment performance.