KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PERSONAL EXPENSE TRACKER

AN APPLICATION PROJECT REPORT

for

WEB TECHNOLOGY (22CST42)

Submitted by

VARNIKA S (23CSR232)

SOWMIYA M V (23CSR211)

SUDEEP K (23CSR218)

INDEX

1. ABSTRACT

Managing personal finances can be particularly challenging for students, who often have limited income and face irregular spending patterns. The absence of structured financial planning tools results in poor budgeting, overspending, and difficulty in identifying areas for potential savings. This report presents the design and development of a web-based Personal Expense Tracker aimed at empowering students to take control of their financial activities. The system offers user authentication, expense logging, categorical organization, data visualization, and budget alert functionalities. By integrating these features, the tracker enables users to record and monitor their daily expenditures, categorize them into logical groups such as food, travel, and shopping, and gain insights into their financial habits through monthly reports. The application also allows users to set custom spending limits and receive notifications when these thresholds are exceeded. This solution not only helps users understand where their money goes but also encourages better financial decision-making through real-time feedback and detailed analytics. Ultimately, the Personal Expense Tracker acts as a digital assistant, promoting responsible spending and aiding students in cultivating sound financial habits.

2. PROBLEM STATEMENT

Students often face difficulties in managing their personal finances due to limited income, lack of financial literacy, and irregular spending habits. Traditional methods like maintaining handwritten logs or spreadsheets are not only time-consuming but also fail to provide meaningful insights into spending behavior. Without real-time feedback and organized data, students tend to lose track of their expenditures, leading to budget overruns and inefficient money management. There is a need for a dedicated digital solution that simplifies the process of recording, categorizing, and analyzing expenses while promoting financial discipline through alerts and reports.

Proposed Solution

To address the financial management challenges faced by students, we propose the development of a web-based **Personal Expense Tracker**. This application will allow users to securely log in, manage their personal profiles, and systematically track their daily expenses. Users can add, edit, or delete expense entries, categorize them (e.g., food, travel, shopping), and view monthly summaries in the form of intuitive visual graphs. The system will also offer a budgeting feature where users can set monthly spending limits and receive alerts when these limits are about to be or have been exceeded. By automating data tracking and providing real-time analytics, the solution aims to empower students to take control of their finances, make informed decisions, and build better financial habits.

3. TECHNOLOGIES USED

To develop the Personal Expense Tracker web application, the following technologies were utilized:

- **Express.js**: A fast, unopinionated, and minimalist web framework for Node.js. It was used to handle server-side routing, manage API requests, and integrate middleware for handling sessions, authentication, and data processing.
- **EJS (Embedded JavaScript Templates)**: A templating engine that allows embedding JavaScript code within HTML pages. EJS was used to render dynamic content on the client side, enabling a seamless user experience for expense listing, graph display, and form management.
- **MongoDB**: A NoSQL database used for storing user profiles and expense records. MongoDB’s document-based architecture allows flexible schema design, making it ideal for storing categorized expense data with varying structures.
- **Mongoose**: An Object Data Modeling (ODM) library for MongoDB that simplifies data validation, schema definition, and querying. Mongoose was used to define models for users and expense entries, enforce data integrity, and perform efficient database operations.

4. FEATURES OF THE WEB APPLICATION

1. **Home Page**

   The home page serves as the central hub of the application, providing users with easy access to various functionalities such as browsing recipes, searching for specific dishes, and viewing featured recipes. It is designed to be visually appealing and user-friendly, with intuitive navigation and quick links to popular categories..

2. **Expense Entry Management**

   Users can add, edit, or delete expense entries through a dedicated form. Each entry includes fields such as amount, category (e.g., food, travel, shopping), date, and description. This section enables users to log their daily spending activities accurately and maintain organized financial records

3. **Category-wise Expense View**

   Expenses are automatically grouped under predefined or user-defined categories, helping users track how much they spend on different aspects of life. A category-wise summary with visual representations (like pie charts or bar graphs) helps identify spending patterns and areas for potential savings.

4. **Monthly Reports**

   This feature provides detailed monthly summaries of a user's expenses, including total spending, category breakdown, and trends over time. Interactive charts and graphs help users visualize their financial habits, making it easier to plan budgets and assess progress toward financial goals.

5. **Search and Filter Expenses**

   Users can search for specific expenses using keywords, dates, or categories. This functionality helps quickly locate past transactions and analyze particular spending habits. Filters make it easy to sort through large volumes of data based on time periods or categories..

6. **Spending Limit Alerts**

   Users can set monthly spending limits, and the system will monitor expenditures in real time. Alerts are triggered when spending approaches or exceeds the defined threshold, encouraging users to stay within budget and manage their finances more responsibly.

7. **Database Storage Area**

   The MongoDB database stores all recipe data securely. The data is structured efficiently to allow quick retrieval and manipulation, ensuring a seamless user experience. The database is designed to handle large volumes of data and support complex queries, making it scalable and reliable.

5. IMPLEMENTATION DETAILS

1. **Express Framework**

Routing: Express.js is used as the primary backend framework to handle routing. Routes are defined for various functionalities such as user authentication, adding/editing/deleting expense entries, retrieving monthly reports, and setting budget limits. Each route corresponds to a specific HTTP method and is organized in a modular structure to promote maintainability and scalability.

Middleware: Middleware functions are implemented to manage tasks such as user authentication, session handling, request parsing, and error logging. Custom middleware is also used to validate user inputs and ensure that only authenticated users can access protected routes like expense management and report generation.

2. **EJS Templating**

Dynamic Content Rendering: EJS is used to dynamically render HTML pages based on user data from the MongoDB database. This includes generating views for dashboards, category summaries, and visual graphs. EJS helps in maintaining a responsive and interactive interface by embedding JavaScript directly into HTML.

Partial Views: Reusable partial templates are employed for common components such as headers, footers, and navigation menus. This reduces code duplication, ensures a consistent UI across the application, and simplifies updates to the design

3. **MongoDB and Mongoose**

Database Design: MongoDB stores user profiles and expense records in flexible, schema-less documents. Each expense document contains fields such as amount, category, date, description, and user reference. This format allows for fast and scalable data access, especially when filtering or aggregating data.

Data Modeling with Mongoose: Mongoose is used to define and enforce schemas for both users and expense entries. This enables strict data structure validation and seamless CRUD operations. Models are designed to reflect real-world entities and include references to manage user-specific data effectively.

Connection Management: Mongoose manages the MongoDB connection lifecycle efficiently. It supports indexing for faster querying and uses pooling techniques for improved performance in high-load scenarios.

Security Measures

To protect user data and ensure secure interactions, the following security measures are implemented:

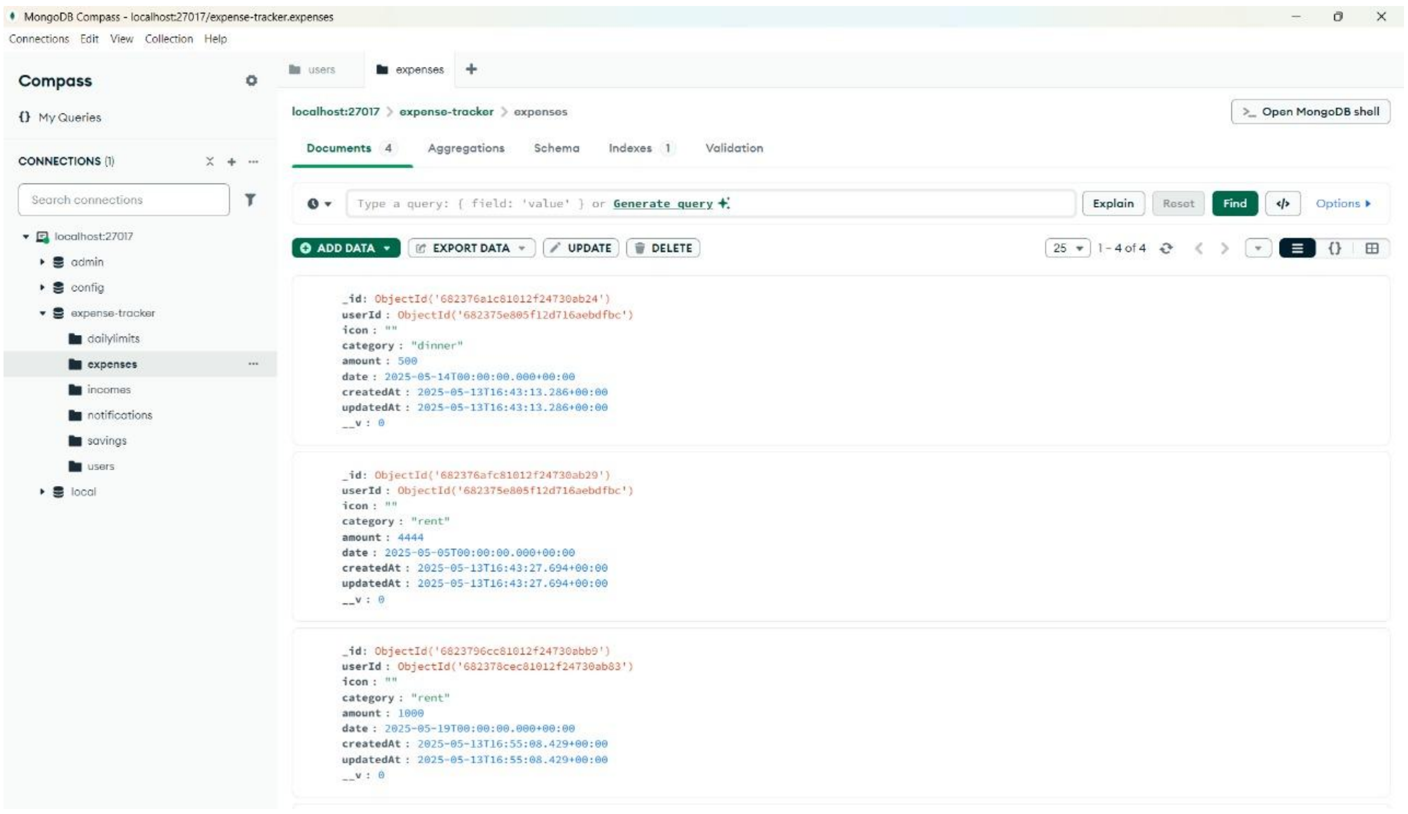
- **Authentication**: Secure user authentication is implemented using session-based login mechanisms. Passwords are hashed using bcrypt before being stored in the database. Middleware ensures that protected routes can only be accessed by authenticated users.
- **Data Validation**: Input validation is performed both client-side and server-side using tools like express-validator or Joi to prevent malicious input, enforce formatting rules, and ensure data integrity.
- **Encryption & HTTPS**: All sensitive data, including user credentials, is encrypted in storage and during transmission. SSL/TLS protocols are used to secure data communication between the client and server.

Documentation and Developer Support

The project is supported with thorough documentation, aimed at simplifying development, maintenance, and future enhancements:

- **API Reference**: Detailed documentation of all RESTful endpoints including request parameters, expected responses, and usage examples.
- **Data Models**: Clear explanations of user and expense schemas, detailing fields, data types, and relationships.
- **Usage Guides**: Step-by-step instructions on user account creation, logging expenses, generating reports, and setting budgets.
- **Code Snippets**: Sample backend and frontend code for integrating different features such as charts, form validations, and AJAX calls..
- **Support Resources**: FAQs, troubleshooting tips, and developer forums (if open-sourced) to foster community support and collaborative development.

6. TABLE DESIGN

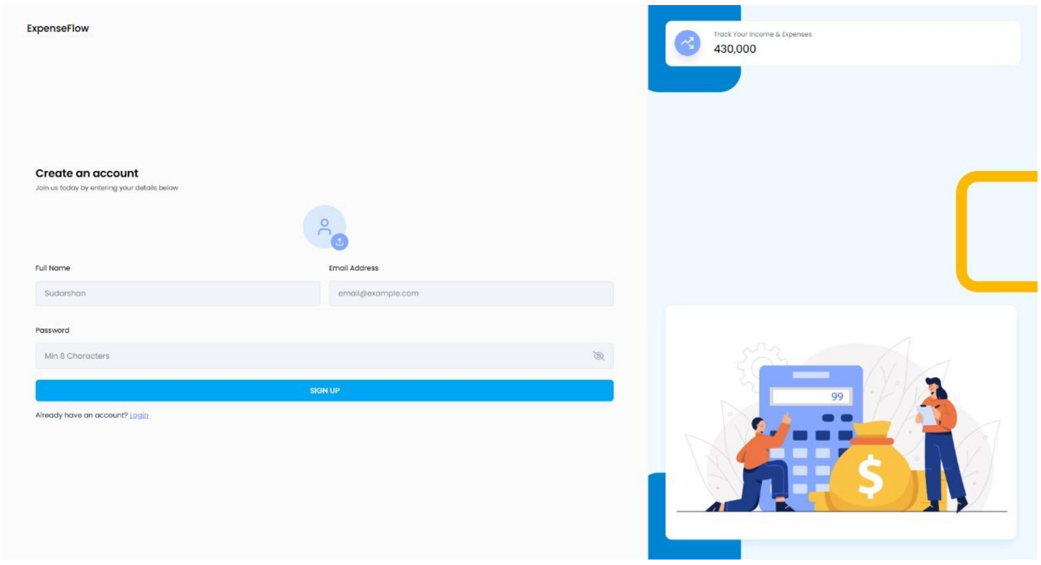


```
_id: ObjectId('682376a1c81012f24730ab24')
userId : ObjectId('682375e805f12d716aebdfbc')
icon : ""
category : "dinner"
amount : 500
date : 2025-05-14T00:00:00.000+00:00
createdAt : 2025-05-13T16:43:13.286+00:00
updatedAt : 2025-05-13T16:43:13.286+00:00
__v : 0

_id: ObjectId('682376afc81012f24730ab29')
userId : ObjectId('682375e805f12d716aebdfbc')
icon : ""
category : "rent"
amount : 4444
date : 2025-05-05T00:00:00.000+00:00
createdAt : 2025-05-13T16:43:27.694+00:00
updatedAt : 2025-05-13T16:43:27.694+00:00
__v : 0

_id: ObjectId('6823796cc81012f24730abb9')
userId : ObjectId('682378cec81012f24730ab83')
icon : ""
category : "rent"
amount : 1000
date : 2025-05-19T00:00:00.000+00:00
createdAt : 2025-05-13T16:55:08.429+00:00
updatedAt : 2025-05-13T16:55:08.429+00:00
__v : 0
```

Here we have used the database “expense-tracker”, and have created a collection “expenses”.

7. FRONTEND DESIGN

:

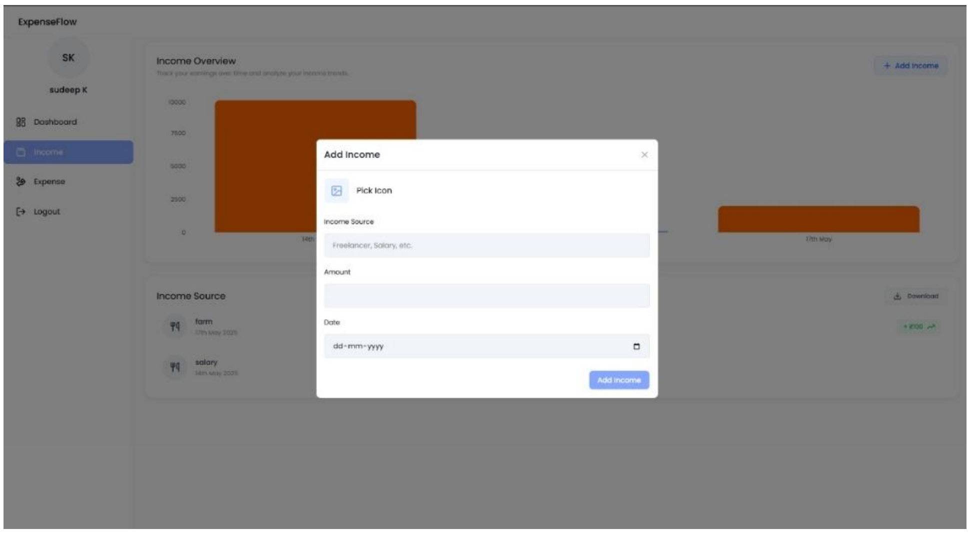


ExpenseFlow

Track Your Income & Expenses
430,000

Create an account
Join us today by entering your details below

Full Name: Sudarshan
Email Address: email@example.com
Password: Min 8 Characters



ExpenseFlow

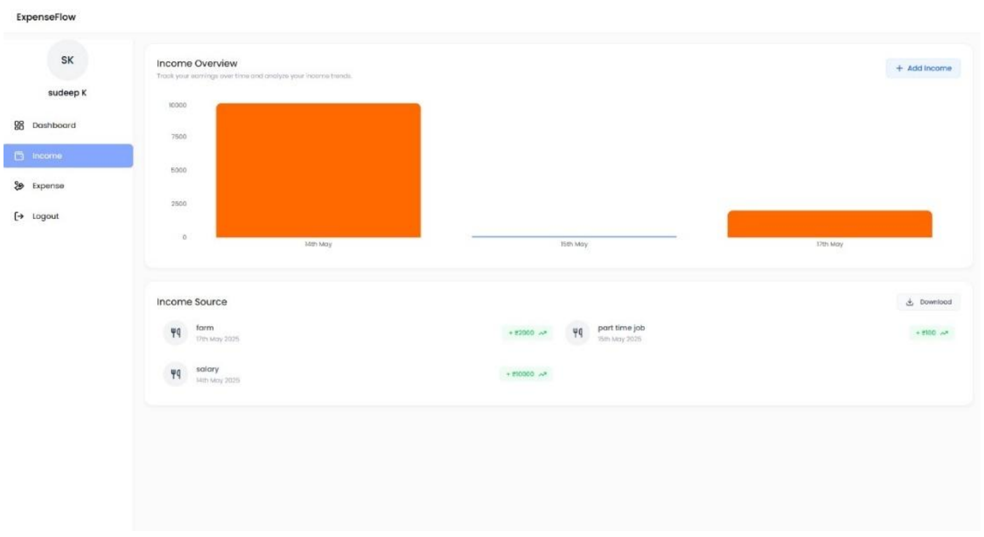
SK
sudeep K

Income Overview
Track your earnings over time and analyze your income trends.

Add Income

Income Source: Freelancer, Salary, etc.
Amount:
Date: dd-mm-yyyy

Income Source
farm — 17th May 2025
salary — 14th May 2025



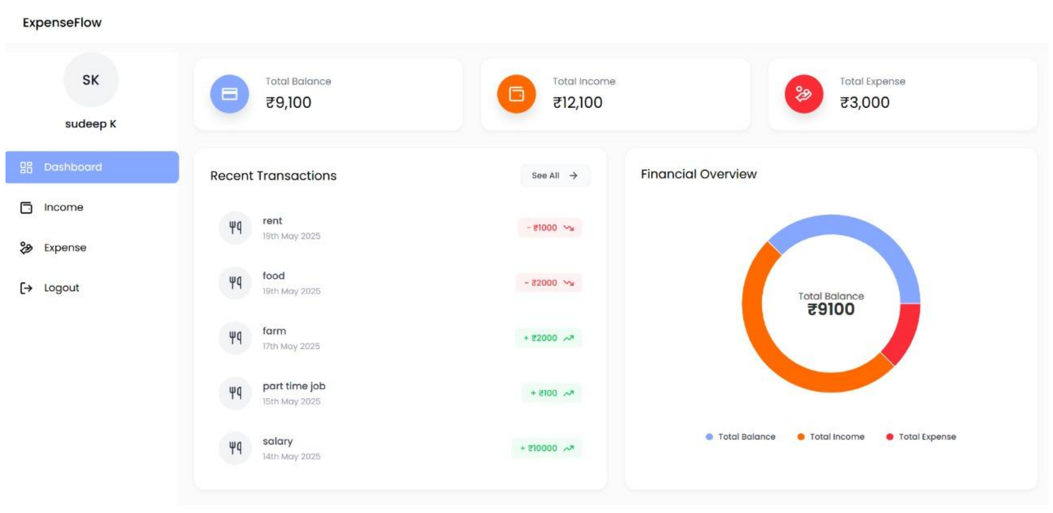
ExpenseFlow

SK
sudeep K

Income Overview
Track your earnings over time and analyze your income trends.

Income Source

farm 17th May 2025	+ ₹2000	part time job 15th May 2025	+ ₹100
salary 14th May 2025	+ ₹10000		



ExpenseFlow

SK

sudeep K

- Dashboard
- Income
- Expense
- Logout

Total Balance ₹9,100

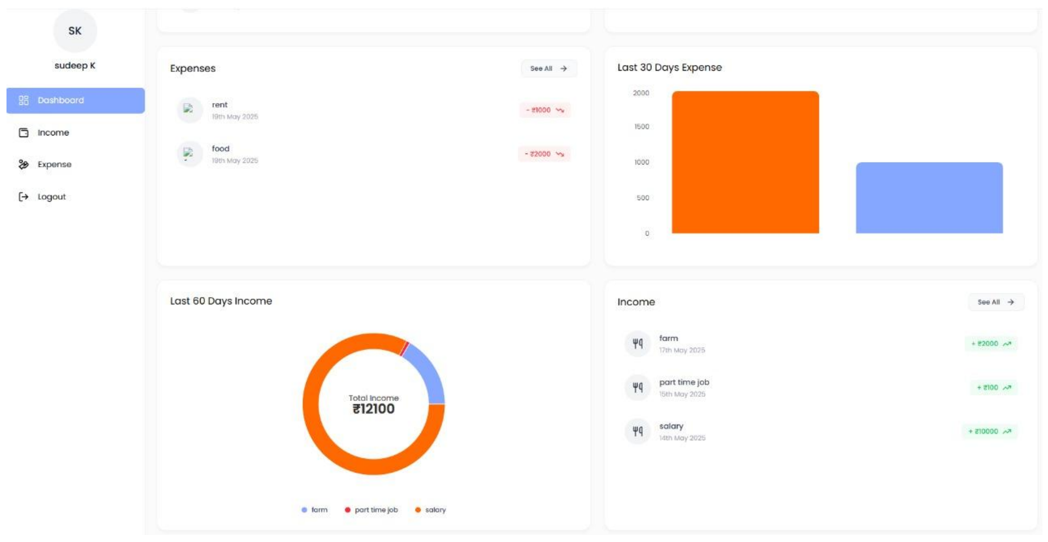
Total Income ₹12,100

Total Expense ₹3,000

Recent Transactions — See All

rent	19th May 2025	- ₹1000
food	19th May 2025	- ₹2000
farm	17th May 2025	+ ₹2000
part time job	15th May 2025	+ ₹100
salary	14th May 2025	+ ₹10000

Financial Overview



SK

sudeep K

- Dashboard
- Income
- Expense
- Logout

Expenses — See All

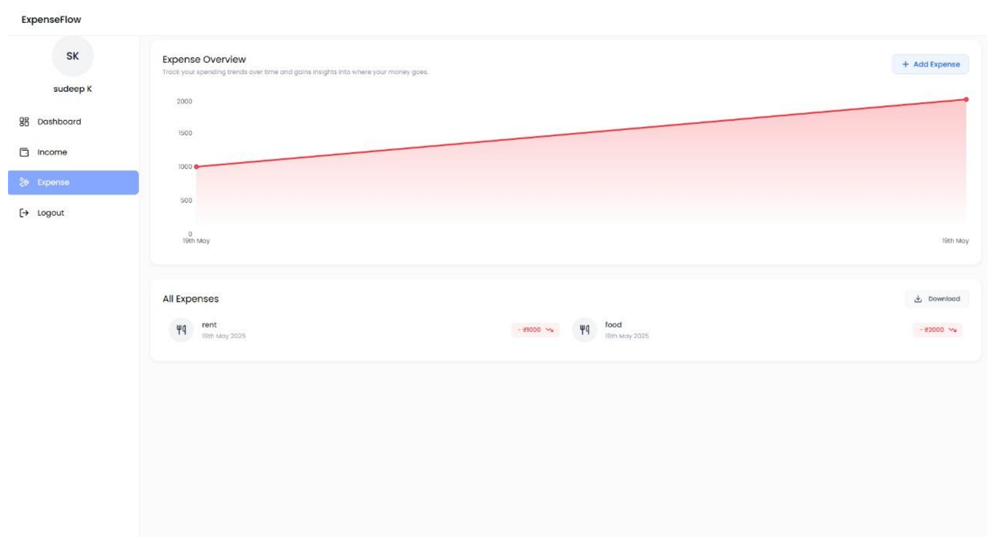
rent	19th May 2025	- ₹1000
food	19th May 2025	- ₹2000

Last 30 Days Expense

Last 60 Days Income

Income — See All

farm	17th May 2025	+ ₹2000
part time job	15th May 2025	+ ₹100
salary	14th May 2025	+ ₹10000



ExpenseFlow

SK

sudeep K

- Dashboard
- Income
- Expense
- Logout

Expense Overview

Track your spending trends over time and gains insights into where your money goes.

+ Add Expense

All Expenses — Download

rent	19th May 2025	- ₹1000
food	19th May 2025	- ₹2000

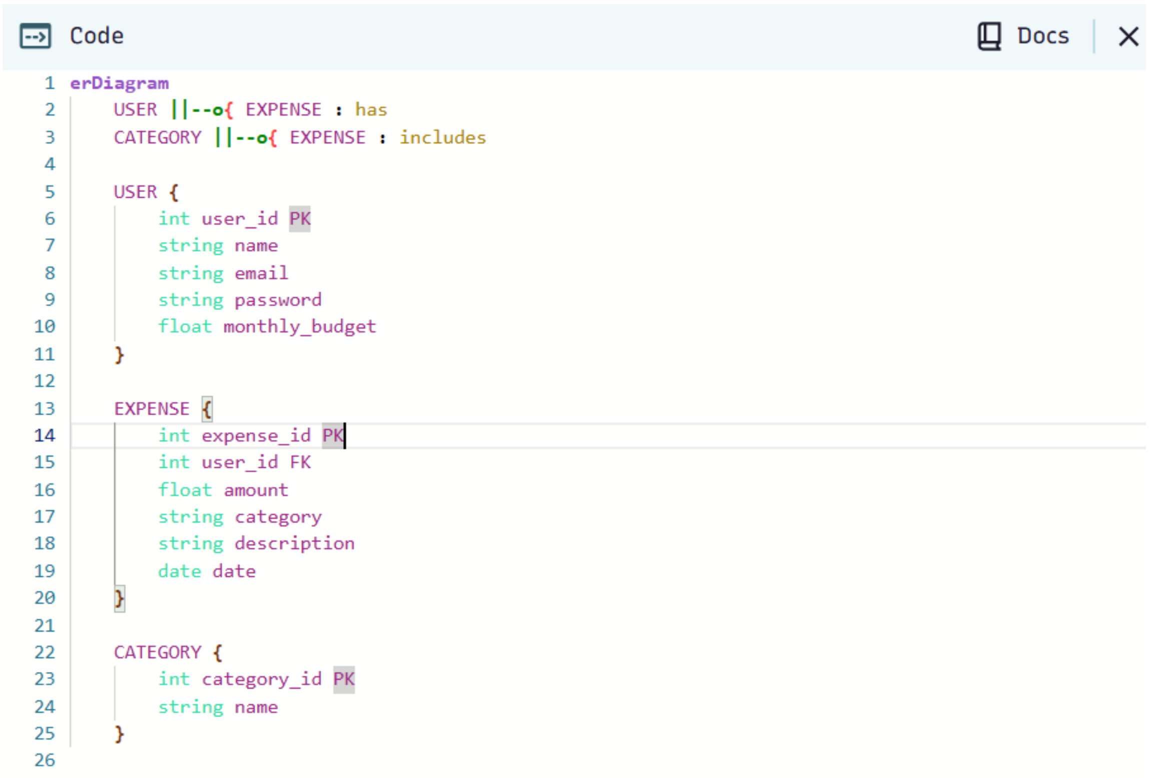
8. CONCLUSION

The Personal Expense Tracker web application addresses a common yet critical challenge faced by students—managing personal finances effectively. By offering features such as secure user authentication, expense logging, category-wise organization, monthly reports with visual charts, and customizable spending limit alerts, the system empowers users to gain better control over their financial behavior.

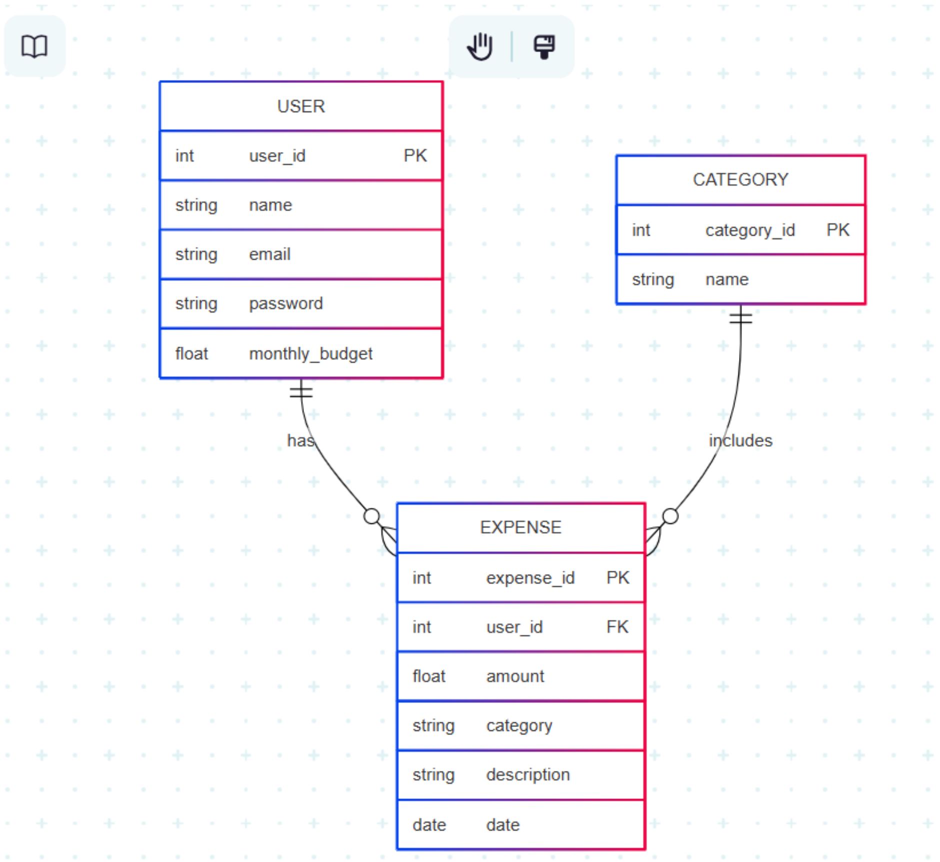
The use of modern technologies like Express.js, MongoDB, EJS, and Mongoose has enabled the development of a scalable, flexible, and user-friendly platform. With its intuitive interface and data-driven insights, the application not only reduces the complexity of manual expense tracking but also encourages financial discipline and informed decision-making.

This project demonstrates how a well-designed digital tool can make a meaningful difference in day-to-day student life. Looking ahead, the system can be further enhanced with features like automated expense categorization, mobile app integration, and AI-based financial recommendations to make it even more valuable and accessible.

9.ER DIAGRAM



```
erDiagram
    USER ||--o{ EXPENSE : has
    CATEGORY ||--o{ EXPENSE : includes

    USER {
        int user_id PK
        string name
        string email
        string password
        float monthly_budget
    }

    EXPENSE {
        int expense_id PK
        int user_id FK
        float amount
        string category
        string description
        date date
    }

    CATEGORY {
        int category_id PK
        string name
    }
```



10.CODING

App.jsx

```
import React from "react";
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Navigate,
} from "react-router-dom";
import Login from "./pages/Auth/Login";
import SignUp from "./pages/Auth/SignUp";
import Home from "./pages/Dashboard/Home";
import Income from "./pages/Dashboard/Income";
import Expense from "./pages/Dashboard/Expense";
import UserProvider from "./context/userContext";
import { Toaster } from "react-hot-toast";
import FloatingChatButton from "./components/FloatingChatButton";
import Recommendations from './pages/Recommendations';
function App() {
  return (
    <UserProvider>
      <div>
        <Router>
```

```
          <Routes>
            <Route path="/" element={<Root />} />
            <Route path="/login" exact element={<Login />} />
            <Route path="/signUp" exact element={<SignUp />} />
            <Route path="/dashboard" exact element={<Home />} />
            <Route path="/income" exact element={<Income />} />
            <Route path="/expense" exact element={<Expense />} />
            <Route path="/recommendations" element={<Recommendations />} />
          </Routes>
        </Router>
        <FloatingChatButton />
      </div>
      <Toaster
        toastOptions={{
          className: "",
          style: {
            fontSize: "13px",
          },}} />
    </UserProvider>  );}
export default App;
const Root = () => {
  const isAuthenticated = !!localStorage.getItem("token");
  return isAuthenticated ? (
    <Navigate to="/dashboard" />
  ) : (
    <Navigate to="/login" />
  );};
```

Main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

ROUTING

```
const express = require("express");
const {
  addExpense,
  getAllExpense,
  deleteExpense,
  downloadExpenseExcel,
} = require("../controllers/expenseController");
const { protect } = require("../middleware/authMiddleware");
const router = express.Router();
router.post("/add", protect, addExpense);
router.get("/get", protect, getAllExpense);
router.get("/downloadexcel", protect, downloadExpenseExcel);
router.delete("/delete/:id", protect, deleteExpense);
module.exports = router;
```

CONTROLLERS

```
const xlsx = require("xlsx");
const Expense = require("../models/Expense");
exports.addExpense = async (req, res) => {
  const userId = req.user.id;
  try {
    const { icon, category, amount, date } = req.body;
    if (!category || !amount) {
      return res.status(400).json({ message: "Category and amount are required." });
    }
    const newExpense = new Expense({
      userId,
      icon: icon || "",
      category,
      amount: Number(amount),
     date: date ? new Date(date) : new Date(),
    });
    await newExpense.save();
        return res.status(201).json({
      success: true,
      data: newExpense,
      message: "Expense added successfully"
    });
  } catch (error) {
    console.error("Add expense error:", error);
    return res.status(500).json({
      success: false,
      message: "Server Error",
      error: error.message
    });
  }
};
exports.getAllExpense = async (req, res) => {
  const userId = req.user.id;
  try {
    const expenses = await Expense.find({ userId }).sort({ date: -1 });
    return res.status(200).json(expenses);
  } catch (error) {
    console.error("Get all expenses error:", error);
    return res.status(500).json({
      success: false,
      message: "Failed to fetch expenses",
      error: error.message
    });
  }
};
exports.deleteExpense = async (req, res) => {
  const expenseId = req.params.id;
  const userId = req.user.id;
  try {
    const expense = await Expense.findOne({ _id: expenseId, userId });
    if (!expense) {
      return res.status(404).json({
        success: false,
        message: "Expense not found or you don't have permission to delete it"
      });
    }
        await Expense.findByIdAndDelete(expenseId);
        return res.status(200).json({
      success: true,
      message: "Expense deleted successfully"
    });
  } catch (error) {
    console.error("Delete expense error:", error);
    return res.status(500).json({
      success: false,
      message: "Failed to delete expense",
      error: error.message
    });
  }
};
exports.downloadExpenseExcel = async (req, res) => {
  const userId = req.user.id;
  try {
    const expenses = await Expense.find({ userId }).sort({ date: -1 });
    if (expenses.length === 0) {
      return res.status(404).json({
        success: false,
        message: "No expense data found to download"
      });
    }
    const data = expenses.map((item) => ({
      Category: item.category,
      Amount: item.amount,
      Date: new Date(item.date).toLocaleDateString(),
      Time: new Date(item.date).toLocaleTimeString(),
    }));
    const wb = xlsx.utils.book_new();
    const ws = xlsx.utils.json_to_sheet(data);
    const colWidths = [
      { wch: 20 }, // Category
      { wch: 15 }, // Amount
      { wch: 15 }, // Date
      { wch: 15 }, // Time
    ];
    ws['!cols'] = colWidths;
    xlsx.utils.book_append_sheet(wb, ws, "Expense Details");
       const filePath = `${__dirname}/../temp/expense_details_${Date.now()}.xlsx`;
    const dir = `${__dirname}/../temp`;
        const fs = require('fs');
    if (!fs.existsSync(dir)) {
      fs.mkdirSync(dir, { recursive: true });
    }
        xlsx.writeFile(wb, filePath);
          return res.download(filePath, "expense_details.xlsx", (err) => {
      if (err) {
        console.error("Download error:", err); }
      fs.unlinkSync(filePath);
    });
```

```
  } catch (error) {
    console.error("Download excel error:", error);
    return res.status(500).json({
      success: false,
      message: "Failed to generate Excel file",
      error: error.message
    });
  }
};
```