**Personal Expense Tracker Project Documentation**

**Problem Statement**

In today’s fast-paced world, individuals need to track and manage their expenses effectively. This project aims to build a personal expense tracker that allows users to log daily expenses, categorize them, and track spending against a monthly budget. The tracker will also have the capability to save and load expenses from a file for future reference.

**Objectives**

1. Design and implement a personal expense tracker that enables users to manage their expenses.
2. Allow users to categorize expenses and set monthly budgets.
3. Implement file-handling functionality to save and load expense data.
4. Create an interactive, menu-driven interface for ease of use.

**Steps to Perform**

**1. Add an Expense**

* **Functionality**: Create a function to prompt the user for the following expense details:
  + **Date of the expense**: In the format YYYY-MM-DD.
  + **Category**: Such as Food, Travel, etc.
  + **Amount spent**: The amount spent on the expense.
  + **Description**: A brief description of the expense.
* **Storage**: Store the expense in a list as a dictionary. Each dictionary will have the following keys: date, category, amount, and description. Example:

{'date': '2024-09-18', 'category': 'Food', 'amount': 15.50, 'description': 'Lunch with friends'}

**2. View Expenses**

* **Functionality**: Write a function to retrieve and display all stored expenses. The function should loop through the list of expenses and display the following information for each entry:
  + Date
  + Category
  + Amount
  + Description
* **Data Validation**: Ensure the function skips any incomplete entries or notifies the user if a required detail (date, category, amount, or description) is missing.

**3. Set and Track the Budget**

* **Set Budget**: Create a function that allows the user to input a monthly budget. Prompt the user to enter the total amount they want to budget for the month.
* **Track Budget**: Create a function to calculate the total expenses recorded so far and compare it with the user’s monthly budget. Implement the following conditions:
  + If the total expenses exceed the budget, display a warning: "You have exceeded your budget!"
  + If the expenses are within the budget, display the remaining balance: "You have [amount] left for the month."

**4. Save and Load Expenses**

* **Save Expenses**: Implement a function to save all expenses to a CSV file. Each row in the file will contain:
  + Date
  + Category
  + Amount
  + Description
* **Load Expenses**: Create a function to load expenses from the CSV file. When the program starts, the following should occur:
  + Read the saved data from the file.
  + Load it back into the list of expenses so the user can view their previous expenses and continue from where they left off.

**5. Create an Interactive Menu**

* **Menu Options**: Build a function that displays a menu with the following options:
  + Add expense
  + View expenses
  + Track budget
  + Save expenses
  + Exit
* **User Interaction**: Allow the user to enter a number to choose an option. Implement the following conditions:
  + If the user selects option 1, call the function to add an expense.
  + If the user selects option 2, call the function to view expenses.
  + If the user selects option 3, call the function to track the budget.
  + If the user selects option 4, call the function to save expenses to the file.
  + If the user selects option 5, save the expenses and exit the program.

**Folder Structure**

* **Frontend**
  + src/
    - assets/
    - components/
    - pages/
    - App.js
    - index.js
* **Backend**
  + server/
    - controllers/
    - models/
    - routes/
    - server.js

**File Names**

**Frontend**

* **App.js**
* **index.js**
* **components**
  + AddExpenseForm.jsx
  + BudgetTracker.jsx
  + ExpenseList.jsx
  + Home.jsx
* **assets**
  + app.css

**Backend**

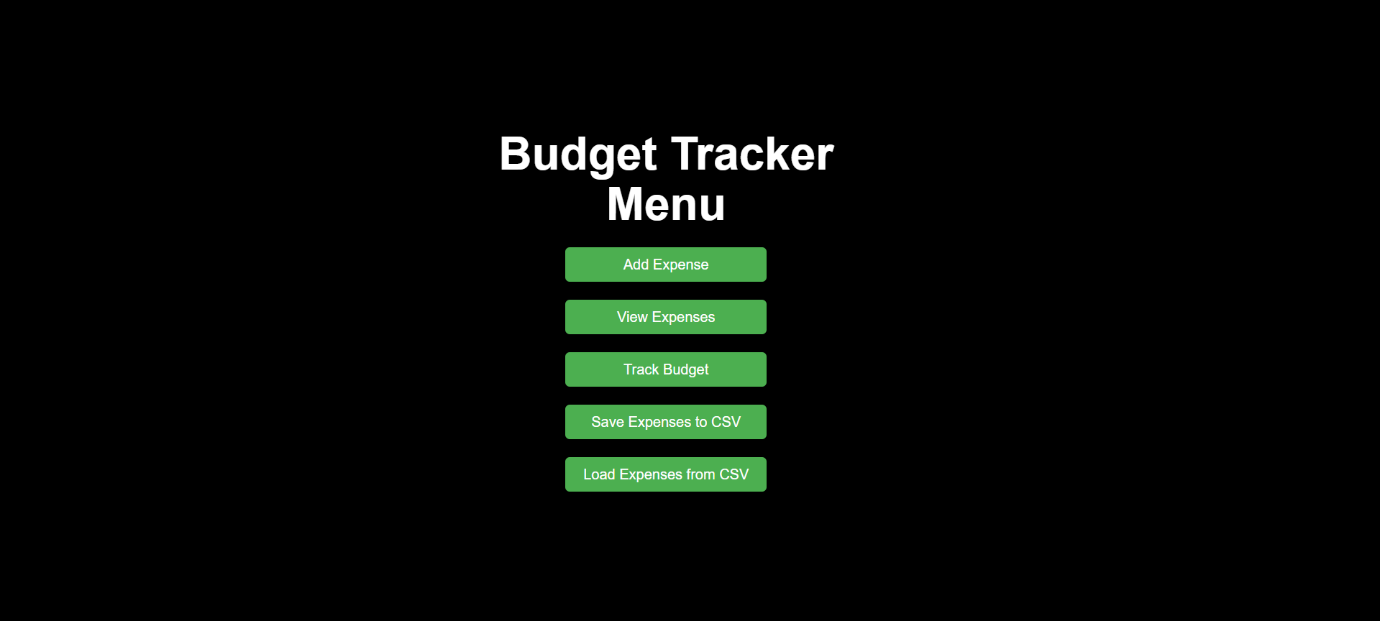
* **server.js**
* **controllers**
  + expenseController.js
* **config**
  + db.js
* **models**
  + Expense.js
* **routes**
  + expenseRoutes.js

|  |  |
| --- | --- |
| **Frontend Code** | |
| **App.jsx** | import React, { useEffect, useState } from "react";  import { fetchExpenses, saveExpenses, loadExpenses } from "./utils/api";  import AddExpenseForm from "./components/AddExpenseForm";  import ExpenseList from "./components/ExpenseList";  import "./styles/App.css";  import BudgetTracker from "./components/BudgetTracker";  import { BrowserRouter as Router, Routes, Route } from "react-router-dom";  import Home from "./components/Home";  const App = () => {    const [expenses, setExpenses] = useState([]);    useEffect(() => {      const getExpenses = async () => {        const response = await fetchExpenses();        setExpenses(response.data);      };      getExpenses();    }, []);    return (      <Router>        <Routes>          <Route path="/" element={<Home />} />          <Route            path="/add-expense"            element={              <AddExpenseForm                onAdd={(expense) => setExpenses([...expenses, expense])}              />            }          />          <Route            path="/view-expenses"            element={<ExpenseList expenses={expenses} />}          />          <Route path="/track-budget" element={<BudgetTracker />} />        </Routes>      </Router>    );  };  export default App; |
| **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>,  ) |
| **AddExpenseForm.jsx** | import React, { useState } from "react";  import { addExpense } from "../utils/api";  import { useNavigate } from "react-router-dom";  const AddExpenseForm = ({ onAdd }) => {    const [form, setForm] = useState({      date: "",      category: "",      amount: "",      description: "",    });    const handleSubmit = async (e) => {      e.preventDefault();      const response = await addExpense(form);      onAdd(response.data);      setForm({ date: "", category: "", amount: "", description: "" });    };    const navigate = useNavigate();    return (      <>        <div className="container">          <h1>Add Expense</h1>          <form className="add-expense-form" onSubmit={handleSubmit}>            <label>Date</label>            <input              type="date"              value={form.date}              onChange={(e) => setForm({ ...form, date: e.target.value })}              required            />            <label>Category</label>            <input              type="text"              placeholder="Category"              value={form.category}              onChange={(e) => setForm({ ...form, category: e.target.value })}              required            />            <label>Amount</label>            <input              type="number"              placeholder="Amount"              value={form.amount}              onChange={(e) => setForm({ ...form, amount: e.target.value })}              required            />            <label>Description</label>            <textarea              placeholder="Description"              value={form.description}              onChange={(e) => setForm({ ...form, description: e.target.value })}              required            />            <button type="submit">Add Expense</button>          </form>          <button className="exit-button" onClick={() => navigate("/")}>            Exit          </button>        </div>      </>    );  };  export default AddExpenseForm; |
| **BudgetTracker.jsx** | import React, { useState } from "react";  import { trackBudget } from "../utils/api";  import { useNavigate } from "react-router-dom";  const BudgetTracker = () => {    const [budget, setBudget] = useState("");    const [status, setStatus] = useState(null);    const navigate = useNavigate();    const handleTrack = async () => {      const response = await trackBudget(budget);      setStatus(response.data);    };    return (      <>        <div className="container">          <h1>Budget Tracker</h1>          <form className="add-expense-form" onSubmit={(e) => e.preventDefault()}>            <input              type="number"              placeholder="Monthly Budget"              value={budget}              onChange={(e) => setBudget(e.target.value)}            />            <button type="button" onClick={handleTrack}>              Track Budget            </button>          </form>          {status && (            <p className="budget-status">              Total Expenses: ₹ {status.total} | Remaining: ₹ {status.remaining}              {status.exceeded && " (You have exceeded your budget!)"}            </p>          )}          <button className="exit-button" onClick={() => navigate("/")}>            Exit          </button>        </div>      </>    );  };  export default BudgetTracker; |
| **ExpenseList.jsx** | import React from "react";  import { useNavigate } from "react-router-dom";  const ExpenseList = ({ expenses }) => {    const navigate = useNavigate();    return (      <>        <div className="container">          <h1 className="page-title">All Expenses</h1>          <table>            <thead>              <tr>                <th>Date</th>                <th>Category</th>                <th>Amount</th>                <th>Description</th>              </tr>            </thead>            <tbody>              {expenses.map((expense, index) => (                <tr key={index}>                  <td>{expense.date}</td>                  <td>{expense.category}</td>                  <td>₹ {expense.amount}</td>                  <td>{expense.description}</td>                </tr>              ))}            </tbody>          </table>          <button className="exit-button" onClick={() => navigate("/")}>            Exit          </button>        </div>      </>    );  };  export default ExpenseList; |
| **Home.jsx** | import React from "react";  import { useNavigate } from "react-router-dom";  import { saveExpenses, loadExpenses } from "../utils/api";  const Home = () => {    const navigate = useNavigate();    const handleSave = async () => {      await saveExpenses();      alert("Expenses saved to CSV!");    };    const handleLoad = async () => {      await loadExpenses();      alert("Expenses loaded from CSV!");      const response = await fetchExpenses();      saveExpenses(response.data);    };    return (      <>        <div className="container">          <h1>Budget Tracker Menu</h1>          <div className="home-buttons">            <button onClick={() => navigate("/add-expense")}>Add Expense</button>            <button onClick={() => navigate("/view-expenses")}>              View Expenses            </button>            <button onClick={() => navigate("/track-budget")}>              Track Budget            </button>            <button onClick={handleSave}>Save Expenses to CSV</button>            <button onClick={handleLoad}>Load Expenses from CSV</button>          </div>        </div>      </>    );  };  export default Home; |
| **api.jsx** | import axios from "axios";  const API = axios.create({ baseURL: "http://localhost:3000/api/expenses" });  export const fetchExpenses = () => API.get("/");  export const addExpense = (expense) => API.post("/add", expense);  export const trackBudget = (budget) =>    API.get(`/track-budget?budget=${budget}`);  export const saveExpenses = () => API.get("/save");  export const loadExpenses = () => API.get("/load"); |
| **App.css** | body {      margin: 0;      padding: 0;      font-family: Arial, sans-serif;      background-color: black;      color: white;      display: flex;      justify-content: center;      align-items: center;      height: 100vh;      overflow: hidden;  }  h1,  h2,  p {      text-align: center;      margin: 0;      padding: 10px;  }  .container {      text-align: center;      width: 90%;      max-width: 600px;  }  button {      padding: 10px 20px;      margin: 10px 0;      font-size: 16px;      background-color: #4caf50;      color: white;      border: none;      border-radius: 5px;      cursor: pointer;      width: 100%;  }  button:hover {      background-color: #45a049;  }  .exit-button {      margin-top: 20px;      background-color: #f44336;  }  .exit-button:hover {      background-color: #d32f2f;  }  .home-buttons {      display: flex;      flex-direction: column;      justify-content: center;      align-items: center;  }  table {      width: 100%;      margin-top: 20px;      border-collapse: collapse;      text-align: left;  }  th,  td {      padding: 10px;      border: 1px solid white;      width: 200px  }  th {      background-color: #333;  }  td {      background-color: #444;  }  .container {      display: flex;      flex-direction: column;      justify-content: center;      align-items: center;      height: 100vh;      background-color: black;      color: white;      text-align: center;  }  .add-expense-form {      display: flex;      flex-direction: column;      align-items: stretch;      width: 100%;      max-width: 400px;      gap: 10px;  }  .add-expense-form label {      text-align: left;      font-size: 1rem;      font-weight: bold;  }  .add-expense-form input,  .add-expense-form textarea,  .add-expense-form button {      padding: 10px;      font-size: 1rem;      border: none;      border-radius: 5px;      outline: none;  }  .add-expense-form button {      background-color: #007bff;      color: white;      cursor: pointer;  }  .add-expense-form button:hover {      background-color: #0056b3;  }  .exit-button {      margin-top: 20px;      padding: 10px 20px;      background-color: #ff0000;      color: white;      font-size: 1rem;      border: none;      border-radius: 5px;      cursor: pointer;  }  .exit-button:hover {      background-color: #cc0000;  } |

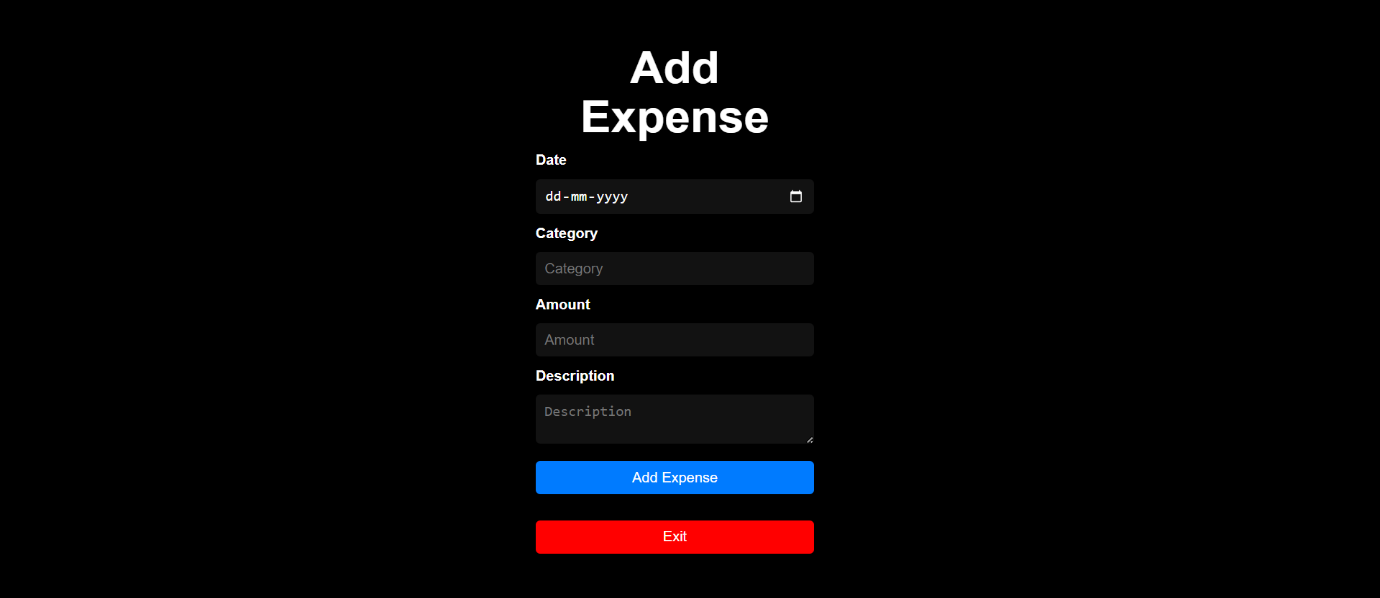
|  |  |
| --- | --- |
| **Backend Code** | |
| **expenseController.js** | const Expense = require("../models/Expense");  const fs = require("fs");  const csv = require("csv-parser");  const path = require("path");  const addExpense = async (req, res) => {    const { date, category, amount, description } = req.body;    try {      const expense = await Expense.create({        date,        category,        amount,        description,      });      res.status(201).json(expense);    } catch (error) {      res.status(400).json({ message: error.message });    }  };  const getExpenses = async (req, res) => {    try {      const expenses = await Expense.find();      res.json(expenses);    } catch (error) {      res.status(500).json({ message: error.message });    }  };  const trackBudget = async (req, res) => {    const { budget } = req.query;    try {      const expenses = await Expense.find();      const total = expenses.reduce((sum, expense) => sum + expense.amount, 0);      const remaining = budget - total;      res.json({ total, remaining, exceeded: remaining < 0 });    } catch (error) {      res.status(500).json({ message: error.message });    }  };  const saveExpensesToCSV = async (req, res) => {    try {      const expenses = await Expense.find();      const data = expenses.map(        (expense) =>          `${expense.date},${expense.category},${expense.amount},${expense.description}`      );      fs.writeFileSync(        "expenses.csv",        ["Date,Category,Amount,Description", ...data].join("\n")      );      res.status(200).json({ message: "Expenses saved to CSV" });    } catch (error) {      res.status(500).json({ message: error.message });    }  };  const loadExpensesFromCSV = async (req, res) => {    try {      const filePath = path.join(\_\_dirname, "../expenses.csv");      if (!fs.existsSync(filePath)) {        return res.status(404).json({ message: "CSV file not found." });      }      const csvData = fs.readFileSync(filePath, "utf8");      const rows = csvData.split("\n").slice(1);      const expenses = rows.map((row) => {        const [date, category, amount, description] = row.split(",");        if (!date || !category || !amount || !description) {          throw new Error(            `Invalid data: ${row}. All fields (date, category, amount, description) are required.`          );        }        return {          date: date.trim(),          category: category.trim(),          amount: amount.trim(),          description: description.trim(),        };      });      await Expense.insertMany(expenses);      res.status(200).json({ message: "Expenses loaded from CSV successfully!" });    } catch (error) {      console.error("Error loading expenses from CSV:", error.message);      res.status(500).json({ message: error.message });    }  };  module.exports = {    addExpense,    getExpenses,    trackBudget,    saveExpensesToCSV,    loadExpensesFromCSV,  }; |
| **db.js** | const mongoose = require("mongoose");  const connectDB = async () => {    try {      const conn = await mongoose.connect(process.env.MONGO\_URI, {        useNewUrlParser: true,        useUnifiedTopology: true,      });      console.log(`MongoDB Connected: ${conn.connection.host}`);    } catch (error) {      console.error(`Error: ${error.message}`);      process.exit(1);    }  };  module.exports = connectDB; |
| **Expense.js** | const mongoose = require("mongoose");  const expenseSchema = mongoose.Schema(    {      date: { type: String, required: true },      category: { type: String, required: true },      amount: { type: Number, required: true },      description: { type: String, required: true },    },    { timestamps: true }  );  module.exports = mongoose.model("Expense", expenseSchema); |
| **expenseRoutes.js** | const express = require("express");  const {    addExpense,    getExpenses,    trackBudget,    saveExpensesToCSV,    loadExpensesFromCSV,  } = require("../controllers/expenseController");  const router = express.Router();  router.post("/add", addExpense);  router.get("/", getExpenses);  router.get("/track-budget", trackBudget);  router.get("/save", saveExpensesToCSV);  router.get("/load", loadExpensesFromCSV);  module.exports = router; |
| **server.js** | require("dotenv").config();  const express = require("express");  const cors = require("cors");  const connectDB = require("./config/db");  const expenseRoutes = require("./routes/expenseRoutes");  connectDB();  const app = express();  app.use(cors());  app.use(express.json());  app.use("/api/expenses", expenseRoutes);  const PORT = process.env.PORT || 5000;  app.listen(PORT, () => console.log(`Server running on port ${PORT}`)); |

**Screenshots of Output**

**1. Home Page with All Buttons**



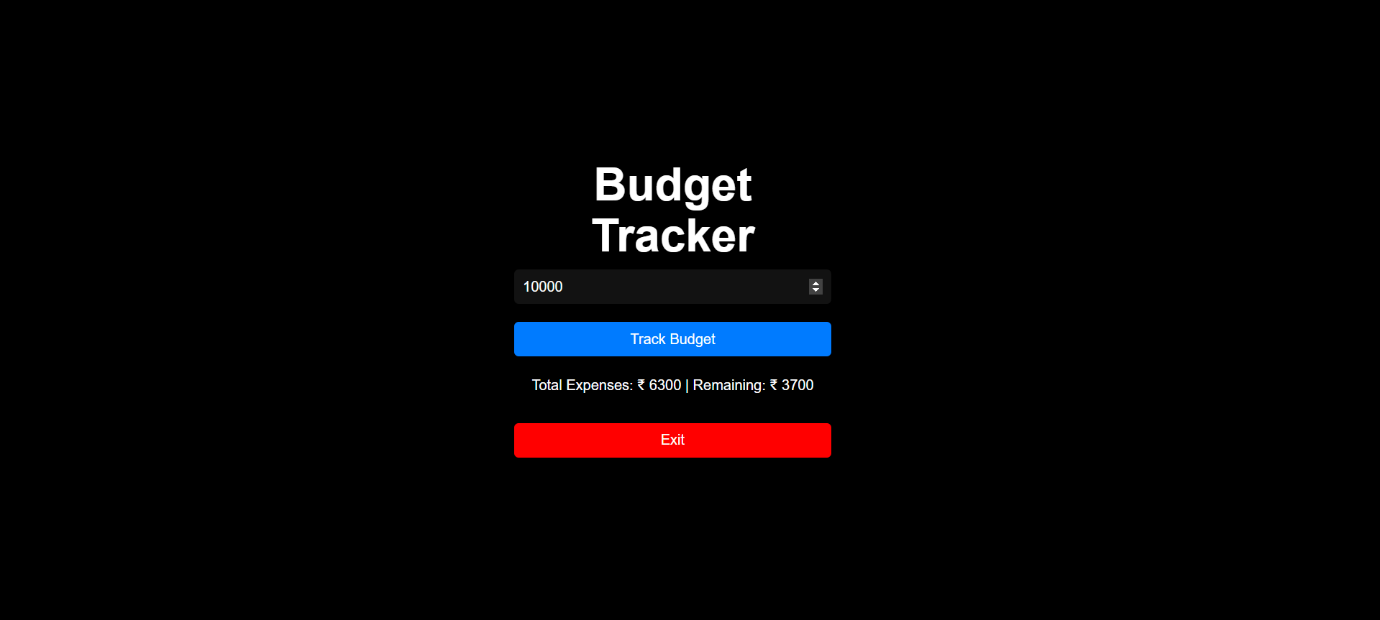
**2. Add Expense Page**



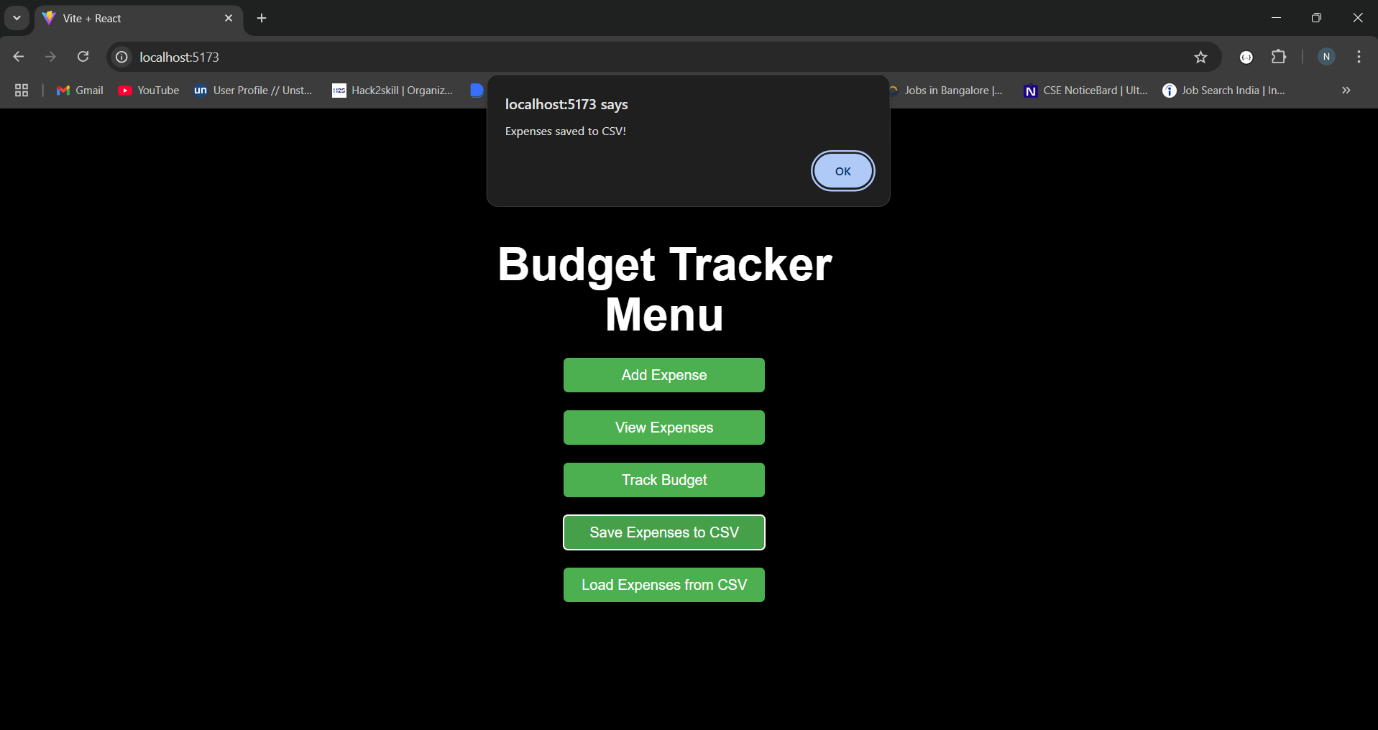
**3. All Expenses Page**



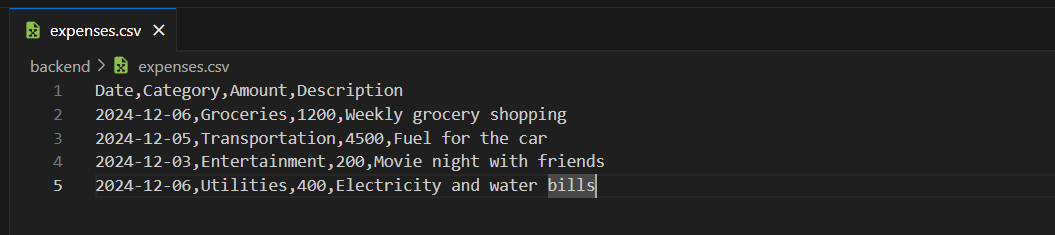
**4. Track Budget Page**



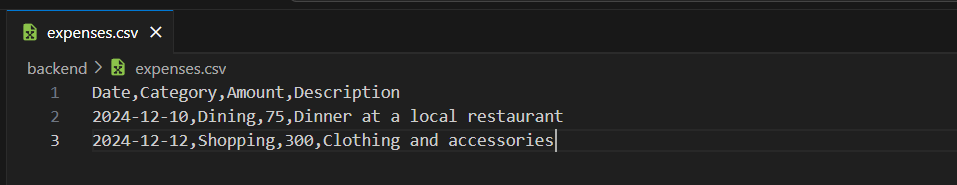
**5. Save All Expenses to CSV File**



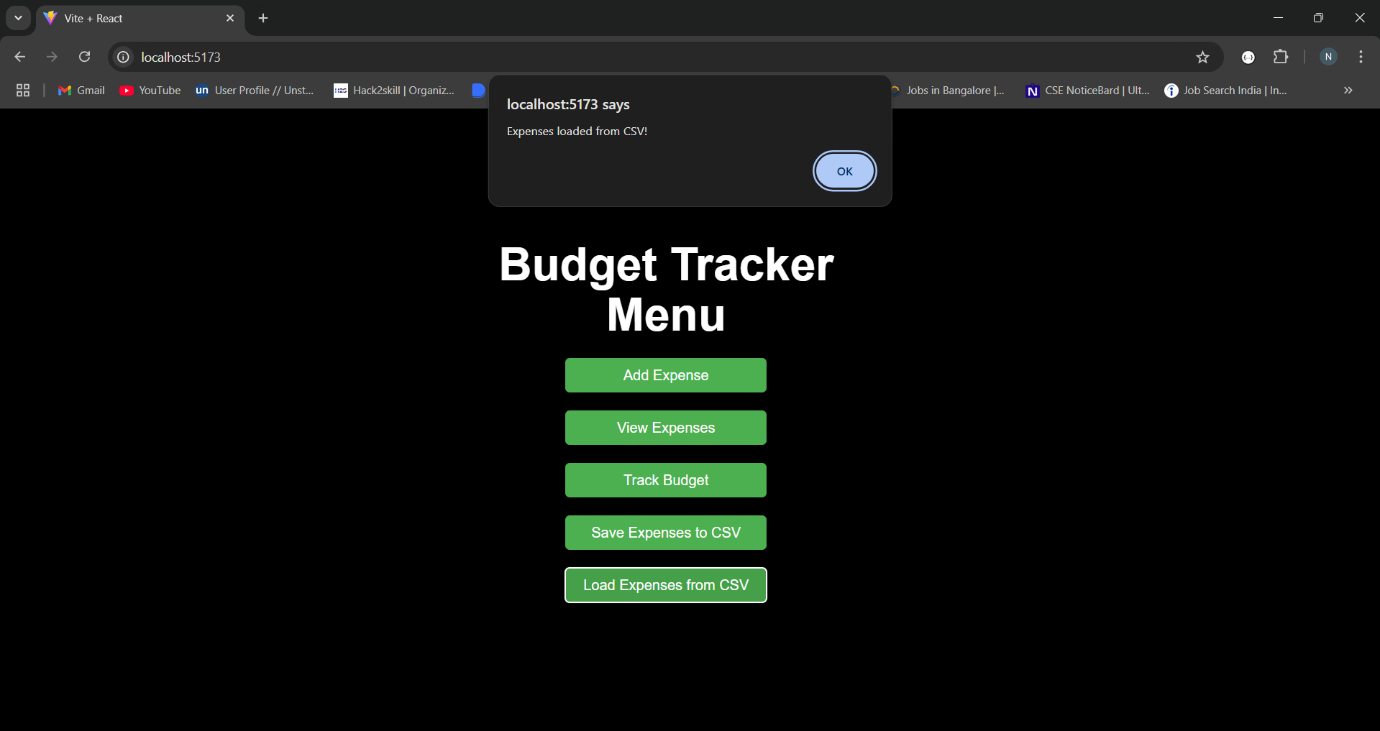
**6. All Expenses Saved in CSV File**



**7. Add Expenses Through CSV File**

****

**8. Load All Expenses from CSV**



**9. Updated All Expenses Page**

