A REPORT

ON

SpendSmart

Offered by the

Department Computer Science and Engineering School of Engineering and Sciences

Submitted by

K Karthikeya Sharma	AP23110010634
N Vivek	AP23110010351
L Vishal	AP23110010621
Sk Arshad	AP23110010316

SRM University–AP
Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522 240

December, 2025

SpendSmart – Personal Expense Tracker (React)

Introduction

Step into the future of personal finance management – a revolutionary financial tracking experience awaits you with our state-of-the-art SpendSmart application, intricately designed using the brilliance of React.js. Merging innovation with a user-centric approach, our expense tracker is poised to redefine how you engage with and immerse yourself in the world of personal financial management.

Crafted for the modern individual, our React-based SpendSmart application seamlessly integrates robust functionality with an intuitive user interface. From tracking daily expenses to planning monthly budgets, our platform ensures a comprehensive financial journey tailored to your unique lifestyle. The core of SpendSmart beats with React, a dynamic and feature-rich JavaScript library. Immerse yourself in a visually stunning and interactive interface, where every transaction entry, budget allocation, and analytics visualization feels like a financial revelation.

Whether you're on a desktop, tablet, or smartphone, our responsive design guarantees a consistent and enjoyable financial management experience across all devices. Bid farewell to the constraints of traditional expense tracking methods and embrace a realm of possibilities with our React-based SpendSmart application. Join us on this financial expedition as we transform the way you connect with and understand your spending habits. Get ready to elevate your financial awareness – it's time to hit "Track" on a new era of personal finance management.

Scenario-Based Introduction

Imagine a typical day in the life of Meera, a young professional who has just started her first job. She arrives at her desk with coffee in hand, excited about her new career but anxious about managing her finances independently. Every month, she struggles to understand where her salary goes – food, entertainment, bills, travel, and subscriptions all seem to add up mysteriously.

One evening, frustrated after checking her bank balance, she discovers SpendSmart.

As she opens the application for the first time, SpendSmart greets her with a clean, intuitive dashboard customized to welcome new users. She quickly sets up her monthly income and defines her budget categories. The next morning, after buying breakfast, she

immediately logs the expense under the "Food" category. SpendSmart instantly updates her dashboard, showing her remaining food budget for the month.

By mid-month, Meera notices something interesting on her dashboard. The colorful pie chart reveals that she's spending nearly 40% of her budget on food and dining out. The visual representation is eye-opening. She decides to cook more at home for the remaining weeks.

Later, when planning a weekend trip with friends, she checks her "Entertainment" budget and realizes she has enough room without compromising her savings goal. The budget tracker gives her the confidence to enjoy life while staying financially responsible.

At month-end, SpendSmart's analytics show Meera that she successfully saved 15% of her income – her first significant savings since starting work. With SpendSmart's clarity, simplicity, and insightful guidance, Meera transforms from a financially anxious beginner to a confident money manager.

Target Audience

SpendSmart is tailored for a diverse community of individuals, including:

Students: Young learners managing allowances and part-time job income who want to develop responsible spending habits early in life.

Working Professionals: Salaried employees seeking to track monthly expenses, plan budgets, and ensure they live within their means while building savings.

Freelancers: Independent workers with irregular income streams who need flexible expense tracking and budget adjustment capabilities.

Families: Households tracking collective expenses across multiple categories to manage family budgets effectively.

Financial Awareness Seekers: Anyone passionate about understanding their spending patterns and cultivating better money management habits.

Project Goals and Objectives

Primary Goal:

The primary goal of SpendSmart is to offer a seamless platform for individuals to manage their personal finances, facilitating an organized and efficient expense tracking experience with real-time analytics and budget monitoring.

Objectives:

User-Friendly Interface: Develop an intuitive interface that enables users to navigate, add transactions, view analytics, and manage budgets effortlessly, promoting a smooth and productive financial management environment.

Comprehensive Financial Tracking: Provide robust features for organizing and categorizing financial data, including advanced filtering options for streamlined transaction discovery and category-based expense analysis.
Real-Time Budget Monitoring: Implement dynamic budget tracking that alerts users when they approach spending limits, helping them make informed decisions throughout the month.
Visual Analytics Dashboard: Integrate interactive charts and graphs using modern visualization libraries to present financial data in an easily digestible format.
Modern Tech Stack: Leverage cutting-edge web development technologies, such as React.js, TailwindCSS, and JSON Server, to ensure an efficient and enjoyable user experience while navigating and interacting with the expense tracking application.

Key Features

Dashboard Overview: A comprehensive dashboard displaying total income, total expenses, current balance, and monthly budget status with visual indicators for quick financial assessment.
Transaction Management: Complete CRUD functionality allowing users to add income and expense entries with details including amount, category, description, and date.
Category-Based Organization: Predefined and customizable expense categories (Food, Transportation, Entertainment, Utilities, Healthcare, Shopping, etc.) for organized financial tracking.
Budget Planning: Monthly budget setup with category-wise allocation, enabling users to set spending limits and receive notifications when approaching thresholds.
Visual Analytics: Interactive charts including pie charts for category distribution, bar graphs for income vs. expense comparison, and line graphs for monthly spending trends.
Responsive Design: Fully optimized interface that adapts seamlessly across desktop, tablet, and mobile devices.

Pre-requisites

Here are the key prerequisites for developing SpendSmart using React.js:
Node.js and npm:
Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. Install Node.js and npm on your development machine.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/package-manager/

React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components.

- Create a new React app: npm create vite@latest
- Enter project name, select React framework and JavaScript variant
- Navigate to the project directory: cd spendsmart
- Install dependencies: npm install
- Running the React App: npm run dev
- Access your React app at http://localhost:5173

TailwindCSS:

TailwindCSS is a utility-first CSS framework for rapidly building custom user interfaces.

- Installation: npm install -D tailwindcss postcss autoprefixer
- Initialize: npx tailwindcss init -p

JSON Server:

JSON Server creates a full fake REST API for prototyping and development.

- Installation: npm install -g json-server
- Run server: json-server --watch db.json --port 3001

Additional Libraries:

- Axios: npm install axios
- Recharts: npm install recharts
- React Router: npm install react-router-dom

HTML, CSS, and JavaScript:

Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Version Control:

Use Git for version control, enabling collaboration and tracking changes throughout the development process.

- Git: https://git-scm.com/downloads

Development Environment:

Choose a code editor or IDE that suits your preferences.

- Visual Studio Code: https://code.visualstudio.com/download
- Sublime Text: https://www.sublimetext.com/download
- WebStorm: https://www.jetbrains.com/webstorm/download

Project Structure

The project structure is organized logically to improve code maintainability:

Project Flow

Milestone 1: Project Demo:

Before starting to work on this project, let's see the demo.

Use the code in: https://github.com/karthikeyasharma20/SpendSmart1

ProjectDemonstration:https://drive.google.com/file/d/1oJrk14m-52bFq0xS_td7K4VMjc7rFLHD/view?usp=drive_link

CodeDemonstration:https://drive.google.com/file/d/12aCrpLtUA2EAEfJ_Zwy_hLopRse3n6N2/view?usp=drive_link

Milestone 2: Project Setup and Configuration

Install Required Tools:

Open the project folder to install necessary tools. In this project, we use:

- React.js
- TailwindCSS
- Axios
- Recharts
- React Router
- JSON Server

Setup Steps:

1. Create project: npm create vite@latest spendsmart
2. Navigate to directory: cd spendsmart
3. Install dependencies: npm install
4. Install additional packages:
5. npm install axios recharts react-router-domnpm install -D tailwindcss postcss autoprefixernpx tailwindcss init -p
6. Install JSON Server: npm install -g json-server

For Further Reference:

- https://react.dev/learn/installation
- https://tailwindcss.com/docs/guides/vite

Milestone 3: Backend Setup - JSON Server Configuration

Setup JSON Server Database:

Create db.json file with initial data structure:

Code Description:

- Creates JSON database file with users, transactions, budgets, and categories collections
- Users collection stores user credentials and profile information
- Transactions collection holds all income and expense records with userId, type, category, amount, description, and date
- Budgets collection manages monthly budget data with category-wise allocations
- Categories collection defines available expense categories with icons and colors
- JSON Server automatically creates RESTful API endpoints for all collections
- Supports GET, POST, PUT, DELETE operations on each endpoint

Running JSON Server:

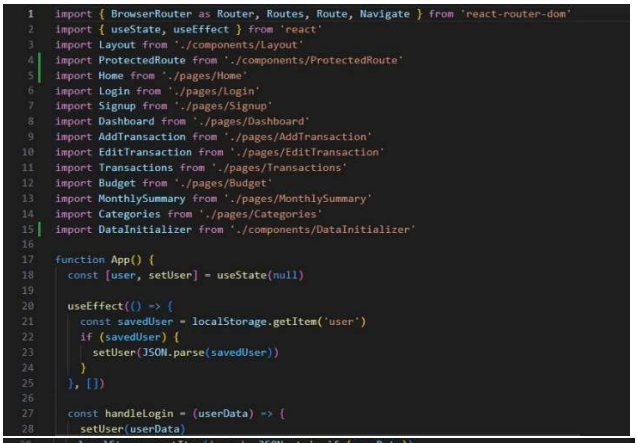
json-server --watch db.json --port 3001

Milestone 4: Web Development

Setup React Application:

- Create React application
- Configure TailwindCSS
- Setup routing
- Install required libraries

App.jsx Component:



```jsx
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom'
import { useState, useEffect } from 'react'
import Layout from './components/Layout'
import ProtectedRoute from './components/ProtectedRoute'
import Home from './pages/Home'
import Login from './pages/Login'
import Signup from './pages/Signup'
import Dashboard from './pages/Dashboard'
import AddTransaction from './pages/AddTransaction'
import EditTransaction from './pages/EditTransaction'
import Transactions from './pages/Transactions'
import Budget from './pages/Budget'
import MonthlySummary from './pages/MonthlySummary'
import Categories from './pages/Categories'
import DataInitializer from './components/DataInitializer'

function App() {
  const [user, setUser] = useState(null)

  useEffect(() => {
    const savedUser = localStorage.getItem('user')
    if (savedUser) {
      setUser(JSON.parse(savedUser))
    }
  }, [])

  const handleLogin = (userData) => {
    setUser(userData)
    localStorage.setItem('user', JSON.stringify(userData))
  }

  const handleLogout = () => {
    setUser(null)
    localStorage.removeItem('user')
  }

  return (
    <Router>
      {/* Initialize data from external APIs when user is logged in */}
      {user && <DataInitializer userId={user.id} />}
      <Routes>
        {/* Public Routes */}
        <Route
          path="/"
          element={!user ? <Home /> : <Navigate to="/dashboard" replace />}
        />
        <Route
          path="/login"
          element={!user ? <Login onLogin={handleLogin} /> : <Navigate to="/dashboard" replac
        />
        <Route
          path="/signup"
          element={!user ? <Signup onLogin={handleLogin} /> : <Navigate to="/dashboard" repla
        />

        {/* Protected Routes with Layout */}
        <Route
          element={
            <ProtectedRoute user={user}>
              <Layout user={user} onLogout={handleLogout} />
            </ProtectedRoute>
          }
        >
          <Route path="/dashboard" element={<Dashboard user={user} />} />
          <Route path="/transactions" element={<Transactions user={user} />} />
          <Route path="/transactions/add" element={<AddTransaction user={user} />} />
          <Route path="/transactions/edit/:id" element={<EditTransaction user={user} />} />
          <Route path="/budget" element={<Budget user={user} />} />
          <Route path="/summary" element={<MonthlySummary user={user} />} />
          <Route path="/categories" element={<Categories user={user} />} />
        </Route>
      </Routes>
    </Router>
  )
}

export default App
```

Code Description:

- ./App.css: The CSS file for styling the App component
- Dashboard, Transactions, Budget, Analytics, Login from pages: Page components that will be displayed
- AppProvider from ./context/AppContext: A context provider component to manage the application's state
- The App function is defined, which returns JSX to render the AppProvider and routing structure
- BrowserRouter wraps the entire application to enable client-side routing
- Routes component defines all application routes with their corresponding page components

- Each Route maps a URL path to a specific page component
- The App component is wrapped within the AppProvider to ensure all components can access the context
- The App component is exported as the default export, making it available for import in main.jsx

Design UI Components:

- Create reusable components
- Implement layout and styling with TailwindCSS
- Add navigation with React Router

Implement Frontend Logic:

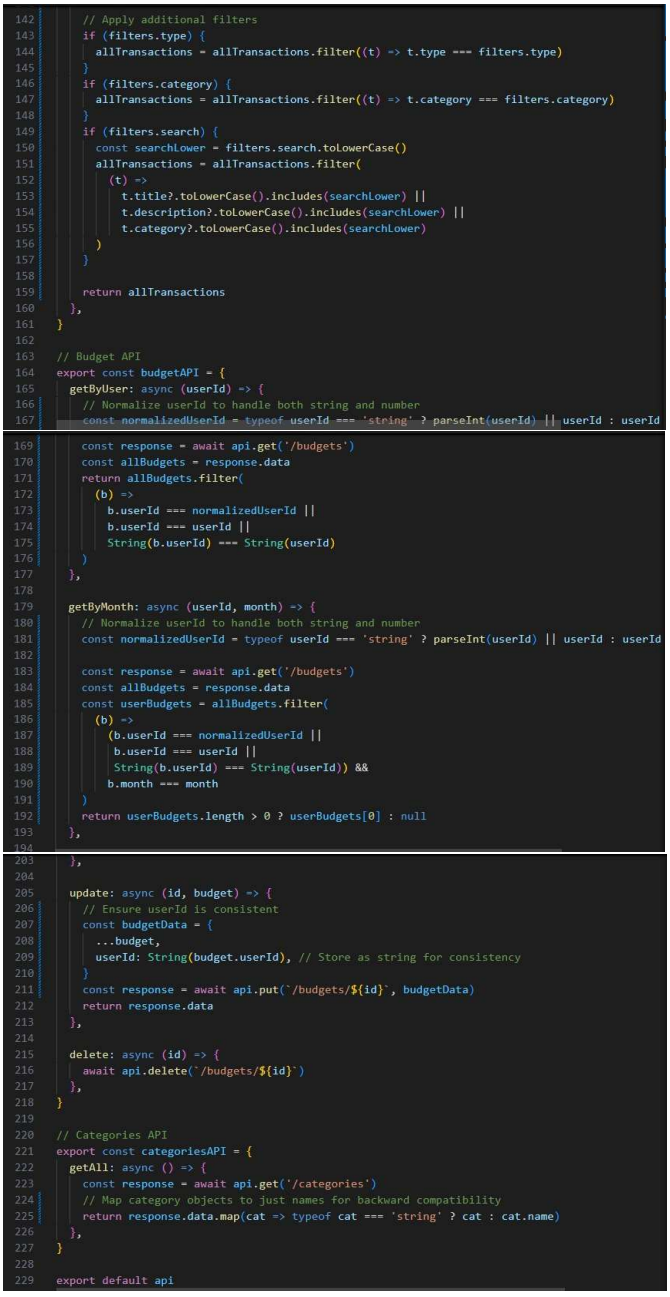
- Integration with JSON Server API endpoints
- Implement data binding using React state and context

Milestone 5: API Service Layer

api.js:

```
import axios from 'axios'

const API_URL = 'http://localhost:3001'

const api = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json',
  },
})

// Users API
export const authAPI = {
  login: async (email, password) => {
    try {
      // Get all users and filter manually for exact match
      const response = await api.get('/users')
      const users = response.data || []

      // Find user with matching email and password
      const user = users.find(
        (u) => u.email === email && u.password === password
      )

      if (user) {
        // Ensure userId is consistent (convert to number if it's a string)
        return {
          ...user,
          id: typeof user.id === 'string' ? parseInt(user.id) || user.id : user.id,
        }
      }

      throw new Error('Invalid credentials')
    } catch (error) {
      // Handle network errors
      if (error.code === 'ECONNREFUSED' || error.message.includes('Network Error')) {
        throw new Error('Cannot connect to server. Please make sure JSON-Server is running on
      }
      // Re-throw other errors
      throw error
    }
  },

  signup: async (userData) => {
    try {
      // Check if email already exists
      const response = await api.get('/users')
      const existingUsers = response.data || []

      const emailExists = existingUsers.some((u) => u.email === userData.email)
      if (emailExists) {
        throw new Error('Email already exists')
      }
```

```
      // Create new user (JSON-Server will auto-generate ID)
      const newUser = {
        username: userData.username,
        email: userData.email,
        password: userData.password,
      }

      const createResponse = await api.post('/users', newUser)
      const createdUser = createResponse.data

      // Ensure userId is consistent
      return {
        ...createdUser,
        id: typeof createdUser.id === 'string' ? parseInt(createdUser.id) || createdUser.id :
      }
    } catch (error) {
      // Handle network errors
      if (error.code === 'ECONNREFUSED' || error.message.includes('Network Error')) {
        throw new Error('Cannot connect to server. Please make sure JSON-Server is running on
      }
      // Re-throw other errors
      throw error
    }
  },
}

// Transactions API
export const transactionsAPI = {
  getAll: async (userId) => {
    // Normalize userId to handle both string and number
    const normalizedUserId = typeof userId === 'string' ? parseInt(userId) || userId : userId

    const response = await api.get('/transactions')
    // Filter transactions for this user (JSON-Server query might not work with mixed types)
    const allTransactions = response.data
    return allTransactions.filter(
      (t) =>
        t.userId === normalizedUserId ||
        t.userId === userId ||
        String(t.userId) === String(userId)
    )
  },

  getById: async (id) => {
    const response = await api.get(`/transactions/${id}`)
    return response.data
  },

  create: async (transaction) => {
    // Ensure userId is consistent
    const transactionData = {
      ...transaction,
      userId: String(transaction.userId), // Store as string for consistency

  update: async (id, transaction) => {
    // Ensure userId is consistent
    const transactionData = {
      ...transaction,
      userId: String(transaction.userId), // Store as string for consistency
    }
    const response = await api.put(`/transactions/${id}`, transactionData)
    return response.data
  },

  delete: async (id) => {
    await api.delete(`/transactions/${id}`)
  },

  filter: async (userId, filters) => {
    // Normalize userId to handle both string and number
    const normalizedUserId = typeof userId === 'string' ? parseInt(userId) || userId : userId

    const response = await api.get('/transactions')
    let allTransactions = response.data

    // Filter by userId first
    allTransactions = allTransactions.filter(
      (t) =>
        t.userId === normalizedUserId ||
        t.userId === userId ||
        String(t.userId) === String(userId)
```



```
    // Apply additional filters
    if (filters.type) {
      allTransactions = allTransactions.filter((t) => t.type === filters.type)
    }
    if (filters.category) {
      allTransactions = allTransactions.filter((t) => t.category === filters.category)
    }
    if (filters.search) {
      const searchLower = filters.search.toLowerCase()
      allTransactions = allTransactions.filter(
        (t) =>
          t.title?.toLowerCase().includes(searchLower) ||
          t.description?.toLowerCase().includes(searchLower) ||
          t.category?.toLowerCase().includes(searchLower)
      )
    }

    return allTransactions
  },
}

// Budget API
export const budgetAPI = {
  getByUser: async (userId) => {
    // Normalize userId to handle both string and number
    const normalizedUserId = typeof userId === 'string' ? parseInt(userId) || userId : userId

    const response = await api.get('/budgets')
    const allBudgets = response.data
    return allBudgets.filter(
      (b) =>
        b.userId === normalizedUserId ||
        b.userId === userId ||
        String(b.userId) === String(userId)
    )
  },

  getByMonth: async (userId, month) => {
    // Normalize userId to handle both string and number
    const normalizedUserId = typeof userId === 'string' ? parseInt(userId) || userId : userId

    const response = await api.get('/budgets')
    const allBudgets = response.data
    const userBudgets = allBudgets.filter(
      (b) =>
        (b.userId === normalizedUserId ||
         b.userId === userId ||
         String(b.userId) === String(userId)) &&
        b.month === month
    )
    return userBudgets.length > 0 ? userBudgets[0] : null
  },

  update: async (id, budget) => {
    // Ensure userId is consistent
    const budgetData = {
      ...budget,
      userId: String(budget.userId), // Store as string for consistency
    }
    const response = await api.put(`/budgets/${id}`, budgetData)
    return response.data
  },

  delete: async (id) => {
    await api.delete(`/budgets/${id}`)
  },
}

// Categories API
export const categoriesAPI = {
  getAll: async () => {
    const response = await api.get('/categories')
    // Map category objects to just names for backward compatibility
    return response.data.map(cat => typeof cat === 'string' ? cat : cat.name)
  },
}

export default api
```

Code Description:

- Imports axios for making HTTP requests to JSON Server
- Defines BASE_URL constant pointing to JSON Server endpoint (http://localhost:3001)
- Creates axios instance with default configuration
- Exports transactionAPI object with methods: getAll(), getById(id), create(data), update(id, data), delete(id), getByUserId(userId)
- Exports budgetAPI object with methods for budget CRUD operations

- Exports categoryAPI object with methods to fetch all categories
- Exports userAPI object with login(credentials) method for authentication
- Each function returns a promise resolving to API response data
- Implements error handling with try-catch blocks
- Uses async/await syntax for cleaner asynchronous code
- Exports api object as default export for import in components

Milestone 6 : Dashboard Components

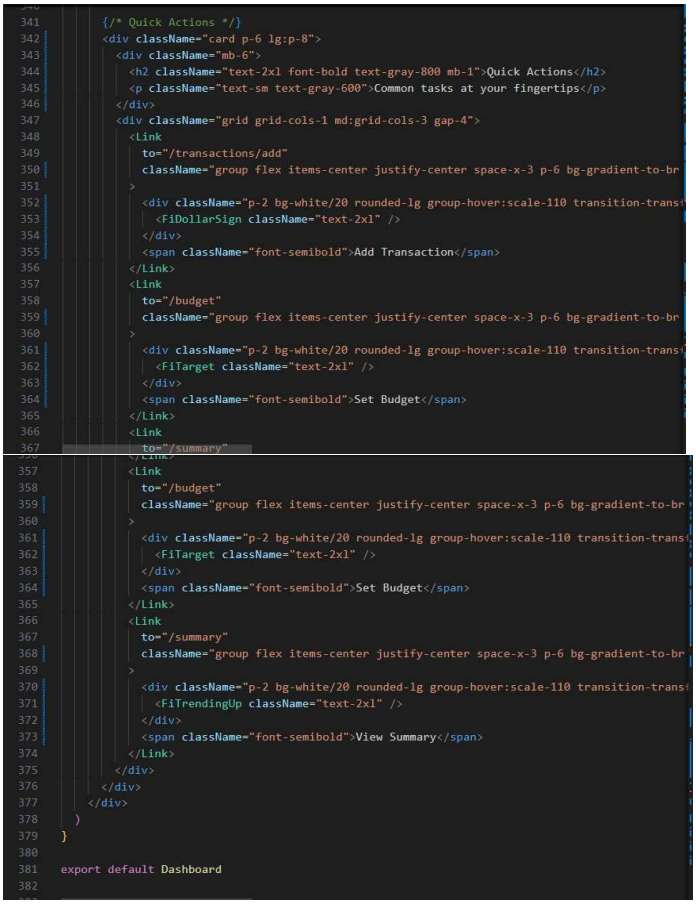
Dashboard.jsx:

```jsx
import { useState, useEffect } from 'react'
import { Link } from 'react-router-dom'
import { transactionsAPI, budgetAPI } from '../services/api'
import { syncDataWithJSONServer } from '../utils/dataInitializer'
import { PieChart, Pie, Cell, BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend, Re
import { FiTrendingUp, FiTrendingDown, FiDollarSign, FiTarget, FiArrowRight } from 'react-ico

const Dashboard = ({ user }) => {
  const [transactions, setTransactions] = useState([])
  const [budget, setBudget] = useState(null)
  const [loading, setLoading] = useState(true)
  const [stats, setStats] = useState({
    totalIncome: 0,
    totalExpense: 0,
    balance: 0,
    monthlySpent: 0,
  })

  const currentMonth = new Date().toISOString().slice(0, 7)

  useEffect(() => {
    loadData()
    // Initialize data from external APIs on component mount
    initializeAPIData()
  }, [user])

  const initializeAPIData = async () => {
    try {
  const loadData = async () => {
    try {
      const [transactionsData, budgetData] = await Promise.all([
        transactionsAPI.getAll(user.id),
        budgetAPI.getByMonth(user.id, currentMonth),
      ])

      setTransactions(transactionsData)
      setBudget(budgetData)

      const currentMonthTransactions = transactionsData.filter(
        (t) => t.date.startsWith(currentMonth)
      )

      const income = currentMonthTransactions
        .filter((t) => t.type === 'income')
        .reduce((sum, t) => sum + t.amount, 0)

      const expense = currentMonthTransactions
        .filter((t) => t.type === 'expense')
        .reduce((sum, t) => sum + t.amount, 0)

      setStats({
        totalIncome: income,
        totalExpense: expense,
        balance: income - expense,
        monthlySpent: expense,
```

```
  const categoryData = transactions
    .filter((t) => t.type === 'expense' && t.date.startsWith(currentMonth))
    .reduce((acc, t) => {
      acc[t.category] = (acc[t.category] || 0) + t.amount
      return acc
    }, {})

  const pieData = Object.entries(categoryData).map(([name, value]) => ({
    name,
    value: parseFloat(value.toFixed(2)),
  }))

  const COLORS = ['#6366f1', '#8b5cf6', '#ec4899', '#f59e0b', '#10b981', '#3b82f

  const recentTransactions = transactions
    .sort((a, b) => new Date(b.date) - new Date(a.date))
    .slice(0, 5)

  const budgetRemaining = budget ? budget.amount - stats.monthlySpent : 0
  const budgetPercentage = budget ? (stats.monthlySpent / budget.amount) * 100 : 0

  if (loading) {
    return (
      <div className="flex justify-center items-center h-64">
        <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-indigo-600"></d
      </div>
    )
  }

  return (
    <div className="space-y-8">
      {/* Header */}
      <div className="flex flex-col sm:flex-row justify-between items-start sm:items-center g
        <div>
          <h1 className="text-4xl font-bold bg-gradient-to-r from-indigo-600 to-purple-600 bg
            Dashboard
          </h1>
          <p className="text-gray-600 mt-1">Overview of your financial activity</p>
        </div>
        <div className="px-4 py-2 bg-white rounded-lg border border-gray-200 shadow-sm">
          <span className="text-sm font-semibold text-gray-700">
            {new Date().toLocaleDateString('en-US', { month: 'long', year: 'numeric' })}
          </span>
        </div>
      </div>

      {/* Stats Cards */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
        <div className="stat-card group">
          <div className="flex items-center justify-between relative z-10">
            <div>
              <p className="text-gray-600 text-sm font-medium mb-1">Total Income</p>
              <p className="text-3xl font-bold text-green-600">
                ${stats.totalIncome.toFixed(2)}
              </p>
            </div>
            <div className="p-3 bg-gradient-to-br from-green-100 to-emerald-100 rounded-xl gr
              <FiTrendingUp className="text-3xl text-green-600" />
            </div>
          </div>
        </div>

        <div className="stat-card group">
          <div className="flex items-center justify-between relative z-10">
            <div>
              <p className="text-gray-600 text-sm font-medium mb-1">Total Expenses</p>
              <p className="text-3xl font-bold text-red-600">
                ${stats.totalExpense.toFixed(2)}
              </p>
            </div>
            <div className="p-3 bg-gradient-to-br from-red-100 to-rose-100 rounded-xl group-ho
              <FiTrendingDown className="text-3xl text-red-600" />
            </div>
          </div>
        </div>

        <div className="stat-card group">
          <div className="flex items-center justify-between relative z-10">
            <div>
              <p className="text-gray-600 text-sm font-medium mb-1">Balance</p>
              <p className={`text-3xl font-bold ${stats.balance >= 0 ? 'text-green-600' : 'te
                ${stats.balance.toFixed(2)}
              </p>
```

```
        <div className="stat-card group">
          <div className="flex items-center justify-between relative z-10">
            <div>
              <p className="text-gray-600 text-sm font-medium mb-1">Budget Remaining</p>
              <p className={`text-3xl font-bold ${budgetRemaining >= 0 ? 'text-green-600' : '
                ${budgetRemaining.toFixed(2)}
              </p>
            </div>
            <div className="p-3 bg-gradient-to-br from-purple-100 to-pink-100 rounded-xl grou
              <FiTarget className="text-3xl text-purple-600" />
            </div>
          </div>
        </div>
      </div>

      {/* Budget Progress */}
      {budget && (
        <div className="card p-8">
          <div className="flex justify-between items-center mb-6">
            <div>
              <h2 className="text-2xl font-bold text-gray-800 mb-1">Monthly Budget Progress</h
              <p className="text-sm text-gray-600">Track your spending against your budget</p
            </div>
            <Link
              to="/budget"
              className="flex items-center space-x-2 px-4 py-2 bg-indigo-50 text-indigo-600 r
            >
              <span>Manage Budget</span>
              <FiArrowRight className="text-lg" />
            </Link>
          </div>
          <div className="space-y-4">
            <div className="flex justify-between items-center">
              <div>
                <p className="text-sm text-gray-600 mb-1">Spent this month</p>
                <p className="text-2xl font-bold text-gray-800">${stats.monthlySpent.toFixed(
              </div>
              <div className="text-right">
                <p className="text-sm text-gray-600 mb-1">Budget limit</p>
                <p className="text-2xl font-bold text-indigo-600">${budget.amount.toFixed(2)}
              </div>
            </div>
            <div className="relative">
              <div className="w-full bg-gray-100 rounded-full h-6 overflow-hidden">
                <div
                  className={`h-6 rounded-full transition-all duration-500 flex items-center
                    budgetPercentage > 100
                      ? 'bg-gradient-to-r from-red-500 to-red-600'
                      : budgetPercentage > 80
                      ? 'bg-gradient-to-r from-yellow-500 to-orange-500'
                      : 'bg-gradient-to-r from-green-500 to-emerald-500'
                  }`}
                  style={{ width: `${Math.min(budgetPercentage, 100)}%` }}
                >
                  {budgetPercentage > 10 && (
                    <span className="text-xs font-semibold text-white">
                      {budgetPercentage.toFixed(0)}%
                    </span>
                  )}
                </div>
              </div>
              <p className="text-sm text-gray-600 mt-2">
                {budgetPercentage > 100
                  ? `⚠️ Exceeded by ${(budgetPercentage - 100).toFixed(1)}%`
                  : `${(100 - budgetPercentage).toFixed(1)}% remaining`}
              </p>
            </div>
          </div>
        </div>
      )}

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        {/* Category Chart */}
        <div className="card p-6 lg:p-8">
          <div className="flex justify-between items-center mb-6">
            <div>
              <h2 className="text-2xl font-bold text-gray-800 mb-1">Expenses by Category</h2>
              <p className="text-sm text-gray-600">Breakdown of your spending</p>
            </div>
          </div>
          {pieData.length > 0 ? (
```

```
          {pieData.length > 0 ? (
            <ResponsiveContainer width="100%" height={300}>
              <PieChart>
                <Pie
                  data={pieData}
                  cx="50%"
                  cy="50%"
                  labelLine={false}
                  label={({ name, percent }) => `${name} ${(percent * 100).toFixed(0)}%`}
                  outerRadius={80}
                  fill="#8884d8"
                  dataKey="value"
                >
                  {pieData.map((entry, index) => (
                    <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />
                  ))}
                </Pie>
                <Tooltip />
              </PieChart>
            </ResponsiveContainer>
          ) : (
            <p className="text-gray-500 text-center py-8">No expense data for this month</p>
          )}
        </div>

        {/* Recent Transactions */}
        <div className="card p-6 lg:p-8">
          <div>
            <h2 className="text-2xl font-bold text-gray-800 mb-1">Recent Transactions</h2>
            <p className="text-sm text-gray-600">Your latest financial activity</p>
          </div>
          <Link
            to="/transactions"
            className="flex items-center space-x-2 px-4 py-2 bg-indigo-50 text-indigo-600 r
          >
            <span>View All</span>
            <FiArrowRight className="text-lg" />
          </Link>
        </div>
        <div className="space-y-3">
          {recentTransactions.length > 0 ? (
            recentTransactions.map((transaction) => (
              <div
                key={transaction.id}
                className="flex items-center justify-between p-4 bg-gradient-to-r from-gray-
              >
                <div className="flex items-center space-x-4">
                  <div className={`p-2.5 rounded-lg ${
                    transaction.type === 'income'
                      ? 'bg-green-100'
                      : 'bg-red-100'
                  } group-hover:scale-110 transition-transform duration-300`}>
                    {transaction.type === 'income' ? (
                      <FiTrendingUp className={`text-xl ${transaction.type === 'income' ? '
                    <div>
                      <p className="font-semibold text-gray-800">{transaction.title}</p>
                      <div className="flex items-center space-x-2 mt-1">
                        <span className="text-xs px-2 py-0.5 bg-indigo-100 text-indigo-700 r
                          {transaction.category}
                        </span>
                        <span className="text-xs text-gray-500">
                          {new Date(transaction.date).toLocaleDateString()}
                        </span>
                      </div>
                    </div>
                  </div>
                </div>
                <div className="text-right">
                  <p
                    className={`text-lg font-bold ${
                      transaction.type === 'income' ? 'text-green-600' : 'text-red-600'
                    }`}
                  >
                    {transaction.type === 'income' ? '+' : '-'}${transaction.amount.toFixe
                  </p>
                </div>
              </div>
            ))
          ) : (
            <div className="text-center py-12">
              <FiDollarSign className="text-4xl text-gray-300 mx-auto mb-3" />
```



```
      {/* Quick Actions */}
      <div className="card p-6 lg:p-8">
        <div className="mb-6">
          <h2 className="text-2xl font-bold text-gray-800 mb-1">Quick Actions</h2>
          <p className="text-sm text-gray-600">Common tasks at your fingertips</p>
        </div>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
          <Link
            to="/transactions/add"
            className="group flex items-center justify-center space-x-3 p-6 bg-gradient-to-br
          >
            <div className="p-2 bg-white/20 rounded-lg group-hover:scale-110 transition-transf
              <FiDollarSign className="text-2xl" />
            </div>
            <span className="font-semibold">Add Transaction</span>
          </Link>
          <Link
            to="/budget"
            className="group flex items-center justify-center space-x-3 p-6 bg-gradient-to-br
          >
            <div className="p-2 bg-white/20 rounded-lg group-hover:scale-110 transition-transf
              <FiTarget className="text-2xl" />
            </div>
            <span className="font-semibold">Set Budget</span>
          </Link>
          <Link
            to="/summary"
```



```
          </Link>
          <Link
            to="/budget"
            className="group flex items-center justify-center space-x-3 p-6 bg-gradient-to-br
          >
            <div className="p-2 bg-white/20 rounded-lg group-hover:scale-110 transition-transf
              <FiTarget className="text-2xl" />
            </div>
            <span className="font-semibold">Set Budget</span>
          </Link>
          <Link
            to="/summary"
            className="group flex items-center justify-center space-x-3 p-6 bg-gradient-to-br
          >
            <div className="p-2 bg-white/20 rounded-lg group-hover:scale-110 transition-transf
              <FiTrendingUp className="text-2xl" />
            </div>
            <span className="font-semibold">View Summary</span>
          </Link>
        </div>
      </div>
    </div>
  )
}

export default Dashboard
```

Code Description:

- Imports SummaryCard, BudgetStatus, and CategoryChart components
- Imports useApp hook to access financial data from context
- Calculates summary metrics: total income, total expenses, balance
- Renders grid layout with four SummaryCard components displaying key financial indicators
- Each SummaryCard receives props: title, amount, icon, and color for visual representation
- Includes BudgetStatus component showing monthly budget progress with visual progress bar
- Displays CategoryChart component rendering pie chart for expense distribution
- Shows recent transactions list with quick links to transaction management
- Implements responsive grid layout using TailwindCSS classes
- Exports Dashboard component as default export

SummaryCard.jsx:

Code Description:

- Creates a reusable card component for displaying financial metrics

- Accepts props: title, amount, icon, color for customization
- Formats currency using Indian Rupee notation with proper thousand separators
- Renders card with icon, title, and formatted amount
- Applies color-coded styling based on card type (income: green, expense: red, balance: blue, budget: purple)
- Uses TailwindCSS for card styling with shadow effects and rounded corners
- Implements hover effects for better interactivity
- Exports SummaryCard as default export

BudgetStatus.jsx:

Code Description:

- Imports useApp hook to access budget and expense data
- Calculates budget utilization percentage from total expenses and monthly budget
- Renders progress bar with color-coded zones: green (0-60%), yellow (60-80%), red (80-100%)
- Displays remaining budget amount and percentage
- Shows warning message when budget exceeds 80% utilization
- Calculates days remaining in current month
- Uses TailwindCSS for styling progress bar and alert messages
- Exports BudgetStatus component as default export

CategoryChart.jsx:

Code Description:

- Imports PieChart, Pie, Cell, ResponsiveContainer, Legend, Tooltip from recharts library
- Imports useApp hook to access category-wise expense data
- Prepares data array by mapping categories with their total expenses
- Defines custom color array for different pie chart segments
- Renders ResponsiveContainer wrapping PieChart for responsive behavior
- Configures Pie component with data, dataKey, nameKey properties
- Maps Cell components with custom colors for each category
- Implements custom Tooltip showing category name, amount, and percentage
- Displays Legend for category identification
- Handles empty state when no expense data exists
- Exports CategoryChart as default export

Milestone 7 : Transaction Management

TransactionForm.jsx:

```jsx
import { useState, useEffect } from 'react'
import { Link } from 'react-router-dom'
import { transactionsAPI, categoriesAPI } from '../services/api'
import { toast } from 'react-toastify'
import { FiPlus, FiEdit, FiTrash2, FiFilter, FiSearch } from 'react-icons/fi'

const Transactions = ({ user }) => {
  const [transactions, setTransactions] = useState([])
  const [filteredTransactions, setFilteredTransactions] = useState([])
  const [categories, setCategories] = useState([])
  const [loading, setLoading] = useState(true)
  const [filters, setFilters] = useState({
    type: '',
    category: '',
    search: '',
  })

  useEffect(() => {
    loadData()
  }, [user])

  useEffect(() => {
    applyFilters()
  }, [transactions, filters])

  const loadData = async () => {
    try {
      const [transactionsData, categoriesData] = await Promise.all([
```

```jsx
  const applyFilters = () => {
    let filtered = [...transactions]

    if (filters.type) {
      filtered = filtered.filter((t) => t.type === filters.type)
    }

    if (filters.category) {
      filtered = filtered.filter((t) => t.category === filters.category)
    }

    if (filters.search) {
      const searchLower = filters.search.toLowerCase()
      filtered = filtered.filter(
        (t) =>
          t.title.toLowerCase().includes(searchLower) ||
          t.description?.toLowerCase().includes(searchLower) ||
          t.category.toLowerCase().includes(searchLower)
      )
    }

    setFilteredTransactions(filtered.sort((a, b) => new Date(b.date) - new Date(a.date)))
  }

  const handleDelete = async (id) => {
    if (window.confirm('Are you sure you want to delete this transaction?')) {
      try {
```

```jsx

  const handleFilterChange = (e) => {
    setFilters({
      ...filters,
      [e.target.name]: e.target.value,
    })
  }

  const clearFilters = () => {
    setFilters({
      type: '',
      category: '',
      search: '',
    })
  }

  if (loading) {
    return (
      <div className="flex justify-center items-center h-64">
        <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-indigo-600"></d
      </div>
    )
  }

  return (
    <div className="space-y-8">
      <div className="flex flex-col md:flex-row justify-between items-start md:items-center g
```

```jsx
    <div className="space-y-6">
      <div className="flex flex-col md:flex-row justify-between items-start md:items-center g
        <div>
          <h1 className="text-4xl font-bold bg-gradient-to-r from-indigo-600 to-purple-600 bg
            Transactions
          </h1>
          <p className="text-gray-600 mt-1">Manage your income and expenses</p>
        </div>
        <Link
          to="/transactions/add"
          className="btn-primary flex items-center space-x-2"
        >
          <FiPlus className="text-lg" />
          <span>Add Transaction</span>
        </Link>
      </div>

      {/* Filters */}
      <div className="card p-6 lg:p-8">
        <div className="flex items-center space-x-2 mb-6">
          <div className="p-2 bg-indigo-100 rounded-lg">
            <FiFilter className="text-indigo-600 text-lg" />
          </div>
          <h2 className="text-xl font-bold text-gray-800">Filters</h2>
        </div>
        <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
          <div className="relative">
            <FiSearch className="absolute left-4 top-1/2 transform -translate-y-1/2 text-gray
            <input
              type="text"
              name="search"
              value={filters.search}
              onChange={handleFilterChange}
              placeholder="Search transactions..."
              className="input-field pl-12"
            />
          </div>
          <select
            name="type"
            value={filters.type}
            onChange={handleFilterChange}
            className="input-field"
          >
            <option value="">All Types</option>
            <option value="income">Income</option>
            <option value="expense">Expense</option>
          </select>
          <select
            name="category"
            value={filters.category}
            onChange={handleFilterChange}
            className="input-field"
          >
            <option value="">All Categories</option>
            {categories.map((cat) => (
              <option key={cat} value={cat}>
            {categories.map((cat) => (
              <option key={cat} value={cat}>
                {cat}
              </option>
            ))}
          </select>
          <button
            onClick={clearFilters}
            className="btn-secondary"
          >
            Clear Filters
          </button>
        </div>
      </div>

      {/* Transactions List */}
      <div className="card overflow-hidden p-0">
        <div className="overflow-x-auto">
          <table className="w-full">
            <thead className="bg-gray-50">
              <tr>
                <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercas
                  Title
                </th>
                <th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercas
                  Category
                </th>
```

```
<tbody className="bg-white divide-y divide-gray-200">
  {filteredTransactions.length > 0 ? (
    filteredTransactions.map((transaction) => (
      <tr key={transaction.id} className="hover:bg-gray-50">
        <td className="px-6 py-4 whitespace-nowrap">
          <div>
            <div className="text-sm font-medium text-gray-900">{transaction.title}
            {transaction.description && (
              <div className="text-sm text-gray-500">{transaction.description}</d
            )}
          </div>
        </td>
        <td className="px-6 py-4 whitespace-nowrap">
          <span className="px-2 inline-flex text-xs leading-5 font-semibold round
            {transaction.category}
          </span>
        </td>
        <td className="px-6 py-4 whitespace-nowrap">
          <span
            className={`px-2 inline-flex text-xs leading-5 font-semibold rounded-
              transaction.type === 'income'
                ? 'bg-green-100 text-green-800'
                : 'bg-red-100 text-red-800'
            }`}
          >
            {transaction.type}
          </span>
        </td>
        <td className="px-6 py-4 whitespace-nowrap">
          <span
            className={`text-sm font-medium ${
              transaction.type === 'income' ? 'text-green-600' : 'text-red-600'
            }`}
          >
            {transaction.type === 'income' ? '+' : '-'}${transaction.amount.toFix
          </span>
        </td>
        <td className="px-6 py-4 whitespace-nowrap text-sm text-gray-500">
          {new Date(transaction.date).toLocaleDateString()}
        </td>
        <td className="px-6 py-4 whitespace-nowrap text-sm font-medium">
          <div className="flex space-x-2">
            <Link
              to={`/transactions/edit/${transaction.id}`}
              className="text-indigo-600 hover:text-indigo-900"
            >
              <FiEdit />
            </Link>
            <button
              onClick={() => handleDelete(transaction.id)}
              className="text-red-600 hover:text-red-900"
            >
              <FiTrash2 />
```

```
{/* Summary */}
{filteredTransactions.length > 0 && (
  <div className="card p-6 lg:p-8">
    <h2 className="text-2xl font-bold text-gray-800 mb-6">Summary</h2>
    <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
      <div>
        <p className="text-sm text-gray-600">Total Income</p>
        <p className="text-2xl font-bold text-green-600">
          $
          {filteredTransactions
            .filter((t) => t.type === 'income')
            .reduce((sum, t) => sum + t.amount, 0)
            .toFixed(2)}
        </p>
      </div>
      <div>
        <p className="text-sm text-gray-600">Total Expenses</p>
        <p className="text-2xl font-bold text-red-600">
          $
          {filteredTransactions
            .filter((t) => t.type === 'expense')
            .reduce((sum, t) => sum + t.amount, 0)
            .toFixed(2)}
        </p>
      </div>
      <div>
        <p className="text-sm text-gray-600">Net Balance</p>
```

```
            <p className="text-sm text-gray-600">Net Balance</p>
            <p
              className={`text-2xl font-bold ${
                filteredTransactions.reduce((sum, t) => {
                  return sum + (t.type === 'income' ? t.amount : -t.amount)
                }, 0) >= 0
                  ? 'text-green-600'
                  : 'text-red-600'
              }`}
            >
              $
              {filteredTransactions
                .reduce((sum, t) => {
                  return sum + (t.type === 'income' ? t.amount : -t.amount)
                }, 0)
                .toFixed(2)}
            </p>
          </div>
        </div>
      </div>
    )}
  </div>
 )
}

export default Transactions
```

Code Description:

- Imports useState and useApp hooks for state and context management
- Creates controlled form with fields: type (income/expense), category, amount, description, date
- Validates form inputs before submission: ensures amount is positive, category is selected, date is valid
- Category dropdown dynamically filtered based on transaction type selection
- Amount input accepts only numeric values with proper validation
- Date picker defaults to current date and prevents future date selection
- Implements handleSubmit function that calls addTransaction or updateTransaction from context
- Shows success/error toast notifications after submission
- Resets form fields after successful transaction creation
- Uses TailwindCSS for form styling with proper spacing and layout
- Exports TransactionForm as default export

TransactionList.jsx:

Code Description:

- Imports useApp hook to access transactions array from context
- Implements table layout for displaying transaction history with columns: date, type, category, description, amount, actions
- Maps through transactions array and renders TransactionItem for each transaction
- Implements filtering functionality by date range, category, and transaction type using state
- Adds search input for filtering transactions by description
- Sorts transactions by date in descending order (newest first)
- Implements pagination for large transaction lists with configurable items per page
- Shows empty state message when no transactions exist or match filters
- Color codes transaction types: green text for income, red text for expense

- Includes action buttons for editing and deleting transactions
- Exports TransactionList as default export

TransactionItem.jsx:

Code Description:

- Creates individual table row component for each transaction
- Accepts transaction object and callback functions (onEdit, onDelete) as props
- Displays formatted transaction date using date formatting utility
- Shows transaction type badge with color coding
- Displays category with icon
- Formats amount as Indian Rupee currency with proper notation
- Implements edit button that triggers onEdit callback with transaction data
- Implements delete button with confirmation dialog before deletion
- Applies hover effects on table row for better user experience
- Uses TailwindCSS utility classes for styling
- Exports TransactionItem as default export

Project Execution

Running the Application:

After completing the code, run the React application and JSON Server:

Start JSON Server:

json-server --watch db.json --port 3001

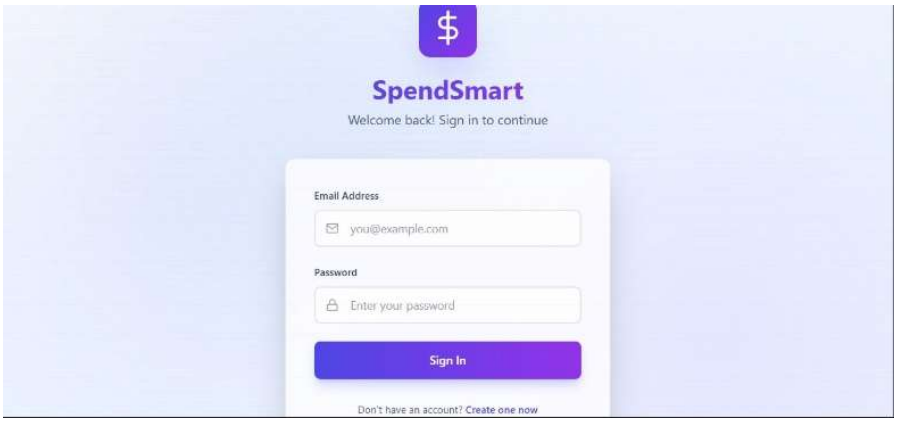
Start React Development Server:

npm run dev

Access the application at http://localhost:5173

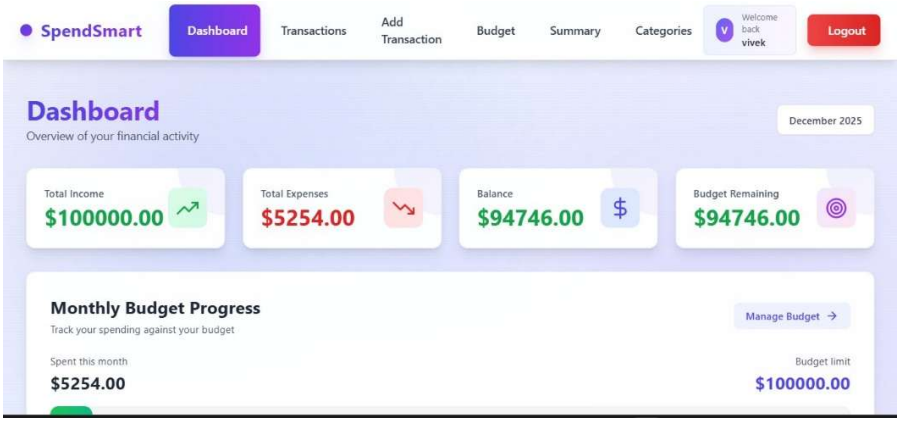
Application Screenshots

Login Page



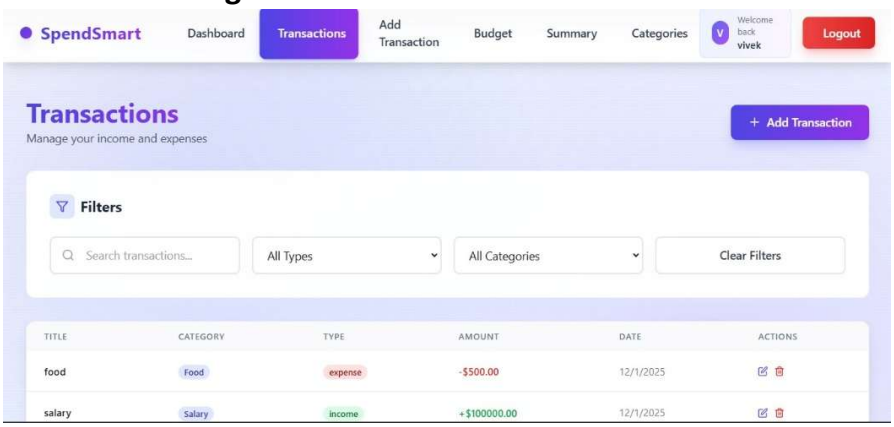
User authentication interface with username and password fields

Dashboard



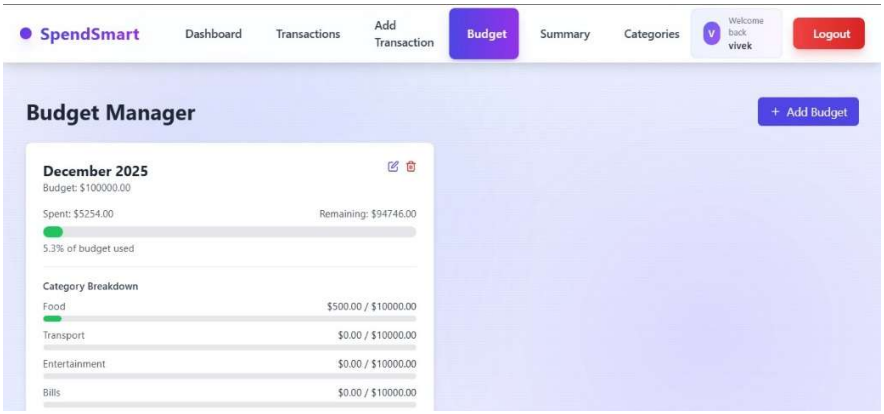
Comprehensive overview showing total income, expenses, balance, budget status, and category-wise expense distribution with pie chart

Transactions Page



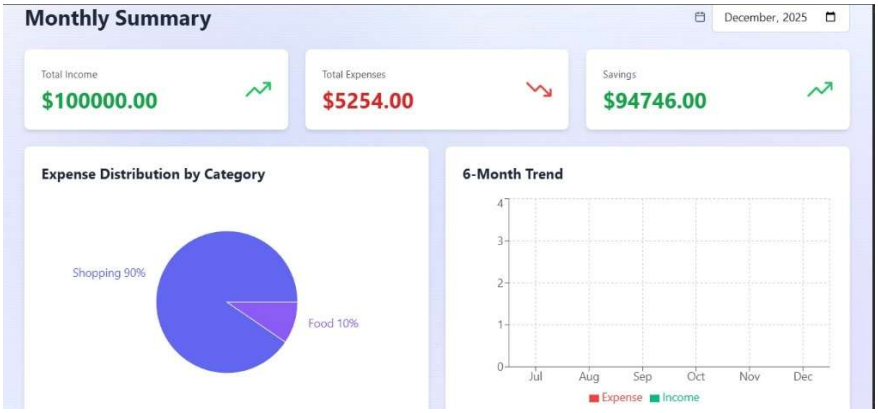
Complete transaction management with add/edit forms, transaction list table, and filtering options

Budget Page



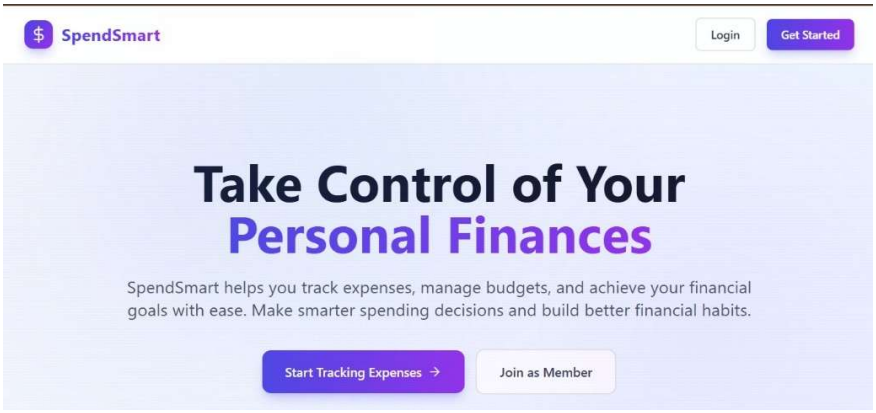
Budget planning interface showing total monthly budget, category-wise allocation, and budget utilization status

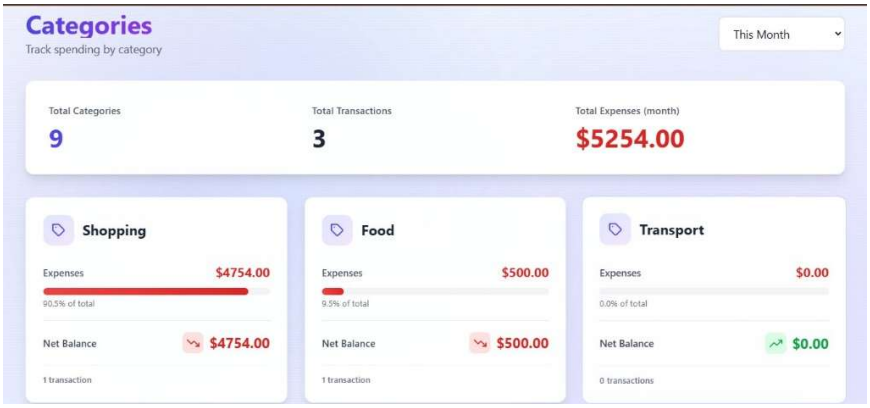
Analytics Page



Visual analytics with multiple charts: pie chart for category distribution, bar chart for income vs expense, and line graph for spending trends

Project Demo





Experience SpendSmart in action and explore all its features through the comprehensive demo.

Happy Coding!!