Final Project Report

1. **Introduction**
    1.1. Project title
    1.2. Objectives
2. **Project Overview**
    1.3. Define Problem Statement
    1.4. Scenario based case-study
3. **Architecture**
    3.1 Frontend
    3.2 Backend
    3.3 Database
4. **ER - Diagram**
    a. User Entity
    b. Expense Entity
5. **Setup Instructions**
    a. Prerequisites
    b. Installation
6. **Folder Structure**
    a. Project structure
    b. Application flow
7. **Project flow**
    7.1 Project setup and configuration
    7.2 Backend development

Spend Smart – Personal Expense Tracker: Your Personal Finance Companion

1.Introduction

1.1Project title: Personal Expense Tracker:

The Expense Tracker app offers a comprehensive solution for users to manage their finances efficiently. With a user-friendly interface, users can easily record their expenses, categorize them, and set budget limits. The app provides real-time notifications to alert users when they approach or exceed their budget limits. Additionally, users can generate detailed reports to analyse their spending patterns and make informed financial decisions. The app prioritizes security and privacy, ensuring users' financial data remains protected. With robust backend functionalities, including user authentication and database management, the Expense Tracker app offers a seamless and hassle-free experience for users to take control of their finances.

With a user-friendly interface and intuitive design, our expense tracker app allows you to effortlessly record your expenses, categorize them, and analyse your spending patterns. You can easily add new transactions, specify the category, and add relevant details such as date, vendor, and payment method.

We prioritize user satisfaction and aim to provide a seamless and hassle-free experience for managing your finances. Our app enables you to set budgets, track your spending against those budgets, and receive notifications when you exceed your limits. You can also generate reports and visualize your expenses through graphs and charts for better financial insights.

For business users, our app offers robust backend functionalities. You can manage multiple accounts, track expenses for different projects or departments, and generate detailed expense reports for accounting purposes. Administrators can efficiently handle user inquiries, ensure data accuracy, and monitor the overall performance of the app.

Our expense tracker app prioritizes your security and privacy. We employ encryption and password protection to safeguard your financial data, ensuring

confidentiality. With our secure platform, you can manage expenses with peace of mind.

We are excited to have you on board and look forward to providing you with a reliable and user- friendly expense tracking experience. Start tracking your expenses with our app today and take control of your finances.

1.2 Objectives:

- **Purpose:**
  The Money Tracker App helps users monitor and manage their personal finances by logging income and expenses. It aims to simplify money management through an intuitive interface, real-time balance updates, and a responsive design.

- **Features:** a) Real-time balance tracking
  b) Add and manage transactions (income and expense)
  c) Transaction history logs
  d) Theming support with light/dark mode
  e) RESTful API for CRUD operations on transactions

2.Project Overview:

2.1 Define Problem Statement:

To create an expense tracking application with features of real-time tracking, adding & managing transactions (income and expense), creating logs, RESTful API for CRUD operations support on transactions.

2.2 Scenario Based Case-study:

Scenario 1: Personal Finance Management

User Profile: Alice - Working Professional

Scenario Overview: Alice, a working professional, aims to take control of her finances using the Expense Tracker app. She diligently tracks her expenses, sets budget limits, and generates reports to analyse her spending patterns.

Key Actions:

1. **Registration and Login:**
    - Alice downloads the Expense Tracker app and completes the registration process.
    - Upon successful registration, she logs in to the app and accesses the dashboard.
2. **Expense Tracking:**
    - Alice starts adding her expenses for the month into the app's interface.
    - She categorizes expenses into groceries, utilities, entertainment, transportation, etc.
    - Alice monitors her spending and receives notifications when approaching or exceeding budget limits.
3. **Budget Analysis:**
    - Throughout the month, Alice continues tracking expenses and adjusting her budget.
    - At the month-end, she generates a report to analyze spending patterns.
    - Alice identifies areas to cut back on expenses and sets new financial goals for the next month.

Scenario 2: Business Expense Management

User Profile: Bob - Small Business Owner

Scenario Overview: Bob, a small business owner, utilizes the Expense Tracker app to manage his company's expenses efficiently. He sets up categories, tracks employee expenses, and generates detailed reports for accounting purposes.

Key Actions:

1. **Account Setup:**
    - Bob creates an account for his business on the Expense Tracker app.
    - He logs in and accesses the dashboard to set up expense categories and budget limits.

2. **Expense Tracking for Employees:**
    - Bob encourages his employees to use the app to record expenses.
    - Employees add expenses, specify categories, and upload receipts or invoices.
3. **Backend Functionality:**
    - As the administrator, Bob has access to backend functionalities.
    - He can view all expense records, track spending for different projects/departments, and generate reports.

Scenario 3: Student Budgeting

User Profile: Charlie - College Student

Scenario Overview: Charlie, a college student, adopts the Expense Tracker app to manage his finances responsibly while studying. He categorizes expenses, sets budget limits, and utilizes notifications to avoid overspending.

Key Actions:

1. **Expense Categorization and Budgeting:**
    - Charlie categorizes expenses into textbooks, dining out, transportation, etc.
    - He sets realistic budget limits based on his monthly allowance and academic expenses.
2. **Expense Monitoring:**
    - Throughout the month, Charlie records expenses promptly using the app.
    - He receives notifications when approaching or exceeding budget limits, helping him prioritize expenses.
3. **Financial Analysis and Adjustment:**
    - At the end of the month, Charlie reviews spending patterns and expense reports.

- He identifies areas to save money and adjusts his budget for the upcoming month accordingly.

Scenario 4: Family Budget Planning

User Profile: David and Emily - Parents with children

Scenario Overview: David and Emily, a couple with children, seek to organize their family finances effectively using the Expense Tracker app. They aim to manage household expenses, save for future goals, and teach their children financial responsibility.

Key Actions:

1. **Family Account Setup:**
    - David and Emily create a joint account for their family on the Expense Tracker app.
    - They customize the account settings to include categories relevant to their family's expenses, such as groceries, childcare, education, utilities, and family outings.
2. **Expense Tracking for Family Activities:**
    - David and Emily record all family-related expenses, including groceries, children's activities, and household bills, in the app.
    - They involve their children in the process, teaching them the importance of tracking spending and budgeting.
3. **Budget Allocation and Saving Goals:**
    - David and Emily set monthly budgets for each expense category based on their family's financial goals and priorities.
    - They allocate a portion of their income towards savings for emergencies, education funds for their children, and family vacations.
4. **Financial Education for Children:**
    - David and Emily use the Expense Tracker app as a tool to educate their children about money management.

- They involve their children in budget discussions, encourage them to track their own spending for allowances or savings, and discuss the family's financial decisions openly.

5. **Regular Financial Reviews:**
    - At regular intervals, David and Emily review their family's financial status using the app's reporting features.
    - They assess their spending patterns, evaluate progress towards savings goals, and make adjustments to their budget as necessary to ensure financial stability and growth for their family.

3.Architecture:

The technical architecture for an expense tracker typically consists of three main components: the User Interface, the Backend/API, and the Database.

3.1 Frontend (User Interface):

This component is responsible for providing a user-friendly interface for users to interact with the app. It includes features such as input forms for adding expenses, displaying expense lists, and generating charts or visualizations of expense data. The User Interface communicates with the Backend/API to send and receive data.

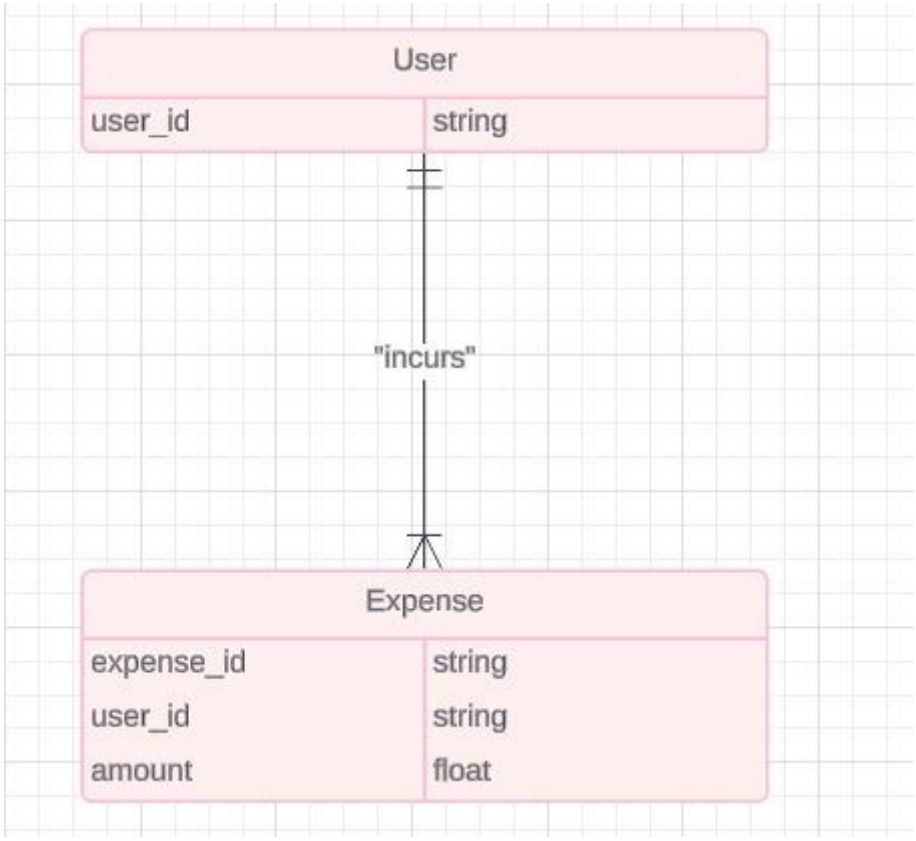
3.2 Backend/API:

The Backend or API serves as the core logic and functionality of the expense tracker app. It handles business operations, data processing, and communication between the User Interface and the Database. The Backend/API includes modules for authentication, expense management, and any other necessary functionalities.

3.3 Database:

The Database stores and manages the data for the expense tracker app. It includes tables or collections to store expense records, user information, and any other relevant data. The Database interacts with the Backend/API to store and retrieve data as needed.

4. ER – Diagram:



User Entity and Expense Entity –

The relationship between the "User" and "Expense" entities is represented by the user_id attribute in the "Expense" entity, which serves as a foreign key referencing the primary key (user_id) of the "User" entity. This indicates that each expense is associated with a specific user.

4.1 User entity:

- user_id (Primary Key): A unique identifier for each user.
- username: The username chosen by the user.
- email: The email address associated with the user.
- password: The password for the user's account.
- created_at: The timestamp indicating when the user account was created.

4.2 Expense entity:

- expense_id: A unique identifier for each expense.
- user_id (Foreign Key): A reference to the user who made the expense.
- amount: The cost or amount spent for the expense.
- category: The category or type of expense (e.g., food, transportation, entertainment).
- date: The date when the expense was incurred.

5.Setup instructions:

5.1 Prerequisites:

To develop a full-stack expense tracker app using React js, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

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

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: https://www.mongodb.com/try/download/community
- Installation instructions: https://docs.mongodb.com/manual/installation/

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

React: React is a JavaScript library for building user interfaces. To create and manage your React project, you can use Create React App, a popular tool for bootstrapping React applications.

Start the development server:

To launch the development server and see your React app in the browser, run the following command:

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.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize React to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

5.2 Installation:

1. Clone the repository –

```
git clone https://github.com/your-username/money-tracker-app.git
cd money-tracker-app
```

2. Set up the backend –

```
cd backend
npm install
```

Create a .env file in the backend folder with –
MONGO_URI=mongodb://localhost:27017/money-tracker

PORT=5000

Start the backend server –

```
npm start
```

3.Set up the frontend –

Navigate to the fronted folder –

```
cd frontend
npm install
```

Start the frontend app –

```
npm run dev
```

6.Project folder Structure:

6.1 Project Structure:

Client:

- **Src/assets –** Contains static assets such as images and logs.
- **Src/Components –** React components including Balance.jsx, ThemeToggle.jsx, TransactionForm.jsx, and TransactionaLogs.jsx.

- **Src/main.jsx –** Entry point for the react app.
- **Vite.config.js –** Configuration for vite.

Server:

- **Models/transaction.js –** Defines the transaction schema using Mongoose.
- **Routes/trasactionRoutes.js –** Defines the API routes for creating, retrieving and deleting transactions.
- **Server.js –** the main entry point for the Node.js server.

6.2 Application flow:

1. **Account Creation and Setup:**
    - User registers for a new account by providing necessary information such as username, email, and password.
    - Upon successful registration, the user sets up and manages personal account details, including profile information like name, contact information, and preferred currency.
2. **Expense Tracking:**
    - User logs into the Expense Tracker app and accesses the dashboard.
    - From the dashboard, the user navigates to the "Add Expense" section.
    - User enters details such as amount, category, and date for each expense incurred.
    - Expenses are categorized appropriately (e.g., groceries, utilities, entertainment) for better organization and analysis.
    - User can edit or delete existing expenses if necessary to maintain accurate records.
    - The app provides real-time updates to ensure users have access to accurate and up-to-date information regarding their expenses.

    Budgeting and Goal Setting:

    - User sets personal budget limits and financial goals for different expense categories (e.g., groceries, dining out, transportation).

- User monitors and tracks expenses against the defined budget using visualizations or progress bars.
- The app sends alerts or notifications to the user when they approach or exceed budget limits, helping them stay on track with their financial goals.

Reporting and Analysis:

- User accesses reports and summaries of expenses from the dashboard.
- User can analyze expenses based on different categories, time periods, or custom filters to gain insights into spending patterns and trends.
- The app provides visual representations such as charts or graphs to facilitate easier analysis and decision-making regarding financial management.

Data Security and Privacy:

- The Expense Tracker app ensures the security and privacy of user data by employing encryption and password protection mechanisms.
- It follows best practices to protect personal financial information and complies with relevant regulations regarding data security and privacy.
- Users are encouraged to use strong passwords and enable two-factor authentication for added security.
- The app has a support or security team that users can contact to report any security concerns or suspicious activities, ensuring prompt resolution and mitigation of potential risks.

Additional Functionality:

- Add Income: Similar to adding an expense, users can add income details, which are saved to the database and associated with the user.
- Delete Month: Users can delete a month, removing all expenses and income records associated with that month for a specific user.
- Download Report: Users can generate and download reports summarizing their expenses, income, or other relevant financial information.

- Add Row/Delete Row: Users can add or delete rows within expense or income records to manage details effectively.

7.Project flow:

7.1 Project setup and configuration:

1.Create project folders and files:

Now, firstly create the folders for frontend and backend to write the respective code and install the essential libraries.

- Client folders.
- Server folders

2. Install required tools and software:

For the backend to function well, we use the libraries mentioned in the prerequisites. Those libraries includes -

- Node.js.
- MongoDB.
- Bcrypt
- Body-parser

Also, for the frontend we use the libraries such as

- React Js.
- Material UI
- Bootstrap
- Axios

7.2 Backend Development:

- **Set Up Project Structure:**
    - Create a new directory for your project and set up a package.json file using npm init command.

    - Install necessary dependencies such as Express.js, Mongoose, and other required packages.
- **Create Express.js Server:**
    - Set up an Express.js server to handle HTTP requests and serve API endpoints.
    - Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.
- **Define API Routes:**
    - Create separate route files for different API functionalities such as authentication, stock actions, and transactions.
    - Implement route handlers using Express.js to handle requests and interact with the database.
- **Implement Data Models:**
    - Define Mongoose schemas for the different data entities like Add, Remove, transactions, deposits and Users.
    - Create corresponding Mongoose models to interact with the MongoDB database.
    - Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.
- **User Authentication:**
    - Implement user authentication using strategies like JSON Web Tokens (JWT) or session-based authentication.
    - Create routes and middleware for user registration, login, and logout.
    - Set up authentication middleware to protect routes that require user authentication.
- **Handle new transactions:**
    - Allow users to make transactions to other users using the user’s account id.
    - Update the transactions and account balance dynamically in real-time.

- **Admin Functionality:**
    - Implement routes and controllers specific to admin functionalities such as fetching all the data regarding users, transactions, stocks and orders.
- **Error Handling:**
    - Implement error handling middleware to catch and handle any errors that occur during the API requests.
    - Return appropriate error responses with relevant error messages and HTTP status codes.

7.3 Database development:

1. **Setup MongoDB:**
    - Install MongoDB locally or use a cloud-based service like MongoDB Atlas.
    - Ensure MongoDB is running and accessible from the application.
2. **Define Schemas:**
    - Create userSchema and expenseSchema using Mongoose.
    - Define the required fields for each schema, such as firstname, lastname, email, and password for users, and userId, monthYear, tables, and calculations for expenses.
3. **Create Models:**
    - Define Mongoose models for User and Expense using the schemas.
    - These models will represent collections in the MongoDB database and provide methods for interacting with the data.
4. **Connect to MongoDB:**
    - Set up a connection to MongoDB using Mongoose in the application's entry point (e.g., app.js or server.js).
    - Use the connection string to connect to the MongoDB database.
5. **Test Connection:**
    - Ensure the application successfully connects to the MongoDB database without any errors.
6. Test the connection by performing basic CRUD operations on the User and Expense models.

7.4 Frontend development:

1. **Setup React Application:**
    - Install required libraries using npm or yarn. This may include libraries for routing, state management (if needed), and any other dependencies.
    - Create the initial application structure with directories for components, containers, styles, and assets.
    - Organize project files for efficient development, ensuring a clear separation of concerns.
2. **Design UI Components:**
    - Create reusable components for interactive elements such as expense listings, input forms, buttons, and notifications.
    - Implement a layout and styling scheme to define the overall look and feel of the application. This should include consistent branding, typography, color palette, and spacing.
    - Design user interface elements with responsiveness in mind, ensuring compatibility across different screen sizes and devices.
    - Integrate a navigation system to allow seamless exploration of different sections of the expense tracker, such as adding expenses, viewing reports, and managing budgets.
3. **Implement Frontend Logic:**
    - Integrate frontend components with backend API endpoints for fetching and updating expense data.
    - Implement data binding to connect user interface elements with the underlying data model.
    - Develop logic for adding, editing, and deleting expenses, including validation to ensure data integrity.
    - Implement features for categorizing expenses, setting budgets, and generating reports based on user preferences.
4. Handle user authentication and authorization to restrict access to certain features or data based on user roles.

8.Running the application:

Frontend –

Start the React app by running the following in the frontend directory –

npm run dev

Backend –

Start the Node.js server by running the following in the backend directory –

npm start

9.Authentication:

Base URL –

https://localhost-5000/api

Endpoints –

a) GET / api / transactions – Fetch all transactions.
b) POST / api / transactions – Create a new transaction.

Example request body:

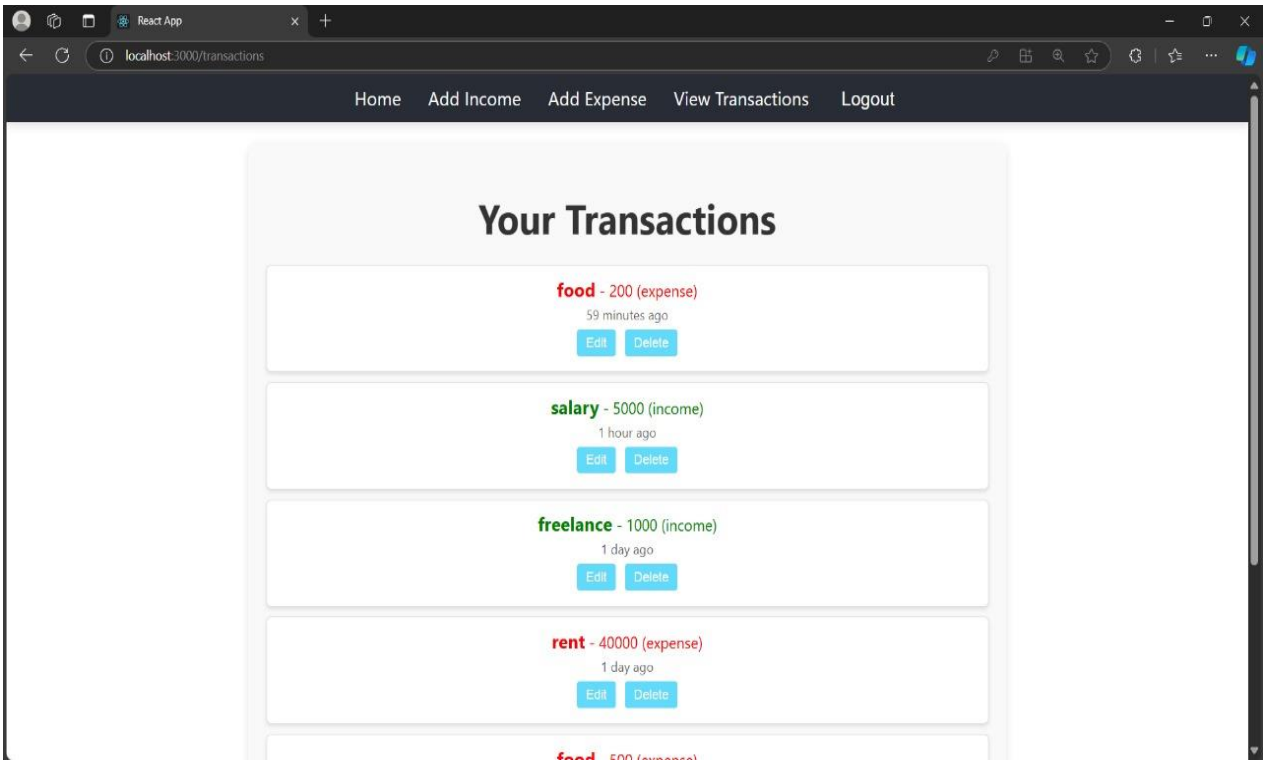
```
{
  “amount” : 1000,
  “description” : “Salary”,
  “type” : “income”
}
```

c) DELETE / api / transactions / id – Delete a transaction by its ID.

10.User Interface:

Screenshots of UI features –

-



11.Testing:

Testing tools –

In the future, testing will be implemented using tools like Jest (for unit tests on the frontend) and Mocha/Chai (for backend API testing).

12.Known Issues:

a) Google account login

b) No visualisations of expenses –

there is no data visualization features like pie charts, graphs, bar graphs etc.

C) Two-factor authentication –

Lack of an additional security layer through two-factor authentication

13.Future Enhancements:

a) Recurring transactions –

Automatically log repeating transactions such as monthly bills or salaries.

b) Category tags –

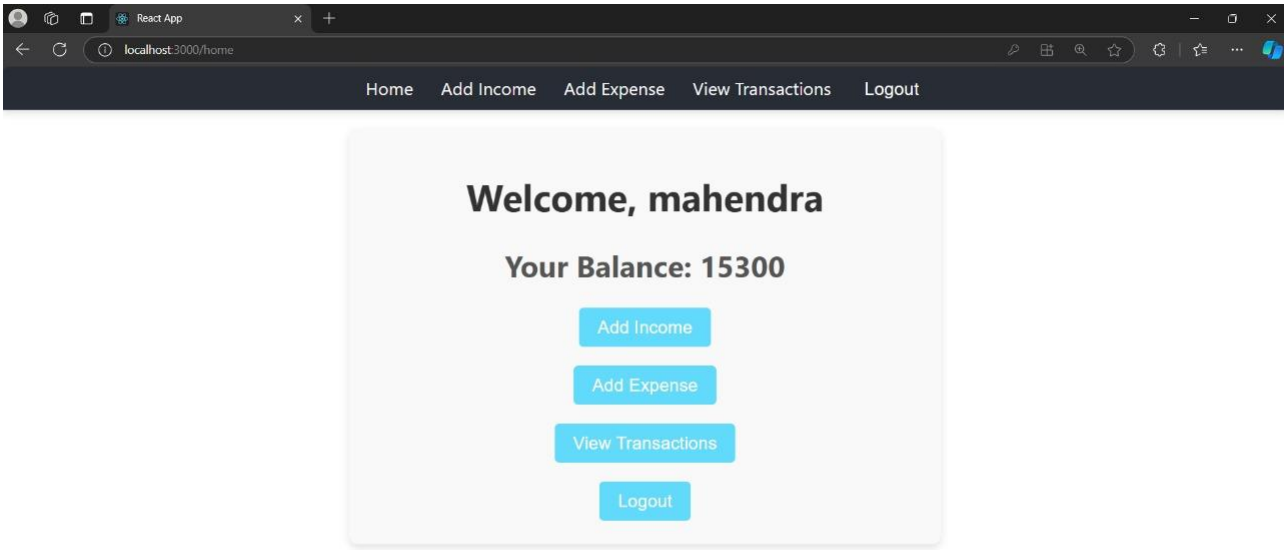
Allow users to categorize expenses and income.

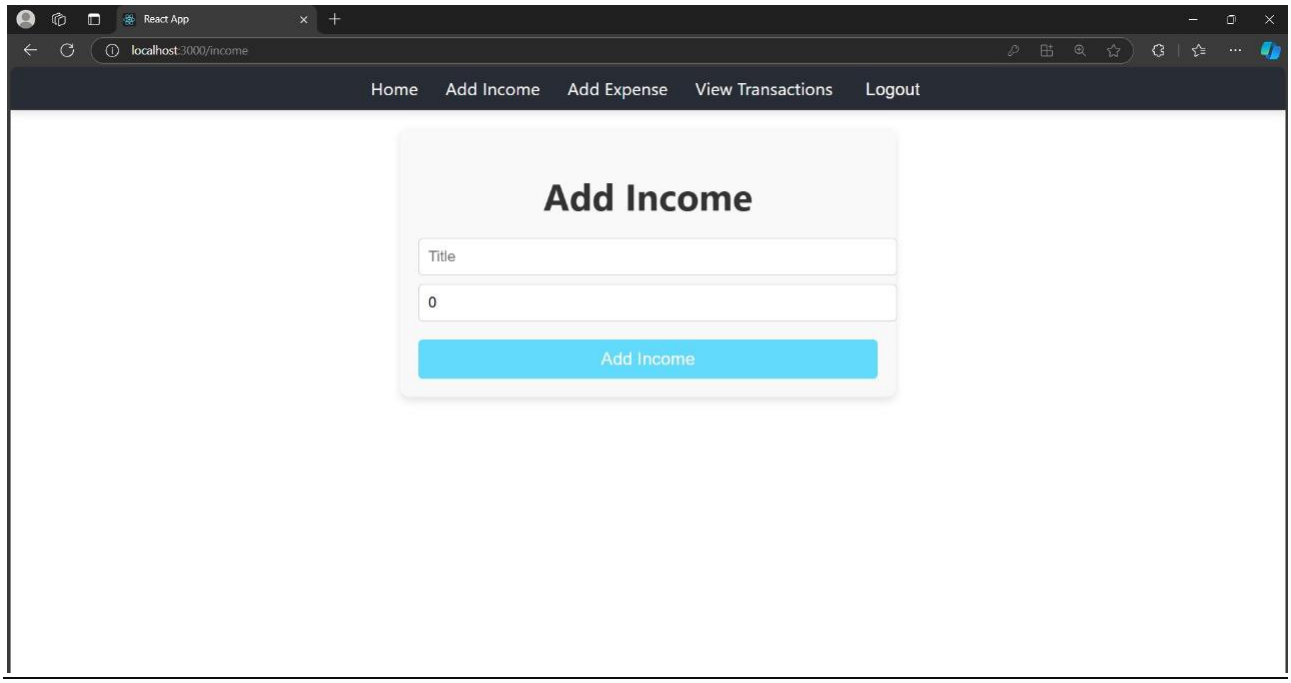
C) Mobile App –

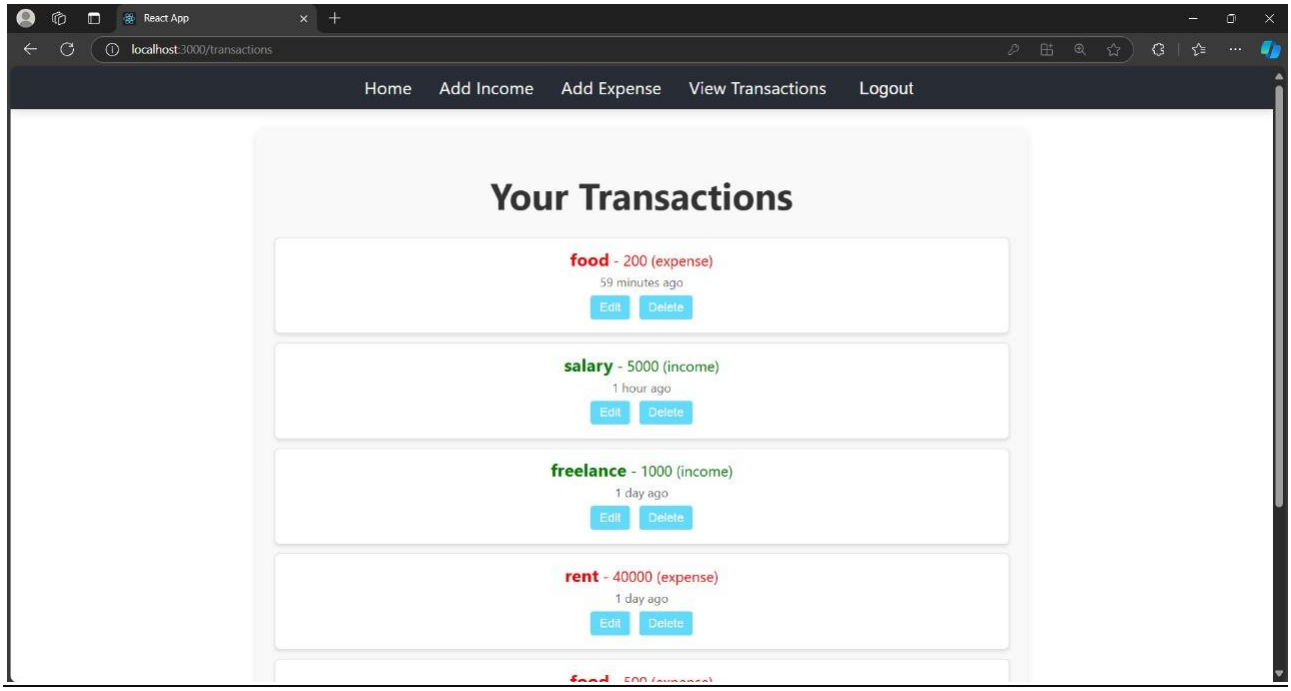
Create a mobile version of this application using React Native.

14.Project Implementation:

14.1 project user interface pages:







14.2 Source Code:

Backend:

Models:

transaction.js:

```
const mongoose = require('mongoose');
const transactionSchema = new mongoose.Schema({
    title: { type: String, required: true },
    amount: { type: Number, required: true },
    type: { type: String, enum: ['income', 'expense'], required: true },
    createdAt: { type: Date, default: Date.now },
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }
});
 const Transaction = mongoose.model('Transaction', transactionSchema);
 module.exports = Transaction;
```

user.js:

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    balance: { type: Number, default: 0 }
});
const User = mongoose.model('User', userSchema);
module.exports = User;
```

Server.js:

```
const express = require('express');
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const cookieParser = require('cookie-parser');
const bodyParser = require('body-parser');
const User = require('./models/User');
const Transaction = require('./models/Transaction');
const app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(cookieParser());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
const cors = require('cors');
app.use(cors({
    origin: 'http://localhost:3000',
```

```
credentials: true
}));
mongoose.connect('mongodb+srv://{username}:{password}@cluster0.plwp0.mongodb.net/products1').then(()=>{console.log("db connected")})
.catch((e)=>console.log(e));
const authenticate = (req, res, next) => {
    const token = req.cookies.token;
    if (!token) return res.status(401).json({ message: 'Unauthorized' });
    jwt.verify(token, 'secret', (err, decoded) => {
        if (err) return res.status(401).json({ message: 'Invalid Token' });
        req.user = decoded;
next();
    });
};
const recalculateUserBalance = async (userId) => {
try {
        const transactions = await Transaction.find({ userId });
        let balance = 0;
        transactions.forEach(transaction => {
            if (transaction.type === 'income') {
                balance += transaction.amount;
            } else if (transaction.type === 'expense') {
                balance -= transaction.amount;
            }
});
        const user = await User.findById(userId);
```

```
        user.balance = balance;
        await user.save();
    } catch (error) {
        console.error('Error recalculating balance:', error);
    }
};
app.post('/api/createUser', async (req, res) => {
    try {
        const { username, password } = req.body;
        const hashedPassword = await bcrypt.hash(password, 10);
        const newUser = new User({ username, password: hashedPassword });
        await newUser.save();
       res.status(201).json({ message: 'User created successfully', user: newUser });
    } catch (error) {
        res.status(500).json({ error: 'Error creating user' });
    }
});
app.post('/api/login', async (req, res) => {
    const { username, password } = req.body;
    const user = await User.findOne({ username });
    if (!user || !await bcrypt.compare(password, user.password)) {
        return res.status(401).json({ message: 'Invalid credentials' });
    }
    const token = jwt.sign({ userId: user._id, username: user.username }, 'secret');
    res.cookie('token', token, { httpOnly: true });
```

```
  res.status(200).json({ message: 'Login successful', token });
});
app.get('/api/home', authenticate, async (req, res) => {
    const user = await User.findById(req.user.userId);
    await recalculateUserBalance(req.user.userId);
    res.status(200).json({ username: req.user.username, balance: user.balance });
});
app.post('/api/income', authenticate, async (req, res) => {
    const { title, amount } = req.body;
    const transaction = new Transaction({ title, amount, type: 'income', userId: req.user.userId });
    await transaction.save();
    await recalculateUserBalance(req.user.userId);
    res.status(201).json({ message: 'Income added successfully', transaction });
});
app.post('/api/expense', authenticate, async (req, res) => {
    const { title, amount } = req.body;
   const transaction = new Transaction({ title, amount, type: 'expense', userId: req.user.userId });
    await transaction.save();
    await recalculateUserBalance(req.user.userId);
    res.status(201).json({ message: 'Expense added successfully', transaction });
});
app.get('/api/transactions', authenticate, async (req, res) => {
const transactions = await Transaction.find({ userId: req.user.userId }).sort({ createdAt: -1 });
```

```
res.status(200).json({ transactions });
});
app.get('/api/transactions/:id', authenticate, async (req, res) => {
    try {
        const { id } = req.params;
        const transaction = await Transaction.findById(id);
        if (!transaction || transaction.userId.toString() !== req.user.userId) {
            return res.status(403).send('Unauthorized');
        }
        res.json(transaction);
    } catch (error) {
      res.status(500).json({ error: 'Error fetching transaction' });
    }
});
app.put('/api/transaction/:id', authenticate, async (req, res) => {
    const { id } = req.params;
    const { title, amount } = req.body;
    let transaction = await Transaction.findById(id);
    if (!transaction || transaction.userId.toString() !== req.user.userId) {
        return res.status(403).json({ message: 'Unauthorized' });
    }
    transaction.title = title;
    transaction.amount = amount;
    await transaction.save();
    await recalculateUserBalance(req.user.userId);
    res.status(200).json({ message: 'Transaction updated successfully', transaction });
```

```
});
app.delete('/api/transaction/:id', authenticate, async (req, res) => {
    const { id } = req.params;
    const transaction = await Transaction.findById(id);
    if (!transaction || transaction.userId.toString() !== req.user.userId) {
        return res.status(403).json({ message: 'Unauthorized' });
    }
    await Transaction.findByIdAndDelete(id);
    await recalculateUserBalance(req.user.userId);
    res.status(200).json({ message: 'Transaction deleted successfully' });
});
app.post('/api/logout', (req, res) => {
    res.cookie('token', '', { httpOnly: true,  sameSite: 'Strict' , expires: new Date(0)});
    res.status(200).json({ message: 'Logout successful' });
});
app.listen(5000, () => console.log('Server running on port 5000'));
```

Frontend:

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, useLocation } from 'react-router-dom';
import Header from './components/Header';
import LoginPage from './components/LoginPage';
import HomePage from './components/HomePage';
import IncomePage from './components/IncomePage';
```

```
import ExpensePage from './components/ExpensePage';
import TransactionsPage from './components/TransactionsPage';
import TransactionEditPage from './components/TransactionEditPage';
import CreateUserPage from './components/CreateUserPage';
import './App.css';
function App() {
  const location = useLocation();
  return (
    <div>
      {location.pathname !== '/login' && location.pathname !== '/createuser' &&
<Header />}
      <Routes>
        <Route path="/login" element={<LoginPage />} />
        <Route path="/" element={<HomePage />} />
        <Route path="/home" element={<HomePage />} />
        <Route path="/income" element={<IncomePage />} />
        <Route path="/expense" element={<ExpensePage />} />
         <Route path="/transactions" element={<TransactionsPage />} />
        <Route path="/edit/:id" element={<TransactionEditPage />} />
        <Route path="/createuser" element={<CreateUserPage />} />
      </Routes>
    </div>
  );
}
function AppWrapper() {
```

```
return (
    <Router>
      <App />
    </Router>
  );
}
export default AppWrapper;
```

CreateUserPage.jsx:

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
function CreateUserPage() {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const navigate = useNavigate();
    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            const response = await axios.post('http://localhost:5000/api/createUser', { username, password },{withCredentials:true});
            if (response.status === 201) {
                navigate('/login');
            }
        } catch (error) {
            console.error('Error creating user:', error);
```

```
    }
   };
   return (
      <div className="create-user-container">
  <h1>Create User</h1>
  <form className="create-user-form" onSubmit={handleSubmit}>
   <input
    className="create-user-input"
    type="text"
    value={username}
    onChange={(e) => setUsername(e.target.value)}
    placeholder="Username"
   />
   <input
    className="create-user-input"
    type="password"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
    placeholder="Password"
   />
   <button className="create-user-button" type="submit">Create Account</button>
  </form>
  <button className="create-user-button" onClick={() =>
navigate('/login')}>Login</button>
</div>
```

```
);
}
export default CreateUserPage;
```

ExpensePage.jsx:

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
function ExpensePage() {
  const [title, setTitle] = useState('');
  const [amount, setAmount] = useState(0);
  const navigate = useNavigate();
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:5000/api/expense', { title, amount
},{withCredentials:true});
      navigate('/home');
    } catch (error) {
      console.error('Error adding expense:', error);
    }
  };
  return (
    <div className="expense-container">
    <h1>Add Expense</h1>
    <form className="expense-form" onSubmit={handleSubmit}>
```

```
      <input className="expense-input" type="text" value={title} onChange={(e) => setTitle(e.target.value)} placeholder="Title" />
        <input className="expense-input" type="number" value={amount} onChange={(e) => setAmount(e.target.value)} placeholder="Amount" />
        <button className="expense-button" type="submit">Add Expense</button>
       </form>
     </div>
   );
}
export default ExpensePage;
```

Header.jsx:

```
import React from 'react';
import { Link, useNavigate } from 'react-router-dom';
import axios from 'axios';
function Header() {
  const navigate = useNavigate();
  const handleLogout = async () => {
try {
     await axios.post('http://localhost:5000/api/logout', {}, { withCredentials: true });
     navigate('/login');
    } catch (error) {
     console.error('Error logging out:', error);
    }
  };
  return (
    <header className="main-header">
```

```
      <Link className="header-link" to="/home">Home</Link>
      <Link className="header-link" to="/income">Add Income</Link>
      <Link className="header-link" to="/expense">Add Expense</Link>
      <Link className="header-link" to="/transactions">View Transactions</Link>
      <button className="header-link logout-button"
onClick={handleLogout}>Logout</button>
    </header>
  );
}
export default Header;
```

HomePage.jsx:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
function HomePage() {
  const [balance, setBalance] = useState(0);
  const [username, setUsername] = useState('');
  const [transactions, setTransactions] = useState([]);
  const navigate = useNavigate();
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/home', {
withCredentials: true });
        setBalance(response.data.balance);
        setUsername(response.data.username);
        setTransactions(response.data.transactions); // Fetch transactions data
```

```
    } catch (error) {
      console.error('Error fetching data:', error);
      navigate('/login');
    }
  };
  fetchData();
}, []);
const handleLogout = async () => {
  try {
    await axios.post('http://localhost:5000/api/logout', {}, { withCredentials: true });
    navigate('/login');
  } catch (error) {
    console.error('Error logging out:', error);
  }
};
return (
  <div className="home-container">
    <h1>Welcome, {username}</h1>
    <h2>Your Balance: {balance}</h2>
    <div className="buttons-container">
      <button className="home-button" onClick={() => navigate('/income')}>Add
Income</button>
      <button className="home-button" onClick={() => navigate('/expense')}>Add
Expense</button>
      <button className="home-button" onClick={() =>
navigate('/transactions')}>View Transactions</button>
      <button className="home-button"
onClick={handleLogout}>Logout</button>
```

```
    </div>
    </div>
   );
  }
  export default HomePage;
```

IncomePage.jsx:

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
function IncomePage() {
    const [title, setTitle] = useState('');
    const [amount, setAmount] = useState(0);
    const navigate = useNavigate();
    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            await axios.post('http://localhost:5000/api/income', { title, amount
},{withCredentials:true});
            navigate('/home');
        } catch (error) {
            console.error('Error adding income:', error);
            navigate('/login')
        }
    };
    return (
        <div className="income-container">
  <h1>Add Income</h1>
```

```
 <form className="income-form" onSubmit={handleSubmit}>
<input
      className="income-input"
      type="text"
      value={title}
      onChange={(e) => setTitle(e.target.value)}
      placeholder="Title"
    />
    <input
      className="income-input"
      type="number"
      value={amount}
      onChange={(e) => setAmount(e.target.value)}
      placeholder="Amount"
    />
    <button className="income-button" type="submit">Add Income</button>
  </form>
</div>
    );
}
export default IncomePage;
```

LoginPage.jsx:

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
function LoginPage() {
```

```
 const [username, setUsername] = useState('');
 const [password, setPassword] = useState('');
 const navigate = useNavigate();
  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('http://localhost:5000/api/login', { username, password }, { withCredentials: true });
      if (response.status === 200) {
        navigate('/home');
      }
    } catch (error) {
      console.error('Error logging in:', error);
      alert("Wrong username or password! Please try again");
    }
  };
  return (
    <div className="login-container">
      <h1 className="site-title">Personal Expense Tracker</h1>
      <h2>Login</h2>
      <form className="login-form" onSubmit={handleLogin}>
        <input
          className="login-input"
          type="text"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          placeholder="Username"
```

```
        />
         <input
         className="login-input"
         type="password"
         value={password}
         onChange={(e) => setPassword(e.target.value)}
         placeholder="Password"
        />
        <button className="login-button" type="submit">Login</button>
      </form>
      <button className="register-button" onClick={() =>
navigate('/createuser')}>Register</button>
    </div>
  );
}
export default LoginPage;
```

TransactionEditPage.jsx:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams, useNavigate } from 'react-router-dom';
function TransactionEditPage() {
    const { id } = useParams();
    const [title, setTitle] = useState('');
    const [amount, setAmount] = useState('');
    const [loading, setLoading] = useState(true);
    const navigate = useNavigate();
    useEffect(() => {
```

```
        const fetchTransaction = async () => {
        console.log(id)
         try {
            const response = await
axios.get(`http://localhost:5000/api/transactions/${id}`, { withCredentials: true });
            // console.log(response.data)
            const  transaction  = await response.data;
            console.log(transaction)
            setTitle(transaction.title);
            setAmount(transaction.amount);
            setLoading(false);
        } catch (error) {
            console.error('Error fetching transaction:', error);
            setLoading(false);
            navigate('/login')
        }
      };
      fetchTransaction();
   }, [id]);
   const handleSubmit = async (e) => {
      e.preventDefault();
      try {
         await axios.put(`http://localhost:5000/api/transaction/${id}`, { title, amount },
{ withCredentials: true });
         navigate('/transactions');
      } catch (error) {
         console.error('Error updating transaction:', error);
```

```
    }
  };
  if (loading) {
    return <div>Loading...</div>;
  }
  return (
    <div className="edit-transaction-container">
    <h1>Edit Transaction</h1>
    <form className="edit-transaction-form" onSubmit={handleSubmit}>
      <input
        className="edit-transaction-input"
        type="text"
        value={title}
        onChange={(e) => setTitle(e.target.value)}
        placeholder="Title"
        required
      />
      <input
        className="edit-transaction-input"
        type="number"
        value={amount}
        onChange={(e) => setAmount(e.target.value)}
        placeholder="Amount"
        required
      />
      <button className="edit-transaction-button" type="submit">Update
Transaction</button>
```

```
        </form>
      </div>
    );
}
export default TransactionEditPage;
```

TransactionsPage.jsx:

```
// export default TransactionsPage;
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
function TransactionsPage() {
    const [transactions, setTransactions] = useState([]);
    const navigate = useNavigate();
    useEffect(() => {
        const fetchTransactions = async () => {
            try {
                const response = await axios.get('http://localhost:5000/api/transactions', {
withCredentials: true });
                setTransactions(response.data.transactions);
            } catch (error) {
                console.error('Error fetching transactions:', error);
            }
        };
        fetchTransactions();
    }, []);
 const handleDelete = async (id) => {
        try {
```

```
            await axios.delete(`http://localhost:5000/api/transaction/${id}`, {
withCredentials: true });
            setTransactions(transactions.filter((transaction) => transaction._id !== id));
        } catch (error) {
            console.error('Error deleting transaction:', error);
            navigate('/login')
        }
    };
    const handleEdit = (id) => {
        navigate(`/edit/${id}`);
    };
    const formatDateTime = (dateString) => {
        const date = new Date(dateString);
        const now = new Date();
        const diffInSeconds = Math.floor((now - date) / 1000);
        if (diffInSeconds < 60) {
            return 'just now';
        } else if (diffInSeconds < 3600) {
            const minutes = Math.floor(diffInSeconds / 60);
            return `${minutes} minute${minutes > 1 ? 's' : ''} ago`;
        } else if (diffInSeconds < 86400) {
            const hours = Math.floor(diffInSeconds / 3600);
            return `${hours} hour${hours > 1 ? 's' : ''} ago`;
        } else if (diffInSeconds < 2592000) {
            const days = Math.floor(diffInSeconds / 86400);
            return `${days} day${days > 1 ? 's' : ''} ago`;
        } else if (diffInSeconds < 31536000) {
```

```
        const months = Math.floor(diffInSeconds / 2592000);
        return `${months} month${months > 1 ? 's' : ''} ago`;
      } else {
        const years = Math.floor(diffInSeconds / 31536000);
        return `${years} year${years > 1 ? 's' : ''} ago`;
      }
    };
    return (
      <div className="transactions-container">
  <h1>Your Transactions</h1>
  <ul className="transactions-list">
    {transactions.map((transaction) => (
      <li
        className="transaction-item"
        key={transaction._id}
        style={{
          color: transaction.type === 'income' ? 'green' : 'red',
        }}
      >
        <strong>{transaction.title}</strong> - {transaction.amount}
({transaction.type})
        <br />
        <small>{formatDateTime(transaction.createdAt)}</small>
        <br />
        <button className="transaction-button" onClick={() =>
handleEdit(transaction._id)}>Edit</button>
        <button className="transaction-button" onClick={() =>
handleDelete(transaction._id)}>Delete</button>
```

```
        </li>
    ))}
  </ul>
</div>
    );
}
export default TransactionsPage;
```

14.3 Github demo link:

https://github.com/218X1A4553/Personal-Expense-tracker-Application

14.4 Demonstration Video Link:

https://drive.google.com/file/d/1e6xeWvLopZYvv2nY_49CiYX_WzDHVCHD/view?usp=sharing