

CHAPTER 1

INTRODUCTION

In today's fast-paced world, managing personal finances effectively has become increasingly important. Individuals need tools that allow them to track their expenses, monitor their budgets, and gain insights into their spending habits. To address this need, we propose the development of an Expense Tracker application using the MERN (MongoDB, Express.js, React.js, Node.js) stack.

The Expense Tracker application aims to provide users with a user-friendly platform to record their expenses, categorize them, set budgets, and visualize their financial data. Leveraging the power of the MERN stack, we can create a robust and scalable application that offers real-time updates, seamless user experiences, and secure data storage.

Through this project, users will have access to features such as transaction recording, expense categorization, budget tracking, and insightful visualizations. Whether managing daily expenses, planning for major purchases, or saving for long-term goals, the Expense Tracker application will empower users to take control of their finances and make informed financial decisions.

By utilizing modern web technologies and best practices in software development, this project aims to deliver a comprehensive solution for personal finance management. The following sections will delve into the architecture, implementation details, features, and future enhancements of the Expense Tracker application, demonstrating how it can revolutionize the way individuals manage their finances in today's digital age.

1.1 Objectives

User Authentication and Authorization: Implement user authentication (signup, login, logout) and authorization to ensure that only authenticated users can access the application and their respective data.

Expense Entry: Allow users to add new expenses including details such as expense category, amount, date, and any additional notes.

Expense Editing and Deletion: Provide functionality for users to edit or delete existing expenses to maintain accurate records.

Expense Categories: Allow users to categorize expenses to better organize and analyze their spending habits.

Expense Visualization: Implement graphical representations (charts, graphs) to visualize expense data over time or by category, helping users to gain insights into their spending patterns.

Expense Filtering and Sorting: Enable users to filter and sort their expenses based on various criteria such as date, category, or amount.

Data Privacy and Security: Ensure that user data is securely stored and protected, adhering to best practices for data privacy and security.

Responsive Design: Create a responsive user interface that works well across different devices and screen sizes, providing a consistent experience for users accessing the application from desktops, tablets, or smartphones.

Scalability: Design the application architecture with scalability in mind, allowing it to handle increasing numbers of users and data volumes efficiently.

1.2 Scope of the project

The scope of a project for an expense tracker management system using the MERN stack encompasses various components and functionalities. Here's a breakdown of the scope:

User Management:

User registration: Allow users to sign up for an account with a unique username and password.

User authentication: Implement a secure authentication system to verify user identities during login.

User profile management: Enable users to update their profiles, including personal information and preferences.

Expense Management:

Expense creation: Allow users to add new expenses with details such as amount, date, category, and description.

Expense editing: Provide functionality for users to edit existing expenses to correct errors or update information.

Expense deletion: Allow users to delete expenses that are no longer relevant or accurate.

Expense categories: Enable users to categorize expenses into different categories (e.g., groceries, utilities, entertainment) for better organization and analysis.

Dashboard and Reporting:

Dashboard overview: Display a summary of total expenses, expense trends over time, and budget status on the user dashboard.

Expense visualization: Generate graphical representations (e.g., charts, graphs) to visualize expense data, allowing users to analyze spending patterns and trends.

Reporting: Provide customizable reports to allow users to generate detailed reports based on specific criteria (e.g., date range, category).

Budgeting and Goal Setting:

Budget creation: Allow users to set monthly or periodic budgets for different expense categories.

Budget tracking: Track actual expenses against budgeted amounts and provide alerts or notifications when users exceed their budget limits..

Data Management:

Data storage: Utilize MongoDB to store user account information, expense data, and other relevant data.

Data retrieval: Implement APIs and endpoints using Node.js and Express.js to retrieve and manipulate data from the database.

Data synchronization: Ensure data consistency and synchronization across multiple devices and sessions.

Security and Privacy:

Data encryption: Encrypt sensitive user information (e.g., passwords) and communication between the client and server to protect against unauthorized access.

Access control: Implement role-based access control (RBAC) to restrict access to certain features or data based on user roles and permissions.

Compliance: Ensure compliance with relevant data protection regulations (e.g., GDPR, CCPA) to protect user privacy and rights.

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 Frontend Requirements:

React.js Framework: The frontend of the application will be built using React.js to create dynamic user interfaces.

Responsive Design: The application should be accessible and usable across various devices, including desktops, tablets, and mobile phones.

User Authentication: Implement user authentication mechanisms such as login, registration, and password reset functionalities.

User Interface Components: Develop UI components for recording transactions, categorizing expenses, setting budgets, and displaying financial data.

2.2 Backend Requirements:

Node.js and Express.js: Use Node.js as the server-side runtime environment and Express.js as the web application framework to handle HTTP requests and responses.

MongoDB Database: Utilize MongoDB as the database management system for storing user data, transactions, categories, and budget information.

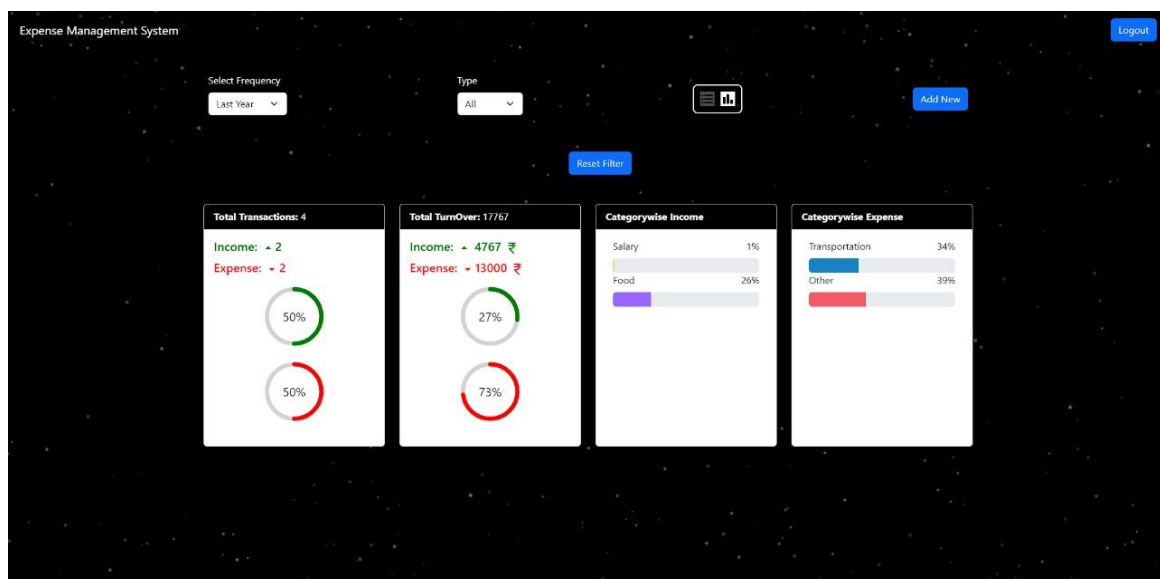
RESTful APIs: Design and implement RESTful APIs to enable communication between the frontend and backend components of the application.

Data Validation and Sanitization: Implement validation and sanitization techniques to ensure the security and integrity of user input data.

CHAPTER 3

DESIGN

Designing the architecture for this project involves structuring the components, services, and modules in a way that promotes modularity, scalability, and maintainability. Below is a high-level overview of the design for this project:



the components you'll need:

Front-end (React.js)

User Interface Components:

Input forms for adding/editing expenses.

List view to display expenses.

Charts or visual representations of expenses (optional but helpful).

State Management:

Redux or Context API for managing application-level state.

State slices for managing expenses, user authentication, etc.

Routing:

React Router for handling client-side routing.

User Authentication:

Authentication components for user sign-up, login, logout.

Authentication tokens (JWT) for securing API calls.

API Interaction:

Axios or Fetch for making API requests to the backend.

Back-end (Node.js with Express.js)

REST API:

Routes for CRUD operations on expenses.

Routes for user authentication (login, register, logout).

Authentication middleware to verify tokens.

Database Integration:

MongoDB with Mongoose for storing expense data.

User model and schema for managing user authentication.

Authentication:

JWT (JSON Web Tokens) for user authentication.

Middleware for verifying tokens on protected routes.

Error Handling:

Middleware for handling errors and sending appropriate responses.

Validation:

Input validation middleware (e.g., using express-validator) to validate incoming data.

Middleware:

Body-parser middleware for parsing request bodies.

CORS middleware for handling Cross-Origin Resource Sharing.

Security:

Use environment variables for sensitive information.

Helmet middleware for setting HTTP headers securely.

Sanitization middleware to prevent injection attacks.

Database (MongoDB)

Database Schema:

Define schemas for expense data.

Define a user schema for managing user accounts.

Data Access:

Use Mongoose ORM for interacting with MongoDB.

Indexes:

Define appropriate indexes for efficient querying.

Data Migration & Seeding:

Scripts for seeding initial data or migrating schema changes.

Deployment & DevOps

Hosting:

Choose a hosting provider (e.g., Heroku, AWS, DigitalOcean) for deploying your backend and front end.

Continuous Integration/Continuous Deployment (CI/CD):

Set up CI/CD pipelines for automated testing and deployment.

Monitoring & Logging:

Implement logging and monitoring solutions (e.g., Sentry, Loggly) for tracking errors and performance.

Security:

Secure your server with HTTPS.

Implement security best practices for server and database

CHAPTER 4

IMPLEMENTATION

The implementation of the Expense Tracker project using the MERN stack involves several key components and steps. Here's a brief explanation of the implementation process:

Setting Up the Development Environment:

Install Node.js, MongoDB, and any necessary dependencies on the development machine.

Set up a new project directory and initialize a Git repository for version control.

Backend Development (Node.js and Express.js):

Create a new Express.js application to serve as the backend of the Expense Tracker.

Define routes and controllers to handle various HTTP requests, such as creating new transactions, retrieving transaction data, and managing user authentication.

Implement middleware for handling authentication, error handling, and other cross-cutting concerns.

Connect the backend application to MongoDB using a MongoDB client library (e.g., Mongoose) to interact with the database.

Frontend Development (React.js):

Set up a new React.js application to serve as the frontend of the Expense Tracker.

Design and develop UI components for different pages and functionalities of the application, such as the dashboard, transaction form, budget management, and reports.

Implement routing using React Router to navigate between different pages of the application.

Integrate state management using libraries like Redux or React Context API to manage application state and data flow.

Database Design and Integration (MongoDB):

Design the MongoDB database schema to store user information, transactions, categories, and budget data.

Set up MongoDB collections and indexes based on the defined schema.

Implement CRUD (Create, Read, Update, Delete) operations to interact with the MongoDB database from the backend application.

User Authentication and Authorization:

Implement user authentication using libraries like Passport.js or JSON Web Tokens (JWT) to secure the application's endpoints.

Develop registration, login, and logout functionalities to allow users to create and access their accounts securely.

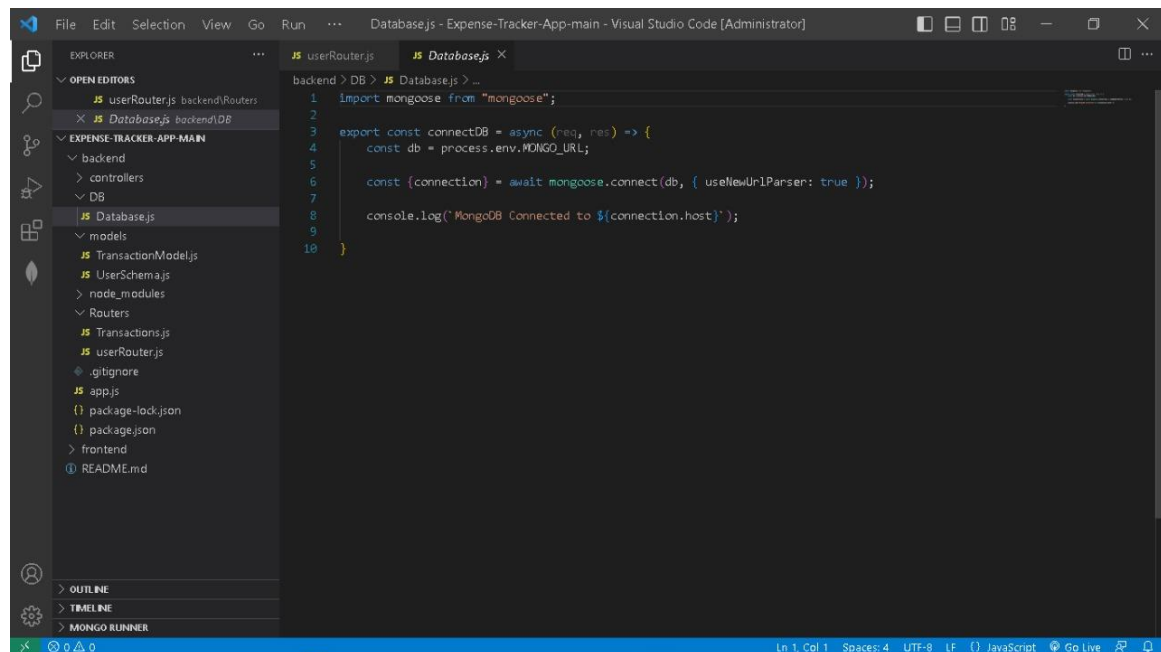
Implement authorization mechanisms to restrict access to certain features or data based on user roles and permissions.

Testing and Quality Assurance:

Write unit tests for backend API endpoints and frontend components using testing frameworks like Jest, Mocha, or React Testing Library.

Perform integration testing to ensure that different components of the application work together as expected.

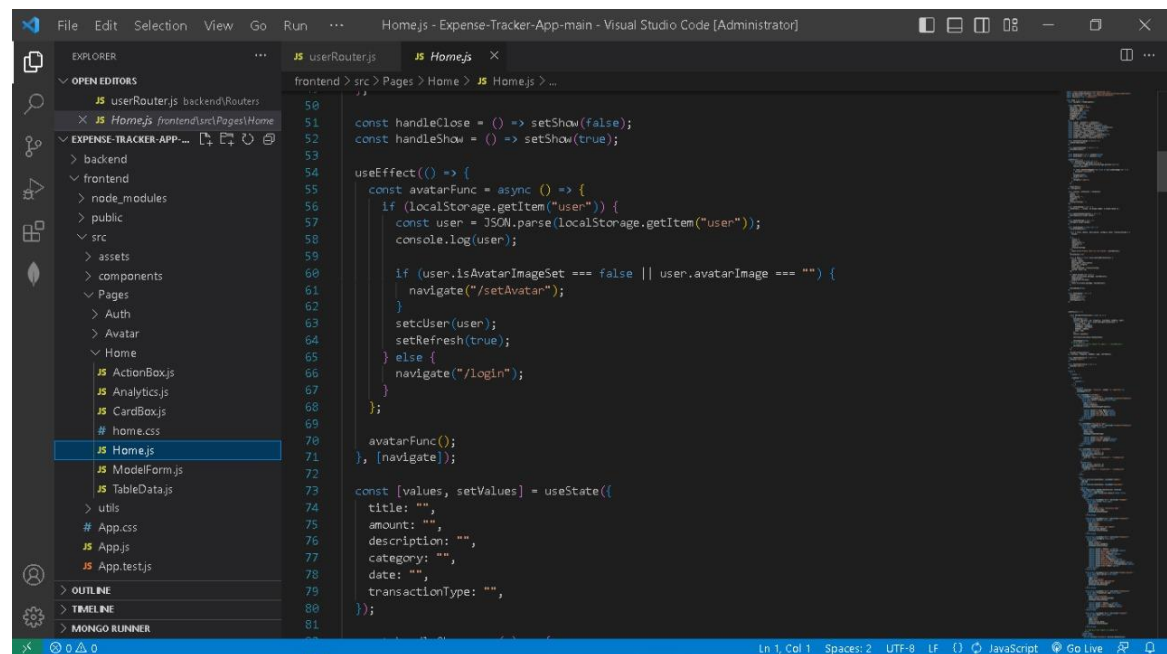
Conduct user acceptance testing to validate that the application meets the specified requirements and user expectations.



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the project structure for 'Expense-Tracker-App-main'. The file explorer is expanded to show the 'Database.js' file. The main editor area displays the content of 'Database.js', which is a JavaScript file for connecting to a MongoDB database. The code includes an import for 'mongoose', a function 'connectDB' that takes 'req' and 'res' as arguments, and a call to 'mongoose.connect' with the database URL from 'process.env.MONGO_URL'. The status bar at the bottom indicates the file is 'Database.js' and the editor is in 'JavaScript' mode.

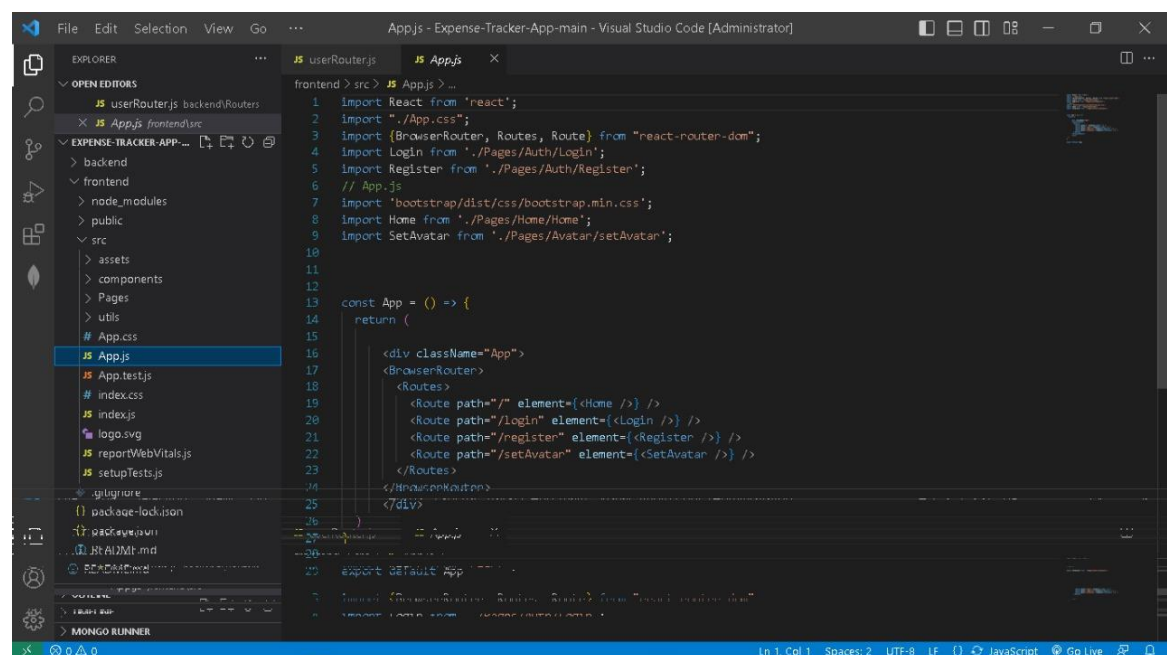
```
1 import mongoose from "mongoose";
2
3 export const connectDB = async (req, res) => {
4   const db = process.env.MONGO_URL;
5
6   const {connection} = await mongoose.connect(db, { useNewUrlParser: true });
7
8   console.log('MongoDB Connected to ${connection.host}');
9
10 }
```

Expense Tracker Management



```

50
51 const handleClose = () => setShow(false);
52 const handleShow = () => setShow(true);
53
54 useEffect(() => {
55   const avatarFunc = async () => {
56     if (localStorage.getItem("user")) {
57       const user = JSON.parse(localStorage.getItem("user"));
58       console.log(user);
59
60       if (user.isAvatarImageSet === false || user.avatarImage === "") {
61         navigate("/setAvatar");
62       }
63       setUser(user);
64       setRefresh(true);
65     } else {
66       navigate("/login");
67     }
68   };
69   avatarFunc();
70 }, [navigate]);
71
72 const [values, setValues] = useState({
73   title: "",
74   amount: "",
75   description: "",
76   category: "",
77   date: "",
78   transactionType: "",
79 });
80
81
  
```



```

1 import React from "react";
2 import "./App.css";
3 import {BrowserRouter, Routes, Route} from "react-router-dom";
4 import Login from "../Pages/Auth/Login";
5 import Register from "../Pages/Auth/Register";
6 // App.js
7 import "bootstrap/dist/css/bootstrap.min.css";
8 import Home from "../Pages/Home/Home";
9 import SetAvatar from "../Pages/Avatar/setAvatar";
10
11
12
13 const App = () => {
14   return (
15     <div className="App">
16       <BrowserRouter>
17         <Routes>
18           <Route path="/" element={<Home />} />
19           <Route path="/login" element={<Login />} />
20           <Route path="/register" element={<Register />} />
21           <Route path="/setAvatar" element={<SetAvatar />} />
22         </Routes>
23       </BrowserRouter>
24     </div>
25   );
26 }
27
28 export default App;
  
```

Expense Tracker Management

The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows the project structure: EXPENSE-TRACKER-APP-main > backend > controllers > transactionController.js. The main editor displays the code for transactionController.js, which includes imports for Transaction, User, and moment, and defines an addTransactionController function.

```

1 import Transaction from "../models/TransactionModel.js";
2 import User from "../models/UserSchema.js";
3 import moment from "moment";
4
5 export const addTransactionController = async (req, res) => {
6   try {
7     const {
8       title,
9       amount,
10      description,
11      date,
12      category,
13      userId,
14      transactionType,
15    } = req.body;
16
17    // console.log(title, amount, description, date, category, userId, transactionType);
18
19    if (
20      !title ||
21      !amount ||
22      !description ||
23      !date ||
24      !category ||
25      !transactionType
26    ) {
27      return res.status(400).json({
28        success: false,
29        messages: "Please Fill all fields",
30      });
31    }
32
33    const user = await User.findById(userId);
  
```

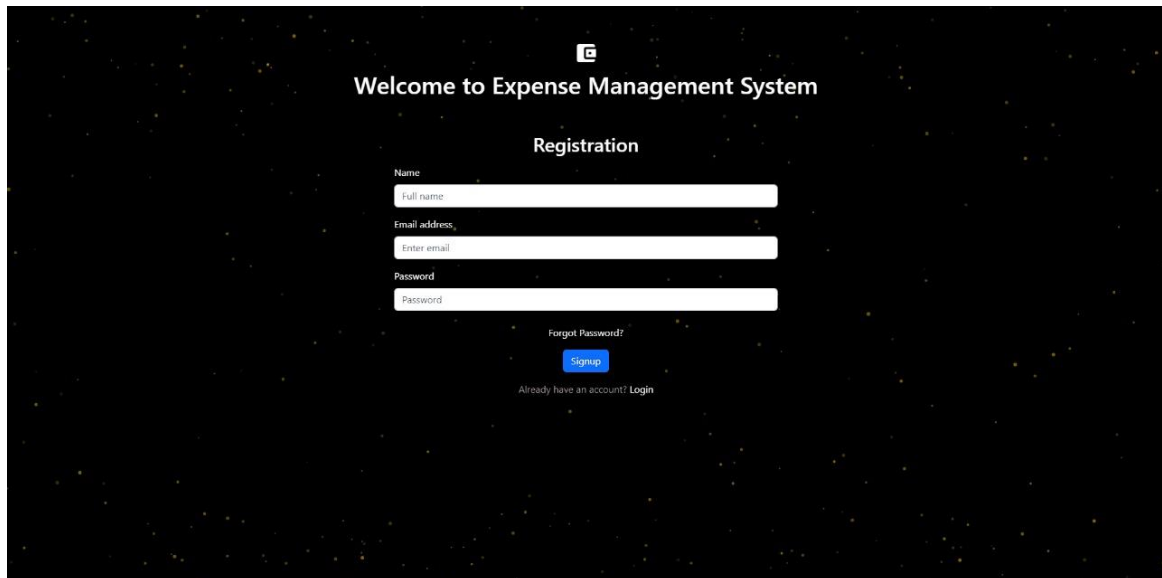
The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows the project structure: EXPENSE-TRACKER-APP-main > backend > Routers > Transactions.js. The main editor displays the code for Transactions.js, which includes imports for express, addTransactionController, deleteTransactionController, getAllTransactionController, and updateTransactionController, and defines a router for these endpoints.

```

1 import express from 'express';
2 import { addTransactionController, deleteTransactionController, getAllTransactionController, updateTransactionController } from '../controllers';
3
4 const router = express.Router();
5
6 router.route("/addTransaction").post(addTransactionController);
7
8 router.route("/getTransaction").post(getAllTransactionController);
9
10 router.route("/deleteTransaction/:id").post(deleteTransactionController);
11
12 router.route("/updateTransaction/:id").put(updateTransactionController);
13
14 export default router;
  
```

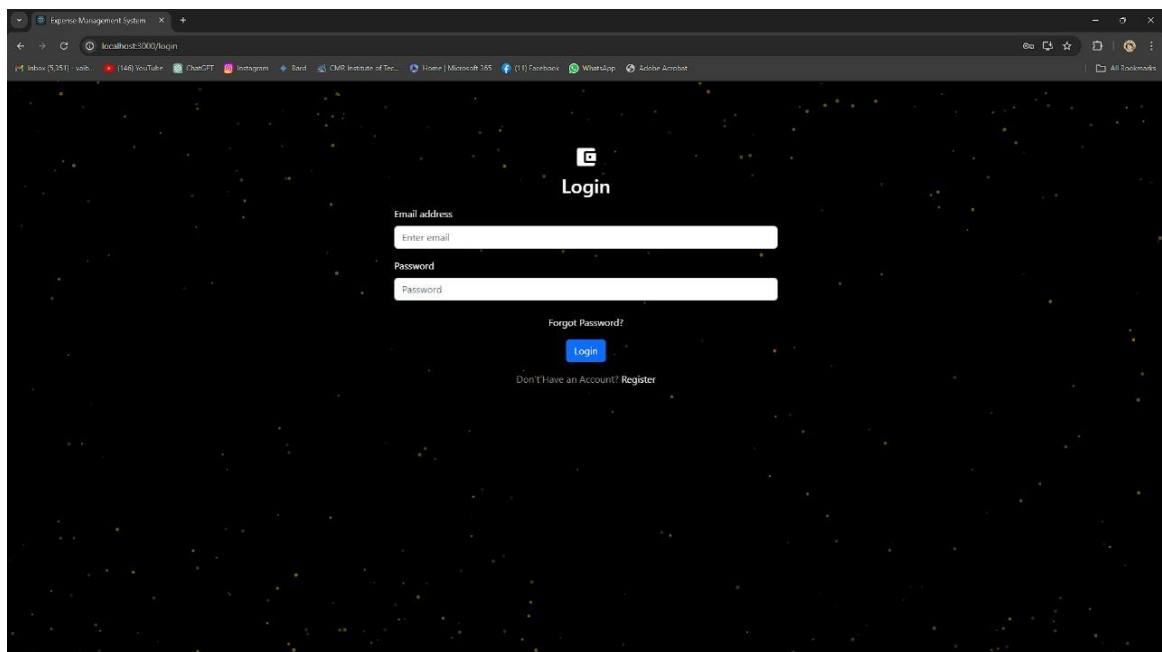
CHAPTER 5

RESULT



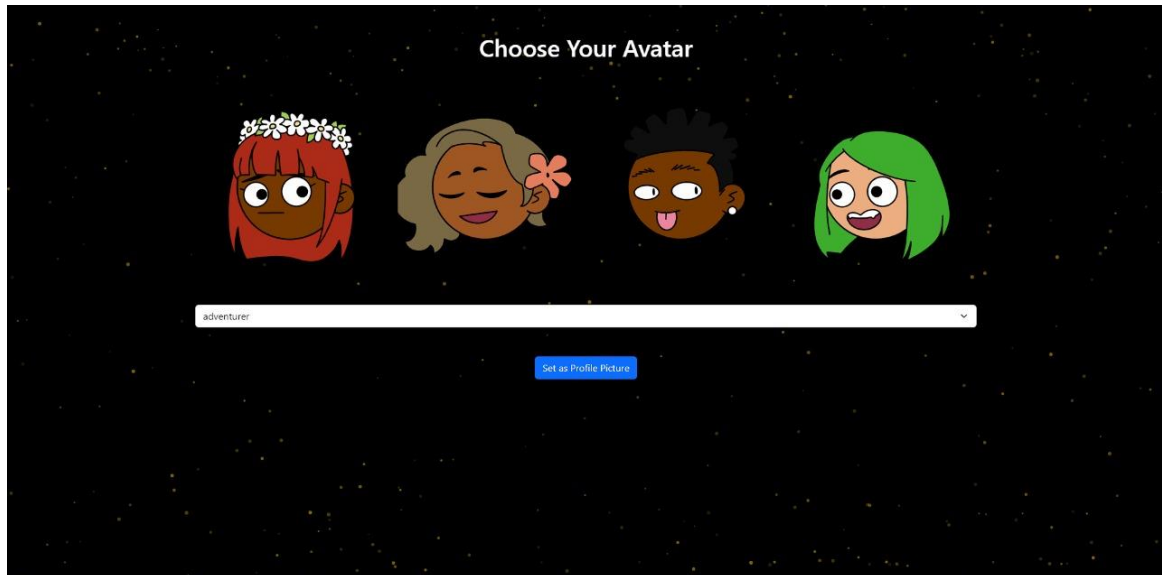
The image shows a web browser window displaying the registration page of an "Expense Management System". The page has a dark background with a subtle pattern of small yellow stars. At the top, there is a small logo and the text "Welcome to Expense Management System". Below this, the heading "Registration" is centered. The form consists of three input fields: "Name" (with a placeholder "Full name"), "Email address" (with a placeholder "Enter email"), and "Password" (with a placeholder "Password"). Below the password field, there is a link "Forgot Password?" and a blue "Signup" button. At the bottom, there is a link "Already have an account? Login".

Rrgistration of Expense Tracker Management

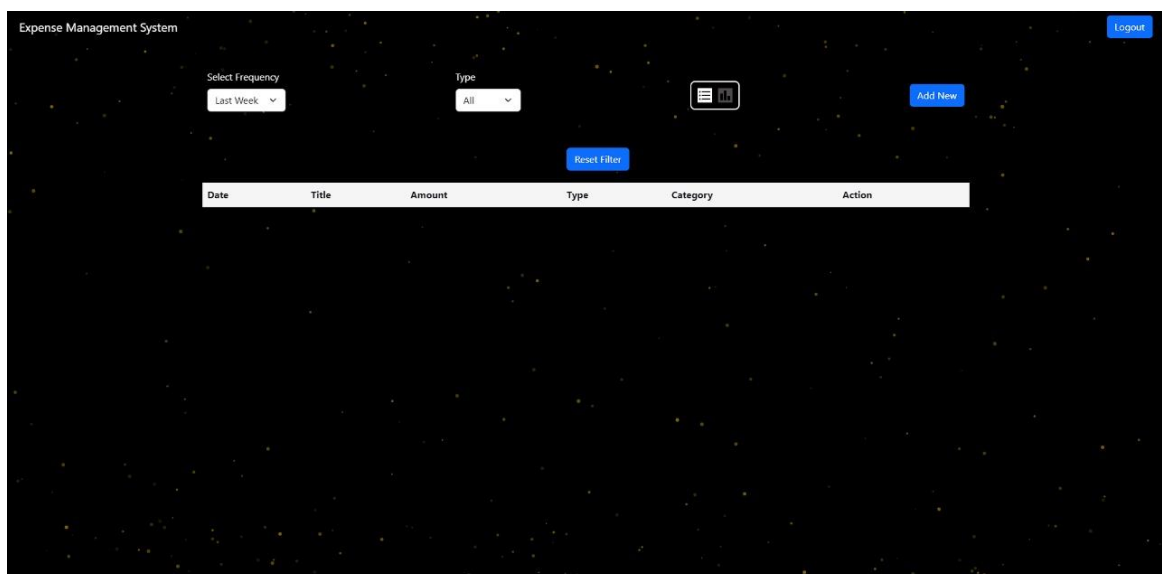


The image shows a web browser window displaying the login page of an "Expense Management System". The page has a dark background with a subtle pattern of small yellow stars. At the top, there is a small logo and the text "Welcome to Expense Management System". Below this, the heading "Login" is centered. The form consists of two input fields: "Email address" (with a placeholder "Enter email") and "Password" (with a placeholder "Password"). Below the password field, there is a link "Forgot Password?" and a blue "Login" button. At the bottom, there is a link "Don't Have an Account? Register".

Login Page



Avathar Selection



Home

Expense Tracker Management

Expense Management System Logout

Select Frequency: Last Year Type: All Add New

Reset Filter

| Date | Title | Amount | Type | Category | Action |
|------------|--------|--------|---------|----------------|--------|
| 2024-02-14 | csdc | 224 | credit | Salary | |
| 2024-03-05 | run | 4543 | credit | Food | |
| 2024-03-05 | travel | 6000 | expense | Transportation | |
| 2024-03-05 | game | 7000 | expense | Other | |

Home page

Expense Management System Logout

Select Frequency: Last Week Type: Add New

Add Transaction Details

Title:

Amount:

Category:

Description:

Transaction Type:

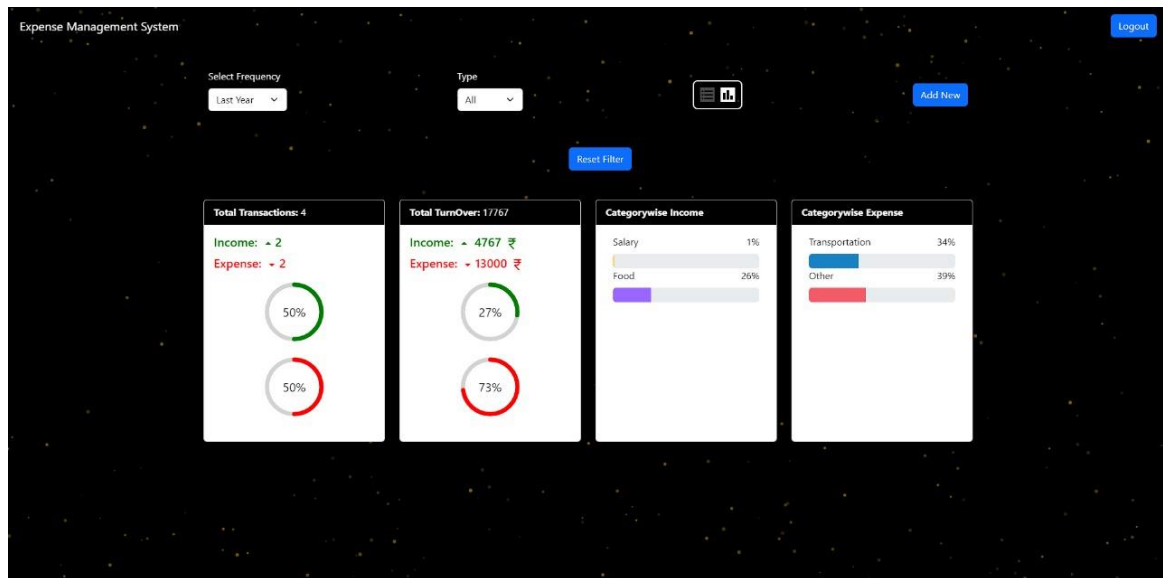
Date:

Close Submit

| Date | Title | Amount | Type | Category | Action |
|------|-------|--------|------|----------|--------|
|------|-------|--------|------|----------|--------|

Detail of Expense

Expense Tracker Management



Expense Track

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

Building an expense tracker management system using the MERN stack provides a robust and scalable solution for tracking and managing expenses. By leveraging MongoDB for data storage, Express.js for building the backend API, React.js for the frontend user interface, and Node.js for server-side logic, you can create a powerful application capable of handling user authentication, CRUD operations on expenses, and providing insightful visualizations of financial data.

Throughout the development process, attention to security, performance, and user experience is crucial. Implementing best practices for authentication, data validation, error handling, and deployment ensures that the application is reliable and user-friendly.

Future scope:

Enhanced Reporting and Insights:

Integrate advanced analytics and reporting features to provide users with deeper insights into their spending habits, trends, and financial health.

Implement machine learning algorithms for predictive analysis and personalized recommendations based on user behavior.

Budgeting and Goal Setting:

Extend the application to include budgeting functionality, allowing users to set spending limits for different categories and track their progress towards financial goals.

Implement notifications and reminders to help users stay on track with their budgets and goals.

Collaborative Features:

Enable collaboration features that allow users to share expense data with family members, roommates, or colleagues, facilitating group budgeting and expense management.

Implement real-time synchronization to ensure that all users have access to the latest expense data.

Integration with External Services:

Integrate with financial institutions or third-party services to automatically import transaction data, eliminating the need for manual entry of expenses.

Enable integration with expense tracking apps, accounting software, or personal finance management tools to provide users with a seamless experience across platforms.

Mobile Applications:

Develop mobile applications for iOS and Android platforms to reach a broader audience and provide users with the flexibility to manage their expenses on the go.

Ensure synchronization between the web and mobile applications to maintain consistency in data across devices.

Localization and Customization:

Add support for multiple languages and currencies to cater to a global audience.

Allow users to customize the application's appearance, preferences, and categories to suit their individual needs and preferences.

REFERENCES

- [1] <https://www.quora.com/>
- [2] <https://chat.openai.com/>
- [3] <https://www.geeksforgeeks.org/>
- [4] <https://www.youtube.com/>
- [5] <https://www.google.com/>