SPLITIT - TECHNICAL DESIGN DOCUMENT

TABLE OF CONTENTS

INTRODUCTION AND SYSTEM OVERVIEW

Introduction

SplitIt is a modern web application built with the MERN stack (MongoDB, Express, React, Node.js) designed to simplify the process of managing and splitting expenses for individuals and groups. Whether you're organizing a trip with friends, managing household expenses, or keeping track of a team’s finances, SplitIt provides a seamless and efficient solution. With a user-friendly interface, SplitIt ensures that everyone involved can easily record, track, and settle expenses.

The application supports group management, detailed expense tracking, and secure user authentication, making it ideal for personal finance management or collaborative financial activities. It is fully responsive, ensuring optimal performance across desktops, tablets, and mobile devices.

System Overview

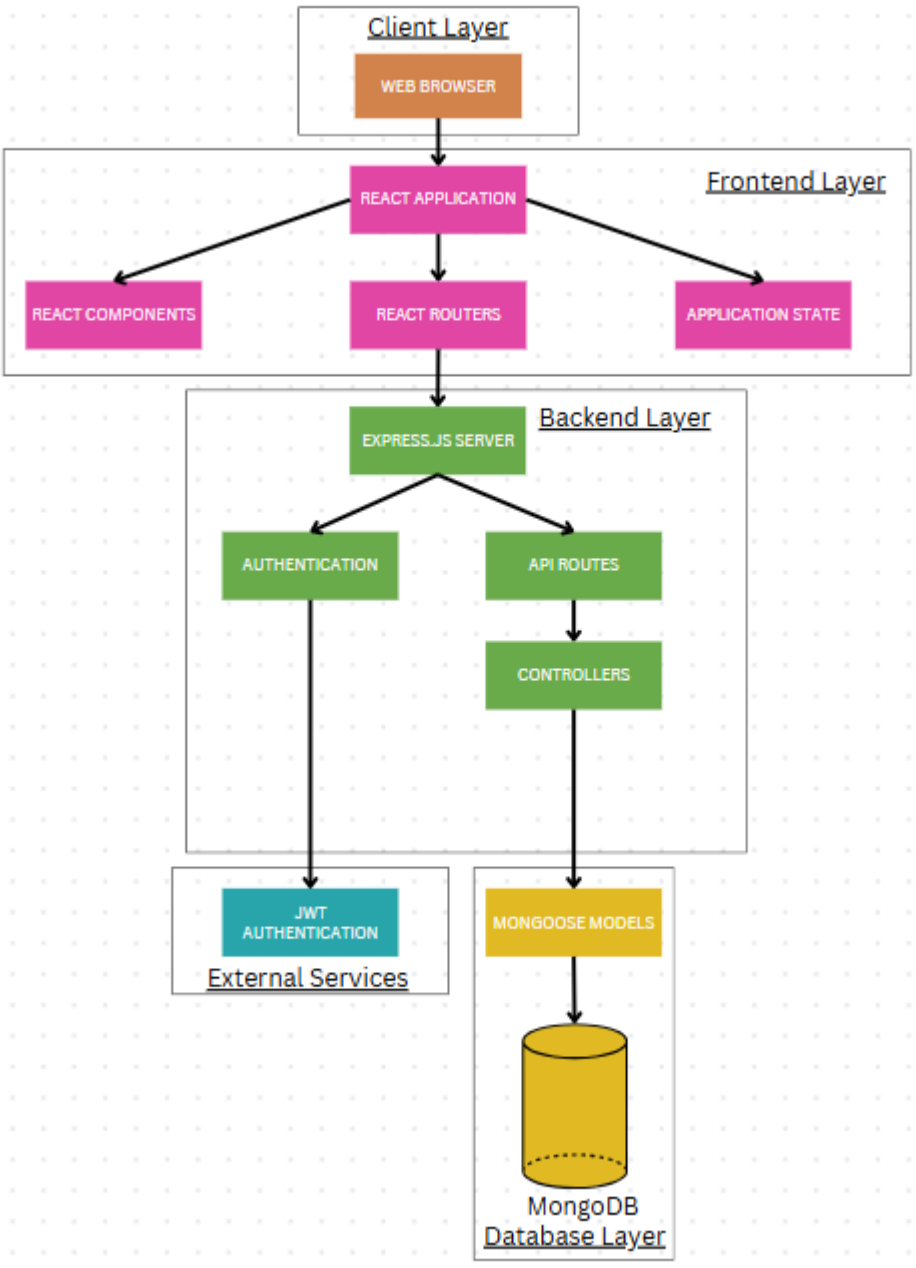
SplitIt follows a client-server architecture consisting of:

- Frontend (React.js): A dynamic, responsive interface allowing users to manage accounts, create groups, and track expenses. It communicates with the backend through RESTful APIs.
- Backend (Node.js with Express): Handles API requests, business logic, and authentication using JWT for secure access. It interacts with the MongoDB database for storing and retrieving user and group data.
- Database (MongoDB): Stores user accounts, groups, members, and expenses, ensuring scalable and efficient data management.

ARCHITECTURE

Components Interaction and High-level Architecture:

- **Frontend**: React.js (Client-side)
- **Backend**: Node.js with Express (Server-side)
- **Database**: MongoDB (For persistent storage)
- **Authentication**: JWT (JSON Web Tokens for secure login)



DETAILED DESIGN

Frontend Design(React.js)

- **Components:**
    1. Authentication:
        - **Login**: Form to input username/password. On successful login, a JWT token is received.
        - **Register**: Form to create a new account.

    2. Dashboard:
        - Displays a list of user's groups and options to create or join a group.
    3. Group Management:
        - Group creation with members.
        - Add, remove, and view group members.

    4. Expense Management:
        - Add and categorize expenses.
        - Track group and individual expenses.
        - Display detailed summaries.

- **State Management**:
    - **Redux** to manage global state, including user authentication, group data, and expenses.

- **UI/UX:**
    - **Tailwind CSS**: Provides a clean, modern, and responsive design across all devices.

Backend Design(Node.js with Express)

- **API Endpoints:**

    1. User:
        - `POST /api/v1/user/login`: Login endpoint to authenticate users.
        - `POST /api/v1/user/register`: Register endpoint for new users.
        - `POST /api/v1/user/logout`: Logout endpoint to authenticate users.
        - `GET/api/v1/user/checkLoggedIn`: Check endpoint for logged in status.
        - `GET/api/v1/user/get-all-groups`: Check endpoint for getting user groups.
    2. Groups:
        - `POST /api/v1/groups/create-group`: Create a new group.
        - `GET /api/v1/get-group-details`: List user's groups details.
        - `DELETE /api/v1/groups/delete-group`: Delete a member from a group.
        - `POST /api/v1/groups/add-member-to-group`: Add a member to a group.
    3. Expenses:
        - `POST /api/v1/expense/add-expense`: Add a new expense.
        - `GET /api/v1/expense/get-all-expenses`: Get all expenses for the user or a specific group.
        - `DELETE /api/v1/expense/delete-expense`: Delete an expense.
    4. JWT Authentication:
        - Middleware to validate tokens and secure endpoints.

- **Database Schema:**
    - **Users**: Store user profile information.
    - **Groups**: Store group details and members.
    - **Expenses**: Store details of each expense.

- **Technologies Used:**
    - **Node.js:** JavaScript runtime environment.
    - **Express.js:** Web application framework for building APIs.
    - **MongoDB:** NoSQL database for data storage.
    - **Mongoose:** ODM(Object Data Modeling) library for MongoDB.
    - **JWT:** JSON Web Tokens for authentication.
    - **Bcrypt:** Library for hashing passwords.
    - **Nodemailer** : Uses SMTP protocol to send emails.

- **Project Structure:**
    - **index.js:** Entry point of the server application.
    - **middleware/:** Contains middleware functions, including Authentication.
    - **models/:** Mongoose schemas for User and Article models.
    - **routes/:** Defines APIs endpoints for authentication, users and articles.
    - **connect.db.js:** Controller to connect with database.

DATABASE SCHEMA EXPLANATION

- **Users: (user.model.js)**
  - userName (String,required) : User's full name.
  - userEmail (String, required,unique) : User's email ID
  - phoneNumber(Number) : User's phone number
  - password(String, required) : User's password

- **Groups: (group.model.js)**
  - **Member Schema:**
    - memberId(Object,required) : Group member's ID.
    - memberName(String, required) : Group member's name.
    - memberEmail(String, required) : Group member's email.
    - memberRole(String) : Owner or member.

  - **Group Schema:**
    - groupName(String, required) : Group's name.
    - groupOwner(Object, required) : Group owner.
    - members(Array, required) : Group members array.
    - groupDescription(String) : Description of the group.

- **Expenses: (expense.model.js)**
  - **Member Schema:**
    - memberId(Object,required) : Group member's ID.
    - memberName(String, required) : Group member's name.

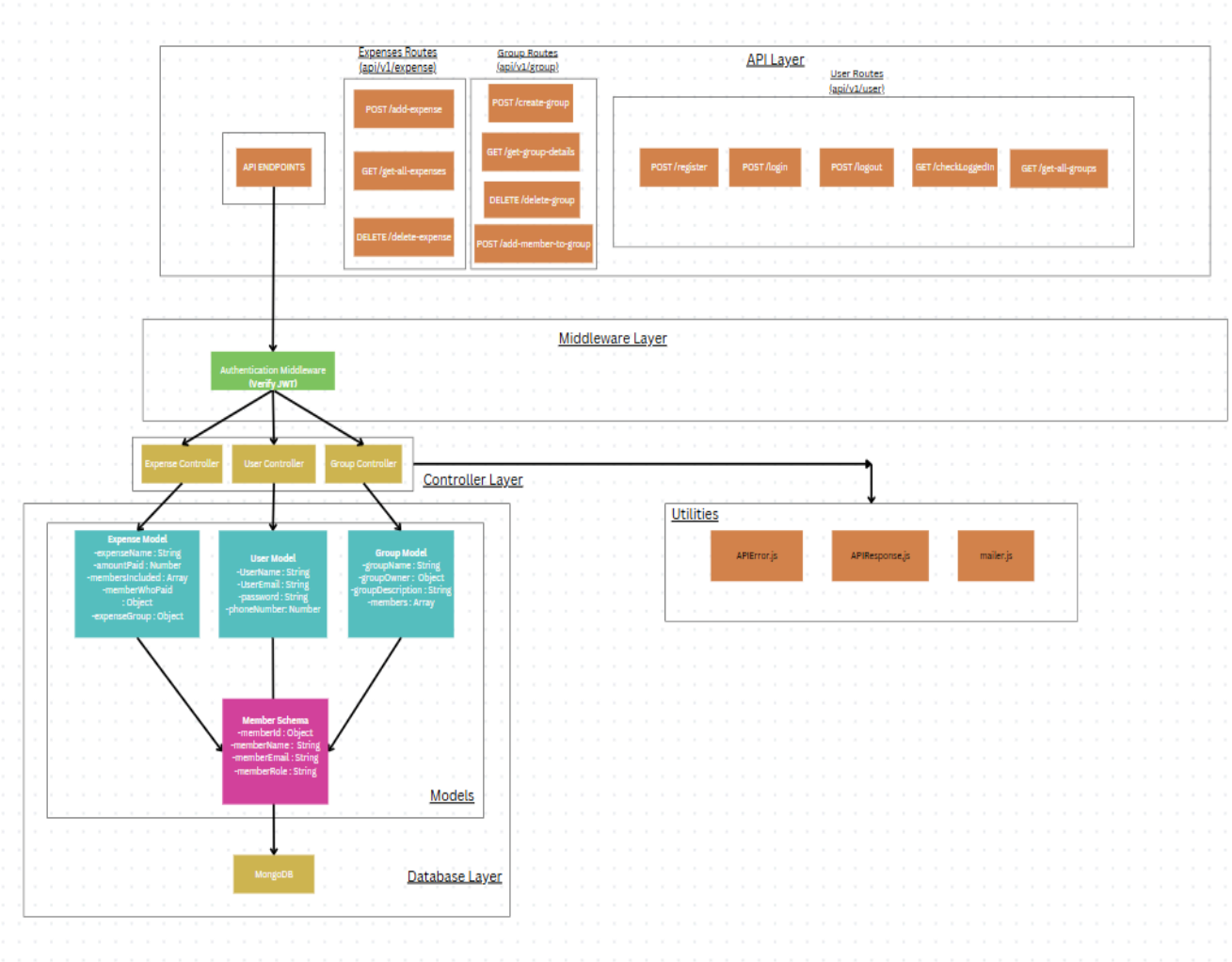
        - memberEmail(String, required) : Group member’s email.
        - memberRole(String) : Owner or member.

    - **Expense Schema:**
        - expenseName(String, required) : Expense’s name.
        - amountPaid(Number, required) : Expense amount paid by the owner.
        - membersIncluded(Array) : Members among whom the expense will split.
        - expenseGroup(Object) : Group to which the expense belongs.

MIDDLEWARE

This middleware function verifies a JWT from the accessToken cookie or Authorization header, decodes the user data, attaches it to the req object, and handles errors if the token is invalid or expired.

BACKEND DESIGN DIAGRAM



SECURITY CONSIDERATIONS

- **JWT Authentication:**
  - Secure login and token storage in the client (preferably in `cookies`).
  - Backend verifies JWT token on every request to protect sensitive data.

API DESIGN (RESTful)

- **/invite :** Store the invited user through email.
- **/api/v1/user :** Performs all user related operations like login, logout, register, get all groups etc.
- **/api/v1/group :** Performs all group related operations like group creation, deletion, add and delete members etc.
- **/api/v1/expense :** Performs all the expenses related operations like add expense, delete expense, get total expense etc.

DATA FLOW AND ERROR HANDLING

- **Data Flow**:
    1. User logs in and receives a JWT token.
    2. User, creates or joins a group.
    3. Expenses are added and categorized within the group.

- **Error Handling**:
    1. Implemented error handling to catch errors in the backend (e.g., invalid token, database errors).
    2. Frontend handles different types of errors and displays appropriate messages to the user.

DEPLOYMENT

- **Frontend:**
    - Deployed on Vercel for easy React.js hosting.

- **Backend:**
    - Deployed on AWS .

- **Database:**
    - MongoDB Atlas .

FUTURE ENHANCEMENTS

- **Progressive Web App (PWA)**: Making SplitIt available as an offline-capable web app for mobile users.
- **Payment Integration**: Integrate with payment gateways (like Paytm) to allow users to settle dues directly through the app.
- **Advanced Analytics**: Provide detailed insights into spending habits, trends, and group comparisons.
- **Group Chats** : The group members can have a real-time chatting experience.

CONCLUSION

SplitIt is designed to offer a seamless, efficient, and secure way for individuals and groups to manage expenses. The system leverages modern web technologies like the MERN stack and provides a responsive user interface to enhance user experience.