Imperial Mess Relay

Mess management made convenient

- Documentation

Contributors:
1. Ayush Vishwakarma
2. Manas Pathak
3. Vishal Pal

Table of Content

Problem Statement

Managing daily mess operations in a student environment can be challenging due to the complexity of handling various tasks such as recording complaints, keeping track of expenses, displaying daily menus, and analyzing related data. Traditional methods often involve manual processes, leading to inefficiencies, lack of transparency, and difficulty in keeping students informed and engaged. There was a need for a streamlined digital solution to manage these tasks more effectively and provide students with an easy-to-use platform for mess-related activities.

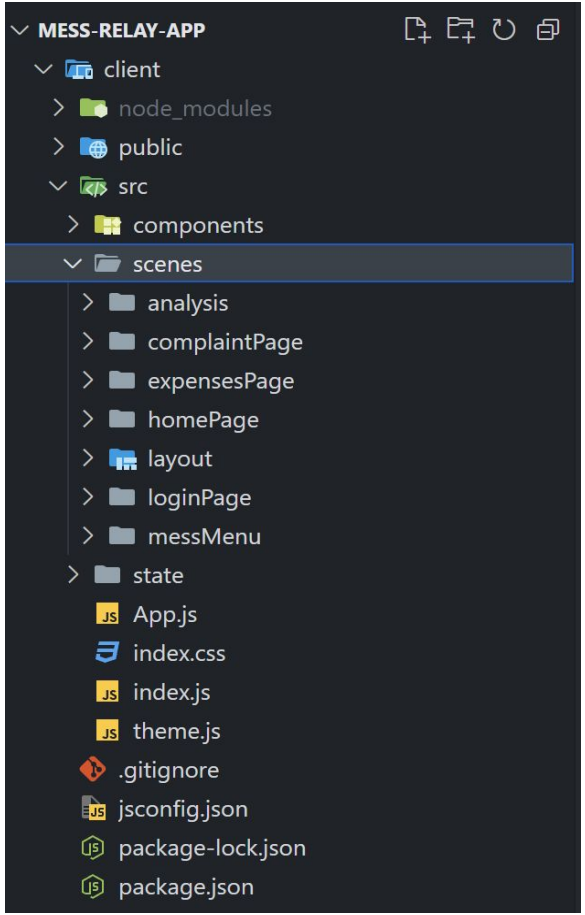
Solution

The Mess Management System is a digital platform developed using the MERN stack that addresses these challenges by providing a comprehensive solution. Here's how the project solves the problem:

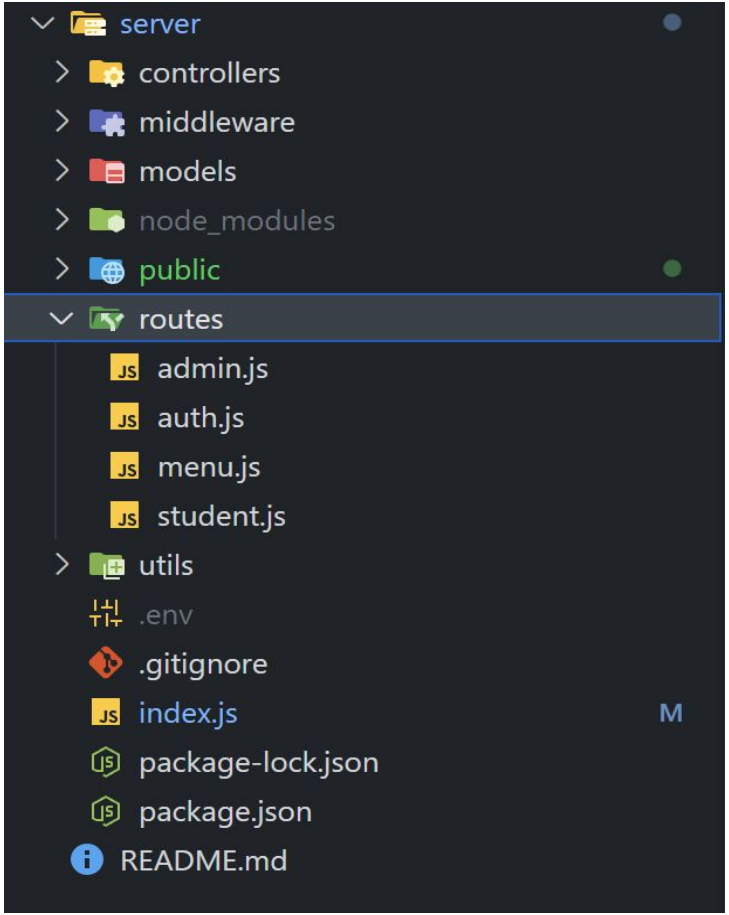
1. **Centralized Management**: The app offers a centralized platform where all mess-related operations can be managed efficiently. This reduces the reliance on manual processes and ensures that all data is stored and accessible in one place.
2. **User Authentication and Security**: The system provides secure user authentication, ensuring that only authorized students and administrators can access or modify data. This security measure helps protect sensitive information like expense records and complaints.
3. **Complaint Handling**: The application allows students to easily file complaints related to mess services. These complaints are stored in the database, and administrators can review and address them promptly, improving responsiveness and student satisfaction.
4. **Expense Tracking**: The app enables students to track their mess-related expenses, providing detailed reports and analysis. This feature helps students manage their finances better and gives them insights into their spending patterns.
5. **Mess Menu Management**: Students can access the daily or weekly mess menu through the app. This keeps them informed about the food offerings and helps them plan their meals accordingly.
6. **Data Analysis**: The platform includes features for analyzing data such as expenses, complaint statistics, and usage trends. This analysis can help both students and administrators make informed decisions, optimize mess operations, and improve service quality.

By implementing this solution, the project successfully addresses the need for a more efficient, transparent, and user-friendly mess management system, enhancing the overall experience for students and easing the administrative burden on mess staff.

Frontend Folder structure



Backend Folder structure



Login Page

Imperial Mess relay

Email

fakemail2@abc.com

Password

••••••••

LOGIN

Forgot password

Haven't registered? Sign Up here

Register Page

Imperial Mess relay

First Name

Dilian

Last Name

shaw

Role

Admin

Hostel

New Boys Hostel

Email

fakemail2@abc.com

Password

••••••••••••••

REGISTER

Already registered? Login

Home Dashboard

MESS RELAY

DILIAN SHAW

- Admin Dashboard
- Mess Menu
- Daily Expenses
- Analysis

CodeSangam

MNNIT

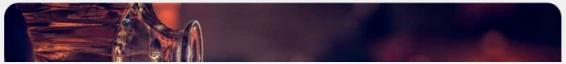
DASHBOARD

Welcome Dilian!

ALL COMPLAINTS

Poha did not have namkeen in sunday breakfast

Glasses are not washed properly and have patches of soap



NOTIFICATIONS

- Mess shall remain closed tomorrow
- Todays dinner is soya chaap temporarily
- students of other hostels(especially manas) will be punished for eating at other than their hostel mess.
- Mess will only run oon weeknds on paid service
- The sunday mess will have choole and rayta instead of daal
- asd

To notify...

NOTIFY

Expenses Details

MESS RELAY

- Admin Dashboard
- Mess Menu
- Daily Expenses
- Analysis

- CodeSangam
- MNNIT

DILIAN SHAW

Details of expenses

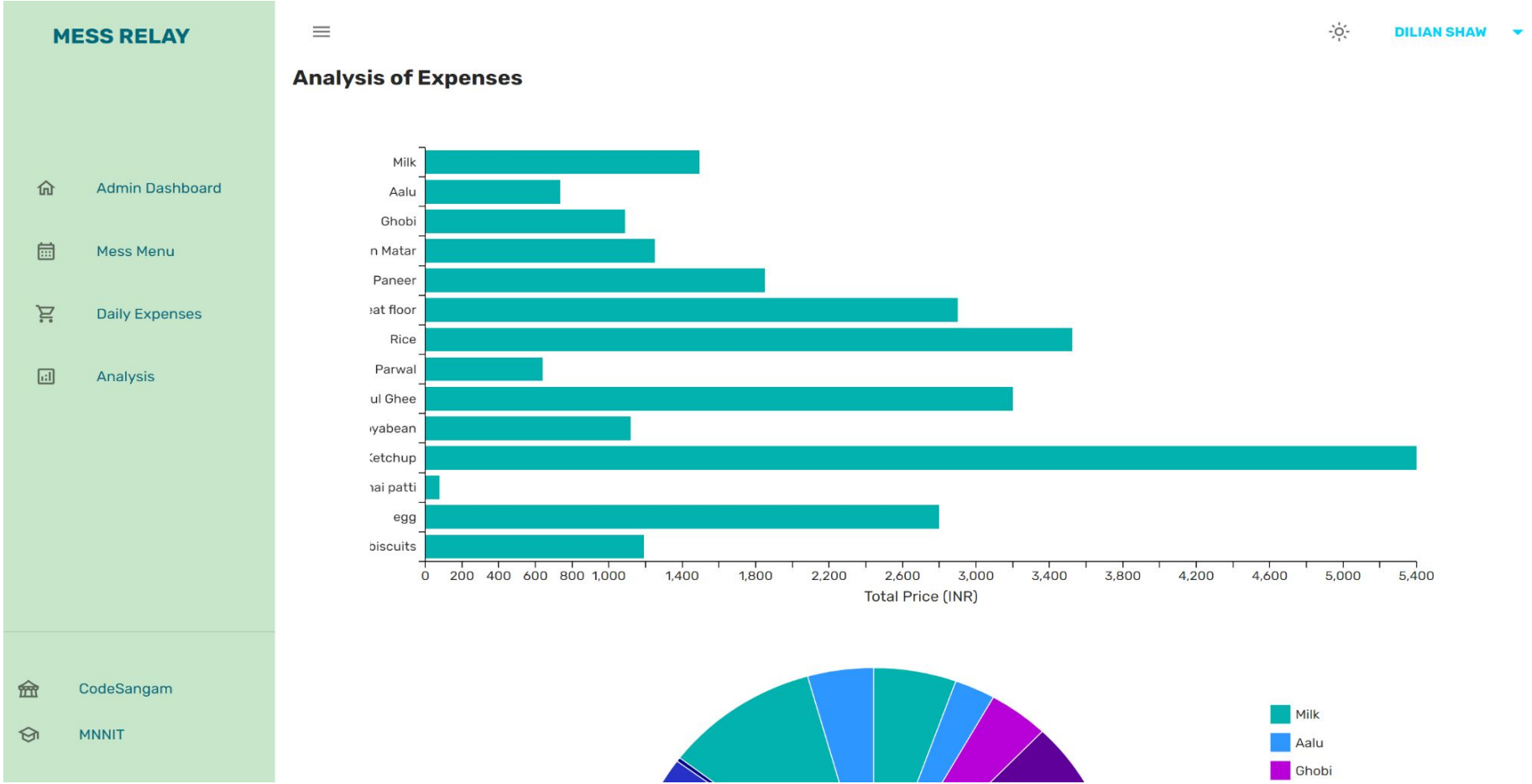
ID	Item name	Rate of item	Units ordered	Total Price
0	Milk	65	23	1,495
1	Aalu	21	35	735
2	Ghobi	34	32	1,088
3	Green Matar	125	10	1,250
4	Paneer	74	25	1,850
5	Wheat floor	290	10	2,900
6	Rice	235	15	3,525
7	Parwal	80	8	640

1–8 of 14

Add orders

Name of the Item | Rate per Unit | Number of Units | ADD

Expenses Visual Analysis



Mess Menu(admin)

MESS RELAY

- Admin Dashboard
- Mess Menu
- Daily Expenses
- Analysis

CodeSangam

MNNIT

DILIAN SHAW

MESS MENU

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
Breakfast	Breakfast	Breakfast	Breakfast	Breakfast	Breakfast
POHA , JALEBI, MILK AND SPROUTS	IDLI SAMBHAR WITH NARIYAL CHUTNEY	AALU PARATHA WITH CHAI	MEDU VADA WOTH SAMBHAR, BANANA AND SPROUTS	PANEER PARATHA & COFEE	SANDWICH, BREAD AND MILK
Lunch	Lunch	Lunch	Lunch	Lunch	Lunch
CHHOLE BHATOORE AND RICE WITH DAHI	MATAR PANEER, ROTI & RICE WITH RAYTA	MIX VEG , TOOR DAAL WITH ROTI & RICE	KADDI, AALU SABZI, RICE & ROTI	BHINDI PYAZA, KALI MASOOR, ROTI & RICE	KALI MASOOR, DAHI, MIC VEG, ROTI & RICE
Snack	Snack	Snack	Snack	Snack	Snack
SAMOSA & CHAI	BREAD PAKORA & COFEE	MAGGI AND MILK	FRUITS & BOURNVITA	PYAAZ PAKORA AND CHAI	CUTLET AND COFEE
Dinner	Dinner	Dinner	Dinner	Dinner	Dinner
LITTI CHOCKA , ARHAR DAAL AND RICE	RAJMA RICE & ROTI WITH ROASTED AALO	AALO SABZI , DAAL, ROTI & RICE	PANEER BHURJI, ROTI & RICE, DAAL AND CUSTURD	SOYABEAN, ARHAR DAAL, ROTI & RICE, KHEER	CHANA DAAL, BHINDI DO PYAZA ROTI & RICE, RABDI

DONE

Project Description

This project is a **Mess Management System** built using the MERN stack (MongoDB, Express.js, React.js, Node.js). The application aims to streamline the management of mess operations for students, allowing them to easily manage complaints, view mess menus, track expenses, and analyze related data.

Frontend (React.js)

- **Routing**: The app utilizes React Router for navigation, providing routes for various functionalities such as login, complaint registration, viewing the mess menu, tracking expenses, and data analysis.
- **Authentication**: The app ensures that only authenticated users can access certain features, redirecting unauthenticated users to the login page.
- **UI Components**: It includes reusable components and pages like `LoginPage`, `ComplaintPage`, `MessMenu`, `Expenses`, and `Analysis`.
- **State Management**: Redux is used for managing global state, including authentication status and theme settings.

Backend (Node.js, Express.js)

- **Express Server**: The backend server is built using Express.js, handling API routes for authentication, student complaints, mess menu management, and expense tracking.
- **File Uploads**: Multer is configured for handling file uploads, such as images for complaints or notifications.
- **Middleware**: Middleware such as Helmet for security, Morgan for logging, and Body-parser for parsing request bodies are utilized.
- **MongoDB**: The application uses MongoDB for data storage, with Mongoose handling the data models and operations.

Features

- **User Authentication**: Users can register and log in, with secure authentication managed by JWT tokens.
- **Complaint Management**: Students can file complaints, which are stored and managed via the backend.
- **Mess Menu Display**: The mess menu is accessible to users, allowing them to view daily or weekly menus.
- **Expense Tracking**: Students can track their expenses related to the mess, with the ability to add or view detailed expense reports.
- **Data Analysis**: Users can analyze mess-related data, such as expenses or complaint statistics, presented in a user-friendly format.

This project demonstrates the integration of various modern web technologies to create a robust and user-friendly application aimed at improving mess management for students.

Frontend Overview in Detail

The frontend of the Mess Management System is built using React.js, providing a dynamic and responsive user interface that ensures a seamless user experience. Below is a detailed overview of the key components and functionalities of the frontend:

1. Routing and Navigation

- **BrowserRouter**: The app uses `react-router-dom`'s `BrowserRouter` to handle client-side routing, allowing users to navigate between different pages without refreshing the entire page. This is critical for maintaining a smooth user experience.
- **Routes and Route**: The application defines multiple routes using the `Routes` and `Route` components. These routes map URLs to specific pages such as the login page, home page, complaint page, mess menu, expenses page, and analysis page. Nested routes are also used to manage layouts and access control.

2. Authentication and Access Control

- **useSelector for Authentication**: The app uses Redux for state management, where `useSelector` retrieves the authentication state from the global store. Specifically, it checks if a valid authentication token exists (`isAuth`), determining whether a user can access restricted pages or should be redirected to the login page.
- **Protected Routes**: Pages like the home page, complaint page, mess menu, expenses, and analysis are protected routes. If a user is not authenticated, they are redirected to the login page using the `Navigate` component, ensuring that only authorized users can access certain features.

3. Layout and Theming

- **Layout Component**: The `Layout` component serves as a wrapper for the protected routes, providing a consistent structure across the app. It typically includes common UI elements like headers, footers, and sidebars.
- **Theming with MUI**: The app uses Material-UI (MUI) for theming, ensuring a consistent and modern design. The theme is dynamically generated using the `createTheme` function and is influenced by the current mode (light or dark). The `ThemeProvider` component wraps the entire application, applying the theme across all components.
- **useMemo for Performance**: The theme is memoized using `useMemo`, ensuring that the theme object is only recalculated when necessary, improving performance by preventing unnecessary re-renders.

4. UI Components and Pages

- **LoginPage**: The entry point for users, where they can log in to access the application. The login page is simple yet secure, handling user authentication and redirecting authenticated users to the home page.
- **ForgotCred and ResetPassword**: These components allow users to recover or reset their credentials if they forget their login information. This functionality improves user experience and reduces the likelihood of account lockouts.
- **HomePage**: The central hub for authenticated users, providing an overview of the available functionalities like viewing the mess menu, lodging complaints, tracking expenses, and accessing analysis features.
- **ComplaintPage**: A dedicated page where users can submit complaints related to mess services. This page includes forms and possibly file upload capabilities, allowing students to describe issues and attach relevant images.
- **MessMenu**: This page displays the daily or weekly mess menu, allowing students to view meal options. It is likely updated regularly and formatted to present the information in an accessible and organized manner.
- **Expenses Page**: A detailed view where students can track their mess-related expenses. This page might include features like adding new expenses, viewing past transactions, and analyzing spending patterns.
- **Analysis Page**: This page provides data visualization and insights, such as trends in complaints or expenses. It helps users and administrators make informed decisions based on the data collected by the system.

5. Global State Management with Redux

- **State Handling**: The frontend uses Redux to manage global state, such as authentication status, user data, and theming preferences. This ensures that all components have access to the necessary data and can update or react to changes consistently across the app.
- **Integration with Backend**: Redux actions and reducers handle the storage of state of application and data coming form backend for tasks such as user login, fetching the mess menu, submitting complaints, and retrieving expense data. This integration ensures that the frontend stays synchronized with the backend, providing real-time updates and interactions.

6. Material-UI and CSS Styling

- **CssBaseline**: The app uses `CssBaseline` from MUI to provide a consistent baseline style across different browsers, ensuring that the application looks uniform and professional.
- **Custom Themes**: The application likely employs custom themes defined in the `themeSettings`, which might include primary and secondary colors, typography settings, and other UI elements tailored to the brand or specific requirements of the mess management system.

Overall, the frontend is designed to be user-friendly, responsive, and efficient, providing students and administrators with a comprehensive tool to manage mess-related activities. The combination of React.js, Redux, and Material-UI ensures that the application is both powerful and visually appealing.

Backend Overview

The backend of the Mess Management System is built using Node.js and Express.js, with MongoDB as the database, providing a robust and scalable architecture for managing the various functionalities of the application. Below is a detailed overview of the backend components and architecture:

1. Express.js Server

- **Express Application**: The backend server is powered by Express.js, a fast and lightweight web application framework for Node.js. The server handles HTTP requests, routes them to the appropriate handlers, and sends responses back to the client.
- **Middleware Integration**: The server utilizes various middleware to enhance security, handle requests, and log activities:
  - **Helmet**: Used to secure the Express app by setting various HTTP headers. It helps protect the app from well-known web vulnerabilities by configuring HTTP headers appropriately.
  - **Morgan**: A logging middleware that logs HTTP requests to the console, providing insights into the traffic and helping with debugging.
  - **Body-parser**: Middleware that parses incoming request bodies in a middleware before your handlers, available under the `req.body` property. It handles JSON and URL-encoded form data, allowing the server to process different types of incoming data.

2. API Routing

- **Modular Route Handling**: The server's routing is organized into modular files, each corresponding to different parts of the application:
  - **Auth Routes (`authRoutes`)**: Handles authentication-related endpoints such as user registration and login.
  - **Admin Routes (`adminRoutes`)**: Manages administrative functionalities like posting notifications and tracking expenses.
  - **Student Routes (`studentRoutes`)**: Includes endpoints related to student interactions, such as filing complaints and viewing the mess menu.
- **Route Organization**: The routes are defined in separate files and then integrated into the main server file (`index.js`). This modular approach makes the codebase more maintainable and easier to extend as new features are added.
-

3. File Uploads with Multer

- **Multer Configuration**: The backend uses Multer, a middleware for handling `multipart/form-data`, which is primarily used for uploading files. Multer is configured to store uploaded files (e.g., images for complaints, receipts for expenses) in a specific directory on the server (`public/assets`).
- **File Storage Strategy**: The file storage is configured to store files with their original names, ensuring that the file structure remains organized and easily accessible.

4. Database Management with MongoDB

- **MongoDB and Mongoose**: MongoDB serves as the database for the application, providing a NoSQL solution that is well-suited for handling the dynamic data structures used in the mess management system. Mongoose is used as an Object Data Modeling (ODM) library to interact with MongoDB, offering a straightforward way to define schemas and perform CRUD operations.
- **Data Models**: The backend likely includes various data models to represent different entities in the system, such as users (students and administrators), complaints, notifications, expenses, and mess menus. These models define the structure of the data and include methods for querying and manipulating the data in the database.
- **Connection Handling**: The server connects to the MongoDB database using the `mongoose.connect` method. Connection details, such as the MongoDB URI, are managed through environment variables (using `dotenv`), ensuring that sensitive information is not hardcoded into the application.

5. Authentication and Authorization

- **JWT Authentication**: The backend likely uses JSON Web Tokens (JWT) for user authentication. Upon successful login, the server generates a JWT and sends it to the client, where it is stored (usually in localStorage or cookies). This token is then included in subsequent requests to authenticate the user and authorize access to protected routes.
- **Role-based Access Control**: The backend may implement role-based access control (RBAC) to differentiate between students and administrators, ensuring that users can only access functionalities appropriate for their role. For example, only administrators might have access to certain routes like posting notifications or adding expenses.

6. Controller Functions

- **Auth Controller**: The `auth.js` controller handles user registration, login, and possibly password management functions. It validates user credentials, interacts with the database to create or retrieve user records, and generates JWT tokens for authentication.
- **Admin Controller**: The `admin.js` controller manages administrative tasks such as posting notifications and managing expenses. These functions may involve creating new records in the database, updating existing records, or processing uploaded files.
- **Student Controller**: The `student.js` controller handles student interactions, including filing complaints and possibly viewing mess menus or personal expenses. This controller ensures that students' actions are recorded in the database and appropriately handled.

7. Security Measures

- **Cross-Origin Resource Policy**: The backend uses Helmet to manage cross-origin resource policies, which control which resources can be loaded by the browser, adding an additional layer of security.
- **Environment Variables**: Sensitive information, such as database credentials and JWT secret keys, is stored in environment variables managed by the `dotenv` package. This ensures that these values are not exposed in the codebase.

8. Scalability and Performance

- **Modular Architecture**: The modular design of the backend, with separate routes and controllers, makes it easier to scale the application. New features or routes can be added without disrupting the existing codebase.
- **Optimized Requests**: The server uses middleware and optimized request handling to ensure that it can handle a large number of concurrent users, making it suitable for environments where multiple students and administrators may be using the system simultaneously.

Summary

The backend of the Mess Management System is a well-structured and secure environment that handles all the critical functionalities required for managing mess operations. It interacts seamlessly with the frontend to provide a complete and user-friendly experience for students and administrators, ensuring that all mess-related activities are efficiently managed and accessible at all times.

Future Scope and Improvements

1. **Mobile Application**: Develop a React Native app for on-the-go access to mess management features.
2. **Advanced Analytics**: Enhance the analytics dashboard with detailed reports, predictive insights, and data visualizations.
3. **Digital Payments**: Integrate with payment gateways for seamless bill payments and enable expense sharing.
4. **Enhanced Security**: Implement role-based access control (RBAC) and end-to-end encryption for sensitive data.
5. **Automated Complaint Management**: Use AI to categorize and prioritize complaints, and allow students to track and rate complaint resolutions.
6. **Dynamic Menu Updates**: Enable real-time menu updates and add nutritional information for better meal choices.
7. **Community Features**: Introduce social features for students to interact, review meals, and organize events.
8. **Scalability**: Implement load balancing, auto-scaling, and optimized database queries to handle growing user numbers.
9. **Multi-Language Support**: Add localization and internationalization for a diverse user base.
10. **Integration with Campus Systems**: Connect with other campus services and offer an API for external applications.