Solution to SaaS Plan with Stripe Integration

Introduction

I have developed a SaaS platform that empowers users to purchase and manage subscription plans using **Stripe payment integration**. The platform includes three subscription tiers: **Basic**, **Standard**, and **Plus**, each with distinct features like **role-based access control**, **organization management**, and user limitations based on the selected plan.

This project is designed with a clean architecture, focusing on scalability and modularity. I utilized **Node.js**, **Express**, **MongoDB**, **React**, **Redux**, and **Tailwind CSS** for the overall development. With seamless communication between the frontend and backend through APIs, I ensured the platform delivers an intuitive and robust user experience.

Tech Stack and Libraries

Backend

- **Node.js**: Backend runtime for building a scalable application.
- **Express.js**: Framework for handling routing and middleware.
- **MongoDB**: NoSQL database for managing users, organizations, and plans.
- **Libraries**:
    - **jsonwebtoken (JWT)**: For secure user authentication and token-based access control.
    - **Stripe**: For payment processing and webhook integration.
    - **dotenv**: For managing environment variables securely.
    - **cors**: To enable cross-origin resource sharing between the backend and frontend.
    - **bcrypt.js**: For securely hashing user passwords.
    - **Mongoose**: To model and interact with MongoDB collections.

Frontend

- **React.js**: Library for building the user interface with reusable components.
- **Redux**: For global state management, ensuring data consistency across the application.
- **Tailwind CSS**: For building a responsive and minimalist design with utility-first styling.
- **Libraries**:
    - **Redux Toolkit**: For managing slices and asynchronous actions effectively.
    - **React Router DOM**: For implementing role-specific routes and navigation.
    - **Axios**: For making API requests to the backend.

Backend Overview

The backend serves as the core of this platform, handling user authentication, role-based access, organization management, plan subscription, and secure payment processing.

Key Features

1. **Authentication and Authorization**
    - I implemented role-based access control for three user roles: **Super Admin**, **Admin**, and **User**.
    - JWT tokens ensure secure session management and authenticated API access.
    - Separate login and registration endpoints are provided for each role.
2. **Plan Management**
    - Super Admins can create, update, and delete plans.
    - Admins can purchase plans and manage users within the limits defined by the selected plan.
3. **Organization and User Management**
    - Organizations are linked to Admins and users, each tied to their respective plans.
    - Admins can add, update, and remove users while adhering to plan restrictions.
4. **Stripe Payment Integration**
    - I used Stripe Checkout for payment processing.
    - Webhooks handle payment confirmation, ensuring real-time updates in the system.

API Endpoints

Authentication

- **POST /api/auth/superadmin/register**: Registers a Super Admin.
- **POST /api/auth/superadmin/login**: Logs in a Super Admin.
- **POST /api/auth/admin/register**: Registers an Admin and creates an organization.
- **POST /api/auth/user/login**: Logs in Admin or User.
- **POST /api/auth/user/logout**: Logs out Admin or User.

Plan Management

- **POST /api/plans**: Super Admin creates a new plan.
- **PUT /api/plans/:id**: Super Admin updates an existing plan.
- **DELETE /api/plans/:id**: Super Admin deletes a plan.
- **GET /api/plans**: Fetches all available plans.

Organization Management

- **GET /api/organizations**: Fetch all organizations (Super Admin only).
- **GET /api/organizations/:id**: Fetch specific organization details.

- **PUT /api/organizations/:id/plans**: Admin purchases or updates a plan for their organization.

User Management

- **POST /api/users**: Admin adds a new user within plan limits.
- **PUT /api/users/:id**: Admin updates user details.
- **DELETE /api/users/:id**: Admin removes a user.
- **GET /api/users/:id**: Fetch user details.

Stripe Integration

- **POST /api/payments/checkout**: Initiates a Stripe checkout session.
- **POST /api/payments/webhook**: Handles payment confirmation and updates the database.

Frontend Overview

The frontend, built with React and Tailwind CSS, offers an interactive and responsive user interface. Role-specific dashboards, navigation, and payment functionality ensure that users can easily access the needed features.

Key Features

1. **Role-Based Dashboards**
    - **Super Admin**: Manage plans and monitor organizations.
    - **Admin**: Purchase plans, manage users, view organization details, and handle payments.
    - **User**: View organization details and active plans.
2. **Stripe Integration**
    - Admins can purchase plans through a checkout flow integrated with Stripe.
3. **Redux for State Management**
    - I used Redux Toolkit for handling global state.
    - Implemented slices for:
        - **authSlice**: Manages authentication state and JWT token storage.
        - **planSlice**: Tracks available plans and current plan details.
        - **userSlice**: Handles user-specific actions like profile updates and organization roles.
        - **paymentSlice**: Manages checkout sessions and payment status, and some other necessary slices.
4. **Service Layer**
    - Separate service files (e.g., authService.js, planService.js, paymentService.js) ensure clean API integration and reusability.

Pages and Navigation

- **Landing Page**: Public-facing page with plan browsing for potential users.
- **Dashboards**: Tailored views for Super Admin, Admin, and User roles.
- **Plan Management**: Displays plans available for purchase and organization upgrades.
- **Cart and Checkout**: Seamlessly integrated with Stripe.
- **Order History**: Displays past orders and active subscriptions.

Although I have completed the majority of the features, I haven't yet integrated some functionalities, like fetching all users in the frontend. However, I've already developed and thoroughly tested the backend APIs for these features using **Postman**, and they are working as expected. I'm planning to integrate these into the frontend soon to enhance the overall functionality and provide a seamless user experience.