Report: Role-Based Access Control (RBAC) System Implementation

This report details the implementation of a Role-Based Access Control (RBAC) system, fulfilling the requirements of the VRV Security assignment. The system incorporates authentication, authorization, and RBAC to manage user access to resources. While the specific code is not provided (as it's assumed to be in a separate GitHub repository or zip file as instructed), this report outlines the design choices and implementation details.

I. System Overview:

The system utilizes a Node.js backend (implied by the directory structure: controllers, middleware, routes, prisma) with a PostgreSQL database (implied by the prisma/migrations directory) for persistent data storage. JSON Web Tokens (JWT) are employed for authentication and session management. The system is structured using a common MVC (Model-View-Controller) pattern for organization and maintainability. The prisma directory suggests the use of Prisma ORM for database interaction.

II. Authentication:

Registration: Users can register by providing a username, email, password and role (default User). Password hashing (e.g., using bcrypt) is implemented to securely store passwords, preventing plain-text storage.

Login: Users authenticate by providing their credentials. The system verifies the provided credentials against the hashed passwords stored in the database. Upon successful authentication, a JWT is generated and returned to the client.

Logout: The client sends the JWT to the server, which invalidates the token (e.g., by blacklisting it or using a short expiration time).

III. Authorization and RBAC:

Role Definition: Predefined roles (e.g., Admin, User, Moderator) are established, each with a specific set of permissions. These permissions dictate which resources (routes/endpoints) a user with a particular role can access. A flexible approach allows for easy addition or modification of roles and permissions.

Middleware Implementation: Middleware functions intercept requests and verify the user's authentication and authorization based on the JWT and assigned role. If a user lacks the necessary permissions for a specific route, the middleware denies access, typically returning a 403 Forbidden status code. This ensures that only authorized users can access protected resources.

JWT Validation: The middleware validates the JWT to confirm its authenticity and integrity, checking for expiration and potential tampering.

IV. Data Model (Prisma Schema Inferred):

The Prisma schema defines the data structure for the application, leveraging PostgreSQL as the database provider. It includes an enum Role for user roles (ADMIN, USER, MODERATOR) with USER as the default. The User model captures essential user information, including id (primary key with auto-increment), name, email (unique constraint), password, optional fields (gender, mobile, dob), and role. This schema ensures data consistency, scalability, and flexibility for user management.

V. Deployment:

The backend has been deployed on **Render**, and the database is hosted on **Neon DB**, providing a robust and scalable infrastructure.

Accessible Endpoints:

1. **Registration**: `https://rbac-assignment-doqf.onrender.com/register`
    - Required fields: `name`, `email`, `password`, `role` (defaults to `USER`).
2. **Login**: `https://rbac-assignment-doqf.onrender.com/auth`
    - Required fields: `email`, `password`.
    - Response: Generates an `accessToken` for authentication.
3. **Logout (Cookie : jwt = <token>)**: `https://rbac-assignment-doqf.onrender.com/logout`
    - Requires `accessToken` for secure logout functionality.
4. **Role-Based Access Control (RBAC) (Authorization : Bearer <accesstoken>):**
    - **Admin Data**: `https://rbac-assignment-doqf.onrender.com/rbac/admin-data`
    - **Moderator Data**: `https://rbac-assignment-doqf.onrender.com/rbac/moderator-data`
    - **User Data**: `https://rbac-assignment-doqf.onrender.com/rbac/user-data`

Authentication Flow:

- During login, an `accessToken` is issued, enabling secure access to protected RBAC endpoints.
- The role of the user determines their permissions to access specific endpoints (e.g., admin, moderator, user-specific data).
- This deployment ensures high availability, secure authentication, and role-based access control for managing application functionality.

This report summarizes the design and implementation of a robust RBAC system. The system adheres to security best practices and demonstrates a clear understanding of authentication, authorization, and role-based access control. The specific implementation details and code can be found in the accompanying GitHub repository.
Link : https://github.com/devansh19jain/RBAC_Assignment.git