APPLICATION ARCHITECTURE AND FLOW

NAME: Navin Krishna

EMAIL: tnkrishnank@gmail.com

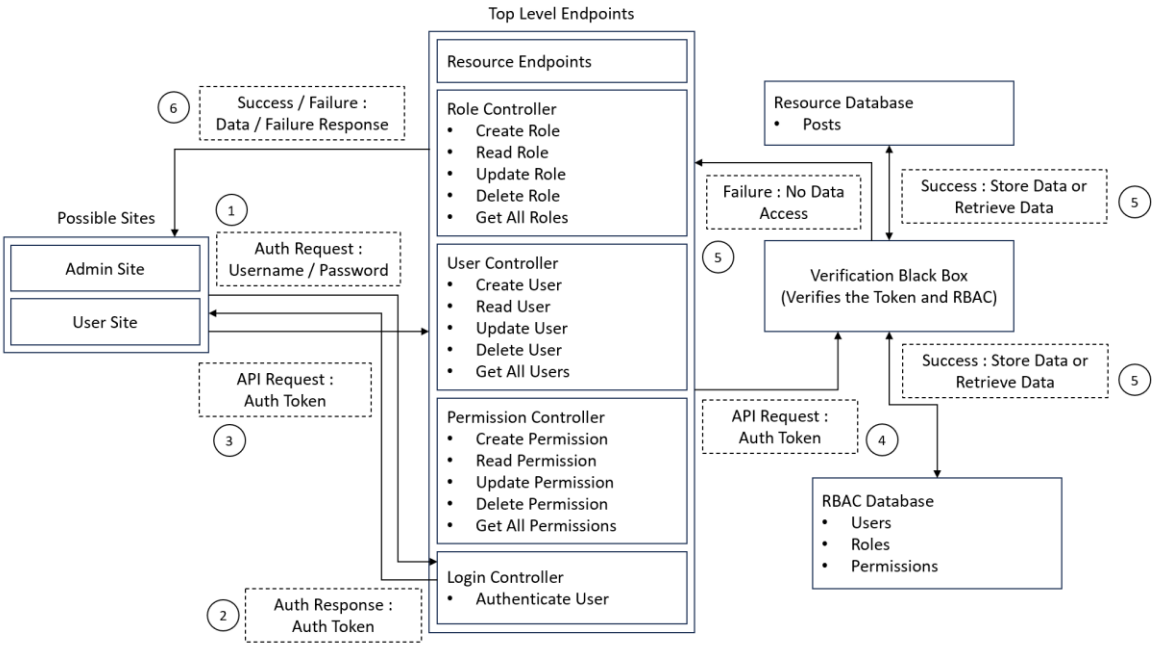
PROJECT LINK: https://github.com/tnkrishnank/RBAC-Node-Express-React

TECH STACK OVERVIEW:

This application follows a MERN-stack style architecture. Below is the tech stack used for the application development.

- **Frontend:** React.js
- **Backend:** Node.js with Express.js
- **Database:** MongoDB Atlas (Cloud)
- **Authentication:** JSON Web Tokens (JWT)
- **Mail Service:** Nodemailer
- **Endpoint Testing:** Postman

ARCHITECTURE:



BACKEND FLOW:

1. User Authentication Request

- Users access either the Admin Site or User Site.
- They send an authentication request with Username and Password.
- This request is handled by the Authenticator at the backend.

2. Authentication Response

- The Login Endpoint verifies the credentials:
    - If correct: an Auth Token (JWT) is generated and returned to the client.
    - If incorrect: an error response is sent.
- The Signup Endpoint checks for username or email already exists in the database, and sends a verification mail to the user if the password satisfies the constraints.
- The user can then click on the link to verify their account and then login.
- This Auth Token is necessary for all subsequent secure API interactions.

3. API Request with Auth Token

- Once authenticated, the client makes further API requests to backend endpoints.
- These requests include the Auth Token in the Authorization Header.

4. Token and RBAC Verification

- Before allowing access to protected resources:
- The Verification Black Box middleware validates the JWT Token.
- It then checks the user's roles and permissions against the RBAC Database (Users, Roles, Permissions).
- If verification fails, access is denied immediately.

5. Database Access Based on Permissions

- After successful verification:
    - The request is allowed to interact with the Resource Database or the RBAC Database.

    - The Resource Database contains Posts in this project and RBAC Database contains Roles, Permissions and Users details.
- Flow:
    - Success: Data is either retrieved from or stored in the database.
    - Failure: If the user lacks necessary permissions, a "No Access" or "Unauthorized" error is returned.

6. Response to Client

- A Success Response (with data) or a Failure Response (with error details) is sent back.
- The client then renders or reacts based on this response.

Note:

- A Postman Collection JSON file is available inside the backend directory.
- Developers can import this collection into Postman and directly test all the backend API endpoints.

FRONTEND FLOW:

1. Application Entry Points

- The app has two main user flows:
    - Admin Portal: /admin
    - User Portal: / or /signup
- Users are routed based on their role after login.

2. Authentication Handling

- On login/signup, the JWT Token is received from the backend and stored in localStorage.
- Every time the user accesses protected routes:
    - The frontend checks if a valid token exists in localStorage.
    - If no token is found, the user is redirected to the Login Page.

3. Route Protection

- Admin Pages:

    - Before loading admin pages, the frontend verifies the user by calling the /verify-admin backend API.
    - If the user is not an admin, they are redirected back to the normal user page (/blogs).
- 404 Handling:
    - If users navigate to an invalid route, the app redirects them to a custom 404 Not Found Page.

4. API Communication

- Axios is used for making API requests.
- Every request includes the Authorization Header with the Bearer token for secure access.
- Responses are handled carefully:
    - On token failure (expired/invalid), the user is logged out and redirected to login.
    - On successful response, data is displayed accordingly.

5. Post-Login Workflow

- After a successful login the user is redirected to:
    - Admin Portal (/admin/dashboard) if the user is admin.
    - User Portal (/blogs) if the user is a normal user.

6. Mail Verification

- On signing up, a verification email is sent to the user's email (handled by nodemailer at the backend).
- The frontend shows a message asking the user to verify their email.
- Only after email verification, the user can login.