**Developer Guide for Audit Management Web Application**

**Introduction**

Welcome to the developer guide for the Audit Management Web Application. This application allows users to manage audits efficiently using React for the front end and Spring Boot for the back end.

**Table of Contents**

1. Getting Started
2. Core Concepts
3. Usage Examples
4. Common Use Cases
5. Best Practices
6. Troubleshooting
7. FAQs
8. Conclusion

**Getting Started**

**Prerequisites**

* **Node.js**: Version 14 or higher
* **Java**: Version 17
* **Maven**: For building the Spring Boot application
* **Database**: MySQL or MSSQL

**Installation**

**1. Clone the Repository**

**bash**

git clone https://github.com/azhar-acnovate/audit-manager-server.git

2. **Set-Up the Backend (Spring Boot)**

Configure your database connection insrc/main/resources/sit.properties:

spring.datasource.url= jdbc:sqlserver://localhost:1433;encrypt=true;trustServerCertificate=true;databaseName=audit\_manager\_db

spring.datasource.username=${AUDIT\_MANAGER\_DB\_USER\_NAME}

spring.datasource.password=${AUDIT\_MANAGER\_DB\_PASSWORD}

Build and run the Spring Boot application:

mvn clean install

mvn spring-boot:run

**3. Set Up the Frontend (React)**

* Navigate to the frontend directory:

git clone  https://github.com/azhar-acnovate/audit-manager-client.git

* Install dependencies: **npm install**
* Run the React application**: npm start**

## Core Concepts

### Key Terminologies

* **User Roles**: access levels (Admin).
* **API Endpoints**: Swagger defined in the Spring Boot application to interact with the database.

Architecture Overview

The application follows a **Microservices Architecture** with a React front end communicating with a Spring Boot back end via RESTful APIs.

**Usage Examples :**

**Create Users:**

To create users, send a POST request:

baseUrl = "/user";

 createUser: async (data) => {

    try {

      return await RestService.CreateData(baseUrl, data);

    } catch (error) {

      console.error("Error creating user:", error);

      throw error;

    }

  },

const CreateData = async (path, data) => {

  const encodedCredentials = localStorage.getItem('encodedCredentials');

  const config = {

    headers: {

      'Authorization': `Bearer ${encodedCredentials}`

    }

  };

  try {

    const response = await axiosInstance.post(path, data, config);

    return response.data;

  } catch (error) {

    console.error(`Error creating ${path}:`, error);

    return error.response.data

  }

};

**Get all Users:**

To get list of users, send a GET request:

findAll: async () => {

    try {

      return await RestService.GetAllData(baseUrl);

    } catch (error) {

      console.error("Error fetching all users:", error);

      throw error; // Handle error as needed

    }

  },

  findPagable: async (pageNo = 0) => {

    try {

      return await RestService.GetAllData(`${baseUrl}?size=${config.DEFAULT\_SIZE\_PAGE}&pageNo=${pageNo}`);

    } catch (error) {

      console.error(`Error fetching users for page ${pageNo}:`, error);

      throw error;

    }

  },

## Best Practices

* **Input Validation**: Always validate user inputs on both front end and back end.
* **Error Handling**: Implement robust error handling for API calls to enhance user experience.
* **JWT for Authentication**: OAUTH2.0 and JSON Web Tokens (JWT) for secure user authentication used.

## Troubleshooting

### Common Errors

* **Error 401**: Unauthorized access—ensure your JWT token is valid and included in the headers.
* **Error 500**: Internal server error—check server logs for details.

## FAQs

**Q: How do I install the application?**  
A: Follow the installation instructions in the Getting Started section.

**Q: What database is supported?**  
A: The application supports both MySQL and MSSQL.