Event Management System Documentation

Introduction

This documentation provides detailed instructions on how to set up and run the Event Management System built with Spring Boot, Keycloak for authentication, and PostgreSQL as the database. It also includes instructions on user registration, authentication, and accessing secured APIs.

Prerequisites

- Java Development Kit (JDK) 11 or later
- Apache Maven
- PostgreSQL
- Keycloak

Setting Up the Application

1. Clone the Repository

```
git clone <repository-url> cd <repository-directory>
```

2. Configure PostgreSQL

- Create a PostgreSQL database named `event_db`.
- Create a user named `event_user` with password `<any>`.
- Grant all privileges on the `event_db` database to `event_user`.

3. Configure Keycloak

- Download and install Keycloak from Keycloak Downloads.
- Start Keycloak server:
- Access Keycloak Admin Console at `http://localhost:8080/auth`.

Create Realm

1. Create a new realm named `event-realm`.

Create Client

1. In the `event-realm`, create a client named `event-client`.
2. Set the client protocol to `openid-connect`.
3. Set the Access Type to `confidential`.
4. Set the Valid Redirect URIs to `http://localhost:8081/*`.

Configure Roles

1. Create roles `ROLE_USER` and `ROLE_ADMIN` in the `event-realm`.

Configure Mappings

1. Configure the client `event-client` to use the roles created.
2. Map the roles to the client.

Update Keycloak Configuration

- Create a `keycloak.json` file with the following content and place it in the `src/main/resources` directory:

json

```
{ "realm"  "event-realm"  "auth-server-url"  "http://localhost:8080/auth"
"ssl-required"  "external"  "resource"  "event-client"  "credentials"  {
"secret" "erkLLtD7lrXx3JshIxq8NMK2gSV3Bevk" }, "use-resource-role-mappings"
true "confidential-port"  0  "principal-attribute"  "preferred_username" }
```

4. Configure Application Properties

- Update `src/main/resources/application.properties` with your database and Keycloak configurations:

properties

```
server.port=8081 spring.application.name=system
spring.datasource.url=jdbc:postgresql://localhost:5432/event_db
spring.datasource.username=event_user
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.security.oauth2.resourceserver.jwt.jwk-set-uri=http://localhost:8080/rea
lms/event-realm/protocol/openid-connect/certs
```

5. Build and Run the Application

sh

```
mvn clean install mvn spring-boot:run
```

User Registration, Authentication, and Accessing Secured APIs

Registering a New User

1. Send a POST request to `/api/auth/register` with the user details:

json

```
POST
http://localhost:8081/api/auth/register
Content-Type: application/json
{"username"  "newuser"  "password"  "password" }
```

Authenticating a User

1. Authenticate the user with Keycloak at `http://localhost:8080/auth/realms/event-realm/protocol/openid-connect/token` by sending a POST request with `username`, `password`, `client_id`, and `client_secret`.

json

```
POST
http://localhost:8080/auth/realms/event-realm/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded
client_id=event-client&client_secret=erkLLtD7lrXx3JshIxq8NMK2gSV3Bevk&grant_typ
e=password&username=newuser&password=password
```

2. Extract the `access_token` from the response.

Accessing Secured APIs

1. Include the `access_token` in the Authorization header when accessing secured endpoints.

Example: Accessing Events API

json

```
GET http://localhost:8081/api/events Authorization: Bearer <access_token>
```

API Endpoints

- **User Registration**: `POST /api/auth/register`
- **Get All Events**: `GET /api/events` (Roles: USER, ADMIN)
- **Create Event**: `POST /api/events` (Role: ADMIN)
- **Update Event**: `PUT /api/events/{id}` (Role: ADMIN)
- **Delete Event**: `DELETE /api/events/{id}` (Role: ADMIN)
- **Get User by ID**: `GET /api/users/{id}` (Roles: USER, ADMIN)
- **Update User**: `PUT /api/users/{id}` (Roles: ADMIN, USER)
- **Delete User**: `DELETE /api/users/{id}` (Role: ADMIN)

Error Handling

Custom error messages are defined in the `AppUserErrorMessages` enum. The global exception handler `GlobalExceptionHandler` handles various exceptions and returns appropriate responses.

Conclusion

Follow the above steps to set up and run the Event Management System. Use the provided endpoints to register users, authenticate, and access secured APIs. Ensure to handle errors appropriately using the provided error messages and exception handler.