**Week 4: Spring REST using Spring Boot 3**

**5. JWT-Hands-on Summary**

**Student Name: Jitesh Sen**

**Email: jiteshsen2002@gmail.com**

**Superset ID: 6387362**

* **Create authentication service that returns JWT**:

**🔍 What I Achieved:**

I secured my Spring Boot REST API using **JWT (JSON Web Token)** to ensure only authenticated users can access specific resources.

**🔐 How the Authentication Flow Works (Conceptually):**

**1️⃣ Authentication Endpoint (/authenticate):**

* I created a dedicated authentication endpoint where the user provides credentials (username/password).
* After successful authentication, the server **generates a JWT token** containing the user's information.
* This token is returned to the client for future requests.

**2️⃣ JWT Token Purpose:**

* The JWT token serves as a **proof of authentication**.
* It contains information like the username, issue time, expiry time, and is signed with a secret key.
* The client uses this token in future API calls instead of repeatedly sending username/password.

**3️⃣ Accessing Protected Resources:**

* The client includes the JWT token in the **Authorization header** using the Bearer scheme when making requests to secured endpoints (e.g., /countries).
* This removes the need for session-based authentication.

**4️⃣ Custom JWT Authorization Filter:**

* I implemented a **custom security filter** that intercepts all incoming requests.
* This filter looks for a **Bearer token** in the Authorization header.
* It **validates the JWT**:
  + Ensures it’s not tampered with (verifies the signature).
  + Checks if it’s expired.
* If the token is valid, the filter allows the request to proceed.
* If the token is invalid or missing, access is denied.

**🎯 Why JWT + Custom Filter is Industry Standard:**

✅ **Stateless Security:** No session storage needed on the server.  
✅ **Scalable:** Works well in distributed systems and microservices.  
✅ **Secure:** Prevents unauthorized access if the token is validated properly.  
✅ **Reusable Knowledge:** This pattern is widely used in modern API design.

**📥 Testing Outcome (Summary):**

|  |  |
| --- | --- |
| **Action** | **Result** |
| Call /authenticate | Received JWT token |
| Call /countries with JWT | Access granted (valid token) |
| Call /countries without JWT | Access denied (unauthorized) |
| Output | Screenshots |

**Step 1: Authenticate & Get Token**

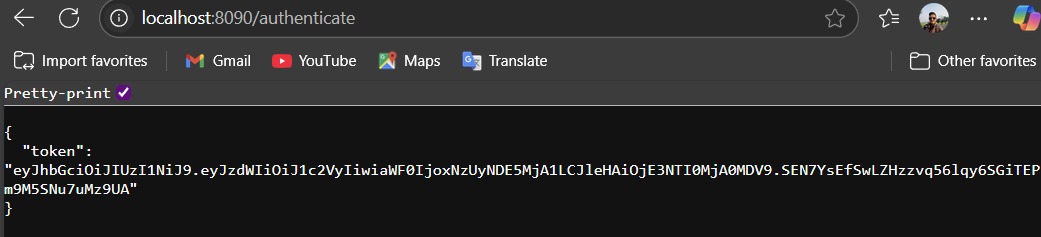
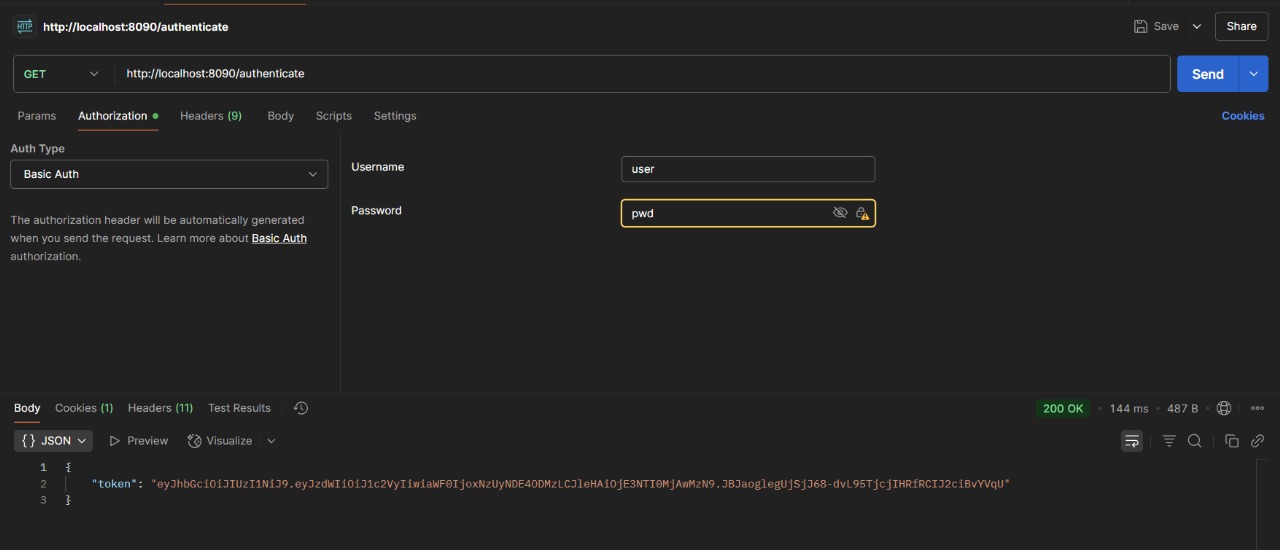
* **GET** http://localhost:8090/authenticate
* **Authorization Tab:** Basic Auth
  + Username: user
  + Password: pwd
* **Response:** Copy the token from JSON.

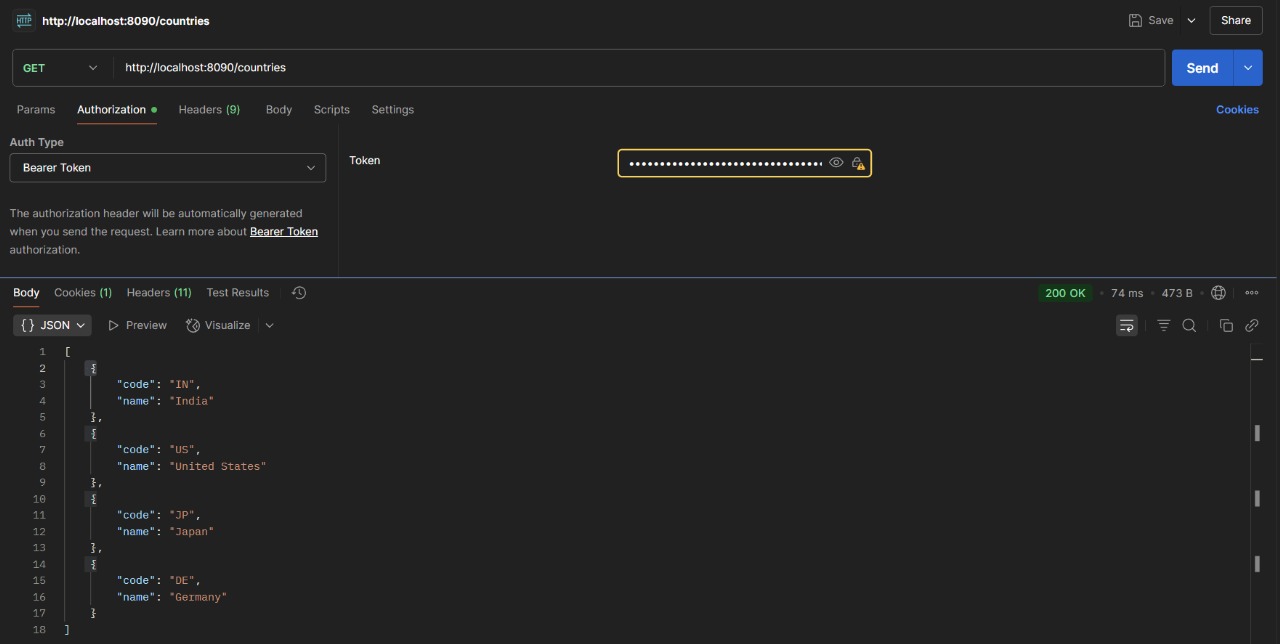
**Step 2: Access Protected Resource**

* **GET** http://localhost:8090/countries
* **Authorization Tab:** Bearer Token
  + Token: Paste the copied JWT
* **OR Headers Tab:**

Key: Authorization

Value: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNzUyNDE4ODMzLCJleHAiOjE3NTI0MjAwMzN9.JBJaoglegUjSjJ68-dvL95TjcjIHRfRCIJ2ciBvYVqU







**📝 Final Takeaway**

I implemented JWT authentication by issuing tokens after validating credentials and built a custom filter to ensure only requests with valid tokens can access protected APIs. This aligns with modern, stateless, and secure practices for REST API development.