Role-Based Authentication in Spring REST API- with JWT

Below is a **complete, end-to-end explanation and implementation guide** that answers **everything you asked**, in a **structured, interview-ready, enterprise-grade way**.

Sections:

1. **Why JWT over Basic Authentication**
2. **JWT Authentication – Theory**
3. **JWT Architecture & Control Flow**
4. **JWT Role-Based Login – Spring Boot Implementation**
5. **Testing JWT APIs using Postman**
6. **Consuming JWT APIs using HTML + Bootstrap 4 + jQuery + AJAX**

1⃣ Why Choose JWT Authentication Over Basic Authentication

❌ Problems with Basic Authentication

Issue	Explanation
Credentials sent every request	Username & password are Base64 encoded (not encrypted)
Session dependent	Uses `HttpSession` → scalability issues
Not mobile friendly	Hard to manage across devices
Logout not real	Browser keeps sending auth header
CSRF vulnerable	When session-based
Not suitable for microservices	Session sharing is complex

☞ **Basic Auth is good for demos, NOT for production**

✅ Why JWT is Preferred (Industry Standard)

Feature	JWT
Stateless	No session stored on server
Secure	Signed token (cannot be tampered)
Scalable	Perfect for microservices
Mobile/API friendly	Token-based
Logout controlled	Token expiry / blacklist
Role support	Roles embedded in token
Performance	No DB lookup on every request

☞ **JWT is the de-facto standard for REST APIs**

2️⃣ JWT Authentication – Theory

🔐 What is JWT?

JWT = **JSON Web Token**

It is a **self-contained token** that carries:

- User identity
- Roles
- Expiry time
- Signature

JWT Structure

```
HEADER.PAYLOAD.SIGNATURE
```

Example

```
eyJhbGciOiJIUzI1NiJ9
.eyJzdWIiOiJhZG1pbiIsInJvbGVzIjpbIlJPTEVfQURNSU4iXSwiZXhwIjoxNzAwMDAwMDB9
.xxxxxxxx
```

◆ JWT Parts Explained

1️⃣ Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2️⃣ Payload

```
{
  "sub": "admin",
  "roles": ["ROLE_ADMIN"],
  "exp": 1700000000
}
```

3️⃣ Signature

```
HMACSHA256(
  base64(header) + "." + base64(payload),
  secretKey
)
```

✔ Prevents token tampering

3️⃣ JWT Authentication – Control Flow

🔄 Authentication Flow

```
Client → /auth/login
        → Validate username/password
        → Generate JWT
        → Return JWT

Client → API request
        → Authorization: Bearer <JWT>
        → JWT Filter validates token
        → Extract user + roles
        → Set SecurityContext
        → Role check
```

📌 **No session, no cookies, no server memory**

4️⃣ JWT Role-Based Login – Spring Boot Implementation

📦 Dependencies

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>
```

🔑 JWT Utility Class

```java
@Component
public class JwtUtil {

    private final String SECRET = "mySecretKey123";

    public String generateToken(String username, List<String> roles) {

        Map<String, Object> claims = new HashMap<>();
        claims.put("roles", roles);

        return Jwts.builder()
                .setClaims(claims)
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(
                    new Date(System.currentTimeMillis() + 1000 * 60 * 30)
                )
                .signWith(Keys.hmacShaKeyFor(SECRET.getBytes()))
                .compact();
    }

    public Claims extractClaims(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(SECRET.getBytes())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }
}
```

🔐 JWT Authentication Filter

```java
@Component
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain chain
    ) throws ServletException, IOException {

        String header = request.getHeader("Authorization");

        if (header != null && header.startsWith("Bearer ")) {

            String token = header.substring(7);
            Claims claims = jwtUtil.extractClaims(token);

            String username = claims.getSubject();
            List<String> roles = (List<String>) claims.get("roles");

            List<GrantedAuthority> authorities = new ArrayList<>();
            for (String role : roles) {
                authorities.add(new SimpleGrantedAuthority(role));
            }

            UsernamePasswordAuthenticationToken auth =
                new UsernamePasswordAuthenticationToken(
                    username, null, authorities
                );

            SecurityContextHolder.getContext()
                .setAuthentication(auth);
        }

        chain.doFilter(request, response);
    }
}
```

🔒 Security Configuration (JWT – Stateless)

```java
@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    @Autowired
    private JwtFilter jwtFilter;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {

        http
          .csrf(csrf -> csrf.disable())
          .sessionManagement(session ->

session.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
          )
          .authorizeHttpRequests(auth -> auth
              .requestMatchers("/auth/login").permitAll()
              .requestMatchers("/admin/**").hasRole("ADMIN")
              .requestMatchers("/user/**").hasAnyRole("USER","ADMIN")
              .anyRequest().authenticated()
          )
          .addFilterBefore(jwtFilter,
              UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
}
```

🔑 Login Controller (JWT Issued Here)

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest req) {

    // validate username/password (DB)
    User user = userRepo.findByUsername(req.getUsername()).orElseThrow();

    List<String> roles = new ArrayList<>();
    for (Role role : user.getRoles()) {
        roles.add(role.getRoleName());
    }

    String token = jwtUtil.generateToken(user.getUsername(), roles);

    return ResponseEntity.ok(Map.of("token", token));
}
```

5️⃣ Testing JWT Authentication Using Postman

🔐 Step 1: Login

```
POST http://localhost:8080/auth/login
```

Body

```
{
  "username": "admin",
  "password": "admin123"
}
```

Response

```
{
  "token": "eyJhbGciOiJIUzI1NiJ9..."
}
```

🔐 Step 2: Access Secured API

Header

```
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9...
```

Example

```
GET /admin/dashboard
```

✓ 200 OK (ADMIN)
✕ 403 Forbidden (USER)

6️⃣ Consuming JWT APIs using HTML + Bootstrap + jQuery

🖥️ index.html

```
<input id="username">
<input id="password">
<button onclick="login()">Login</button>

<button onclick="callAdmin()">Admin</button>

<pre id="output"></pre>
```

📜 app.js

```
var token = "";

function login() {

  $.ajax({
    url: "/auth/login",
    type: "POST",
    contentType: "application/json",
    data: JSON.stringify({
      username: $("#username").val(),
      password: $("#password").val()
    }),
    success: function(res) {
      token = res.token;
      alert("Login success");
    }
  });
}

function callAdmin() {

  $.ajax({
    url: "/admin/dashboard",
    type: "GET",
    beforeSend: function(xhr) {
      xhr.setRequestHeader(
        "Authorization",
        "Bearer " + token
      );
    },
    success: function(data) {
      $("#output").text(data);
    },
    error: function(xhr) {
      alert(xhr.status);
    }
  });
}
```

JWT Authentication – Step-Wise Control Flow

□ STEP 1: Client sends Login Request

Client (UI / Postman / Browser) sends credentials:

```
POST /auth/login
{
  "username": "admin",
  "password": "admin123"
}
```

📌 No token yet
📌 Public endpoint (`permitAll()`)

□ STEP 2: Authentication Validation (Server)

1. Controller receives login request
2. Username fetched from database
3. Password validated using `BCryptPasswordEncoder`
4. Roles fetched from DB (`ROLE_ADMIN`, `ROLE_USER`)

✔ If invalid → `401 Unauthorized`
✔ If valid → proceed to token generation

□ STEP 3: JWT Token Generation

Server creates **JWT token** containing:

- Subject → username
- Claims → roles
- Expiry → e.g. 30 minutes
- Signature → secret key

```
HEADER.PAYLOAD.SIGNATURE
```

✔ Token is **signed** (cannot be tampered)
✔ No session is created

□ STEP 4: Token Returned to Client

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiJ9..."
}
```

📌 Client stores token:

- Browser → JS variable / localStorage
- Mobile → secure storage
- Postman → environment variable

□ STEP 5: Client Calls Secured API

Client sends token with every request:

```
Authorization: Bearer <JWT_TOKEN>
```

Example:

```
GET /admin/dashboard
```

📌 No username/password sent again
📌 Stateless request

□ STEP 6: JWT Filter Intercepts Request

`JwtAuthenticationFilter` executes **before controller**:

1. Reads `Authorization` header
2. Extracts JWT token
3. Validates signature & expiry
4. Extracts username & roles

❌ Invalid token → `401 Unauthorized`
✔ Valid token → proceed

□ STEP 7: SecurityContext is Created

Spring creates:

```
Authentication object
 ├─ principal → username
 ├─ authorities → roles
```

Then stores it in:

```
SecurityContextHolder (ThreadLocal)
```

📌 No HttpSession
📌 Exists only for this request

□ STEP 8: Authorization Decision

Spring checks endpoint rules:

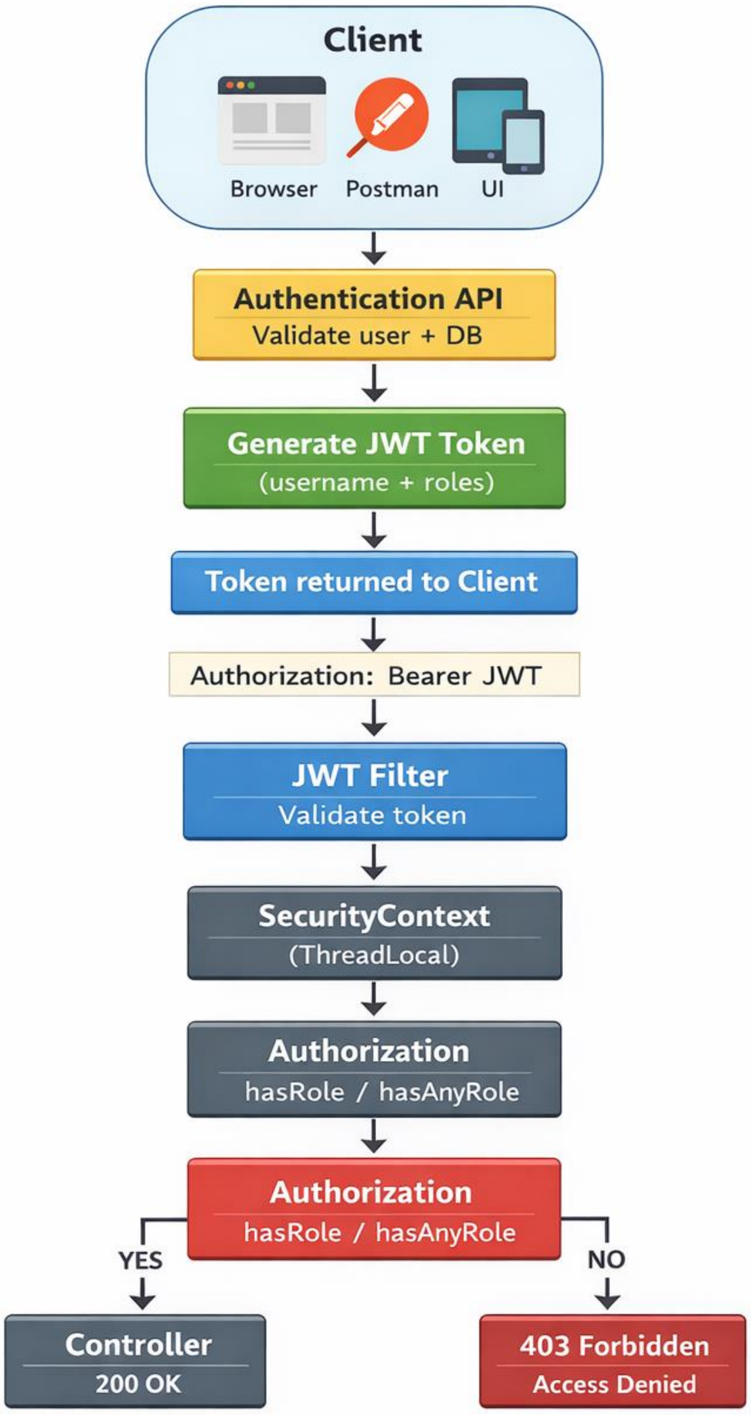
```
.hasRole("ADMIN")
```

Comparison:

```
Required: ROLE_ADMIN
Token has: ROLE_ADMIN
```

□ STEP 9: Final Result

Condition	Result
Role matches	✅ 200 OK
Role mismatch	❌ 403 Forbidden
Token missing / invalid	❌ 401 Unauthorized


Client
Browser
Postman
UI
Authentication API
Validate user + DB
Generate JWT Token
(username + roles)
Token returned to Client
Authorization: Bearer JWT
JWT Filter
Validate token
SecurityContext
(ThreadLocal)
Authorization
hasRole / hasAnyRole
Authorization
hasRole / hasAnyRole
YES
NO
Controller
200 OK
403 Forbidden
Access Denied