Spring Boot Security & JWT Authentication 🔐

- ✅ What is Spring Security?
- ✅ Why use JWT (JSON Web Token)?
- ✅ Implementing User Authentication in Spring Boot
- ✅ Securing REST APIs with JWT Tokens
- ✅ Adding Role-Based Access Control (RBAC)
- ✅ Protecting API Endpoints

1.1 What is Spring Security? 🔐

Spring Security is a powerful authentication and access-control framework. It helps:

- Protect APIs & applications from unauthorized access.
- Implement authentication mechanisms (JWT, OAuth, Basic Auth, etc.).
- Manage user roles & permissions.
- Prevent security threats like CSRF, CORS, and brute-force attacks.

1.2 Why Use JWT (JSON Web Token)? 🤔

JWT is a stateless authentication mechanism, meaning the server doesn't need to store session data.

🔹 Advantages of JWT:

- ✅ Secure – Uses digital signatures.
- ✅ Scalable – No need to store sessions in memory.
- ✅ Fast – Tokens are passed in HTTP headers.
- ✅ Works with microservices – Can be shared across services.

1.3 Setting Up Spring Security in Spring Boot

Step 1: Add Dependencies in pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

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

1.4 Creating a User Entity(@Entity)

```
import jakarta.persistence.*;
import java.util.Set;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String username;

    private String password;

    @ElementCollection(fetch = FetchType.EAGER)
    private Set<String> roles; // e.g., ["ROLE_USER", "ROLE_ADMIN"]

    // Getters and Setters
}
```

✅ Users table with roles and encrypted password.

1.5 Creating a User Repository(JpaRepository)

```
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

✅ Find users by username for authentication.

1.6 Password Encryption with Bcrypt

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

✅ Encrypts passwords before storing in the database.

1.7 Implementing JWT Token Generation

```
import io.jsonwebtoken.*;
import org.springframework.stereotype.Component;
import java.util.Date;
import java.util.function.Function;

@Component
public class JwtUtil {

    private String secretKey = "mySecretKey"; // Change this in production!

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 *
60 * 60)) // 1 hour validity
                .signWith(SignatureAlgorithm.HS256, secretKey)
                .compact();
    }

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T>
claimsResolver) {
        final Claims claims = Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(token)
                .getBody();
        return claimsResolver.apply(claims);
    }
```

```
    public boolean validateToken(String token, String username) {
        return (extractUsername(token).equals(username) &&
!isTokenExpired(token));
    }

    private boolean isTokenExpired(String token) {
        return extractClaim(token, Claims::getExpiration).before(new
Date());
    }
}
```

1.8 Implementing Authentication Service

```
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.Optional;

@Service
public class AuthService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private JwtUtil jwtUtil;

    public String registerUser(String username, String password) {
        User user = new User();
        user.setUsername(username);
        user.setPassword(passwordEncoder.encode(password));
        user.setRoles(Set.of("ROLE_USER")); // Default role
        userRepository.save(user);
        return "User registered successfully!";
    }

    public String authenticateUser(String username, String password) {
        Optional<User> userOptional =
userRepository.findByUsername(username);
        if (userOptional.isPresent()) {
            User user = userOptional.get();
            if (passwordEncoder.matches(password, user.getPassword())) {
                return jwtUtil.generateToken(username); // Return JWT Token
            }
        }
        throw new RuntimeException("Invalid Credentials");
```

```
    }
}
```

✅ Registers users and generates JWT tokens after login.

1.9 Creating Auth Controller (@RestController)

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private AuthService authService;

    @PostMapping("/register")
    public String register(@RequestParam String username, @RequestParam
String password) {
        return authService.registerUser(username, password);
    }

    @PostMapping("/login")
    public String login(@RequestParam String username, @RequestParam String
password) {
        return authService.authenticateUser(username, password);
    }
}
```

✅ Provides APIs for user registration and login.

1.10 Protecting API Endpoints using JWT Security

🔹 Create a Security Filter to Validate JWT Tokens

```
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticat
```

```
ionToken;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;
import java.util.Collections;

@Component
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        String token = request.getHeader("Authorization");
        if (token != null && token.startsWith("Bearer ")) {
            token = token.substring(7);
            String username = jwtUtil.extractUsername(token);
            if (username != null && jwtUtil.validateToken(token, username))
{
                UsernamePasswordAuthenticationToken auth = new
UsernamePasswordAuthenticationToken(
                        new User(username, "", Collections.emptyList()),
null, Collections.emptyList());
                SecurityContextHolder.getContext().setAuthentication(auth);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

✅ Validates JWT tokens before accessing secured APIs.