How do you handle API security in your application ? How can you make your API secure ? What are possible ways to do it in Springboot? Assume you are using Spring security 6.x+ ,then how will you do it ?

To secure APIs in a modern Spring Boot 3 + Spring Security 6.x environment, the standard approach is using **OAuth2 Resource Server** with **JWT** (stateless), combined with **Method Security** (`@PreAuthorize`) for fine-grained control.[1] [2]

Here is how I handle it, structured as a clear interview answer:

1. Core Security Strategy

"I handle API security by implementing a **stateless, token-based architecture** using **OAuth2 and JWT**. I use Spring Security 6.x as the framework, configuring it as a **Resource Server**. My strategy includes:

- **Authentication:** Verifying identity via JWT tokens issued by an Identity Provider (like Keycloak, Okta, or a custom Auth Service).
- **Authorization:** Enforcing access control at two levels:
  - **URL Level:** Using `SecurityFilterChain` to protect endpoints globally.
  - **Method Level:** Using `@EnableMethodSecurity` and `@PreAuthorize` for fine-grained business logic access."[3] [1]

2. How I implement it in Spring Boot 3 (Spring Security 6.x)

Step A: Dependency
"I add `spring-boot-starter-oauth2-resource-server`. This single starter handles JWT validation, parsing, and context setup automatically."[1]

Step B: Configuration (`SecurityFilterChain`)
"In Spring Security 6, we no longer extend `WebSecurityConfigurerAdapter`. Instead, we define a bean."[4]

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity // Replaces @EnableGlobalMethodSecurity in v6
public class SecurityConfig {
```

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        return http
            .csrf(csrf -> csrf.disable()) // Disable CSRF for stateless APIs
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPo
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/public/**").permitAll() // Public endpoints
                .anyRequest().authenticated() // Everything else requires a token
            )
            .oauth2ResourceServer(oauth2 -> oauth2.jwt()) // Enable JWT validation
            .build();
    }
}
```

Step C: Handling Roles (The "Tricky" Part)
"By default, Spring Security 6 maps JWT scopes to authorities (e.g., `SCOPE_read`). To use roles like `ROLE_ADMIN`, I configure a **JwtAuthenticationConverter**." [2] [1]

```
@Bean
public JwtAuthenticationConverter jwtAuthenticationConverter() {
    JwtGrantedAuthoritiesConverter grantedAuthoritiesConverter = new JwtGrantedAuthoritie
    grantedAuthoritiesConverter.setAuthorityPrefix("ROLE_"); // Prefixes roles
    grantedAuthoritiesConverter.setAuthoritiesClaimName("roles"); // Reads "roles" claim

    JwtAuthenticationConverter jwtConverter = new JwtAuthenticationConverter();
    jwtConverter.setJwtGrantedAuthoritiesConverter(grantedAuthoritiesConverter);
    return jwtConverter;
}
```

3. Making APIs Secure (Best Practices I Follow)

Beyond just "login," I enforce:

- **HTTPS Only:** I enforce TLS (SSL) so tokens aren't intercepted. [5] [6]
- **Fine-Grained Access:** I use `@PreAuthorize` on service methods.
  - *Example:* `@PreAuthorize("hasRole('ADMIN') or #username == authentication.name")` to ensure users can only edit their own profile. [7] [3]
- **Input Validation:** I use `@Valid` and JSR-380 annotations on DTOs to prevent injection attacks before data even reaches business logic. [6]
- **Rate Limiting:** I implement rate limiting (using libraries like Bucket4j or via API Gateway) to prevent DDoS.
- **CORS Configuration:** I strictly configure CORS to allow only trusted frontend domains, preventing unauthorized cross-origin calls. [6]

4. Interview Follow-Up: "What if you need Custom Auth?"

"If I can't use an external IDP and need custom auth, I implement a custom `JwtAuthenticationFilter` that intercepts the request, parses the Bearer token, validates it (signature + expiry), and manually populates the `SecurityContextHolder`. But in Spring 6, standard OAuth2 Resource Server is preferred for standard JWTs." [8] [2]

⁂

1. https://devlach.com/blog/java/spring-security-oauth2-resource-server
2. https://www.youtube.com/watch?v=clMl_jhQcR4
3. https://www.baeldung.com/spring-enablemethodsecurity
4. https://www.tothenew.com/blog/migrating-to-spring-security-6/
5. https://moldstud.com/articles/p-securing-restful-apis-with-spring-security-best-practices
6. https://escape.tech/blog/security-best-practices-for-spring-boot-applications/
7. https://docs.spring.io/spring-security/reference/6.0/servlet/authorization/method-security.html
8. https://blog.csdn.net/m0_60551898/article/details/130458451
9. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg
10. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg
11. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg
12. https://spring.io/projects/spring-security
13. https://spring.io/guides/gs/securing-web
14. https://www.geeksforgeeks.org/advance-java/securing-rest-apis-with-spring-security/
15. https://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works
16. https://www.youtube.com/watch?v=oeni_9g7too
17. https://stevenpg.com/posts/spring-boot-oauth2-multiple-resource-servers/
18. https://docs.spring.io/spring-security/reference/servlet/configuration/java.html
19. https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/index.html
20. https://www.geeksforgeeks.org/java/spring-security-filter-chain-with-example/
21. https://bootify.io/spring-security/rest-api-spring-security-with-jwt.html
22. https://docs.spring.io/spring-security/reference/servlet/authorization/method-security.html
23. https://snyk.io/blog/spring-boot-security-best-practices/
24. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg