Spring Security

**🛡️ Spring Security: Basic Default Features**

Spring Security is a powerful and customizable **authentication and access-control framework**. It is the **standard security framework** for Spring-based applications.

**✅ 1. Default Login Form**

* When Spring Security is on the classpath (spring-boot-starter-security), it **automatically protects all endpoints**.
* On accessing a secured endpoint (like /), you’re redirected to a **default login form** at /login.

🧩 **Username:** user  
🔐 **Password:** Auto-generated on app startup (printed in logs)

Example:

Using generated security password: 3c9e0b5e-b12d-4e98-b0f9-34c2c749fc0a

**✅ 2. Default Authentication**

* Uses an in-memory user store.
* A default user with username user and random password is created.
* Authentication is session-based (cookie: JSESSIONID).

You can override with:

spring.security.user.name=admin

spring.security.user.password=admin123

**✅ 3. All Endpoints Are Secured by Default**

* Every HTTP endpoint is secured (requires authentication).
* By default, **no endpoint is publicly accessible**, unless explicitly configured.

Example:

http.authorizeHttpRequests(auth ->

auth.anyRequest().authenticated()

);

**✅ 4. Form-Based Login & Logout**

* Enabled by default.
* Login page at /login
* Logout URL at /logout (HTTP POST)
* On logout:
  + Session is invalidated.
  + Cookies are cleared.

**✅ 5. Basic Authentication Support**

* HTTP Basic Auth is also available.
* Can be enabled explicitly for APIs:

http.httpBasic();

Then, Postman/cURL can send:

Authorization: Basic base64(username:password)

**✅ 6. CSRF Protection**

* Enabled by default for state-changing requests (POST, PUT, DELETE).
* A **CSRF token** must be included in such requests.
* Works with form submissions or via headers.

To disable (for APIs):

http.csrf().disable();

**✅ 7. Session Management**

* Uses **HTTP session (JSESSIONID)** by default.
* Session is created upon login and reused for authenticated requests.
* You can configure stateless sessions (for JWT or REST APIs):

http.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

**✅ 8. Password Encoding**

* Spring Security uses **bcrypt** by default for encoding passwords.
* You can use it like:

PasswordEncoder encoder = new BCryptPasswordEncoder();

String encodedPassword = encoder.encode("admin123");

For user authentication, compare encoded password with stored value.

**✅ 9. Method-Level Security (Optional, Explicit)**

* Not enabled by default, but commonly used.
* Add @EnableMethodSecurity and then use:

@PreAuthorize("hasRole('ADMIN')")

public void adminOnlyMethod() { }

**✅ 10. Security Filters**

Spring Security works via a **chain of servlet filters** that:

* Check authentication
* Apply CSRF rules
* Handle login/logout
* Enforce access control
* More...

**🚀 Summary Table**

|  |  |
| --- | --- |
| Feature | Default Behaviour |
| Authentication | In-memory with user/auto-password |
| Form Login | /login, basic login form |
| CSRF Protection | Enabled for all state-changing requests |
| URL Access Control | All URLs require authentication |
| Logout | POST to /logout |
| Session Management | Session-based (JSESSIONID) |
| Password Encoding | Uses BCrypt by default |
| Method-Level Security | Off by default; opt-in via annotations |
| Basic Auth for APIs | Supported; can be enabled manually |
| Filter-Based Architecture | Built on servlet filters to secure HTTP requests |

Customization of Spring Security :

@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http.build();  
 }  
}

This will implement default Security behaviour in Spring Boot

1. CSRF is enabled
2. All requests are denied unless authenticated
3. No login mechanism is enabled (e.g., no httpBasic() or formLogin() configured)
4. No user is defined, unless provided in application.yml/application.properties

CSRF 🡪 Cross-Site Request Forgery

Attribute name = \_csrf (type = hidden)

By default, spring security give csrf security enabled. Due to this without the csrf token anyone can send GET request but not POST, PUT, DELETE. This secures the application from any malicious or hidden request.

To disable csrf security:

@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http  
 .csrf(AbstractHttpConfigurer::disable)  
 .build();  
 }  
}

After this anyone can send GET, POST, PUT, DELETE request only with the username and password.

To implement a basic auth for any request:

@EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http  
 .csrf(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(auth-> auth.anyRequest().authenticated())  
 .formLogin(Customizer.*withDefaults*())  
 .httpBasic(Customizer.*withDefaults*())  
 .build();  
 }  
}

This implements a basic auth for any request (authorizeHttpRequests(auth-> auth.anyRequest().authenticated()).

Also, formLogin() and httpBasic() is added to authenticate the credentials without these all response will be 403 : Forbidden.

401: Unauthorized 🡪 Application trying to authorize but credentials are wrong.

403: Forbidden 🡪 Application is rejected to authorize credentials.

To implement HttpSession STATELESS :

@EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http  
 .csrf(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(auth-> auth.anyRequest().authenticated())  
 .httpBasic(Customizer.*withDefaults*())  
 .sessionManagement(session->session.sessionCreationPolicy(SessionCreationPolicy.*STATELESS*))  
 .build();  
 // .formLogin(Customizer.withDefaults()) // As stateless session policy is used to this is not needed.  
 }  
}

Stateless Session Policy : For every session credential requires