CSRF

https://docs.spring.io/spring-security/reference/servlet/exploits/csrf.html
https://www.baeldung.com/spring-security-csrf

If you're not using HTML for views, but instead relying on other technologies or frameworks, such as a JavaScript frontend framework like React, Angular, or Vue.js, or a mobile application framework like React Native or Flutter, the approach to CSRF protection may vary.

When you're not using HTML for views generated by a server-side template engine like Thymeleaf or JSP, you typically need to handle CSRF protection manually in your frontend code.

Here's a general approach:

1. **Generate CSRF Token**: Your backend (Spring Security) should still generate CSRF tokens. These tokens are typically stored in session or cookies. When your frontend application initializes, it should request this token from the backend.

2. **Include CSRF Token in Requests**: Once your frontend application has obtained the CSRF token, it should include it in subsequent requests to the backend. This usually involves including the CSRF token in the headers of your HTTP requests. For example, in a JavaScript application, you might set the CSRF token as a custom header in your AJAX requests.

3. **Validate CSRF Token**: On the backend (Spring Security), you need to ensure that incoming requests include a valid CSRF token. Spring Security provides mechanisms to validate CSRF tokens included in HTTP headers, so you would configure it to look for the CSRF token in the headers of incoming requests.

4. **Handling CSRF Token Expiration**: CSRF tokens typically have a limited lifespan for security reasons. When the token expires, you need to handle refreshing it on the frontend and ensuring that subsequent requests include the new token.

In summary, when not using HTML for views, you need to manually handle CSRF protection in your frontend code by generating, including, and validating CSRF tokens in your HTTP requests. However, the specifics of implementation may vary depending on the frontend framework or technology you're using.

No, you typically don't manually write CSRF tokens into HTML when using Spring Security. Spring Security provides built-in protection against CSRF (Cross-Site Request Forgery) attacks. It automatically generates and includes CSRF tokens in your HTML forms when using Thymeleaf or JSP (JavaServer Pages) with the appropriate Spring Security tags.

When a form is rendered using Thymeleaf or JSP with Spring Security tags, the CSRF token is automatically included in the form as a hidden field. When the form is submitted, Spring Security validates the CSRF token on the server-side to ensure that the request originated from the same application and not from a malicious site.

You typically just need to ensure that your Spring Security configuration is properly set up to enable CSRF protection, and Spring Security will handle the rest for you. This includes generating and validating CSRF tokens as part of the request/response cycle.