**Day 9**





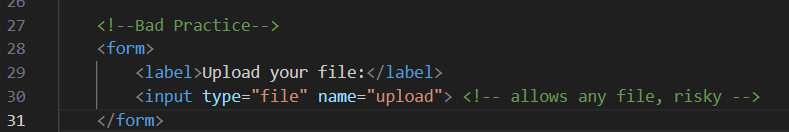
**“Web Development + Security”**

**Security Practices for Forms and Input tags:**

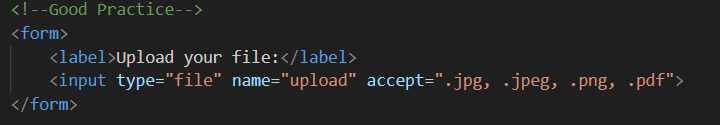
**Security practices for <input> tag:**

1. Restrict file types using accept.
2. Limit text length with maxlength.
3. Always validate input server-side.

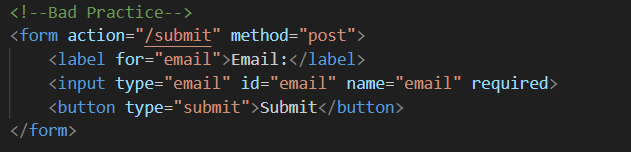
Bad Practice example: (1)



Good Practice example: (1)

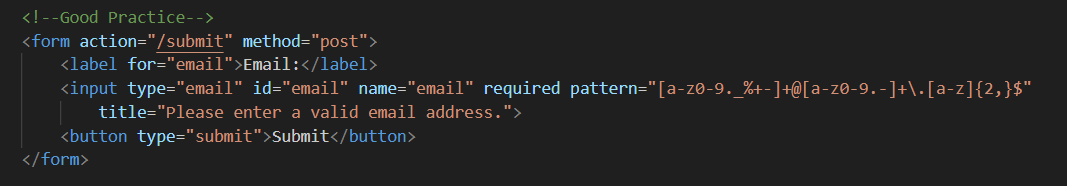


Bad Practice example: (2)



The browser checks if it looks like an email, but an attacker can disable JS, use dev tools, or send a direct POST request with invalid data like not-an-email or malicious scripts. If the server doesn’t check it, it can store invalid or dangerous data.

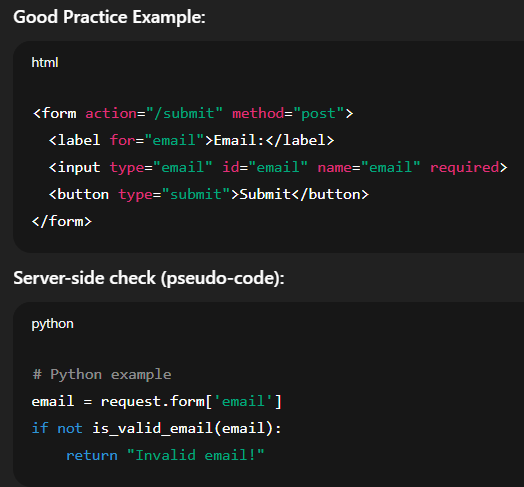
Good Practice example: (2) (simple HTML fix)



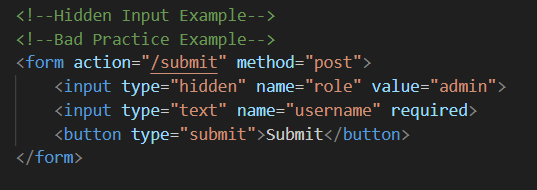
**Why it’s good:**

1. **type="email"** → basic HTML5 email validation.
2. **required** → ensures the user cannot submit the form empty.
3. **pattern="[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,}$"** → enforces a stricter email format.
4. **title="..."** → gives a helpful message if the pattern doesn’t match.
5. **label for="email"** → improves usability and accessibility.

Good Practice example: (2) (Server-side fix)



Bad Practice example: (3)



It is not secure because it stores sensitive information, like the user’s role, in a hidden field. Hidden fields can be easily modified by anyone using browser developer tools or by sending a custom request, which means a normal user could change their role to “admin” and gain unauthorized access. Additionally, if the server blindly trusts this hidden value without validation, it creates a serious security risk. In short, sensitive data should never be stored or relied on in client-side hidden fields; all critical logic must be handled securely on the server.

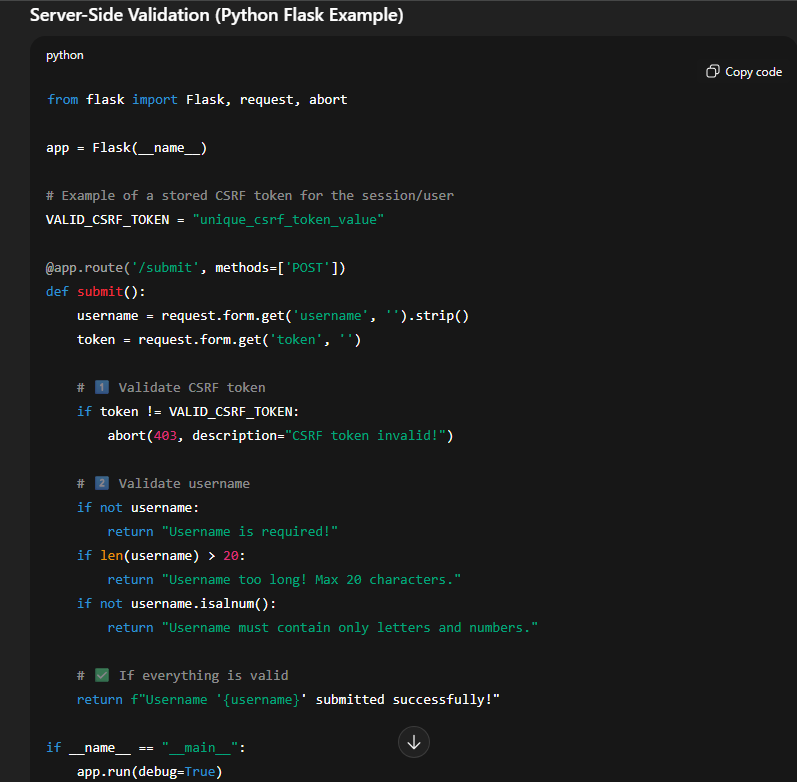
Good Practice example: (3)



The hidden field contains a CSRF token, which is a unique, randomly generated value for each user session. When the form is submitted, the server checks this token to ensure the request is coming from a legitimate user and not from a malicious site trying to perform actions on the user’s behalf. This prevents Cross-Site Request Forgery (CSRF) attacks. The form does not include sensitive data like roles or passwords in hidden fields, keeping critical logic on the server side.

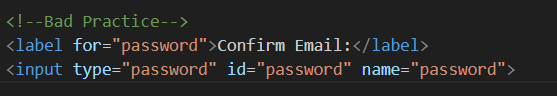
But, If the token is static, predictable, or poorly implemented, attackers could guess it and bypass CSRF protection. Also, if developers start adding sensitive information (like roles, permissions, or passwords) in hidden fields alongside the token, it would create a security risk similar to the first bad example.

Good Practice example: (3) (Server-side check)

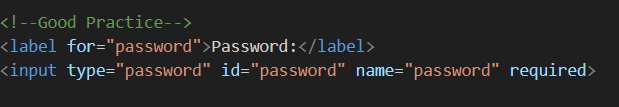


**Security practices for <label> tag:**

1. Label clearly describes the input.
2. Always link for to input id.

Bad practice:

Good Practice:

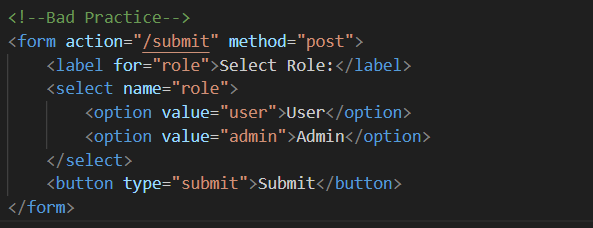


**Security practices for <select> tag:**

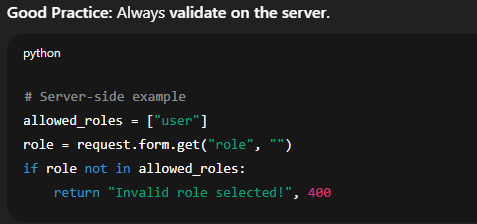
1. Assume dropdown value is safe
2. No default placeholder
3. Rely on required

Bad Practice example 1:

Relying on the <select> values on the client side without server validation. Attackers can manipulate the value before submitting. A user can manually change role=admin and gain unauthorized privileges if the server trusts the value blindly.

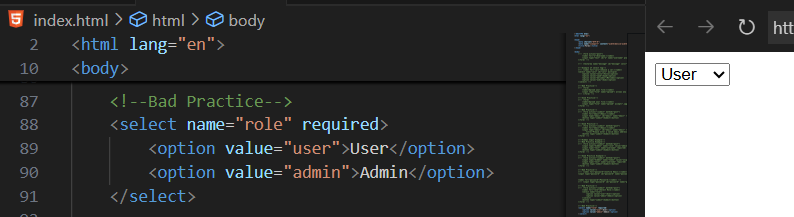


Good Practice example 1:

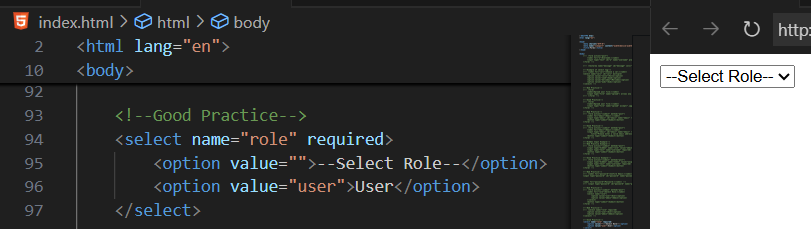


Bad Practice example 2:

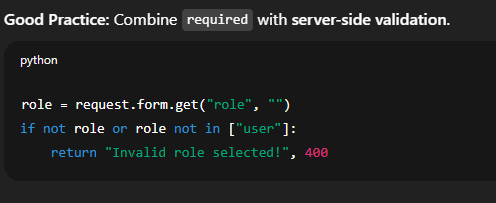
Not having a placeholder can cause the first option to be submitted unintentionally. If the user doesn’t actively select, user is automatically chosen, which may not reflect their intent.



Good Practice example 2:



Good Practice example 3:



--The End--