**RECOMMEND STEPS TO BE FOLLOWED TO CREATE A SECURED SYSTEM**

Following the steps bellow will mitigate many vulnerabilities, which includes, **XSS**, **SQL injection**, **privacy violation**, **data leakage**, **Session Fixation**, **User Enumeration**, **Bruteforce attack**, **MiTM**etc.

1. Login page with username/mail\_id (as input text field) and password (with input type as password) and a submit button which will send the data to the server via POST Http method (not GET Http Method). *This is to disallow anyone near you to see the credentials in plain text URL strings or leaking by browser history.*
2. While creating user in database, store the password as SHA256 hash of password and salt***[SHA256(password+salt)]***. ***Salt*** can be any random number specifically generated for each user, or if your application does not want is not that secure, you can use same salt for each user as well, but store that salt securely. *This should be used to secure user’s passwords in case of any data compromise.*
3. The password received in step 1 at the server should be hashed with the same method & salt (which you used while storing) and that hash should be sent back to the database for matching.
4. The server should have a method to validate input (in this case username and password) coming from the client side and should generate an error when any unexpected character would get identified in the input (you can create a whitelist of allowed characters for this). *This would prevent most of the input validation related attacks.* If you specifically want any special character to be stored in database, you should not miss the next step.
5. Always use parameterized queries with data binding, for all database operations from any language. You can use PDO in php (Parameterized queries will generally send the user input as data to database, when you bind variables). This is a very important step from security point of view and would ***prevent SQL Injection*** attacks.
6. Always send generic error messages to the user. For e.g., if authentication fails, send only “Wrong username/password”; Do not specifically send which part is wrong, username or password. *This will prevent****User Enumeration****, which is again an issue from security point of view.* Also, in condition of any other error, redirect user to some custom error page showing generic error page.
7. ***Recreate a new session id*** and authenticate it as soon as credentials got verified and than set it in user’s browser. By authenticating a session i meant, storing some user specific information like email id or username in session variable(or object) to track in later requests. Authenticating a new session and setting it in cookie would prevent **Session Fixation***related attacks.*
8. *To prevent****Bruteforce****attack*, you can use ***CAPTCHA***at the login, which should appear after some number of failed login attempts (usually depends upon data classification of application), lets say 3 or 5. And those failed login attempts should be handled at the server side and not client side. Also, after some attempts you can even lock the user account from server and create another functionality to approach admin for unlocking it, that should only be done when you are dealing with highly restricted kind of information.
9. Always host your application over ***https***instead of http. This would help against ***MiTM***attacks.
10. Set ***HTTPOnly***and ***secure***attributes with session id cookie. ***HTTPOnly***is to prevent your session id being hijacked when ***XSS***is present. ***Secure flag***should only be set with ***https***if you do not want to break your application connectivity. Secure flag will ensure that the cookie (where it is set) would only transfer over secure https channel.
11. You can use ***HSTS***header as well, if you want one more level of security. This would secure you against ***MiTM***attacks.
12. Always set password policy for users when you are giving them rights to choose their own passwords. Creating a password policy would ensure prevention against ***dictionary attack***. This policy should be managed over the server. Passwords should follow below criteria.
    1. Minimum 8 characters
    2. Atleast 1 character in Upper Case
    3. Atleast 1 special character from (!@#$&%. etc)
    4. Atleast 1 number
    5. Does not contain word which has been used in the username.

Some points from above can be avoided if the application is not dealing with any financial data or user’s personal information. But some points like SQL injection and Brute force one should always be implemented.