Checklist

|  |  |
| --- | --- |
| Set Strong Password   * Min 12 chars * Lower-case * Upper-case, * Numbers, * Special characters | yes |
| Securing user data and passwords | Yes |
| * Provide a Secured Session (Fixed Sessions issues) * Session timeout * Route to homepage/login page after session timeout. | Yes  Yes  Yes |
| * Able to login to system after registration. * Account lockout after 3 login failures. * Clean logout * Perform audit log | Yes  Yes  Yes  Yes |
| Implement Google reCaptcha v3 service | Yes |
| * Prevent SQLi and XSS and perform proper input filtering, validation and verification. (e.g email) * Client and server input validation | Yes  Yes |
| Graceful error handling on all pages (including 404, 403 error pages etc) | Yes |
| Use external tools to perform software testing:   * Github (check week 14 eLab) * Implement the recommendation to clear the security vulnerability for your source code. * Save your source code into Github and provide the public link. | Yes |
| Account Policies  • Automatic account recovery after x mins of lockout.  • Avoid password reuse (max 2 password history)  • Change password  • Minimum and Maximum password age  • 2FA | Yes  Yes  Yes  Yes  Yes |

**Set strong password**

* Use of .Length to determine if length of password is more than 12 characters.
* Use of regex to determine lower-case, upper-case, numbers and special characters.
* Both server and client side implemented the same technique.

**Securing user data and passwords**

* **Hash with Salt**: Password & CVC number. Used for comparison.
  + Saves both Password hash and salt into DB, for future comparisons.
  + Assumption: As an extra layer of security, each time a user wants to pay using their stored credit card, they would be required to type the CVC numbers in again. Hence CVC numbers will be hashed instead of using encryption algorithms.
* **Rijndael (AES) encryption**: Credit card number and Expiry date.
  + Store the IV and key

**Session**

* Session Fixature prevention
  + Generate a new set of guid each time user logs in.
  + Invalidate the old sessions and cookies
* Session timeout set in web.config file
* Added client side javascript to refresh the client browser after the session timeout time is out.

**Login**

* Attempt to retrieve the user password hash and salt with the email user keyed in. If email given does not match any record, return a generic message.
* Else, hash the user keyed in password with salt and compare if both hash from DB and the current one matches.
* If it matches, proceed to bring user to OTP page. Else return generic message.

Login – Account lockout after 3 login failures;

* Keep a variable “LoginFailAttempts” inside table account, each time a password is incorrectly given, the count will increase by 1
* Once the count reaches 3, the user will not be able to login.

Login – Clean logout

* Clear all cookies and session.

Audit Log

* A table that contains fields of Id, UserId, Action, Timestamp.
* Each time a user logins or logout, it will show the respective action “login-success”, “login-fail” & “logout-success”.

**Anti-bot**

* Implemented Google reCAPTCHA v3.

**Validation – Whitelisted values only.**

Email – Check if it’s a valid email using regex and check against the DB to see if any existing email exists.

Password – Checks against a list of criteria to advise user on what other characters to add in to fit into the password requirement.

First/Last name – Check if name is only alphabets and single space.

Credit Card Numbers

* Check if card number matches exactly 16 numbers, and format of what a typical card number would be.

CVV

* Check if CVV matches exactly 3 numbers.

Credit Card Expiry

* reject all inputs that month does not fit January to December
* check against the current year, if it is less than the current year then card will not be accepted
* or card that is expiring this month or had already expired will be rejected.

Image Validation

* Checks if image field is filled, if it’s an accepted image file type (.jpg) and if image size is below 0.2mb.

Email Verification

* Use of MailKit to transmit email.
* A token will be generated each time a user account is created using RNGCryptoServiceProvider. This token will be saved to Authentication table, together with userid, timestamp and expiry time.
* Append the token to a confirm page link and mail it to the email indicated.
* Upon user clicking on the link in their email, the system will check if the token exists and check against the expiry time.
* If it’s valid, it will update the user account table on “IsEmailVerified” to true and set the expiry time of the Authentication table to the datetime now. This will invalidate any other requests made using the token as the expiry time has past.
* If it’s not valid, a generic message will appear to alert the user.
* When the user attempts to login, with the correct credentials entered, user will need “IsEmailVerified” to be true in order to proceed.

**SQL Injection Prevention**: Prevented by using Parameterized Queries

**Proper Error Handling**

* Custom errors made in all try and catch loops.
* Custom errors configured in web.config.
* Test case: <https://localhost:44377/Registration1.aspx>

Github Link:

Advance feature – **Automatic account recovery**

* Every time after user exceeds 3 tries, a date time of +1 minute (account lockout time) will be recorded and stored in AccRecovery field of Account table.
* Upon account lockout time is up, and user tries to login, a function will clear the time recorded and also resets the lockout count to 0.

Advance feature – **Change password**

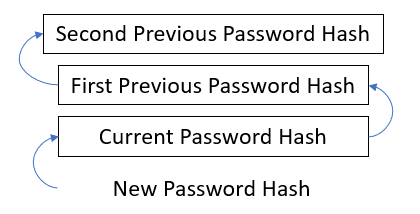
* Security stamp is implemented to invalidate sessions every time a password is changed.
* Once system detected security stamp stored in session differs from the one in the DB, the system will log the user out of the account. This prevents user logged in from multiple sessions/devices to continue using the account after the password is changed.
* Update password hash upon validation.

Advance feature – **Min/max password age**

* Min Age: 1 min
* Max Age: 5 mins
* Added in MinAge & MaxAge fields into Account Table.
* These values will be issued out when the user has successfully registered or changed password.
* When user logs in, and max age is reached, the user will be redirected to change password.
* Validation will be done on change password page on submit, to see if the current datetime is less than the minage stored in db.

Advance feature – **Avoid password reuse**

* Have the previous 2 password stored as a separate field in the Account Table.
* When user changes password, hash the new password that is being updated, compare against the current password hash, the first previous password hash and the second previous password hash to see if there any values match. If any values match, deny the change.
* If none of the values matches, proceed to make a change and store the first password hash (if any) to the second password hash value and current password hash into the first password hash history.



Advance feature – **2FA via Email**

* Every time a user logs in with a valid ID and password, there will be an email containing a 6 digit onetime password sent to their email address, only providing them with a session of their email address (which will be used for further verification later). This OTP has a 5 minute expiry.
* They will be redirected to the OTP page requesting them to submit the OTP.
* Once user has keyed in the OTP and press submit, system will check against the db to check if it’s a correct OTP, and if the OTP entered within before the expiry time.
* If all checks pass, user will be issued GUID, and security stamp to proceed. (Without GUID and security stamp, user will not have enough rights to access the core system).