Developing Secure Software Coursework 2: Client Report

100242165, 100233844, 100263597, 100263297

1 Ethical Considerations

During the design and development of this blog there were some ethical considerations to take into account. A main point was the decision to avoid using or storing and sensitive user data - the only personal details we collect are a first name and email. This is both to respect privacy as well as avoiding outright several potential threats under attack. An example of this is that users are required to declare that they are over the age of 13 upon registration. This is to ensure that content on the site is safe to be consumed by those who are allowed access without having the user declare their specific age or birth date.

2 Security Vulnerabilities and their Mitigation

As with all web development, sites are often at risk of being maliciously or unintentionally exploited. This may be in the form of attempting to leak sensitive data stored in our databases or bringing the site down. In this section, we cover different security vulnerabilities and how we attempt to mitigate them.

2.1 Account enumeration

To prevent attackers from gaining information from the login process, we refrain from providing detailed information for failed logins. This comes in the form of a generic incorrect message for failed login attempts. Additionally, the response times of the site for logging in is randomised, by adding up to one second of extra delay. This prevents attackers from being able to identify a difference in the server response time if they successfully guess a credential present in our database. The short extra time minimises usability impact.

```
let sleep = ms => new Promise(resolve => setTimeout(resolve,
                        Math.random() * 1000));
```

2.2 Session Hijacking

For this blog, we utilised the Express-session library (4). The library generates a "session cookie" (a unique identifier) for each user. Our mitigation also includes having the cookie expire after a day, where the user will be required to reauthenticate. This is done so unattended computers or logged in accounts will have to authenticate. The length of time is a compromise between usability and security - if the system was more critical like a banking website, this time span would be shorter. The cookie is also protected from being able to be accessed by client-side JavaScript as we see no need to allow front-end users to see the cookie. The default name of the session cookie is also renamed to obfuscate our authentication process from would-be attackers. We also set the cookie to be only sent via same-site requests. As our blog all runs on the same site, we found that we can enable this feature to prevent external requests to our client's cookie. Finally, one final step we take is to regenerate the session cookie upon successful user login. This is to ensure that an attacker cannot use the same cookie from an unauthenticated user to hijack an authenticated session.

2.3 SQL Injection

To solutions were used for SQL Injection. Firstly a basic safeguard prevents the user from creating and entering non-alphanumeric characters in most text fields. This prevents users from inputting special characters which are associated with SQL code. We also implemented a more modern approach - almost all databases allow for query parameterization. As an example, instead of passing a username and password directly into the SQL statement, we utilise placeholders and pass the username and password as parameters. This means any user input is always treated as text and not database instructions (10).

2.4 Cross Site Scripting (XSS)

Though our simple alphanumeric filter implemented in SQL injection would work in prevent Cross Site Scripting, it would mean that all posts on the site would be without many special characters. Creating our own more advanced filter would be near impossible due to all the edge cases (OWASP). As such we utilise the cross site scripting (XXS) library XSS (5), to sanitize any HTML and JavaScript found in posts stored to our database. The library replaces these illegal characters thus preventing the HTML script from running when displayed on our site.

```
let title = xss(results[0].title)
```

2.5 Cross Site Request Forgery (CSRF)

We use the cross site request forgery (CSRF) library CSURF (2) to create a CSRF token to authenticate our user. CSURF creates a token which is added to requests. When a user makes a request, the token is validated against the user's CSRF cookie or session.

2.6 (Distributed) Denial of Service Attacks

To prevent DDoS attacks, we utilise the toobusy-js library to prevent the server from going down when overwhelmed with traffic (11). The library works by monitoring average response times, and will begin to stop further requests if response times drastically increase. This means that the server remains responsive under the load while serving the requests it can manage. Another tool, Express-Rate-Limit (9), limits the number of requests one location can make within a time span of ten minutes - a reasonable figure. We also utilise a manually implemented Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) in the registration process therefore preventing unregistered users from brute-forcing requests through our registration. This is done by generating a random string of numbers, then sending images of visually distorted numbers for the users to read and input.

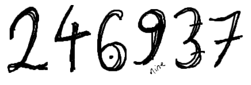
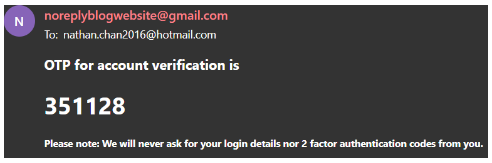


Figure 1: An example of a generated CAPTCHA image.

2.7 Phishing

To prevent phishing, warnings are placed over the site that the admins/moderators of our blog will never ask users for their personal information including their log in details. We also implemented and perform 2 Factor Authentication in the form of email login verification which prevents an inactive user from losing their account to a hacker - if an account has not been regularly logged in for more than a period of time (one month in this case), logging in requires inputting an additional code sent to the user's email.


N
noreplyblogwebsite@gmail.com
To: nathan.chan2016@hotmail.com

OTP for account verification is

351128

Please note: We will never ask for your login details nor 2 factor authentication codes from you.

Figure 2: An example of a generated 2FA code email.

2.8 Other Authentication Methods

To maintain usability, we decided to stick to a standard username/password system. Other methods have multiple weaknesses, such as being unfamiliar to users, and having their technical security hampered by aspects such as the human factor (13) (12) (3). Other methods of authentication have also been avoided like biometrics - users may feel they are invasive, and the majority of computers do not feature biometric hardware.

To secure passwords securely they are obfuscated into "hash", in a way where no parties (including our own) can realistically decipher them. The inputted password is combined, or "salted", with a salt, semi random generated text that is stored on the database and is unique to each user, and not based off of any public information (1). Once this is done, the combined output is then hashed - a unique and irreversible output. This output can then be stored on the database. The method to produce this hash output should have a few attributes - its output should be completely unique, of the same length, and feature the "avalanche effect" - even small changes to the input greatly change the output. (8) These features maximise obfuscation. The "salting" process also eliminated the same password producing the same output. Only the salt and hash are the only stored fields on our database - no passwords can ever be seen in original form. To authenticate a user, their input is put through this same process and compared with the stored field. This was implemented twice, one using the BCrypt library (6) and its pre-made functions, and one done manually for proof of concept.

As further authentication methods, we also use a CAPTCHA to authenticate users when they complete the registration process as shown previously in Figure 1. While we debated adding CAPTCHA to our login process, we decided it would be too large an impact to the usability of the site. In the future it could be implemented if multiple failed login attempts were made. Furthermore, as we cover in the Ethical Considerations section 1, as we do not store sensitive user information on the site, there is less to lose should a breach occur. Finally, we also require the user to input a one time password (OTP) if their last login was more than 1 month ago. This comes in the form of sending an email through our own Simple Mail Transfer Protocol server. To increase security in our emails, we use Google's authentication and authorisation service to ensure that the data between the web application and emails are kept private.

3 Testing

3.1 User Testing: Site Wireframing/Prototyping

To ensure that our blog meets usability standards, we prototyped with Lo-Fi and Mid-Fi diagrams. After development of the Lo-Fi, we sought the feedback of 3 users who were unrelated to the development of this application. The feedback was noted and considered for the development of the Mid-Fi. Then, the feedback process for repeated for the Mid-Fi and feedback used in the development of the final front-end.

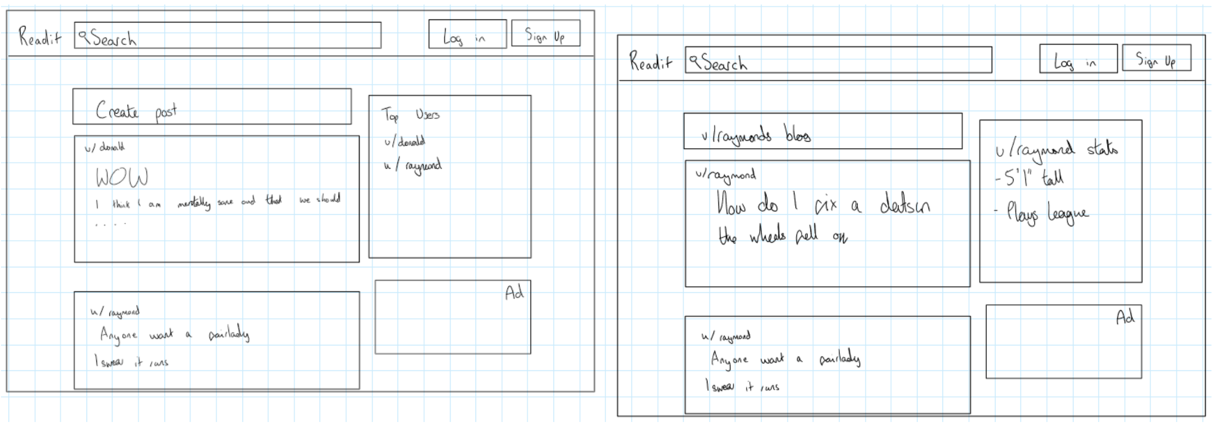


Figure 3: Lo-Fi diagrams of the site.

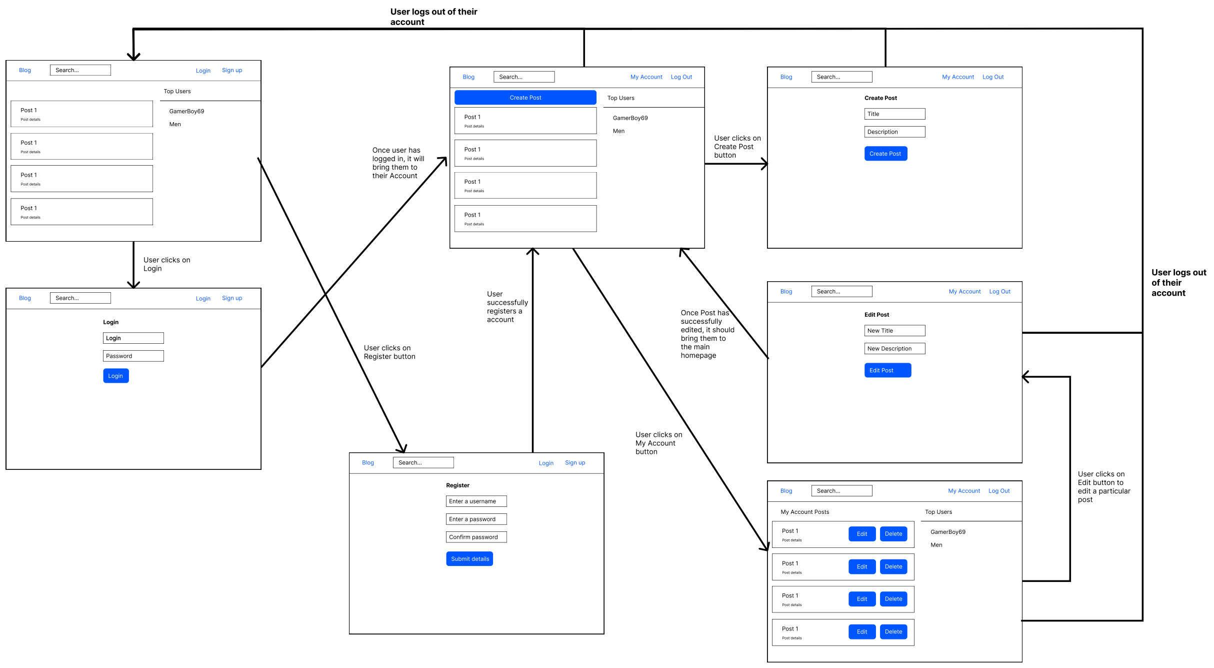

User logs out of their account
Blog
Search...
Login
Sign up
Top Users
Post 1
Post details
GamerBoy69
Men
User clicks on Login
Login
Password
Once user has logged in, it will bring them to their Account
My Account
Log Out
Create Post
User clicks on Create Post button
Title
Description
User clicks on Register button
User successfully registers a account
Once Post has successfully edited, it should bring them to the main homepage
Edit Post
New Title
New Description
User logs out of their account
User clicks on My Account button
Register
Enter a username
Enter a password
Confirm password
Submit details
My Account Posts
Edit
Delete
User clicks on Edit button to edit a particular post

Figure 4: Mid-Fi diagrams of the site.

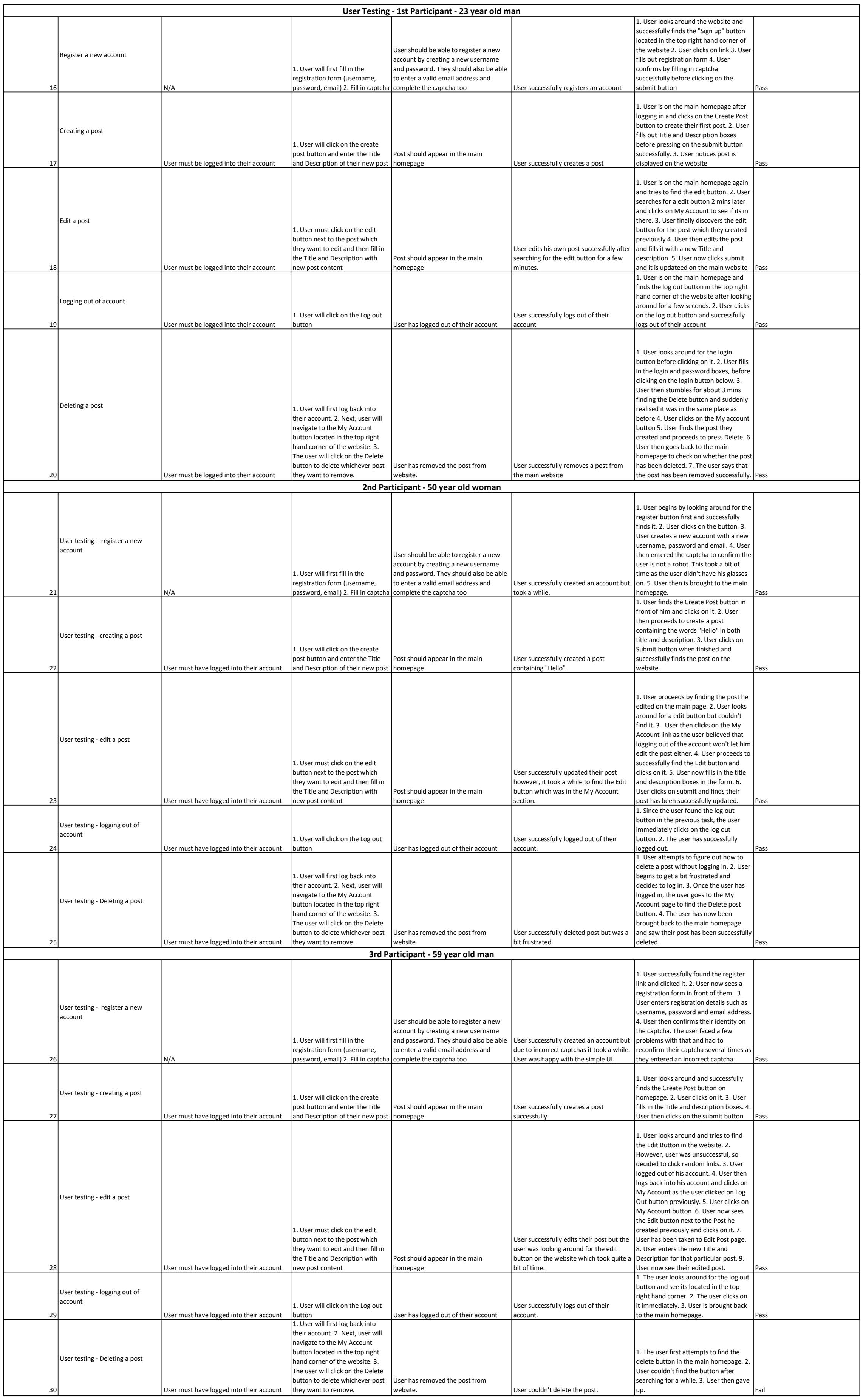
3.2 User Testing: Think Aloud Tasks

After the completion of the base application, we developed tasks which aimed to cover the main functionality of the site for test subjects to perform while thinking their thought process aloud for feedback. These were namely: register an account, login, create a post, edit a post, and logout. The documentation can be found in the form of a test plan in Figure 5.

3.3 System Unit Testing

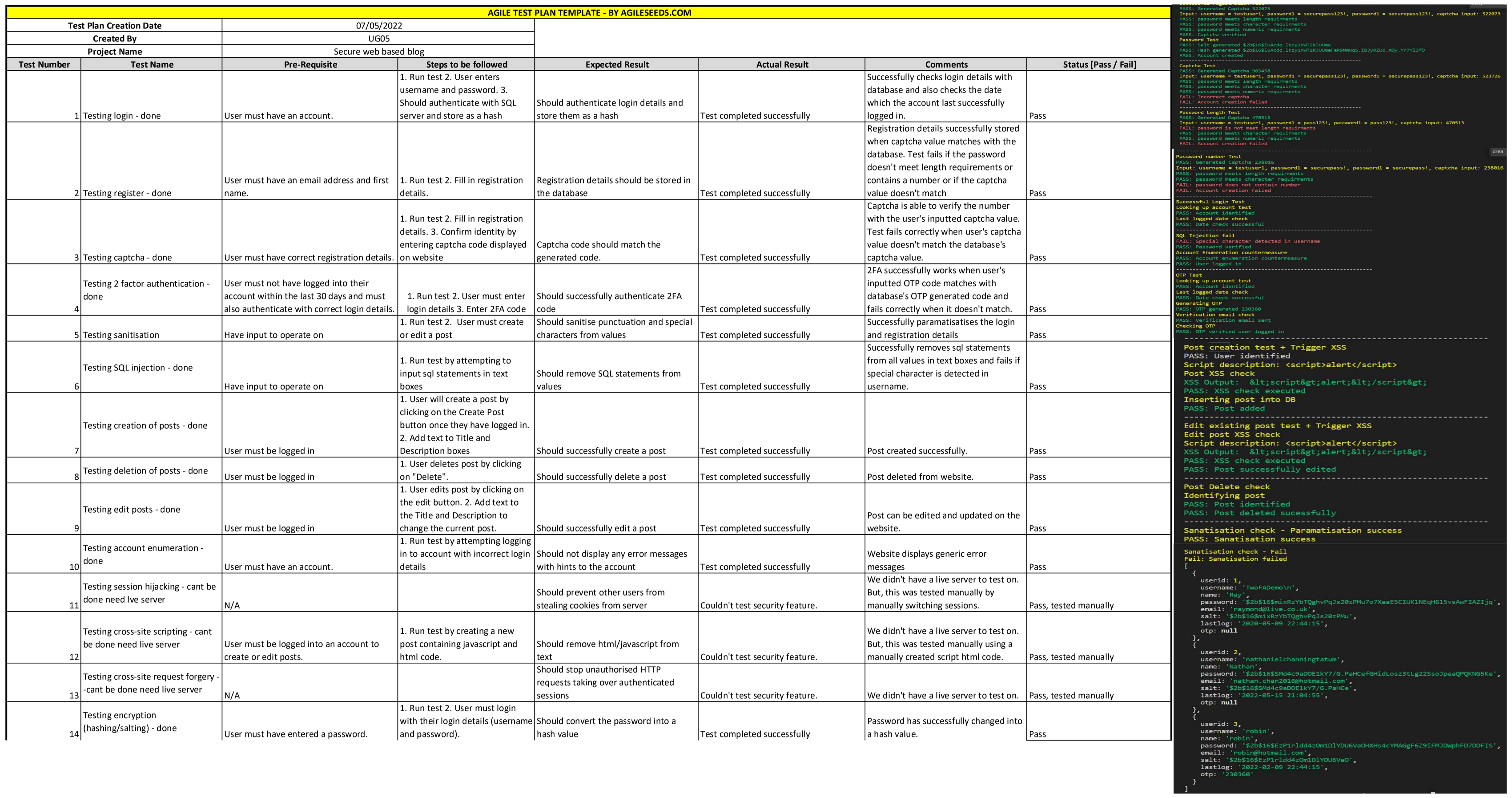
We also performed system unit testing - tests were written and performed for all functions the site uses. The expected results and actual results of each test were recorded and documented in the form of a test plan 6.

4 Appendix



User Testing - 1st Participant - 23 year old man							
16	Register a new account	N/A	1. User will first fill in the registration form (username, password, email) 2. Fill in captcha	User should be able to register a new account by creating a new username and password. They should also be able to enter a valid email address and complete the captcha too	User successfully registers an account	1. User looks around the website and successfully finds the "Sign up" button located in the top right hand corner of the website 2. User clicks on link 3. User fills out registration form 4. User confirms by filling in captcha successfully before clicking on the submit button	Pass
17	Creating a post	User must be logged into their account	1. User will click on the create post button and enter the Title and Description of their new post	Post should appear in the main homepage	User successfully creates a post	1. User is on the main homepage after logging in and clicks on the Create Post button to create their first post. 2. User fills out Title and Description boxes before pressing on the submit button successfully. 3. User notices post is displayed on the website	Pass
18	Edit a post	User must be logged into their account	1. User must click on the edit button next to the post which they want to edit and then fill in the Title and Description with new post content	Post should appear in the main homepage	User edits his own post successfully after searching for the edit button for a few minutes.	1. User is on the main homepage again and tries to find the edit button. 2. User searches for a edit button 2 mins later and clicks on My Account to see if its in there. 3. User finally discovers the edit button for the post which they created previously 4. User then edits the post and fills it with a new Title and description. 5. User now clicks submit and it is updateed on the main website	Pass
19	Logging out of account	User must be logged into their account	1. User will click on the Log out button	User has logged out of their account	User successfully logs out of their account	1. User is on the main homepage and finds the log out button in the top right hand corner of the website after looking around for a few seconds. 2. User clicks on the log out button and successfully logs out of their account	Pass
20	Deleting a post	User must be logged into their account	1. User will first log back into their account. 2. Next, user will navigate to the My Account button located in the top right hand corner of the website. 3. The user will click on the Delete button to delete whichever post they want to remove.	User has removed the post from website.	User successfully removes a post from the main website	1. User looks around for the login button before clicking on it. 2. User fills in the login and password boxes, before clicking on the login button below. 3. User then stumbles for about 3 mins finding the Delete button and suddenly realised it was in the same place as before 4. User clicks on the My account button 5. User finds the post they created and proceeds to press Delete. 6. User then goes back to the main homepage to check on whether the post has been deleted. 7. The user says that the post has been removed successfully.	Pass
2nd Participant - 50 year old woman							
21	User testing - register a new account	N/A	1. User will first fill in the registration form (username, password, email) 2. Fill in captcha	User should be able to register a new account by creating a new username and password. They should also be able to enter a valid email address and complete the captcha too	User successfully created an account but took a while.	1. User begins by looking around for the register button first and successfully finds it. 2. User clicks on the button. 3. User creates a new account with a new username, password and email. 4. User then entered the captcha to confirm the user is not a robot. This took a bit of time as the user didn't have his glasses on. 5. User then is brought to the main homepage.	Pass
22	User testing - creating a post	User must have logged into their account	1. User will click on the create post button and enter the Title and Description of their new post	Post should appear in the main homepage	User successfully created a post containing "Hello".	1. User finds the Create Post button in front of him and clicks on it. 2. User then proceeds to create a post containing the words "Hello" in both title and description. 3. User clicks on Submit button when finished and successfully finds the post on the website.	Pass
23	User testing - edit a post	User must have logged into their account	1. User must click on the edit button next to the post which they want to edit and then fill in the Title and Description with new post content	Post should appear in the main homepage	User successfully updated their post however, it took a while to find the Edit button which was in the My Account section.	1. User proceeds by finding the post he edited on the main page. 2. User looks around for a edit button but couldn't find it. 3. User then clicks on the My Account link as the user believed that logging out of the account won't let him edit the post either. 4. User proceeds to successfully find the Edit button and clicks on it. 5. User now fills in the title and description boxes in the form. 6. User clicks on submit and finds their post has been successfully updated.	Pass
24	User testing - logging out of account	User must have logged into their account	1. User will click on the Log out button	User has logged out of their account	User successfully logged out of their account.	1. Since the user found the log out button in the previous task, the user immediately clicks on the log out button. 2. The user has successfully logged out.	Pass
25	User testing - Deleting a post	User must have logged into their account	1. User will first log back into their account. 2. Next, user will navigate to the My Account button located in the top right hand corner of the website. 3. The user will click on the Delete button to delete whichever post they want to remove.	User has removed the post from website.	User successfully deleted post but was a bit frustrated.	1. User attempts to figure out how to delete a post without logging in. 2. User begins to get a bit frustrated and decides to log in. 3. Once the user has logged in, the user goes to the My Account page to find the Delete post button. 4. The user has now been brought back to the main homepage and saw their post has been successfully deleted.	Pass
3rd Participant - 59 year old man							
26	User testing - register a new account	N/A	1. User will first fill in the registration form (username, password, email) 2. Fill in captcha	User should be able to register a new account by creating a new username and password. They should also be able to enter a valid email address and complete the captcha too	User successfully created an account but due to incorrect captchas it took a while. User was happy with the simple UI.	1. User successfully found the register link and clicked it. 2. User now sees a registration form in front of them. 3. User enters registration details such as username, password and email address. 4. User then confirms their identity on the captcha. The user faced a few problems with that and had to reconfirm their captcha several times as they entered an incorrect captcha.	Pass
27	User testing - creating a post	User must have logged into their account	1. User will click on the create post button and enter the Title and Description of their new post	Post should appear in the main homepage	User successfully creates a post successfully.	1. User looks around and successfully finds the Create Post button on homepage. 2. User clicks on it. 3. User fills in the Title and description boxes. 4. User then clicks on the submit button	Pass
28	User testing - edit a post	User must have logged into their account	1. User must click on the edit button next to the post which they want to edit and then fill in the Title and Description with new post content	Post should appear in the main homepage	User successfully edits their post but the user was looking around for the edit button on the website which took quite a bit of time.	1. User looks around and tries to find the Edit Button in the website. 2. However, user was unsuccessful, so decided to click random links. 3. User logged out of his account. 4. User then logs back into his account and clicks on My Account as the user clicked on Log Out button previously. 5. User clicks on My Account button. 6. User now sees the Edit button next to the Post he created previously and clicks on it. 7. User has been taken to Edit Post page. 8. User enters the new Title and Description for that particular post. 9. User now see their edited post.	Pass
29	User testing - logging out of account	User must have logged into their account	1. User will click on the Log out button	User has logged out of their account	User successfully logs out of their account.	1. The user looks around for the log out button and see its located in the top right hand corner. 2. The user clicks on it immediately. 3. User is brought back to the main homepage.	Pass
30	User testing - Deleting a post	User must have logged into their account	1. User will first log back into their account. 2. Next, user will navigate to the My Account button located in the top right hand corner of the website. 3. The user will click on the Delete button to delete whichever post they want to remove.	User has removed the post from website.	User couldn't delete the post.	1. The user first attempts to find the delete button in the main homepage. 2. User couldn't find the button after searching for a while. 3. User then gave up.	Fail

Figure 5: User test chart



AGILE TEST PLAN TEMPLATE - BY AGILESEEDS.COM							
Test Plan Creation Date		07/05/2022					
Created By		UG05					
Project Name		Secure web based blog					
Test Number	Test Name	Pre-Requisite	Steps to be followed	Expected Result	Actual Result	Comments	Status [Pass / Fail]
1	Testing login - done	User must have an account.	1. Run test 2. User enters username and password. 3. Should authenticate with SQL server and store as a hash	Should authenticate login details and store them as a hash	Test completed successfully	Successfully checks login details with database and also checks the date which the account last successfully logged in.	Pass
2	Testing register - done	User must have an email address and first name.	1. Run test 2. Fill in registration details.	Registration details should be stored in the database	Test completed successfully	Registration details successfully stored when captcha value matches with the database. Test fails if the password doesn't meet length requirements or contains a number or if the captcha value doesn't match	Pass
3	Testing captcha - done	User must have correct registration details.	1. Run test 2. Fill in registration details. 3. Confirm identity by entering captcha code displayed on website	Captcha code should match the generated code.	Test completed successfully	Captcha is able to verify the number with the user's inputted captcha value. Test fails correctly when user's captcha value doesn't match the database's captcha value.	Pass
4	Testing 2 factor authentication - done	User must not have logged into their account within the last 30 days and must also authenticate with correct login details.	1. Run test 2. User must enter login details 3. Enter 2FA code	Should successfully authenticate 2FA code	Test completed successfully	2FA successfully works when user's inputted OTP code matches with database's OTP generated code and fails correctly when it doesn't match.	Pass
5	Testing sanitisation	Have input to operate on	1. Run test 2. User must create or edit a post	Should sanitise punctuation and special characters from values	Test completed successfully	Successfully paramatisatises the login and registration details	Pass
6	Testing SQL injection - done	Have input to operate on	1. Run test by attempting to input sql statements in text boxes	Should remove SQL statements from values	Test completed successfully	Successfully removes sql statements from all values in text boxes and fails if special character is detected in username.	Pass
7	Testing creation of posts - done	User must be logged in	1. User will create a post by clicking on the Create Post button once they have logged in. 2. Add text to Title and Description boxes	Should successfully create a post	Test completed successfully	Post created successfully.	Pass
8	Testing deletion of posts - done	User must be logged in	1. User deletes post by clicking on "Delete".	Should successfully delete a post	Test completed successfully	Post deleted from website.	Pass
9	Testing edit posts - done	User must be logged in	1. User edits post by clicking on the edit button. 2. Add text to the Title and Description to change the current post.	Should successfully edit a post	Test completed successfully	Post can be edited and updated on the website.	Pass
10	Testing account enumeration - done	User must have an account.	1. Run test by attempting logging in to account with incorrect login details	Should not display any error messages with hints to the account	Test completed successfully	Website displays generic error messages	Pass
11	Testing session hijacking - cant be done need lve server	N/A		Should prevent other users from stealing cookies from server	Couldn't test security feature.	We didn't have a live server to test on. But, this was tested manually by manually switching sessions.	Pass, tested manually
12	Testing cross-site scripting - cant be done need live server	User must be logged into an account to create or edit posts.	1. Run test by creating a new post containing javascript and html code.	Should remove html/javascript from text	Couldn't test security feature.	We didn't have a live server to test on. But, this was tested manually using a manually created script html code.	Pass, tested manually
13	Testing cross-site request forgery - cant be done need live server	N/A		Should stop unauthorised HTTP requests taking over authenticated sessions	Couldn't test security feature.	We didn't have a live server to test on.	Pass, tested manually
14	Testing encryption (hashing/salting) - done	User must have entered a password.	1. Run test 2. User must login with their login details (username and password).	Should convert the password into a hash value	Test completed successfully	Password has successfully changed into a hash value.	Pass

```
PASS: Generated Captcha 522073
Input: username = testuser1, password1 = securepass123!, password1 = securepass123!, captcha input: 522073
PASS: password meets length requirments
PASS: password meets character requirments
PASS: password meets numeric requirments
PASS: Captcha verified
Password Test
PASS: Salt generated $2b$16$EuAcdq.lkzy1nWT2RJUUme
PASS: Hash generated $2b$16$EuAcdq.lkzy1nWT2RJUUmeFaR9MmUqS.IbJyRZUc.XDy.Yr7Yl3fO
PASS: Account created
------------------------------------------------------------
Captcha Test
PASS: Generated Captcha 903458
Input: username = testuser1, password1 = securepass123!, password1 = securepass123!, captcha input: 523726
PASS: password meets length requirments
PASS: password meets character requirments
PASS: password meets numeric requirments
FAIL: Incorrect captcha
FAIL: Account creation failed
------------------------------------------------------------
Password Length Test
PASS: Generated Captcha 470513
Input: username = testuser1, password1 = pass123!, password1 = pass123!, captcha input: 470513
FAIL: password is not meet length requirments
PASS: password meets character requirments
PASS: password meets numeric requirments
FAIL: Account creation failed
------------------------------------------------------------
Password number Test
PASS: Generated Captcha 238016
Input: username = testuser1, password1 = securepass!, password1 = securepass!, captcha input: 238016
PASS: password meets length requirments
PASS: password meets character requirments
FAIL: password does not contain number
FAIL: Account creation failed
------------------------------------------------------------
Successful Login Test
Looking up account test
PASS: Account identified
Last logged date check
PASS: Date check successful
------------------------------------------------------------
SQL Injection fail
FAIL: Special character detected in username
PASS: Password verified
Account Enumeration countermeasure
PASS: Account enumeration countermeasure
PASS: User logged in
------------------------------------------------------------
OTP Test
Looking up account test
PASS: Account identified
Last logged date check
PASS: Date check successful
Generating OTP
PASS: OTP generated 230360
Verification email check
PASS: Verification email sent
Checking OTP
PASS: OTP verified user logged in
------------------------------------------------------------
Post creation test + Trigger XSS
PASS: User identified
Script description: <script>alert</script>
Post XSS check
XSS Output:  &lt;script&gt;alert;&lt;/script&gt;
PASS: XSS check executed
Inserting post into DB
PASS: Post added
------------------------------------------------------------
Edit existing post test + Trigger XSS
Edit post XSS check
Script description: <script>alert</script>
XSS Output:  &lt;script&gt;alert;&lt;/script&gt;
PASS: XSS check executed
PASS: Post successfully edited
------------------------------------------------------------
Post Delete check
Identifying post
PASS: Post identified
PASS: Post deleted sucessfully
------------------------------------------------------------
Sanatisation check - Paramatisation success
PASS: Sanatisation success

Sanatisation check - Fail
Fail: Sanatisation failed
[
  {
    userid: 1,
    username: 'TwoFADemo\n',
    name: 'Ray',
    password: '$2b$16$mixRzYbTQghvPqJs20zPMu7o7XaaE5CIUK1NEqH615vsAwFIAZIjq',
    email: 'raymond@live.co.uk',
    salt: '$2b$16$mixRzYbTQghvPqJs20zPMu',
    lastlog: '2020-05-09 22:44:15',
    otp: null
  },
  {
    userid: 2,
    username: 'nathanielchanningtatum',
    name: 'Nathan',
    password: '$2b$16$SMd4c9aDOE1kY7/G.PaHCefGHidLosz3tLg22SsoJpeaQPQKNG5Ke',
    email: 'nathan.chan2016@hotmail.com',
    salt: '$2b$16$SMd4c9aDOE1kY7/G.PaHCe',
    lastlog: '2022-05-15 21:04:55',
    otp: null
  },
  {
    userid: 3,
    username: 'robin',
    name: 'robin',
    password: '$2b$16$EzP1rldd4zOm1DlYDU6VaOHXHs4cYMAGgF6Z9iFMJDWphFD7DDFIS',
    email: 'robin@hotmail.com',
    salt: '$2b$16$EzP1rldd4zOm1DlYDU6VaO',
    lastlog: '2022-02-09 22:44:15',
    otp: '230360'
  }
]
```

Figure 6: Unit test chart

References

[1] Arias, D. (2021). Adding salt to hashing: A better way to store passwords. https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/.

[2] Csurf (2020). Csurf. https://www.npmjs.com/package/csurf.

[3] Devlin, M., Nurse, J. R. C., Hodges, D., Goldsmith, M., and Creese, S. (2015). Predicting graphical passwords. In Tryfonas, T. and Askoxylakis, I., editors, *Human Aspects of Information Security, Privacy, and Trust*, pages 23–35, Cham. Springer International Publishing.

[4] Express-session (2022). Express-session. https://www.npmjs.com/package/express-session.

[5] Lei, Z. (2022). Xss. https://www.npmjs.com/package/xss.

[6] Mazières, D. and Provos, N. (2021). Bcrypt. https://www.npmjs.com/package/bcrypt.

[OWASP] OWASP. Xss filter evasion - owasp cheat sheet series. https://cheatsheetseries.owasp.org/cheatsheets/XSS$_{F}ilter_{E}vasion_{C}heat_{S}heet.html$.

[8] Project, T. T. (2003). The hash function design problem. https://tracer.lcc.uma.es/problems/avalanche/avalanche.html.

[9] rate limit, E. (2022). Express-rate-limit. https://www.npmjs.com/package/express-rate-limit.

[10] SQL-Server-Team, M. (2019). How and why to use parameterized queries. https://techcommunity.microsoft.com/t5/sql-server-blog/how-and-why-to-use-parameterized-queries/ba-p/383483.

[11] Toobusy-JS (2016). Toobusy-js. https://www.npmjs.com/package/toobusy-js.

[12] Uellenbeck, S., Dürmuth, M., Wolf, C., and Holz, T. (2013). Quantifying the security of graphical passwords. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. ACM Press.

[13] Wang, Z. (2018). Pattern lock data set for ndss'17 paper entitled ”cracking android pattern lock in five attempts” version 2.