**A black and white sign

Description automatically generated**

**Secure Programming**

**Assignment 2**

|  |  |
| --- | --- |
| **Name** | **LIM ZENG KAI** |
| **Matric Number** | **A20EC0068** |
| **Lecturer’s Name** | **Dr. MOHD. ZAMRI BIN OSMAN** |

**Introduction on Different Technology used**

In Assignment 2, I have changed the technology stack used compared to Assignment 1. In Assignment 1, I used HTML, CSS, JavaScript, and PHP. However, for continued development of the web application with enhanced security, it is beneficial to use modern frameworks and technologies. Therefore, I have chosen React.js and Tailwind CSS for the frontend, Node.js and Express.js for the backend, and MySQL as the database.

Similar security functions implemented in Assignment 1 have been carried over to Assignment 2. These include password hashing, email and password verification, strong password implementation, and Google reCAPTCHA v2. Below is some code and screenshot which is different in Assignment 1.

* Password Hashing with the used of bcrypt.

The code below is to allow create a new account, and first will check the email or username is existing or not to avoid mistake. After that, the password will be hashed and store in the database

A screenshot of a computer program

Description automatically generated

Figure 1: code snippet of Creating User in server side

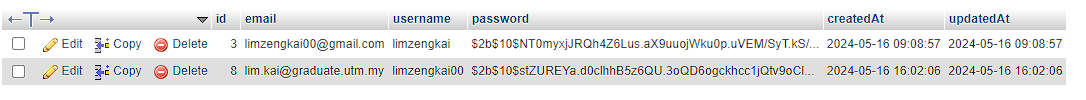


Figure 2: The output of the data stored in database

* Username, Email and password verification. The website enforces the user to use a strong passwords by utilizing a password field that requires user input to be at least 1 uppercase letter, 1 lowercase letter, 1 digit, 1 special character, and have a minimum length of 8 characters. This will be check on client side and server side to ensure the password must be meet to the requirement.

A screenshot of a computer program

Description automatically generated

Figure 3: code snippet of validation in client side

A screen shot of a computer code

Description automatically generated

Figure 4: code snippet of validation in server side

A screenshot of a login form

Description automatically generated

Figure 5: Website Output for create new account will wrong input

* Google reCAPTCHA v2

A screen shot of a computer program

Description automatically generated

Figure 6: Code snippet to get the results

A screenshot of a screenshot of a bicycle

Description automatically generated

Figure 7: Google reCAPTCHA v2 output

4. SQL Injection Prevention

In this project, I used Sequelize to handle SQL operations, which helps prevent SQL injection attacks. Sequelize includes several features designed to mitigate the risk of SQL injection:

* Parameterized Queries: Sequelize automatically uses parameterized queries, ensuring that user inputs are treated as data rather than executable code. This prevents attackers from injecting malicious SQL into the query.

A screenshot of a computer program

Description automatically generated

Figure 8: parameterized Query in the code snippet

* Escaping User Inputs: When Sequelize constructs SQL queries, it escapes user inputs, treating special characters as data rather than part of the SQL command. This escaping mechanism is crucial in preventing SQL injection.
* ORM Methods: By using Sequelize's ORM methods such as findOne, create, and update, raw SQL queries are abstracted away. These ORM methods internally handle the construction of safe SQL queries, adding an additional layer of security.

A computer screen shot of text

Description automatically generated

Figure 9: Code snippet of login by using sequelize

**Countermeasure against various session attack**

In this assignment, we are using JSON Web Token (JWT) authentication to enhance the security of our web application. JWTs are secure, compact, and self-contained tokens that are used to transmit information between parties. Here are the countermeasures we have implemented to protect against various session attacks:

1. Session Hijacking Attacks

Session hijacking involves an attacker stealing or intercepting a valid session token to gain unauthorized access. To mitigate this risk, we use the following techniques:

* HttpOnly Cookies: By setting the httpOnly flag on cookies in figure 10, we ensure that the cookies cannot be accessed via JavaScript, which protects them from cross-site scripting (XSS) attacks. We can see the result in Figure 12 and 13 to show that the benefit to use httponly cookie to avoid XSS attack.
* Secure Cookies: The secure flag ensures that cookies are only sent over HTTPS, protecting them from being intercepted by man-in-the-middle (MITM) attacks. However, we are develop in localhost so setting the node\_env to use when in production.
* Token Expiry: Setting an expiration time for JWTs limits the window of opportunity for an attacker to use a stolen token as shown in the Figure 11.

A screen shot of a computer code

Description automatically generated

Figure 10: Code Snippet of store JWT in the cookie

A screen shot of a computer screen

Description automatically generated

Figure 11: Code Snippet to generate JWT

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedFigure 12: cookie without httponly

Figure 13: cookie with httponly

1. Spoofing Authentication Cookies Attacks

Spoofing authentication cookies involves creating fake cookies to gain unauthorized access. To prevent this:

* JWT Signature Verification: Each JWT is signed with a secret key with can refer to Digure 11. On the server side, we always verify the JWT signature to ensure the token is valid and has not been tampered with. Figure 14 show the cookieJwtAuth code snippet which is used to verify the JWT used in the website. After the user login and generated JWT then for all the side will be protect by the proctectedRoute to ensure the JWT is valid and can used to access the pages. The process can refer to the Figure 15, 16, and 17
* HttpOnly and Secure Cookies: As mentioned earlier, using httpOnly and secure flags for cookies prevents them from being accessed via JavaScript and ensures they are only sent over HTTPS after deploy with https proctection

A screen shot of a computer program

Description automatically generated

Figure 14: cookieJwtAuth

A screen shot of a computer

Description automatically generated

Figure 15: server route for verify the token

A screen shot of a computer program

Description automatically generated

Figure 16: ProctectedRoute

A screen shot of a computer code

Description automatically generated

Figure 17: Route with is protected by ProctectRoute

1. Session Fixation Attacks

Session fixation attacks involve forcing a user's session ID to a known value, allowing an attacker to hijack the session. To prevent this:

* Regenerate Session ID: Upon successful login, we regenerate the session ID to ensure that the session ID is unique and unpredictable. The Figure 18 show the code snippet to regenerate new session when login is successfully.

A screen shot of a computer program

Description automatically generated

Figure 18: Session Regenerate