**COMP3911 Coursework 2**

**Analysis of Flaws**

1. Password Complexity – We should add complexity/security requirements to passwords to make them harder to guess or crack with a bruteforce attack.
2. Password Encryption – We should encrypt the passwords so that they are not stored in raw text format in the database to prevent anyone from seeing them.
3. Request Rate Limit – We should limit the amount of requests that a user can make from their device to stop them either crashing the web server or attempting a bruteforce attack.
4. Database Password Protection – Currently anyone with access to the database file can make modifications or view any of the data. We should add a password to prevent unauthorised access.
5. Device Authorisation – We could verify device IP addresses to ensure that we only allow requests to be made from authorised devices within the network

**Detailed Analysis:**

***Password Complexity***

Currently there is no security criteria for passwords to ensure that they are secure. If we open the SQL database with a database viewer, we can see three passwords as shown below. The first password ‘wysiwyg0’ would be considered relatively secure. However, the second could easily be guessed as it is composed of just the users first name. It is common practise to not allow a user’s name to be included in a password. Although we do not have access to the sign-up function, there is also clearly no requirement for numbers or special characters to be included in a password which makes them much easier to be cracked through a bruteforce attack.

Graphical user interface, application, table

Description automatically generated

***Password Encryption***

We can also see that the user passwords are stored in plaintext format in the database. This is very unsecure as anyone who had access to the database could gain access to a user’s password and then use this to access the system, or potentially other systems if the user uses the same login information. We should only store a hashed password and verify that the hashed version of the inputted user password matches this.

***Database Password Protection***

Currently anyone who has access to the database file can view any of the information in there. If a user gained access to the file they could read any information, and as this contains confidential information, it should have a password requirement so that only authorised users can read it.

**Fixes Implemented**

***Password complexity***

I have added a signup function which allows users to create an account via a POST request. This will take the name, email, and password of the user, as well as a secret key as a parameter which verifies that the user is authorised to create an account. This sign-up function first ensures that the user is authorised to make the request, and then verifies that there is not already an existing user with the same username. If these checks are passed, we then verify that the password is secure, by ensuring that it is at least 8 characters long, then checking that it contains at least one number and special character. Finally, we verify that it does not contain the users name as a substring, to make it harder to crack. If all these conditions are met, we return a response to the user and add the new account to the database.

***Password encryption***

I have modified the above sign-up function to now store a hashed version of the password in the database using SHA-256 hashing. This means that if someone gained access to the database then they would be unable to view the password. The authentication function now uses the same hashing technique to generate a SHA-256 hash of whatever password the user enters and verify that this matches the hash stored in the database. This will only match if it is the same password that the user originally signed up with.