Criteria:

**Properly store passwords on the server (15 points)**

To securely store passwords on the server, a salt and an SHA256 hash have been used. The code for this can be found under the signup route handler within webapp/routes.py, with screenshots provided below.

First, the username and password imputed by the user are collected. Some basic checks are applied to these, in order to force users to input a username and password combination that is at least somewhat secure. The username must be unique and greater than three characters long, ensuring that users can be differentiated from one another and there are no double up accounts with the same usernames. The password must be greater than five characters long, so that it is not too easy to guess, and it must match what has been inputted to the “confirm password” box, ensuring that the user has correctly inputted their desired password. (This can be seen in lines 44-57)

Second, a salt is generated. The salt is a random piece of data which is added to the password before it is hashed. Here, the salt is generated using the os.urandom() method of python’s OS module, which generates a random string suitable for cryptographic use of the specified size – in this case 16 bytes. Adding the 16 byte salt to the password helps protect against attacks using a rainbow table or other precomputed table, as it makes the required table for such an attack prohibitively large. It especially helps protect users who input generic passwords that might have identifiable hashes, as it makes the hash code that it actually stored in the database unique. (This can be seen in line 59)

Third, the computed salt and the inputted are hashed together using SHA256. A hash function takes an input and generates an output string that is (theoretically) unique to that input. It is a one way, meaning that the hash can be computed from the input, but the input cannot be discerned from the hash. By hashing the password and the salt, the plaintext password is never stored on the server. This means even if a hacker were to gain access to the database, they would not be able to determine what a user’s passwords are. However, the passwords can still be authenticated, because when the user later enters their password when attempting to log in, the same hash can be generated based on their input and compared to the hash stored in the database. (This can be seen in line 61)

Finally, this hash, alongside the username and the unique salt for this user, are added to the database. Specifically, for this task the database python module has been used. (This can be seen in lines 61-63)

Code:

Graphical user interface, text, application, email

Description automatically generated

**When log in, first check server’s certificate (e.g., you can manually create one using a hardcoded CA public key in your code) (25 points)**

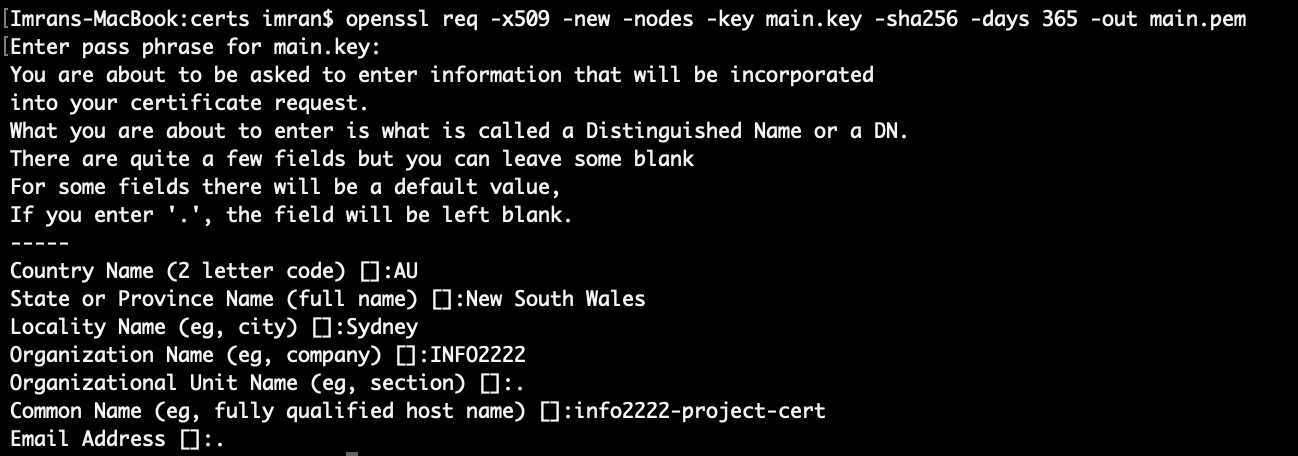
The following describes the process of generating a self-signed certificate and key, which are stored in the ‘certs’ folder of the project and are parsed in main.py.

Using openssl, a private key was generated:

Text

Description automatically generated

This private key was then used to generate a root certificate:

****

Two additional files were then created. Firstly, a config file for a certificate signing request (csr) file:



And secondly, a file to specify an X509 v3 certificate:



Finally, these two files were then used to create the domain certificate, using the command:

*openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out server.crt -days 500 -sha256 -extfile v3.ext*

**Securely transmitting a pwd to server (leveraging secure protocols or design the secure transmission properly) (10 points)**

To ensure that passwords are transmitted to the server securely, the whole application has been run over HTTPS. HTTPS stands for Hypertext Transfer Protocol Secure. It is an extension of HTTP, or Hypertext Transfer Protocol. HTTPS is more secure than HTTP, because it provides authentication and protects data. The key difference between HTTPS and HTTP is that HTTPS encrypts the communication protocol using SSL, or Secure Sockets Layer. SSL uses an asymmetric public key infrastructure to secure the data whilst it is in transit and protect user data from man-in-the-middle attacks, eaves dropping and tampering. In the code below, SSL is set up in line 13. “certs/server.crt” represents the location of the certificate file, and “certs/server.key” represents the location of the private key.

Code:

Graphical user interface, text, application

Description automatically generated

**Properly check whether password is correct (at least use the simple method that defends against offline pre-computation attacks) (10 points)**

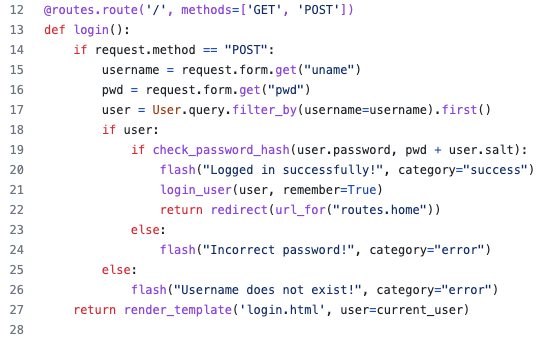
To check if the password is correct, the username and password inputted by the user are first collected. This occurs in lines 15 and 16 of webapp/routes.py. Next, the username entered is matched to one of the usernames stored in the database in line 17. If the username cannot be found, then the user is alerted to the error in line 26.

If the user can be found, the password entered by the user is then verified against the stored password associated with that user. This is performed by the function check\_password\_hash() in line 19. This function takes the password entered by the user, combines it with the stored salt of the user, and then hashes the resultant string. The hash code produced is then compared against the hash code stored in the database, which was previously hashed in the same way when the user first signed up. If the password is correct, then the hash produced by the password/salt combination should be identical to the hash stored in the database, since identical inputs produce identical hash codes.

In this method, it is the addition of a salt which protects against offline pre-computation attacks. Because it is a large, random and unique string, the salt adds significant complexity to the password before it is hashed. It therefore means that even if a hacker were to input a large quantity of generic passwords in an attempt to break into a user’s account, even a user who had used a generic password would be protected.

Code:

Webapp/routes.py



**Securely transmitting the message from A to B, even the server who can forward communication transcript cannot read the message, or modify the ciphertext (leveraging secure protocols or design the authenticated secure transmission properly) (40 points)**

The code relating to sending and receiving messages can be found in the file webapp/templates/home.html, and is written in javascript. Here, AES256 encryption is performed using the CryptoJS model to ensure that the communication is secure.

Lines 52 to 65 represent the code responsible for sending messages. The encryption is performed in line 60. Two parameters are passed to the encrypt() method, the plaintext of the message being sent, and a password. The password is a random integer provided as a context variable in line 35 when the page is loaded. The password enables the text to be correctly encrypted and decrypted at either end, without malicious actors being able to view the message. The result of the encrypt() function is a ciphertext. Since the plain text is converted into a ciphertext here, before the message is sent to the server, not even the server is able to read and understand the message. Finally, in line 65, the ciphertext is sent to the socket handler in main.py. In turn, the socket handler then sends the ciphertext back to the site in line 10 of main.py.

Lines 39 to 51 are responsible for receiving messages. The process here is essentially the reverse of sending messages. When the ciphertext is received, it is decrypted using the same CryptoJS Module and AES256. This occurs in line 45, using the decrypt() method. Again, two parameters are provided, but this time it is the ciphertext of the message being received and the password. After the decryption, the plaintext is made into a readable string in line 46 and shown to the user in line 49.

Code:

Main.py



Webapp/templates/home.html

A picture containing text

Description automatically generated