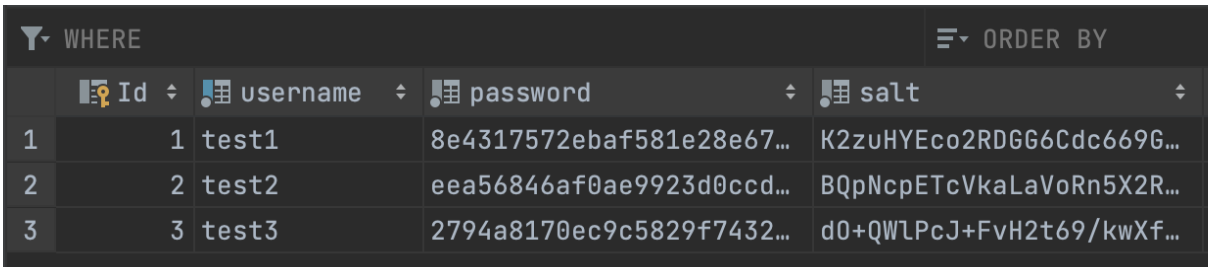
Register:

Users can use our web to create an account, the server will utilise SQL database to store user's account and password.

For storage part, the server will:

1. Generate a salt value
2. Combine the salt value with encoded password and hash it
3. Store username,hashed password,salt value and the public key(for the purpose of end-to-end encryption when sending message) generated in front end in table `Users`

Here is the Table screenshot:

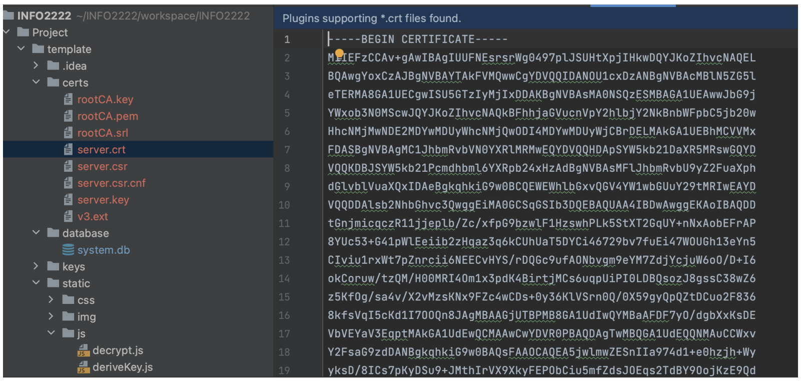


	Id	username	password	salt
1	1	test1	8e4317572ebaf581e28e67…	K2zuHYEco2RDGG6Cdc669G…
2	2	test2	eea56846af0ae9923d0ccd…	BQpNcpETcVkaLaVoRn5X2R…
3	3	test3	2794a8170ec9c5829f7432…	dO+QWlPcJ+FvH2t69/kwXf…

Server's Certificate:

We reference the method provided by this link:https://www.freecodecamp.org/news/how-to-get-https-working-on-your-local-development-environment-in-5-minutes-7af615770eec/ to set up our local certificate and store them in our server.

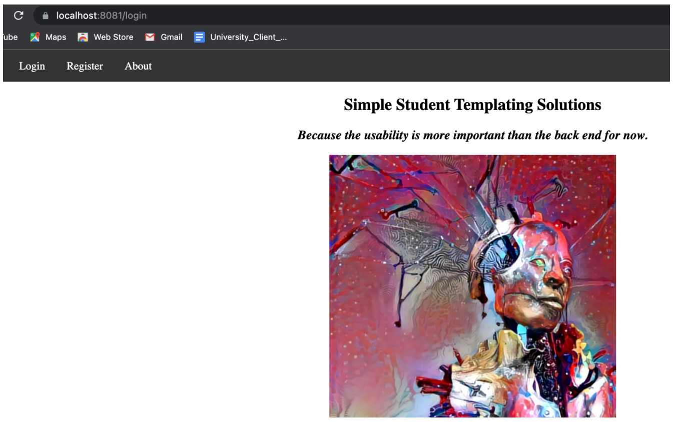
Here are the cert files:



```
-----BEGIN CERTIFICATE-----
MIIEFzCCAv+gAwIBAgIUUFNEsrsrWg0497plJSUHtXpjIHkwDQYJKoZIhvcNAQEL
BQAwgYoxCzAJBgNVBAYTAkFVMQwwCgYDVQQIDANOU1cxDzANBgNVBAcMBlN5ZG5l
eTERMA8GA1UECgwISU5GTzIyMjIxDDAKBgNVBAsMA0NSQzESMBAGA1UEAwwJbG9j
YWxob3N0MScwJQYJKoZIhvcNAQkBFhhjaGVucnVpY2hlbjY2NkBnbWFpbC5jb20w
HhcNMjMwNDE2MDYwMDUyWhcNMjQwODI4MDYwMDUyWjCBrDELMAkGA1UEBhMCVVMx
FDASBgNVBAgMC1JhbmRvbVN0YXRlMRMwEQYDVQQHDApSYW5kb21DaXR5MRswGQYD
VQQKDBJSYW5kb21Pcmdhbml6YXRpb24xHzAdBgNVBAsMFlJhbmRvbU9yZ2FuaXph
dGlvblVuaXQxIDAeBgkqhkiG9w0BCQEWEWhlbGxvQGV4YW1wbGUuY29tMRIwEAYD
VQQDDAlsb2NhbGhvc3QwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDD
tGnjmicqczR11jjeplb/Zc/xfpG9bzwlF1HzswhPLk5StXT2GqUY+nNxAobEFrAP
8YUc53+G41pWlEeiib2zHqaz3q6kCUhUaT5DYCi46729bv7fuEi47WOUGh13eYn5
CIviu1rxWt7pZnrcii6NEECvHYS/rDQGc9ufAONbvgm9eYM7ZdjYcjuW6oO/D+I6
okCoruw/tzQM/H00MRI4Om1x3pdK4BirtjMCs6uqpUiPI0LDBQsozJ8gssC38wZ6
z5Kf0g/sa4v/X2vMzsKNx9FZc4wCDs+0y36KlVSrn0Q/0X59gyQpQZtDCuo2F836
8kfsVqI5cKd1I70OQn8JAgMBAAGjUTBPMB8GA1UdIwQYMBaAFDF7yO/dgbXxKsDE
VbVEYaV3EqptMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgTwMBQGA1UdEQQNMAuCCWxv
Y2FsaG9zdDANBgkqhkiG9w0BAQsFAAOCAQEA5jwlmwZESnIIa974d1+e0hzjh+Wy
yksD/8ICs7pKyDSu9+JMthIrVX9XkyFEPObCiu5mfZdsJOEqs2TdBY9OojKzE9Qd
```

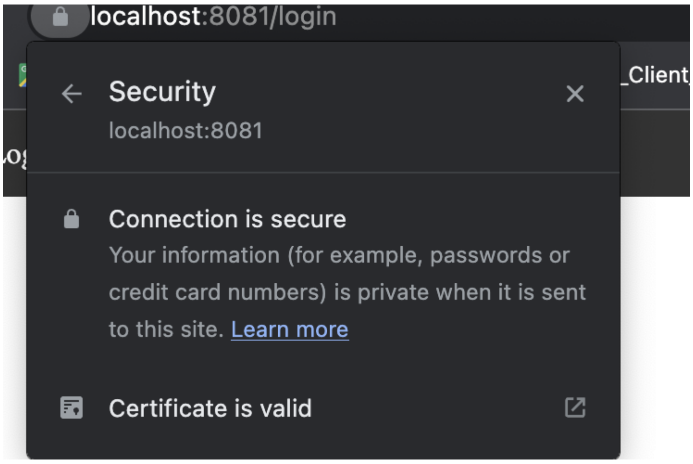
Based on the cert files we created, we utilise `gunicorn` server and apply HTTPS protocol to secure communication between user's web browser and our server.

Here are the screenshots of HTTPS and certificate:



Simple Student Templating Solutions

Because the usability is more important than the back end for now.



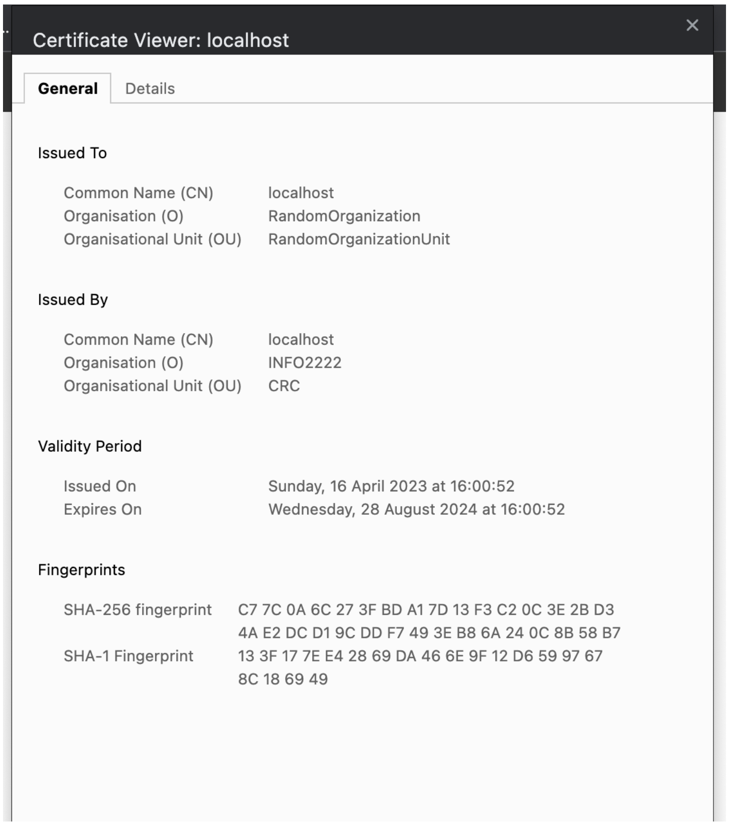
Security

localhost:8081

Connection is secure

Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more

Certificate is valid



Certificate Viewer: localhost

Issued To

Common Name (CN)	localhost
Organisation (O)	RandomOrganization
Organisational Unit (OU)	RandomOrganizationUnit

Issued By

Common Name (CN)	localhost
Organisation (O)	INFO2222
Organisational Unit (OU)	CRC

Validity Period

Issued On	Sunday, 16 April 2023 at 16:00:52
Expires On	Wednesday, 28 August 2024 at 16:00:52

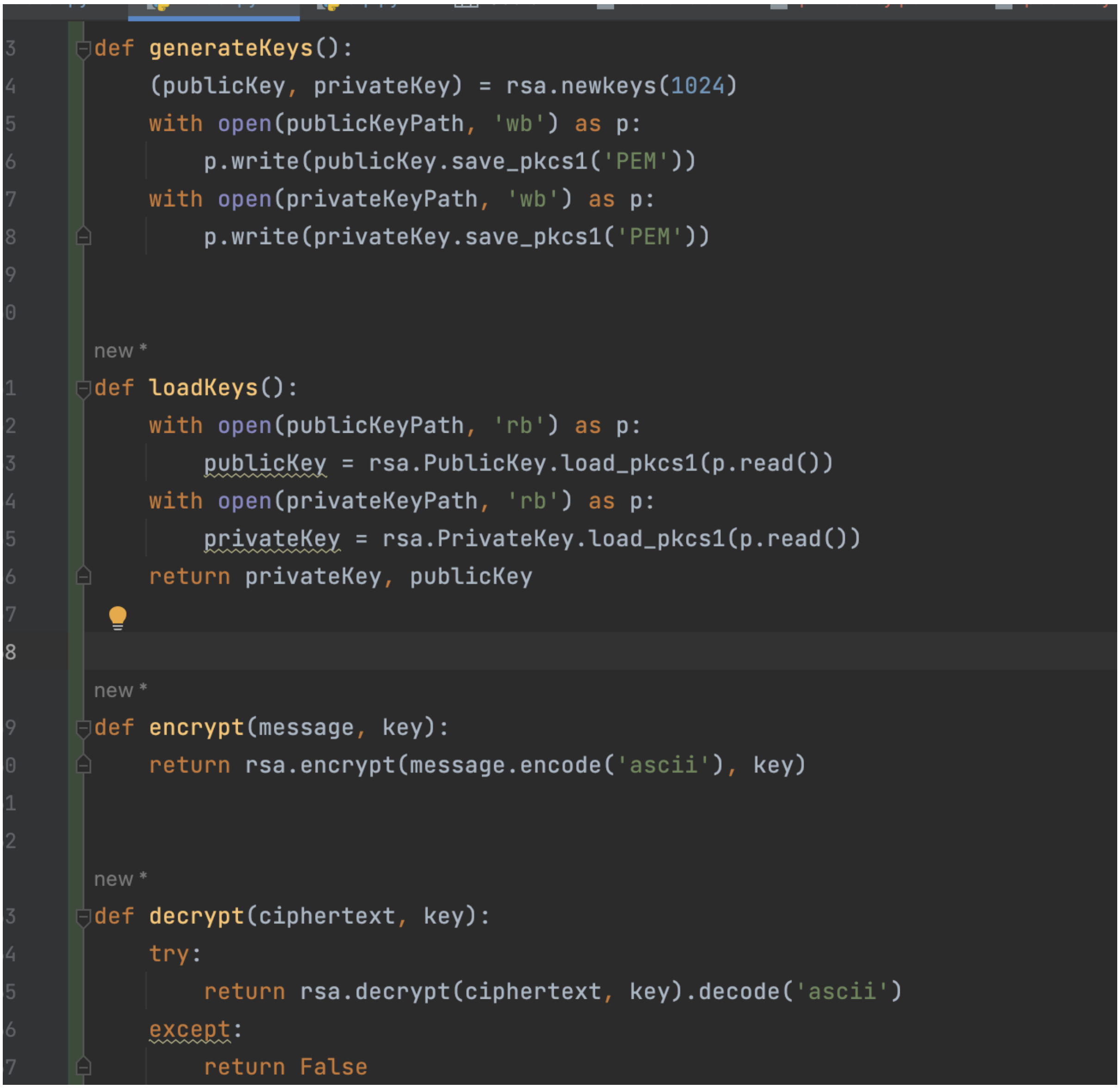
Fingerprints

SHA-256 fingerprint	C7 7C 0A 6C 27 3F BD A1 7D 13 F3 C2 0C 3E 2B D3 4A E2 DC D1 9C DD F7 49 3E B8 6A 24 0C 8B 58 B7
SHA-1 Fingerprint	13 3F 17 7E E4 28 69 DA 46 6E 9F 12 D6 59 97 67 8C 18 69 49

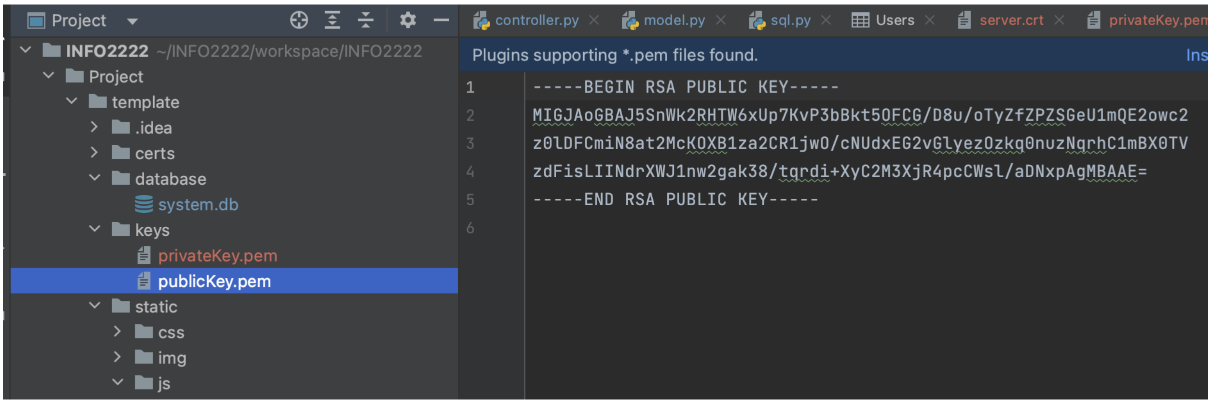
For password transmission part:

We utilise Public Key Encryption(RSA). The server will generate a public-private key pair and store them. When a user register or login by typing the password, the front end will get the public key and encrypted with the password and sent the data back to the server. The server will utilise its private key to decrypt the data and retrieve the password and store or check it by invoking methods in sql file.

Here are the screenshots of Public Key Encryption:



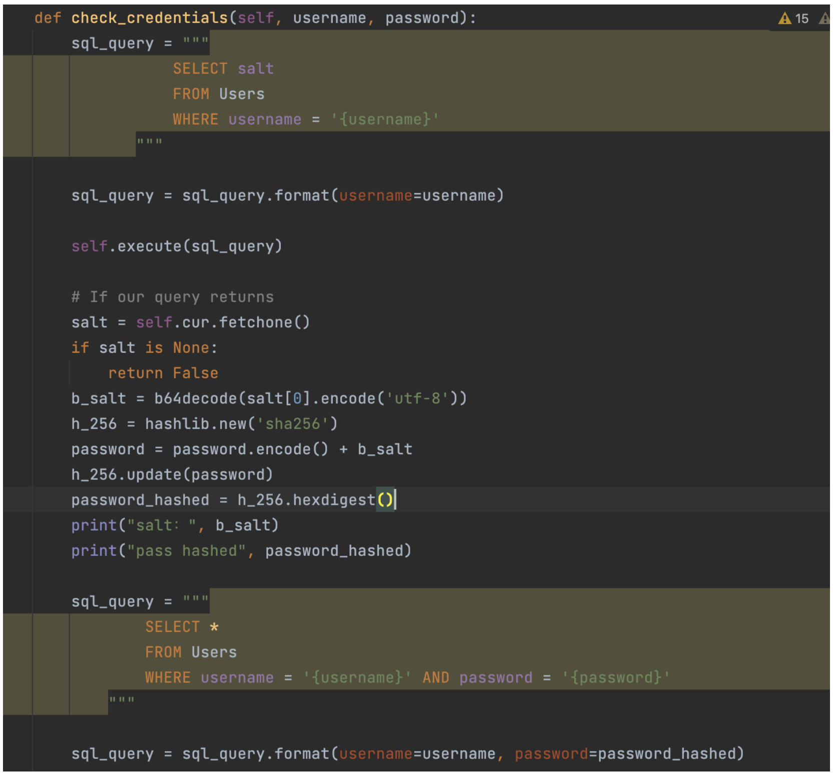
```
def generateKeys():
    (publicKey, privateKey) = rsa.newkeys(1024)
    with open(publicKeyPath, 'wb') as p:
        p.write(publicKey.save_pkcs1('PEM'))
    with open(privateKeyPath, 'wb') as p:
        p.write(privateKey.save_pkcs1('PEM'))


new *
def loadKeys():
    with open(publicKeyPath, 'rb') as p:
        publicKey = rsa.PublicKey.load_pkcs1(p.read())
    with open(privateKeyPath, 'rb') as p:
        privateKey = rsa.PrivateKey.load_pkcs1(p.read())
    return privateKey, publicKey


new *
def encrypt(message, key):
    return rsa.encrypt(message.encode('ascii'), key)


new *
def decrypt(ciphertext, key):
    try:
        return rsa.decrypt(ciphertext, key).decode('ascii')
    except:
        return False
```



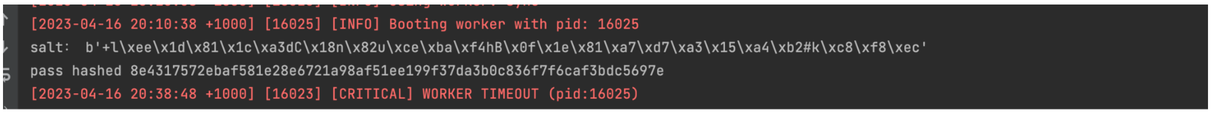
Plugins supporting *.pem files found.

```
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAJ5SnWk2RHTW6xUp7KvP3bBkt5OFCG/D8u/oTyZfZPZSGeU1mQE2owc2
z0lDFCmiN8at2McKOXB1za2CR1jwO/cNUdxEG2vGlyezOzkq0nuzNqrhC1mBX0TV
zdFisLIINdrXWJ1nw2gak38/tqrdi+XyC2M3XjR4pcCWsl/aDNxpAgMBAAE=
-----END RSA PUBLIC KEY-----
```

Check password:
After receive the password sent by the user, our server will get the salt value stored in our db and use the same method to get the hashed password and compare it with the hashed password stored in db. If everything is correct, the users will successfully log in and see their friends list.
Here are the screenshots of checking password:



```
def check_credentials(self, username, password):
    sql_query = """
                SELECT salt
                FROM Users
                WHERE username = '{username}'
            """

    sql_query = sql_query.format(username=username)

    self.execute(sql_query)

    # If our query returns
    salt = self.cur.fetchone()
    if salt is None:
        return False
    b_salt = b64decode(salt[0].encode('utf-8'))
    h_256 = hashlib.new('sha256')
    password = password.encode() + b_salt
    h_256.update(password)
    password_hashed = h_256.hexdigest()
    print("salt: ", b_salt)
    print("pass hashed", password_hashed)

    sql_query = """
            SELECT *
            FROM Users
            WHERE username = '{username}' AND password = '{password}'
        """

    sql_query = sql_query.format(username=username, password=password_hashed)
```



```
[2023-04-16 20:10:38 +1000] [16025] [INFO] Booting worker with pid: 16025
salt:  b'+l\xee\x1d\x81\x1c\xa3dC\x18n\x82u\xce\xba\xf4hB\x0f\x1e\x81\xa7\xd7\xa3\x15\xa4\xb2#k\xc8\xf8\xec'
pass hashed 8e4317572ebaf581e28e6721a98af51ee199f37da3b0c836f7f6caf3bdc5697e
[2023-04-16 20:38:48 +1000] [16023] [CRITICAL] WORKER TIMEOUT (pid:16025)
```

Message Sending:
For the message sending part, we utilise end-to-end encryption to ensure that only the two messaging users can know the text. When messaging, the front end will retrieve the public key of the receiver and encrypt it with the sender's private key. The derived key will be used to encrypt the message and data will be stored in the database. When the receiver want to receive message from the sender, the receiver click on the receive button and get the encrypted message and the public key of the sender and decrypt the message with the receiver's private key.
The javascript files of this part is reference from this site:
https://getstream.io/blog/web-crypto-api-chat/

Limitations:
Our group is not familiar with HTML and Java Script, for the message sending part, we could not fix some bugs in time so we disabled the encryption part to make the basic exemplary flow to work properly.

Contributions:

Front end pages: Emily
Backend database: Derrick