**7DEVELOPMENT GUIDELINE**

**Description.**

In this document, you will be introduced with the application data flows and received the instruction of using its functions properly. All related methods, modules and functions will be described clearly, then we can jump to the overall workflow of the application.

**Summary.**

There are 3 modules that operated in the application : a server side, a client interface and an API module which manages the paths between server and database.

The client and server, communicating with each other by using socket connection. Sent and received data are executed with socket command lines.

The client is the module which user inputs data in and extracts data out, it communicates with the application through the server module.

The server module, receives data from client, will connect to the API module through the URL <http://127.0.0.1:5000/> , based on the unique data sent from client, the URL will be designed to be able to transfer the data to API module.

The server also holds some methods that aren’t contained by the API module, they play some small parts for a function.

In API module, the transferred data from server will go through various methods here. A database is connected to this module, so it will call a link path to database every time a new data arrived, and the data could be put in the database, replacing the current ones/used for extracting another data from the database.

**Cryptanalytic reference.**

The key pair of both trainee and supervisor is used for most of the functions. You will see supervisor’s public key is used when trainee just finished half of the record. Lately, supervisor will decrypt the record with this own secret key for marking. When the record is filled up, it is encrypted by trainee’s public key then stored in database for once. If the trainee wants to observe his record, he just needs to use his secret key for decryption.

**Data storage.**

The data is stored in the database, which is used with SQLite Python – supporting for python editor. All methods with related database used SQLite commands to execute the function.

**Security requirement.**

We have satisfied the prerequisite conditions that : user secret key, will only be used in the client module, since it is not security if the server could hold both public and secret keys.

**Modules, methods and functionalities.**

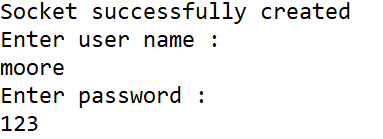
The rating application will operated with 5 functions within 3 modules, they are :

* Task performing: made by the trainee, information that filled by trainee will be uploaded to the database.
* Assessment making: supervisor will mark and rate the tasks that answered by trainee. The marked record and supervisor’s new assessment will be uploaded to the database for the last time.
* Record observation: trainee could view his record by entering a few unique information so the system could locate then deliver his record to him.
* Profile sharing: trainee will send his decrypted record to the receiver by using email.
* Employee, hasing checking: user could check if the employee/hashing is available (this function is mainly from Blockchain features.

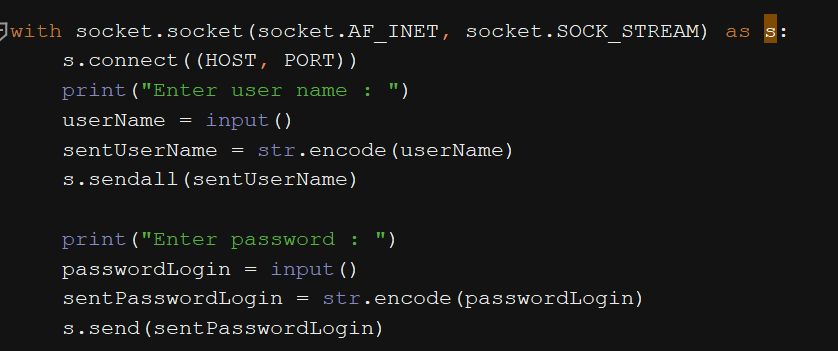
An unique condition is made when APIs module is about to be called by the URL from server. This condition will clarify then place the data from server to the rightful method in API modules.

**1.The login. (return the Hello, <Employee\_Name>)**

When logging in the application, user will need to enter their username and password. Username “moore” and password “123” are the data that needed to be verified before accessing the application.



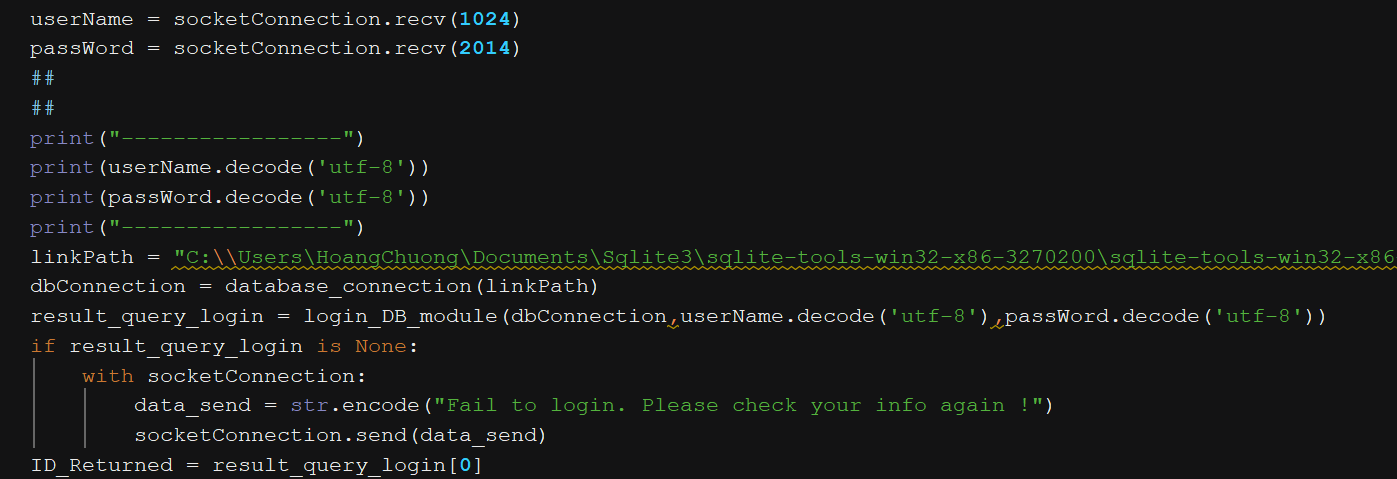
In the client side, username and password fields are filled then sent by socket.



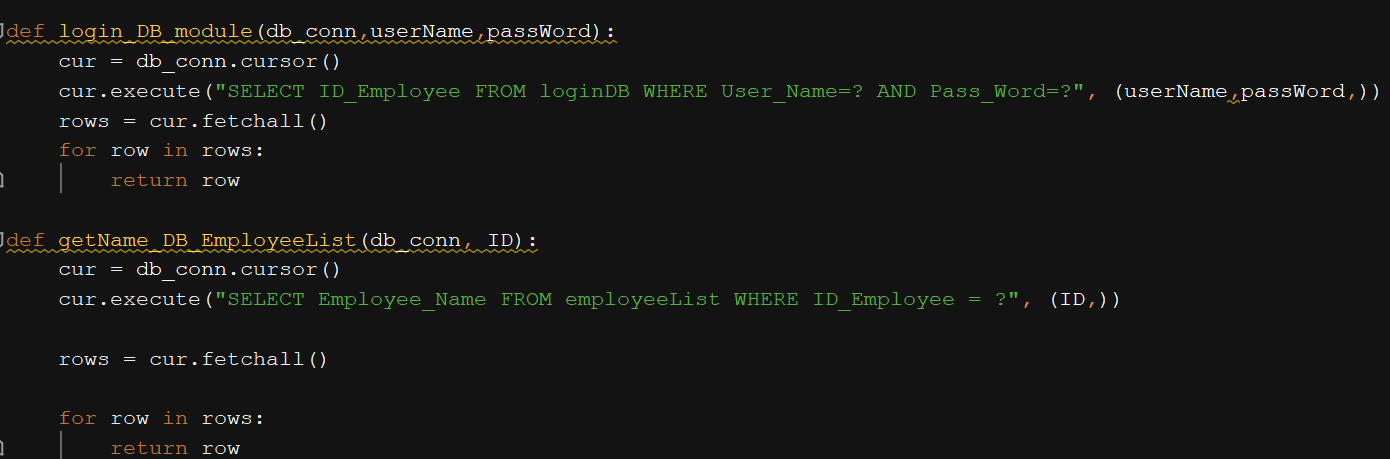
*<RatingApp\_Client.py>*

They are received in the server side

A link path to the database is opened. The received username and password are used to get the ID employee from the loginDB table.

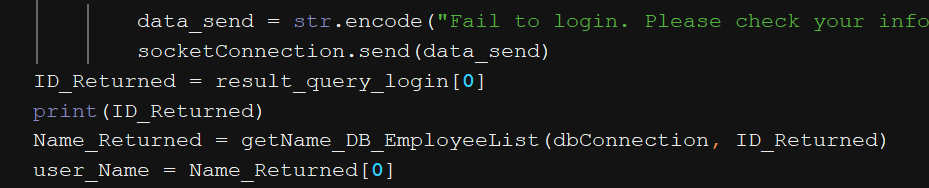


*<RatingApp\_Server.py>*



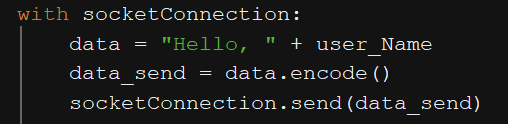
*<RatingApp\_Server.py>*

The ID employee is then queried to the employeeList table, to get the Employee\_Name field.



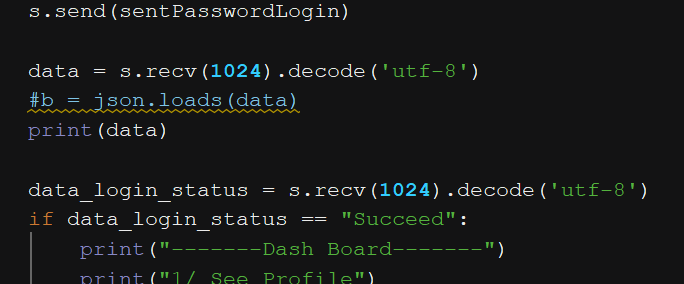
*<RatingApp\_Server.py>*

Server now sent the Employee\_Name to the client side through socket.

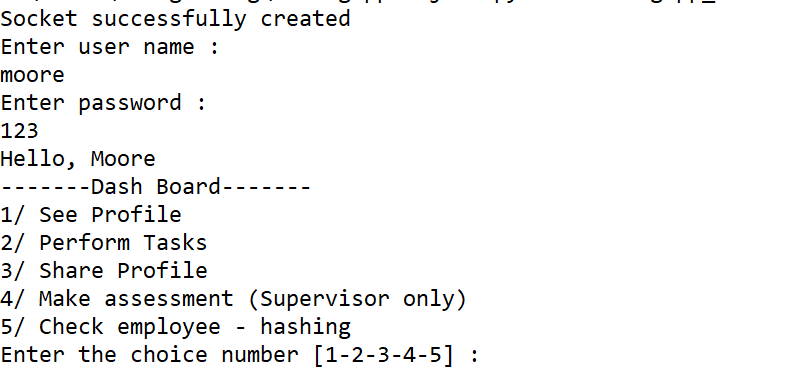


*<RatingApp\_Server.py>*

In the client interface, a “Hello, Moore” appeared, along with it, the dashboard consists of 5 functions for user.



*<RatingApp\_Client.py>*



**2. Function addressing.**

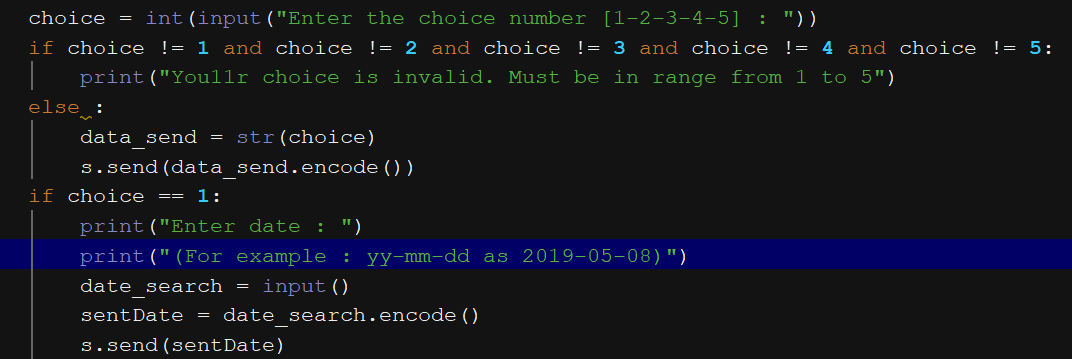
There are 5 options that user could choose. The 4th option is initially for supervisor when making assessment to trainee record. The 1st 2nd 3rd are for trainee when doing tasks, seeing record and sharing record.

The 5th option, however, any user could use it since it doesn’t require cryptographic method.

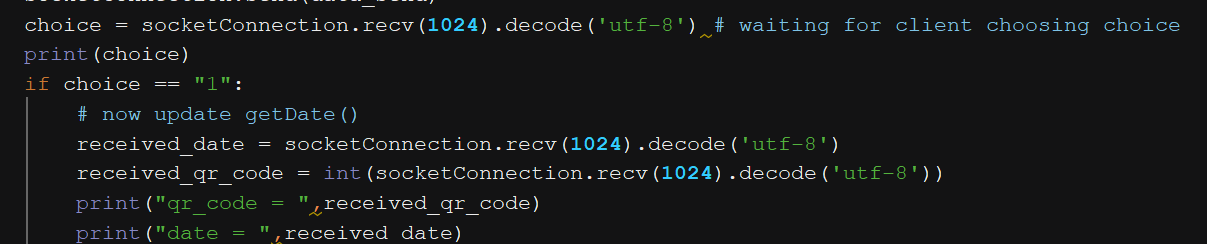
When user chose the choice (by entering the number) it would be sent to the server.

In server side, depending on the choice number, it will initialize a “waiting state” for client to send data.

(for example, if the choice is 1, client continuously enters the date and ID record fields, the server side will wait until it received the data from client)



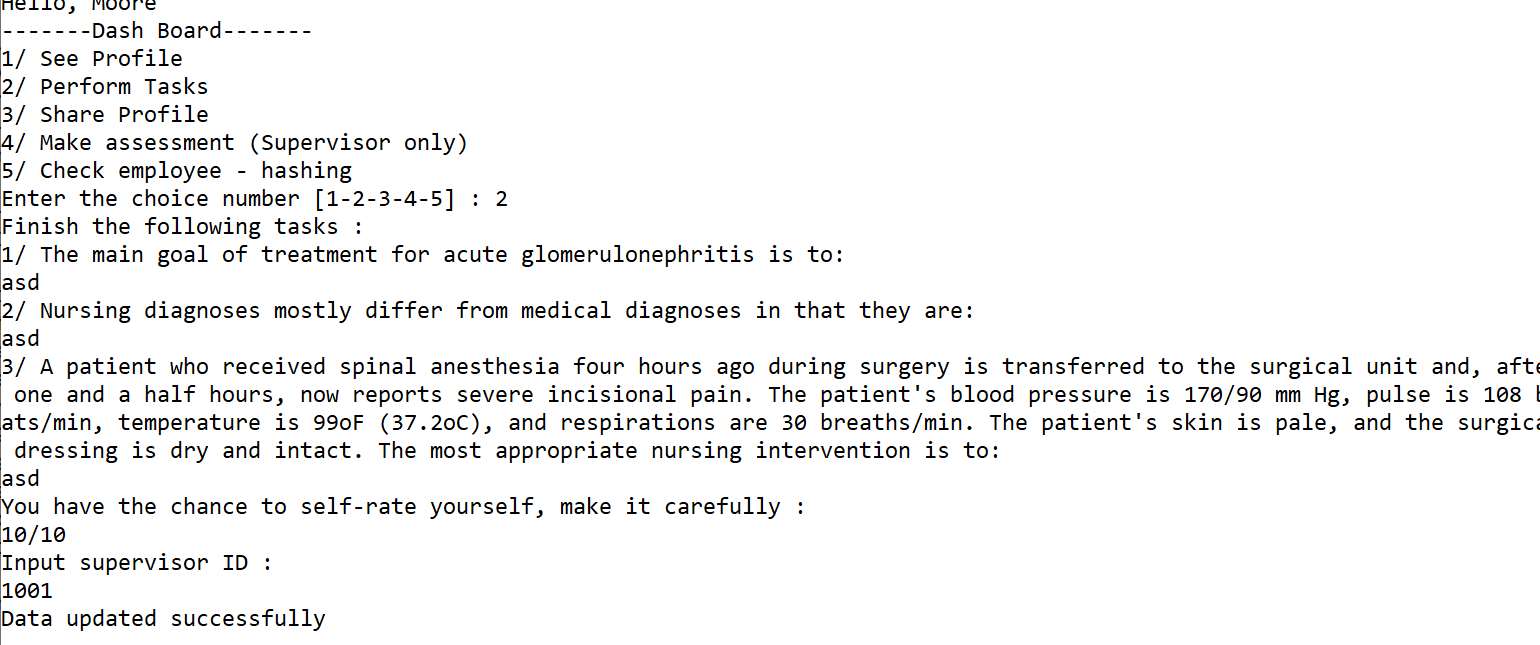
*<RatingApp\_Client.py>*



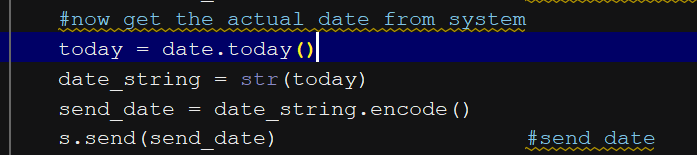
*<RatingApp\_Server.py>*

**3. Task performing.**

As the choice is 3, a list of task questions will appear in client interface, answering one by one will address user to a new blanked information to type in.

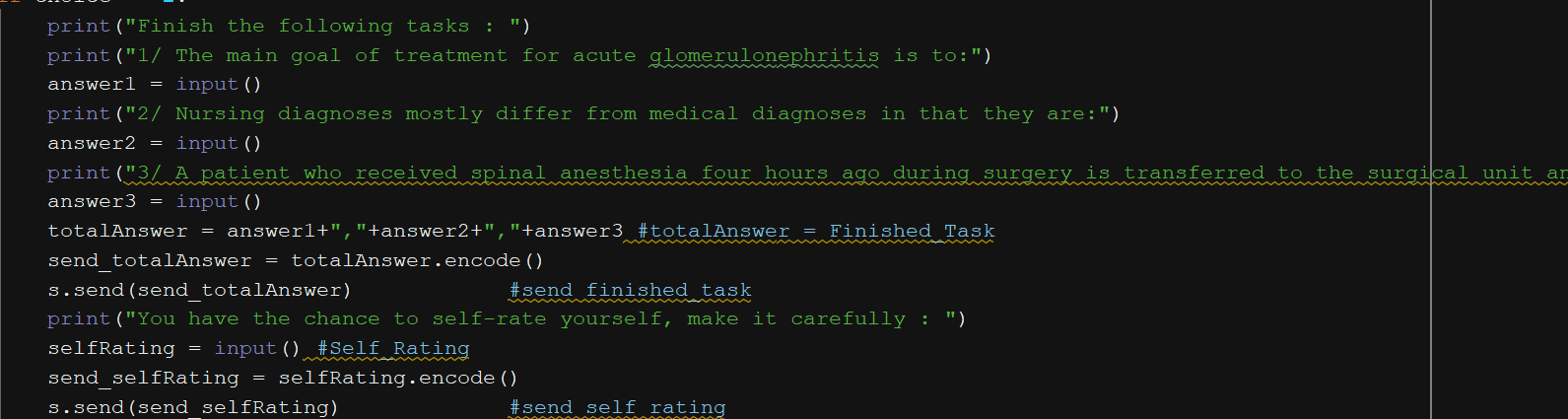


The date that trainee performing the task is created by a random method, taken from real life date.



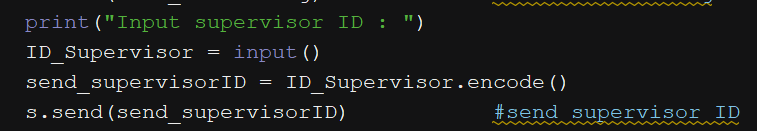
*<RatingApp\_Client.py>*

All 3 questions and a self-rating are filled, sent to server side.



*<RatingApp\_Client.py>*

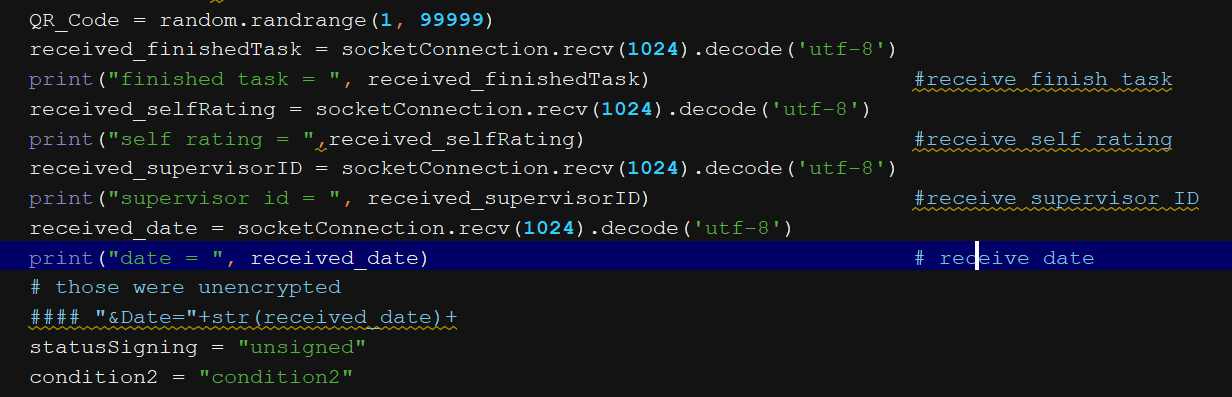
An ID supervisor is required to be entered by trainee, as this person is chosen to mark trainee record.



*<RatingApp\_Client.py>*

In server side, a condition is added – Signing\_Status = “unsigned”, which means the record haven’t been marked by supervisor. Lately, supervisor will find the records that haven’t been assessed based on this condition.

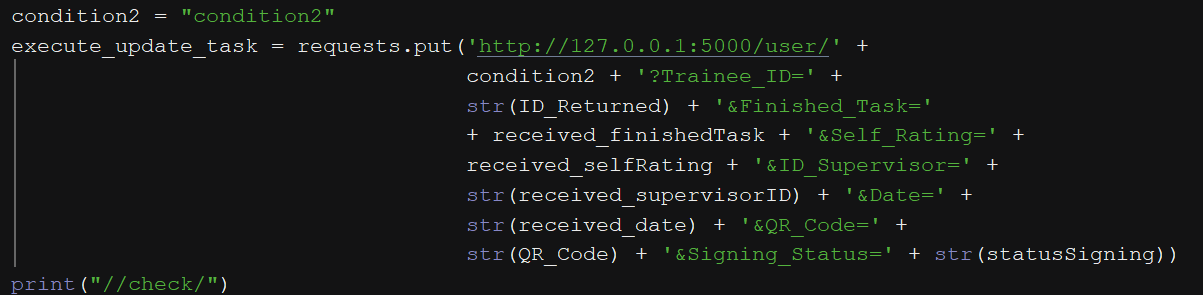
A random ID record (QR code) is also created, this ID record will then be updated to the database, initialized as the holder for that record.



*<RatingApp\_Server.py>*

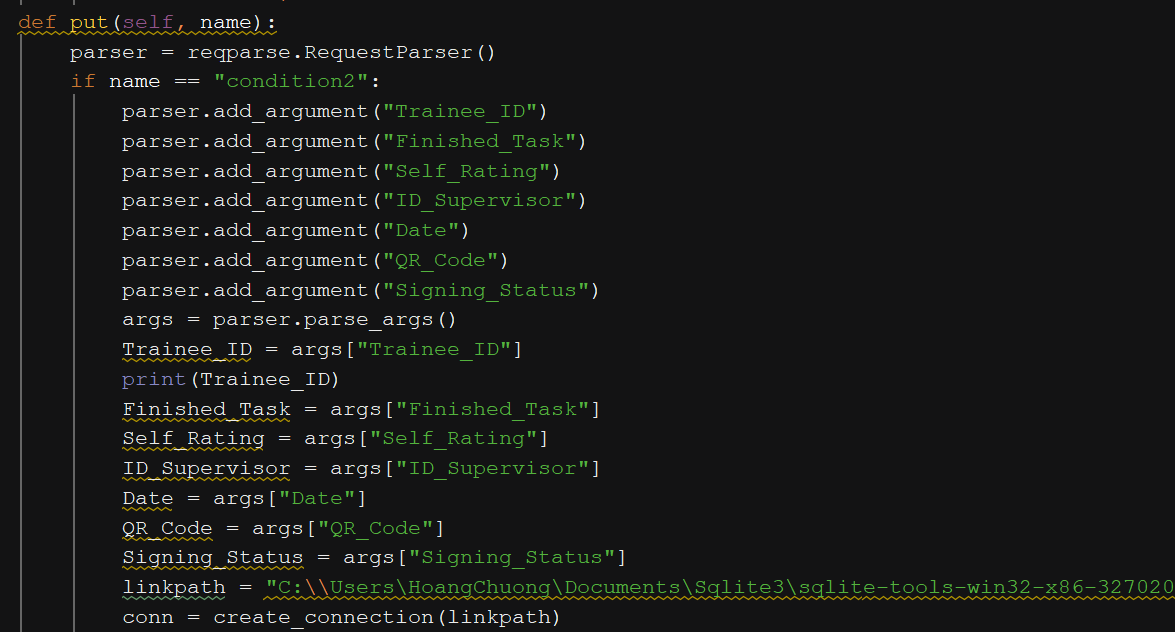
An URL is initialized, with the data that sent by client and the signing condition.

The condition for requiring the specified API is “condition2”



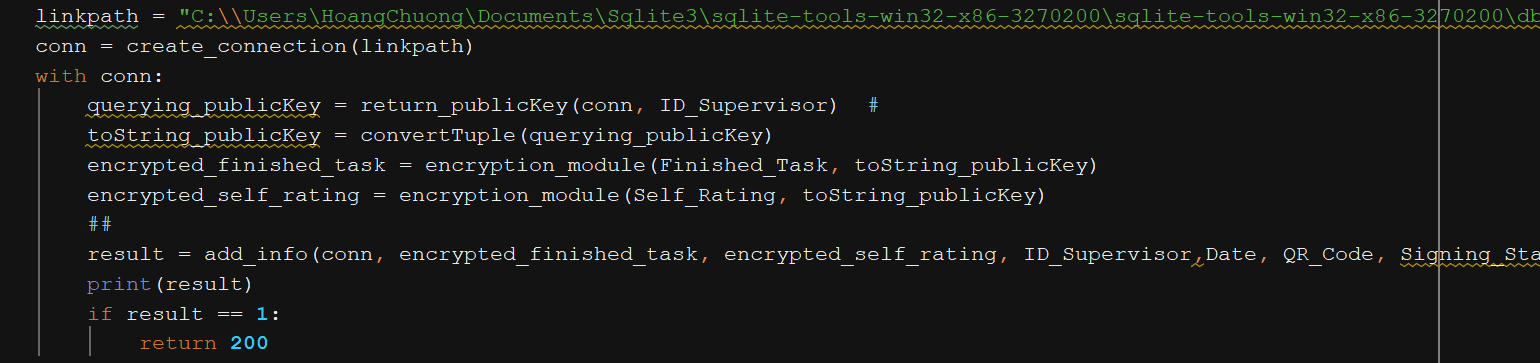
*<RatingApp\_Server.py>*

When it reaches APIs module, the functionality will address the data in the specified condition.

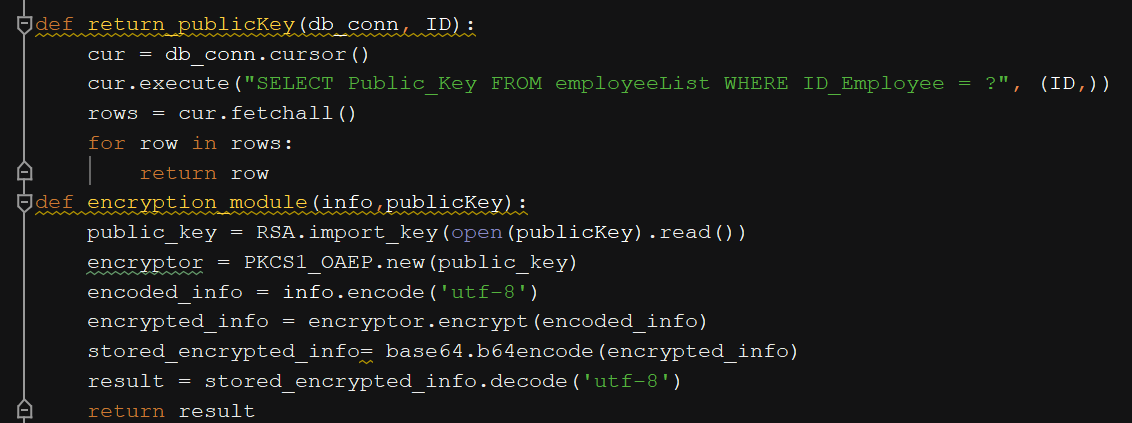


*<RatingApp\_Full\_APIs.py>*

In the APIs module, the unique data is been through the cryptographic module : The previous ID supervisor sent by trainee is queried to the database through the method public key returning, it will return that supervisor’s public key, used for encrypting the unique data (Finished\_Task and Self\_Rating)

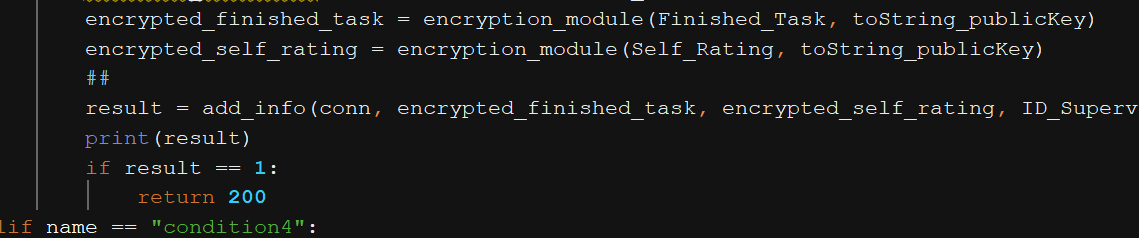


*<RatingApp\_Full\_APIs.py>*

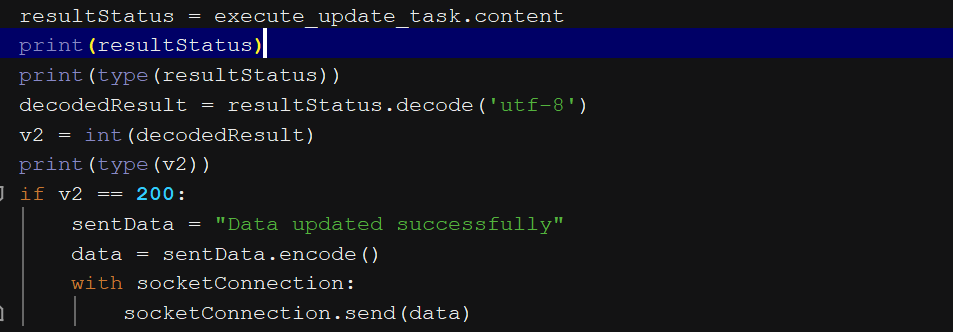


*<RatingApp\_Full\_APIs.py>*

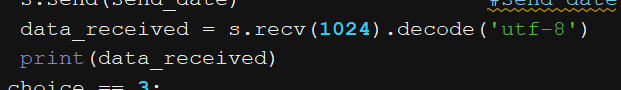
An updating method is initialized, updated all data to the database, when it successes, the APIs module will return an alarm to server, if the alarm is matched with the condition, server will send the “Data updated successfully” to client, ending the progress.



*<RatingApp\_Full\_APIs.py>*



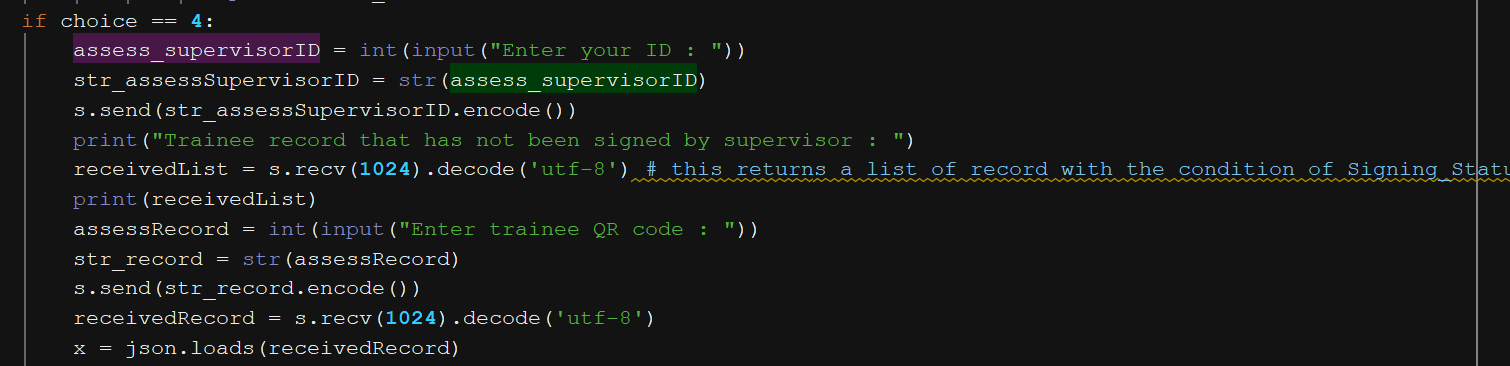
*<RatingApp\_Server.py>*



*<RatingApp\_Client.py>*

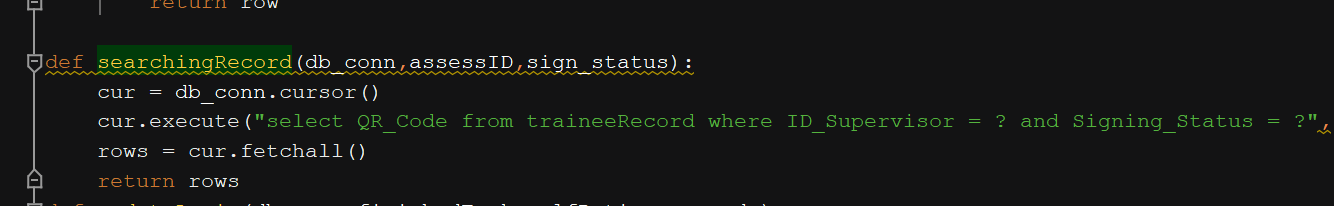
**4. Assessment initializing.**

The supervisor (badcc), choosing the option 4th to access the assessment making function. He will enter his ID employee. The ID is queried to the database to find the record which was entered with his ID by the previous trainee and condition of signing as “unsigned”.



*<RatingApp\_Client.py>*

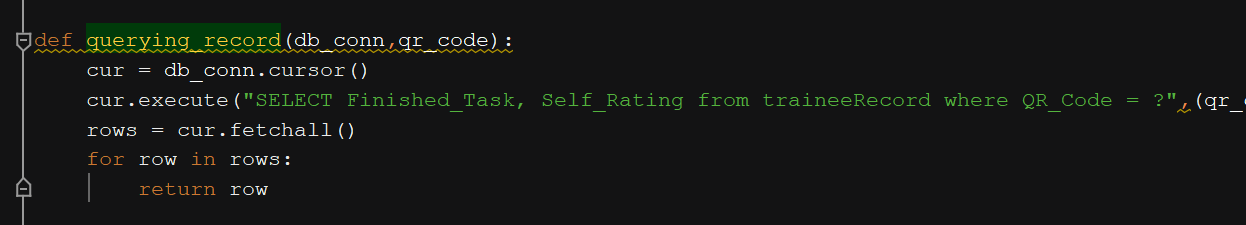
It will return to client interface a list of record which was assigned to the chosen supervisor by trainee and the signing condition is “unsigned”.



*<RatingApp\_Server.py>*

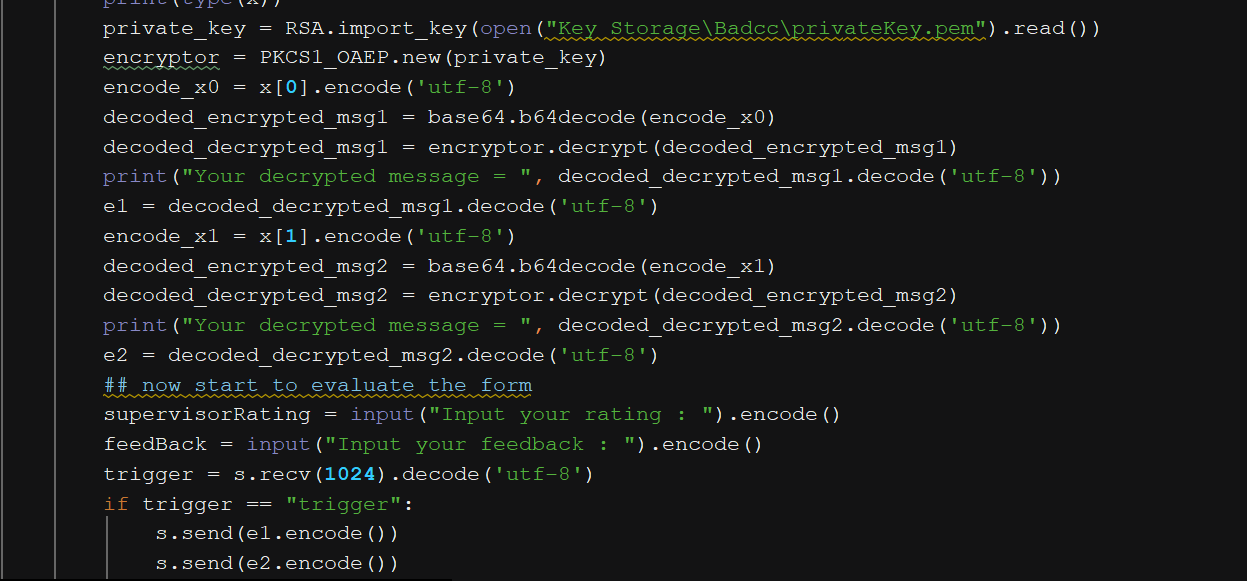
Supervisor inputs the ID record he wants to mark, application will return the encrypted task information to him.

Now, supervisor will decrypt this task information by using his secret key.



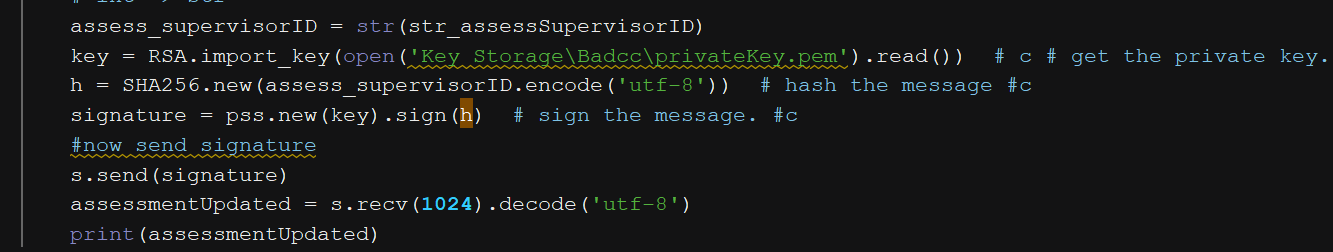
*<RatingApp\_Server.py>*

As the task answers are revealed, supervisor will start marking based on it.



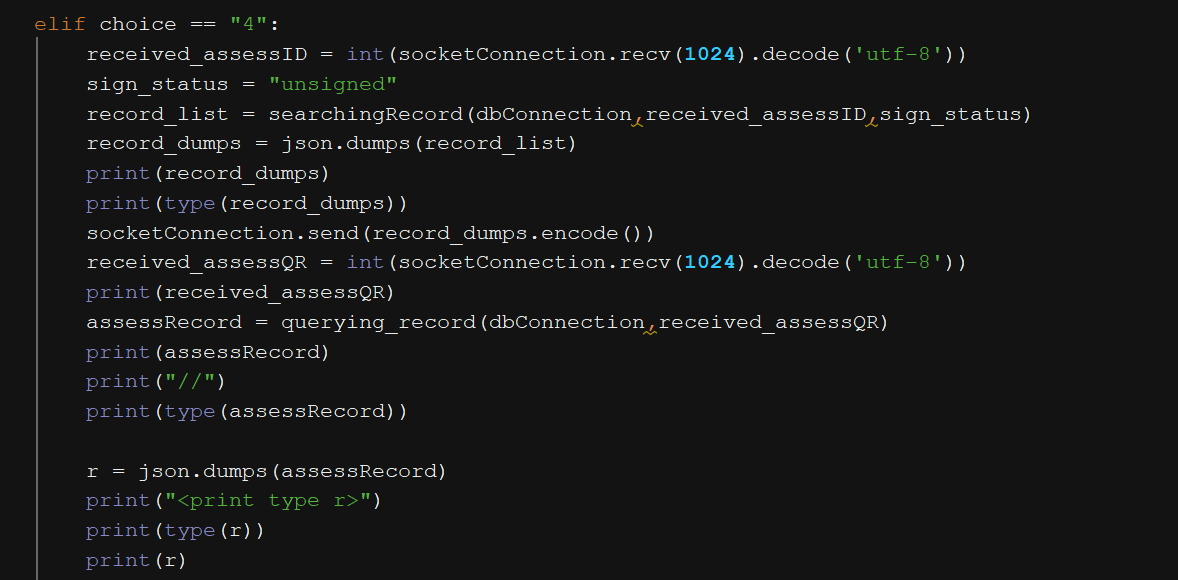
*<RatingApp\_Client.py>*

A new digital signature data is created from ID supervisor, signed by supervisor’s secret key.



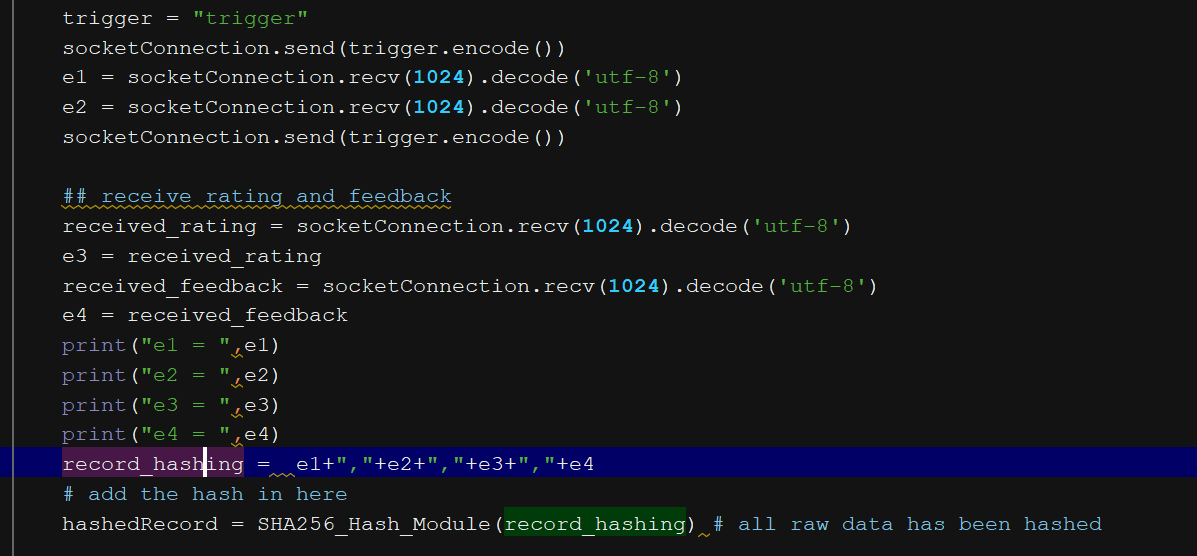
*<RatingApp\_Client.py>*

When he finished marking, all data is sent to the server side.

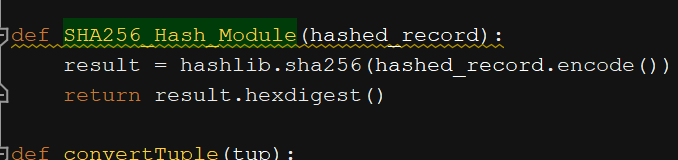


*<RatingApp\_Server.py>*

The decrypted trainee record (Finished\_Task, Self\_rating) and supervisor’s assessment (Supervisor\_rating, Supervisor\_FeedBack) are hashed in the hashing method, return a string of hashing.

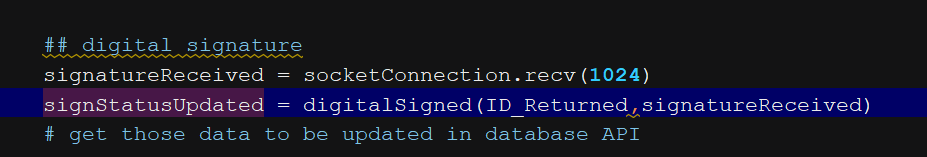


*<RatingApp\_Server.py>*

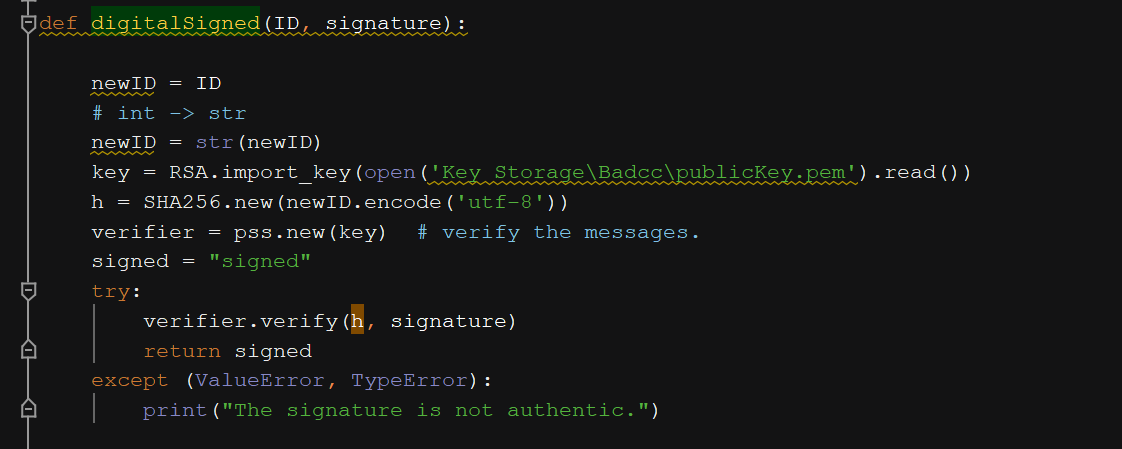


*<RatingApp\_Client.py>*

In server side, the signature that created from the ID supervisor, will be through a method of verifying record. The previous ID ( that was received when logging in application with username and password) is also used to verify this signature. The verification is executed by supervisor public key.



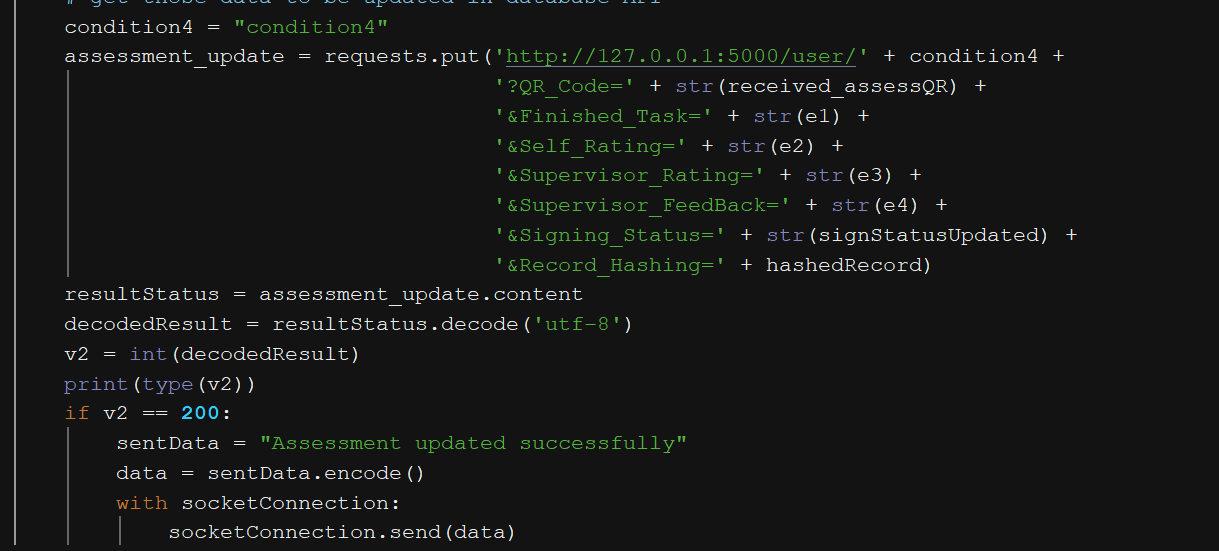
*<RatingApp\_Server.py>*



*<RatingApp\_Server.py>*

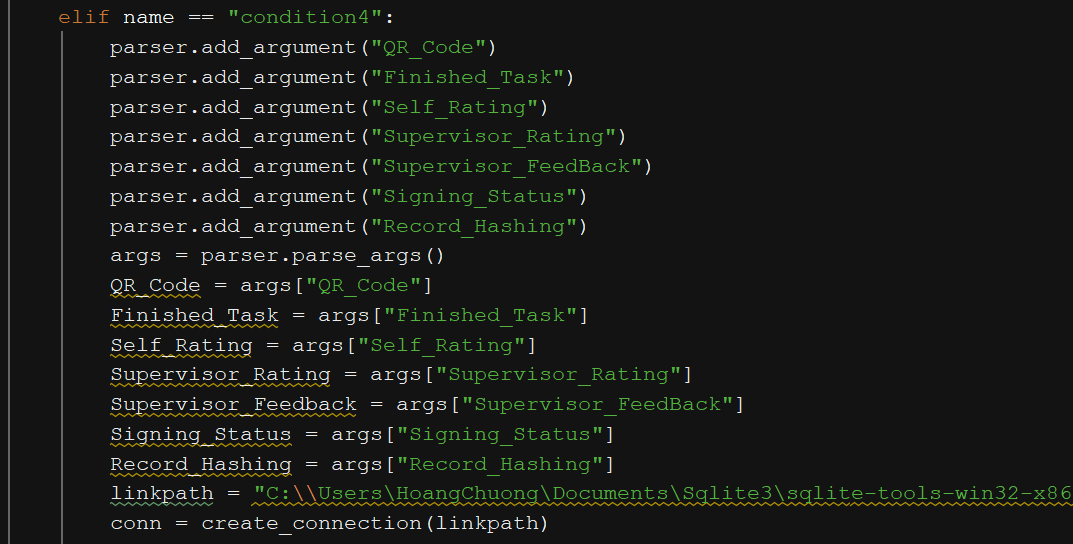
As the digital signature is successfully verified, the signing status as “signed” is returned.

A new URL is initialized, with the specified condition for the rightful function in APIs module, all decrypted data, a new updated signing status and a new record hashing, are pushed to the module.



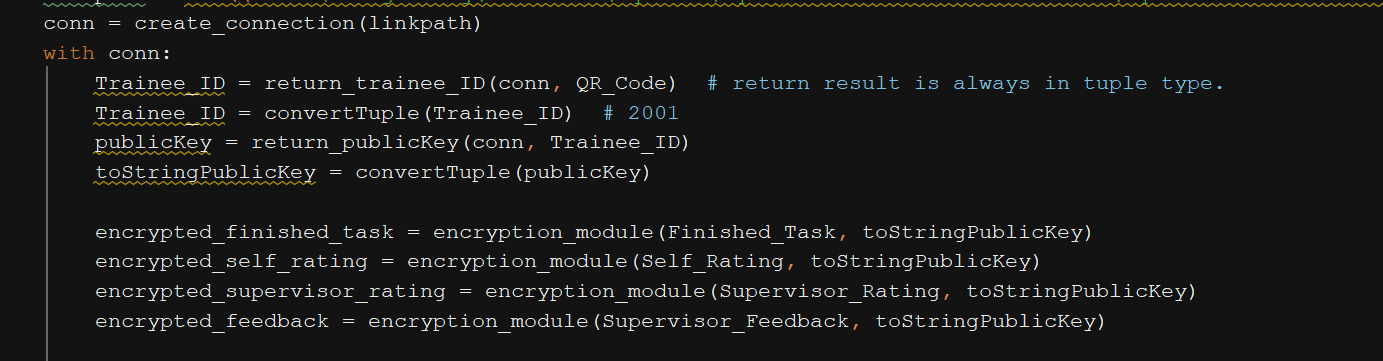
*<RatingApp\_Server.py>*

The condition “condition4” is recognized, data is addressed to the method.

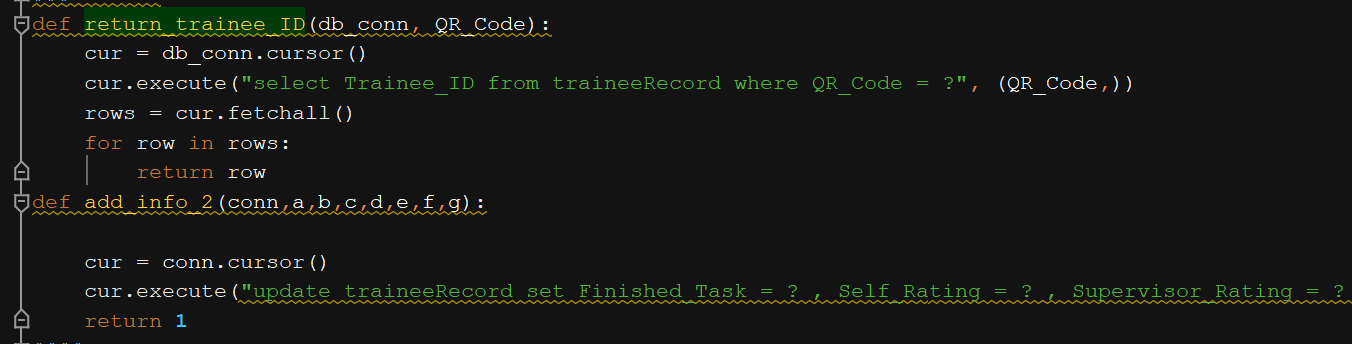


*<RatingApp\_Full\_APIs.py>*

The ID record that was entered by supervisor before, is used to get the ID of trainee who performed the tasks. Then the ID is queried to the database to return the his public key. Those functions are through the 2 methods of return\_trainee\_ID and return\_publicKey.

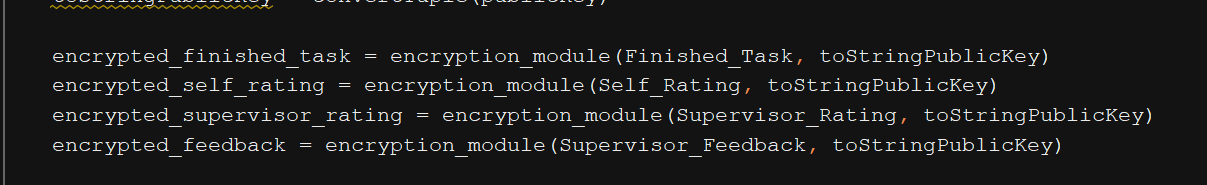


*<RatingApp\_Full\_APIs.py>*



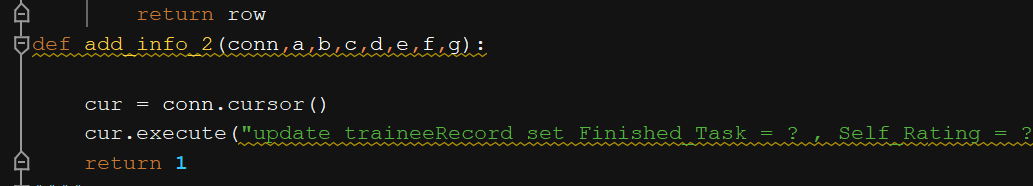
*<RatingApp\_Full\_APIs.py>*

The public key is then used to encrypt all specified data (Finished\_Task, Self\_Rating, Supervisor\_Rating, Supervisor\_FeedBack)



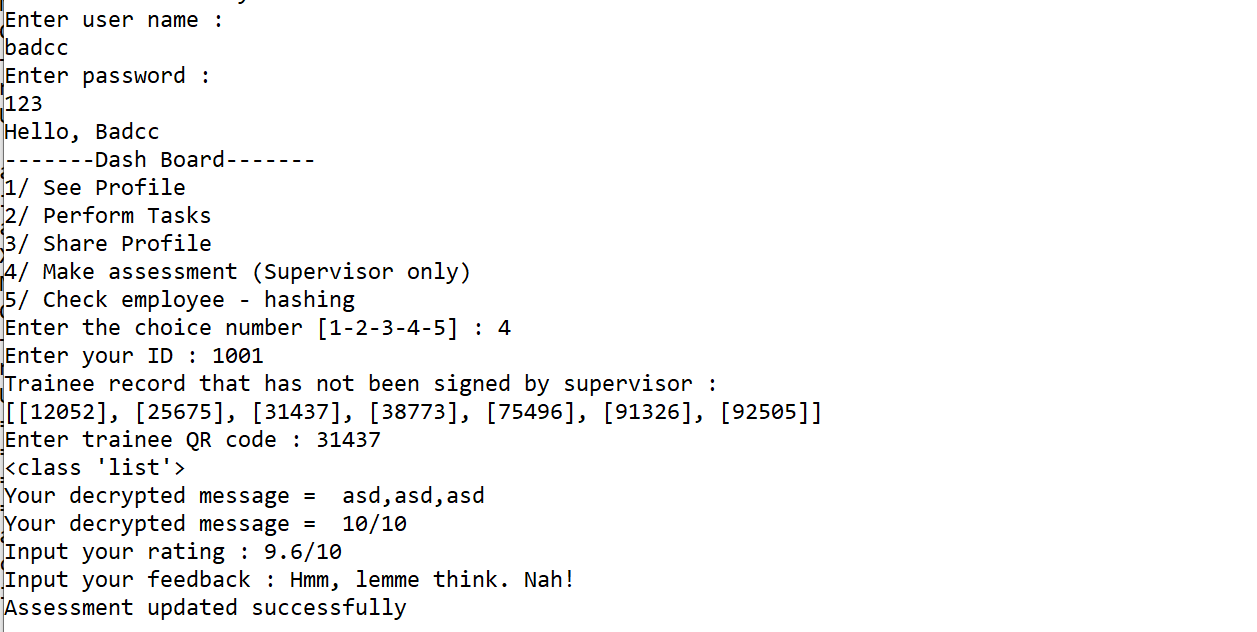
*<RatingApp\_Full\_APIs.py>*

All the encrypted data, signing status and new record hashing are updated to the record database.



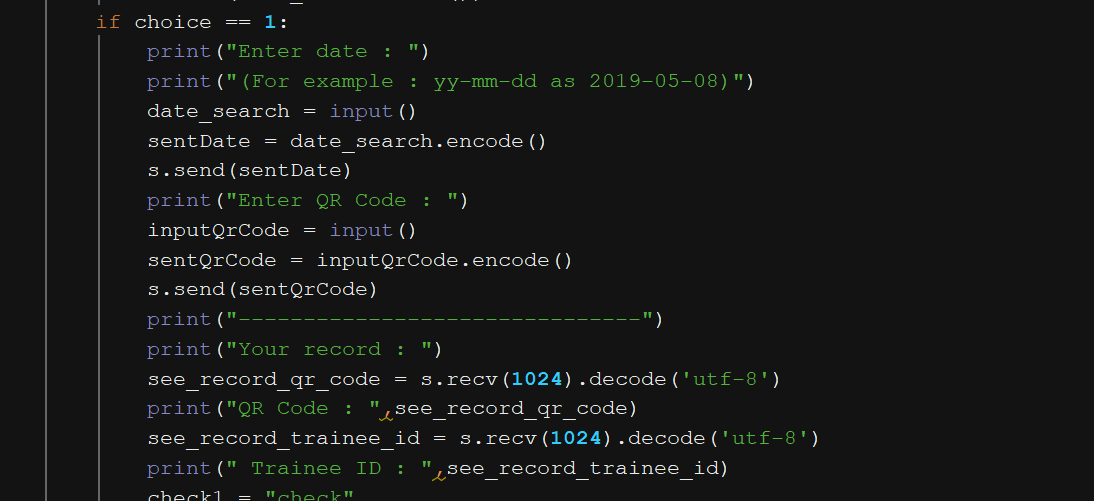
*<RatingApp\_Full\_APIs.py>*

As the result is returned from APIs module, server will send the announcement to client that assessment has been updated.



**5. Record observing.**

Trainee needs to enter the date he did the task questions, then the ID record which was created on that day.



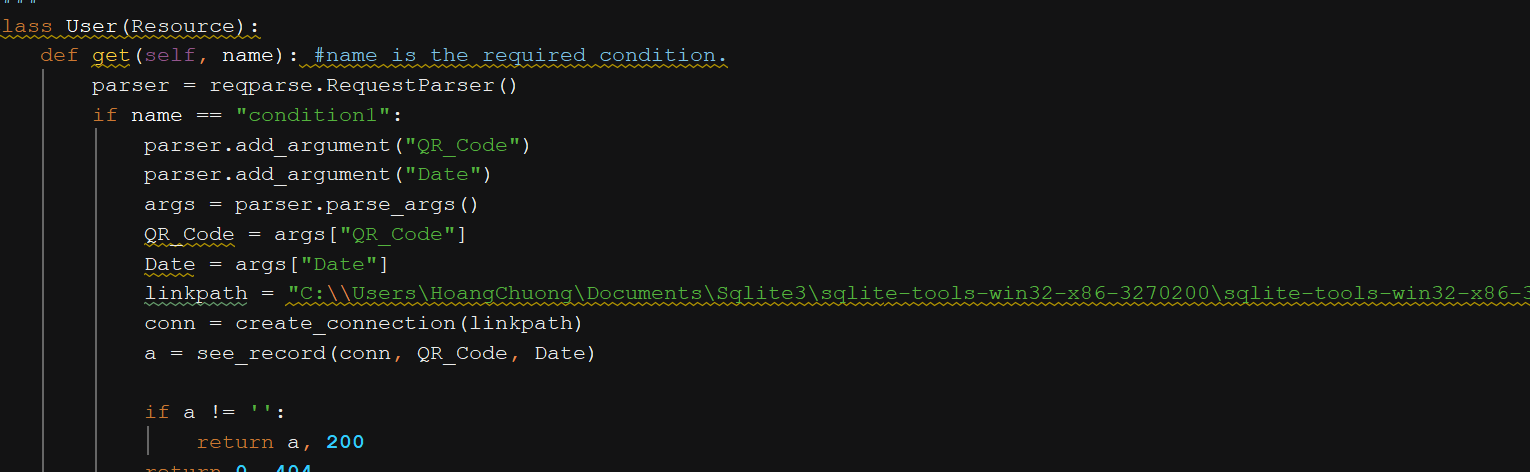
*<RatingApp\_Client.py>*

In server side, the date and ID record are put to the initialized URL, with the condition “condition1”.

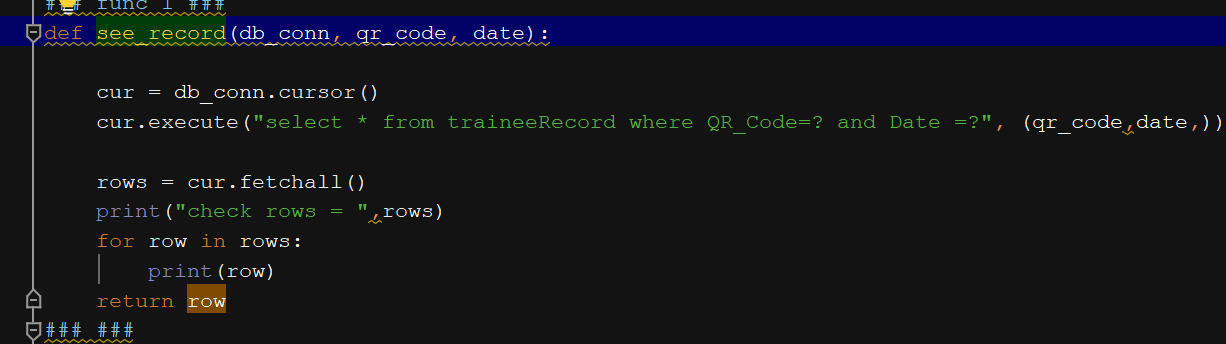


*<RatingApp\_Server.py>*

In APIs module, the condition is recognized, the arrived date and ID record are then used for querying the full record, through the method see\_record.

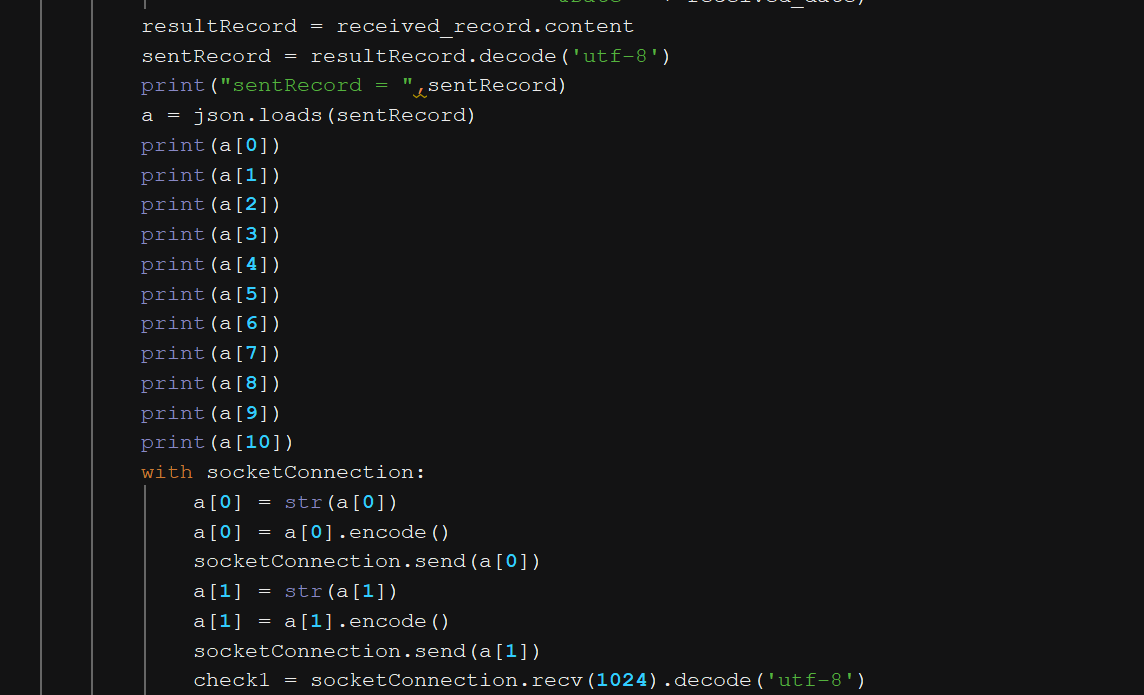


*<RatingApp\_Full\_APIs.py>*

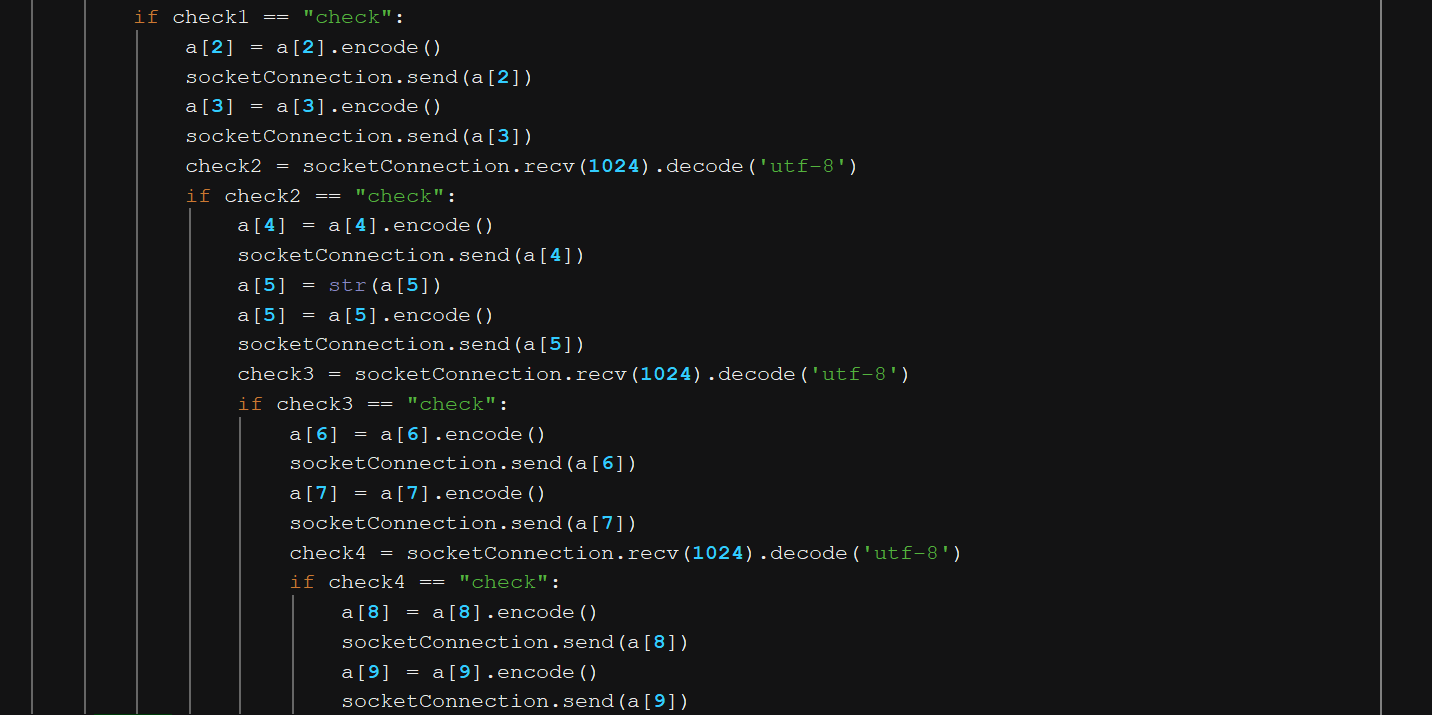


*<RatingApp\_Full\_APIs.py>*

The full record is returned to the server side, now ready for transferring to client interface.

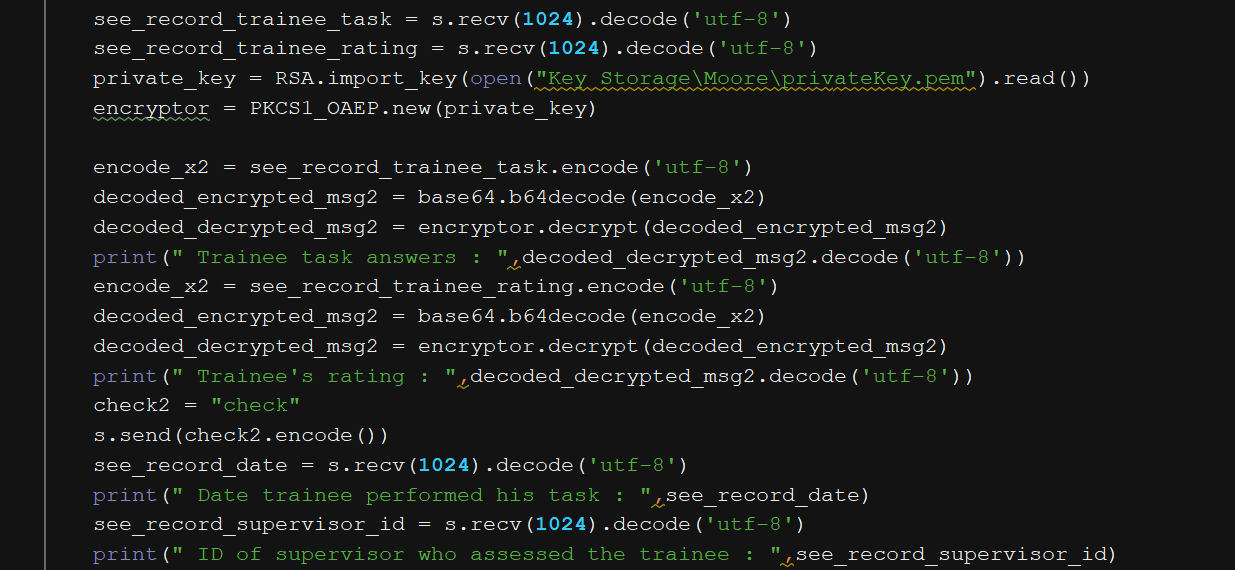


*<RatingApp\_Server.py>*

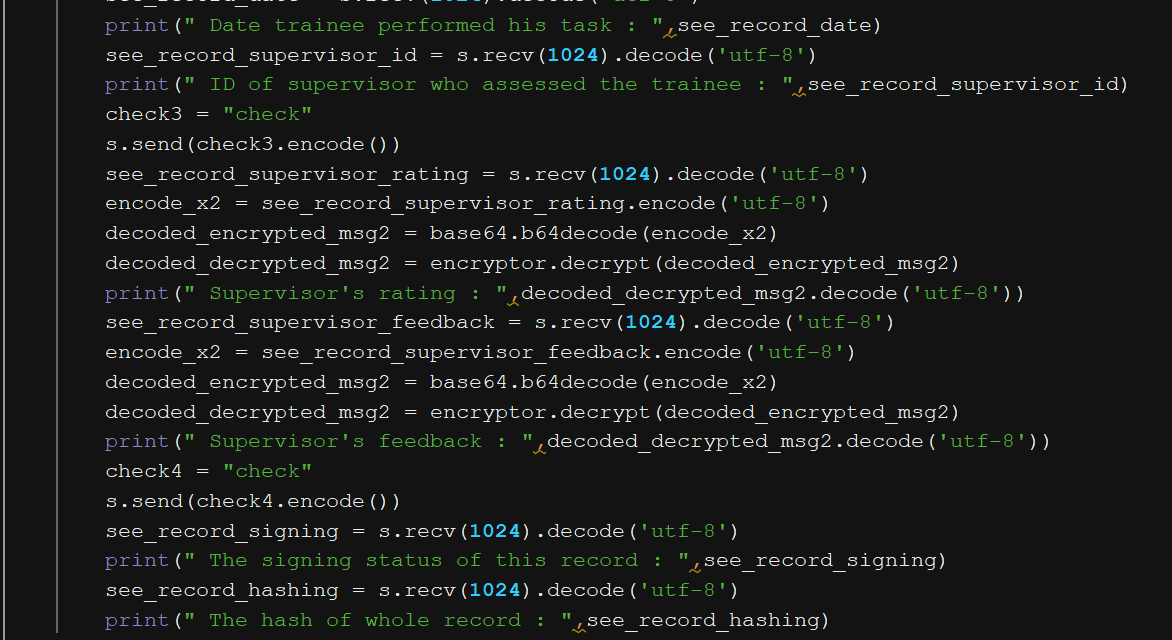


*<RatingApp\_Server.py>*

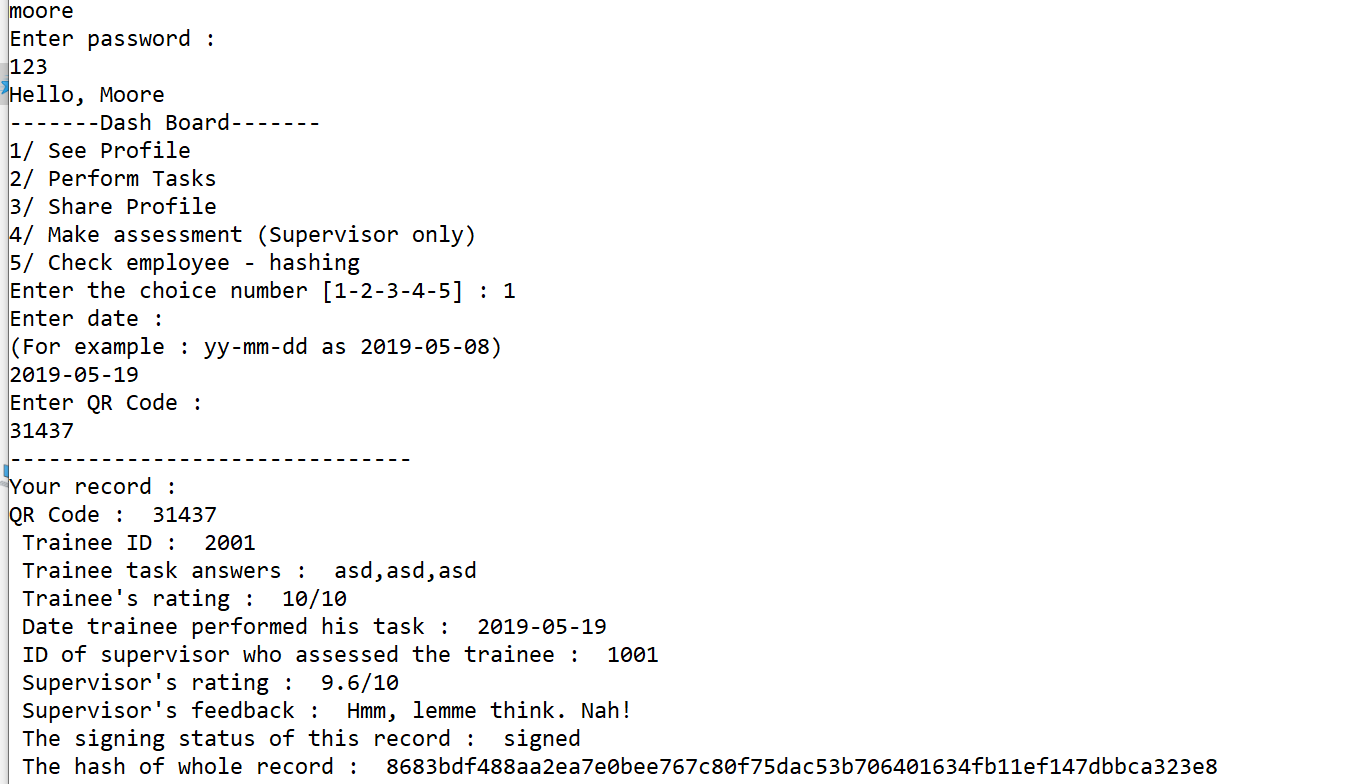
In client interface, after received the record, trainee will use his private key to decrypt the encrypted data, revealing the whole record for observation.



*<RatingApp\_Client.py>*

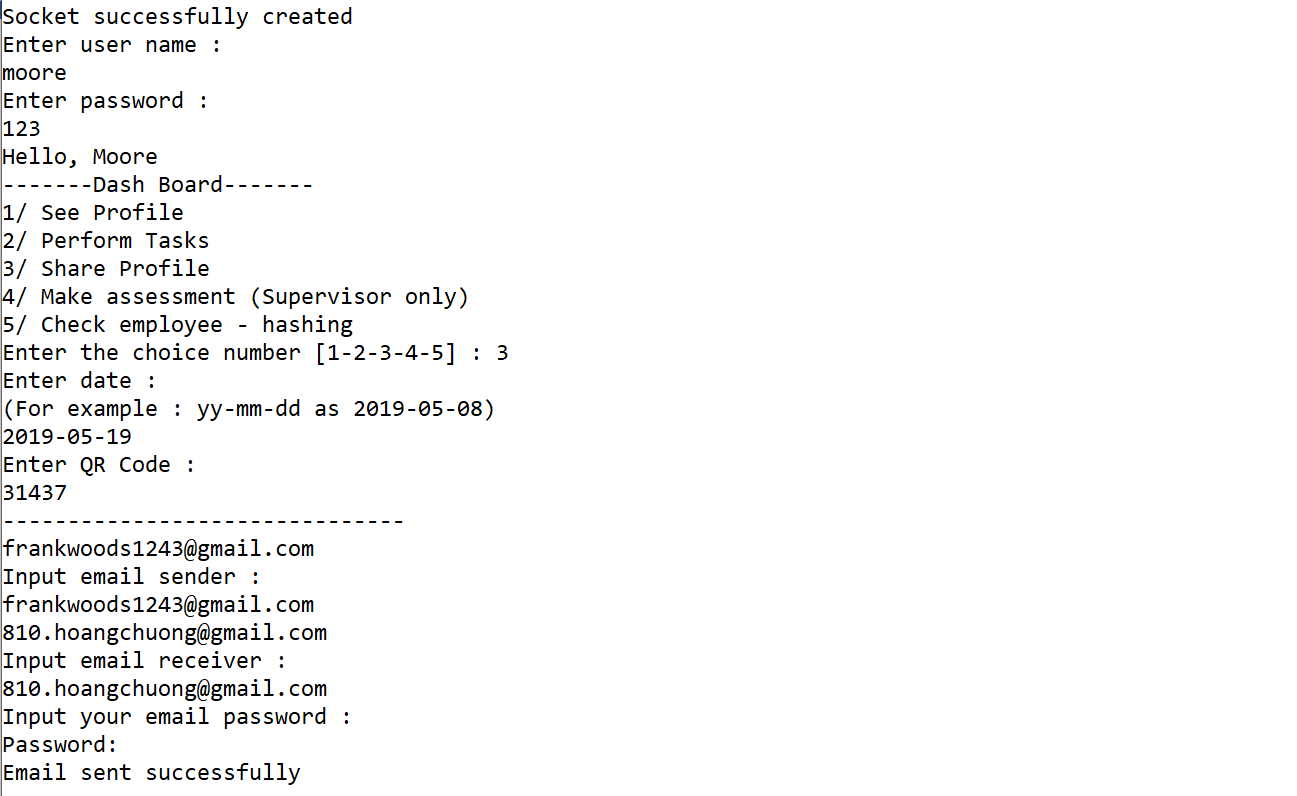


*<RatingApp\_Client.py>*

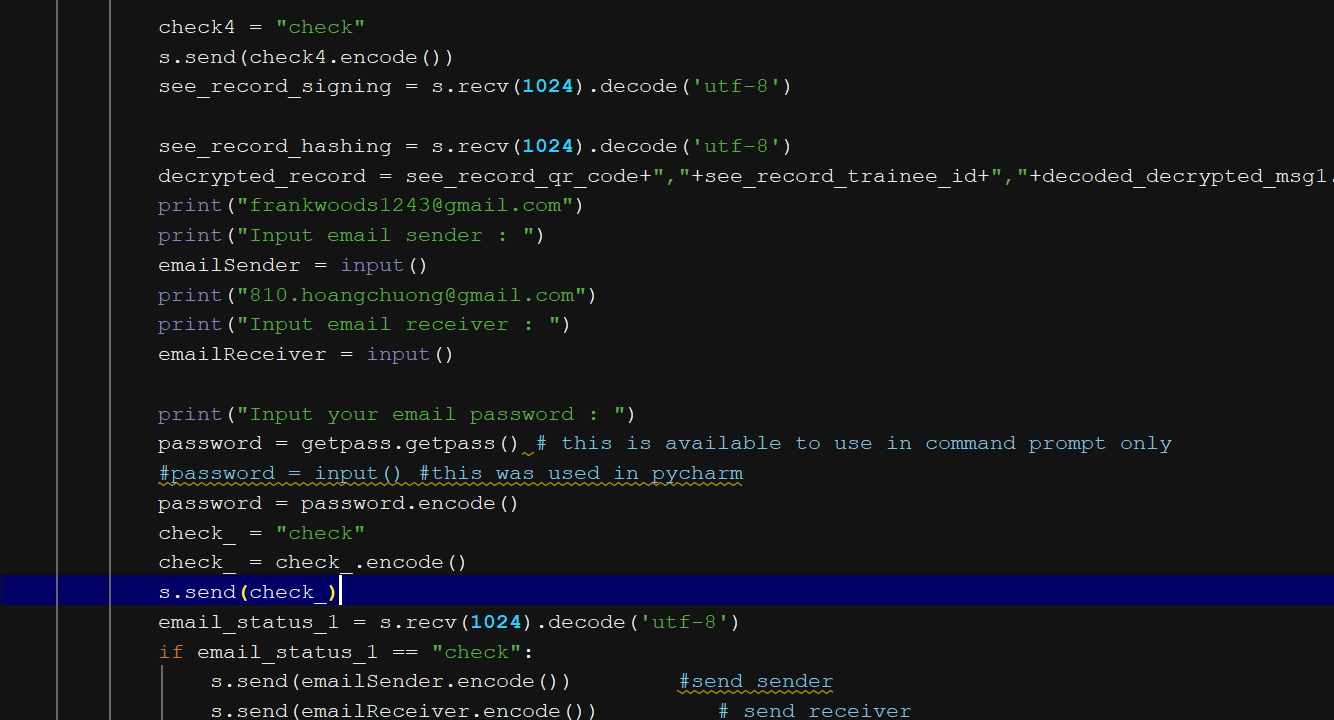


**6. Profile sharing.**

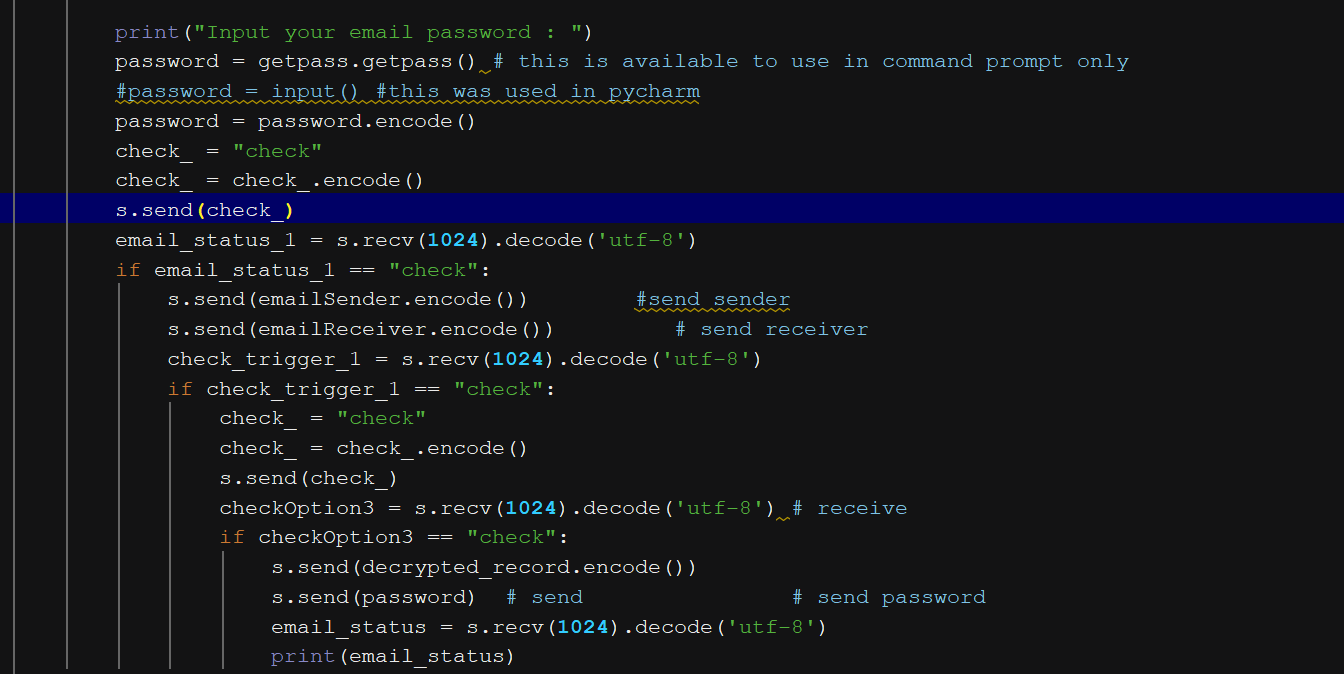
The progress is same as record observation at first - with the related APIs module, getting all encrypted data to client then decrypt it - but next you need to enter the email of person you want to send, then entering both your email and password.



So you just need to enter the receiver’s email and sender’s email – your own email.

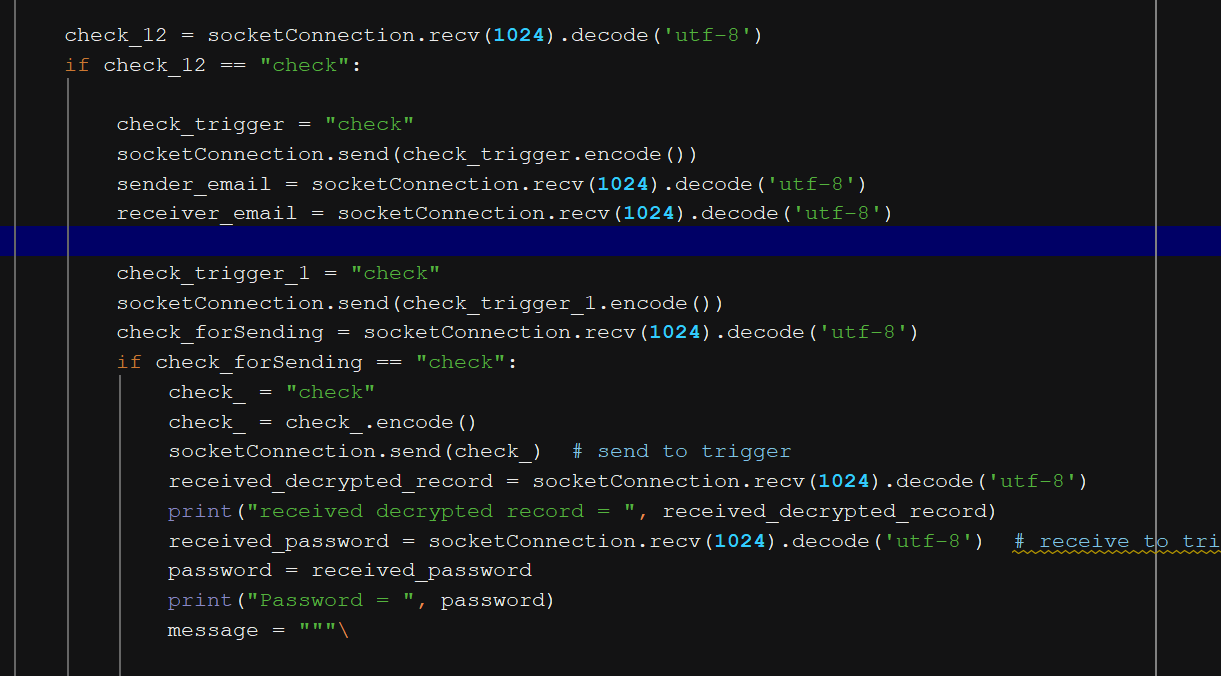


*<RatingApp\_Client.py>*

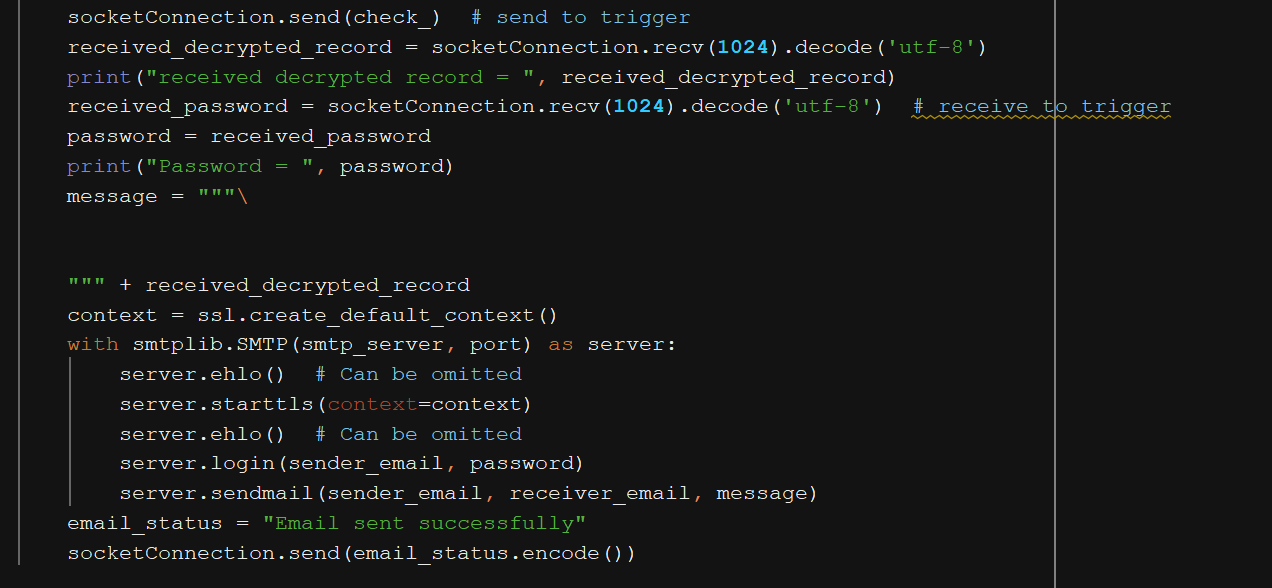


*<RatingApp\_Client.py>*

Then you enter your email password.

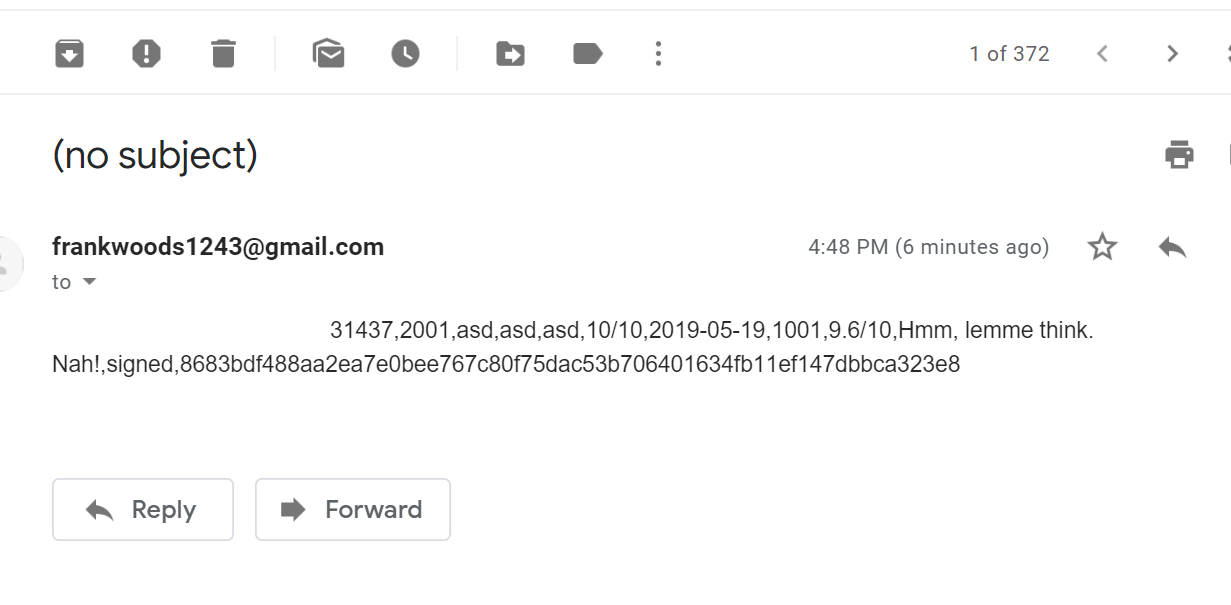


*<RatingApp\_Server.py>*



*<RatingApp\_Server.py>*

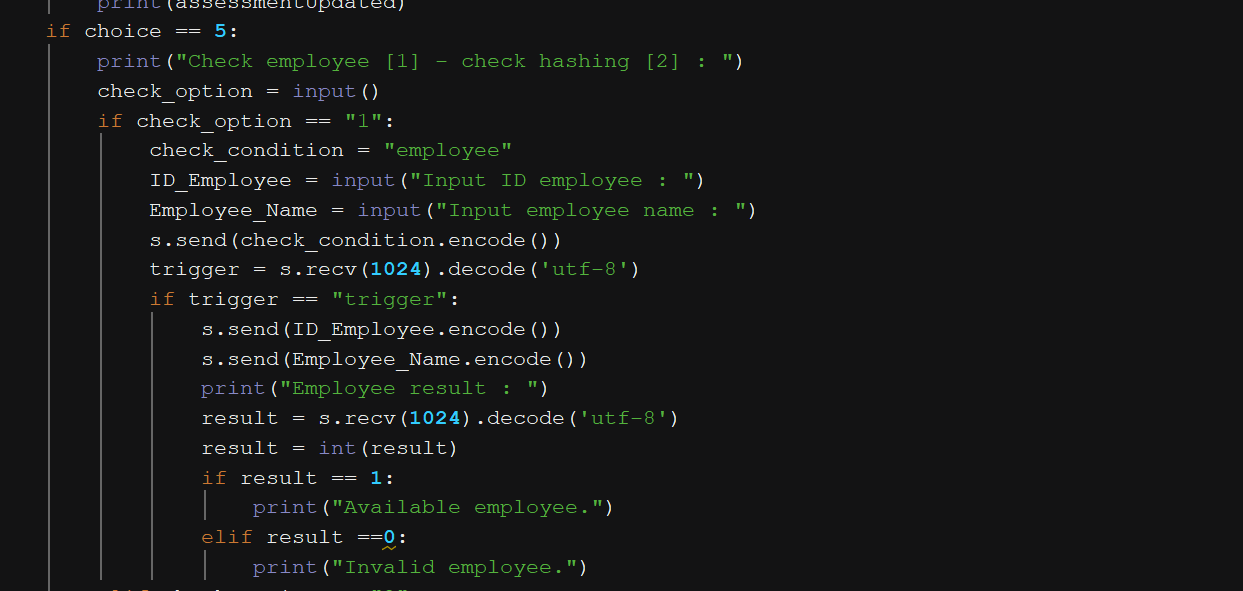
When the screen displays “Email sent successfully”, which means the record has been shared to that receiver. You can see the result in his email inbox.



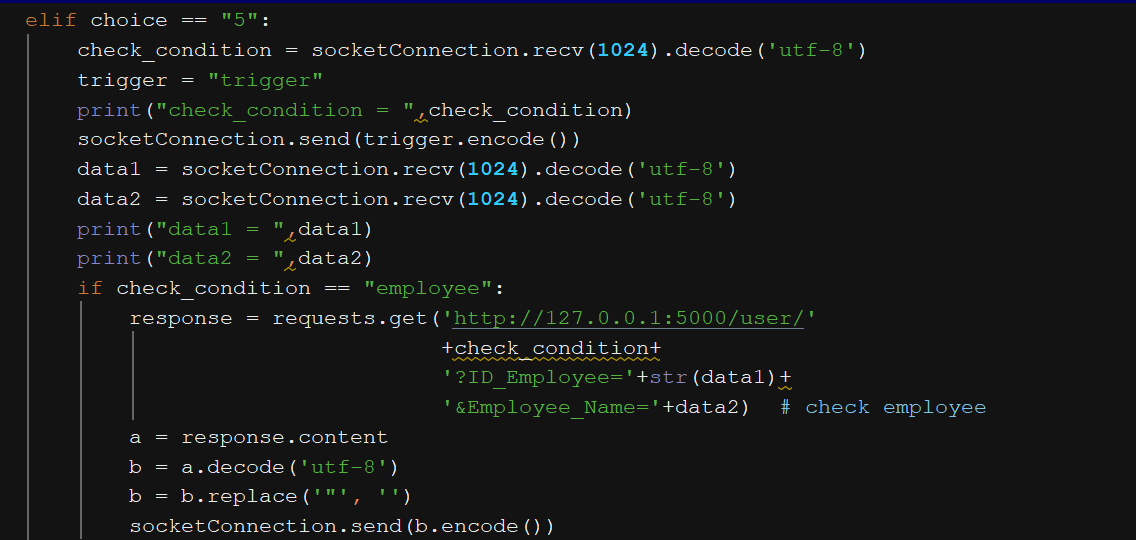
**7. Employee – hashing checking.**

In the option 5th, you can choose 1 or 2 for checking employee or hashing, each function will be initialized with a specific condition linked to the method held by APIs module.

If you enter 1, you need to fill data fields ID employee and employee name. These data is sent to APIs module with the condition “employee”

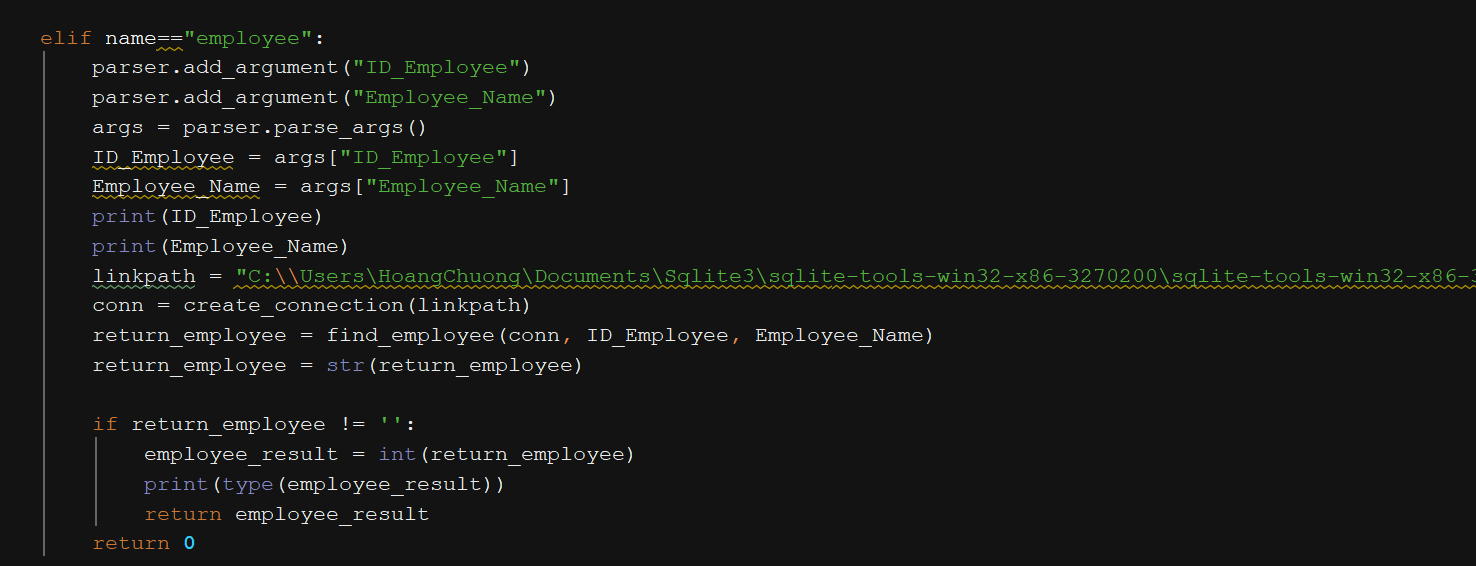


*<RatingApp\_Client.py>*

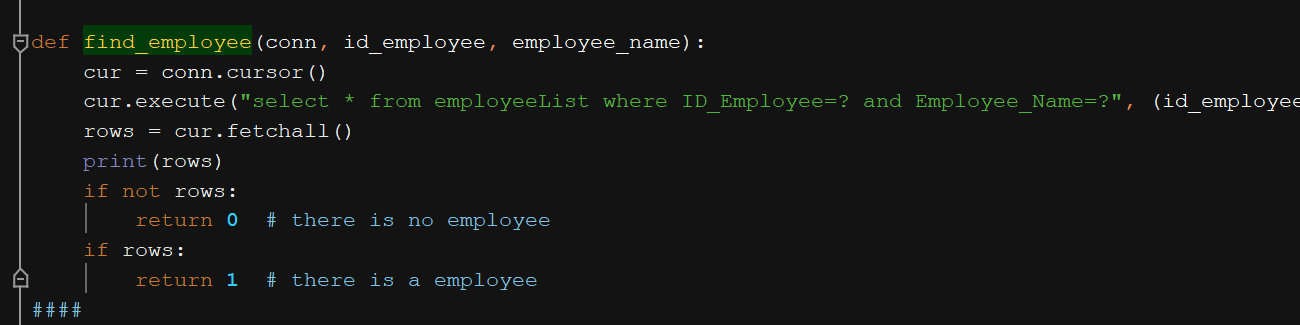


*<RatingApp\_Server.py>*

In the API modules, the condition “employee” is recognized, data is sent to the method of finding employee.

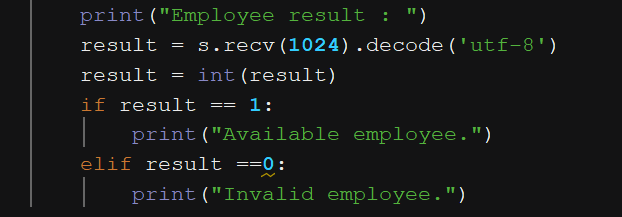


*<RatingApp\_Full\_APIs.py>*



*<RatingApp\_Full\_APIs.py>*

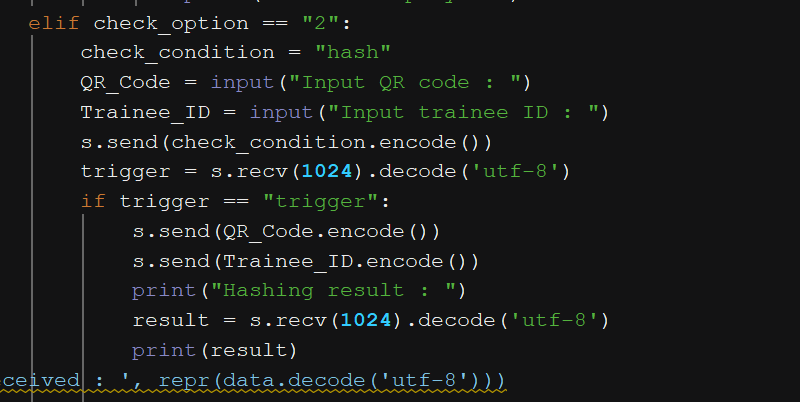
Based on the result, if it was 1, then the screen will display “Available employee”, or else “Invalid employee”.



*<RatingApp\_Client.py>*

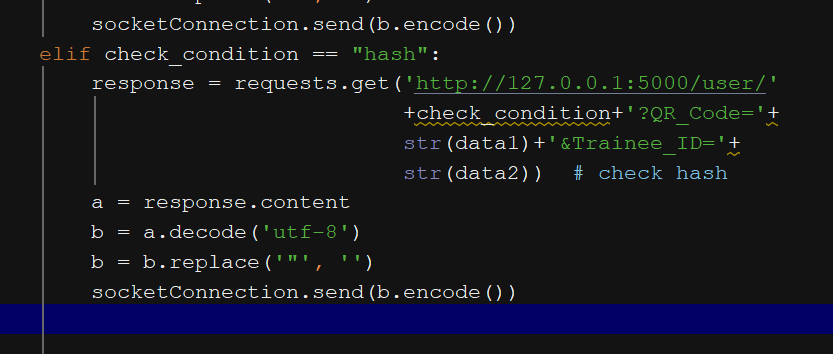


Same as employee checking, if the option was 2, user will be asked to fill in information as ID record and ID of trainee.



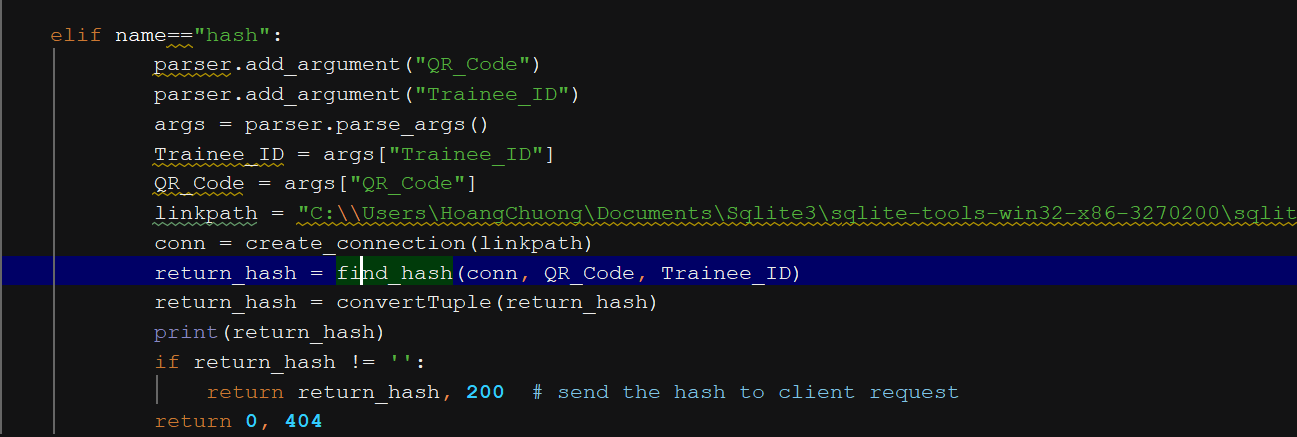
*<RatingApp\_Client.py>*

In server side, after received the data, condition “hash” and a specific URL are initialized, addressing to APIs module.

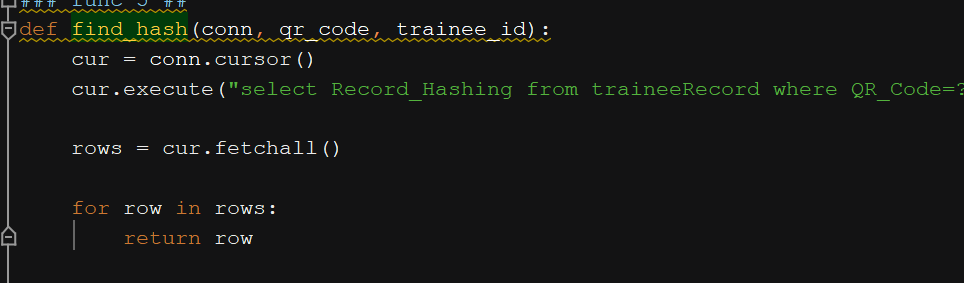


*<RatingApp\_Server.py>*

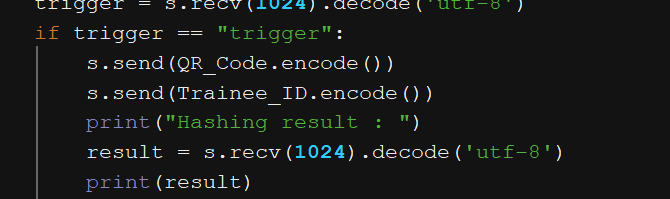
In APIs module, the condition “hash” is accepted, now method of hashing searching will be executed to return the hashing, through method find\_hash.



*<RatingApp\_Full\_APIs.py>*



*<RatingApp\_Full\_APIs.py>*



*<RatingApp\_Client.py>*

As the hashing result is returned, it will print to client interface.

