CSI3344 - DISTRIBUTED SYSTEMS

ASSIGNMENT 3: DISTRIBUTED SYSTEMS PROJECT AND VIDEO DEMONSTRATION

DISTRIBUTED SYSTEMS REPORT

Due Date: 05/02/2024

Student ID: 10470851

Student Name: Samarakoon Mudiyanselage Sanuri Dinusha

Executive Summary

The Open University of Science and Technology (*OUST*) has successfully developed the Honors Enrolment Pre-assessment System (*HEPaS*) to empower graduates and third-year students in self-evaluating their eligibility for Honors studies.

The primary goal of *HEPaS* is to facilitate a convenient and efficient self-assessment process for students aiming for Honors studies. This project responds to the necessity for a user-friendly application that simplifies the evaluation of eligibility criteria.

HEPaS has adopted a robust three-tier distributed system architecture, comprising a client-side application (*Client*), a server-side application (*Server-1*), and a database server (*Server-2*). This architecture ensures seamless data flow, user authentication, and efficient assessment processes through validation and verification.

The system encompasses essential functionalities, including user authentication and retrieval of academic records for OUST students, input of *Unit Scores* from non-OUST students, and the assessment of Honors studies eligibility criteria. The Client provides a user-friendly interface, allowing for user data input and result presentation. *Server-2* efficiently manages learning records in OUST students' course learning records (*OSCLR*) database, facilitating transient and accurate data retrieval for the assessment process to ensure efficiency of *HEPaS*.

Since *Client* directly communicates with *Server-1*, *Server-2* is relayed with requests by *Server-1*; therefore, remote servers *Server-1* and *Server-2* were communicated using *Synchronous* communication pattern and *RPC* mechanism through "*XMLRPC"*. *OSCLR database* was developed in *"SQL"* using *"MySQL Workbench"* and the other system components were developed in *"Python"* through *"VS Code"*.

Possible future enhancements include implementing a GUI for both OUST and non-OUST students and a database UI for database administrators and enhancing security and scalability of the system.

The outlined project requirements are satisfied, as the *Client* and *Servers* can run on different machines and communicate remotely to fulfil system requirements as well.

Table of Contents

1.0 Introduction

1.1 Background

The Open University of Science and Technology (*OUST*) is concerned about creating a simple Honors Enrolment Pre-assessment System (*HEPaS*) for graduates and third-year students to self-evaluate their eligibility for Honors studies. The system design is a three-tier distributed architecture, comprising a client-side application (*Client*), one or more server-side applications (*Server-1*), and a database server (*Server-2*) storing the learning records of OUST students.

1.2 Scope

The project entails designing and implementing the *HEPaS* to fulfil the specified requirements.

The *Client* should collect user data such as *Person ID* and *Unit Scores* with *Unit Codes* from non-OUST students, while *Peron ID* and *OUST Email* are only requested from OUST students to access student records from OUST students' course learning records (*OSCLR*) database. After performing preliminary processing by the *Client*, data is transmitted to remote server-side applications; *Server-1* and *Server-2*.

Server-1 is responsible for displaying each score in <*unit_code*, *mark*> pairs, calculating *Course Average* from all units and *Best 8 Average* from highest 8 scores of the student, assessing eligibility based on Honors evaluation criteria, and sending evaluation result back to *Client*. *Server-2* is responsible in retrieving student records from *OSCLR database*, when an OUST student has requested to perform Honors eligibility assessment through *Client* to *Server-1*, as *Client* can directly communicate with *Server-1* only. Then, *Server-1* communicates back the evaluation results to the *Client*. Finally, then the *Client* presents the assessment results to the user.

Furthermore, a *"MySQL"* database was built as *OSCLR database*, and connection was established between the database and *Server-2*. *Synchronous* communication pattern and "*XMLRPC"* built-in library was utilised to enable remote communication between *Client* and *Server-1*, and *Server-1* and *Server-2* through Remote Procedure Call (RPC).

1.3 Assumptions

The following **assumptions** were considered for the system implementation and simplicity.

- *Person ID* should be an 8-digit number.
- *OUST Email* address should end with "@our.oust.edu.au".
- OUST students are authenticated through their *Person ID* and *OUST Email* address.
- Non-OUST students can enter *Unit Scores* ranging from 16 to 30.
- Each *Unit Score* should be entered in pairs in <*unit_code*, *mark*> format (without "<" and ">").
- *Unit Code* has a maximum length of 7 and contains alphanumeric characters.
- *Mark* is a real number that falls within the inclusive range of 0 to 100.
- If non-OUST students have 3 scores for the same unit, it should be 2 *"Fail"* grades and 1 *"Pass"* grade.
- Users with 3 or more *"Fail"* grades are warned through the assessment.

2.0 Introduction to Remote Procedure Call (RPC)

RPC is a technology that enables communication between different remote machines of a distributed system. Such RPC applications typically have a two-tiered structure; client and server and may extend to three-tiered architecture as well, when a second server is implemented such as a database server to retrieve or store data.

Functionality of RPC

Figure 1 elaborates the RPC procedure. The following elements are required for the functionality of RPC.

Client: This process initiates RPC by making a standard call triggering a correlated process in *Client Stub* (Geeks for Geeks, 2022).

Client Stub: Used for semantic transparency (having a similarity in the meaning of a remote and local procedures) and called by the *Client* (Geeks for Geeks, 2022). Two tasks are performed by *Client Stub*:

1. When a *Client* request is received, the parameters and required specifications of the remote procedure are marshalled (packed) in a message (Geeks for Geeks, 2022).
2. Once the result values are received after execution, the results are unmarshalled (unpacked) and sent to the *Client* (Geeks for Geeks, 2022).

RPC Runtime: Responsible for message transmission between *Client* and *Server* through the network (Geeks for Geeks, 2022). It performs acknowledgment, retransmission, encryption, and routing (Geeks for Geeks, 2022). The client-side receives result values through a message from the server-side; then sent to the client stub (Geeks for Geeks, 2022). On the server-side, the *RPC Runtime* receives the same message from the *Server Stub* and forwards it to the *Client* (Geeks for Geeks, 2022). Additionally, call request messages are accepted from the *Client* and forwarded to the *Server Stub* (Geeks for Geeks, 2022).

Server Stub: Two tasks are performed by the *Server Stub*:

1. Call request message received from local *RPC Runtime* is unmarshalled and a regular call is made to invoke the necessary procedure in the *Server* (Geeks for Geeks, 2022).

2. When the procedure execution result of *Server* is received, it is marshalled into a message and the local *RPC Runtime* is requested to transmit it to *Client Stub* to be unmarshalled (Geeks for Geeks, 2022).

Server: A call request received from *Client* is passed to the *Server* through *Server Stub* (Geeks for Geeks, 2022). The necessary procedure is executed, and result is returned to *Server Stub* to proceed with passing to *Client* using local *RPC Runtime* (Geeks for Geeks, 2022).

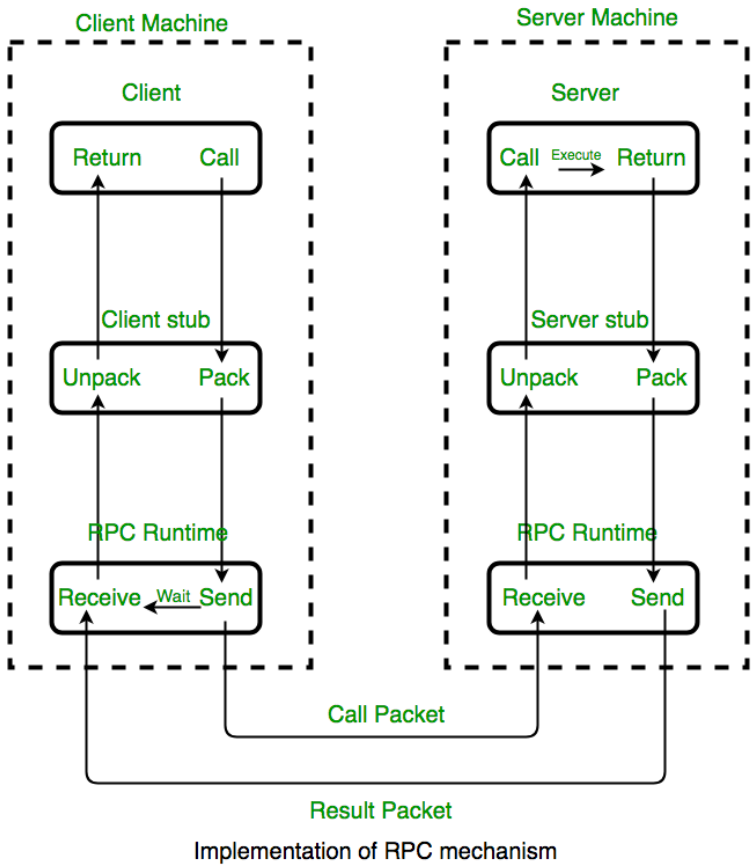

Figure 1: Implementation of RPC mechanism (Geeks for Geeks, 2023)

3.0 Application Requirements

The requirements of the application are as follows:

3.1. *Client* application should prompt the user to answer whether the user is an OUST student or not.

3.2. *Client* application should prompt the user to enter their *Person ID,* which must be an 8-digit number.

3.3. *Client* application should prompt an OUST student to input their *OUST Email* to verify the student status.

3.4. *Client* application should communicate the OUST student details; *Person ID* and *OUST Email* to *Server-1*, and *Server-1* should communicate them to *Server-2*, where *Server-2* authenticates the user as an OUST student by comparing entered details to the details stored in *OSCLR database* and sends *Unit Scores* of the student to *Server-1*.

3.5. *Client* application should prompt a non-OUST student to enter their *Unit Scores* in pairs of <*unit_code, mark*> and send them to *Server-1*.

3.6. *Server-1* should display the entered unit scores, calculate *Course Average* and *Best 8 Average* and then, evaluate eligibility for the Honors studies program using *Unit Scores* of the user (OUST student or non-OUST student) based on Honors evaluation criteria, and send the evaluation result back to the *Client*, which displays the evaluation result to the user.

4.0 Application Design and Implementation Procedure

4.1 The HEPaS System Architecture

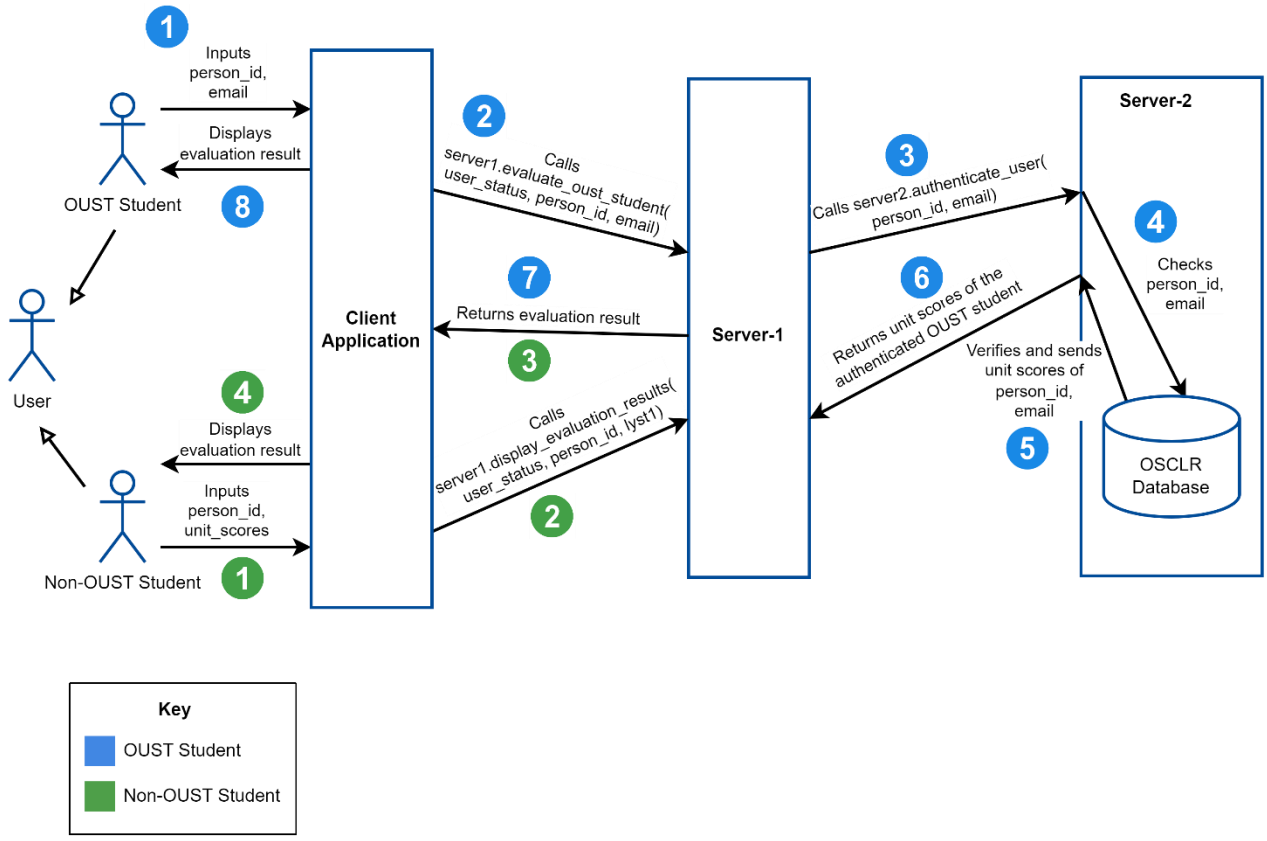

Figure 2: HEPaS architecture and interactions between components and users

As mentioned before, *HEPaS* has two main users; OUST students and non-OUST students. Figure has been created to elaborate the interactions and communication between users and the *Client*, *Client* and *Server-1*, and *Server-1* and *Server-2*, where blue colour numbers indicate the evaluation process for OUST student users and green colour numbers indicate the evaluation process for non-OUST student users.

4.2 The HEPaS System Implementation Procedure

4.2.1. Firstly, the *Client* program was created to input *Unit Scores* in pairs of <*unit_code*, *mark*>, and validate the permitted number of units (minimum 16 and maximum 30 units) based on the entered *Unit Scores*.

4.2.2. Then, the *Server-1* program was built employing *Synchronous* communication pattern to input these *Unit Scores* and separate them into validated *Unit Codes* and *Marks* and validate the counts of "Fail" and "Pass" grades for repeating units, then calculate *Course Average* from all *Unit Scores* and return the average to *Client*.

4.2.3. Then, built-in *"XMLRPC"* library was used to establish communication with *RPC* between *Client* and *Server-1* after registering functions of *Server-1* at port 8000. *Server-1* program contains the *Client* request processing methods such as calculating average and displaying evaluation result and *Unit Scores*; therefore, a proxy for *Server-1* was created in *Client* program.

4.2.4. After that, the Honors studies eligibility criteria were implemented in *Server-1*; therefore, eligibility assessment result could be returned along to *Client*, while entered *Unit Scores* are displayed on *Server-1*.

4.2.5. After testing this two-tiered system implementation, added a prompt to confirm whether the user is an OUST student or not. Then added another prompt for the user to enter *Person ID*. If the user is an OUST student, an additional prompt is presented to enter their *OUST Email*. Both *Person ID* and *Email* were implemented with validation methods; validate whether the *Person ID* is an 8-digit number and validate whether the *Email* is an *OUST Email* by confirming if it ends with *"@our.oust.edu.au"*.

4.2.6. After further testing and validation procedures, *OSCLR database* was built using *"MySQL Workbench 8.0"*. Separate tables that store OUST student information, course information, and *Unit Scores* of OUST students were created. The inserted data were extracted from modified data in *"SampleData sets.xlsx"* provided along with the assignment document.

4.2.7. Then, *Server-2* program was developed using *Synchronous* communication pattern. It contains the *OSCLR database* connection, query requests, and OUST Student authentication through *Person ID* and *OUST Email,* and returns *Unit Scores* (*Unit Codes* and *Marks*). A proxy for *Server-2* was created in *Server-1* program for authentication and returning *Unit Scores* of a specific OUST Student. This *RPC*-based communication between *Server-1* and *Server-2* was established using *"XMLRPC"* library as well by registering functions of *Server-2* at port 8001.

4.2.8. Finally, the entire three-tiered distributed system was tested for both user types; OUST students and non-OUST students. An executable file was created to ensure seamless execution of the application by clicking it.

4.3 Languages, Libraries, and Tools

Python

"Python" is the main programming language that was used to conveniently develop *Client*, *Server-1*, and *Server-2* programs by applying *Synchronous* communication pattern.

Structured Query Language (SQL)

"SQL" is the query language used to create the *OSCLR database* and its tables, and extract data from the *OSCLR database*, when authenticating OUST students and fetching unit scores to proceed with Honor studies eligibility assessment.

XMLRPC Library

"XMLRPC" is an in-built library that was utilised to establish *RPC*-based communication between *Client* and remote server *Server-1*, and remote severs *Server-1* and *Server-2*; since *Client* can directly communicate with *Server-1* only.

MySQL Workbench 8.0

"MySQL Workbench" was employed in creating the *OSCLR database* and its tables *"student_info"*, *"course_info"*, and *"student_unit"* to store OUST student information, course information, and unit scores of OUST students respectively. The connection from *Server-2* Interface to the database was established by importing *"mysql.connector"* library.

Visual Studio Code (VS Code)

VS Code is the source code editor used to write and test codes during the development period of *HEPaS* system. It supports an extensive range of languages including *"Python"* and *"SQL"*.

5.0 User Manual

5.1 System Installation Requirements

The system is required to have installed the following tools and software to run this *HEPaS* application.

- At least *"Python"* version 3.11 (Python, 2023).
- *"MySQL version 8.0.36"* (MySQL Community Downloads, n.d.)
- *"MySQL Workbench 8.0"* (MySQL Community Downloads, n.d.); once the *"MySQL Installer"* is installed, *"MySQL Workbench"* can be installed as well.
- *"MySQL Connector"* for Python (MySQL, n.d.)
    - Execute this command on *"Command Prompt"* to install *"MySQL Connector"* for *"Python"*:

      pip install mysql-connector-python

5.2 HEPaS Application Setup

To setup the application, follow the below steps:

1. Open the *"server2.py"* file, which is in *"HEPaS_v6"* folder in a code editor as shown in Figure 3.

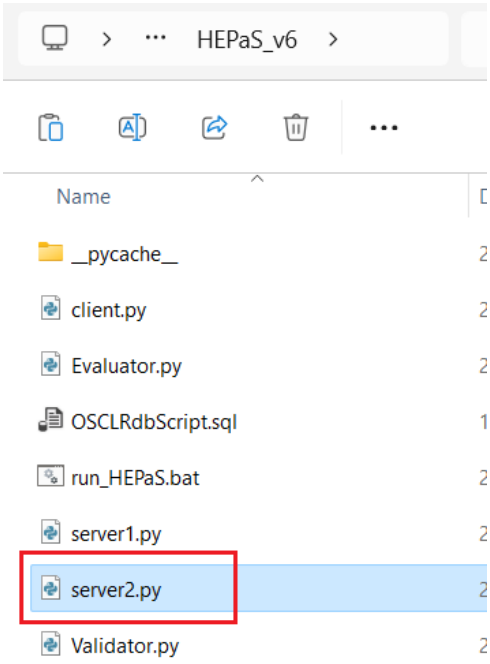

Figure 3: Locating *"Server2_Interface.py"* file inside *"HEPaS_v2"* folder

2. As indicated in Figure 4, locate the *"password"* parameter of the *"sql_connector"* variable and change the value of the *"password"* to the root password of the *"MySQL Server"* and save the *"server2.py"* file.

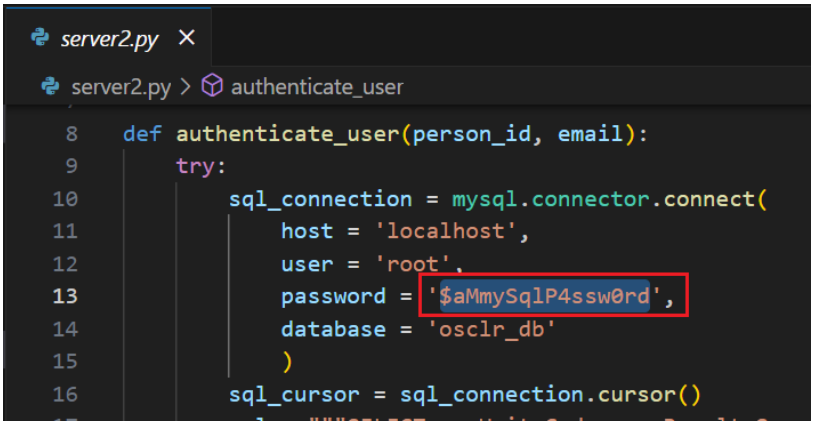

Figure 4: Locating *"password"* parameter's value in *"server2.py"* file

3. Open *"MySQL Workbench"* and open *"OSCLRdbScript.sql"* through it and run the script to create *OSCLR database* and its tables *"course_info"*, *"student_info"*, and *"student_unit"* and populate them with data as shown in Figure 5.

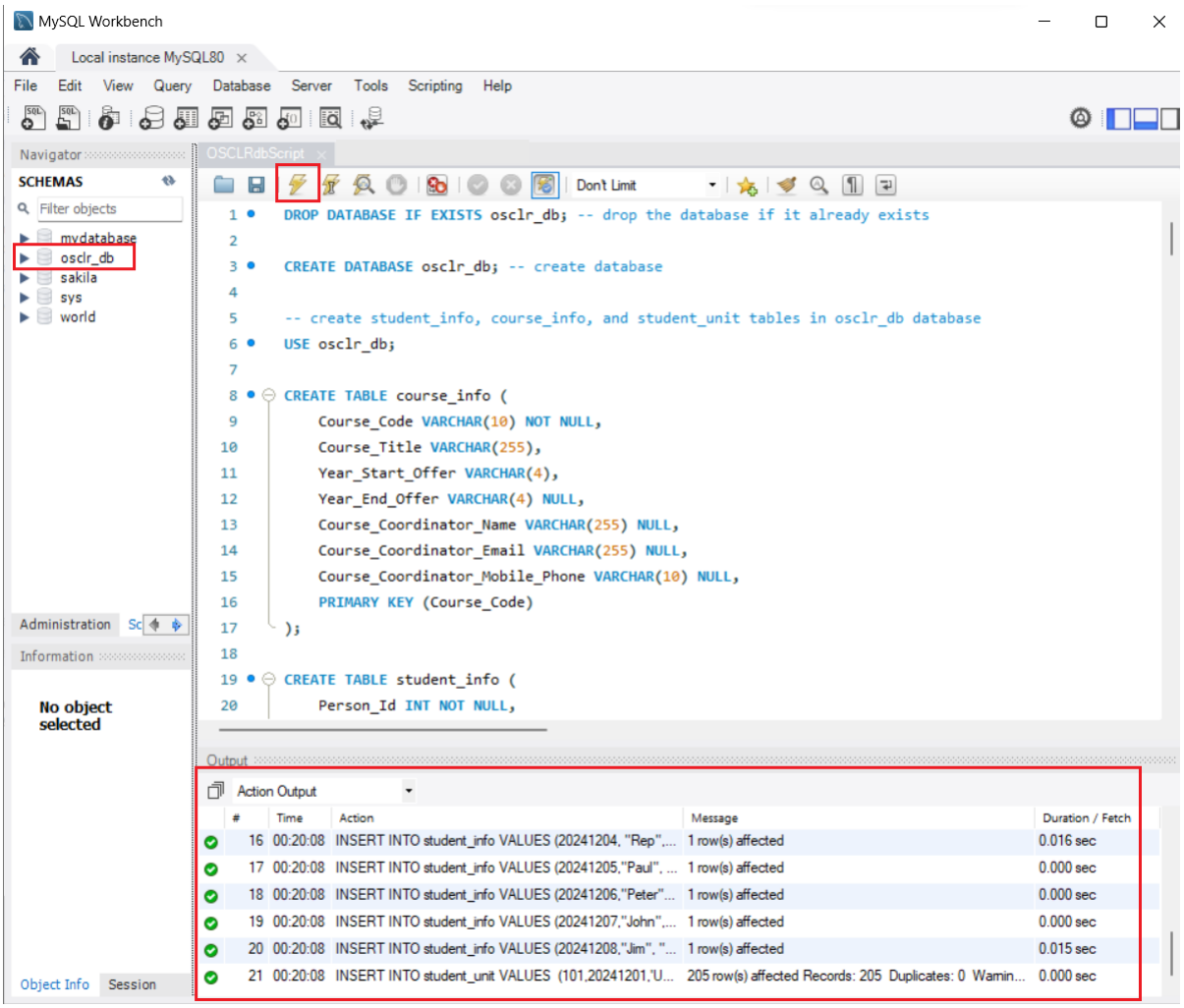

Figure 5: *OSCLR database* creation and data population

4. Now the setup is completed.

5.3 HEPaS Application Usage

5.3.1 Run Client and Servers in the Same Machine

This option is to run *Client*, *Server-1*, and *Server-2* in three separate *"Command Prompt"* windows to mimic the implementation of remote servers and client.

1. Open *"HEPaS_v6"* folder in one machine, which is shown in Figure 6.

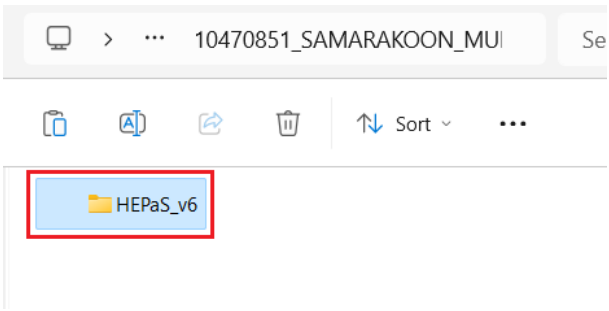

Figure 6: *"HEPaS_v6"* folder

2. Find *"run_HEPaS.cmd"* in *"HEPaS_v6"* folder and double click on it to run the *HEPaS* application, which is indicated in Figure 7.

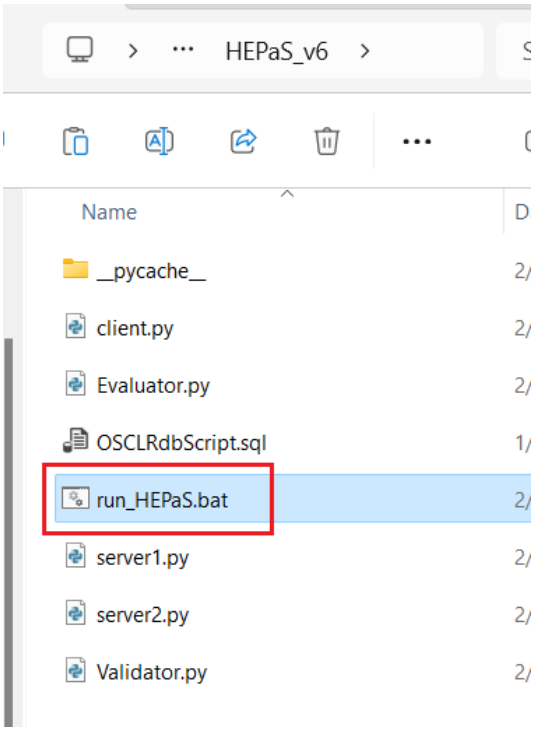

Figure 7: *"run_HEPaS.cmd"* in *"HEPaS_v6"* folder

3. It will show 3 separate *"Command Prompts"* being opened to run *Client*, *Server-1*, and *Server-2* programs separately as in different remote machines. Figure 8 shows the opened Command Prompt windows; in the order of *Client*, *Server-1*, and *Server-2*.

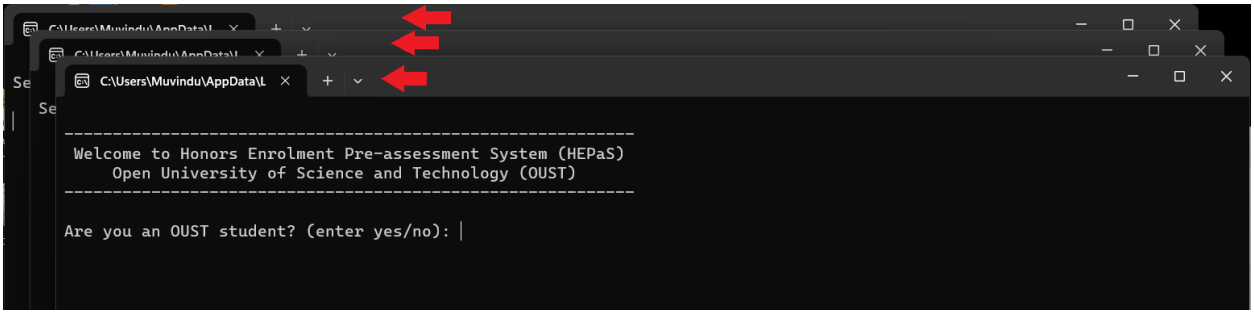

Figure 8: Opened three windows of *"Command Prompt"*

4. Input data to the questions asked in *Client* prompt as shown in Figure 9. This is *TV01* test case.

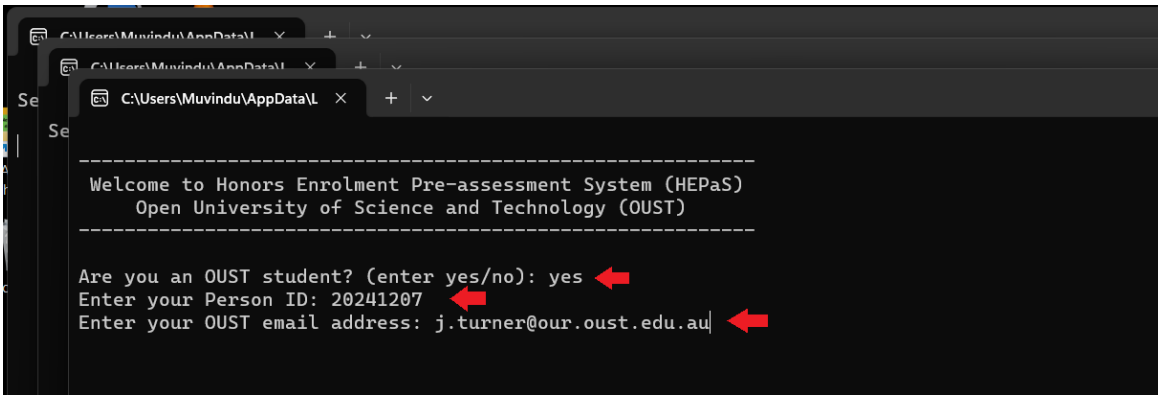

Figure 9: Input data to the *Client* prompt

5. Press *"Enter"* key. The *Client* window will display evaluation result as in Figure 10. *Server-1* window, which is underneath it will display the *Unit Scores* and displays the *Server-2* window after executing query to fetch OUST student data from *OSCLR database*.as shown in Figure 11.

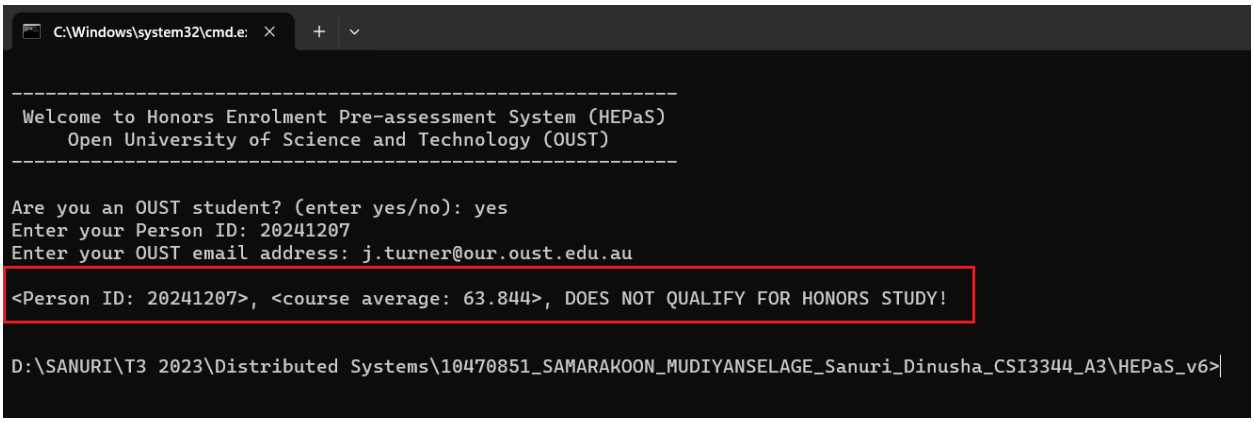

Figure 10: Honors evaluation result displayed at *Client*

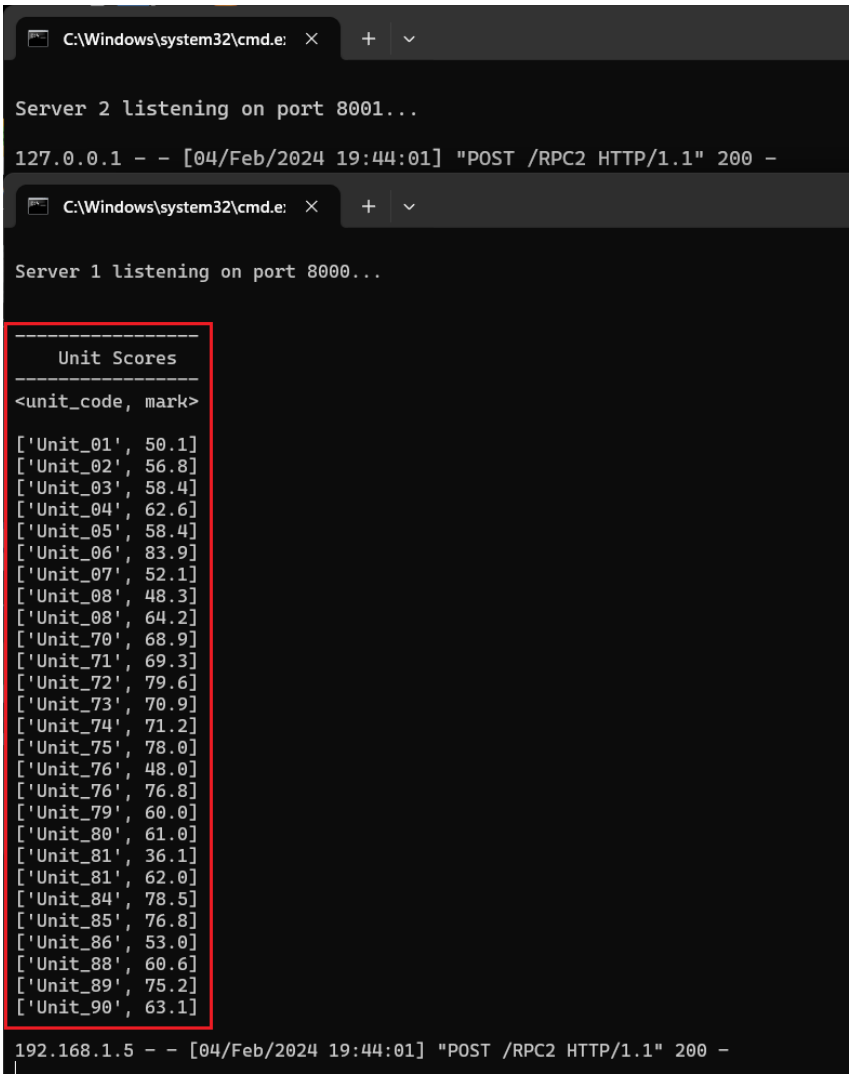

Figure 11: *Server-1* window with *Unit Scores* and *Server-2* window after fetching OUST student data from *OSCLR database*

5.3.2 Run Client and Servers in Different Machines

1. Open *"HEPaS_v6"* folder (shown in Figure 6) through VS Code in two different machines as in Figures 12 and 13.

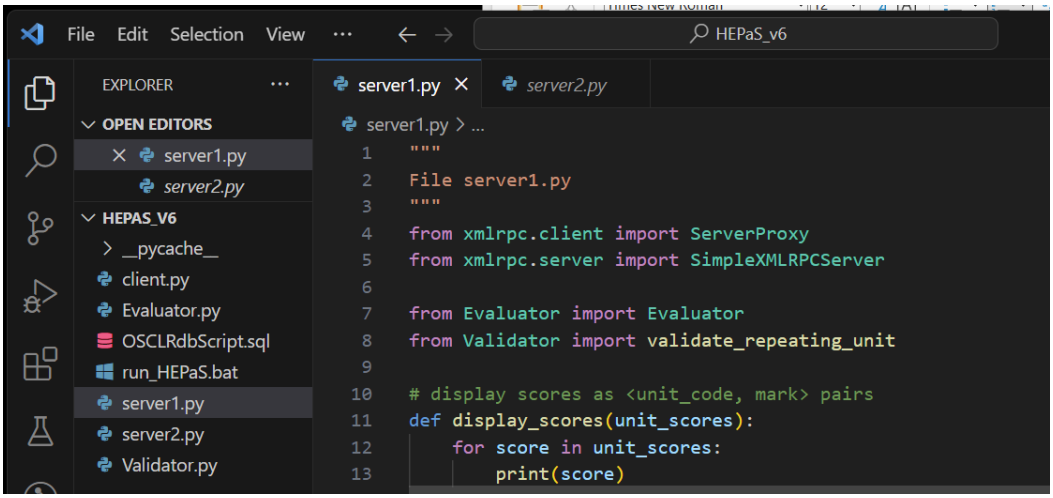

Figure 12: *"HEPaS_v6"* folder opened through VS Code on a Windows OS machine

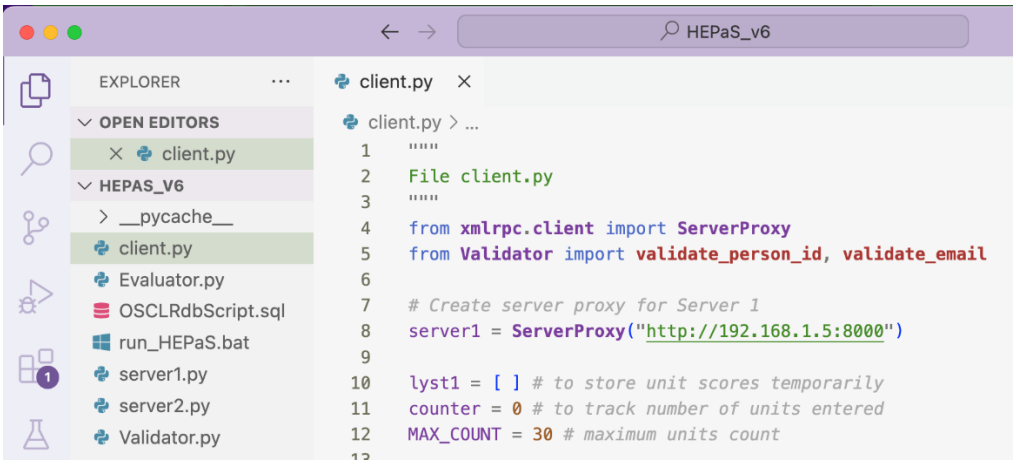

Figure 13: *"HEPaS_v6"* folder opened through VS Code on a MacOS machine

2. Choose one machine to run both servers and the other machine to run client. Obtain the IP address of the machine that will run the servers, according to the connected network. Figure 14 shows running *"ipconfig"* as the command on *Powershell"* of the Windows machine.

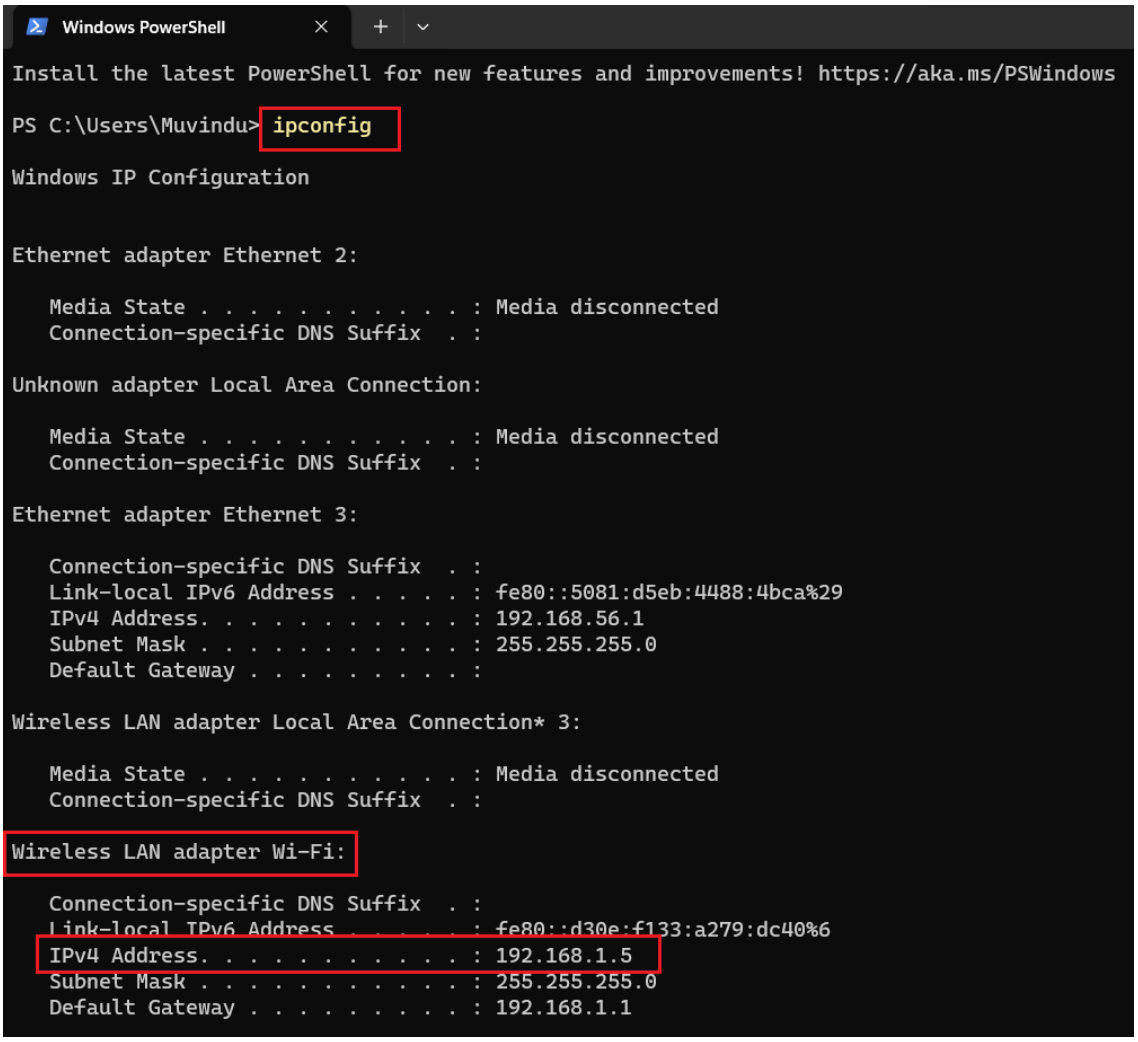

Figure 14: Windows machine selected as server machine and obtaining IP address

3. Change the IP address in line 8 of *"client.py"* file in the chosen *Client* machine into the IP address of the machine that runs servers as shown in Figure 15.

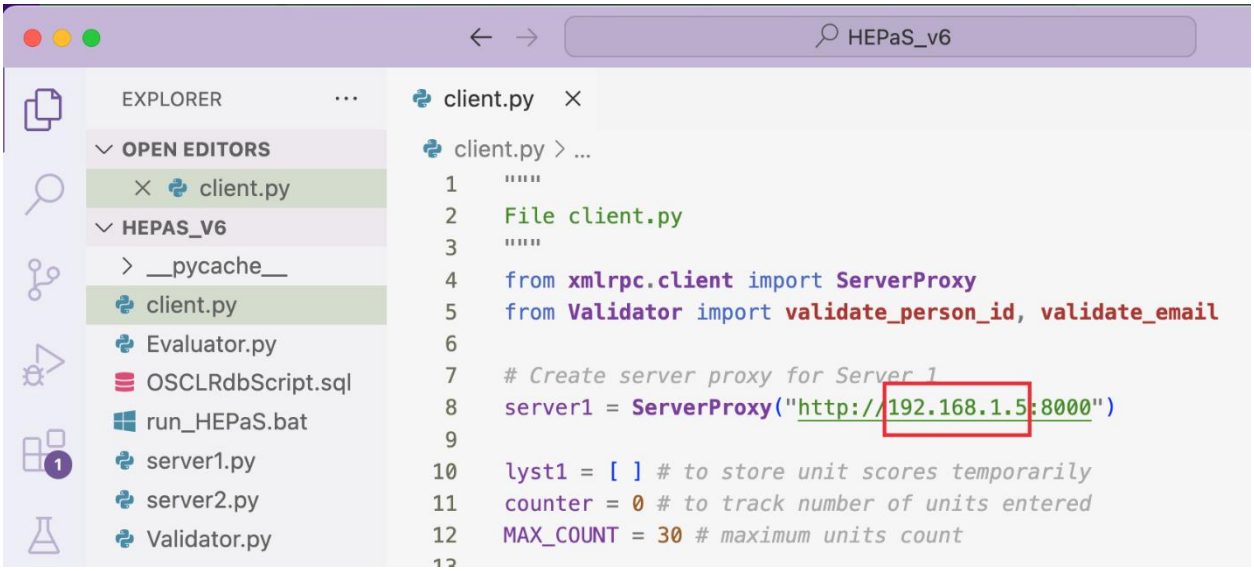

Figure 15: Setting IP address in *Client* program to Windows machine's IP address

4. Change the IP address in line 55 of *"server1.py"* into the same IP address (IP of the machine that runs servers) set in *"client.py"* as shown in Figure 16. *Server-2* may run on *localhost (127.0.0.1)* as specified in the code.

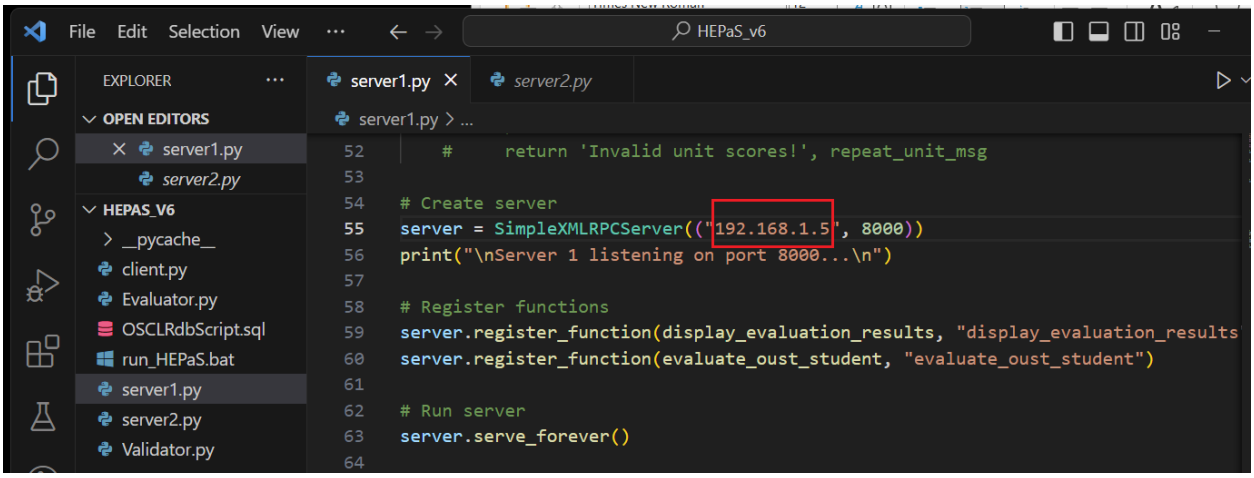

Figure 16: Setting IP address of *Server-1* program into the same IP address of the machine that runs servers

5. Now, open two split terminal on VS Code in server machine and run *Server-1* (*"server1.py"*) using *"python server1.py"* and *Server-2* (*"server2.py"*) using *"python server2.py"* on two different *Terminals* in VS Code as indicated in Figure 17. Open another *Terminal* on VS Code on *Client* machine and run *Client* (*"client.py"*) using *"python client.py"* if the *Client* is also a

Windows machine. If the *Client* is a Mac machine, run *"python3 client.py"* instead as in Figure 18. Figure 19 indicates discovering IP address (192.168.1.4) of *Client* to show that IP address shown on POST request on *Server-1* in Figure 17 is the same as in Figure 19.

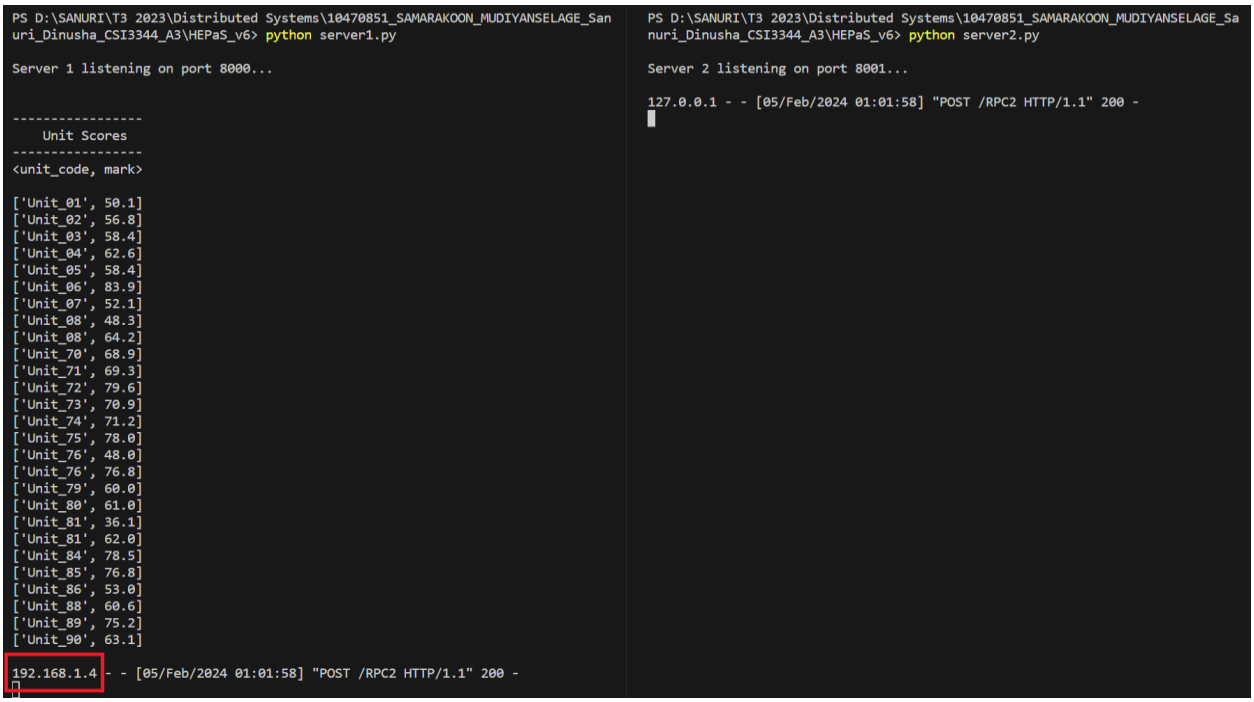

Figure 17: Both servers running on the same Windows machine via VS Code but in two *Terminals* and two different IPs; *Client* (192.168.1.4) has requested data for an OUST student, so both servers have successfully responded with status code 200

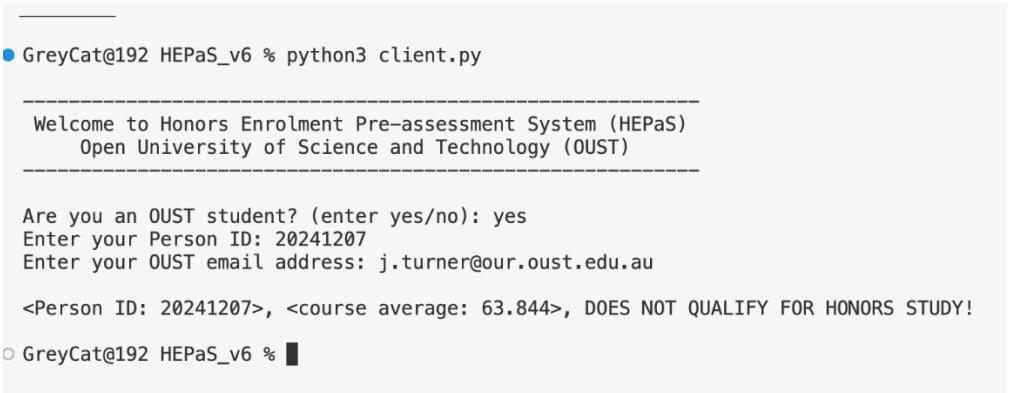

Figure 18: *Client* running on a Mac machine via VS Code *Terminal* and displaying evaluation result for an OUST student

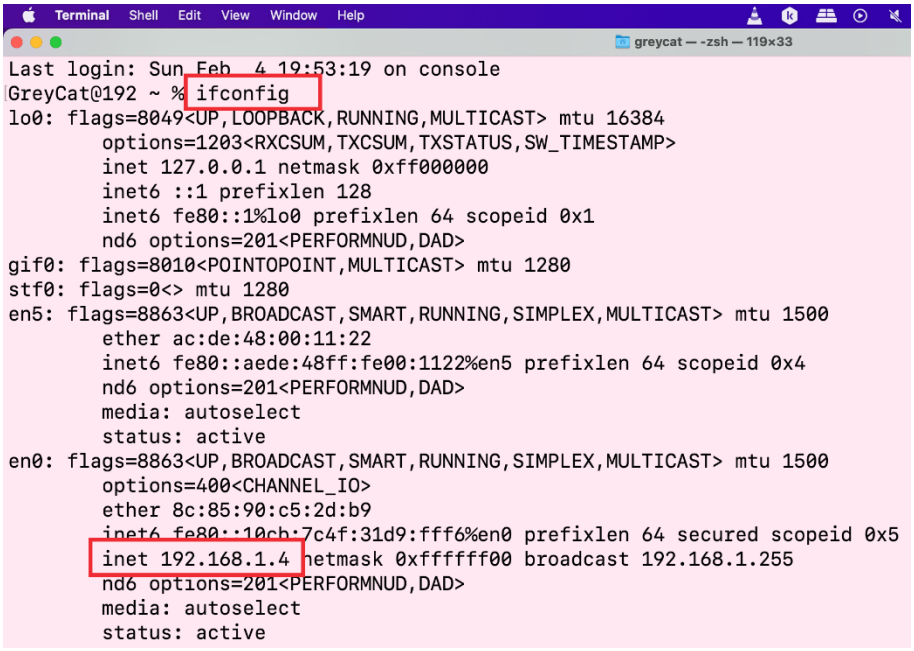

Figure 19: Discovering IP address of *Client* using command *"ifconfig"* on Mac *Terminal*

6.0 Test Cases

6.1 Test Cases with Valid Inputs

Test ID	Test Case Description	Input	Expected Client Output	Actual Client Output
TV01	An OUST student enters valid *Person ID* and *OUST Email* and receives evaluation result (based on *Course Average* < 60 and *Best 8 Average* < 80) as not qualifying for Honors studies	user_status: yes person_id: 20241207 email: j.turner@our.oust.edu.au	<Person ID: 20241207>, <course average: 63.844>, DOES NOT QUALIFY FOR HONORS STUDY!	<Person ID: 20241207>, <course average: 63.844>, DOES NOT QUALIFY FOR HONORS STUDY!
TV02	An OUST student enters valid *Person ID* and *OUST Email* and receives evaluation result (based on *Course Average* >= 65 and *Best 8 Average* >= 80) as qualified for Honors studies	user_status: yes person_id: 20241203 email: l.max@our.oust.edu.au	<Person ID: 20241203>, <course average: 66.429>, <best 8 average: 84.362>, QUALIFIES FOR HONORS STUDY!	<Person ID: 20241203>, <course average: 66.429>, <best 8 average: 84.362>, QUALIFIES FOR HONORS STUDY!
TV03	An OUST student enters valid *Person ID* and *OUST Email* and receives evaluation result	user_status: yes person_id: 20241201 email: j.max@our.oust.edu.au	<Person ID: 20241201>, <course average: 70.353>, with	<Person ID: 20241201>, <course average: 70.353>, with

	(Fails count >= 6) as not qualifying for Honors studies		6 or more Fails! DOES NOT QUALIFY FOR HONORS STUDY!	6 or more Fails! DOES NOT QUALIFY FOR HONORS STUDY!
TV04	An OUST student enters valid *Person ID* and *OUST Email* and receives evaluation result (based on *Course Average* >= 70) as qualified for Honors studies	user_status: yes person_id: 20241202 email: jt.max@our.oust.edu.au	<Person ID: 20241202>, <course average: 72.863>, QUALIFIES FOR HONOURS STUDY!	<Person ID: 20241202>, <course average: 72.863>, QUALIFIES FOR HONOURS STUDY!
TV05	An OUST student enters valid *Person ID* and *OUST Email* and receives evaluation result (based on *Course Average* >= 60 and *Best 8 Average* >= 80) as may be chosen for Honors studies (with coordinator's permission)	user_status: yes person_id: 20241205 email: p.landan@our.oust.edu.au	<Person ID: 20241205>, <course average: 63.789>, <best 8 average: 82.037>, MAY HAVE A CHANCE! Must be carefully reassessed and get the coordinator's permission!	<Person ID: 20241205>, <course average: 63.789>, <best 8 average: 82.037>, MAY HAVE A CHANCE! Must be carefully reassessed and get the coordinator's permission!

TV06	A non-OUST student enters valid *Person ID* and 16 *Unit Scores* in <*unit_code*, *mark*> pairs with 2 scores for only 1 unit (1 *"Fail"* and 1 *"Pass"*), receives evaluation result (based on *Course Average* >= 65 and *Best 8 Average* < 80) as may be chosen for Honors studies	user_status: no person_id: 12345678 unit_scores: Unit1,60 Unit2,65 Unit3,45 Unit4,50 Unit3,55 Unit5,70 Unit6,70 Unit7,68.5 Unit8,69.9 Unit9,73 Unit10,75 Unit11,71 Unit12,78 Unit13,80 Unit14,67 Unit15,82.5 -1	<Person ID: 12345678>, <course average: 67.494>, <best 8 average: 74.938>, MAY HAVE A GOOD CHANCE! Need further assessment!	<Person ID: 12345678>, <course average: 67.494>, <best 8 average: 74.938>, MAY HAVE A GOOD CHANCE! Need further assessment!
TV07	A non-OUST student enters valid *Person ID* and 6 *Unit Scores* in <*unit_code*, *mark*> pairs with 2 scores for only 1 unit (2 *"Fail"*s and 1 *"Pass"*), receives evaluation result (minimum unit count < 16) as not	user_status: no person_id: 23456789 unit_scores: Unit1,60 Unit2,65 Unit3,45 Unit3,47 Unit3,55 Unit4,50 -1	<Person ID: 23456789>, <course average: 53.667>, completed less than 16 units! DOES NOT QUALIFY FOR HONORS STUDY!	<Person ID: 23456789>, <course average: 53.667>, completed less than 16 units! DOES NOT QUALIFY FOR HONORS STUDY!

	qualifying for Honors studies			

Table 1: Test cases for valid user inputs

6.2 Test Cases with Invalid Inputs

Test ID	Test Case Description	Input	Expected Client Output	Actual Client Output
TI01	An OUST student enters invalid *Person ID* (incorrect length; less than 8) and receives error message and prompt to re-enter a valid *Person ID*	user_status: yes person_id: 1111222	Person ID should be an 8-digit number! Enter your Person ID:	Person ID should be an 8-digit number! Enter your Person ID:
TI02	An OUST student enters invalid *Person ID* (incorrect length; greater than 8) and receives error message and prompt to re-enter a valid *Person ID*	user_status: yes person_id: 111122233	Person ID should be an 8-digit number! Enter your Person ID:	Person ID should be an 8-digit number! Enter your Person ID:
TI03	An OUST student enters valid *Person ID* (correct length; 8, exists in database) and	user_status: yes person_id: 20241203 email: a@a.a	This is not an OUST email address! Enter your OUST email address:	This is not an OUST email address! Enter your OUST email address:

	invalid *Email* (incorrect domain name) and receives error message and prompt to re-enter a valid *OUST Email*		Enter your OUST email address:	
TI04	An OUST student enters invalid *Person ID* (correct length; 8, not in database) and invalid *OUST Email* (with correct domain name; our.oust.edu.au, exists in database)	user_status: yes person_id: 11112222 email: j.max@our.oust.edu.au	Invalid Person ID or OUST Email!	Invalid Person ID or OUST Email!
TI05	An OUST student enters valid *Person ID* (correct length; 8, exists in database) and invalid *OUST Email* (with correct domain name; our.oust.edu.au, not in database).	user_status: yes person_id: 20241202 email: max@our.oust.edu.au	Invalid Person ID or OUST Email!	Invalid Person ID or OUST Email!
TI06	An OUST student enters invalid *Person ID* (correct length; 8, not a number, not in	user_status: yes person_id: 2222aaaa	Person ID should be an 8-digit number!	Person ID should be an 8-digit number! Enter your Person ID:

	database) and receives error message and prompt to re-enter a valid *Person ID*		Enter your Person ID:	
TI07	A non-OUST student enters valid *Person ID* and 16 *Unit Scores* in <*unit_code, mark*> pairs with 4 scores for only 1 unit (3 *"Fail"*s and 1 *"Pass"*), receives evaluation result (based on *Course Average* >= 60) as not qualifying for Honors studies	user_status: no person_id: 12345678 unit_scores: Unit1,60 Unit2,65 Unit3,35 Unit3,45 Unit3,45 Unit3,55 Unit4,50 Unit5,70 Unit6,70 Unit7,68 Unit8,60 Unit9,73 Unit10,75 Unit11,71 Unit12,78 Unit13,80 -1	The same unit can have up to 3 scores ONLY! <Person ID: 12345678>, <course average: 62.5>, DOES NOT QUALIFY FOR HONORS STUDY!	The same unit can have up to 3 scores ONLY! <Person ID: 12345678>, <course average: 62.5>, DOES NOT QUALIFY FOR HONORS STUDY!
TI08	A non-OUST student enters valid *Person ID* and attempts to enter more than 30 *Unit Scores* in <*unit_code, mark*>	user_status: no person_id: 12345678 unit_scores: Unit1,60 Unit2,60 Unit3,60 Unit4,60 Unit5,70	Cannot enter more than 30 unit scores! <Person ID: 12345678>, <course average:	Cannot enter more than 30 unit scores! <Person ID: 12345678>, <course average: 68.0>,

	pairs, but automatically exit from entering *Unit Scores* after the 30th and receives evaluation result (based on *Course Average* >= 65 and *Best 8 Average* >= 80) as qualified for Honors studies	Unit6,70 Unit7,70 Unit8,70 Unit9,70 Unit10,70 Unit11,70 Unit12,70 Unit13,80 Unit14,80 Unit15,80 Unit16,80 Unit17,60 Unit18,60 Unit19,60 Unit20,60 Unit21,80 Unit22,80 Unit23,60 Unit24,60 Unit25,60 Unit26,60 Unit27,80 Unit28,80 Unit29,60 Unit30,60	68.0>, <best 8 average: 80.0>, QUALIFIES FOR HONOURS STUDY!	<best 8 average: 80.0>, QUALIFIES FOR HONOURS STUDY!
TI09	A non-OUST student enters valid *Person ID* and 7 *Unit Scores* in <*unit_code*, *mark*> pairs with 3 scores	user_status: no person_id: 12345678 unit_scores: Unit1,60 Unit2,60 Unit3,30 Unit3,40	A unit has more than 2 Fail grades! <Person ID: 12345678>, <course average:	A unit has more than 2 Fail grades! <Person ID: 12345678>, <course average: 49.375>,

	for only 1 unit (3 *"Fail"*s), receives evaluation result (minimum unit count < 16) as not qualifying for Honors studies	Unit3,45 Unit4,50 Unit5,50 Unit6,60 -1	49.375>, completed less than 16 units! DOES NOT QUALIFY FOR HONORS STUDY!	completed less than 16 units! DOES NOT QUALIFY FOR HONORS STUDY!
TI10	A non-OUST student enters valid *Person ID* and invalid *Unit Scores*	user_status: no person_id: 12345678 unit_scores: uu dd,60 -1	Invalid unit scores!	Invalid unit scores!
TI11	A non-OUST student enters valid *Person ID* and does not enter *Unit Scores* (enters -1 to stop)	user_status: no person_id: 12345678 unit_scores: -1	No unit scores found!	No unit scores found!

Table 2: Test cases for invalid user inputs

7.0 Results

The results shown below are only based on the valid inputs from example main two users of the system.

7.1 OUST Student Honors Eligibility Evaluation

Test case *TV02* has been used as an example for OUST student Honors eligibility evaluation.

1. OUST student entering details; *Person ID* and *OUST Email* to *HEPaS* application as in Figure 12.

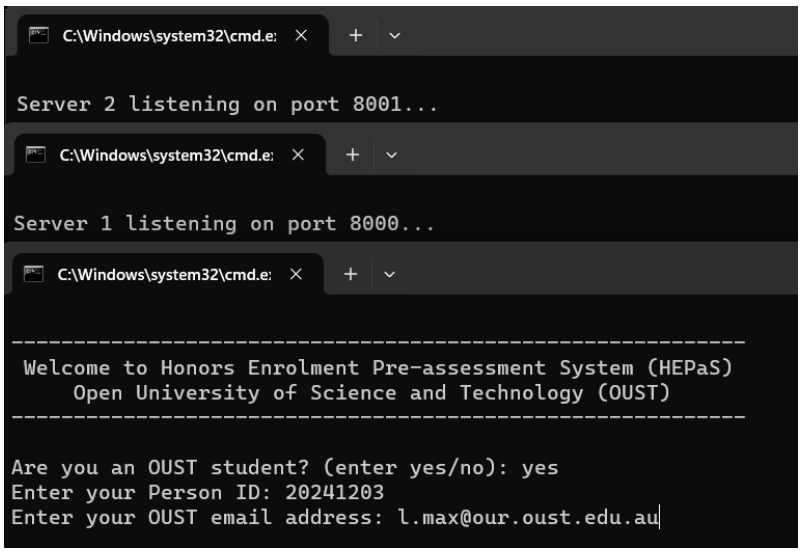

Figure 12: Entered data of an existing OUST student in *OSCLR database*

2. Displaying Honors assessment result on *Client* and *Unit Scores* on *Server*-1 for the above entered student details (Figure 12) is shown in Figure 13. *Server-2* displays a successful request with code 200.

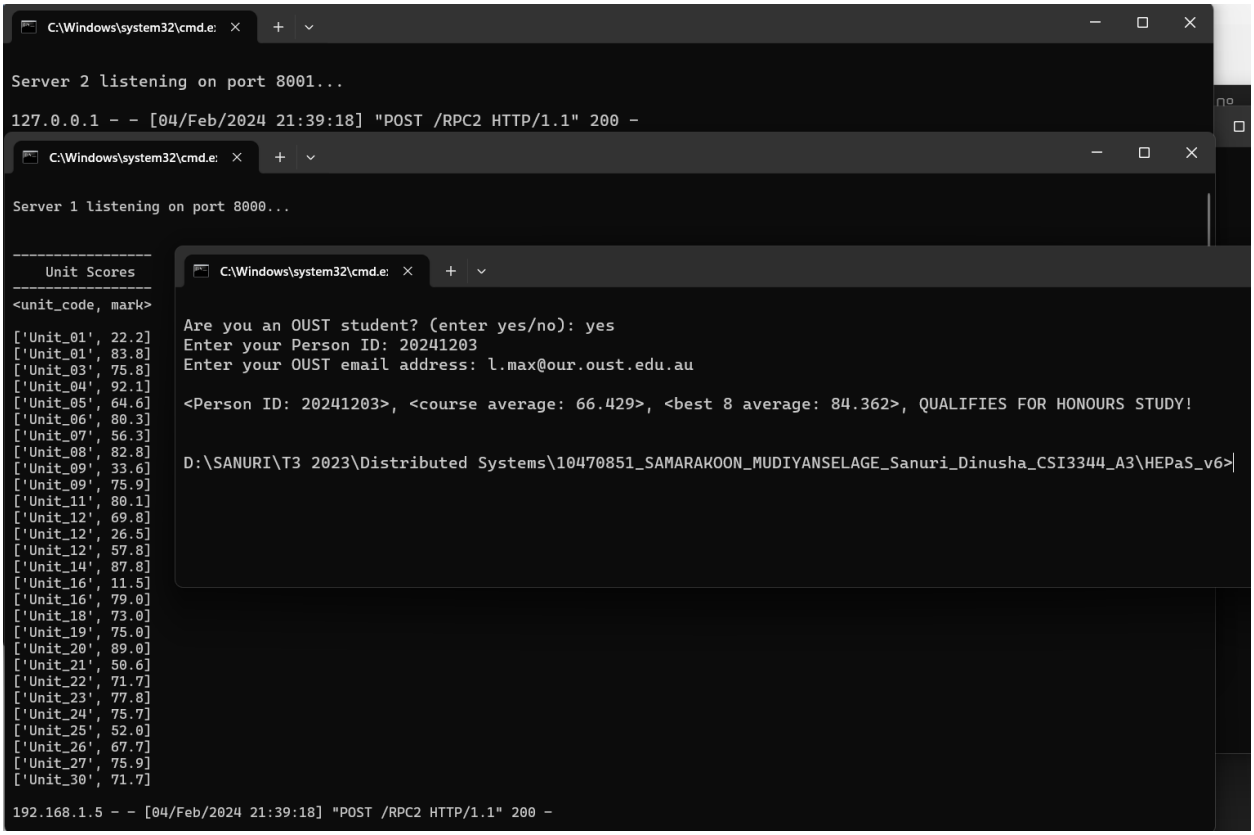

Figure 13: *Unit Scores* and Honors evaluation result of the above OUST student

7.2 Non-OUST Student Honors Eligibility Evaluation

Test case *TV06* has been used as an example for non-OUST student Honors eligibility evaluation.

A non-OUST student has entered details; *Person ID* and *Unit Scores* to *HEPaS* application and Honors assessment result is displayed on *Client* and *Unit Scores* are displayed at *Server-1* as in Figure 14.

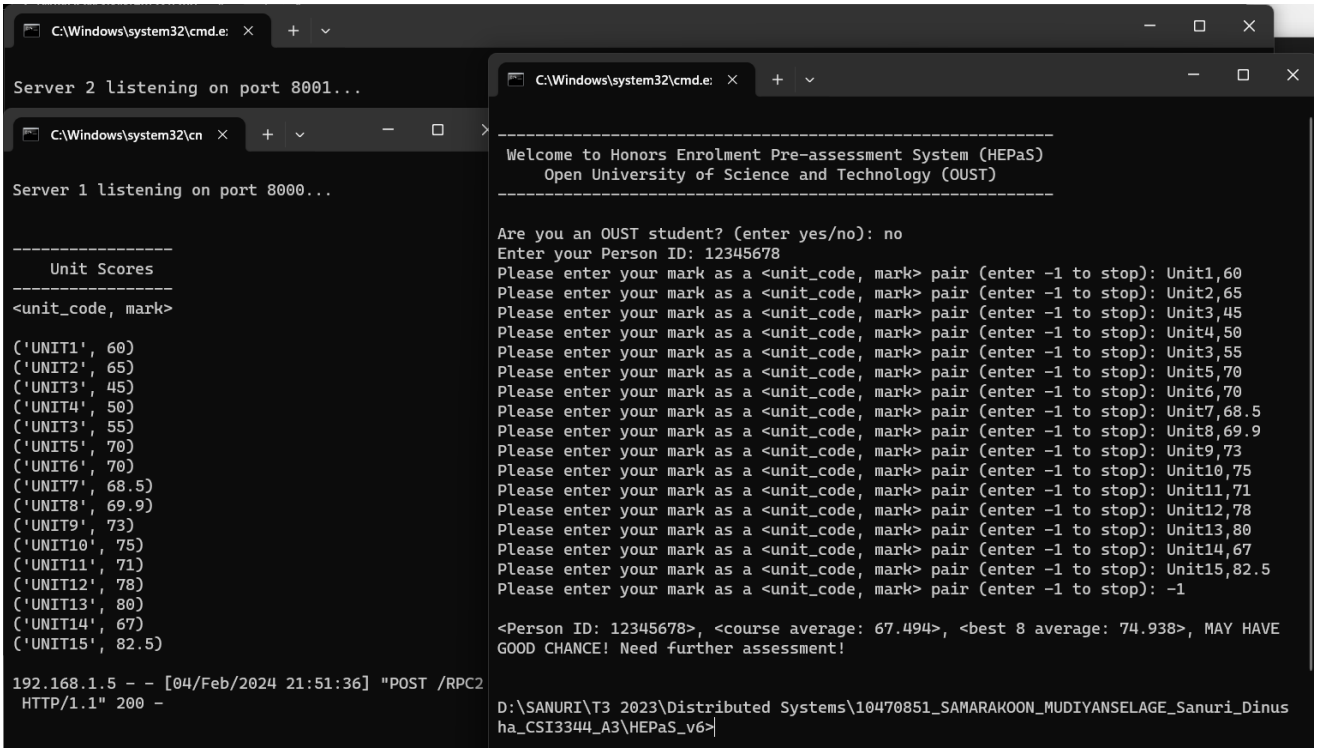

Figure 14: Displaying Honors evaluation result and entered *Unit Scores* for a non-OUST student

8.0 Discussion

HEPaS application implementation is involved with designing and developing a simple system to satisfy the system requirements. The main purpose of developing this system is to allow former and current OUST students and non-OUST students to self-assess the eligibility for Honors studies.

Since OUST students' learning records have been stored in a database, only the non-OUST students are required to enter their *Unit Scores*. However, both students are required to input a *Person ID*, but only OUST students are necessary to input their *OUST Email* as well for student status verification and retrieving *Unit Scores* from the database. Thereafter, the Honors assessment will be conducted by evaluating the criteria that requires *Course Average* and/or *Best 8 Scores' Average* and result will be displayed to the user (OUST/non-OUST student).

The HEPaS is a three-tiered distributed system that includes a client application, *Server-1*, and *Server-2* with *OSCLR database*. The user interface, which displays the prompt and result to the user was developed in *Client* program. The user entered details are communicated to *Server-1* that evaluates the user eligibility and returns the result to *Client*. When the user is an OUST student, before evaluating Honors studies eligibility, *Server-1* communicates with *Server-2* to retrieve *Unit Scores* of the student from *OSCLR database* after verification. *Server-2* is communicated with the OUST student details by *Server-1*. The *Unit Scores* are displayed at *Server-1* and the assessment result is displayed at *Client*.

The communication between *Client* and *Server-1*, and *Server-1* and *Server-2* were done through *RPC* by employing the in-built *"XMLRPC"* library to establish remote communication in this distributed system. Since *Client* can directly communicate with *Server-1* only, *Server-1* contains the proxy for *Server-2*; to relay requests from *Client*.

In addition to calculating average, authenticating OUST students and retrieving *Unit Scores*, and evaluating Honors eligibility; the system verifies every *Unit Score* by validating *Unit Codes* and scores entered (integer or float), tracking number of failed units, and displaying error messages for seamless user experience. The *OSCLR database* stores student information, course information, and *Unit Scores* of OUST students.

One of the challenges encountered while designing and implementing *HEPaS* was mainly related to establishing *RPC* communication with remote servers. As it was a novel experience and only

have seen an implementation example on a two-tiered system using *"Pyro4"* (a third-party library supporting *RMI* and not allowed to use for *Synchronous* communication pattern as per assignment documentation). Besides, *Client* can directly communicate with *Server-1*, had to ponder about techniques on relaying *Client* requests to *Server-2* through *Server-1*. Hence, a reference to *Server-2* was placed in *Server-1* to make *Server-2* available for communication.

As discussed before, basic requirements of the *HEPaS* application have been fulfilled through implementation. Moreover, the system has been tested on various test cases for valid and invalid inputs as tabulated in Tables 1 and 2 to ensure that possible exceptions caused by database connection and user inputs will be handled.

Potential future enhancements are implementing a graphical user interface (GUI) for OUST/non-OUST students and a database user interface (UI) for the database administrators (DBA) and enhancing security and scalability of the system to manage various user loads; hence, convenient access and reduced user errors could be guaranteed.

Conclusion

In this project, *HEPaS* application was designed and implemented as a three-tiered distributed system; *Client*, *Server-1*, and *Server-2*. Since *Client* could directly communicate with *Server-1*, *Server-1* relayed requests on authenticating OUST students and retrieving OUST *Unit Scores* to *Server-2*, which accesses *OSCLR database* tables and records. Communication between these components were established by employing *Synchronous* communication and *"XMLRPC"* library that allows remote server communication through *RPC*. A *"MySQL"* database was created for *OSCLR database* and populated its tables with the necessary data covering most test cases as well. Necessary user input validations and database connection establishment through handling exceptions to ensure a seamless operation of the programs, which can be found in the test cases. Finally, it was deployed as an executable file for convenient deployment.

In conclusion, *Synchronous* communication, and *RPC* through *"XMLRPC"* enabled convenient and seamless communication between remote servers and a client by managing their communication details; hence, this project has enlightened with designing and developing flowless distributed systems that have potential for enhanced security and scalability, and GUI and database UI for OUST/non-OUST students and DBAs respectively.

References

Geeks for Geeks. (2022, August 16). RPC implementation mechanism in distributed system. https://www.geeksforgeeks.org/rpc-implementation-mechanism-in-distributed-system/

Geeks for Geeks. (2023, April 22). Remote call procedure (RPC) in operating system. https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/

MySQL Community Downloads. (n.d.). *MySQL Installer* (Version 8.0.36) [Software]. Oracle. https://dev.mysql.com/downloads/installer/

MySQL. (n.d.). *Installing connector/Python with pip*. https://dev.mysql.com/doc/connector-python/en/connector-python-installation.html

Python. (2023, August 24). *Python 3.11.5*. https://www.python.org/downloads/release/python-3115/