CSI3344 Distributed Systems

Assignment 3: Group project & report

Objectives from the Unit Outline

- Implement a small distributed system using RPC or RMI.
- Apply algorithms to the solution of complex distributed system problems.
- Discuss the structure and functionality of distribution algorithms for distributed systems.
- Present outcomes of the research and/or development of a distributed system.

General Information

- Assignment 3 is a team-based (or group) assessment for up to two students. It consists of two parts therefore requires two lots submissions.
  - The first submission is a project report, together with the implementation codes (source and executable) of the mini programming project.
  - The second submission consists of two short video presentations: one is the project demonstration (one per group) and the other is an individual video reflection (one per team member). The project demonstration video is a video demonstration of the mini project completed, while the video reflection is one in which you have the opportunity to explain your own contribution to the group project, and your individual comments to the team work, etc.
- This assignment focuses on implementation of a small distributed system for an application scenario. It is a continuous task of (task 2 of) Assignment 2. However, while some basic design consideration from Assignment 2 is still valid, more detailed requirements are listed in this assignment so you should attempt your project based on what listed in this assignment document (instead of ones listed in content of Assignment 2).
- You are responsible for forming your team for the team-based project. You can form your team by signing in a group via the link underneath the “Assignments” in the “Assessment” section on BB, or alternatively if you have decided to form a group with a friend/classmate, you may email the details of all team members to your tutor so that he can help you set up a group. Please make sure you form your group by week 09.
- If you really wanted to do the Assignment 3 by yourself, that is fine (please let you tutor know in this case). However, in such a case, you should not expect any reduction in the workload of the assignment.

Due date: *(See Due date in Assignments section on BB)*

Marks: 100 marks (which will be converted to 50% of unit mark)

Format requirement of the Assignment report (5 marks)

Report content	**Cover/Title page** Must show unit code, assignment title, student IDs and name/s of all team members, and due date, etc.
	Executive Summary This should represent a snapshot of the entire report that your tutor will browse through and should contain the most vital information requested. This part should be in between Cover page and ToC.
	Table of Contents (ToC) This must accurately reflect the content of your assignment/report and should be generated automatically in Microsoft Word with clear page numbers.
	Introduction Provide background information of the project, define its scope, and state assumptions if any; Use in-text citation /reference where appropriate. **Main body** (this should be organized in a few sections) This part should contain (but not limited to): • Understanding of concepts/techniques involved in the report. • The problems being solved. • Strategies used to solve the problem (e.g., an approach/es to develop your solution/s, etc.). • A *User Manual* to install your project on to (two or more) computers and run your programs. • Test cases that you use to demonstrate your project in your group video presentation. • Comments/discussion or a critique of the solutions developed /achieved, etc. • Any other content you think necessary to be included. **Conclusion** Main achievement or finding/s from the assignment. **References** A list of end-text reference, if any, formatted according to the ECU requirements using the APA format (Web references are OK)
Other requirements	The length of the assignment/report should be, generally, of 2000~3000 words (excluding references and diagrams). The text must use font **Times New Roman** and **be not smaller than 12pt**.

Submission requirements (5 marks)

- *Report submission* (one submission per team): This submission should include a written report, all implementation codes (source or executable) and any additional documentation associated with the report/project. This submission should be submitted by the team leader only (i.e., through the team leader's BB submission portal - the other team member should not submit the same copy version).

  Video submissions: The submissions should include:
  (1) video demonstration of the project (one submission per team).
  (2) individual video reflection (one submission per team member).

  Note that separate submission links are available for each submission.

- The Project/Report submitted should be one single compressed file (e.g., in .zip format), which should contain all your documents (e.g., written report, source & executable codes/files and any other support documents, if any). The written report file must be in Word or PDF format (please refer to the section of *Format requirement of the Assignment report*). Please rename the .zip file in a format of

  <Team-leader's Student ID>_< team-leader's first & last name>_< team member's ID_ first name>_CSI3344_A3.zip.

  If the assignment is done by an individual, please rename the .zip file in a format of

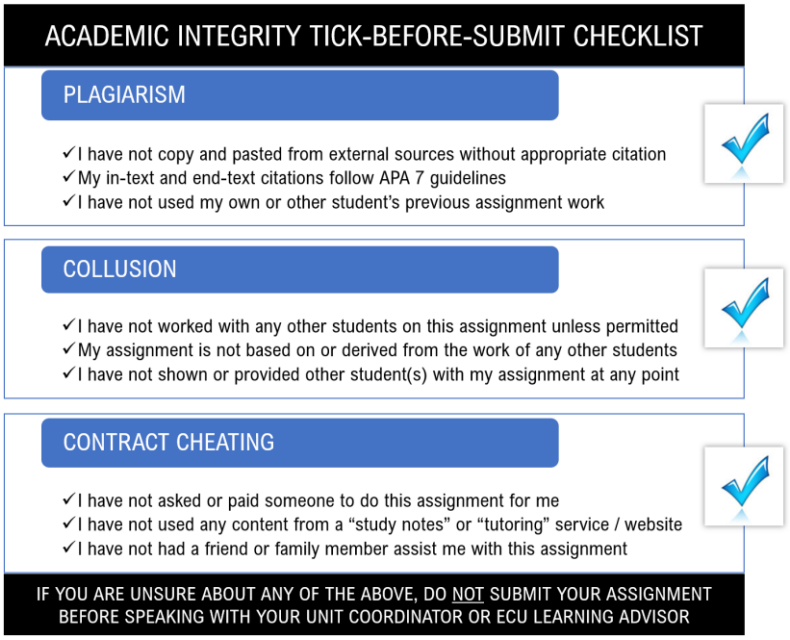
  <Student ID>_< first name>_<last name>_CSI3344_A3.zip.

  > *As an example, if your team leader's ID is 12345678, his name is Ben SMITH, the team member's ID is 15555555 and his first name is Max, the submission file should be named 12345678_Ben SMITH_15555555_Max_CSI3344_A3.zip.*
  > *If the assignment is done by Ben SMITH alone, the submission file should be named 12345678_Ben SMITH_CSI3344_A3.zip*

- Students Identity Verification (SIV) requirements
  For assessment integrity, the team's video demonstration requires SIV. All team members should be present at the demonstration/presentation (i.e., unless otherwise discussed with the tutor, each team member should show or demonstrate a part of the project in the video), and SIV should be arranged in some way, e.g., either by capturing your face using computer camera (at least for a few seconds) in the video, or capturing your student ID card during the demo, etc.
- Note that files found to contains viruses or other infecting mechanisms shall be awarded a zero mark.
- Your attention is drawn to the university rules governing cheating/plagiarism and referencing. Please refer to the section of *Academic Integrity and Misconduct* in the next page.
- ECU rules/policies will apply for late submission of this assignment.

Academic Integrity and Misconduct

- This assignment is a **group assignment** (for up to two students per team). Your entire assignment must be your own work of the team (unless quoted otherwise) and produced for the current instance of the unit. Any use of uncited content that was not created by your team constitutes *plagiarism* and is regarded as Academic Misconduct. All assignments will be submitted to plagiarism checking software, which compares your submission version with previous copies of the assignment, and the work submitted by all other students in the unit (in the current semester or previous years/semesters).
- *Never give any part of your assignment to someone else or any other group,* even after the due date or the results are released. Do not work on your team assignment with other students or teams. – You can help someone (in other team) by explaining concepts, demonstrating skills, or directing them to the relevant resources. But doing any part of the assignment for them or with them or showing them your work is inappropriate. An unacceptable level of cooperation between students/groups on an assignment is *collusion* and is deemed an act of Academic Misconduct. If you are not sure about plagiarism, collusion or referencing, simply contact your lecturer or unit coordinator.
- You may be asked to explain and demonstrate your understanding of the work you have submitted. Your submission should accurately reflect your understanding and ability to apply the unit content and the skills learnt from this unit.
- Before submitting your assignment 3 (and video demos), team leader should make sure you have checked all items in the **Academic Integrity tick-before-submit checklist** below and watch the video by clicking the link next to the checklist image:



Watch this video before submitting your assignment

The tasks:

Background Information

A university is interested in developing a simple ***Honours Pre-assessment System*** (HPaS) that can be used by their students or other users to assess if they are eligible for an *Honours study.*

This is a continuous work of Task 2 of your Assignment 2. In order to eliminate confusion, in this assignment, we reproduce the application scenario and added more detailed requirement (see below).

HPaS is a simple distributed system with one client-side application and one or more server-side applications. The client (application) collects a user's data, performs necessary pre-processing of the data items, sends the data to the remote application/s at server-side to assess whether the user meets the conditions of *Honours* study, and then waits for server's evaluation results, and finally displays evaluation outcome/s to the user.

In this project, you are requested to implement a version of the above-described HPaS under certain practical requirements.

1. The mini project

You should complete the project in two stages. The first stage is to implement a simple *"two-tier" interaction* between the client application and server applications, while the second stage is to upgrade the *"two-tier" interaction* into a *"three-tier" interaction* by adding a data server that acts as a database server (or data center). The database server can process data request / service from the server-side application/s that was completed in the first stage.

Stage 1: the *"two-tier" version implementation* (*50 marks*):

Refer to the source Java-like pseudo-code in Appendix A that defines a class *Average*. It is an application that runs on a local machine. It accepts a number of integers as marks to some units. Then it adds them up, and finally returns the average (as a real number of double type). The result is displayed in a message box.

This application code can be converted to a client-server interaction. For example, the client application (or process) accepts integers (of unit marks) from local machine (or process), and then sends the data to the server, requesting a server-side application (or process) to calculate the *Average* value, which was then sent back to the client by the server. The result is then displayed in the interface at the client side.

Based on this idea, practice using RMI programming to complete a mini programming project for a two-tier client-server application. You may start by modifying (or converting) the above application/code to a simple client-server application for *HPaS*. That is, the client makes RMI to a remote application/method, say, *Evaluator*, which is used to assess if a user is qualified for an Honours study based on the record of his/her Bachelor's course.

The basic requirements of the two-tier application include:

(1). Only authenticated users are allowed to use this application (from client-side).

(2). Client application allows a user to enter the Person ID and a sequence of unit scores /marks (or in <unit code, mark> pair), one by one, through keyboard, and also provides functions to display the evaluation result received from the server.
The number of input marks should be between 12 and 30, including "Fail" (e.g., unit score less than 50) and repeated marks of the same unit.
The client application also does some necessary data pre-processing, for instance, a user is automatically disqualified for Honours study if he/she has 6 or more "Fail" results during study of his/her bachelor's course.
The client then sends the collected data to the server, requesting a service (i.e., evaluation) and then waits for a return from the server, and finally displays the evaluation results to the user.

(3). Server application provides services/operations that process the input data, calculate the averages, assess the qualification against the evaluation criteria, and return the client the evaluation result.

The basic operations on the server include:

- displaying individual scores on the server in their input order;
- selecting the best (or highest) 12 marks, and calculating the *average* of the best 12 marks;
- calculating the *course average*;
- evaluating the qualification according to the Honours evaluation criteria (see below); and finally
- sending the evaluation result/s back to the client.

The *Honours evaluation criteria* are as follows:

- If the course average is greater than or equal to 70, return a message
  "<Person ID>, <course average>, QUALIFIED FOR HONOURS STUDY!";
- If the course average is less than 70, but the average of best 12 marks is 80 or higher, return message,
  "<Person ID>, <course average>, <best 12 average>, MAY HAVE GOOD CHANCE! Need further assessment!".
- If the course average is less than 70, but the average of best 12 marks is between 70 and 79, return message
  "<Person ID>, <course average>, <best 12 average>, MAY HAVE A CHANCE! Must be carefully reassessed and get the coordinator's special permission!".
- If the course average is less than 70, and the average of best 12 marks is also less than 70, return message

"<Person ID>, <course average>, <best 12 average>, DOES NOT QUALIFY FOR HONORS STUDY! Try Masters by course work."

(4). The client and server-side applications should be able to run in different machines.

(5). The client should do necessary data validations, for example, while a user may have up to three marks for a same unit, no more than two "Fail" grades and no more than one Pass (i.e., unit mark is greater than or equal to 50) grade is allowed, etc.

Recommendations & restrictions to Stage-1 programming:

You are encouraged to choose from one of the two implementation styles:

(i) Use traditional *synchronous communication* pattern to implement the system, that is, use RMI (or RPC) technique within one programming environment only. While it is OK to use the class library available in the language environment to implement the two- (and/or three-tiered) application, no third-party package/s (outsides the language environment) can be used.
In this case, the corresponding third-tiered database server (to be implemented in Stage-2) could be simplified to using a set of arrays to mimic a relational database.

(ii) Use either *synchronous* or *asynchronous communication* pattern (or a combination of them) to implement the system. In this case, you can use RMI (or RPC) technique, but also use all class libraries from the chosen programming environment and third-party package/s to implement the application.
In this case, the corresponding third-tiered database server (to be implemented in Stage-2) must use (or connect to) a real/existing database server (e.g., SQL server, MySQL, or Oracle server, etc.).

Remember the steps to create an RMI application (e.g., in Java, etc.):
1. define the interface for the remote object 2. implement this remote interface 3. create stub and/or skeleton 4. implement the server software 5. implement the client software

Stage 2 the *"third-tier" version implementation* (15 marks):

This stage consists of two sub-tasks:

(1) Using various data to test the client/server application that is completed in *Stage 1* (for simplicity, we call the server-side application *server-1*).

(2) If a user is a (past or current) student of the university where HPaS is to be installed/run, he/she may request to use his own Bachelor's course study historic data stored in the *Student Records* database (of the university) to do the Honours study pre-assessment, rather than entering his/her individual unit results.
In this stage, you are requested to create a *third-tiered* server, a simplified database server (called *server-2*) to mimic the server of the *Student Records* database. The database records students' course study historic data, such as unit selection history, unit results (each unit allows up to 3 attempts if there was a "Fail" grade/s recorded). You should also make necessary extensions to the client and server-1 applications that were implemented in Stage-1 so as to complete the three-tiered system.

The server-1 and server-2 communicates by RMI as well, however the client-side application has no direct access/link to server-2.

- Extension to the Client-side applications:

Client-side application is updated to allow a user to do the honours study pre-assessment by one of the two ways:

(i). enter his individual Person ID and a sequence of unit results to do the honours study pre-assessment as usual (i.e., the same as what were done in stage 1), with an optional action: whether or not the user wanted his newly entered data to be stored or updated in the database for future enquiries; or

(ii). use his/her student records to do the honours study pre-assessment. In this case, the user should enter a student/Person ID only, and the client submit a request to the server, requesting the server to use the student's unit results stored in the *Student Records* database for an evaluation.

- Extension to the Server-side applications:

(iii). Server-2: create server-2 database and implement following functions at server-2 side:
Server-2 stores student records, which may include student's course selection history and unit selection history, and related unit results, etc.
For simplicity, we assume that server-2 store unit results for one course per user, and each student's record can hold results for up to 30 units only. In addition, server-2 allows new users to be created and their own unit records stored, say after

received data for honors pre-assessment. That is, server-2 receives data from client application by way of server-1 and creates a NEW student record if the Student ID is not existed in the database. For example, when a user entered an ID and marks of individual units (see step 2 in Stage 1), server-1 may pass the data to server-2 if the user requested to do so. In such a case, server-2 will make necessary data validation, create a new user, and save the data into the student's records, if applicable.
Server-2 also allows updating data and enquiring data based on the request from server-1.

(iv). Server-1 extension:
When a client user wanted to use his/her student records to do the honours study pre-assessment, Server-1 makes a request to server-2 (with the Person ID as a parameter), asking for the student records. Once received data, it then uses the data to do evaluation and then returns evaluation result back to the client.
In the case of client user passing a list of (up to 30) unit results/marks for the honours study pre-assessment, if the client user also wanted to store his/her unit results on to the Student Records database, server-1 application makes necessary data validation, forms a data set and passes it to server-2 for creating/updating the user's entry using the database at server-2 side.
Necessary data validation should be done at server-1 side, if applicable.

Bonus Marks (up to 5 marks):

For data checking and verification purpose, you are encouraged to develop an additional mini database interface for server-2 that helps preparing and/or checking data stored in server-2 side. This is more important/convenient when your project connects to the server-1 with a real database server (e.g., MySQL server, SQL server, Oracle server etc.).

- Develop a mini interface using SQL or other tool from the database server such that it
  (i) allows users input (or enter information for) their unit study data. Use an integer to keep track of how many record have been created so far. (Be careful – Don't allow more than 30 unit results for any user)
  (ii) allows users search unit result information by Student ID + Unit ID, etc., instead of by Student ID only, etc.
- Explain and extend to accommodate error messages, etc.

This additional work and effort to improve the program and make it more useful can be worth bonus marks. Up to 5 marks can be added for the above improvement. However, Bonus marks are not required and will not improve your score above 100% of the assignment score.

Other basic requirements

- The applications should include all required components.
- The observable behaviours of your application should be consistent with what is described.
- The application provides necessary error handling mechanisms.
- The application must be implemented using one single programming language, chosen from a list of o-o programming language set (Python, Java, C#, C++). However, a SQL-compatible language can be used in implementing the third-tier server application/s in Stage-2.
- The system must be runnable *off-the-shell*. That is, executable codes should be produced, and can be run in Windows' Commend Line prompt (or by double clicking on the executable in a Window's folder). In other words, the final product of the project (i.e., the system developed) should be runnable independent of the programming environment used (e.g., Java NetBeans, .Net, etc.)
- The project report should be well structured and informative (no more than 20 pages)*, and not contains codes of your project (Note: all code files should be included separately the submission .zip file).

2. The written report and possible structure (15 marks)

Refer to the **Format requirement of the Assignment report** in page 2. The Main Body should include:

i. (A brief) Introduction of RMI
ii. Application requirements
iii. Application design and implementation procedure
iv. User manual: application setting-up and usage steps (with possible screenshots) (Note: this is to allow your tutor to install your project code/s to their/lab computer/s for testing your project/codes)
v. Example snapshots demonstrating application running status and input/outputs

(Note: source codes are required and must be includes separately in .zip files)

3. The video demonstration/presentation (20 marks)

This video should mainly be a demonstration of the group project, although you may also present other contents such as a brief introduction to RMI and explanation of the project report, etc.
Some requirements:

(1) The duration of video should be between 5 and 10 minutes.
(2) SIV should be presented.
(3) The demo should include
- a brief introduction of software used to develop the project,

- the architecture of the system completed,
- user authentication/registration;
- the demonstration over some test cases:
  (i) the case/s where a sequence of valid unit data is entered for Honours study pre-assessment;
  (ii) the case where a user enters a student ID and requests a Honours study pre-assessment using existing data stored in the Student records at server-2 ;
  (iii) the cases with some invalid data input (e.g., less than 12 unit marks was entered, or greater than 30 unit marks is entered, non-authenticated user used the system, etc.), causing the system to show error handling messages, etc.

(4) Any special features that your team wanted to show.

4. The video reflection (10 marks)

This video is an individual one. It should not longer than 5 minutes.

Each team member should, in this video, explain his/her own contribution to the group project, provide their individual comments to the team work that they want the tutor to know, plus any further discussion into the project matter, and possibly some lessons learnt from the project, etc.

(Go to the next page)

Indicative Marking Guide:

	Criteria	Out of marks	You got
Implementation of the project	Stage 1: Two-tiered version completed + authentication, etc.	40	
	Stage 2: three-tiered version completed, tested	15	
(Project) Bonus mark	(up to 5 marks)		
The Report	Executive summary: abstraction of the report & vital information as a whole; Thought/idea organization: conjunctions & cohesion; Clarity: discussion flow and integrity etc.; Citations to References; User manual; Test cases; Conclusions achieved; Quality of the report: Report presented as per Format requirement; report length - not too long and too short (e.g., 2000-3000 words, of 10-20 pages?).	15	
submission	Document presented as per format requirement Submitted as per submission requirement		
Video demo	As per requirement	20	
Video reflection	As per requirement	10	
Penalty	(possible Plagiarism or Late submission penalty)		
Total	Total mark of 100 (which is then converted to 50% of unit mark)	100 / (50%)	

(Go to the next page)

Appendix A:

```
// Fig. 4.9: Average2.java from Deitel & Deitel <<Java How to Program>> (5th ed)
// Class average program with sentinel-controlled repetition.
import java.text.DecimalFormat;
import javax.swing.JOptionPane;
public class Average2 {
        public static void main( String args[] ) {
               int gradeCounter,  // number of grades entered
                   gradeValue,    // grade value
                   total;         // sum of grades
                double average;    // average of all grades
                String input;      // grade typed by user
         // Initialization phase
               total = 0;         // clear total
               gradeCounter = 0;  // prepare to loop
         // Processing phase
               // prompt for input and read grade from user
               input = JOptionPane.showInputDialog(
                       "Enter Integer Grade, -1 to Quit:" );
               // convert grade from a String to an integer
               gradeValue = Integer.parseInt( input );
               while ( gradeValue != -1 ) {
                   // add gradeValue to total
                   total = total + gradeValue;
                   // add 1 to gradeCounter
                   gradeCounter = gradeCounter + 1;

                   // prompt for input and read grade from user
                   input = JOptionPane.showInputDialog(
                      "Enter Integer Grade, -1 to Quit:" );
                   // convert grade from a String to an integer
                   gradeValue = Integer.parseInt( input );
               }

       // Termination phase
              DecimalFormat twoDigits = new DecimalFormat( "0.00" );
              if ( gradeCounter != 0 ) {
                    average = (double) total / gradeCounter;
                   // display average of exam grades
                   JOptionPane.showMessageDialog( null,
                   "Class average is " + twoDigits.format( average ),
                   "Class Average", JOptionPane.INFORMATION_MESSAGE );
              }
              else
                   JOptionPane.showMessageDialog( null,
                   "No grades were entered", "Class Average",
                   JOptionPane.INFORMATION_MESSAGE );
              System.exit( 0 );    // terminate application
       } // end method main
}  // end class Average2
```

Reference:

- Deitel and Deitel, Java How to Program (fifth ed), Prentice Hall, 2003.

========== The end of Assignment 3 description==========