CSI3344 Distributed Systems

Assignment 3: Distributed system project & video demo

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 (or A3 for short) is the major assessment of the unit. It is a group assignment with teamwork and individual work. Therefore, it is divided into two parts: the first part is a **group assessment** of **up to two students.** Your group needs to complete a project of building a small distributed system and write a report on the project. The second part is a **video presentation** to be completed by individual team members. Therefore, two batches of submissions are required.
  – The first batch of submission includes submission of the project implementation (including source code/s, and executables if applicable) and the written report. This part is a *group* task so it should be completed by all group members and only submitted by the team leader (that is, the other team member should *not* submit it repeatedly).
  – The second batch of submission consists of one *video link* and an *optional Peer-Review* document. This is an individual task/work so each team member of the group should prepare and submit their own version of the work seperately. The video link is the web link to a video presentation, which should contain two parts: a demonstration of the project completed by the group, and an individual's *reflection* on the project work. You can also include an (optional) *Peer-Review* document within this batch of submission.
- Group formation for A3:
  - You are responsible for forming your team for the team-based project. You can form your team by signing in a group via the A3 group sign-up page on Canvas (e.g., via People section), or alternatively if you have decided to form a group with a friend/classmate, you can email your tutor with the details of all team members so he can help you set up a group. Make sure to form your team by week 8 or before.
  - Once a group is formed, the team members will be locked. - If you want to change the team members of your group, you must email your tutor to request a change. Note that group membership changes can only be made within the first 9 weeks.
  - In some special circumstances, groups of three students are possible (for example, only one student is left unpaired). If you wanted to form a group of three, that's fine, but your group will need to do some extra work (see more details below). On the other hand, if you really wanted to do A3 yourself, that's fine too. In such a case, you must let you tutor know your decision by or before week 8 and you should not expect a reduction in workload of the assignment.

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

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

Background Information and Assumptions of the Project

A government agency is interested in developing a simple ***personal income tax return estimate*** (PITRE) *system* that can be used by taxpayers to estimate their tax returns themselves. In this project we practice developing a dummy system that mimics the PITRE system.

Preliminary: How a *personal income taxation system* (PITS) works

Taxes are one of the largest sources of state revenue. The *Personal income tax* is the compulsory contribution to state taxation, levied by the government on personal income. It includes taxes from diverse sources such as wages/salaries, dividends, interest, rents, gambling winnings, product sales, and so on.

For example, Australian personal income tax rate on (annual) taxable income is shown in the table below:

Table 1: Australian tax rate (as in 2022)

Taxable income	Tax on this income
0 - $18,200	Nil
$18,201 – $45,000	19c for each $1 over $18,200
$45,001 – $120,000	$5,092 plus 32.5 cents for each $1 over $45,000
$120,001 – $180,000	$29,467 plus 37 cents for each $1 over $120,000
$180,001 and over	$51,667 plus 45 cents for each $1 over $180,000

Generally, a government's taxation system can be divided into several sub-systems, including a *tax reporting* system*,* a *tax returning* system, and so on.

The *tax reporting* system is a system that helps Government's Tax Office (GTO) collect tax and tax related data during a financial year. It involves hundreds of thousands of parties (such as organizations, companies, universities, banks, government departments, etc.) and/or clients. These clients work together (e.g., to collect tax and tax related data, etc. during the financial year) to ensure that taxes can be properly calculated at the end of the financial year. For example, GTO provides a *payroll-tax client* for each company's payroll department so that on payday, the payroll department can use it to calculate payroll taxes on their employees' wages/salaries and collect taxes on behalf of the GTO. Likewise, GTO offers *interest-taxation* clients to banks (and other financial institutions). Before a bank pays interest to their costumers (e.g., when a fixed-term deposit matures), the bank runs the client to calculate the tax liability of the interest earned by the costumer. The bank then collects the tax on behalf of the GTO (and reports it to the GTO) and pays the customer the amount of interest minus the tax. All tax-related data collected by GTO are stored in a database for the current financial year. For ease of reference, we call the database the *Personal Income Tax Database* (or PITD for short).

The *tax returning* system is used by taxpayers or companies, etc., to finalize their annual income tax after the end of the financial year. Personal (or *Individual's*) *income tax return* refers to the process of completing an individual's annual income tax declaration. One of the main tasks of the process is to calculate the individual's annual income from various sources and the corresponding tax due to the government. After the end of the financial year, if an individual's tax withheld during the financial year exceeds the tax amount due to the government, the GTO will refund the difference to the taxpayer within a fixed period (known as the *tax return period*) after the end of the financial year. - This process is sometimes called a *tax return,* or *tax refund*.

During tax return period, GTO makes a *tax-return client* (such as *etax* in Australia) available to all taxpayers so they can use the tax-return client to complete their tax-returns. That is, a taxpayer can run the client that sends a request to one of the GTO's servers, requesting the server to calculate the taxpayer's annual tax amount and compare it with the withholding tax amount recorded in the PITD, thereby determining whether the taxpayer gets tax refunds from the GTO or should he/she pay more tax to the GTO.

Example: Jim Max's annual taxable income in fiscal 2022 was $50,200. His withholding tax was $8,735.00. Based on the tax rate shown in Table 1, Jim's tax liability should be
$5,092 + $0.325 * ($50,200 -$45,000) = $6, 782.00
Jim will also need to pay *Medicare levy,* which is 2% of the taxable income.
$50,200*2% = $1004.00
Therefore, Jim can get a tax refund of
$8,735.00 - $6,782.00 - $1004.00= $949.00

From the technical point of view, a PITS is a complex distributed system. It may consist of many clients and servers. While some clients collect tax related data on behalf of GTO during the financial year, other clients and/or servers may be used to do things like calculating individual's tax return at the time of tax return, storing tax related data to PITD, or accessing and getting tax related data from PITD, and so on.

A simplified *personal income tax return estimate* (PITRE) system

Consider the following simplified PITRE system: it consists of two clients and two servers only.
*(**Note:** Both servers and the TRE client must be implemented by all groups.
The PTC client is only required for groups of three team members).*

The client 1 (or TRE client): This is individual's *tax return estimate* client issued by the GTO at the beginning of the tax return period. An individual can use it to get an estimate of his/her tax return for the last financial year. Details of the TRE client can be found in "The System Behaviour" section and "PITRE behaviour and requirements" section.

The client 2 (or PTC client): This is the *payroll-tax client* available to (the payroll departments of) all companies/organizations. On payday, a PTC client is used to calculate taxes on employees' wages (or salaries). Depending on whether the company pays the wages weekly or bi-weekly, taxes are calculated based on the *mapped weekly-wage-based tax* (WWBT[1]) rate or the *mapped*

[1] The *mapped WWBT* rate is the tax rate used to calculate tax on weekly wages. With this rate, if an employee's current (weekly) gross wage is *x*, it is assumed that the employee's annual income be *x* * 52 (i.e., it is assumed that the employee would receive the same weekly gross wages and be paid 52 times in total during the year). The tax for the current payout cycle can then be calculated and collected. For example, suppose that Jim's current gross (weekly) wage is $960.56. The *payroll-tax client* calculates the tax payable to GTO based on the assumption that Jim's annual income is $960.56 * 52 =$49,949.12. In this example, the annual income of $49,949.12 would place Jim on the third row of Table 1. So, his annual tax will be calculated using the following formula
$5092 + ($49,949 -$45,000) * 0.325 =$6700.43
The pay-roll tax amount that Jim must pay for this pay cycle is
$6700.44 / 52 = $128.85
In addition, he also needs to pay the weekly component of Medicare levy (ML), which is 2% of the grass wage:
$960.56 * 2% = $19.21
His net pay will be: $960.56 - $128.85 – 19.21 = $812.50
Note that the pay-roll tax amount varies depending on the grass wage of the current payout cycle only. If Jim's next (weekly) gross wage becomes $500, based on similar calculation, you can work out that Jim would have to pay payroll tax of $28.50 on that payout round (plus ML). If his gross wage in the following pay-cycle is $300, then he does not need to pay any payroll tax on that round of payout.

biweekly-wage-based tax (BWBT[2]) rate, derived from Table 1. For each employee in the organization, the PTC calculates the pay-roll tax (based on the employee's gross wage/salary of the current pay cycle) so that the payroll department can collect the payroll-tax amount (from the employee's current pay round) on behalf of the GTO before the net-pay (i.e., the gross wage/salary less the tax) is paid to the employee. Note that the payroll tax also includes some other tax component such as *Medicare Levy*, and *Medicare Levy Surcharge*, etc. (see below "The system behaviour" section). In the end of the pay cycle, the PTC client also generates a report containing tax-related data of the current payment cycle. The tax-related data can include information about the employer and all employees who received payments during the current pay cycle. For simplicity, we assume that the employer's information includes its National Business Number (NBN), the business name, and location. The employees' data is a list of tax records, one for each employee. An employee's tax record may include items like employee's ID, name, Tax File Number (TFN), pay period (e.g., the start and end dates of current pay cycle), date of the payday, gross wage/salary, tax levied on the gross wage/salary, and net wage/salary (of the current pay cycle), etc. The PTC client then sends the report to GTO (so that the data can be stored in the PITD).

> *Note:*
> A payroll-tax client can be much more complicated in the real world because there are many situations when an employee's pay-roll tax is calculated. The payroll taxation formulae used to calculate payroll tax varies depending on the employment type, e.g., full-time, part-time, or casual employment, etc. Even with a full-time job, the tax formula used to calculate payroll tax may also vary depending on whether they are paid weekly, bi-weekly, etc. In this assignment, we only consider a *simplified* version of such a payroll-tax client, i.e., the client used for calculating payroll tax for employees who work full-time and are paid in biweekly basis.

The server 1: This server is a simplified GTO's server. It offers two primary functions. During the financial year, the server is used to collect all taxpayers' tax-related data, which are then be stored in a separate database server (i.e., the PITD). After the financial year is ended, this server 1 is used to perform individual's *tax return* (and *tax return estimate*).

The server 2: This is a database server, i.e., PITD. It can only be accessed by the *server 1* (not by any of the two clients).

The system behaviour

Suppose that a taxpayer wants to estimate his/her tax return using the PITRE. The taxpayer enters his/her annual taxable income and the tax withheld and requests the server to calculate his/her tax return, or tax payable if he/she needs to pay more tax to GTO.

In addition to the tax on taxable incomes, an individual also needs to pay *Medicare Levy* (*ML*), which is usually 2% of the taxable income. An individual may also need to pay *Medicare Levy Surcharge* (MLS) - If a person's annual taxable income is greater than a certain threshold, and does not have a *Private Health Insurance Cover*, he/she will have to pay the MLS. The MLS can be calculated based on the rules in the following Table 2.

[2] The *mapped BWBT* rate is similar to the abovementioned mapped WWTR rate, but it applies to the case when employees are paid bi-weekly rather than weekly. If an employee is paid on a bi-weekly basis, and his/her current gross wage is *x*, it is assumed that the employee's annual income be *x* * 26 (i.e., it is assumed that the employee would receive the same (biweekly) gross wage every two weeks of the year and is paid 26 times in total). Tax calculation follows similar rules as described above (however 52 is replaced with 26).

Table 2: Medicare Levy Surcharge (MLS)

Taxable income	**MLS on this income**
0 - $90,000	0%
$90,001 ~ $105,000	1% of the taxable income
$105,000 ~ $140,000	1.25% of the taxable income
$140,001 and over	1.5% of the taxable income

An individual's *net income* is the sum of all his/her incomes taken home during the financial year.

An individual's tax estimate can be calculated using the following formula:

tax_estimate = *taxable_income - net_Income – tax - ML – MLS*

If the tax_estimate is great than 0, it is the amount of money to be received from GTO in this round of tax return. Otherwise, the individual will need to pay more tax to GTO (and the amount of tax payable is |tax_estimate|).

Notes: For simplicity, we have omitted some possible items when calculating the tax estimates in the simplified PITRE system. For example, we do not include taxpayer's allowable deductions in our calculations. In general, a taxpayers may be entitled some allowable deductions (e.g., work-related clothing deduction, such as uniform expenses, work-related self-education deduction, such as textbooks & stationary, work-related travel expenses, etc.). The deduction amount can be entered at the time of data entry and be deducted from the taxable income. In addition, Medicare Levy may be applied only to taxpayers whose annual taxable income are more than a certain threshold. These details have been removed from the formula for calculating tax estimates in the PITRE.

Your tasks:

You are requested to design and implement the simple distributed system, PITRE, as described above, that can be used for taxpayers to estimate their tax returns.

PITRE is a simple *three-tiered* distributed system. The system has one (or two) client-side application/s, one or more server-side applications, and one database server which holds all taxpayers' tax related data/records.

- The TRE client (application) collects a user's data, performs preliminary pre-processing of the data items, and then sends the data to the remote application/s at server-side.
- The server-side application authenticates the user using the data received from the client. Upon a successful authentication, the server-side application either does a tax-return estimate based on the data received from the client or sends a request to the database server to fetch the user's tax related data/records and then uses that data to do the tax return estimate. The estimate result is then sent back to the client application.
- After receiving the data from the server, the TRE client application finally displays the estimate to the user.

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

1. The mini project

PITRE behavior and requirements

(1). Only authenticated users can use this application (from the client-side).

(2). Client application (i.e., the TRE client) allows a user to enter data for a tax return estimate (TRE). A user may or may not have a tax file number (TFN), which is a 9-digital number (e.g., 331421123). (Note: The GTO uses a TFN to uniquely identify a taxpayer, therefore the TFN can uniquely identify a taxpayer's data records in PITD).

The client application first prompts the user to answer if he/she has a TFN.

If the user has a TFN, the client then asks the user to enter her/her Person ID, his/her TFN, and other information (e.g., first and last name, and email address, etc.) for authentication purpose.

If the user doesn't have an TFN, the client requests the user to enter a Person ID and a sequence of biweekly net-wages and the corresponding tax withheld (e.g., in <`net_wage, Tax_withheld`> pair), one by one, through keyboard. The number of data items entered should be between 1 and 26.

Then the client application asks user to answer if he/she has a *Private Health Insurance Cover* (PHIC), which determines if he/she needs to pay a Medicare Levy Surcharge.

The client then passes all the collected data to the server for a TRE.

The client then waits for the estimate result from the server, and finally displays the result to the user.

(3). Server application (i.e., the *server-1*) provides services/operations that process the data received from the TRE client, calculates user's annual taxable income and the tax withheld, and generates the tax return estimate against the estimate criteria, etc. It then returns the estimate result to the TRE client. The server may send a request/s to the database server (i.e., server-2) to fetch a user's payroll tax data, etc., if necessary.

The basic operations on the server-1 include (but not limited to):

- displaying individual tax related data (e.g., a sequence of the biweekly net-wages and tax withheld, in <`net_wage, Tax_withheld`> pairs) on the screen in their input order;
- calculating the annual taxable_income;

- calculating the tax return estimate according to the individual tax return estimate criteria (see below);
- sending the estimate result/s back to the TRE client;
- using a user's TFN, server-1 may send a request to server-2 to get the tax-related data of the given user, such as the total amount of taxable income (or total net income), tax withheld, etc. It may then use the data to do tax return estimate, and finally send the estimate result to the TRE client.

Individual tax return estimate criteria are as follows (in sequence):

- Basic tax: tax on table income can be calculated using Table 1. This generates two results: `<individual's taxable income>` and `<tax>`;
- Additional taxes:
  - Each individual needs to pay *Medicare Levy* (*ML*), which is 2% of his/her taxable income.
  - An individual may also need to pay *Medicare Levy Surcharge (MLS)*.
    If a person's taxable income is greater than a certain threshold, and does not have a *Private Health Insurance Cover*, he/she will have to pay the MLS. The MLS can be calculated using the rules/rates listed in Table 2.
- If the user doesn't have a TFN, the server 1 returns the following data items:
  `"<Person ID>", "No TFN", "<annual taxable income>", "<total tax-withheld>", "<total net-income>", "<estimated tax refund amount>".`

- If the user has a TFN, the server 1 returns the following data items:
  `"<Person ID>", "<TFN>", "<annual taxable income>", "<total tax-withheld>", "<total net-income>", "<estimated tax refund amount>".`
- In any other cases, server-1 returns an error message to the user, with proper description of the error, e.g., `"The TFN entered does not exist…", or "No tax records found for the person with TFN = <TFN>", etc.`

(4). Database server (or PITD, also called *server-2*): It stores individual's tax-related data. For simplicity, we assume that server-2 can only store tax related data for one employer per person (or per TFN), and each person's tax record can hold tax data items for up to 26 payroll records only - This implies that his/her wages were paid on bi-weekly basis.
The PIDT uses taxpayer's TFN to identify a user's tax-related data records. The payroll-tax related data records in PIDT are read-only. The server-1 cannot make any change to the data records in the PITD database.

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

(6). The TRE client should do necessary data validations, for example, a taxpayer' tax withholding should not be greater than his/her taxable income, etc.

Recommended procedure for coding:

You should complete the project in two phases. The first phase is to implement a simple *"two-tier" interaction* between the client application and server-1 application/s. The second phase is to upgrade the *"two-tier" interaction* to a *"three-tier" interaction* by adding server-2 that acts as a database server (or data center). The database server can process some requests/services from the server-1 application/s that was completed in the first phase.

Phase 1: the *"two-tier" version implementation* (*30 marks*):

Refer to the attached Python-like pseudocode in Appendix A that defines a method *Average*. It is an application that runs on a local machine. The *Average* method takes a list of integers as input, calculates, and finally returns the average (as a real number of double type). The main function tests the *Average* method with a list of students' unit marks, and displays the result (i.e., average of the units) on the screen.

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 to complete a mini programming project for a two-tier client-server application. That is, in this phase, it is assumed that the users do not have TFNs, therefore it is not necessary to consider any functionalities listed in item (4) in the PITRE behavior and requirements section (see above).

You may start by modifying (or converting) the above application/code to a simple client-server application for the *PITRE*. That is, the client makes RMI to a remote application/method, say, *Estimator*, which is located in the server-1 side, and is used to estimate a user's (or taxpayer's) tax return estimate based on the data entered.

Recommendations & restrictions to Phase-1 programming:

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

(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 programming 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 Phase-2) could be simplified to using a set of arrays to mimic a relational database.

(ii) Use *asynchronous communication* pattern (or a combination of the *synchronous* and *asynchronous communication* patterns) to implement the system. In this case, you can not only use RMI (or RPC) technique, but also all class libraries from the chosen programming environment and any third-party package/s to implement the application/s.
In this case, the corresponding third-tiered database server (to be implemented in Phase-2) must use (or connect to) a real/existing database server (e.g., SQL server, MySQL, or Oracle server, etc.).

Phase 2: The *"three-tier" version implementation* (*15 marks*):

This phase should be started after you completed the "two-tiered" client-server application in Phase 1. This phase consists of two sub-tasks:

(1) Using various data to test the client/server application completed in *Phase 1*.
(2) If a user (i.e., a taxpayer) has a TFN, he/she may use his own payroll-tax data stored in the *PITD* database to do the tax-return estimate, rather than entering his/her tax-related data via the interface at the TRE client.
In this phase, you are requested to create a *third-tiered* server, a simplified database server to mimic the server of the *PITD* database. The database stores taxpayers' pay-roll tax related data of the last financial year. A taxpayer's pay-roll tax related data is a list of up to 26 pay-roll records, each may include the pay period, date of the payday, gross wage/salary, tax levied on the gross wage/salary, and net wage/salary of each pay cycle, etc. You should also make necessary extensions to the client and server-1 applications that were implemented in Phase-1 so as to complete the three-tiered system.

The server-1 and server-2 can communicate (say by RMI), however the client application has no direct access to server-2.

- Extension to the client-side applications:

Client application should be updated to allow a user to do the tax-return estimate by one of the two ways:

(i). The user enters his Person ID and a sequence of payroll tax records to do the tax return estimate as usual (i.e., the same as what was done in phase 1, note that users are not required to enter his/her TFN in the case), or
(ii). The user uses his/her tax-related data records to do the tax return estimate. In this case, the user should enter his/her Person ID and TFN (and other related information) for authentication purpose. The client submits a request to the server-

1, requesting the server-1 to use the user' tax-related data stored in the *PITD* (i.e., server-2) for the estimate.

- Extension to the Server-side applications:

(iii). Server-1 extension:
When a user wanted to use his/her tax-related data to do the tax-return estimate, Server-1 makes a request to server-2 (with the Person ID, and FTN as parameters), asking for the users' tax-related data. After receiving data from the server-2, it extracts related data and uses it to do tax-return estimate and then returns the result back to the client.
Necessary data validation should be done at server-1 side, if applicable.

Bonus Marks (up to 5 marks):

For data initialization, checking and verification purpose, you are encouraged to develop an additional mini database interface for server-2 that can be used to help database administrators (DBAs) to enter, delete, and update data stored in server-2. This is more important /convenient when your server-1 connects to a real database server (e.g., MySQL server, SQL server, Oracle server etc.).

- Develop a mini user interface (e.g., by using SQL or other development tool/s) for the database server (i.e., server-2) such that
  (i) it can display all available data records stored in *PITD* database.
  (ii) it allows users enter/delete/edit data records stored in the database.
  (Note: You should use an integer variable to keep track of the number of pay-roll tax records created so far. Be careful – Don't allow more than 26 data records for any taxpayer)
- Explain and extend to accommodate error messages, etc.
- Attach a simple *User Manual* showing how to install and run this user interface.
- Add a short video clip (as a part of the demo video) to demonstrate how you use the interface.

This additional work and effort to improve the program and make it more convenient 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 project score.

(Note: This bonus mark applies to the case where the project is completed by a group of two students or by an individual. Groups with three students will not get a bonus mark because this is a part of required tasks for groups of three).

Other basic requirements

- The application/system should include all required components.
- The observable behaviors of your application should be consistent with what is described.
- The application provides necessary error handling mechanisms.

- The application must be implemented using Python – If you got permission to use a different programming language, please use ONLY one single programming language, chosen from a list of o-o programming language set (e.g., Java, C#, C++), to implement all client and server applications. The only exception is that a SQL-compatible language can be used to implement the third-tier server application/s in Phase-2.
- The system must be runnable *off-the-shell*. That is, executable codes should be produced, and can be run in OS shell (e.g., Windows' Command Line prompt) or by double clicking on the executable in a folder. In other words, the final product of the project (i.e., the system developed) should be runnable independent of any programming environment.
- The project report should be well structured and informative (e.g., no more than 20 pages), and not contain codes of your project. All code files should be included, separately, in the submitted .zip file.

2. The written report and its possible structure (12 marks)

Refer to the **Format requirement of the Assignment report** in page 13. The *main body* should include (but not limited to):

i. (A brief) Introduction to 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. Test cases that can be used to test your system
vi. Example snapshots demonstrating application running status and input/outputs

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

3. The video demonstration and reflection (individual work, 40 marks)

This is an individual task (not a group task). Each team member of the group should prepare and submit their own version of video, seperately.
The video should cover two parts: the first part is a demonstration of the project completed by the group, and the second part is an individual's *reflection* on the project work.

(a) The video demonstration/presentation

This first video part 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 are:
(1) The duration of this part of the video should be between 5 and 10 minutes.

(2) Students Identity Verification (SIV): For academic integrity, a SIV is required for the video demonstration. When the video starts, you should take a few seconds to briefly introduce yourself while your computer camera captures your face, or your student ID.

(3) The demo should include
- a brief introduction of software used to develop the project;
- the architecture of the system completed;
- user authentication (and/or registration if applicable);
- the demonstration over some test cases:
  (i) the case/s for a user who has no TFN, e.g., a user enters his/her Person ID, and a sequence of valid payroll tax related data for tax-return estimate, where each payroll data item may be in a format of <net_wage, Tax_withheld> pair;
  (ii) the case/s for a user who has a TFN, e.g., a user enters a Person ID, TFN and any other information to do tax-return estimate using his/her payroll tax-related data stored in the PITD at server-2;
  (iii) the cases with some invalid data input (e.g., more than 26 payroll data items are entered, non-authenticated user trying to use the system, etc.), triggering the system to display error handling message/s, etc.

(4) Any special features that you want to show.

(b) The video reflection

This is the second part of the video. The duration of this video part should not be longer than 5 minutes.
In this video part you should explain your own contribution to the teamwork, provide a personal commentary of the teamwork. The following is a list of points that you may consider in your reflection:

- What was your project about?
- How successful was your team?
- What was your key role in completing this project?
- How did completion of this task link to your work in this unit or in other units in your study?
- What was your key takeaway from the teamwork/task?
- What did you learn from the process?
- What difficulties you (or your team) encountered during the project development and how these were overcome?
- Which parts of the tasks were the most challenging, if any?
- Which parts were done well, and which parts of the work could be improved if you are given a chance? etc.

This video part may be used to support an un-even mark distribution between team members of the group work if the project contributions made by team members are significantly different.

(Note: It is fine if you wish to separate the video in two short video clips, e.g., one for the project demo and the other for personal reflection. In this case, submit two separate video links instead of one)

4. The peer review (*optional – 0 mark*)

This is to grade performance for each team member, including yourself.

This task is *optional* – If you choose to do it, please download the *peer review* form template from Canvas, fill in the form and make necessary remarks/comments, and finally submit it.

Format requirement of the *written report*

Report content	**Cover/title page** Must show unit code/title, assignment title, assignment due date, student IDs and name/s of all team members, 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 placed 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 sections, each with an appropriate title) 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 to run your programs. • Test cases that your team members may use to demonstrate the project in the video demo/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, screen shots and diagrams). The text must use font **Times New Roman** and **be not smaller than 12pt**.

Submission requirements (3 marks)

- *Project submission* (one submission per team):
  This submission should include a written report, all implementation codes (source codes, and executables if applicable) and any additional documentation associated with the report/project (if any). This submission should be submitted by the team leader only (i.e., through the team leader's submission portal). The other team member/s should not submit duplicated assignment version.

  Submit the last project product only (that is, only submit the 3-tiered system, and do not submit the earlier version of the 2-tiered system that you completed in the first phase, unless you have not completed the 3-tiered system).
  The project submitted should be in one single compressed file (e.g., in .zip format), which contains 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 rename the .zip file in a format of

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

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

  <Student ID>_<last name>_< first name>_CSI3344D_A3.zip.

  > *As an example, if your team leader's ID is 12345678, with last name SMITH and fist name Ben, the team member's ID is 15555555 and his Last name is MAX, the submission file should be named 12345678_SMITH_ Ben_15555555_MAX_CSI3344_A3.zip.*
  > *If the assignment is done by Ben SMITH alone, the submission file should be named 12345678_ SMITH_Ben_CSI3344_A3.zip*

  Note that files found to contain 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.

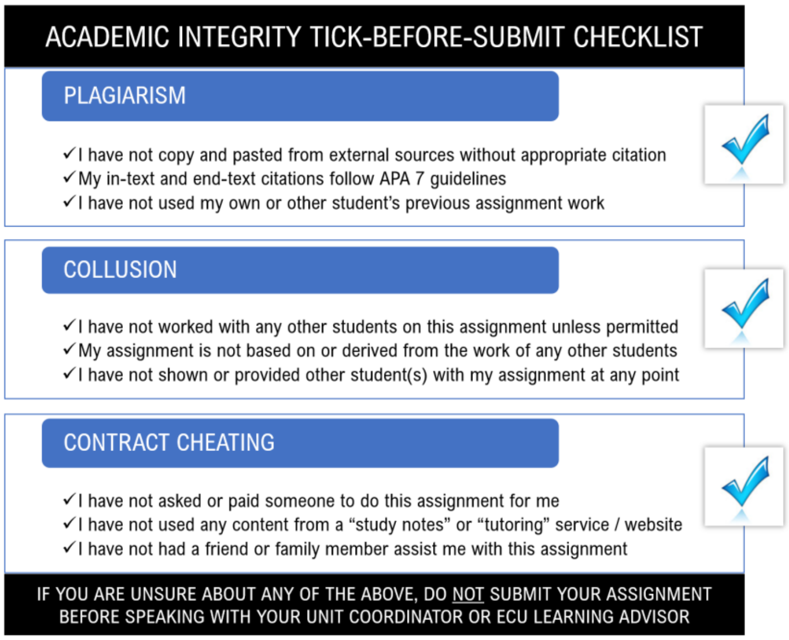
- *Video submission*:
  The submission should be a video link (or two video links if you separate the video in two video clips).

- *Peer review submission (optional):*
  Fill in the form and submit it (note that a form template is provided on Canvas).

  Note that separate submission links are available for each submission.

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:



ACADEMIC INTEGRITY TICK-BEFORE-SUBMIT CHECKLIST

PLAGIARISM

✓I have not copy and pasted from external sources without appropriate citation
✓My in-text and end-text citations follow APA 7 guidelines
✓I have not used my own or other student's previous assignment work

COLLUSION

✓I have not worked with any other students on this assignment unless permitted
✓My assignment is not based on or derived from the work of any other students
✓I have not shown or provided other student(s) with my assignment at any point

CONTRACT CHEATING

✓I have not asked or paid someone to do this assignment for me
✓I have not used any content from a "study notes" or "tutoring" service / website
✓I have not had a friend or family member assist me with this assignment

IF YOU ARE UNSURE ABOUT ANY OF THE ABOVE, DO NOT SUBMIT YOUR ASSIGNMENT BEFORE SPEAKING WITH YOUR UNIT COORDINATOR OR ECU LEARNING ADVISOR

Watch this video before submitting your assignment

Indicative Marking Guide:

	Criteria	Out of marks	You got
Implementation of the project	Phase 1: Two-tiered version completed + authentication, etc.	30	
	Phase 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?).
		12	

Submission	Report document presented as per format requirement, Files submitted as per submission requirement.	3	
Video demo &reflection	As per requirement	40	
Penalty	(possible Plagiarism or Late submission penalty)		
Total	Total mark of 100 (which is then converted to 50% of unit mark)	100 / (50%)	

Appendix A

Example Python-like pseudo-code (for reference only):

```
"""
File Average.py.
An exemplar Python code.
"""
import random

def  Average (lyst):
        #lyst: an array of unit scores, ended with -1, which is not a valid score
        #double average: average of all scores
        #Initialization phase
        total = 0        #total - sum of scores
        gradeCounter = 0 #gradeCounter: to track the number of scores in the array lyst[])
        #Processing phase
        n=len(lyst)
        i=0
        if (n ==0):
                return 0
        while lyst[i] != -1:
                gradeValue = lyst[i]
                total = total + gradeValue
                gradeCounter +=  1
                i +=1
        if (gradeCounter != 0 ):
                average = total / gradeCounter
                return average
        else:
                return 0

def main(average = Average):
    lyst = [ 10, 20, 30, 40, 50, 60 ,70, 80, 90, 100, -1]
    size = 10
    #for count in range(size):
     #   lyst.append(random.randint(1, 100 ))
    print(lyst)
    a = average (lyst)
    print("The average value is:", a)
    lyst1=[ ]
    while True:
        v = input("Please enter an integer as a unit mark (enter -1 to stop):")
        x=int (v)
        if x !=-1:
            lyst1.append(x)
        else:
            lyst1.append(-1)
            break
    print(lyst1)
    a = average (lyst1)
    print("The average value is: a = ", a)


if __name__ == "__main__":
    main()
```