TORRENS UNIVERSITY AUSTRALIA

ASSESSMENT 1 BRIEF

Subject Code and Title	PBT205—Project-based Learning Studio: Technology
Assessment	Prototype Development
Individual/Group	Group
Length	Three (3) prototypes (tasks) and a group report for each
Learning Outcomes	The Subject Learning Outcomes demonstrated by the successful completion of the task below include: a) Work cooperatively to plan, design, produce and manage a software prototype utilising advanced techniques and where applicable, emerging technologies; c) Investigate middleware and other technologies to assist in efficient and effective software development; d) Explore hardware and software limitations of various middleware solutions; and e) Demonstrate software engineering skills required in a software development team.
Submission	Due by 11.55 pm AEST/AEDT on the Sunday at the end of Module 6
Weighting	45%
Total Marks	100 marks

Task Summary

- THREE (3) separate tasks make up this assessment. Each of the tasks deals with different domains: 1) chatting; 2) trading; and 3) contact tracing.
- In each of the three (3) tasks, you will be required to get middleware up and running to enable the applications you develop to communicate with one another.
- You will develop command line–based applications that will do one or both of the following:
  - Subscribe to topics in the middleware; and/or
  - Perform behaviours based on messages received.
- You will publish to topics in middleware based on user input or start-up parameters. It is a prerequisite that:
  - You have access to a machine that is capable of running Docker or similar containerisation software (e.g., Podman). However, it should be noted that the instructions provided here are for Docker only.
- In relation to the middleware:
  - RabbitMQ is the prescribed middleware; and
  - You can reuse the middleware from the first prototype for all subsequent prototypes.

Please refer to the Task Instructions (below) for details on how to complete this task.

Context

The assumptions for all of the following three (3) tasks is that you already have some basic programming knowledge from prior courses and are able to use some general-purpose language (e.g., C++, Java or JavaScript).

Some prior knowledge of basic programming and general purpose language (i.e., C++, Java or JavaScript) will prove advantageous in completing the task requirements.

In the course of completing all **three (3)** tasks, you will need to divide the work appropriately among your team members to complete the assignments. The areas of focus may include: eliciting requirements from the instructor, understanding the chosen middleware technology being used (e.g., RabbitMQ), determining the choice of serialisation (e.g., JSON), defining contracts and protocols between systems, designing GUIs and planning for the demonstration.

> Note: The instructor may choose to switch or fix some of these technology decisions. (e.g., this may occur if the company you joined already has a middleware solution).

It is also likely and anticipated that you will need to develop parts of the solution(s) independently and then test them against one another (i.e., you will need to engage in integration testing). Teams should recognise that integration is a key area of risk (from the first or second prototype) and allow ample time for it in subsequent assignments.

Hopefully, you will quickly realise that you will be developing many standalone command-line applications that simply talk to the middleware. This realisation will assist you to identify useful abstractions to carry forward into subsequent assignments.

Setup Instructions

Note: These instructions are common to all themes/assignments.

You will need access to a machine capable of running **Docker**.

First, you will need to get your middleware up and running. The solution you will use to do this is **RabbitMQ**—it is free and easily launchable using **Docker**. Use the following command to launch an instance of RabbitMQ:

```
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

(Available at from https://www.rabbitmq.com/download.html as at 24/June/2020).

Task 1 Instructions—Chatting Application

Specification

This basic command-line chat application (app) allows multiple chat participants to send and receive messages from a room.

Tasks

- Set up your middleware.
- Set up a new topic on the middleware named ‘room’ (or something equivalent).
  - (Note: Keep this portion flexible and re-usable. You will need to create topics for every subsequent assignment).
- Write a ‘chat’ command-line application:
  - Start-up arguments:
    - A username;
    - The middleware endpoint (TCP port).
  - Behaviour:
    - Upon starting up, it should subscribe to the above topic;
    - Any messages that other users post should be printed to the console, along with the username of whoever posted the message;
    - Any messages entered by the user (terminated by a carriage return) should be sent to the chat room for the other participants to see.

Final Product

- Expose the chat application over a simple GUI interface in which multiple users can log in and join a ‘room’ and begin chatting.
- Extend the application to allow for multiple rooms, so that a user can select the room they wish to join upon logging in.

Task 2 Instructions—Trading System

Specification

You will be modelling a very simple market and exchange in which there is only one stock (XYZ Corp). All orders must have a desired price expressed on the order (this should be distinguished from ‘market orders’ in industry).

This will be enabled by two (2) actors:

1. The Exchange: There can only be one (1) exchange. It is responsible for matching orders from traders against one another and publishing ‘done trades’ along with the price they were traded at; and
2. The Trader: There can be multiple traders, each with a unique identifier.

Tasks:

- Create two (2) topics: ‘Orders’ and ‘trades’.
- Write a ‘sendOrder’ command-line application.
    - Start-up arguments:
        - A username;
        - Middleware endpoint;
        - A SIDE (‘BUY’ or ‘SELL’);
        - A QUANTITY (an integer). For this assignment, the quantity will be fixed at 100 shares on every order;
        - A price (a double).
    - Behaviour:
        - This program should simply start up, connect to the endpoint, submit its order to the ‘orders’ topic and then exit immediately.
- Write an ‘exchange’ command-line application:
    - Start-up arguments:
        - Middleware endpoint.
    - Behaviour:
        - Subscribe to the ‘orders’ topic;
        - Whenever it receives an order:
            - If there is a matching opposite-side order with an acceptable price for this order (the buyer’s price is acceptable to the seller or vice versa), then:
                - Take that order out of the order book and publish trade information to the ‘trades’ topic;
            - If there is no suitable order, then add the order to the order book.
- You may find it useful to explore the tooling that the chosen middleware solution offers for printing the contents of a topic.
- The team is free to choose the most suitable data structure for the order book.

Final Product

- Expose a simple GUI interface that displays the latest price at which XYZ Corp traded.
- Extend the functionality to allow for multiple stocks.

Task 3 Instructions—Contact Tracing

Specification

You are modelling a given environment as a two-dimensional system of coordinates (like a very large chess board). The virtual size of the board is 10 x 10, but should be configurable for up to 1,000 x 1,000.

You and your colleagues will start at a randomised position in this environment and can move one square only in any direction (similar to a King’s movement in chess). Each person will have a unique identifier (for simplicity, you can simply use a first name as an identifier in this environment (provided that the name is unique) to monitor the activity.

Your role is to list colleagues that you have come into contact with (i.e., other people who have occupied the same square at the same time).

Unlike chess, multiple people can occupy a single square simultaneously and there is no concept of ‘rounds‘ (in which each player is allowed to make a single move, turn by turn). Further, each person can frequent the same square as often as they wish and can move to new squares as frequently as they wish.

Tasks

- Set up two (2) middleware topics: ‘position’ and ‘query’.
- Write a ‘tracker’ command-line application:
    - Start-up arguments:
        - Middleware endpoint.
    - Behaviour:
        - Subscribe to the ‘position’ topic:
            - Keep a ‘view of the environment’ detailing each person’s current position;
            - If two (2) people occupy the same position, log this fact in a suitable data structure.
        - Subscribe to the ‘query’ topic:
            - When a person identifier is placed into the ‘query’ topic, respond to the ‘query-response’ topic with all the names that person came into contact with onto the console in reverse-chronological order.
- Write a ‘person’ command-line application:
    - Start-up arguments:
        - Middleware endpoint;
        - Person identifier;
        - Movement speed (team’s discretion; that is, ‘moves per second’ or ‘fast/slow’ or other descriptor).
    - Behaviour:
        - Connect to the middleware endpoint;
        - Communicate initial (randomised) position to the ‘position’ topic, along with the person’s identifier;
        - Continually make a move in a random direction (one square at a time) and publish that move to the ‘position’ topic. (Movement speed should accord with the start-up argument provided).
- Write a ‘query’ command-line application:
    - Start-up arguments:
        - Middleware endpoint;
        - Person identifier.
    - Behaviour:
        - Connect to the middleware;
        - Publish the person identifier provided at start-up onto the ‘query’ topic;

        - Await the response on the ‘query-response’ topic from the tracker and print that response to the console, then exit.
- Introduce a process representing a person that starts with a position and randomly moves ‘squares’ once every ‘n’ seconds (where ‘n’ is configurable). By ‘moving’, it is simply publishing a new (x,y coordinate)
- All coordinates are represented by numbers (x,y). You must cater for boundary conditions, such as a person moving off the board.

Final Product

- Provide a GUI that:
    - Gives a visual representation of the ‘environment’; and
    - Allows a user to query a person identifier and to see with whom they have come into contact.
- Extend the solution to work for boards/environments of different sizes.

Referencing

It is essential that you use APA style to cite and reference your research. For more information on referencing, visit our Library website at https://library.torrens.edu.au/academicskills/apa/tool.

Submission Instructions

Submit this task via the Assessment link in the main navigation menu in PBT205—Project based Learning Studio: Technology. The learning facilitator will provide feedback via the Grade Centre in the LMS portal. Feedback can be viewed in My Grades.

The following page contains a guide to constructing rubrics. The learning facilitator can adapt or modify the rubric to suit the assessment outcomes and in accordance with Torrens University assessment policy.

Academic Integrity Declaration (Student Undertaking)

We declare that the work we are submitting for this assessment task is our own work. We have read and are aware of the Torrens University Australia Academic Integrity Policy and Procedure, viewable online at http://www.torrens.edu.au/policies-and-forms.

We are also aware that we need to keep a copy of all submitted material and any drafts and we agree to do so.

TORRENS UNIVERSITY AUSTRALIA

Assessment Rubric

Assessment Attributes	**Fail (Yet to achieve minimum standard) 0–49%**	**Pass (Functional) 50–64%**	**Credit (Proficient) 65–74%**	**Distinction (Advanced) 75–84%**	**High Distinction (Exceptional) 85–100%**
Knowledge and understanding of Functional middleware set up or reuse **10%**	Demonstrates a partially developed understanding *Functional middleware set up or reuse*	Demonstrates a functional knowledge of *Functional middleware set up or reuse*	Demonstrates proficient knowledge of *Functional middleware set up or reuse*	Demonstrates advanced knowledge of *Functional middleware set up or reuse*	Demonstrates exceptional knowledge of *Functional middleware set up or reuse*
Knowledge and understanding of Complied code - *free from errors* **30%**	Demonstrates a partially developed understanding Complied code - *free form errors*	Demonstrates a functional knowledge of Complied code - *free form errors*	Demonstrates proficient knowledge of Complied code - *free form errors*	Demonstrates advanced knowledge of Complied code - *free form errors*	Demonstrates exceptional knowledge of Complied code - *free form errors*
Knowledge and understanding of assessment requirements, associated outputs and arguments implemented as specified in the task **40%**	Demonstrates a partially developed understanding of main topics, arguments implemented and outputs as specified in the task(s)	Demonstrates a functional knowledge of main topics, arguments implemented and outputs as specified in the task(s)	Demonstrates proficient knowledge of main topics, arguments implemented and outputs as specified in the task(s)	Demonstrates advanced knowledge of main topics, arguments implemented and outputs as specified in the task(s)	Demonstrates exceptional knowledge of main topics, arguments implemented and outputs as specified in the task(s)

TORRENS UNIVERSITY AUSTRALIA

Assessment Attributes	Fail (Yet to achieve minimum standard) 0–49%	Pass (Functional) 50–64%	Credit (Proficient) 65–74%	Distinction (Advanced) 75–84%	High Distinction (Exceptional) 85–100%
Teamwork and collaboration between members **Percentage for this criterion = 20%**	Does not participate effectively in a team environment. Team goals compromised and milestones impacted due to lack of effective input .	Participates effectively in teams. Identifies team and individual goals, tasks, responsibilities and schedules. Contributes to group processes. Supports the team.	Contributes to small group discussions to reach agreement on issues. Works together with others towards shared goals. Renegotiates responsibilities to meet needed change.	Understands group dynamics and team roles. Facilitates team development. Renegotiates responsibilities, tasks and schedules to meet needed change.	Builds team's identity and commitment. Leads teams. Evaluates teams' outcomes. Implements strategies for enhancing team effectiveness.

The following Subject Learning Outcomes are addressed in this assessment	
SLO a)	Work cooperatively to plan, design, produce and manage a software prototype utilising advanced techniques and where applicable, emerging technologies.
SLO c)	Investigate middleware and other technologies to assist in efficient and effective software development.
SLO d)	Explore hardware and software limitations of various middleware solutions
SLO e)	Demonstrate software engineering skills required in a software development team.