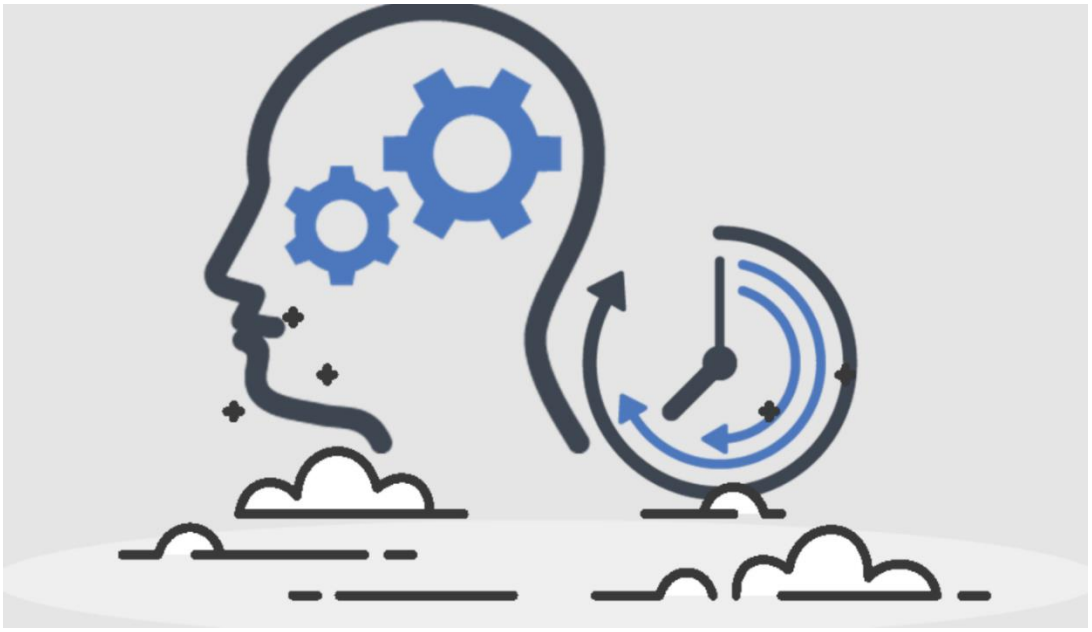
Prototype Report



Group 3

Akshdeep Sandhu (A00082999)

Niroj Khadka (A00061981)

Raman Shrestha (A00078894)

Razat Siwakoti (A00043365)

Project Based Technology (PBT205)

Torrens University

Ultimo, NSW, Australia

Date: 12/08/2023

Abstract

A four-person team's evaluation of a number of incrementally improved prototypes is presented in this report. The prototypes use RabbitMQ to create a three working prototypes: Chatting, Trading, and Contact Tracing application. The prototypes successfully meet the assessment criteria and demonstrate the team's mastery of middleware usage and application development.

Table of Contents

Introduction

Docker, RabbitMQ and Python are some of the most powerful technologies that can be used together to create variety of application. The ability of Docker to build isolated environments allows for consistent deployment across several systems (*What Is Docker? | AWS*, 2023). Real-time data sharing is made possible by RabbitMQ's middleware features, which promote seamless connection between application components (*RabbitMQ*, 2017). The team is able to create capabilities ranging from messaging and trading logic to player movement simulation and interaction monitoring because to Python's versatility and numerous libraries.

In this project, three different prototypes were created: Chatting Application, Trading System and Contact Tracing by group of 4 people. The team skilfully uses mentioned tools in efficient teamwork to produce functioning prototypes that correspond to evaluation requirements. The study evaluates the group's thorough knowledge of middleware configuration, functional coding, and collaboration, which finally resulted in the creation of diverse apps that demonstrate their mastery of combining Docker, RabbitMQ, and Python.

Prototype 1: Chatting Application

The first prototype aims to provide a straightforward command-line chat programme using RabbitMQ as the message communication middleware. In a chat room, the application enables many users to send and receive messages.

Prototype 2: Trading System

A trade and order matching system are included in Prototype 2 to advance the application. The programme analyses buy and sell orders, makes an effort to match them, and logs trades. To construct a more complex financial trading environment, the codebase expands upon the earlier prototype.

Prototype 3: Contact Tracing

In the third Prototype, player movement simulation on a grid and player interaction tracking is introduced. The programme mimics player motions, keeps track of their locations, and monitors player interactions based on proximity.

Knowledge and Understanding of Functional Middleware Set Up or Reuse

The team showed remarkable mastery of using RabbitMQ to set up functioning middleware in each prototype. To start a RabbitMQ instance, Docker was used.

In the first prototype, middleware was built, allowing for effective message transfer between users as the foundation of communication. The team uses Docker to segregate RabbitMQ instances inside of containers, guaranteeing consistency between deployments. Using RabbitMQ's publish-subscribe paradigm, the team expertly creates communication channels that enable real-time message flow between users. The Python scripts' successful communication with RabbitMQ channels demonstrates their mastery of messaging middleware integration.

In the second prototype, RabbitMQ was reused from the first prototype encapsulating the message middleware efficiently. The implementation of order processing and matching logic demonstrates the team's comprehension of RabbitMQ's message queuing processes. This level of competence enables effective order book and trader communication.

In the third prototype, the group successfully creates communication channels using RabbitMQ, allowing for effective player interaction recording. Python scripts exhibit a sophisticated command of middleware integration as they deftly exchange interaction data with RabbitMQ channels. The team's use of interaction tracking and player movement simulation highlights their expertise in using middleware for practical applications. The contact tracing mechanism prototype essentially shows off the team's skill at setting up functional middleware for challenging simulation and tracking jobs.

Knowledge and Understanding of Compiled Code

Each Prototype offers code samples in Python Programming Language, which is interpreted, but there is no direct evidence of compiled code. The first prototype focuses on middleware setup, message exchange, and basic functionalities, while the second one centres on the functional implementation of the trading system, and the third prototype emphasizes the functional implementation of player movement and interaction tracking.

Knowledge and Understanding of Assessment Requirements:

In each prototype, setting up RabbitMQ using Docker for middleware communication was the fundamental part. The prototype aligns with the assessment requirements by:

Prototype 1: Chatting Application

- Creating a command-line chat programme that allows users to communicate in a chat room
- Using threading to implement sender and consumer functionalities to allow simultaneous sending and receiving
- Establishing a logical framework for user interactions and communication processing

Prototype 2: Trading System

- Expanding the application to include an interface for matching orders and trading
- Implementing logic into structure for order submission and processing
- Exemplifying order matching using simple conditions
- Keeping trade records and showing trade data

Prototype 3: Contact Tracing

- Utilising random directions to simulate player movement on a grid
- Tracking the position of players and updating the canvas display accordingly
- Installing the Tracker class to capture player interactions based on proximity
- Upon user request, displaying tracked interactions in a separate window

Teamwork and Collaboration

The teamwork and collaboration between the members are evident in the code's organization and functionality (GitLab, 2023). The group has a common knowledge of RabbitMQ configuration and message ideas. Effective communication is demonstrated by the proper division of concerns into separate files (chat.py and rabbit.py). The team improved the functionality of the system in the second and third prototypes by utilising their understanding of RabbitMQ gained from the first. Effective teamwork and collaboration are demonstrated by the way the code is organised and by how the 'Tracker' class is used.

Conclusion

The team's combined efforts lead to a number of prototypes that demonstrate their knowledge of functional programming, middleware, and teamwork. In all prototypes, RabbitMQ served as an efficient communication hub for message delivery. The team's ability to gradually add new features on top of existing ones exemplifies its cooperation and understanding.

References

GitLab. (2023, April 5). *What are software team collaboration best practices?* Gitlab.com; GitLab. https://about.gitlab.com/topics/version-control/software-team-collaboration/

RabbitMQ:— RabbitMQ. (2017). Rabbitmq.com. https://www.rabbitmq.com/

What is Docker? | AWS. (2023). Amazon Web Services, Inc. https://aws.amazon.com/docker/#:~:text=Docker%20is%20a%20software%20platform,tools%2C%20code%2C%20and%20runtime.