Orange and white text on a black background

Description automatically generated

|  |  |
| --- | --- |
|  | **ASSESSMENT 1 – Group Assessment** |
| **Student Name** | Mohammaduzair Mohammadyahya Shaikh | A00126661 |
| **Student Name** | Mohammed Isa | A00118725 |
| **Subject Code and Title** | PBT205—Project-based Learning Studio: Technology |
| **Assessment** | Prototype Development |
| **Learning Facilitator** | Nyasha Shoniwa |

# Task – 1 Console-Based Chat Application

### Introduction

Greetings for our report about our chat application based on terminals! This project demonstrates the process of creating a basic but successful chat platform with Javascript and RabbitMQ. The tool enables users to chat instantly via a command-line interface, which is ideal for understanding message brokering and real-time communication.

1. Application Setup
   1. Prerequisites

Before starting the setup process, ensure that you have the following items installed:

* Node.js is necessary for our chat application to function properly.
* We will employ Docker to operate RabbitMQ, the core foundation of our messaging platform.
  1. Setting Up RabbitMQ

Configuring RabbitMQ is simple. Simply adhere to these instructions:

1. Get Docker: If you have not yet done so, get and install Docker from the official Docker website.
2. Start the RabbitMQ container by entering the following command in your terminal:

docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management

This instruction will launch a RabbitMQ container with a user-friendly management interface, simplifying tasks.

1. To verify if the RabbitMQ is running or not, to check that we can execute a simple command-prompt

(docker ps)

1. Access RabbitMQ management interface by visiting http://localhost:15672 on your web browser. Utilize the standard login credentials:

Username: guest

Password: guest

* 1. Configuring The Application

Now we must configure our application

**Update `. env` File**: Create a `. env` file in the project directory with the following content:

**RABBITMQ\_URL=amqp://localhost:5672**

**RABBITMQ\_EXCHANGE=chat\_room**

### Now Run and Test the Application

To test and run our application, we must open our terminal or command and execute the following code line  
Terminal: npm install

Terminal: node chat\_app.js <username>

\*In place of <username>, we must write the actual username

### How the Chatting Application Operates.

So, we will see that how our chat application works

1. Using the URL we provide, the application first establishes a connection to RabbitMQ.
2. async function connectToRabbitMQ() {
3. try {
4. const connection = await amqp.connect(config.rabbitMQ.url);
5. console.log(kleur.green('Connected to RabbitMQ'));
6. return connection;
7. } catch (error) {
8. console.error(kleur.red('Failed to connect to RabbitMQ:'), error.message);
9. process.exit(1);
10. }
11. }
12. Channel and Queue Creation: It establishes a channel and a queue in which all messages are sent out loud.
13. async function setupChannel(connection) {
14. try {
15. const channel = await connection.createChannel();
16. await channel.assertExchange(config.rabbitMQ.exchange, 'fanout', { durable: false });
17. console.log(kleur.green('Channel and exchange setup complete'));
18. return channel;
19. } catch (error) {
20. console.error(kleur.red('Failed to setup channel:'), error.message);
21. process.exit(1);
22. }
23. }
24. Managing Messages: Messages sent by users are immediately broadcast to the exchange so that anyone who has subscribed can read them.
25. async function publishMessage(channel, username, message) {
26. try {
27. await channel.publish(config.rabbitMQ.exchange, '', Buffer.from(`${username}: ${message}`));
28. } catch (error) {
29. console.error(kleur.red('Failed to publish message:'), error.message);
30. }
31. User Interaction: You may type messages and view incoming ones in real time with this program because it uses read line for user input.
32. async function startConsuming(channel, queueName, messageHandler) {
33. try {
34. await channel.consume(queueName, messageHandler, { noAck: true });
35. console.log(kleur.green('Started consuming messages'));
36. } catch (error) {
37. console.error(kleur.red('Failed to start consuming:'), error.message);
38. process.exit(1);
39. }
40. }
41. Benefits of the Chat Application

* This Chatting Application has several benefits which are listed below
* Real-time communication: It enables instant message sending and receiving.
* Lightweight and Efficient: It doesn't need a lot of resources because it is terminal-based.
* Scalable Architecture: It can effortlessly manage numerous users because RabbitMQ is at the heart of it.
* Learning Opportunity: This is an excellent approach to understand real-time applications and message brokering ideas.

### References:

RabbitMQ Documentation Team. (n.d.). *RabbitMQ documentation* [Website]. Retrieved from <https://www.rabbitmq.com/docs>

[Node.js](https://node.js/)Foundation. (n.d.). [*Node.js*](https://node.js/)*documentation* [Website]. Retrieved from <https://nodejs.org/en/docs/>

Docker Inc. (n.d.). *Docker documentation* [Website]. Retrieved from <https://docs.docker.com/>

# Project Charter Document

**Project Overview**

**Project Name:** Console-Based Chat Application

**Project Manager:** Mohammaduzair & Mohammed Isa

**Start Date: 3 October**

**End Date: 26 October**

### Purpose

Using Node.js and RabbitMQ, the project intends to develop a console-based chat application that showcases message brokering and real-time messaging via a terminal interface.

### Objectives

* Create a working Node.js chat application.
* For broadcasting and message handling, integrate RabbitMQ.
* Allow users to communicate with one another in real time by using terminal commands.

### Scope

Within scope are the development of the chat application, RabbitMQ setup, necessary configuration creation, and console testing.

### Deliverables

* Console-based chat application code.
* Documentation detailing setup, functionality, and user instructions.

### Stakeholders

Mohammaduzair: Project Manager and Reviewer

Mohammed Isa: Developer and Designer

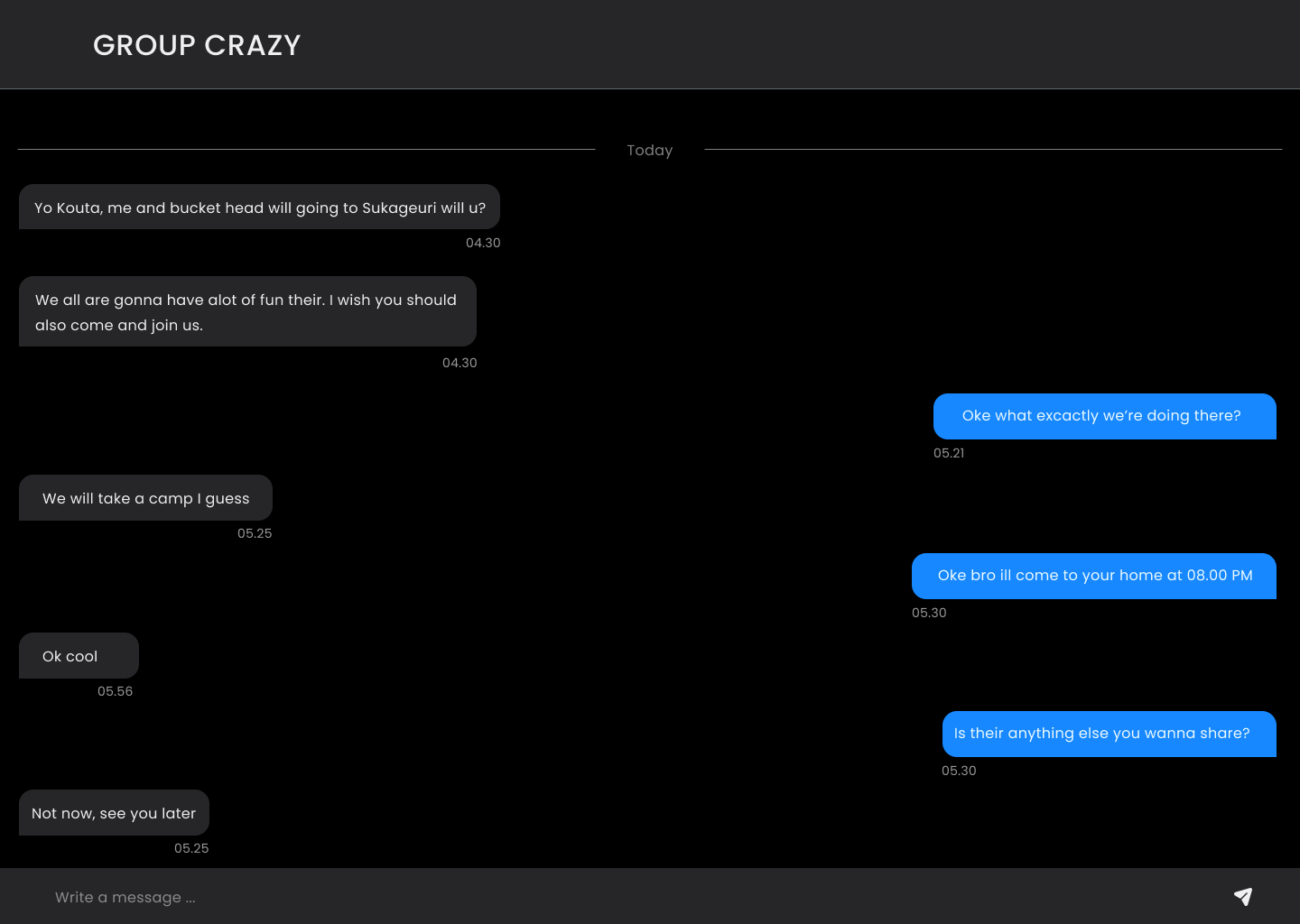
### Risks

* Dependency Issues: Possible problems setting up RabbitMQ. Mitigation: Verify that environment and Docker variables are set up appropriately.
* Connectivity Issues: RabbitMQ may fail to connect if the server is not running. Mitigation: Document steps to troubleshoot RabbitMQ setup.

### Team Members

|  |  |
| --- | --- |
| Name | Role |
| Mohammaduzair Mohammadyahya Shaikh | Project Manager and Reviewer |
| Mohammed Isa | Developer and Designer |

Here are some screenshots of the output and our UI designs which we prepared:



A screenshot of a computer

Description automatically generated