Problem statement: CHAT APPLICATION USING RPC.

Group Details:

1. Padshah Rohan Chirag (2022H1030121P)
2. Rukhsar Parveen (2022H1030122P)

Overview

The objective of this project is to demonstrate the use of RPC using any industry standard RPC frameworks.We have created a chat application that will involve a single server with one or more clients connecting to the server and communicating with each other using simple text messages.

Basic Features:

1. Registration for new users.
2. Login for existing users.
3. Allows 1 to 1 chat between users where client and server are communicating using gRPC framework.

Advanced Features:

1. Group Chat **:** Ability to create a Group by selecting multiple users and chat amongst the group members (multi-cast).

Why gRPC?

1. A gRPC messages are serialized with ProtoBuf, a light weight message format. A gRPC message is always smaller than an equivalent JSON message.
2. gRPC has both unidirectional and bidirectional streaming support.

Tech Stack:

1. RPC Framework - gRPC
2. Frontend - AngularJS,Bootstrap v4, gRPC angular dependency, protobuf google dependency, proto compiler dependency
3. Backend - Java 11, Spring boot, gRPC spring boot dependency, MS SQL Database
4. Docker containers
5. Envoy proxy server

Server Operating Modes: Iterative vs Concurrent

We are using the tomcat server to deploy the backend application. Once the server starts the default configuration for a maximum number of connections allowed and maximum request processing threads allowed as mentioned in the documentation are 8192 and 200 respectively. Which means the server is by default **concurrent** in nature.

From the documentation:

Each incoming, non-asynchronous request requires a thread for the duration of that request. If more simultaneous requests are received than can be handled by the currently available request processing threads, additional threads will be created up to the configured maximum (the value of the maxThreads attribute). If still more simultaneous requests are received, Tomcat will accept new connections until the current number of connections reaches maxConnections. Connections are queued inside the server socket created by the Connector until a thread becomes available to process the connection. Once maxConnections has been reached the operating system will queue further connections. The size of the operating system provided connection queue may be controlled by the acceptCount attribute. If the operating system queue fills, further connection requests may be refused or may time out.

Hence if we want to run the application in **Iterative mode we need to limit the number of connections to 1 and number of maximum threads to 1**

The same is done in the code and would be decided based on the arguments passed.

Iterative Mode:

1. **Frontend: npm start --server_ip=IP --is_iterative=true**
2. **Backend: gradlew bootRun --args='server=iterative;ip=IP'**

Concurrent Mode:

1. **Frontend: npm start --server_ip=IP --is_iterative=false**
2. **Backend: gradlew bootRun --args='server=concurrent;ip=IP'**

Problems observed while running server in Iterative Mode :

In general any chat-application is a real time interactive application, which means as a user we want to communicate in real time. Now when we start the server in iterative mode, number of

connections and threads are limited to 1, which means that only 1 client request would be served at any point of time, thereby defeating the purpose of real time communication.

More over if the application runs in concurrent mode, the client will have a open long lived stream which is enough to support real time communication, while in case of iterative the stream will close immediately and the client needs to continuously poll the server, which is an overhead on the server. This also means that the request will get queued and may be served at a later point of time , due to number of requests received by the server. The config of same is shown below:

```
Properties properties = new Properties();
if (serverType.equalsIgnoreCase( anotherString: "iterative")) {
    System.out.println("Starting in iterative mode");
    properties.put("server.tomcat.threads.max", "1");
    properties.put("server.tomcat.max-connections", "1");
    properties.put("myserver.iterative", true);
    properties.put("server.address", ip);
} else {
    System.out.println("Starting in concurrent mode");
    properties.put("server.tomcat.threads.max", "200");
    properties.put("server.tomcat.max-connections", "8192");
    properties.put("myserver.iterative", false);
    properties.put("server.address", ip);
}
```

NOTE: So even though iterative mode is supported, it won't serve the purpose of the chat application and application may not work as expected, hence it is advisable to run the server in concurrent mode only

Why envoy proxy is required?

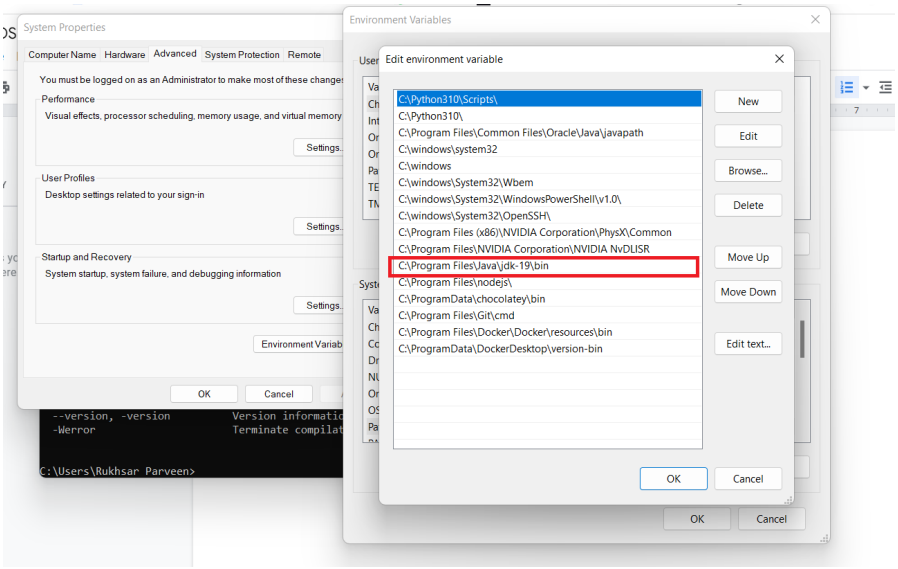
1. gRPC uses HTTP v2 under the hood while all the browsers use HTTP v1.x to communicate with backend servers. Hence there is no way a browser would be able to communicate directly with the gRPC server.In order to accomplish this we need a proxy

server that can convert and redirect the HTTP v1.x calls to HTTP v2 calls as mentioned in the documentation. We are using envoy docker image/container for the same

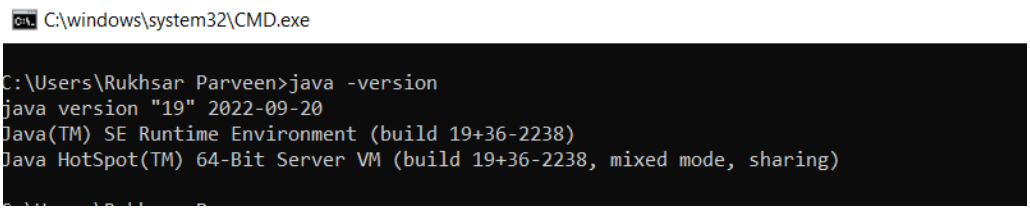
Softwares Required:

NOTE: The following setup is tried and tested on WINDOWS OS, in order to run it on LINUX OS it will need some extra adjustments.Please run exact commands to avoid failures.

1. **Java**
    a. Click to download the Java 11 version. Unzip the file on completion and add a new entry to PATH environment variable.



    b. Once installed then run the **java -version** in cmd and it should result as below.



2. **Docker**
    a. Go to this page and download the windows docker installer.Run the installer to install the docker.

b. **NOTE**: Once the docker installation is complete, it will prompt to activate **WSL 2** on your machine**,** follow the steps mentioned and restart the system. OR you can download it from here and run the installer.
c. Once docker is installed Open command prompt and run **docker -version**

```
C:\Users\Rukhsar Parveen>docker --version
Docker version 20.10.17, build 100c701
```

3. **Node JS**
   a. Download the installer from here and execute the installer. After installation is complete go to command prompt and run **node --version** which should show the following:

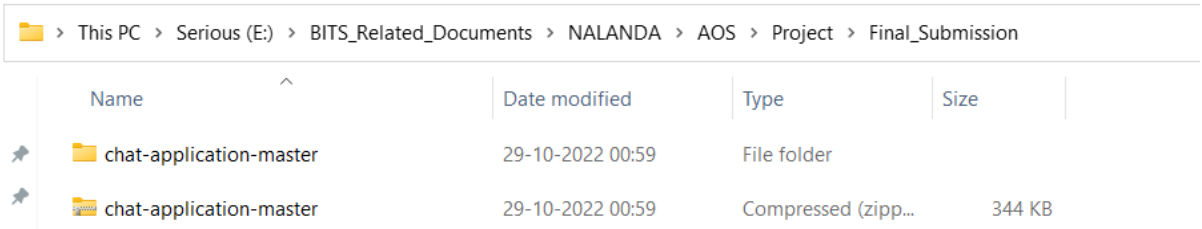
```
C:\Users\Rukhsar Parveen>node --version
v16.17.1
```

   b. Now run the **npm --version** which should show the following:

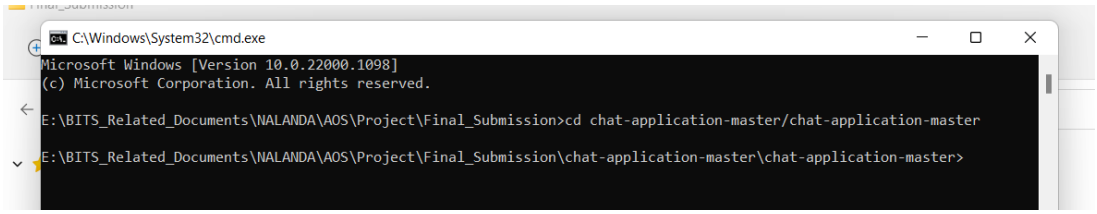
```
C:\Users\Rukhsar Parveen>npm --version
8.15.0
```

Download the source code

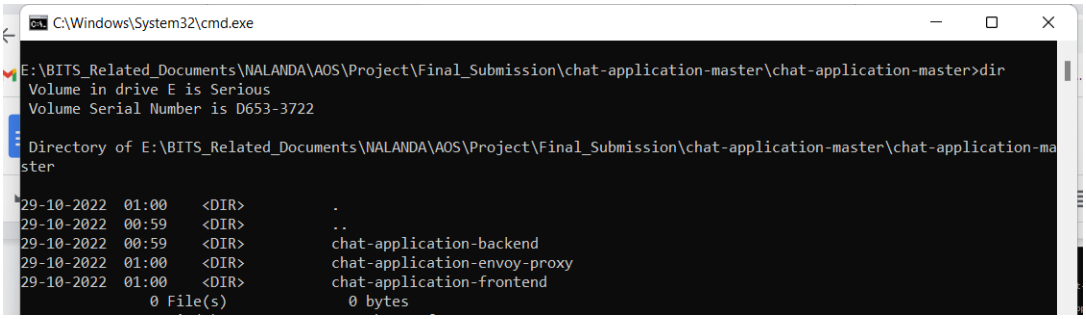
1. Download and extract the zip to get the source code. Should look like below:



2. Run command promt here inside this directory.
   a. Run **cd chat-application-master/chat-application-master**

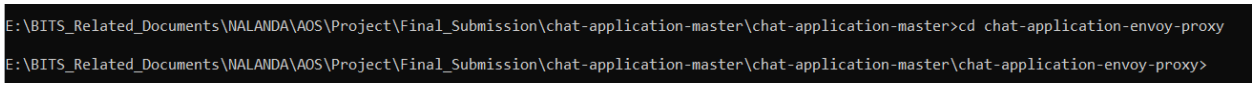


3. The folder would contain 3 subfolders
    a. chat-application-backend
    b. chat-application-frontend
    c. chat-application-envoy-proxy

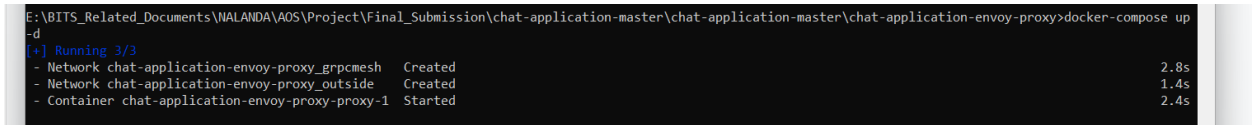


Run the application

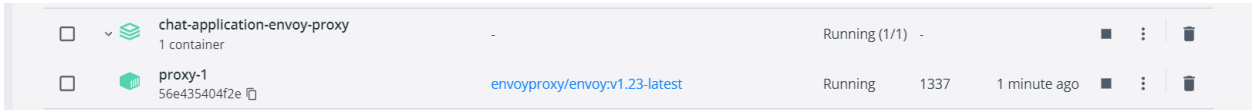
1. Continuing the above steps: Run **cd chat-application-envoy-proxy**



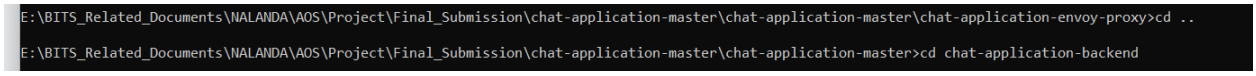
2. The first step is to run the docker-compose file to start proxy server as mentioned above using the following command: **docker-compose up -d**



To verify open Docker Desktop from start menu. Once its running it would look like this



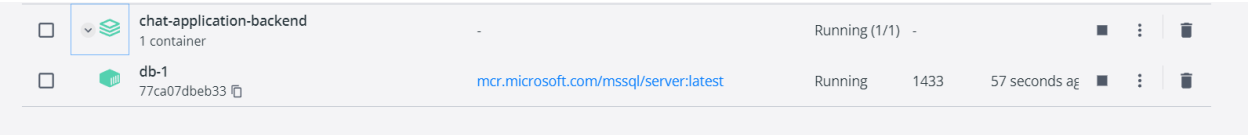
3. Run **cd ..** and move to parent directory and move to **cd chat-application-backend**



4. Run **docker-compose up -d** to start the database.

```
E:\BITS_Related_Documents\NALANDA\AOS\Project\Final_Submission\chat-application-master\chat-application-master\chat-application-backend>docker-compose up -d
[+] Running 2/2
 - Network chat-application-backend_default  Created                                                    0.8s
 - Container chat-application-backend-db-1   Started                                                    2.0s
```

Verifying using docker desktop



5. Run ipconfig and note down the LAN wifi IP address as shown in the below screen shots.

```
E:\BITS_Related_Documents\NALANDA\AOS\Project\Final_Submission\chat-application-master\chat-application-master\chat-application-frontend>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : bits-pilani.ac.in
   Link-local IPv6 Address . . . . . : fe80::49c5:637f:bc06:7dbe%20
   IPv4 Address. . . . . . . . . . . : 172.17.87.124
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Default Gateway . . . . . . . . . : 172.17.86.1
```

6. Let's run the backend application in **Concurrent Mode**. Open a new command prompt from the folder where the source code is unzipped

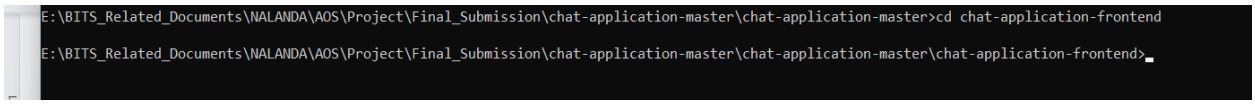
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.1098]
(c) Microsoft Corporation. All rights reserved.

E:\BITS_Related_Documents\NALANDA\AOS\Project\Final_Submission\chat-application-master\chat-application-master\chat-application-backend>
```

7. Run the following command: **gradlew bootRun --args='server=concurrent;ip=IP'**
8. IP is the same we noted in step 5 Eg (gradlew bootRun --args='server=concurrent;ip=172.17.87.124')

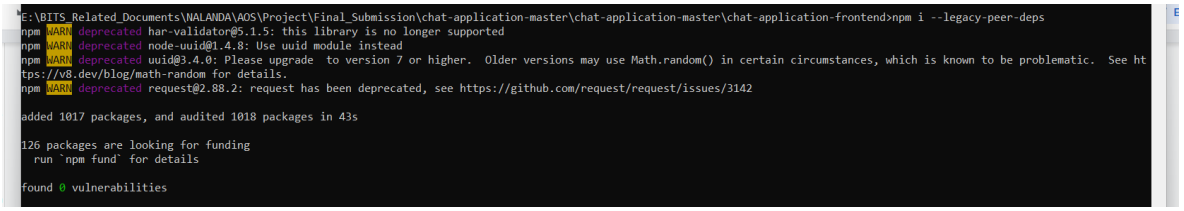
9. Once the application is running it would look like this:

10. Let's run the frontend application in **Concurrent Mode**. Do **cd ..** and **cd chat-application-frontend** and move to that directory

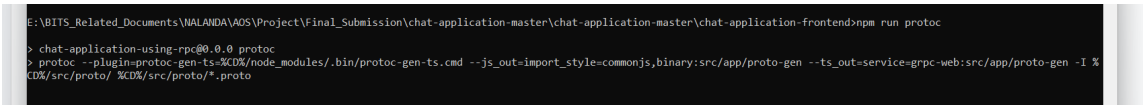


Run the following commands

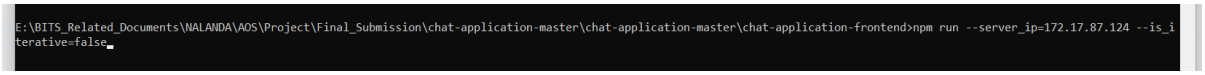
1. **npm i --legacy-peer-deps**



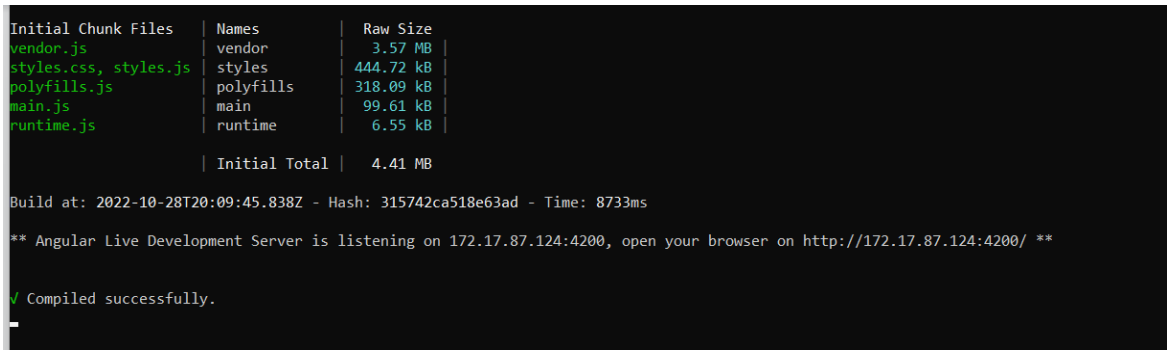
2. **npm run protoc**



3. **npm start --server_ip=IP (that we obtained in step 5) --is_iterative=false**
   Eg: npm start –server_ip=172.17.87.124



Once the server is started we can access it on the mentioned URL



The URL would be : http://IP:4200/ eg (http://172.17.87.124:4200/) and we can access the application on that URL.

NOTE: In order to connect different clients to this server we need to disable the firewall of the machine where server is running.

System Behavior in case of some edge cases:

1. Currently it is assumed that all server will run on same machine, which is configurable and can be changed later.
2. Right now, there is no mechanism to track if user abruptly logs out of the application by closing the browser window, browser tab or shutting down the system,hence it is advisable to log out by clicking the log out button on home page.
3. Chats of the users (1-1 or Group) are not stored in database, hence refreshing the page, logging out and logging back in will mean the chat history is lost.