Distributed System

Google Remote Procedure Call GRPC

EL MAJJODI Abdeljalil

Distribution System and Artificial Intelligence Master's
Department of Mathematics and Informatics
University of Hassan 2 Casablanca

March 26, 2023

Introduction

In this assignment, we will go over the basics of the Google remote procedure call (GRPC), how to use it to build services, and how to install it on a server. All of this will be put to use when designing a guessing game and the chat application.
Java and Python are the languages we use to put this homework into practice.

Contents

Introduction **i**

1 Create GRPC Models **1**
1.1 Create Proto File 1
1.2 Define services 2
1.2.1 Unary Model 2
1.2.2 Server Sreaming Model 2
1.2.3 Client Sreaming Model 3
1.2.4 Bi-Directional Streaming Model 3
1.3 Create Server GRPC 4
1.4 Test Server With BloomRPC 4
1.5 Create Client GRPC Java 6
1.5.1 Unary Client 6
1.5.2 Server Stream Client 6
1.5.3 Client Stream Client 7
1.5.4 Bi-Dirictional Stream Client 9

2 Chat Application GRPC **11**
2.1 Create Proto File 11
2.2 Define GRPC Sevices 11
2.3 Create Server GRPC 12
2.4 Test with BloomRPC 13
2.5 Create Chat Client GRPC 14
2.5.1 Implimentation with Java 14
2.5.2 Test Client 16
2.5.3 Create Client With Python 16

3 Guess Number Game With GRPC **17**
3.1 Create Proto File 17
3.2 Define GRPC Services 17
3.3 Deploy Game Services in GRPC Server 19
3.4 Create Player GRPC 19
3.5 Test 20

Conclusions **21**

List of Figures

1.1 Unary Model Test 4
1.2 Server Stream Model Test 5
1.3 Client Stream Model Test 5
1.4 Bi-Dirictinal Stream Model Test 5

2.1 BloomRPC 13
2.2 One To One Messages 13
2.3 One To Many Message 14
2.4 User Chat Test 16

3.1 Game Test 20

Chapter 1

Create GRPC Models

1.1 Create Proto File

Protocol buffers (protobuf) are used as the Interface Definition Language (IDL) by default. The .proto file contains:

- The definition of the gRPC service.
- The messages sent between clients and servers.

```
syntax="proto3";
option java_package="ma.elma_dev.stubs";
service bankServices{
  rpc convert(messageReq) returns(messageResp); //unary module
  rpc getCurrencyStream(messageReq) returns(stream messageResp);//server
      stream module
  rpc perfomCurrencyStream(stream messageReq)
      returns(messageResp);//sclient stream module
  rpc fullCurrencyStream(stream messageReq) returns(stream
      messageResp);//Bi_Directional
}

message messageReq{
  string messageFrom=1;
  string messageTo=2;
  double amount=3;
}

message messageResp{
  string messageFrom=1;
  string messgaeTo=2;
  double amount=3;
  double result=4;

}
```

1.2 Define services

After compiling the proto file we should define the services as a java class or python class.

1.2.1 Unary Model

```
public void convert(BankServices.messageReq request,
    StreamObserver<BankServices.messageResp> responseObserver) {
        String messageFrom=request.getMessageFrom();
        String messageTo=request.getMessageTo();
        double amount=request.getAmount();

        BankServices.messageResp response=
            BankServices.messageResp.newBuilder().
                setMessageFrom(messageFrom).setMessgaeTo(messageTo).
                setAmount(amount).setResult(amount*11.30)
        .build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
}
```

1.2.2 Server Sreaming Model

```
public void getCurrencyStream(BankServices.messageReq request,
    StreamObserver<BankServices.messageResp> responseObserver) {
    String messageFrom = request.getMessageFrom();
    String messageTo = request.getMessageTo();
    double amount = request.getAmount();
    Timer timer = new Timer();
    timer.schedule(new TimerTask() {
        int counter=0;
        @Override
        public void run() {
            BankServices.messageResp messageResp =
                BankServices.messageResp.newBuilder().
            setMessageFrom(messageFrom).setMessgaeTo(messageTo).setAmount(amount).setResu
                * Math.random() * 10).build();
            responseObserver.onNext(messageResp);
            if(counter++ ==20){
                responseObserver.onCompleted();
                timer.cancel();
            }
            counter++;
        }
    },1000,1000);
}
```

1.2.3 Client Sreaming Model

```
public StreamObserver<BankServices.messageReq>
    perfomCurrencyStream(StreamObserver<BankServices.messageResp>
    responseObserver) {
    return new StreamObserver<BankServices.messageReq>() {
        double sum=0;
        @Override
        public void onNext(BankServices.messageReq messageReq) {
            System.out.println(messageReq.getAmount());
            sum+=messageReq.getAmount();

        }

        @Override
        public void onError(Throwable throwable) {

        }

        @Override
        public void onCompleted() {
            BankServices.messageResp resp =
                BankServices.messageResp.newBuilder()
                    .setResult(sum*11.30).build();
            responseObserver.onNext(resp);
            responseObserver.onCompleted();

        }
    };
}
```

1.2.4 Bi-Directional Streaming Model

```
public StreamObserver<BankServices.messageReq>
    fullCurrencyStream(StreamObserver<BankServices.messageResp>
    responseObserver) {
    return new StreamObserver<BankServices.messageReq>() {
        @Override
        public void onNext(BankServices.messageReq messageReq) {
            System.out.println(messageReq.toString());
            BankServices.messageResp resp =
                BankServices.messageResp.newBuilder().setResult(messageReq.getAmount()
                * 11.30).build();
            responseObserver.onNext(resp);
        }

        @Override
        public void onError(Throwable throwable) {
```

```
        }

        @Override
        public void onCompleted() {
            responseObserver.onCompleted();
        }
    };
}
```

1.3 Create Server GRPC

In this section, we try to deploy our services in **localhost** server with **port 2001**.

```
package ma.elma_dev.server;

import io.grpc.ServerBuilder;
import ma.elma_dev.services.BankSercvices;
public class bankServer {
    public static void main(String[] args) throws Exception {
        //build a server with bank services in localhost port 2001
        ServerBuilder.forPort(2001).addService(new
            BankSercvices()).build().start().awaitTermination();
    }
}
```

1.4 Test Server With BloomRPC

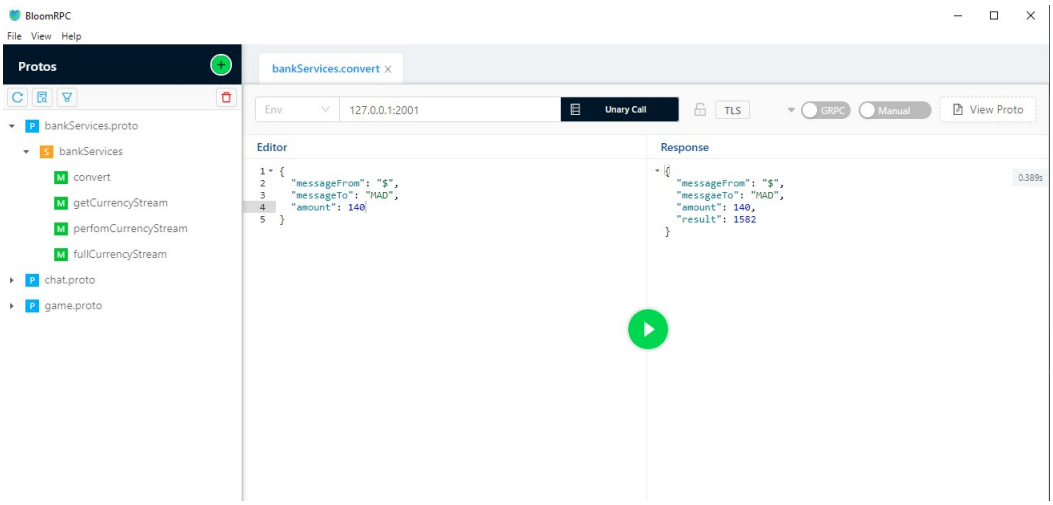

Figure 1.1: Unary Model Test

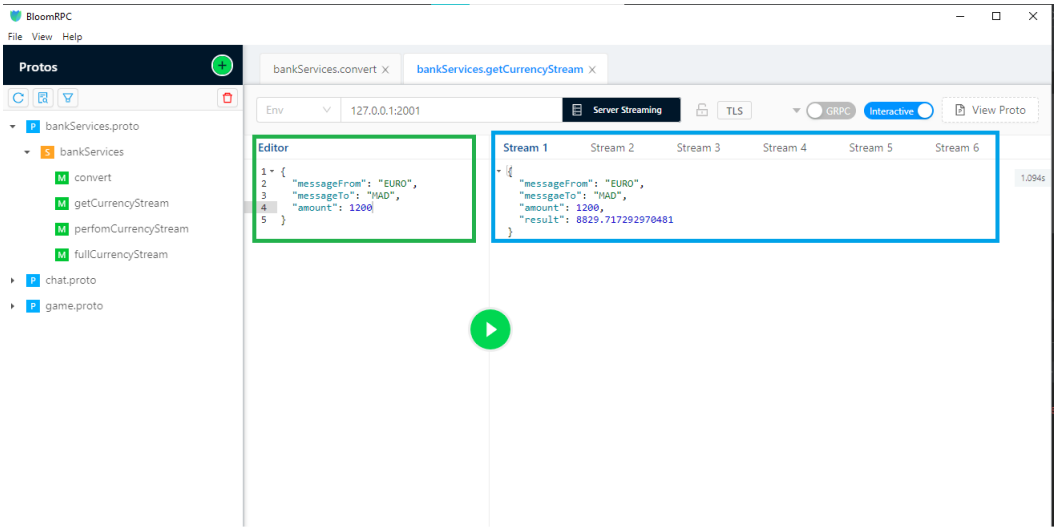

Figure 1.2: Server Stream Model Test

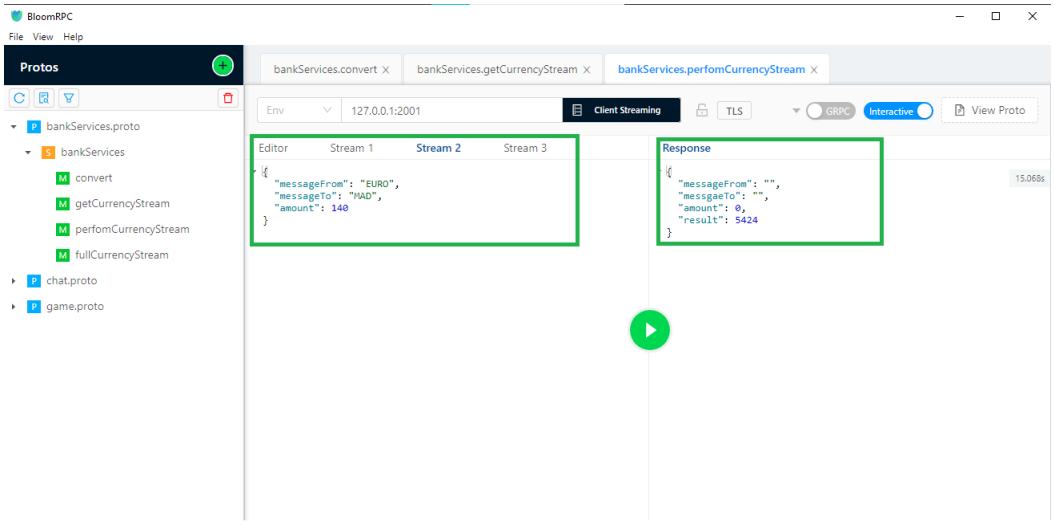

Figure 1.3: Client Stream Model Test

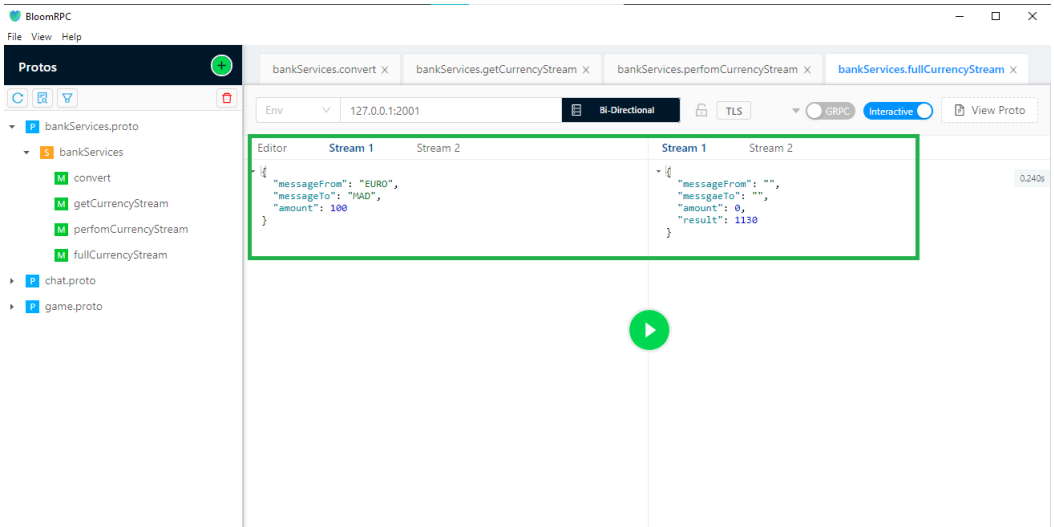

Figure 1.4: Bi-Directinal Stream Model Test

1.5 Create Client GRPC Java

We will build a GRPC client in this section that can use the server's services.

1.5.1 Unary Client

```
package dev.elma.clients;

import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import dev.elma.stubs.BankServices;
import dev.elma.stubs.bankServicesGrpc;

public class ClientGrpcUnary {
    public static void main(String[] args) {
        //create connection with server
        ManagedChannel localhost =
            ManagedChannelBuilder.forAddress("localhost",
            2001).usePlaintext().build();
        //create stub between client and srv
        bankServicesGrpc.bankServicesBlockingStub bankServicesBlockingStub
            = bankServicesGrpc.newBlockingStub(localhost);
        //create req msg
        BankServices.messageReq
            messageReq=BankServices.messageReq.newBuilder().
                setMessageFrom("MAD").setMessageTo("$").setAmount(200).build();
        //create resp msg
        BankServices.messageResp
            messageResp=bankServicesBlockingStub.convert(messageReq);

        //show the response
        System.out.println(messageResp);

    }
}
```

1.5.2 Server Stream Client

```
package dev.elma.clients;

import dev.elma.stubs.BankServices;
import dev.elma.stubs.bankServicesGrpc;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;
import java.io.IOException;

public class ClientGrpcSStream {
```

```
    public static void main(String[] args) throws IOException {
        ManagedChannel connection =
            ManagedChannelBuilder.forAddress("localhost",
            2001).usePlaintext().build();
        bankServicesGrpc.bankServicesStub bankServicesStub =
            bankServicesGrpc.newStub(connection);

        //create Request:
        BankServices.messageReq mesgReq =
            BankServices.messageReq.newBuilder().
                setMessageTo("MAD").
                setMessageTo("EURO").
                setAmount(200).build();

        bankServicesStub.getCurrencyStream(mesgReq, new
            StreamObserver<BankServices.messageResp>() {
            @Override
            public void onNext(BankServices.messageResp messageResp) {
                System.out.println(messageResp.toString());
            }
            @Override
            public void onError(Throwable throwable) {

            }
            @Override
            public void onCompleted() {
                System.out.println("The Last One...");
            }
        });

        System.out.println("Waiting the Response...");
        System.in.read();

    }
}
```

1.5.3 Client Stream Client

```
package dev.elma.clients;

import dev.elma.stubs.BankServices;
import dev.elma.stubs.bankServicesGrpc;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;

import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;
```

```
public class ClientGrpcCS {
    public static void main(String[] args) throws IOException {
        ManagedChannel localhost
            =ManagedChannelBuilder.forAddress("localhost",2001).usePlaintext().build();
        bankServicesGrpc.bankServicesStub bankServicesStub =
            bankServicesGrpc.newStub(localhost);

        StreamObserver<BankServices.messageReq> messageReqStreamObserver =
            bankServicesStub.perfomCurrencyStream(new
            StreamObserver<BankServices.messageResp>() {
            @Override
            public void onNext(BankServices.messageResp messageResp) {
                System.out.println("Response of server is:
                    "+messageResp.getResult());
            }
            @Override
            public void onError(Throwable throwable) {

            }
            @Override
            public void onCompleted() {
                //System.out.println("The last One...");
            }
        });

        Timer timer = new Timer();
        timer.schedule(new TimerTask() {
            int counter=0;
            @Override
            public void run() {
                BankServices.messageReq messageReq =
                    BankServices.messageReq.newBuilder().
                        setMessageFrom("MAD").setMessageTo("EURO").
                        setAmount(Math.random() * 100).build();
                messageReqStreamObserver.onNext(messageReq);
                if(counter++==20){
                    messageReqStreamObserver.onCompleted();
                    timer.cancel();
                }
            }
        }, 1000, 1000);

        System.out.println("-----------Wait Resp----------");
        System.in.read();


    }
}
```

1.5.4 Bi-Dirictional Stream Client

```
package dev.elma.clients;

import dev.elma.stubs.BankServices;
import dev.elma.stubs.bankServicesGrpc;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;
import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;

public class ClientBiDirectionalStream{
    public static void main(String[] args) throws IOException {
        ManagedChannel localhost =
            ManagedChannelBuilder.forAddress("localhost",
            2001).usePlaintext().build();//Channel with server
        bankServicesGrpc.bankServicesStub bankServicesStub =
            bankServicesGrpc.newStub(localhost);//Stubs

        StreamObserver<BankServices.messageReq> messageReqStreamObserver =
            bankServicesStub.fullCurrencyStream(new
            StreamObserver<BankServices.messageResp>() {
            @Override
            public void onNext(BankServices.messageResp messageResp) {
                System.out.println(messageResp.toString());
            }

            @Override
            public void onError(Throwable throwable) {

            }

            @Override
            public void onCompleted() {
                System.out.println("That's The Last One...");
            }
        });

        Timer timer = new Timer();
        timer.schedule(new TimerTask() {
            int counter=0;
            @Override
            public void run() {
                BankServices.messageReq msg =
                    BankServices.messageReq.newBuilder().setMessageFrom("MAD").
                setMessageTo("EURO").setAmount(Math.random() + 100).build();
                messageReqStreamObserver.onNext(msg);
                if(counter++==20){
                    messageReqStreamObserver.onCompleted();
```

```
                    timer.cancel();
                }
            }
        }, 1000, 1000);

        System.out.println("Waiting...");
        System.in.read();
    }
}
```

Chapter 2

Chat Application GRPC

2.1 Create Proto File

```
syntax="proto3";
option java_package="dev.elma.stubs";

service chat{
  rpc send(stream request) returns (stream request); //bi-directional stream
  rpc connectReq(connect) returns (request); //request of connection
  rpc disconnectReq(connect) returns(request); //request of disconnect
}
message request{
  string messageFrom=1;
  string messageTo=2;
  string content=3;
}

message connect{
  string username=1;
}
```

2.2 Define GRPC Sevices

```
    package dev.elma.services;

    import dev.elma.stubs.Chat;
    import dev.elma.stubs.chatGrpc;
    import io.grpc.stub.StreamObserver;

    import java.util.HashMap;

    public class ServicesDefine extends chatGrpc.chatImplBase {
        HashMap<String,StreamObserver<Chat.request>> clientsBuff=new
            HashMap<>();
```

```
    @Override
    public void disconnectReq(Chat.connect request,
        StreamObserver<Chat.request> responseObserver) {
        String username = request.getUsername();
        clientsBuff.get(username).onCompleted();
        clientsBuff.remove(username);
        Chat.request serverRep =
            Chat.request.newBuilder().setMessageFrom("Server").setContent("You
            are disconnected").build();
        responseObserver.onNext(serverRep);
        responseObserver.onCompleted();
    }

    @Override
    public StreamObserver<Chat.request> send(StreamObserver<Chat.request>
        responseObserver) {
        return new StreamObserver<Chat.request>() {
            @Override
            public void onNext(Chat.request request) {
                String messageFrom = request.getMessageFrom();
                String messageTo = request.getMessageTo();
                if(!clientsBuff.containsKey(messageFrom)){
                    clientsBuff.put(messageFrom,responseObserver);
                }
                if(messageTo.isEmpty()){
                    for(String s : clientsBuff.keySet()){
                        if(!s.equals(messageFrom))
                            clientsBuff.get(s).onNext(request);
                    }
                }
                else if(clientsBuff.containsKey(messageTo)){
                    StreamObserver<Chat.request> requestStreamObserver =
                        clientsBuff.get(messageTo);
                    requestStreamObserver.onNext(request);
                }
            }
            @Override
            public void onError(Throwable throwable) {}
            @Override
            public void onCompleted() {}
        };
    }
}
```

2.3 Create Server GRPC

```
package dev.elma.server;
import dev.elma.services.ServicesDefine;
import io.grpc.ServerBuilder;
```

```
import java.io.IOException;

public class ServerGRPC {
    public static void main(String[] args) throws InterruptedException,
        IOException {
        ServerBuilder.forPort(2001).addService(new
            ServicesDefine()).build().start().awaitTermination();
    }
}
```

2.4 Test with BloomRPC

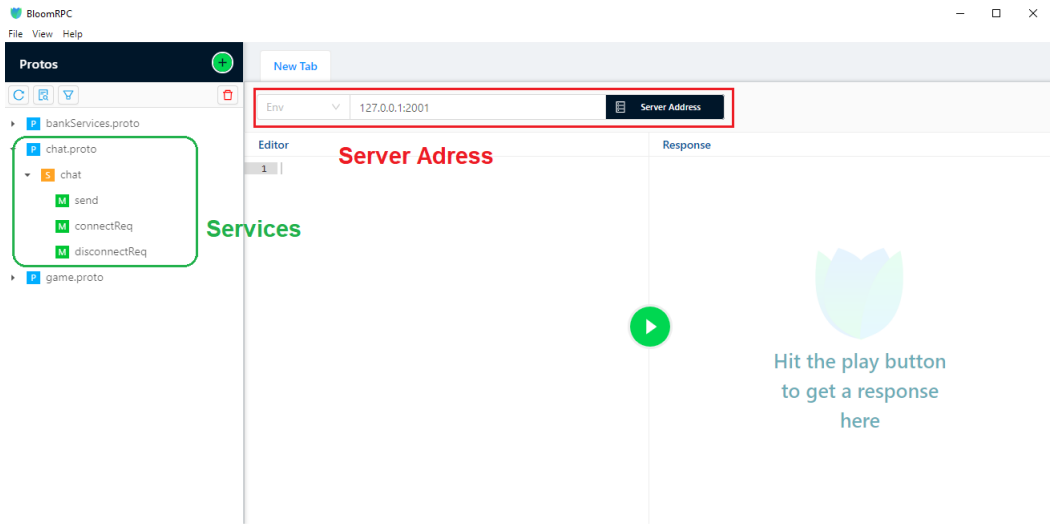

Figure 2.1: BloomRPC

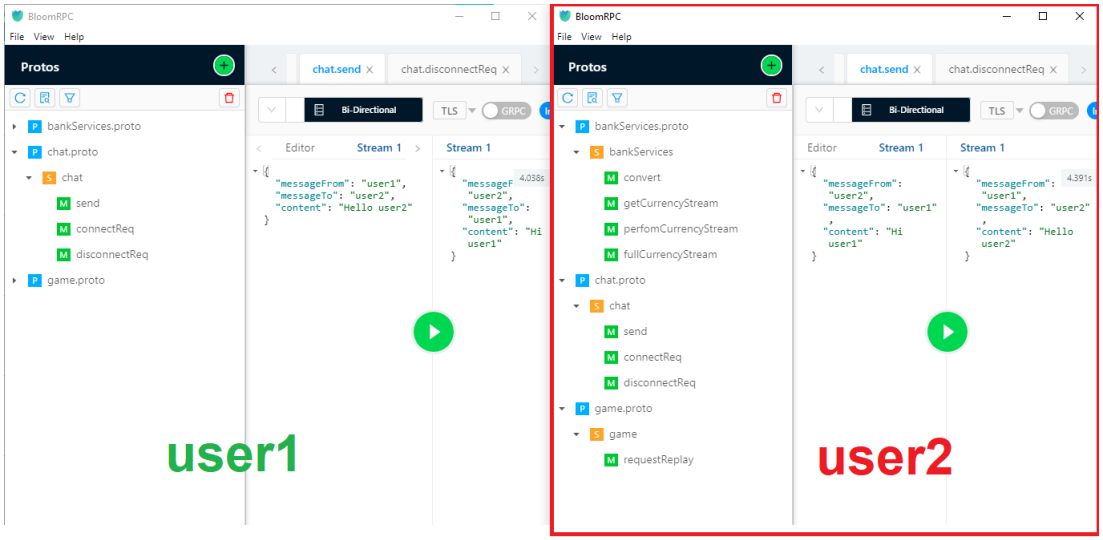

Figure 2.2: One To One Messages

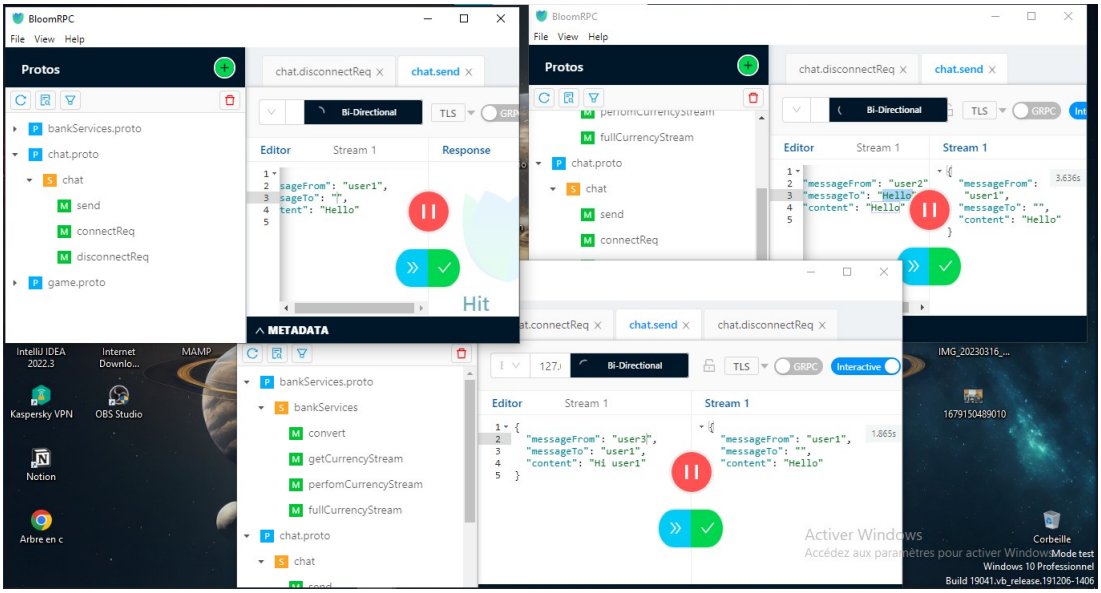

Figure 2.3: One To Many Message

2.5 Create Chat Client GRPC

We will create a user that can access the three chat modes available in the chat application:

- One To One Mode
- One To Many
- One To All

2.5.1 Implimentation with Java

```
package dev.elma.users;

import dev.elma.stubs.Chat;
import dev.elma.stubs.chatGrpc;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;

import java.io.IOException;
import java.util.Scanner;

public class User {
    public static void main(String[] args) throws IOException {
        ManagedChannel localhost =
            ManagedChannelBuilder.forAddress("localhost",
            2001).usePlaintext().build();
        chatGrpc.chatStub chatStub = chatGrpc.newStub(localhost);
```

```
String username;
String message;
Scanner scanner=new Scanner(System.in);

System.out.print("Username : ");
username=scanner.next();
System.out.println(username+ " connected...");
System.out.println("To disconnect write (disc) in message...");



StreamObserver<Chat.request> send = chatStub.send(new
    StreamObserver<Chat.request>() {
     @Override
     public void onNext(Chat.request request) {
         String messageFrom = request.getMessageFrom();
         String content = request.getContent();
         System.out.println(messageFrom+" : "+content);
     }

     @Override
     public void onError(Throwable throwable) {

     }

     @Override
     public void onCompleted() {

     }
 });

while (true){
    System.out.println("-----------------------------------------------");
    System.out.print("Message To : ");
    String messageTo=scanner.next();
    System.out.print("Message : ");
    message=scanner.next();
    System.out.println(message);
    if(message.equals("disc")) {
        Chat.connect disconnect =
            Chat.connect.newBuilder().setUsername(username).build();
        chatStub.disconnectReq(disconnect, new
            StreamObserver<Chat.request>() {
            @Override
            public void onNext(Chat.request request) {
                System.out.println(request.getContent());
            }

            @Override
            public void onError(Throwable throwable) {
```

```
                    }

                    @Override
                    public void onCompleted() {

                    }
                });
                //System.in.read();
                break;
            }
            Chat.request request =
                Chat.request.newBuilder().setMessageFrom(username).
            setMessageTo(messageTo).setContent(message).build();
            send.onNext(request);
        }
    }
}
```

2.5.2 Test Client

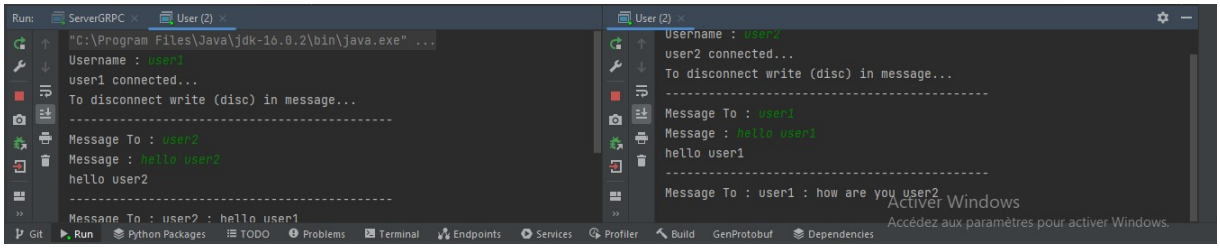

Figure 2.4: User Chat Test

2.5.3 Create Client With Python

Chapter 3

Guess Number Game With GRPC

In this section, we will utilize GRPC to make a guess-number game. The rules of the game are that the user can guess the number after connecting by using the random number we choose as a secret number when we start the server.

3.1 Create Proto File

```
syntax="proto3";
option java_package="dev.elma.stubs";

service game{
  rpc requestReplay(stream guessMsg) returns (stream repMsg);
}

message guessMsg{
  string username=1;
  double number=2;
}

message repMsg{
  string content=1;
}
```

3.2 Define GRPC Services

```
package dev.elma.services;

import dev.elma.stubs.Game;
import dev.elma.stubs.gameGrpc;
import io.grpc.stub.StreamObserver;
import java.util.HashMap;

public class Services extends gameGrpc.gameImplBase{
    //Secret number...
```

```
int number=(int)(Math.random()*1000);
//list of users
HashMap<String,StreamObserver<Game.repMsg>> players=new HashMap<>();
@Override
public StreamObserver<Game.guessMsg>
    requestReplay(StreamObserver<Game.repMsg> responseObserver) {
    return new StreamObserver<Game.guessMsg>() {
        @Override
        public void onNext(Game.guessMsg guessMsg) {
            System.out.println(number);

            String player=guessMsg.getUsername();
            //if is the new user
            if(!players.containsKey(player)){
                players.put(player,responseObserver);
            }
            int nbr=(int)(guessMsg.getNumber());
            Game.repMsg repMsg;
            //if guessing number is correct
            if(nbr==number){
                for(String pl:players.keySet()){
                    if(pl.equals(player)){
                        repMsg=Game.repMsg.newBuilder().setContent("You
                            WIIIN ").build();
                        players.get(pl).onNext(repMsg);
                        players.get(pl).onCompleted();
                    }
                    else {
                        repMsg=Game.repMsg.newBuilder().setContent(player+"
                            WIIIN ").build();
                        players.get(pl).onNext(repMsg);
                        players.get(pl).onCompleted();
                    }
                }
            }
            //guessing number is less than the secret number
            else if(nbr<number){
                repMsg=Game.repMsg.newBuilder().setContent("Greater
                    than "+nbr).build();
                responseObserver.onNext(repMsg);
            }
            //guessing number is Greater than the secret number
            else{
                repMsg=Game.repMsg.newBuilder().setContent("Less than
                    "+nbr).build();
                responseObserver.onNext(repMsg);
            }

        }
        @Override
        public void onError(Throwable throwable) {}
```

```
            @Override
            public void onCompleted() {}
        };
    }
}
```

3.3 Deploy Game Services in GRPC Server

```
package dev.elma.server;

import dev.elma.services.Services;
import io.grpc.ServerBuilder;
public class Server {
    public static void main(String[] args) throws Exception {
        ServerBuilder.forPort(2001).addService(new
            Services()).build().start().awaitTermination();
    }
}
```

3.4 Create Player GRPC

```
package dev.elma.players;

import dev.elma.stubs.Game;
import dev.elma.stubs.gameGrpc;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;
import java.util.Scanner;

import static java.lang.System.exit;

public class player {
    public static void main(String[] args) {
        ManagedChannel localhost =
            ManagedChannelBuilder.forAddress("localhost",
            2001).usePlaintext().build();
        gameGrpc.gameStub gameStub = gameGrpc.newStub(localhost);
        StreamObserver<Game.guessMsg> guessMsgStreamObserver =
            gameStub.requestReplay(new StreamObserver<Game.repMsg>() {
            @Override
            public void onNext(Game.repMsg repMsg) {
                String content = repMsg.getContent();
                System.out.println(content);
            }
            @Override
            public void onError(Throwable throwable) {}
```

```
            @Override
            public void onCompleted() {
                exit(0);
            }
        });

        String username;
        System.out.print("Username:");
        Scanner scanner = new Scanner(System.in);
        username=scanner.next();

        while(true) {
            System.out.println("Guess Number: ");
            int number=new Scanner(System.in).nextInt();
            Game.guessMsg build =
                Game.guessMsg.newBuilder().setUsername(username).setNumber((double)
                number).build();
            guessMsgStreamObserver.onNext(build);
        }
    }
}
```

3.5 Test

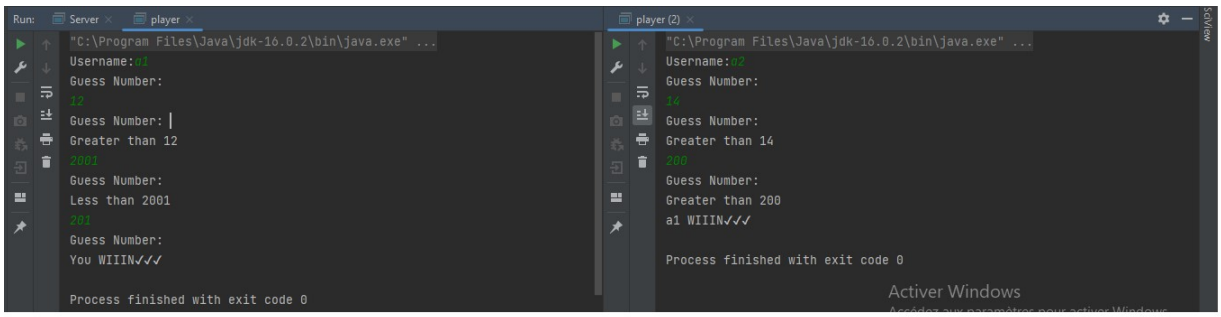

Figure 3.1: Game Test

Conclusions

I learned what a buffer file is through this task, and I have gotten a lot out of it. how do you make it? How can I use buffer files in Python and Java? What are grpc models, how are they implemented, and how else?