Rozproszone Systemy Informatyczne

Raport - Ćwiczenie 1b

Paweł Kluska, 260391

Katsiaryna Ziatsikava, 245891

System składa się z 3 modułów:

- GrpcClient

- GrpcInterface

- GrpcServer

GrpcInterface służył do sprecyzowania typów danych oraz metody, która będzie dostępne na serwerze.

GrpcServer implementował metodę, która została podana w GrpcInterface.

GrpcClient mógł połączyć się z serwerem, a następnie uruchomić metodę, która została tam wystawiona.

Na początku odpowiednio przygotowaliśmy pliki konfiguracyjne pom.xml dla projektu i poszczególnych modułów.

Main pom.xml

W main pom.xml zdefiniowane moduły i wspólne komponenty takie jak properties i dependencies.

```xml
<modules>
    <module>GrpcClient</module>
    <module>GrpcInterface</module>
    <module>GrpcServer</module>
</modules>

<properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <grpc.version>1.43.2</grpc.version>
    <protobuf.maven.plugin.version>0.6.1</protobuf.maven.plugin.version>
    <protobuf.version>3.19.4</protobuf.version>
    <os.maven.plugin.version>1.6.2</os.maven.plugin.version>
</properties>
```

```xml
<dependencies>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-protobuf</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-stub</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-netty</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.tomcat</groupId>
        <artifactId>annotations-api</artifactId>
        <version>6.0.53</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

GrpcInterface pom.xml

W pliku jest zdefiniowany moduł rodzica,os-maven-plugin i protobuf-maven-plugin.

```
<parent>
    <artifactId>Rsi-lab3</artifactId>
    <groupId>com.example</groupId>
    <version>1.0-SNAPSHOT</version>
</parent>
```

```
<build>
    <extensions>
        <extension>
            <groupId>kr.motd.maven</groupId>
            <artifactId>os-maven-plugin</artifactId>
            <version>${os.maven.plugin.version}</version>
        </extension>
    </extensions>
```

```
    <plugins>
        <plugin>
            <groupId>org.xolstice.maven.plugins</groupId>
            <artifactId>protobuf-maven-plugin</artifactId>
            <version>${protobuf.maven.plugin.version}</version>
            <configuration>
                <protocArtifact>com.google.protobuf:protoc:${protobuf.version}:exe:${os.detected.classifier}
                </protocArtifact>
                <pluginId>grpc-java</pluginId>
                <pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc.version}:exe:${os.detected.classifier}
                </pluginArtifact>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>compile-custom</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

GrpcServer pom.xml

W GrpcServer pom.xml jest zdefiniowany moduł rodzica, zależność do GrpcInterface i maven-assembly-plugin.

```
<parent>
    <artifactId>Rsi-lab3</artifactId>
    <groupId>com.example</groupId>
    <version>1.0-SNAPSHOT</version>
</parent>
```

```
<dependencies>
    <dependency>
        <groupId>com.example</groupId>
        <artifactId>GrpcInterface</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
```

```
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.3.0</version>
        <configuration>
            <descriptorRefs>
                <descriptorRef>
                    jar-with-dependencies
                </descriptorRef>
            </descriptorRefs>
            <archive>
                <manifest>
                    <mainClass>GrpcServer</mainClass>
                </manifest>
            </archive>
        </configuration>
        <executions>
            <execution>
                <id>make-assembly</id>
                <phase>package</phase>
                <goals>
                    <goal>single</goal>
                </goals>
            </execution>
        </executions>
```

GrpcClient pom.xml

Posiada identyczną konfigurację jak GrpcServer.

Proto file

W module GrpcInterface w folderze main był stworzony plik GrpcInterface.proto, który umożliwia korzystanie z usług gRPC oraz komunikację między klientami gRPC i komunikatami serwera. W tym pliku definiujemy typy danych dla Request, Response i metodę rpc grpcProcedure, która przyjmuje GrpcRequest i zwraca GrpcResponse.

```
syntax = "proto3";
package org.example;
// The service definition.
service ServiceName {
// Remote procedures:
rpc unaryProcedure (TheRequest) returns (TheResponse) {}
}
// The request message containing the user's name and age.
message TheRequest {
string name1 = 1;
string name2 = 2;
double latitude1 = 3;
double longitude1 = 4;
double latitude2 = 5;
double longitude2 = 6;
double latitude3 = 7;
double longitude3 = 8;
}
// The response message containing the hello text
message TheResponse {
double distance = 1;
}
```

Po określeniu tego pliku automatycznie generujemy kod źródłowy, którego dalej używamy w klasach GrpcServer i GrpcClient.

GrpcServer class

Ta klasa jest odpowiedzialna za wystawienie serwera na określonym ip, porcie .

```java
public class GrpcServer {
    no usages
    public static void main(String[] args) {
        MyDate.info();
        int port = 50000;
        System.out.println("Starting gRPC server...");
        Server server = ServerBuilder.forPort(port).addService(new MyServiceImpl()).build();
        try {
            server.start();
            System.out.println("...Server started");
            server.awaitTermination();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Została zaimplementowana metoda haversine() , która wylicza odległość pomiędzy dwoma punkatami.

```java
static class MyServiceImpl extends ServiceNameGrpc.ServiceNameImplBase {
    3 usages
    private double haversine(double lat1, double lon1, double lat2, double lon2) {
        // distance between latitudes and longitudes
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);

        // convert to radians
        lat1 = Math.toRadians(lat1);
        lat2 = Math.toRadians(lat2);

        // apply formulae
        double a = Math.pow(Math.sin(dLat / 2), 2) +
                Math.pow(Math.sin(dLon / 2), 2) *
                        Math.cos(lat1) *
                        Math.cos(lat2);
        double rad = 6371;
        double c = 2 * Math.asin(Math.sqrt(a));
        return rad * c;
    }
```

Również została zaimplementowana metoda haversine3, która używa haversine() i wylicza odległość na podstawie sumy odległości powiedzy 1 i 2 punktami i pomiędzy 2 i 3.

```
private double haversine3(double lat1, double lon1, double lat2, double lon2, double lat3, double lon3){
    double dst3 = haversine(lat1,lon1,lat2,lon2) + haversine(lat2,lon2,lat3,lon3);
    return dst3;
}
```

Ponadto została zaimplementowana metoda grpcProcedure, która będzie dostępna na zewnątrz. Tu sprawdzamy, czemu są równe Latitude3 i Longitude3. I w zależności od tego wybieramy metodę dla 2 lub 3 punktów. I wysyłamy responce dla klienta.

```
    public void unaryProcedure(GrpcInterface.TheRequest req,
                               StreamObserver<GrpcInterface.TheResponse> responseObserver) {
        double dst;
        System.out.println("...called UnaryProcedure - start");
        //System.out.println(req.getLatitude3() + " " + req.getLongitude3());
        if (req.getLatitude3() == 1000 || req.getLongitude3() == 1000){
            dst = haversine(req.getLatitude1(),req.getLongitude1(),req.getLatitude2(),req.getLongitude2());
        }else {
            dst = haversine3(req.getLatitude1(),req.getLongitude1(),req.getLatitude2()
                    ,req.getLongitude2(), req.getLatitude3(), req.getLongitude3());
        }
        GrpcInterface.TheResponse response = GrpcInterface.TheResponse.newBuilder()
                .setDistance(dst)
                .build();
        try {
            Thread.sleep( millis: 2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        responseObserver.onNext(response);
        responseObserver.onCompleted();
        System.out.println("...called UnaryProcedure - end");
        }
    }
```

GrpcClient class

Ta klasa odpowiada za łączenie oraz wywoływanie metody z serwera. Dodatkowo zostało zaimplementowane proste menu, które pozwalało na wybór obliczania długości pomiędzy 2 lub 3 punktami. Dane były wpisywane z klawiatury przy pomocy Scannera.

```
2 usages
private static City inputCity() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Wpisz nazwę punktu:");
    String nam = sc.nextLine();
    System.out.println("Wpisz szerokość geograficzną:");
    double lat = sc.nextDouble();
    System.out.println("Wpisz długość geograficzną:");
    double log = sc.nextDouble();
    return new City(nam, lat, log);
}
```

```
no usages
public static void main(String[] args) {
    MyData.getData();

    String address = "192.168.43.30";
    int port = 50000;
    System.out.println("Running gRPC client...");
    ArrayList<City> cities;
    int number;

    do {
        cities = new ArrayList<City>();
        System.out.println("Wpisz dla ilu miast (2 lub 3) chcesz policzyć dystans.");
        System.out.println("Jeśli chcesz wyjść wpisz 0.");
        Scanner sc = new Scanner(System.in);
        number = sc.nextInt();
        switch (number) {
            case 2:
                for (int i = 0; i < 2; i++) {
                    cities.add(inputCity());
                }
                break;
            case 3:
                for (int i = 0; i < 3; i++) {
                    cities.add(inputCity());
                }
                break;
            case 0:
                System.exit( status: 0);
            default:
                System.out.println("Coś poszło nie tak.");
        }
```

```
            if(number==2 || number ==3){
                double lat1 = cities.get(0).getLatitudes();
                double lat2 = cities.get(1).getLatitudes();
                double lat3 = cities.size() == 3 ? cities.get(2).getLatitudes() : 1000;
                double lon1 = cities.get(0).getLongitudes();
                double lon2 = cities.get(1).getLongitudes();
                double lon3 = cities.size() == 3 ? cities.get(2).getLongitudes() : 1000;

                ServiceNameGrpc.ServiceNameBlockingStub bStub;
                ManagedChannel channel = ManagedChannelBuilder.forAddress(address, port).usePlaintext().build();
                bStub = ServiceNameGrpc.newBlockingStub(channel);
                GrpcInterface.TheRequest request = GrpcInterface.TheRequest.newBuilder()
                        .setLatitude1(lat1).setLongitude1(lon1)
                        .setLatitude2(lat2).setLongitude2(lon2)
                        .setLatitude3(lat3).setLongitude3(lon3).build();
                System.out.println("...calling unaryProcedure");
                GrpcInterface.TheResponse response = bStub.unaryProcedure(request);
                System.out.println("...after calling unaryProcedure");
                System.out.println("Distance - " + response.getDistance() + " km");
                channel.shutdown();
            }
        }while( number !=0);
    }
}
```

Na powyższych zrzutach ekranu jest widoczne wykorzystanie metody info z klasy MyDate, która została zaimportowna do projektu przy użyciu pliku jar.

Działanie systemu

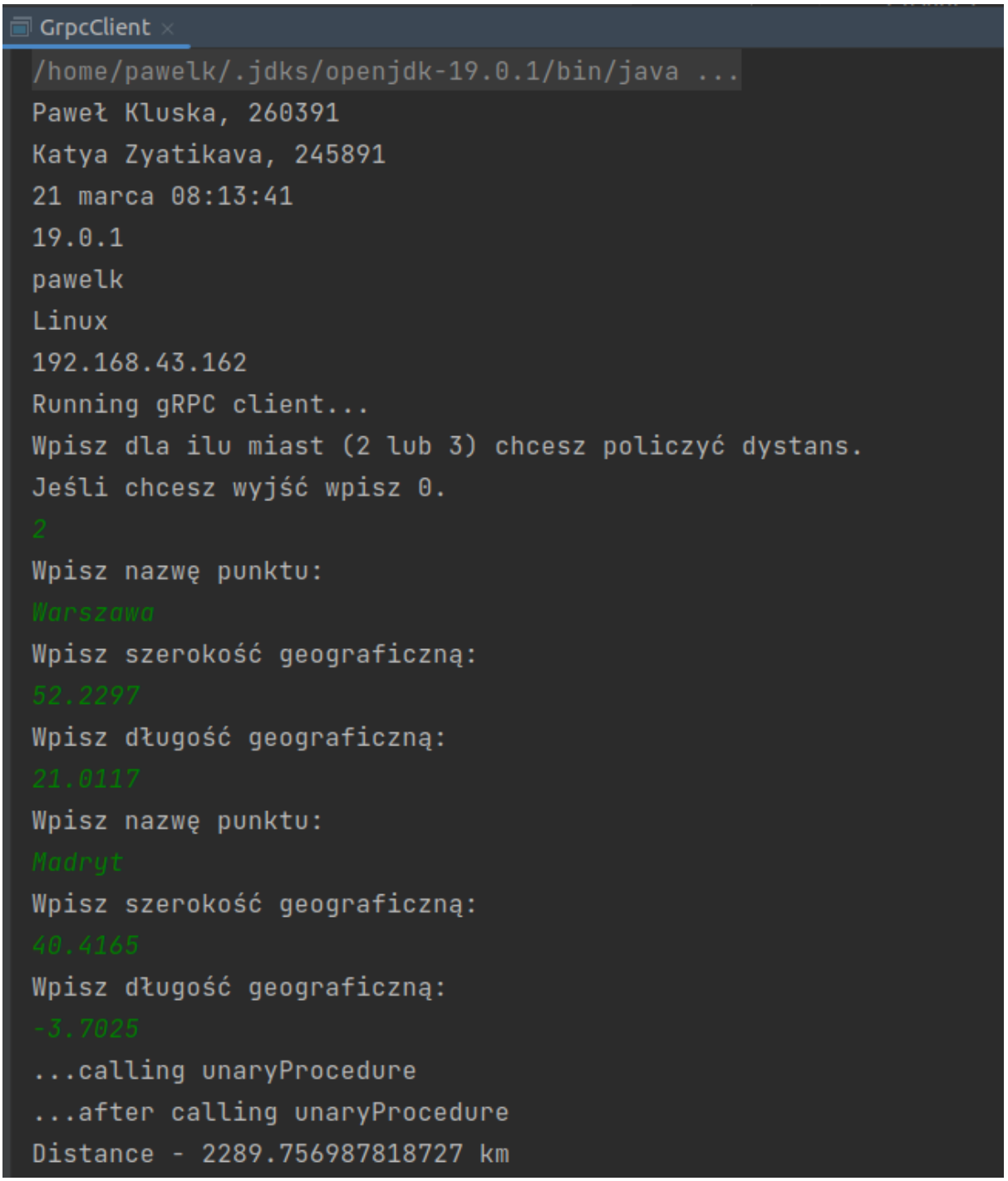
System uruchomiliśmy w konfiguracji dwumaszynowej, na jednej maszynie znajdował się klient a na drugiej serwer. Oba komputery zostały podłączone do tej samej sieci lokalnej. Działanie systemu prezentuje się następująco:

Serwer

```
C:\Users\katya\.jdks\openjdk-19.0.2\bin\java.exe ...
Katsiaryna Ziatsikava - 245891
Paweł Kluska - 260391
21 MARCH 08:30:07
19.0.2+7-44
Windows 10
192.168.56.1
Starting gRPC server...
...Server started
...called UnaryProcedure - start
...called UnaryProcedure - end
```

Działanie klienta

Przykładowe uruchomienie klienta dla 2 miast



Przykładowe uruchomienie klienta dla 3 miast

```
Wpisz dla ilu miast (2 lub 3) chcesz policzyć dystans.
Jeśli chcesz wyjść wpisz 0.
3
Wpisz nazwę punktu:
Warszawa
Wpisz szerokość geograficzną:
52.2297
Wpisz długość geograficzną:
21.0117
Wpisz nazwę punktu:
Madryt
Wpisz szerokość geograficzną:
40.4165
Wpisz długość geograficzną:
-3.7025
Wpisz nazwę punktu:
Buenos Aires
Wpisz szerokość geograficzną:
-34.6131
Wpisz długość geograficzną:
-58.3772
...calling unaryProcedure
...after calling unaryProcedure
Distance - 12335.319685914183 km
```