Rozproszone Systemy Informatyczne

Raport - Ćwiczenie 1

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";
option java_multiple_files = true;
option java_outer_classname = "GrpcAppProto";
option objc_class_prefix = "GAP";
// The greeting service definition.
service GrpcService {
  // Greeting procedure
  rpc grpcProcedure (GrpcRequest) returns (GrpcResponse) {}
}
// The request message containing the user's name and age.
message GrpcRequest {
  string name = 1;
  int32 age = 2;
}
// The response message containing the greetings
message GrpcResponse {
  string message = 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 oraz zaimplementowania metody grpcProcedure, która będzie dostępna na zewnątrz.

```
public class GrpcServer {
    no usages  PawelK
    public static void main(String[] args) {
        MyData.getData();
        int port = 50001;
        System.out.println("Starting server...");
        Server server = ServerBuilder
                .forPort(port)
                .addService(new GrpcServiceImpl()).build();
        try {
            server.start();
            System.out.println("...Server started");
            server.awaitTermination();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Ponadto zostało dodane wyświetlenie aktualnej daty i czasu na serwerze dla odpwiedzi dla klienta.

```
static class GrpcServiceImpl extends GrpcServiceGrpc.GrpcServiceImplBase {
    1 usage
    public void grpcProcedure(GrpcRequest req, StreamObserver<GrpcResponse> responseObserver) {
        String msg;
        LocalDateTime now = LocalDateTime.now();
        DateTimeFormatter time = DateTimeFormatter.ofPattern("dd MMMM HH:mm:ss");
        String newTime = now.format(time);
        System.out.println("...called GrpcProcedure");
        if (req.getAge() > 18)
            msg = "Mr/Ms " + req.getName();
        else
            msg = "Boy/Girl";
        GrpcResponse response = GrpcResponse.newBuilder()
                .setMessage("Hello " + msg + "/n" + newTime).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
```

GrpcClient class

Ta klasa odpowiada za łączenie się z serwerem oraz wywołuje metodę z serweru. Oprócz tego zostało zrobione wpisywanie imienia klienta z klawiatury.

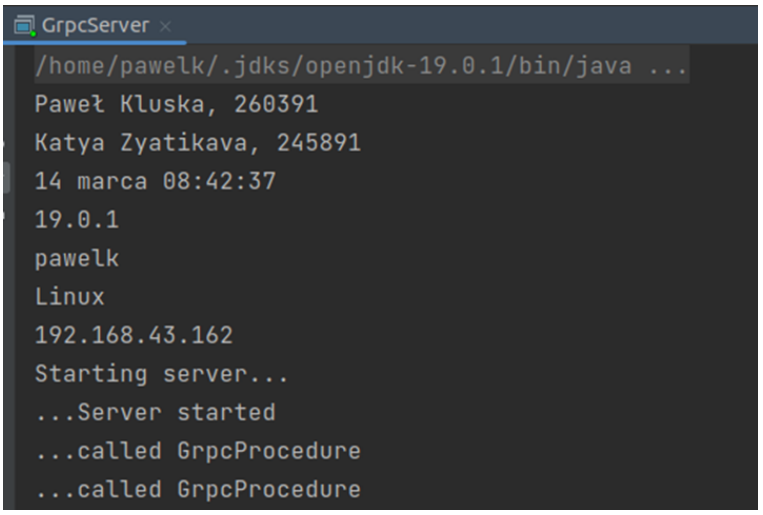
```java
public class GrpcClient {
    no usages
    public static void main(String[] args) {
        MyDate.info();
        String address = "192.168.43.162";
        int port = 50001;
        Scanner sc = new Scanner(System.in);
        System.out.println("Wpisz Imię:");
        String name = sc.nextLine();

        System.out.println("Running grpc client...");
        ManagedChannel channel = ManagedChannelBuilder.forAddress(address, port).usePlaintext()
                .build();
        GrpcServiceGrpc.GrpcServiceBlockingStub stub = GrpcServiceGrpc.newBlockingStub(channel);
        GrpcRequest request = GrpcRequest.newBuilder().setName(name)
                .setAge(24)
                .build();
        GrpcResponse response = stub.grpcProcedure(request);
        System.out.println(response.getMessage());
        channel.shutdown();
    }
```

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

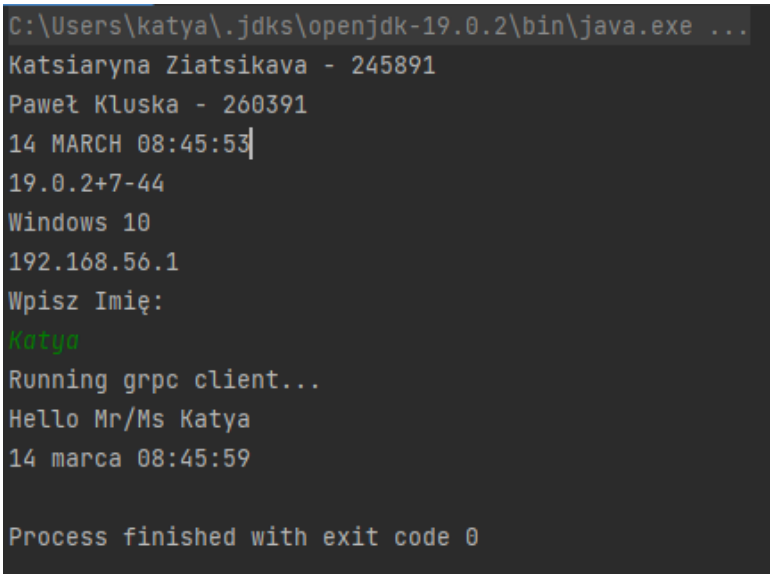
Działanie systemu

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

Serwer



Klient


C:\Users\katya\.jdks\openjdk-19.0.2\bin\java.exe ...
Katsiaryna Ziatsikava - 245891
Paweł Kluska - 260391
14 MARCH 08:45:53
19.0.2+7-44
Windows 10
192.168.56.1
Wpisz Imię:
Katya
Running grpc client...
Hello Mr/Ms Katya
14 marca 08:45:59
Process finished with exit code 0