Ćwiczenie 2

gRPC

(tymczasowo w języku angielskim)

Autor: Mariusz Fraś

1 Objectives of the exercise

The purpose of the exercise is:

1. Familiarization with the development environment for gRPC applications.
2. Understanding the general architecture of the gRPC application.
3. Mastering the basic techniques of creating gRPC applications.

2 Development environment

The developing of gRPC applications can be performed using various platforms. Here is considered the following environment:

- Windows 7 (or higher) Operating System (eventually MacOS).
- Visual Studio 2019 (or 2017),
- Proper NuGet packages.
- Java programming platform (optionally).

3 Exercise – Part I
Creating simple gRPC application project.

3.1 Visual Studio

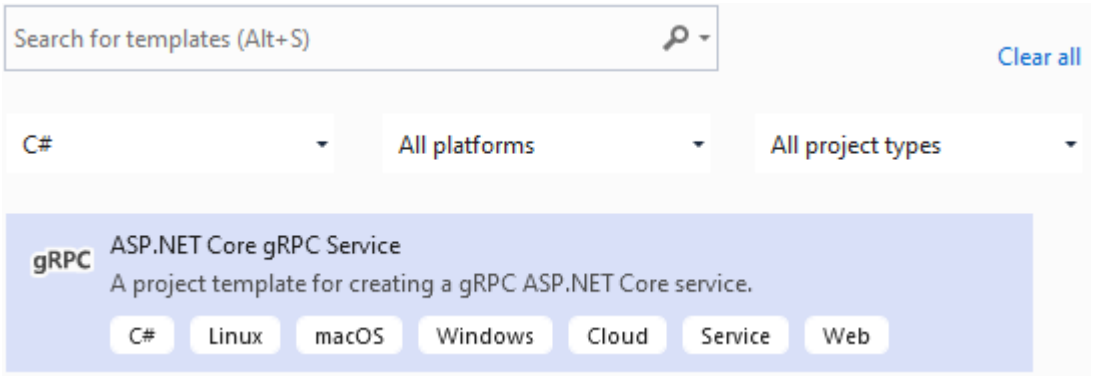
Visual Studio 2019 contains dedicated gRPC service application project. It is also possible to use several NuGet packages to build gRPC application components.

Note (from nugget.org):
"*NuGet is the package manager for .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers.*"
NuGet is a package manager aimed to enable developers to share reusable code. NuGet's client (nuget.exe) is a free and open-source command-line app that can create and consume packages. NuGet is distributed as a Visual Studio extension, and it can natively consume NuGet packages.

A. Visual Studio 2019.

3.1.1 C# project (new solution) for gRPC service (server).

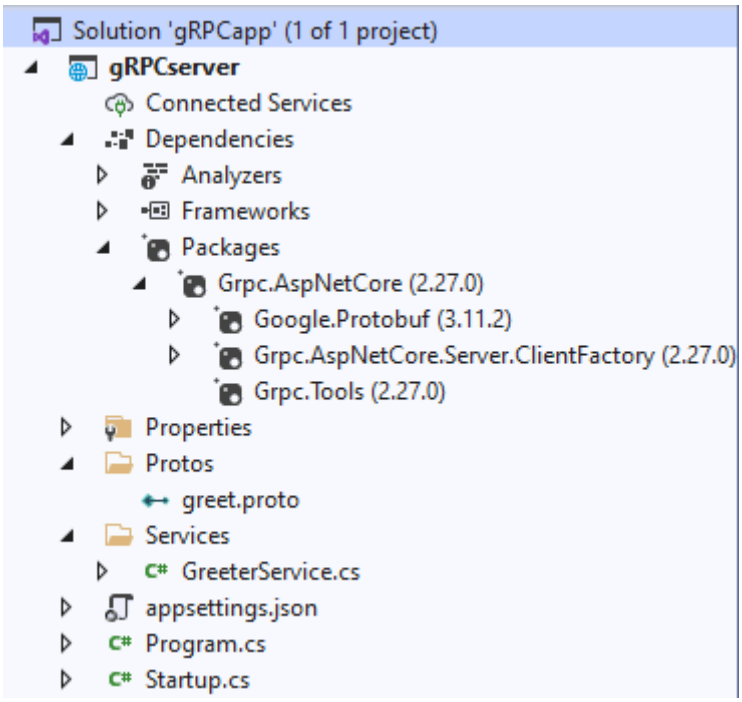
- To create new gRPC app select *ASP.NET Core gRPC Service*. You can use Search field to find the project faster.



- After selection of names (in next window – here project name gRPCserver in gRPCapp solution) you can specify *Target Framework* (.Net 5.0 or .NET Core 3.1), and eventually enable *Docker* option (don't do it now).

3.1.2 The project structure

The initial default project consist of (among the others):
- already included necessary packages – **Grpc.AspNetCore**.
- the initial proto file (file defining procedure interface) – **greet.proto**,
- the initial gRPC service - **GreeterService.cs**,
- **appSettings.json** – contains configuration data
- the code for service host (by default Kestrel web server is used) – **Program.cs**,
- the service configuration code – **Startup.cs**,



To create your own solution you can as well edit *.cs and proto files as rename or add new ones.

Note the following:

- The default main NuGet package for the default gRPC service project is *GrpcASPNetCore* package which include: *Google.Protobuf*, *Grpc.Tools*, *Grpc/ASPNetCore.Server*, and some other packages.
- The project includes files with methods called by the runtime. It includes:
  - **`Startup`** class that configures services and the app's request pipeline. It has two default methods:
    - **`Configure`** method to create the app's request processing pipeline,
    - **`ConfigureServices`** optional method to configure the app's services. Services are registered here.
  - Enabling gRPC service
    gRPC is enabled with the **`AddGrpc`** method in `ConfigureServices` method.
  - Routing
    Routing in ASP..NET Core is responsible for matching incoming requests (especially HTTP ones) and dispatching those requests to the app's service endpoints.

    The service hosted by ASP.NET Core gRPC, should be added to the routing pipeline with use of the **`UseRouting()`**, the **`UseEndpoints()`**, and the **`MapGrpcService<TService>()`** methods called in the `Configure` method. It adds endpoints to the gRPC service.

- Startup class is typically called by `UseStartup<TStartup>` method when the app's host is built.
- Hosting the service

  Host is the component that, among the others, encapsulates (and run) the services – when it is run it calls `StartAcync` on each service implementation..

  - **`Program`** class (with **`Main`** method) is used to create and run the host for gRPC service.
  - The default host for gRPC service is Kestrel – a cross-platform web server for ASP.NET Core (other host options are also available). Kestrel doesn't have some of the advanced features but provides the best performance and memory utilization. It requires HTTP/2 transport and should be secured with TLS. **Attention**: MacOS doesn't support ASP.NET Core gRPC with TLS.
  - Creation and running the host

    In the main method it is built and run the host with builder method:

    `CreateHostBuilder``(args).Build().Run();`

```
public static void Main(string[] args)
{
    CreateHostBuilder(args).Build().Run();
}
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
         {
           webBuilder.UseStartup<Startup>();
         }
    );
```

  - Defining the host

    The default HTTP host (Kestrel Web server) is built using:

    `CreateDefaultBuilder(args)`**`.ConfigureWebHostDefaults`**`(…);`

    In the method it is specified `Startup` as the startup class with use of `UseStartup<TStartup>` method.

3.1.3 gRPC procedure interface defined in proto file

In proto file the interface used for creating stubs for server and client is defined. In the example the remote procedure named **`GrpcProc`** with two parameters is defined. The procedure returns some calculation result and some message (string value).

- Access the **proto.file** and enter (or modify) the following content that defines:
  - Protocol Buffers version,
  - service and procedure definition (with input and output parameters (messages)),

- input (request) message – parameters in call,
- output (response) message – response for call.

```
syntax = "proto3";
option csharp_namespace = "gRPCserver";
package mygrpc;
// Service definition.
service GrpcService {
  rpc GrpcProc (GrpcRequest) returns (GrpcResponse);
}
// The request message
message GrpcRequest {
  string name = 1;
  int32 age = 2;
}
// The response message
message GrpcResponse {
  string message = 1;
  int32 days = 2;
}
```

- Ensure that namespace in proto file is the same as for gRPC service implementation. When changing names in proto file they must be changed in server implementation accordingly.

3.1.4 The gRPC service (gRPC procedure)

In the solution's Services node skeleton code of remote gRPC procedure is generated.

The class must extend the base class with the name of ***service name defined in proto file*** tagged with ***Base*** word. This class is defined in the class generated from proto file named as ***service name defined in proto file***. (i.e. ***SrvName.SrvName*Base** class where ***SrvName*** is the name of the service defined in **proto file**.

In the class the method of remote procedure (with the name as in proto file) takes the request and returns the Task object with response.
Note:
- the `Task<…>` is used to allow asynchronous processing,
- the `ServerCallContext` is not used here (is for potential future use).

- Add/modify the code of GrpcProc that returns some message and number of days for provided years:

```
public class MyGrpcService : GrpcService.GrpcServiceBase
{
  ...
  public override Task<GrpcResponse> GrpcProc(GrpcRequest request,
                                              ServerCallContext context)
  {
```

```
        string msg;
        int val;
        val = request.Age * 12 * 365;
        msg = "Hello "+request.Name+" being "+request.Age+" years old.";
        return Task.FromResult(new GrpcResponse { Message=msg, Days=val });
    }
}
```

Pay attention that request and response field names are with capital first letter!

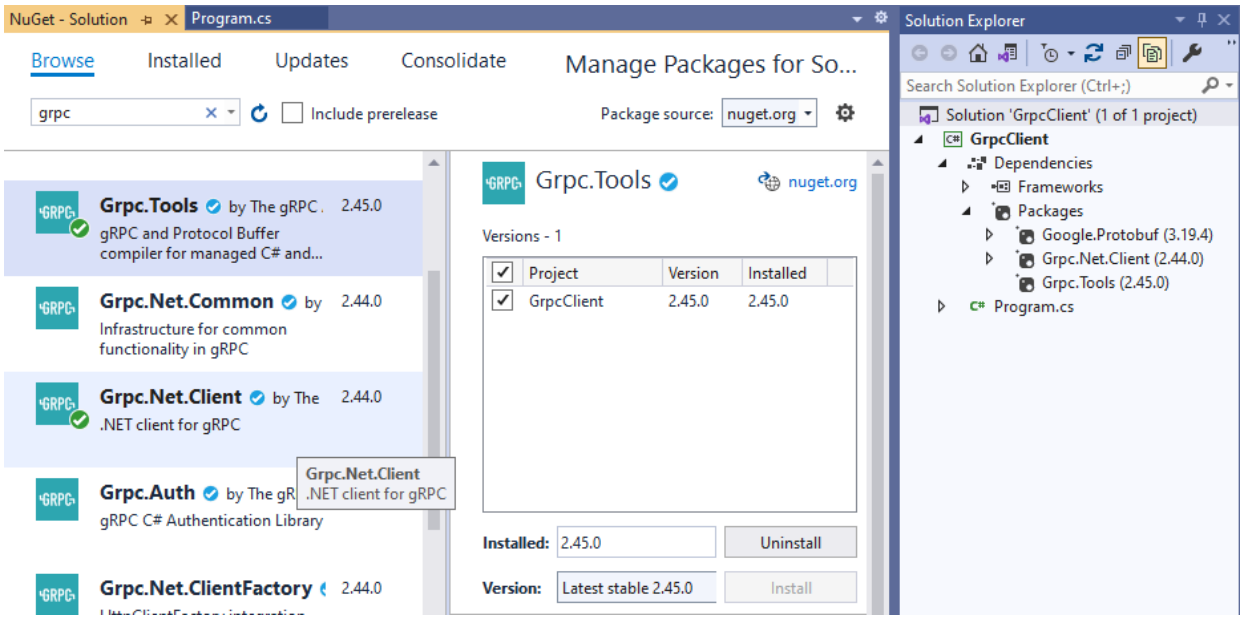
- Build and run the application to verify the correctness.

3.1.5 The gRPC client

Here the client .Net console application will be created.

In order to compile proto files it must be included in the project several packages with use of NuGet manager.

- Create a new project (eventually add one to previous) a *Console Application (.NET Core)* project – here named **gRPCclient**.
- Add NuGet packages: Grpc.Net.Client, Grpc.Tools, Google.Protobuf.
  - Select the project in Solution Explorer window, and from platform menu or context menu select *Manage NuGet Packages* option (or select this option from bar menu).
  - Select *Browse* tab, enter **grpc** in *Search* field.



  - Select and add the following packages to the project:
    - Grpc.Net.Client – .Net client for gRPC,
    - Grpc.Tools – gRPC and Protocol Buffers compiler.
  - Enter **protobuf** in *Search* field.
    Select and add the following package:
    - Google.Protobuf – C# runtime library for Protocol Buffers.

- Add/create new folder (e.g. Protos) in project and copy/create **greet.proto** proto file. Ensure (change) the namespace to the same as for the client.
- Select the client project node in *Solution Explorer*, select the context menu *Edit Project File*, and ensure (alternatively enter/correct) the in the file is included section:

```
<ItemGroup>
  <Protobuf Include="Protos\mygrpc.proto" GrpcServices="Client" />
</ItemGroup>
```

  Especially check if **`Client`** is set as `GrpcServices` value.

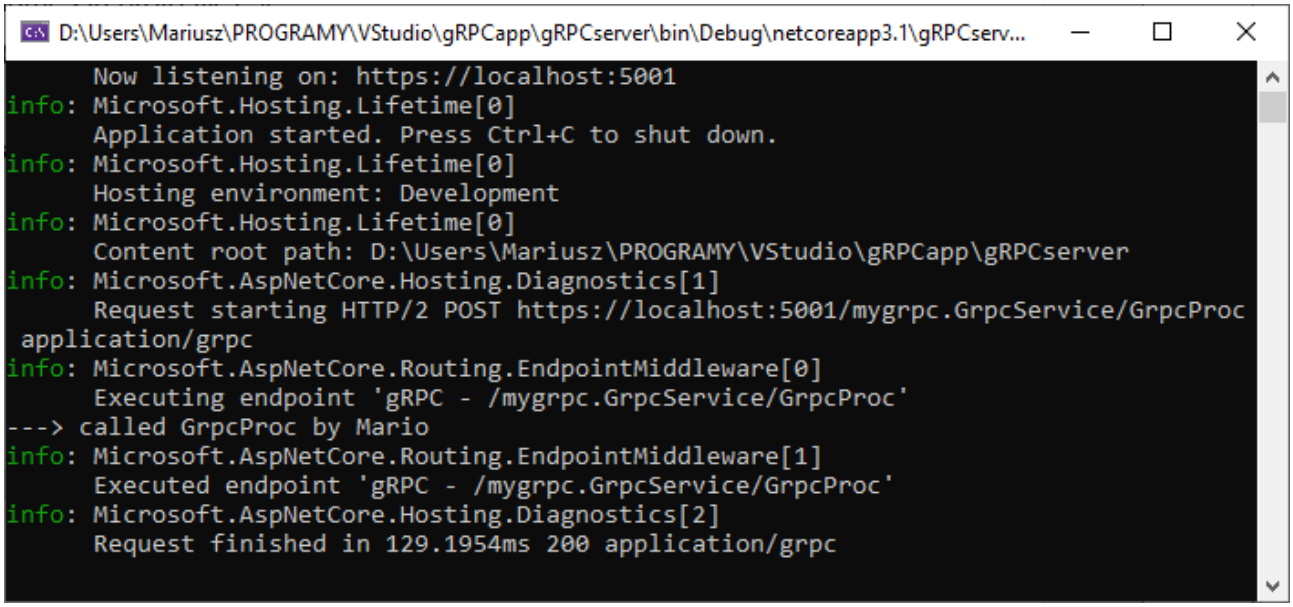
- Enter the code of client `Main` method in which you will:
  - create a channel for communication with the server (specifying the address and the same port as in server (here we run the server on localhost),
  - create client object – to make requests,
  - call the remote procedure asynchronously passing input parameters,
  - display results,
  - close the channel when no more used.

```
static async Task Main(string[] args)
{
  Console.WriteLine("Starting gRPC Client");
  using var channel = GrpcChannel.ForAddress("https://localhost:5001");
  var client = new GrpcService.GrpcServiceClient(channel);
  Console.Write("Enter the name: ");
  String str = Console.ReadLine();
  int val = 21;
  var reply = await client.GrpcProcAsync(new GrpcRequest { Name=str ,
                                                           Age=val});
  Console.WriteLine("From server: " + reply.Message);
  Console.WriteLine("From server: "+val+" years = "+reply.Days+" days");
  Console.WriteLine("Press any key to exit...");
  Console.ReadKey();
  channel.ShutdownAsync().Wait();
}
```

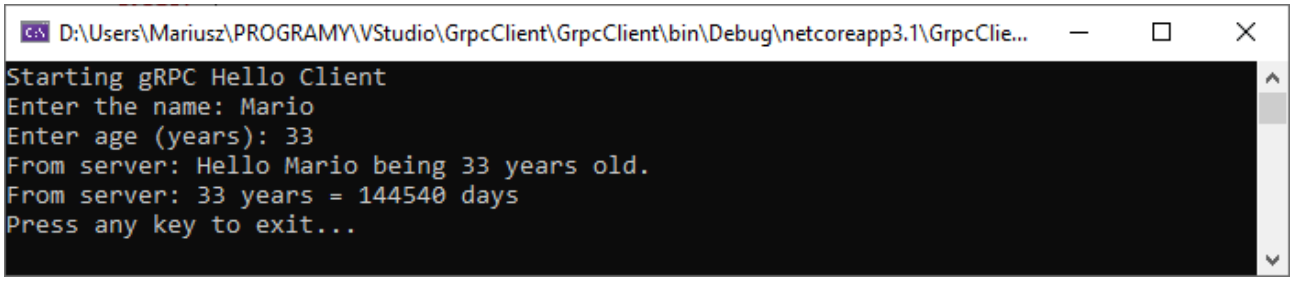
  The client class is generated from the interface defined in proto file. This class is defined in the class named as ***service name in proto file***. The client class name is ***service name in proto file*** tagged with ***Client*** word. Here the client object is built with the GrpcService.GrpcServiceClient class generated from proto file (names correspond to names in proto file).

1. Build the solution (compile/build all projects) and run the application (server first, and then client).
   - Check the operation of the application.
   - The result should be similar to below figures:

Server running in separate window (one process):



Client running in separate window (other process):



3.2 Visual Studio 2017

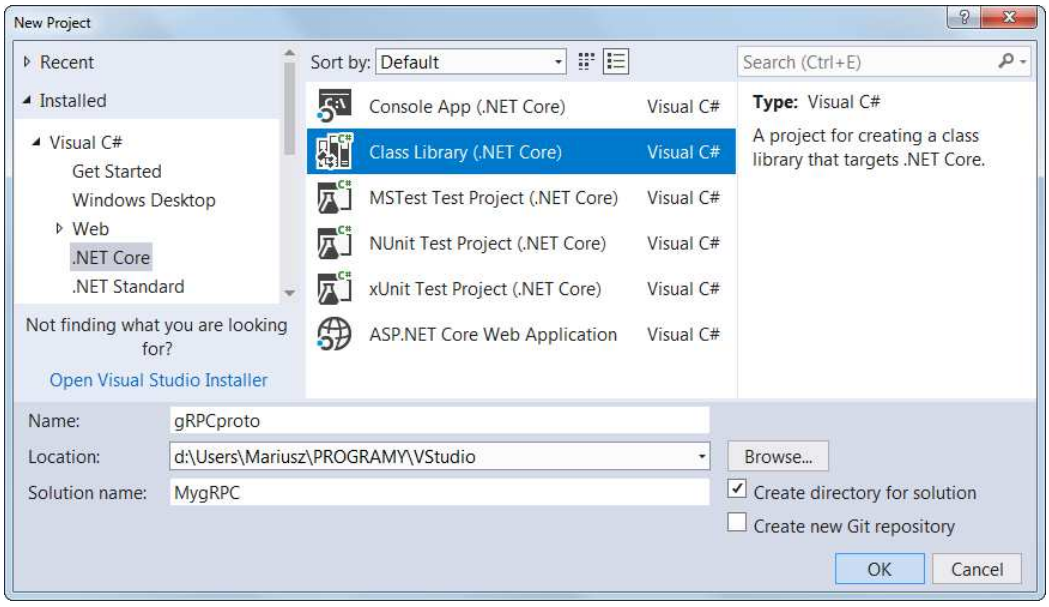
Visual Studio 2017 has got slightly worse, however still quite good support for building gRPC applications with help of NuGet packages management.

The following steps describe developing simple gRPC application (client and server) with VS 2017. In this instruction one solution with 3 projects will be created to develop simple gRPC server and gRPC client.

1. **Create a new C# project (new solution) for proto interface**.

   This will be for creating code of stubs for server and client – after compilation the proper source code for server and client will be generated.
   It can be used Class Library (.NET Core) or . Class Library (.NET Standard) project – both will work. Here we use the first one:

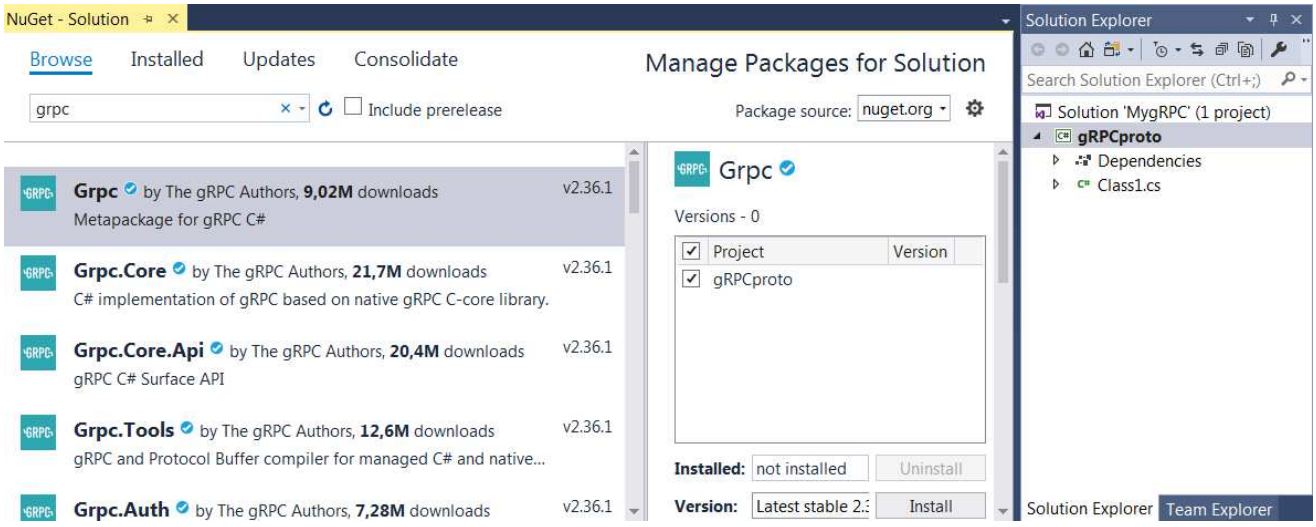
   - Create Class Library (.NET Core) project in a new solution (here named gRPCproto in MygRPC solution).



2. **Configure the project** (here named gRPCproto) **for generation of interface stubs**.

   In order to compile proto files it must be included several packages with use of NuGet manager.

   - Select the project in Solution Explorer window, and from platform menu or context menu select *Manage NuGet Packages* option.
   - Select *Browse* tab, enter **grpc** in *Search* field.



   - Select and add the following packages to the project:
     - Grpc (metapackage for gRPC) – this will also add other necessary packages,
     - Grpc.Tools (gRPC and Protocol Buffers compiler)
   - Enter **protobuf** in *Search* field.
     Select and add the following package:
     - Google.Protobuf (C# runtime library for Protocol Buffers)
   - You may delete initially created Class1.cs file – it will be not used.

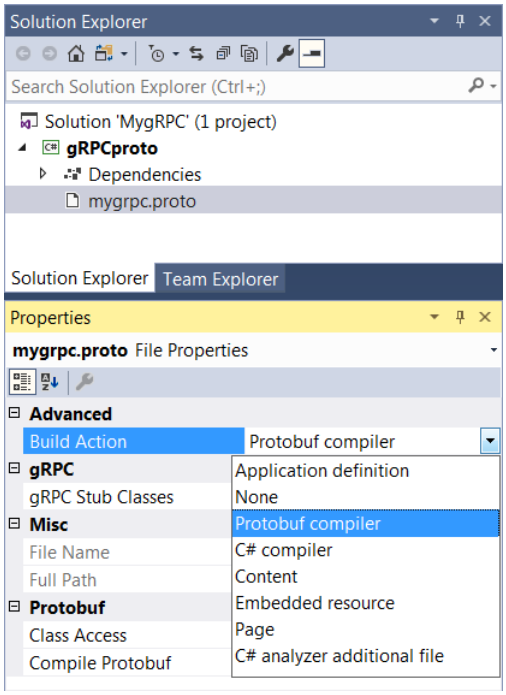
3. **Create proto file and interface defined in proto file**.

   Here the interface for creating stubs for server and client will be defined in proto file. The remote procedure **`addInt`** with two parameters that adds two integers will be defined. The procedure returns the result and some comment (string value).

   - Create manually or using platform options (*Add New Item* in menu) a text proto file with proto filename extension – here named mygrpc.proto).
   - Enter the following content that defines:
     - Protocol Buffers version,
     - service name, remote procedure name, output, and input parameters (messages),
     - input (request) message – parameters in call,
     - output (response) message – replay for call.

```
syntax = "proto3";
package mygrpcproto;
// Service definition.
service MyGrpcSrv {
     rpc addInt (AddIntRequest) returns (AddIntReply) {}
}
// The request message
message AddIntRequest {
  int32 num1 = 1;
  int32 num2 = 2;
}
// The response message
message AddIntReply {
  int32 result = 1;
  string comment = 2;
}
```

   - Select the proto file in *Solution Explorer* window and bellow in *Properties* window select in *Build Action* field the *Protobuf compiler* action.

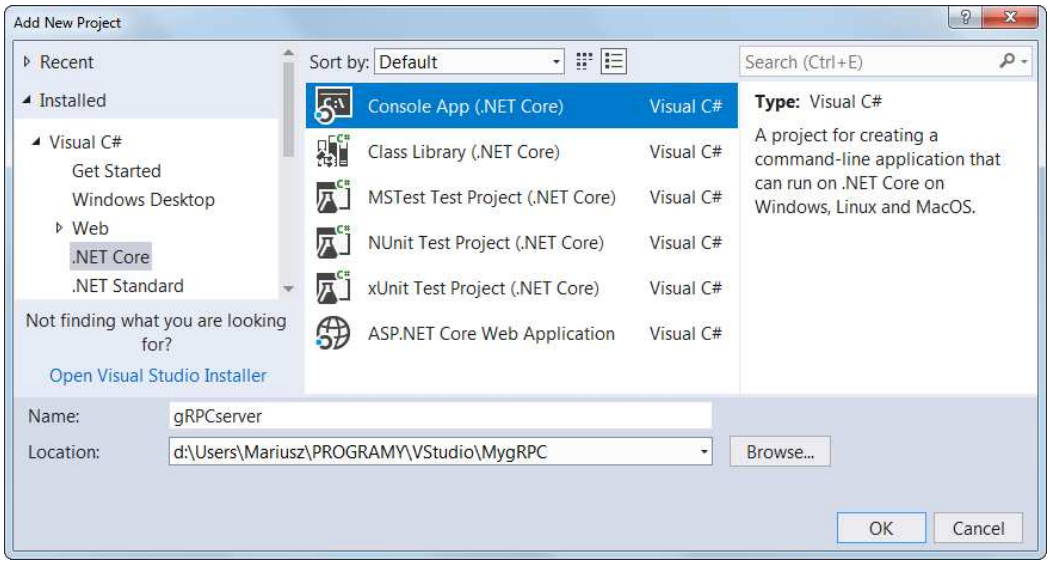


- You may compile (build option) the project to check that all works well.
  Pay attention for generated files with classes in **obj** directory of the project – among the others source code for server and client stubs.

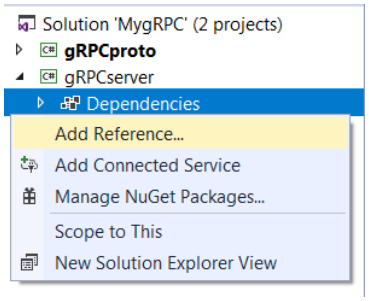
4. Create in the same solution the gRPC server project.

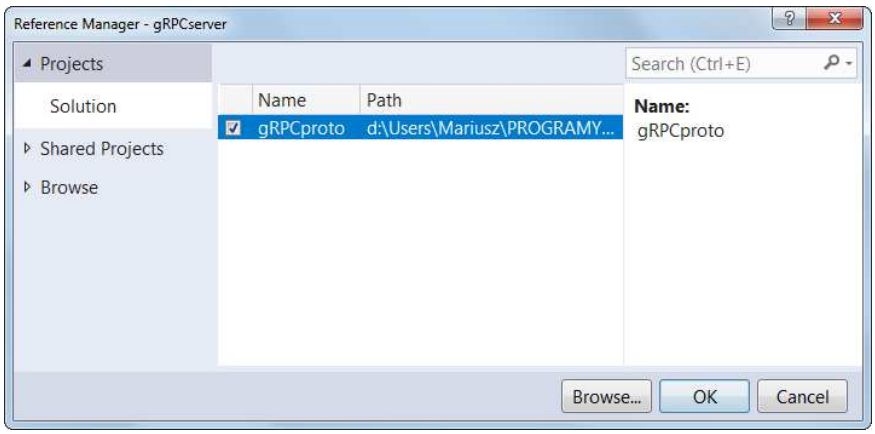
Here the server .Net console application will be created.

- Add to the solution the Console App (.NET Core) project – here named gRPCserver.



- Add the dependency (reference) to gRPCproto project (to use their generated classes). Select *Add Reference* from context menu (see the figure below), or edit *Project Dependences*.





- Add the class implementing the remote procedure (the interface defined in proto file). The class must extend the base class with the name of ***service name in proto file*** tagged with ***Base*** word. This class is defined in the class generated from proto file named as ***service name in proto file***.

```
namespace gRPCserver {
  class MyGrpcSrvImpl : MyGrpcSrv.MyGrpcSrvBase {
  public override Task<AddIntReply> addInt(AddIntRequest req,
       ServerCallContext ctx)  {
```

```
    string comment;
    int result;
    ...
    return Task.FromResult(new AddIntReply { Result = result,
 Comment = comment });
  } }
  ...
}
```

Set (calculate) `result` and `comment` values properly (e.g. comment the sign of the result).

Note:
- here the `Task<>` is used – this allow asynchronous processing,
- the `ServerCallContext` is not used here (is for potential future use).

- Add the code in Program class to create the server object and to run the server listening on given port:

```
namespace gRPCserver {
  ...
  class Program  {
  const int port = 10000;
  static void Main(string[] args)  {
    Console.WriteLine("Starting Hello gRPC server");
    Server myServer = new Server
    {
    Services = { MyGrpcSrv.BindService(new MyGrpcSrvImpl()) },
    Ports = { new ServerPort("localhost", port,
  ServerCredentials.Insecure) }
    };
    myServer.Start();
    Console.WriteLine("Hello gRPC server listening on port " + port);
    Console.WriteLine("Press any key to stop the server...");
    Console.ReadKey();
    myServer.ShutdownAsync().Wait();
  }
  }
}
```

Note:
- for the `Server` object the available services are defined (here our one service implementation) and ports where services calls come to,
- the services are created with `BindService` method,
- here we specify localhost and port=10000 as the procedure endpoint,
- for simplicity we use not secure connection (hint: on Mac OS there are problems to use secure connection for gRPC).

- Correct compiler warnings/errors properly (usually by importing definitions – with using keyword). Make a note that name for proto file (name of package) start with capital letter (not like in name in proto file).

5. Create in the same solution the gRPC client project.

Here the client .Net console application will be created.

- Add to the solution the new Console App (.NET Core) project – here named gRPCclient.
- Add the dependency (reference) to gRPCproto project (to use their generated classes). Select *Add Reference* from context menu.
- Implement in `Main` method:
  - Creating a channel which will be used to communicate with server
    - specifying the address and port (here we run the server on localhost),
    - specifying no security mechanisms (for simplicity).
  - Creating client object – to make requests.

    The client class is generated from the interface defined in proto file. This class is defined in the class named as ***service name in proto file***. The client class name is ***service name in proto file*** tagged with ***Client*** word.
  - Calling remote procedure – here named **addInt** – with use of **AddIntRequest** object (from the class generated from proto file).
  - Closing the channel when no more used.

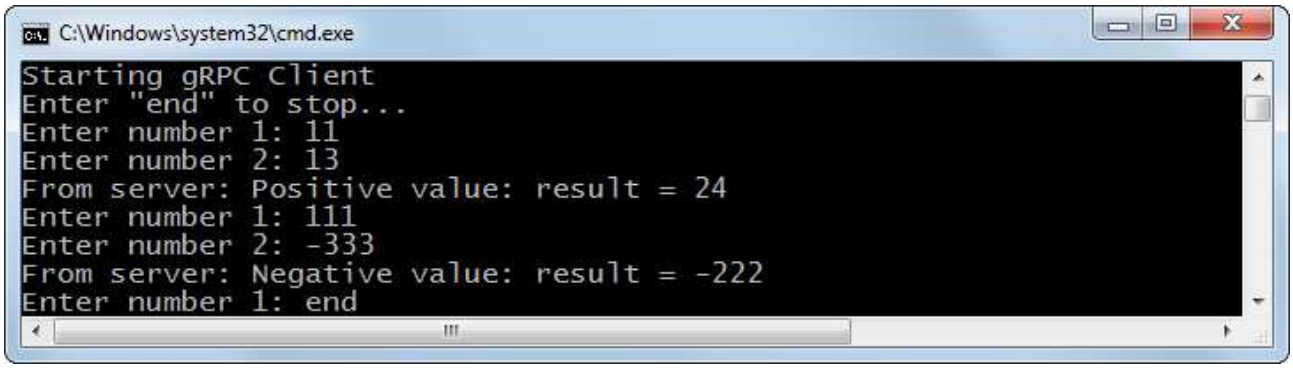
```
static void Main(string[] args)  {
  Console.WriteLine("Starting gRPC Client");
  Channel channel = new Channel("127.0.0.1:10000",
ChannelCredentials.Insecure);
  var client = new MyGrpcSrv.MyGrpcSrvClient(channel);
  String str;
  int num1, num2;
  Console.Write("Enter number 1: ");
  str = Console.ReadLine();
  if (int.TryParse(str, out num1))
  {
  Console.Write("Enter number 2: ");
  str = Console.ReadLine();
  if (int.TryParse(str, out num2))
  {
    var reply = client.addInt(new AddIntRequest { Num1 = num1,
  Num2 = num2 });
    Console.WriteLine("From server: " + reply.Comment + reply.Result);
  }
  else
    Console.WriteLine("Wrong value!");
```

```
        }
        else
        Console.WriteLine("Wrong value!");
        Console.WriteLine("Stopping gRPC Client");
        channel.ShutdownAsync().Wait();
    }
```

6. Build the solution (compile/build all projects).
   - Check the operation of the application.
   - The result should be similar to below figures:

   Server running in separate window (one process):

   Client running in separate window (other process):

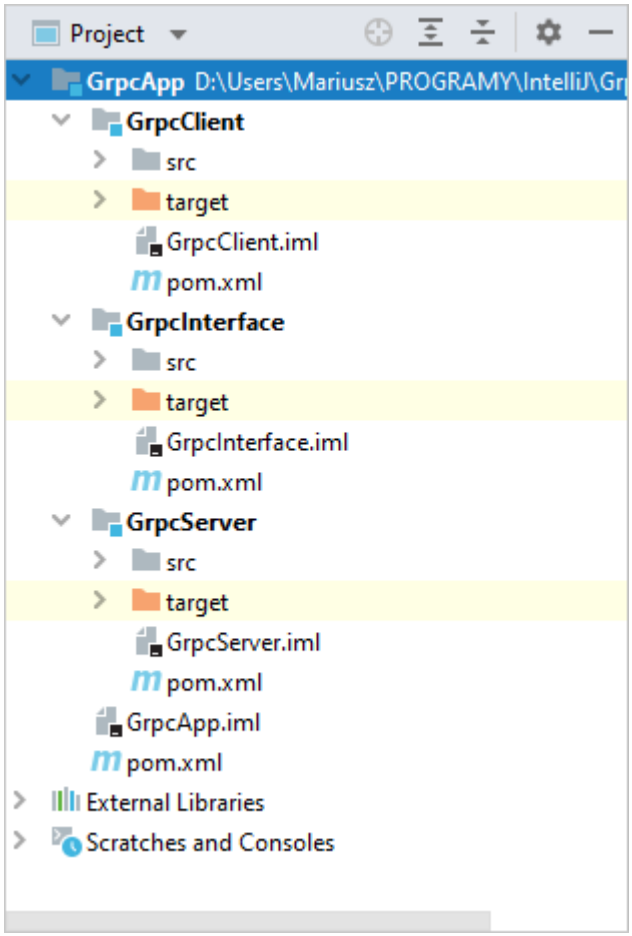


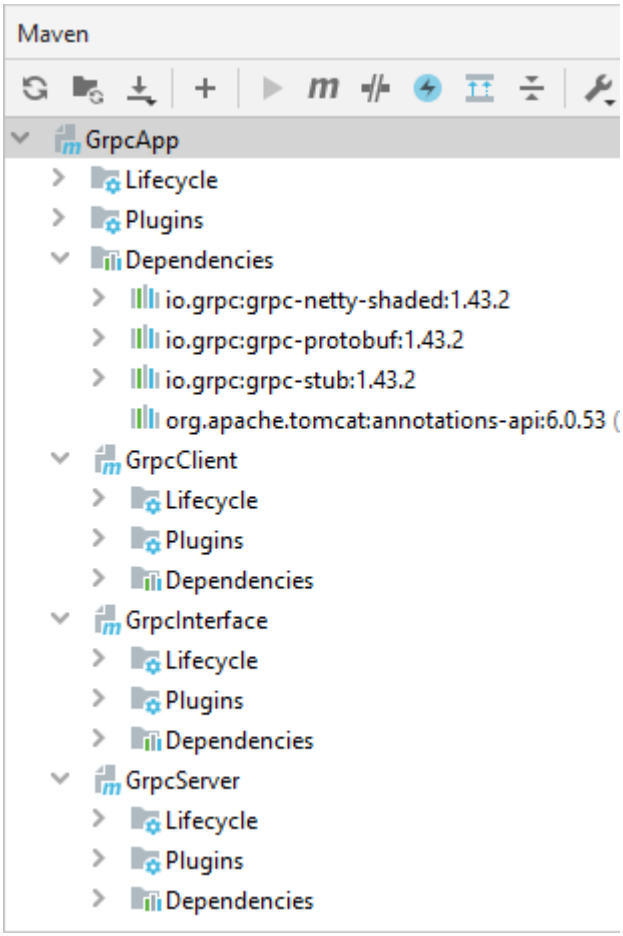
Note:
After proper creating and building the solution (all projects), the VS 2107 platform sometimes shows (after re-opening the solution (projects)) that errors exist for uses directive (as if proto namespace was not visible and classes generated from it). Anyway all works well (after some time errors may disappear) and the solution can be rebuilt without errors. This behavior of Visual Studio 2017 is strange and hard to explain.

3.3 Java based platform – maven project

Detailed description will be completed later.

3.3.1 General project stucture





3.3.2 POMs

Main POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>pl.edu.pwr.wit.rsi</groupId>
    <artifactId>GrpcApp</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>GrpcServer</module>
        <module>GrpcClient</module>
        <module>GrpcInterface</module>
    </modules>
```

```
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</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>

    <dependencies>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-netty-shaded</artifactId>
            <version>${grpc.version}</version>
        </dependency>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-protobuf</artifactId>
            <version>${grpc.version}</version>
        </dependency>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-stub</artifactId>
            <version>${grpc.version}</version>
        </dependency>
        <dependency> <!-- necessary for Java 9+ -->
            <groupId>org.apache.tomcat</groupId>
            <artifactId>annotations-api</artifactId>
            <version>6.0.53</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

</project>
```

Interface POM:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>GrpcApp</artifactId>
        <groupId>pl.edu.pwr.wit.rsi</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>GrpcInterface</artifactId>

    <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>
                            <goal>test-compile</goal>
                            <goal>test-compile-custom</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>

        </plugins>
    </build>

</project>
```

Server POM:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>GrpcApp</artifactId>
        <groupId>pl.edu.pwr.wit.rsi</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>GrpcServer</artifactId>

    <dependencies>
        <dependency>
            <groupId>pl.edu.pwr.wit.rsi</groupId>
            <artifactId>GrpcInterface</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <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>
            </plugin>
        </plugins>
    </build>

</project>
```

Client POM:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>GrpcApp</artifactId>
        <groupId>pl.edu.pwr.wit.rsi</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>GrpcClient</artifactId>

    <dependencies>
        <dependency>
            <groupId>pl.edu.pwr.wit.rsi</groupId>
            <artifactId>GrpcInterface</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <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>GrpcClient</mainClass>
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

3.3.3 Proto file

In main\proto directory/node.

```
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;
}
```

3.3.4 Server code

```
public class GrpcServer {
    public static void main(String[] args) {
        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();
        }
    }

    static class GrpcServiceImpl extends GrpcServiceGrpc.GrpcServiceImplBase {
        public void grpcProcedure(GrpcRequest req,
                                  StreamObserver<GrpcResponse> responseObserver) {
            String msg;
            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).build();
            responseObserver.onNext(response);
            responseObserver.onCompleted();
        }
    }
}
```

3.3.5 Client code

```
public class GrpcClient {
    public static void main(String[] args) {
        String address = "localhost";
        int port = 50001;
        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("Mariusz")
                .setAge(21)
                .build();
        GrpcResponse response = stub.grpcProcedure(request);
        System.out.println(response);
        channel.shutdown();
    }
}
```

4 Exercise – Part II

The detailed and final requirements for Part II are set by the teacher.

A. Develop or prepare to develop a program according to the teacher's instructions.