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 2017 or 2019 (eventually Java programming platform (also with SpringBoot)),
- Proper NuGet packages.

3 Exercise – Part I Creating simple gRPC application project.

3.1 Visual Studio

Visual Studio 2017 has got quite good support for building gRPC applications with help of NuGet packages management. Visual Studio 2019 has got dedicated gRPC app project.

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 2017.

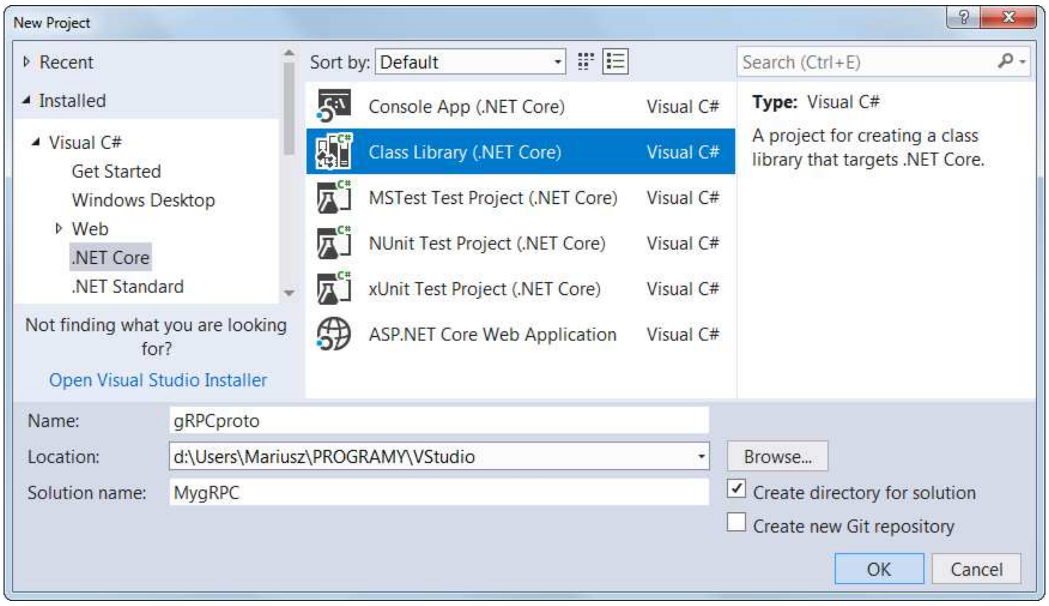
The following steps describe developing simple gRPC application (client and server) with VS 2017.

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).


New Project
Recent
Installed
Visual C#
Get Started
Windows Desktop
Web
.NET Core
.NET Standard
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
Console App (.NET Core) Visual C#
Class Library (.NET Core) Visual C#
MSTest Test Project (.NET Core) Visual C#
NUnit Test Project (.NET Core) Visual C#
xUnit Test Project (.NET Core) Visual C#
ASP.NET Core Web Application Visual C#
Browse...
OK
Cancel

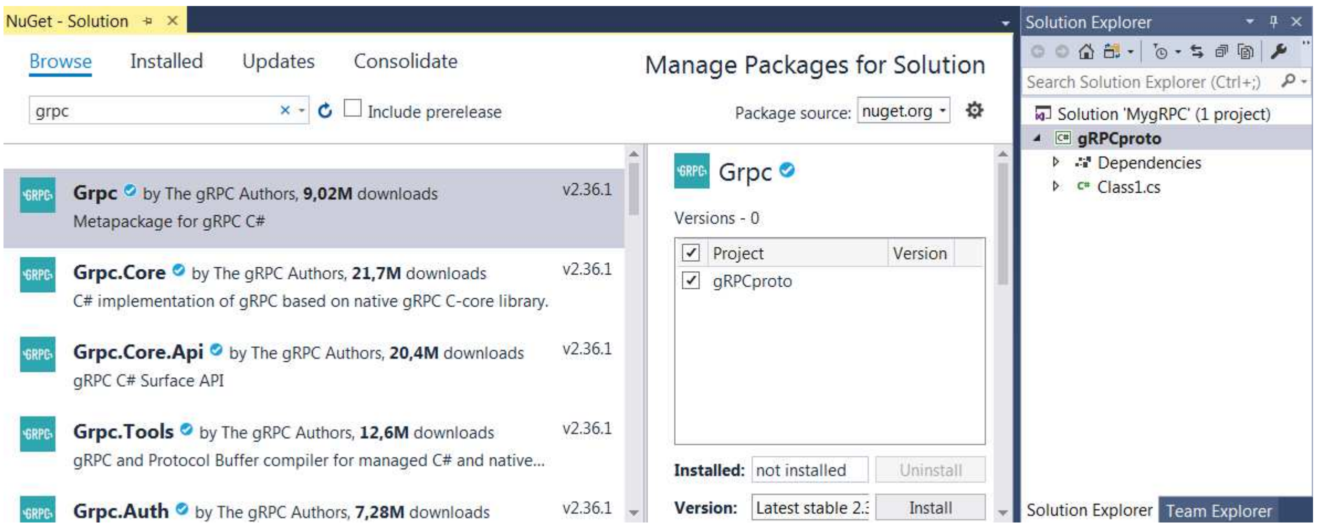
Type: Visual C#
A project for creating a class library that targets .NET Core.

Name: gRPCproto
Location: d:\Users\Mariusz\PROGRAMY\VStudio
Solution name: MygRPC
[✓] Create directory for solution
[ ] Create new Git repository

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.



NuGet - Solution

Browse | Installed | Updates | Consolidate

Manage Packages for Solution

grpc — Include prerelease — Package source: nuget.org

Grpc by The gRPC Authors, 9,02M downloads — v2.36.1
Metapackage for gRPC C#

Grpc.Core by The gRPC Authors, 21,7M downloads — v2.36.1
C# implementation of gRPC based on native gRPC C-core library.

Grpc.Core.Api by The gRPC Authors, 20,4M downloads — v2.36.1
gRPC C# Surface API

Grpc.Tools by The gRPC Authors, 12,6M downloads — v2.36.1
gRPC and Protocol Buffer compiler for managed C# and native...

Grpc.Auth by The gRPC Authors, 7,28M downloads — v2.36.1

Grpc

Versions - 0

Project	Version
✓ gRPCproto	

Installed: not installed — Uninstall

Version: Latest stable 2.3 — Install

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'MygRPC' (1 project)
- gRPCproto
  - Dependencies
  - Class1.cs

Solution Explorer | Team Explorer

   - 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.

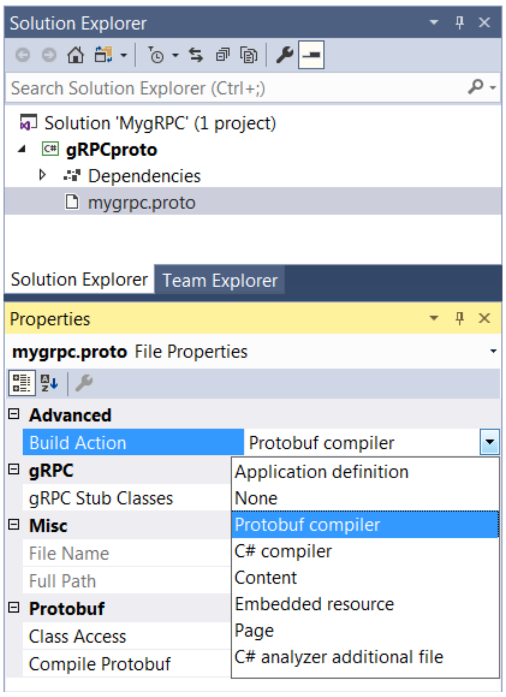
3. **Create proto file and interface defined in proto file**.

   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.

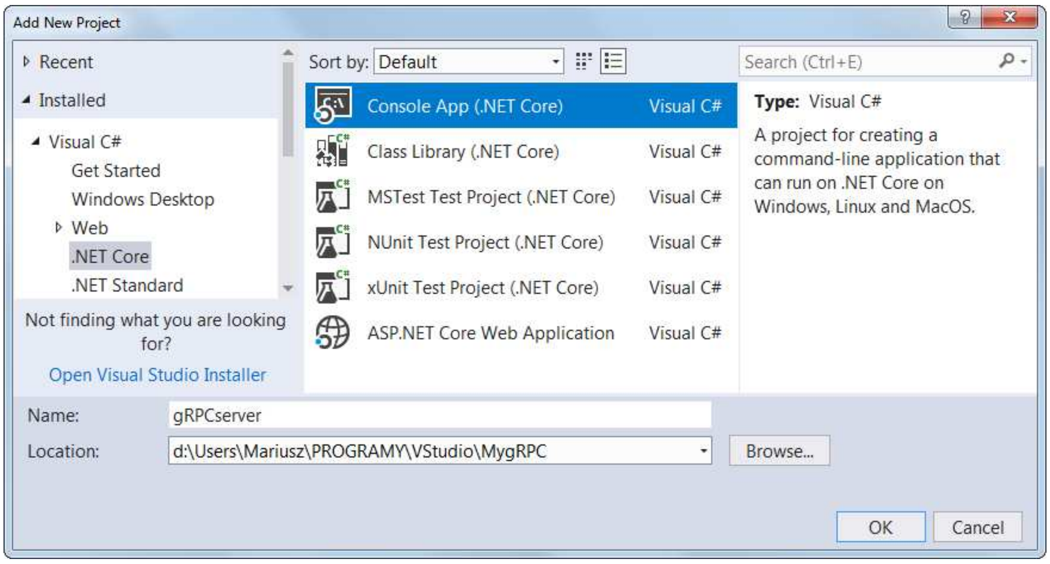


- You may compile (build option) the project to check that all works well.
  Pay attention for generated files with classes in **obj** directory of the project – among the others source code for server and client stubs.

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 New Project
Recent
Installed
Visual C#
Get Started
Windows Desktop
Web
.NET Core
.NET Standard
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Console App (.NET Core) Visual C#
Class Library (.NET Core) Visual C#
MSTest Test Project (.NET Core) Visual C#
NUnit Test Project (.NET Core) Visual C#
xUnit Test Project (.NET Core) Visual C#
ASP.NET Core Web Application Visual C#
Search (Ctrl+E)
Browse...
OK
Cancel

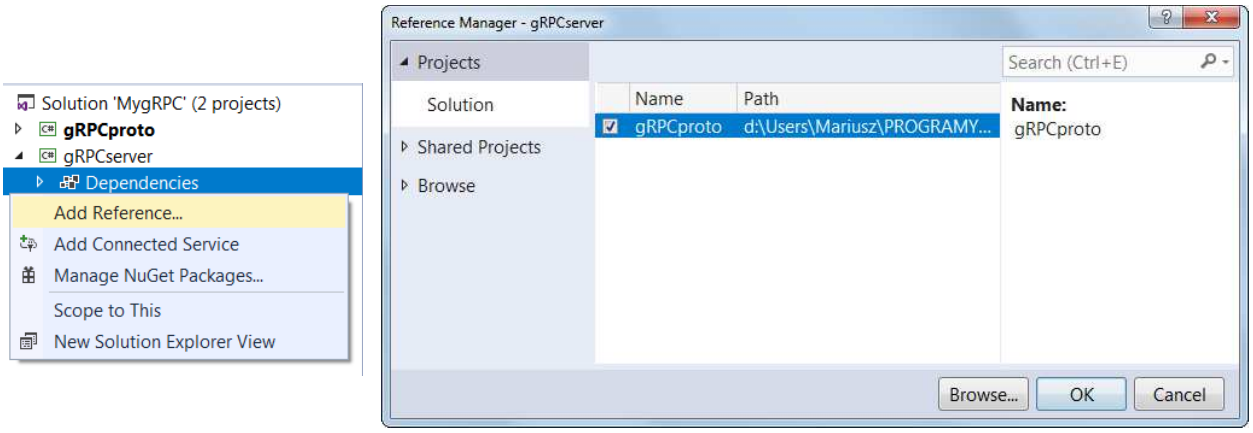
Type: Visual C#

A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.

Name: gRPCserver

Location: d:\Users\Mariusz\PROGRAMY\VStudio\MygRPC

- 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*.



Name	Path
☑ gRPCproto	d:\Users\Mariusz\PROGRAMY...

Name: gRPCproto

- 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.

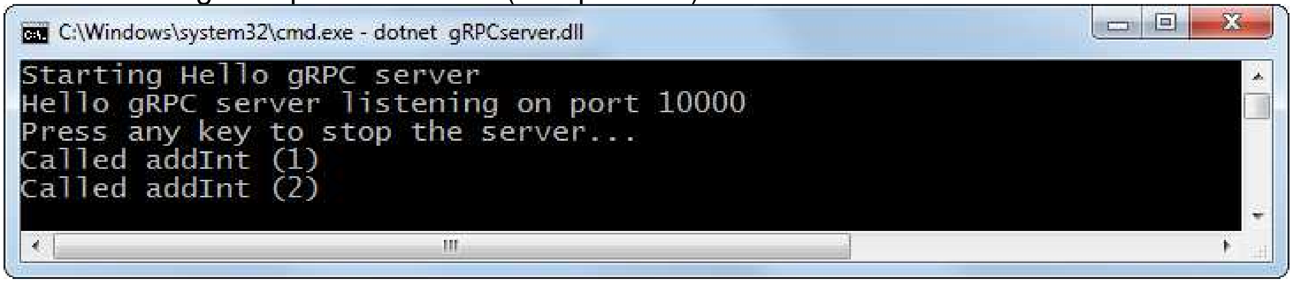
- Add the dependency (reference) to gRPCproto project (to use their generated classes). Select *Add Reference* from context menu.
- Implement in `Main` method:
  - Creating a channel which will be used to communicate with server
    - specifying the address and port (here we run the server on localhost),
    - 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();
}
```

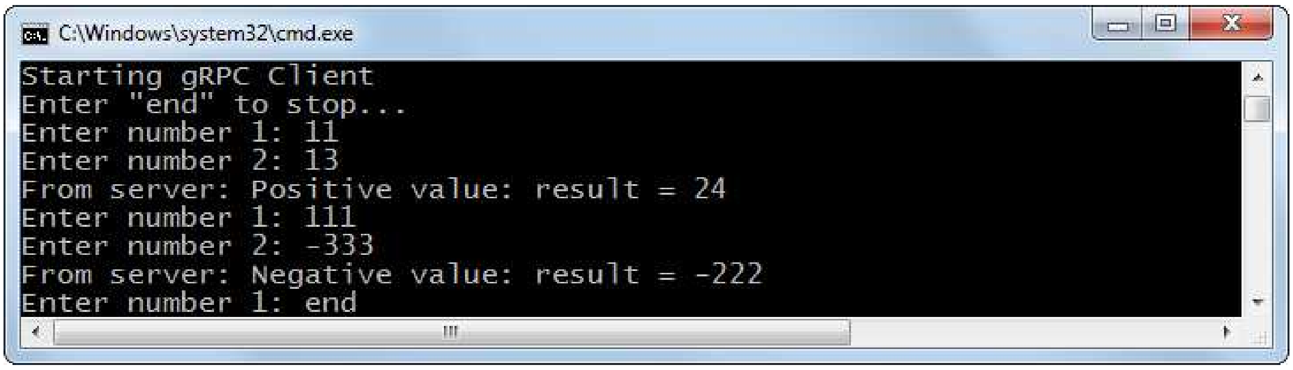
6. Build the solution (compile/build all projects).
   - Check the operation of the application.
   - The result should be similar to below figures:

Server running in separate window (one process):



```
Starting Hello gRPC server
Hello gRPC server listening on port 10000
Press any key to stop the server...
Called addInt (1)
Called addInt (2)
```

Client running in separate window (other process):



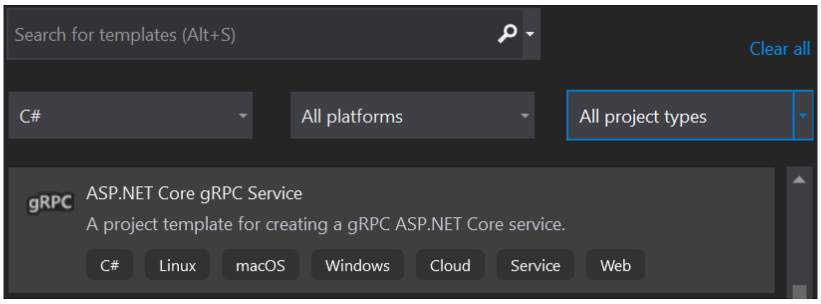
```
Starting gRPC Client
Enter "end" to stop...
Enter number 1: 11
Enter number 2: 13
From server: Positive value: result = 24
Enter number 1: 111
Enter number 2: -333
From server: Negative value: result = -222
Enter number 1: end
```

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.

B. Visual Studio 2019.

Visual Studio 2019 has got dedicated gRPC app project. It is recommended also to use several different NuGet packages.

1. **Creation of C# project (new solution) for gRPC service (server)**.
   - To create new gRPC app select *ASP.NET Core gRPC Service*. You can use Search field find the project faster.


Search for templates (Alt+S)
Clear all
C#
All platforms
All project types
gRPC
ASP.NET Core gRPC Service
A project template for creating a gRPC ASP.NET Core service.
C#
Linux
macOS
Windows
Cloud
Service
Web

After selection of names (in next window) you can specify *Target Framework* (default is .NET core 3.1), and eventually enable *Docker* option (don't do it now).

2. **The project structure**

The initial default project consist of:

- the initial proto file – **greet.proto**,
- the initial gRPC service - **GreeterService.cs**,
- the service configuration code – **Startup.cs**,
- the code for service hosting (by default Kestrel web server is used) – **Program.cs**,
- **appSettings.json** – contains configuration data
- the necessary dependences (including NuGet packadges).

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*, and *GrpcASPNetCore.Server* 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 specified by calling the `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. The gRPC service (gRPC procedure)

- The service code is similar as described for VS 2017.
- The proto file format is obviously exactly the same (but other operation and messages are defined). Here, ensure that namespace in proto file is the same as for gRPC service implementation.

4. The gRPC client

The client can be developed similarly. Here different building channel will be presented and calling the service procedure asynchronously. The difference is using one different NuGet package.

- Add to the solution (or create in new solution) a new *Console App (.NET Core)* project.
- Add (similarly as for VS 2017) NuGet packages:
  - Grpc.Net.Client – note: here is the difference,
  - Grpc.Tools,
  - Google.Protobuf.

- 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\greet.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 the channel for communication with server (use the same port number as in server),
  - create client,
  - call the remote procedure asynchronously.

```
static async Task Main(string[] args)
{
  Console.WriteLine("Starting gRPC Hello Client");
  using var channel = GrpcChannel.ForAddress("https://localhost:5001");
  var client = new Greeter.GreeterClient(channel);
  String str = Console.ReadLine();
  var reply = await client.SayHelloAsync(new HelloRequest { Name = str });
  Console.WriteLine("From server: " + reply.Message);
  Console.WriteLine("Press any key to exit...");
  Console.ReadKey();
}
```

  The class and message names are as the default ones in project.

5. Build the solution (compile/build all projects) and run the application (server first, and then client).
   - Check the operation of the application.

3.2 Visual Studio Code

Maybe will be completed later.

3.3 Java based platform

Maybe will be completed later.

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.