Discussion about advantages of choosing gRPC over REST based architecture



Photo by [https://cncf-branding.netlify.app](https://cncf-branding.netlify.app/)

REST is architectural style which has been a de facto while designing large scale systems these days. REST idealogy disrupted the market around 15 years back replacing SOAP based architectures because of its high performance, lighter and flexible nature. Yes of course there are many more reason to it. But its been a long time while developers tried hard to struggle with few problems while using REST such as costly JSON objects, streaming back the response from server and lack of support for HTTP 2.

**Why gRPC**

As the world is getting more and more connected, much more data load needs to catered by our giant servers and in this, Google came up with a faster and more power efficient architectural framework for communication called gRPC.

As the name suggests, it is an extension of traditional remote procedure calls (RPC). gRPC is designed very thoughfully to overcome pain points of REST based architectures with following advantages

1. gRPC is based on HTTP 2 which allows multiplexing over the network, gRPC is even superior than similar solutions like Thrift because of the above mentioned reasons
2. gRPC uses serialised protocol buffers which consume less network bandwidth than JSON while data transfers. Protocol buffer’s contract definition language (**proto3** as of today) is technology agnostic
3. gRPC supported bidirectional streaming of data over a single RPC call. Means server can send a stream of data back to client efficiently unlike REST’s *hit and get* nature.
4. gRPC’s implementation is time consuming as compare to REST but it is 7 times faster than REST in receiving and 10 times faster than REST in sending data.

**Support for gRPC**

gRPC is a relatively new approach than REST. Hence, not every technology or tool currently support gRPC but most of the do. gRPC is available with following languages as of today

* [C# / .NET](https://grpc.io/docs/languages/csharp/)
* [C++](https://grpc.io/docs/languages/cpp/)
* [Dart](https://grpc.io/docs/languages/dart/)
* [Go](https://grpc.io/docs/languages/go/)
* [Java](https://grpc.io/docs/languages/java/)
* [Kotlin](https://grpc.io/docs/languages/kotlin/)
* [Node](https://grpc.io/docs/languages/node/)
* [Objective-C](https://grpc.io/docs/languages/objective-c/)
* [PHP](https://grpc.io/docs/languages/php/)
* [Python](https://grpc.io/docs/languages/python/)
* [Ruby](https://grpc.io/docs/languages/ruby/)

Moreover gRPC APIs use their own Protoc compiler which allows you to create your own code. It works in multiple languages and can be used in polyglot environments. This refers to groups of microservices that run on separate platforms and are coded in multiple languages.

**gRPC with REST**

For enterprise , most of their current products and cloud solutions work fine with REST and that is a big reason for their resistance to migrate to gRPC. But there are solutions out there which allows you run gRPC with REST approach such as **gRPC-Gateway.**

As per definition, gRPC-Gateway is a plugin of [protoc](https://github.com/protocolbuffers/protobuf" \t "_blank). It reads a [gRPC](https://grpc.io/" \t "_blank) service definition and generates a reverse-proxy server which translates a RESTful JSON API into gRPC.

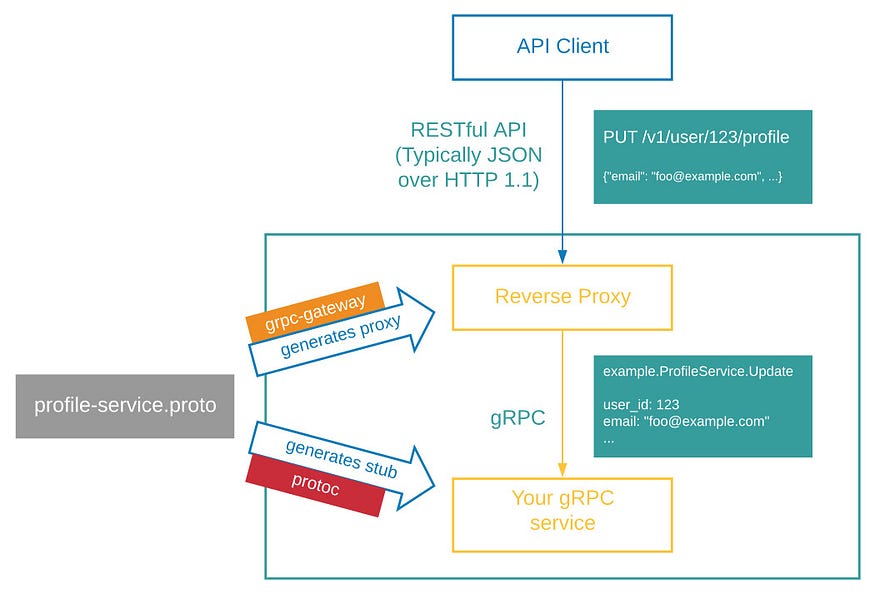


Photo by https://grpc-ecosystem.github.io

All thanks to gRPC-Gateway, you can now support gRPC over your existing REST based services and enjoy benefits of both.

**By the way, incase you are a fan of Microservices, you can check out this article to have a new viewpoint on it.**

[Why Microservices are not always the best choice](https://medium.com/geekculture/why-microservices-is-not-always-the-best-choice-df5b3b10babb)

**Conclusion**

In the tech community, more we work more we tends to fall in the trap of ‘*the best approach possible*’ stuff while solving the problem. Even the most experience guys talks like the advocate of REST. Without getting technologically polar, I would like to conclude gRPC can be used along with REST in some cases to get maximum advantage of both.

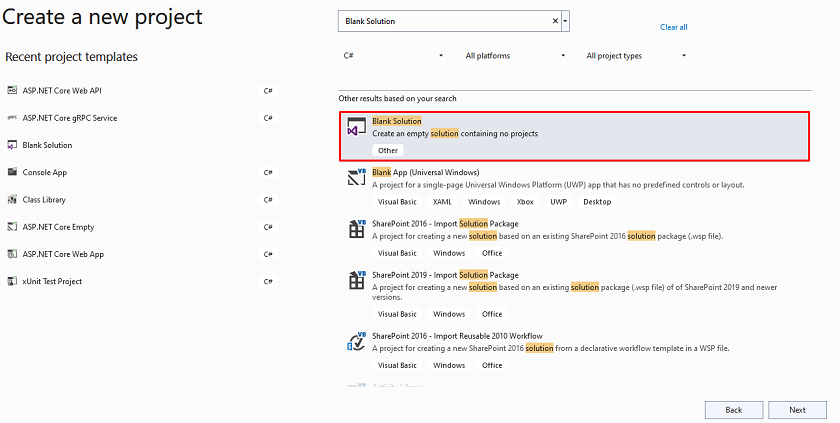
“ gRPC’s major strength comes from the underlying HTTP 2 support which makes it 7 times faster than REST in receiving data and 10 times faster to send the data. While designing a solution which revolves around the expectations of 5 nines (99.999), 7–10x improvement is game changer. “

Growth of support and community for gRPC is in our hands as techies in the end. **Hope gRPC can save a lot of I/O (s) for you in furture.**

**mplementation of ProductOfferGrpcService Service**

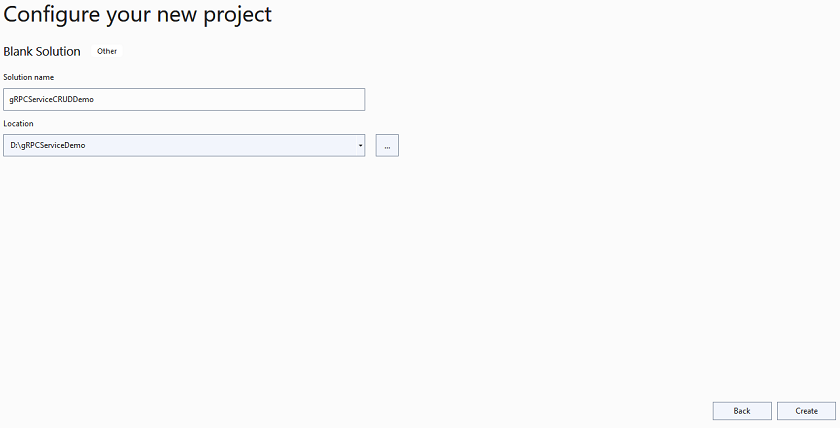
**Step 1**

Create a new Blank Solution



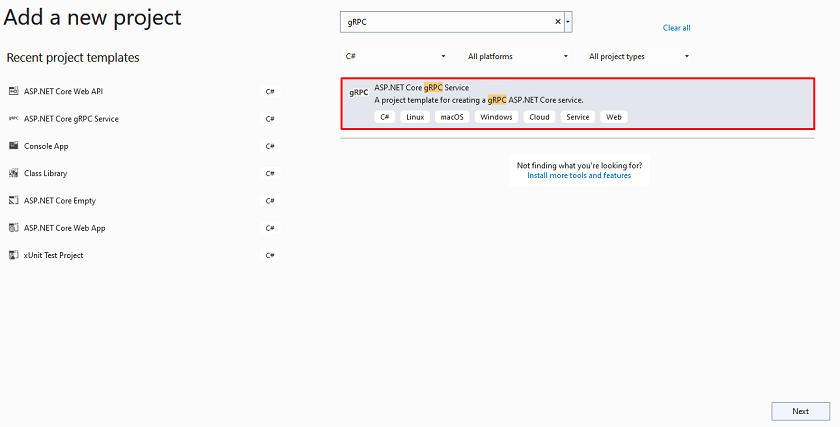
**Step 2**

Configure Project



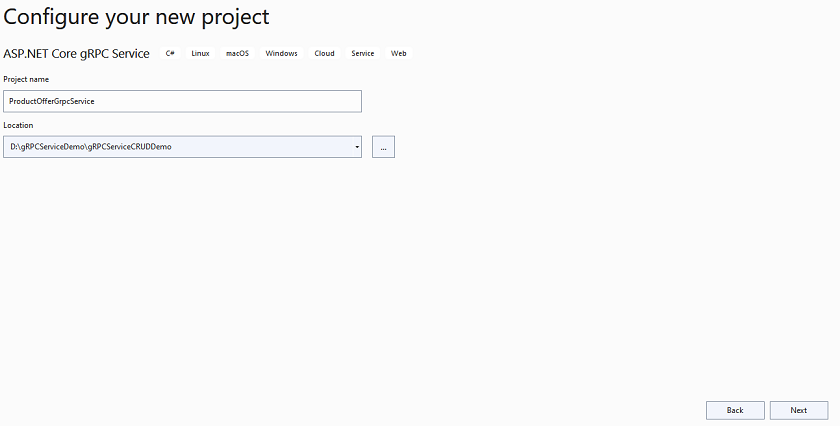
**Step 3**

Add a new gRPC Service Project inside the Blank Solution



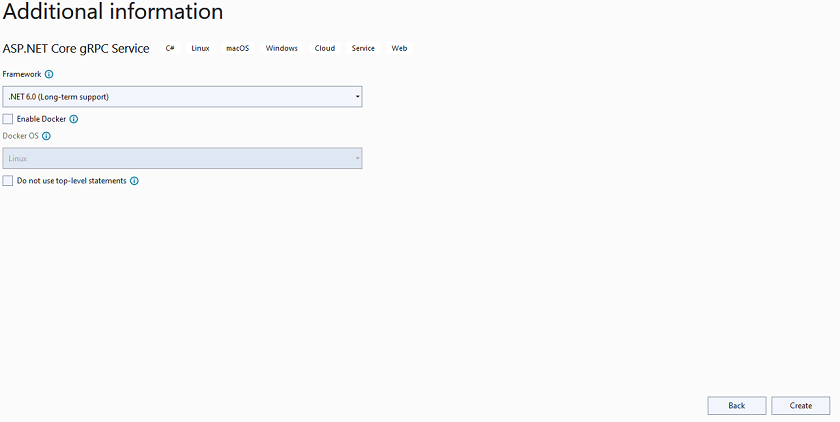
**Step 4**

Configure your new project



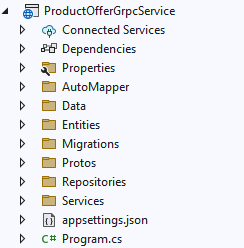
**Step 5**

Provide additional information



Remove default protobuf and service file from the project

**Project Structure**



**Step 6**

Create Offer Class inside the Entities

**Step 7**

Next, Create a new DbContextClass inside the Data folder

**Step 8**

Later on, create IProductOfferService and ProductOfferService inside the Repositories folder

**IProductOfferService**

**ProductOfferService**

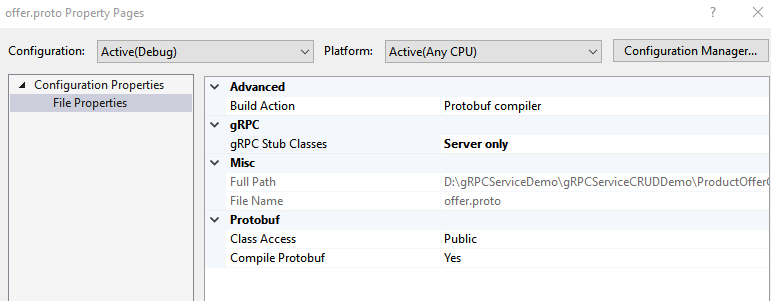
**Step 9**

Create OfferMapper inside AutoMapper folder

**Step 10**

Next, create a new offer proto file inside Proto

Also, make sure proto file properties are correct as I showed below and if that will be correct then build your project



**Step 11**

Add a new Offer Service inside Services

**Step 12**

Configure the database connection string inside the app settings file

**Step 13**

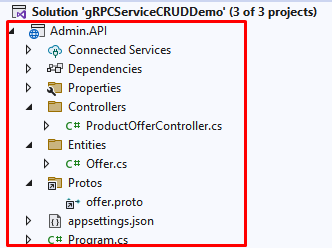
Register and configure a few services inside the Program class

**Implementation of Admin.API Microservice**

**Step 1**

Create a new Admin.API Web API Project

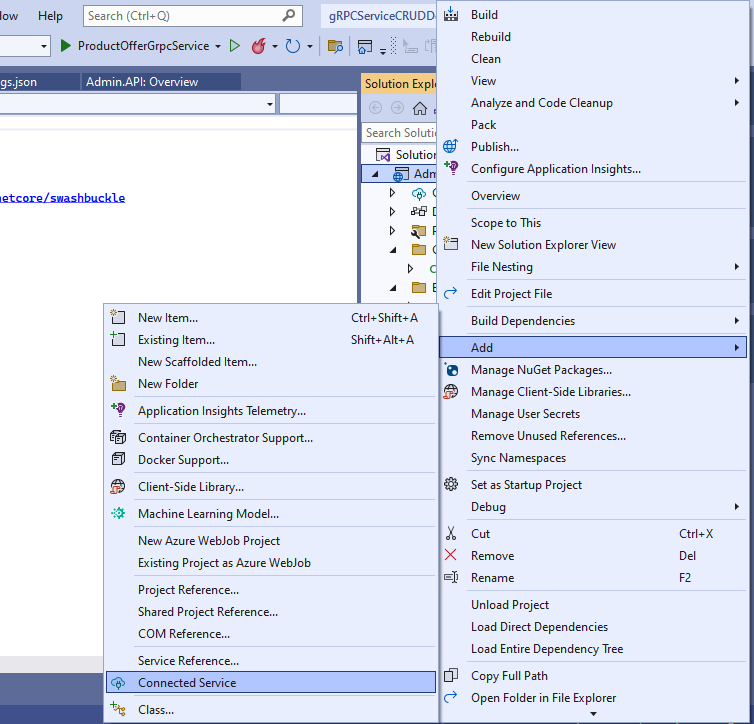
**Project Structure**



**Step 2**

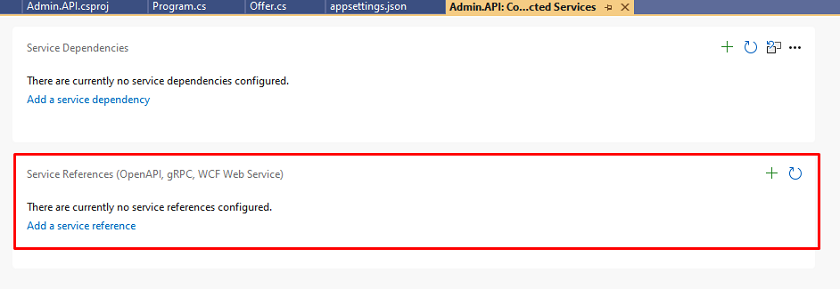
Connect the **ProductOfferGrpcService**

Right-Click on Admin.API and click on Connected Service in Add section



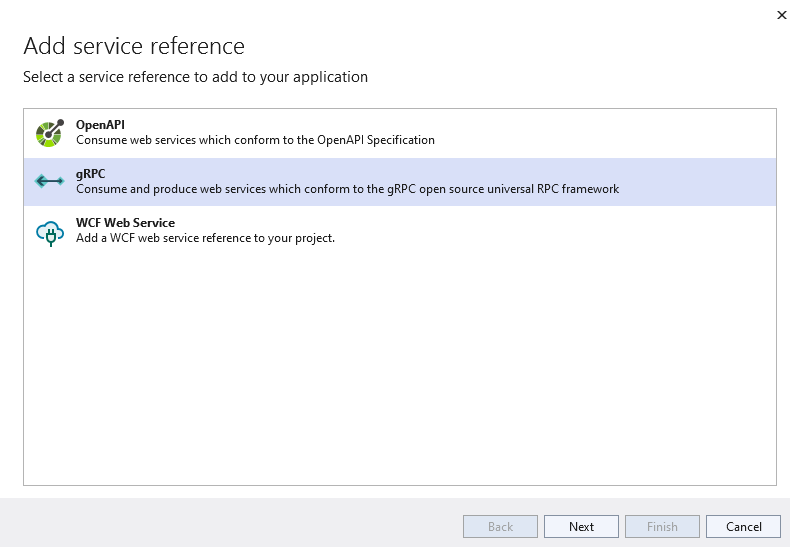
**Step 3**

Add a new gRPC Service



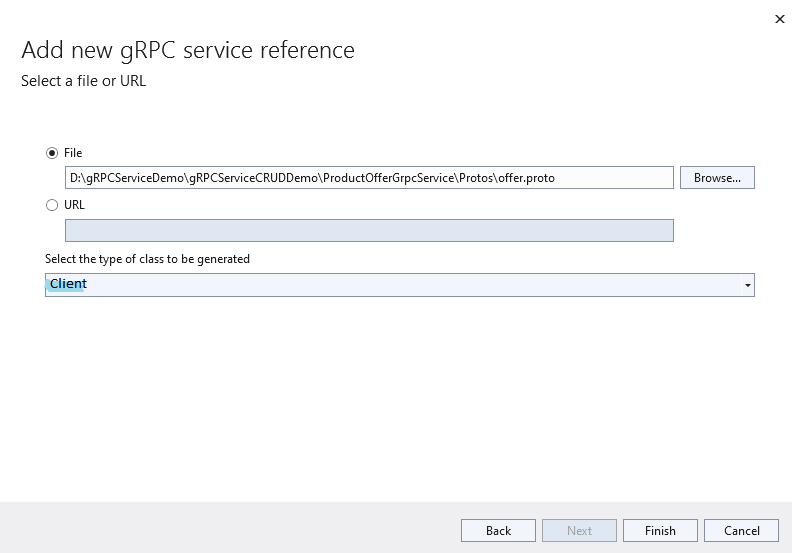
**Step 4**

Click on gRPC



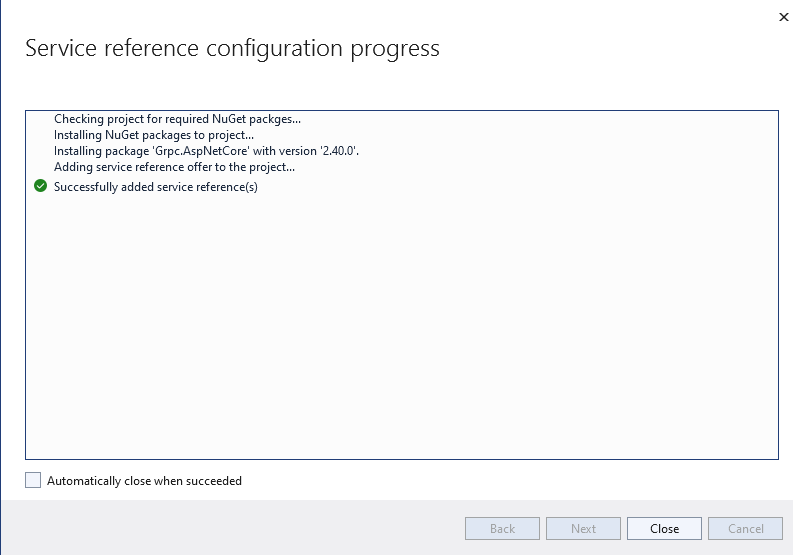
**Step 5**

Provide the protobuf file path and make sure your type of class is also correct



**Step 6**

Click on finish it will configure all things



**Step 7**

Create the Offer Class inside Entities

**Step 8**

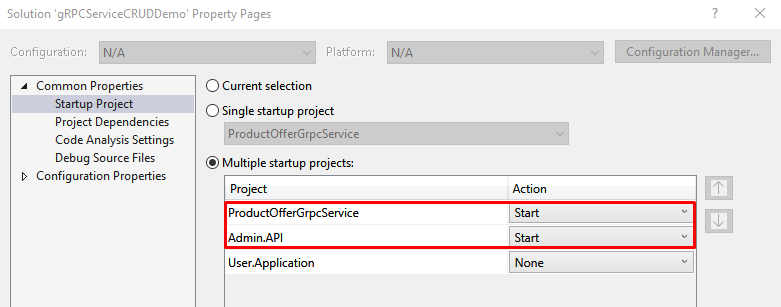
Next, add ProductOfferController

**Step 9**

Configure the gRPC Service URL inside the app settings file

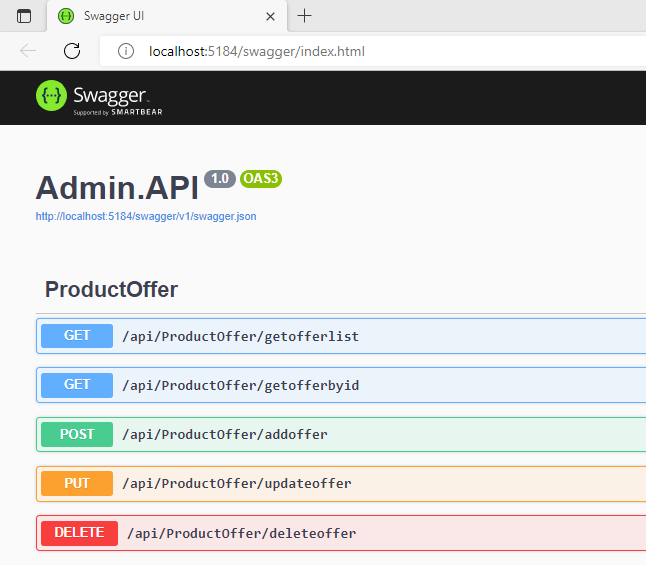
**Step 10**

Right-Click on the solution and set both projects as a startup project



**Step 11**

Finally, run your project



Here you can use endpoints to manage offers through gRPC Service

**Implementation of User Console Application**

We are going to create a console application to get a list of offers that which admin adds through gRPC

**Step 1**

First, we create a new proto file inside **ProductOfferGrpcService**named as user offer proto file

**Step 2**

Create UserOfferMapper inside AutoMapper of**ProductOfferGrpcService**

**Step 3**

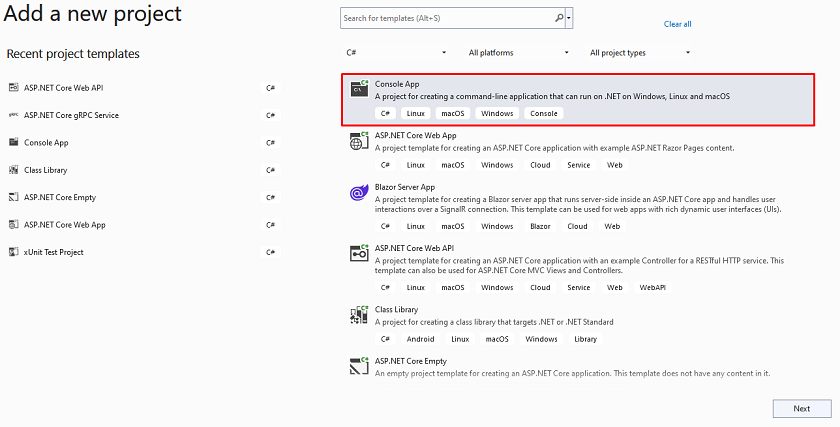
Next, add a new UserOfferService inside the Services of **ProductOfferGrpcService**

**Step 4**

Configure UserOfferService inside the Program class of **ProductOfferGrpcService**

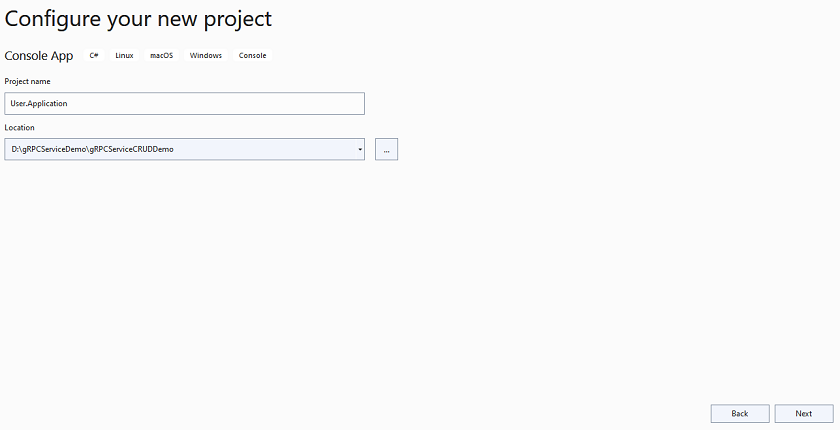
**Step 5**

Create a new User console application



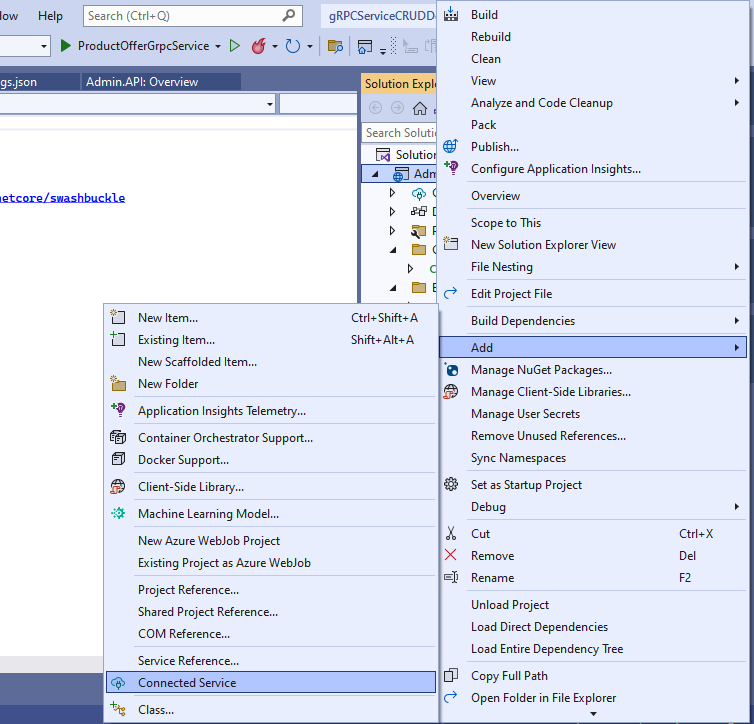
**Step 6**

Configure your project



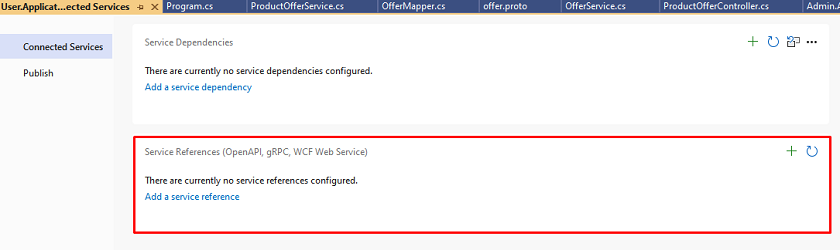
**Step 7**

Add a user offer proto file inside the User Application



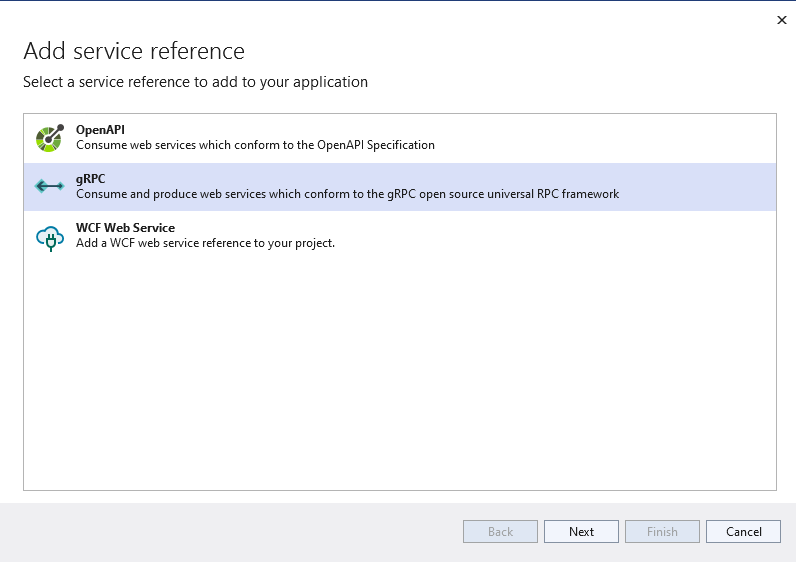
**Step 8**

Add a new reference service



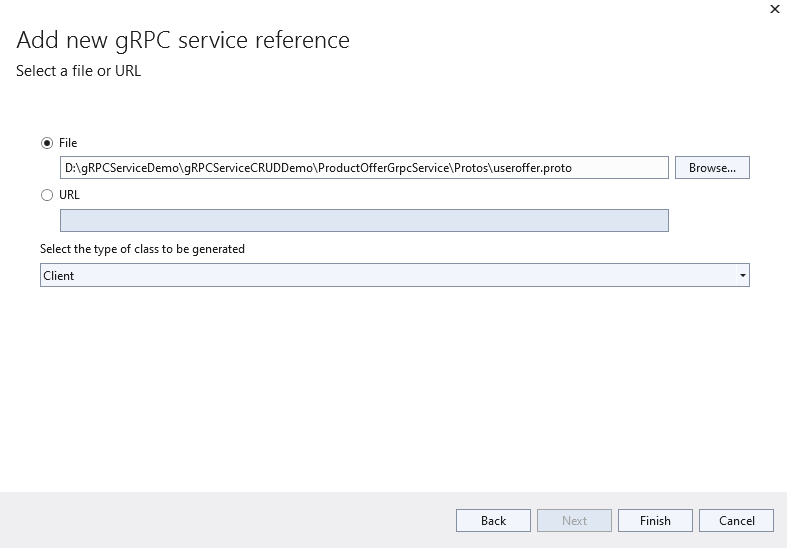
**Step 9**

Click on gRPC



**Step 10**

Select file URL and type of class properly

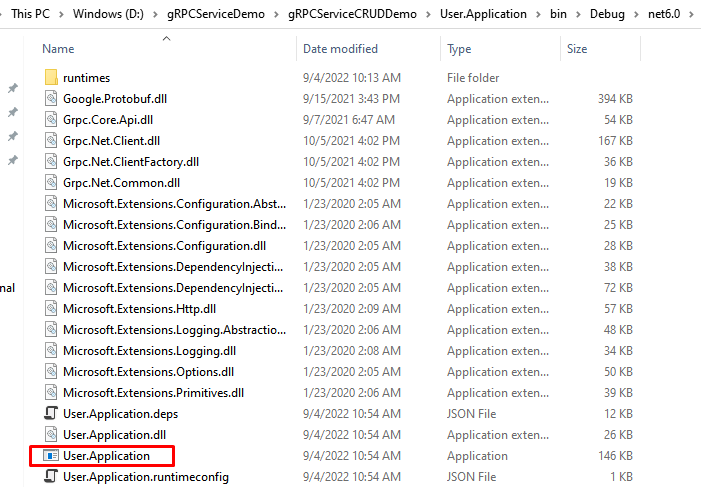


**Step 11**

Add code inside Program class which connects gRPC Service and takes a list of offers which is added by Admin

**Step 12**

Finally, whenever you run your console application will see the latest offer added by the admin



Output

