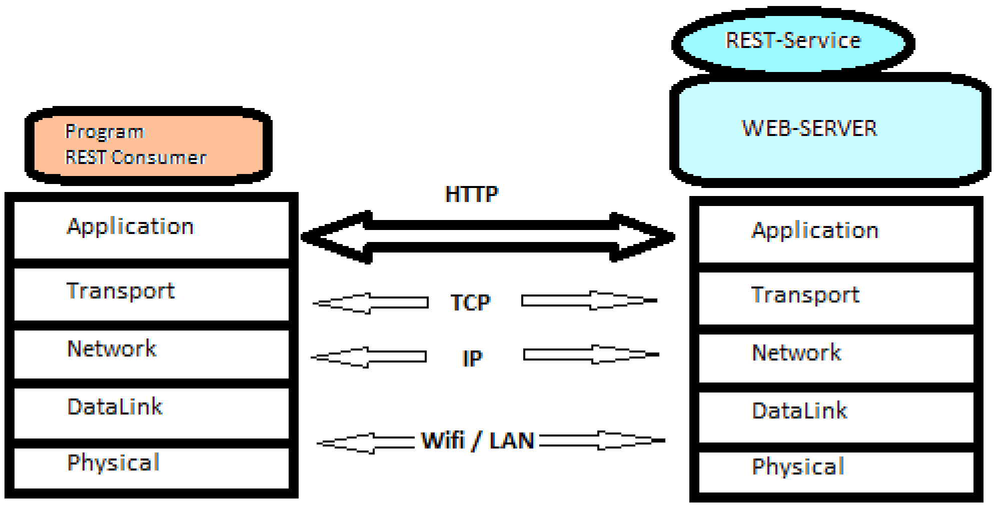
The REST Service Concept and the Underlying Network

When you are to publish persistence i.e. tables in a database, you can use the REST-Service[1][2]. The REST service is provided as a Web-Service, which is hosted on a Web-Server. To understand this setup in more detail you need to have some basic knowledge of how the network is organised, addressed and set up.

The overall picture is like this:



Figur A: Overall Architecture for a REST-Service

As demonstrated in the figure the REST-Service and the REST-Consumer are divided into each site of a network. Therefore, you need a network communication running to use the REST-Service.

The underlying Network

Let see briefly at the network. The Network is now a day described through five layers (in the early days we had seven layers in the ISO-OSI model).

From the top, we have the Application layer which major concerning is to provide specific communication (protocols) for different purposes like mail, file transfer and as in this case the WEB i.e. the **HTTP** protocol (HyperText Transfer Protocol).

Just below the Application, we have first the Transport layer, which have **TCP** for reliable communication i.e. for HTTP and for streaming service the UDP protocol, and second we have the **IP** (the internet) protocol.

At the bottom, we have the low layer network communication like LAN, Wi-Fi, and Bluetooth etc. In this case, we do not need to take much attention to these layers.

Addressing in the Network

In the case of the REST-Service especially two network addresses are important.

[1] For more information see: https://en.wikipedia.org/wiki/Representational_state_transfer
[2] For more information see: https://www.codecademy.com/articles/what-is-rest

The first one is the address of which computer the Web-Server is running. Theses addresses are given in either direct **IP-Address** format e.g. 177.33.45.234 or in a more memorable logical form e.g. “localhost” if the Web-Server (thereby the REST-Service) is located on you own computer or it could be “SomeName.azurewebsites.net” if the web-server is located at Azure.

The second address is the address of your Web-Server on the computer specified by the first address, the IP-Address. These addresses are named **Port-numbers** and the default port-number for your Web-Server (HTTP) is 80. On one computer you can only have one of each port-number i.e. if you already have a Web-Server running on port-number 80, you cannot have another. This is the reason that, if you run your Web-Server locally you will often be given other port-numbers than 80; it could by example be 2372 (it will be a random port-number between 2048 and 65535).

The HTTP protocol

The last element to have knowledge about is the HTTP protocol. This protocol is use to transfer REST-request from the program (The REST-Consumer) to the REST-Service (at the Web-Server). The HTTP protocol has two formats one for the **request** and one for the **response**, next we look shortly at these two format.

The HTTP Request[3] looks like:

Method	‘ ‘	**URI**	‘ ‘	Version	‘new line’
Header	‘: ‘	**Value**	‘new line’		

…

‘new line’
Body

For the REST-Service the important parts are:

- **Method** – which can be **GET, POST, PUT** or **DELETE**
  These method are mapped into the CRUD Principe i.e. **Create** will be **POST** in the HTTP method, **Read** will be **GET**, **Update** will be **PUT** (note both Update and Put has a ‘U’) and **Delete** will be **Delete**
- **URI** – which is a path e.g. ‘API/BOOKS’
  These URI will point out which resource we would like to do (get, post, put, delete) something with – in this example we would like ‘the Book’ (could be a database table Book).
- **HEADER**: **VALUE** – which could be many different information. We mostly use “Content-Type: application/json” to tell we send and retrieve information as JSON[4] formatted.
- **Body** – which hold information presumably JSON-formatted information. For GET and Delete the body would be empty, but for Put or Post the object would be JSON-formatted

[3] For more information see: https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html
[4] For more information see: https://www.w3schools.com/js/js_json_intro.asp

The HTTP Response[5] looks like:

Version	‘ ‘	**Status-code**	‘ ‘	Status-text	‘new line’
Header	‘: ‘	**Value**	‘new line’		

…

‘new line’
Body

Header and Body is the same as for the request. The **Status-code**[6] hold response information about how the request have been maintained i.e. if the status-code is 2xx then it had gone ok, where if you get 4xx as a response something went wrong.

If anything, goes wrong look at either:

1. Have you used the right method?
2. Have you used the right URI?
3. Is your object correctly, JSON formatted?

Some Example of REST-Service request

Let see some examples

Read all books from a REST-Service located at azure (could be anywhere)

```
GET SomeName.azurewebsites.net/API/Books HTTP/1.1

… several header fields

-- a blank line, so in real life there is nothing here

-- no body
```

This will then be sent to the computer at `SomeName.azurewebsites.net` using default port-number 80. It will access the resource books and it would like to read information

Read one book from a REST-Service located locally

```
GET localhost:12555/API/Books/6 HTTP/1.1

… several header fields

-- a blank line, so in real life there is nothing here

-- no body
```

This will be sent to the own computer i.e. localhost using port-number 12555. It will access the resource books reading book No 6.

[5] For more information see: https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html
[6] For more information see: https://www.w3schools.com/tags/ref_httpmessages.asp

The response could look like:

```
HTTP/1.1 200 OK

… several header fields
Content-Type: application/json

-- a blank line, so in real life there is nothing here

{"Id":23,"Author":"J.K.Rowling","Title":"Harry Potter - the Philosopher's
Stone "}
```

Create one book from at the REST-Service located locally

```
POST localhost:12555/API/Books HTTP/1.1

… several header fields

Content-Type: application/json

-- a blank line, so in real life there is nothing here

{"Id":314,"Author":"J.K.Rowling","Title":"Harry Potter - the Chamber of
Secrets "}
```

Like before, the request will be sent to local computer at port 12555 to the resource Books. The method is now POST meaning create and a header-field telling the new book is JSON formatted. The body contain the values for the new book.

How to create a REST-Service

Around the net exists many good tutorials for creating REST-Services. Here some of these are mentioned.

General topics of REST-Services

- API: Using HTTP Methods for RESTful Services
- API: Resource Naming

Creating REST-Service in C# ASP.NET-projects Server site

- http://www.c-sharpcorner.com/UploadFile/0c1bb2/creating-Asp-Net-web-api-rest-service/

Creating REST-Service in C# WCF-projects Server site

- Ghani: CRUD Operations using WCF RESTful Service - part 1

Consuming REST-Service in C# i.e. from a program

- Calling REST-service from a WCF service
- Lochow: HttpClient makes GET and POST very simple