RAWDATA Assignment 3 – Network

This assignment concerns development of a network service using the RDJTP – RawData JSON Transport Protocol. The protocol is defined below. The task is to create a network service that provide the functionality defined by the RDJTP. There are some similarities between the service to implement and a web server providing a web service, the difference is primary the protocol used to define requests and responses.

How and when to hand in

A 1-2 pages document defining the members of the group, the URL to a GitHub repository where the source code can be found, and a table (or a screen dump from your testing environment) showing the status of running all tests in the test suite attached to this assignment. Upload the document to Moodle "Assignment 3" no later than Marts 12 at 23.55.

Important

Hand in one submission from your group (from one of the members), but DO REMEMBER to write ALL NAMES of participants in your group in the top of the file.

RDJTP – RawData JSON Transport Protocol

Here you find the description of the protocol to implement.

Elements marked with a * is mandatory**.**

Request Format:

```
{
        method: <METHOD*>,
        path: <PATH*>,
        date: <DATE*>,
        body: <BODY>
}
```

Note: path is mandatory for all methods except the "echo" method.

Response Format:

```
{
        status: <STATUS*>,
        body: <BODY*>
}
```

<METHOD>
"create", "read", "update", "delete", "echo"

<PATH>
path to the resource in the format /foo/resource, i.e. standard path where elements are separated by slashes

<DATE>*
in Unix time (i.e. number of seconds that have elapsed since 1970-01- 01T00:00:00Z)

<BODY>
The payload of the message. If it contains data it must be in JSON format.

<STATUS>
One of the following status codes followed by their reason phrase:

Status Code	Reason phrase
1	Ok
2	Created
3	Updated
4	Bad Request
5	Not found
6	Error

the level of description is not specified, except for bad requests with missing or erroneous elements which should be reported in the form

`<STATUS CODE> reason`

where reason has the form

missing <element>

or

illegal <element>

where `<element>` is the name of the protocol element, e.g. method, date, path)

if more than one problem is found return a list of reasons, e.g.

`"4 missing date, missing body, illegal method"`

the order is not important.

Question a)

Implement a web service by use of the `TcpListener`[1]`/TcpClient`[2] classes from the .NET framework. The service must use the client/server model[3] and the request-response design pattern[4] Your service must listen on port 5000[5] and the implementation should be multithreaded, i.e. every connection to your service is handled by a new thread. The common pattern would be that clients connect to your service and send a request and that your service handles that request and send back a response. Nevertheless, you must be prepared for connections that do not send a request, and just ignore such connections.

Your service must implement the RDJTP and be able to verify all the constraints given in the description, regarding requirements on structure and content. The structural constrains concerns correct use of the fields in requests and responses, e.g.

```
{
    method: “update”,
    path: “/api/products/1”,
    date: 1507318869
}
```

which is an example on a violation of the constraint that specify that the `update` method requires a `body`, which is not present in the request. The content constraints are about the format of the values, e.g.

```
{
    method: “update”,
    path: “/api/categories/1”,
    date: “06-10-2017 19:41:09”,
    body: “{cid: 1, name: "NewName"}”
}
```

where the structure is correct, but the content of the date is not in the Unix time format.

The test suite will tell you if you missed some constraints :-)

Question b)

Provide an API to the following data model:

```
category(cid, name)
```

[1] https://msdn.microsoft.com/en-us/library/system.net.sockets.tcplistener(v=vs.110).aspx
[2] https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.110).aspx
[3] https://en.wikipedia.org/wiki/Client%E2%80%93server_model
[4] http://www.servicedesignpatterns.com/clientserviceinteractions/requestresponse
[5] Very import, since the test suite expect a service on this port

and the following data:

category	
cid	**name**
1	Beverages
2	Condiments
3	Confections

You do not need to implement any real database stuff. Just keep the data in memory, but changes to the data on create, update and delete must be reflected. (but not persisted, i.e. losing any changes on restart)

Note: The test suite expects this data to be available on your RDJTP service.

The API provide the following path: **`/api/categories`**

Likewise, the protocol requests and responses, all data transported between client and server must be in JSON, except for the payload on the `echo` method. Sending, for instance, the category with id 1 over the protocol you must transform it into JSON

```
{cid: 1, name: "Beverages"}
```

Here are some examples on how to use the service:

Method	Path	Example input	Status code	Example output
read	/categories/1		1 Ok	{cid: 1, name: “Beverages”}
read	/categories		1 Ok	[{cid: 1, name: “Beverages”}, {cid: 2, name: “Condiments”} {cid: 3, name: “Confections”}]
update	/categories/3	{ cid: 3, name: “Test”}	3 Updated	
update	/categories	{ cid: 3, name: “Test”}	4 Bad Request	
create	/categories	{ name: “Seafood”}	2 Created	{cid: 4, name: “Seafood”}
delete	/categories/3		1 Ok	
delete	/ categories/123		5 Not Found	
read	/ categories/123		5 Not Found	

read
The list of all elements can be retrieved by use of the path, individual elements can be retrieved by adding the “/<id>” to the path (the id of the element to be retrieved without the <>). The latter must return status “5 Not found” if the requested element is not in the database, otherwise the status is “1 Ok”.

create

New elements can be added by use of the path and the new element in the body of the request. Using path and an id is invalid and should return “4 Bad request”. On successful creation return the “2 Created” status plus the newly create element in the body.

update
All elements can be updated by use of the path extended with the id and the updated element in the body. Updates without an id in the path is not allowed and should return “4 Bad request”. On successful update return the “3 Updated” status.

delete
All elements can be deleted by use of the path extended with the id. If the element is not in the database “5 Not found” should be returned otherwise “1 Ok”.

echo
This method does not take any path, it is just ignored if provided, and will simply return the body of the request as the body of the response with status “1 Ok”.

Appendix

Converting to and from JSON

JSON conversion can be done with the Newtonsoft.Json package (get it from NuGet), after adding a using statement to your file

```
using Newtonsoft.Json;
```

conversion can be done like this:

```
var category = new Category();
// from objects to JSON
var categoryAsJson = JsonConvert.SerializeObject(category);
// from JSON text to object
var categoryFromJSON = JsonConvert.DeserializeObject<Category>(categoryAsJson);
```