REST WEBSERVICE APPLICATION

English Manual

20 DE AGOSTO DE 2023

JUAN CARLOS SEPULVEDA

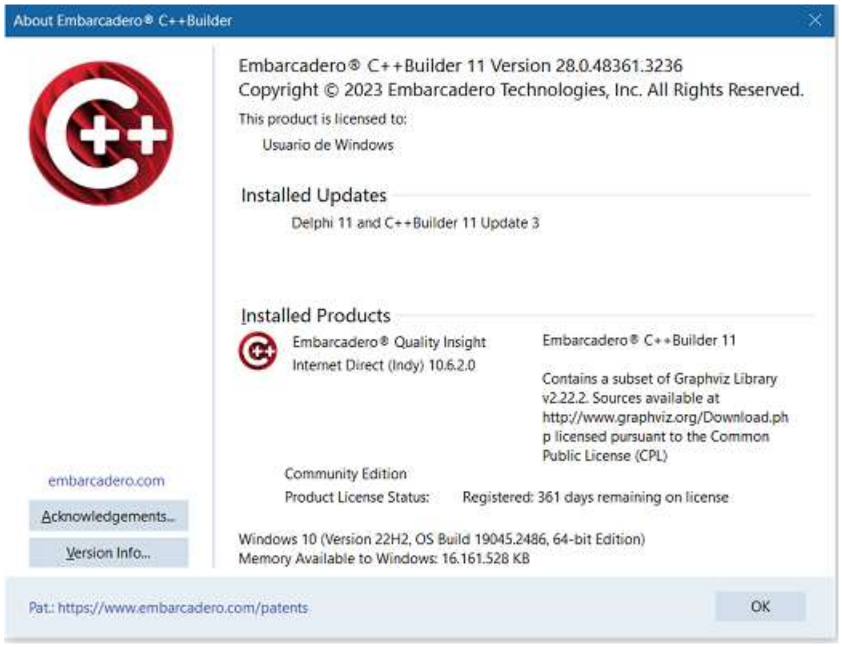
Contenido

Application description.

Used tools.

The “Rest WebService Application” application was made using the tools:

Embarcadero® C++ Buider 11.



Embarcadero® C++Builder 11 Version 28.0.48361.3236
Copyright © 2023 Embarcadero Technologies, Inc. All Rights Reserved.
This product is licensed to:
Usuario de Windows

Installed Updates
Delphi 11 and C++Builder 11 Update 3

Installed Products
Embarcadero® Quality Insight
Internet Direct (Indy) 10.6.2.0

Embarcadero® C++Builder 11

Contains a subset of Graphviz Library v2.22.2. Sources available at http://www.graphviz.org/Download.php licensed pursuant to the Common Public License (CPL)

Community Edition
Product License Status: Registered: 361 days remaining on license

Windows 10 (Version 22H2, OS Build 19045.2486, 64-bit Edition)
Memory Available to Windows: 16.161.528 KB

Pat.: https://www.embarcadero.com/patents

The "Community Edition" version license was used, whose license is free for 1 year, even for commercial applications, although it has some limitations such as the absence of some components that are present in others version, among other limitations. Among the missing components are the “DataSnap” and “REST Server” palette components.

As these components were not present, the **TIdHTTPServer** component was used to implement the REST server

REST Debugger 10.5

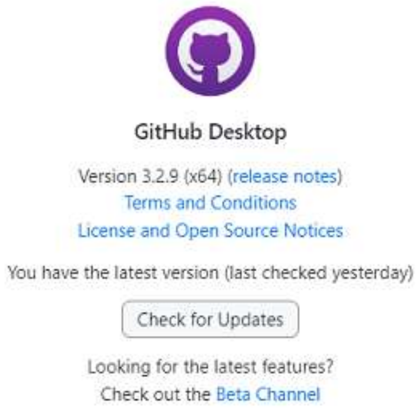
REST Debugger 10.5
Embarcadero Technologies

To debug the application, the "REST Debugger 10.5" tool was used, simulating the JSon requests.

SQLite Maestro.

For the design and work with the Database.

GitHub Desktop


GitHub Desktop

Version 3.2.9 (x64) (release notes)
Terms and Conditions
License and Open Source Notices

You have the latest version (last checked yesterday)

Looking for the latest features?
Check out the Beta Channel

To upload the source codes to the GitHub repository.

Used Components

For the development of the application the following components were used.

Indy Servers Palette

TIdHTTPServer : It was used to receive HTTP requests.

This component was used since the components of the "REST Server" palette were not present. The **TIdHTTPServer** component was used to implement the REST server. Knowledge of TCP and HTTP is required to use this component, as well as a greater amount of code than if the "REST Server" component is used.

The following components events were handled:

OnCommandGet: To handle GET and POST type requests.

OnCommandOther: To handle PUT and DELETE type requests.

FireDAC Palette

TFDConnection: It was used to make the connection with the SQLite database.

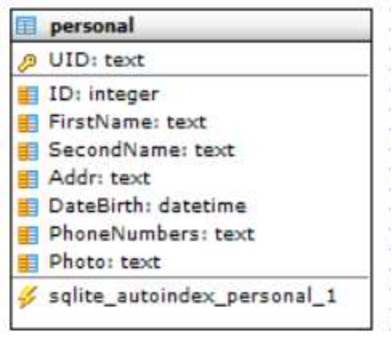
TFDQuery: It was used to perform the SQL database queries.

DataBase.

Se utilizo una BBDD de tipo SQLite, conteniendo una sola Tabla denominada

An SQLite type database was used, containing a single Table called “Personal” with the following main fields:

Fields of personal table:



```
personal
UID: text
ID: integer
FirstName: text
SecondName: text
Addr: text
DateBirth: datetime
PhoneNumbers: text
Photo: text
sqlite_autoindex_personal_1
```

The “UID” field was used as the key. This field stores an Universal Identifier also known as UUID, very useful when you want to use a universally unique identifier. A numerical “ID” field was also used to identify the contact for educational purposes and to show the use of the application at work. This field (“ID”) could have been used as a key by using an autoincrement type, but with the following disadvantages:

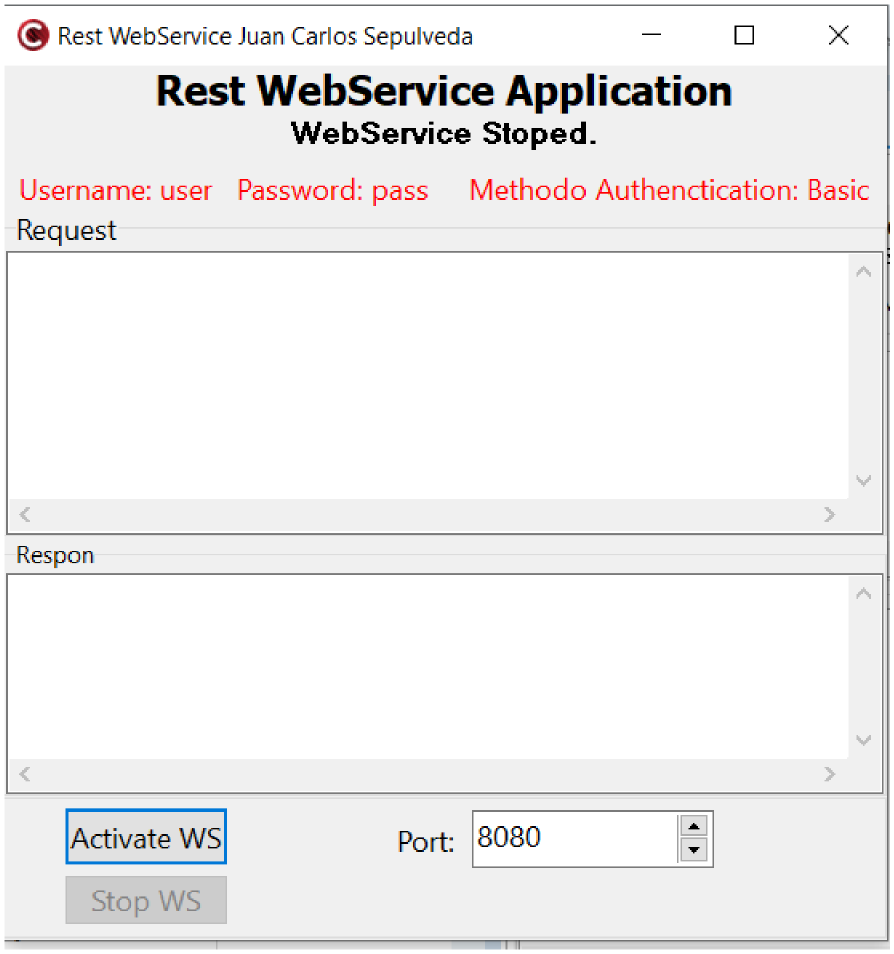
1. Limitation in the selection of values: Autoincrement type fields only allow consecutive numeric values, which limits the selection of custom values.
2. Value cannot be modified: Once a value has been assigned to an auto-incrementing field, it cannot be manually modified. This can be a problem if a specific value needs to be changed in the future.
3. Synchronization problems: If you work with several databases and use an auto-incrementing field, there may be synchronization problems between the different databases.
4. Scalability issues: If the database grows too large, there may be problems with the auto-incrementing fields, as they may reach their maximum limit and not be able to generate new values.
5. Security issues: If an auto-incrementing field is used as the primary key, there may be security issues, as users can easily guess the values in the records.

Of course none of these drawbacks are important in this application but to give you an idea of how a real application should be implemented.

Note: In the case of images, the link is saved and it is assumed that they are physically stored in the cloud or on an external server.

Application description.

The application is a "Stand Alone" type application, it does not need to be executed within any Web Server, such as IIS, apache or other. When you lift it, the following interface is displayed:


Rest WebService Juan Carlos Sepulveda
Rest WebService Application
WebService Stoped.
Username: user
Password: pass
Methodo Authenctication: Basic
Request
Respon
Activate WS
Stop WS
Port:
8080

The application because is test application, has the values of UserName and Password fixed in the code. It currently uses the following hardcoded values.

Username: user Password: pass

In a real application, these values should be loaded from the database to be able to use several users and even give a personalized response based on the connected user.

Before "Activate" the server can be selected which will be the listening port. By default it uses 8080, but this can be changed while the application is not active.

The "Memo" labeled as "Request" and "Response" are used to show the logs of what was sent, received and what happened in the Request and Response messages. In a real application this Logs information should be able to be saved and also have a time stamp for each even to be saved.

Using the App

Install the app from the installer. It is recommended not to install it in the “Program File” directories because the App includes a database and the database is installed in the same installation directory and Windows security policies do not allow access without Administration permissions to the files in those folders.

Copy the installer to the same folder where you want to install it so you don't need to change the directory during install process.

To test the application some client must be used.

Clients used during development were mainly the “Web Firefox 116.0.3 (64-bit)” browser and the “REST Debugger 10.5” tool.

nstale la aplicación desde el instalador. Se recomienda no instalarla en los directorios Program File pues incluye una BBDD que se instala en el mismo directorio de instalación y las directivas de seguridad de Windows no permiten acceder sin permisos de Administacion a los archivos en esas carpetas.

Copie el instalador en la misma carpeta donde desea instalarlo para que no necesite cambiarlo de directorio

Para probar la aplicación debe usarse algún cliente.

Los clientes que se usaron durante el desarrollo fueron principalmente el navegador “Web Firefox 116.0.3 (64-bit)” y el “REST Debugger 10.5”

Testing requests of the GET Type.

The Port 8095 was selected to carry out the tests in this manual. It could be another.

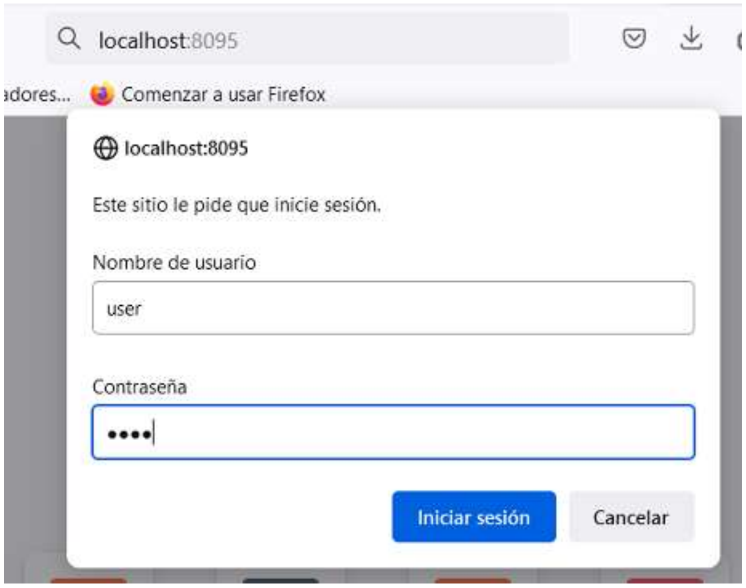
Using Firefox

The first time.

When entering the URL for the first time, a password will be requested

Enter: http://localhost:8095/

With which it will ask for username and password:



Este sitio le pide que inicie sesión.

Nombre de usuario: user

Contraseña: ••••

Requesting all the data

To request all the data, simply enter the URL: http://localhost:8095/

Once the values have been entered, all the content of the database will be displayed in the browser, something similar to the following:

[{"UID":"97F9D2BF-23BB-4A37-B851-9D83AC675FAA","ID":1,"FirstName":"Juan","SecondName":"Carlos","Addr":"Marianao, Habana","DateBirth":"1976-08-01 00:00:00","PhoneNumbers":"55525659","Photo":"","Age":20,"UID":"4AC7B2C5-6AF7-49B0-B090-24F55A1C86F7","ID":2,"FirstName":"Lazaro","SecondName":"Sep","Addr":"La Lisa, Habana","DateBirth":"2020-10-16 00:00:00","PhoneNumbers":"5555555","Photo":"","Age":3,"UID":"BD2B7FD0-55D8-4987-B77E-145208F2D122","ID":3,"FirstName":"Lucia","SecondName":"Sep","Addr":"Camaguey, Camaguey","DateBirth":"2020-10-16 00:00:00","PhoneNumbers":"4444444","Photo":"","Age":3,"UID":"{220CDA9E-E91E-492B-B15E-0BA17B00BB64}","ID":222,"FirstName":"Mama","SecondName":"Moya","Addr":"UCI,116 104","DateBirth":"","PhoneNumbers":"5678","Photo":"www.pictures.com\/yuri\/carnet.jpg","Age":80,"UID":"{73DFF37B-100E-4261-8F32-5F49994D8D84}","ID":221,"FirstName":"Yurisleidy","SecondName":"Moya","Addr":"UCI,116 104","DateBirth":"","PhoneNumbers":"5678","Photo":"www.pictures.com\/yuri\/carnet.jpg","Age":31}]

In this case 5 values were returned in JSon format.

Requesting specific contacts by a field

If you only wanted to return the values of a specific field. For example: ID=1. Type one of the 2 following URLs. They are equivalent.

http://localhost:8095/1

http://localhost:8095/?ID=1

Returned result:

```
[{"UID":"97F9D2BF-23BB-4A37-B851-9D83AC675FAA","ID":1,"FirstName":"Juan","SecondName":"Carlos","Addr":"Marianao,
Habana","DateBirth":"1976-08-01 00:00:00","PhoneNumbers":"55525659","Photo":"","Age":20}]
```

You can also make requests for other fields:

http://localhost:8095/?Age=3

Returned result:

```
[{"UID":"4AC7B2C5-6AF7-49B0-B090-24F55A1C86F7","ID":2,"FirstName":"Lazaro","SecondName":"Sep","Addr":"La Lisa,
Habana","DateBirth":"2020-10-16 00:00:00","PhoneNumbers":"5555555","Photo":"","Age":3,"UID":"BD2B7FD0-55D8-4987-B77E-
145208F2D122","ID":3,"FirstName":"Lucia","SecondName":"Sep","Addr":"Camaguey, Camaguey","DateBirth":"2020-10-16
00:00:00","PhoneNumbers":"4444444","Photo":"","Age":3}]
```

http://localhost:8095/?Age>40

Returned result:

```
[{"UID":"{220CDA9E-E91E-492B-B15E-0BA17B00BB64}","ID":222,"FirstName":"Mama","SecondName":"Moya","Addr":"UCI,116
104","DateBirth":"","PhoneNumbers":"5678","Photo":"www.pictures.com\/yuri\/carnet.jpg","Age":80}]
```

You can also make requests for various conditions

Examples of condition requests for 2 fields

http://localhost:8095/?ID=1&Age=3

Returned result:

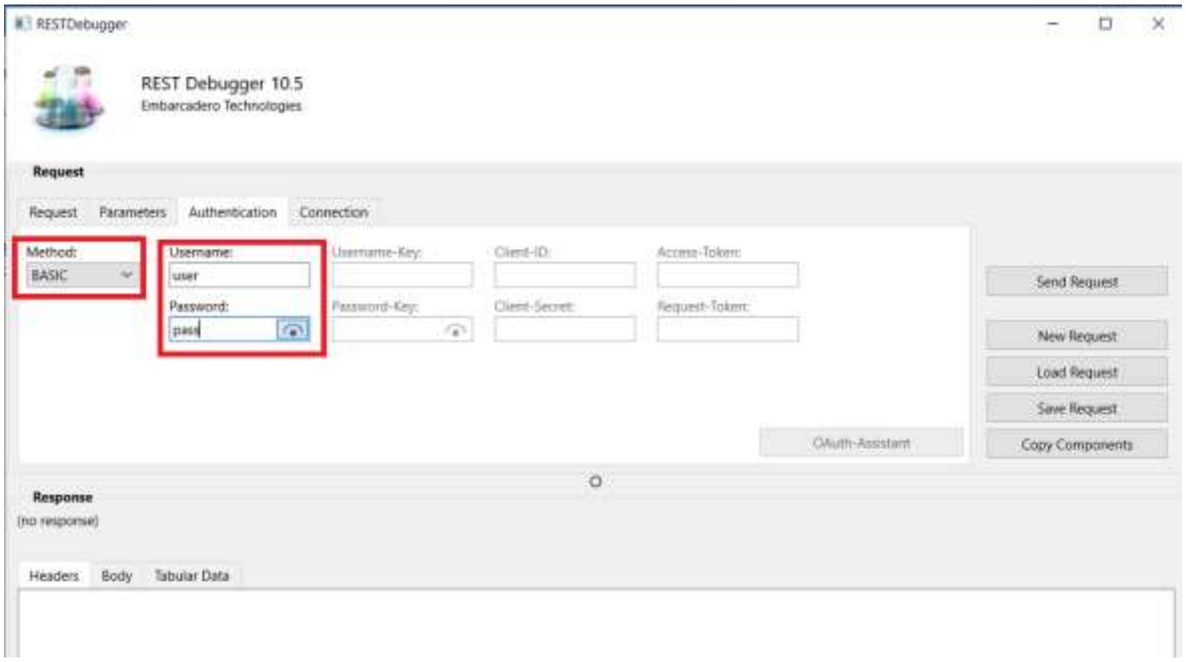
```
[{}]
```

http://localhost:8095/?ID=2&Age=3

```
[{"UID":"4AC7B2C5-6AF7-49B0-B090-24F55A1C86F7","ID":2,"FirstName":"Lazaro","SecondName":"Sep","Addr":"La Lisa,
Habana","DateBirth":"2020-10-16 00:00:00","PhoneNumbers":"5555555","Photo":"","Age":3}]
```

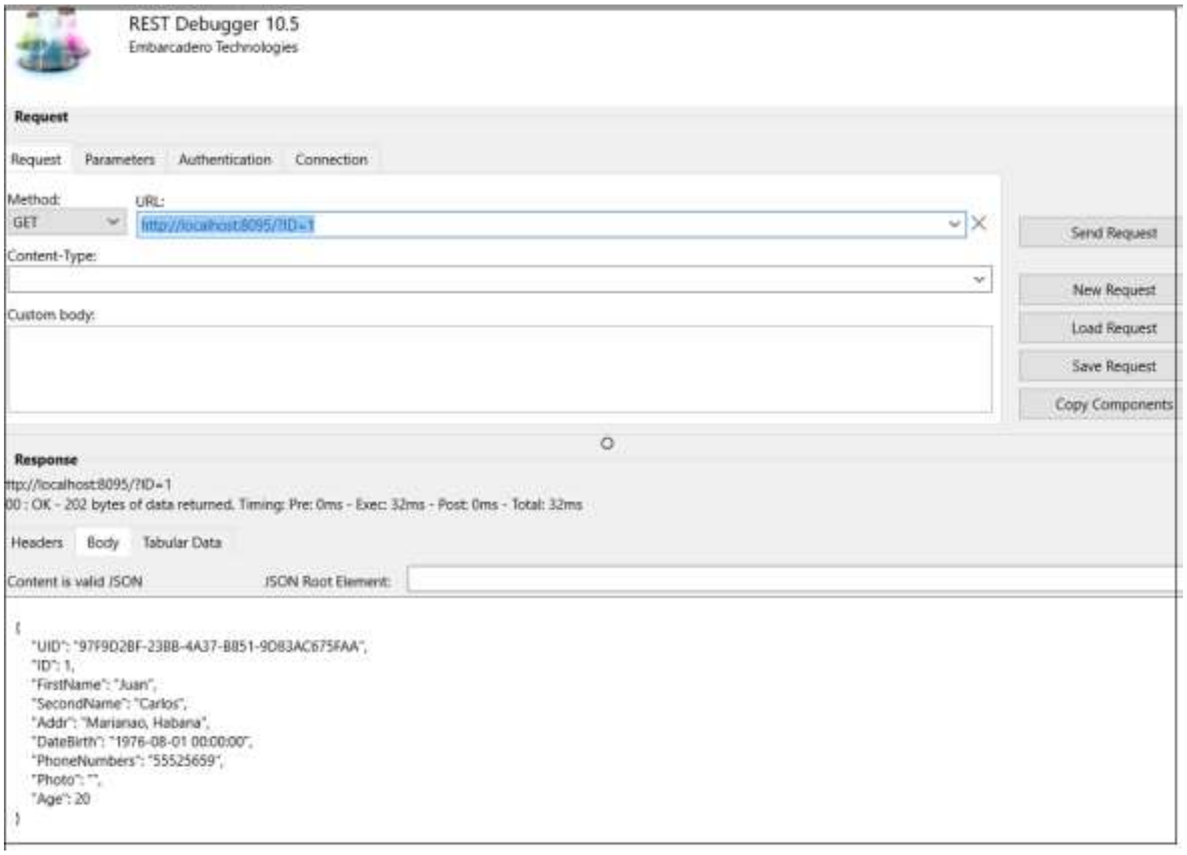
Using REST Debugger 10.5

The first step in using REST Debugger to test your application is to set up authentication for the site. In this case, type "Basic" and with the login and password already mentioned before.



Method: BASIC
Username: user
Password: pass
(no response)

To make requests, just indicate the desired URL as already explained for firefox.



Method: GET

URL: http://localhost:8095/?ID=1

Content-Type:

Custom body:

```
ttp://localhost:8095/?ID=1
00 : OK - 202 bytes of data returned. Timing: Pre: 0ms - Exec: 32ms - Post: 0ms - Total: 32ms
```

Content is valid JSON   JSON Root Element:

```
{
   "UID": "97F9D2BF-23BB-4A37-B851-9D83AC675FAA",
   "ID": 1,
   "FirstName": "Juan",
   "SecondName": "Carlos",
   "Addr": "Marianao, Habana",
   "DateBirth": "1976-08-01 00:00:00",
   "PhoneNumbers": "55525659",
   "Photo": "",
   "Age": 20
}
```

Using REST Debugger 10.5

To insert a contact you must create a JSON text to send. For example use this:

{"UID":"4AC7B2C5-6AF7-49B0-B090-24F55A1C86F7","ID":222,"FirstName":"Lazaro","SecondName":"Sep","Addr":"La Lisa, Havana", "DateBirth":"2015-10-16 00:00:00","PhoneNumbers":"5555555","Photo":"","Age":3}

Select POST method, and Content-Type = application/json

Testing requests of Type POST.

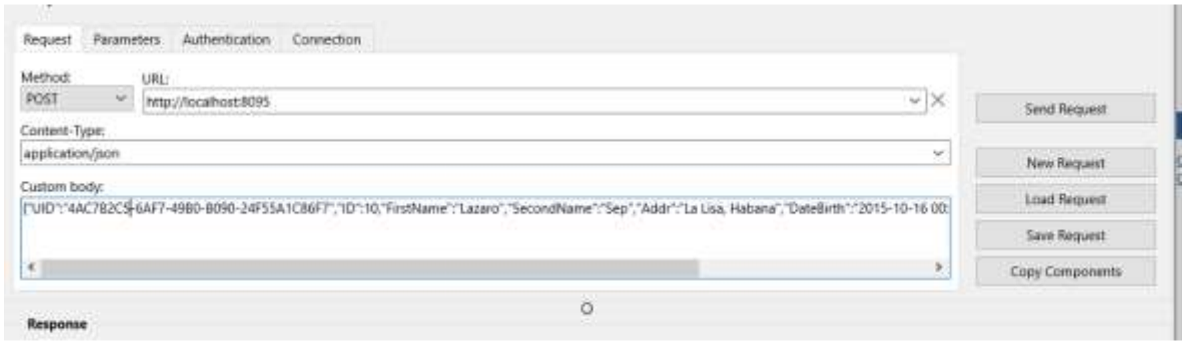
The Port 8095 was selected to carry out the tests in this manual. It could be another.

Using REST Debugger 10.5

To insert a contact you must create a JSON text to send. For example use this:

> {"UID":"4AC7B2C5-6AF7-49B0-B090-24F55A1C86F7","ID":222,"FirstName":"Lazaro","SecondName":"Sep","Addr":"La Lisa, Habana","DateBirth":"2015-10-16 00:00:00","PhoneNumbers":"5555555","Photo":"","Age":3}

Select **POST** method, and Content-Type = **application/json**



Method: POST

URL: http://localhost:8095

Content-Type: application/json

Custom body:

```
{"UID":"4AC782C5-6AF7-49B0-B090-24F55A1C86F7","ID":10,"FirstName":"Lazaro","SecondName":"Sep","Addr":"La Lisa, Habana","DateBirth":"2015-10-16 00:
```

If you then check the current data you can see that a new contact value was inserted. The UID value that was specified in the JSon command is discarded as a UUID value is automatically generated internally to ensure it is unique.

Testing requests of the PUT Type.

Using REST Debugger 10.5

Updates are made to contacts that have an ID equal to the one specified as part of the URL.

http://localhost:8095/15

If there are repeated contacts with the same ID, all of them will be updated with the data sent.

To update the previously inserted value, send for example the following JSon text with the values to update.

{"FirstName":"Haydee","SecondName":"Pena","Addr":"Marianao, Habana","DateBirth":"1976-08-01 00:00:00","PhoneNumbers":"55525659","Photo":"","Age":20}

In the JSon string it is not necessary to specify neither the "UID" nor the "ID" since neither of them will be modified.

Testing requests of Type DELETE.

Using REST Debugger 10.5

To delete a contact, you only have to indicate the value of the "ID" field of those you wish to delete.

For example: http://localhost:8095/15

It must be specified that the Method is DELETE