Documentation Microservice: Set

Goal: Allow to us to insert some element in our database through a HTML Page.

First, we start with the main function:

Main.go file:



First of all, in my code, i call a function name CreateDir. I will explain later this function. you can also see that a call a function name http.HandleFunc with 2 parameters inside. this function will create our web server and also call my function name GetData.

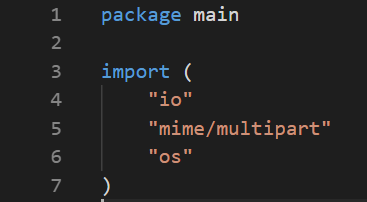
‘/’ define the URL path that we have to use.

After that, i just print that we are connected and i make a verification on a function.

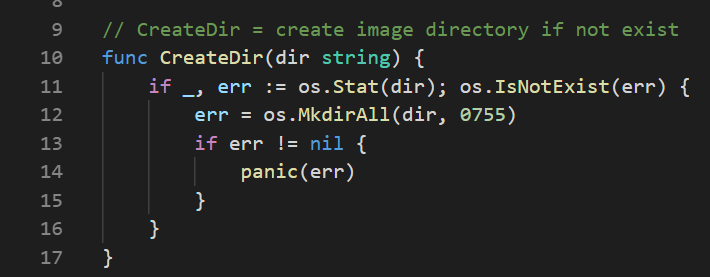
I call a function name http.ListenAndServe which allow to use to define a port. here, I choose the port 8080. If this port is taken by another app or everything i will have an error and I display it.

Let's continue by the CreateDir function in create.go file:

Import:



CreateDir function:



Here, I call CreateDir function with 1 parameter. this function will allow to us to create a directory where our image that we send to our database will be store.

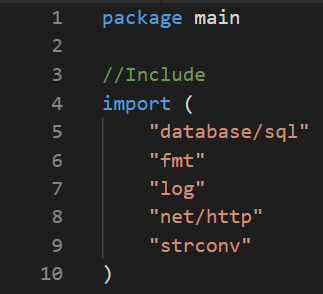
As you can see, this function takes 1 parameter, a string. this string contains "image". It will be our directory name.

First of all in this function, we check if the directory that we want to create exist or not.

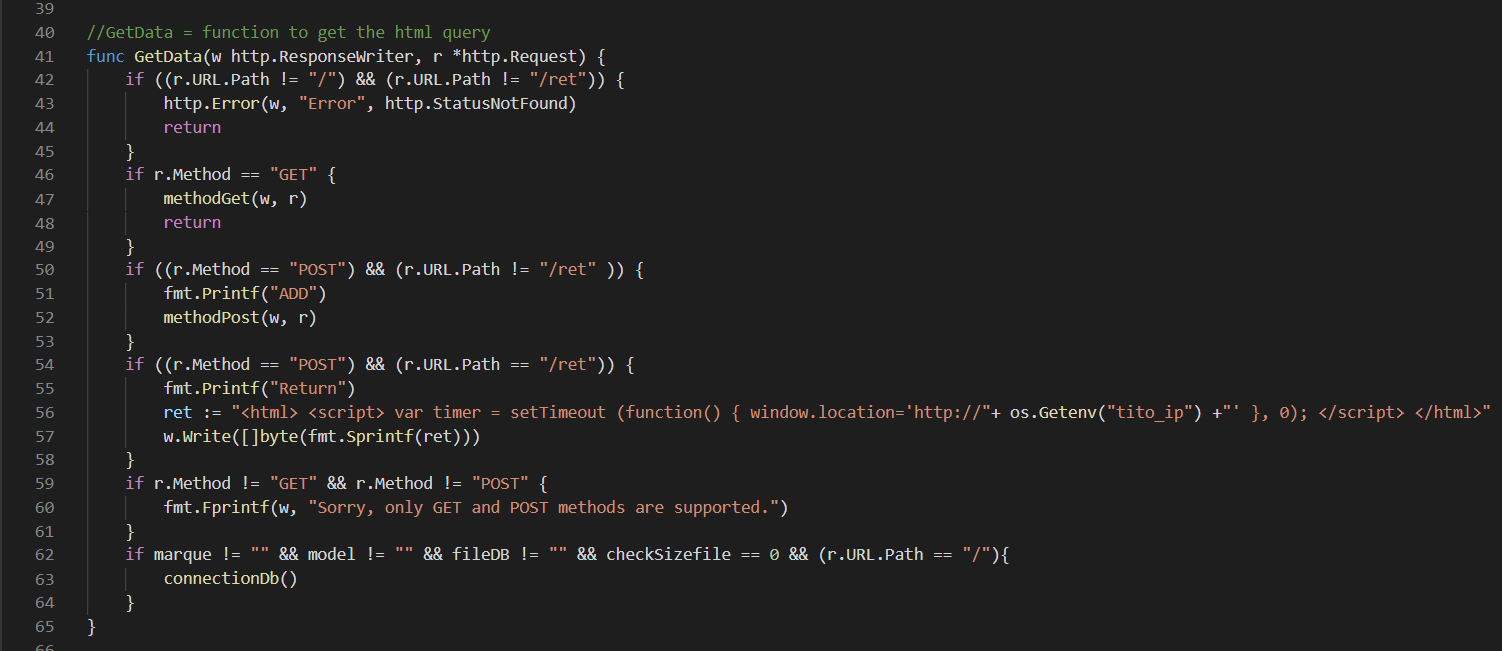
If the directory doesn't exit, it will create one, on the contrary, it will end the function. That's all.

Go back to our main and i will explain the GetData function, the more important one in the getData.go file.

Import:



GetData function:



As you can see, our function take 2 parameters.

-A http.ResponseWriter value assembles the HTTP server's response; by writing to it, we send data to the HTTP client.

-A http. Request is a data structure that represents the client HTTP request. r.URL.Path is the path component of the request URL.

The HTTP queries/requests are Get and Post.

So first of all, we have a verification on our URL.

Secondly, we have a verification on our first HTTP query. If the query is GET, we call another function name methodGet.

Thirdly, we have another verification on our second HTTP query. If the query is POST and our URL is not “/ret”, we call a function name methodPost.

Fourthly, we have another verification, if our query is POST and our URL is “/ret”, we redirect

our URL to the Tito main page.

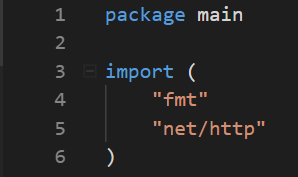
Fifthly, there is a verification of our method. if the HTTP query is not GET or POST, that means that we have an error.

And finally, we have a fifthly verification that allow to us to launch the connection to our database MySQL.

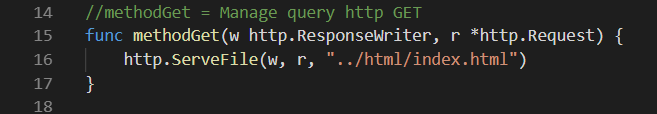
This function will allow to us to verify if there are some errors. And also, if everything is good, launch our connection to database.

Go to our method file which contain methodGet & methodPost functions.

Import:



methodGet function:



We have also 2 parameters like the getData function.

Here, we just specify that we want to display in our web server, our html page call index.html.

methodPost:



Line 37: img2html := "<html> <style> .form-img { background-image: url(data:image/jpeg;base64," + imgBackground + "); height: 195px; background-position: center; background-repeat: no-repeat;} </style> <body class=\"form-img\"> <meta http-equiv=\"refresh\" content=\"0\";URL=\"/\"/> </body> </html>"

In this function we have also 2 parameters, the same as the getData function.

Here, we want to recover our field that we fill in our html page.

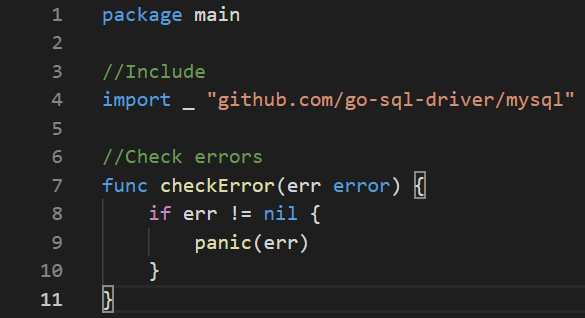
You can also see that we use our global variables. So we recover our field model, marque, gazLevel, location variable.

We also recover our file (that is an image don't worry) but with in a different way. The fact that it is an image it will be different.

After that, we check if our variables are not empty. If one of them is empty, we display a html page with an error written.

If there any errors, we decided to call a function name checkError.

checkError.go & checkError function:

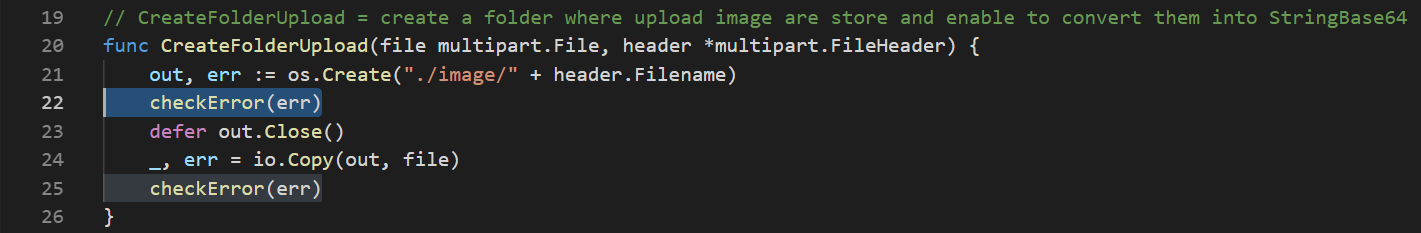


Allow to us to check if there is no error.

Go back to our methodPost, we call another function name CreateFolderUpload with 2 parameters.

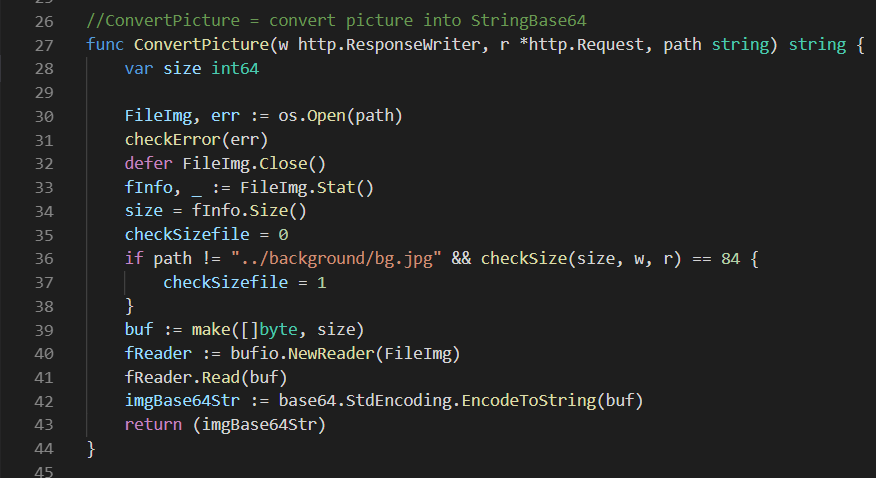
The CreateFolderUpload function is in the same file like CreateDir function above:

CreateFolderUpload function:



This function stores our image that we choose in our directory that we create before. That's all to understand.

For the end of this function (methodPost), we call a last function name Convertpicture which stores in picture.go file.



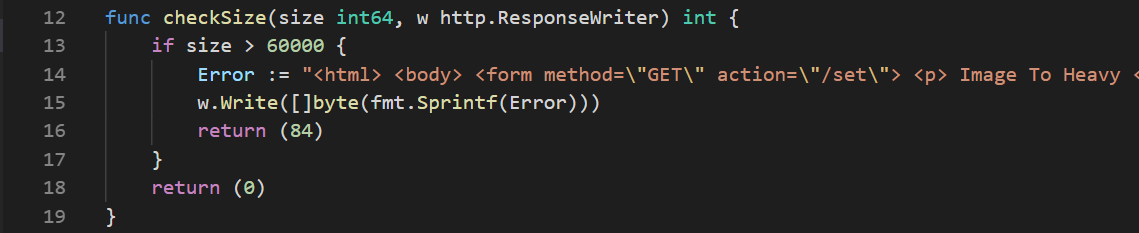
This function allows to us to convert our image that we want to send to our database into string because MySQL database convert the image in BLOB and it is more useful to convert them into string and send them into database.

First, we decided to open our image that we want.

To convert the image into Stringbase64, we have to create a buffer to stock our string of our image. To do that, we need the size of the image to rent the exact space in our memory. So, we call FileImg.Stat sotck in fInfo and we call also fInfo.Size() to know exactly the size.

After that, we call a function name checkSize, just to verify if the size of the image is not to heavy for the sending. And we return the StringBase64.

checkSize function (file: picture.go):



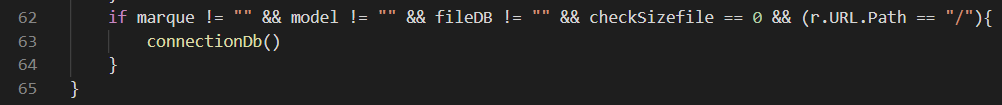
If the size is upper than 60 000, we display a HTML page with an error written.

Back to our ConvertPicture function.

So, we create our buffer with our optimal size that we check just above and we launch a method to convert our image.

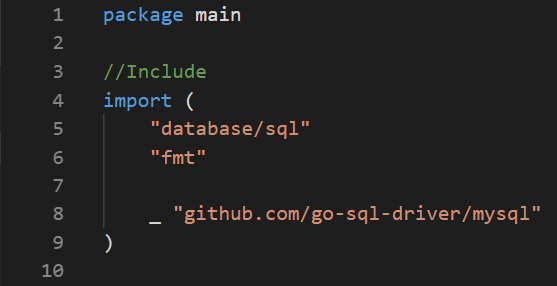
After that, we display this image in a html page to have a visual on it. And we stock our string in our global variable.

Once our methodPost function is done, we go back to our getData function. And i will check if every global variable is not empty to launch my connection to my MySQL database:

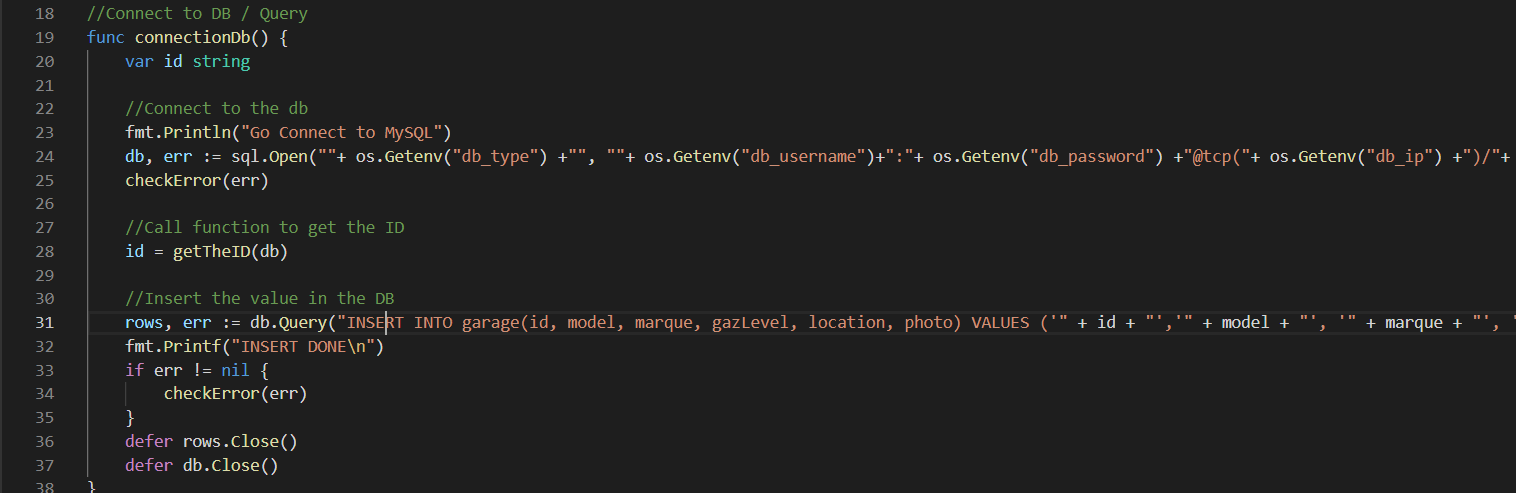


connect.go file.

Import:



connectionDb function:



Line24 : db, err := sql.Open(""+ os.Getenv("db\_type") +"", ""+ os.Getenv("db\_username")+":"+ os.Getenv("db\_password") +"@tcp("+ os.Getenv("db\_ip") +")/"+ os.Getenv("db\_name") +"")

Line 31 : rows, err := db.Query("INSERT INTO garage(id, model, marque, gazLevel, location, photo) VALUES ('" + id + "','" + model + "', '" + marque + "', '" + gazLevel + "', '" + location + "', '" + fileDB + "');")

First of all, we open our sql table with the command :

sql.Open("mysql", "username:password@protocol(IP)/YourDatabase")

We use our env variable that we set in our Dockerfile to connect to our database.

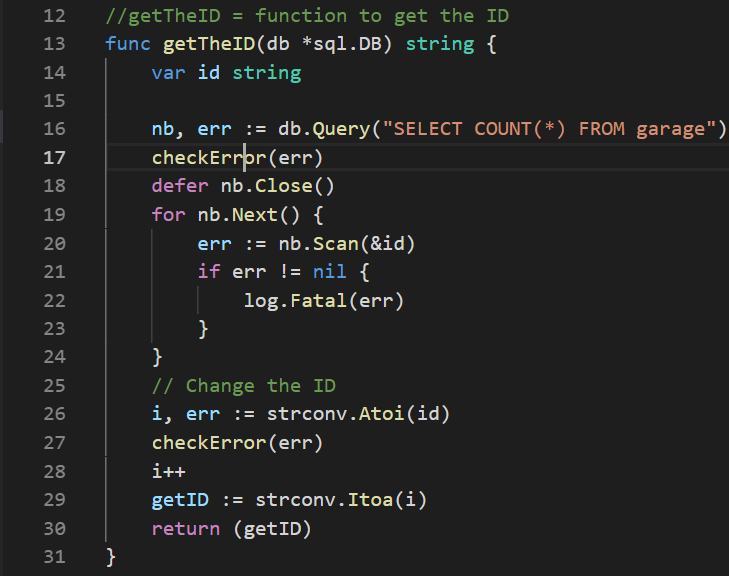
The first parameter is the type of your db. It can be mariaDB or everything you have choosen.

The second parameter is your log to reach to the database.

After that we call a function to get the id and increment it. I will show to you later.

After that, we launch a query to insert our data in our database. and we check if there are no errors.

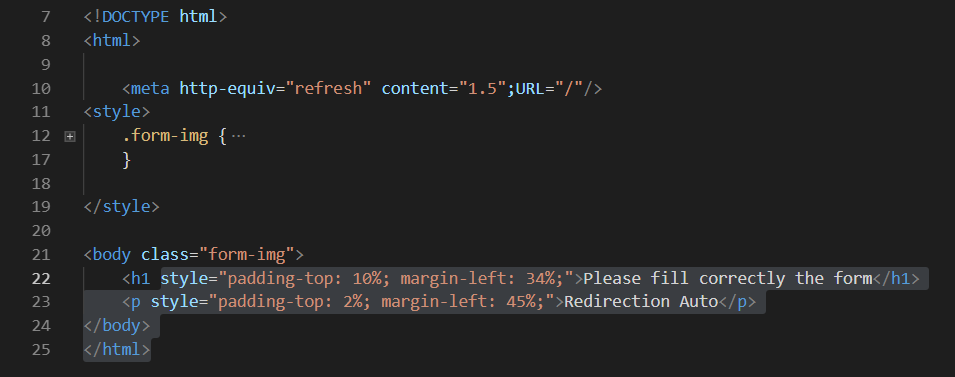
getTheId function (file -> getData.go):



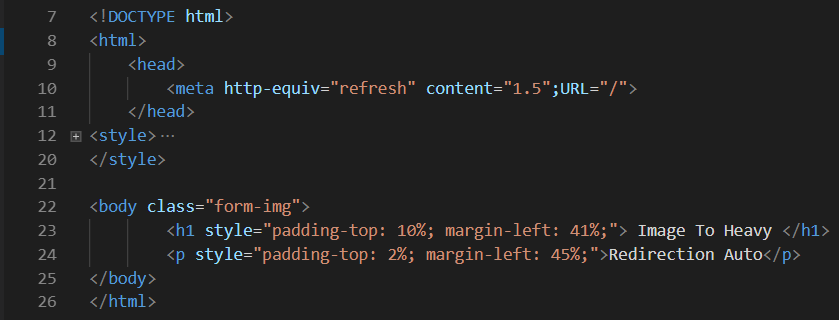
In this function, we recover the id maximum of our database with a sql query. We convert this result into a number because we recover it like a character. And we increment it by 1.

HTML file:

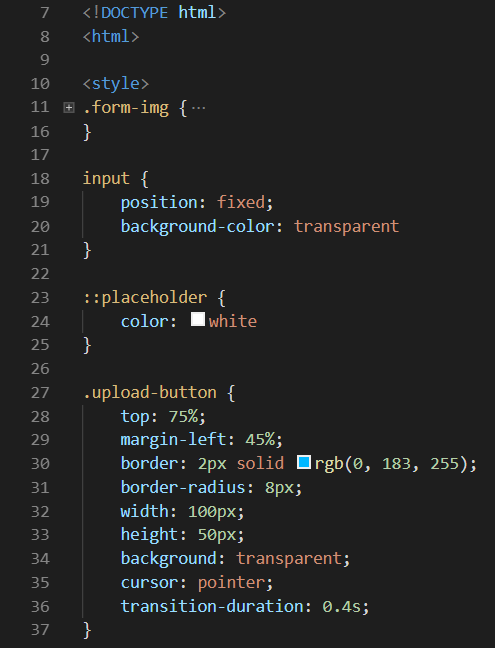
Err.html:

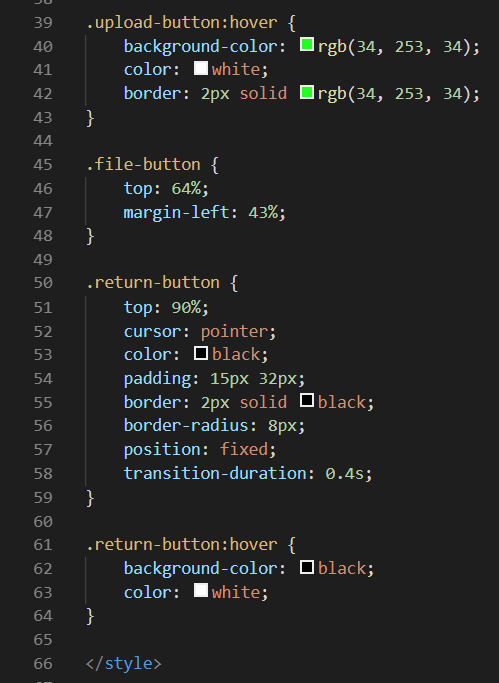


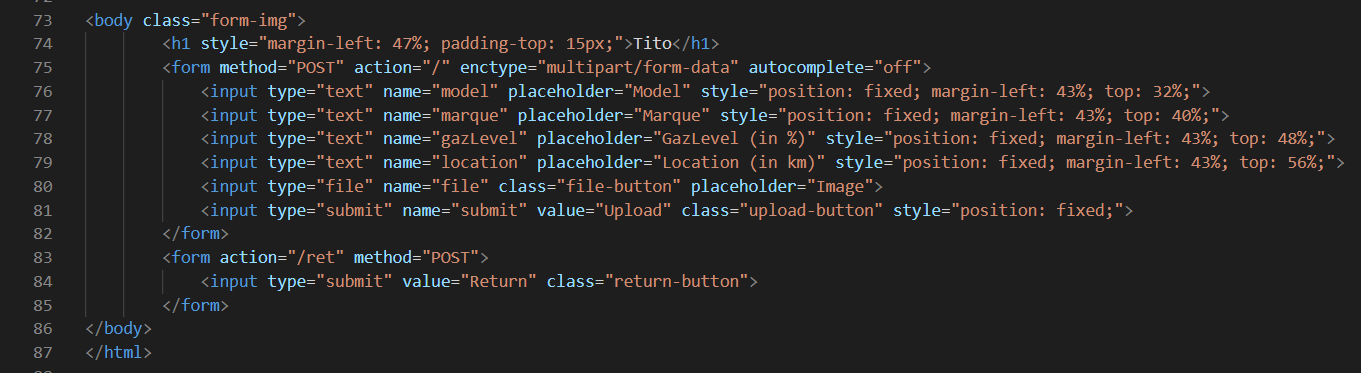
Imgheavy.html:



Index.html:







Dockerfile:



This Microservice allow to us to insert some element in our MySQL database through a html page.

Ip: 172.18.12.219:3333