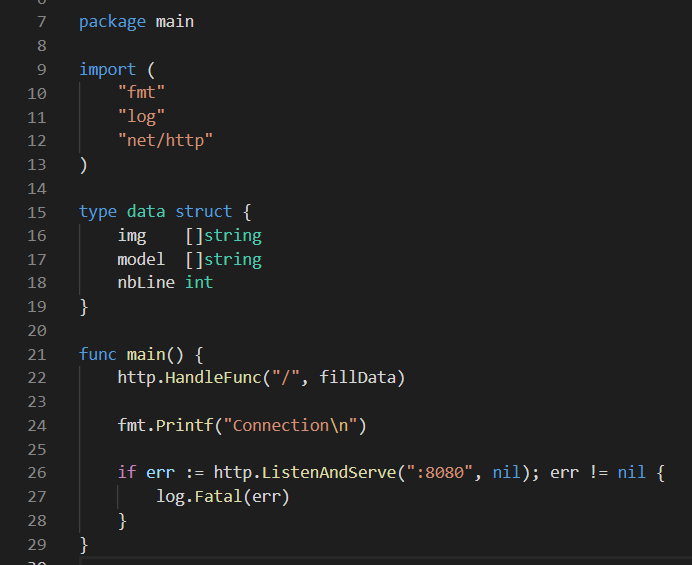
Documentation Microservice: Booking

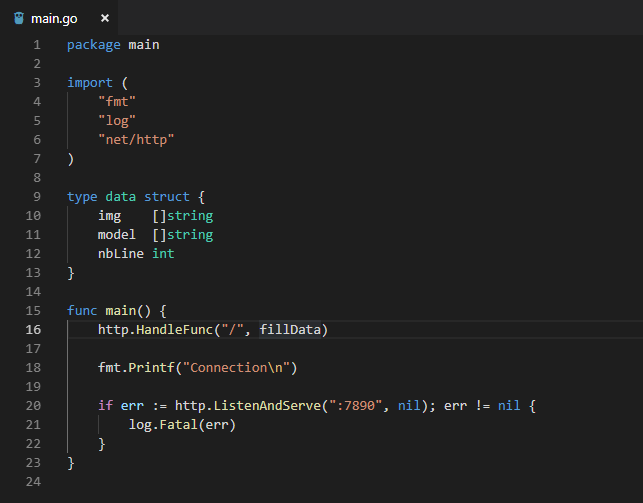
Goal: Allow to book our car through our virtual garage.

First, we go with the main:

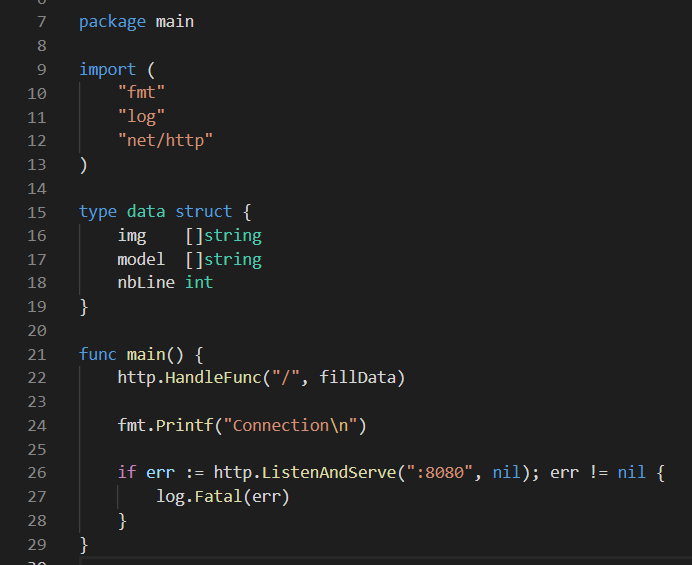
Main.go



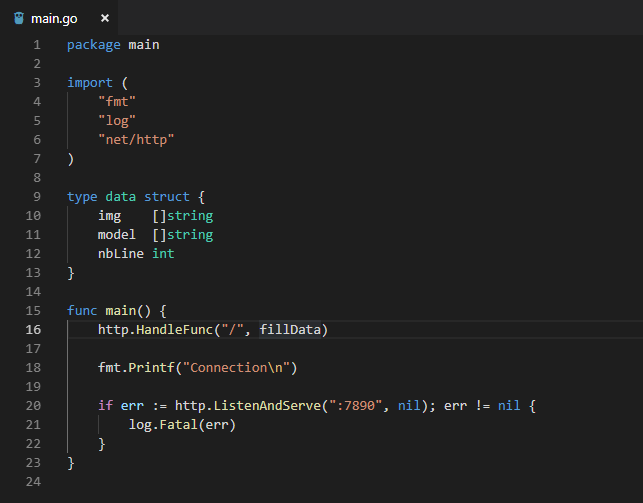
So in the main function, we declare our URL where we are going to work on:



And we declare our port that we want to work on. We have to check if there are no errors



We have also declare a structure to store our data. We will discuss of it forward in the documentation

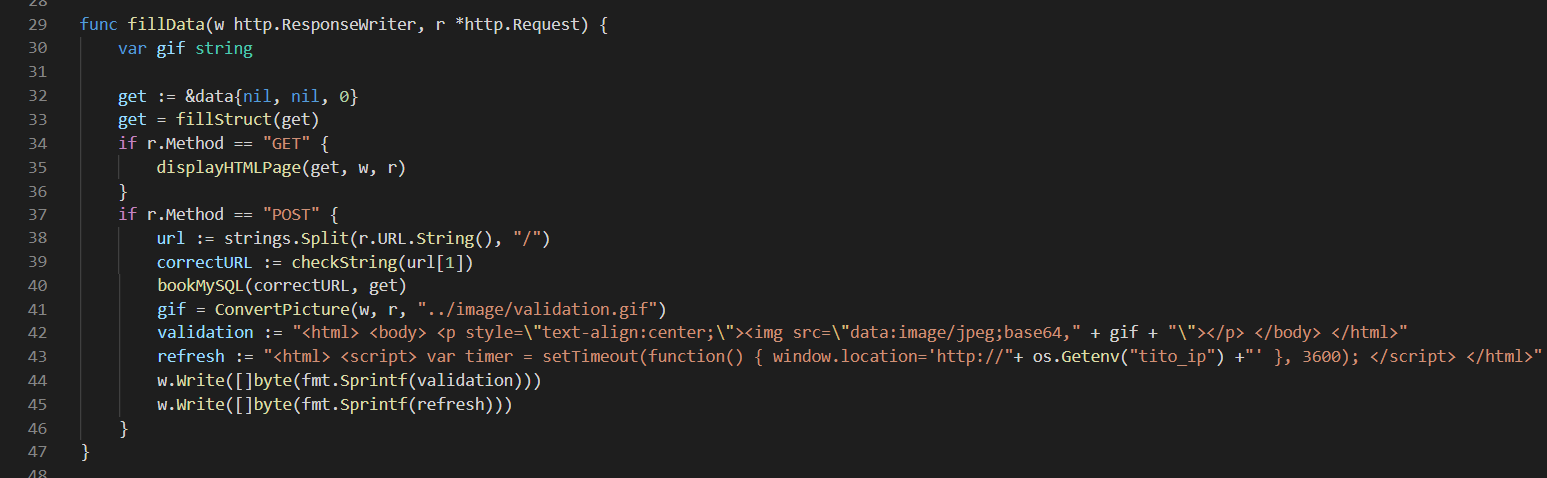


Our main function is here to declare our URL and the port that we are going to work on. We have also declare a structure

Data.go file:

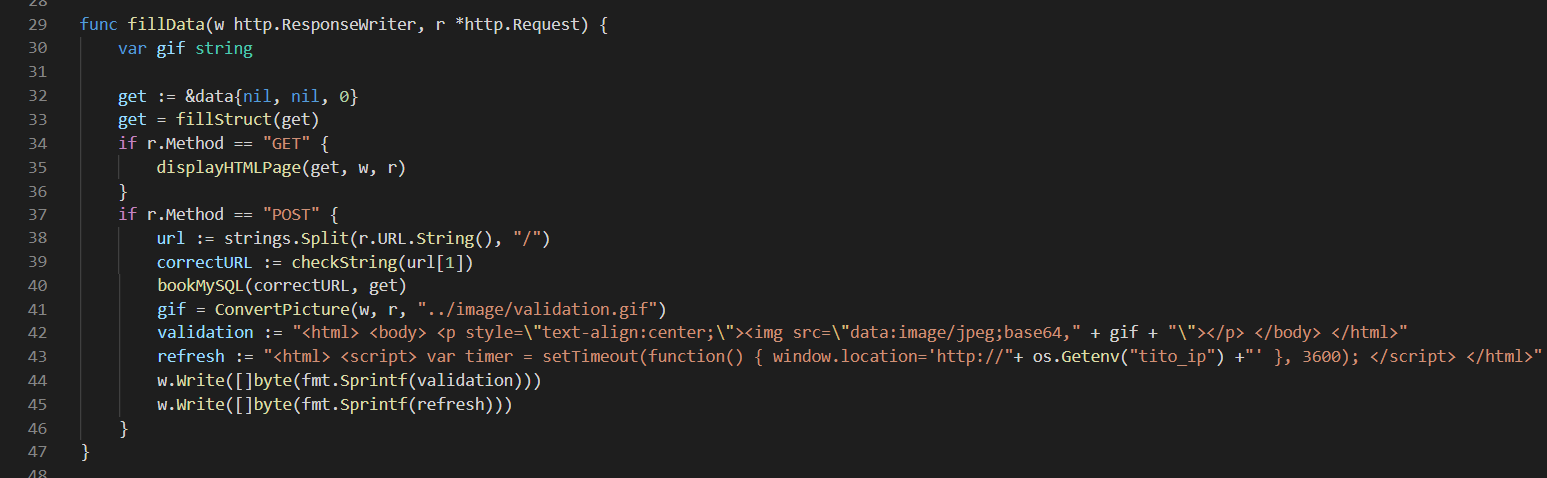


Line 20: db, err := sql.Open(""+ os.Getenv("db\_type") +"", ""+ os.Getenv("db\_username")+":"+ os.Getenv("db\_password") +"@tcp("+ os.Getenv("db\_ip") +")/"+ os.Getenv("db\_name") +"")

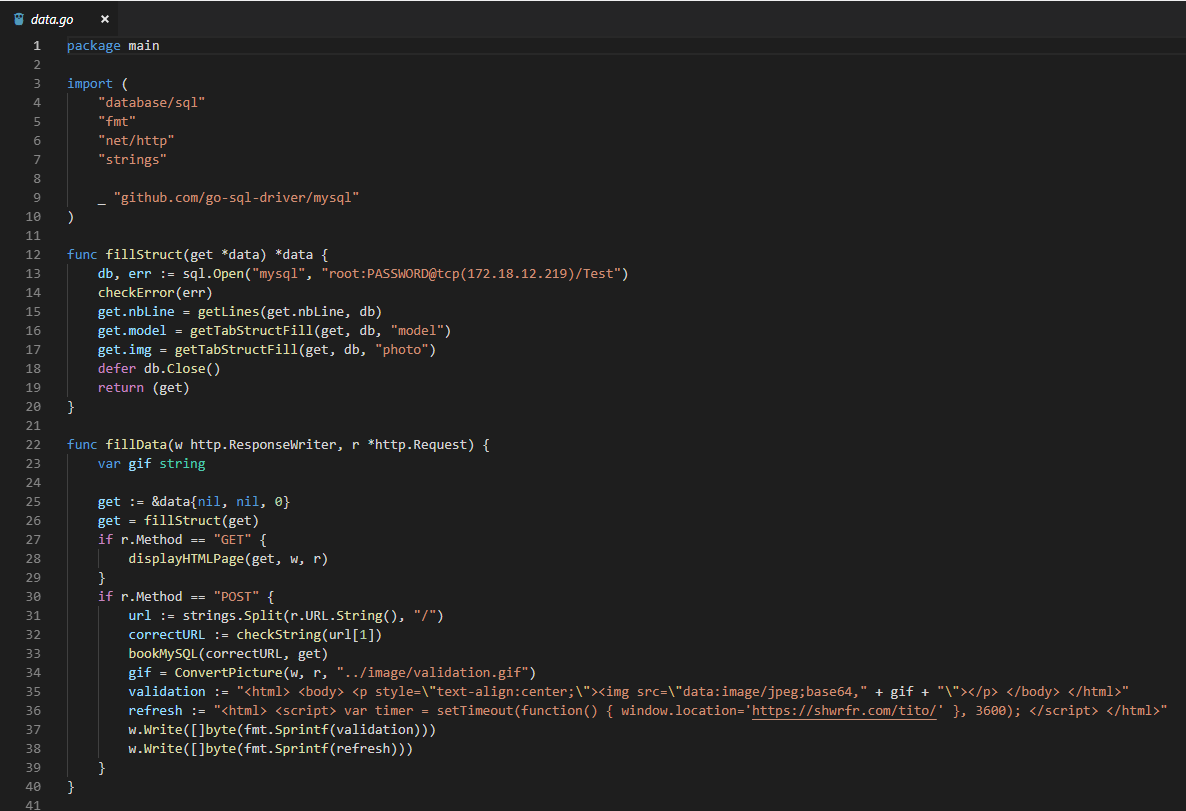


We are using en variable set in our Dockerfile.

First we start with the fillData function



Here in this function, we fill the element in our structure with this two lines



First we initialize our data in the structure and we fill them with the second line.

Then, in terms of the http query, we will launch different function.

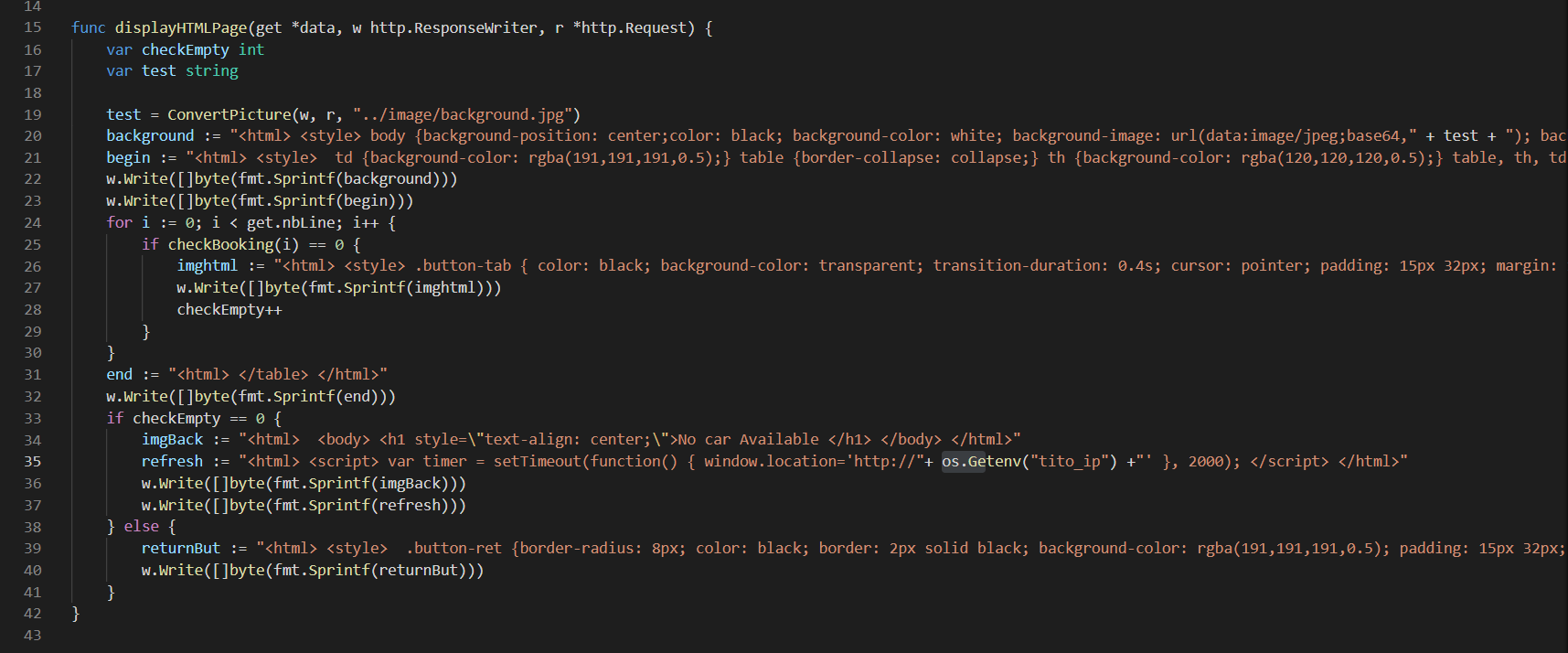
The GET request is when we launch our microservice. It will display a HTML page like this:



Here in this page, we can see our car that we can book. Also when a car is already book, it will not display it. To book a car, you have to press the button next to the car that you want.

Display.go file:

displayHTMLPage function :



Line 20: background := "<html> <style> body {background-position: center;color: black; background-color: white; background-image: url(data:image/jpeg;base64," + test + "); background-repeat: no-repeat; background-attachment: fixed; -webkit-background-size: cover; -moz-background-size: cover; -o-background-size: cover; background-size: cover;} </style> </html>"

Line 21: begin := "<html> <style> td {background-color: rgba(191,191,191,0.5);} table {border-collapse: collapse;} th {background-color: rgba(120,120,120,0.5);} table, th, td {border: 1px solid black} </style> <body> <table align=\"center\"> <th> Image </th> <th> Reservation </th> </body> </html>"

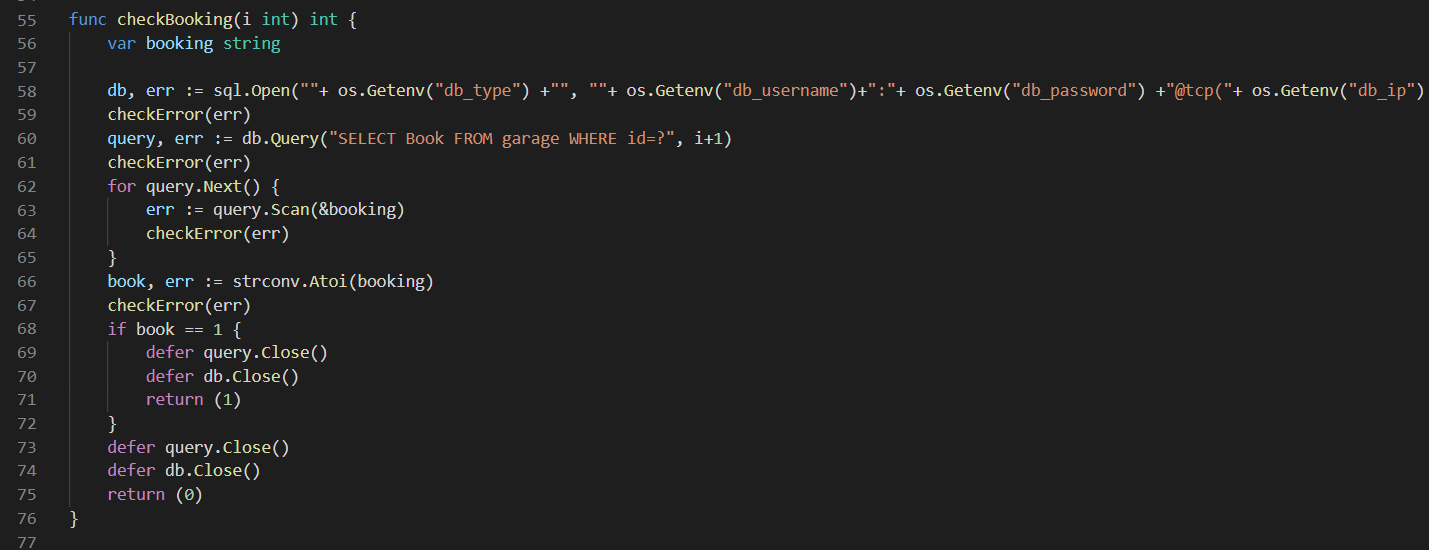
Line 26: imghtml := "<html> <style> .button-tab { color: black; background-color: transparent; transition-duration: 0.4s; cursor: pointer; padding: 15px 32px; margin: 10px; border-radius: 8px; border: 1px solid black; } .button-tab:hover { background-color: #01A30A; color: white; } </style> <body> <tr> <td> <img src=\"data:image/png;base64," + get.img[i] + " \" height=\"200\" width=\"240\"> </td> <form method=\"POST\" action=\"" + get.model[i] + "\"> <td> <input type=\"submit\" id=\"" + get.model[i] + "\"value=\"Book\" class=\"button-tab\"> <input type=\"hidden\" id=\"" + get.model[i] + "\"> </form> </td> </tr> </body> </html>"

Line 39: returnBut := "<html> <style> .button-ret {border-radius: 8px; color: black; border: 2px solid black; background-color: rgba(191,191,191,0.5); padding: 15px 32px; cursor: pointer; transition-duration: 0.4s;} .button-ret:hover { background-color: black; color: white; } </style> <body> <form action=\"http://"+ os.Getenv("tito\_ip") +"\"> <input type=\"submit\" Value=\"Return\" class=\"button-ret\"> </form> </body> </html>"

It only display our page just below.

We call a function name checkBooking. This function allow to us to display only the car that they are not book. It launch a mysql request to check if the car that we want to display is book are not.

Check.go file



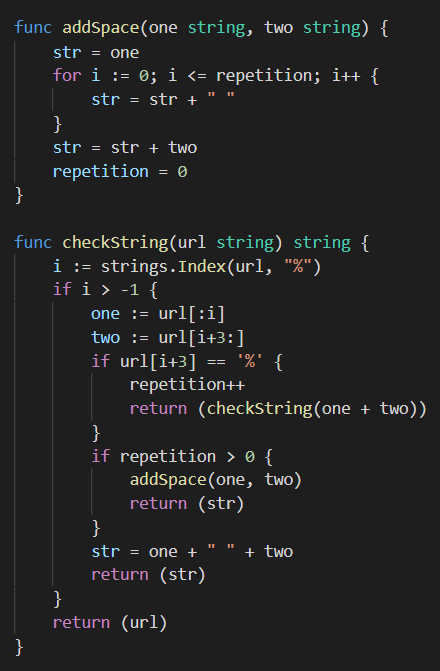
Line 58: db, err := sql.Open(""+ os.Getenv("db\_type") +"", ""+ os.Getenv("db\_username")+":"+ os.Getenv("db\_password") +"@tcp("+ os.Getenv("db\_ip") +")/"+ os.Getenv("db\_name") +"")

The POST request is on when you click on the button. It will send a request to our mysql database that we want to book this car. In the databe, it will change the boolean from 0 to 1, Meaning that the car is book.

Our method to check the button that we pressed is simple. We passed the model name of our car in the URL and we compare with our database.

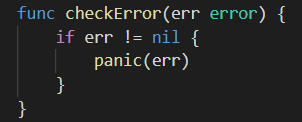
We aslo check if there are some spaces in the model. Because through the GET request, it will transform the space into a URL alphabet ( %20). We resolve this with this 2 functions:

Check.go file



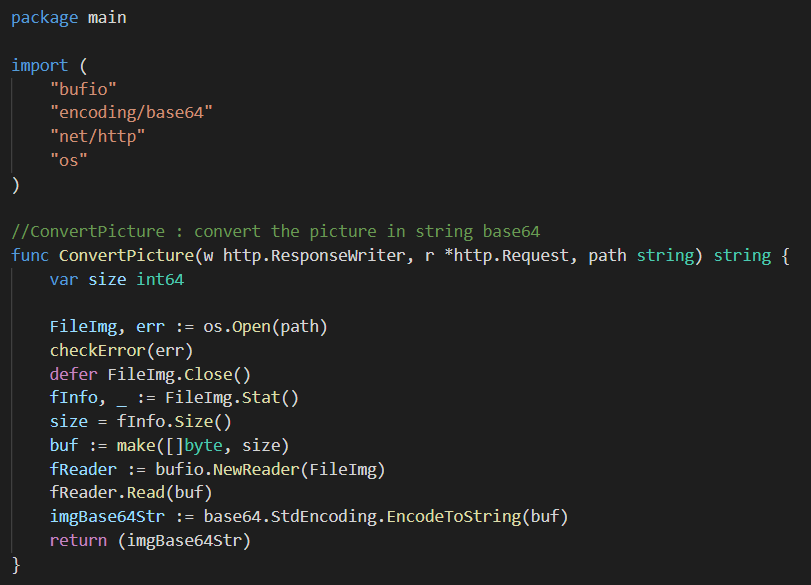
This function check if we have a % in our URL. If there is 1 or more, we increment our index and we copy the URL string in another variable that we return. If there is no space. We juste return the URL.

CheckError, a fucntion to check if there is an error when we call some function:



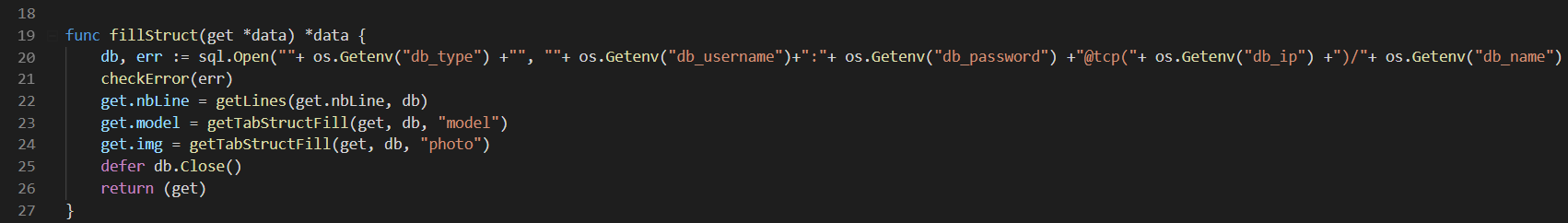
Also in the fillData function, we call ConvertPicture function that will convert a picture in stringBase64:

Picture.go file:



That allow to us to print the image in our HTML page.

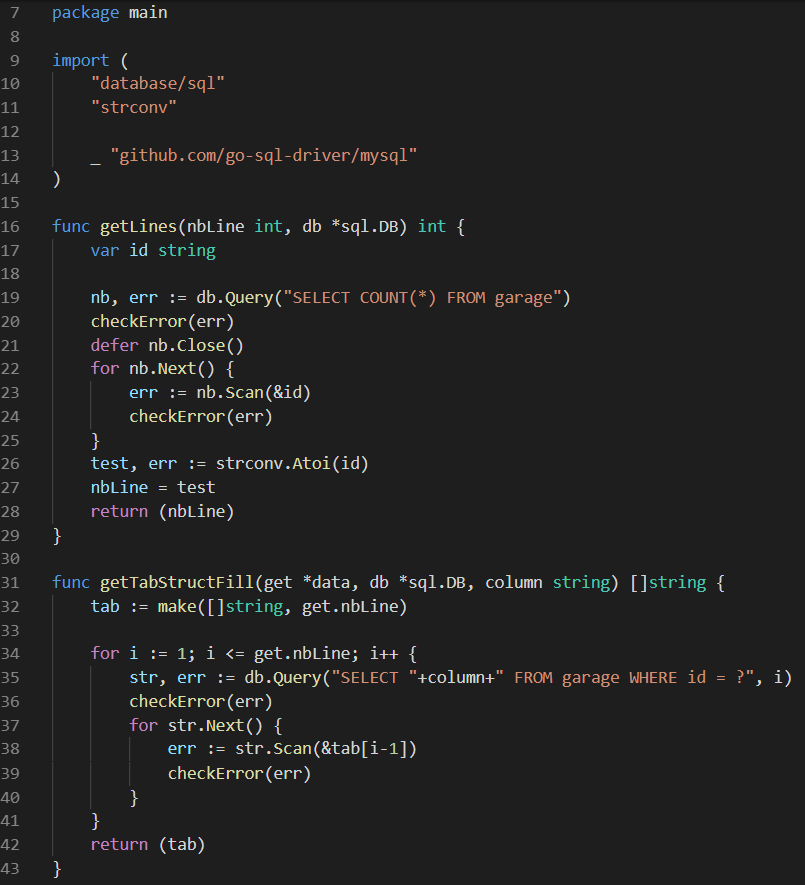
Then in the Data.go file we have also another function : fillStruct



Line 20: 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") +"")

It is just a mysql call to get the element in the database.

Get.go file:



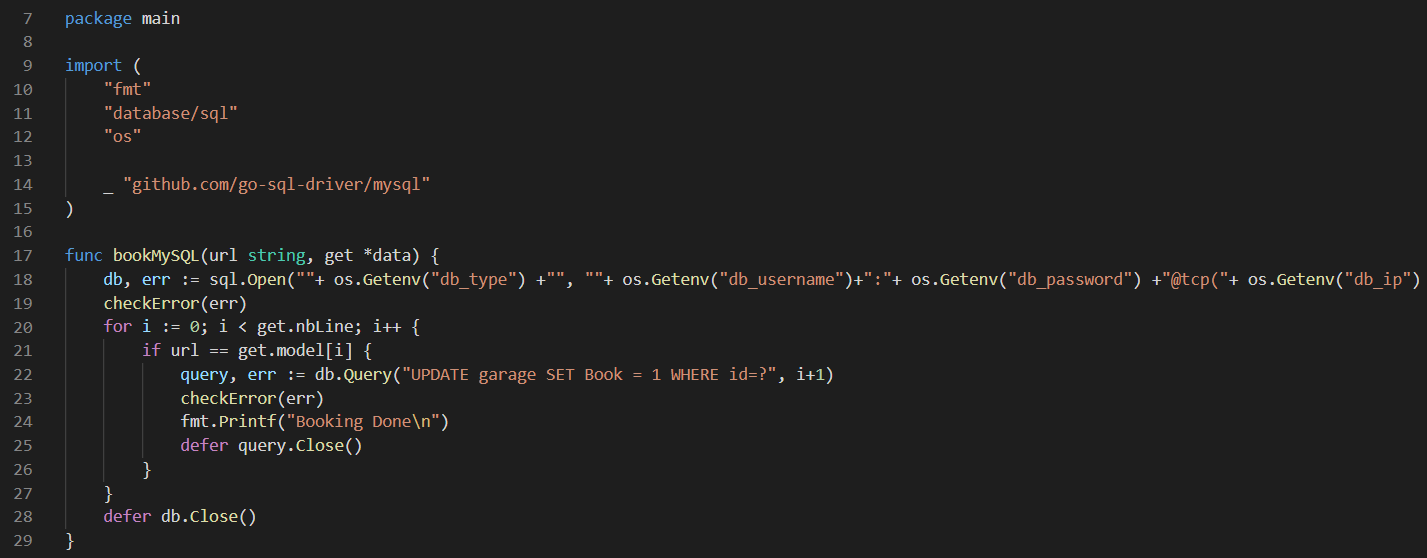
. nbLine get te number of line that we have in our database.

. model get the name of the car’s model

. img get the StringBase64 of each car in our database.

At the end, in the fillData function, we launch our booking of our car:

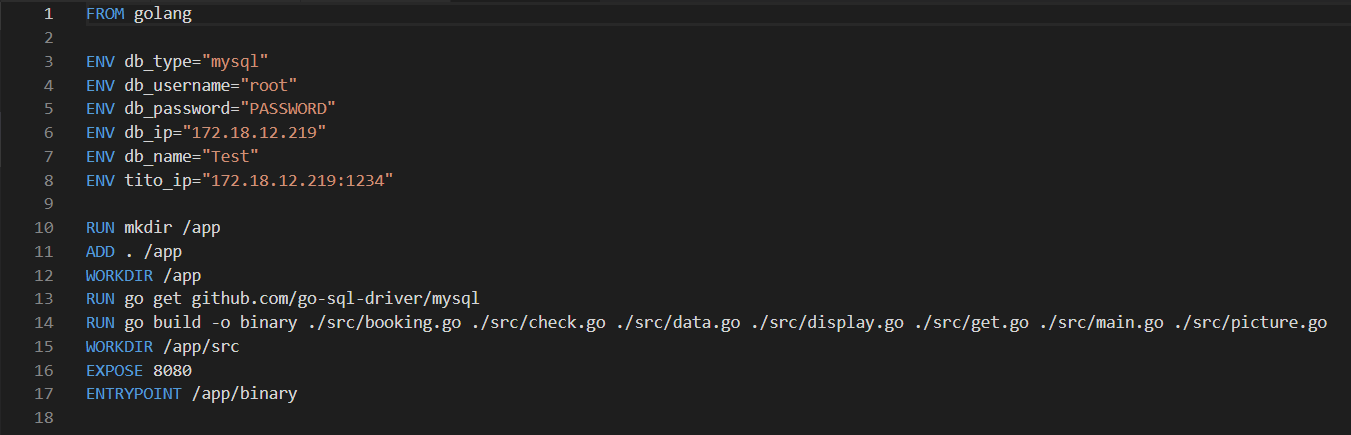
Booking.go file:



Line 18: 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") +"")

Here, we update our data in our database.

Dockerfile:



This microservice allow to us to book a car through a HTML page.

Ip: 172.18.12.219:2222