

GOLANG Midterm2 report

Ayaulym Parmash	200103265
Myrzaliyeva Zhibek	200103281
Amangeldi Diana	200103455
Zhanna Kanal	200103157

INTRODUCTION TO SERVICE

We continued to enhance our website . It contains html, css of filtering , rating , commenting parts by providing the required images, information about the product. There was a developed filtration part with customized range slider, interval in order to detect, classify, and organize products according to their price and rating by CSS/HTML. We add a new page for description of product's characteristics like price, color, size. A customer could arrive at the characteristic page by clicking the product's card on the main page . In addition , there was provided rating and comment in the description page . By appending those activities we tried to make a convenient , flexible user interface with eye-catching design, readable data and deliver insightful data and progress reports to website users.

TEAM MEMBERS (DESCRIPTION/CONTRIBUTION)

Ayaulym Parmash (200103265) :

GO -> Making a commenting part

GO -> Making a rating part

HTML/CSS -> Front of rating and comment

Zhibek Myrzaliyeva (200103281) :

GO -> Filtering items based on rating

GO -> Making the session

HTML/CSS -> Front of product's card

Diana Amangeldi (200103455) :

GO -> Filtering items based on price

Database -> Filing the database with needed information

Database -> Making the logic of table's connection

Zhanna Kanal (200103157) :

GO -> Making a rating part

HTML/CSS -> Front of filter

HTML/CSS -> Front of description page

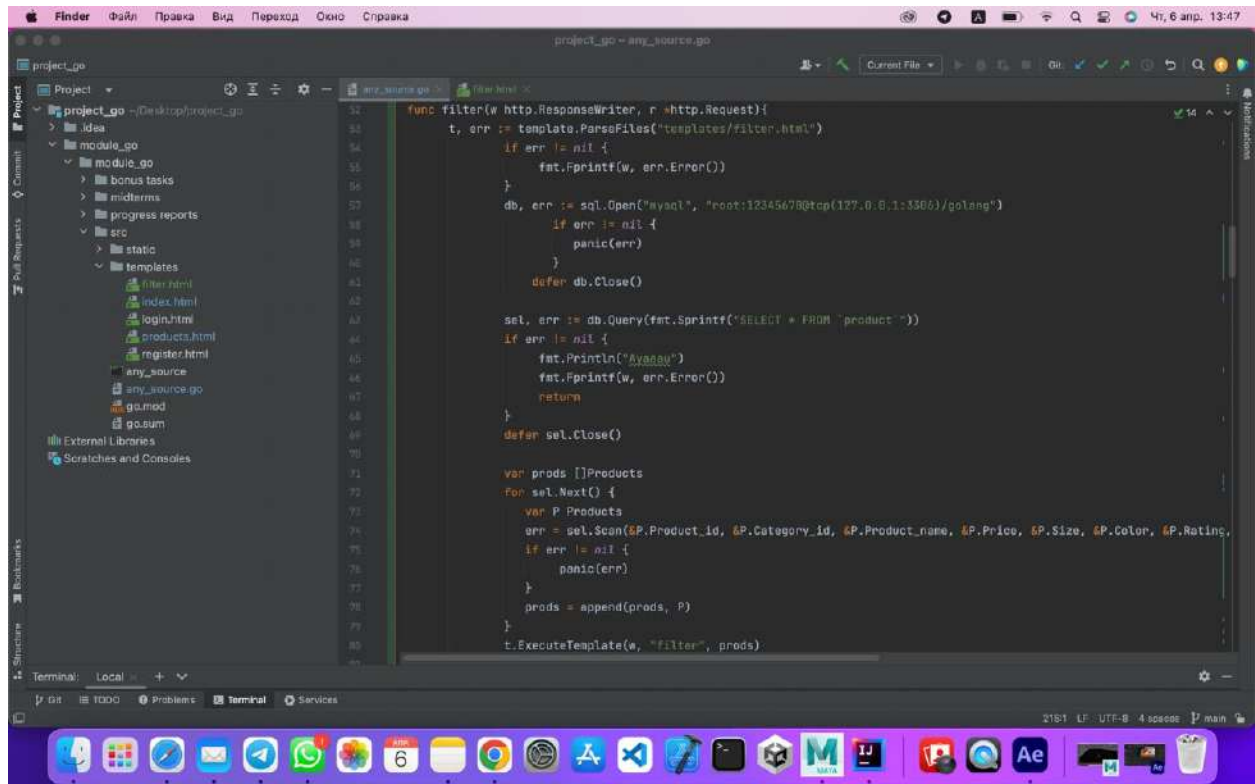
HOW TO RUN

To run the project you need to open in the git module_go/src and type "go run any_source.go".

EXPLANATION

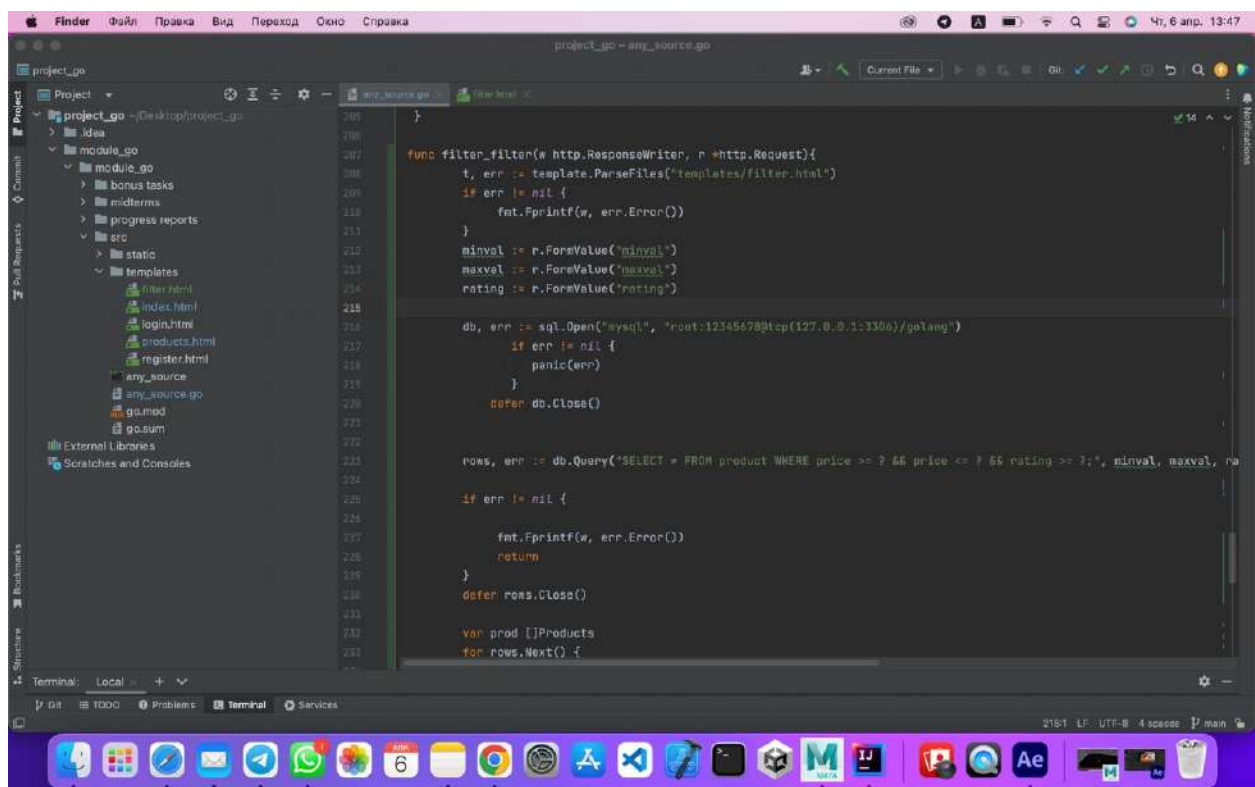
Customers can choose between a range of minimal and maximum qualities. It helps to control the price to obtain the necessary goods. It comes with 0 and 1000000 by default. A rating filter reveals products that go beyond the customer's desire by accepting data from the range slider. There are two functions in our code: filter and filter_filter. The filter function can be used by selecting the filter button on the home page. From the main page, where the filter is located, it sends a client to the filter page. The second filter_filter function displays information that matches the value supplied. Additionally, even if one

of the filters is changed, the default settings will still apply.



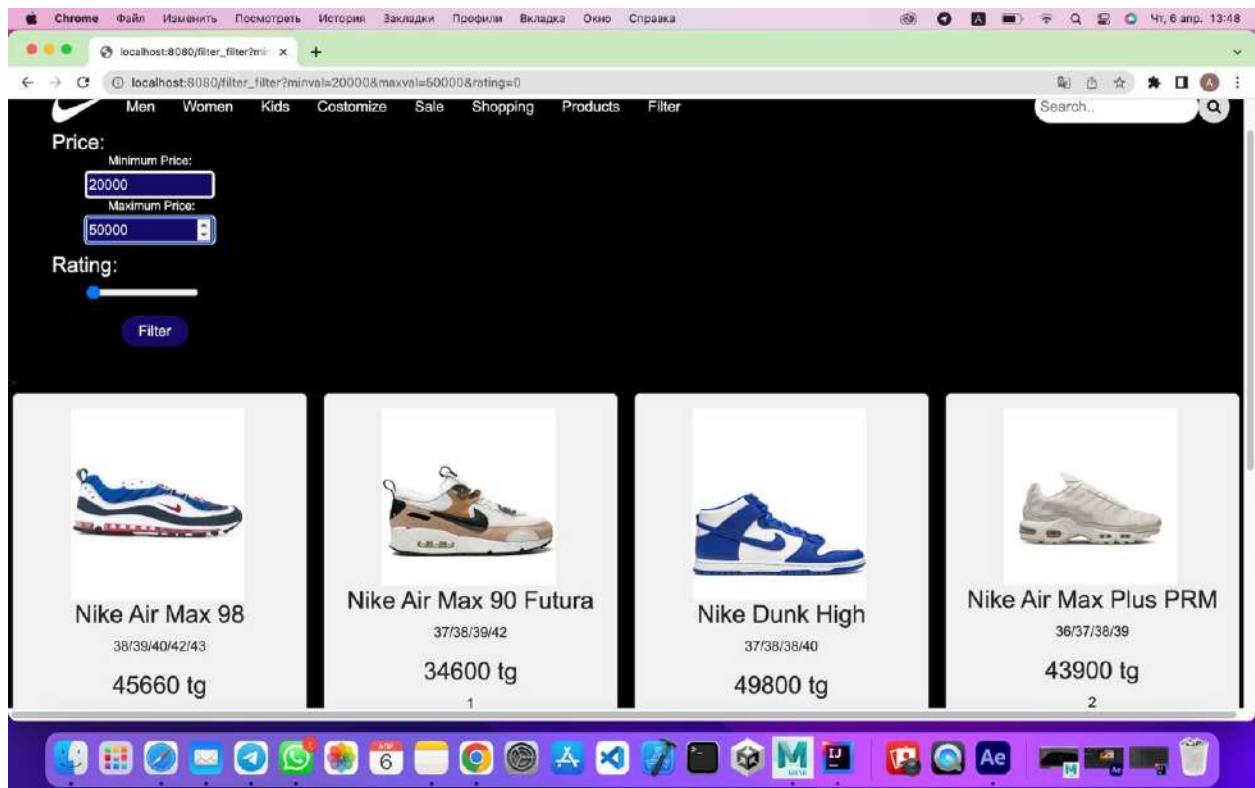
The screenshot shows an IDE window titled "project_go - any_source.go". The left sidebar displays a project structure with folders like "project_go", "idea", "module_go", "bonus tasks", "midterms", "progress reports", "src", "static", and "templates". The "templates" folder is expanded, showing files like "filter.html", "index.html", "login.html", "products.html", "register.html", "any_source", "any_source.go", "go.mod", and "go.sum". The main editor area shows the "any_source.go" file with the following Go code:

```
32 func filter(w http.ResponseWriter, r *http.Request){
33     t, err := template.ParseFiles("templates/filter.html")
34     if err != nil {
35         fat.Fprintf(w, err.Error())
36     }
37     db, err := sql.Open("mysql", "root:12345678@tcp(127.0.0.1:3306)/golang")
38     if err != nil {
39         panic(err)
40     }
41     defer db.Close()
42
43     sel, err := db.Query(fmt.Sprintf("SELECT * FROM `product`"))
44     if err != nil {
45         fat.Println("Aaaaaa")
46         fat.Fprintf(w, err.Error())
47         return
48     }
49     defer sel.Close()
50
51     var prods []Products
52     for sel.Next() {
53         var p Products
54         err = sel.Scan(&p.Product_id, &p.Category_id, &p.Product_name, &p.Price, &p.Size, &p.Color, &p.Rating)
55         if err != nil {
56             panic(err)
57         }
58         prods = append(prods, p)
59     }
60     t.ExecuteTemplate(w, "filter", prods)
61 }
```

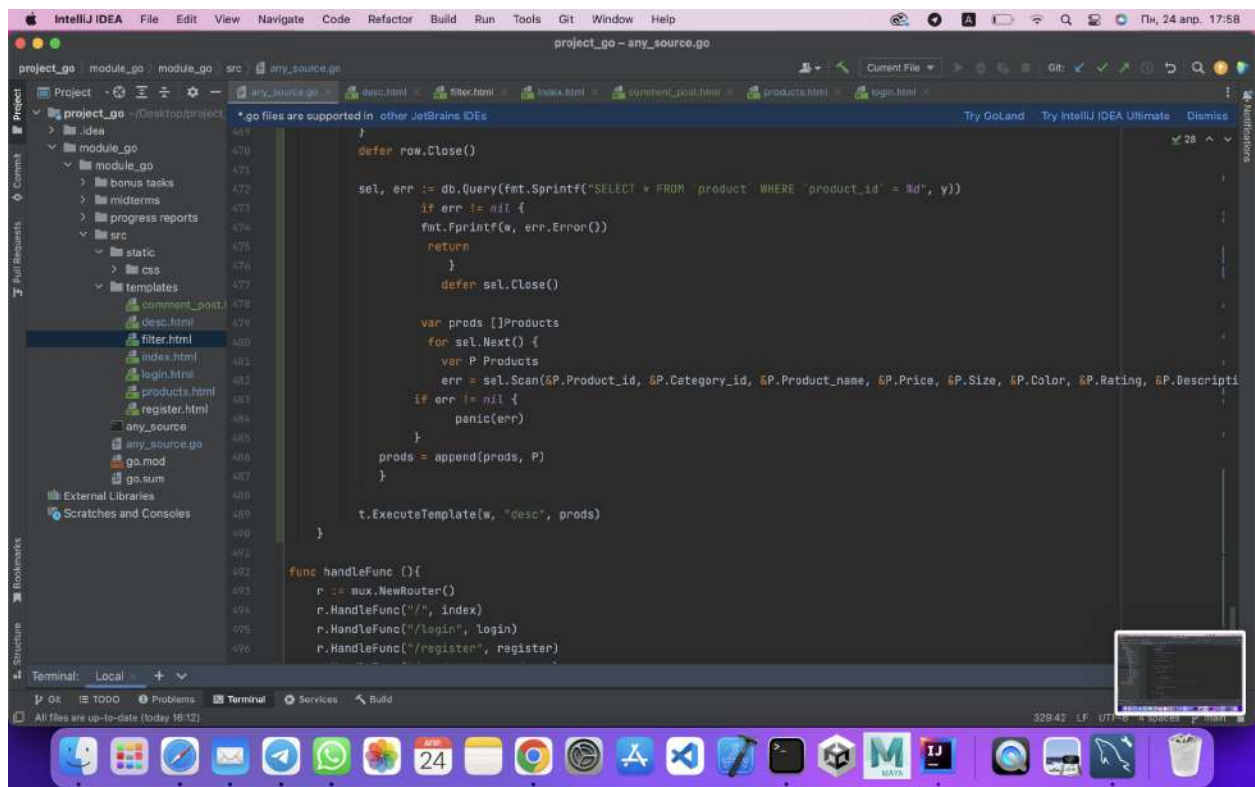
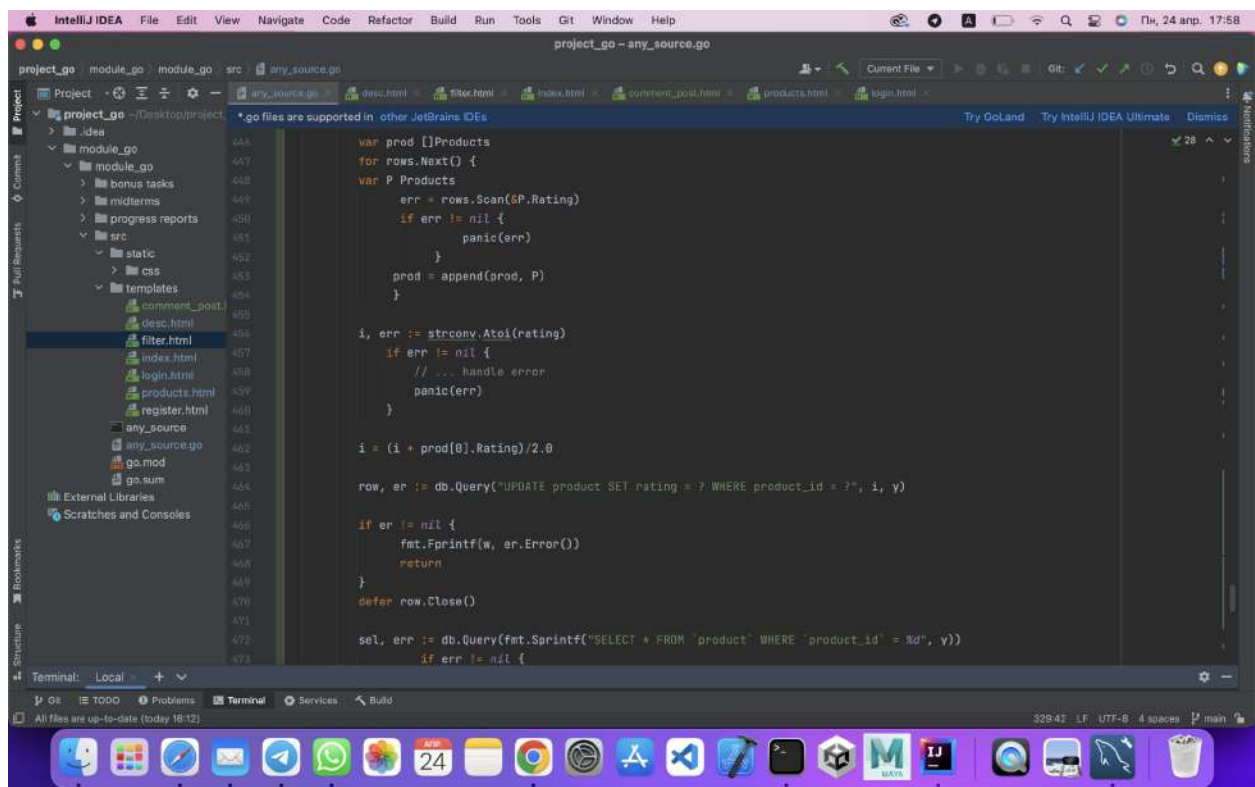


The screenshot shows the same IDE window, but the code in the "any_source.go" file has been updated to include a "filter_filter" function. The code is as follows:

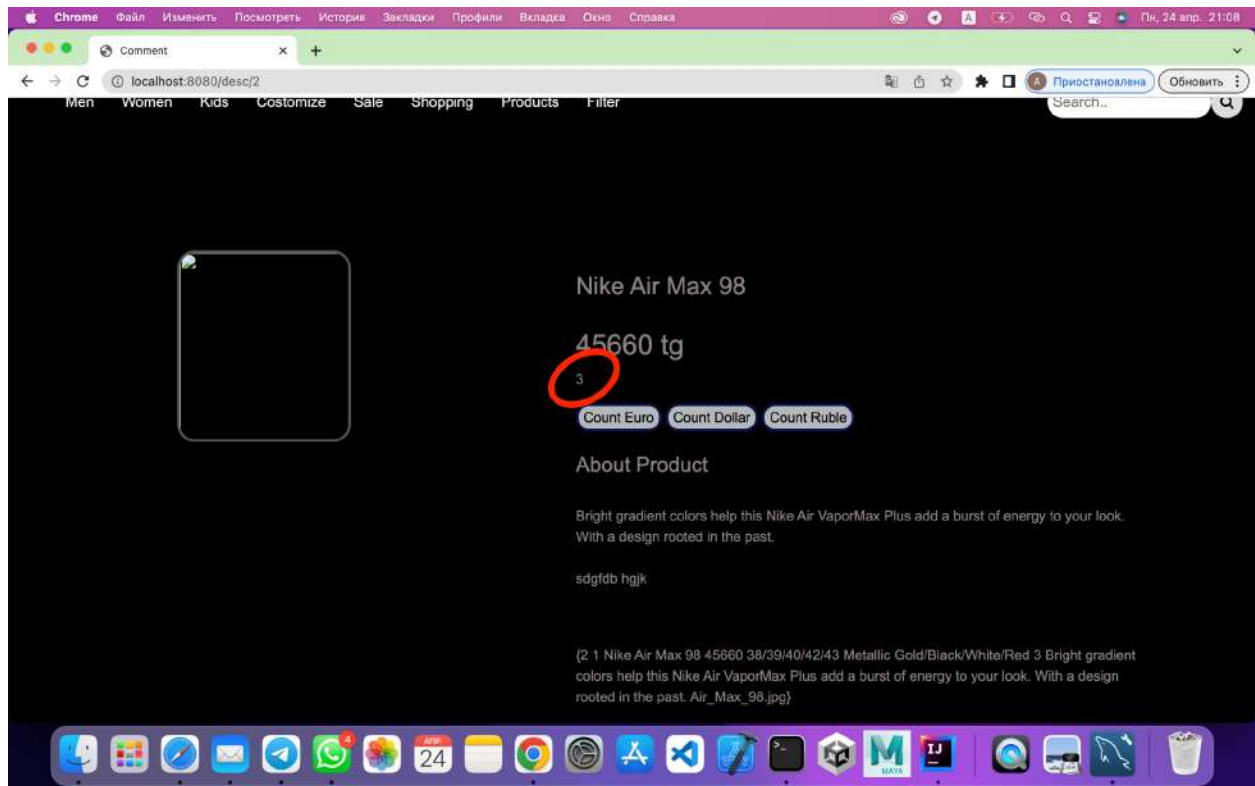
```
205 }
206
207 func filter_filter(w http.ResponseWriter, r *http.Request){
208     t, err := template.ParseFiles("templates/filter.html")
209     if err != nil {
210         fat.Fprintf(w, err.Error())
211     }
212     minval := r.FormValue("minval")
213     maxval := r.FormValue("maxval")
214     rating := r.FormValue("rating")
215
216     db, err := sql.Open("mysql", "root:12345678@tcp(127.0.0.1:3306)/golang")
217     if err != nil {
218         panic(err)
219     }
220     defer db.Close()
221
222     rows, err := db.Query("SELECT * FROM product WHERE price >= ? && price <= ? && rating > ?;", minval, maxval, rating)
223     if err != nil {
224         fat.Fprintf(w, err.Error())
225         return
226     }
227     defer rows.Close()
228
229     var prod []Products
230     for rows.Next() {
231         var p Products
232         err = p.Scan(rows.Scan(&p.Product_id, &p.Category_id, &p.Product_name, &p.Price, &p.Size, &p.Color, &p.Rating))
233         if err != nil {
234             panic(err)
235         }
236         prod = append(prod, p)
237     }
238     t.ExecuteTemplate(w, "filter", prod)
239 }
```

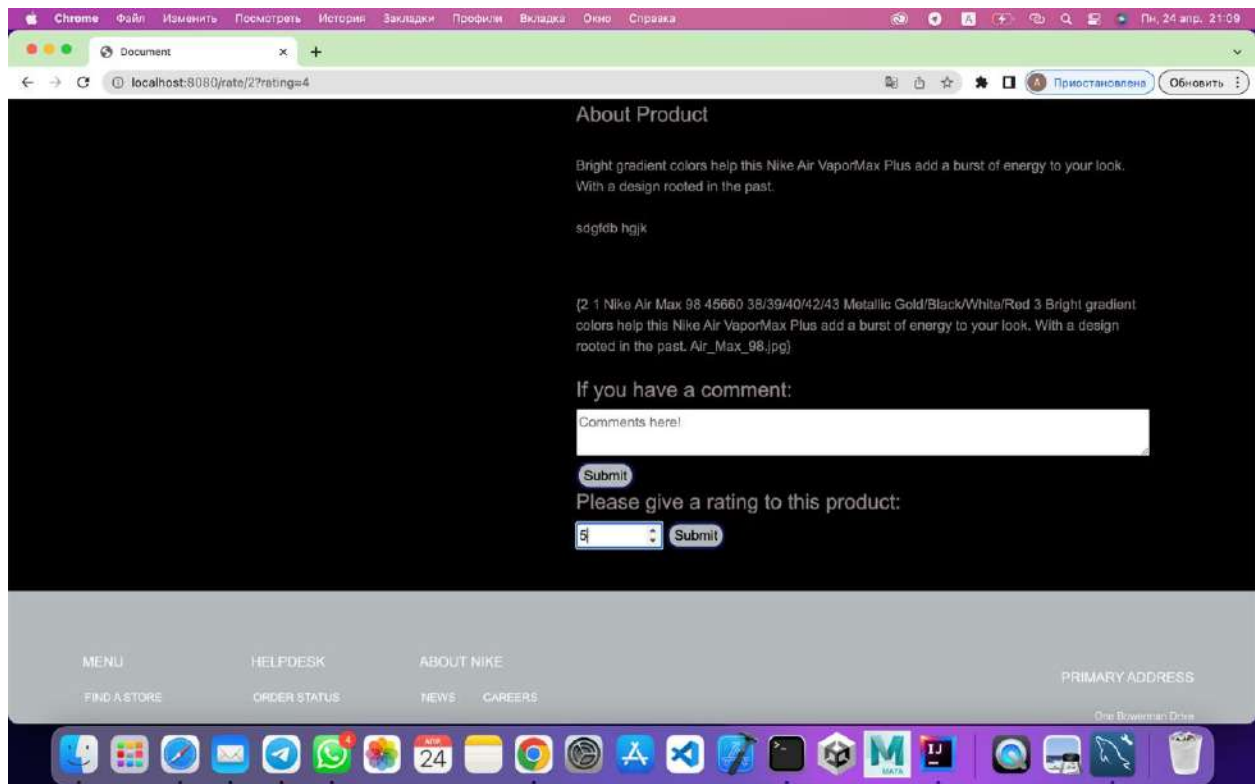
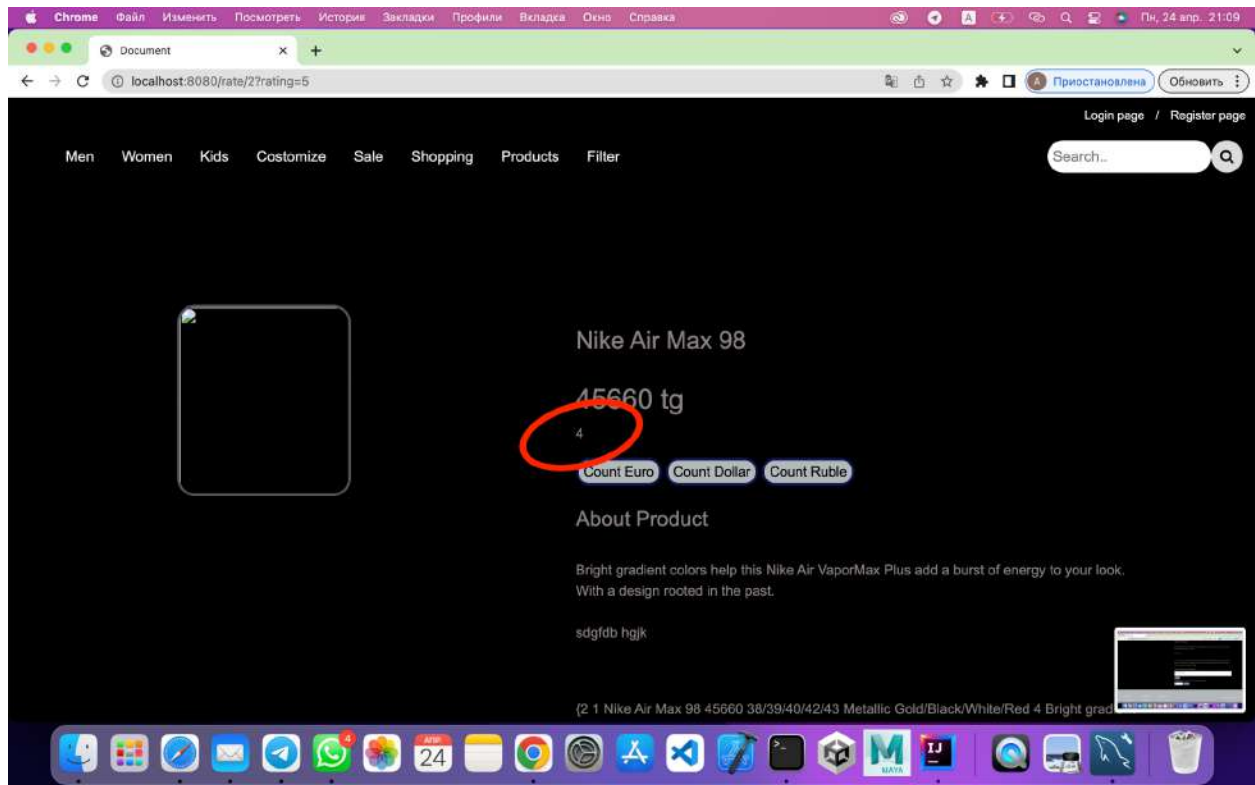


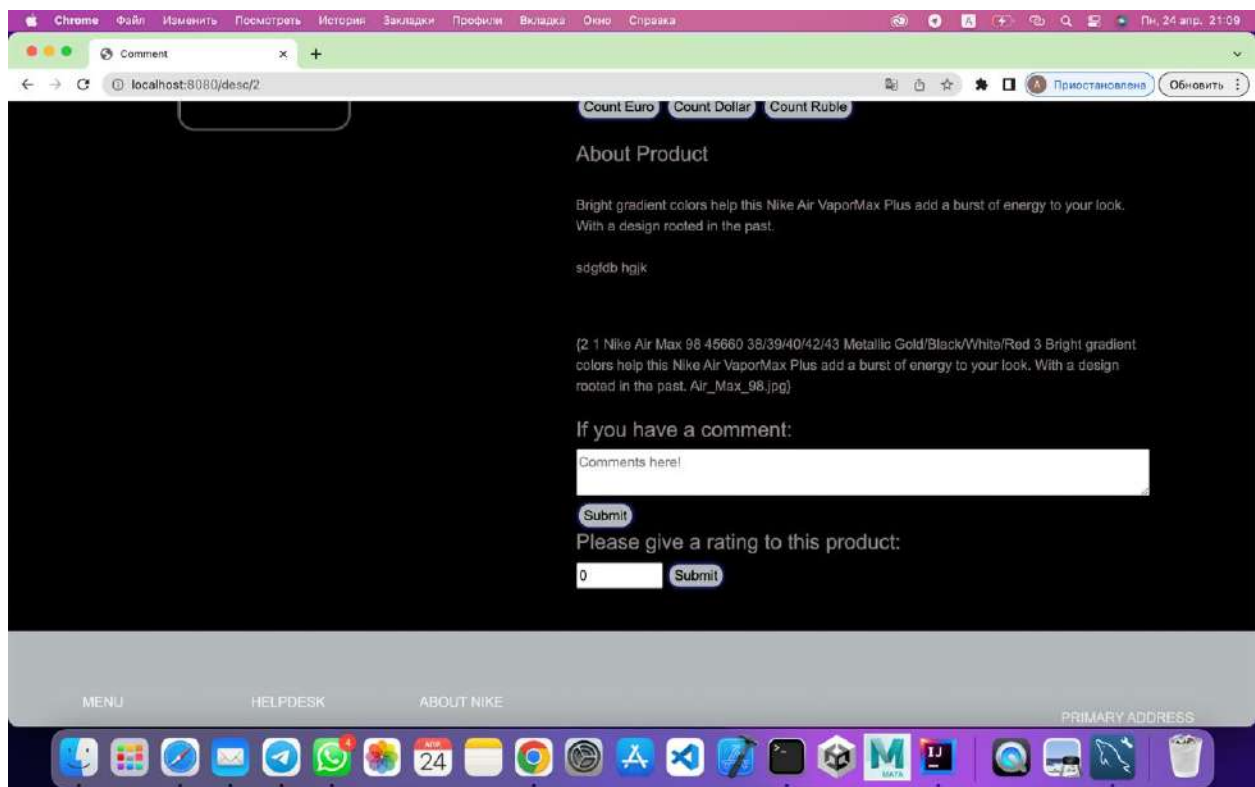
In the giving rating part, we created a new input area of type Number, which takes a number in the range of 0 and 5. After this form method, the value goes to the rate() func where the rating of the object will be taken from the database and added to the form value which we created. Then we update a table in the database with this value by “UPDATE product SET rating = ? WHERE product_id = ?” query. And after, the rate function will execute a description page with updated info.



Below, you can see an example of our web page where the rating of the product was 3 and we input 5 as our rating, so at the end we took $(3+5)/2=4$ as the rate of this product. Then, the value of the product's ranking is changed after each alteration in rating on the description page. As a rating we choose the input number type with the submit button and indication text. Each product has its own rating value which means average ratings do not affect others .







We added a textarea to front of our code and Submit button which leads to Comment func and created Comments struct. In the comment part, the Comment function takes Product_id of the product and uses it for queries and comment var where we store the front end's form value. After that, we do simple actions like opening the template and connecting to the db and in the main part we insert a comment to the comment table in the database by the query `Insert into `comment` (`product_id`, `comment`) Values ('%d', '%s')` where %d is the product_id of the product which we took before with mux.vars and %s is our comment which we input in the front.

And then, `SELECT * FROM `comment` WHERE `product_id` = %d` by the help of this query we selected all the info of the product in the comment table and retrieved and stored it as a Comment struct and saved every row of info to the array. After that, we created a map with string key and interface value to store and pass all this data to the desc template. In the front part, with this part we added all this info to our page.

```
{{ range .prod }}

    <p style="color: rgb(141, 131, 131); ">{{ .Comment }}</p>

{{ end }}
```


Here `{{range .prod}}` retrieves all the info in the prod array and `{ { .Comment } }` prints comments of these rows.

REFERENCES

GitHub -> https://github.com/Ayaulym2003/module_go.git