Rakhymbayeva Zhaniya
Madiyar Ernar

Our website for searching about cars selling.
Work parts:
Zhaniya

- Sending comment
- sending rating
- showing all comments and current rating in product page
- trigger to update avgRate

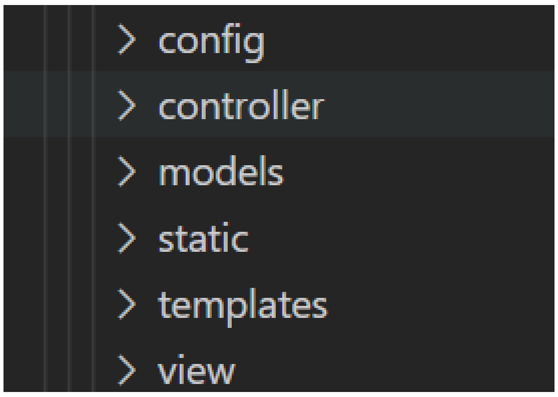
Ernar

- creating comments table
- table ratings
- filtering by price and rating in main page

To run code should run database in github named as sqlDB.sql and run code with go run main.go

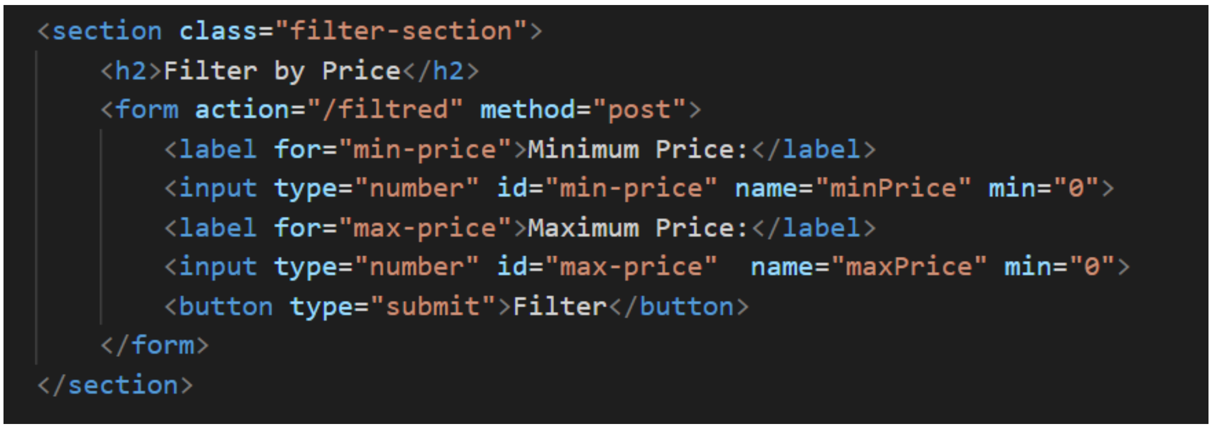
Во первых мы переделали структуру чтобы было легче понимать и сделать функционал более продуктивным.Для этого поделили 4 папки

1. Config
   a. db.go - connection to database
2. Controller - handlers for functionalities
   a. CommentHandler
   b. FiltredHandler -next
   c. ProductHandler - saves all functions and methods to show products info
   d. SearchHandle - the function to show searched list of product
3. Models - all data we need
   a. Comment
   b. Product
   c. ProductPage
   d. Search
   e. ToIndexData
4. View
   a. product
   b. login
   c. registration



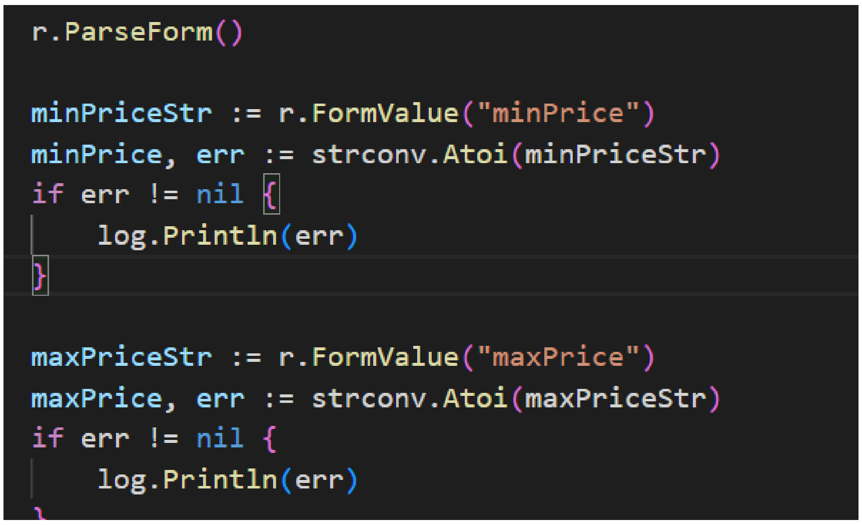
Filtering by price,rate

Form to post data



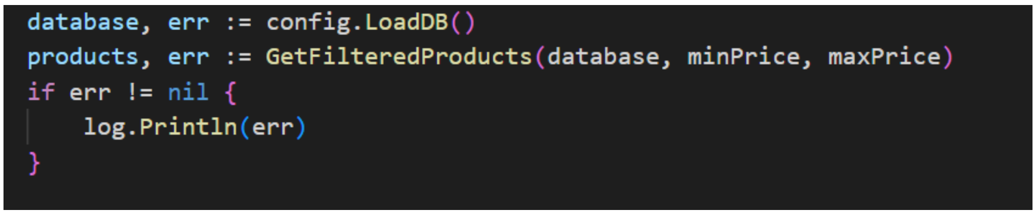
```
<section class="filter-section">
    <h2>Filter by Price</h2>
    <form action="/filtred" method="post">
        <label for="min-price">Minimum Price:</label>
        <input type="number" id="min-price" name="minPrice" min="0">
        <label for="max-price">Maximum Price:</label>
        <input type="number" id="max-price"  name="maxPrice" min="0">
        <button type="submit">Filter</button>
    </form>
</section>
```

Firstly, it gets data from Form with r.FormValue(“name”)
With convert it to string



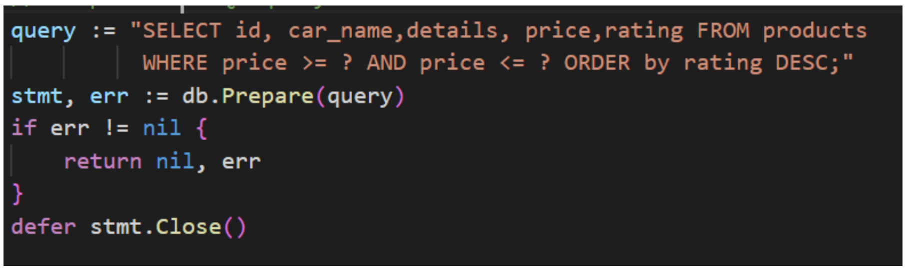
```
r.ParseForm()

minPriceStr := r.FormValue("minPrice")
minPrice, err := strconv.Atoi(minPriceStr)
if err != nil {
    log.Println(err)
}

maxPriceStr := r.FormValue("maxPrice")
maxPrice, err := strconv.Atoi(maxPriceStr)
if err != nil {
    log.Println(err)
```

Next step is connection to DB and calling method to get list of products
To the method it sends db and range of price



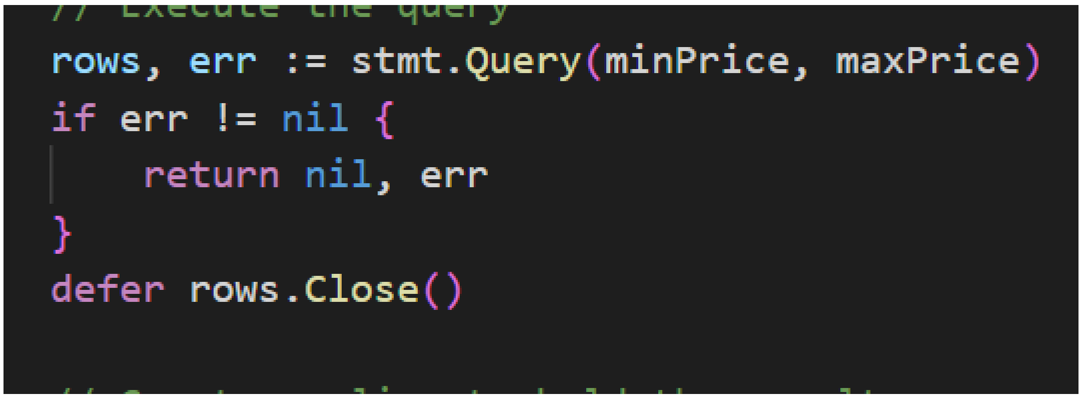
```
database, err := config.LoadDB()
products, err := GetFilteredProducts(database, minPrice, maxPrice)
if err != nil {
    log.Println(err)
}
```

Query to get a list of products which price between minPrice and maxPrice.



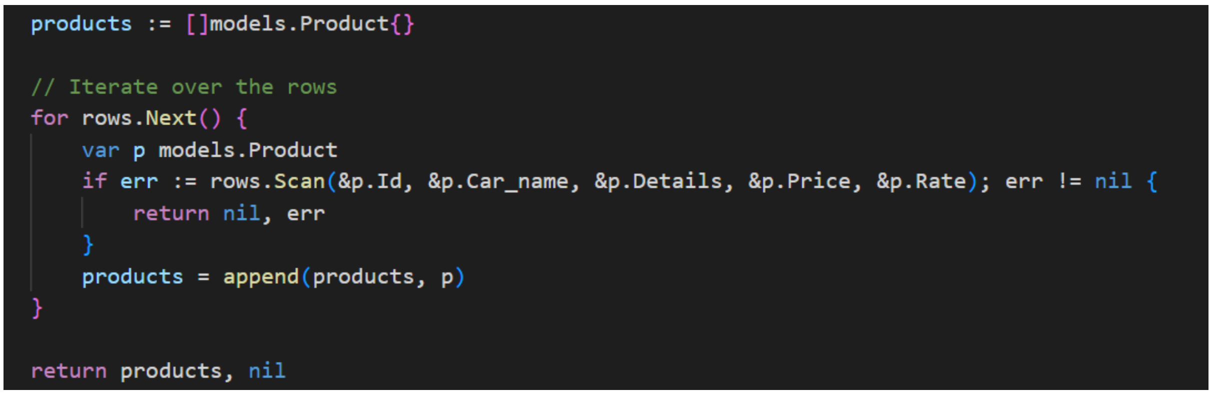
```
query := "SELECT id, car_name,details, price,rating FROM products
            WHERE price >= ? AND price <= ? ORDER by rating DESC;"
stmt, err := db.Prepare(query)
if err != nil {
    return nil, err
}
defer stmt.Close()
```

Executing the query to our DB which gives us rows of products



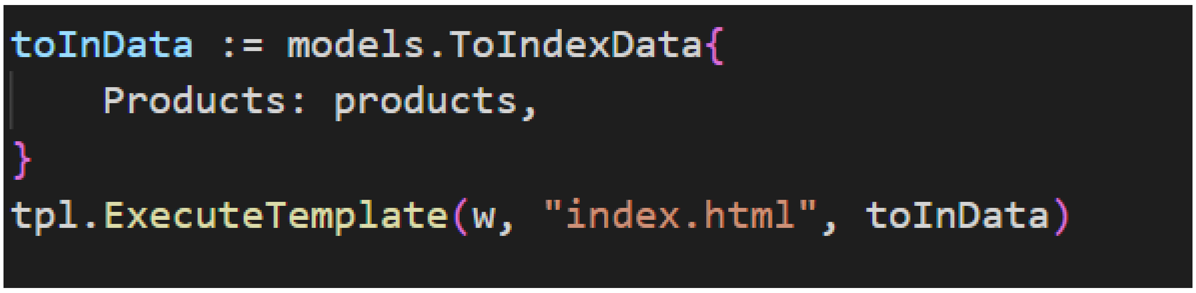
```go
// Execute the query
rows, err := stmt.Query(minPrice, maxPrice)
if err != nil {
    return nil, err
}
defer rows.Close()
```

From sql Rows rewrites it to Product and saves it to arrayist.
Return products list.



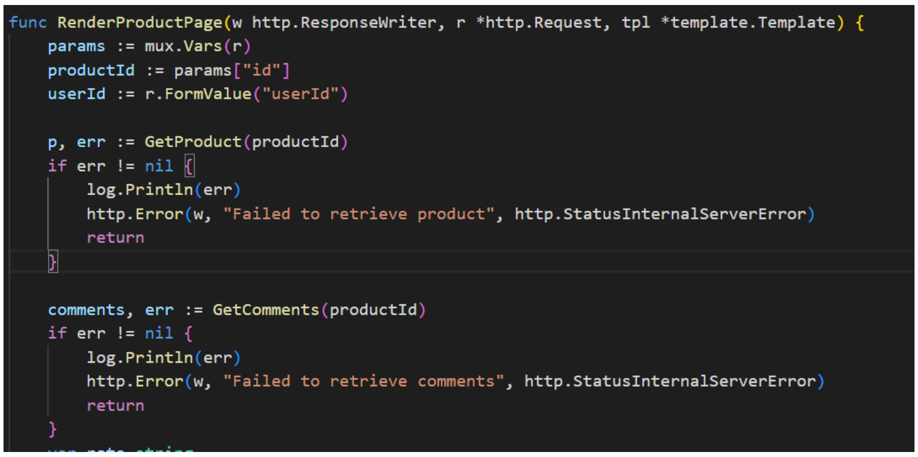
```go
products := []models.Product{}

// Iterate over the rows
for rows.Next() {
    var p models.Product
    if err := rows.Scan(&p.Id, &p.Car_name, &p.Details, &p.Price, &p.Rate); err != nil {
        return nil, err
    }
    products = append(products, p)
}

return products, nil
```

Saves data and sends to index.html template with the data.



```go
toInData := models.ToIndexData{
    Products: products,
}
tpl.ExecuteTemplate(w, "index.html", toInData)
```

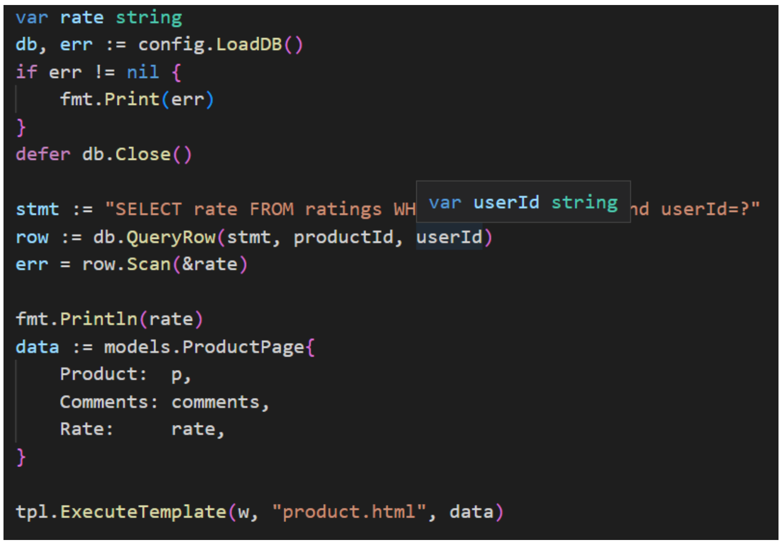
Rendering Product Page with comments and rating.



```go
func RenderProductPage(w http.ResponseWriter, r *http.Request, tpl *template.Template) {
    params := mux.Vars(r)
    productId := params["id"]
    userId := r.FormValue("userId")

    p, err := GetProduct(productId)
    if err != nil {
        log.Println(err)
        http.Error(w, "Failed to retrieve product", http.StatusInternalServerError)
        return
    }

    comments, err := GetComments(productId)
    if err != nil {
        log.Println(err)
        http.Error(w, "Failed to retrieve comments", http.StatusInternalServerError)
        return
    }
```

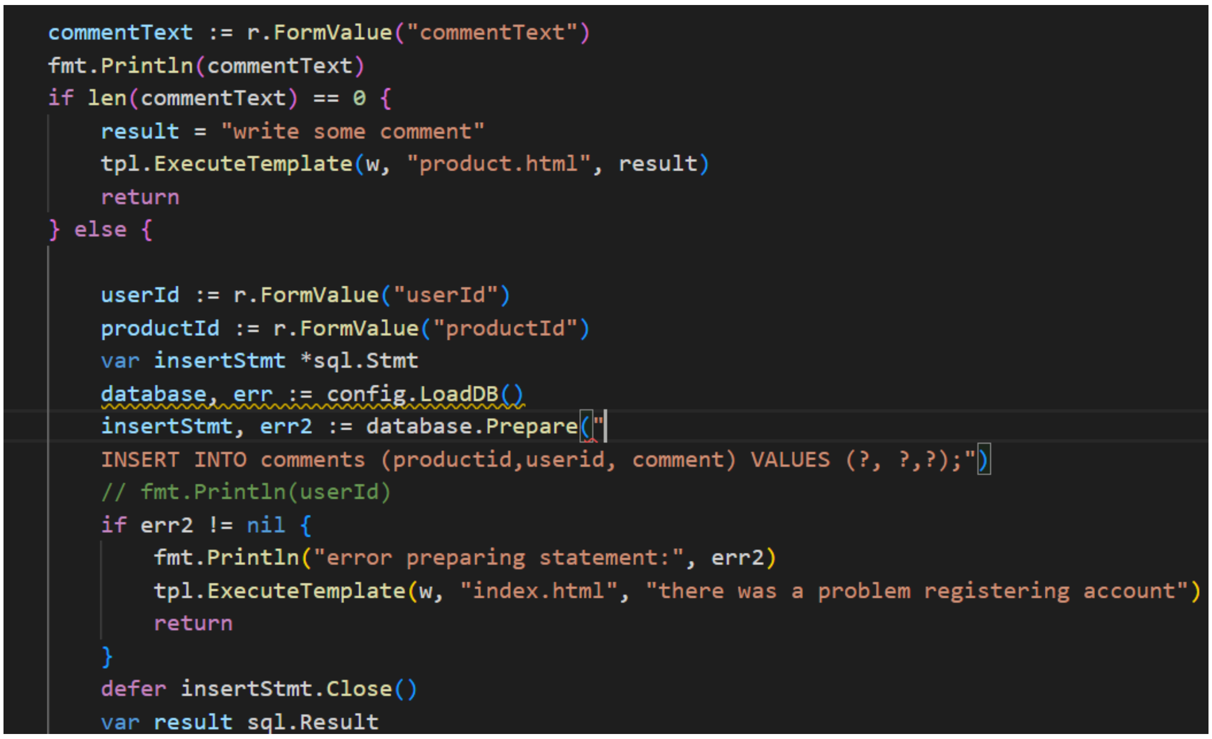
To show all info about product

1. get product info in total (in products table we alter column rating also)
2. get comments of the product
3. query to get rate of that user to show in front (already sent rating)
4. saves all data into one and sends to product.html template



```go
var rate string
db, err := config.LoadDB()
if err != nil {
    fmt.Print(err)
}
defer db.Close()

stmt := "SELECT rate FROM ratings WH var userId string nd userId=?"
row := db.QueryRow(stmt, productId, userId)
err = row.Scan(&rate)

fmt.Println(rate)
data := models.ProductPage{
    Product:  p,
    Comments: comments,
    Rate:     rate,
}

tpl.ExecuteTemplate(w, "product.html", data)
```

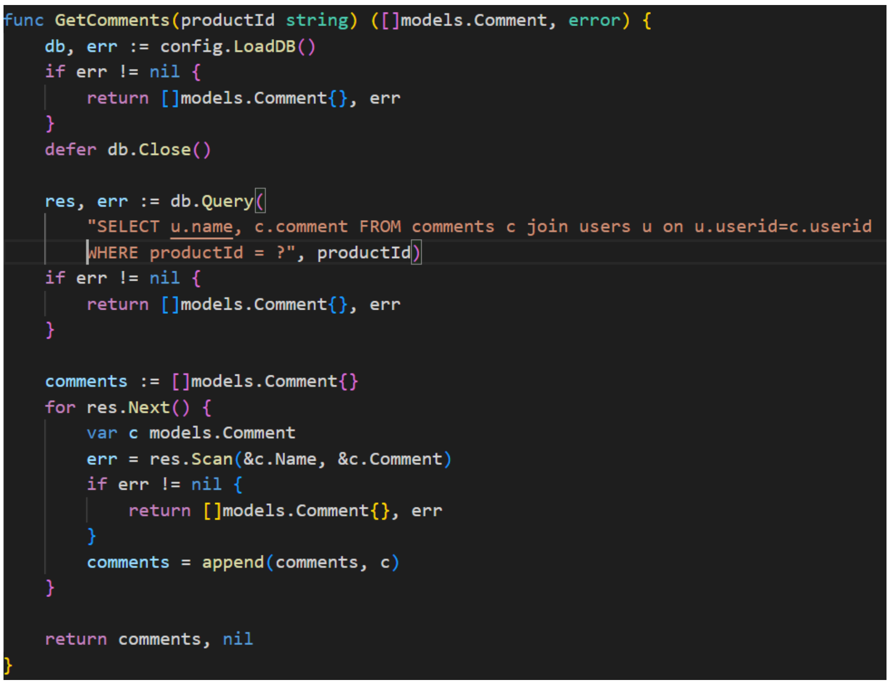
Comments section



```go
commentText := r.FormValue("commentText")
fmt.Println(commentText)
if len(commentText) == 0 {
    result = "write some comment"
    tpl.ExecuteTemplate(w, "product.html", result)
    return
} else {

    userId := r.FormValue("userId")
    productId := r.FormValue("productId")
    var insertStmt *sql.Stmt
    database, err := config.LoadDB()
    insertStmt, err2 := database.Prepare("
    INSERT INTO comments (productid,userid, comment) VALUES (?, ?,?);")
    // fmt.Println(userId)
    if err2 != nil {
        fmt.Println("error preparing statement:", err2)
        tpl.ExecuteTemplate(w, "index.html", "there was a problem registering account")
        return
    }
    defer insertStmt.Close()
    var result sql.Result
```

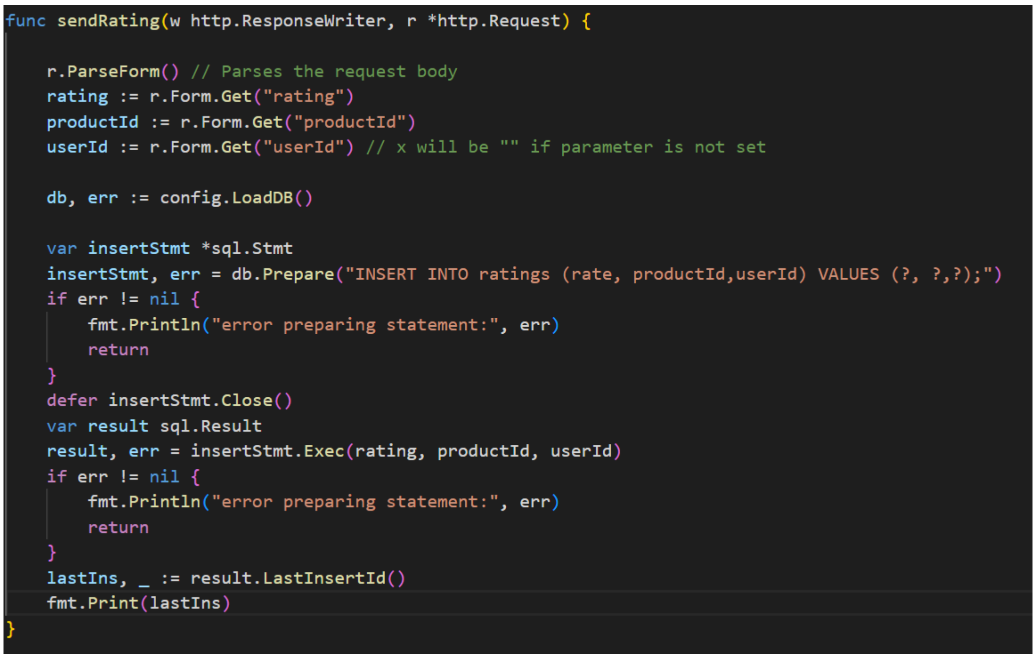
First it gets comments text and if its null response to write some comments.
Else it saves data (userId, productId) ,and connects to DB.
There showed a query to insert the comments.

After sending comment it reload page and shows all comments.(+just inserted one)



```go
func GetComments(productId string) ([]models.Comment, error) {
    db, err := config.LoadDB()
    if err != nil {
        return []models.Comment{}, err
    }
    defer db.Close()

    res, err := db.Query(
        "SELECT u.name, c.comment FROM comments c join users u on u.userid=c.userid
        WHERE productId = ?", productId)
    if err != nil {
        return []models.Comment{}, err
    }

    comments := []models.Comment{}
    for res.Next() {
        var c models.Comment
        err = res.Scan(&c.Name, &c.Comment)
        if err != nil {
            return []models.Comment{}, err
        }
        comments = append(comments, c)
    }

    return comments, nil
}
```

GetComment return a list of comments of that product.



```go
func sendRating(w http.ResponseWriter, r *http.Request) {

    r.ParseForm() // Parses the request body
    rating := r.Form.Get("rating")
    productId := r.Form.Get("productId")
    userId := r.Form.Get("userId") // x will be "" if parameter is not set

    db, err := config.LoadDB()

    var insertStmt *sql.Stmt
    insertStmt, err = db.Prepare("INSERT INTO ratings (rate, productId,userId) VALUES (?, ?,?);")
    if err != nil {
        fmt.Println("error preparing statement:", err)
        return
    }
    defer insertStmt.Close()
    var result sql.Result
    result, err = insertStmt.Exec(rating, productId, userId)
    if err != nil {
        fmt.Println("error preparing statement:", err)
        return
    }
    lastIns, _ := result.LastInsertId()
    fmt.Print(lastIns)
}
```

To send Rating it gets all the info and connects to DB. After as u see there is a query to insert it into the ratings table.

Also this is a trigger to update product AvgRate
CREATE TRIGGER update_product_rating
AFTER INSERT ON ratings
FOR EACH ROW
BEGIN
UPDATE products p SET p.rating =
(SELECT ROUND(AVG(rate),2)

```
FROM ratings WHERE productId = NEW.productId)
WHERE p.id = NEW.productId;
END;
```