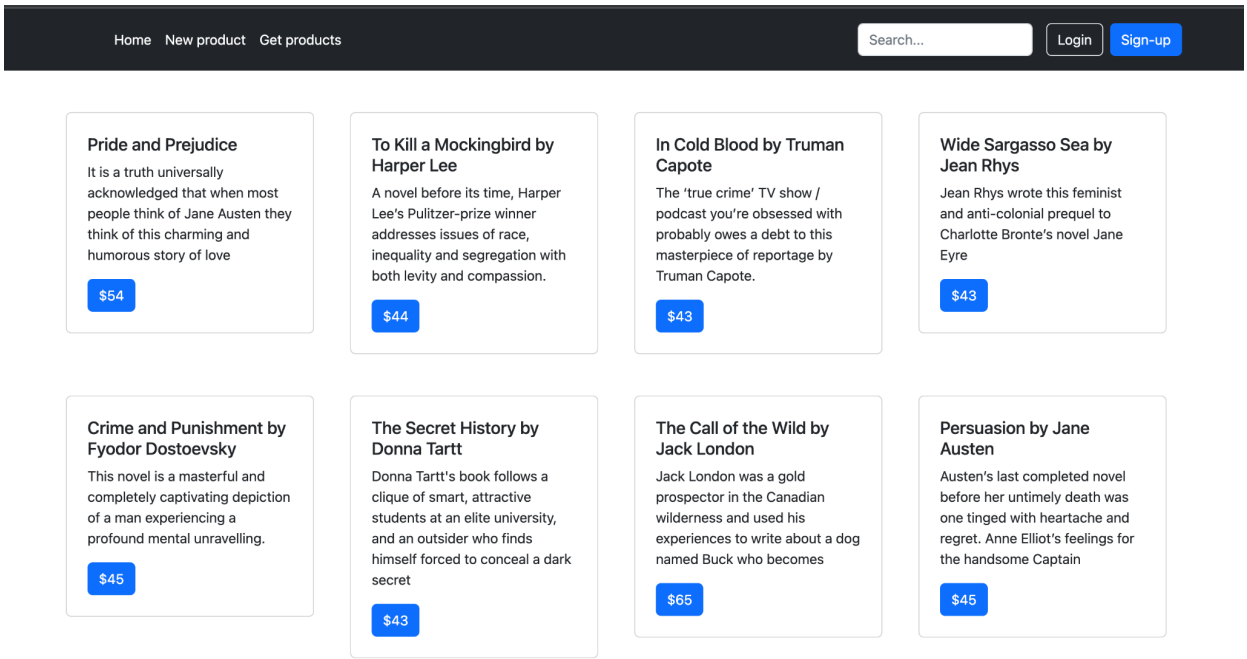
1 Introduction to the Service

Our project is based on a bookstore like Marwin, Meloman.



As you can see in the main page the books are displayed.

2. Team members:

Zhandos Saparbayev 200103201 - Did authorization form, the searching function

Nursultan Myrzagulov 200103440 - Made a connection with the database, registration form

Sairan Zhiger 200113005 - Created a new database products, made a connection with it, and did the “add products” function

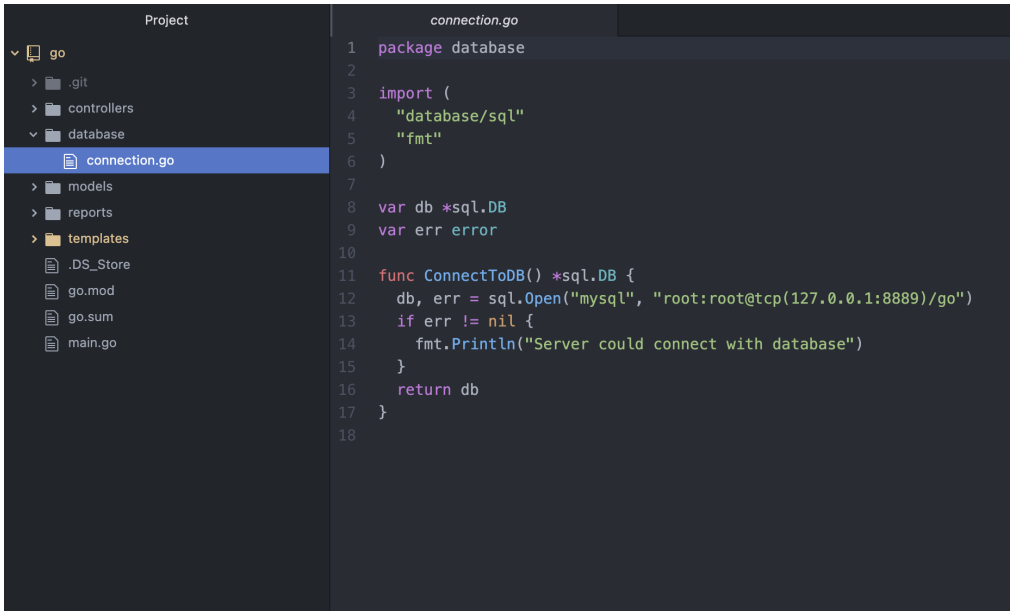
3. How to run the code.
Explanation: Firstly you need to download our project to your laptop or PC. If you don't have a PL Go, then we need to install it. Secondly you need to change the connection to the database for yours. For this you need to go to the connection.go file inside of the database folder. There are some comments which may help you.
Then open the terminal from the project folder and type “ go run main.go”. Packages will be downloaded automatically, if it doesn’t you need to download it by hand. All package dependencies you can find in the go.mod file. Ex: “`go get github.com/go-sql-driver/mysql`”. Then again run the “go run main.go” command. Then all should work normally.

4 Explanation of each feature with screen of code and the output result

“main.go” - Here we call the functions for different links

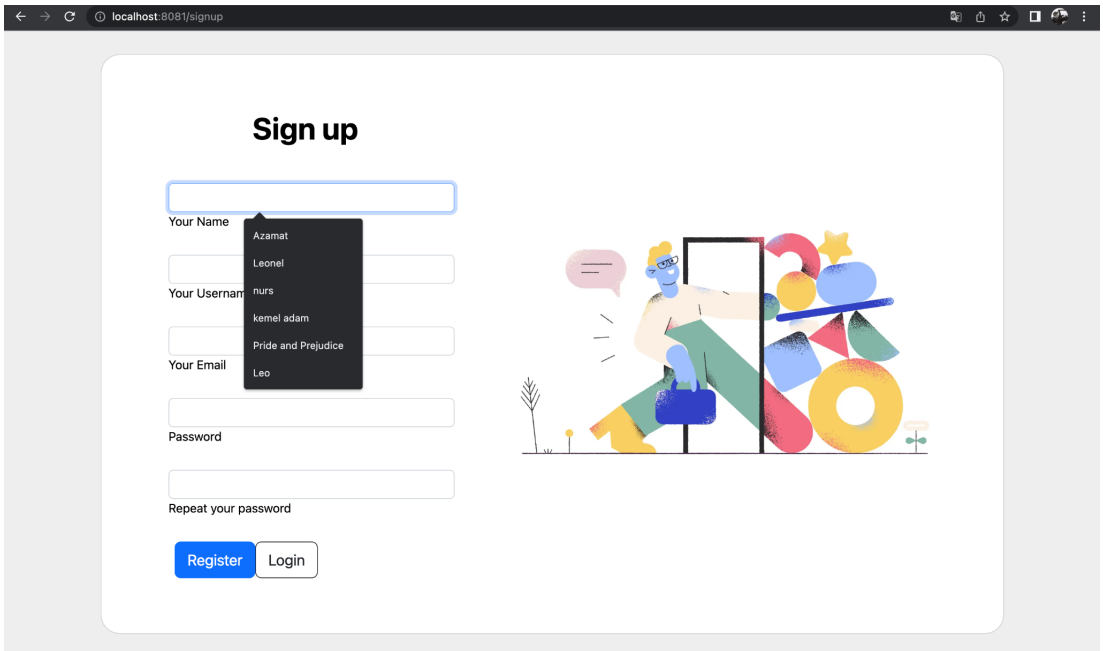
```go
package main

import (
  "Goland/controllers"
  _ "github.com/go-sql-driver/mysql"
  "html/template"
  "net/http"
)

var tpl *template.Template

func main() {
  tpl, _ = tpl.ParseGlob("templates/*.html")

  http.HandleFunc("/signup", controllers.Signup)
  http.HandleFunc("/products", controllers.ShowProducts)
  http.HandleFunc("/search", controllers.GetProduct)
  http.HandleFunc("/login", controllers.Login)
  http.HandleFunc("/logout", controllers.Logout)
  http.HandleFunc("/add_product", controllers.AddProduct)
  http.HandleFunc("/", controllers.Home)
  http.ListenAndServe(":8081", nil)
}
```

In this page we connect go project with database. For this we enter name of sql user, his password then database name which you want to connect.

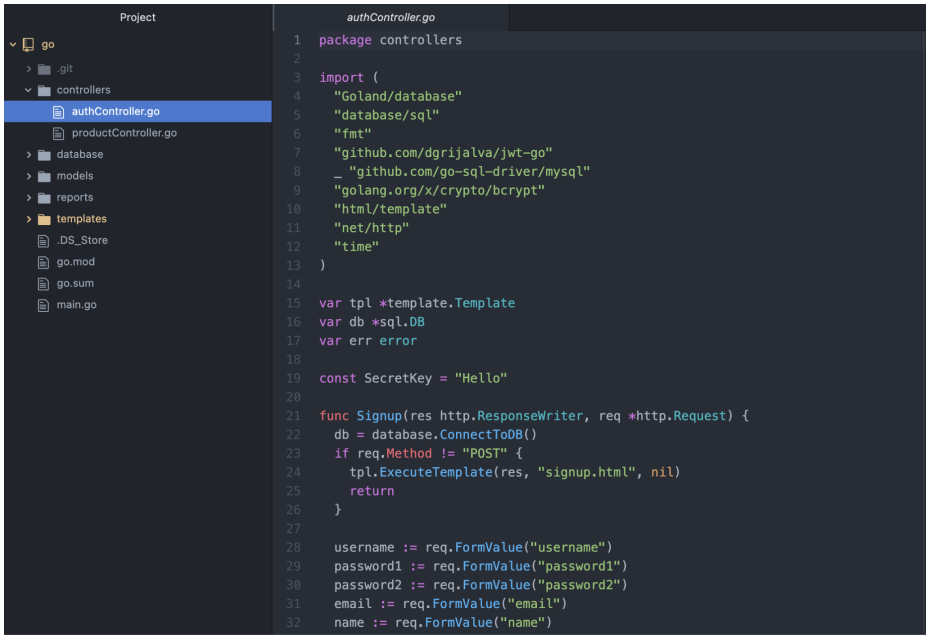


Connecting to database. "username:password@(127.0.0.1:8889)/databasename"

Registration form



With by this page we can add a new users
 Firstly you should have users table on your database with "username", "email", "name", "password" fields.



line 21 -> Function for registration
line 22 -> we connect to the database
line 23-26 -> if method GET we return html file

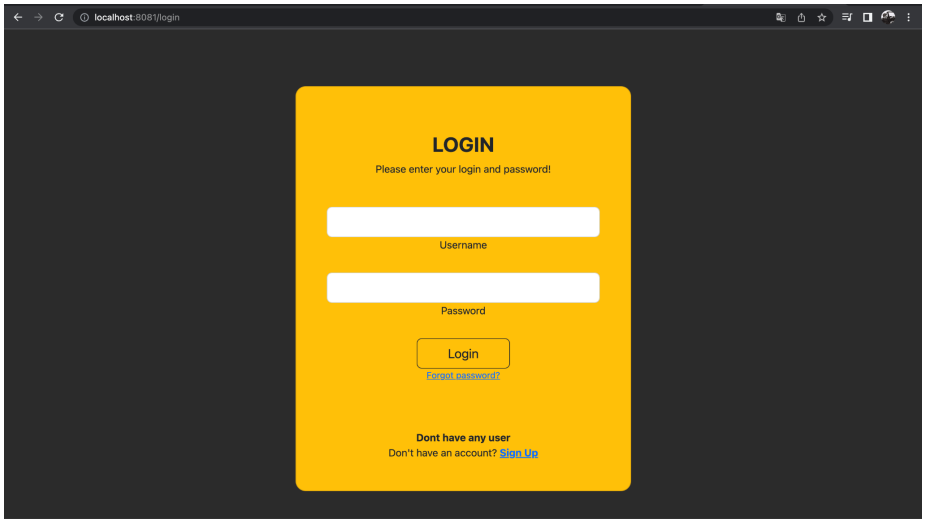
line 28-32 -> if method is POST then we should get new user data from post request and insert it to the database getting users data from post request by field name.

```
Project
go
  .git
  controllers
    authController.go
    productController.go
  database
  models
  reports
  templates
  .DS_Store
  go.mod
  go.sum
  main.go
```

```
authController.go
34  var user string
35
36  err := db.QueryRow("SELECT username FROM users WHERE username=?", username).Scan(&user)
37
38  if password1 == password2 {
39    switch {
40    case err == sql.ErrNoRows:
41      hashedPassword, err := bcrypt.GenerateFromPassword([]byte(password1), bcrypt.DefaultCost)
42      if err != nil {
43        http.Error(res, "Server error, unable to create your account.", 500)
44        return
45      }
46
47      _, err = db.Exec("INSERT INTO users(username, password, email, name) VALUES(?, ?, ?, ?)", username, hashedPassword, email,
48      if err != nil {
49        http.Error(res, "Server error, unable to create your account.", 500)
50        return
51      }
52
53      res.Write([]byte("User created!"))
54      return
55    case err != nil:
56      http.Error(res, "Server error, unable to create your account.", 500)
57      return
58    default:
59      http.Redirect(res, req, "/", 301)
60    }
61  } else {
62    http.Error(res, "Password doesn't match. Both passwords should be same!", 500)
63    return
64  }
65 }
```

line 36 -> getting data from database with giving username
line 38 -> comparing two passwords
line 40 -> if username doesn't have in database then we create a new user with this username
line 41-46 -> we hash the password
line 47-52 -> we add new data to the database
line 53 -> return message “User created!”
line 55-65 -> if user with this username exists, then we show error with this text.

Authorization form



```go
var Logerror string

func Login(res http.ResponseWriter, req *http.Request) {
  db = database.ConnectToDB()

  if req.Method != "POST" && Logerror != "" {
    tpl.ExecuteTemplate(res, "login.html", Logerror)
    return
  }

  username := req.FormValue("username")
  password := req.FormValue("password")

  var databaseUsername string
  var databasePassword string

  err := db.QueryRow("SELECT username, password FROM users WHERE username=?", username).Scan(&databaseUsername, &databasePassword

  if err != nil {
    http.Redirect(res, req, "/login", 301)
    Logerror = "Dont have any user"
    return
  }

  err = bcrypt.CompareHashAndPassword([]byte(databasePassword), []byte(password))
  if err != nil {
    http.Redirect(res, req, "/login", 301)
    Logerror = "Password is incorrect"
    return
  }

  claims := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.StandardClaims{
    Issuer:    databaseUsername,
    ExpiresAt: time.Now().Add(time.Minute * 60).Unix(), //1 day
  })
  token, err := claims.SignedString([]byte(SecretKey))
```

line 69 -> function for logging
line 70 -> connecting to the database
line 72 -> if request method id GET then we show login.html
line 77-81 -> if request method is POST then we are checking the user by password and username
line 83 -> getting user by username from database
line 85-89 -> id user doesn't exist we return the value "logger" with text below
line 91-96 -> if user exist, then we hash the password and compare with password from database
line 98-102 -> if user exist and password also correct we generate the new token and add it to http cookies.

```go
  newCookie := http.Cookie{
    Name:     "jar",
    Value:    token,
    Expires:  time.Now().Add(time.Hour * 24),
    HttpOnly: true,
    MaxAge:   99,
  }

  http.SetCookie(res, &newCookie)
  http.Redirect(res, req, "/", 301)
}

func Logout(res http.ResponseWriter, req *http.Request) {
  newCookie := http.Cookie{
    Name:     "jar",
    Value:    "",
    Expires:  time.Now().Add(-time.Hour),
    HttpOnly: true,
  }

  http.SetCookie(res, &newCookie)
  http.Redirect(res, req, "/", 301)
}
```

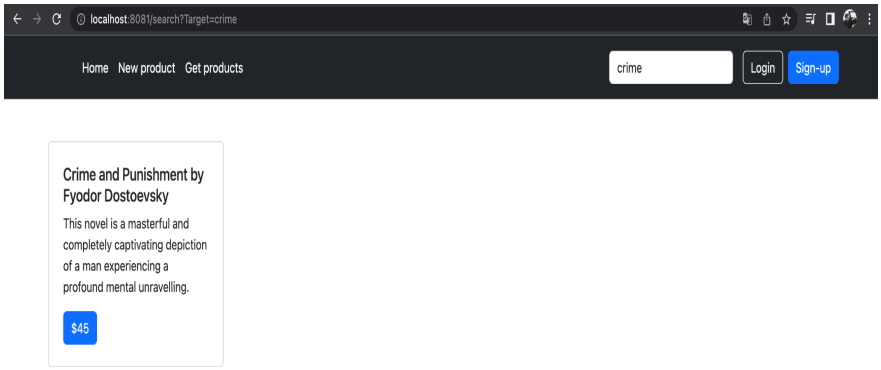
line 104 -> creating new cookie for this user

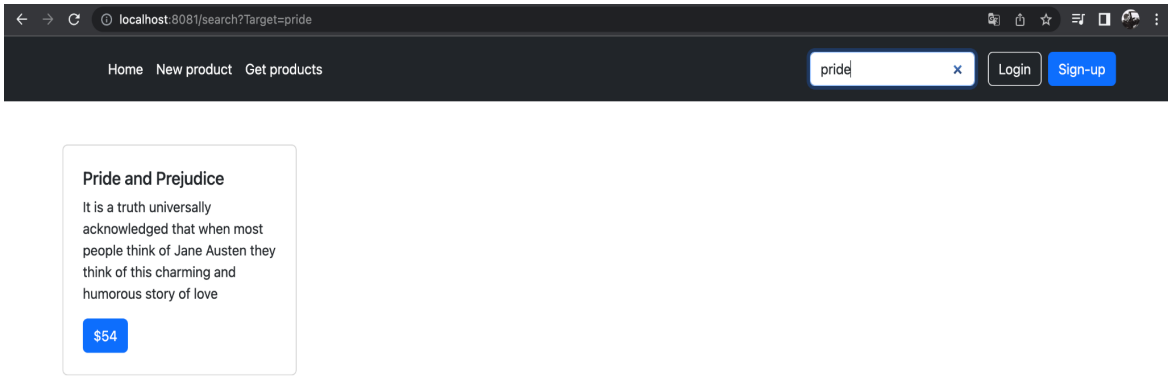
line 112 -> add the cookie for http
line 113 -> then redirect to main page
line 116 -> Logout for removing cookie for user
line 117 -> creating new cookie with expired date
line 124 -> add this cookie for changing old cookie

```
models
reports
templates
.DS_Store
go.mod
go.sum
main.go

127
128 func Home(res http.ResponseWriter, req *http.Request) {
129   tpl, _ = tpl.ParseGlob("templates/*.html")
130
131   username, _ := GetUser(req)
132   tpl.ExecuteTemplate(res, "index.html", username)
133
134 }
135
136 func GetUser(req *http.Request) (string, error) {
137   cookie, err := req.Cookie("jar")
138   if err != nil {
139     fmt.Println("Something was wrong")
140     return "", err
141   }
142
143   token, err := jwt.ParseWithClaims(cookie.Value, &jwt.StandardClaims{}, func(token *jwt.Token) (interface{}, error) {
144     return []byte(SecretKey), nil
145   })
146
147   if err != nil {
148     fmt.Println("Something was wrong")
149     return "", err
150   }
151
152   claims := token.Claims.(*jwt.StandardClaims)
153   fmt.Println(claims.Issuer)
154
155   return claims.Issuer, nil
156 }
```

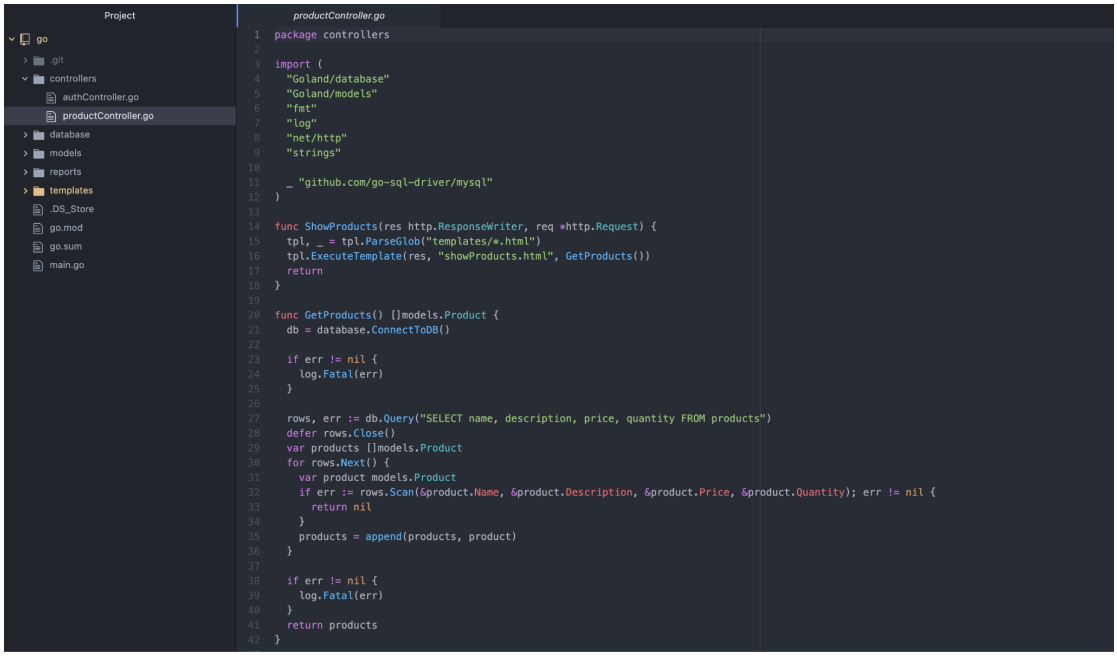
line 128 -> homepage gets user and put it home.html file
line 136 -> function which return user if its has or empty value
line 137 -> we get the cookie from http for checking authentication of user
line 143 -> if http has token then we take user name and return it for home function

Searching items based on name

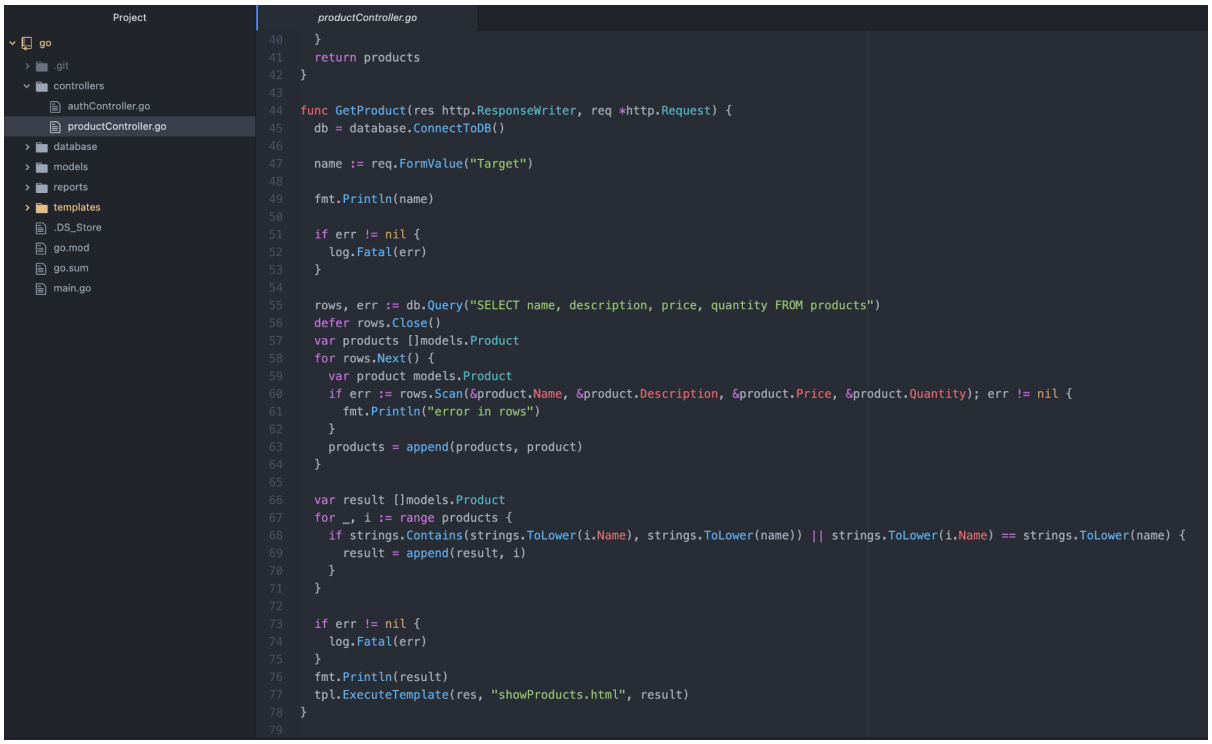




you need to create a table products in database

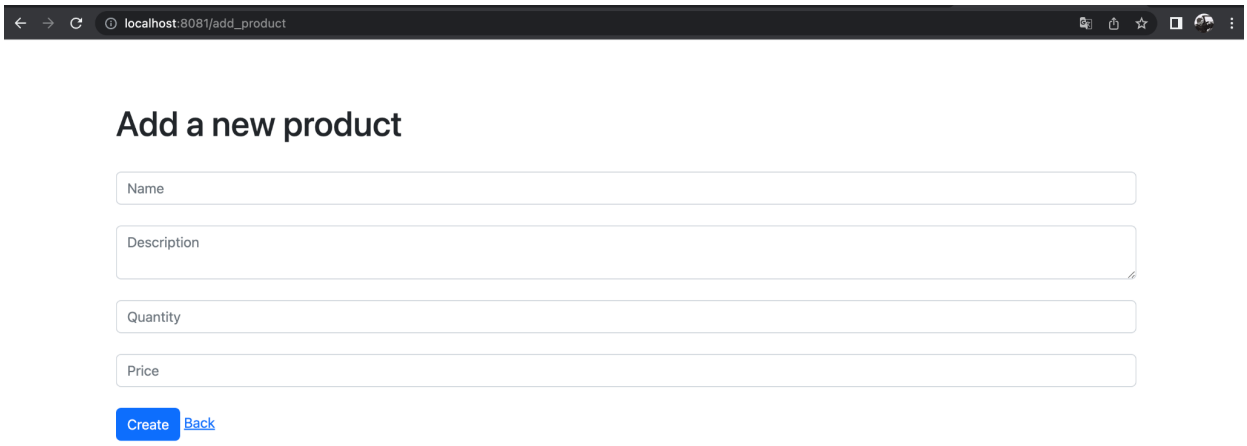


line 14 -> we get all products from database with by GetProductsfunction
line 20 -> getting all products from database
line 21 -> connecting to database
line 27 -> selecting all products
line 29 -> declaration of slice
line 35 -> adding each products for products slice
line 41 -> then return products for showProducts function



line 44 -> getting all product with by search input from header
line 45 -> connecting to database
line 47 -> getting target value from request post with by name Target
line 55 -> selecting all products to slice
line 67-68-> comparing product name with target value from search
line 69 -> if products exist with target name, then we add it to result slice
line 77-> then return all products with target value name

Add Product function




```
80  func AddProduct(res http.ResponseWriter, req *http.Request) {
81    db = database.ConnectToDB()
82
83    if req.Method != "POST" {
84      http.ServeFile(res, req, "templates/add_product.html")
85      return
86    }
87
88    description := req.FormValue("description")
89    price := req.FormValue("price")
90    quantity := req.FormValue("quantity")
91    name := req.FormValue("name")
92
93    _, err = db.Exec("INSERT INTO products(name, description, price, quantity) VALUES(?, ?, ?, ?)", name, description, price, quantity)
94    if err != nil {
95      http.Error(res, "Server error, unable to create your account.", 500)
96      return
97    }
98    http.Redirect(res, req, "/show_products", 301)
99  }
100
```

line 80-> adding a new product from website to database
line 81-> connecting to database
line 88-> getting data from form
line 93-> inserting a new data to database
line 98-> redirecting the page