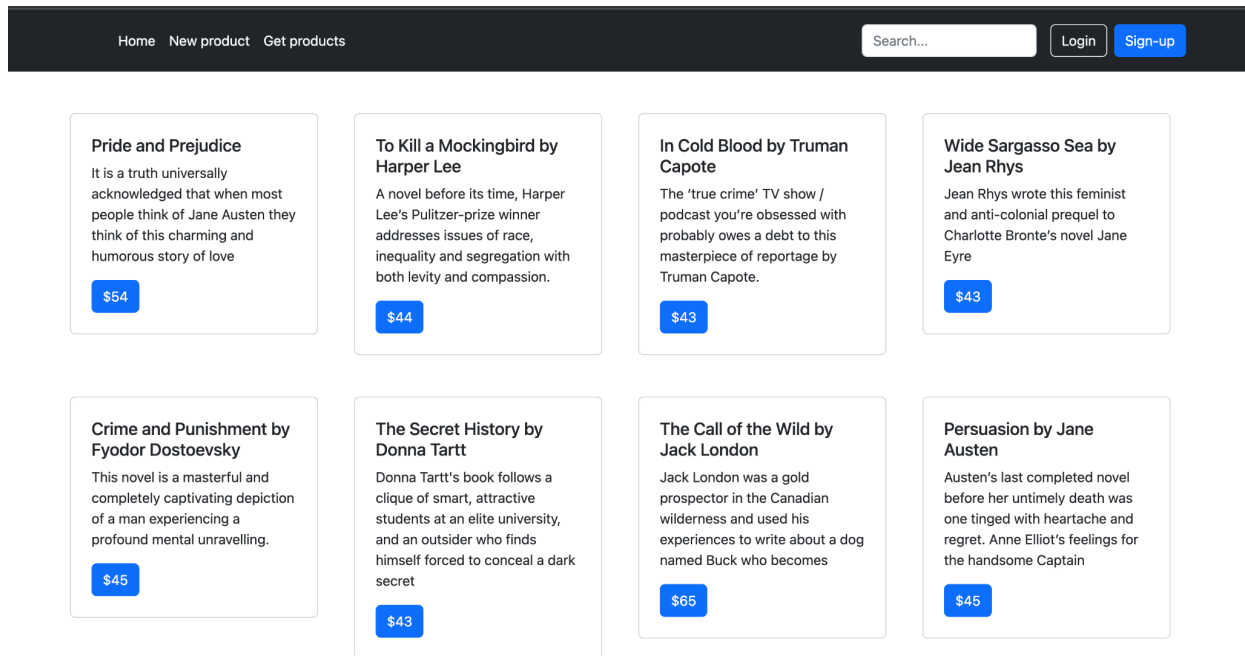


1 Introduction to the Service

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.

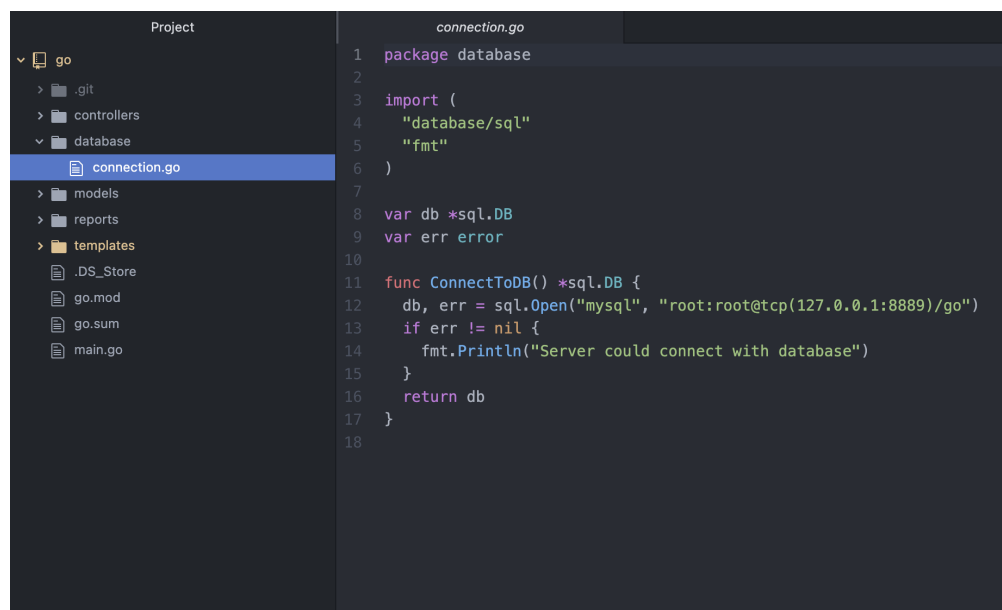
Then open the terminal from the project folder and type "go run main.go". Packages will be downloaded automatically, if it doesn't you need to download it by hand. All package dependencies you can find in the go.mod file. Ex: `go get github.com/go-sql-driver/mysql`. Then again run the "go run main.go" 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

```
1 package main
2
3 import (
4     "Goland/controllers"
5     _ "github.com/go-sql-driver/mysql"
6     "html/template"
7     "net/http"
8 )
9
10 var tpl *template.Template
11
12 func main() {
13     tpl, _ = tpl.ParseGlob("templates/*.html")
14
15     http.HandleFunc("/signup", controllers.Signup)
16     http.HandleFunc("/products", controllers.ShowProducts)
17     http.HandleFunc("/search", controllers.GetProduct)
18     http.HandleFunc("/login", controllers.Login)
19     http.HandleFunc("/logout", controllers.Logout)
20     http.HandleFunc("/add_product", controllers.AddProduct)
21     http.HandleFunc("/", controllers.Home)
22     http.ListenAndServe(":8081", nil)
23 }
24
```

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.



The screenshot shows an IDE with a project structure on the left and a Go file named `connection.go` on the right. The project structure includes a `go` directory with subdirectories `.git`, `controllers`, `database`, `models`, `reports`, and `templates`. The `database` directory is expanded, showing `connection.go`, `.DS_Store`, `go.mod`, `go.sum`, and `main.go`. The `connection.go` file contains the following code:

```
1 package database
2
3 import (
4     "database/sql"
5     "fmt"
6 )
7
8 var db *sql.DB
9 var err error
10
11 func ConnectToDB() *sql.DB {
12     db, err = sql.Open("mysql", "root:root@tcp(127.0.0.1:8889)/go")
13     if err != nil {
14         fmt.Println("Server could connect with database")
15     }
16     return db
17 }
18
```

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

Registration form

Sign up

Your Name

Your Username

Your Email

Password

Repeat your password

Register Login

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.

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

1 package controllers
2
3 import (
4     "Goland/database"
5     "database/sql"
6     "fmt"
7     "github.com/dgrijalva/jwt-go"
8     _ "github.com/go-sql-driver/mysql"
9     "golang.org/x/crypto/bcrypt"
10    "html/template"
11    "net/http"
12    "time"
13 )
14
15 var tpl *template.Template
16 var db *sql.DB
17 var err error
18
19 const SecretKey = "Hello"
20
21 func Signup(res http.ResponseWriter, req *http.Request) {
22     db = database.ConnectToDB()
23     if req.Method != "POST" {
24         tpl.ExecuteTemplate(res, "signup.html", nil)
25         return
26     }
27
28     username := req.FormValue("username")
29     password1 := req.FormValue("password1")
30     password2 := req.FormValue("password2")
31     email := req.FormValue("email")
32     name := req.FormValue("name")
```

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.

```

23  authController.go
24
25  var user string
26
27  err := db.QueryRow("SELECT username FROM users WHERE username=?", username).Scan(&user)
28
29  if password1 == password2 {
30      switch {
31      case err == sql.ErrNoRows:
32          hashedPassword, err := bcrypt.GenerateFromPassword([]byte(password1), bcrypt.DefaultCost)
33          if err != nil {
34              http.Error(res, "Server error, unable to create your account.", 500)
35              return
36          }
37          _, err = db.Exec("INSERT INTO users(username, password, email, name) VALUES(?, ?, ?, ?)", username, hashedPassword, email, name)
38          if err != nil {
39              http.Error(res, "Server error, unable to create your account.", 500)
40              return
41          }
42          res.Write([]byte("User created!"))
43          return
44      case err != nil:
45          http.Error(res, "Server error, unable to create your account.", 500)
46          return
47      default:
48          http.Redirect(res, req, "/", 301)
49      }
50  } else {
51      http.Error(res, "Password doesn't match. Both passwords should be same!", 500)
52      return
53  }
54  }
55  }

```

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

← → ↻ localhost:8081/login

LOGIN

Please enter your login and password!

Username

Password

Login

[Forgot password?](#)

Don't have an user

Don't have an account? [Sign Up](#)

```
66
67 var Logerror string
68
69 func Login(res http.ResponseWriter, req *http.Request) {
70     db = database.ConnectToDB()
71
72     if req.Method != "POST" && Logerror != "" {
73         tpl.ExecuteTemplate(res, "login.html", Logerror)
74         return
75     }
76
77     username := req.FormValue("username")
78     password := req.FormValue("password")
79
80     var databaseUsername string
81     var databasePassword string
82
83     err := db.QueryRow("SELECT username, password FROM users WHERE username=?", username).Scan(&databaseUsername, &databasePassword)
84
85     if err != nil {
86         http.Redirect(res, req, "/login", 301)
87         Logerror = "Dont have any user"
88         return
89     }
90
91     err = bcrypt.CompareHashAndPassword([]byte(databasePassword), []byte(password))
92     if err != nil {
93         http.Redirect(res, req, "/login", 301)
94         Logerror = "Password is incorrect"
95         return
96     }
97
98     claims := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.StandardClaims{
99         Issuer:    databaseUsername,
100         ExpiresAt: time.Now().Add(time.Minute * 60).Unix(), //1 day
101     })
102     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.

```
103
104 newCookie := http.Cookie{
105     Name:    "jar",
106     Value:   token,
107     Expires: time.Now().Add(time.Hour * 24),
108     HttpOnly: true,
109     MaxAge:  99,
110 }
111
112 http.SetCookie(res, &newCookie)
113 http.Redirect(res, req, "/", 301)
114 }
115
116 func Logout(res http.ResponseWriter, req *http.Request) {
117     newCookie := http.Cookie{
118         Name:    "jar",
119         Value:   "",
120         Expires: time.Now().Add(-time.Hour),
121         HttpOnly: true,
122     }
123
124     http.SetCookie(res, &newCookie)
125     http.Redirect(res, req, "/", 301)
126 }
127
```

line 104 -> creating new cookie for this user

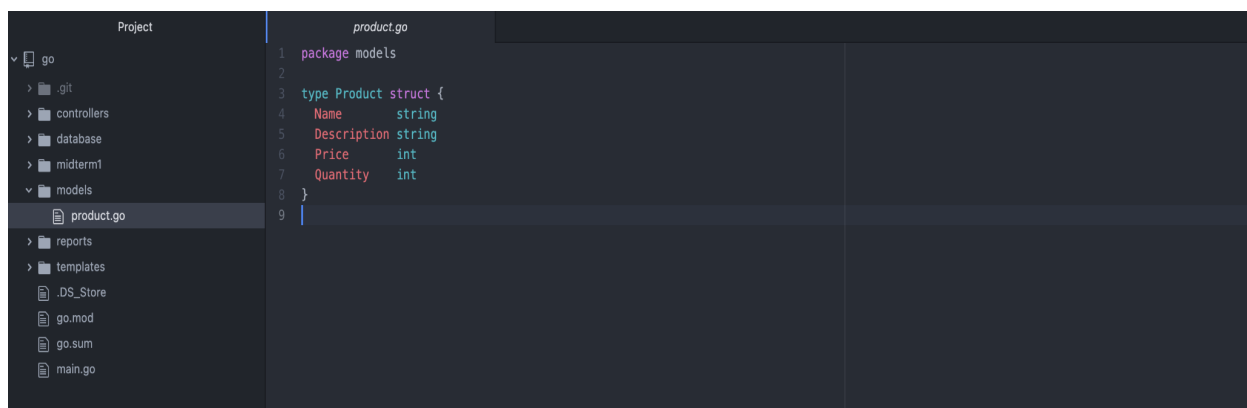
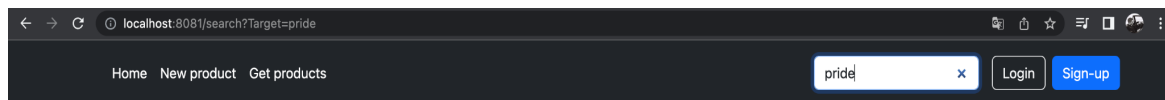
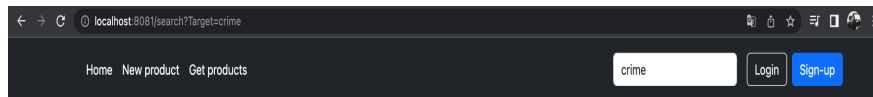
line 112 -> add the cookie for http
line 113 -> then redirect to main page
line 116 -> Logout for removing cookie for user
line 117 -> creating new cookie with expired date
line 124 -> add this cookie for changing old cookie



```
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

```
1 package controllers
2
3 import (
4     "Goland/database"
5     "Goland/models"
6     "fmt"
7     "log"
8     "net/http"
9     "strings"
10
11     _ "github.com/go-sql-driver/mysql"
12 )
13
14 func ShowProducts(res http.ResponseWriter, req *http.Request) {
15     tpl, _ = tpl.ParseGlob("templates/*.html")
16     tpl.ExecuteTemplate(res, "showProducts.html", GetProducts())
17     return
18 }
19
20 func GetProducts() []models.Product {
21     db = database.ConnectToDB()
22
23     if err != nil {
24         log.Fatal(err)
25     }
26
27     rows, err := db.Query("SELECT name, description, price, quantity FROM products")
28     defer rows.Close()
29     var products []models.Product
30     for rows.Next() {
31         var product models.Product
32         if err := rows.Scan(&product.Name, &product.Description, &product.Price, &product.Quantity); err != nil {
33             return nil
34         }
35         products = append(products, product)
36     }
37
38     if err != nil {
39         log.Fatal(err)
40     }
41     return products
42 }
```

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

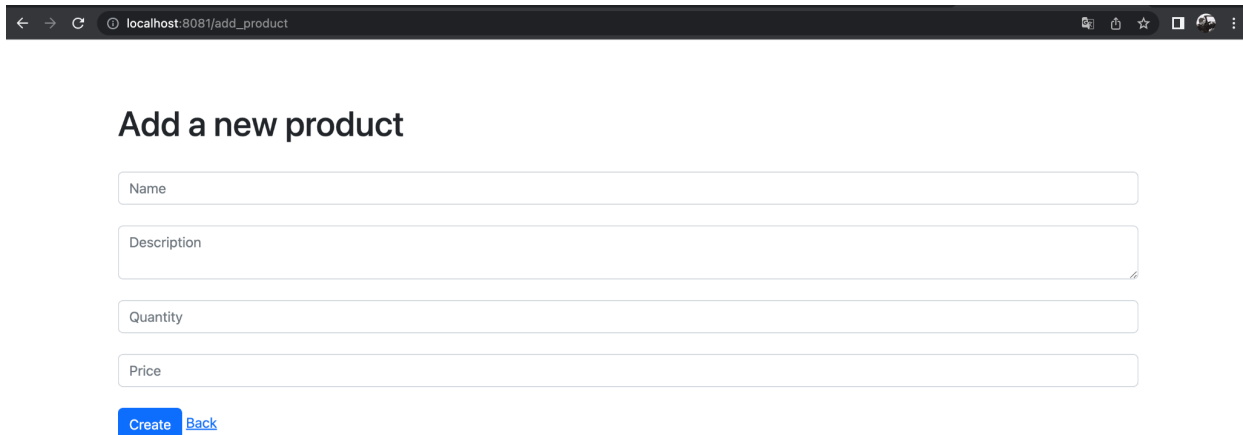
line 35 -> adding each products for products slice

line 41 -> then return products for showProducts function

```
40 }
41 return products
42 }
43
44 func GetProduct(res http.ResponseWriter, req *http.Request) {
45     db = database.ConnectToDB()
46
47     name := req.FormValue("Target")
48
49     fmt.Println(name)
50
51     if err != nil {
52         log.Fatal(err)
53     }
54
55     rows, err := db.Query("SELECT name, description, price, quantity FROM products")
56     defer rows.Close()
57     var products []models.Product
58     for rows.Next() {
59         var product models.Product
60         if err := rows.Scan(&product.Name, &product.Description, &product.Price, &product.Quantity); err != nil {
61             fmt.Println("error in rows")
62         }
63         products = append(products, product)
64     }
65
66     var result []models.Product
67     for _, i := range products {
68         if strings.Contains(strings.ToLower(i.Name), strings.ToLower(name)) || strings.ToLower(i.Name) == strings.ToLower(name) {
69             result = append(result, i)
70         }
71     }
72
73     if err != nil {
74         log.Fatal(err)
75     }
76     fmt.Println(result)
77     tpl.ExecuteTemplate(res, "showProducts.html", result)
78 }
79 }
```


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



The screenshot shows a web browser window with the address bar displaying 'localhost:8081/add_product'. The page title is 'Add a new product'. The form contains four input fields: 'Name', 'Description', 'Quantity', and 'Price'. Below the fields are two buttons: 'Create' and 'Back'.

```
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