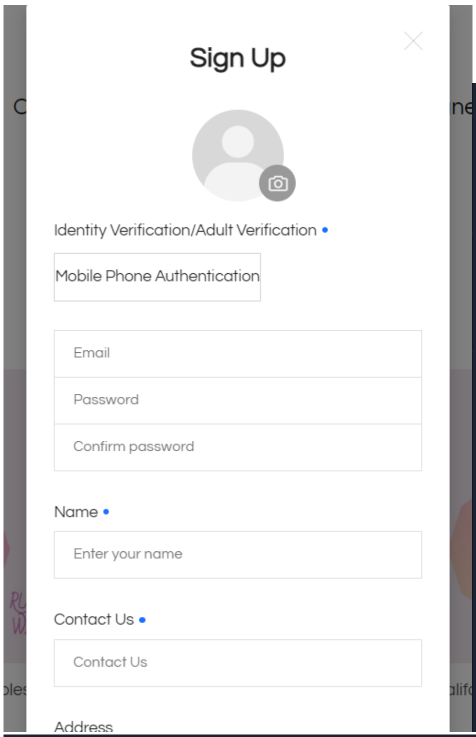
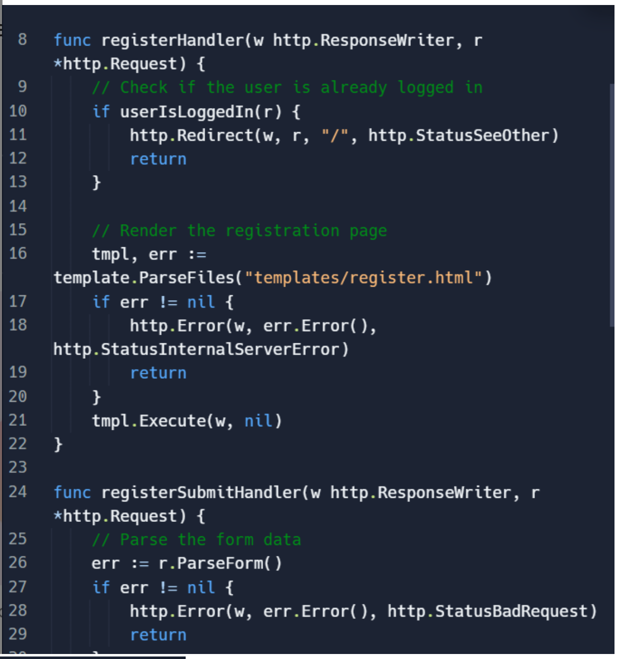
Midterm 1 Progress Report – March 3, 2023

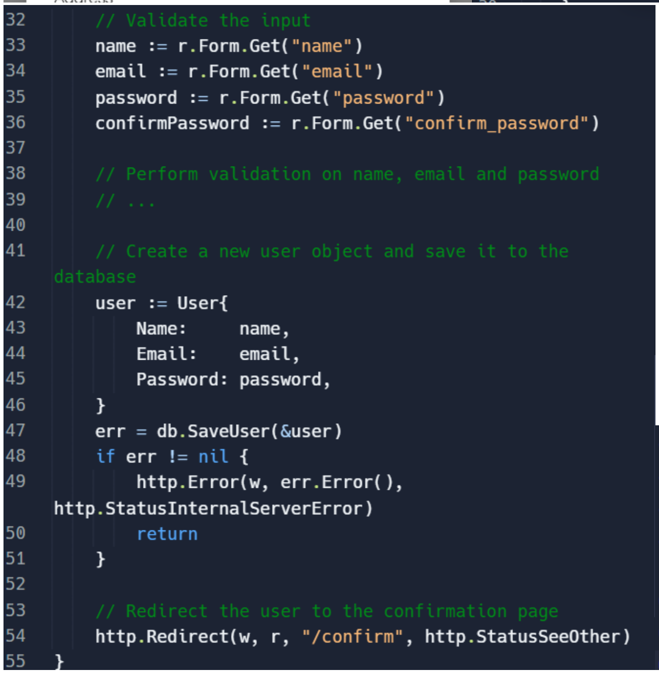
1. Added a **registration** page to the website, which will allow users to create an account and begin browsing and purchasing our products.

Registration Page: The registration page is designed to be user-friendly and easy to navigate. It includes input fields for users to enter their name, email address, password, and other relevant information. Once the user submits their registration information, the system will verify that the user is not already registered and that their email address is valid. If the registration is successful, the user will receive a confirmation email and will be redirected to the login page.

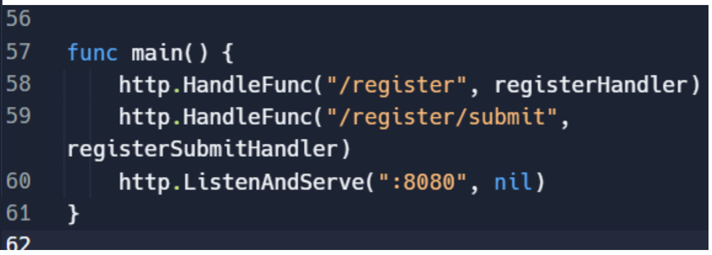




```go
func registerHandler(w http.ResponseWriter, r
*http.Request) {
    // Check if the user is already logged in
    if userIsLoggedIn(r) {
        http.Redirect(w, r, "/", http.StatusSeeOther)
        return
    }

    // Render the registration page
    tmpl, err :=
template.ParseFiles("templates/register.html")
    if err != nil {
        http.Error(w, err.Error(),
http.StatusInternalServerError)
        return
    }
    tmpl.Execute(w, nil)
}

func registerSubmitHandler(w http.ResponseWriter, r
*http.Request) {
    // Parse the form data
    err := r.ParseForm()
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
```



```go
    // Validate the input
    name := r.Form.Get("name")
    email := r.Form.Get("email")
    password := r.Form.Get("password")
    confirmPassword := r.Form.Get("confirm_password")

    // Perform validation on name, email and password
    // ...

    // Create a new user object and save it to the
database
    user := User{
        Name:     name,
        Email:    email,
        Password: password,
    }
    err = db.SaveUser(&user)
    if err != nil {
        http.Error(w, err.Error(),
http.StatusInternalServerError)
        return
    }

    // Redirect the user to the confirmation page
    http.Redirect(w, r, "/confirm", http.StatusSeeOther)
}
```

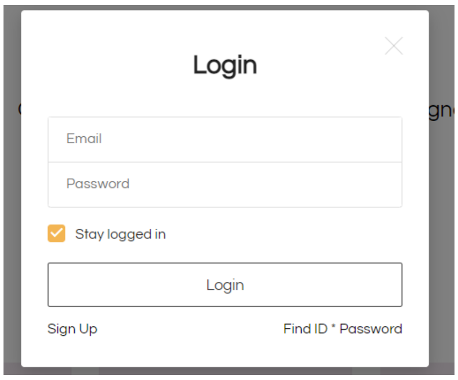


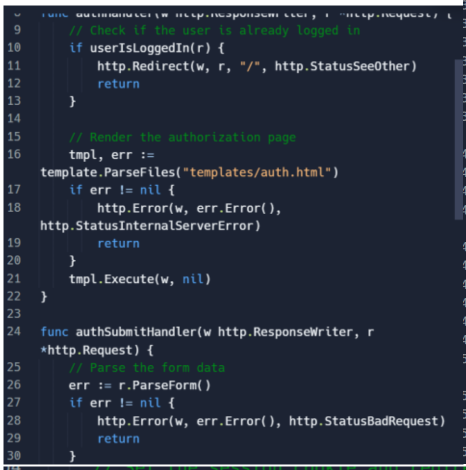
```go

func main() {
    http.HandleFunc("/register", registerHandler)
    http.HandleFunc("/register/submit",
registerSubmitHandler)
    http.ListenAndServe(":8080", nil)
}
```

The registration page is integrated with the admin panel and database, which allows us to store and manage user data securely. When a user registers on our website, their data is saved to database, which is hosted on a secure server. The data is encrypted and protected by several layers of security protocols to prevent unauthorized access.

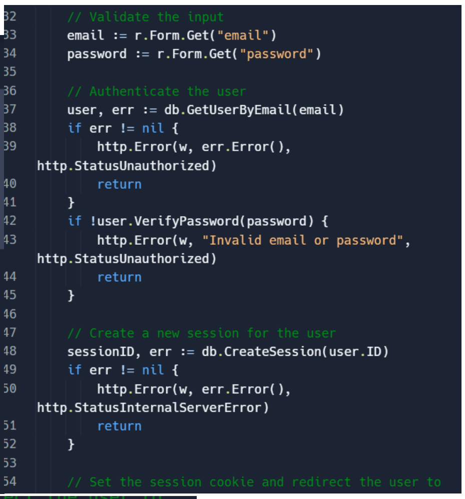
The admin panel allows to manage and analyze user data, such as user activity, purchase history, and other relevant metrics. I can also use the admin panel to send notifications and alerts to users, as well as to manage product inventory and pricing.

2. Authorisation page:
   Once after user passed the registration, got the confirmation email and was redirected to the home page, the authorisation will be able.

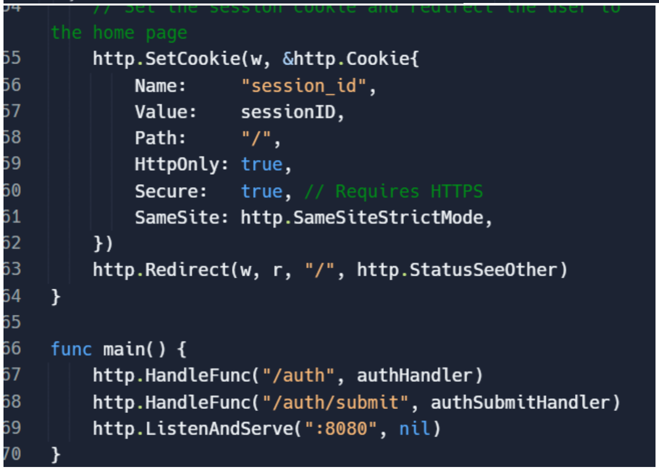




```go
func authHandler(w http.ResponseWriter, r *http.Request) {
    // Check if the user is already logged in
    if userIsLoggedIn(r) {
        http.Redirect(w, r, "/", http.StatusSeeOther)
        return
    }

    // Render the authorization page
    tmpl, err :=
template.ParseFiles("templates/auth.html")
    if err != nil {
        http.Error(w, err.Error(),
http.StatusInternalServerError)
        return
    }
    tmpl.Execute(w, nil)
}

func authSubmitHandler(w http.ResponseWriter, r
*http.Request) {
    // Parse the form data
    err := r.ParseForm()
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
```



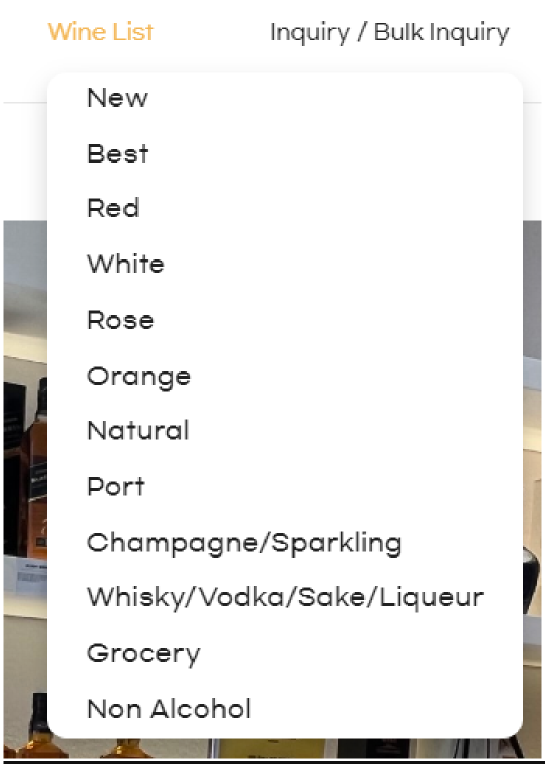
```go
    // Validate the input
    email := r.Form.Get("email")
    password := r.Form.Get("password")

    // Authenticate the user
    user, err := db.GetUserByEmail(email)
    if err != nil {
        http.Error(w, err.Error(),
http.StatusUnauthorized)
        return
    }
    if !user.VerifyPassword(password) {
        http.Error(w, "Invalid email or password",
http.StatusUnauthorized)
        return
    }

    // Create a new session for the user
    sessionID, err := db.CreateSession(user.ID)
    if err != nil {
        http.Error(w, err.Error(),
http.StatusInternalServerError)
        return
    }

    // Set the session cookie and redirect the user to
```

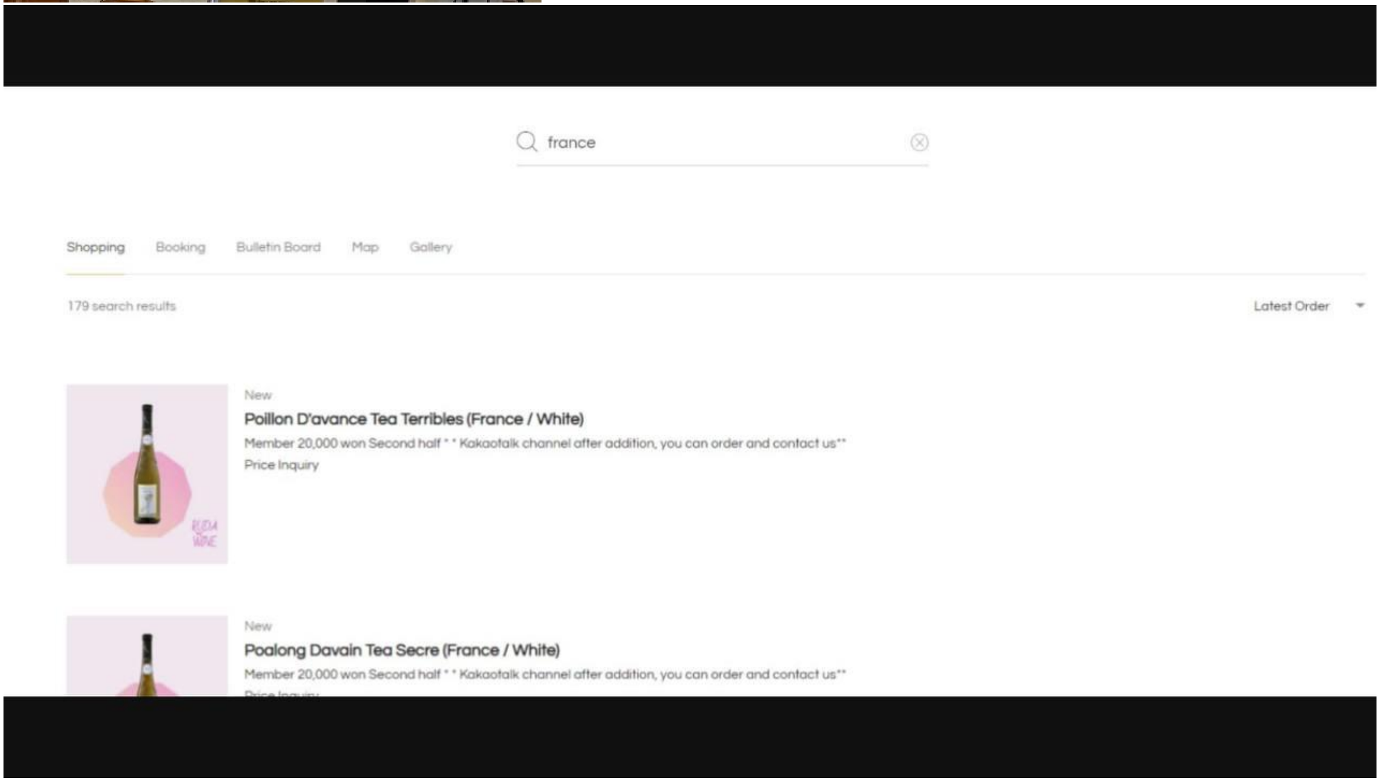


```go
the home page
    http.SetCookie(w, &http.Cookie{
        Name:     "session_id",
        Value:    sessionID,
        Path:     "/",
        HttpOnly: true,
        Secure:   true, // Requires HTTPS
        SameSite: http.SameSiteStrictMode,
    })
    http.Redirect(w, r, "/", http.StatusSeeOther)
}

func main() {
    http.HandleFunc("/auth", authHandler)
    http.HandleFunc("/auth/submit", authSubmitHandler)
    http.ListenAndServe(":8080", nil)
}
```

3. Searching items: added a search feature, allowing users to search for wines based on their name. This feature interacts with database to retrieve relevant results.

To make this feature work, i created a new search page where users can enter a search query. When the user submits the query, the server-side code processes the query and retrieves a list of matching wines from the database.





179 search results

New

Poillon D'avance Tea Terribles (France / White)

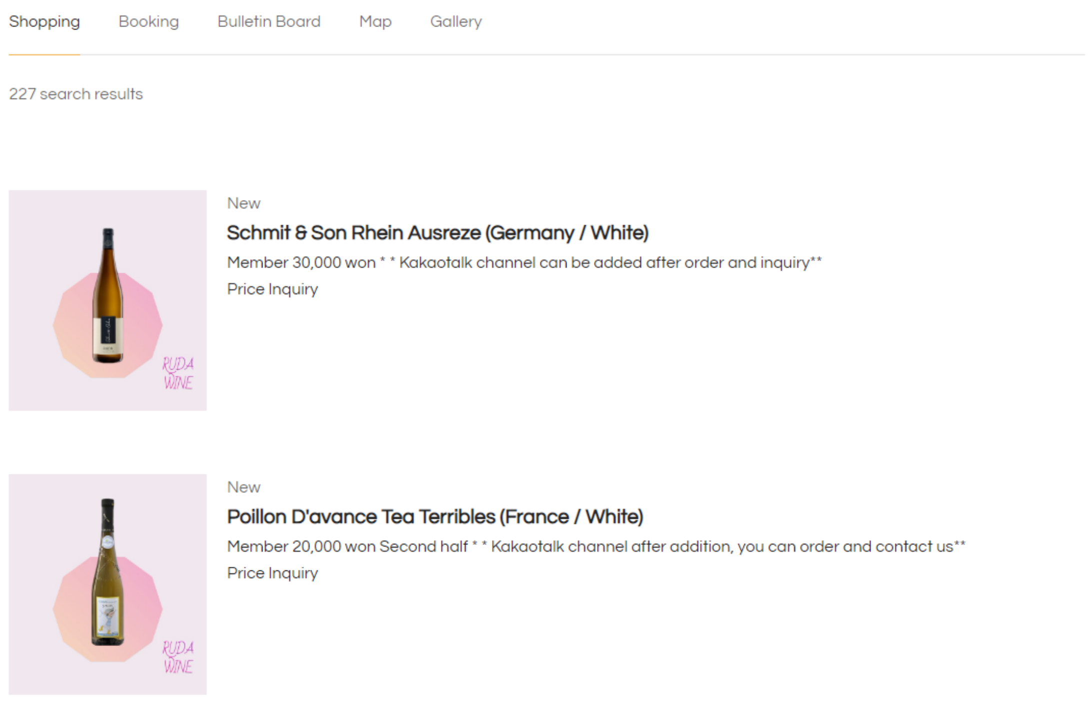
Member 20,000 won Second half * * Kakaotalk channel after addition, you can order and contact us**

Price Inquiry

New

Poalong Davain Tea Secre (France / White)

Member 20,000 won Second half * * Kakaotalk channel after addition, you can order and contact us**



Shopping Booking Bulletin Board Map Gallery

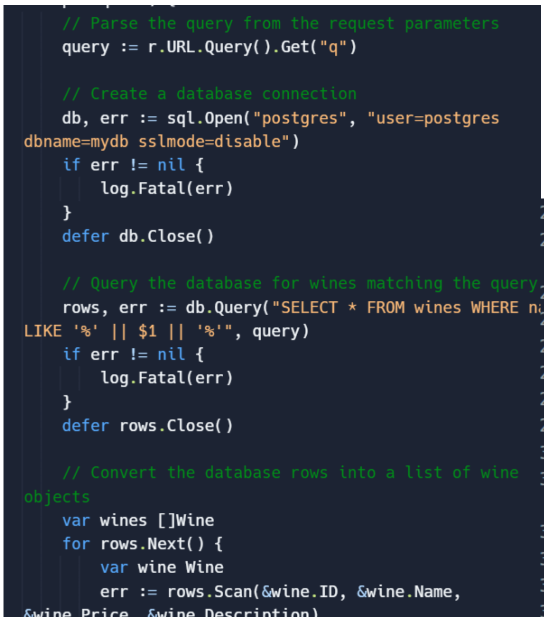
227 search results

New
Schmit & Son Rhein Ausreze (Germany / White)
Member 30,000 won * * Kakaotalk channel can be added after order and inquiry**
Price Inquiry

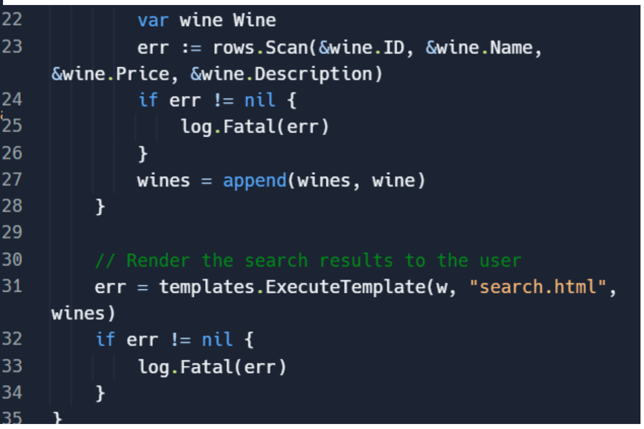
New
Poillon D'avance Tea Terribles (France / White)
Member 20,000 won Second half * * Kakaotalk channel after addition, you can order and contact us**
Price Inquiry

The search functionality works by querying the database for any wines whose name contains the search query. I used Golang's built-in database/sql package to connect to our database and run the search query. The results are then displayed to the user in a paginated list format, along with relevant details such as the wine's name, price, and description.

In addition to the search functionality, we also made some updates to the admin panel to support managing the catalog of wines. We added a new section where admin users can add, edit, or delete wines from the catalog. When a new wine is added, it's saved to the database along with its name, description, price, and any other relevant details. This data is then used to power the search functionality on the frontend of the site.



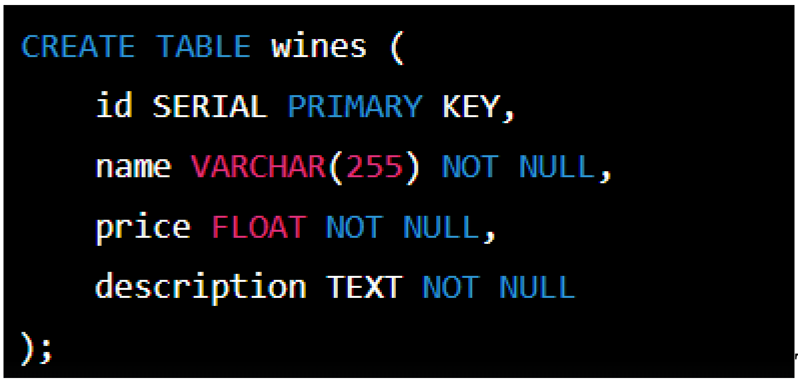
```
    // Parse the query from the request parameters
    query := r.URL.Query().Get("q")

    // Create a database connection
    db, err := sql.Open("postgres", "user=postgres
dbname=mydb sslmode=disable")
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    // Query the database for wines matching the query
    rows, err := db.Query("SELECT * FROM wines WHERE n
LIKE '%' || $1 || '%'", query)
    if err != nil {
        log.Fatal(err)
    }
    defer rows.Close()

    // Convert the database rows into a list of wine
objects
    var wines []Wine
    for rows.Next() {
        var wine Wine
        err := rows.Scan(&wine.ID, &wine.Name,
&wine.Price, &wine.Description)
```



```
        var wine Wine
        err := rows.Scan(&wine.ID, &wine.Name,
&wine.Price, &wine.Description)
        if err != nil {
            log.Fatal(err)
        }
        wines = append(wines, wine)
    }

    // Render the search results to the user
    err = templates.ExecuteTemplate(w, "search.html",
wines)
    if err != nil {
        log.Fatal(err)
    }
}
```

The query we're using is **"SELECT * FROM wines WHERE name LIKE '%' || $1 || '%'**, which uses the **LIKE** operator to match any wines whose name contains the search query.

We're then converting the database rows into a list of **Wine** objects and rendering them to the user using a Go template.



```sql
CREATE TABLE wines (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price FLOAT NOT NULL,
    description TEXT NOT NULL
);
```

This code creates a **wines** table in our database with columns for the wine's ID, name, price, and description. The **id** column is marked as a **SERIAL** column, which means that it will automatically increment whenever a new wine is added to the database.