Module Code	CT108-3-2-IOS
Intake Code	APD2F2211IT(MBT)
Lecturer Name	Mr. Amad Arshad
Hand-out Date	16 December 2022
Hand-in Date	24 March 2023

Student ID	**Student Name**
TP059963	Yip Zi Xian

Table of Contents

1.0 Introduction

1.1 Scenario

Based on brief research through Malaysian App Store, there isn't a single application that can provide a full-fledge user experience as a wine retailer with delivery orders. Although there are some applications that provide wine delivery services to customers, but the user interface is badly designed. Another issue is the feedback provided by customers itself varies because everyone has a different taste and opinion about wines.

To tackle the problems mentioned above, a full-fledge wine retailer application should have:

- Login and Registration, Guest Mode
- Wine Recommendations, Search and Filter function to ease customers' experience.
- Customisable profile and purchase history
- Cart, Payment, Delivery Service
- Wine Rating and Description from wine connoisseur and oenologists instead of customers
- User-friendly and elegant user interface

1.2 Assumptions

However, since this is just a prototype application and only can be run on a simulator, there are several assumptions needed to address:

- The payment transaction will be automatically deducted from customers' debit or credit card.
- Orders are topped at one carton per unique wine or 12 bottles per unique wine, but customers can order different wine at the same time.
- Orders are distributed via sea or air shipping and local delivery services and are received by customers based on their address.

2.0 Draft Diagrams

Before working on the SwiftUI application, two diagrams are drawn in order to help to visualize how and where the data will be transferred between multiple users. Use case diagram is drawn with four different users, two internal users and two external users. Use case diagram can benefit software developers to program the application from the user's perspective and satisfy the business processes (Cacoo, 2021).

Another diagram is entity relationship diagram. According to Sullivan, M. (2022), an entity relationship diagram allows software developers, project managers, mostly everyone to have an insight into the basic level of the database with simplified visualization.

2.1 Use Case Diagram

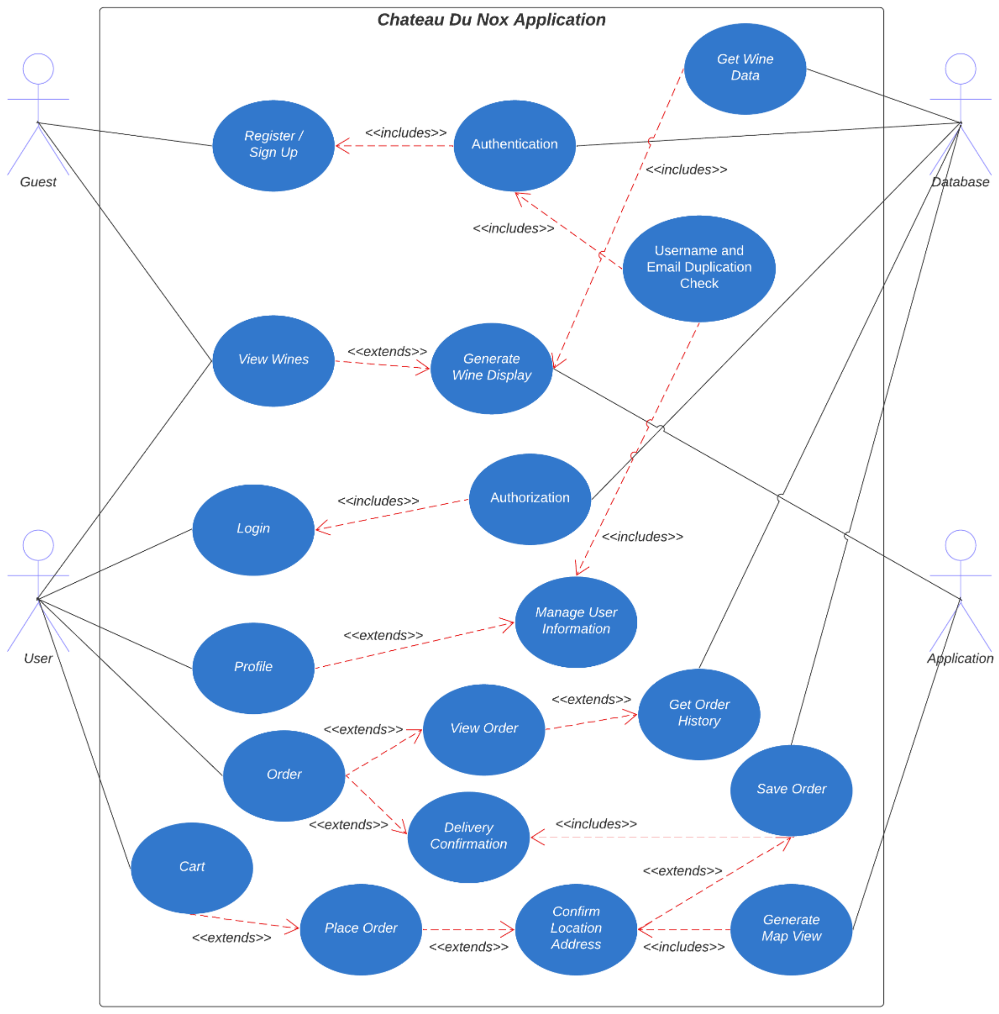


Diagram Key

● Use case

● Relationship

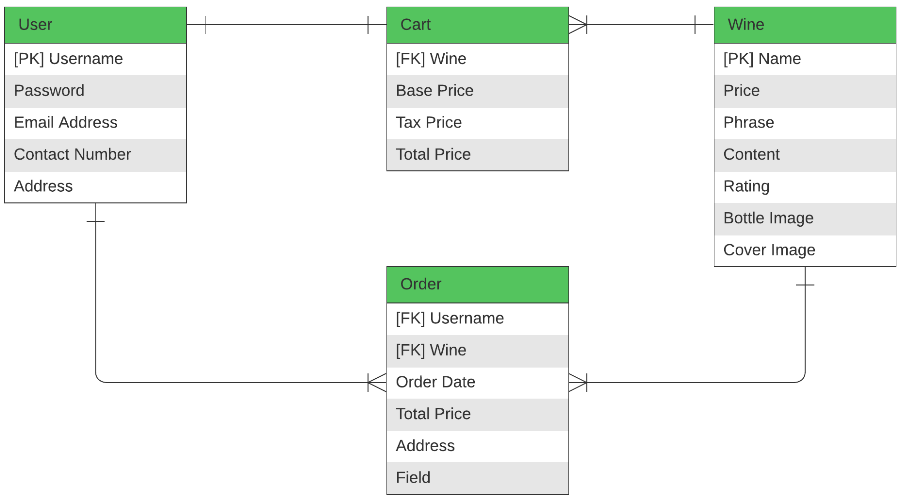
● Actor

Chateau Du Nox application will mainly consist of two external users, which are the unregistered user or guest and registered user or customer, and two internal users, which are the application itself and the database storing all data and information relevant to the application.

Guest will have only two basic functions, registration or browsing wines. When the guest first launch the application, the database will fetch the information about the wines and the application will generate a display for each wine for the guest to explore further. If the guest wishes to register or sign up an account with Chateau Du Nox, the database will automatically check for duplicated username and email within the existing customer information given that the guest has inputted all required fields. If there are no duplications in the database, then a new account will be created and guest can now login with their username and password.

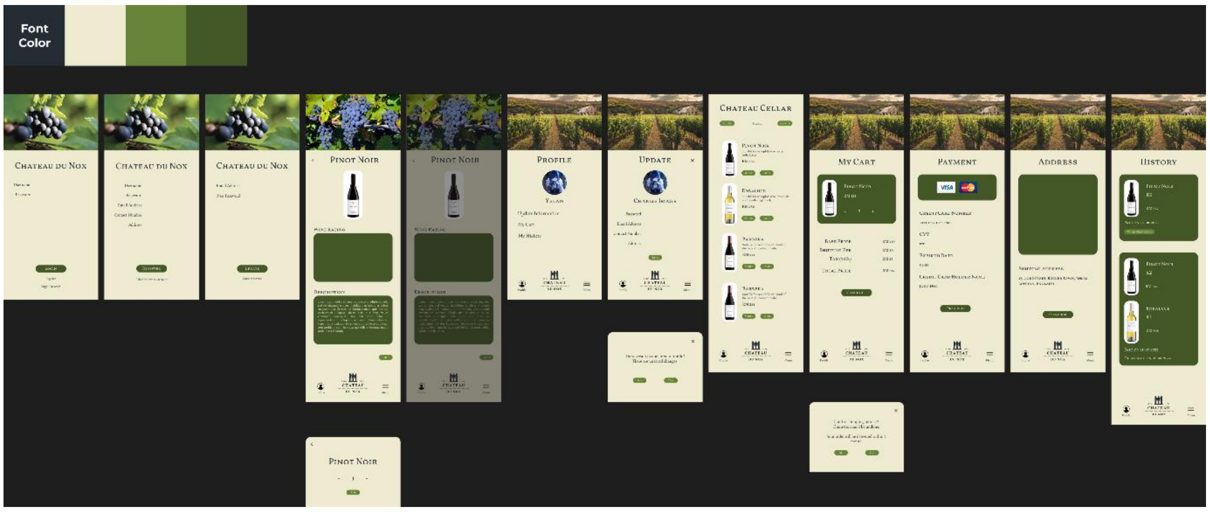
As for registered users, they can directly login and access their account or if they forget their password, they can reset their password using their email address. They can contact the customer service if they can't recall their email address too. After accessing their account, they can browse for wines, view their carts, and check their pending orders and history. Registered customers can edit their personal information.

2.2 Entity Relationship Diagram



The database of Chateau Du Nox application contains four different entity which are user, cart, order, and wine. Each user will have one and only one cart while is allowed to have one or multiple orders. Wine can be included in one or multiple customers' cart and order. Cart and order entity will not have a unique primary key instead they will have foreign keys to connect to other relevant entities.

3.0 User Interface Draft Design



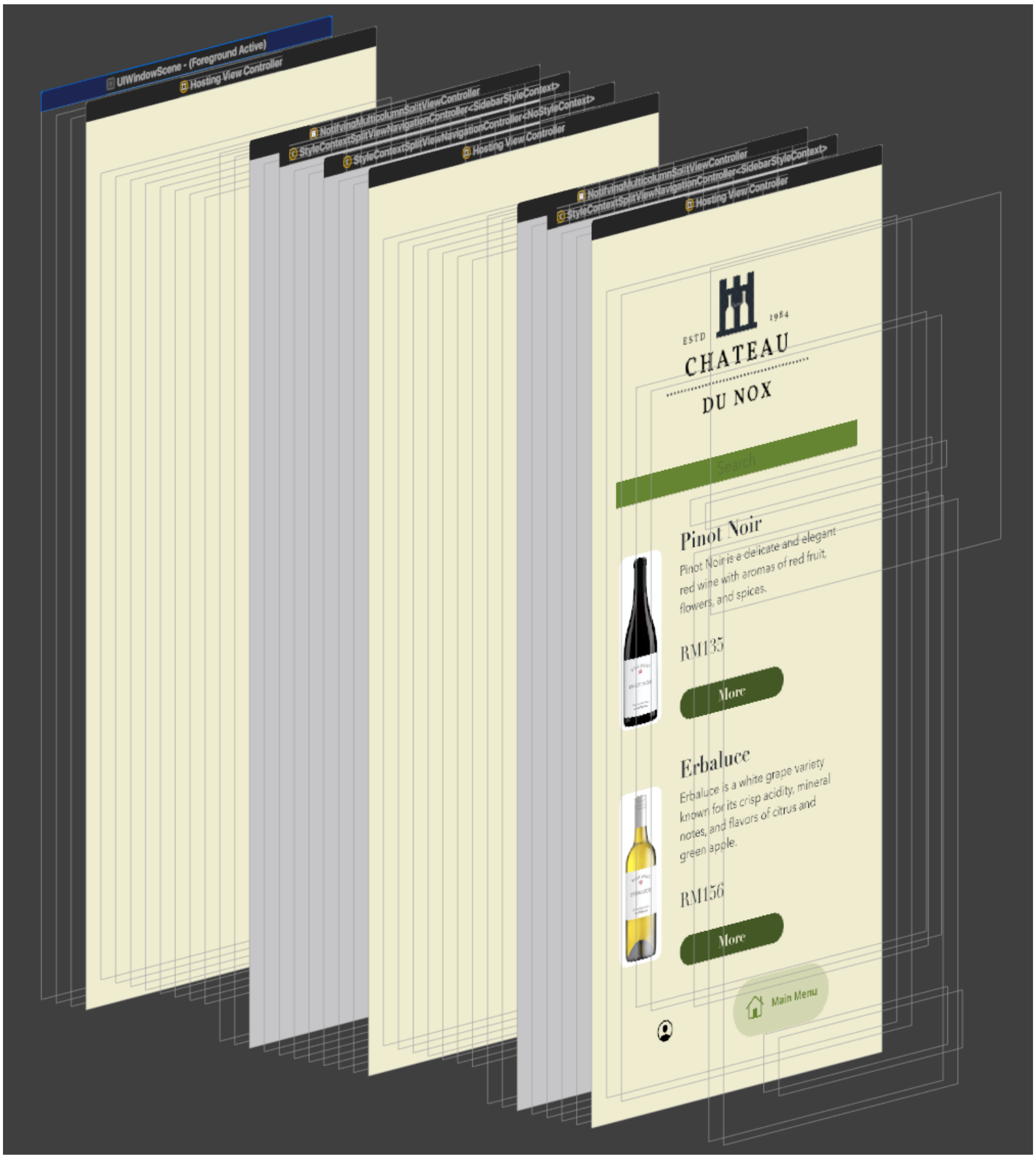
This is the initial drafting of the user interface including popups notification for Chateau Du Nox using Figma tools. There will be some changes during the programming phase because some user interface designs are time-consuming or not applicable. For instance, the fonts used in Figma design are Volkhorn SC and Mate but during programming phase, the fonts were changed to Italiana and Avenir Next due to the incompatibility of the downloaded font file. Other than that, a bottom tab bar is added to enhance users' experience and better navigation while the logo of Chateau Du Nox was moved to the top of the screen.

Based on an article by Fox, E. (2017), creating a user interface design is to help software developers to visualize the actual application and create the application faster without the need to think of the user interface and writing the code at the same time which will increase the efficiency and productivity.

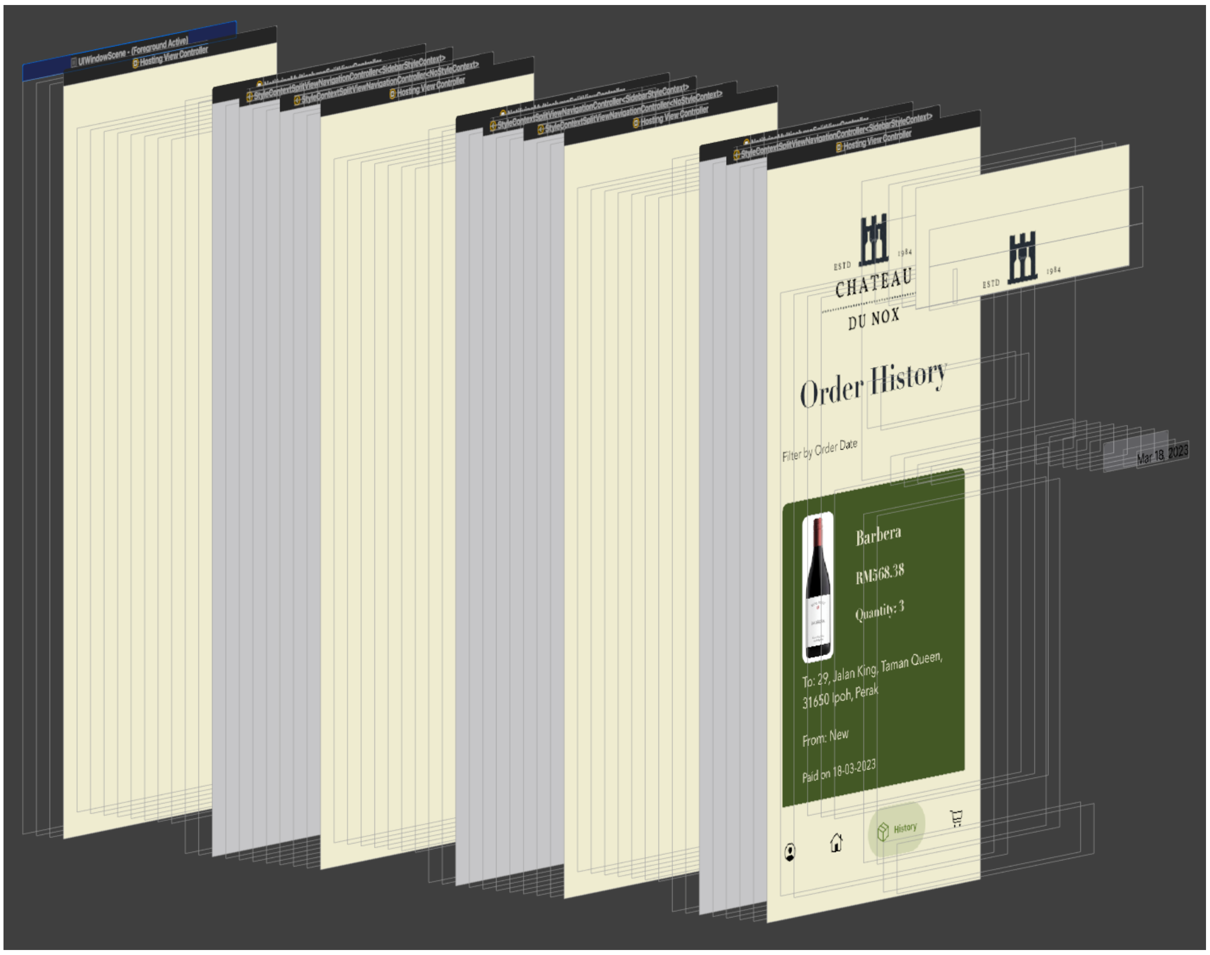
4.0 Application View Hierarchy

Adapting an article explanation from Anokhin, D. (2019), the application view hierarchy allows the visualization of the relationships between views in the application. By analysing the view hierarchy, a deeper understanding of the structure of the application and how different views are interconnected can be gained. This is very beneficial as it can help in identifying and resolving issues related to the layout, navigation, and user interaction.

Below are two application view hierarchies, the application when it is in guest mode and the application when users has logged in, respectively:


UIWindowScene - (Foreground Active)
Hosting View Controller
NotifyingMulticolumnSplitViewController
StyleContextSplitViewNavigationController<SidebarStyleContext>
StyleContextSplitViewNavigationController<NoStyleContext>
Hosting View Controller
NotifyingMulticolumnSplitViewController
StyleContextSplitViewNavigationController<SidebarStyleContext>
Hosting View Controller
ESTD 1984
CHATEAU DU NOX
Search
Pinot Noir
Pinot Noir is a delicate and elegant red wine with aromas of red fruit, flowers, and spices.
RM135
More
Erbaluce
Erbaluce is a white grape variety known for its crisp acidity, mineral notes, and flavors of citrus and green apple.
RM156
More
Main Menu

It is obvious that there are lesser view when the application is in guest mode because it is just to allow users to have a brief overall look of the application and just to browse for the wines that are available and in sales.



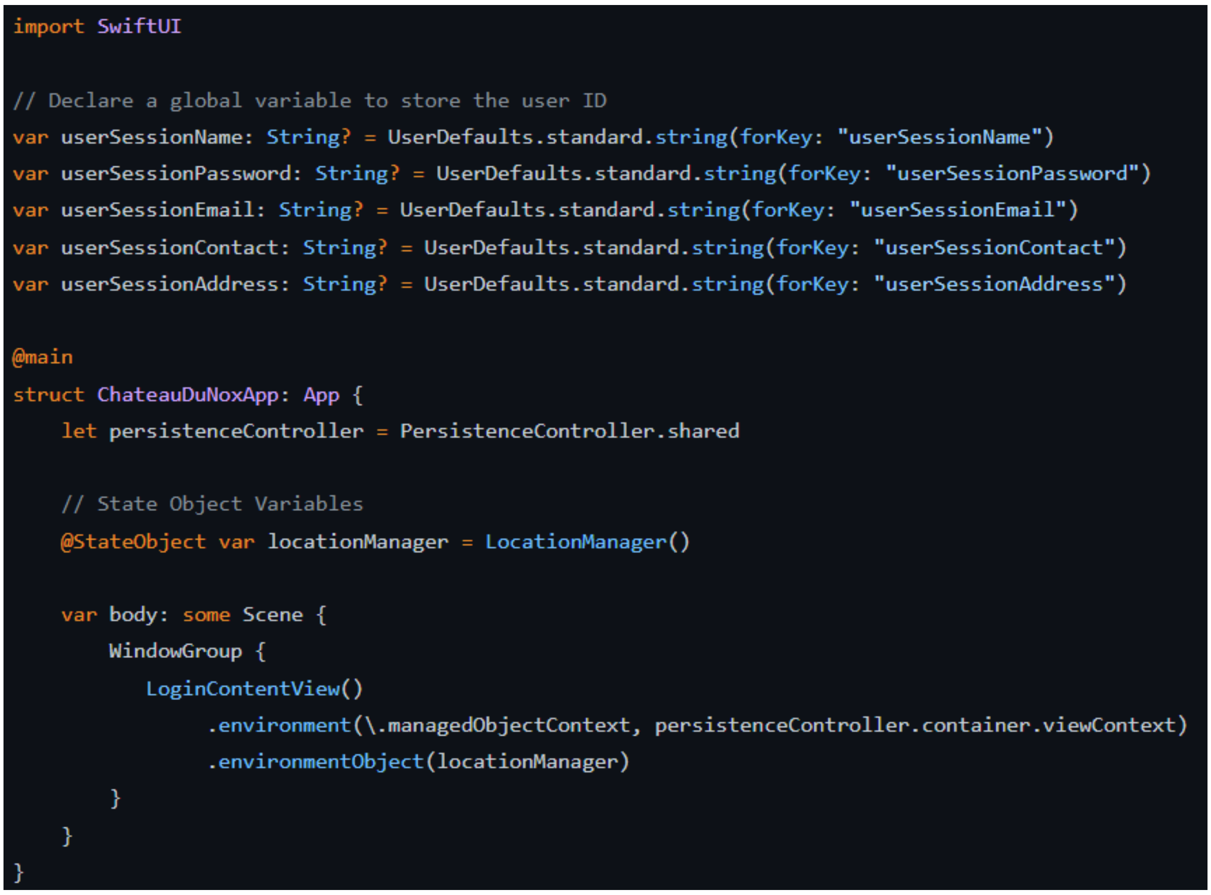
After users logged in with a verified account, they will have more access to the application where they can edit their personal information, place orders, view their carts and order history and browsing the store. Comparing to the previous application view hierarchy, the current hierarchy is much complex and complicated because there are various functions and views for users to access.

5.0 Frameworks

5.1 SwiftUI

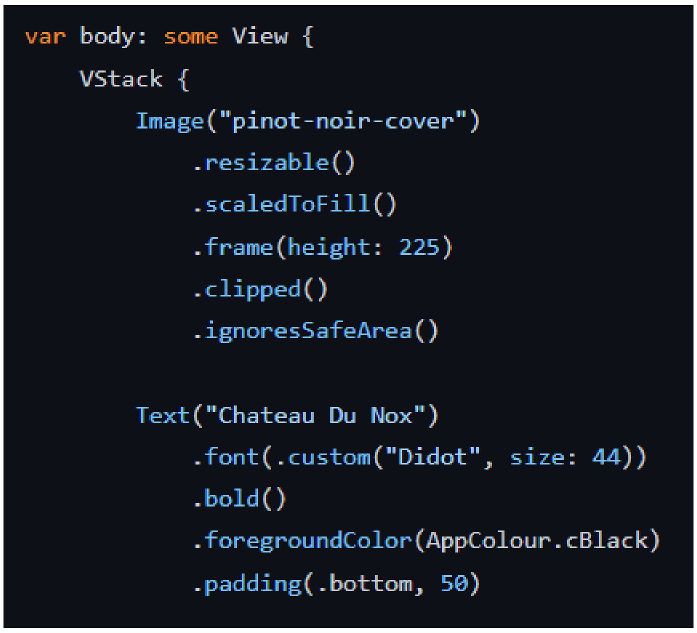
According to Jacobs B. (n.d.), SwiftUI offers developers the ability to create user interface like lists, buttons, stepper, text field, label, and etc. It also offers functions to include customize views, adding animations and implementing gestures like long-pressed and drag gesture.

Chateau Du Nox's application is built with SwiftUI as it is more effective and efficient in terms of programming compared to Storyboard's drag and drop. There are several code snippets about SwiftUI can be discussed in this project.



```swift
import SwiftUI

// Declare a global variable to store the user ID
var userSessionName: String? = UserDefaults.standard.string(forKey: "userSessionName")
var userSessionPassword: String? = UserDefaults.standard.string(forKey: "userSessionPassword")
var userSessionEmail: String? = UserDefaults.standard.string(forKey: "userSessionEmail")
var userSessionContact: String? = UserDefaults.standard.string(forKey: "userSessionContact")
var userSessionAddress: String? = UserDefaults.standard.string(forKey: "userSessionAddress")

@main
struct ChateauDuNoxApp: App {
    let persistenceController = PersistenceController.shared

    // State Object Variables
    @StateObject var locationManager = LocationManager()

    var body: some Scene {
        WindowGroup {
           LoginContentView()
                .environment(\.managedObjectContext, persistenceController.container.viewContext)
                .environmentObject(locationManager)
        }
    }
}
```

When creating a SwiftUI application, the first and foremost important lines of code are as above. The **@main** provides an entry point for the application where it will determine which view to render and start to build and run the simulator. During development, developers can change the view within **WindowGroup** to control which view to be rendered on startup of the simulator. This can increase the productivity of the developers and avoid the need to go through each completed view.



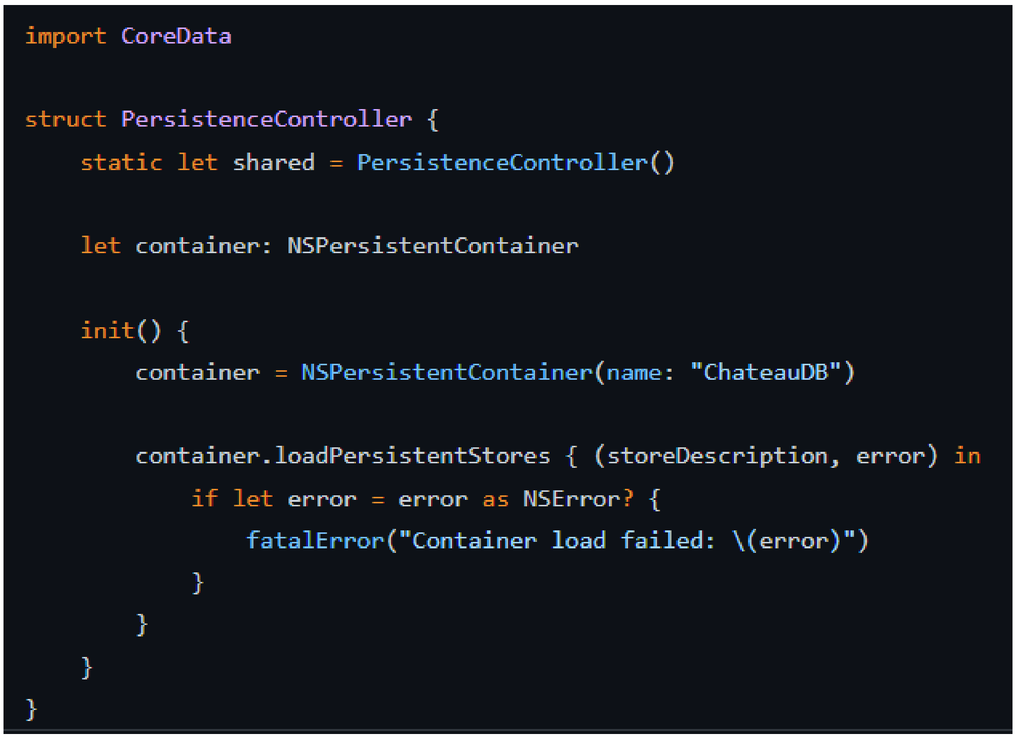
```swift
var body: some View {
    VStack {
        Image("pinot-noir-cover")
            .resizable()
            .scaledToFill()
            .frame(height: 225)
            .clipped()
            .ignoresSafeArea()

        Text("Chateau Du Nox")
            .font(.custom("Didot", size: 44))
            .bold()
            .foregroundColor(AppColour.cBlack)
            .padding(.bottom, 50)
```

As mentioned before, SwiftUI allows developers to create user interface easier. For example, in the code above, a vertical row can be created by using **VStack** where everything inside the stack are aligned from top to bottom. If a horizontal row is required, it can be created by using **HStack** instead of VStack**.** SwiftUI also offers the ability to create elements that stacked on top of each other with **ZStack** (DevTechie, 2022)**.** Continuing with the code snippet above, an **image** and a **text label** are created with just a few lines of code and relevant attributes are included to customize the appearance of the image and text.

5.2 CoreData

CoreData is a client-side caching framework that is developed by Apple which allows users to go offline while using an application. It provides a set of predefined modules that can perform simple create-read-update-delete methods or **CRUD** methods in a persistence layer (Assumani, M., 2019).

Chateau Du Nox's application stores customers' personal information, cart items, and order history using CoreData. Although it is better to store data in a cloud like Firebase, CoreData provides a faster access to the database because it is cached in the local devices where Firebase sometimes might encounter requesting or fetching issues, result in a timeout error.

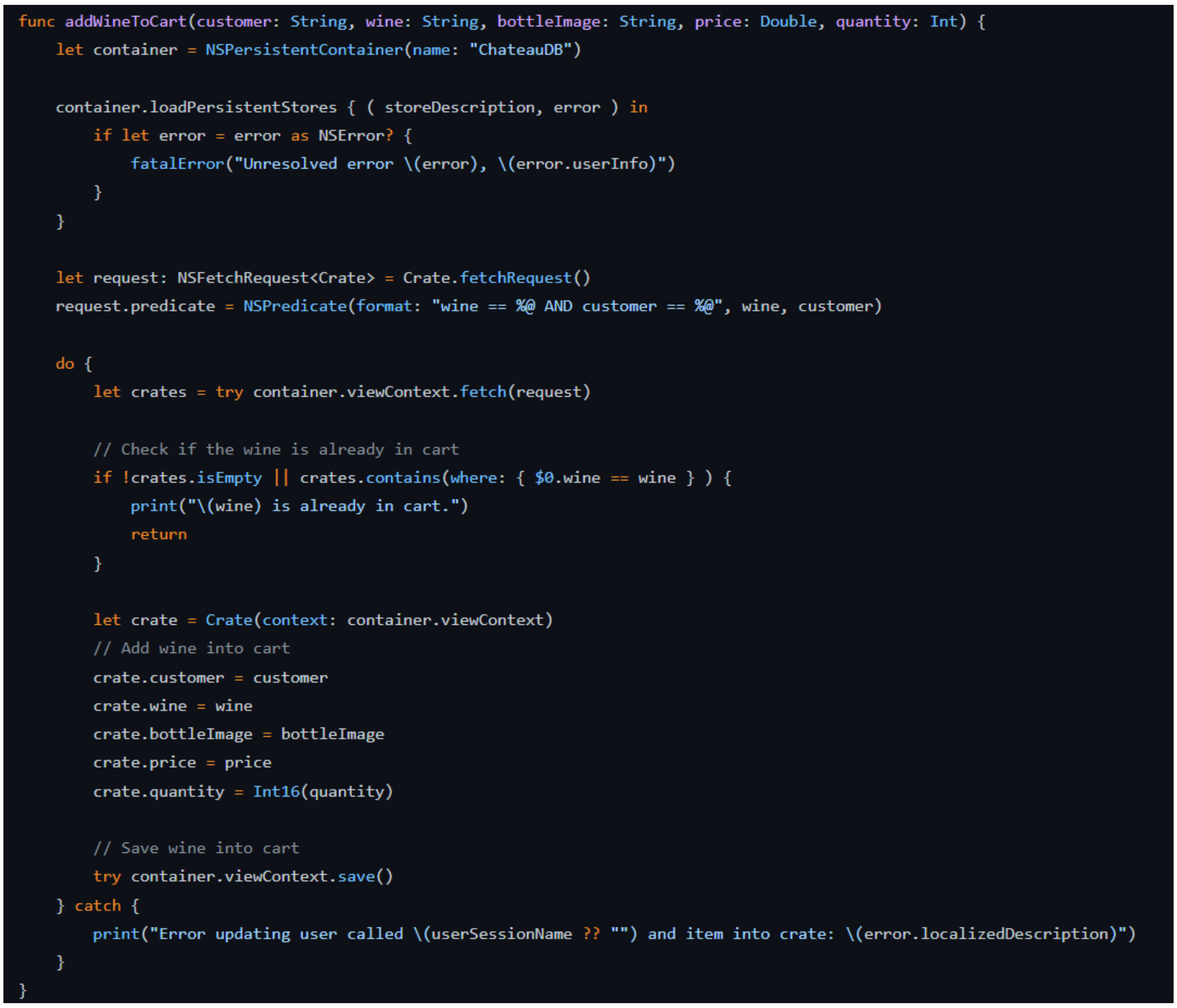


```swift
import CoreData

struct PersistenceController {
    static let shared = PersistenceController()

    let container: NSPersistentContainer

    init() {
        container = NSPersistentContainer(name: "ChateauDB")

        container.loadPersistentStores { (storeDescription, error) in
            if let error = error as NSError? {
                fatalError("Container load failed: \(error)")
            }
        }
    }
}
```

The code snippet above is the most important part of implementing CoreData in any application. It is used to manage the persistence of data in iOS or macOS application. The code provided contains two properties, **shared** and **container**. The property **shared** is a static instance of the **PersistenceController** struct while **container** is an instance of the **NSPersistentContainer** class that manages the CoreData stack.

Within the **init** function, NSPersistentContainer with the name of **ChateauDB** is initialized and the persistent store is loaded. When the loading process is completed, the closure passed to the **loadPersistentStores** is executed. An error fallback is provided for the information of developers in case the loading process failed.

On the other hand, the property shared is defined as a static instance of the PersistenceController struct, making it accessible globally throughout the application without a new instance of the struct. It also ensures that there is only one instance of NSPersistentContainer and CoreData stack is properly managed throughout the application.



```swift
func addWineToCart(customer: String, wine: String, bottleImage: String, price: Double, quantity: Int) {
    let container = NSPersistentContainer(name: "ChateauDB")

    container.loadPersistentStores { ( storeDescription, error ) in
        if let error = error as NSError? {
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    }

    let request: NSFetchRequest<Crate> = Crate.fetchRequest()
    request.predicate = NSPredicate(format: "wine == %@ AND customer == %@", wine, customer)

    do {
        let crates = try container.viewContext.fetch(request)

        // Check if the wine is already in cart
        if !crates.isEmpty || crates.contains(where: { $0.wine == wine } ) {
            print("\(wine) is already in cart.")
            return
        }

        let crate = Crate(context: container.viewContext)
        // Add wine into cart
        crate.customer = customer
        crate.wine = wine
        crate.bottleImage = bottleImage
        crate.price = price
        crate.quantity = Int16(quantity)

        // Save wine into cart
        try container.viewContext.save()
    } catch {
        print("Error updating user called \(userSessionName ?? "") and item into crate: \(error.localizedDescription)")
    }
}
```

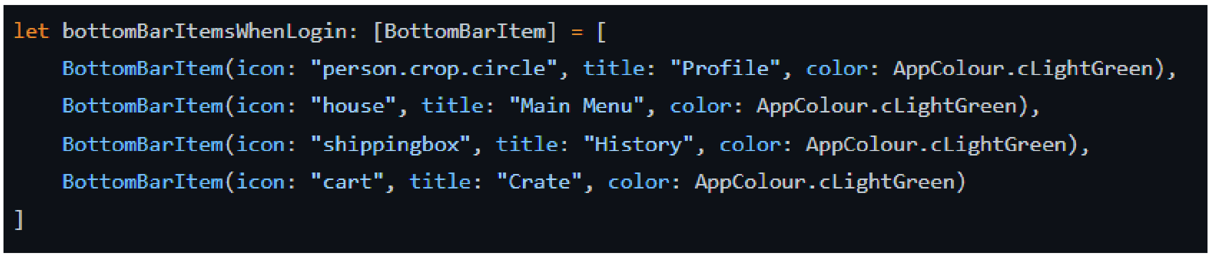
Here is an example of adding data into CoreData. When customers add a wine to his or her cart, **addWineToCart** function is called. In this function, five parameters, **customer name, wine, the image of the wine, price** and **quantity** are required.

It starts with by initializing a **NSPersistentContainer** and loading the persistent store and an error fallback is provided in case the application crashed unexpectedly. After that, the fetch request is created to retrieve **Crate** objects which the information about customers' cart items. The fetch request is filtered using the customer's name and the wine to check whether the customer has the same items in the wine. This is included to avoid item redundancy within the cart.

If the wine is not the cart, then the wine will be added along with the customer's name to determine which order belongs to each customer, and the data is save by using **container.viewContext.save** function. An error message is provided in the error fallback in case there is an error in adding the item into the cart.

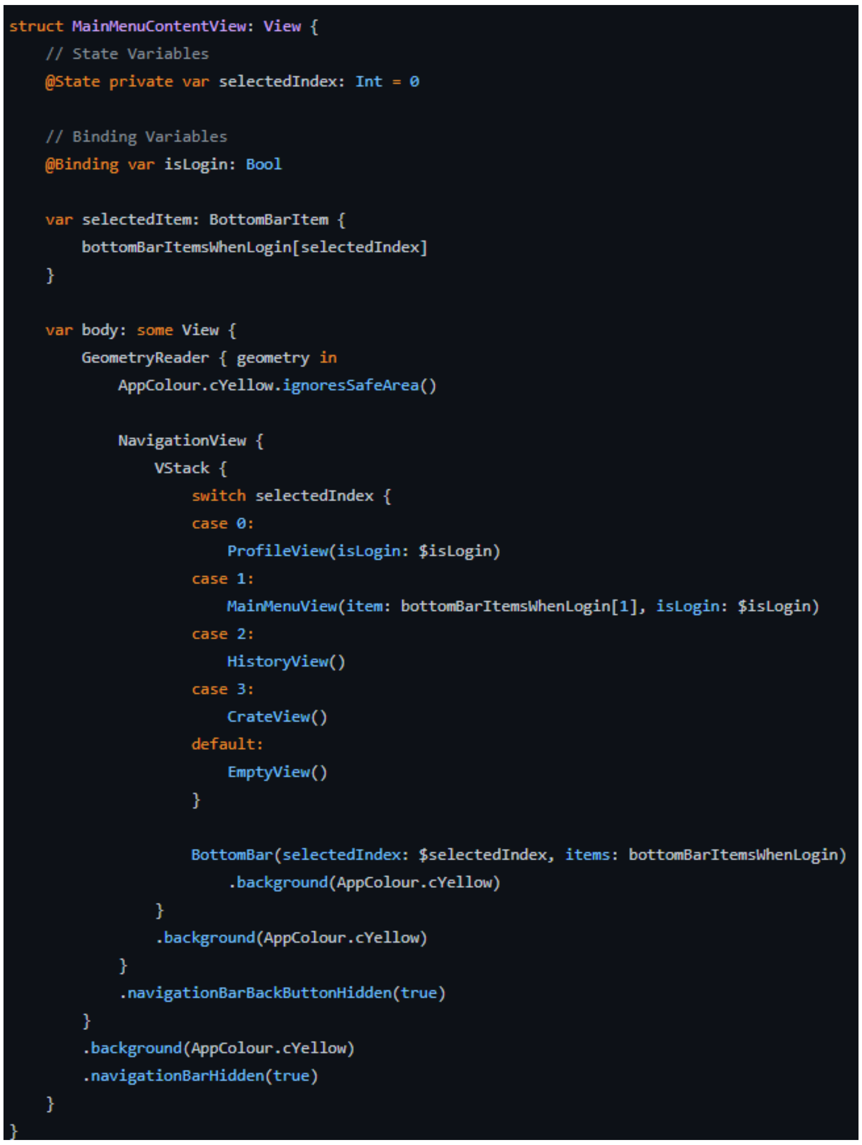
5.3 BottomBar_SwiftUI

BottomBar_SwiftUI is an external framework created by **smartvipere75** from GitHub. It provides the ability to create a bottom tab bar to access multiple view in a single view without the need of redirecting or moving back or forth between views (smartvipere75, 2022).



```swift
let bottomBarItemsWhenLogin: [BottomBarItem] = [
    BottomBarItem(icon: "person.crop.circle", title: "Profile", color: AppColour.cLightGreen),
    BottomBarItem(icon: "house", title: "Main Menu", color: AppColour.cLightGreen),
    BottomBarItem(icon: "shippingbox", title: "History", color: AppColour.cLightGreen),
    BottomBarItem(icon: "cart", title: "Crate", color: AppColour.cLightGreen)
]
```

The code snippet above is the first requirement of creating a bottom tab bar with four items. The maximum items can be created in the tab bar is **four**. An array of **BottomBarItem** is created outside of the struct in order to be used in the second requirements.



```swift
struct MainMenuContentView: View {
    // State Variables
    @State private var selectedIndex: Int = 0

    // Binding Variables
    @Binding var isLogin: Bool

    var selectedItem: BottomBarItem {
        bottomBarItemsWhenLogin[selectedIndex]
    }

    var body: some View {
        GeometryReader { geometry in
            AppColour.cYellow.ignoresSafeArea()

            NavigationView {
                VStack {
                    switch selectedIndex {
                    case 0:
                        ProfileView(isLogin: $isLogin)
                    case 1:
                        MainMenuView(item: bottomBarItemsWhenLogin[1], isLogin: $isLogin)
                    case 2:
                        HistoryView()
                    case 3:
                        CrateView()
                    default:
                        EmptyView()
                    }

                    BottomBar(selectedIndex: $selectedIndex, items: bottomBarItemsWhenLogin)
                        .background(AppColour.cYellow)
                }
                .background(AppColour.cYellow)
            }
            .navigationBarBackButtonHidden(true)
        }
        .background(AppColour.cYellow)
        .navigationBarHidden(true)
    }
}
```

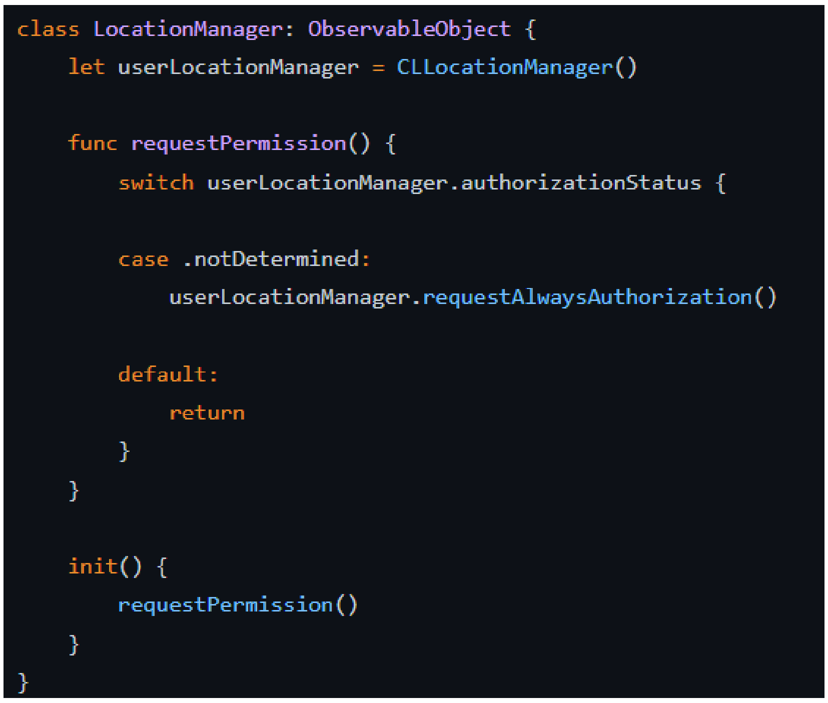
After that, a **@State** variable is created to govern the index of the bottom tab bar when users want to navigate between views using the tab bar. The bottom tab bar is then created using **BottomBar** with the parameters of **selectedIndex** and **items** referring to the array creating in the first requirement.

An important thing to ensure is that another struct called **MainMenuContentView** is created to encapsulate four different views, **ProfileView, MainMenuView, HistoryView** and **CrateView**. When the user logged in successfully, the redirection will be **MainMenuContentView** so that the four views will be accessible to the user through the bottom tab bar.

5.4 CoreLocation

According to Kharchyshyn, A. (2018), CoreLocation is an iOS framework that allows developers to track the user locations and it works along maps. For example, it can be used to suggest different cafes or gyms nearby the current user's location or track the user's movements as a sport fitting application.

In Chateau Du Nox's application, CoreLocation is used to determine the current user location if the user accept the request permission. The current user location is required to determine where the user is from assuming there is an imaginary function or methods is called when the user is using the application to track the demographic of all users of Chateau Du Nox for business purposes.



```
class LocationManager: ObservableObject {
    let userLocationManager = CLLocationManager()

    func requestPermission() {
        switch userLocationManager.authorizationStatus {

        case .notDetermined:
            userLocationManager.requestAlwaysAuthorization()

        default:
            return
        }
    }

    init() {
        requestPermission()
    }
}
```

The code snippet above defined a class called **LocationManager** that conforms to the **ObservableObject** protocol. The class contains a property **userLocationManager** which is an instance of **CLLocationManager** class, a standard iOS class that provides location-related services.

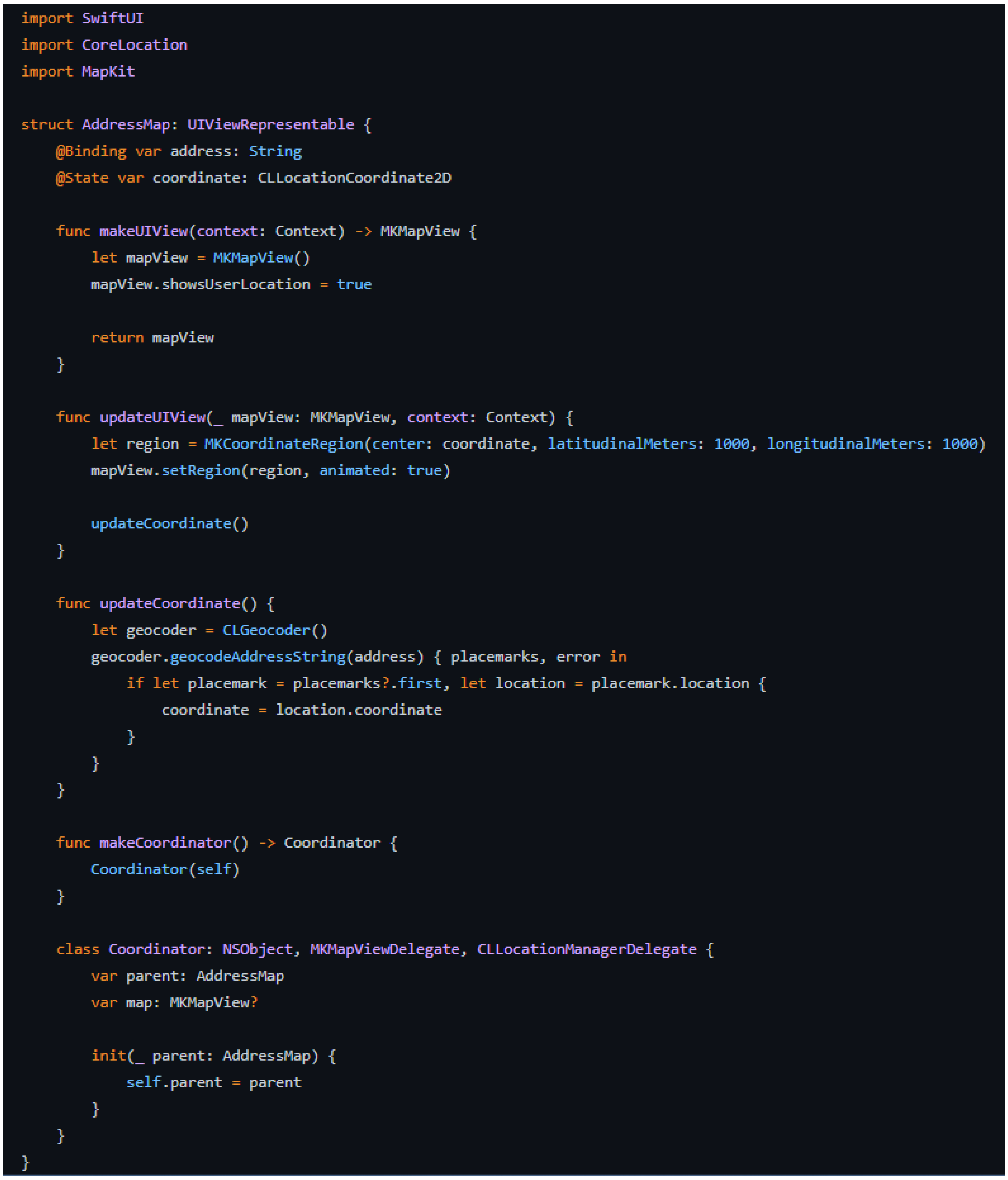
The LocationManager class also has a method called **requestPermission** which is to request the user's permission to access the current user's location. If the status is **notDetermined**, the method requests **always authorization** while if the users has granted the permission, it will return without doing anything.

When the **init** function is called when an instance of LocationManager is created and runs the method of requestPermission. By conforming ObservableObject protocol, instances of this class can be observed by other objects in SwiftUI framework and when the properties changed, SwiftUI will automatically updates any views that are bound to those properties.

5.5 MapKit

Based on an article written by Tetlaw, A. (2020), MapKit is another framework developed by Apple similar to CoreLocation and both can be used in tandem. MapKit enables developers to display map into an application and mark locations with pins and enhance customized data and create routes in the map.

Chateau Du Nox's application take advantage of this framework during address confirmation when user wants to confirm their delivery address. It allows users to change their delivery address even though they have an address in their personal information already. This avoids the hassle to edit user's address in their personal information and directly change their delivery address as they proceed to checkout.



```swift
import SwiftUI
import CoreLocation
import MapKit

struct AddressMap: UIViewRepresentable {
    @Binding var address: String
    @State var coordinate: CLLocationCoordinate2D

    func makeUIView(context: Context) -> MKMapView {
        let mapView = MKMapView()
        mapView.showsUserLocation = true

        return mapView
    }

    func updateUIView(_ mapView: MKMapView, context: Context) {
        let region = MKCoordinateRegion(center: coordinate, latitudinalMeters: 1000, longitudinalMeters: 1000)
        mapView.setRegion(region, animated: true)

        updateCoordinate()
    }

    func updateCoordinate() {
        let geocoder = CLGeocoder()
        geocoder.geocodeAddressString(address) { placemarks, error in
            if let placemark = placemarks?.first, let location = placemark.location {
                coordinate = location.coordinate
            }
        }
    }

    func makeCoordinator() -> Coordinator {
        Coordinator(self)
    }

    class Coordinator: NSObject, MKMapViewDelegate, CLLocationManagerDelegate {
        var parent: AddressMap
        var map: MKMapView?

        init(_ parent: AddressMap) {
            self.parent = parent
        }
    }
}
```

The code snippet shows that **AddressMap** struct that conforms to **UIViewRepresentable.** The struct contains two properties, a **Binding String** called **address** which holds the user’s address and a **State** called **coordinate** which holds the location coordinates of the user’s address.

The **makeUIView** method is called when the view is first created and it returns an instance of **MKMapView** which is a standard iOS class that provides a map interface. It also sets the **showsUserLocation** property of the map view to **true** to display the user's current location.

Another method called **updateUIView** is called whenever the view needs to be updated. It sets the map view's region to a specific coordinate using **MKCoordinateRegion** and **setRegion** methods. It also calls **updateCoordinate** methods which uses **CLGeocoder** instance to convert the user's address into a location coordinate and updates the coordinate property.

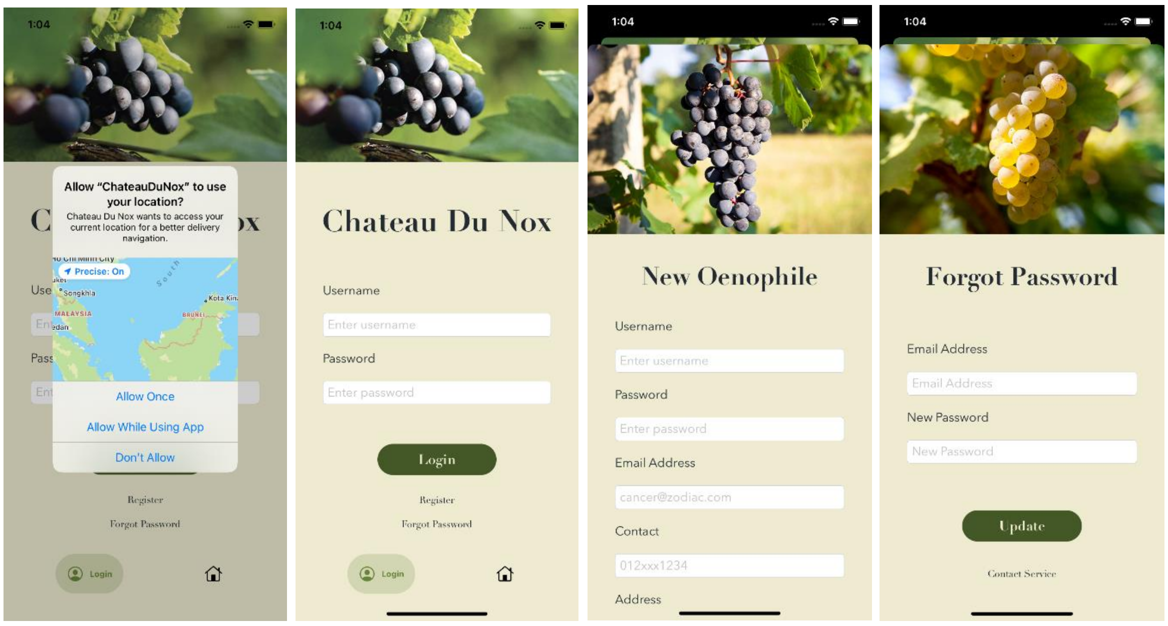
The **makeCoordinator** method is used to create a coordinator object that manages the interaction between SwiftUI view and the underlying UIKit view. It returns an instance of **Coordinator** class which is a subclass of **NSObject** that conforms to both **MKMapViewDelegate** and **CLLocationManagerDelegate.**

The Coordinator class has two properties, **parent** which is referring to the parent AddressMap instance and **map** which is an optional reference to the **MKMapView** instance. The Coordinator is used to handle events and delegate methods from the map view and location manager and update the parent view accordingly.

Using these lines of code, the map will automatically redirect and re-centre from a random location to the address that the user typed in the text field when they are checking out their cart items. This is to allow user to check whether the address they typed are the correct address that they wished their order to be delivered to.

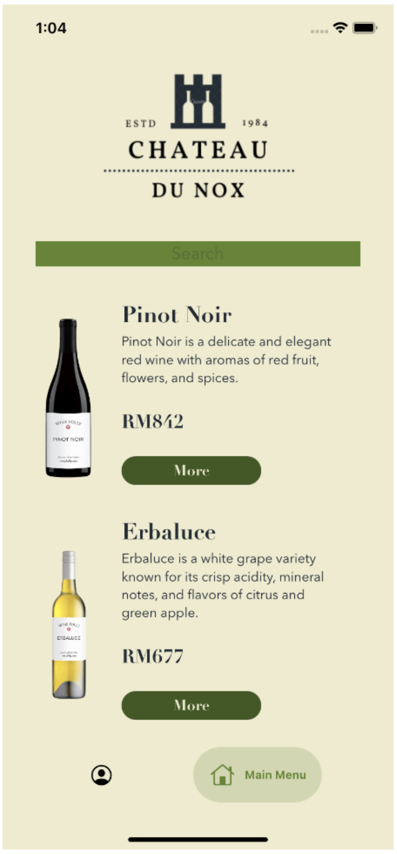
6.0 User Manual

6.1 Login / Register / Forgot Password



Before proceeding further, the application will request the location access from the users for demographic business purposes. In the login view, users are presented with various options to access their account securely. They can create a new account by providing their email address or reset the password of their existing account by verifying their email address. For added convenience, users can also login with their username and password to gain access to their account. To ensure data security, the login view includes robust data validation features that protect against unauthorized access. If a user forgets their email address, they can easily contact customer service for further assistance.

6.2 Guest Main Menu


1:04

ESTD 1984 CHATEAU DU NOX

Search

Pinot Noir

Pinot Noir is a delicate and elegant red wine with aromas of red fruit, flowers, and spices.

RM842

More

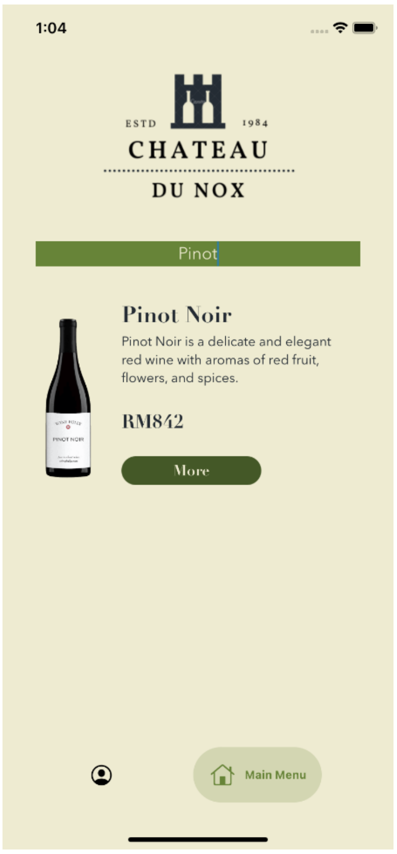
Erbaluce

Erbaluce is a white grape variety known for its crisp acidity, mineral notes, and flavors of citrus and green apple.

RM677

More

Main Menu


1:04

ESTD 1984 CHATEAU DU NOX

Pinot

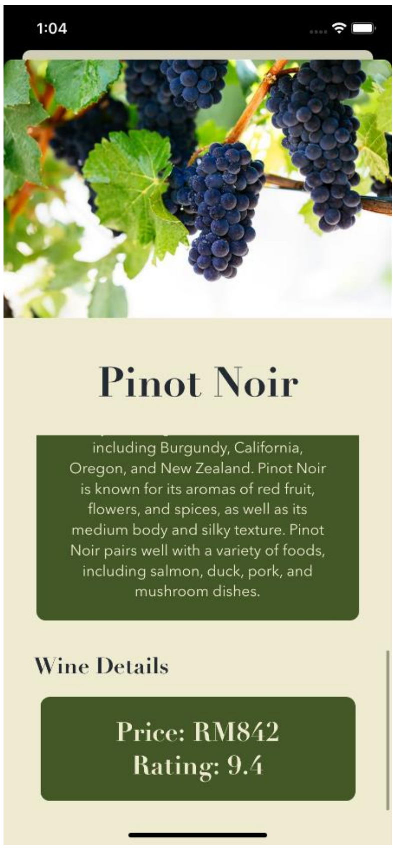
Pinot Noir

Pinot Noir is a delicate and elegant red wine with aromas of red fruit, flowers, and spices.

RM842

More

Main Menu


1:04

Pinot Noir

including Burgundy, California, Oregon, and New Zealand. Pinot Noir is known for its aromas of red fruit, flowers, and spices, as well as its medium body and silky texture. Pinot Noir pairs well with a variety of foods, including salmon, duck, pork, and mushroom dishes.

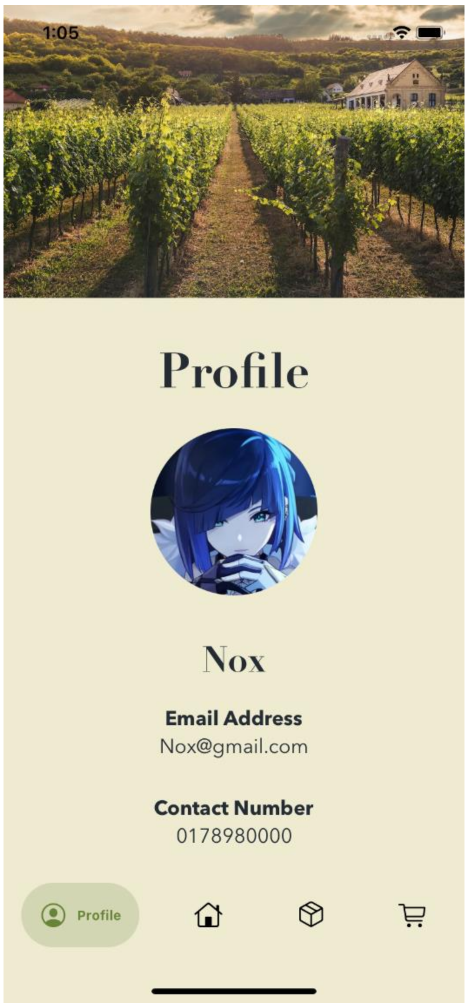
Wine Details

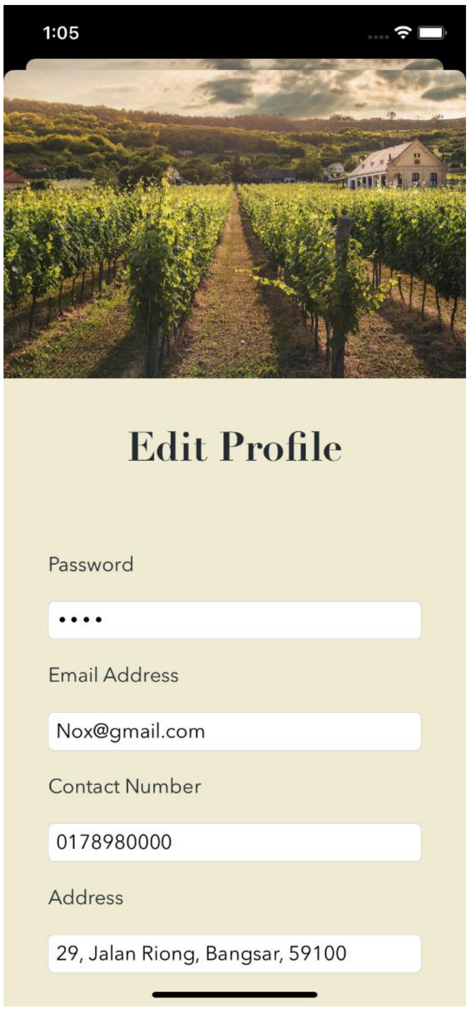
Price: RM842
Rating: 9.4

For an immersive experience for unregistered users who are interested in exploring the wine selection at Chateau Du Nox, they can easily navigate to the menu view through the bottom tab bar. Once there, users can immediately browse through the entire selection of available wines. To help users further refine their search, the menu view includes a robust filtering system, allowing users to sort through the wines by name.

For more information about a particular wine, users can simply tap the "More" button to open the wine detail view. Here, users can find a beautiful cover image showcasing the wine's grapes, as well as an enlarged image of the wine bottle. Detailed content descriptions of the wine, including tasting notes and suggested pairings, are also available to help users make informed decisions about their selection. In addition, users can view the wine's rating and price, allowing them to easily compare various wines in the collection.

6.3 Profile

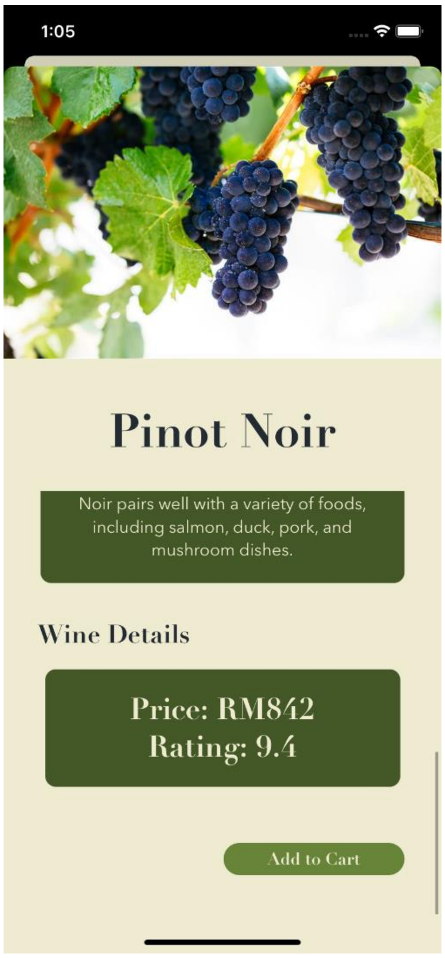




Upon a successful login, users are directed to their profile view, where they can easily manage and update their personal details. Although image may be customized in the future version, they still have the ability to update their password, email address, contact number, and address by simply clicking on the "Edit" button.

To ensure the security of the user's data, the profile view includes robust data validation features that check for errors and ensure that the information provided is accurate and valid. In addition to updating their personal details, users can also easily logout from their account by clicking on the "Logout" button. This helps to ensure the security of the user's account and protect against unauthorized access.

6.4 Main Menu

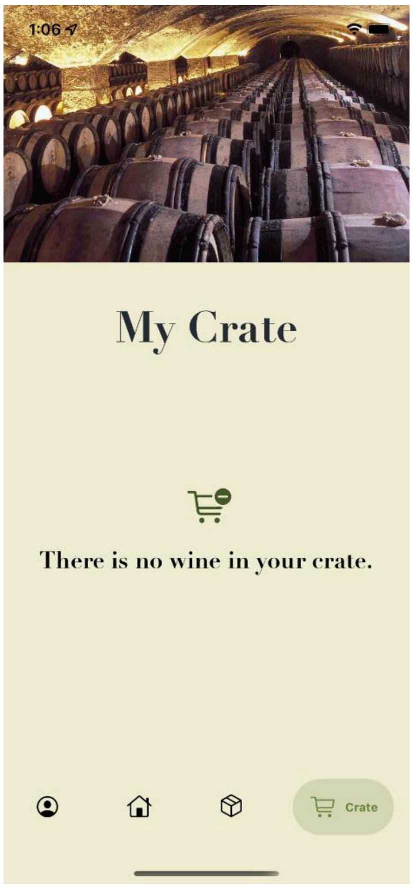


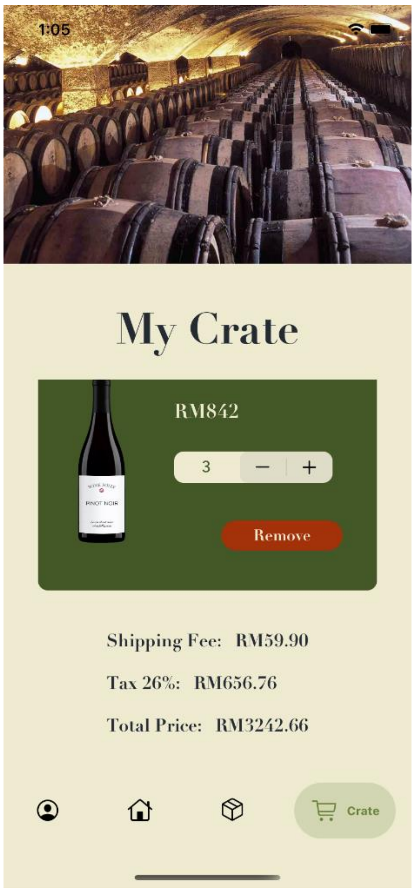
Noir pairs well with a variety of foods, including salmon, duck, pork, and mushroom dishes.

Price: RM842
Rating: 9.4

Similar to the guest main menu, logged-in users can browse through all wines available in the menu view, and have the ability to filter the wines by name. Additionally, users can access the wine detail view by clicking on the "More" button to gain a deeper understanding of the wines they are interested in. However, unlike the guest main menu, the wine detail view for logged-in users includes an "Add to Cart" button, allowing users to easily add wines to their cart for purchase.

6.5 Cart / Crate



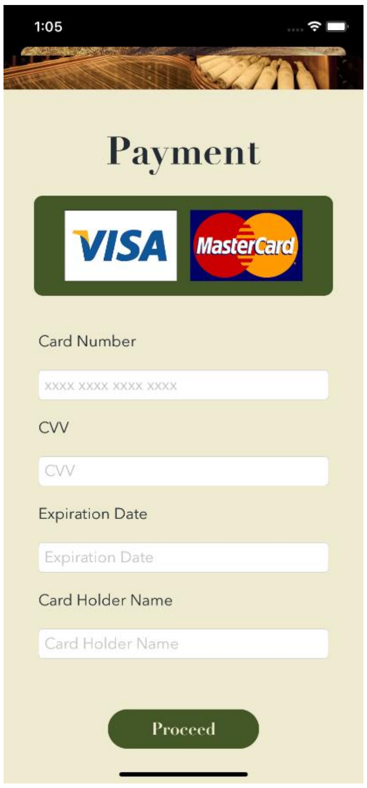


In the cart view, users will be initially presented with a message that says "No items in cart" until they add wines into their cart. Once they have added wines to their cart, they can then adjust the quantity of each wine they wish to purchase. However, to maintain the integrity of the ordering process and ensure that all customers can obtain the wine they want, there is a limit per order. Each order must contain at least one bottle of wine and a maximum of 12 bottles or a carton of each wine per order.

For example, a user can order up to 12 bottles of Barbera wine in a single order. If they wish to purchase more than 12 bottles, they can create a new order for additional bottles, assuming that the delivery service will process both orders on the same day and deliver them together in the same batch.

The cart view also includes shipping fees of RM59.90 and 26% tax price per order, which are automatically calculated and displayed to the user. As users adjust the quantity of wine in their cart, the total amount they need to pay will be updated in real-time to reflect the changes. All quantities of wine in the cart are updated in the database to ensure accurate inventory management.

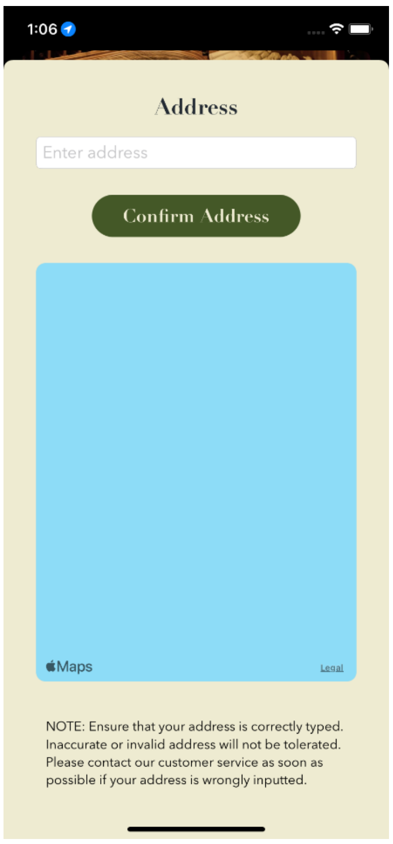
6.6 Payment



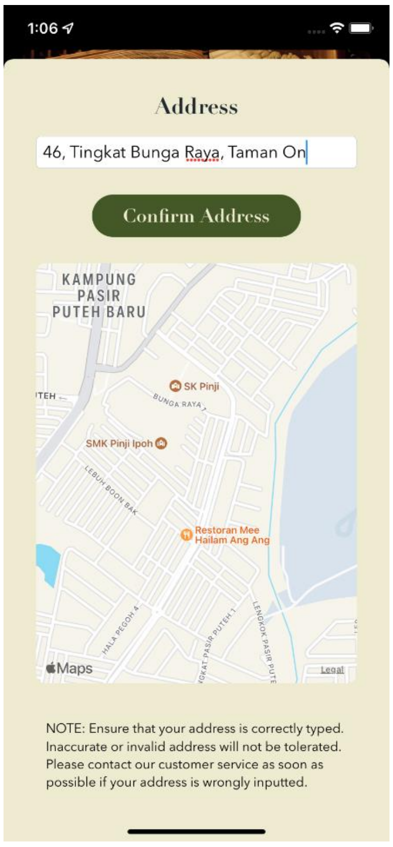
After users click on the "Checkout" button, they will be directed to a payment modal where they can input their credit or debit card information, including the card number, CVV, expiration date, and cardholder name. These fields are necessary to mimic the transaction process and allow users to pay for their orders.

The payment modal will include clear examples of the format required for each field to guide users in inputting their information correctly. Additionally, data validation will be included to ensure that users enter valid and complete information. For example, the card number field will require a 16-digit number, the CVV field will require a 3-digit number, and the expiration date field will require a valid month and year format.

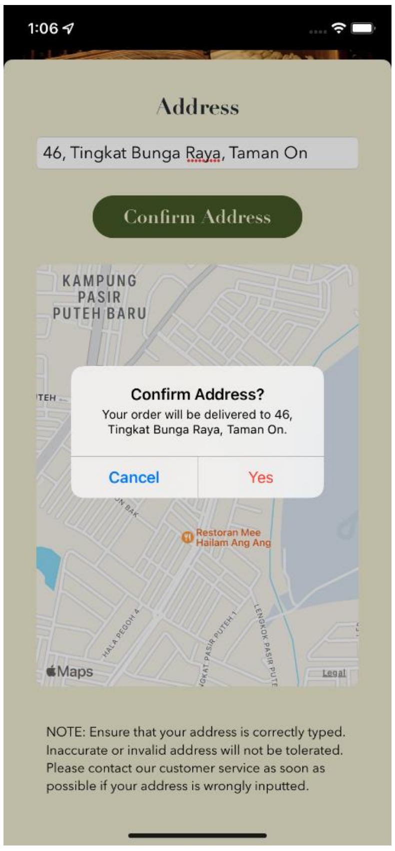
6.7 Address



NOTE: Ensure that your address is correctly typed. Inaccurate or invalid address will not be tolerated. Please contact our customer service as soon as possible if your address is wrongly inputted.



NOTE: Ensure that your address is correctly typed. Inaccurate or invalid address will not be tolerated. Please contact our customer service as soon as possible if your address is wrongly inputted.



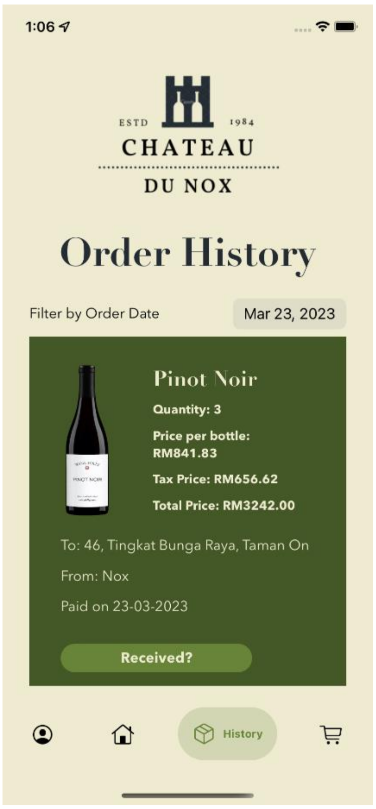
Confirm Address?
Your order will be delivered to 46, Tingkat Bunga Raya, Taman On.

NOTE: Ensure that your address is correctly typed. Inaccurate or invalid address will not be tolerated. Please contact our customer service as soon as possible if your address is wrongly inputted.

Assuming that users have typed a valid and correct address, the map will automatically re-center to the location that users have entered. This ensures that the delivery driver will have an accurate location to deliver the order.

To prevent potential issues caused by incorrect address information, a popup alert will be triggered before the payment and order transaction is processed. This alert will remind users to verify their address information and confirm that it is accurate. If the user believes that they have entered incorrect information, they will have the opportunity to correct the address before proceeding with the transaction.

6.8 Order History


1:06

ESTD 1984 CHATEAU DU NOX

Order History

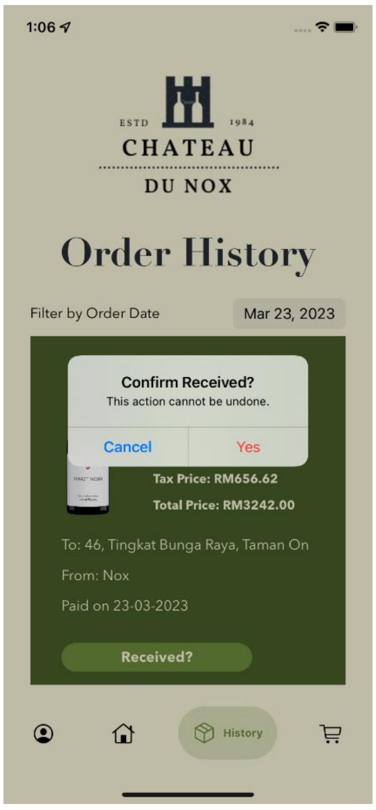
Filter by Order Date Mar 23, 2023

Pinot Noir
Quantity: 3
Price per bottle: RM841.83
Tax Price: RM656.62
Total Price: RM3242.00

To: 46, Tingkat Bunga Raya, Taman On
From: Nox
Paid on 23-03-2023

Received?

History


1:06

ESTD 1984 CHATEAU DU NOX

Order History

Filter by Order Date Mar 23, 2023

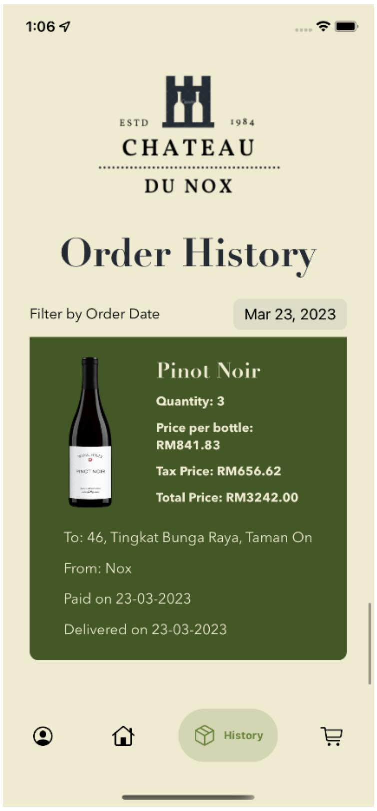
Confirm Received?
This action cannot be undone.
Cancel Yes

Tax Price: RM656.62
Total Price: RM3242.00

To: 46, Tingkat Bunga Raya, Taman On
From: Nox
Paid on 23-03-2023

Received?

History


1:06

ESTD 1984 CHATEAU DU NOX

Order History

Filter by Order Date Mar 23, 2023

Pinot Noir
Quantity: 3
Price per bottle: RM841.83
Tax Price: RM656.62
Total Price: RM3242.00

To: 46, Tingkat Bunga Raya, Taman On
From: Nox
Paid on 23-03-2023
Delivered on 23-03-2023

History

After users have placed an order successfully, the items in the cart will be cleared, and users will be redirected to the order history view. In the order history view, users will be able to see their recent orders along with order details and the intended delivery address.

Assuming that users have received their wine orders and there is a delivery date attached to the parcel, they can navigate back to the order history view and filter through the dates to find the relevant order. For each order that has been delivered but not yet marked as received, there will be a "Received?" button available.

When users click on the "Received?" button, the application will record the date and time based on the users' device current date and time. This information will be stored in the application's database, allowing users to track their received orders easily. Additionally, this information will help the application to provide better service to its users by accurately tracking the status of each order.

7.0 Future Versions and Updates

7.1 Auto Refresh

Auto-refreshing may not seem like a critical feature at first glance, but in applications like Chateau Du Nox, it's a cornerstone feature. By automatically refreshing the view, users can enjoy a seamless and hassle-free experience, without having to manually refresh the page or navigate to other views to see updated data. This not only streamlines the user experience, but it also helps to eliminate frustration and save valuable time. In short, auto-refreshing is a small yet significant feature that can greatly enhance the usability and functionality of an application.

7.2 Longer User Session

As of now, the Chateau Du Nox application only maintains the user's login session until the user quits the app. This means that if the user swipes up and closes the application, their session will be lost, and they will have to log in again when they open the app.

To provide a better user experience, we can enhance the app by introducing a longer user session. With a longer session, users can leave the app and return to it later without having to log in again. This will save them time and effort and provide a more seamless experience.

To implement this, we can extend the duration of the user session by increasing the time that the app keeps the user logged in. We can also add a feature to allow users to manually log out if they wish to do so. This will give users more control over their account security and privacy.

By enhancing the app with a longer user session, we can improve the user experience and increase user engagement. This can lead to greater customer satisfaction and loyalty, and ultimately benefit the business by driving increased usage and revenue.

7.3 Easier and Friendlier User Interface

Chateau Du Nox's user interface has the potential to be significantly enhanced for improved navigation and increased user-friendliness. Currently, some minor issues exist that may lead to confusion among users. For example, the calculation of base price and tax price in order history is based on the total price paid by customers, resulting in a slight difference in the base price displayed in the main menu. Rectifying this issue will go a long way in enhancing the application's usability.

Furthermore, when users view their personal information after logging in, the layout is not as professional as the standard set by other applications in the market. To improve the user experience, the layout can be revamped to enhance its aesthetic appeal and make it more intuitive to navigate. Other issues worth mentioning is that when users updating their personal information excluding email address, the application will not update the personal information because the email address of the user existed in the database. This problem can be solved in future version by adding another logic parameter to allow users to update their personal information if there is no similar email address or the email address is the same as the user’s current email address.

Additionally, the application's wine details can be further improved by adding multiple photos in a slider, allowing users to inspect wines before purchasing. This will enable users to make more informed decisions and increase the likelihood of repeat purchases. A more specific filter function can be introduced so users can filter through the category of wines either red or white wines, range of price, wine rating etc.

Finally, implementing shortcut keys or widgets is another way to enhance the user experience. For example, including a collapsed menu or an accordion menu will enable users to navigate different views with fewer clicks or taps, streamlining the process and making the application more efficient to use. These enhancements will significantly improve Chateau Du Nox's user interface, making it a more attractive option for users in the market.

7.4 Enhance Map Tracking during Ordering Process

The map tracking feature in Chateau Du Nox has the potential to be improved to enhance the user experience. Currently, the map displays the location of the user's inputted address, but there is no pin or marker displayed on the location. This can cause confusion when users want to confirm their delivery address, leading to unnecessary errors and delays in the ordering process.

To mitigate this issue, the map can be enhanced with a function that automatically detects the address when users long-press on the map and drops a marker on the intended location. This will simplify the ordering process for users by eliminating the need to type out long addresses, increasing the likelihood of repeat orders and improving the overall user experience.

Additionally, some addresses are not recognized by the map, potentially due to an outdated map version. To ensure that the service of Chateau Du Nox can cover more users in different locations, the map version needs to be updated regularly to ensure accurate address recognition and expanded coverage.

In conclusion, improving the map tracking feature in Chateau Du Nox will significantly enhance the user experience, simplifying the ordering process and reducing the likelihood of errors or delays. Upgrading the map version will also ensure that the application can serve a wider range of users, improving its overall accessibility and appeal.

7.5 Provide More In-depth Details about Wines

Chateau Du Nox can significantly improve its appeal to oenophiles by expanding its wine database. Including more wines in the database will allow the application to better satisfy the needs of the large community of wine enthusiasts. Additionally, enhancing the details of each wine, such as the terroir, aging, body, tannin, acidity, sweetness, and other factors, will provide more in-depth information about each wine, further enhancing the user experience.

Collaborating with other vineyards to broaden the wine data while increasing profits and sales is another strategy that can benefit Chateau Du Nox. By promoting and reselling the wines of other vineyards, the application can earn a certain amount of commission, creating a mutually beneficial partnership. This will not only increase the variety of wines offered but also attract new customers who may be loyal to these vineyards.

Furthermore, including a small feature like a wiki in the application will enable new oenophiles or customers to learn more about wine grapes, the process of making wine, and aging wine, among other relevant information. This feature will promote wine in a beneficial way, allowing Chateau Du Nox to educate users on wine-related topics and provide them with a more comprehensive understanding of wine.

In conclusion, expanding the wine database, collaborating with other vineyards, and including a wine wiki feature are all strategies that can enhance Chateau Du Nox's appeal to oenophiles. These enhancements will improve the user experience, broaden the application's reach, and ultimately increase sales and profits.

8.0 References

Anokhin, D. (2019, January 30). Exploring view hierarchy. *Medium.* Retrieved from: https://dmytro-anokhin.medium.com/exploring-view-hierarchy-332ea63262e9

Assumani, M. (2019, May 17). An introduction to CoreData: Intro to the low-level theory of Apple’s CoreData framework. *Medium.* Retrieved from: https://betterprogramming.pub/a-light-intro-to-core-data-part-un-e344f9d1528

Cacoo. (2021, December 23). How a UML use case diagram can benefit any process. *Nulab.* Retrieved from: https://nulab.com/learn/software-development/how-a-uml-use-case-diagram-can-benefit-any-process/#:~:text=The%20greatest%20advantage%20of%20a,and%20serves%20the%20user's%20goals.

DevTechie. (2022, August 26). ZStack in SwiftUI. *Medium.* Retrieved from: https://medium.com/devtechie/zstack-in-swiftui-6764022ce51a

Fox, E. (2017, February 23). UI/UX design before development is important. *PMG Digital Made for Humans.* Retrieved from: https://www.pmg.com/blog/uiux-design-development-important

Jacobs, B. (n.d.). SwiftUI fundamentals – what is SwiftUI. *Cocoacast.* Retrieved from: https://cocoacasts.com/swiftui-fundamentals-what-is-swiftui

Karhchyshyn, A. (2018, August 6). Core Location tutorial for iOS: Tracking visited locations. *Kodeco.* Retrieved from: https://www.kodeco.com/5247-core-location-tutorial-for-ios-tracking-visited-locations

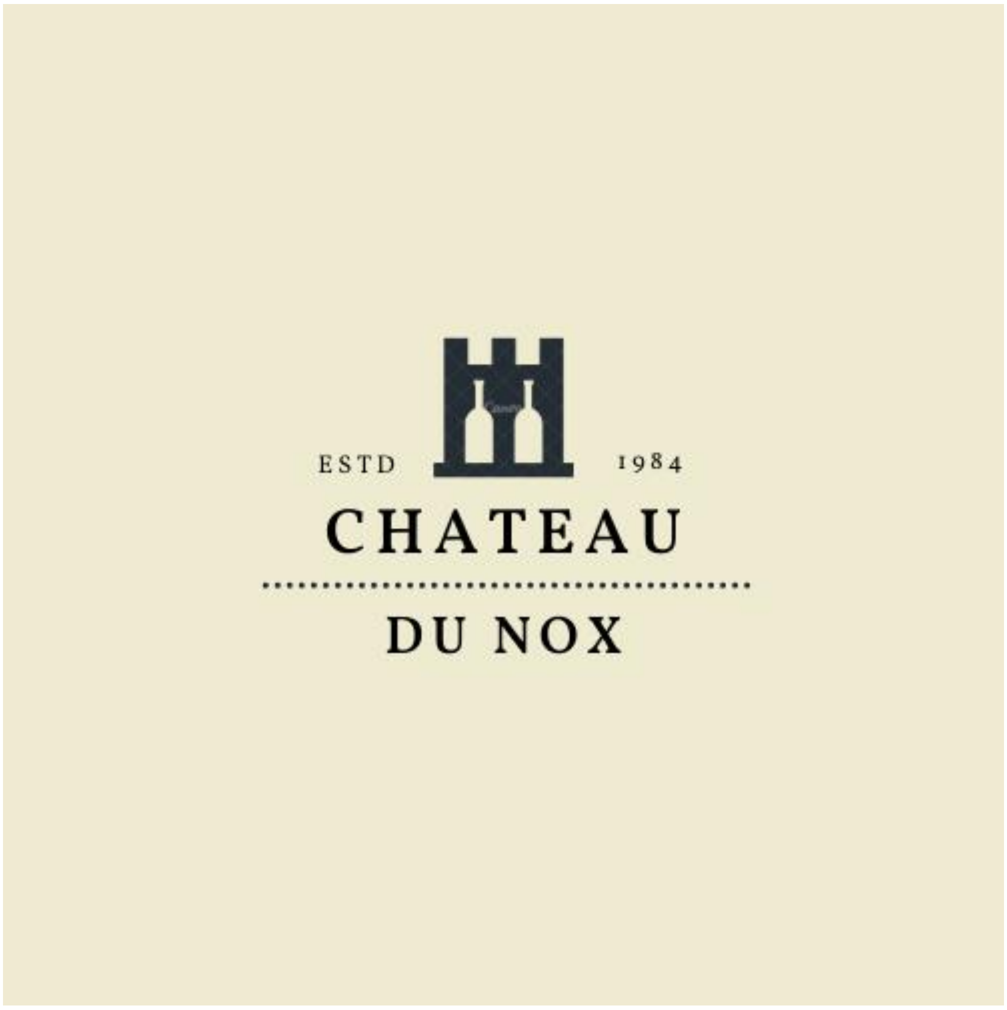
Smartvipere75. (2022, August 15). BottomBar-SwiftUI. *GitHub.* Retrieved from: https://github.com/smartvipere75/bottombar-swiftui

Sullivan, M. (2022, July 27). The importance of entity relationship diagrams. *Hubgem.* Retrieved from: https://blog.hubgem.co.uk/the-benefits-of-entity-relationship-diagrams

Tetlaw, A. (2020, April 1). MapKit tutorial: Getting started. *Kodeco.* Retrieved from: https://www.kodeco.com/7738344-mapkit-tutorial-getting-started

9.0 Appendix

9.1 Chateau Du Nox Logo Design


ESTD
1984
CHATEAU
DU NOX