Go Advanced Assignment Report

Title: Home Baker Application

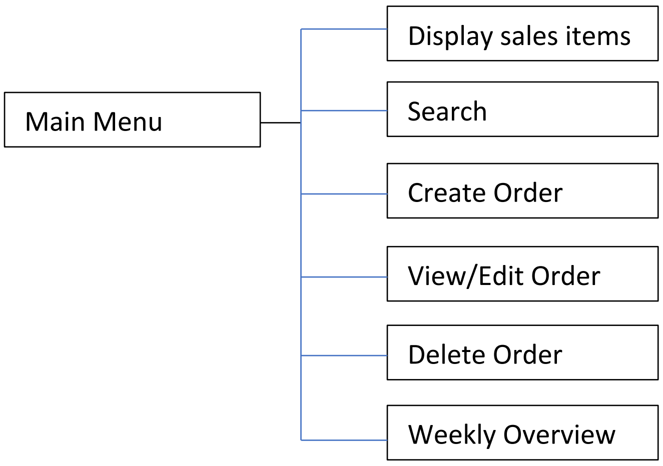
Submitted by: Tho Yan Bo

Application Overview

Home Baker is a simple application that is used to manage the orders of a home based bakery for the upcoming week (Monday to Sunday).

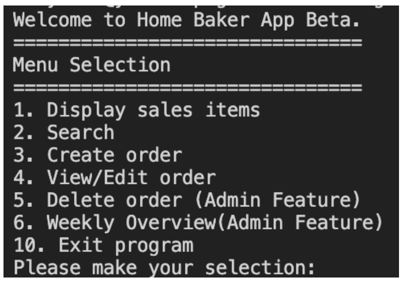
Application Features

The Home Baker application consists of the following features.



Main Menu Feature

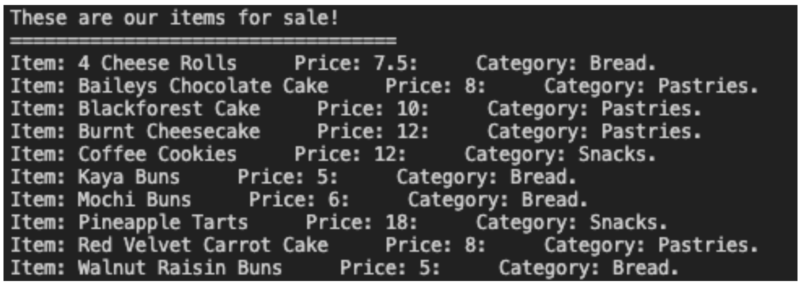
The main menu feature will welcome the user on runtime prompt and allows the user to make the following selections. Upon making a selection, the respective feature shall be executed. This main menu feature is a for loop that will only exit if user enters "10", which is the exit condition for this menu feature.



```
Welcome to Home Baker App Beta.
===============================
Menu Selection
===============================
1. Display sales items
2. Search
3. Create order
4. View/Edit order
5. Delete order (Admin Feature)
6. Weekly Overview(Admin Feature)
10. Exit program
Please make your selection:
```

1. Display Sales Items

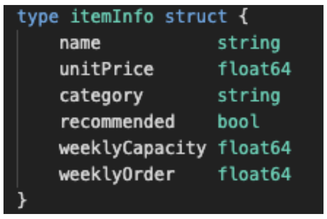
This feature will display all the items that are being sold by the home baker. The sales items are being stored in an **array based list** and are displayed based on alphabetical order due to the underlying function first sorting the array's items' names alphabetically before printing the item details based on array index.



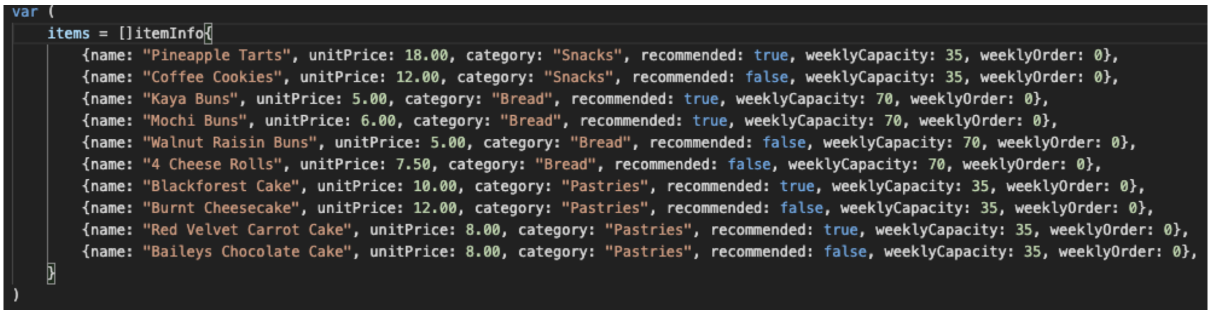
```
These are our items for sale!
==================================
Item: 4 Cheese Rolls     Price: 7.5:     Category: Bread.
Item: Baileys Chocolate Cake     Price: 8:     Category: Pastries.
Item: Blackforest Cake     Price: 10:     Category: Pastries.
Item: Burnt Cheesecake     Price: 12:     Category: Pastries.
Item: Coffee Cookies     Price: 12:     Category: Snacks.
Item: Kaya Buns     Price: 5:     Category: Bread.
Item: Mochi Buns     Price: 6:     Category: Bread.
Item: Pineapple Tarts     Price: 18:     Category: Snacks.
Item: Red Velvet Carrot Cake     Price: 8:     Category: Pastries.
Item: Walnut Raisin Buns     Price: 5:     Category: Bread.
```

1.1 Choice of array-based list for sales items

An array based list was chosen here as the number of sales items that the home baker puts up is relatively small and unlikely to change in quantity (assuming home baker does not have a dynamic item menu). Having the sales item set up as an array based list will allow for easy manipulation of the data list. Since the number of items on sale in this app are small (a total of 10 items), this would make the sorting algorithm (selection sort) appropriate due to the small array with a worst case efficiency case of O(n^2).



```go
type itemInfo struct {
    name           string
    unitPrice      float64
    category       string
    recommended    bool
    weeklyCapacity float64
    weeklyOrder    float64
}
```



```go
var (
    items = []itemInfo{
        {name: "Pineapple Tarts", unitPrice: 18.00, category: "Snacks", recommended: true, weeklyCapacity: 35, weeklyOrder: 0},
        {name: "Coffee Cookies", unitPrice: 12.00, category: "Snacks", recommended: false, weeklyCapacity: 35, weeklyOrder: 0},
        {name: "Kaya Buns", unitPrice: 5.00, category: "Bread", recommended: true, weeklyCapacity: 70, weeklyOrder: 0},
        {name: "Mochi Buns", unitPrice: 6.00, category: "Bread", recommended: true, weeklyCapacity: 70, weeklyOrder: 0},
        {name: "Walnut Raisin Buns", unitPrice: 5.00, category: "Bread", recommended: false, weeklyCapacity: 70, weeklyOrder: 0},
        {name: "4 Cheese Rolls", unitPrice: 7.50, category: "Bread", recommended: false, weeklyCapacity: 70, weeklyOrder: 0},
        {name: "Blackforest Cake", unitPrice: 10.00, category: "Pastries", recommended: true, weeklyCapacity: 35, weeklyOrder: 0},
        {name: "Burnt Cheesecake", unitPrice: 12.00, category: "Pastries", recommended: false, weeklyCapacity: 35, weeklyOrder: 0},
        {name: "Red Velvet Carrot Cake", unitPrice: 8.00, category: "Pastries", recommended: true, weeklyCapacity: 35, weeklyOrder: 0},
        {name: "Baileys Chocolate Cake", unitPrice: 8.00, category: "Pastries", recommended: false, weeklyCapacity: 35, weeklyOrder: 0},
    }
)
```

1.2 Items data list

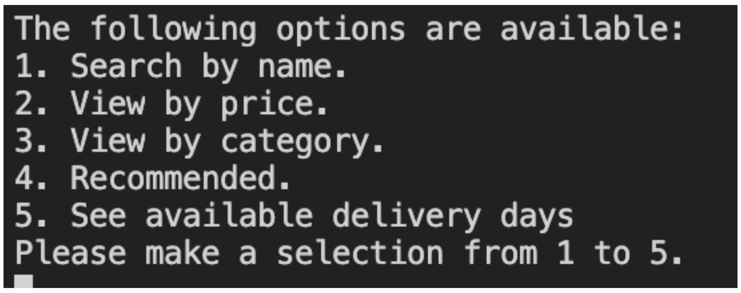
The data stored in the items data list is of itemInfo struct as follows:

name	string	Name of sales item
unitPrice	float64	Unit price of sales item
category	string	Category of sales item
recommended	bool	Boolean type for if item is recommended
weeklyCapacity	float64	Weekly baking capacity of the item
weeklyOrder	float64	Weekly order quantity (initialized at 0)

The above table shows the data variables stored within the itemInfo struct. The weekly baking capacity (weeklyCapacity) and orders (weeklyOrders) are stored into the struct for processing of item availability within the application.

2. Search

The ‘Search’ feature allows the user to narrow their search option via the following options.



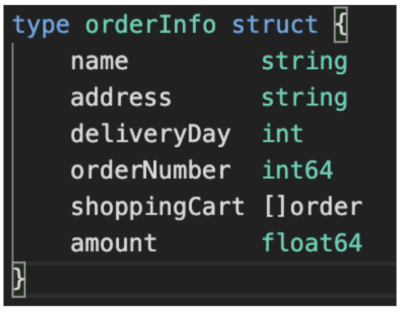
```
The following options are available:
1. Search by name.
2. View by price.
3. View by category.
4. Recommended.
5. See available delivery days
Please make a selection from 1 to 5.
```

1. ‘Search by name’ returns the sales item when matched with the exact sales item name (case sensitive).
2. ‘View by price’ sorts the array using selection sort algorithm and returns the full list from lowest to highest price.
3. ‘View by category’ sorts the array using selection sort algorithm and returns the full list in category.
4. ‘Search by recommended’ returns the recommended sales item based on the recommended Boolean value stored within the objects’ itemInfo struct.
5. ‘See available delivery’ days returns the available delivery days of the week (the maximum number of delivery per day has been pre-determined by the program).

3. Create order

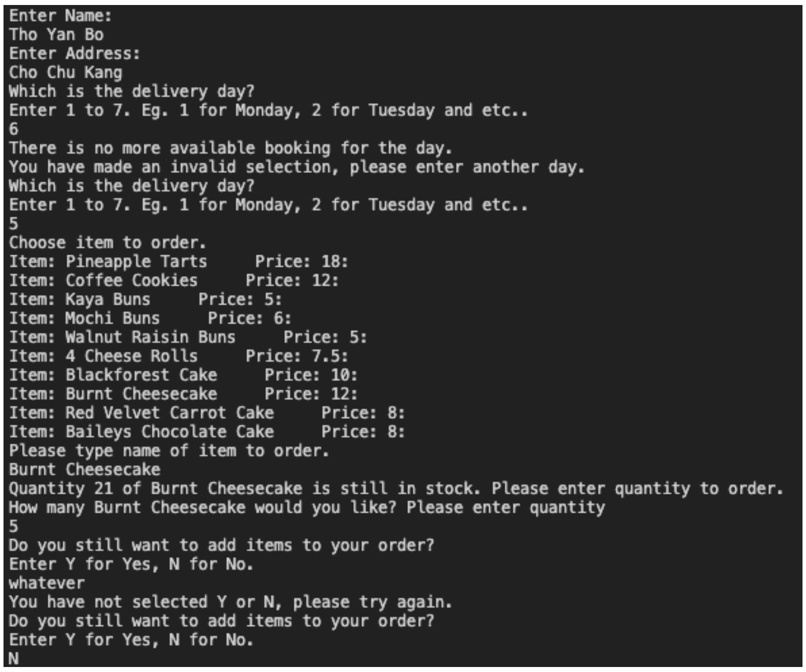
The 'Create order' feature allows the user to create a new order which will be added to an order data list. The order data structure is a pointer-based list (Linked List).

The feature will prompt the user to insert details required to create an order. See below screenshot for the information stored inside an orderInfo struct.



```
type orderInfo struct {
    name         string
    address      string
    deliveryDay  int
    orderNumber  int64
    shoppingCart []order
    amount       float64
}
```

User will be required to key in their name, address, preference of delivery day and items they wish to order. The orderNumber and amount values will be generated using program functions and automatically inserted into orderInfo as the order is created.



```
Enter Name:
Tho Yan Bo
Enter Address:
Cho Chu Kang
Which is the delivery day?
Enter 1 to 7. Eg. 1 for Monday, 2 for Tuesday and etc..
6
There is no more available booking for the day.
You have made an invalid selection, please enter another day.
Which is the delivery day?
Enter 1 to 7. Eg. 1 for Monday, 2 for Tuesday and etc..
5
Choose item to order.
Item: Pineapple Tarts     Price: 18:
Item: Coffee Cookies     Price: 12:
Item: Kaya Buns     Price: 5:
Item: Mochi Buns     Price: 6:
Item: Walnut Raisin Buns     Price: 5:
Item: 4 Cheese Rolls     Price: 7.5:
Item: Blackforest Cake     Price: 10:
Item: Burnt Cheesecake     Price: 12:
Item: Red Velvet Carrot Cake     Price: 8:
Item: Baileys Chocolate Cake     Price: 8:
Please type name of item to order.
Burnt Cheesecake
Quantity 21 of Burnt Cheesecake is still in stock. Please enter quantity to order.
How many Burnt Cheesecake would you like? Please enter quantity
5
Do you still want to add items to your order?
Enter Y for Yes, N for No.
whatever
You have not selected Y or N, please try again.
Do you still want to add items to your order?
Enter Y for Yes, N for No.
N
```

This feature has in-built functions to check if the delivery day is available and thus will prompt the user to select a different delivery day if the number of deliveries for the day has been maxed.

The name of items to order must also be entered correctly (case sensitive) otherwise the program will prompt the user to type in the correct name.

Also, the available quantity remaining for the week is also shown by the feature and the user is not allowed to order more than the remaining quantity.

Program feature will prompt user if they are done with order before finally recording the order into the order list.

3.1 Choice of pointer-based linked list for taking orders

A pointer-based list was chosen for the order list data structure due to the nature of the order list requiring constant addition to the list as new orders are being created. The size of a link list is not fixed and can grow as large as is necessary.

Also, throughout the application, the order list is used to obtain specific order data by search traversing through the linked list before obtaining the necessary data. Addition or deleting of the orders is relatively simple by update of pointers, but again traversing through the list is required for such operation. Furthermore, there is no requirement from the application to sort this pointer-based list which makes the selection of a linked list appropriate for the order list. The worst case complexity for a linked list is O(n) when traversing to the last node is required but this is still relatively manageable due to the order list being relatively small at maximum weekly capacity (maximum weekly order is 50).

3.2 Data stored in order list

The data stored in the order linked list is of type orderInfo struct as follows:

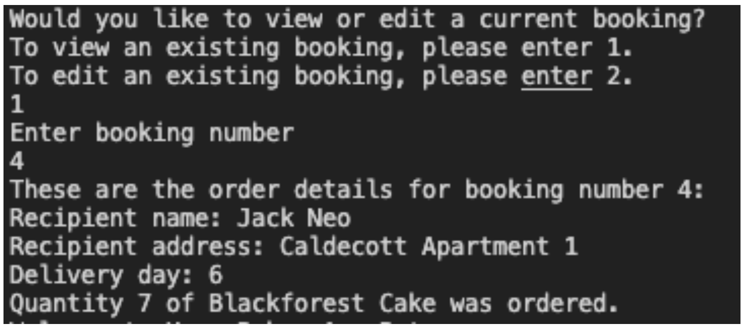
name	string	Name of customer
address	string	Address of customer
deliveryDay	int	Delivery day of order
orderNum	int64	Order number
shoppingCart	[]order	Shopping cart array of type struct order
amount	float64	Total amount of orders

The order struct records the following data:

orderItem	string	Name of item to be ordered
qty	int	Quantity of item ordered

4. View/Edit Order feature

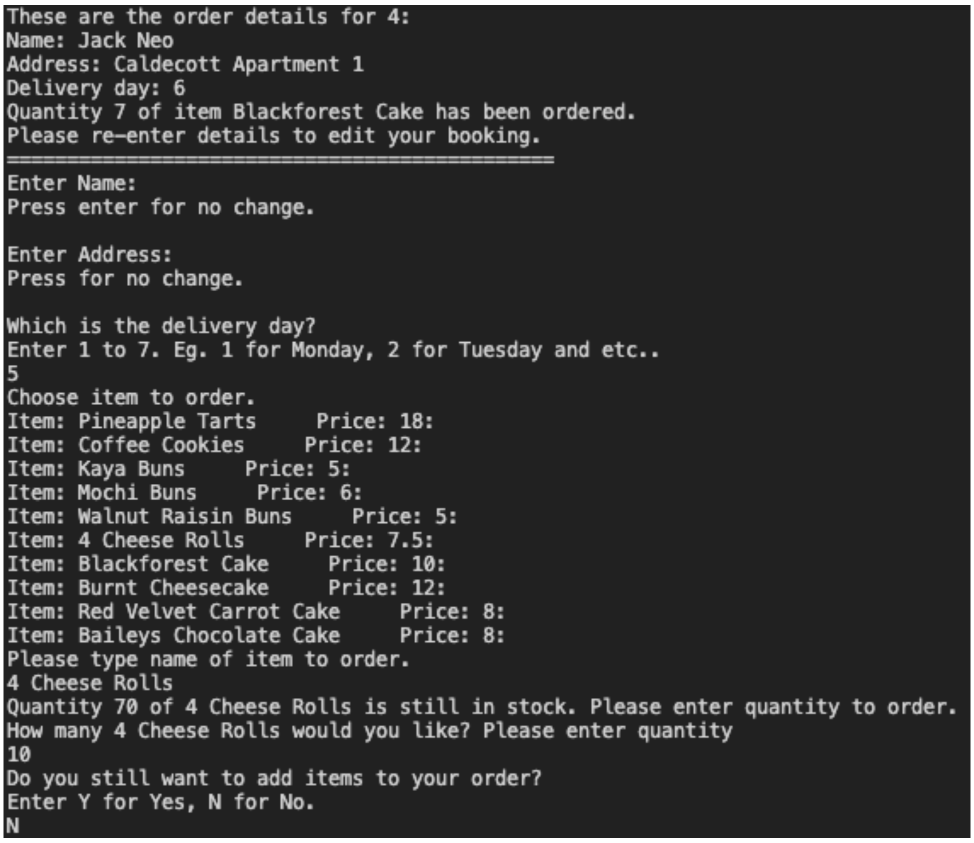
The 'View/Edit Order' feature would allow the user to view or edit orders based on the order number input. Based on selection of either view or edit order, the program will prompt the user for an order number input, before searching for the order via traversal through the order list to either view or modify the order details.



```
Would you like to view or edit a current booking?
To view an existing booking, please enter 1.
To edit an existing booking, please enter 2.
1
Enter booking number
4
These are the order details for booking number 4:
Recipient name: Jack Neo
Recipient address: Caldecott Apartment 1
Delivery day: 6
Quantity 7 of Blackforest Cake was ordered.
```

Screenshot of a view booking selection.

Selection of editing an existing booking will prompt user to re-enter booking details for modification.



```
These are the order details for 4:
Name: Jack Neo
Address: Caldecott Apartment 1
Delivery day: 6
Quantity 7 of item Blackforest Cake has been ordered.
Please re-enter details to edit your booking.
==============================================
Enter Name:
Press enter for no change.

Enter Address:
Press for no change.

Which is the delivery day?
Enter 1 to 7. Eg. 1 for Monday, 2 for Tuesday and etc..
5
Choose item to order.
Item: Pineapple Tarts     Price: 18:
Item: Coffee Cookies     Price: 12:
Item: Kaya Buns     Price: 5:
Item: Mochi Buns     Price: 6:
Item: Walnut Raisin Buns     Price: 5:
Item: 4 Cheese Rolls     Price: 7.5:
Item: Blackforest Cake     Price: 10:
Item: Burnt Cheesecake     Price: 12:
Item: Red Velvet Carrot Cake     Price: 8:
Item: Baileys Chocolate Cake     Price: 8:
Please type name of item to order.
4 Cheese Rolls
Quantity 70 of 4 Cheese Rolls is still in stock. Please enter quantity to order.
How many 4 Cheese Rolls would you like? Please enter quantity
10
Do you still want to add items to your order?
Enter Y for Yes, N for No.
N
```

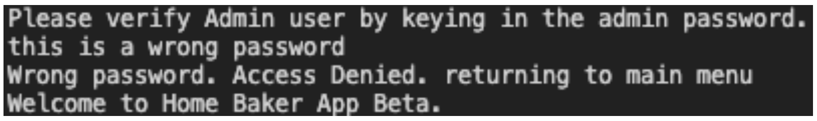
Screenshot of an edit booking selection.

5. Delete Order (Admin Feature)

The ‘delete order’ features allows the user to delete an order from the order list, provided they are authenticated to be an Admin user.

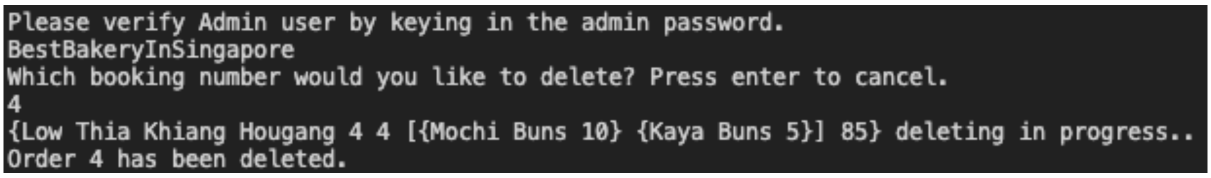
The password for this application is “BestBakeryInSingapore”.

Upon successful authentication, the user is able to delete an order which is performed through deleting of the pointer data in the order list, and reconnecting the next pointer address back into the order list.



```
Please verify Admin user by keying in the admin password.
this is a wrong password
Wrong password. Access Denied. returning to main menu
Welcome to Home Baker App Beta.
```

Example of wrong password entered.



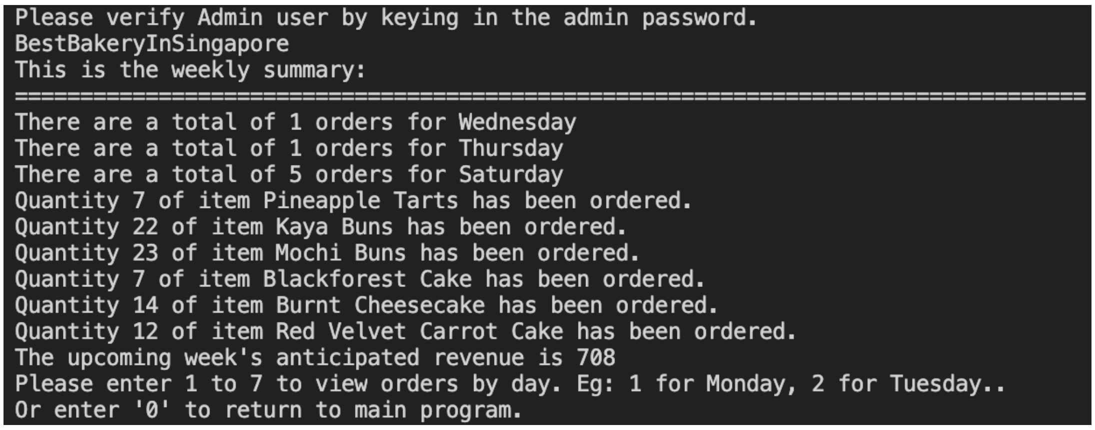
```
Please verify Admin user by keying in the admin password.
BestBakeryInSingapore
Which booking number would you like to delete? Press enter to cancel.
4
{Low Thia Khiang Hougang 4 4 [{Mochi Buns 10} {Kaya Buns 5}] 85} deleting in progress..
Order 4 has been deleted.
```

Example of successful deletion of order.

6. Weekly Overview

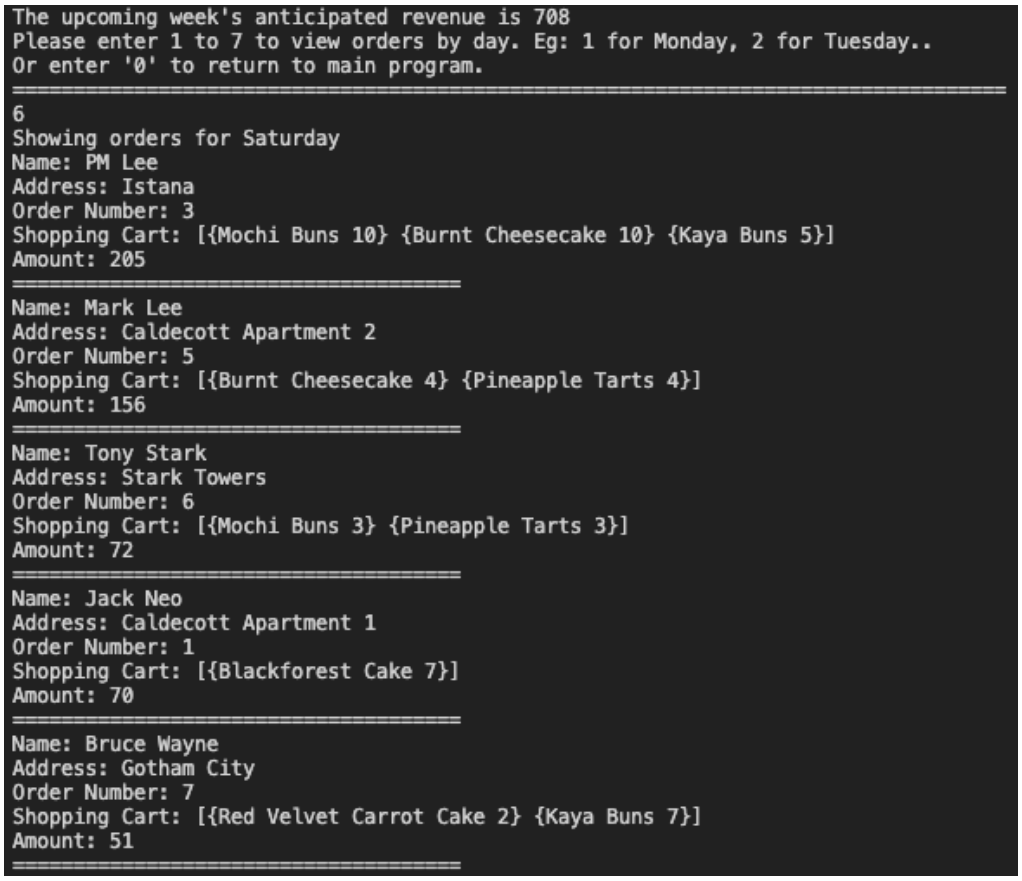
The weekly overview feature is an admin access feature which allows the admin user to have a summarized view for the week's upcoming orders.

It shows the breakdown of the number of orders by day and the number of orders by item, and calculates the expected week's order revenue amount.



```
Please verify Admin user by keying in the admin password.
BestBakeryInSingapore
This is the weekly summary:
================================================================================
There are a total of 1 orders for Wednesday
There are a total of 1 orders for Thursday
There are a total of 5 orders for Saturday
Quantity 7 of item Pineapple Tarts has been ordered.
Quantity 22 of item Kaya Buns has been ordered.
Quantity 23 of item Mochi Buns has been ordered.
Quantity 7 of item Blackforest Cake has been ordered.
Quantity 14 of item Burnt Cheesecake has been ordered.
Quantity 12 of item Red Velvet Carrot Cake has been ordered.
The upcoming week's anticipated revenue is 708
Please enter 1 to 7 to view orders by day. Eg: 1 for Monday, 2 for Tuesday..
Or enter '0' to return to main program.
```

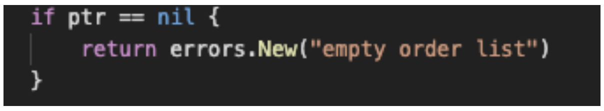
It also gives the user the option to view the orders of any specific day by keying in the day to view orders.



```
The upcoming week's anticipated revenue is 708
Please enter 1 to 7 to view orders by day. Eg: 1 for Monday, 2 for Tuesday..
Or enter '0' to return to main program.
================================================================================
6
Showing orders for Saturday
Name: PM Lee
Address: Istana
Order Number: 3
Shopping Cart: [{Mochi Buns 10} {Burnt Cheesecake 10} {Kaya Buns 5}]
Amount: 205
====================================
Name: Mark Lee
Address: Caldecott Apartment 2
Order Number: 5
Shopping Cart: [{Burnt Cheesecake 4} {Pineapple Tarts 4}]
Amount: 156
====================================
Name: Tony Stark
Address: Stark Towers
Order Number: 6
Shopping Cart: [{Mochi Buns 3} {Pineapple Tarts 3}]
Amount: 72
====================================
Name: Jack Neo
Address: Caldecott Apartment 1
Order Number: 1
Shopping Cart: [{Blackforest Cake 7}]
Amount: 70
====================================
Name: Bruce Wayne
Address: Gotham City
Order Number: 7
Shopping Cart: [{Red Velvet Carrot Cake 2} {Kaya Buns 7}]
Amount: 51
====================================
```

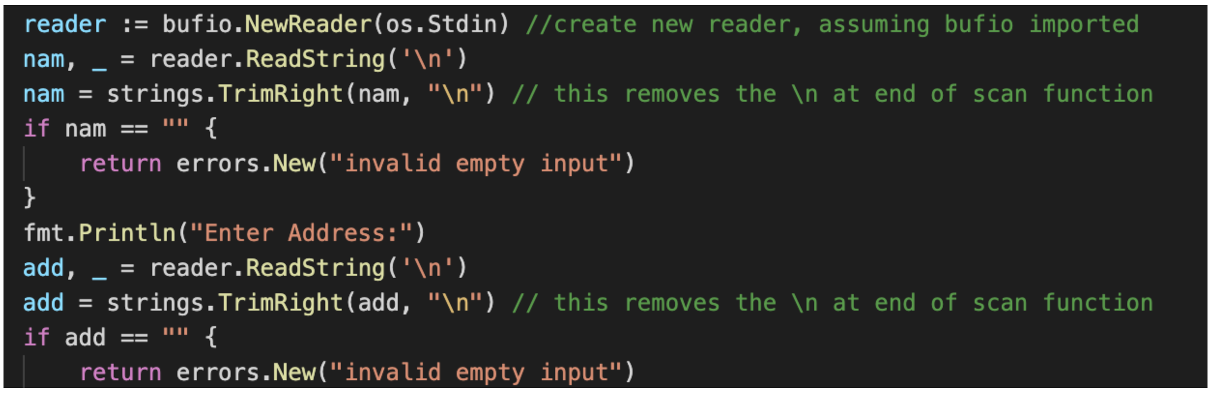
7. Error handling and panic recovery mechanisms

In general, most functions in this application have been coded with error handling mechanism in mind. In fact, most features that require user inputs such as order item name and order delivery day have been built with recursive functions to request that user enters a valid input before allowing the program to proceed. This minimizes the risk of errors occurring and helps the program to run more smoothly.



```go
if ptr == nil {
    return errors.New("empty order list")
}
```

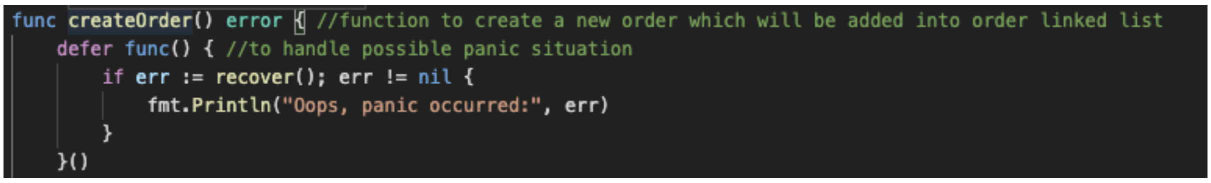
Example of an error returning if order linked list is empty



```go
reader := bufio.NewReader(os.Stdin) //create new reader, assuming bufio imported
nam, _ = reader.ReadString('\n')
nam = strings.TrimRight(nam, "\n") // this removes the \n at end of scan function
if nam == "" {
    return errors.New("invalid empty input")
}
fmt.Println("Enter Address:")
add, _ = reader.ReadString('\n')
add = strings.TrimRight(add, "\n") // this removes the \n at end of scan function
if add == "" {
    return errors.New("invalid empty input")
```

Example of an error returning if entries are empty

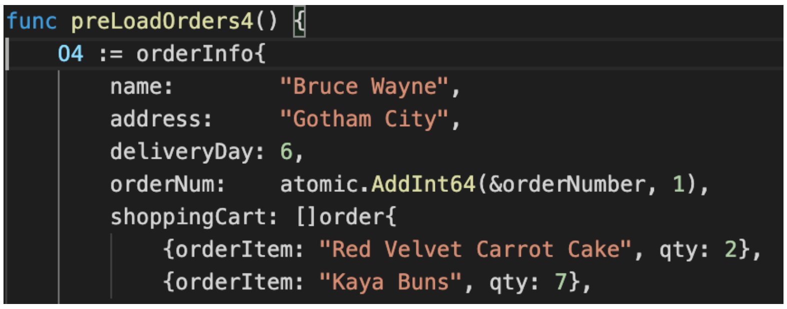
Also, basic panic recovery functions are included in features such as create order, edit order and delete order where multiple functions are called upon internally and have higher risk of a panic occurring should errors occur in those functions.



```go
func createOrder() error { //function to create a new order which will be added into order linked list
    defer func() { //to handle possible panic situation
        if err := recover(); err != nil {
            fmt.Println("Oops, panic occurred:", err)
        }
    }()
```

8. Concurrency and Goroutines

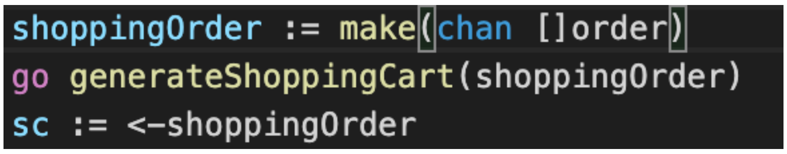
Goroutines are used at the start of the program to load multiple user orders into the program. This is done using preLoadOrders() function at the start of program to trigger multiple Goroutines to insert various orders into the order linked list. Since order number (global variable declaration orderNumber) is required for each individual order and each go routine will race to read and write with orderNumber, the atomic function is used to increment this global variable so as to ensure only 1 Goroutine can access this variable at any single time.



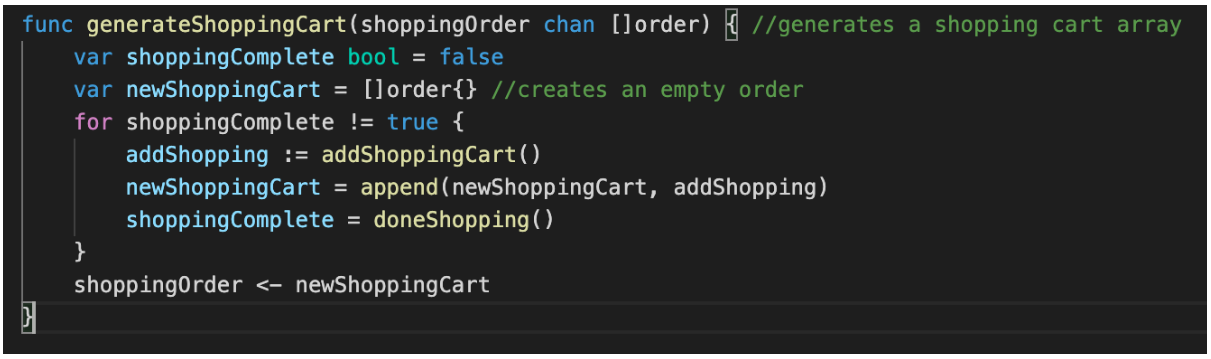
```go
func preLoadOrders4() {
    o4 := orderInfo{
        name:        "Bruce Wayne",
        address:     "Gotham City",
        deliveryDay: 6,
        orderNum:    atomic.AddInt64(&orderNumber, 1),
        shoppingCart: []order{
            {orderItem: "Red Velvet Carrot Cake", qty: 2},
            {orderItem: "Kaya Buns", qty: 7},
```

Example of atomic function being used to write over orderNum

Channels are also used in 'Create Order' and 'Edit Order' features to permit a separate Goroutine (generate shopping cart) to process the creation of new shopping cart (type order array). The new shopping cart that is created in this Goroutine is made synchronous with the main function through use of channel communication. As such, the function that triggers the GoRoutine will wait to receive the shopping order (type order array) before it is able to continue processing the remaining lines of code in the function. It is not really required for a separate Goroutine to be created in this case but nonetheless this was done to demonstrate how channels can be used to achieve concurrency and synchronous interaction between the Goroutines.



```go
shoppingOrder := make(chan []order)
go generateShoppingCart(shoppingOrder)
sc := <-shoppingOrder
```



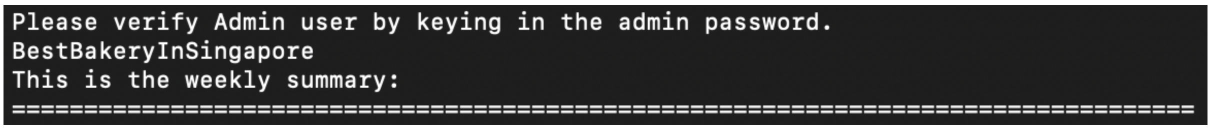
```go
func generateShoppingCart(shoppingOrder chan []order) { //generates a shopping cart array
    var shoppingComplete bool = false
    var newShoppingCart = []order{} //creates an empty order
    for shoppingComplete != true {
        addShopping := addShoppingCart()
        newShoppingCart = append(newShoppingCart, addShopping)
        shoppingComplete = doneShopping()
    }
    shoppingOrder <- newShoppingCart
}
```

Instructions to run program

A simple text based unix executable terminal has been created for this application (goAdvancedAssignment).

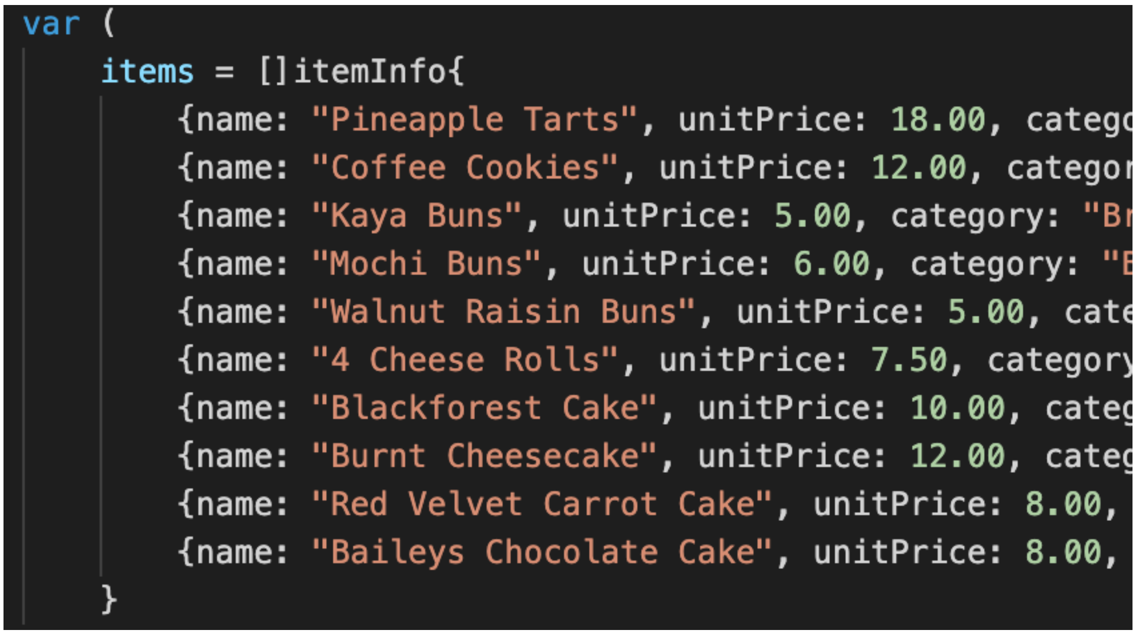
When the program is executed, the program will be pre-loaded with 7 orders into the home baker program. 5 order have been submitted into Saturday and thus new orders cannot be created on Saturday (this is also used to test if the program allows Saturday order creation).

Features 5 and 6 of the main menu are admin accessible features and will require a password entry to proceed. **The admin password for this program is 'BestBakeryInSingapore'.**



```
Please verify Admin user by keying in the admin password.
BestBakeryInSingapore
This is the weekly summary:
====================================================================================
```

Navigation within the program will be prompted with text instructions from the program. Do note that for text based entries when creating or editing an order, the item names must match an existing sales item name for the program to proceed, otherwise it will return an error. Please refer to screenshot below for the list of item names when using the program.



```
var (
    items = []itemInfo{
        {name: "Pineapple Tarts", unitPrice: 18.00, categ
        {name: "Coffee Cookies", unitPrice: 12.00, catego
        {name: "Kaya Buns", unitPrice: 5.00, category: "Br
        {name: "Mochi Buns", unitPrice: 6.00, category: "B
        {name: "Walnut Raisin Buns", unitPrice: 5.00, cate
        {name: "4 Cheese Rolls", unitPrice: 7.50, category
        {name: "Blackforest Cake", unitPrice: 10.00, categ
        {name: "Burnt Cheesecake", unitPrice: 12.00, categ
        {name: "Red Velvet Carrot Cake", unitPrice: 8.00,
        {name: "Baileys Chocolate Cake", unitPrice: 8.00,
    }
```

Names entered when creating order must match name variables defined in program (case sensitive).