Command GoInAction2Assignment

Package main contains the main() func to run the application.

Subdirectories

Name	Synopsis
..	
pkg	
order	Package order implements a Queue data structure to enqueue/dequeue orders in a FIFO order.
pizza	Package pizza implements a LinkedList data structure to add/edit/delete pizzas.
server	Package server implements all the handlers functions and is separated into 5 go files to segregate the functionalities of the application.

Directory /src/GoInAction2Assignment/pkg

Name	Synopsis
..	
order	Package order implements a Queue data structure to enqueue/dequeue orders in a FIFO order.
pizza	Package pizza implements a LinkedList data structure to add/edit/delete pizzas.
server	Package server implements all the handlers functions and is separated into 5 go files to segregate the functionalities of the application.

Package order

```
import "GoInAction2Assignment/pkg/order"
```

Overview
Index

Overview ▾

Package order implements a Queue data structure to enqueue/dequeue orders in a FIFO order.

Index ▾

func CalOrderTotal(pizzaList *pizza.Linkedlist, orderSlice []OrderItem) float64
func GenerateOrderNo(orderNo int, orderSlice []OrderItem, ch chan<- int)
func PrintOrderReceipt(pizzaList *pizza.Linkedlist, orderNo int, orderSlice []OrderItem, totalCost float64)
func SearchPizzaInSlice(pizzaNo int, orderSlice []OrderItem) bool
func printDividerLine()
type Node
type Order
type OrderItem
type Queue
    func (p *Queue) Dequeue(orderChannel chan<- Order)
    func (p *Queue) Enqueue(orderNo int, orderSlice []OrderItem, totalCost float64, userName string) error
    func (p *Queue) GetAllOrders(userName string, isAdmin bool) ([]Order, error)
    func (p *Queue) IsEmpty() bool
    func (p *Queue) SearchOrder(orderNo int) (Order, error)
    func (p *Queue) SearchPizzaInOrder(pizzaNo int) (bool, error)
    func (p *Queue) UpdateOrder(orderNo int, orderSlice []OrderItem, totalCost float64, wg *sync.WaitGroup, mutex *sync.Mutex)

Package files

order.go orderQueue.go

func **CalOrderTotal**

```
func CalOrderTotal(pizzaList *pizza.Linkedlist, orderSlice []OrderItem) float64
```

CalOrderTotal calculates the total amount of an order

func **GenerateOrderNo**

```
func GenerateOrderNo(orderNo int, orderSlice []OrderItem, ch chan<- int)
```

GenerateOrderNo takes []OrderItem slice and checks if any OrderItem exists. If so, it will increment the orderNo by 1. A channel is use to receive the orderNo prevent multiple order no being generated at the same time

func **PrintOrderReceipt**

```
func PrintOrderReceipt(pizzaList *pizza.Linkedlist, orderNo int, orderSlice []OrderItem, totalCost float64)
```

PrintOrderReceipt prints the receipt of the order made on the server terminal/cmd prompt

func **SearchPizzaInSlice**

```
func SearchPizzaInSlice(pizzaNo int, orderSlice []OrderItem) bool
```

SearchPizzaInSlice takes in the pizzaNo and orderSlice as parameters and return true if pizzaNo is found in the slice. Otherwise, returns false.

func **printDividerLine**

```
func printDividerLine()
```

printDividerLine prints a divider line to segregate sections for easy viewing when printing on the terminal/cmd prompt

type **Node**

Node item for the Queue is an Order struct

```
type Node struct {
    Item Order
    Next *Node
}
```

type **Order**

Define an Order struct. OrderSlice can contain more than 1 OrderItem

```
type Order struct {
    OrderNo    int
    OrderSlice []OrderItem
    TotalCost  float64
    UserName   string
}
```

type **OrderItem**

Define an OrderItem struct

```
type OrderItem struct {
    PizzaNo  int
    OrderQty int
}
```

type **Queue**

Queue struct for orders

```
type Queue struct {
    Front *Node
    Back  *Node
    Size  int
}
```

func (*Queue) Dequeue

```
func (p *Queue) Dequeue(orderChannel chan<- Order)
```

Dequeue removes an order from the queue

func (*Queue) Enqueue

```
func (p *Queue) Enqueue(orderNo int, orderSlice []OrderItem, totalCost float64, userName string) error
```

Enqueue adds an order to the queue

func (*Queue) GetAllOrders

```
func (p *Queue) GetAllOrders(userName string, isAdmin bool) ([]Order, error)
```

GetAllOrders appends the current node Order item that belongs to a user into an Order slice. Admin user is allowe to get all the orders.

func (*Queue) IsEmpty

```
func (p *Queue) IsEmpty() bool
```

IsEmpty return true/false if the LinkedList is empty or not

func (*Queue) SearchOrder

```
func (p *Queue) SearchOrder(orderNo int) (Order, error)
```

SearchOrder finds the Order struct in the LinkedList node item. It then returns the Order item.

func (*Queue) SearchPizzaInOrder

```
func (p *Queue) SearchPizzaInOrder(pizzaNo int) (bool, error)
```

SearchPizzaInOrder finds the current node's OrderSlice and call the func SearchPizzaInSlice using pizzaNo given as parameters and returns true if found and false otherwise.

func (*Queue) UpdateOrder

```
func (p *Queue) UpdateOrder(orderNo int, orderSlice []OrderItem, totalCost float64, wg *sync.WaitGroup, mutex *sync.Mutex)
```

UpdateOrder updates the current node item that matches the orderNo given with a new orderSlice and totalCost.

Package pizza

```
import "GoInAction2Assignment/pkg/pizza"
```

Overview
Index

Overview ▾

Package pizza implements a LinkedList data structure to add/edit/delete pizzas.

Index ▾

type Linkedlist
  func (p *Linkedlist) AddPizza(pizzaNo int, pizzaName string, pizzaPrice float64) error
  func (p *Linkedlist) CreateStartMenu(standardPizza []string, standardPrice float64) error
  func (p *Linkedlist) DeletePizza(pizzaNo int) error
  func (p *Linkedlist) EditPizza(pizzaNo int, pizzaName string, pizzaPrice float64) error
  func (p *Linkedlist) GetAllPizza() ([]Pizza, error)
  func (p *Linkedlist) GetAt(pos int) *Node
  func (p *Linkedlist) SearchPizza(pizzaNo int) (Pizza, error)
type Node
type Pizza

Package files

pizzaList.go

type **Linkedlist**

LinkedList struct for the pizza menu

```
type Linkedlist struct {
    Head *Node
    Size int
}
```

func (*Linkedlist) AddPizza

```
func (p *Linkedlist) AddPizza(pizzaNo int, pizzaName string, pizzaPrice float64) error
```

AddPizza creates a Pizza struct which is then added to the LinkedList node item.

func (*Linkedlist) CreateStartMenu

```
func (p *Linkedlist) CreateStartMenu(standardPizza []string, standardPrice float64) error
```

CreateStartMenu creates a standard pizza menu.

func (*Linkedlist) DeletePizza

```
func (p *Linkedlist) DeletePizza(pizzaNo int) error
```

DeletePizza remove the node in the LinkedList where pizzaNo is found.

func (*Linkedlist) EditPizza

```
func (p *Linkedlist) EditPizza(pizzaNo int, pizzaName string, pizzaPrice float64) error
```

EditPizza updates the Pizza struct in the LinkedList node item.

func (*Linkedlist) GetAllPizza

```
func (p *Linkedlist) GetAllPizza() ([]Pizza, error)
```

GetAllPizza finds all the Pizza struct in the LinkedList node item and appends it to a pizzaSlice. It returns the pizzaSlice.

func (*Linkedlist) GetAt

```
func (p *Linkedlist) GetAt(pos int) *Node
```

GetAt finds the position where a node is located and returns the node pointer.

func (*Linkedlist) SearchPizza

```
func (p *Linkedlist) SearchPizza(pizzaNo int) (Pizza, error)
```

SearchPizza finds the Pizza struct in the LinkedList node item. It then returns the Pizza item.

type **Node**

Node item for the LinkedList is a Pizza struct

```
type Node struct {
    Item Pizza
    Next *Node
}
```

type **Pizza**

Define a Pizza struct

```
type Pizza struct {
    PizzaNo    int
    PizzaName  string
    PizzaPrice float64
}
```

Package server

```
import "GoInAction2Assignment/pkg/server"
```

Overview
Index

Overview ▾

Package server implements all the handlers functions and is separated into 5 go files to segregate the functionalities of the application.

```
server.go: Initialises the application variables and starts the server.

indexHandler.go: Manages the index page and implements the functionalities for user logins.

orderHandler.go: Implements the functionalities to manage orders.

pizzaHandler.go: Implements the functionalities to manage pizzas.

userHandler.go: Implements the functionalities to manage users.
```

Index ▾

Variables
func InitServer()
func StartServer()
func addOrder(orderSlice []order.OrderItem, pizzaNo int, orderQty int) []order.OrderItem
func addorder(res http.ResponseWriter, req *http.Request)

```
func addpizza(res http.ResponseWriter, req *http.Request)
func alreadyLoggedIn(req *http.Request) bool
func calOrderTotal(orderSlice []order.OrderItem) float64
func calTotalSales(viewPizzaSalesSlice []viewPizzaSales) float64
func checkPizzaInOrder(pizzaNo int) (bool, error)
func completeorder(res http.ResponseWriter, req *http.Request)
func deletepizza(res http.ResponseWriter, req *http.Request)
func deleteuser(res http.ResponseWriter, req *http.Request)
func editorder(res http.ResponseWriter, req *http.Request)
func editpizza(res http.ResponseWriter, req *http.Request)
func edituser(res http.ResponseWriter, req *http.Request)
func generateOrderNo(orderSlice []order.OrderItem)
func generatePizzaNo()
func getPizzaSales(viewOrderSlice []viewOrder, ch chan<- []viewPizzaSales)
func index(res http.ResponseWriter, req *http.Request)
func isValidEmail(e string) bool
func logout(res http.ResponseWriter, req *http.Request)
func pizzasales(res http.ResponseWriter, req *http.Request)
func printDividerLine()
func printOrderReceipt(orderNo int, orderSlice []order.OrderItem, totalCost float64)
func signup(res http.ResponseWriter, req *http.Request)
func updateCompletedOrders(completedOrder order.Order, myUser user)
func updateLoginDate(myUser user)
func validatePassword(password string) error
func validatePrice(price string) (float64, error)
func validateQuantity(qty string) (int, error)
func validateUserInput(userName string, password string, cmfPassword string, firstName string, lastName string, email string) error
func vieworders(res http.ResponseWriter, req *http.Request)
func viewpizza(res http.ResponseWriter, req *http.Request)
type user
    func getUser(res http.ResponseWriter, req *http.Request) user
type viewOrder
    func getCompletedOrders(userName string, isAdmin bool) []viewOrder
    func getCurrentOrders(userName string, isAdmin bool) ([]viewOrder, error)
type viewOrderItem
type viewPizza
type viewPizzaSales
    func updatePizzaInSlice(vOrderItem viewOrderItem, viewPizzaSalesSlice []viewPizzaSales) []viewPizzaSales
```

Package files

indexHandler.go orderHandler.go pizzaHandler.go server.go userHandler.go

Variables

Global variables

```
var (
    wg    sync.WaitGroup
    mutex sync.Mutex

    tpl         *template.Template
    mapUsers    = map[string]user{}
    mapSessions = map[string]string{}

    log  = logrus.New()
    file *os.File

    newPizzaNo    int     // To generate a new Pizza No
    newOrderNo    int     // To generate a new Order No
    maxOrderQty   int     // Set the max order quantity
    standardPrice float64 // Set the standard price of a pizza

    minUserName int // Set the min length for new Username
    maxUserName int // Set the max length for new Username
    minPassword int // Set the min length for new Password
    maxPassword int // Set the max length for new Password

    bFirst = true

    // Create an empty LinkedList
    pizzaList = &pizza.Linkedlist{
        Head: nil,
        Size: 0,
```

```
    }

    // Create an empty Queue
    orderQueue = &order.Queue{
        Front: nil,
        Back:  nil,
        Size:  0,
    }

    // Create an empty []viewOrder slice. Use for displaying completed orders.
    completedOrderSlice = make([]viewOrder, 0)
)
```

func **InitServer**

```
func InitServer()
```

InitServer will start the required workflow before server starts. It will complete all initialisation and run only once. First it will parse templates. Then it will open/create the file for logging. After which, it will load variables from .env and initialise the global variables. It will then create an admin user and a standard pizza menu for testing purpose.

func **StartServer**

```
func StartServer()
```

StartServer initialise all the handler func then listen and start the server on the given port using https. At the end, it will close the logging file.

func **addOrder**

```
func addOrder(orderSlice []order.OrderItem, pizzaNo int, orderQty int) []order.OrderItem
```

addOrder takes in the OrderItem slice, a pizzaNo and orderQty. It creates an OrderItem with the pizzaNo and orderQty, then adds it to the slice.

func **addorder**

```
func addorder(res http.ResponseWriter, req *http.Request)
```

addorder is a handler func to add a new order. Redirects to index page if user has not login.

func **addpizza**

```
func addpizza(res http.ResponseWriter, req *http.Request)
```

addpizza is a handler func to add a new pizza. Redirects to index page if user has not login.

func **alreadyLoggedIn**

```
func alreadyLoggedIn(req *http.Request) bool
```

alreadyLoggedIn func checks if a user has already logged in. Checks for valid session token. Returns true if already logged in, false otherwise.

func **calOrderTotal**

```
func calOrderTotal(orderSlice []order.OrderItem) float64
```

calOrderTotal calculates the total amount of an order

func **calTotalSales**

```
func calTotalSales(viewPizzaSalesSlice []viewPizzaSales) float64
```

calTotalSales calculates the total sales of all pizzas that were ordered

func **checkPizzaInOrder**

```
func checkPizzaInOrder(pizzaNo int) (bool, error)
```

checkPizzaInOrder checks whether a pizza exists in any orders. Returns true if found, otherwise returns false.

func **completeorder**

```
func completeorder(res http.ResponseWriter, req *http.Request)
```

completeorder is a handler func to display orders that are currently in the queue. It allows the admin user to dequeue the first order in the queue. Redirects to index page if user has not login.

func **deletepizza**

```
func deletepizza(res http.ResponseWriter, req *http.Request)
```

deletepizza is a handler func to delete an existing pizza. Selected pizza cannot be deleted if an order exists with it. Redirects to index page if user has not login.

func **deleteuser**

```
func deleteuser(res http.ResponseWriter, req *http.Request)
```

deleteuser is a handler func to delete user account. Redirects to index page if user has not login. Only admin user has access to delete users and admin is not allowed to delete oneself.

func **editorder**

```
func editorder(res http.ResponseWriter, req *http.Request)
```

editorder is a handler func to edit an existing order. Redirects to index page if user has not login.

func **editpizza**

```
func editpizza(res http.ResponseWriter, req *http.Request)
```

editpizza is a handler func to edit an existing pizza. Selected pizza cannot be edited if an order exists with it. Redirects to index page if user has not login.

func **edituser**

```
func edituser(res http.ResponseWriter, req *http.Request)
```

edituser is a handler func to edit user account information. Redirects to index page if user has not login. Validates user input and updates the information.

func **generateOrderNo**

```
func generateOrderNo(orderSlice []order.OrderItem)
```

generateOrderNo takes []OrderItem slice and checks if any OrderItem exists. If so, it will increment the global value of newOrderNo by 1. A mutex lock is implemented to prevent multiple orders being generated at the same time

func **generatePizzaNo**

```
func generatePizzaNo()
```

generatePizzaNo increments the global pizza no for new pizza creation

func **getPizzaSales**

```
func getPizzaSales(viewOrderSlice []viewOrder, ch chan<- []viewPizzaSales)
```

getPizzaSales takes in a channel and received a []viewPizzaSales slice

func **index**

```
func index(res http.ResponseWriter, req *http.Request)
```

index is a handler func that display the home page of the application. On start, it will default as the login page first. Once user login, the page will change to show the main menu for the users. If user is an admin, it will display the admin menu as well.

func **isValidEmail**

```
func isValidEmail(e string) bool
```

isValidEmail validates if the string parameter is a valid email using regexp

func **logout**

```
func logout(res http.ResponseWriter, req *http.Request)
```

logout func is a handler to logout the current user. Redirects to index page if user has not login. Otherwise, delete session token from server and client, then redirects to index page.

func **pizzasales**

```
func pizzasales(res http.ResponseWriter, req *http.Request)
```

pizzasales is a handler func to display the sales of all the pizzas, its total quantity and total cost Redirects to index page if user has not login.

func **printDividerLine**

```
func printDividerLine()
```

printDividerLine prints a divider line to segregate sections for easy viewing when printing on the terminal/cmd prompt

func **printOrderReceipt**

```
func printOrderReceipt(orderNo int, orderSlice []order.OrderItem, totalCost float64)
```

printOrderReceipt prints the receipt of the order made on the server terminal/cmd prompt

func **signup**

```
func signup(res http.ResponseWriter, req *http.Request)
```

signup is a handler func to create a new user account. Validates user information and creates a new user account.

func **updateCompletedOrders**

```
func updateCompletedOrders(completedOrder order.Order, myUser user)
```

updateCompletedOrders updates the global var for completedOrderSlice for display of orders that have been dequeued. Mutex lock and unlock is implemented for concurrency

func **updateLoginDate**

```
func updateLoginDate(myUser user)
```

updateLoginDate updates the LastLoginDT to previous CurrentLoginDT. Then updates the CurrentLoginDt to time.Now(). No changes to all other information.

func **validatePassword**

```
func validatePassword(password string) error
```

validatePassword validates that the input user password must contain as least one upper case, lower case, numeric and special characters.

func **validatePrice**

```
func validatePrice(price string) (float64, error)
```

validatePrice will parse the price string as float and returns the parsed value

func **validateQuantity**

```
func validateQuantity(qty string) (int, error)
```

validateQuantity will parse the qty string as int and returns the parsed value

func **validateUserInput**

```
func validateUserInput(userName string, password string, cmfPassword string, firstName string, lastName string, email
string) error
```

validateUserInput func checks if a user has already logged in. Checks for valid session token.

func **vieworders**

```
func vieworders(res http.ResponseWriter, req *http.Request)
```

vieworders is a handler func to view orders created by a user or view all orders if user is admin Redirects to index page if user has not login.

func **viewpizza**

```
func viewpizza(res http.ResponseWriter, req *http.Request)
```

viewpizza is a handler func to view all pizzas. Redirects to index page if user has not login.

type **user**

user struct for storing user account information

```
type user struct {
    UserName       string
    Password       []byte
    FirstName      string
    LastName       string
    Email          string
    IsAdmin        bool
    CreatedDT      time.Time
    LastModifiedDT time.Time
    CurrentLoginDT time.Time
```

```
    LastLoginDT    time.Time
}
```

func getUser

```
func getUser(res http.ResponseWriter, req *http.Request) user
```

getUser func gets the current user. Checks for valid session token. Add a new session token cookie to the client if one is not found. Return user struct if found.

type viewOrder

viewOrder is used for display in the html templates

```
type viewOrder struct {
    IdxNo          int
    OrderNo        int
    ViewOrderItems []viewOrderItem
    TotalCost      string
    UserName       string
}
```

func getCompletedOrders

```
func getCompletedOrders(userName string, isAdmin bool) []viewOrder
```

getCompletedOrders retrieves the completed orders into []viewOrder

func getCurrentOrders

```
func getCurrentOrders(userName string, isAdmin bool) ([]viewOrder, error)
```

getCurrentOrders retrieves the current orders into []viewOrder

type **viewOrderItem**

viewOrderItem is used for display in the html templates

```
type viewOrderItem struct {
    ItemNo     int
    PizzaNo    int
    PizzaName  string
    PizzaPrice string
    OrderQty   int
    Checked    string
    ErrorMsg   string
}
```

type **viewPizza**

viewPizza is used for display in the html templates

```
type viewPizza struct {
    PizzaNo     int
    PizzaName   string
    PizzaPrice  float64
    SPizzaPrice string
    Selected    string
}
```

type **viewPizzaSales**

viewPizzaSales is used for display in the html templates

```
type viewPizzaSales struct {
    PizzaNo     int
    PizzaName   string
    OrderQty    int
    TotalSales  float64
    STotalSales string
}
```

func updatePizzaInSlice

```
func updatePizzaInSlice(vOrderItem viewOrderItem, viewPizzaSalesSlice []viewPizzaSales) []viewPizzaSales
```

updatePizzaInSlice updates the total quantity and total sales of each type of pizzas that were ordered