Go In Action 2 - Rachel Tan

*new material added in from Go In Action 1

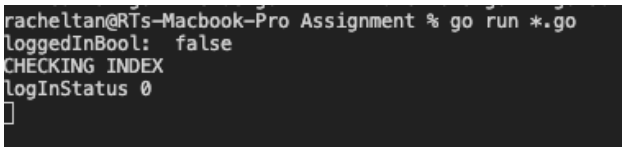
Description

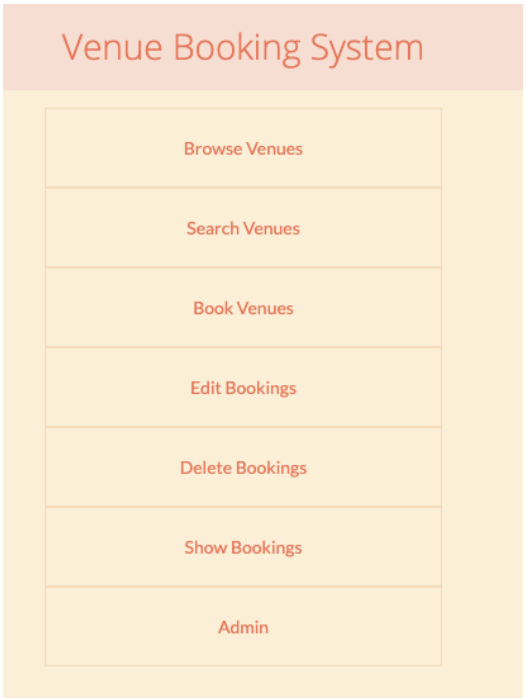
What's In the Zip File:

- Main.go - handler that spins up and handles the response and requests
- functionHelper.go - helper functions to perform processing within handlers
- Initializer.go - seed some test data (seeds venues and bookings)
- vApp - holds code from previous assignment
    - LinkedList.go - implementation of LinkedList to hold bookings nodes
    - Tree.go - implementation of AVLTree
    - Venue.go - holds methods to traverse through LinkedList
    - BookingNodes.go - holds struct for booking information
    - Functions - new methods added in this assignment
- Css - contains app.css for formatting
- Templates - gohtml files
- userDB.go - helper functions added to replace previous map with a mysql db

Running Application:

1. Go run *.go
2. Login → **username: "admin", pw: "1234"** to access the entire application
3. Signup → for any other username





Description of Application:

- Header: Venue Booking System : link to homepage /
    - Conditional Header
        - SignUp Login: if the user doesn't have a cookie it'll prompt login
        - UserName Logout: if user is logged in, the userName will be passed into header to indicate the user is logged in and can access certain features
- Menu:
    - Browse Venues: Open to everyone to view all the venues
    - Search Venues: Open to everyone to view, uses a form to capture what form of search choice:
        - Type
        - Capacity
        - Date Availability
    - Book Venues: have to be logged in, the booking will be using the username
    - Edit Bookings: have to be logged in, it will search for the venue and booking details and update with new booking information
    - Delete Bookings: have to be logged in, will ask for venue and booking information to be deleted
    - Show Bookings: open to everyone, it will run through the entire database to search for bookings
    - Admin: only available for admin
        - Add Venues: enter new information for a new venue
        - Delete Sessions: the server will run through the session data and display the users attached to the session to be deleted

        - Delete Users: the server will run through the user map and display the username to be picked
            - “admin” is protected from user deletion

Global Variables:

- myMap map[string]*AVLTree - holds the entire structure for venues and bookings
- capacityMap map[int]bool - a quick map to hold the available capacity for easier searching when going through the trees to look for a Venue
- mapUsers - mysql database that holds username and hashed password
- mapSession map[string]string{} - map of sessions to usernames that will tell if the user is logged in or not

Requirement Features

Security

SQL Injection

Queries are written with ? to prevent query injection for the user database.

```
results, _ := db.Query("SELECT * FROM VenueDB.Users where Username=?", username)
```

Password Hashing

Passwords are hashed using bcrypt and stored in the database. Upon logging in the password is verified using bcrypt’s CompareHashAndPassword.

```
bs, err := bcrypt.GenerateFromPassword([]byte(pw), bcrypt.MinCost)
err := bcrypt.CompareHashAndPassword([]byte(u.Password), []byte(pw))
```

AutoEscaping

html/template is chosen since it has automatic auto escaping to prevent html tags from being injected into the webpage.

```
"html/template"
```

Documentation

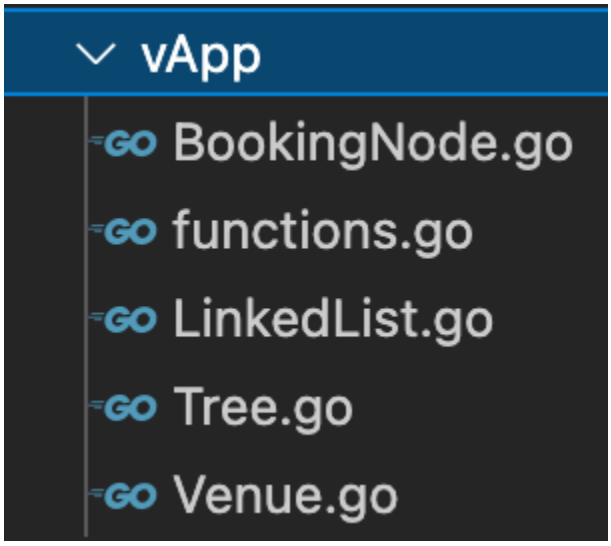
All .go files have comments explaining what that file holds.

```
/*
-------Main.go-------
host the main function
global variables are declared here
server is declared here as well as handlers that serve the pages
*/
```

A readme.md file has been added for additional commentary.

Idiomatic Go

Commentary
Each go file have been separated to demonstrate their grouping, Tree.go contains the AVL implementation and will contain all tree related functions. HelperFunctions contains all the functions that are used to bridge the gap between what was created in vApp and what the server needs to serve the application to the client.



Idiomatic Go General
All unused variables and packages are not included in the file. Unused returned variable are passed into a _ to be discarded, but it is usually error checking if there is a need to double check the error.

```
func deleteBooking(res http.ResponseWriter, req *http.Request) {
   req.ParseForm()
   m := make(map[string]interface{})
   logIn, userInfo := headerPasser(res, req, &m)
   if !logIn {
       m["notLoggedInError"] = "Please login to do this!"
       tpl.ExecuteTemplate(res, "index.gohtml", m)
   } else {
       if req.Method == "GET" {
           err := tpl.ExecuteTemplate(res, "delete.gohtml", nil)
           if err != nil {
               log.Fatalln(err)
           }
       } else {
```

```
            venueName := req.Form["venue"][0]
            capacity, _ := strconv.Atoi(req.Form["capacity"][0])
            venueType := req.Form["venuetype"][0]
            oldmonth, _ := strconv.Atoi(req.Form["month"][0])
            oldday, _ := strconv.Atoi(req.Form["day"][0])
            oldstarttime, _ := strconv.Atoi(req.Form["starttime"][0])
            oldendtime, _ := strconv.Atoi(req.Form["endtime"][0])
            foundVenue, err := findVenue(venueName, capacity, venueType)
```