Function Rooms Booking System For Co-working Space In Singapore

Version 3.0 → Updated with secure software practises, Idiomatic Go approaches and go documentation.

Created by
Oon Xin Ni
(Batch 2)
Go Advanced Assignment

Table of Contents

1. Overview

For this assignment, the function room booking system has been revised to Version 3 to include secure software development features, idiomatic go practices and go documentations. To have a quick recap, the application is a booking system application for function rooms of a coworking space. This application was revised previously to a server client application which allow 3 modes of accessing: 1. Guest, 2. Registered user, and 3. Admin. The application comes with the following features.

	Features	Admin	Registered user	Guest
1	Browse Venue Availability	✓	✓	✓
2.	Search Based on Criteria	✓	✓	✓
3.	Venue Booking	✓	✓	✗
4.	Browse My Booking	✗	✓	✗
5.	Browse Whole Booking List	✓	✗	✗
6.	Delete Booking	✓	✗	✗
7.	Login/ Sign Up	✓	✓	✓

Further details of every features can refer to Go Advance and Go in Action 1 report.

2. Secure software development features

2.1 Input Validation

All areas that allow user input poses security risk. By default, all users' inputs should be treated as insecure. For this assignment, there are few ways input validations are being done.

1. Front end input validation through html code

```
<form method="post">
    <label for ="date">Please enter your preferred date: (DDMM)</label><br>
    <input type="text" name="date" placeholder="Enter date (DDMM) "required pattern="[0-9]{2}[0-9]{2}"><br>
    <label for ="time">Please enter your preferred time:</label><br>
    <input type="text" name="time" placeholder="Enter time (eg:1300) "required pattern="[0-9]{2}[0]{2}"><br>
    <label for ="duration">Please enter duration of the event in hours:</label><br>
    <input type="text" name="duration" placeholder="Enter duration (hrs) "required pattern="[0-9]{1}"><br>
    <label for ="participantSize">Please enter total number of participant:</label><br>
    <input type="text" name="participantSize" placeholder="Enter no of participant"required pattern="[0-9]{1,3}"><br>
    <label for ="kind">Please enter preferred type of function room: (1, 2, 3)</label><br>
    <ol>
        <li> Meeting Room</li>
        <li> Activity Room</li>
        <li> Auditorium</li>
    </ol>
    <p>Leave blank if there is no preference</p>
    <input type="text" name="kind" placeholder="Enter preferred type"><br>
    <input type="submit">
```

2. Input validation using regexp/whitelisting exercise

```
dateS := user.Policy.Sanitize(strings.TrimSpace(req.FormValue("date")))
regexdateS := regexp.MustCompile(`^[0-3][0-9](0|1)[0-9]$`)
if !regexdateS.MatchString(dateS) {
    http.Error(res, "Invalid date selection, please try again.", http.StatusBadRequest)
    log.Info.Println("Booking, invalid date selection by", myUser.UserName)
    return
}
runes := []rune(dateS) // this is to take care of condition like 0802, without trimming, it will have issue
if runes[0] == '0' {
    dateS = strings.TrimLeft(dateS, "!0")
}
date, _ := strconv.Atoi(dateS)
```

3. Input validation through third party package (Go playground/validator/v10)

```
username := strings.ToLower(req.FormValue("username"))
err1 := validate.Var(username, "required,min=3,max=30,alphanum")
if err1 != nil {
    http.Error(res, "Username and/or password do not match", http.StatusForbidden)
    log.Warning.Println("Attempt to login with invalid username.-", err1)
    return
}
```

4. Some post validation actions have also been done for non-sensitive user data to ensure proper data cleaning before storing the data.
    a. Capitalize the first word of every name.
    b. Change the username to all lowercase.
    c. Trim any additional spaces before and after user desired inputs.
    d. Convert the numeric user input to integer to allow post-processing.

2.2 Sanitization of Input

Proper sanitization of user input after data validation is important to ensure there is no undesired malicious script that was inserted by the user. For input sanitization of this application:

1. Sanitization of user input after user validation was done using third party package, bluemonday. The package will sanitize user input against a whitelist of approved HTML element and attributes.

```
firstname = Policy.Sanitize(firstname)
lastname = Policy.Sanitize(lastname)
username = Policy.Sanitize(username)
password = Policy.Sanitize(password)
```

2.3 Output Encoding

Output encoding was done to prevent any unwanted/malicious html elements being executed.

1. Before execute the template, html/templates was used in this case to escape the unwanted character. The security model used by this package assumes that template authors are trusted, while Execute's data parameter is not. All data parameter that are inserted into the templates was changed to text/plain.

2. Sanitization of URL request path and restriction of request method was done by gorilla/mux.

```
r := mux.NewRouter()
r.HandleFunc("/", Index).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/menu", menu).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/browseVenue", venue.BrowseVenue).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/searchVenue", venue.SearchVenue).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/bookVenue", booking.BookVenue).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/removeBooking", booking.MainBookingList.RemoveBooking).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/browseBooking", booking.MainBookingList.BrowseBooking).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/signup", user.Signup).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/login", user.Login).Methods("GET", "POST").Schemes("https")
r.HandleFunc("/logout", user.Logout).Methods("GET", "POST").Schemes("https")
r.Handle("/favicon.ico", http.NotFoundHandler())

//connect to the port according to the assignment requirement
err := http.ListenAndServeTLS(":5221", "cert/cert.pem", "cert/key.pem", r)
if err != nil {
    log.Fatal.Fatalln("ListenAndServe: ", err)
}
```

2.4 Proper Error Handling & Logging

Proper error handling was done throughout the application to cover all the possible errors by the application. Besides error handling, the application was also coded to recover from any unexpected panic. All error, panic and fatal execution will be properly log inside the custom loggers. For this application, we have separated the log information into 5 types. Common log will log our Info and Warning log while for errors log, it will log all error, panic and fatal events. Below is the list of events that will be log inside the logger.

1. Info
    a. Authentication activities, sign up
    b. Failure in input validation (text input) -possible due to user unintended/intended errors.
    c. Zero search result based on criteria (for search venue & book venue)
    d. Deletion of booking
    e. Booking failures
    f. Completion of initialization of bookings
    g. Successful booking

2. Warning
    a. Record authentication failure
    b. Failure in input validation for login/signup
    c. Failure in input validation for user input (radio input, list selection)
        - The failure of this kind of input was log as warning as this has much lesser chance for user to make mistake as there are selection for them to choose from.
    d. Multi-login attempt
    e. There is no longer any booking.

3. Error
    a. Failure in initialization of bookings, registered user or slots.
    b. Failure in brycpt hashing of password
    c. Failure in remove valid booking ID, booked slots

4. Panic
    a. Panic message log after the panic is captured and recovered.

5. Fatal
    a. Failure in establish HTTPS connection
    b. Failure in executing templates
    c. Failure in opening log file

Sample logs are as below:

log > errorslog.txt

```
1  FATAL: 2021/03/25 11:15:13 main.go:56: ListenAndServe:  listen tcp :5221: bind: address already in use
2  ERROR: 2021/03/27 19:44:38 main.go:47: File Tampering of common log detected.
3  ERROR: 2021/03/27 19:44:38 main.go:65: File Tampering of error log detected.
4
```

Checksum will be performed every time the application is initialized; however, this number have to be updated whenever we close the application. This might be able to improve in the future to automate the process of updating the checksum of the file while exiting the server.

```go
logChecksum, err := ioutil.ReadFile("log/checksumcommonlog.txt")
if err != nil {
    fmt.Println(err)
}

str := string(logChecksum)

if b, err := ComputeSHA256("log/commonlog.txt"); err != nil {
    fmt.Printf("Err: %v", err)
} else {
    hash := hex.EncodeToString(b)
    if str == hash {
        log.Info.Println("Log integrity of common log is OK.")
    } else {
        log.Error.Println("File Tampering of common log detected.")
    }
}
```

```
log > checksumcommonlog.txt
  1   8bf1d4025fc22d9c29e3fdb9e2f74555b95e8179b39632f531814a3448011e6a
```

The SHA256 hash number can be generated from the terminal using below command line.
shasum -a 256 /Users/xinnioon/Projects/Go/src/goInAction2Assignment/log/commonlog.txt
shasum -a 256 /Users/xinnioon/Projects/Go/src/goInAction2Assignment/log/errorslog.txt

2.5 Cryptography & Encryption

For the application, hashing was done for the user password and file checksum as we do not need to know the content after the hashing was done. However, encryption was done for the communication as data still need to be access after it is encrypted.

2.5.1 Password hashing

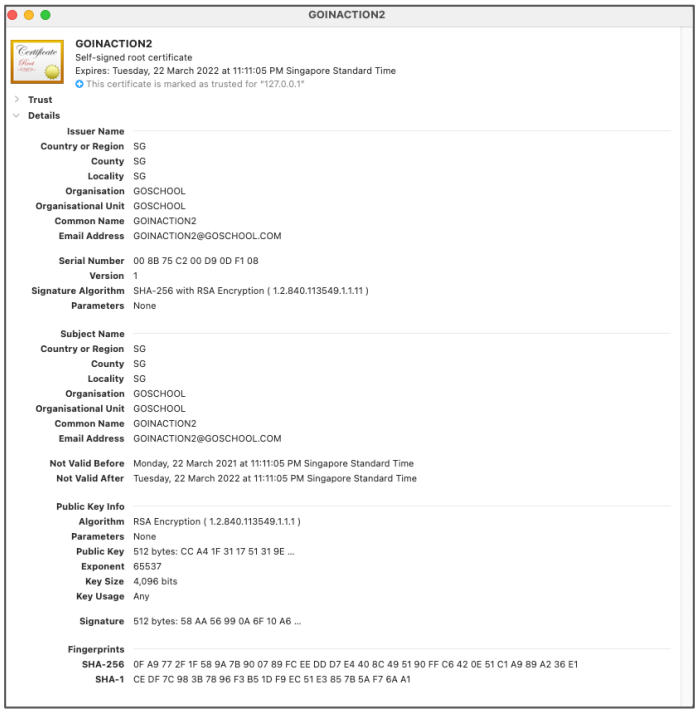
Third party package crypto/bcrypt was selected to hash our registered user password. This method of hashing is one of the ways recommended by OWASP. The package enabled the password to be hashed in a robust way.

2.5.2 File Checksum SHA-256

SHA-256 and BLAKE2 are the preferred hashing algorithm. SHA-256 was selected as the file checksum hashing. More details on this can refer back to 2.4 Proper error handling and logging.

2.5.3 Communication encryption

Communication between client and server was done encrypted to prevent man in the middle attack. This was done through the setup of ***https***. Generation of digital certification required to be done in order to establish an https connection. Below is a sample of SSL certificate generated. Besides the digital certificate, a private key was also generated together using OpenSSL. Both the certificate and key were stored inside the goInAction2Assignment/cert folder.



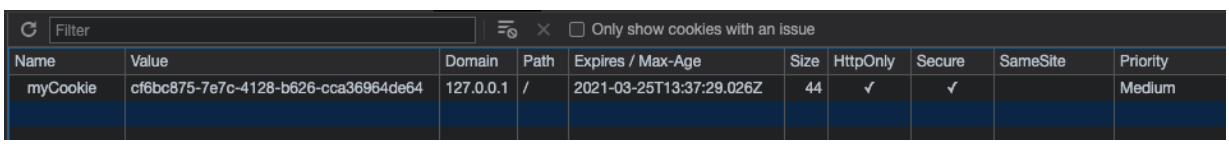
2.6 Session Management

Proper session management is important in order to prevent any MITM attacks. For this application, a few setups have been done to secure the user session.

1. Third party package satori/go.uuid was selected to generate the cookie value. This UUID is sufficiently random based on RFC 4122 standard.

2. A secured cookie was setup, and the application will expire the cookie every 30 minutes. (for non-critical operation). The user will be log out every 30 minutes with this setup, it has been tested with shorter timeframe.

```go
// create session
id := uuid.NewV4()
myCookie := &http.Cookie{
    Name:     "myCookie",
    Value:    id.String(),
    Expires:  time.Now().Add(30 * time.Minute),
    HttpOnly: true,
    Path:     "/",
    Domain:   "127.0.0.1",
    Secure:   true,
}
```



3. The system will generate a new session with a new cookie value whenever the user trying to do a new sign in.

4. Logout option from all pages was setup and the session will be deleted once the user select to log out or the session is logged in more than 30 minutes.

5. Multiple login was checked whenever user entered a matching username and password. If the username was found in any mapSession value, the new attempt to login will be stopped.

```
for _, v := range MapSessions {
    if v == username {
        return true
    }
}
```

2.7 Authentication & Password Management

1. All information of registered users are stored in json file with the password hashed using third party package bcrypt. This is to ensure that there is no sensitive information exposed in the source code.

2. Password entry is also obscured on user screen and autocomplete attribute has been turned off.

3. All login/sign up html form was sent to the server through HTTP Post method to ensure credentials leakage.

4. A generic error message will be thrown to the user if there are some errors detected in login process. The error can be either no existing username or the password is not matched.

5. All failed login attempt and signup attempt will also be log inside the custom logger.

3. Idiomatic Go

Below are the idiomatic Go practices that has been implemented in this assignment.

1. Go doc comments are done in complete sentences.
2. Every package has a comment to explain its purpose and usage.
3. All exported package has a upper case first character and all the variable are defined in camelCase form.
4. Codes are written defensively, and error handling are done wherever possible.
5. The return values that are not required will be define as blank identifier and this value will be discarded.

4. Appendix (Go Documentation)

Go Documentation Server

Search

Command goInAction2Assignment

goInAction2Assignment is an application of a function room booking system for coworking space in Singapore.

```
Function of the application including:
1. Login/logout
2. Browse venue
3. Search venue
4. Book venue
5. Browse booking
6. Delete booking
```

Subdirectories

Name	Synopsis
..	
booking	Package booking stores all the booking variables and functions.
log	Package log is a custom logger created to log various important information.
user	Package user contains all the registered user for the application and login logout feature.
venue	Package venue store information of all the function rooms available.

GoDoc

Search

Package log

```
import "goInAction2Assignment/log"
```

Overview
Index

Overview

Package log is a custom logger created to log various important information.

Index

Variables
func init()

Package files

log.go

Variables

Info, Warning, Error, Panic, Fatal were declared.

```
var (
    Info    *log.Logger //to record information action done by the apps, also it will record the user input mistake.
    Warning *log.Logger //to record information that worth server attention such as fail login attempt or fail action.
    Error   *log.Logger //to record any undesired apps error that are not caused by user.
    Panic   *log.Logger //to record any unexpected panic event.
    Fatal   *log.Logger //to record any fatal event.
)
```

func init

```
func init()
```

GoDoc

Search

Package user

```
import "goInAction2Assignment/user"
```


Overview

Package user contains all the registered user for the application and login logout feature.

Index

Package files

loginlogout.go

Variables

```
var (
    MapUsers    = map[string]User{}   //mapUsers uses username as a key and map with all its related information in user
struct.
    MapSessions = map[string]string{} //mapSessions map registered user to a particular cookie value.
    tpl         *template.Template

    //Unique policy creation for the life of the program.
    Policy = bluemonday.UGCPolicy()
    // use a single instance of Validate
    validate *validator.Validate
)
```

func AlreadyLoggedIn

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

AlreadyLoggedIn function enabled quick check of if a particular session is checked in.

func GenerateUser

```
func GenerateUser()
```

GenerateUser initiate preliminary user list and brcrypt their password in a secure manner.

func Login

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

Login function allow registered user to login to the system. This function will also check for the existence of multiple login. Concurrent/mulitple login is not allowed for this apps.

func Logout

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

Logout function is able to access by user in all the pages. Once registered user is log out, the current session cookies will be cleared.

func **Signup**

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

Signup function enabled all the unregistered user to signup before they are able to access registered user feature.

func **init**

```
func init()
```

func **multiLogin**

```
func multiLogin(username string) bool
```

multiLogin is to check through all the sessions to see if particular user login twice

type **User**

User struct is a type that stored all registered user information.

```
type User struct {
    FirstName string
    LastName  string
    UserName  string
    Password  []byte
}
```

func **GetUser**

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

GetUser function can helped to check if a particular session is map to any registered user. If an cookie value is able to map to registered user, it will return it. Else, the return would be empty.

GoDoc

Search

Package venue

```
import "goInAction2Assignment/venue"
```


Overview

Package venue store information of all the function rooms available. It also store the slot availability of a venue in hour. User of the application can view the venue availability using the functions in this package. Any modification of the availability of the slot can be performed using the function in this package as well.

Index

Package files

browse.go search.go update.go

Variables

Rooms available to pick from are stored in slice of struct data type. This slice contains all the details information of the function room such as name, kind, capacity and slot availability.

```
var (
    Room = []roomInfo{
        {Name: "MR01", Kind: "MeetingRoom", Capacity: 10, Slot: TotalAvailableSlot[:279]},
        {Name: "MR02", Kind: "MeetingRoom", Capacity: 10, Slot: TotalAvailableSlot[279:558]},
        {Name: "MR03", Kind: "MeetingRoom", Capacity: 10, Slot: TotalAvailableSlot[558:837]},
        {Name: "MR04", Kind: "MeetingRoom", Capacity: 20, Slot: TotalAvailableSlot[837:1116]},
        {Name: "MR05", Kind: "MeetingRoom", Capacity: 20, Slot: TotalAvailableSlot[1116:1395]},
        {Name: "AR06", Kind: "ActivityRoom", Capacity: 50, Slot: TotalAvailableSlot[1395:1674]},
        {Name: "AR07", Kind: "ActivityRoom", Capacity: 50, Slot: TotalAvailableSlot[1674:1953]},
        {Name: "AR08", Kind: "ActivityRoom", Capacity: 50, Slot: TotalAvailableSlot[1953:2232]},
        {Name: "AD09", Kind: "Auditorium", Capacity: 100, Slot: TotalAvailableSlot[2232:2511]},
        {Name: "AD10", Kind: "Auditorium", Capacity: 100, Slot: TotalAvailableSlot[2511:2790]},
    }
    availableSlot      = make(SlotArr, 0, 279)
    TotalAvailableSlot = make(SlotArr, 0, 2790)
)
```

```
var (
    wg           sync.WaitGroup
    err          error
    mapBookingID = map[string][]int{} //mapBookingID store the results from checkCriteria function.
)
```

AvailableVmap store all the search results for a particular user.

```
var AvailableVmap = map[string][]VenueDetail{}
```

DayOfMonth assign days to different months Not really useful at this stage of project,setup upfront for possible future expansion

```
var DayOfMonth = map[string]int{
    "January": 31, "February": 28, "March": 31, "April": 30, "May": 31, "June": 30,
    "July": 31, "August": 31, "September": 30, "October": 31, "November": 30, "December": 31,
}
```

```
var tpl *template.Template
```

func **BrowseVenue**

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

BrowseVenue function show table of slot availabiltiy based on user venue selection.

func **CheckCriteria**

```
func CheckCriteria(date int, time int, duration int, size int, kind string) ([]int, error)
```

CheckCriteria based on criteria provided, search and sort available slot

func **GenerateSlots**

```
func GenerateSlots()
```

GenerateSlots initiate slice of slots based on date, month and venue (XXXXXX)

func **SearchVenue**

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

SearchVenue function enable the user to key in all their criteria in the browser and perform a search on available venue based on the inputs.

func **indexOfLargest**

```
func indexOfLargest(arr []int, n int) int
```

indexOfLargest return the largest number in the slice.

func **init**

```
func init()
```

func **printTableOfSlot**

```
func printTableOfSlot(res http.ResponseWriter, req *http.Request, num int)
```

printTableOfSlot print out the availability of all slots for a particular venue.

func **screenSlot**

```
func screenSlot(find chan int, slotDetail int, duration int) error
```

ScreenSlot was launched (10 goroutines) to perform slot availability binary search

func **selectionSort**

```
func selectionSort(arr []int, n int)
```

Selection sort was performed based on room number on the available slots array.

func **swap**

```
func swap(x *int, y *int)
```

swap perform a swap in location between two variable.

type **Slot**

Each slot is setup with 2 information, slot details and its availability.

```
type Slot struct {
    Info      int
    Available bool
}
```

type **SlotArr**

All the slots are store in slice of struct data type.

```
type SlotArr []Slot
```

func (SlotArr) **RemoveBookedSlot**

```
func (s SlotArr) RemoveBookedSlot(bookings []int) error
```

RemovedBookedSlot help to update the slot back to true when the admin remove the slot.

func (SlotArr) **UpdateSlotAvailability**

```
func (s SlotArr) UpdateSlotAvailability(bookings []int) error
```

UpdateSlotAvailability help to update the slot to false when the venue is booked.

type **VenueDetail**

VenueDetails is a data type that store details of user search criteria of venue.

```
type VenueDetail struct {
    Day         int
    Month       int
    Time        int
    Room        int
    Duration    int
    Participant int
}
```

type **roomInfo**

RoomInfo equipped with basic info of room and also an array of its slot availability.

```
type roomInfo struct {
    Name     string
    Kind     string
    Capacity int
    Slot     SlotArr
}
```

GoDoc

Search

Package booking

```
import "goInAction2Assignment/booking"
```


Overview

Package booking stores all the booking variables and functions. It includes function to initialise preloaded bookings, performing venue booking search, and lastly generate booking record

Index

Package files

booking.go browseDeleteBooking.go

Variables

```
var (
    MainBookingList = &bookingList{nil, 0} //MainBookingList store all bookings in linked list.
    bookingCount    int64                  //bookingCount will be treated as booking ID. It is locked inside mutex so
that there will be access once at a time.
    mu              sync.Mutex             //mu is used to lock the booking process where only one booking can be done at
a time.
    tpl             *template.Template
    wg              sync.WaitGroup //wg is used to wait the go routine that being launched to preloaded some bookings.
    err             error
    mapBookingID    = map[string][]int{} //mapBookingID store the searchCriteria result from the booking query for every
user.
)
```

func BookVenue

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

BookVenue function will first perform a search based on the booking input by user. After that, all available venue that matches the criteria will be listed out. Registered user can select the venue and hit the submit button to book it.

func GenerateBookings

```
func GenerateBookings() error
```

GenerateBookings initiate the application with some preloaded booking list

func init

```
func init()
```

type booking

booking struct is a type of every booking information.

```
type booking struct {
    BookingID       int64
```

```
    Venue           int
    BookingSlot     []int
    Duration        int
    participantSize int
    Host            string
    next            *booking
}
```

type **bookingList**

bookingList struct is setup for linked list of booking.

```
type bookingList struct {
    head *booking
    size int
}
```

func (*bookingList) BrowseBooking

```
func (b *bookingList) BrowseBooking(res http.ResponseWriter, req *http.Request)
```

BrowseBooking function will be triggered in two scenario. 1.Registered user can choose to browse their own booking record. 2.Admin can browse the whole booking list. He/she can also filter the booking list by user name.

func (*bookingList) RemoveBooking

```
func (b *bookingList) RemoveBooking(res http.ResponseWriter, req *http.Request)
```

RemoveBooking function will perform remove booking feature based on booking ID. Only admin can perform remove booking.He/she can select the booking ID based on the full booking list displayed below.

func (*bookingList) addBooking

```
func (b *bookingList) addBooking(option int, userName string, bookings []int, duration int, size int, bookingError chan
error) error
```

addBooking function traverse through booking linked list to add new booking record.