Function Rooms Booking System For Co-working Space In Singapore

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

Table of Contents

1 Background

For this assignment, a text-based application of function rooms booking system is created. In this application, there will be a few different types of function rooms that can accommodate different capacities available for booking. Each booking slot will be an hour long. There are 9 slots per day, starting from 10am to 7pm. For the scope of this assignment, booking slots are only initiated for the month of February. The application has the capability to expand to other months in the future.

2 Features

The text-based application has been designed with the following features in order to meet the assignment requirements. When the application is run, it will initiate functions to preload functions rooms, create available slot for the month and add in a few upfront bookings for better illustration of the application.

2.1 Function Rooms Booking System Application Menu

Below is the menu printout of the application.

```
Welcome to Function Rooms Booking System
========================================
1. Browse venue and availability
2. Search for available rooms
3. Book Venue
4. Remove Booking (for admin only)
5. List whole booking list (for admin only)
6. Exit the Booking System
Select your choice:
1
```

User has to select choice 1 to 6 (integer type) to proceed, else message of “Please select 1 to 6” will prompt out.

2.2 Browse venue and availability

The first feature will be to browse the venue and the overview of its slot availability. To create this feature, 10 different types of function rooms has been added into the application. All 10 function rooms are stored in *array-based list*. Array based list has been selected to store this data as we don’t foresee changes to the selection of function rooms in this coworking space. All the details of function room are stored in struct form.

This co-working space has a total of 10 different function rooms (3 different types).

1. Meeting room (MR01, MR02, MR03 with a capacity of 10)
2. Meeting room (MR04, MR05 with a capacity of 20)
3. Activity room (AR06, AR07, AR08 with a capacity of 50)
4. Auditorium (AD09, AD10 with a capacity of 100)

The room structure is setup as below.

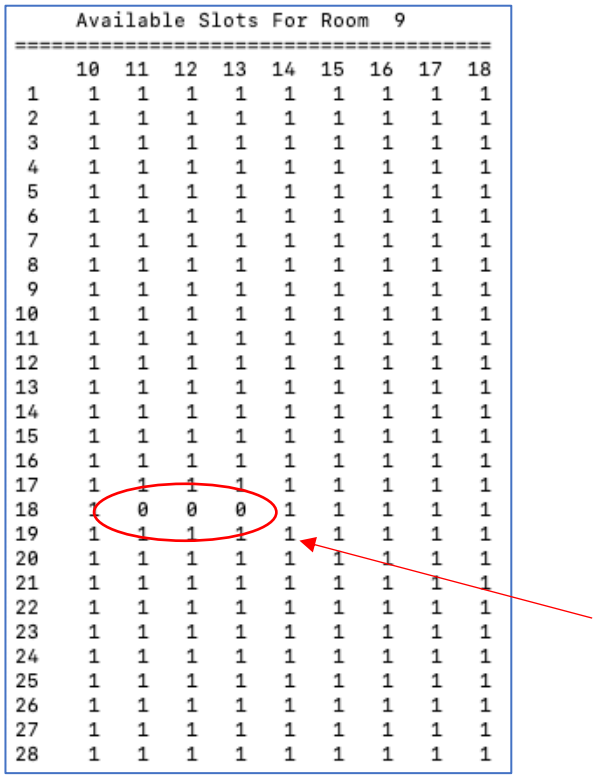
<table>
<tr><td colspan="2">Venue</td></tr>
<tr><td>Name (string)</td><td>MR01</td></tr>
<tr><td>Type (string)</td><td>Meeting Room</td></tr>
<tr><td>Capacity (integer)</td><td>10</td></tr>
<tr><td>Slot Availability
(struct slice- length 252)</td><td>for illustration only
<table><tr><td>161001</td><td>161101</td></tr><tr><td>True</td><td>False</td></tr></table></td></tr>
</table>

The bookings can only be done for the month of February for the scope of this assignment. However, this system is built in a way that it can be expanded to other months in the future.

- There will a total of 28 days for February and available booking slots will be from 10am to 7pm (Total slots per venue: 28 X 9 = 252)
- The slots availability will be stored as a simple struct with slot details in a code format (DDTTVV) & slot availability in boolean type in a one-dimensional *slice-based struct list*.
- Example of slot details being computed: slot of Feb 16 11:00am Meeting Room 3 will be stored as 161103
- There will be in total of 2520 slots open up for bookings. 252 slots per venue in month of February.

Once option 1 is selected, the following will be displayed. To continue to view full slot availability of the venue, user has to select 1. Select 2 will bring the user back to the menu. Other option will show as invalid selection.

```
List of venue:
Room: 1 --> Name: MR01 , Type: MeetingRoom, Capacity: 10
Room: 2 --> Name: MR02 , Type: MeetingRoom, Capacity: 10
Room: 3 --> Name: MR03 , Type: MeetingRoom, Capacity: 10
Room: 4 --> Name: MR04 , Type: MeetingRoom, Capacity: 20
Room: 5 --> Name: MR05 , Type: MeetingRoom, Capacity: 20
Room: 6 --> Name: AR06 , Type: ActivityRoom, Capacity: 50
Room: 7 --> Name: AR07 , Type: ActivityRoom, Capacity: 50
Room: 8 --> Name: AR08 , Type: ActivityRoom, Capacity: 50
Room: 9 --> Name: AD09 , Type: Auditorium, Capacity: 100
Room: 10 --> Name: AD10 , Type: Auditorium, Capacity: 100
Select 1 to check venue availability
Select 2 to return
1
Which venue to view?
5
```



```
     Available Slots For Room  9
=======================================
     10  11  12  13  14  15  16  17  18
 1    1   1   1   1   1   1   1   1   1
 2    1   1   1   1   1   1   1   1   1
 3    1   1   1   1   1   1   1   1   1
 4    1   1   1   1   1   1   1   1   1
 5    1   1   1   1   1   1   1   1   1
 6    1   1   1   1   1   1   1   1   1
 7    1   1   1   1   1   1   1   1   1
 8    1   1   1   1   1   1   1   1   1
 9    1   1   1   1   1   1   1   1   1
10    1   1   1   1   1   1   1   1   1
11    1   1   1   1   1   1   1   1   1
12    1   1   1   1   1   1   1   1   1
13    1   1   1   1   1   1   1   1   1
14    1   1   1   1   1   1   1   1   1
15    1   1   1   1   1   1   1   1   1
16    1   1   1   1   1   1   1   1   1
17    1   1   1   1   1   1   1   1   1
18    1   0   0   0   1   1   1   1   1
19    1   1   1   1   1   1   1   1   1
20    1   1   1   1   1   1   1   1   1
21    1   1   1   1   1   1   1   1   1
22    1   1   1   1   1   1   1   1   1
23    1   1   1   1   1   1   1   1   1
24    1   1   1   1   1   1   1   1   1
25    1   1   1   1   1   1   1   1   1
26    1   1   1   1   1   1   1   1   1
27    1   1   1   1   1   1   1   1   1
28    1   1   1   1   1   1   1   1   1
```

Once a function room is selected, an updated table of the slot availabilities will be printed according to all the bookings done. The slot that has been booked will be reflected as 0. Through this table, all the users of the application (no login is required) can has a quick view which slot is still available.

2.3 Search for available rooms

Besides viewing the availability of rooms through the table of slot availability, user can also let the application do the hard work through providing the criteria for the search. Inputs requested will be as below.

```
Please enter your preferred date. (DDMM)
1602
Please enter your preferred time. (eg.1000)
1100
Please enter duration of the event in hours.
4
Please enter total number of participant.
35
Please enter the preferred type(1, 2 or 3). If there is none, press enter
1. Meeting Room
2. Activity Room
3. Auditorium

16/2 11:00 AR06 is available.
16/2 11:00 AR07 is available.
16/2 11:00 AR08 is available.
16/2 11:00 AD09 is available.
16/2 11:00 AD10 is available.
```

In order to perform the search for the available rooms to fit the whole duration of the bookings, based on the input of user above, the application will compile the input details into four slot details (based on duration) which is 161101, 161201, 161301,161401 for room 1, it will perform the same for other 9 rooms. After that *10 go routines* will be called to check these four slots details for each room. If all these four slots are not booked, the go routine will send back the slot details back to the function through *unbuffered channel*. On the other hand, if the slots are booked, the go routine will send back 0.

The go routine performs the search of the location of slot details in the array-based list by using *binary search* as the slot availability array (length of 252) is a sorted array. Binary search has enabled slot details to be located quickly.

The results received from 10 different go routines will not be in sequence of the room number. Hence, sorting will be required before it can be presented to user nicely. Due to the size of the data (data size of 10), *selection sort* is selected for this sort.

Besides that, the last filter will be the available rooms has to fit the number of participants. Only room that can fit the number of participants will be printed out. As shown in the above screen shot, only function room 6-10 can fit 35 peoples and above.

2.4 Book Venue

Viewing list of function rooms, displaying table of slot availability and perform search based on criteria can be done without login in. However, in order to book a venue. Users need to login to the application. Registered users are stored in a *singly linked list*. As you can see, when 3 is selected, a message will be prompt out to let the user select whether you are a registered user or do you plan to register as a new user. If 1 is selected, the application will request your user ID and password from you.

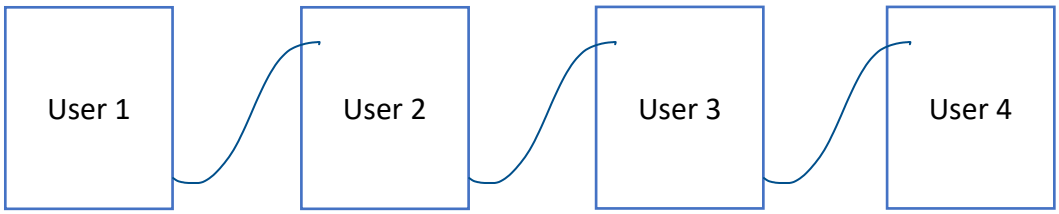
```
Select your choice:
3
Login is needed for venue booking
Are you registered user?
Select 1 for registered user, select 2 to register new user
1
Please enter your userID
lee.matthew
Please enter your password
leematthew
Hold on, logging in......
Successful Login!
Hello lee.matthew !
Please enter your preferred date. (DDMM)
```

If 2 is selected, the application will prompt you to key in the information to register as a new user as below. After new user is registered, the user can continue to book the venue.

```
Login is needed for venue booking
Are you registered user?
Select 1 for registered user, select 2 to register new user
2
What is your first name?
Cathie
What is your last name?
Wood
What is your preferred userID?
cathie.wood
Please enter password:
cathiewood
New user registered!
Hello cathie.wood !
```

For all the user, it is stored in a linked list in a form of struct. The user structure is as below.

User	
First Name (string)	Matthew
Last Name (string)	Lee
User ID (string)	lee.matthew
Password (string)	leematthew
Category (int)	2 for Admin 1 will be selected as default for all the new user
next	*user (pointer of the next user)



When the registered user key in their user ID and password, the application will *traverse through the user linked list* to search for user ID and password for the particular user if it is able to find a matching user ID in the linked list. If user ID is not found, the user has to register as new user before proceeding. If user ID is found and the password stored for that user is matching, the application will print out Login Successful and Hello (user ID)!

After user successfully login, they will require to input details for their booking. An example of input requested are shown below. A similar search algo as choice 2 – search for available rooms will be run and suggest to the user the available rooms for booking based on the criteria. User can proceed their booking by selecting their choice of room.

After that, booking will be processed, the slot availability of relevant slots will be change to false which in turn reflected in the table of slot availabilities for the particular function room (as shown in the screen shot below).

```
Please enter your preferred date. (DDMM)
1802
Please enter your preferred time. (eg.1000)
1400
Please enter duration of the event in hours.
3
Please enter total number of participant.
35
Please enter the preferred type(1, 2 or 3). If there is none, press enter
1. Meeting Room
2. Activity Room
3. Auditorium

18/2 14:00 AR06 is available.
18/2 14:00 AR07 is available.
18/2 14:00 AR08 is available.
18/2 14:00 AD09 is available.
18/2 14:00 AD10 is available.
Select your venue based on availability
7
Creating booking...
Booking 181607 , change availability to false
Booking 181507 , change availability to false
Booking 181407 , change availability to false
Booking done!
```

```
     Available Slots For Room  7
=========================================
     10  11  12  13  14  15  16  17  18
 1    1   1   1   1   1   1   1   1   1
 2    1   1   1   1   1   1   1   1   1
 3    1   1   1   1   1   1   1   1   1
 4    1   1   1   1   1   1   1   1   1
 .
 .
 .
16    1   1   1   1   1   1   1   1   1
17    1   1   1   1   1   1   1   1   1
18    1   1   1   1   0   0   0   1   1
19    1   1   1   1   1   1   1   1   1
```

Every booking is stored in a struct and the whole booking list are connected as a singly linked list. In order to add a booking, the application have to first traverse through the linked list and add the new booking list to the end of the list. The booking details stored as below.

Booking Form	
Booking ID (int)	5
Booking Venue (int)	3
Booking Slots ([]int)	{121103, 121203, 121303}
Duration (Hours) (int)	3
Participant Size (int)	8
Host (string)	lee.matthew
next	*booking (pointer of the next user)

2.5 Remove booking (for admin only)

In order to remove booking, user has to perform login. The application will then check if the user is of category 2 (which indicate the user is an admin). If yes, the application will check with the user which booking ID the user wanted to remove. Based on that, booking will be removed from the booking list. The slot availability will also be updated and change from false to true.

```
(Admin right is required to delete booking/view whole booking list)
Login is needed for venue booking
Are you registered user?
Select 1 for registered user, select 2 to register new user
1
Please enter your userID
lee.matthew
Please enter your password
leematthew
Hold on, logging in......
Successful Login!
Welcome lee.matthew
Displaying booking list.....
Booking ID: 0, Bookings for 3 hours: [201401 201501 201601] , Host: lee.matthew, Venue: 1
Booking ID: 1, Bookings for 2 hours: [171001 171101] , Host: tan.alison, Venue: 1
Booking ID: 2, Bookings for 2 hours: [221203 221303] , Host: tan.alison, Venue: 3
Booking ID: 3, Bookings for 4 hours: [31104 31204 31304 31404] , Host: ong.ryan, Venue: 4
Booking ID: 4, Bookings for 5 hours: [281006 281106 281206 281306 281406] , Host: tan.alison, Venue: 6
Booking ID: 5, Bookings for 3 hours: [181109 181209 181309] , Host: lim.christina, Venue: 9
Which booking ID you wish to remove?
3
Removed Booked Slot
[31104 31204 31304 31404]
```

A request to print the whole booking list after the booking ID 3 is deleted are as below. Booking ID 3 is no longer linked to the booking list linked list as shown below.

```
Welcome lee.matthew
Displaying booking list.....
Booking ID: 0, Bookings for 3 hours: [201401 201501 201601] , Host: lee.matthew, Venue: 1
Booking ID: 1, Bookings for 2 hours: [171001 171101] , Host: tan.alison, Venue: 1
Booking ID: 2, Bookings for 2 hours: [221203 221303] , Host: tan.alison, Venue: 3
Booking ID: 4, Bookings for 5 hours: [281006 281106 281206 281306 281406] , Host: tan.alison, Venue: 6
Booking ID: 5, Bookings for 3 hours: [181109 181209 181309] , Host: lim.christina, Venue: 9
```

Below is a demo entry of a non-admin registered user. He/she can login successful but cannot remove booking or view the whole booking list because the user is not an admin.

```
(Admin right is required to delete booking/view whole booking list)
Login is needed for venue booking
Are you registered user?
Select 1 for registered user, select 2 to register new user
1
Please enter your userID
tan.alison
Please enter your password
tanalison
Hold on, logging in......
Successful Login!
sorry, remove booking/print whole booking list are not allowed. You are not an admin
```

2.6 List whole booking list

The final feature is to allow admin to either view the whole booking list or to view the booking list done by certain user. The booking list will show the slots that has been booked out and number of participants that will use the facility. Lastly, the host of the event is also shown in the booking details. First screen shot show the demo of displaying the full booking list. The second screen shot display the booking done by Alison Tan.

```
Select your choice:
5
(Admin right is required to delete booking/view whole booking list)
Which booking list you would like to view?
1. Complete Booking List (based on booking ID)
2. Booking List of particular user
1
Are you registered user?
Select 1 for registered user, select 2 to register new user
1
Please enter your userID
lee.matthew
Please enter your password
leematthew
Hold on, logging in......
Successful Login!
Welcome lee.matthew
Displaying booking list.....
Booking ID: 0, Bookings for 3 hours: [201401 201501 201601] , Host: lee.matthew, Venue: 1
Booking ID: 1, Bookings for 3 hours: [181109 181209 181309] , Host: lim.christina, Venue: 9
Booking ID: 2, Bookings for 2 hours: [221203 221303] , Host: tan.alison, Venue: 3
Booking ID: 3, Bookings for 2 hours: [171001 171101] , Host: tan.alison, Venue: 1
Booking ID: 4, Bookings for 4 hours: [31104 31204 31304 31404] , Host: ong.ryan, Venue: 4
Booking ID: 5, Bookings for 5 hours: [281006 281106 281206 281306 281406] , Host: tan.alison, Venue: 6
```

```
Select your choice:
5
(Admin right is required to delete booking/view whole booking list)
Which booking list you would like to view?
1. Complete Booking List (based on booking ID)
2. Booking List of particular user
2
Which user ID you would like to view?
tan.alison
Are you registered user?
Select 1 for registered user, select 2 to register new user
1
Please enter your userID
lee.matthew
Please enter your password
leematthew
Hold on, logging in......
Successful Login!
Welcome lee.matthew
Displaying booking list.....
Booking ID: 2, Bookings for 2 hours: [221203 221303] , Host: tan.alison, Venue: 3
Booking ID: 3, Bookings for 2 hours: [171001 171101] , Host: tan.alison, Venue: 1
Booking ID: 5, Bookings for 5 hours: [281006 281106 281206 281306 281406] , Host: tan.alison, Venue: 6
```

Lastly, select choice 6 will allow you to exit the application entirely. Else, the menu will keep showing as a result of the loop.

3 Data Structures and algorithms

This section will summarise the data structures and algorithms that has been used for this application. A brief discussion on the logic behind the selection will also be done.

3.1 Room List (slice of struct – array-based list)

As discussed earlier in the feature of application, room list was created as an array-based list because we do not foresee the change in the infrastructure of this coworking space in the near future. If there is any, we foresee that the changes would not be frequent or drastic, hence select an array-based list is appropriate as we most likely would not take up a lot extra memory allocation as the changes in the array length would be kept at a minimal. One of the benefits of setting the room list in array-based list is the array lookup will be O (1).

3.2 Slot availability List (slice of struct – array-based list)

Slot availability slice of every function room is store within the slice of the room list. They are store using a six-digit slot details comprises of date, time and room number.

3.2.1 Binary Search

Due to the way the slot is initialised, the array itself will be a *sorted array* within every room. Hence binary search can be used to search availability of the slot. Binary search through the slot availability list of 252 slots has reduce the complexity of searching the data by screening the whole list (O(n)) to $O(\log_2 n)$.

3.2.2 Selection Sort

Not much sorting algorithm was implemented in this application as majority of the data has been sorted when they are introduced. One of the areas that sorting algorithm can be used is to sort the sequence of slot availability according to room number after the availability of the room for a particular date & time is sent back through go routines. There are in total 10 data that were sent back. Based on the size of this data, selection sort would be a suitable choice to do the sorting. Although selection sort has a high complexity of $O(n^2)$, but this would not be significant for a small n. Besides selection sort, we can also consider insertion sort to sort this data if needed, however, since it is stored in an array-based list, it might be less suitable to use insertion sort.

3.2.3 Update slot availability

There are two steps in update the slot availability to false once the user has chosen to book for the particular slot. Firstly, we will need to locate the particular slot using binary search. After that, we need to update the slot availability of the particular slot through updating the information in the address of the slot.

3.3 User list (singly linked list)

As user list will be constantly increase over time, it would be appropriate to use linked list or BST to store the full user list. Double linked list has been considered as well, however; it was not chosen in the end as it would takes up double memories allocation although it can make adding a new user much easier as transversing through the complete list will not be needed. The complexity of adding a new user for doubly linked list is O (1). Reverse linked list can be considered in this case as well. Linked list was chosen in the end instead of a binary search tree due to the way of the user list was added. An unbalance binary search tree might be built. This might defeat the purpose of setting up the list as a binary search tree in the first place. A brief search of BST shows that there are a few ways we can actually balance the BST. These can be further explored in the future. For this assignment purpose, singly linked list has been chosen in the end.

3.3.1 Add item

For user list, one of the main features that needed to be done is to add new user to the linked list. In order to do that, transversing across the whole linked list is required. This would take a time complexity of O(n).

3.4 Booking List (singly linked list)

Booking list was also setup in a singly linked list as the booking list size was unpredictable. The dynamic memory allocation of a linked list ensure that no unnecessary extra memory was allocated to the list. One of the extra details for our booking list is the booking ID. It was created inclemently every time a new booking was created. Original it was done using the relationship with the booking list size. However, this link was omitted in the end as booking list need to have the delete item feature which will in turn affected the booking ID in the long run as the booking list size might get reduced if delete item was done and this will make the booking list to have a repetitive booking ID.

3.4.1 Add item

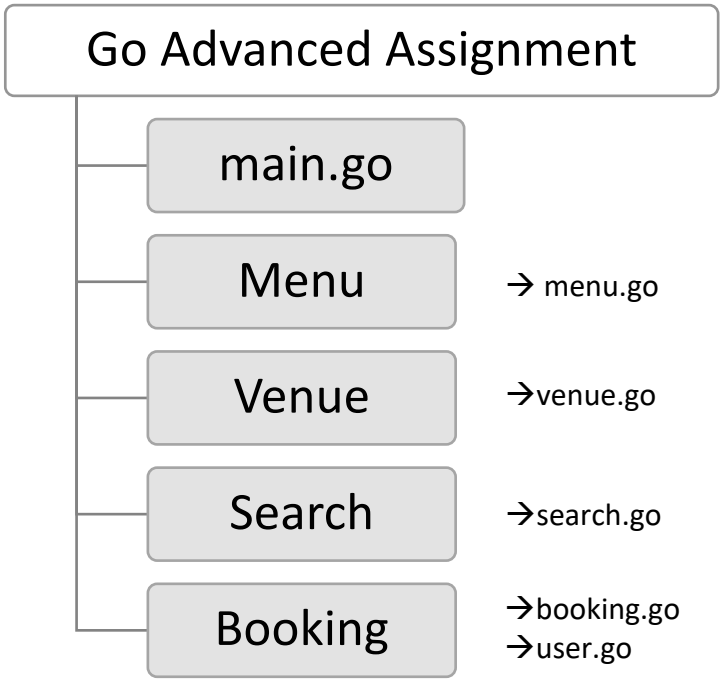
Adding booking to the booking list would be the same as section 3.3.1.

3.4.2 Delete item

In order to delete item of the booking list based on the booking ID, the booking list will need to be traverse until the matching booking ID is found, and the traversing will be stopped after the booking ID is found. Hence, in worst case, this action will also have a complexity of O(n).

4 Packages

For this application, it has been grouped into four different packages. This is a small application and might not required different packages. However, modularised the application while it is still small might help in its future expansion of features. The data were all stored in go file for this application.



5 Error handling and concurrency mechanism

5.1 Error handling and panic

For this application, a lot of error handlings has been inserted especially on the input requested from the user. The application was trying to ensure that meaningful inputs have been accepted so that the result presented in the end would be meaningful. Besides that, error handling has been inserted into the search algorithm in order to make sure that if what is needed is not found, it would be reflected as an error. All the errors have been returned from the function and the error will be printed from the menu package in the end. However, error handling was not included in some simple function that we do not foresee any error such as printing function.

Besides error handling, panic handling has been incorporated into this application. Majority of the errors have been taken care by error handling. However, panic handling has been incorporated as well in case there is any unexpected error occurred. Panic was captured using a deferred function and calling recover () in menu.go stop the function stack to unwind further.

5.2 Using channel for search

Channel was used in this application in returning the availability of a particular slot. In order to increase the efficiency of the binary search, 10 go routines has been launched to perform binary search for the slot in 10 different rooms concurrently. However, due to the result received would not be in sequence of the room number, we will need to perform sort after we receive the result of all available slots. The booked slot will send a zero integer to the channel. After all the go routines has finished running, the user can have a quick view on all available rooms for a particular time and date.

5.3 Tackling concurrency issue

Besides performing the binary search concurrently, another trial to incorporate concurrency in the application is to initiate the upfront bookings of the function rooms when the application is launched. Since there are no preference in sequence of making different bookings, all 6 upfront bookings were done concurrently using 6 different go routines. However, doing booking in this matter will caused race issue when all the go routine were trying to get a booking ID (through reading the booking count-shared resource) at the same time. Reading the booking count can only be done by one go routine in a time. After that booking count should be increase before it was allowed to be read by another go routine. Hence atomic function was used to prevent the race condition of the concurrency.

6 Instruction to run application

An executable file (FunctionRoomsBookingSystem.exe) has been built for this text-based application. Appropriate input for the application has been demo in the features of the application in Section 2 of the report. Below are the data that has been preloaded into the system. In order to try out remove booking/view booking list feature, user ID : lee.matthew and password: leematthew should be used as he is the admin of this application.

```
List of venue:
Room: 1 --> Name: MR01 , Type: MeetingRoom, Capacity: 10
Room: 2 --> Name: MR02 , Type: MeetingRoom, Capacity: 10
Room: 3 --> Name: MR03 , Type: MeetingRoom, Capacity: 10
Room: 4 --> Name: MR04 , Type: MeetingRoom, Capacity: 20
Room: 5 --> Name: MR05 , Type: MeetingRoom, Capacity: 20
Room: 6 --> Name: AR06 , Type: ActivityRoom, Capacity: 50
Room: 7 --> Name: AR07 , Type: ActivityRoom, Capacity: 50
Room: 8 --> Name: AR08 , Type: ActivityRoom, Capacity: 50
Room: 9 --> Name: AD09 , Type: Auditorium, Capacity: 100
Room: 10 --> Name: AD10 , Type: Auditorium, Capacity: 100
```

```
Displaying booking list.....
Booking ID: 0, Bookings for 3 hours: [201401 201501 201601] , Host: lee.matthew, Venue: 1
Booking ID: 1, Bookings for 3 hours: [181109 181209 181309] , Host: lim.christina, Venue: 9
Booking ID: 2, Bookings for 2 hours: [221203 221303] , Host: tan.alison, Venue: 3
Booking ID: 3, Bookings for 2 hours: [171001 171101] , Host: tan.alison, Venue: 1
Booking ID: 4, Bookings for 4 hours: [31104 31204 31304 31404] , Host: ong.ryan, Venue: 4
Booking ID: 5, Bookings for 5 hours: [281006 281106 281206 281306 281406] , Host: tan.alison, Venue: 6
```

```
mainList.addUser("Matthew", "Lee", "lee.matthew", "leematthew", 2)
mainList.addUser("Alison", "Tan", "tan.alison", "tanalison", 1)
mainList.addUser("Christina", "Lim", "lim.christina", "limchristina", 1)
mainList.addUser("Ryan", "Ong", "ong.ryan", "ongryan", 1)
```