References for Administrator and user

1. Administrator Part:

1. The Database Design:

Requirements analysis:

For this Hotel Management system, it needs two major parts including room booking or cancelling part and meal booking or cancelling part. Meanwhile, it has two branches, which are guest branch and staff branch.

The guest should be able to register their account at first, then they can use this account to login the system for room booking/cancelling and meal booking/cancelling. Guests must provide personal information to register account: 1. Username, 2. Their Real Name, 3. Passport ID, 4. Telephone Number, 5. Email Address. The information updating is optional in the requirements. To login account of guests, they need to type in their username and password correctly. Afterwards, if they login successful, (Use correct username and password) they will have some options such as “Book Room”, “Book Meal”, “Cancel Room”, “Cancel Meal”, and “Log out”. (Important options) When guest booking a room, they need to specify the check-in date, check-out date, and room type. If there are more than one available room are consistent with their specified condition, then the system would allow them to choose the specific room they want to stay.

For the food service of guests, they can book the meal two days in advance. The chefs only work in certain days in each week. Most of the meals are different in their meal lists, but two meals could be cooked by three of them. These meals are “fried egg” and “curry rice”. Meanwhile, the meal is only available between 7:00AM and 10:00PM.

For the staff part, their account should be associated with “Username”, “Password”, “Telephone Number”, and “Staff ID”. Staff should be able to check the room booking situation at least.

Table Design in Database:

Table Design:

I have created 10 tables to store all information, the name of these tables listed as below:
(Due to the Server connection collation (Character Set) is “utf8mb4_unicode_ci” (case-insensitive), the actual name in the database schema is a little different from the presented name in this list. For instance, “RoomType” shown as “roomtype” in the database.)

1. Guest
2. Staff
3. Room
4. RoomType
5. BookedRoom
6. Chef
7. Meal
8. Schedule (This name is a key word in SQL, you need to add “ ` ” such as `Schedule` to avoid accidents)

9. OneWeek
10. BookedMeal

1. Guest Table has 8 attributes in it.

   These attributes include “userID”, “username”, “password”, “realName”, “passportID”, “telephoneNumber”, “email”, and “operationTime”.

   The `userID` is an **INT** type attribute, it has a **Primary Key (UNSIGNED AUTO_INCREMENT)** of this table, and it will not be presented to guests. This is because the `userID` is only used for record and prevent duplicates. Furthermore, the `userID` would be utilised as a communication code with different tables or function blocks in the Java program.

   The `username` attribute could store the specified username of each guest. It has a **UNIQUE** Key (and **NOT NULL**) to prevent username duplicates and it is a **VARCHAR (255)** attribute in this table.

   The `password` (**VARCHAR (255) NOT NULL**) is similar to the `username` attribute. However, it does not have a UNIQUE key restriction because the passwords from different guests could be the same.

   The `realName` (**VARCHAR (255) NOT NULL**) is an attribute to store the Real Name of the guest. They cannot choose to skip filling in the blank of this attribute when they are registering an account.

   `passportID` (**VARCHAR (9) NOT NULL**) is an attribute to store the Passport ID of each guest. Due to the international standards, **the length of Passport ID cannot exceed 9 characters and digits combination**.

   `telephoneNumber` (**BIGINT UNSIGNED NOT NULL**) is an attribute to store the Phone Number of guests. The Phone Number format in this system is the same as the guests’ Phone Number **excluding blank, underscore, or dashed line (“ + ”, “ ”, “ _ ”, “ - ”)**. For example, “+86 12332112356” should be written as “8612332112356”.

   `email` (**VARCHAR (255) DEFAULT ‘’**) attribute is used to store the email of guests. It is an optional attribute because there is a possibility that the guest does not have an email. Thus, it could choose to skip the filling step on this attribute.

   `operationTime` (**TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP**) is an attribute to capture the latest information modified date and time. It would be initialised by “NOW()” function in the Java program. It could also be utilised to promote this system with mechanism of optimistic locking to support multiple threads in the future.

   An improvement suggestion of this table: The `username` and `password` are always used when guests log into their accounts. Meanwhile, the DBMS will scan through the entire table to find corresponding data. Thus, a large-scale Guest table which includes detailed information of each guest would reduce the running efficiency. One possible solution is to separate the guest table into two tables. The `Login` table only has userID (Primary Key), username and password, but another `Information` table has the detailed information of each guest. This `Information` table also has the `userID` attribute and reference the Primary Key of `Login` table. This solution will not only enhance the running efficiency of DBMS, but also provide other possibilities which means that users could add multiple personal information as the alternative information in the database. The reason why I

do not choose to apply the above solution into this table is that the data set is not extremely large at the initial stage. Although the solution provided above has two advantages, the Cartesian Product will decrease the running efficiency when guests want to update their personal information. Therefore, the Cartesian Product is a more noticeable problem compared with scanning efficiency reduction problem when the data of guests is rare. Consequently, I chose to use the large `Guest` table in this program. In the future, when the data of guests are numerous, I could consider alter the table structure as the same one as the solution stated above.

2. Staff Table has 5 attributes.

   `staffID` (**INT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) is used to distinguish the different staff in this hotel.

   `username` (**VARCHAR (255) NOT NULL**) is an attribute to store the staff name or alias in this table.

   `password` (**VARCHAR (255) NOT NULL**) is used for staff login operation.

   `telephoneNumber` (**BIGINT UNSIGNED NOT NULL**) is an attribute to store the phone number of staff. It has the same format rule as guests phone number input in the Java program.

   `operationTime` (**TIMESTAMP NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP**) could record the latest update time on staff information. (Staff could only change their password in this system.) This attribute is also set for future function extension, it can cooperate with optimistic locking to implement a multiple threads operation.

   Notice: The new Staff could only register their account by submitting their application to the Administrator of this database. Afterwards, the Administrator would tell him or her the `username` and default `password` in this hotel management system (HMS). They are recommended to change their `password` as soon as possible after they get their own account.

   The improvement suggestion of this table: (Almost the same as the Guest Table improvement suggestion. However, this alteration method is not very recommended for this table, because staff do not have complex personal information compared with guests' information.)

3. Room Table has 2 attributes.

   `roomID` (**INT UNSIGNED PRIMARY KEY**) is used to store the Room ID of each floor, it was formatted as "XY" (X stands for the number of floors, Y stands for the room number in each floor.) For example, the room number 6 in the 6^{th} floor is "606".

   `roomTypeID` (**TINYINT UNSIGNED NOT NULL**) is specify the four different types of room in the hotel. It references the `roomTypeID` attribute in `RoomType` Table.

4. RoomType Table has 2 attributes.

   `roomTypeID` (**TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) is used as the index to represent 4 types of room in this hotel.

   `roomType` (**VARCHAR (30) NOT NULL**) stored the actual name of these four types of room.

5. BookedRoom Table has 6 attributes.

   `bookedRoom_ID` (**INT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) is an order code of booked room. It can separate different booked room orders with different ID code. This design could help guests to modify their booked room with only order codes specified in program. It is more convenient than input other information to specify a room order.

   `userID` (**INT UNSIGNED NOT NULL**) is used to specify who want to live in the hotel. It has referenced `userID` attribute in Guest Table.

   `roomID` (**INT UNSIGNED NOT NULL**) is used to specify which room that the guest wants to live in. It has referenced `roomID` attribute in Room Table.

   `checkInDate` (**DATE NOT NULL**) stores the guest's check-in date.
   `checkOutDate` (**DATE NOT NULL**) stores the guest's check-out date.

   (Two attributes shown above determine the time interval that a specific guest live in the hotel.)

   `operationTime` (**TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP**) records the latest modify time of a booked room order. It would be initialised by "NOW()" function in the Java program. This attribute could cooperate with optimistic locking to implement a multiple threads operation.

6. Chef Table has 2 attributes.

   `chefID` (**TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) stands for the ID of different chefs. It will work as a communication code in the Java program. Due to the number of chefs are not extremely large, the `chefID` was set to TINYINT data type to save the storage (or memory) space. It could be changed to INT if the number of chefs is exceeded 127.

   `chefName` (**VARCHAR (20) NOT NULL**) stores the Real Name of chefs. Because of the longest name in these chefs are not exceed 20 characters, the VARCHAR was set to 20 length for storage (or memory) saving.

7. Meal Table has 4 attributes.

   `dishesType_ID` (**INT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) is the ID of different combination between dishes and chefs. It shows all available meal in the hotel.

   `chefID` (**TINYINT UNSIGNED NOT NULL**) refers to the chef of each meal. It has referenced the attribute `chefID` in Chef Table.

   `dishes` (**VARCHAR (60) NOT NULL**) stores the name of each dish. Each dish corresponds to a chef in this table. For the aim of storage or memory space saving, the VARCHAR is set to 60 characters. It can be changed if new added name of dish exceeds this length.

   `price` (**FLOAT UNSIGNED NOT NULL**) stores the price of per dish. It could be a float type data, but in this table all price data are integer. However, the new added dish might have float point number of prices.

An improvement suggestion of this table: This table could be separated into two tables, because there exists a condition that one meal corresponds to three chefs. According to Normalisation, it should be separated into two tables consistent with the third Normal form. However, there are only two meals that have this type of relationship with chefs. Meanwhile, the final result of these two separated tables should be recombined together to display a valid relation. It is noticeable that there exists Cartesian Product that will reduce the running efficiency. Therefore, I stayed it as an entire Meal Table at this initial stage. If the meal and chefs have increased to a large number and the "1:m" relationship has become regular, I will choose to separate this table to consistent with the third normal form.

8. `Schedule` Table has 3 attributes.

   `weekday_ID` (**INT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) stores the different ID to separate different chefs and their workday.

   `chefID` (**TINYINT UNSIGNED NOT NULL**) stores the chefs' ID. It has referenced the `chefID` attribute in Chef Table.

   `day_ID` (**TINYINT UNSIGNED NOT NULL**) stores the corresponding workday for different chefs. It has referenced `day_ID` attribute in OneWeek Table.

9. OneWeek Table has 2 attributes.

   `day_ID` (**TINYINT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) stores the corresponding day number for one week. It is reasonable that one week could be represented by 7 different numbers. Therefore, it does not need to declare a large data type for this attribute. The TINYINT type is the appropriate minimum data type for this range.

   `day_Name` (**VARCHAR (10) NOT NULL**) stores the full text name of each day in one week. Because of the name which has the maximum length in one week is not exceed 10 characters, the VARCHAR set as 10 characters for storage or memory space saving.

10. BookedMeal Table has 8 attributes.

    `bookedMeal_ID` (**INT UNSIGNED PRIMARY KEY AUTO_INCREMENT**) stores the order code of meal. It can help guests to modify the meal in convenient approach.

    `userID` (**INT UNSIGNED NOT NULL**) stores the guest who book the meal. It has referenced `userID` attribute inGuest Table.

    `bookedRoom` (**BIT NOT NULL**) verify that whether the customer living in a room when he or she enjoying the meal in the hotel. "0" stands for false (Not live in the hotel) and "1" stands for true (Live in the hotel).

    `dishesType_ID` (**INT UNSIGNED NOT NULL**) stores the meal type and chef. This attribute represents most of meal information in each tuple in this table. It has referenced `dishesType_ID` attribute in Meal Table.

`orderDate` (**DATETIME NOT NULL**) stores the order date and time of guests.

`serveDate` (**DATETIME NOT NULL**) stores the service date and time for guests to enjoy the meal.

`count` (**INT UNSIGNED NOT NULL DEFAULT 1**) stores the specified count for each meal by guests. The default count for each dish is 1.

`totalPrice` (**FLOAT UNSIGNED NOT NULL**) stores the total price of each meal order. If the guest book the meal two days in advance and living in hotel on the service date, they will get 20% discount on the total price.

The Entity-Relationship (E/R) Diagram:

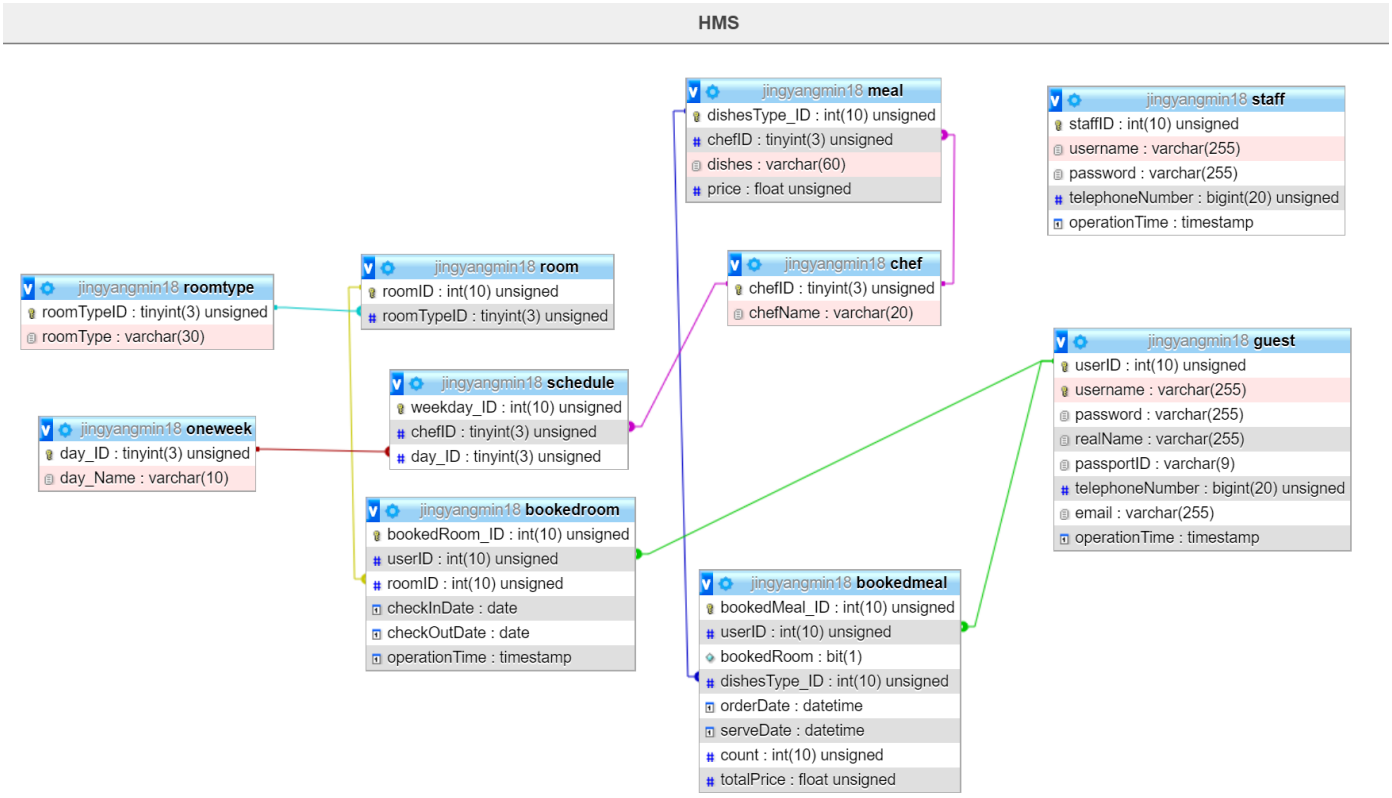


Figure 1: The E/R Diagram of all relations in Hotel Management System

2. The Program Design:

<u>General structure and preparation in early stage</u>:

Although the MySQL DBMS has provided numbers of approaches to implement data storage issue, the table design is also important because it will affect program and DBMS running efficiency. In this program, I have not chosen to utilise the absolutely OOP concept when I was designing it. It is feasible that directly insert an object into one table, but it is not necessary. This is because the user inputs should be checked before insert these data into a table, and create an object is still need these processes. It is noticeable that create an object for “Guest” or “Booked Room” would need intermediate steps. Thus, I just verify these data at first and then insert them into a table directly.

In this program, to establish the connection with the specified database is one of the most important

tasks. In order to get a connection with the XJTLU mariaDB database schema which has been named as **jingyangmin18**, (The URL is **jdbc:mysql://csse-mysql.xjtlu.edu.cn:3306/jingyangmin18**) we need to configure the Connector/J package (JDBC) into the library of our project in the IDE. I used **jdk1.8.0_251** and **mariadb-java-client-2.6.0.jar** in my **IntelliJ IDEA project**. To optimise the configuration efficiency, I used **java.util.ResourceBundle** to directly configure the URL, Username, Password in it. You can also find it in (Path: **project/Hotel_Management_System/src/information.properties**). When you open this properties file, you will find 3 configuration contents including: **1. URL, 2. administrator, 3. password**. You can modify the contents shown on the right side of equal sign ("=") to connect the program to your own database. The prepared contents in this file are the configuration of my database schema, you can also unchanged it and directly connect to my database.

All the function in this program have been divided into 8 classes. The project includes one main class named as "HMS", and a package named as "utils". Almost all the function was implemented in this utils package. This utils package includes: "Guest", "Staff", "Booking", "StaffOperation", "DB_Utility", "TablePrinter", and "ShowWindow". The last two class is used to display the table in GUI format. They involve the java swing to implement this display function.

This program could be descripted as a robust system, it supports not only number codes interaction, but also supports full text name interaction, message verification (with regular expression), cancel input process at any time, transaction, and a number of functions about concurrency supporting. This program also provided large extension space for following developers to create new function in its body. Because of my design concept is to directly insert data into table, so most of classes in the program is not support object instantiation. To prevent it happened, except the "ShowWindow" class, the constructors of these classes have been already set to private and prevent object instantiation from outside of their class body.

For the different panel display, I used "byte code" to implement the communication among different function blocks. For example, 0 stands for welcome page, 1 stands for Guest welcome page, 2 for Staff welcome page, -1 for quit the system. There are 23 different codes in the program, each of them has different function. They could be separated into different function blocks, but all of them combined together to form an entire HMS program. Therefore, it has a high portability because of different part is not very tightly connected together.

To prevent resources consuming when the program is running, I set the Connection, Statement, and ResultSet to be created in separate methods, then close all of them (De-allocate resources) after the operation between program and database has done. This strategy could decline the stream consumption of each separate running program when they running concurrently.

The program also shown accurate "exception" message to user, these messages are surrounded or upper bounded by "=======" sign to help users identify what error they made in the past time typing contents.

For more detailed information of this program, you can also refer to the Javadoc comments in these source codes.

The detail functions in this program:

When you start the program, you will get a welcome message that remind you are using this system. Following this, the main(String args[]) method would call the processing() method to do all other parts in the program. In the processing() method, it will firstly call the welcome() method to display the initial

control panel of whole system, which includes options of Staff Mode, Guest Mode, and Quit system. Then user could type in their options and program will process this input with switch-case block in an infinite loop (while(true)). They must choose one of these three options, otherwise they cannot do other things in this program. This options design is also consistent with all the other options choosing mode in the whole program.

When the user chooses one option from these three options (Staff Mode, Guest Mode, Quit system), the welcome() method would return a communication code for the caller processing to decide which mode it should to engage in on the next step. The "byte step" is a holder of this communication code, and the mode selector has the same structure as options choosing mode. If a user log in their account in the following steps, the information int array would hold both communication code and login ID simultaneously. The ID will help program to know that which account is running on this platform, and the ID could also help the sql statement to filter out tuples in a table.

When either guest or staff log in their account, they will get a feedback from program. The message notify you have log in the system successfully, and your username will also be shown on the control panel. The normal feedback message is surrounded or upper bounded by "---------". Meanwhile, either staff or guest could update their personal information (staff could only update their password) in this program no matter he/she log in their accounts before or after the login stage.

In guest register function block, the regular expression was used to verify whether the user input is valid or not. This is also the same when guests want to update their personal information. In the update information function block, the transaction is supported by using **connection.setAutoCommit(false)** and providing **rollback()** and **commit()** options. That means user could choose cancel all changes and back to previous options, or save all changes and back. Meanwhile, they can also continue their updating process if they want to change some other information. The transaction supporting has the same operation in the cancel room or meal function blocks.

The transaction isolation settings are different between modifying room booking information and cancel meal orders. This is because they use the different mechanism to implement their function.

The booked room modification implemented by passing connection argument to the specified method, and these invoked methods would set the transaction isolation level to read committed. In this program it shows as (**Connection.TRANSACTION_READ_COMMITTED**).
The reason why I utilised this mechanism is that when a guest is changing the booked room orders, their specified new data might be occupied by other guest at the same time. Due to the default isolation level is (**Connection.TRANSACTION_REPEATABLE_READ**), the program would check their new data based on the valid information at the transaction started moment. There is possible to happened other guest occupied this time interval in this short period. Therefore, in the modify booked room information function block, I only need to consider other guests' operation at the same time.

The cancel meal orders function block is different from modify booked room information. This is because I chose independently create new connection in the invoked methods instead of passing connection argument to them. This strategy could simplify the operations of programmers, but it needs to set the transaction isolation level to read uncommitted (**Connection.TRANSACTION_READ_UNCOMMITTED**) to support the transaction in the caller method. (But it still needs to reset to the default repeatable reads: **Connection.TRANSACTION_REPEATABLE_READ**) In the cancel meal function block, there is no need of avoid time conflict among different guests. This point is a reason that why I did not choose to pass connection argument to the invoke methods. However, if the system wants to become multiple threads

supporting in the future, this design would refuse updating without downtime. For example, administrator might use transaction to update meal table or chef table when the program is running on different devices. Due to the uncommitted reads, some guests might read the dirty data when the data in these tables are alerting by administrator in his own transaction. As a consequence, if this program wants to support multiple threads, it needs some tiny changes in the specific structure.

There are some test data in these tables. For example, the following table presented some key information in Staff Table:

Table 1: Some Key Attributes in Staff Table

staffID	username	password	telephoneNumber	……
1	mike	2333666	8615833356871	…
2	tigger	2333	987658632	…
3	scott	111	2123535768	…

2. User Part:

User manual:

Notice: You must connect with database before you run the program. Otherwise, you cannot use this system properly. The connection failure would cause serious problem.

1. Guest Mode

When you start this system, you will find three options initially. (The current time is closely related to your current time of system clock.)

```
                              *-------------------------------------*
                              *     Welcome to use this system!     *
                              *-------------------------------------*
***********************************
Current time --> 2020-06-05 22:30:06
***********************************
1. Guest Mode   (Type in "1" or full name of mode to select this mode)
2. Staff Mode   (Type in "2" or full name of mode to select this mode)
3. quit the system (Exit)
Please type in the corresponding number of different modes to select your identity:
```

You can choose 1 to select guest mode.

```
                              *-------------------------------------*
                              *     Welcome to use this system!     *
                              *-------------------------------------*
***********************************
Current time --> 2020-06-05 22:30:06
***********************************
1. Guest Mode   (Type in "1" or full name of mode to select this mode)
2. Staff Mode   (Type in "2" or full name of mode to select this mode)
3. quit the system (Exit)
Please type in the corresponding number of different modes to select your identity: 1

***********************************
Current time --> 2020-06-05 22:31:47
***********************************
1. Guest Login
2. Guest Sign Up
3. Update Personal Details
4. Back to previous options
5. quit the system

(Now in: Guest Mode)
Please type in the corresponding number to choose your option:
```

Afterwards, you can sign up a new account and login it. You can also update your previous account by choosing 3.

```
**********************************
Current time --> 2020-06-05 22:34:06
**********************************
Please type in your username (type "Return" to cancel input): minicap
Please type in your password (type "Return" to cancel input): 333666
```

On this stage, you need to type in the correct username and password to verify your identity. If you input your username and password correctly, you will get these options.

```
------------------------------------------------------------------
Verification succeed! You can change your personal information now.
------------------------------------------------------------------

Your latest information listed as below:
--------------------------------------------------
```

Continue. . .

```
Your latest information listed as below:
--------------------------------------------------
Your username: minicap
Your password: 333666
Your Real Name: mike simon
Your Passport ID: WCMM22233
Your Phone Number: 22222256
Your email: 2553755@123.com
--------------------------------------------------
1. Username
2. Password
3. Real Name
4. Passport ID
5. Phone Number
6. Email
7. Cancel All Updates and Return
8. Save and Return to previous page
9. quit the system (Not record your changes)

(Now in: Guest Information Alteration Mode)
Please type in your choice:
```

You can modify your personal information here. After changes, you can cancel all of them or save them and back to previous page. Then you can log in your account.

```
**********************************
Current time --> 2020-06-05 22:40:00
**********************************
1. Guest Login
2. Guest Sign Up
3. Update Personal Details
4. Back to previous options
5. quit the system

(Now in: Guest Mode)
Please type in the corresponding number to choose your option: 1

**********************************
Current time --> 2020-06-05 22:40:23
**********************************
Please type in your username (type "Return" to cancel input): minicap
Please type in your password (type "Return" to cancel input): 333666
```

You can do these operations in the system.

```
--------------------------------------------------------------------------------
Welcome back! minicap
--------------------------------------------------------------------------------

**********************************
Current time --> 2020-06-05 22:41:04
**********************************
1. Book Rooms
2. Modify Booked Rooms
3. Book Meal
4. Cancel Booked Meal
5. Update Personal Details
6. Log out
7. quit the system

Account: minicap
Please type in the corresponding number to choose your option:
```

If you have already booked a room in option “1”, you can choose modify booked rooms to change the room order information.

```
**********************************
Current time --> 2020-06-05 22:47:31
**********************************
1. Modify Check-in Date
2. Modify Check-out Date
3. Cancel Booked Room
4. Back to previous page
5. Log out
6. quit the system

Please type in the corresponding number to choose your option:
```

You can change your check-in date and check-out date, or cancel the booked room order. However, there is a restriction is that you cannot modify the check-in date if today is the check-in date. Similarly, you cannot modify the check-out date if you have already passed the check-out date. You cannot set check-in or check-out date before today. The room cannot be cancelled if today is the check-in date. You can find the specific notification message when you operating this system.

For the functions of Guest mode, you can login your account to book or cancel rooms or meals. You can also register an account at first, or modify your personal information before login. You can cancel current operation at almost any time by type in “return”. The characters for all operations are case insensitive. (Except passport ID (Uppercase), but it will be automatically converted by the system.)

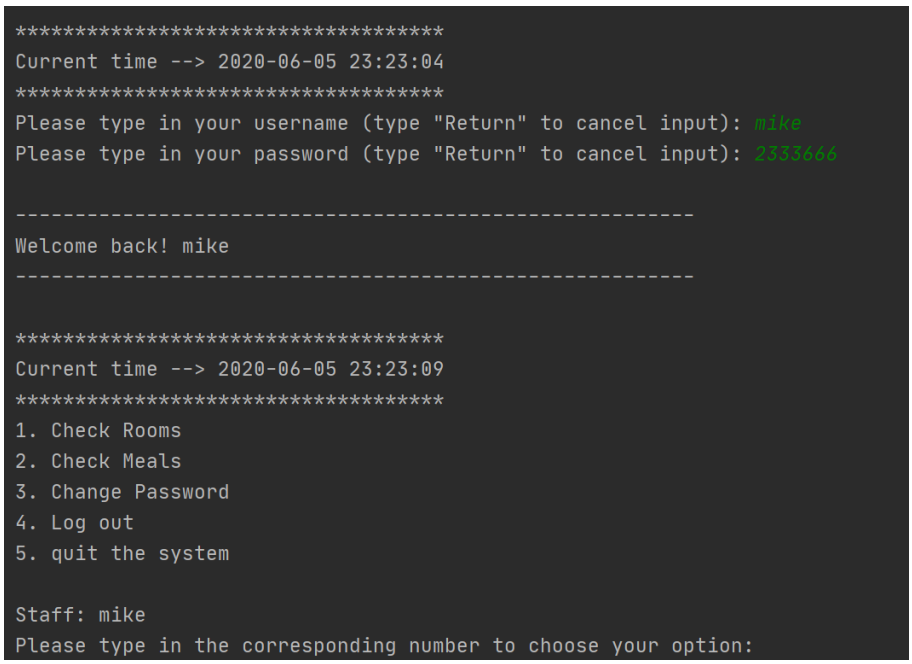
2. Staff Mode

If you choose to engage in the guest control panel in accident, you can choose “back to previous options” and choose the correct staff mode again. You are also capable to quit the system in the guest login panel. After you choose the correct option, you will find these options.

```
**********************************
Current time --> 2020-06-05 23:20:30
**********************************
1. Staff Login
2. Change Password
3. Back to previous options
4. quit the system

(Now in: Staff Mode)
Please type in the corresponding number to choose your option:
```

The password changing is also supporting transaction. After log in your account, you can see these options.



You can check booked rooms information by choosing one specific type of room, or you can search all booked rooms information in detail. You can also check the meal orders with guests' real name, or just search all the detailed meal booked information of all guests.