INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

AAPP004-4-2-JP

JAVA PROGRAMMING

HAND OUT DATE: 10th JANUARY 2022

HAND IN DATE: 10th APRIL 2022

WEIGHTAGE: 60%

INSTRUCTIONS TO CANDIDATES:

1 **Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing).**

2 **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**

3 **Cases of plagiarism will be penalised.**

4 **The assignment should be bound in an appropriate style (comb bound or stapled).**

5 **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**

Student Name: Teo Kai Yii

Student TP Number: TP058618

Table of Contents

1.0 Introduction

This room reservation system is created for Magical Paradise Resort Room Booking System. The primary objective of this system is to manage room bookings for Magical Paradise Resort with two views which includes Jungle View and Sea View. Each view contains ten rooms. The daily price of Magical Paradise Resort is RM350 per room. The main user of this system is the staffs of the Magical Paradise Resort.

Staff can login to the system and reserve a room for customers. The system provides access for employees to select a room for a particular day. Furthermore, the system will display all available rooms on a specific day. The staff may choose the room the customer want to reserve. When making a reservation, staff need to request information from the customer which are customer's name, IC/Passport number, phone number, email address, room number, number of days and etc. When customer make a reservation, the booking details will be saved in txt file and the reserved room will be restricted for the chosen days. Other than that, staff can manage the details of room reservation which modify, delete, search, view bookings is allowed.

After staff help customer to book a room, the system will generate receipt for the customers. The receipt will include customer's information, room details, total amount, and taxes. There are also a 10% of service tax of total accommodation rates and a tourist tax of RM10 per night.

There are also some additional features which staff can sort bookings by past bookings, bookings now, and future bookings. This would allow staff to differentiate bookings easily which staff can sort past, now, or future bookings and find a specific booking more efficiently. Furthermore, staff can also register a booking while payment is not done. There will be a make payment function for the staff to update payment.

2.0 Source Code with explanation

2.1 Java Packages and API

The Java API is a free-to-use library of prewritten classes provided in the Java Development Environment. There are components for handling input, database programming, and even more in the library. Packages and classes are used to arrange the library. To access Java API, import keyword is necessary to import a class or package from the library. Classes can be imported individually together with its methods and attributes. Other than that, classes can also be imported in an entire package which includes all the classes in the provided package (W3Schools, 2022).

2.1.1 Import a Package

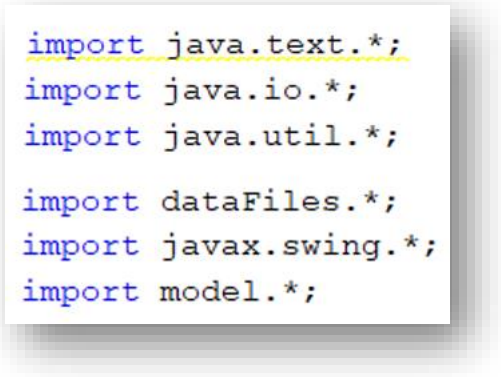


Figure 1: import packages

As shown in the figure above, there are an asterisk sign (*) at the end of each statement to import the entire package. The table below explains the purpose of each package.

Table: Java Packages (Oracle, 2020)

Package	Description
java.text	Offer classes and interfaces for processing text, dates, numbers, and messages in a natural language independent way.
java.io	Support system input and output via data streams, serialisation, and the file system.
java.util	The collections framework, heritage collection classes, event model, date and time facilities, internationalisation, and various utility classes.
dataFiles	Self-created package to manage and check data from txt file.
javax.swing	Contains a collection of components that function the same on all platforms to the greatest possible extent.
model	Self-created package to define fields and ensures all variable can be accessed by setter and getter method.

2.1.2 Import a Class and Exception

The figures below show how to import a class or exception from packages which is to place the class name behind of the package name.

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

Figure 2: Classes of the Package java.io

The classes in the figure above are from the Package java.io. BufferedWriter class write text to a character-output stream, buffering characters to allow efficient writing of single characters, arrays, and strings. Furthermore, FileWriter class can create character files. Moreover, PrintWriter class print text-output stream in the formatted representations of objects. Additionally, FileInputStream class includes another input stream which it utilises as its primary source of data, or even modifying that data to offer extra functionality. IOException indicates an input or output exception had occurred. Lastly, FileNotFoundException indicates the action to open a file in the provided pathname has failed (Oracle, 2020).

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
```

Figure 3: Classes of the Package java.text

The classes in the figure above are from the Package java.text. SimpleDateFormat class is used to format and parse date. ParseException is used to indicate errors occurred unexpectedly while parsing (Oracle, 2020).

```
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.HashSet;
import java.util.concurrent.TimeUnit;
```

Figure 4: Classes of the Package java.util

The classes in the figure above are from the Package java.util, Package java.util.concurrent and Package java.util.logging. ArrayList class allows list interface to be implemented as resizable array. Calendar class contains methods to convert a given field of time and the calendar fields. Date class includes precise point in time, down to the millisecond. Locale class define the particular geographical location. Scanner class uses regular expressions to parse primitive types and strings. HashSet class can implement set interface that is supported by hash table. TimeUnit define time durations that can convert between units as well as performing timing and delay operations in these units. Level class specifies a collection of standard logging levels that may be used to manage output of logs. Lastly, Logger is used to record messages for a particular system or application component (Oracle, 2020).

```
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
```

Figure 5: Classes of the Package javax.swing

The classes in the figure above are from the Package javax.swing and Package javax.table . JOptionPane class is used to provide a pop-up message dialog box. JTextField class allow user to edit a text. DefaultTableModel store cell value objects (Oracle, 2020).

```
import model.BookingDetails;
import model.Staff;
```

Figure 6: Classes of the Package model

The classes above are from the self-created Package model. BookingDetails class is used to define fields and access booking details with the get set method. Staff class is used to define fields and access staff information with the get set method.

```
import dataFiles.CheckAllData;
import dataFiles.DataIntoFiles;
import dataFiles.CheckBookings;
import dataFiles.EditBookings;
import dataFiles.UpdateBookings;
import dataFiles.CheckStaff;
```

Figure 7: Classes of the Package dataFiles

The classes above are from the self-created Package dataFiles. CheckAllData class will get a string array of all the information from a txt file. DataIntoFiles class will store information into txt file. CheckBookings class will return information required to do checking. EditBookings class will allow staff to edit and delete rows from the txt file. UpdateBookings class will update the modified string array to the file. CheckStaff class will check staff information and validation.

2.2 Variables

In Java, a variable is a data container that saves data values during the execution of a Java application. Every variable is allocated a data type, which specifies the kind and amount of value it may hold (GeeksforGeeks, 2022).

```
//get value from text field
String getFullName = txtFullName.getText();
String getIc = txtIC.getText();
String getContactNumber = txtContactNumber.getText();
String getEmergencyNumber = txtEmergency.getText();
String getEmail = txtEmail.getText();
String getHomeAddress = txtHomeAddress.getText();
String getStartDate = txtStartDate.getText();
String getEndDate = txtEndDate.getText();
String getNumOfDays = txtNumOfDays.getText();
String getRoomID = txtRoomID.getText();
```

Figure 8: String data type

As shown in the figure above, String is initialized with “String” keyword. The name of String variable is followed by the “String” keyword. After the equal sign (=), there will be the data to store in the variable. String is a non-primitive data type as it is defined by programmer. The example in the figure above is used to store value retrieved from the JTextField into the String variable.

```
//calculate total & service charge
double dTotalRoomPrice = 350 * Integer.parseInt(getNumOfDays);
double dServiceCharge = dTotalRoomPrice * 0.1;
double dTotalAmount = dTotalRoomPrice + dServiceCharge + 10;
```

Figure 9: double data type

The example above shows the double data type. “double” keyword is used to store the calculated numbers. After “double” keyword, there will name of the double variable. The value of the variable is defined after the equal sign (=). Double datatype can store up to 15 decimal digits. As shown in figure above, the double is used to store variable of total room price, service charge and total amount. Calculations is done by using these variables.

```
int i = existingID.size() - 1; //get index
```

Figure 10: int data type

As shown in the figure above, integer is initialized with “int” keyword. The name of int variable is followed by the “int” keyword. After the equal sign (=), there will be the data to store in the variable. int is a primitive data type as the size and type is set. The example in the figure above is used to store the index number by getting the size of list and minus by one.

```
Boolean available = true;
```

Figure 11: Boolean data type

The example above shows the Boolean data type. “Boolean” keyword is used to store the true or false value. After “Boolean” keyword, there will name of the Boolean variable. The value of the variable is defined after the equal sign (=). As shown in figure above, the Boolean is used to store true or false value of the room availability.

```
Date start, end, ytd;

Date ytdDate = sdf.parse(yDate);
```

Figure 12: Date data type

As shown in the figure above, Date is initialized with "Date" keyword. The name of Date variable is followed by the "Date" keyword. After the equal sign (=), there will be the data to store in the variable. Date is a non-primitive data type as it is defined by programmer. The example in the figure above is used to store the date of start date, end date and yesterday's date.

```
String[] bookingDetails = CheckAllData.checkAllData("bookings.txt");

String[] tableRow = {roomBookingData[0],roomBookingData[1],roomBookingData[3],
    roomBookingData[7],roomBookingData[8],roomBookingData[11],roomBookingData[9],
    roomBookingData[10],roomBookingData[17]};
```

Figure 13: Array data type

The example above shows the array data type. Array is used to store various values into an individual variable. Array can be defined by a variable type followed by the square brackets "[]". After variable type and square brackets, there will name of the string array variable. The value of the variable is defined after the equal sign (=). The bookingDetails string array get the values from the checkAllData class. The tableRow string array used the curly braces ({}) and separate with commas to insert values into the string array.

```
final long MILLI_A_DAY = 1000 * 60 * 60 * 24;//millisecond in one day
long ytdDate = currDate.getTime() - MILLI_A_DAY;
```

Figure 14: long data type

As shown in the figure above, long is initialized with "long" keyword. The name of long variable is followed by the "long" keyword. After the equal sign (=), there will be the data to store in the variable. Date is a primitive data type as the size of type is set. The example in the figure above is used to store the milliseconds in one day.

2.3 Control Structure

Control structure in Java provide control statements to control the flow of Java codes. “if” statement is used to examine specific conditions. “else” statement will be executed if the “if” statement is false (javaTpoint, n.d.)..

2.3.1 if Statement

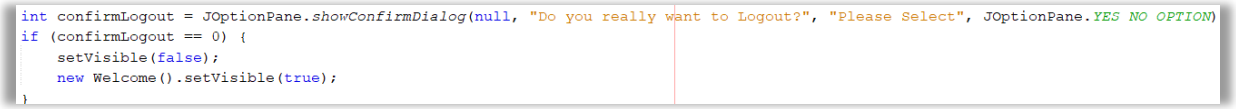
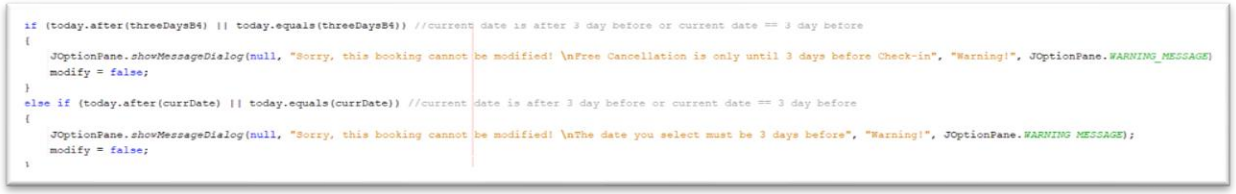


Figure 15: if Statement

As shown in the figure above, confirmLogout integer is used to store the decision of staff. There will be a pop-up message dialog box to ask staff to confirm logout. If staff answer yes, confirmLogout will store value of “0”. The “if” statement will examined if the confirmLogout integer is equal to “0”, if the statement is true, the current page will be closed and the staff will be redirect to the welcome page.

2.3.2 if-else Statement



```
if (today.after(threeDaysB4) || today.equals(threeDaysB4)) //current date is after 3 day before or current date == 3 day before
{
    JOptionPane.showMessageDialog(null, "Sorry, this booking cannot be modified! \nFree Cancellation is only until 3 days before Check-in", "Warning!", JOptionPane.WARNING_MESSAGE)
    modify = false;
}
else if (today.after(currDate) || today.equals(currDate)) //current date is after 3 day before or current date == 3 day before
{
    JOptionPane.showMessageDialog(null, "Sorry, this booking cannot be modified! \nThe date you select must be 3 days before", "Warning!", JOptionPane.WARNING_MESSAGE);
    modify = false;
}
```

Figure 16: if-else Statement

The purpose of this "if-else" statement is to validate the rules to modify bookings. In figure above, *today* variable refers to the today's date, *threeDaysB4* variable refers to three days before the previous Check-in date, and *currDate* variable refers to three days before the current date. The if-else statement is used to evaluate if the selected date is available to modify. If today's date is after three days before the previous Check-in date or today's date is equal to three days before the previous Check-in date, there will be pop up message to tell staff that booking cannot be modified and free cancellation is only available until three days before the previous Check-in date. If this is not true, "else" statement will be executed. There will be a second "if" statement which check if the today's date is after three days before current date or today's date is equals to three days before current date, a pop-up message will notify staff that the modify date selected is not three days before the current date. Boolean modify is set to false if the booking is not available to modify, else the Boolean modify is remain true.

2.3.3 Nested if-else Statement

```
Boolean valid = loginValidation(getUsername,getPassword);

if(valid)
{
    Staff staff = new Staff();
    staff.setUsername(getUsername);
    staff.setPassword(getPassword);
    Boolean loginCheck = CheckStaff.checkLoginCredential(staff);
    if(loginCheck){
        clean(); //if anything happen
        setVisible(false);
        new Dashboard().setVisible(true);
    }else {
        JOptionPane.showMessageDialog(null, "Username or Password wrong !","Error!",JOptionPane.WARNING_MESSAGE);
    }
}
```

Figure 17: Nested if-else Statement

The purpose of this nested if-else statement is to validate staff login details. There will be a Boolean valid at first which the username and password will be validated and return a true or false value. If the Boolean valid is true, the codes within the curly braces will be executed. In the if statement, there is another if-else statement within which makes it a nested if-else statement. If the staff credential is valid, staff will be lead to the dashboard page, else there will be pop up message to warn staff that the username or password entered is wrong.

2.4 Selection Control Structure

Switch statements in Java are comparable to if-else-else statements. A single case is run based on the variable that is being switched in the switch statement, which includes different blocks of code called cases (javaTpoint, n.d.).

```
switch(type) {
    case "view":
        paymentBtn.setVisible(false);
        modifyBtn.setVisible(false);
        deleteBtn.setVisible(false);
      break;

    case "manage":
      break;

    case "modify":
        paymentBtn.setVisible(false);
        deleteBtn.setVisible(false);
      break;

    case "delete":
        paymentBtn.setVisible(false);
        modifyBtn.setVisible(false);
      break;

    case "payment":
        modifyBtn.setVisible(false);
        deleteBtn.setVisible(false);
      break;

    default:
      break;
  }
```

Figure 18: ViewBooking.java

This Switch case is used to manage button functions of the view booking page. This switch case is located in the constructor of ViewBooking with parameters of booking ID and type. Switch statement has a bracket that contains type variable as its expression. Case will follow by variable, if expression matches the variable, statements will be executed, and break statement will terminate the switch case. If none of the case is executed, default statement will be executed. If the type in the given parameter is view, payment button, modify button and delete button will be hidden. If type is equal to case of manage, the break statement will terminate the switch block. If type is equal to case of modify, payment button and delete button will be hidden, which only modify button will be shown. If type is equal to case of payment, modify button and delete button will be hidden, which only payment button will be shown. Lastly, the default statement will use the break statement to terminate the switch block.

2.5 Looping Structure

Looping structure run a piece of code several times while a specified condition is true. The set of code is repeated until it reached a certain circumstance. There are three different forms of loops that all work about the same (javaTpoint, n.d.).

2.5.1 for loop

```
//store everything
StringBuilder bookingBd = new StringBuilder();
for (int i=0; i<roomBookingData.length; i++) {

    bookingBd.append(roomBookingData[i]);

    //last element no need seperator
    if (i == (roomBookingData.length - 1)){
        break;
    }
    bookingBd.append("//");
}
```

Figure 19: CheckBookings.java

The for loop above is used to append each booking details into a string with a separator for each detail. StringBuilder object is created and used to append each detail and separator with the help of the append method (Oracle, 2020). The for loop starts with an initialization of “int i = 0” which means integer i has a value of “0”. Condition of the for loop is “i<roomBookingData.length” which means the loop will be repeated as i is less than the length of string array, roomBookingData. Increment of the for loop is “i++” which means the value of i will increase by 1 when each time the for loop is looped. In the for loop above, the loop starts from 0 and increase by 1 each time until each string of roomBookingData is append. In the for loop, if variable i is equal to length of string array minus one, that single loop will break and the new loop will start again. This is because the last element does not need the separator.

2.5.2 Nested for loop

```
for(int i = 0; i < dtm.getRowCount(); i++)
{
    StringBuilder bookingBd = new StringBuilder();
    for(int j = 0; j < dtm.getColumnCount(); j++)
    {
        bookingBd.append(dtm.getValueAt(i, j).toString());
        //last element no need seperator
        if (j == (dtm.getColumnCount() - 1)){
            break;
        }

        bookingBd.append("//");
    }
    String booking = bookingBd.toString();
    tableInfo.add(booking);
}
```

Figure 20: ViewAllBookings.java

The for loop above is used to append each booking details into a string with a separator for each detail. StringBuilder object is created and used to append each detail and separator with the help of the append method (Oracle, 2020). The outer for loop starts with an initialization of “int i = 0” which means integer i has a value of “0”. Condition of the for loop is “i<dtm.getRowCount()” which means the loop will be repeated as i is less than the length of row count of table. Increment of the for loop is “i++” which means the value of i will increase by 1 when each time the for loop is looped. The inner for loop starts with an initialization of “int j = 0” which means integer j has a value of “0”. Condition of the for loop is “i<dtm.getColumnCount()” which means the loop will be repeated as i is less than the length of column count of table. Increment of the for loop is “j++” which means the value of j will increase by 1 when each time the for loop is looped. In the for loops above, the loop starts from 0 and increase by 1 each time until each cell of data is appended. In the for loop, if variable is equal to column count of table minus one, that single loop will break and the new loop will start again. This is because the last element does not need the separator. The outer loop gets the index number of table row, inner loop get the index number of column row.

2.5.3 for-each loop

```
//loop to get each booking from booked details arraylist
for (String eachBooking : bookingDetails) {
    String[] roomBookingData = eachBooking.split("//"); //split the booking info

    //get details of one booking
    if(bookingID.equals(roomBookingData[0]))
    {
        for (String eachDetail : roomBookingData)
        {
            existingList.add(eachDetail); //store each value in array list
        }
    }
}
```

Figure 21: CheckBookings.java

The for-each loop above is used to get the booking details that matches the booking ID. This loop does not need a variable, the loop will loop each value in the string array. In the example above, String data type and variable eachBooking is follow by ":" and with the string array bookingDetails. The for loop has a string array of roomBookingData created and split details of eachBooking with a separator of "//". After each detail is split and store into the roomBookingData. If the bookingID given and the bookingID in the loop matches, all booking details of that bookingID will be added into the arraylist.

2.5.4 while loop

```
// get all rows from file
while (s.hasNextLine()) {
    list.add(s.nextLine());// adding each booking to booking LIst
}
```

Figure 22: CheckAllData.java

The while loop above is used to get all rows from the txt file and add each booking to the arraylist. “s” is the scanner object to scan the file. “hasNextLine()” is the method to check if the file has another line. “nextLine()” is the method to get next line in the txt file. While the condition of “s.hasNextLine()” is true, following line of txt file will be added into the arraylist (javaTpoint, n.d.).

2.6 OOPs Concept

OOP refers to Object-Oriented Programming. Object-oriented programming is about building objects that include both data and methods, whereas procedural programming is about creating procedures or methods that execute actions on the data. OOP is a more efficient and straightforward method of programming. This Resort Booking System have a good structure thanks to OOP (W3Schools, 2022).

2.6.1 Class

A class is an object's blueprint. We must first declare the class before we can construct an object

Figure 23: Staff.java

As shown in the figure above Staff class is created so the object can be created when object is needed. This class contains fields and methods about staff.

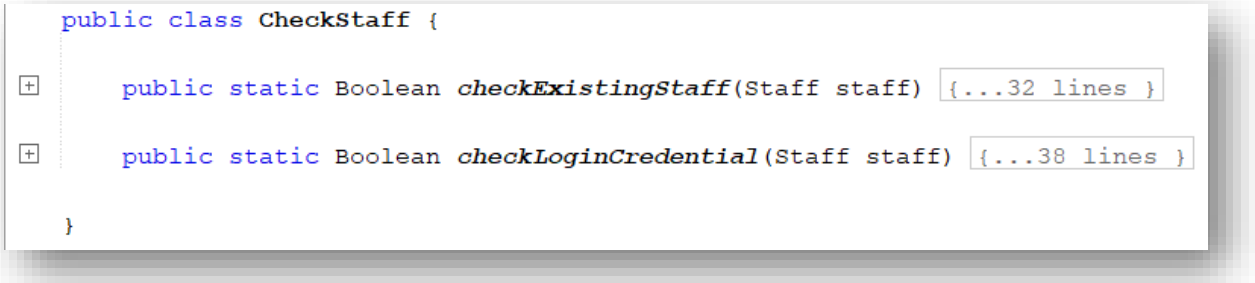


Figure 24: Staff.java

As shown in figure above, CheckStaff class is created to check if username exist and check the validity of login credentials. This class contains methods for two different purposes. checkExistingStaff method returns true if the username already exists. checkLoginCredential method return true if username and password are accurate.

2.6.2 Object

```
BookingDetails BD = new BookingDetails();
BD.setId(idLbl.getText());
BD.setContactNum(contactLbl.getText());
BD.setEmail(emailLbl.getText());
BD.setEmergencyNum(emergencyLbl.getText());
BD.setFullName(nameLbl.getText());
BD.setHomeAddress(homeLbl.getText());
BD.setIC(icLbl.getText());
BD.setInDate(startDateLbl.getText());
BD.setNumOfDay(numOfDaysLbl.getText());
BD.setOutDate(endDateLbl.getText());
BD.setRoomID(roomIdLbl.getText());
BD.setView(roomTypeLbl.getText());
BD.setDate(dateLbl.getText());
BD.setTime(timeLbl.getText());
BD.setTotalAmount(totalAmountLbl.getText());
BD.setServiceCharge(serviceChargeLbl.getText());
BD.setTotalRoomPrice(totalRoomPriceLbl.getText());
```

Figure 25: BookingDetails.java

A class in Java is used to build an object. The BookingDetails class have been constructed, so the class can be used to create the object “BD”. As shown in figure above, the object is used to set the value of fields with the set method in the BookingDetails class.

2.6.3 Encapsulation

```
public class Staff {
    private String fullName;
    private String username;
    private String ic;
    private String email;
    private String contactNumber;
    private String password;
    private String confirmPassword;

    public String getFullName() {...3 lines }

    public void setFullName(String fullName) {...3 lines }

    public String getUsername() {...3 lines }

    public void setUsername(String username) {...3 lines }

    public String getIc() {...3 lines }

    public void setIc(String ic) {...3 lines }

    public String getEmail() {...3 lines }

    public void setEmail(String email) {...3 lines }

    public String getContactNumber() {...3 lines }

    public void setContactNumber(String contactNumber) {...3 lines }

    public String getPassword() {...3 lines }

    public void setPassword(String password) {...3 lines }

    public String getConfirmPassword() {...3 lines }

    public void setConfirmPassword(String confirmPassword) {...3 lines }

}
```

Figure 26: Staff.java

The Encapsulation concepts is implemented in this Staff class which class variables is declare as private and include public get and set method to access and set the value of private variable. This encapsulation concept improves the control over class attributes and methods. The data security has been improved with this encapsulation concept (W3Schools, 2022).

2.6.4 Generalization

```
public class Register extends javax.swing.JFrame {
```

Figure 27: Register.java

The Generalization concept is implemented here which Register class inherits the attributes and methods from javax.swing.Jframe class. Jframe contains a main window and have elements such as buttons, text fields and labels to create a GUI (javaTpoint, n.d.). The “extends” keyword extend the Register class. Register class is the child which is the subclass. While on the other hand, javax.swing.JFrame will be the parent which is also the superclass (W3Schools, 2022).

2.6.5 Constructor method

Constructor is similar to method. When constructor is called, memory for the object is allocated in the memory which constructor is used to set up an object. There are two types of constructors in java which are parameterized constructor and default constructors (javaTpoint, n.d.).

Default Constructor

```
Staff staff = new Staff();
```

Figure 28: Register.java

The default constructor of Staff class is called with the new keyword. As shown in Figure 23: Staff.java, there is no constructor in the Staff class, so Java compiler will automatically create a default constructor for Staff class (javaTpoint, n.d.).

Parameterized Constructor

```
public class DateFormatting {

    Date start, end, ytd;
    String sDate, eDate, yDate;
    SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");

    public DateFormatting(Date startDate,Date endDate, Date ytdDate){

        //Date type
        start = startDate;
        end = endDate;
        ytd = ytdDate;

        //String type
        sDate = sdf.format(start);
        eDate = sdf.format(end);
        yDate = sdf.format(ytd);
    }

    //format dates to string to eliminate seconds

    public String StartDateString()
    {...6 lines }

    public String EndDateString()
    {...6 lines }

    public String YtdDateString()
    {...6 lines }

    //format string to date to do comparison

    public Date StartDate() throws ParseException{...6 lines }

    public Date EndDate() throws ParseException{...6 lines }

    public Date YtdDate() throws ParseException{...6 lines }


}
```

Figure 29: DateFormatting.java

As shown in figure above, there is a parameterized constructor for the DateFormatting class constructor. DateFormatting constructor has three date parameters. These parameters is useful to get the formatted date and string.

```
DateFormatting df = new DateFormatting(sd, ed, yes);
//format dates to string to eliminate milliseconds
String startDate = df.StartDateString();
String endDate = df.EndDateString();
String ytdDate = df.YtdDateString();

//format string to date to do comparison
Date start = df.StartDate();
Date end = df.EndDate();
Date ytd = df.YtdDate();
```

Figure 30: BookARoom.java

This DateFormatting class can be used to create object “df” and three dates is passed into the constructor as parameters. With the object created, String value and Date value can be retrieved with the methods in the DateFormatting class.

2.6.6 Get and Set method

```
public class Staff {
    private String fullName;
    private String username;
    private String ic;
    private String email;
    private String contactNumber;
    private String password;
    private String confirmPassword;

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public String getUsername() {...3 lines }

    public void setUsername(String username) {...3 lines }

    public String getIc() {...3 lines }

    public void setIc(String ic) {...3 lines }

    public String getEmail() {...3 lines }

    public void setEmail(String email) {...3 lines }

    public String getContactNumber() {...3 lines }

    public void setContactNumber(String contactNumber) {...3 lines }

    public String getPassword() {...3 lines }

    public void setPassword(String password) {...3 lines }

    public String getConfirmPassword() {...3 lines }

    public void setConfirmPassword(String confirmPassword) {...3 lines }

}
```

Figure 31: Staff.java

As shown in the figure above, Staff class contain getter and setter for all of its field. The value of the variable fullName is returned by the getFullName() method. The setFullName() method accepts fullName parameter and set it to the fullName variable. The current object is referred with “this” keyword (W3Schools, 2022).

Get

```
Staff staff = new Staff();
staff.setFullName(getFullName);
staff.setUsername(getUsername);
staff.setIc(getIc);
staff.setEmail(getEmail);
staff.setContactNumber(getContactNumber);
staff.setPassword(getPassword);
staff.setConfirmPassword(getConfirmPassword);

//check existing
Boolean staffExist = CheckStaff.checkExistingStaff(staff);
if(!staffExist){
    //store info
    DataIntoFiles.register(staff);//storing the staff to file
    clean(); //if anything happen
    setVisible(false);
    new Login().setVisible(true);
}else {
    JOptionPane.showMessageDialog(null, "Username already taken !\nKindly pick up another name.","Error!",JOptionPane.WARNING_MESSAGE);
}
```

Figure 32: Register.java

As shown in figure above, object is created for the Staff class. With the object created, set method is used to update the fields in the Staff class. After validating the staff username, the staff object is passed into the method of register as parameter.

Set

```
public static void register(Staff staff) {
    try {
        FileWriter fw = new FileWriter("staff.txt", true);
        BufferedWriter bw = new BufferedWriter(fw);
        PrintWriter pw = new PrintWriter(bw);
        pw.println(staff.getFullName()+"///"+staff.getUsername()+"///"+staff.getIc()+"///"+staff.getEmail()+"///"+staff.getContactNumber()+"///"+staff.getConfirmPassword());
        pw.close();
        JOptionPane.showMessageDialog(null, "Registered Successfully !","Congratulations!",JOptionPane.INFORMATION_MESSAGE);
    }catch(IOException e){
        JOptionPane.showMessageDialog(null, e);
    }
}
```

Figure 33: DataIntoFiles.java

To retrieve the staff information, get method is used to get the value of staff information with the object given from the method parameter.

2.6.7 Normal method

```
//date validation
public Boolean DateValidation(Date start, Date end, Date ytd){
    Boolean valid = false;

    if (start.before(ytd))
    {
        JOptionPane.showMessageDialog(this, "Your start date should be later than yesterday","Warning!",JOptionPane.WARNING_MESSAGE);
        Refresh();
    }
    else if (start.equals(ytd))
    {
        JOptionPane.showMessageDialog(this, "Your start date should be later than yesterday","Warning!",JOptionPane.WARNING_MESSAGE);
        Refresh();
    }
    else if (start.after(end))
    {
        JOptionPane.showMessageDialog(this, "Your end date should be later than start date","Warning!",JOptionPane.WARNING_MESSAGE);
        Refresh();
        System.out.println("3");
    }
    else if (start.equals(end))
    {
        JOptionPane.showMessageDialog(this, "Your end date should be later than start date","Warning!",JOptionPane.WARNING_MESSAGE);
        Refresh();

    }
    else
    {
        valid = true;
    }

    return valid;
}
```

Figure 34: BookARoom.java

This DateValidation method is created, and data can be passed within method with the use of parameters. This method is also a function for date validation. The start date and end date given by the staff is validated with if-else statement. There are JOptionPane to show warning message if the date given is invalid. Furthermore, Boolean valid is set as false at first, if the dates are valid, Boolean valid will be set to true. This method will return Boolean value to control the flow of Java codes.

```
Boolean valid = DateValidation(start, end, ytd);
```

Figure 35: BookARoom.java

As shown in figure above, the DateValidation method is called and return a true or false value for the Boolean valid.

2.6.8 Exception Handling

When running Java code, a variety of mistakes might occur, including programmer errors, problems caused by incorrect input, and other unforeseen events. When an issue occurs, Java usually comes to a halt and displays an error message which also refers to throw an exception (W3Schools, 2022).

Try-Catch Block

```
try {
    String[] staffDetails = CheckAllData.checkAllData("staff.txt");

    //create arraylist to store all username
    ArrayList<String> existingUsername
            = new ArrayList<String>();

    //loop to get each staff from staffdetails arraylist
    for (String eachStaff : staffDetails) {
        String[] staffValue = eachStaff.split("//"); //split the staff info
        existingUsername.add(staffValue[1]); //add all username to arraylist
    }

    //loop the existingUsername arraylist to check username
    for (int j = 0; j < existingUsername.size() ; j++) {
        if(username.equalsIgnoreCase(existingUsername.get(j))){
            usernameExist = true;
            break;
        }
    }

}catch(IOException e){
    JOptionPane.showMessageDialog(null, e);
}
```

Figure 36: CheckStaff.java

The try block is used to surround code that may cause an exception to be thrown. It has to be utilised as part of the method. The rest of the block code will not run if an exception occurs at a specific statement in the try block. As a result, code that does not throw an exception should not be kept in a try block. A catch block must be included after the try block in Java (javaTpoint, n.d.). In the figure above, the code to check previous username from the staff.txt file is included in the try block. This is because this part of code my throw an error. In the catch block, IOException is included, this type of exception is for input and output exception. The exception will be shown in a pop-up message box.

Multiple Catch Block

```
}catch(IOException e){
    JOptionPane.showMessageDialog(null, e);
} catch (ParseException ex) {
    JOptionPane.showMessageDialog(null, ex);
}
```

Figure 37: CheckBookings.java

Try block can have multiple catch block after it. As shown in both figure above, the catch blocks must have different exception handlers. IOException is used to catch input and output exception. ParseException is used to throw unexpected error while parsing (javaTpoint, n.d.).

Throw Keyword

```
public static String[] checkSearch(String searchValue) throws FileNotFoundException{...52 lines }
public static String[] checkFuture(String[] tableDetails) throws ParseException {...37 lines }
public static String[] checkPast(String[] tableDetails) throws ParseException {...37 lines }
public static String[] checkNow(String[] tableDetails) throws ParseException {...41 lines }
```

Figure 38: CheckBookings.java

Exception Handling in Java can also be handled by using the “throws” keyword to declare exception in the method. As shown in figure above, exception can be place after the throws clause.

2.6.9 File Concept

A File is an abstract data type in Java. A File is a specified location where relevant information is stored. Create a new File, retrieve information about a File, write into a File, read from a File, and delete a File are all examples of File Operations (javaTpoint, n.d.).

Write File

```
public static void updateFile(String[] modifiedBookings) {
    try {
        FileWriter fw = new FileWriter("bookings.txt", false); //write the file
        BufferedWriter bw = new BufferedWriter(fw);
        PrintWriter pw = new PrintWriter(bw);
        for(String eachBooking : modifiedBookings)
        {
            pw.println(eachBooking);
        }
        pw.close();
        JOptionPane.showMessageDialog(null, "Successfully Updated!","Congratulations!",JOptionPane.INFORMATION_MESSAGE);
    }catch(IOException e){
        JOptionPane.showMessageDialog(null, e);
    }
}
```

Figure 39: UpdateBookings.java

File can be written with the use of the FileWriter class. As shown in figure above, there is a Boolean value in the parameter of the FileWriter constructor. The parameters for the constructor are the file name and Boolean to determine to write or append the file. If Boolean is true, the data will be appended. On the other hand, the file will be written when Boolean value is false. The code is places in a try-catch block to catch exception. The exception is the IOException which displays error related to input and output (Oracle, 2020).

Furthermore, to insert lines into the txt file, BufferedWriter class and PrintWriter class are used. BufferedWriter class will get the output of the file and PrintWriter class will have the println() method to print data line by line into the txt file. After that, close() method will close the file (Oracle, 2020).

Read File

```
public static String[] checkAllData(String fileName) throws FileNotFoundException {

    // arraylist to store all data
    ArrayList<String> list
        = new ArrayList<String>();

    //open file to read
    FileInputStream fis = new FileInputStream(fileName);

    //scan file
    Scanner s = new Scanner(fis);

    // get all rows from file
    while (s.hasNextLine()) {
        list.add(s.nextLine());// adding each booking to booking LIst
    }

    // convert booking List to string array
    String[] arrayData
        = list.toArray(new String[0]);

    return arrayData;
}
```

Figure 40: CheckAllData.java

Data in files can be read and store into a string array to do further operations. FileInputStream class is used to read data from the txt file. The name of the file is given with a String variable, the FileInputStream class will establish a connection with it (Oracle, 2020). Furthermore, Scanner class is used to create an object. With the object created, method of hasNextLine() and nextLine() can be use to retrieve data from the file. While the txt file has another line, the next line will be added into the arraylist. After all lines is added into the arraylist, the arraylist will be convert into string array. The method will return this string array so the program can use this string array to do further operations (javaTpoint, n.d.).

Search row in File

```
public static String[] checkBookingDetails(String bookingID) throws FileNotFoundException {

    String[] bookingDetails = CheckAllData.checkAllData("bookings.txt");

    //create arraylist to store all booking ID
    ArrayList<String> existingList
            = new ArrayList<String>();

    //loop to get each booking from booked details arraylist
    for (String eachBooking : bookingDetails) {
        String[] roomBookingData = eachBooking.split("//"); //split the booking info

        //get details of one booking
        if(bookingID.equals(roomBookingData[0]))
        {
            for (String eachDetail : roomBookingData)
            {
                existingList.add(eachDetail); //store each value in array list
            }
        }
    }

    // convert existing list to string array
    String[] existingDetails
            = existingList.toArray(new String[0]);

    return existingDetails;
}
```

Figure 41: CheckBookings.java

To search a row from the file, all the file data is converted into a string array with the help of checkAllData method. Furthermore, a for loop is used to get each booking from the bookingDetails arraylist. The roomBookingData string array is used to store each element in the booking details. The booking details will be split with a separator of "//". If the search booking ID matches the booking ID in the file data string array, the booking details will be store into the existingList arraylist. This arraylist is declared at first. Lastly, this arraylist is converted into a string array which the string array will be used to display booking details on the screen.

Delete row in File

```
public static String[] DeleteBookings(String bookingID) throws FileNotFoundException{

    String ID = bookingID;

    // modifiedlist to store all updated bookings
    ArrayList<String> modifiedList
        = new ArrayList<String>();

    String[] bookingDetails = CheckAllData.checkAllData("bookings.txt");

    //loop to get each booking from booking details arraylist
    for (String eachRoomBooking : bookingDetails) {
        String[] roomBookingData = eachRoomBooking.split("//"); //split the booking info

        //store booking info of that room
        if (ID.equals(roomBookingData[0]))
        {
            continue;
        }
        else
        {
            StringBuilder bookingBd = new StringBuilder();
            for (int i=0; i<roomBookingData.length; i++) {

                bookingBd.append(roomBookingData[i]);

                //last element no need seperator
                if (i == (roomBookingData.length - 1)){
                    break;
                }
                bookingBd.append("//");
            }

            String booking = bookingBd.toString();

            modifiedList.add(booking);
        }
    }

    // convert booking List to string array
    String[] modifiedBookings
            = modifiedList.toArray(new String[0]);

    return modifiedBookings;
}
```

Figure 42: EditBookings.java

To delete a row from the file, all the file data is converted into a string array with the help of checkAllData method. Furthermore, a for loop is used to get each booking from the bookingDetails arraylist. The roomBookingData string array is used to store each element in the booking details. The booking details will be split with a separator of "//". The modifiedList arraylist is declared at first. If the booking ID to delete matches the booking ID in the file data string array, the individual booking will not be added into the arraylist. Bookings other than the booking ID to delete will append all booking details to a string with separator of "//". This string will be added into the modifiedList arraylist. This arraylist will contains the bookings

except the deleted booking. Lastly, the arraylist is converted into string array and updated to the file with the updateFile method as shown in Figure 39: UpdateBookings.java. The file will be written and the new set of data, the modified list will be added into the file.

Update row in File

```
public static String[] updateBookings(String getId, String getContactNumber, String getEmail, String getEmergencyNumber, String getFullName, String getHomeAddress,
        String getIc, String getStartDate, String getNumOfDays, String getEndDate, String getRoomID, String getView, String getDate, String getTime,
        String getTotalAmount, String getServiceCharge, String getTotalRoomPrice) throws FileNotFoundException{
```

Figure 43: EditBookings.java

```
String ID = getId;

// modifiedlist to store all updated bookings
ArrayList<String> modifiedList
    = new ArrayList<String>();

String[] bookingDetails = CheckAllData.checkAllData("bookings.txt");

//loop to get each booking from booking details arraylist
for (String eachRoomBooking : bookingDetails) {
    String[] roomBookingData = eachRoomBooking.split("//"); //split the booking info

    //store booking info of that room
    if (ID.equals(roomBookingData[0]))
    {
        roomBookingData[1] = getFullName;
        roomBookingData[2] = getIc;
        roomBookingData[3] = getContactNumber;
        roomBookingData[4] = getEmergencyNumber;
        roomBookingData[5] = getEmail;
        roomBookingData[6] = getHomeAddress;
        roomBookingData[7] = getStartDate;
        roomBookingData[8] = getEndDate;
        roomBookingData[9] = getView;
        roomBookingData[10] = getRoomID;
        roomBookingData[11] = getNumOfDays;
        roomBookingData[12] = getDate;
        roomBookingData[13] = getTime;
        roomBookingData[14] = getServiceCharge;
        roomBookingData[15] = getTotalRoomPrice;
        roomBookingData[16] = getTotalAmount;
    }

    StringBuilder bookingBd = new StringBuilder();
```

Figure 44: EditBookings.java

```
    for (int i=0; i<roomBookingData.length; i++) {

        bookingBd.append(roomBookingData[i]);

        //last element no need seperator
        if (i == (roomBookingData.length - 1)){
            break;
        }
        bookingBd.append("//");
    }

    String booking = bookingBd.toString();

    modifiedList.add(booking);
}

// convert booking List to string array
String[] modifiedBookings
        = modifiedList.toArray(new String[0]);

return modifiedBookings;
```

Figure 45: EditBookings.java

The method shown in the figure above is to update a single row or booking. To update a row from the file, all the file data is converted into a string array with the help of checkAllData method. Furthermore, a for loop is used to get each booking from the bookingDetails arraylist. The roomBookingData string array is used to store each element in the booking details. The booking details will be split with a separator of "//". The modifiedList arraylist is declared at first. If the booking ID to modify matches the booking ID in the file data string array, each value in the roomBookingData string array will be reset using the parameter passed into the method. With this, each value of the booking details will be updated. Bookings other than the booking ID to modify will remain the same. All booking details will then append to a string with separator of "//". This string will be added into the modifiedList arraylist. This arraylist will contains the updated bookings. Lastly, the arraylist is converted into string array and updated to the file with the updateFile method as shown in Figure 39: UpdateBookings.java. The file will be written and the new set of data, the modified list will be added into the file

3.0 Sample Outputs Screens

3.1 Welcome

Figure 46: Welcome Page

This is the welcome page of the room booking system for Magical Paradise Resort. The purpose of this page is to let staff to login or sign up. “Login Now!” button will let staff to login and “Sign Up!” button allow staff to register an account.

3.2 Sign Up

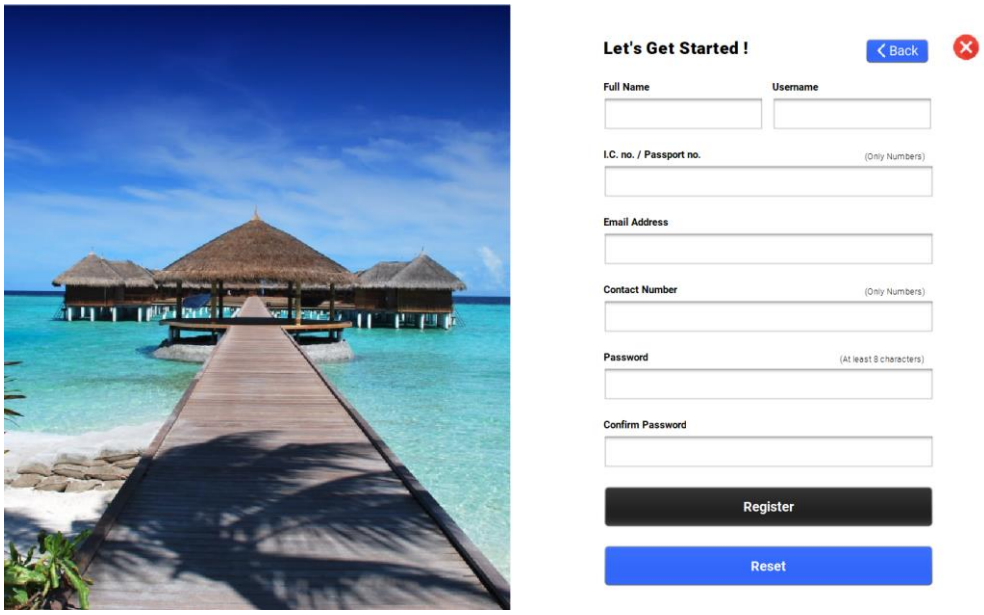


Figure 47: Sign Up Page

This is the sign-up page where staff can register an account with personal details. There will be a "Reset" button for staff to reset everything in the form. "Back" button will bring staff back to the welcome page. After staff filled in all the personal details, staff must click on the "Register" button to validate and store into the txt file.

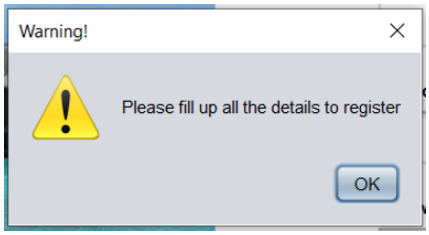


Figure 48: Sign Up Page Validation

If the personal detail in the sign-up form is not completed, this warning message will appear. This message is to ask staff to fill up all the details.

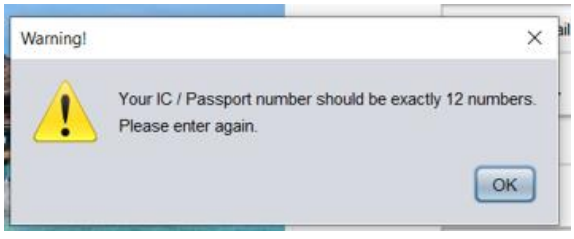


Figure 49: Sign Up Page Validation

If the IC number or passport number in the sign-up form do not have exactly 12 digit, this warning message will appear. This message is to ask staff to give a validate IC number or passport number to the system.

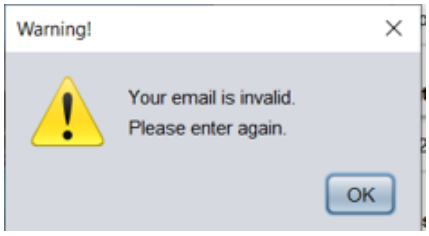


Figure 50: Sign Up Page Validation

If the email in the sign-up form is invalid, this warning message will appear. This message is to ask staff to give a valid email.

Figure 51: Sign Up Page Validation

If the contact number in the sign-up form do not have exactly 10 digit, this warning message will appear. This message is to ask staff to give a validate contact number to the system.

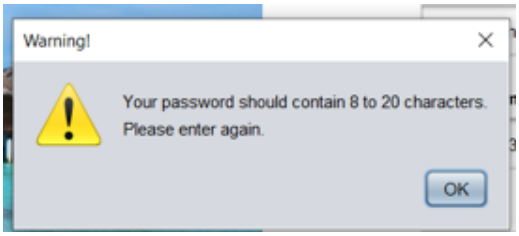


Figure 52: Sign Up Page Validation

If the password in the sign-up form do not contain 8 to 20 characters, this warning message will appear. This message is to ask staff to provide a validate password.

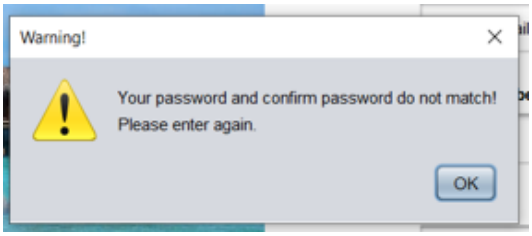


Figure 53: Sign Up Page Validation

If the password and confirm password in the sign-up form do not match, this warning message will appear. This message is to ask staff to provide an accurate password and avoid typing error.

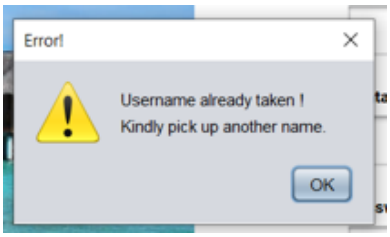


Figure 54: Sign Up Page Validation

If the username in the sign-up form already exists, this warning message will appear. This message is to ask staff to provide another username that are not taken.

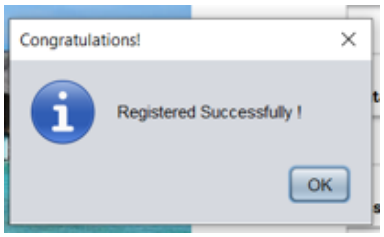


Figure 55: Sign Up Successfully

If all personal details in the sign up form is validated, the register details will be saved to the txt file. This success message will notify staff that their account has been created. The staff will then redirected to the login page after clicking on the “OK” button.

staff.txt ×

Source History

```
1  Sophie Teo//sophietky//020113140500//sophietky@gmail.com//0128003983//sophietky
2  Casty Lau//casty//019283948283//casty@gmail.com//0192839482//castylau
3  Cynyi Yap//cynyi//019203930293//cynyi@gmail.com//0192930293//cynyiyap
4  Hira//hira//918273828283//hira@gmail.com//9182938293//hirahira
5  Ming Roong//Ron//010101019203//ron@gmail.com//0129302930//ronronron
6  Akram//akram//029302930293//akram@gmail.com//9283928302//99999999
7  Jane//jane//019202930293//jane@gmail.com//0192003920//janejane
8  Bala//bala//019283746573//bala@gmail.com//9283748573//balabala
9  Sophie Teo//sophie//020113140500//sophie@gmail.com//0128003983//sophietky
```

Figure 56: staff.txt

The register details will be saved into the txt file.

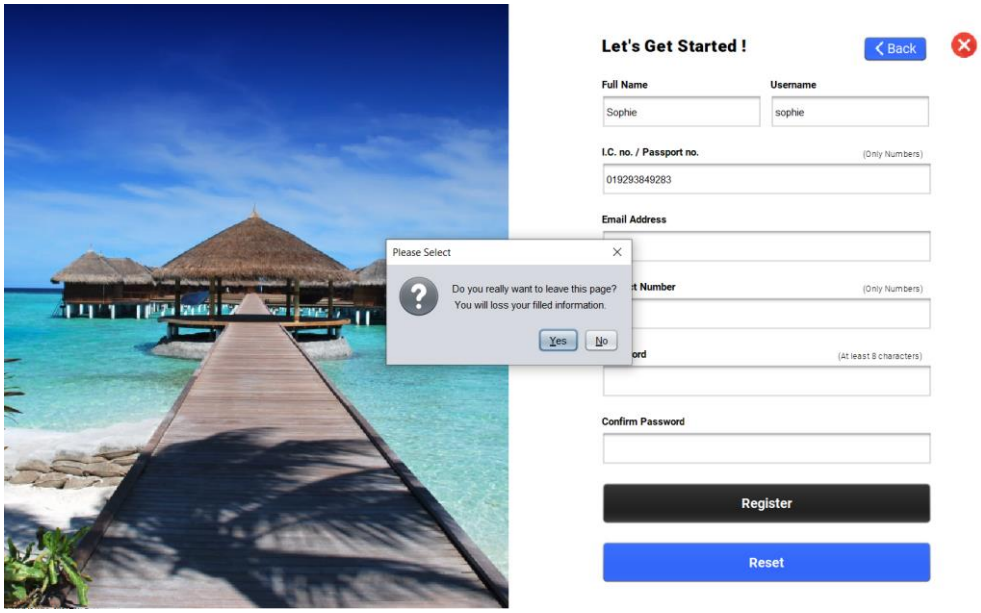


Figure 57: Back Button

This “Back” button will lead staff back to the welcome page. This message is to remind staff that they will lost the filled-up information. If staff insist to go back, staff will be redirected to the welcome page, else staff stay in the same page.

3.3 Login

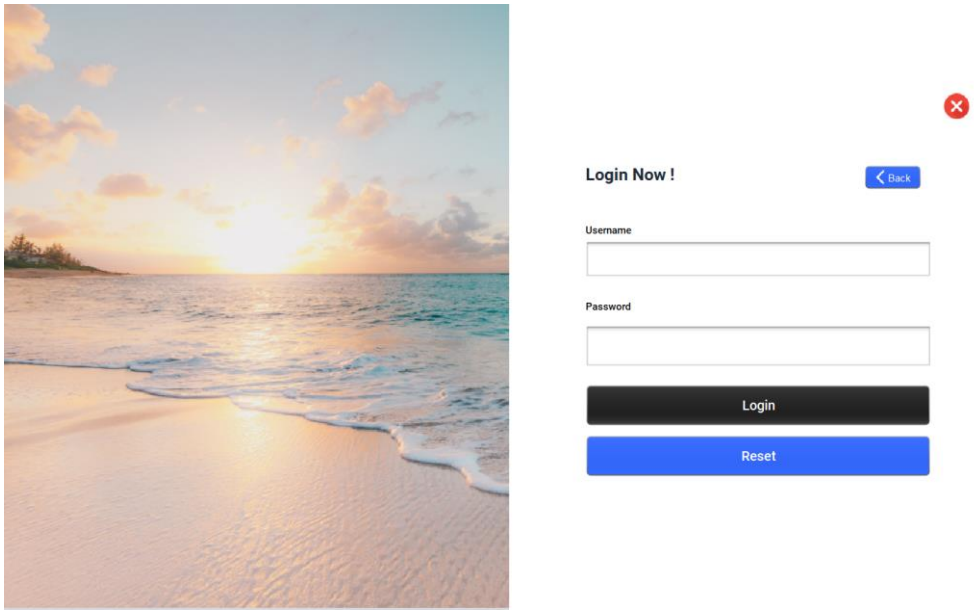


Figure 58: Login Page

This is the login page. Staff must provide a correct username and password to login to the system. To go back to the previous page, click on the “Back” button. The “Reset” button lets staff to clear login details. Staff must click on the “Login” button in order to validate username and password and enter the system. If correct username and password is provided, staff will be redirected to the dashboard.

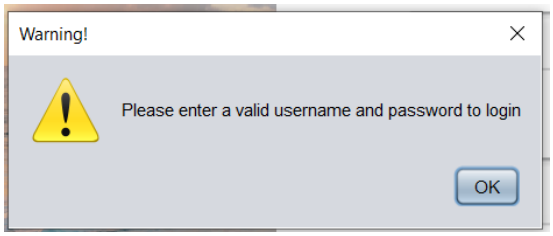


Figure 59: Login Page Validation

If login details are not complete, this message will appear to ask staff to provide all login details.

Figure 60: Login Page Validation

If staff do not provide username, this message will ask the staff to provide a username.

Figure 61: Login Page Validation

If staff do not provide password, this message will ask the staff to provide a password.

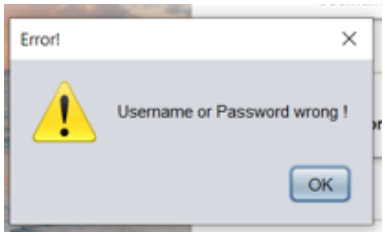


Figure 62: Login Page Validation

If username or password is incorrect or does not exist, this message will notice the staff so staff can re-enter their username and password.

3.4 Dashboard

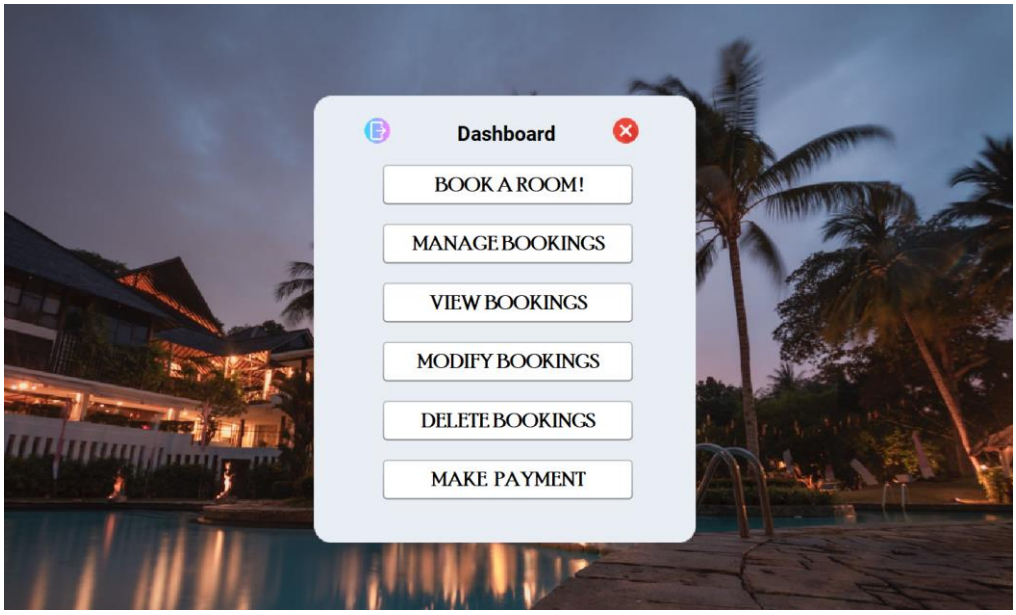


Figure 63: Dashboard Page

This is the dashboard page which is the main menu of the system. There are options add a booking, manage booking, view booking, modify booking, delete booking and make payment. Staff can easily access these functions.

3.5 Book A Room

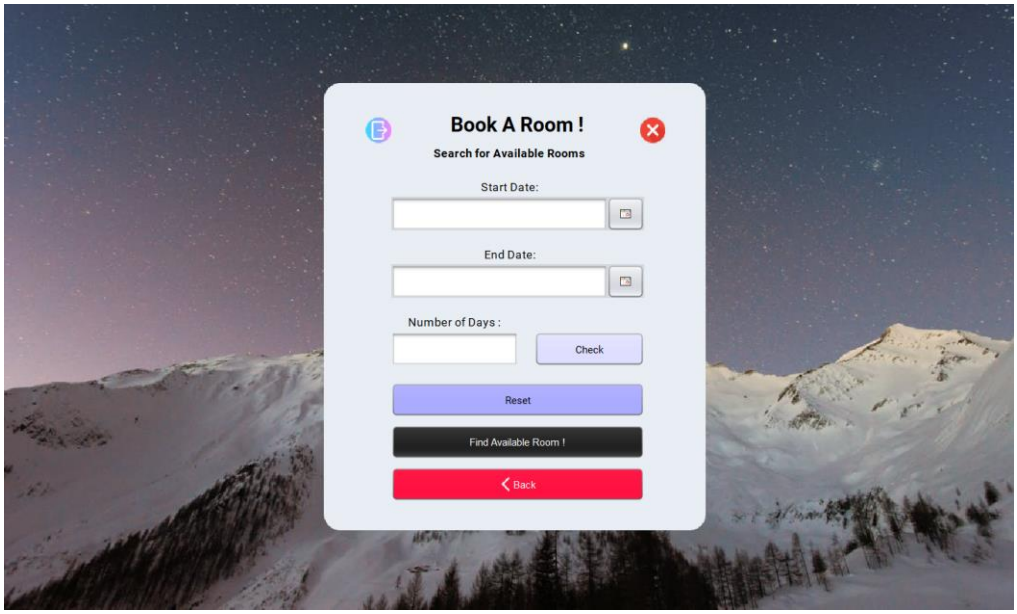


Figure 64: Book A Room Page

This is the booking page where staff can input dates and check room availability of the dates. There will be a “Reset” button to clear all entered details. The “Back” button will lead staff back to the dashboard. The validations applied for “Check” and “Find Available Room!” are the same. After staff clicked on the “Find Available Room!”, staff will be redirected to select an available room.

Figure 65: Book A Room Validation

If staff did not provide start date and end date, this message will ask staff to provide both dates.

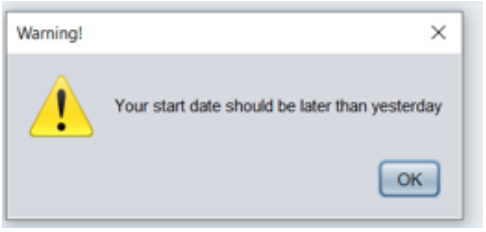


Figure 66: Book A Room Validation

The earliest start date provided should be today's date. If staff provided a start date that is later than yesterday, this message will ask staff to provide another valid start date.

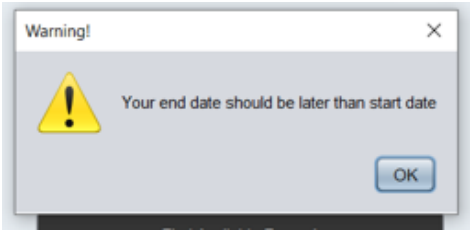


Figure 67: Book A Room Validation

The earliest end date provided should be at least one day later than the start date. If staff provided an end date that is not later than start date, this message will ask staff to provide another valid end date.

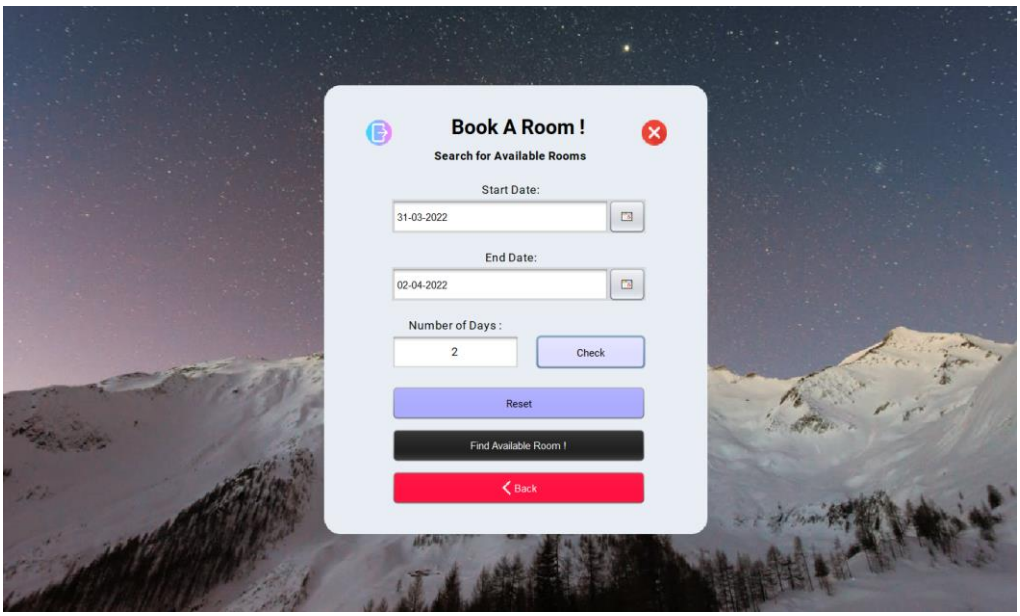


Figure 68: Book A Room Page

After clicking on the "Check" button, staff will be able to check the number of days between start date and end date.

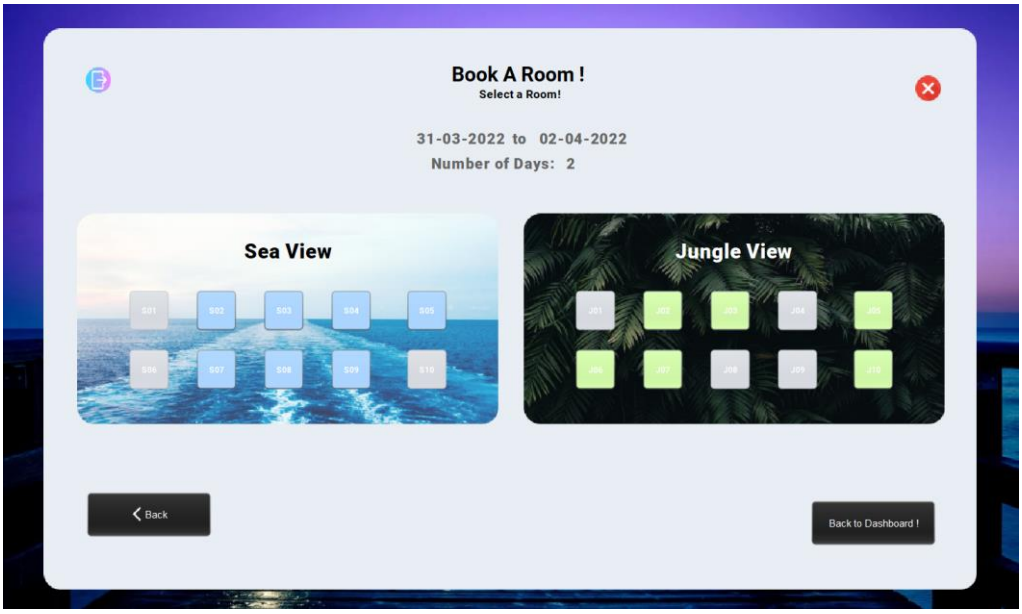


Figure 69: Select Room Page

This is the page where staff can select an available room. The unavailable room will be disabled. The start date, end date and number of days will be shown on the screen. “Back” button will lead staff back to booking page to choose date again. “Back to Dashboard!” button will take staff back to the dashboard.

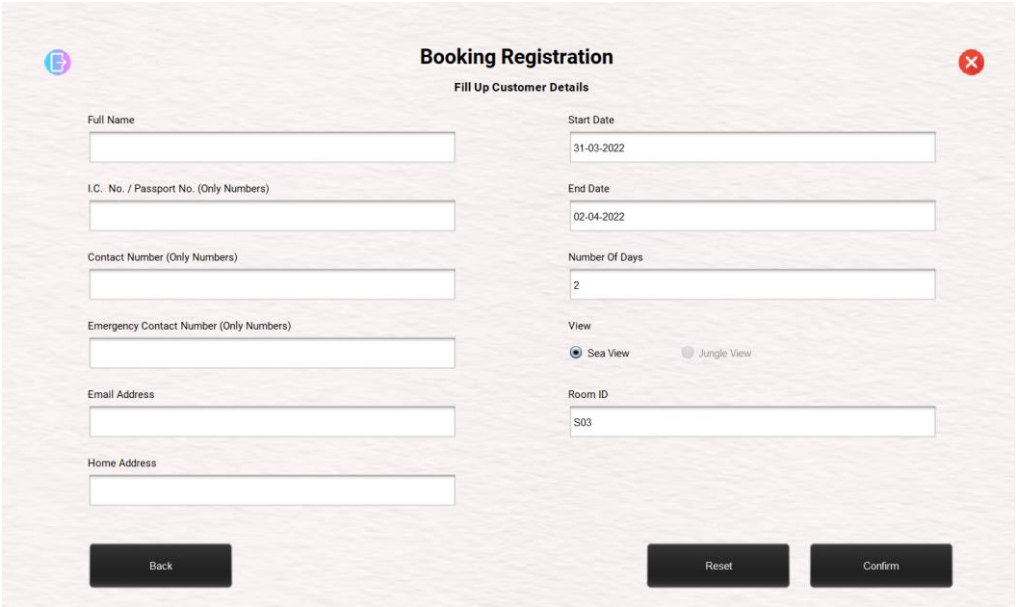


Figure 70: Booking Registration Page

This is the booking registration page where staff needs to fill up the details needed for bookings. “Back” button will take staff back to select another room. “Reset” button will clear customer details in the booking registration page. The room booking details will not be able to edit in the

booking registration page. After filling up customer details, staff must click confirm to check payment details.

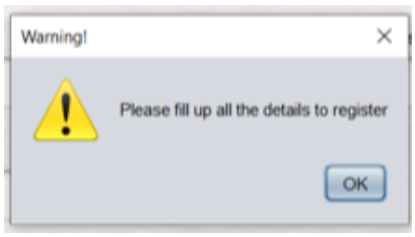


Figure 71: Booking Registration Page Validation

If the booking details provided by the staff is not complete, the message will appear to ask staff to fill up all booking details.

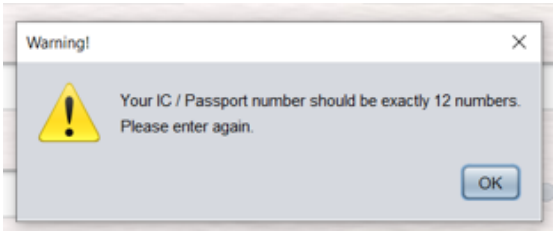


Figure 72: Booking Registration Page Validation

If the IC number or passport number in the booking registration form do not have exactly 12 digit, this warning message will appear. This message is to ask staff to give a validate IC number or passport number to the system.

Figure 73: Booking Registration Page Validation

If the contact number in the booking details form do not have exactly 10 digit, this warning message will appear. This message is to ask staff to give a validate contact number to the system.

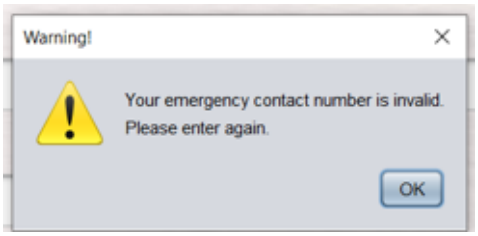


Figure 74: Booking Registration Page Validation

If the emergency contact number in the booking details form do not have exactly 10 digit, this warning message will appear. This message is to ask staff to give a validate contact number to the system.

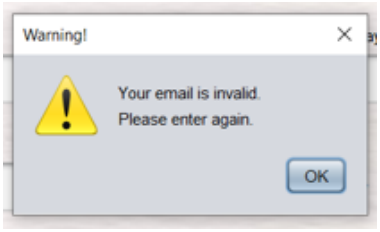


Figure 75: Booking Registration Page Validation

If the email in the booking details form is invalid, this warning message will appear. This message is to ask staff to give a valid email.

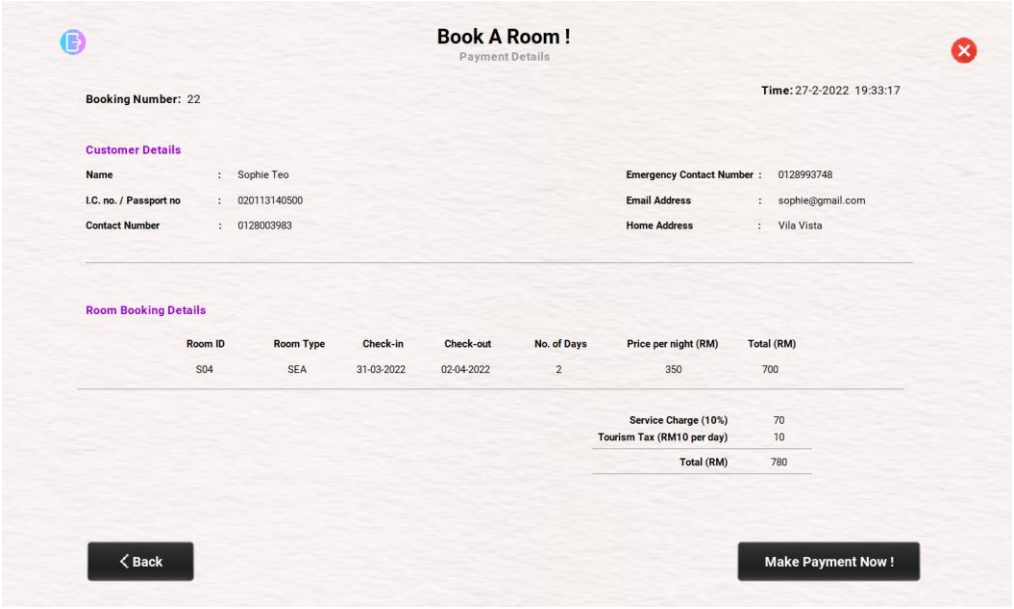


Figure 76: Payment Details Page

This is the payment details page where all booking details are shown and prices are calculated. The “Back” button will lead staff back to the booking registration page. However, as shown in the figure below, the customer information will be lost.

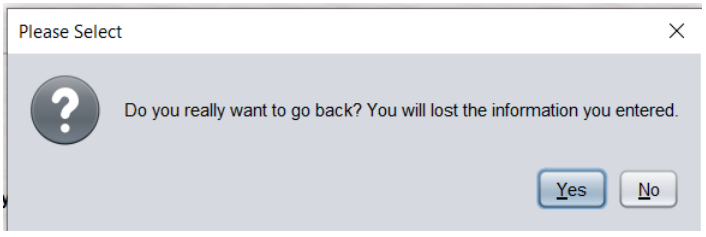


Figure 77: Back button

Figure 78: Booking Registered Successfully

If staff click on the "Make Payment Now!" button, the booking will be registered and store into the txt file. The booking's payment status will be set to "PENDING".

bookings.txt

```
1//Sophie//019283928321//0192039402//0192039203//soph@gmao.com//asdf asdf asdf//20-03-2022//23-03-2022//SEA//S10//3//20-2-2022//1:22:53//35//350//395//PAID
4//Kate//019203920392//0192039203//0192039203//sss@gmail.com//Sunway Velocity//30-03-2022//31-03-2022//JUN//J09//1//21-2-2022//11:49:35//35//350//395//PENDING
6//Lincoln Teo//0192830293002//9102938293//0192039203//sophie@gmail.com//Vila Vista Condo//30-03-2022//31-03-2022//JUN//J08//1//21-2-2022//0:40:3//35//350//395//PAID
7//Cynyi//029382938293//0192839283//0192839283//cyn@gmail.com//sungai buluh//22-03-2022//26-03-2022//JUN//J07//4//20-2-2022//2:15:40//280//2,800//3,090//PENDING
8//Canyon//019203920395//0192039302//0192830293//can@gmail.com//kuala lumpur, malaysia//26-03-2022//27-03-2022//JUN//J10//1//21-2-2022//0:24:35//35//350//395//PAID
9//Katee//019283928392//0192839283//0192839243//ka@gmail.com//KL//27-03-2022//29-03-2022//JUN//J08//2//20-2-2022//2:1:43//35//350//395//PENDING
10//Tony Teo//029301938492//0192839243//tony@gmail.com//Perak Kuala Pilah//24-03-2022//26-03-2022//JUN//J05//2//20-2-2022//2:4:17//70//700//780//PAID
12//Karl1//019203948293//0192938394//0192039348//karl@gmail.com//Ampang Jaya//24-03-2022//31-03-2022//SEA//S10//7//20-2-2022//5:5:18//245//2,450//2,705//PAID
13//Triston//019203920932//0192039203//0192839283//triston@gmail.com//Redang//20-03-2022//21-03-2022//JUN//J08//1//20-2-2022//8:5:25//35//350//395//PAID
14//Sean//019202930192//0192039203//0293453403//sean@gmail.com//Pangkor//25-03-2022//27-03-2022//SEA//S03//2//20-2-2022//8:11:28//70//700//780//PAID
15//Jace//019293939392//0192920293//0192039203//sophie@gmial.com//adsfasdfasdfsdf//25-03-2022//26-03-2022//JUN//J07//1//20-2-2022//9:15:20//35//350//395//PAID
16//Kitty//019203920392//0192039203//0919283923//kitty@gmila.com//Jalan Jalan Jalan//28-03-2022//31-03-2022//SEA//S06//3//21-2-2022//11:52:40//105//1,050//1,165//PAID
17//Jane Jerry//019203920392//0192029302//0192039202//jane@gmail.com//Jalan alor//23-03-2022//26-03-2022//JUN//J08//3//21-2-2022//11:34:32//105//1,050//1,165//PENDING
18//Yao Zi Jing//019203920392//0192039203//0192039203//ziijing@gmail.com//Zi Jin Cheng//29-03-2022//31-03-2022//JUN//J01//2//21-2-2022//11:58:35//70//700//780//PAID
19//Teo Kai Yii//020113140500//0192837482//0192837472//tky@gmail.com//Vila Vista Condo//25-03-2022//29-03-2022//SEA//S08//4//22-2-2022//10:59:16//140//1,400//1,550//PAID
20//Jessi//019203829382//0192938372//0192839283//jessi@gmail.com//Taman Muda//27-03-2022//09-04-2022//SEA//S01//13//24-2-2022//5:34:44//455//4,550//5,015//PAID
21//Katty//019283746573//0192837482//0192837462//katty@gmail.com//Jalan mewah bunga//31-03-2022//09-04-2022//JUN//J04//9//24-2-2022//23:18:37//315//3,150//3,475//PENDING
22//Sophie Teo//020113140500//0128003983//0128993748//sophie@gmail.com//Vila Vista//31-03-2022//02-04-2022//SEA//S04//2//27-2-2022//19:33:17//70//700//780//PENDING
```

Figure 79: booking.txt

This is the booking.txt file where all bookings will be stored here. The last word of each line will be the payment status.

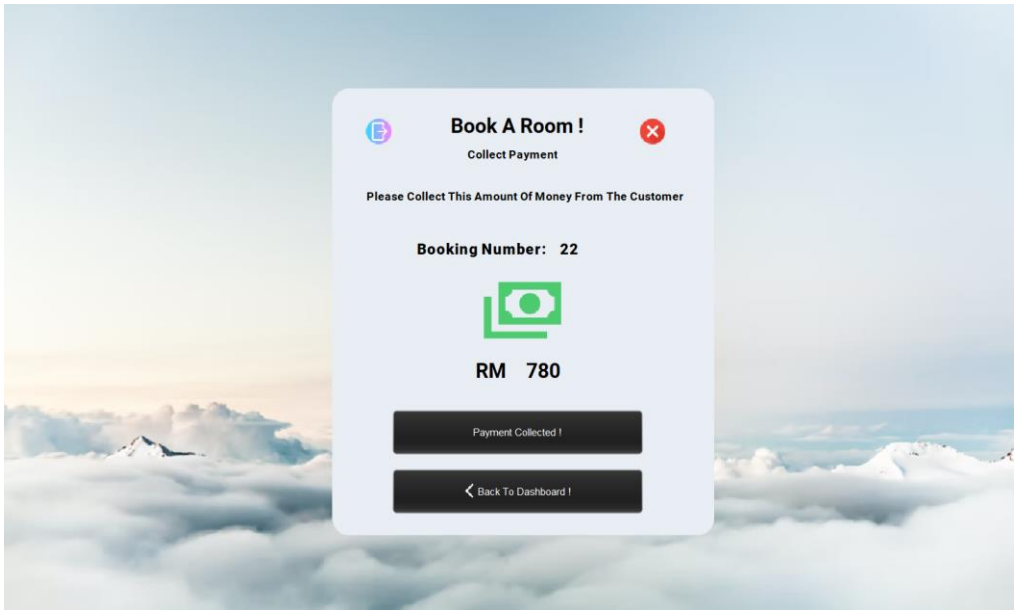


Figure 80: Collect Payment Page

This is the collect payment page where staff can update payment status. Staff can go back to the dashboard with “Back To Dashboard!” button if customer decided not to pay first. The payment status will be updated after staff click on the “Payment Collected!” button.

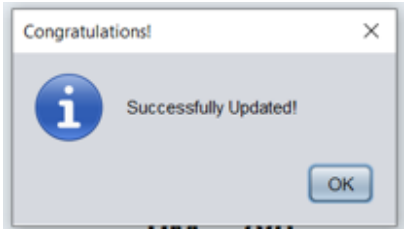


Figure 81: Payment Updated Successfully

This message appear when the booking payment status is successfully updated to “PAID”.

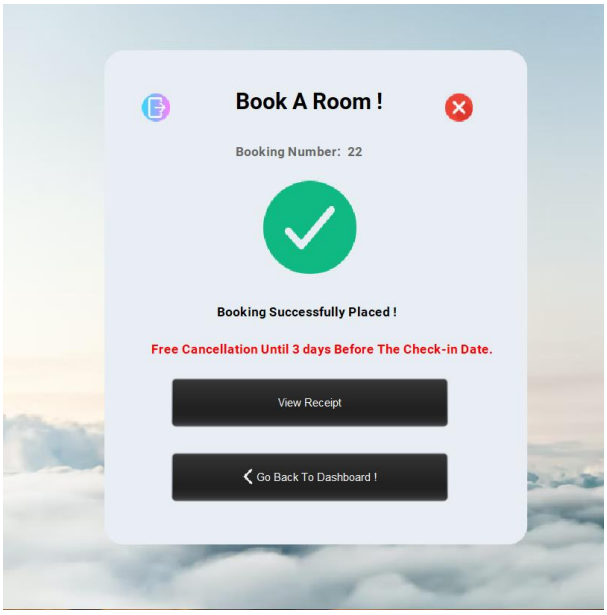


Figure 82: Booked Successfully Page

This is the booked successfully page where staff can view the receipt and go back to the dashboard. “View Receipt” button lets staff to view bookings. “Go Back to Dashboard!” button lead staff to the dashboard.

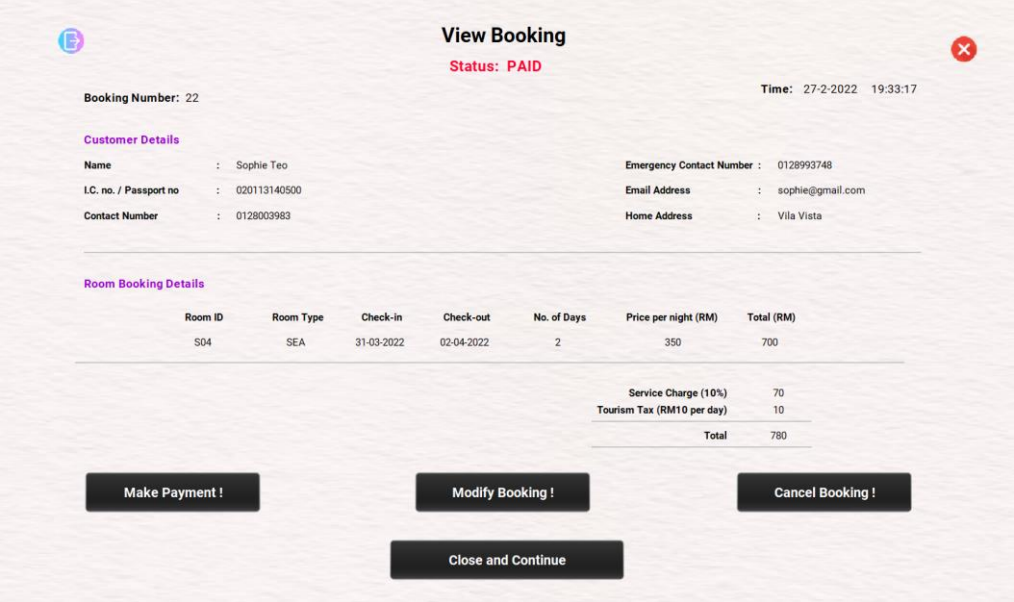


Figure 83: View Booking Page

This is the view booking page where staff can manage booking easily. Staff can make payment, modify booking and cancel booking with “Make Payment!” button, “Modify Booking” button, and “Cancel Booking” button. “Close and continue” button allow staff to close the receipt.

3.6 Manage Bookings

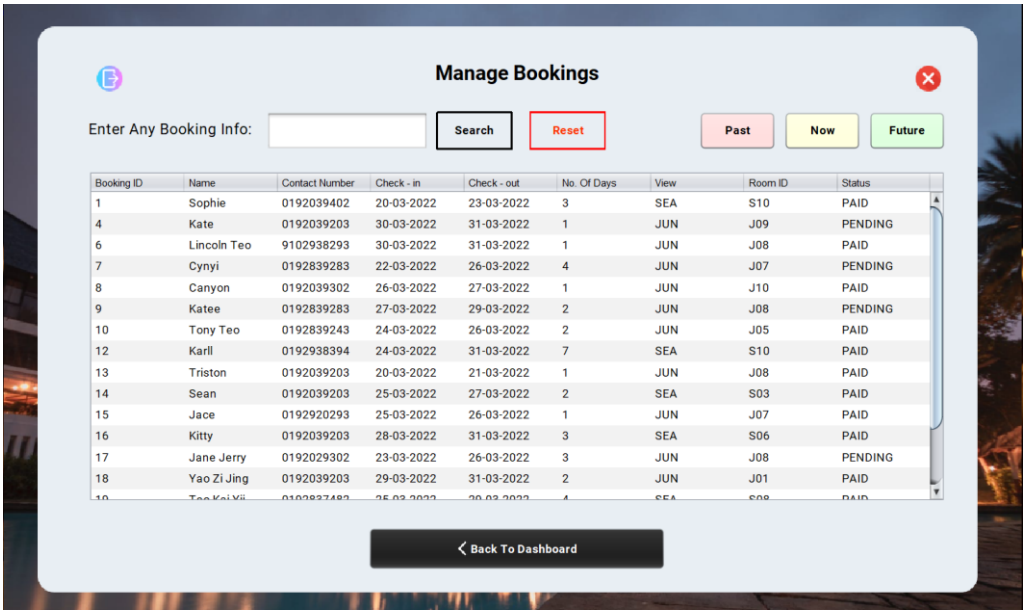


Figure 84: Manage Bookings Page

This is the manage bookings page where all bookings will be included into the table in default. To filter the bookings, staff can search booking with keyword or filter booking according to Past, Now or Future. If staff do not find the desired booking in the filter result, staff can use "Reset" button to reset the table to show all bookings. "Back To Dashboard" button will allow staff to go back the dashboard. To manage a specific booking, staff can just click on the row and the view booking page will pop up.

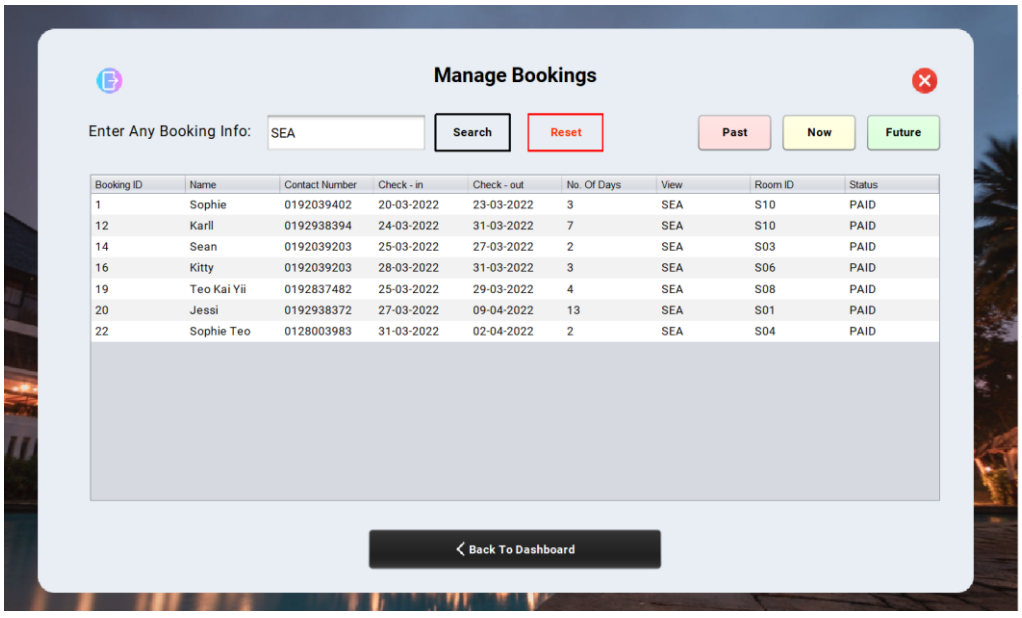


Figure 85: Manage Bookings Page

The figure above shows how to search bookings by keyword. The filter result shows all the Sea View Room.

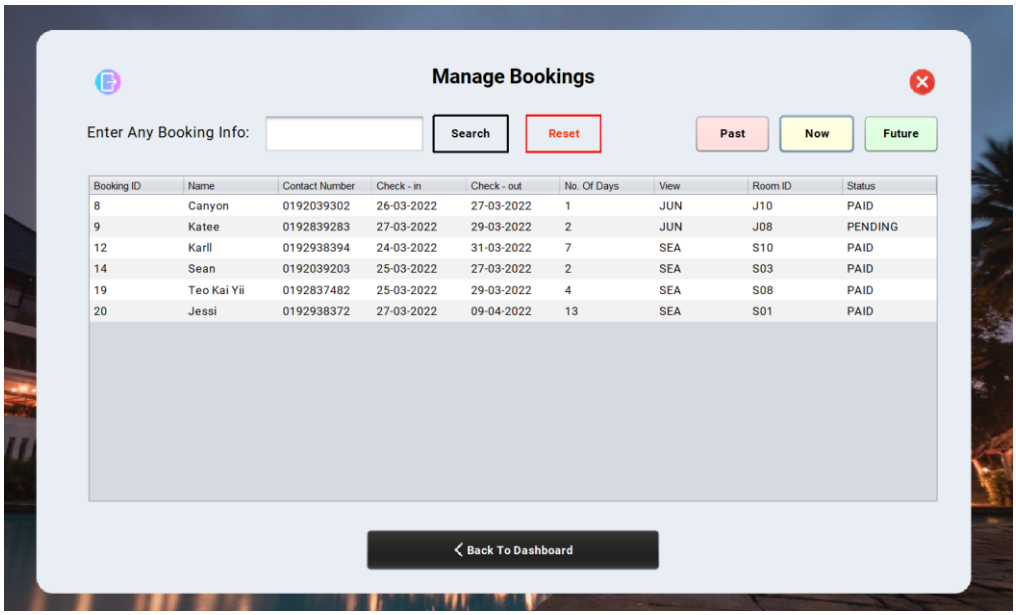


Figure 86: Manage Bookings Page

The result above shows the filter result of bookings now which is filtered by “Now” button on the date of 28 March 2022. Furthermore, staff can also filter past bookings and future bookings by clicking on the “Past” button and “Future” button respectively.

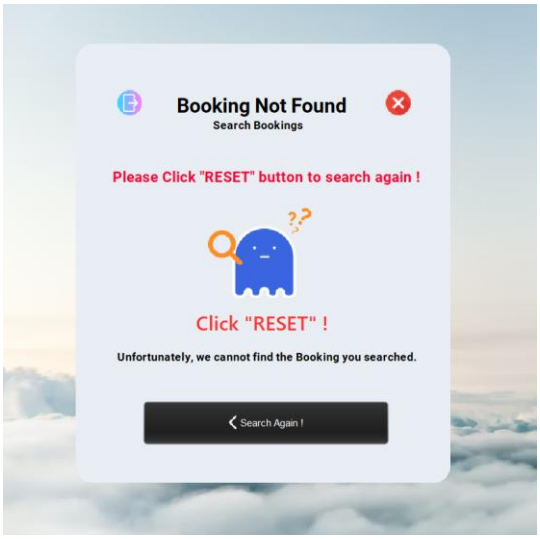


Figure 87: Booking Not Found

If there are no result from the search key, which means no rows appear in the table, this booking not found page will be shown to remind staff to press the “Reset” button before searching again. By clicking on the “Search Again!” button, this booking not found page will be closed.

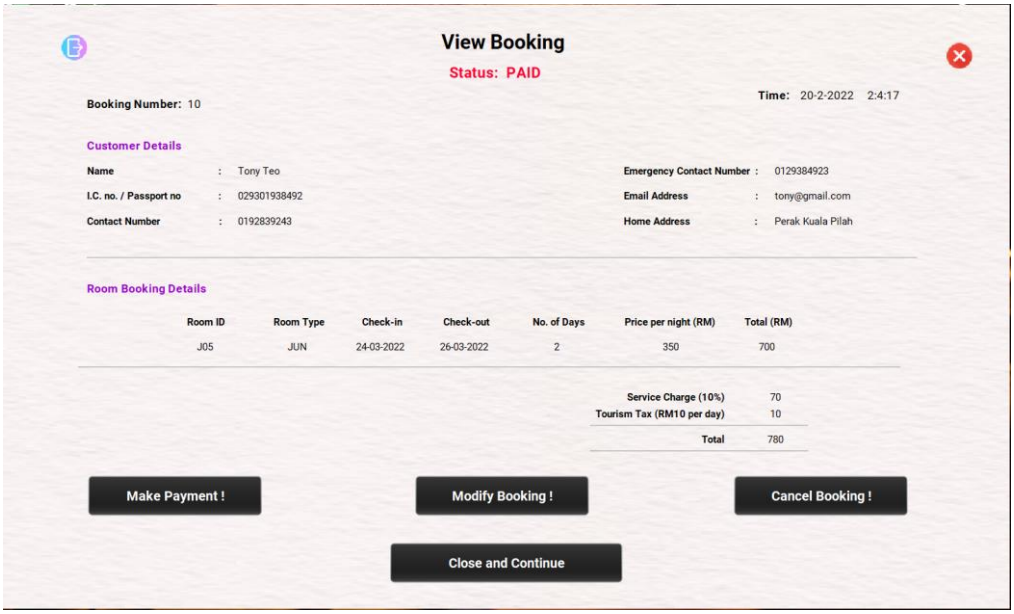


Figure 88: View Bookings Page

This the view bookings page for managing bookings. This page will pop up and show all the booking details. The buttons would allow staff to manage bookings with making payment, modifying booking and cancel booking.

3.7 Make Payment

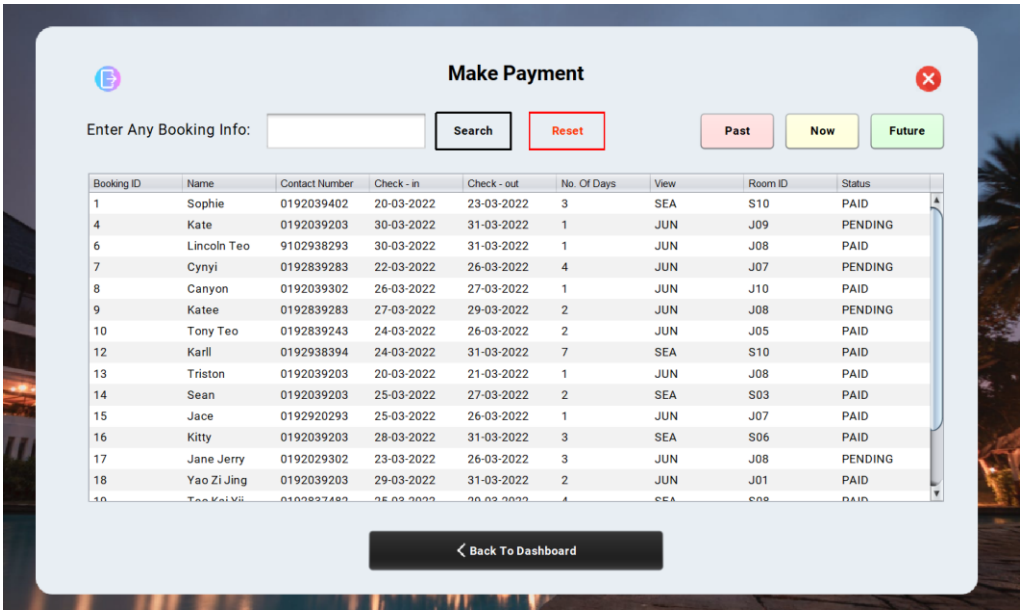


Figure 89: Make Payment Page

This is the make payment page where all bookings will be included into the table in default. This make payment page will allow staff to help customers to make payment. Staff can filter the table to find a specific booking. To make payment for a specific booking, staff can just click on the row and the view booking page will pop up.

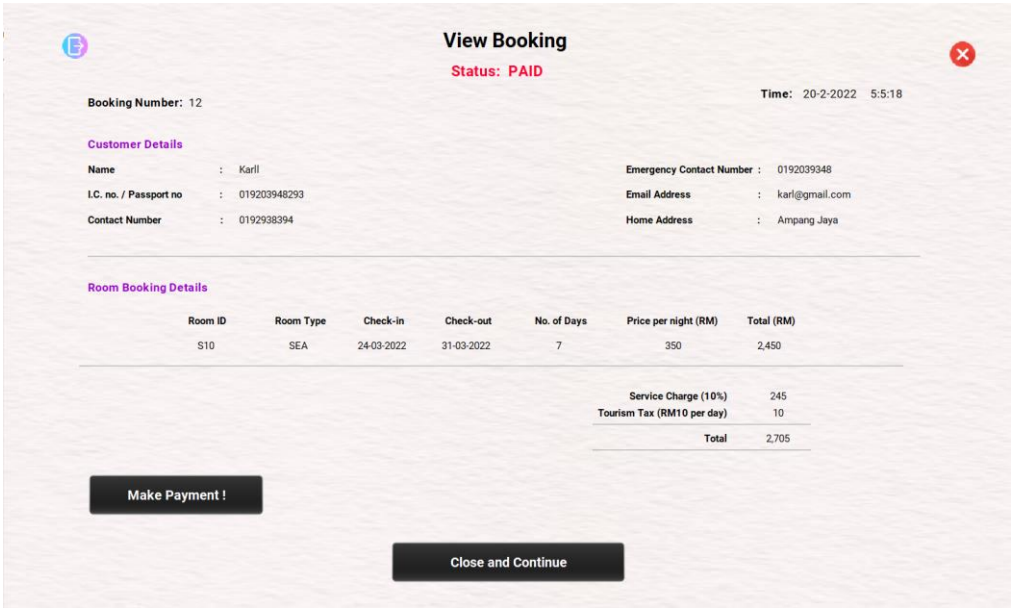


Figure 90: Make Payment Page

This is the view bookings page for making payment This page will pop up and show all the booking details. The “Make Payment” button will allow staff to collect payment for a specific booking and update the payment status.

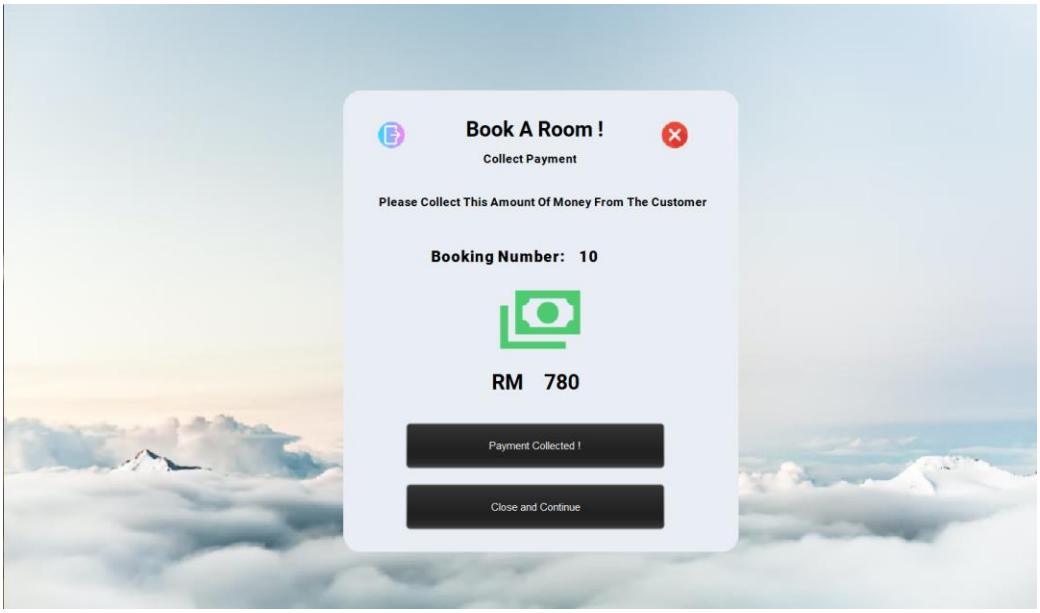


Figure 91: Collect Payment Page

This is the collect payment page which the purpose is to let staff to collect payment from customer and confirm the payment. The payment status will then be updated to “PAID”. After payment is collected, staff can use “Close and Continue” button to close this page.

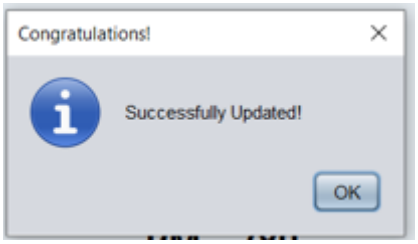


Figure 92: Payment Successfully Updated

This message will appear when staff confirm payment by clicking on the “Payment Collected !” button. The purpose of this message is to notify the staff payment have been updated.

3.8 Modify Booking

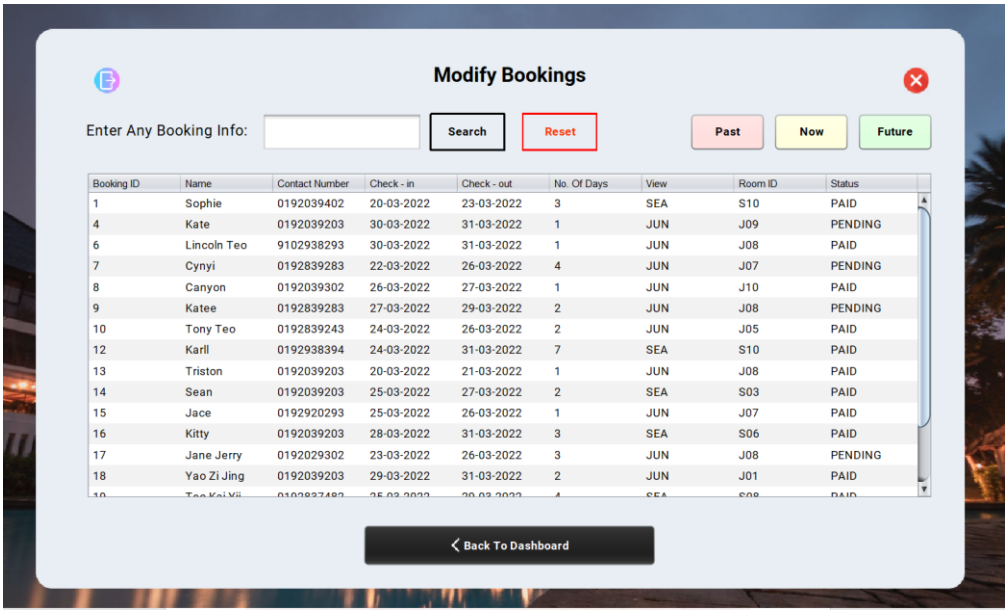


Figure 93: Modify Bookings Page

This is the modify bookings page where all bookings will be included into the table in default. This modify booking page will allow staff to modify customer details and room details. Staff can filter the table to find a specific booking. To modify a specific booking, staff can just click on the row and the view booking page will pop up.

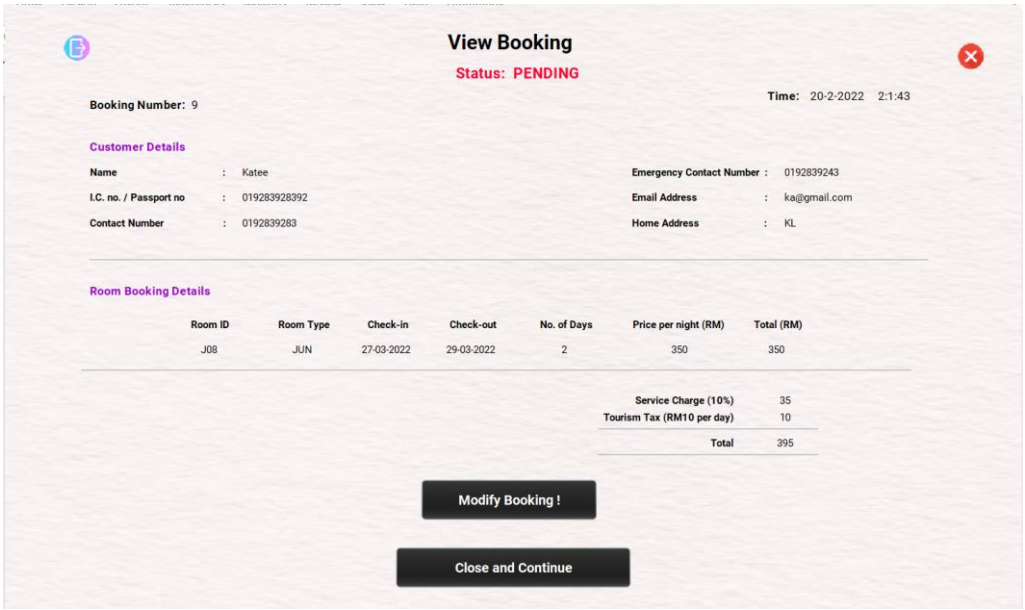


Figure 94: Modify Bookings Page

This view bookings page for modifying bookings. This page will pop up and show all the booking details. The “Modify Booking” button will allow staff to modify the customer details and room details of a specific bookings.

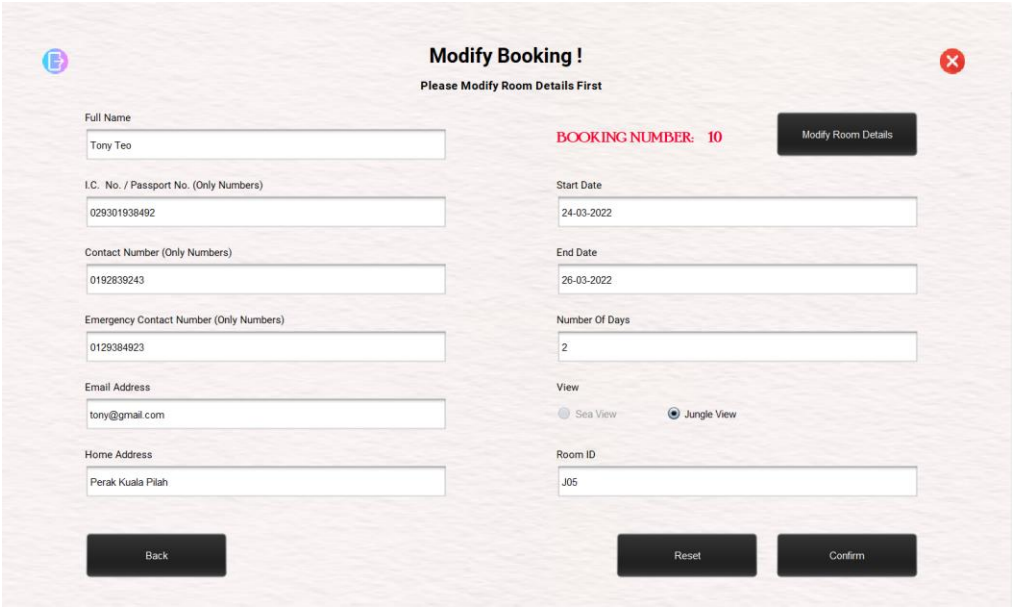


Figure 95: Modify Bookings Page

This is the modify booking page where staff can modify the room details with “Modify Room Details” button and modify the customer details in the text field. “Back” button will bring staff back to the previous page. By clicking on the “Reset” button, all the customer details will be cleared.

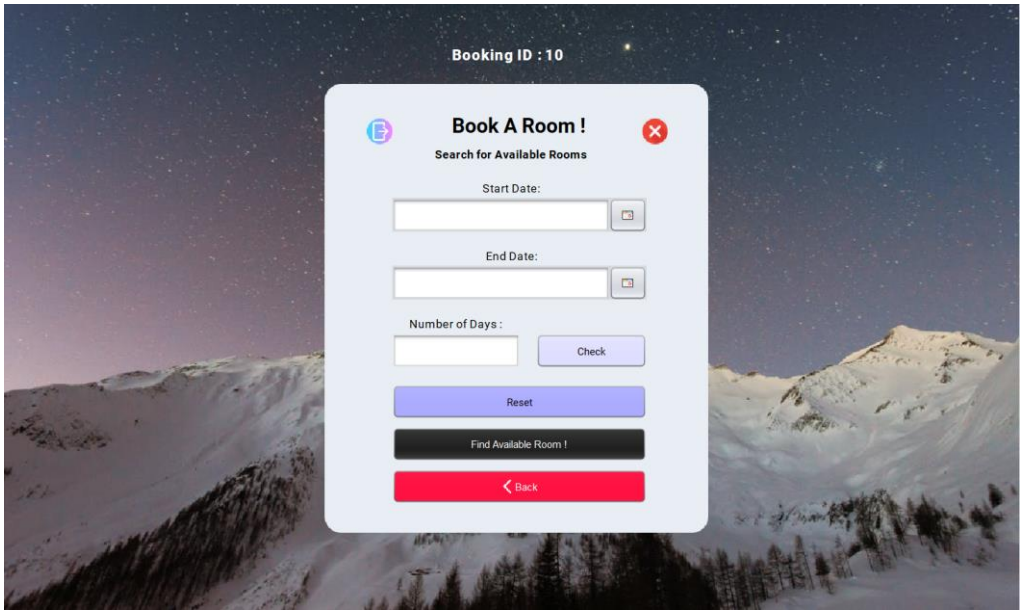


Figure 96: Modify Bookings Page

After clicking on the "Modify Room Details" button, staff will be redirected to the booking page where staff can start to find available rooms. There will also be validations for start date and end date input.

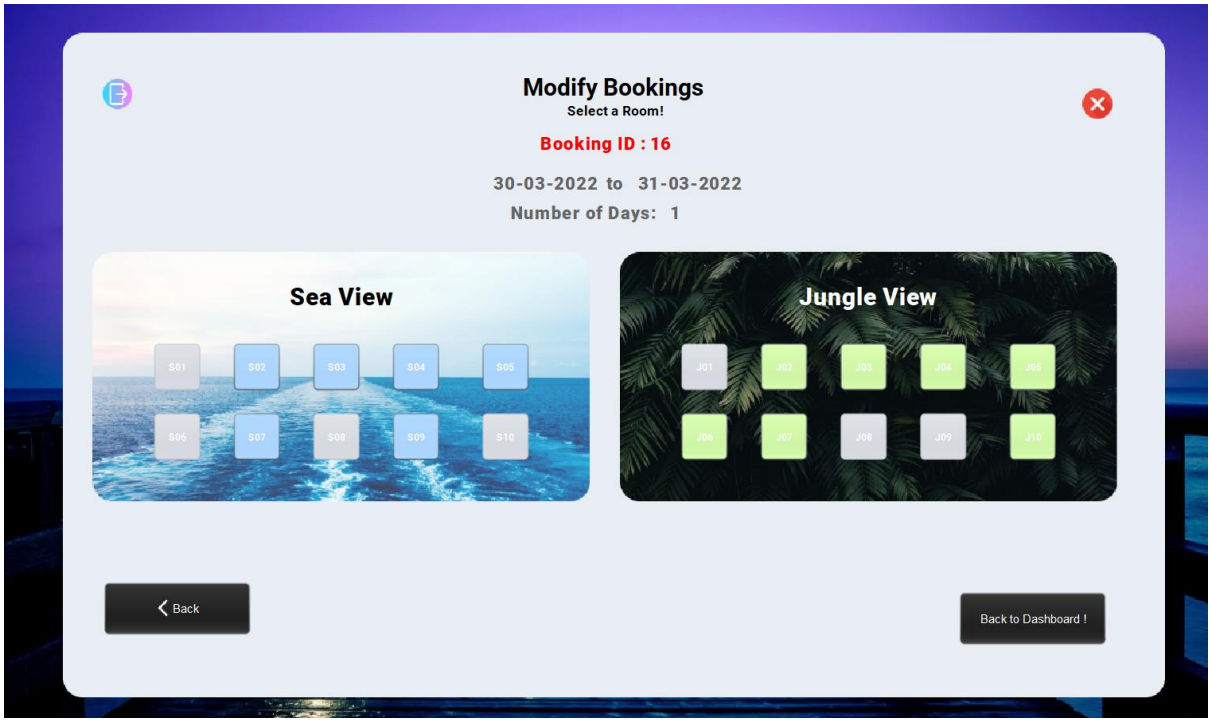


Figure 97: Modify Bookings Page

This is the page where staff can select an available room. The unavailable room will be disabled. The start date, end date and number of days will be shown on the screen. "Back" button will lead staff back to booking page to choose date again. "Back to Dashboard!" button will take staff back to the dashboard.

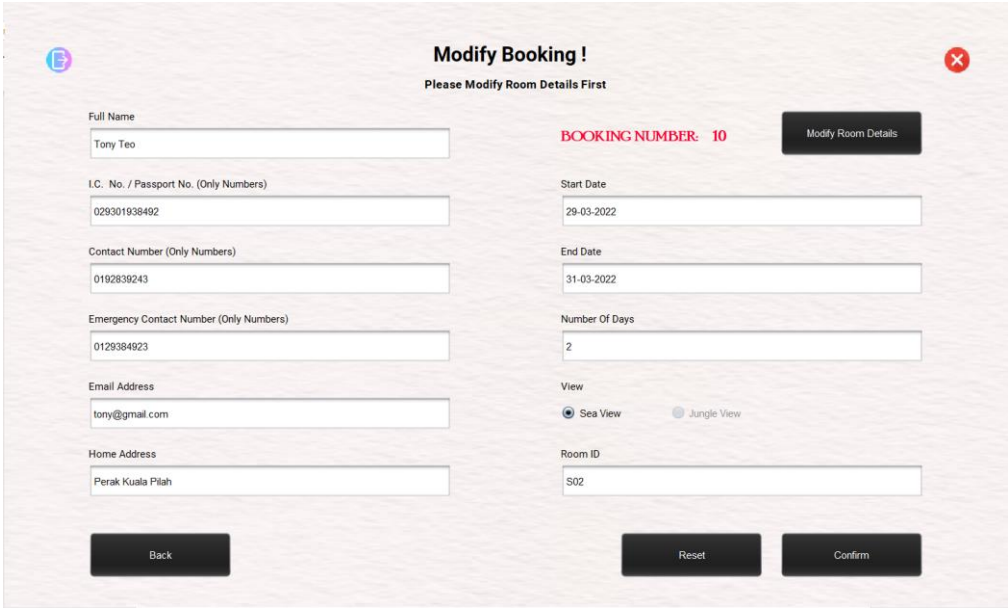


Figure 98: Modify Bookings Page

"Back" button will take staff back to the modify bookings page which shows the bookings table. "Reset" button will clear customer details in the booking registration page. The room booking details will not be able to edit in the booking registration page. After filling up customer details, staff must click confirm to check booking details.

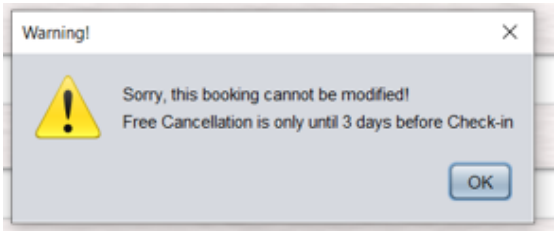


Figure 99: Modify Bookings Validation

This message will appear if the booking staff intended to modify a booking that is not available for modification anymore. This is because staff can only modify booking 3 days before the check in date.

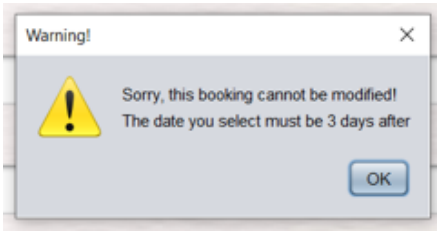


Figure 100: Modify Bookings Validation

This message will appear if the booking staff intended to modify a booking with the unavailable new selected date. This is because staff can only select new start date that are three days after the current date.

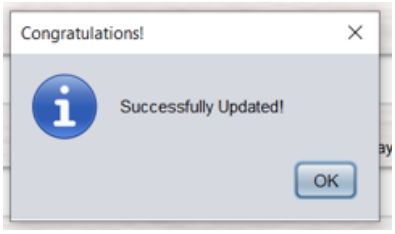


Figure 101: Modify Bookings Successfully

If the modified details is validated and successfully updated, this message will appear to notify staff that the bookings have been successfully updated.

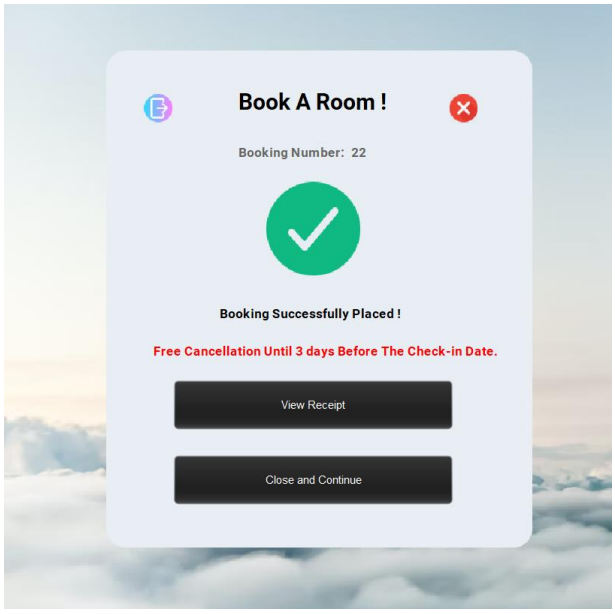


Figure 102: Booked Successfully Page

This is the booked successfully page where staff can view the bookings and manage bookings by clicking on the “View Receipt” button. Staff can close this page by clicking on the “Close and Continue” button.

3.9 Cancel Booking

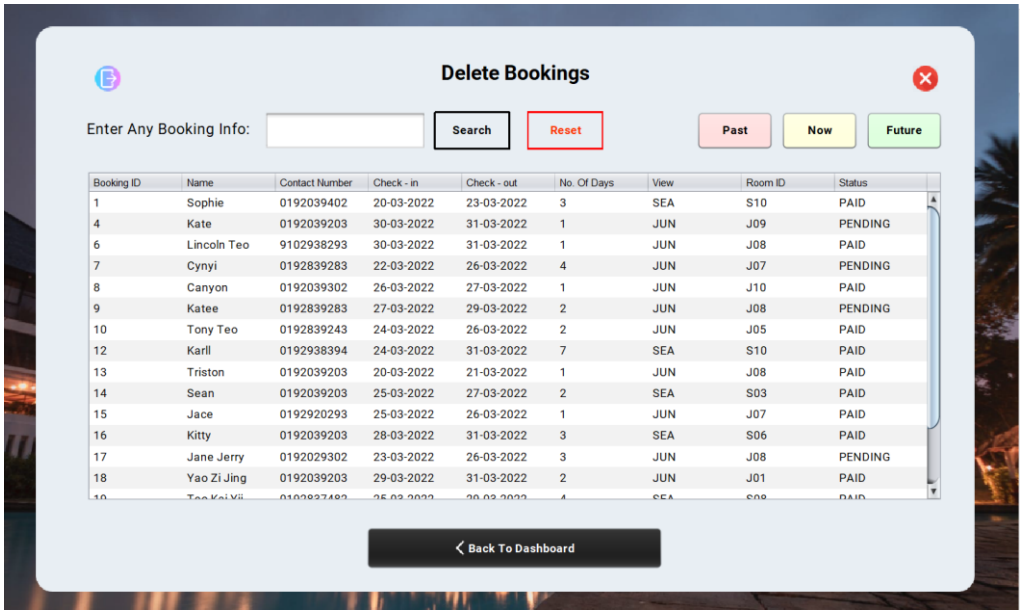


Figure 103: Delete Bookings Page

This is the delete bookings page where all bookings will be included into the table in default. This delete booking page will allow staff to delete customer details and room details. Staff can filter the table to find a specific booking. To delete a specific booking, staff can just click on the row and the view booking page will pop up.

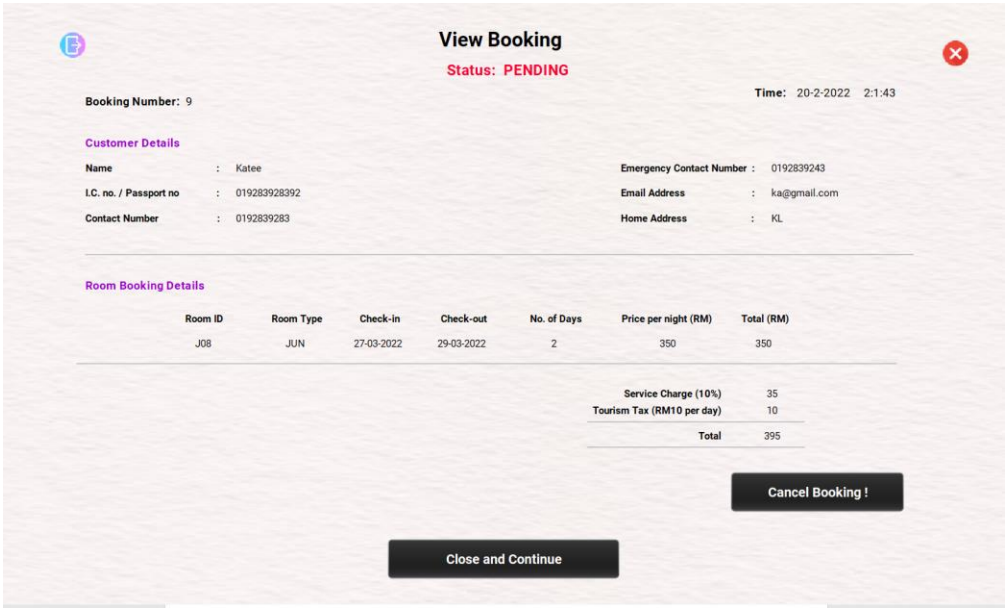


Figure 104: Delete Bookings Page

This is the view bookings page for delete bookings. This page will pop up and show all the booking details. The “Delete Booking” button will allow staff to delete a specific booking.

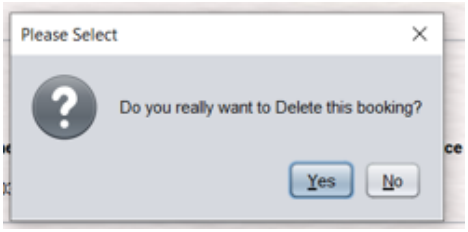


Figure 105: Manage Bookings Page

After staff click on the "Cancel Booking!" button, this message will appear to let staff confirm their choice. If yes, the booking will be deleted, else staff will remain in the same page.

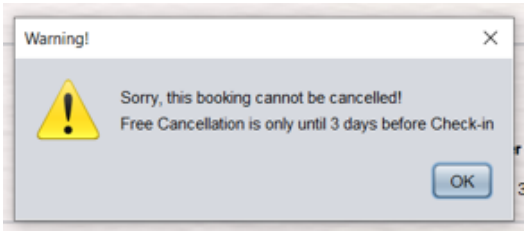


Figure 106: Manage Bookings Validation

After staff confirm decision to delete a booking, this message will appear if the booking is not available for cancellation. This is because bookings can only be cancelled 3 days before the check in date. After staff click "OK" button, the view booking page for delete booking will be closed.

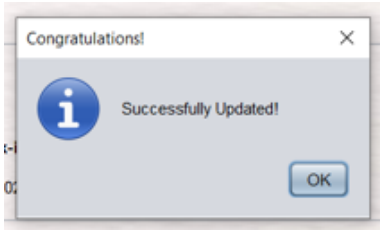


Figure 107: Bookings Cancelled Successfully

After staff confirm decision to delete a booking, this message will appear if the booking is successfully deleted. After staff click "OK" button, the view booking page for delete booking will be closed.

3.10 View All Bookings

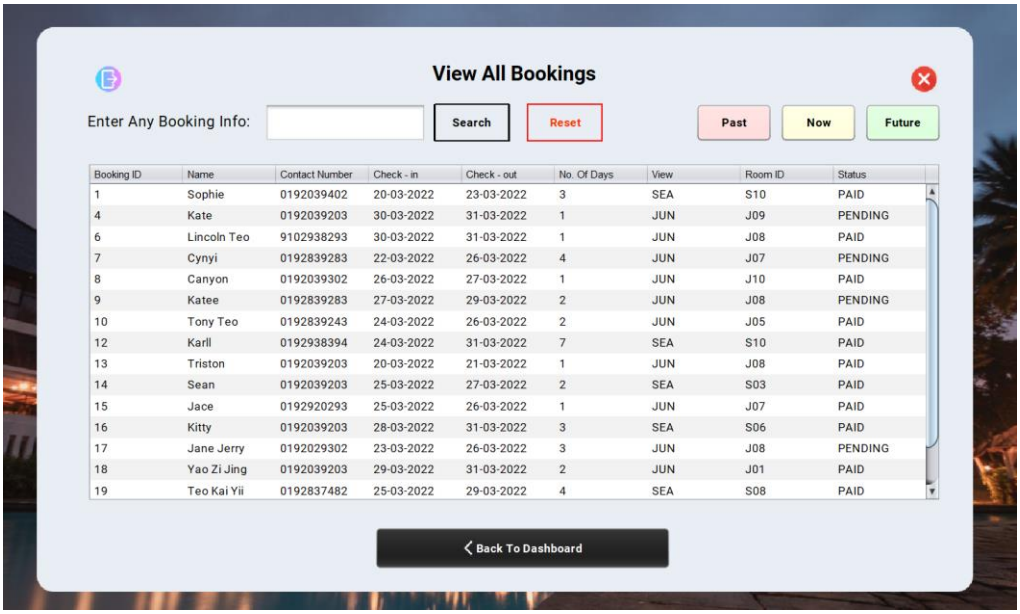


Figure 108: View All Bookings Page

This is the view all bookings page where all bookings will be included into the table in default. This view all booking page will allow staff to view customer details and room details. Staff can filter the table to find a specific booking. To view a specific booking details, staff can just click on the row and the view booking page will pop up.

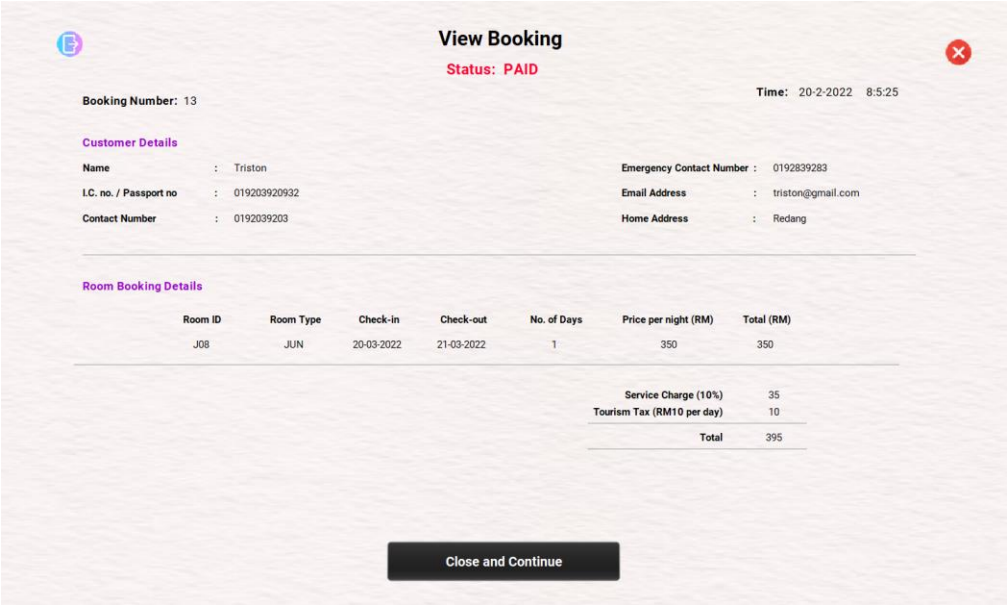


Figure 109: View Bookings Page

This is the view bookings page which will pop up and show all the booking details.

3.11 Logout

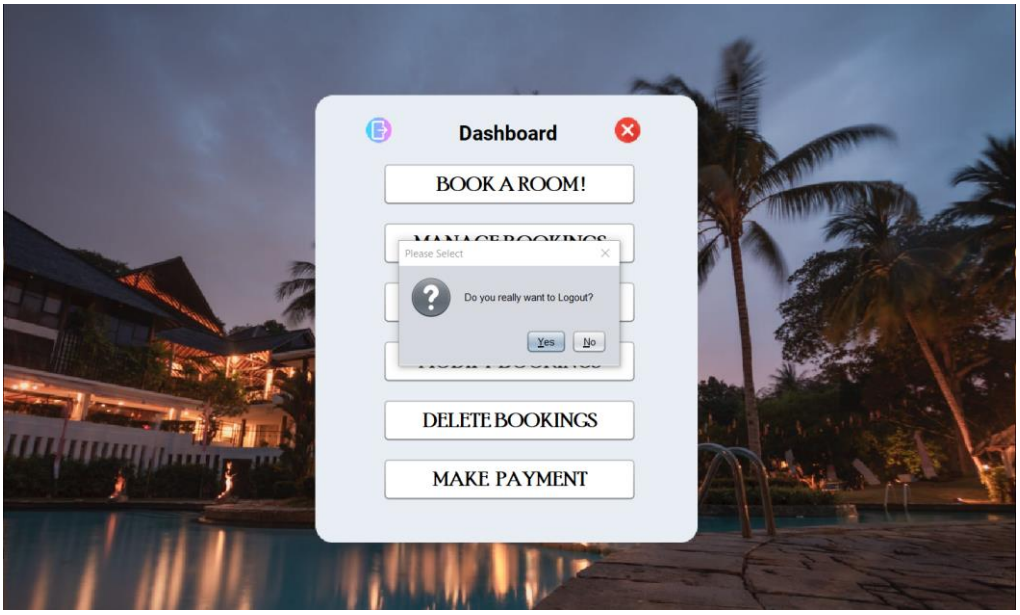


Figure 110: Logout

There is a logout function in this system on the top left corner. After clicking on the logout button, the message will appear to let staff to confirm logout. If yes, staff will be redirected to the welcome page, else staff will remain in current page.

Figure 111: Welcome Page

This is the page where staff will be redirected when staff has successfully logout. Staff can choose to login or sign up.

4.0 Additional Features

4.1 Array

```
//loop to get each booking from booking details arraylist
for (String eachRoomBooking : searchDetails) {
    String[] roomBookingData = eachRoomBooking.split("//"); //split the booking info
    String[] tableRow = {roomBookingData[0],roomBookingData[1],roomBookingData[3],roomBookingData[7],roomBookingData[8],
        roomBookingData[11],roomBookingData[9],roomBookingData[10],roomBookingData[17]};
    dtm.addRow(tableRow); //add to table
}
```

Figure 112: ViewAllBookings.java

As shown in figure above, the array is declared with the data type at first and follow by a square bracket ([]). Furthermore, to declare a string array that consists of booking details for the table row, a curly brace ({}) and with comma to separate each string is used. Other than that, specific element of array can be get with the index number of array. This is shown in the figure above where different elements is retrieved from the string array with different index number. To get elements in a string array, "for-each" loop can be use to get each element in the string array (W3Schools, 2022).

```
for (int i=0; i<roomBookingData.length; i++) {

    bookingBd.append(roomBookingData[i]);

    //last element no need seperator
    if (i == (roomBookingData.length - 1)){
        break;
    }
    bookingBd.append("//");
}
```

Figure 113: CheckBookings.java

There is another way to loop an array. As shown in figure above, length can be used to identify the number of times the loop should repeat. Initial variable is "i=0" and variable i increment by 1 each time. The variable i will repeat as variable i met the condition of "i<roomBookingData.length".

4.2 ArrayList

```
//create arraylist to store all booking ID
ArrayList<String> existingList
        = new ArrayList<String>();

//loop to get each booking from booked details arraylist
for (String eachBooking : bookingDetails) {
    String[] roomBookingData = eachBooking.split("//"); //split the booking info

    //get details of one booking
    if(bookingID.equals(roomBookingData[0]))
    {
        for (String eachDetail : roomBookingData)
        {
            existingList.add(eachDetail); //store each value in array list
        }
    }
}

// convert existing list to string array
String[] existingDetails
        = existingList.toArray(new String[0]);
```

Figure 114: CheckBookings.java

Package java.util has a class call ArrayList which allow resizable array which is different from array. Array size cannot be changed unless a new array is created. ArrayList object is created called existingList. This ArrayList class has add() method. The code above shows each detail of a particular booking is added into the arraylist. Furthermore, the arraylist is covert into string array with the toArray method (W3Schools, 2022).

```
int i = existingID.size() - 1; //get index
String lastID = existingID.get(i); //get the last id
```

Figure 115: CheckBookings.java

In example above, method of size() is use to get the size of the arraylist so index number can be determined. To get the last id in the arraylist, get() method is used to get the element with the index number (W3Schools, 2022).

4.2 JTable

```
DefaultTableModel dtm = (DefaultTableModel)bookingTable.getModel();

for(int i = 0; i < dtm.getRowCount(); i++)
{
    StringBuilder bookingBd = new StringBuilder();
    for(int j = 0; j < dtm.getColumnCount(); j++)
    {
        bookingBd.append(dtm.getValueAt(i, j).toString());
        //last element no need seperator
        if (j == (dtm.getColumnCount() - 1)){
            break;
        }

        bookingBd.append("//");
    }
    String booking = bookingBd.toString();
    tableInfo.add(booking);
}

// convert booked room list to string array
String[] tableDetails
= tableInfo.toArray(new String[0]);

//get string array
String[] pastBookings = CheckBookings.checkPast(tableDetails);

//clear previous rows
dtm.setRowCount(0);

//loop to get each booking from booking details arraylist
for (String eachRoomBooking : pastBookings) {
    String[] roomBookingData = eachRoomBooking.split("//"); //split the booking info
    String[] tableRow = {roomBookingData[0],roomBookingData[1],roomBookingData[2],roomBookingData[3],
        roomBookingData[4],roomBookingData[5],roomBookingData[6],roomBookingData[7],roomBookingData[8]};
    dtm.addRow(tableRow); //add to table
}

if (dtm.getRowCount() == 0 ){
    new BookingNotFound().setVisible(true);
}
```

Figure 116: ViewAllBookings.java

JTable class is used to display rows of bookings in table form in JFrame class. DefaultTableModel is used to copy the data of table, with getModel method, JTable's data can be retrieved. In the nested for loop above, getRowCount() method returns the number of rows in the table and getColumnCount() method returns the number of columns in the table. Each method is used to set the condition of the for loops. With the help of nested for loop, getValueAt() method will get the cell value with variable i and j the initialization of both loop. The toString() method help to change the cell value to String. This will allow each cell value of the table to be append and added into the arraylist. Furthermore, the setRowCount() method

is set to zero so there are no rows in the table. This will clear the previous row in the table. Next, rows of the table is added one by one with the for loop and addRow() method which will add a row at the end of the table (Oracle, 2020).

As shown in figure above, all table data is store into the tableInfo arraylist. Then, this arraylist is converted into string array. The checkPast() method will return the list of past bookings. Additionally, all previous rows in the table will be cleared. Next, the string array of past bookings is looped, and value of the table rows is set. Rows will be added into the table with model. If there are no rows found in the table, the booking not found page will show to notify staff that there are no booking found.

4.3 JCalendar

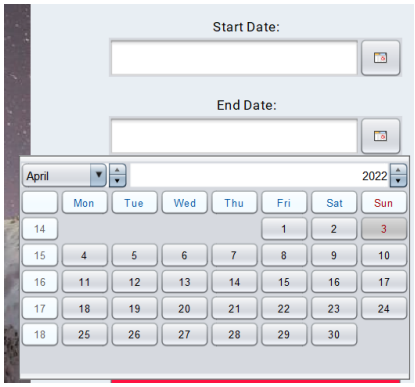


Figure 117: BookARoom.java

This Magical Paradise Resort Booking System uses JCalendar to provide GUI to choose Check-in date and Check-out date. JCalendar is a Java date picker bean that allows staff to choose a date visually as shown in figure above (JCalendar, 2021).

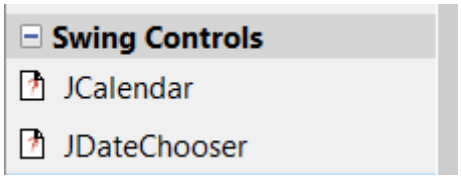


Figure 118: Palette

The JCalendar and JdateChooser is downloaded and install to the NetBeans. Now, both can be find in the Palette under the Swing Controls section.

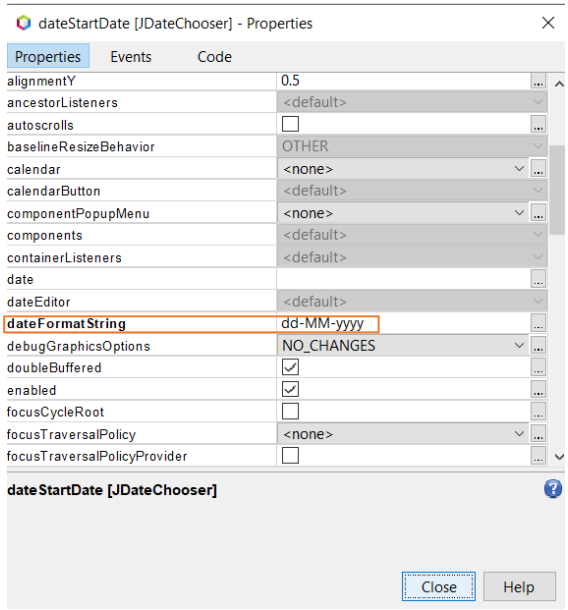


Figure 119: BookARoom.java

The dateFormatString properties of the JDateChooser is set to “dd-MM-yyyy” format. This means that the format of the date chosen will be given in “31-03-2022”.

```
//startdate, enddate
Date sd = dateStartDate.getDate();
Date ed = dateEndDate.getDate();
```

Figure 120: BookARoom.java

Method of getDate() is used to get the chosen date with the variable name of the date chooser.

5.0 Assumptions

1. I assume my room booking system can let staff to manage room bookings of two views with a total of 20 rooms.
2. I assume staff can register account in my room booking system.
3. I assume staff can login to my room booking system with the registered username and password.
4. I assume staff can view available room of a specific date range.
5. I assume staff can select the room to make reservation.
6. I assume staff can use this room reservation system to complete a booking registration.
7. I assume my room booking system will store necessary customer's information.
8. I assume the registered information of booking and staff will be stored into the specific txt file.
9. I assume the room will be restricted for the date range that are reserved by customers.
10. I assume that my system can calculate and generate receipt which include customer's information, room details, total amount, and taxes for customers.
11. I assume this room booking system allow staff to modify, delete, search and view all bookings.
12. I assume there are policy for staff to delete or modify bookings which the policy are "Bookings can only be delete or modify 3 days before Check-in".
13. I assume that staff can sort bookings by past, now or future bookings.
14. I assume that my room booking system allow staff to make payment and update payment status.
15. I assume that staff can reset to view all bookings if the searched booking is not found.

6.0 References

- GeeksforGeeks. (24 March, 2022). *Variables in Java*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/variables-in-java/
- javaTpoint. (n.d.). *Constructors in Java*. Retrieved from javaTpoint: https://www.javatpoint.com/java-constructor
- javaTpoint. (n.d.). *File Operations in Java*. Retrieved from javaTpoint: https://www.javatpoint.com/file-operations-in-java
- javaTpoint. (n.d.). *Java Control Statements | Control Flow in Java*. Retrieved from javaTpoint: https://www.javatpoint.com/control-flow-in-java
- javaTpoint. (n.d.). *Java JFrame*. Retrieved from javaTpoint: https://www.javatpoint.com/java-jframe
- javaTpoint. (n.d.). *Java Scanner*. Retrieved from javaTpoint: https://www.javatpoint.com/Scanner-class
- javaTpoint. (n.d.). *Java try-catch block*. Retrieved from javaTpoint: https://www.javatpoint.com/try-catch-block
- *JCalendar*. (2021). Retrieved from TOEDTER.COM: https://toedter.com/jcalendar/
- Oracle. (2020). *Class DefaultTableModel*. Retrieved from Oracle: https://docs.oracle.com/javase/7/docs/api/javax/swing/table/DefaultTableModel.html
- Oracle. (2020). *Class FileInputStream*. Retrieved from Oracle: https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html
- Oracle. (2020). *Class StringBuilder*. Retrieved from Oracle: https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html
- Oracle. (2020). *Java™ Platform, Standard Edition 7*. Retrieved from Oracle: https://docs.oracle.com/javase/7/docs/api/overview-summary.html
- Oracle. (2020). *Package java.io*. Retrieved from Oracle: https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html

- W3Schools. (2022). *Java ArrayList*. Retrieved from W3Schools: https://www.w3schools.com/java/java_arraylist.asp
- W3Schools. (2022). *Java Arrays*. Retrieved from W3Schools: https://www.w3schools.com/java/java_arrays.asp
- W3Schools. (2022). *Java Encapsulation*. Retrieved from W3Schools: https://www.w3schools.com/java/java_encapsulation.asp
- W3Schools. (2022). *Java Exceptions - Try...Catch*. Retrieved from W3Schools: https://www.w3schools.com/java/java_try_catch.asp
- W3Schools. (2022). *Java extends Keyword*. Retrieved from W3Schools: https://www.w3schools.com/java/ref_keyword_extends.asp
- W3Schools. (2022). *Java OOP*. Retrieved from W3Schools: https://www.w3schools.com/java/java_oop.asp
- W3Schools. (2022). *Java Packages*. Retrieved from W3Schools: https://www.w3schools.com/java/java_packages.asp