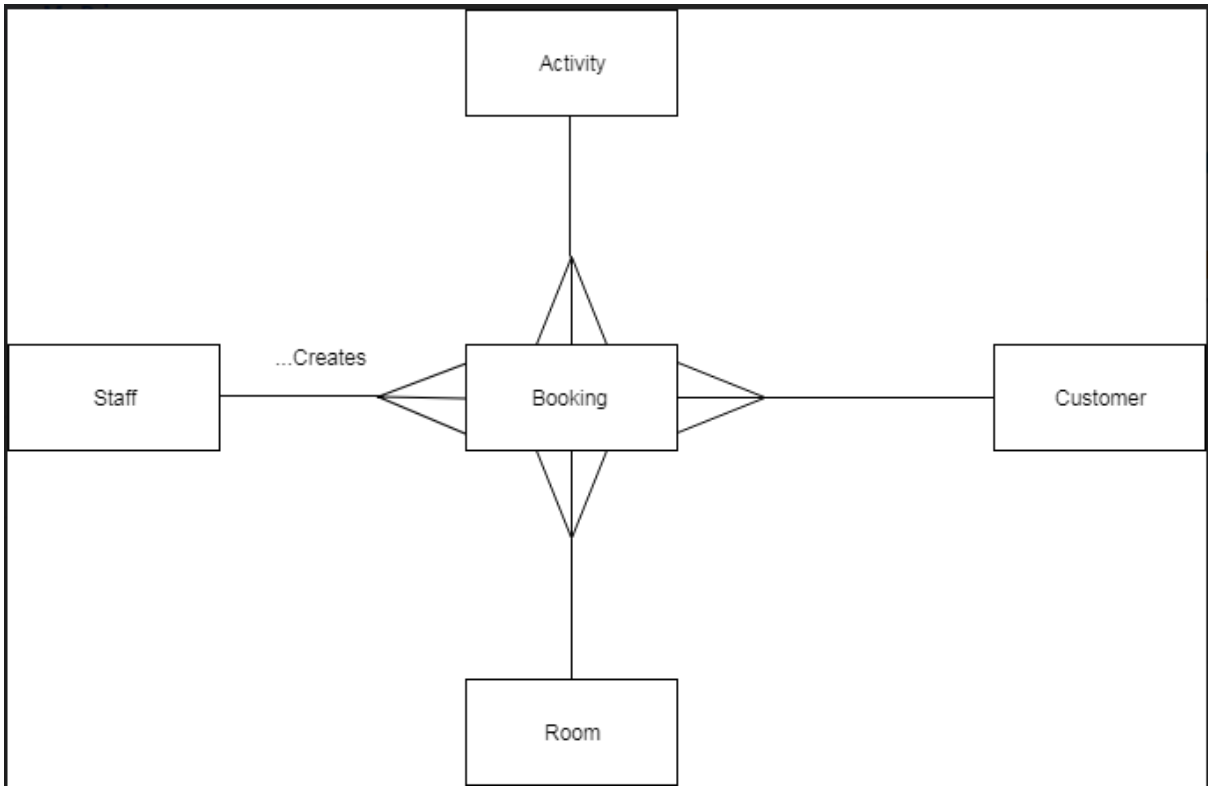
3 – Design

Contents

3.1 Breakdown of Problem

3.11 ER Diagram for Proposed System



The ER diagram shows the entities involved in my system and how they interact with each other. The system revolves around the bookings and staff or customers can create many bookings with each booking being created by one member of staff or one customer. Once the booking is created the booking is assigned to a room, with each room having many other bookings. The booking is also assigned an activity and similarly each activity can have many bookings.

Staff and customers can add bookings, view bookings, edit/delete bookings and search bookings. Along with this once a booking is made a calculation will be made to determine price of the booking. The data for both, bookings and payments are written and stored to files.

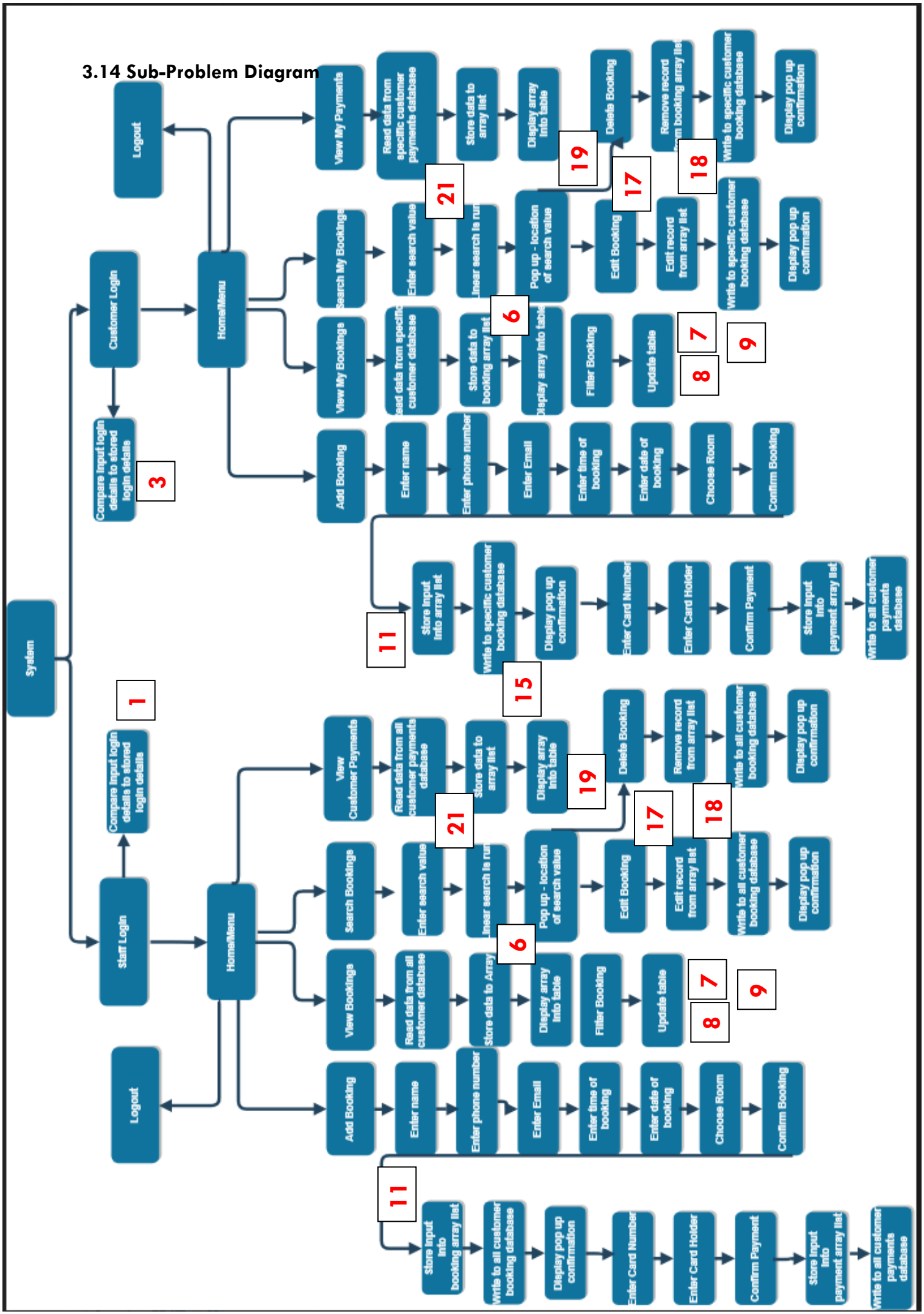
The booking will then be assigned to a room which is chosen when the booking is being made, the activity will also be assigned, both of which, will be able to be edited.

3.12 Justification of Breakdown

3.13 Links between Objectives and Sub Problems

Objective # (Investigation)	Objective Description (Investigation)	Sub-Problem # (Diagram)
1	Security – verify staff logins	1
2	Read from a text file – customer logins	2
3	Security - Verify customer logins	3
4	Read from a text file – staff logins	4
5	Read from a text file – booking file	5
6	View bookings – display array data into a table	6
7	Sort/Filter bookings by date	7
8	Sort/Filter bookings by name	8
9	Sort/Filter customers/members lists by name	9
10	Add bookings to the database	10
11	Write to a file – booking list	11
12	Write to a file – customer info	12
13	Write to file –customer history	13
14	Write to a file – list of customers	14
15	Calculations to decide price	15
16	Calculations to apply discount on customers fee if required	16
17	Amend booking list	17
18	Update/ edit booking database	18
19	Delete bookings from the database	19
20	Searching for specific customer details	20
21	Search by dates of booking	21

3.14 Sub-Problem Diagram



3.2 Input and Output Designs

3.21 Identification of Data

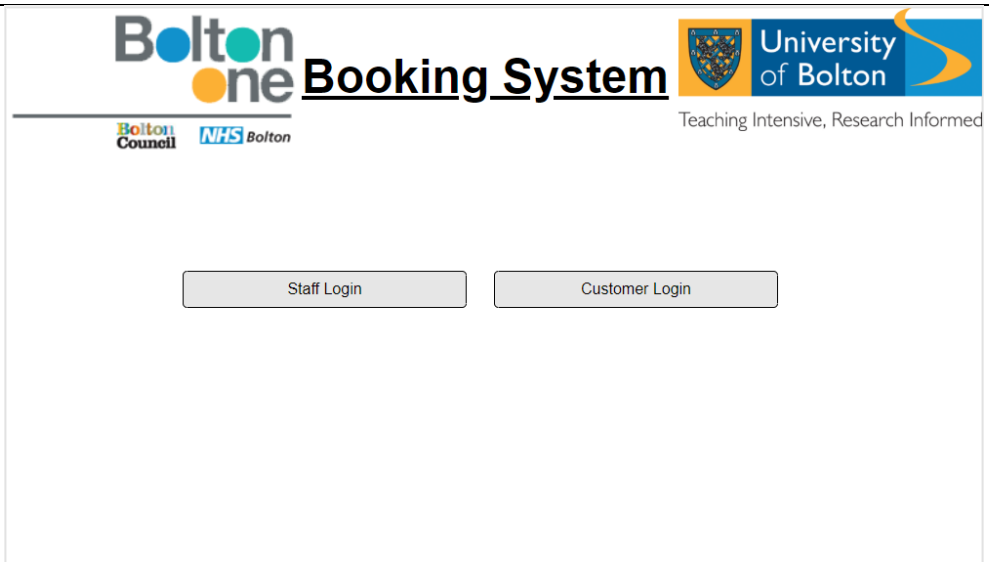
Objective #	Feature/ Process	Input Required	Output Created	Consideration of Output
1	Verify staff logins	Button to confirm Username and password entered Username and password from database	Text fields displayed for data to be entered Pop up which states that the user has logged in successfully Moves to next screen	The text fields are displayed to the user clearly allowing them to enter the username and password to login to the system This acts as feedback to the user to clearly indicate that they have logged onto the system successfully. The page will also change to the system menu screen.
3	Verify customer logins	Button to confirm Username and password entered Username and password from database	Text fields displayed for data to be entered Pop up which states that the user has logged in successfully Moves to next screen	The text fields are displayed to the user clearly allowing them to enter the username and password to login to the system This acts as feedback to the user to clearly indicate that they have logged onto the system successfully. The page will also change to the system menu screen.
4	Read from a text file – staff logins	Data within the text file	An array of the data is created	This is created to allow the comparisons of user input with the array list to verify access onto the system. This is required security purposes
5	Read from a text file – customer logins	Data within the text file	An array of the data is created	This is created to allow the comparisons of user input with the array list to verify access onto the system. This is required security purposes
2	Read from a text file – booking file	Button to move to view bookings page Data within the text file	An array list of booking data is created	This is created to allow for further output to the user in the form of a table therefore necessary to store data in an array
6	View bookings – display array data into a table	Array list of booking data	Table displayed onto the screen	This output displays the booking data in an effective and clear way with headings to the user
7	Sort/Filter bookings by date	Button choosing filter Booking data (Unfiltered)	Sorted data in table	This will update the table for the user which is an effective output to the user
8	Sort/Filter bookings by name	Button choosing filter Booking data (Unfiltered)	Sorted data in table	This will update the table for the user which is an effective output to the user
9	Add bookings to the database	Entering booking details Selecting Booking date, time and room	Pop up confirming the booking Array of booking data is updated adding new data	Simple output to clearly inform the user that the booking has been added. An array list of this data is updated, required to write to a file.

10	Write to a file – booking list	array containing objects of a record of the booking list	Text file updated	The database will be updated to include the new booking, which is necessary to update the tables too.
11	Write to a file – customer info	Customer array containing objects of a record	Text file updated	The database will be updated to include the new customer information, which is necessary to hold the customers information, where it can be stored securely
12	Write to file – customer history	Customer booking history array containing objects of a record	Text file updated	The database will be updated to include the new booking, which will allow customers to view their own booking history, as well as staff.
13	Write to a file – list of customers	array containing objects of a record of customers	Text file updated	The database will be updated to include the new booking, which is necessary to update the tables too.
14	Calculations to decide price	Calculation on length of booking with set price.	Displays price of booking on screen	Once the price is calculated it will take the user to the payments section where the price will be displayed on screen easy to see.
15	Calculations to apply discount on customers fee if required	Original price calculated and discount percentage	New total price of booking displayed on screen	Once the new total price is calculated it will take the user to the payments section where the price will be displayed on screen easy to see.
16	Amend booking list	User input on the new booking details	Array list of booking data is amended at a specific index	This will allow the array to be used in updating the database with the new amended booking
17	Update/ edit booking database	Amended array list of booking data	The database file is updated	This will therefore allow the tables displaying the bookings to be updated for staff and customers which is required to maintain consistency of the system.
18	Delete bookings from the database	Select the booking from the table Button to confirm deletion	Pop up confirming the deletion of the booking Updated database	The pop up is to confirm to the user that the booking has been deleted, the database will also be updated to remove the bookings
19	Searching for specific customer details	User inputs ID search value	Pop up displaying location of customer in table	This will let the user know the location of the booking which they will have the option to edit or delete the booking
20	Search by dates of booking	User inputs date search value	Pop up displaying location of customer in table	This will let the user know the location of the booking which they will have the option to edit or delete the booking

3.22 House Style

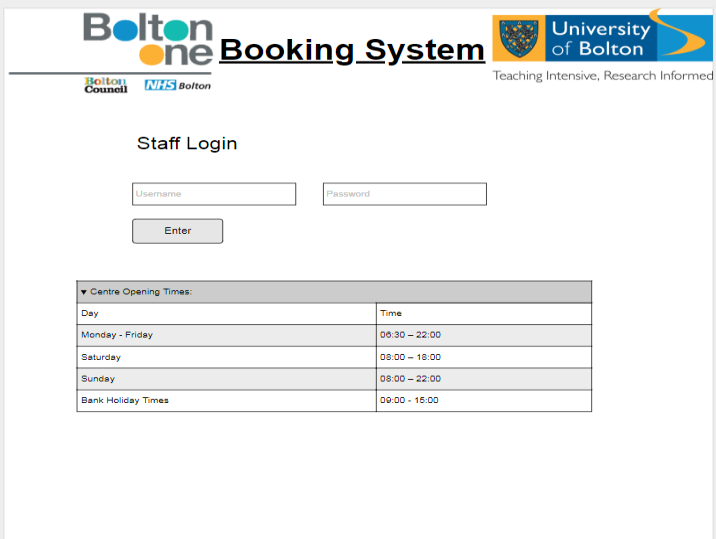
Element	Font Type, Size, Colour
Titles	Tw Cen Mt, Large Text size, Black
Labels	Tw Cen Mt, small text size, Black
Branding	Bolton one Bolton Council NHS Bolton University of Bolton Teaching Intensive, Research Informed
Jtexfields	Medium size
Jbutton	Medium size
Background	White background

3.22 Designs of Input & Output



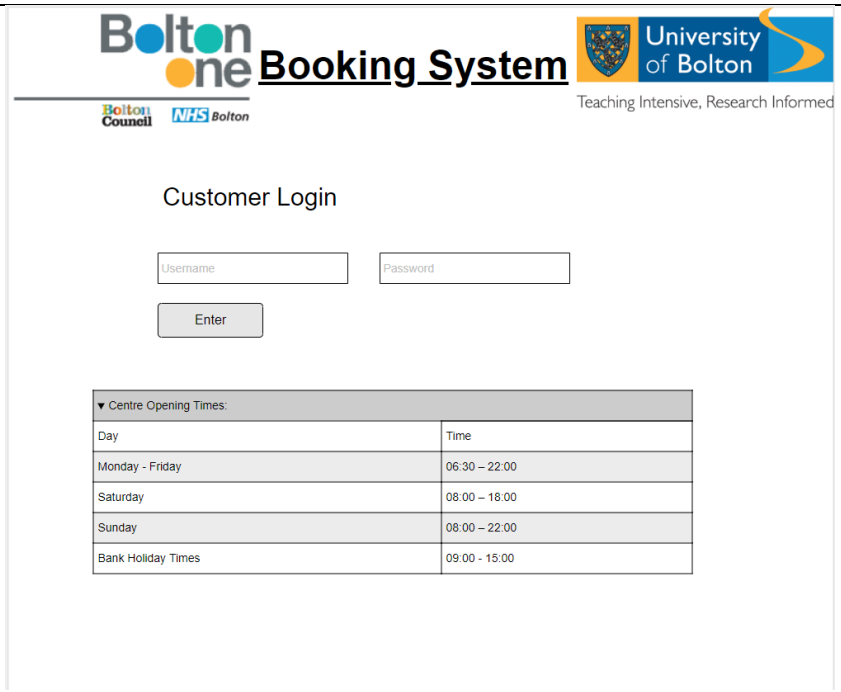
THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Home screen	Screen	The design for this screen is simple to the user and is clear what the purpose of the screen is. They have the option to login depending on if they are a staff member or customer. There is also a title again making it clear for the user to understand the system	Two buttons to choose the option of staff or customer login



THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Staff Login	Screen	The staff login screen is also clear and simple with large text boxes, for the username and password, to allow easy input into the system. There is also a button for the user to confirm the login and enter the system. I have chosen this design as it is a similar design to many other systems, therefore the user may be familiar with its touch and feel. There is also a table displaying opening times for the business. This is again large and clear with headings, allowing the user to understand the purpose of the table.	• Staff Login • Table • Consistent use of house style.

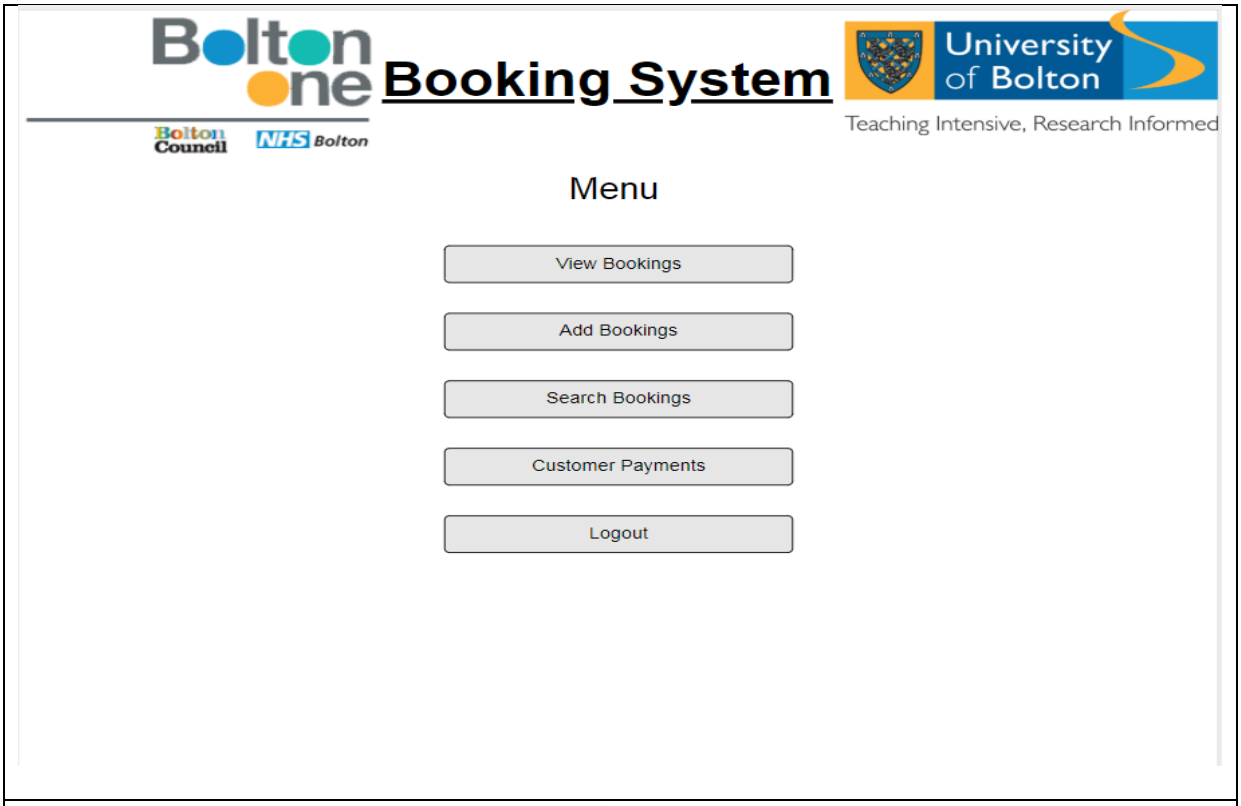


THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Staff Customer	Screen	The customer login screen is like the staff login with it being simple with large text boxes, for the username and password, to allow easy input into the system. There is also a button for the user to confirm the login and enter the system. I have chosen this design as it is a similar design to many other systems, therefore the user may be familiar with its touch and feel. There is also a table displaying opening times for the business. This is again large and clear with headings, allowing the user to understand the purpose of the table.	• Customer Login • Table • House style use

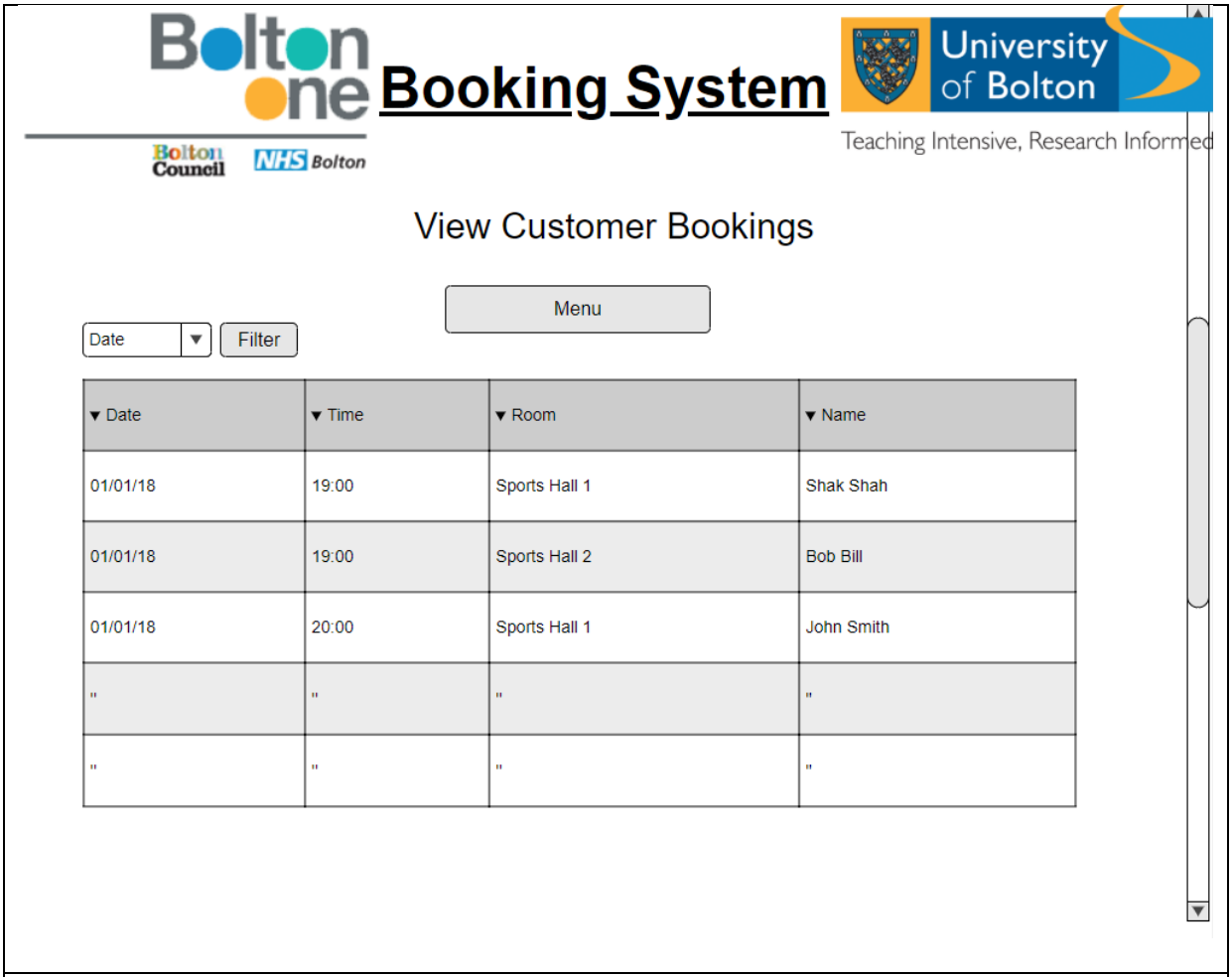
THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Customer Menu	Screen	For the customer menu page there is a simple design with multiple buttons each clearly stating the page it will turn to. There is also a large heading again showing the purpose of the screen to the user. This will be an easy page to navigate as again I have decided on a general look and feel, which should help the user comfortably navigate the system. The buttons are also large with large text for users to read. The customer will be allowed to view their own bookings, add to their own booking and view their payments. There is also a button allowing the user to logout which will return the user to the home screen.	• House style • Buttons



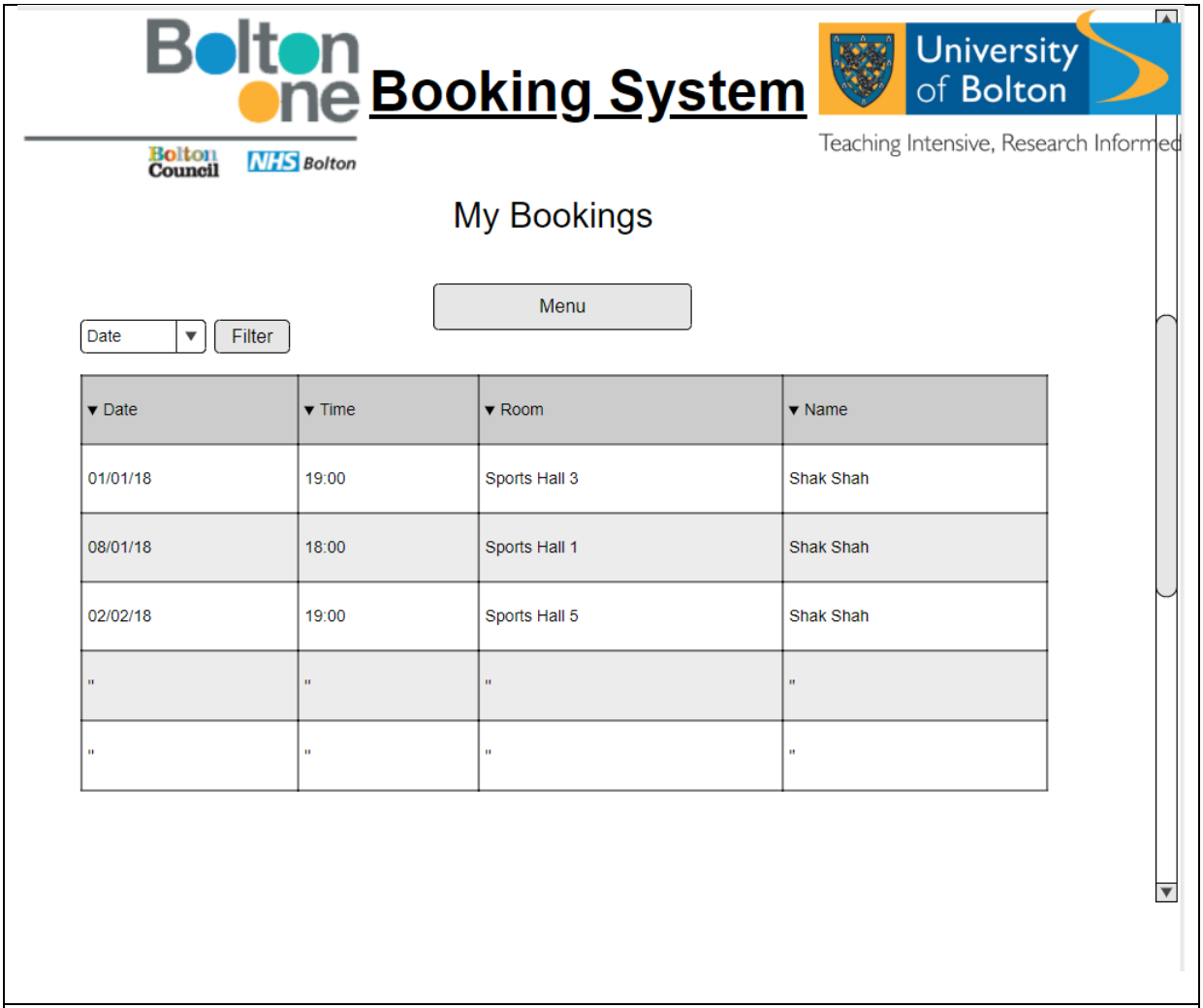
THIS DESIGN WAS CREATED IN MS-VISIO/Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Staff Menu	Screen	For the staff menu page there is the same design to the customer menu design with multiple buttons each clearly stating the page it will turn to. There is also a large heading again showing the purpose of the screen to the user. This will be an easy page to navigate as again I have decided on a general look and feel, which should help the user comfortably navigate the system. The buttons are also large with large text for users to read. The staff will be allowed to view all bookings, add/edit all booking and view all customer payments. There is also a button allowing the user to logout which will return the user to the home screen.	



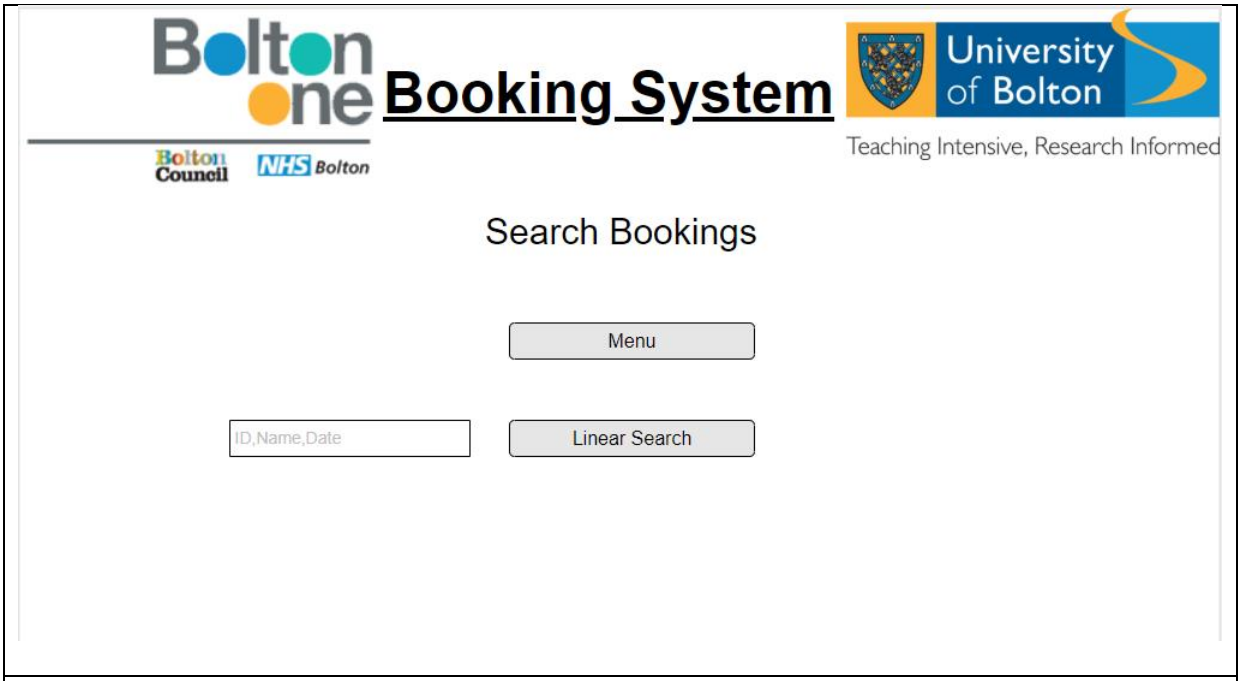
THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
View Customer booking – for staff use	Screen	The view bookings screen is also a simple design with a large table holding the booking data to be displayed easily to the user, allowing them to view bookings. There is also a large menu button to allow the user to return to the menu.	• Menu button • Bookings table • house style



THIS DESIGN WAS CREATED IN MS-VISIO/Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
View Customer booking – for staff use	Screen	The view bookings screen is also a simple design with a large table holding the booking data to be displayed easily to the user, allowing them to view bookings. There is also a large menu button to allow the user to return to the menu. I have also included a drop-down menu to allow the filtering of the bookings, by date, alphabetically, room etc. there is a filter button to apply the filters once the user has chosen.	• Menu button • Filter button • Drop-down menu • Bookings table • house style

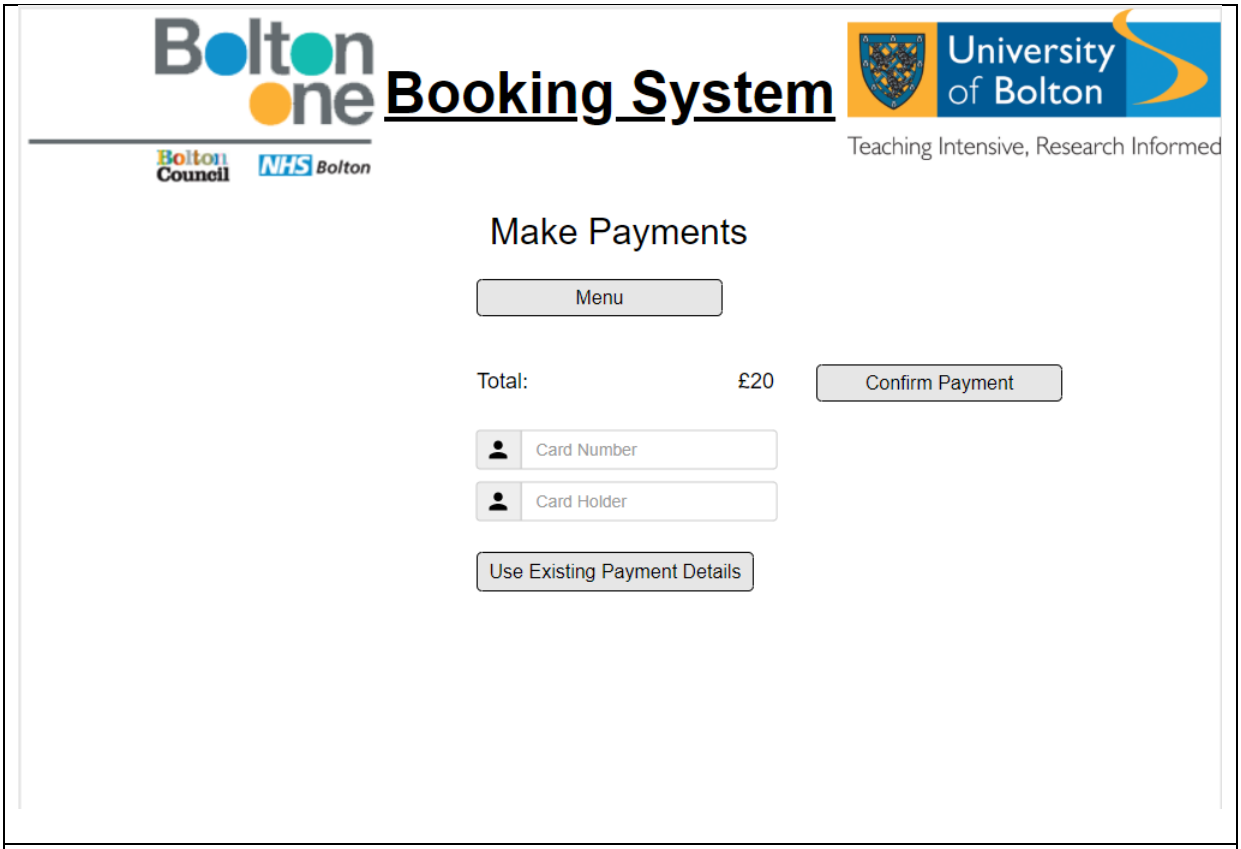


THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Search Bookings – Both customer and staff use	Screen	The purpose of this screen is the search bookings screen. There is a text field and a search button to allow the user to enter in the value that they are searching for. They can search by booking ID, name, date etc. Once the user clicks search a pop up will be displayed. This is again a simplistic design clearly showing its purpose to the user.	• Menu button • Filter button • Drop-down menu • Bookings table • house style

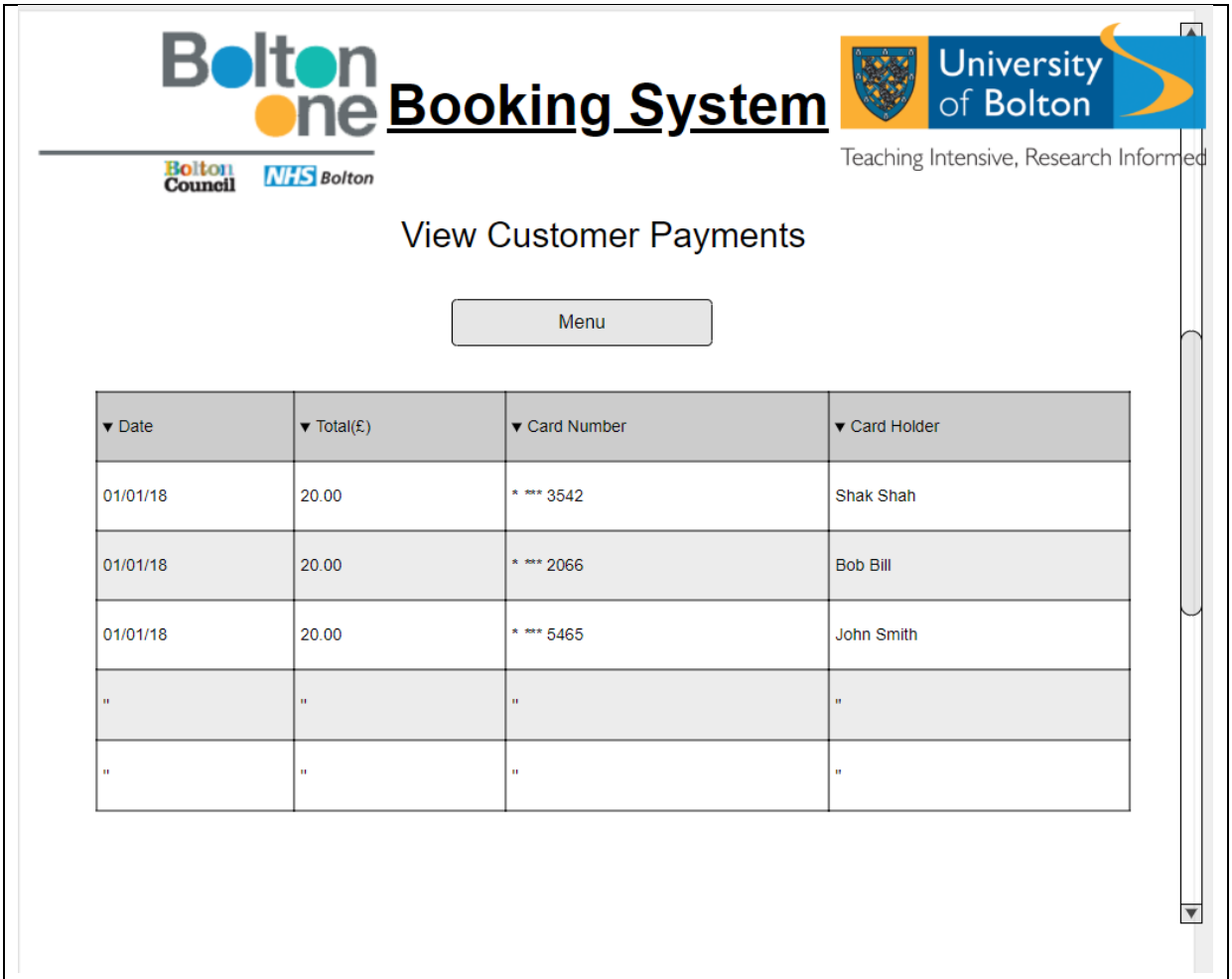
THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Add Bookings – Both customer and staff use	Screen	The add bookings screen follows a straight forward design with multiple text fields which allow the user to input all relevant details to the booking. The user can also choose a room using the drop-down menu. There is also a confirm button to save the booking. This design is very simple to understand and again follows a similar design to other systems. The is also a continued use of house style, following the theme of the system to make each page feel familiar.	• Menu button • Multiple text fields • Drop down menu • Confirm button • House style



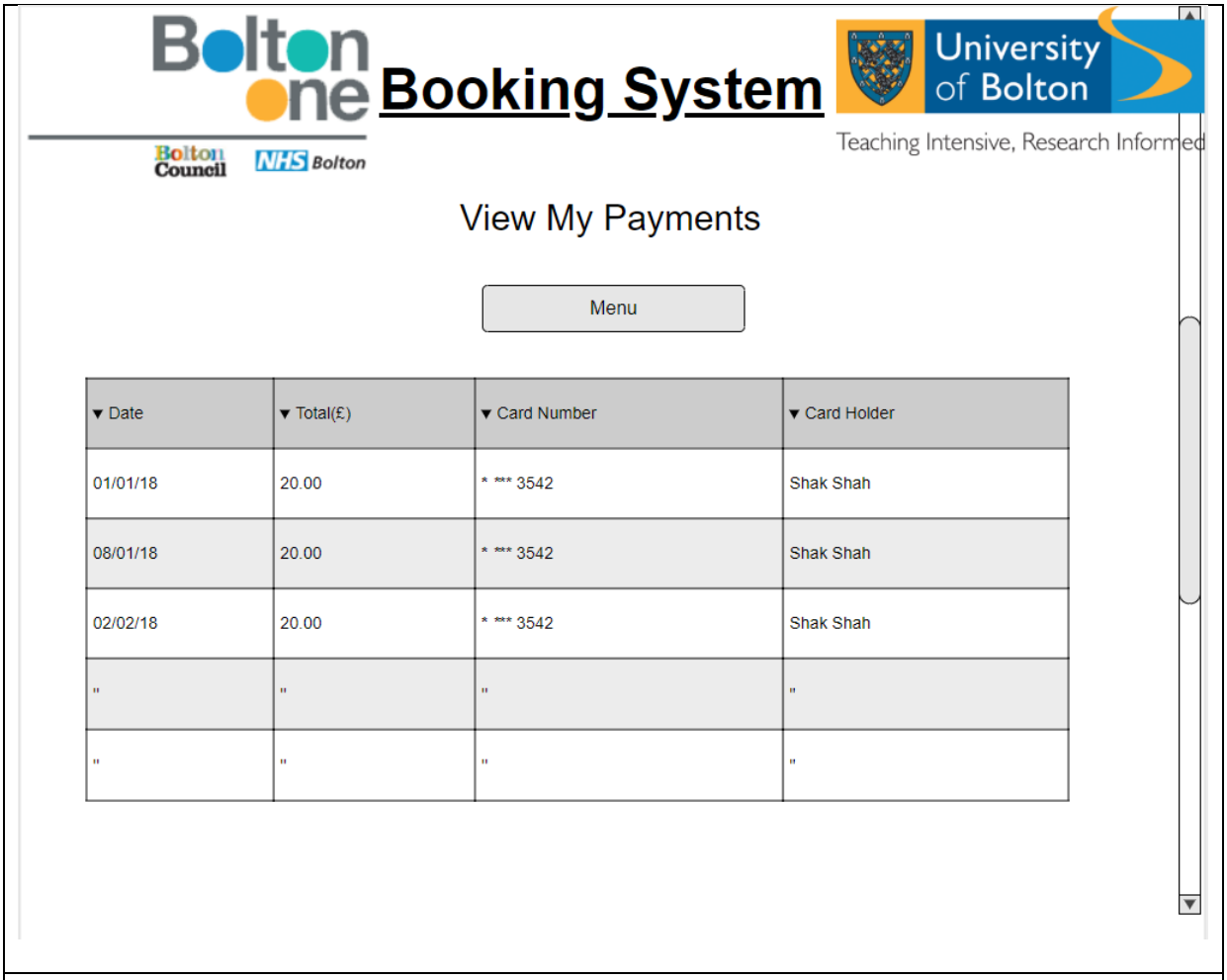
THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
Make Payments – Both customer and staff use	Screen	The make payments screen is very simple and easy to understand for the user. There are text fields to enter relevant payment details with a label clearly stating the price of their booking. There is also a button to use existing details which will auto enter the details into the text fields. This design is quite basic yet effective in displaying its purpose as it follows a similar theme to the rest of the system.	• Menu button • Payment text fields • Label to display price • Use existing button • Confirm button • House style



THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
View all Payments – for staff use	Screen	This screen is similar to the view bookings page. This output is simple and clear in displaying the list of all customer payments to the staff user. This again will have a general touch and feel for the user making it easy to use. There is also continued use of house style branding.	• Menu button • Payments table • House style

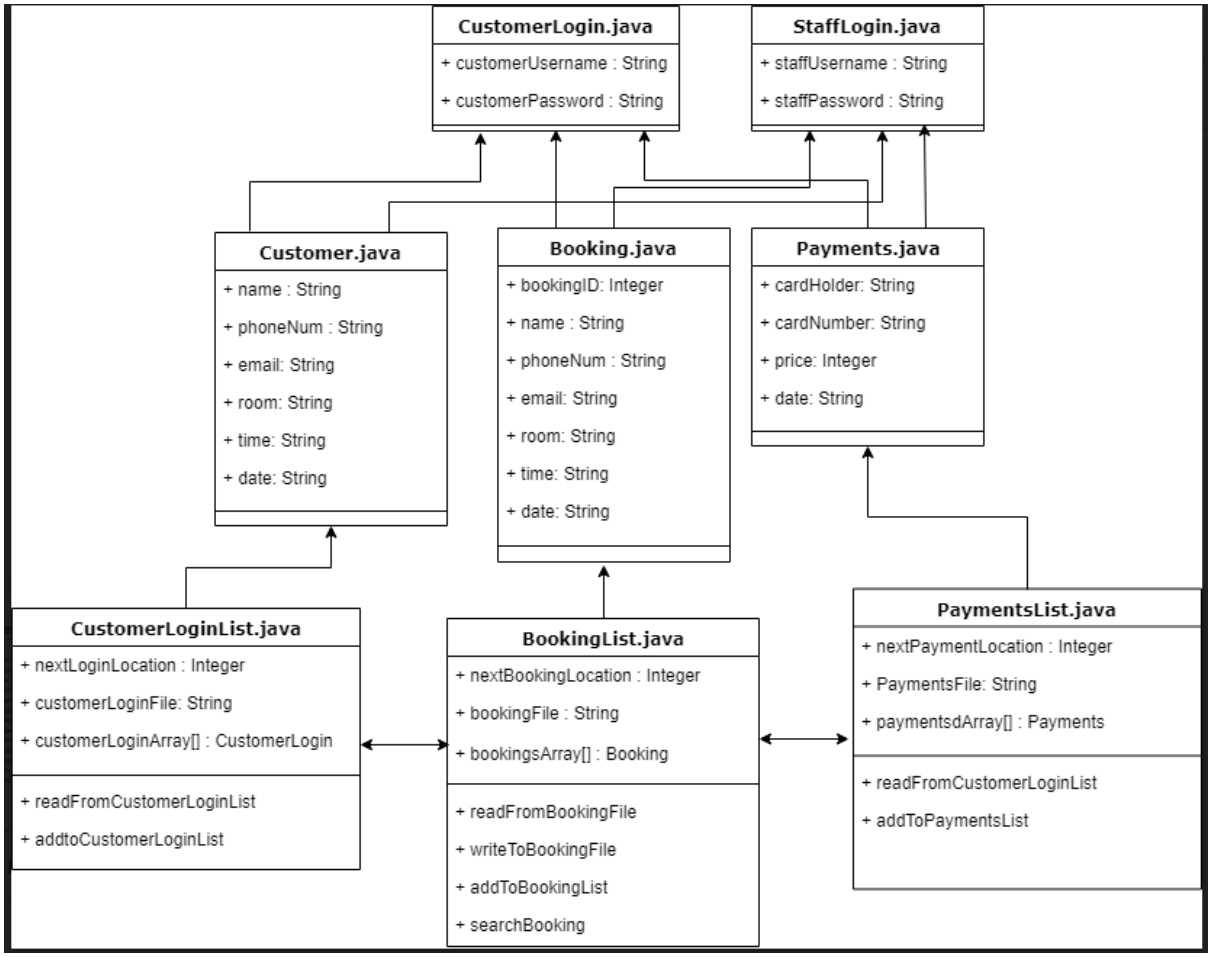


THIS DESIGN WAS CREATED IN Moqups AND IS NOT AN IMPLEMENTATION

Name of Design	Format	Consideration of Design & Purpose	Features Included
View Payments – for customer use	Screen	This screen is identical to the staff payments page but shows only the logged in customers payments only. This again will have a general touch and feel for the user making it easy to use. There is also continued use of house style branding.	• Menu button • Payments table • House style

3.3 Files & Data Structures

3.3 Class Diagram



3.32 Methods of Access inc. Files

Entity & Classes	Justification for Method of Access
Staff StaffLogin.java StaffLoginList.java	**5 Records accessed serially** There are only 5 members of staff, so it would only hold the records on those staff, holding their logins. The access to these records is serial access as the logins are stored in which ever order they have been created therefore the file will not be ordered. Also, as it is a small list it will not cause any inefficiencies even though the file is accessed every time the system is used by a staff member.
Customers CustomerLogin.java CustomerLoginList.java CustomerInfo.java CustomerInfoList.java	**100+ records accessed index sequentially for the customer logins** The login details are stored in **an unordered list** therefore the login details entered could be anywhere within the list of logins. I have decided **that accessing the data serially**, to allow each item compared to the input data, would **be too long of a process** as there could be many customers in the list. This is data is also **accessed frequently** as many customers can log on to the system at once therefore could cause strain on the system. Instead I have decided to read the file and store the data from the file into an array of data. This will allow me to access the data within the array via an index which will reduce search times significantly helping improve the efficiency of the system. **100+ records accessed index sequentially for the customer information** The customer information records are stored in a text file just as the logins are. These are also stored in an unordered list and for the same reasoning the data will not be accessed serially as it will be inefficient to do so. The data will therefore be accessed in sequentially using an index as the data will be stored into an array, as it will need to be accessed regularly, which if stored into an array will mean it can be accessed from RAM which has a faster retrieval speed.
Bookings Booking.java BookingList.java [customer]Booking.java [customer]BookingList.java	**1000+ records accessed index sequentially** The bookings are stored into a text file the same as the others, in the order entered, making the booking file an unordered list. This again would make it inefficient to use serial access to read the file for tables, searches etc. therefore I also decided to access the file data by reading the file once when the system is run which stores the data into an array which could be easily and efficiently accessed multiple times during the processes of the system. As there is so many records for the booking this is the most suitable as well as time and cost-effective method of access.
Payments Payments.java PaymentsList.java [customer]Payments.java [customer]PaymentsList.java	**1000+ records accessed index sequentially** Again, I will access the payments through indexed sequential access as there is a large amount of records meaning serial access will not be a viable option. Instead the data will be read from a file into an array once the system starts running allowing access to the data at any time, a lot faster. This will be done as the file is also unordered meaning it will be easier to access files using the fastest method.

3.33 Data Structure Designs

StaffLogin.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
userName	-	String	<= 15	Staff1	Y
password	-	String	<= 15	StaffPassword	Y

CustomerLogin.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
customerUserName	-	String	<= 15	Shak	Y
customerPassword	-	String	<= 15	Shak12345	Y

CustomerLoginList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
customerLoginArray[]	-	CustomerLogin	[100]	Customer login Record	-
nextLogin Location	-	Integer	-	3	-
customerLoginFile	-	String	-	CustomerLoginInfo.txt	-

Booking.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
bookingID	PK	Integer	-	5	-
name	FK	String	<= 50	Shak Shah	Y
phoneNum	-	String	9-12	01204 553366	Y
email	-	String	<= 50	Shak@gmail.com	Y
room	-	String	6	Room 1	Y
time	-	String	5	12:00	Y
date	-	String	8	01/12/18	Y

BookingList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
bookingsArray[]	-	Booking	[1000]	Booking Record	-
nextBooking Location	-	Integer	-	3	-
BookingFile	-	String	-	BookingData.txt	-

Customer.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
name	PK	String	<= 50	Shak Shah	N
phoneNum	-	String	9-12	01204 553366	N
email	-	String	<= 50	Shak@gmail.com	N
room	-	String	6	Room 1	N
time	-	String	5	12:00	N
date	-	String	8	01/12/18	N

CustomerList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
customerArray[]	-	Customer	[100]	Customer Record	-
nextCustomer Location	-	Integer	-	3	-
customerFile	-	String	-	CustomerInfo.txt	-

Payments.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
cardNumber	-	String	16	1531 8216 713 5468	Y
cardHolder	-	String	<= 50		Y
price	-	Float	6	£20.00	N
date	-	String	8	01/12/18	N

PaymentList.java

Field Name	Key Field	Data Type	Length	Example Data	Validate?
paymentsArray[]	-	Payment	[10]	Payment Record	-
nextPayment Location	-	Integer	-	3	-
PaymentsFile	-	String	-	PaymentInfo.txt	-

3.4 Validation

#	Class	Field	Val. Type Val. Rule/Description Error Message
1	StaffLogin.java CustomerLogin.java	Username	**Lookup** **Presence** Validation for this field is to check whether the username entered by the user matches with a username in the database containing usernames This is to check if the username entered is valid. *"Error, please enter a correct username."* *"Error, please enter a username"*
2	StaffLogin.java CustomerLogin.java	Password	**Lookup** **Presence** Validation for this field is to check whether the password entered by the user matches with a password in the database containing usernames. This is to check if the username entered is valid. *"Error, please enter a correct password."* *"Error, please enter a password"*
3	Booking.java	Name	**Type** **Presence** **Length** For this field the validation rule is that it must be a string entered by the user, to allow a sensible name. Another validation rule for this field would be a presence check, as this would allow the system to detect if input has been entered A length check could also be implemented to have a sensible amount of characters entered. E.g no more than 30 characters *"Error please enter a valid name"* *"Error, please enter a name"* *"Error, please enter a name less than 30 characters"*
4	Booking.java	Phone Number	**Type** **Presence** **Length** The type check would ensure the data entered is valid for a phone number. A string would be required. The presence check would again detect if the data has been inputted by the user The length check would detect whether the phone number is the correct length. *"Error, please enter a valid phone number"* *"Error, please enter a phone number"* *"Error, phone number is too short or too long"*
5	Booking.java	Email	**Presence** **Format** The presence check would again detect if the data has been inputted by the user

			The format check would detect if an email address has been entered legally, including the '@' *"Error, please enter an email address"* *"Error, please enter a suitable email address"*
6	Booking.java	Time	**Format** The format check would detect if the time has been entered legally, as 12:00 *"Error, please enter a time with the correct format"*
7	Booking.java	Date	**Format** The format check would detect if date has been entered legally, as dd/mm/yy *"Error, please enter a date with the correct format"*
8	Booking.java	Room	**Presence** This presence check will detect if the value of the combo box chosen has a value. *"Error, please select a room"*
9	BookingList.java	Search value	**Presence** The presence check would again detect if the data has been entered by the user *"Error, please enter a search"*
10	Payments.java	Card holder name	**Presence** **Type** The presence check would again detect if the data has been entered by the user For this field the validation rule is that it must be a string entered by the user, to allow a sensible name. *"Error, please enter a card holder"* *"Error, please enter a suitable name"*
11	Payments.java	Card number	**Presence** The presence check would again detect if the data has been entered by the user The type check would ensure the data entered is valid for a card number. A string would be required. *"Error, please enter a card number"*

3.5 Processes

3.51 Objectives Links with Data/Files

Objective # and Name (Investigation)	Explanation of Process inc. Array / Files Used
5 - Read Staff Login Details	StaffLoginInfo.txt This process allows the staff file to be read in order to verify logins. The staff file will be opened and read from with each line in the text file being compared to the user input to find the correct username and password.
2 - Read Customer Login Details	CustomerLoginInfo.txt customerLoginArray[] This process allows for the customer login file to be read. Once the file is opened each line within the file is read as long as the line is not null.
2 - add to array Customer Login Details	CustomerLoginInfo.txt customerLoginArray[] This will allow for the customer logins within the file to be stored into an array. This array will then be stored into an object of the class CustomerLogin. The logins entered by the user will then be verified as the user input would be compared to different indexes of the array.
6 – Read from the booking file	BookingData.txt bookingsArray[] Payments.txt This process allows for the booking file to be read. Once the file is opened each line within the file is read if the line is not null. The data read is split up where there is a ','
7– add booking file data to an array	BookingData.txt bookingsArray[] Once the data from the text file has been read, after each line the data is stored into an array containing a list of the booking data. This array is then stored within an object within the class Booking to be called, to access the array data inside. This will allow the data of the bookings to be accessed for many objectives such as searching, adding, and view booking data.
13 - Write to booking file	BookingData.txt bookingsArray[] To add a new booking, the data entered by the user is stored into an object of the booking class. This object is used to store the data within the array of bookings by adding it to the next free index of the array. Using this array, the data is written to the text file, storing each record of data on a new line.
14 - Write to the customer file	customerFile.txt customerArray[] To add new customer data to the customer file some of the attributes of the customers related bookings are stored into an object of the customer class. This object is used to add the data to the customer array which is written to the file each index on a new line of the file.
15 - Write to the payments	payments.txt payments[] The data for payments is stored into an object which is used to add the data into the array holding the list of payments. This data is then written to the file with each record being stored on a new line.

3.52 Processing Routines

```
// Objective 5 – Read from Staff Login
START
      DECLARE username as String
      DECLARE password as String
      DECLARE file = "StaffLoginList.txt"
      INPUT username
      INPUT password
      OPEN file
      WHILE Each line in file != NULL
      CALL staff verify method
      ENDWHILE
END
```

```
// Objective 1 – Verify Staff Login
START
      IF username = line.staffUsername AND password =
line.staffPassword
      THEN
            OPEN SystemGUI
      ENDIF
      ELSE
            OUTPUT "Invalid Username and Password"
      END ELSE
END
```

```
// Objective 2 – Read from Customer Login
START
      DECLARE username as String
      DECLARE password as String
      DECLARE file = "CustomerLoginList.txt"
      INPUT username
      INPUT password
      WHILE Each line in file != NULL
      CALL add to customer login array method //OBJ 3
      CALL customer verify method
      ENDWHILE
END
```

```
// Objective 3 – Add to Customer Login Array
START
     FOR i=0 to customerLoginArray[nextFreeLocation]
          customerLoginArray[i] = line
          next line
         i++
     ENDFOR
END
```

```
// Objective 4 – Verify Customer Login
START
     FOR i=0 to customerLoginArray[nextFreeLocation]
         IF username = customerLoginArray[i].username AND
            password = customerLoginArray[i].password
         THEN
             OPEN SystemGUI
         ENDIF
         ELSE
            OUTPUT "Invalid Username and Password"
         END ELSE
END
```

```
// Objective 6 – Read from a Text File: Booking File
START
      DECLARE bookingsArray as String Array
      DECLARE file = "BookingData.txt"
      OPEN file
      WHILE Each line in file != NULL
           CALL Add to booking array method //OBJ 6
      ENDWHILE
END
```

```
// Objective 7 – Add to Booking Data Array
START
      FOR i=0 to bookingsArray[nextFreeLocation]
           bookingsArray[i] = line
           next line
          i++
      ENDFOR
END
```

```
// Objective 8 – View Bookings
START
      CALL read booking file method //OBJ 6
      CALL Add to booking array method //OBJ 7
      CREATE TableModel
      DELCARE TableModelHeadings = "ID","Date","Time","Room","Name"
      OUTPUT TableModel
END
```

```
// Objective 9 – Sort/Filter Bookings by Date
START
      DECLARE bookingsArray as String Array
      DECLARE file = "BookingData.txt"
      OPEN file
      PERFORM bubble sort on date
      CALL view bookings method //OBJ 8
END
```

```
// Objective 10 – Sort/Filter Bookings by Name
START
      DECLARE bookingsArray as String Array
      DECLARE file = "BookingData.txt"
      OPEN file
      PERFORM bubble sort on name
      CALL view bookings method //OBJ 8
END
```

```
// Objective 11 – Add Bookings to the bookings Array
START
      DECLARE bookingsArray as String Array
      DECLARE customerArray as String Array
      DECLARE file = "BookingData.txt"
      INPUT name as String
      INPUT phoneNumber as String
      INPUT email as String
      INPUT date as String
      INPUT time as String
      INPUT room as String
      bookingsArray[nextFreePostion]=name+phoneNum+email+date+time+room
      customerHistoryArray[nextFreePostion]= date+time+room
      customerArray[nextFreePostion]=name+phoneNum+email

END
```

```
// Objective 12 – Write to a Text File: Booking List
START
      DECLARE bookingsArray[] as String Array
      DECLARE file = "BookingData.txt"
      OPEN file
      WHILE i< bookingsArray.[nextFreeLocation]
            THEN
                  READ file
                  Write.(bookingsArray[i]) to line
                  New line
                  i++
      ENDWHILE
      CLOSE file

END
```

```
// Objective 13 – Write to a Text File: Customer Info
START
      DECLARE customerArray[] as String Array
      DECLARE file = "CustomerInfo.txt"
      OPEN file
      WHILE i< customerArray.[nextFreeLocation]
            THEN
                  READ file
                  WRITE (customerArray[i]) to line
                  New line
                  i++
      ENDWHILE
      CLOSE file

END
```

```
//Objective 14 – Write to a Text File: Customer History
START
      DECLARE customerHistoryArray[] as String Array
      DECLARE file = "CustomerHistory.txt"
      OPEN file
      WHILE i< customerHistoryArray.[nextFreeLocation]
            THEN
                  READ file
                  WRITE (customerHistoryArray[i]) to line
                  New line
                  i++
      ENDWHILE
      CLOSE file
END
```

```
// Objective 15 – Write to a Text File: Customer List
START
      DECLARE customerArray[] as String Array
      DECLARE file = "CustomerInfo.txt"
      OPEN file
      WHILE i< customerArray.[nextFreeLocation]
            THEN
                  READ file
                  WRITE (customerArray[i]) to line
                  New line
                  i++
      ENDWHILE
      CLOSE file

END
```

```
// Objective 16 – Calculations for price
START
      SET HOURLYRATE = 20
      DECLARE startVal as Integer
      DECLARE endVal as Integer
      DECLARE duration as Integer
      DECLARE price as Integer

      duration = endVal – startVal
      price = duration *  HOURLYRATE
      OUTPUT "Total Price:" + price
END
```

```
// Objective 17 – Amend Booking List
START
      DECLARE bookingsArray as String Array
      DECLARE customerHistoryArray as String Array
      DECLARE customerArray as String Array
      INPUT bookingID as Integer
      INPUT name as String
      INPUT phoneNumber as String
      INPUT email as String
      INPUT date as String
      INPUT time as String
      INPUT room as String
      bookingsArray[bookingID]=name+phoneNum+email+date+time+room
      customerHistoryArray[bookingID]= date+time+room
      customerArray[bookingID]=name+phoneNum+email
END
```

```
// Objective 18 – Update booking database
START
      DECLARE bookingsArray[] as String Array
      DECLARE file = "BookingData.txt"
      OPEN file
      WHILE i< bookingsArray.[nextFreeLocation]
            THEN
                  READ file
                  Write.(bookingsArray[i]) to line
                  New line
                  i++
      ENDWHILE
      CLOSE file
END
```

```
// Objective 19 - Delete booking from database
START
      DECLARE bookingsArray as String Array
      DECLARE customerArray as String Array
      DECLARE customerArray as String Array
      INPUT bookingID as Integer
      Remove.bookingsArray[bookingID]
      Remove.customerHistoryArray[bookingID]
      Remove.customerArray[bookingID]
END
```

```
// Objective 20 - search for specific customer by name
START

      DECLARE customerArray as String Array
      DECLARE file = ".txt"
      OPEN file
      WHILE Each line in file != NULL
           FOR i=0 to customerArray[nextFreeLocation]
               customerArray[i] = line
               next line
              i++
           ENDFOR
     ENDWHILE
      INPUT searchVal
     FOR i< customerArray[nextItemLocation]

          IF customerArray[i].name == search

              OUTPUT  searchVal + "present at location "+ i
               i++
          END IF
          ELSE
              OUTPUT "Search not found"
          END ELSE
END
```

```
// Objective 21 - search for booking by date
START

      DECLARE bookingArray as String Array
      DECLARE file = ".txt"
      OPEN file
      WHILE Each line in file != NULL
           FOR i=0 to bookingArray[nextFreeLocation]
               bookingArray[i] = line
               next line
              i++
           ENDFOR
      ENDWHILE
      INPUT searchVal
      FOR i<  bookingArray[nextItemLocation]

           IF  bookingArray[i].date == search

               OUTPUT  searchVal + "present at location "+ i
                i++
          END IF
          ELSE
              OUTPUT "Search not found"
          END ELSE
END
```