

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

Name: Salah Abdo

Student ID: 6179614

Course: Computer Science

Module: 260CT Software Engineering

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

Group Members:

- Salah Abdo
- Sam Kelly
- Zehua Zhou
- Martin metodiev
- Adhitama Budi
- Reece Holness

Functionalities:

Booking – Salah

Registration – Adhitama

Viewing Booking – Zehua

Time Table – Sam

Manager deleting – Reece

Manager Adding – Martin

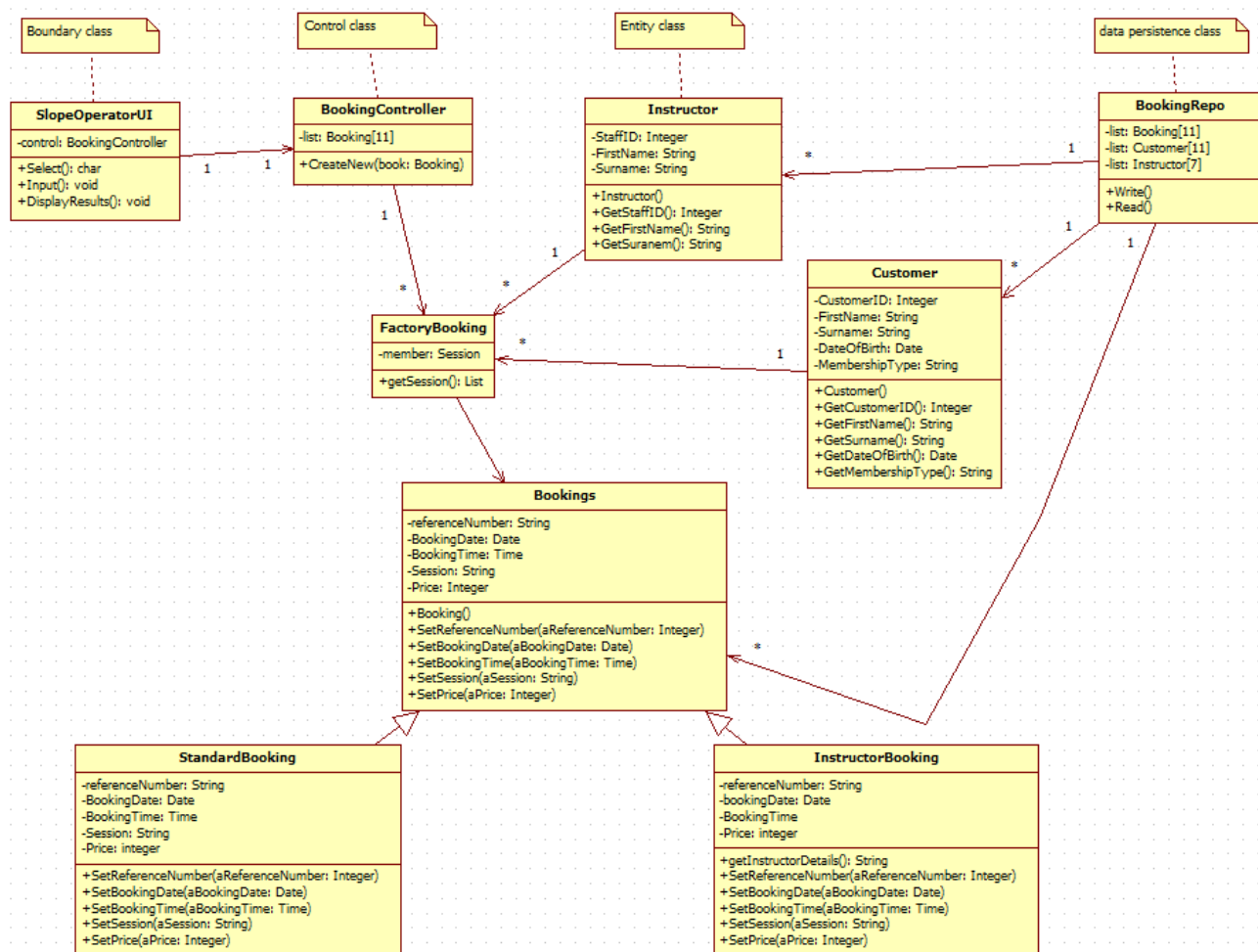
Manager updating - Adhitama

Student Name: Salah Abdo

SID: 6179614

Task 1: To complete design specification, implementation, testing and integration – evidence with a report supported by research with references in the CU Harvard referencing style in a maximum of 1,000 words

Design specification



For my Functionality, I had to Create a design model for the booking of the system. I used the Four Layer Architecture layer this allowed me to structure in the most appropriate way for the overall application architecture, it allowed me to structure my functionality which helped me use as a guide to provide my code. The four-layered model; User interface layer, application/business logic layer, domain layer and data access layer. Each of these layers provides a different purpose. The User interface layer is the boundary class which represents the graphical user interface to handle the input and output. With my design, I created a class called the Slope operator user interface, this represents the graphical user interface of the booking interface. The Application/business logic layer is the control class which coordinates the application logic for each use case. With my design, I added 3 different class; the instructor, customer, and booking. In my design, I've got a class called the booking Controller, it allows the other class to be controlled through this controller creating low coupling since I'm reducing the amount of connection between objects. The domain layer is the entity class which data objects contained within that class from the business domain from where the layer of the those that are not recognized at the analysis stage. The instructor class is used primarily for getting information about the instructor which can be used in the booking. The customer class is needed as when making a booking the customer details are required linking the booking to the customer. The booking class I used for the booking information. The data access layer is the data persistence class which handles the reading and writing to a database. In the design, I've got a class that's called booking repo which handles the reading from the customer and instructor and writing for the booking.

For my design, I used GRASP design Principles, what grasp does is that it helps decide which responsibility should be assigned to which class or object. It helps identify the objects and responsibilities from the problem domain and identify how different object interact with each other. (Danya Rao, grasp design principles n.d).

The Gang of Four Design pattern needed to be used for my design specification, there is 23 Gang of four design pattern I'm going to research about a couple of them and evaluating them. This give me the best knowledge so I can make an appropriate decision on which of the design pattern to use.

Creational design pattern would be the post appropriate one to use as my functionally is creating a booking. This design pattern allows me to give ways to instantiate single objects of related objects (black wasp 2009). There is a total of 5 creational design patterns which are; Abstract factory, builder, factory method, prototype, and singleton.

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

Abstract factory pattern- This design pattern is one of the Gang of four design pattern which is used to control class instantiation it provides a client with a set of related / dependent objects. The family of objects is created by the factory and is determined at the run time (black wasp 2008).

Builder pattern- This is another design pattern of Gang of four. This pattern is also used to control class instantiation. The pattern is used to create complex objects with established parts which must be created in the same order. The external class controls the construction algorithm (black wasp 2008). The benefits of the builder pattern are the when you need to do a lot of different things to build an object. You must create plenty of nodes and attributes to get your final object which will have the level of dependencies between objects creating high coupling. A factory is used when the factory can easily create the object with one method call (stack overflow 2008).

Prototype pattern- The prototype is the third Gang of four design pattern. This design pattern is also used to control class instantiation and objects generation. The prototype pattern allows to instantiate a new object, this is done by copying all of the properties of another object by creating an independent clone (black wasp 2008). The benefits of the prototype pattern are that it eliminates the overhead of initializing an object. Another benefit is that it also simplifies and can enhance the use case where multiple objects of which have similar types will have mostly similar data.

Singleton pattern- The singleton pattern is another gang of four design pattern. This pattern also used to control class instantiation. The singleton pattern makes sure that only one objects of a particular class are ever created (black wasp 2008).

For my design, I used one of the gang of four design patterns, the factory method. The factory method is used to control class instantiation and used to replace the class constructors and abstracting the process of the object. This allows different types of objects instantiated which can be determined at run time (black wasp 2008). The benefits of using the factory pattern are that if you want to match creation of different objects through a shared factory (stack overflow 2012). This can be used to abstract away constructors and details for each subclass of a common parental class, or it can provide default arguments to a constructor. I used the factory method for this design because it allowed an object to be created without exposing the creation logic the external using a common interface. As you can see with my design the factory pattern was applied for the booking functionality as there can be two types of booking, a normal booking with no instructor or a booking with an instructor

Implementation

Prototype 1

Booking

Sphere Booking and Check-in

First Name

Surname

Date of birth

Session

Date

Time

Book

Timetable

Home

The booking GUI allows the slope operators to enter the customer detail and booking detail to make the booking.

Each entry will allow the slope operator to input data which will store the variable into a variable which can then be used later on. The data entries will be used to calculate price, get customer membership information and also to create the booking itself.

Each button serves a different purpose the booking will calculate the booking cost, generate a reference number and store the information into the date base. The Time table button is an integration of Sam's functionality, which displays the time table of the slope and the instructor. The home button will bring the user back to the main menu.

Salah Abdo
260CT- Software Engineering
Student ID: 6179614
Booking Code

```
'''  
Sphere Booking and Check-in  
260CT  
prototype  
Salah Abdo  
Python 3
```

This code is for the booking. It allows the slope operator to make a booking by inputting all the correct information required. It will also calculate the price. The price depends of what kind of session the customer book and if they have a membership do they have the correct one to give a discount.

```
'''  
from Main import *  
  
class Booking():  
  
    def __init__(self, master, mainwnd):  
  
        self.mainwnd= mainwnd # store the 'self.master` of the main  
window  
        self.master = master  
  
        self.master.geometry("1080x800+200+200")  
        self.master.title("Sphere Booking and Check-in")  
  
        self.conn = sqlite3.connect('Database.db') # connects to the  
database  
        self.c = self.conn.cursor()  
  
        Label(self.master, text="Sphere Booking and Check-  
in", fg="black", font=("Helvetica", 25)).grid(row=0, column=2)  
        Label(self.master, text=" ").grid(row=1)  
        Label(self.master, text="First Name").grid(row=2)  
        Label(self.master, text=" ").grid(row=3)  
        Label(self.master, text="Surname").grid(row=4)  
        Label(self.master, text=" ").grid(row=5)  
        Label(self.master, text="Date of birth").grid(row=6)  
        Label(self.master, text=" ").grid(row=7)  
        Label(self.master, text="Session").grid(row=2, column=2)  
        Label(self.master, text=" ").grid(row=3)  
        Label(self.master, text="Date").grid(row=4, column=2)  
        Label(self.master, text=" ").grid(row=5)  
        Label(self.master, text="Time").grid(row=6, column=2)  
        Label(self.master, text=" ").grid(row=7)  
  
        self.firstName = Entry(self.master)  
        self.surname = Entry(self.master)  
        self.dob = Entry(self.master)  
        self.session = Entry(self.master)  
        self.date = Entry(self.master)  
        self.time = Entry(self.master)  
        # stores the entry made into a variable
```

```
self.firstName.grid(row=2, column=1)
self.surname.grid(row=4, column=1)
self.dob.grid(row=6, column=1)
self.session.grid(row=2, column=3)
self.date.grid(row=4, column=3)
self.time.grid(row=6, column=3)
# positioning of the entry

Button(self.master, text='Book', command=self.checkValue,
font=("Helvetica",15, "bold italic")).grid(row=11, column=4,
sticky=W, pady=4)
Button(self.master, text='Timetable', command=self.table,
font=("Helvetica",15, "bold italic")).grid(row=12, column=4,
sticky=W, pady=4)
Button(self.master, text='Home', command=self.goHome,
font=("Helvetica",15, "bold italic")).grid(row=13, column=4,
sticky=W, pady=4)

# padx = horizonta
# pady = verticle
# .grid(row=6, column=2)
# row = increase the number you move down, decrease the
number you move up.
# column = increase the number you move right, decrease the
number you move left.

def goHome(self):
    self.master.destroy() # close the current Member window
    self.mainwnd.update() # update the home window
    self.mainwnd.deiconify() # un-minimize the home window

def table(self): # Runs the time table functionality
    root1=Toplevel(self.master)
    timetable=timeTable(root1,self.master)
    # Opens the slope or instructor timetable

def closeDB(self): # closes the database
    self.c.close()
    self.conn.close()
    self.goHome()

def insertBooking(self):
    refNum = self.ref
    customerID = self.customer
    session = self.session.get()
    date = self.date.get()
    time = self.time.get()
    checking = "False"
    price = self.price
    sessionNumber = self.sessionNum

    self.c.execute("INSERT INTO Booking (Ref_number, customerID,
session, date, time, checking, price) VALUES(?, ?, ?, ?, ?, ?, ?)", #
stores all the data into booking table
```



```
(refNum, customerID, session, date, time,
checking, price))

        self.c.execute("UPDATE Customer SET number_of_sessions = (?)
WHERE CustomerID = (?) ",
                        (sessionNumber, customerID))
        self.conn.commit()

        self.closeDB()

    def id_generator(self,size=16, chars=string.ascii_uppercase +
string.digits): # Creates a random 16 digit value with numbers and
characters, used as reference number
        self.ref=''.join(random.choice(chars) for _ in range(size))
        self.insertBooking()

    def calculate(self):
        session = self.session.get()
        cID = self.customer

        normal = 50 # price per hour
        instructor = 20 #price per hour
        self.price = 0

        self.c.execute("SELECT Membership_Type FROM Member WHERE
Customer_ID = ?", # selects what membership the customer had, this
will be used to calculate the price
                        (cID))

        self.status =self.c.fetchone()

        membershipType = self.status[0]

        if membershipType.lower() == "standard": # use the customers
membership to give a discount if they have loyalty membership
            if session.lower() == "normal":
                self.price = normal
            else:
                self.price = normal + instructor
        else:
            if session.lower() == "normal":
                discount = (normal * 20) / 100 #gives 20% off the
price since the customer has a loyalty membership
                self.price = normal - discount

            else:
                discount = (normal + instructor * 20) / 100
                self.price = normal + instructor - discount

        print(self.price)

        self.id_generator()
```

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

```
def checkValue(self):

    firstName = self.firstName.get()
    surname = self.surname.get()
    dob = self.dob.get()

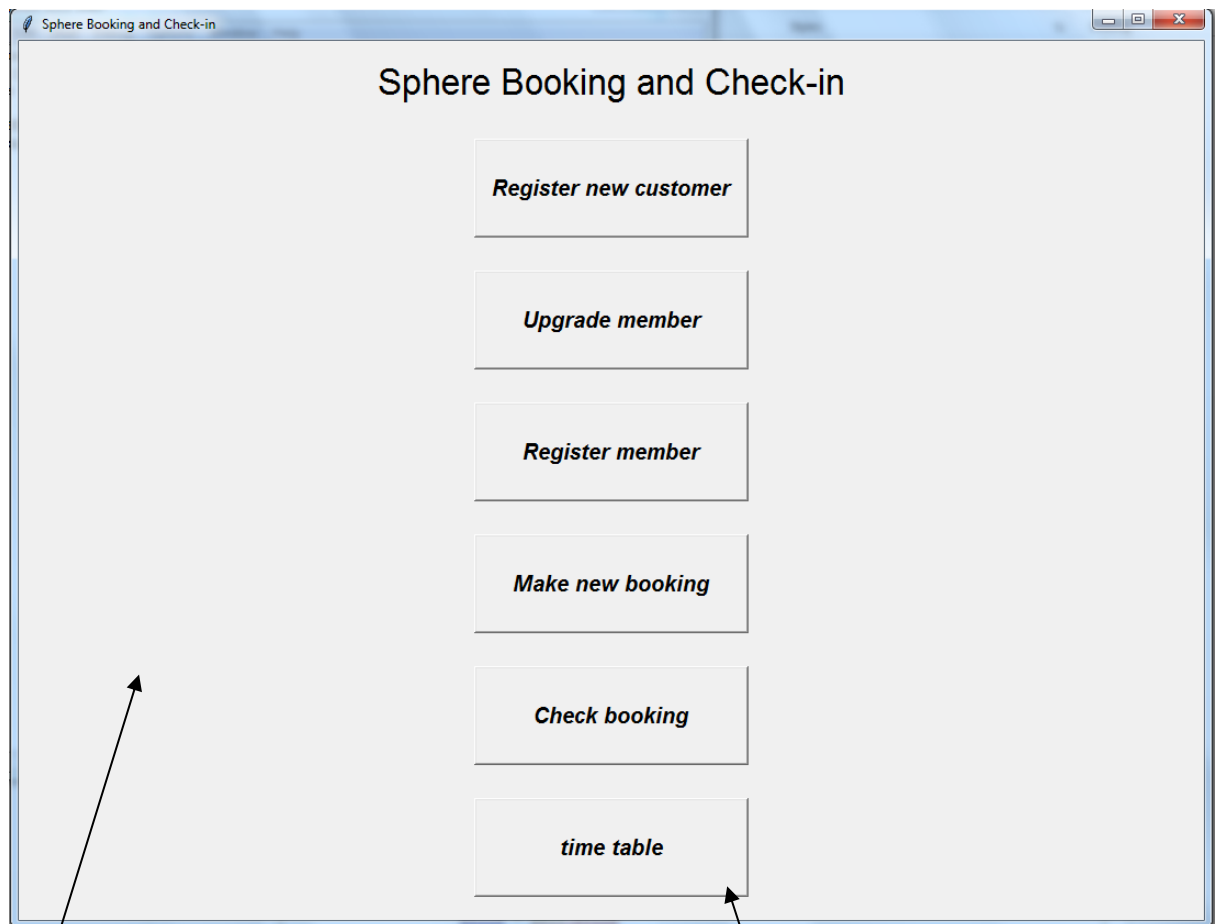
    self.c.execute("SELECT CustomerID FROM Customer WHERE
FirstName = ? AND surname = ? AND date_of_birth = ?", # Get customer
ID from the data base where the customer name, surname and date of
birth matches.
                    (firstName, surname, dob))
    self.ID = self.c.fetchone()
    self.customer = str(self.ID[0])

    self.c.execute("SELECT number_of_sessions FROM Customer WHERE
FirstName = (?) AND surname = (?) AND date_of_birth = (?)", # Get the
amount of sessions the customer has booked from the customer table
                    (firstName, surname, dob))

    session = self.c.fetchone()
    self.sessionNum = int(session[0])
    self.sessionNum +=1
    print(self.sessionNum)

    self.calculate()
```

Prototype 2



Main Menu allows the functionality to be accessed in one location. Everyone's functionality is linked to the buttons which integrates all of our work together.

Each button links to one of the group members' functionality.

Sphere Booking and Check-in

Customer Information

First Name:

Surname:

Date of birth:
Select
Select
Select

Booking Information

Session:

Time:

Date:
Select
Select
Select

Book

Timetable

Home

This is the main user interface for the booking, I've separated the customer information and booking information to make it clear and more structured. As you can see I've also added a drop down select menu which have all been assigned "select" as its first value. This should help keep incorrect data being entered into the database. However, for the date of birth and date I did not add a calendar to select the date this is because it would be extremely time consuming and extremely difficult. But, I've done instead was have 3 different dropdown select menus, the results will then be concatenated and be stored in a different variable to be used later.

Each button serves a different purpose. The booking button will validate all the entered data and will then store the information into the database once it passes the validation checks.

The Timetable shows the timetables of each of the instructors, allowing the slope operator to book at empty slot available.

The Home button will take the slope operator back to the main menu if the customer decided to cancel or the booking has been confirmed.

Sphere Booking and Check-in

Customer Information

First Name:

Surname:

Date of birth:

Booking Information

Session:

Time:

Date:

Book

Timetable

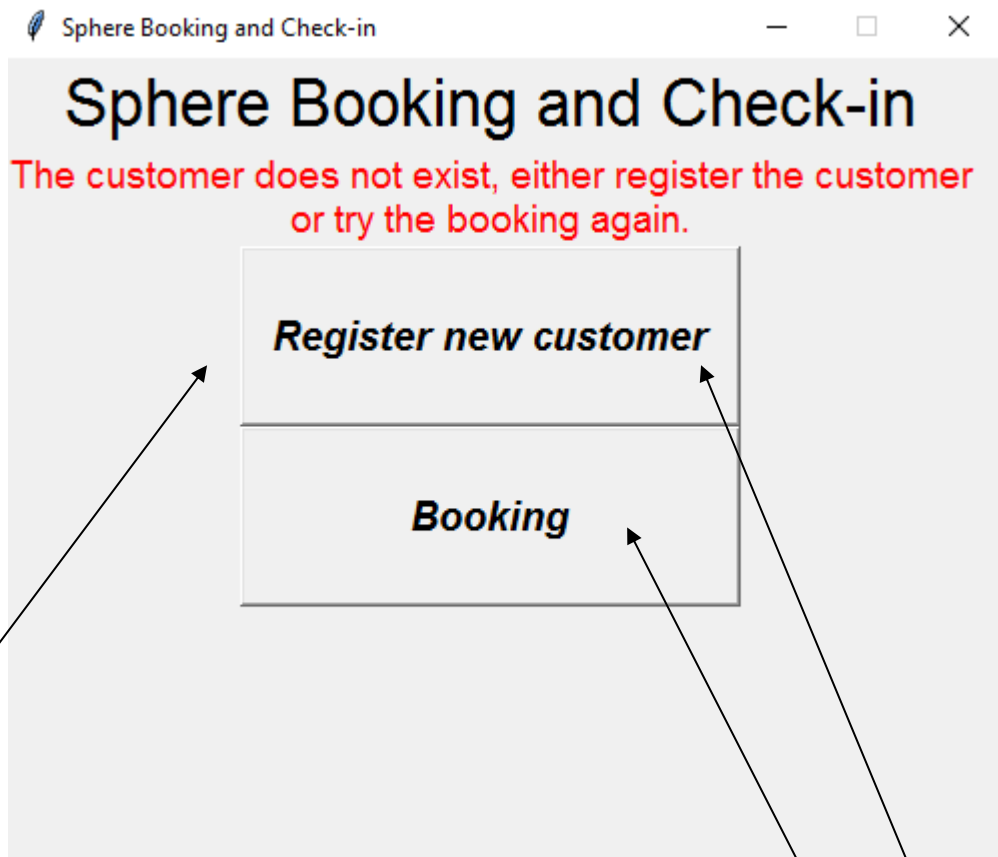
Home

Sphere Booking and Check-in

Date:
(DD/MM/YYYY leave zeros such as 01 as 1)

Staff ID:

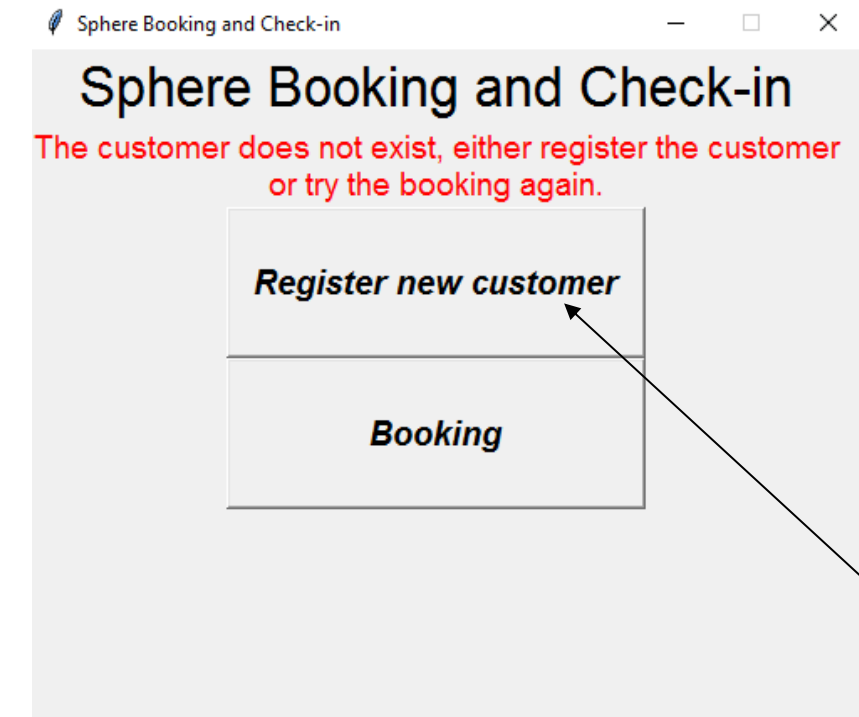
The timetable button is an integration of Sam's functionality. It allows the slope operator to check the timetable of the slope and instructors.



This window will only display if the customer details do not exist within the database. It lets the slope operator know, giving them the option to register the customer or try again.

These two buttons each serve a different purpose. The Register new customer button will redirect the slope operator to the register user interface allowing them to register the new customer.

The booking button will redirect them back to the booking page so that they can try the booking again because maybe they entered the details incorrectly.

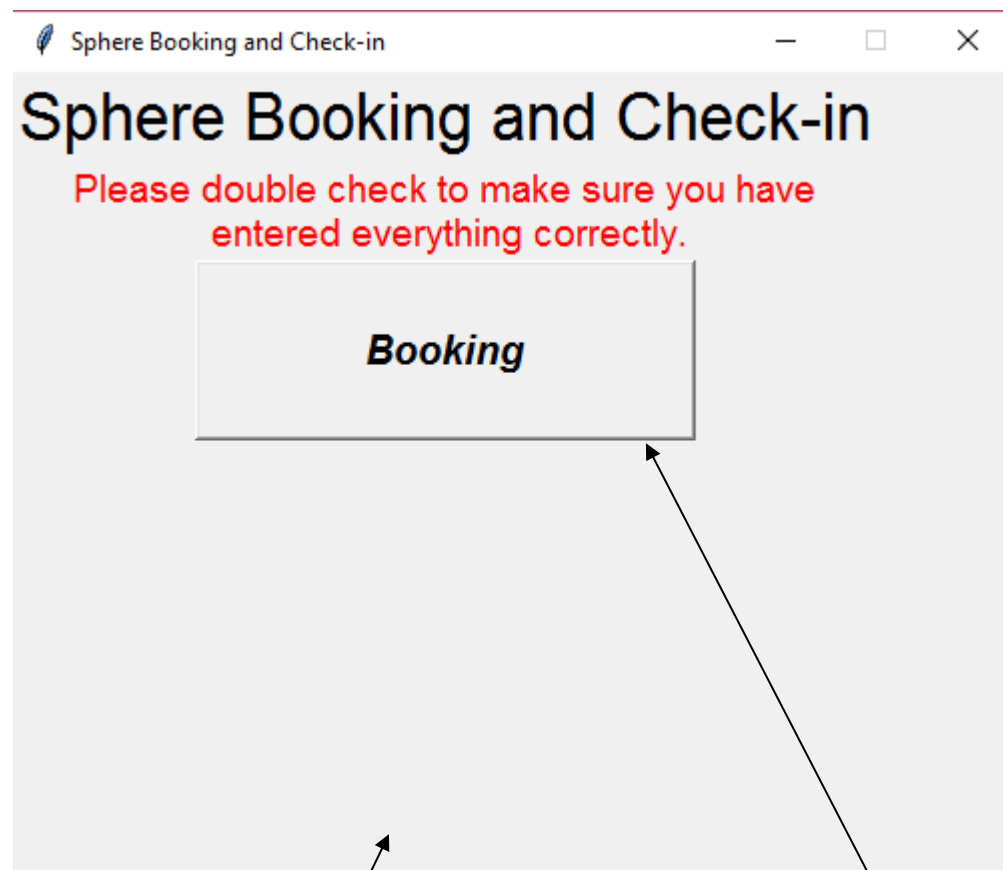


This is another integration created by Adhitama. The slope operator has the option to register a customer if a customer doesn't exist when making a booking. The button on the warning page redirects the slope operator to this window where they can register the customer.

The screenshot shows a window titled "Sphere Registration Table". It contains a form with the following fields and controls:

- First Name: Text input field
- Surname: Text input field
- Date of Birth: Text input field with a date picker icon and the format "DD/MM/YYYY"
- Customer Experience: Two radio buttons labeled "Novice" and "Member"
- Number of Session: Text input field
- Account Balance: Text input field with the value "0"
- Buttons: "Insert Data" and "Quit"

An arrow points from the "Novice" radio button to the same text box on the right.



This windows will only display if the booking details haven't been fully entered. All the field need to be entered before booking can be validated.

The booking button will redirect them back to the booking page so that they can try the booking again because maybe they entered the details incorrectly.

The screenshot displays the 'Sphere Booking and Check-in' application interface. It features two main sections: 'Customer Information' and 'Booking Information'. The 'Customer Information' section includes input fields for 'First Name' (filled with 'salah'), 'Surname' (filled with 'abdo'), and 'Date of birth' (filled with '30/07/1997'). The 'Booking Information' section includes input fields for 'Session' (filled with 'normal'), 'Time' (filled with '14:00'), and 'Date' (filled with '27/11/2024'). Below these sections are three buttons: 'Book', 'Timetable', and 'Home'. A modal window is open in the center, displaying a success message: 'The booking has been successful.' followed by 'Booking reference: XJ7J3X7AXMHYJZAN', 'Date: 2024-11-27', 'Time: 14:00', and 'Price: £40.0'.

Sphere Booking and Check-in

Customer Information

First Name: salah

Surname: abdo

Date of birth: 30/07/1997

Booking Information

Session: normal

Time: 14:00

Date: 27/11/2024

Book

Timetable

Home

Sphere Booking and Check-in

The booking has been successful.

Booking reference: XJ7J3X7AXMHYJZAN

Date: 2024-11-27

Time: 14:00

Price: £40.0

The booking will only confirm once all the validation has been met. It will display a message telling the slope operator the booking has been successful, and it will also display the booking reference number, date, time of the booking and the price.

Salah Abdo
260CT- Software Engineering
Student ID: 6179614
Booking Code2

```
'''  
Sphere Booking and Check-in  
260CT  
prototype  
Salah Abdo  
Python 3
```

This code is for the booking. It allows the slope operator to make a booking by inputting all the correct information required. It will also calculate the price, the price depends of what kind of session the customer book and if they have a membership do they have the correct one to give a discount.

```
'''  
  
from Main import *  
from TimeTable import *  
from home import *  
  
class Booking(object):  
  
    def __init__(self, master, mainwnd):  
  
        self.mainwnd= mainwnd # store the 'self.master` of the main  
window  
        self.master = master  
        self.ref = " "  
        self.price = 0  
        self.date = " "  
        self.time = " "  
  
        self.master.geometry("1080x800")  
        self.master.title("Sphere Booking and Check-in")  
  
        self.conn = sqlite3.connect('Database.db') # connects to the  
database  
        self.c = self.conn.cursor()  
  
        self.time = StringVar(self.master)  
        self.time.set("Select") # initial value  
  
        self.day = StringVar(self.master)  
        self.day.set("Select")  
  
        self.day2 = StringVar(self.master)  
        self.day2.set("Select")  
  
        self.month = StringVar(self.master)  
        self.month.set("Select")  
  
        self.month2 = StringVar(self.master)  
        self.month2.set("Select")
```

```
self.year = StringVar(self.master)
self.year.set("Select")

self.year2 = StringVar(self.master)
self.year2.set("Select")

self.session = StringVar(self.master)
self.session.set("Select")

Label(self.master, text="Sphere Booking and Check-
in", fg="black", font=("Helvetica", 25)).grid(row=0, column=2)

Label(self.master, text="Custromer
Information", fg="black", font=("Helvetica", 11, 'bold')).grid(row=1)
Label(self.master, text="First Name:").grid(row=2)
Label(self.master, text="").grid(row=3)
Label(self.master, text="Surname:").grid(row=4)
Label(self.master, text="").grid(row=5)
Label(self.master, text="Date of birth: ").grid(row=6)
Label(self.master, text="").grid(row=7)

Label(self.master, text="Booking
Information", fg="black", font=("Helvetica", 11, 'bold')).grid(row=1,
column=3)
Label(self.master, text="Session:").grid(row=2, column=3)
Label(self.master, text="").grid(row=3, column=3)
Label(self.master, text="Time: ").grid(row=4, column=3)
Label(self.master, text="").grid(row=5, column=3)
Label(self.master, text="Date:").grid(row=6, column=3)
Label(self.master, text="").grid(row=7, column=3)

self.firstName = Entry(self.master)
self.surname = Entry(self.master)
# stores the entry made into a variable

session = OptionMenu(self.master, self.session, "normal",
"instructor")
time = OptionMenu(self.master, self.time, "8:00", "10:00",
"12:00", "14:00", "16:00", "18:00")
year = OptionMenu(self.master, self.year, "2017", "2018",
"2019", "2020", "2021", "2022", "2023", "2024",
"2025", "2026", "2027", "2028", "2029", "2030", "2031", "2032", "2033", "2034",
"2035")
month = OptionMenu(self.master, self.month, "01", "02", "03",
"04", "05", "06", "07", "08", "09", "10", "11", "12")
day = OptionMenu(self.master, self.day, "1", "2", "3", "4",
"5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16",
"17", "18", "19", "20", "21", "22", "22", "23", "24", "25", "26",
"27", "28", "29", "30", "31")

year2 = OptionMenu(self.master, self.year2, 1917, 1918, 1919,
1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930,
1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941,
1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952,
1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963,
```

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

```
1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974,
1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985,
1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996,
1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007,
2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017)
    month2 = OptionMenu(self.master, self.month2, "01", "02",
"03", "04", "05", "06", "07", "08", "09", "10", "11", "12")
    day2 = OptionMenu(self.master, self.day2, "1", "2", "3", "4",
"5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16",
"17", "18", "19", "20", "21", "22", "22", "23", "24", "25", "26",
"27", "28", "29", "30", "31")
    #drop down select menue

    self.firstName.grid(row=2, column=1)
    self.surname.grid(row=4, column=1)
    day2.grid(row=6, column=1)
    month2.grid(row=7, column=1)
    year2.grid(row=8, column=1)

    session.grid(row=2, column=4)
    day.grid(row=6, column=4)
    time.grid(row=4, column=4)
    month.grid(row=7, column=4)
    year.grid(row=8, column=4)

    # positioning of the entry

    Button(self.master, text='Book', command=self.confirmation,
font=("Helvetica",15, "bold italic")).grid(row=11, column=4,
sticky=W, pady=4)
    Button(self.master, text='Timetable', command=self.table,
font=("Helvetica",15, "bold italic")).grid(row=12, column=4,
sticky=W, pady=4)
    Button(self.master, text='Home', command=self.goHome,
font=("Helvetica",15, "bold italic")).grid(row=13, column=4,
sticky=W, pady=4)

    # padx = horizonta
    # pady = verticle
    # .grid(row=6, column=2)
    # row = increase the number you move down, decrease the
number you move up.
    # column = increase the number you move right, decrease the
number you move left.

def table(self):
    root=Toplevel(self.master)
    timetable=timeTable(root,self.master)
    # Opens the slope or instructor timetable

def goHome(self):
    self.master.destroy() # close the current Member window
    self.mainwnd.update() # update the home window
    self.mainwnd.deiconify() # un-minimize the home window
```

```
def bookingConfirmation(self):
    root1=Toplevel(self.master)

bookingcon=BookingConfirmation(root1,self.master,self.ref,self.date,s
elf.time,self.price)

def closeDB(self): # closes the database
    self.c.close()
    self.conn.close()
    self.bookingConfirmation()

def insertBooking(self):
    refNum = self.ref
    customerID = self.customer
    day = self.day.get()
    year = self.year.get()
    month = self.month.get()
    self.time = self.time.get()
    checking = "False"
    price = self.price
    sessionNumber = self.sessionNum
    session = self.session.get()

    dash = "-"
    self.date = year + dash + month + dash + day #concatenates
day month and year into required format "2017-03-23"

    self.c.execute("INSERT INTO Booking (Ref_number, customerID,
session, date, time, checking, price) VALUES(?, ?, ?, ?, ?, ?, ?)", #
stores all the data into booking table
                    (refNum, customerID, session, self.date,
self.time, checking, price))

    self.c.execute("UPDATE Customer SET number_of_sessions = (?)
WHERE CustomerID = (?) ",
                    (sessionNumber, customerID))
    self.conn.commit()

    self.closeDB()

def id_generator(self,size=16, chars=string.ascii_uppercase +
string.digits): # Creates a random 16 digit value with numbers and
characters, used as reference number
    self.ref=''.join(random.choice(chars) for _ in range(size))
    self.insertBooking()

def calculate(self):
    session = self.session.get()
    cID = self.customer

    normal = 50 # price per hour
    instructor = 20 #price per hour
    self.price = 0
```

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

```
self.c.execute("SELECT count(1) FROM Member WHERE Customer_ID = ?", # selects what membership the customer had, this will be used to calculate the price
               (cID))
membershipCheck = self.c.fetchone()
numberCheck = int(membershipCheck[0])

if numberCheck == 1:
    self.c.execute("SELECT Membership_Type FROM Member WHERE Customer_ID = ?", # selects what membership the customer had, this will be used to calculate the price
                   (cID))

    membershipType = True

else:
    membershipType = False

# If customer has normal membership or no membership then no discount
if membershipType == False: # use the customers membership to give a discount if they have loyalty membership
    if session.lower() == "normal":
        self.price = normal
    else:
        self.price = normal + instructor

# if customer has the correct membership then they get a discount
else:
    if session.lower() == "normal":
        discount = (normal * 20) / 100 #gives 20% off the price since the customer has a loyalty membership
        self.price = normal - discount

    else:
        discount = (normal + instructor * 20) / 100
        self.price = normal + instructor - discount

self.id_generator()

def getUserInfo(self):

    self.c.execute("SELECT CustomerID FROM Customer WHERE FirstName = ? AND surname = ? AND date_of_birth = ?", # Get customer ID from the data base where the customer name, surname and date of birth matches.
                   (self.firstName, self.surname, self.dob))
    self.ID = self.c.fetchone()
    self.customer = str(self.ID[0])

    self.c.execute("SELECT number_of_sessions FROM Customer WHERE FirstName = (?) AND surname = (?) AND date_of_birth = (?)", # Get the amount of sessions the customer has booked from the customer table
                   (self.firstName, self.surname, self.dob))
```

```
        session = self.c.fetchone()
        self.sessionNum = int(session[0])
        self.sessionNum +=1

    def calculate():

    def noCustomer(self):
        root2=Toplevel(self.master)
        self.master.withdraw()
        customer=CustomerNotFound(root2,self.master)

    def wrongEntry(self):
        root3=Toplevel(self.master)
        self.master.withdraw()
        entry=bookingError(root3,self.master)

    def checkEntry(self):
        day = self.day.get()
        y = self.year.get()
        month = self.month.get()
        time = self.time.get()
        session = self.session.get()

        if session != "Select":
            if time != "Select":
                if day != "Select":
                    if month != "Select":
                        if y != "Select":
                            self.getUserInfo()
        else:
            self.wrongEntry()

    def confirmation(self):

        self.firstName = self.firstName.get()
        self.surname = self.surname.get()

        day = self.day2.get()
        y = self.year2.get()
        month = self.month2.get()
        dash = "-"
        year = str(y)

        self.dob = year + dash + month + dash + day #concatenates day
month and year into required format "2017-03-23"

        self.c.execute("SELECT count(1)FROM Customer WHERE FirstName
= (?) AND surname = (?) AND date_of_birth = (?)", # Get the amount of
sessions the customer has booked from the customer table
                        (self.firstName, self.surname, self.dob))

        confirmation = self.c.fetchone()
```

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

```
        checkConf = int(confirmation[0])

        if checkConf == 1:
            self.checkEntry()
        else:
            self.noCustomer()

class BookingConfirmation(object): # only gets called if the booking
validation have been approved

    def __init__(self, master, mainwnd, ref, date, time, price):

        self.mainwnd= mainwnd # store the 'self.master` of the main
window
        self.master = master

        self.bookingRef = ref # variables passed between classes
        self.date = date
        self.time = time
        self.price = price

        self.master.geometry("500x400") # size of the window
        self.master.resizable (0, 0) #disables resizing

        Label(self.master, text="Sphere Booking and Check-
in", fg="black", font=("Helvetica", 25)).grid(row=0, column=2)

        Label(self.master, text="The booking has been
successful.\nBooking reference: " + str(self.bookingRef) + "\nDate: "
+ str(self.date) + "\nTime: " + str(self.time) + "\nPrice: £" +
str(self.price), fg="red", font=("Helvetica", 14)).grid(row=2, column=2)
        Label(self.master, text=" ").grid(row=3, column=2)

class CustomerNotFound(): # only gets called if the customers does
not exist

    def __init__(self, master, mainwnd):

        self.mainwnd= mainwnd # store the 'self.master` of the main
window
        self.master = master

        self.master.geometry("500x400") # size of the window
        self.master.resizable (0, 0) #disables resizing
        self.master.title("Sphere Booking and Check-in")

        Label(self.master, text="Sphere Booking and Check-
in", fg="black", font=("Helvetica", 25)).grid(row=0, column=2)
```


Salah Abdo
260CT- Software Engineering
Student ID: 6179614

```
Label(self.master, text="The customer does not exist, either
register the customer\nor try the booking
again.", fg="red", font=("Helvetica", 14)).grid(row=2, column=2)

self.button1=Button(self.master, text="Register new customer",
command=self.register, fg="black", width=20,
height=3, font=("Helvetica", 15, "bold italic")).grid(row=4, column=2)
self.button1=Button(self.master, text="Booking",
command=self.gotoBooking, fg="black", width=20,
height=3, font=("Helvetica", 15, "bold italic")).grid(row=5, column=2)

def gotoBooking(self):

    root1=Toplevel(self.master)
    self.master.withdraw()
    booking=Booking(root1, self.master)

def register(self): # Runs the register functionality

    self.master.withdraw()
    run()

class bookingError(): # only gets called of the booking information
have not been filled in

def __init__(self, master, mainwnd):

    self.mainwnd= mainwnd # store the 'self.master` of the main
window
    self.master = master

    self.master.geometry("500x400") # size of the window
    self.master.resizable (0, 0) #disables resizing
    self.master.title("Sphere Booking and Check-in")

    Label(self.master, text="Sphere Booking and Check-
in", fg="black", font=("Helvetica", 25)).grid(row=0, column=2)

    Label(self.master, text="Please double check to make sure you
have\n entered everything
correctly.", fg="red", font=("Helvetica", 14)).grid(row=2, column=2)

    self.button1=Button(self.master, text="Booking",
command=self.gotoBooking, fg="black", width=20,
height=3, font=("Helvetica", 15, "bold italic")).grid(row=5, column=2)

def gotoBooking(self):

    root1=Toplevel(self.master)
    self.master.withdraw()
    booking=Booking(root1, self.master)
```

Salah Abdo
260CT- Software Engineering
Student ID: 6179614
Automated unit testing

```
import unittest
import sqlite3

def calculate(session, cID):
    conn = sqlite3.connect('Database.db') # connects to the database
    c = conn.cursor()

    normal = 50 # price per hour
    instructor = 20 #price per hour
    price = 0

    c.execute("SELECT count(1) FROM Member WHERE Customer_ID = ?", #
selects what membership the customer had, this will be used to
calculate the price
              (cID))
    membershipCheck = c.fetchone()
    numberCheck = int(membershipCheck[0])

    if numberCheck == 1:
        c.execute("SELECT Membership_Type FROM Member WHERE
Customer_ID = ?", # selects what membership the customer had, this
will be used to calculate the price
              (cID))

        membershipType = True

    else:
        membershipType = False

    # If customer has normal membership or no membership then no
discount
    if membershipType == False: # use the customers membership to
give a discount if they have loyalty membership
        if session.lower() == "normal":
            price = normal
        else:
            price = normal + instructor

    # if customer has the correct membership then they get a discount
    else:
        if session.lower() == "normal":
            discount = (normal * 20) / 100 #gives 20% off the price
since the customer has a loyalty membership
            price = normal - discount

        else:
            discount = (normal + instructor * 20) / 100
            price = normal + instructor - discount
    return(price)
```

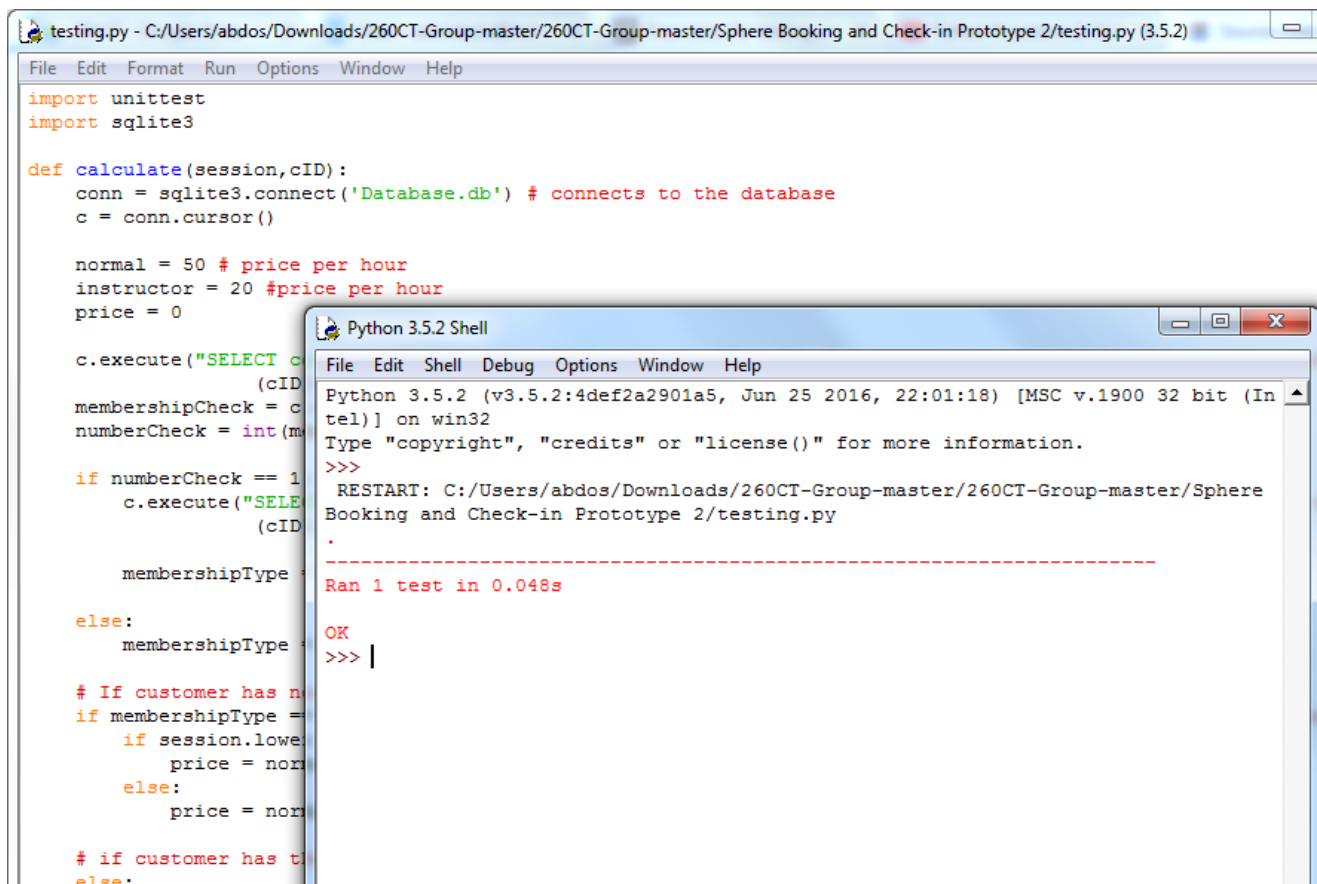
Salah Abdo
260CT- Software Engineering
Student ID: 6179614

```
class MyTest(unittest.TestCase):  
    def test_calculate(self):  
        self.assertEqual( calculate("normal", "6"), 40.0)  
        self.assertEqual( calculate("instructor", "6"), 65.5)  
        self.assertEqual( calculate("normal", "5"), 50)  
        self.assertEqual( calculate("instructor", "5"), 70)
```

```
unittest.main()
```

The automated testing calls the function that you want to test, it gives it input values.

The automated testing also want answer for the return value, this is to make sure that the function is actually working.



The screenshot shows a Python IDE with a file named `testing.py` open. The file contains Python code for testing a price calculation function. The code imports `unittest` and `sqlite3`, defines a `calculate` function that connects to a database and calculates prices based on booking type and membership status, and includes test cases. A `Python 3.5.2 Shell` window is overlaid on the code, showing the execution of a test. The shell output indicates that the test passed successfully, with the message "Ran 1 test in 0.048s".

```
testing.py - C:/Users/abdos/Downloads/260CT-Group-master/260CT-Group-master/Sphere Booking and Check-in Prototype 2/testing.py (3.5.2)
File Edit Format Run Options Window Help

import unittest
import sqlite3

def calculate(session, cID):
    conn = sqlite3.connect('Database.db') # connects to the database
    c = conn.cursor()

    normal = 50 # price per hour
    instructor = 20 #price per hour
    price = 0

    c.execute("SELECT cID, membershipType FROM bookings WHERE cID = ?")
    (cID, membershipType) = c.fetchone()

    membershipCheck = cID == 1
    numberCheck = int(membershipType)

    if numberCheck == 1:
        c.execute("SELECT price FROM bookings WHERE cID = ?")
        (cID, price) = c.fetchone()

        membershipType = membershipType

    else:
        membershipType = membershipType

    # If customer has no membership
    if membershipType == 0:
        if session.lower() == 'normal':
            price = normal
        else:
            price = normal

    # if customer has a membership
    else:
        if session.lower() == 'normal':
            price = normal
        else:
            price = normal

    return price
```

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/abdos/Downloads/260CT-Group-master/260CT-Group-master/Sphere Booking and Check-in Prototype 2/testing.py
.
-----
Ran 1 test in 0.048s
OK
>>> |
```

For this testing, I used python built in automated unit testing called “unittest”. It allows me to write test. For this test, I tested the price calculation, the prices can vary depending on the type of booking that has made and the customer membership status. So, for the test I tested 4 different types of probabilities the price calculation could get. The first one would be a customer with a loyalty membership that has made a normal booking, so the price should be £40 as they get a discount which is correct. The second test would be a customer with a loyalty membership that has made a booking with an instructor, so the price should be £65.50 which is correct. The Thirds test is a customer with no membership that has made normal booking, so the price should be £50 as they get no discount which is correct. The forth test is a customer with no membership that has made a booking with an instructor, so the price should be £70 which is correct.

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

```
class MyTest(unittest.TestCase):  
    def test_calculate(self):  
        self.assertEqual( calculate("normal","6"), 40.0)  
        self.assertEqual( calculate("instructor","6"), 65.5)  
        self.assertEqual( calculate("normal","5"), 50)  
        self.assertEqual( calculate("instructor","5"), 100)  
  
unittest.main()
```

```
testing.py - C:/Users/abdos/Downloads/260CT-Group-master/260CT-Group-master/Sphere Booking and Check-in Prototype 2/testing.py (3.5.2)  
File Edit Format Run Options Window Help  
import unittest  
import sqlite3  
  
def calculate(session,cID):  
    conn = sqlite3.connect('Database.db') # connects to the database  
    c = conn.cursor()  
  
    normal = 50 # price per hour  
    instructor = 20 #price per hour  
    price = 0  
  
    c.execute("SELECT cID, membershipType, numberCheck FROM bookings WHERE cID = ?")  
    (cID, membershipType, numberCheck) = c.fetchone()  
    numberCheck = int(numberCheck)  
  
    if numberCheck == 1:  
        c.execute("SELECT price FROM bookings WHERE cID = ?")  
        (cID, price) = c.fetchone()  
        membershipType = membershipType  
    else:  
        membershipType = membershipType  
  
    # If customer has no booking  
    if membershipType == 'normal':  
        if session.lower() == 'normal':  
            price = normal  
        else:  
            price = normal  
    # if customer has a booking  
    else:  
        if session.lower() == 'normal':  
            discount = 0  
            price = normal  
        else:  
            discount = 0  
            price = normal  
  
    return price  
  
Python 3.5.2 Shell  
File Edit Shell Debug Options Window Help  
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
RESTART: C:/Users/abdos/Downloads/260CT-Group-master/260CT-Group-master/Sphere Booking and Check-in Prototype 2/testing.py  
.br/>-----  
Ran 1 test in 0.048s  
OK  
>>>  
RESTART: C:/Users/abdos/Downloads/260CT-Group-master/260CT-Group-master/Sphere Booking and Check-in Prototype 2/testing.py  
F  
-----  
FAIL: test_calculate (__main__.MyTest)  
-----  
Traceback (most recent call last):  
  File "C:/Users/abdos/Downloads/260CT-Group-master/260CT-Group-master/Sphere Booking and Check-in Prototype 2/testing.py", line 51, in test_calculate  
    self.assertEqual( calculate("instructor","5"), 100)  
AssertionError: 70 != 100  
-----  
Ran 1 test in 0.056s  
FAILED (failures=1)  
>>> |
```

As you can see I purposely changed one of the potential return value. The automated testing is showing me that one of the test was a failure. As you can see the automated testing is saying that the `self.assertEqual(calculate("instructor","5"), 100)` test is incorrect and the answer should be 70 not 100.

Task 2: To evaluate the management of the project – evidence with a report supported by research with references in the CU Harvard referencing style in a maximum of 1,000 words

Evidence of how the method was applied to the project

Gant chart

Week	Group Introduction	Design specification	Implementation	intergration prototype 1	intergration prototype 2	Testing
	Set up GitHub repository, facebook group chat, agree to functionalities	Create 4 layerd model including Grasp and GoF	Implement your design into python	Intergrate everyones work and get feedback of the client	implement improvements that client recommended to prototype 2.	Test 1 function in your code
1: 23/01-27/01	*					
2: 30/01-3/02		*				
3: 6/02-10/02		*				
4: 13/02-17/02			*			
5: 20/02-24/02			*			
6: 27/02-3/03			*			
7: 6/03-10/03				*		
8: 13/03-17/03				*		
9: 20/03-24/03					*	
10: 27/03-31/03						*
11: 3/04-7/04						

Booking System

Meeting Minutes 23/01/2017

Opening

The regular meeting of the Booking System was called to order at 9:00 on 23/01/2017

Present

Salah

Martin

Reece

Sam

Adhitama

Zehua

Discussion

- **Functionalities**
 - Booking session – Salah
 - Time Table – Sam
 - Registering Customer – Adhitama
 - Admin Reece, Adhitama, Martin
 - Viewing session - Zehua
- **Programming language decision**
 - Python 3
- **Data base decision**
 - SQLite3

Agenda for Next Meeting

Adjournment

Meeting was adjourned at 10:00 by Salah the next general meeting will be at 11:00 on February 8, 2017.

Minutes submitted by: Salah

Meeting Minutes
8/02/2017

Opening

The regular meeting of the Group 2: Smart Parking was called to order at 11:00 on 8/02/2017

Present

Salah

Sam

Adhitama

Zehua

Absence

Martin

Reece

Discussion

- **Functionalities Update**
 - Booking functionality is under progression, GUI has been started and will be ready by next meeting.
 - Data Base was set up and working
 - Admin functionality hasn't been started on
 - Registration functionality is under progression, GUI has been started and will be ready by next meeting.
 - Viewing session has been started but no GUI interface has been made.

Agenda for Next Meeting

Functionalities Update

Adjournment

Meeting was adjourned at 12:00 by Salah the next general meeting will be at 9:00 on February 13, 2017.

Minutes submitted by: Salah

Booking System

Meeting Minutes
13/02/2017

Opening

The regular meeting of the Booking System was called to order at 9:00 on 13/02/2017

Present

Salah

Sam

Adhitama

Zehua

Martin

Reece

Discussion

- **Functionalities Update**
 - Booking functionality is under progression, GUI has been fully completed.
Now just waiting for the back-end of the functionality
 - Data Base was set up and working
 - Admin functionality has been completed
 - Registration functionality is under progression, GUI has been fully completed.
Now just waiting for the back-end of the functionality
 - Viewing session is under progression, GUI will be finished by next week
 - Time table functionality is under progression, GUI has been fully completed.
Now just waiting for the back-end of the functionality

Agenda for Next Meeting

Functionalities Update

Adjournment

Meeting was adjourned at 10:00 by Salah the next general meeting will be at 9:00 on February 20, 2017.

Minutes submitted by: Salah

Booking System

Meeting Minutes
20/02/2017

Opening

The regular meeting of the Booking System was called to order at 9:00 on 20/02/2017

Present

Salah

Sam

Adhitama

Zehua

Martin

Reece

Discussion

- **Functionalities Update**
 - Booking functionality has been completed
 - Admin functionality has been completed
 - Registration functionality has been completed
 - Viewing session is under progression, GUI has been fully completed. Now just waiting for the back-end of the functionality
 - Time table functionality has been completed

Agenda for Next Meeting

Functionalities Merging

Adjournment

Meeting was adjourned at 10:00 by Salah the next general meeting will be at 9:00 on February 27, 2017.

Minutes submitted by: Salah

Booking System

Meeting Minutes
27/02/2017

Opening

The regular meeting of the Booking System was called to order at 9:00 on 27/02/2017

Present

Salah

Sam

Adhitama

Zehua

Martin

Reece

Discussion

- **Functionalities Update**
 - Viewing session has been completed
- **Functionalities Merging**
 - Salah will merge functionalities by creating a Main menu and linking all the code together by next week Wednesday.

Agenda for Next Meeting

Functionalities merging update

Adjournment

Meeting was adjourned at 10:00 by Salah the next general meeting will be at 11:00 on March 8, 2017.

Minutes submitted by: Salah

Booking System

Meeting Minutes
08/03/2017

Opening

The regular meeting of the Booking System was called to order at 11:00 on 08/03/2017

Present

Salah

Sam

Adhitama

Zehua

Absencent

Martin

Reece

Discussion

- **Functionalities Merging update**
 - The Merge is complete, the first Prototype for the booking system work and is ready for feedback from the client.
 - The Main Menu is the interface which allow the user to guide through the booking system.

Adjournment

Meeting was adjourned at 11:20 by Salah.

Minutes submitted by: Salah

Feedback

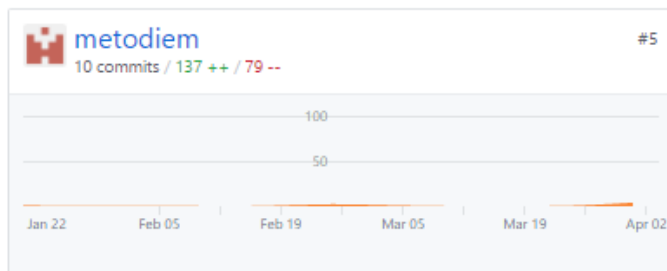
I presented Prototype 1 to the client, the client did other some improvement that I could make to the prototype. These improvements include drop down entry box where required, this includes time, sessions types and date. This will help prevent incorrect data being entered and anomalies acquiring. Another improvement included confirmation page which lets the slope operator now the booking has been successful and displaying information such as booking reference number, date, time of the booking and the session. Data validation was another improvement required for the next prototype, the client wanted some sort of validation letting her now if the customer exist before the booking is processed or if the booking information has been filled in. Lastly, the structure of GUI the client wanted the GUI to be more structured clearly indicating what's the customer information and what's the booking information and instead of the slope operator entering the date or date of birth they can manually select it on a calendar if it's possible.

GitHub

Jan 22, 2017 – Apr 5, 2017

Contributions: Commits ▾

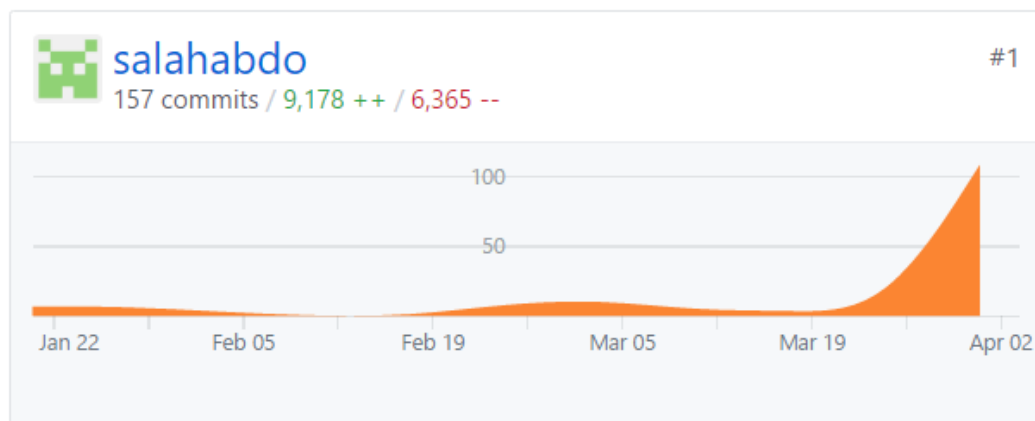
Contributions to master, excluding merge commits



As you can see the group and have regularly committed our work on to GitHub. This graph shows that we have regularly committed from January the 22th to April the 5th. My GitHub name is salahabdo, you can see that have I've made 157 commits since January to April.

GitHub repository: <https://github.com/adhitama19/260CT-Group>

Salah Abdo
260CT- Software Engineering
Student ID: 6179614



260CT Software Engineering group project

242 commits

1 branch

0 releases

6 contributors

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

salahabdo committed on GitHub Update README.md

Latest commit 41b6d30 2 hours ago

Project management	Add files via upload	2 hours ago
Prototype 2	Add files via upload	2 hours ago
Prototype1	Add files via upload	2 hours ago
other	Add files via upload	2 hours ago
README.md	Update README.md	2 hours ago

README.md

260CT-Group

Group Members: • Salah Abdo • Sam Kelly • Zehua Zhou • Martin metodiev • Adhitama Budi • Reece Holness

Functionalities: Booking – Salah, Registration – Adhitama, Viewing Booking – Zehua, Time Table – Sam, Manager deleting – Reece, Manager Adding – Martin, Manager updating - Adhitama

Extra: Membership- Salah, Main Menu - Salah

Case Study – Sphere Booking and Check-in (SBC) System Introduction

The Sphere is a new indoor artificial snow ski-slope in the Midlands. Skiers can book sessions to use the slope. Each day is divided into a number of one-hour sessions. At the moment, the company have a slow inflexible Booking and Check-in System that needs replacing. It is currently considering an electronic booking system to manage their business. The following outlines the required functionality for the Booking and Checkin System.

Requirements for the Bookina and Check-in Svstem Customers can phone in. or come to the front desk to make all session

Salah Abdo
260CT- Software Engineering
Student ID: 6179614
My role in the project

My main role within the project was the booking functionality, I was responsible for creating the design for my functionality and then implementing it into code. I had to make sure that the design met the requirement of the project and kept getting feedback from my group members and also the client. I made sure to get feedback from the client by providing different prototypes.

Reflection:

From my perspective, I found that the project was successful. The main success was due to communication, I tried to communicate with the team as much as possible getting their input and feedback on my designs which allowed me to produce a detailed design specification. I tried to make myself available to help each of my member to maximize the work produced, such as creating the main menu for the booking system which allowed us to integrate our work together

Skills I have gained with in the project would be firstly team work. My team work skill has improved, this will benefit with other project to come and will help during professional practice with in industry, as team work is key within any group project. This improvement was due to my communication within the group chat and input during meeting, I made sure that I was there for my team when need providing input and feedback.

What went wrong with the project was he kept having disagreement with the database, this was because some of the members decided to edit the database without telling the rest of team., this cause problems with other people code. We had to agree with one person being able to edit the database and if you needed to edit the data base you would either notify that member or edit it yourself and say what you have edited.

For my prototyping methodology, I used the prototyping model, this is a system development methods in which prototype is built, tested and then reworked as required until the prototype is acceptable either deemed by the client or its meeting or the specifications (searchcio.techtarget 2005). The prototyping model works best in situations where not all of the project requirements are known, making it trial and error process which takes place between the developer and client in this case. With this project, we were given a case study which gave the requirements of the system. However, it did mention that we could make assumption on the case study where necessary and could also add a feature which was not listed in the requirement. Making the prototyping model perfect for what I'm doing.

The downfall of using the prototype model is that its insufficient analysis. By that, I mean it the focus on a limited prototype can distract the developer from analysing the project properly, which can lead to overseeing potential better solution, preparation of incomplete specification (iotap 2009). Furthermore, this method is limited in functionality it may not scale if the prototype is used as the outcome, this may not be overlooked if the developer is too focused on building a prototype mode. Another downfall of using this method is that the developer can misunderstand the client objectives. This means that the developer can assume that the client shares their objective without thinking about the bigger picture such as commercial issues. (iotap 2009). Lastly, the developer attachment to prototype can also be a downfall. This is because the developer can also become attached to produce porotypes as they would be constantly working on making them spend a great amount of time and effort in producing it. This can lead to lead to many different problems such as attempting to convert a limited prototype into the fished system. This is a problem because it does not have the appropriate underlying architecture (iotap 2009).

However, the benefits of using the prototyping method are that it reduces time and cost. The way it does this is by improving the quality of requirement and specification provided to developers. This can help change the cost by a lot more to integrated as they are noticed later in development, the early determination of what the client wants can result in faster and cheaper system (iotap 2009). Another benefit would be that it improves and increases user involvement. This is because prototyping requires user involvement and allows them to see and interreact with the prototype, this gives them to provide a better feedback and specifications (iotap 2009). The prototype being looked at by the client prevents many confusions/ misunderstanding that occurs when the developer and the client think that they know what they said. Since the client know what they want

Salah Abdo
260CT- Software Engineering
Student ID: 6179614

getting the prototype looked at by the client helps increased interaction which can result in having a better final system and is more likely to satisfy the client.

The comparison between prototyping model and agile scrum is that with prototyping model it's mainly used as a backup or an in addition to a formal functional specification, this model work well with situations where not all the project requirements are known or you have several ideas of how a specific section might work. We use prototyping as a practical means to help develop the system at the same time as allowing the client to try out the development of the system rather than trying to guess the functionality of the specification (impact technology n.d). While the agile method is mainly face to face communication rather than a formal way of communicating like a written document, the agile method usually breaks the project down the project itself into smaller bits which then follows the software development life cycle; planning, analysis, design, implementation, testing & integration, maintenance. The software development life cycle would be used in a formal manner where the working demonstration would be used to review the progress, analyse the effectiveness and re-evaluate the priorities (impact technology n.d). This is because in order to help improve the return on investment for the client.

To conclude the prototyping model allowed me to get the best result possible. This is because this prototyping method work on the situation where not all the project requirement as know making it trial and error process. So, by communicating with the client increased the interaction which resulted in a better final product that has greater tangible and intangible qualities since they are the one that knows what they want. this allowed me to create a final product that is more likely to satisfy them for the look, feel and performance.

References

Report 1:

Black wasp (2008) *Factory Method design pattern*. [Online] Available from
<<http://www.blackwasp.co.uk/FactoryMethod.aspx>> [05/04/2017]

Black wasp (2008) *Prototype design pattern*. [Online] Available from
<<http://www.blackwasp.co.uk/Prototype.aspx>> [05/04/2017]

Black wasp (2008) *Builder design pattern*. [Online] Available from
<<http://www.blackwasp.co.uk/Builder.aspx>> [05/04/2017]

Black wasp (2008) *Abstract Factory design pattern*. [Online] Available from
<<http://www.blackwasp.co.uk/AbstractFactory.aspx>> [05/04/2017]

Black wasp (2008) *Singleton design pattern*. [Online] Available from
<<http://www.blackwasp.co.uk/Singleton.aspx>> [05/04/2017]

Black wasp (2009) *Gang of four design pattern*. [Online] Available from
<<http://www.blackwasp.co.uk/gofpatterns.aspx>> [05/04/2017]

JoshBerke. (2008) 'When would you use the Builder Pattern'. [30/11/2008]
stack overflow [online]. available from
<<http://stackoverflow.com/questions/328496/when-would-you-use-the-builder-pattern>> [05/04/2017]

dbrumann. (2012) 'what's the advantage of factory pattern?'. [4/02/2012] stack
overflow [online]. available from <
<http://stackoverflow.com/questions/9141178/whats-the-advantage-of-factory-pattern>> [05/04/2017]

Danya Rao. (n.d) '*GRASP design Principles*.' [online] 3. available from
<<https://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/rao.pdf>> [05/04/2017]

Report 2:

iotap (2009) *advantages & disadvantages of prototyping process model*. [Online]
Available from < <http://iotap.com/blog/entryid/124/advantages-disadvantage-of-prototyping-process-model> > [05/04/2017]

searchcio.techtarget (2005) *Prototyping Model* . [Online] Available from
<<http://searchcio.techtarget.com/definition/Prototyping-Model>> [05/04/2017]

impact technology (2005) *software prototyping and agile development*. [Online]
Available from <<http://www.impacttechnology.co.uk/software-consultancy/prototyping-agile-dev>> [05/04/2017]