A logo of a planet with yellow rings around it

Description automatically generated

**GROUP ASSIGNMENT**

**PYTHON PROGRAMMING**

 HALL BOOKING SYSTEM DOCUMENTATION

|  |  |
| --- | --- |
| **Group Members** | **TP Number** |
| DEMETRIUS KOH ZIHIN | TP069529 |
| ROOSHA FATIMAH JOOMUN | TP074366 |
| SHRINIVAS HOH | TP071957 |
| AMAL DJASREEL BIN AMRIL | TP072135 |

**Lecturer:** MUHAMMAD HUZAIFAH BIN ISMAIL

**Module:** PYTHON PROGRAMMING

**Module Code:** CT108-3-1-PYP

**Intake Code:** APD1F2309CS(DF), APU1F2309CS(AI), APD1F2309CS(AI)

**Hand Out Date:** 31st October 2023

**Hand In Date:** 5th January 2024

Table of Contents

[**1.0 INTRODUCTION AND ASSUMPTIONS** 4](#_Toc155389735)

[**1.1 Program Introduction** 4](#_Toc155389736)

[**1.2 Program Assumptions** 5](#_Toc155389737)

[1.2.1 User Interface 5](#_Toc155389738)

[1.2.2 Admin Interface 5](#_Toc155389739)

[**2.0 PROGRAM DESIGN** 6](#_Toc155389740)

[**2.1 Main Pseudocode** 6](#_Toc155389741)

[**2.2 Pseudocode for user interface** 10](#_Toc155389742)

[**2.3 Pseudocode for admin interface** 27](#_Toc155389743)

[**3.0 SOURCE CODE AND PROGRAM BREAKDOWN** 52](#_Toc155389744)

[**3.1 User Source Code** 52](#_Toc155389745)

[3.1.1 Line 1-3 52](#_Toc155389746)

[3.1.2 Line 4-5 52](#_Toc155389747)

[3.1.3 Line 8-48 52](#_Toc155389748)

[3.1.4 Line 51-63 53](#_Toc155389749)

[3.1.5 Line 66-71 53](#_Toc155389750)

[3.1.6 Line 74-77 53](#_Toc155389751)

[3.1.7 Line 80-82 53](#_Toc155389752)

[3.1.8 Line 85-87 53](#_Toc155389753)

[3.1.9 Line 90-130 54](#_Toc155389754)

[3.1.10 Line 133-205 55](#_Toc155389755)

[3.1.11 Line 208-225 56](#_Toc155389756)

[3.1.12 Line 228-259 56](#_Toc155389757)

[3.1.13 Line 262-360 57](#_Toc155389758)

[3.1.14 Line 363-380 58](#_Toc155389759)

[3.1.15 Line 383-412 59](#_Toc155389760)

[3.1.16 Line 415-506 59](#_Toc155389761)

[3.1.17 Line 509-527 61](#_Toc155389762)

[3.1.18 Line 530-552 61](#_Toc155389763)

[3.1.19 Line 555-666 62](#_Toc155389764)

[3.1.20 Line 669-721 64](#_Toc155389765)

[**3.2 Admin Source Code** 65](#_Toc155389766)

[3.2.1 Line 724-738 65](#_Toc155389767)

[3.2.2 Line 741-760 65](#_Toc155389768)

[3.2.3 Line 763-784 66](#_Toc155389769)

[3.2.4 Line 787-868 67](#_Toc155389770)

[3.2.5 Line 871-909 69](#_Toc155389771)

[3.2.6 Line 912-933 70](#_Toc155389772)

[3.2.7 Line 936-986 71](#_Toc155389773)

[3.2.8 Line 989-1018 72](#_Toc155389774)

[3.2.9 Line 1021-1151 74](#_Toc155389775)

[3.2.10 Line 1154-1178 77](#_Toc155389776)

[3.2.11 Line 1181-1199 78](#_Toc155389777)

[3.2.12 Line 1202-1230 79](#_Toc155389778)

[3.2.13 Line 1233-1350 80](#_Toc155389779)

[3.2.14 Line 1353-1421 83](#_Toc155389780)

[3.2.15 Line 1424 84](#_Toc155389781)

[**4.0 SCREENSHOTS OF SAMPLE INPUT/OUTPUT AND EXPLANATION** 85](#_Toc155389782)

[**5.1 Sign up as User** 85](#_Toc155389783)

[**5.2 Login as User** 86](#_Toc155389784)

[**5.3 Login as Admin** 94](#_Toc155389785)

[**5.0 CONCLUSION** 103](#_Toc155389786)

[**6.0 REFERENCES** 104](#_Toc155389787)

[**7.0 Workload Matrix** 105](#_Toc155389788)

# **1.0 INTRODUCTION AND ASSUMPTIONS**

## **1.1 Program Introduction**

We were tasked with programming a fully functional booking hall system. The system consists of 2 parts, the administrator interface, and the user interface.

The program begins in a menu interface, to access the administrator interface, admins are to provide a username and password that the system recognizes belong to an admin. After providing the required credentials, admins can access the following options;

1. **Performs Hall Management:** Admins can create, view, search, edit and delete hall information.
2. **Performs Booking Management:** Admins can view, search, edit and delete/cancel booking information.
3. **Performs User Management:** Admins can view, search, edit and delete/block user information.
4. **Logout From System:** Admins can logout of system and return to the menu interface.

On the other hand, the user interface can be accessed by entering username and password credentials that the system recognizes as a user. However, before that, users are required to sign up by providing the following information:

* Username
* Password
* First Name
* Last Name
* Date of Birth
* Contact Number
* Email Address

Once user signs up, users can log in by providing the required credentials and access the following features:

1. **Perform Booking:** Users can create a booking by providing following information;
   1. Event Name
   2. Event Description
   3. Hall ID
   4. Pax
   5. Date and Time
   6. Payment Price
2. **View Bookings**
3. **Delete/Cancel Bookings**
4. **Edit Booking Information**
5. **Search Booking Information**
6. **Update Profile Information**
7. **Logout the System:** User will return to the menu interface

Data such as Username, Password, Halls, Bookings etc. Will be stored in text files.

## **1.2 Program Assumptions**

As the system consists of 2 parts, we decided to split the tasks where 2 people would program the admin interface and another 2 would program the user interface. The user interface is then divided into 2 further subparts user sign in and user log in.

### 1.2.1 User Interface

Under the user sign in section, users must provide data regarding the individual, hence proper validation is necessary for password, date of birth, email address etc. Moreover, data collected should be stored in a database to be used later, in this project we are required to use txt file as a means for this.

Besides that, as usernames are necessary for the system to identify the user and provide access to information only regarding the user, the username should be unique for everyone. Thus, a validation process for comparing usernames with ones already stored in database should be made.

Once a user profile is created, the user may log in via the menu interface using the username and password they registered with. Any previous bookings made by the user can be accessed via the view option whereby the program would compile the relevant information and display it to the user in a readable format.

* The information provided by the user is assumed to be accurate and valid.
* The phone number provided matches the geographic location of the user.
* The user is booking the hall for himself and not for another person.
* Once a booking has been saved, it is assumed that the user has paid as per stated in the booking information.

### 1.2.2 Admin Interface

Under the admin section of the code, after logging in with the credentials recognized by the system, the admin can perform several exclusive functions and has access to all records stored for this system. As there is no restriction as to what the admin is allowed to do, here are a few assumptions we made as to regarding why the admin will choose an option proposed by our menu-driven code.

* Assume a hall will be available after an event and the admin will update its availability status.
* The user will be updated on changes made by admin to the details of a hall they have booked.
* The user will know if the admin has edited or deleted their booking information.
* The user will know if the admin edits or deletes their profile information.

# **2.0 PROGRAM DESIGN**

## **2.1 Main Pseudocode**

START

FROM datetime IMPORT datetime

IMPORT re

IMPORT json

CONSTANT ADMIN\_USERNAME = "Banana"

CONSTANT ADMIN\_PASSWORD = "B4n4n4"

TRY

FILEOPEN “username\_database.txt” FOR READ AS f

username\_database = json.Load(f)

EXCEPT FileNotFoundError

PRINT “Could not run code as the file “username\_database.txt” could not be found!”

exit()

END TRY

TRY

FILEOPEN “user\_info\_database.txt” FOR READ AS f

user\_info\_database.txt = json.Load(f)

EXCEPT FileNotFoundError

PRINT “Could not run code as the file “user\_info\_database.txt” could not be found!”

exit()

END TRY

TRY

FILEOPEN “available\_hall\_ids.txt” FOR READ AS f

available\_hall\_ids = json.Load(f)

EXCEPT FileNotFoundError

PRINT “Could not run code as the file “available\_hall\_ids.txt” could not be found!”

exit()

END TRY

TRY

FILEOPEN “available\_halls.txt” FOR READ AS f

available\_halls = json.Load(f)

EXCEPT FileNotFoundError

PRINT “Could not run code as the file “available\_halls.txt” could not be found!”

exit()

END TRY

TRY

FILEOPEN “booked\_hall\_ids.txt” FOR READ AS f

booked\_hall\_ids = json.Load(f)

EXCEPT FileNotFoundError

PRINT “Could not run code as the file “booked\_hall\_ids.txt” could not be found!”

exit()

END TRY

TRY

FILEOPEN “booked\_halls.txt” FOR READ AS f

“booked\_halls = json.Load(f)

EXCEPT FileNotFoundError

PRINT “Could not run code as the file ““booked\_halls.txt” could not be found!”

exit()

END TRY

FUNCTION Valid Password

lower, upper, number = 0, 0, 0

for c in \_password:

if 97 <= ord(c) <= 122:

lower += 1

if 65 <= ord(c) <= 90:

upper += 1

if 48 <= ord(c) <= 57:

number += 1

IF lower < 2 or upper < 2 or number < 2 or len(\_password) < 8:

RETURN False

ELSE:

RETURN True

FUNCTION validdateformat

REPEAT

SET datetime

UNTIL ValueError

RETURN False

FUNCTION Time

pattern = re.compile("^(0[1-9]|1[0-2]):[0-5][0-9] [APap][Mm]$")

IF Pattern is unmatched related to Time

RETURN False

FUNCTION Valid Email

format = r'\b[A-Za-z0-9.%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

RETURN bool(re.fullmatch(format, \_email))

FUNCTION Valid Contact Number

pattern = re.compile(r'^[+\d-]+$')

RETURN bool(re.fullmatch(pattern, phone\_number))

FUNCTION main

DISPLAY "==Welcome to Banana Halls! What would you like to do?=="

DISPLAY "(1) Sign up as User"

DISPLAY "(2) Login as User"

DISPLAY "(3) Login as Admin"

DISPLAY "(4) Quit"

INPUT number

IF option not in "1234" OR Len(option) != 1

DISPLAY "Please enter a valid option!"

ELSE IF option == "1"

ELSE IF option == "2"

ELSE IF option == "3"

ELSE

DISPLAY "Goodbye!"

END

## **2.2 Pseudocode for user interface**

FUNCTION Signup

DISPLAY "==New account creation=="

While True

INPUT Username

IF Username < 1

Return to main page

ELSE IF username in username\_database

DISPLAY "That username already is already taken! Please enter a different username"

ELSE

BREAK

DISPLAY "Your password should contain at least 8 characters."

DISPLAY "Your password should also include at least of 2 numbers, 2 uppercase and 2 lowercase characters."

INPUT Password

While Password is not valid

DISPLAY "Your password is not strong enough! Please follow the format above."

INPUT "Please re-enter your password: "

While True

INPUT first name

IF first name < 1

PRINT "Please don't leave this empty!"

ELSE

Break

While True

INPUT last name

IF last name < 1

PRINT "Please don't leave this empty!"

ELSE

Break

While True

INPUT date of birth

IF date of birth not valid

PRINT "Invalid date of birth! Please re-enter."

ELSE

Break

While True

INPUT contact number

IF len(contact\_number) < 1

DISPLAY "Please don't leave this empty!"

ELSE IF contact number not valid

DISPLAY "Invalid contact number! Please re-enter."

ELSE

BREAK

While True

INPUT Email

IF len(email) < 1

DISPLAY "Please don't leave this empty!"

ELSE IF Email not valid

DISPLAY " "Invalid email address! Please re-enter."

ELSE

BREAK

DISPLAY "New user successfully created! Please login from the main page."

DISPLAY "Returning to main page..."

Return To main()

FUNCTION Login

DISPLAY "==Login to your account=="

INPUT Username

IF Username < 1

Return to Main ()

ELSE IF Username not in Database

DISPLAY f"That user does not exist!"

SET User Index

INPUT password

IF password not equivalent to the user\_info\_database

DISPLAY "Incorrect password!"

ELSE

Return to User Interface

FUNCTION User Interface

DISPLAY f"==Welcome, {user\_info\_database[user\_index]}! What would you like to do?=="

DISPLAY "(1) Make a new booking"

DISPLAY "(2) View all halls"

DISPLAY "(3) View/Edit/Delete bookings"

DISPLAY "(4) View/Edit account into"

DISPLAY "(5) Delete account"

DISPLAY "(6) Logout"

While True

INPUT Number between 1-6

IF option not in "123456" or len(option) != 1

DISPLAY "Please enter a valid option!"

ELSE IF option == "1"

Redirect to new\_booking(user\_index)

ELSE IF option == "2"

Redirect to view\_all\_halls(user\_index)

ELSE IF option == "3"

Redirect to view\_edit\_delete\_bookings(user\_index)

ELSE IF option == "4"

Redirect to edit\_account\_info(user\_index)

ELSE IF option == "5"

REPEAT

IF type(user\_info\_database[user\_index + 7]) is dict

DISPLAY “You cannot delete your account with halls booked! Please delete your bookings before deleting your account”

ELSE

delete\_account(user\_index)

UNTIL IndexError

delete\_account(user\_index)

ELSE

DISPLAY f"See you again, {user\_info\_database[user\_index]}!"

RETURN to Main()

FUNCTION Edit Account Information

DISPLAY "==View/Edit Account Info=="

DISPLAY "Current Info"

DISPLAY f"Username: {user\_info\_database[user\_index]}"

DISPLAY f"Password: {user\_info\_database[user\_index + 1]}"

DISPLAY f"First name: {user\_info\_database[user\_index + 2]}"

DISPLAY f"Last name: {user\_info\_database[user\_index + 3]}"

DISPLAY f"Date of birth: {user\_info\_database[user\_index + 4]}"

DISPLAY f"Contact number: {user\_info\_database[user\_index + 5]}"

DISPLAY f"Email: {user\_info\_database[user\_index + 6]}"

INPUT Response to edit account information

IF len(response) != 0

user\_interface(user\_index)

RETURN

ELSE

PASS

username\_database.remove(user\_info\_database[user\_index])

While True

INPUT new Username

IF new Username < 1

username = user\_info\_database[user\_index]

break

ELSE IF username in username\_database

DISPLAY "That username already is already taken! Please enter a different username"

ELSE

Break

username\_database.append(username)

DISPLAY "Your password should contain at least 8 characters."  
DISPLAY "Your password should also include at least of 2 numbers, 2 uppercase and 2 lowercase characters."

WHILE True

INPUT New Password

IF len(password) < 1

password = user\_info\_database[user\_index + 1]

ELSE IF

Not a Valid Password

DISPLAY "Your password is not strong enough! Please follow the format above."

INPUT Re-Enter Password

valid\_password(password)

ELSE

BREAK

INPUT New First Name

IF len(first\_name) < 1

first\_name = user\_info\_database[user\_index + 2]

INPUT New Last Name

IF len(last\_name) < 1

last\_name = user\_info\_database[user\_index + 3]

while True:

INPUT New Date Of Birth

IF len(date\_of\_birth) < 1

date\_of\_birth = user\_info\_database[user\_index + 4]

break

ELSE IF Date of Birth not valid

DISPLAY "Invalid date of birth! Please re-enter."

ELSE

Break

while True:

INPUT New Contact Number

IF len(contact\_number) < 1

contact\_number = user\_info\_database[user\_index + 5]

Break

ELSE IF Contact Number Invalid

DISPLAY "Invalid contact number! Please re-enter."

INPUT New Contact Number

Valid\_contact( New Contact Number)

ELSE:

Break

while True

INPUT New Email

IF len(email) < 1

email = user\_info\_database[user\_index + 6]

Break

ELSE IF Email Invalid

DISPLAY "Invalid email address! Please re-enter."

ELSE

Break

DISPLAY "Account successfully updated! Here's your latest account info"

DISPLAY f"Username: {user\_info\_database[user\_index]}"

DISPLAY f"Password: {user\_info\_database[user\_index + 1]}"

DISPLAY f"First name: {user\_info\_database[user\_index + 2]}"

DISPLAY f"Last name: {user\_info\_database[user\_index + 3]}"

DISPLAY f"Date of birth: {user\_info\_database[user\_index + 4]}"

DISPLAY f"Contact number: {user\_info\_database[user\_index + 5]}"

DISPLAY f"Email: {user\_info\_database[user\_index + 6]}"

DISPLAY "Returning to user interface..."

user\_interface(user\_index)

FUNCTION Delete User Account

while True:

INPUT User deleting their account or maybe not

IF Answer == "y"

username\_database.remove(user\_info\_database[user\_index])

SET i = 0

WHILE i < 7

user\_info\_database.remove(user\_info\_database[user\_index])

INCREMENT i

DISPLAY "Your account has been deleted successfully. Sorry to see you go!"

Return to main ()

Break

ELSE IF answer == "n":

DISPLAY “Returning to user interface…”

user\_interface(user\_index)

Break

ELSE:

DISPLAY "Please provide a valid response!"

FUNCTION Viewing All Halls

DISPLAY "== Viewing hall information =="

DISPLAY f"Available halls (by ID): {available\_hall\_ids}"

DISPLAY f"Booked halls (by ID): {booked\_hall\_ids}"

WHILE True:

INPUT Hall ID for the user to view

IF len(hall\_id) < 1:

user\_interface(user\_index)

RETURN

ELSE IF hall\_id in available\_hall\_ids:

hall\_index = available\_halls.index(hall\_id)

DISPLAY f"Displaying hall info of {hall\_id}"

DISPLAY f"ID: {available\_halls[hall\_index]}"

DISPLAY f"Name: {available\_halls[hall\_index + 1]}"

DISPLAY f"Description: {available\_halls[hall\_index + 2]}"

DISPLAY f"Hall pax: {available\_halls[hall\_index + 3]}"

DISPLAY f"Status: Available"

DISPLAY f"Rate price per day: ${available\_halls[hall\_index + 4]}"

ELSE IF hall\_id in booked\_hall\_ids:

hall\_index = booked\_halls.index(hall\_id)

DISPLAY f"Displaying hall info of {hall\_id}"

DISPLAY f"ID: {booked\_halls[hall\_index]}"

DISPLAY f"Name: {booked\_halls[hall\_index + 1]}"

DISPLAY f"Description: {booked\_halls[hall\_index + 2]}

DISPLAY f"Hall pax: {booked\_halls[hall\_index + 3]}"

DISPLAY f"Status: Booked"

DISPLAY f"Rate price per day: ${booked\_halls[hall\_index + 4]}"

ELSE

DISPLAY f"That hall does not exist!"

FUNCTION New Booking

DISPLAY "==Making a new booking=="

DISPLAY f"Available halls (by ID): {available\_hall\_ids}"

WHILE True:

INPUT Hall ID Information

IF len(hall\_id) < 1

user\_interface(user\_index)

exit ()

ELSE IF hall\_id in available\_hall\_ids

BREAK

ELSE

DISPLAY "That hall ID does not exist!"

DISPLAY "Hall Info"

DISPLAY f"ID: {available\_halls[hall\_index]}"

DISPLAY f"Name: {available\_halls[hall\_index + 1]}"

DISPLAY f"Description: {available\_halls[hall\_index + 2]}"

DISPLAY f"Hall pax: {available\_halls[hall\_index + 3]}"

DISPLAY f"Status: Available"

DISPLAY f"Rate price per day: ${available\_halls[hall\_index + 4]}"

INPUT To Proceed with The Booking

IF len(response) != 0

new\_booking(user\_index)

RETURN

WHILE True

INPUT Event Name

IF len(event\_name) < 1

DISPLAY "This cannot be empty!"

ELSE

Break

WHILE True

INPUT Event Description

IF len(event\_description) < 1

DISPLAY "This cannot be empty!"

ELSE

Break

WHILE True:

INPUT New Number of Pax

IF number\_of\_pax.isdigit()

IF int(number\_of\_pax) > int(available\_halls[hall\_index + 3]):

DISPLAY f"That amount is too big! Maximum amount of pax is {available\_halls[hall\_index + 3]}"

ELSE IF

Break

ELSE

DISPLAY "Please enter a valid number!"

WHILE True:

INPUT Date Of Rental

IF Date Format Invalid

DISPLAY "Invalid date! Please re-enter."

ELSE:

Break

WHILE True:

INPUT Time Of Rental

IF valtime(time) == False

DISPLAY "Invalid time or incorrect format! Please re-enter"

ELSE

BREAK

WHILE True:

INPUT Payment Price

IF payment\_price.isdigit()

IF payment\_price < available\_halls[hall\_index + 4]

DISPLAY f"That is not enough! The rate price per day is ${available\_halls[hall\_index + 4]}!"

ELSE IF:

BREAK

ELSE:

DISPLAY "Please enter a valid number!"

available\_hall\_ids.remove(available\_halls[hall\_index])

i = 0

WHILE i < 5:

available\_halls.remove(available\_halls[hall\_index])

i += 1

DISPLAY "Hall successfully booked!"

DISPLAY "Returning to user interface..."

user\_interface(user\_index)

FUNCTION view/edit/delete bookings

DISPLAY "==View/Edit/Delete Bookings=="

DISPLAY "(1) View all bookings"

DISPLAY "(2) Edit a booking"

DISPLAY "(3) Delete a booking"

DISPLAY "(4) Go back"

WHILE True

INPUT type a number between 1-6 only

IF option not in "1234" or len(option) != 1

DISPLAY "Please enter a valid option!"

ELSE IF option == "1"

view\_bookings(user\_index)

ELSE IF option == "2"

edit\_a\_booking(user\_index)

ELSE IF option == "3"

delete\_a\_booking(user\_index)

ELSE:

user\_interface(user\_index)

FUNCTION View Bookings

DISPLAY "==Viewing all bookings=="

DISPLAY "Here is the list of your bookings"

DISPLAY "(If nothing shows up, that means you haven't made any bookings yet)"

SET x = 1

WHILE True:

REPEAT

IF type(user\_info\_database[user\_index + 6 + x]) is in Dictionary

DISPLAY f"Booking {x}"

DISPLAY f'Hall ID: {user\_info\_database[user\_index + 6 + x]["hall\_id"]}'

DISPLAY f'Event Name: {user\_info\_database[user\_index + 6 + x]["event\_name"]}'

DISPLAY f'Event Description: {user\_info\_database[user\_index + 6 + x]["event\_description"]}'

DISPLAY f'Number of Pax: {user\_info\_database[user\_index + 6 + x]["number\_of\_pax"]}'

DISPLAY f'Date: {user\_info\_database[user\_index + 6 + x]["date"]}'

DISPLAY f'Time: {user\_info\_database[user\_index + 6 + x]["time"]}'

DISPLAY f'Payment Price: ${user\_info\_database[user\_index + 6 + x]["payment\_price"]}'

DISPLAY ""

INCREMENT x

Else:

BREAK

UNTIL IndexError:

BREAK

view\_edit\_delete\_bookings(user\_index)

FUNCTION Editing the booking hall

DISPLAY "==Edit a booking=="

DISPLAY "Here is the list of your bookings by your event name"

DISPLAY "If nothing shows up, that means you haven't made any bookings yet"

SET x = 1

While True:

REPEAT

IF type(user\_info\_database[user\_index + 6 + x]) is in Dictionary

DISPLAY f"Booking {x}"

DISPLAY f'Event Name: {user\_info\_database[user\_index + 6 + x]["event\_name"]}'

DISPLAY ""

INCREMENT x

Else:

Break

UNTIL IndexError

Break

while True:

INPUT Booking Number to Edit

IF len(option) == 0:

view\_edit\_delete\_bookings(user\_index)

return

ELSE IF option.isdigit()

REPEAT

if type(user\_info\_database[user\_index + 6 + int(option)]) is in Dictionary

BREAK

ELSE

DISPLAY"Please select a valid option!"

UNTIL IndexError:

DISPLAY "Please select a valid option!"

ELSE

DISPLAY "Please select a valid option!"

hall\_id = user\_info\_database[user\_index + 6 + int(option)]["hall\_id"]

hall\_index = booked\_halls.index(hall\_id)

DISPLAY f"You are now editing booking {int(option)}"

INPUT New Event Name

IF len(event\_name) < 1

event\_name = user\_info\_database[user\_index + 6 + int(option)]["event\_name"]

INPUT Event Description

IF len(event\_description) < 1

event\_description = user\_info\_database[user\_index + 6 + int(option)]["event\_description"]

WHILE True:

INPUT New Number Of Pax

IF len(number\_of\_pax) < 1

number\_of\_pax = user\_info\_database[user\_index + 6 + int(option)]["number\_of\_pax"]

BREAK

ELSE IF number\_of\_pax.isdigit():

IF int(number\_of\_pax) > int(booked\_halls[hall\_index + 3]):

DISPLAY f"That amount is too big! Maximum amount of pax is {booked\_halls[hall\_index + 3]}"

ELSE:

BREAK

ELSE:

DISPLAY "Please enter a valid number!"

WHILE True:

INPUT New Date Of Rental

IF len(date) < 1:

date = user\_info\_database[user\_index + 6 + int(option)]["date"]

Break

ELSE IF Date Format Invalid

DISPLAY "Invalid date! Please re-enter."

ELSE

Break

WHILE True:

INPUT New Time of Rental

IF len(time) < 1:

time = user\_info\_database[user\_index + 6 + int(option)]["time"]

BREAK

ELSE IF valtime(time) == False:

DISPLAY "Invalid time or incorrect format! Please re-enter"

ELSE

BREAK

WHILE True:

INPUT New Payment Price ($)

IF len(payment\_price) < 1:

payment\_price = user\_info\_database[user\_index + 6 + int(option)]["payment\_price"]

BREAK

ELSE IF payment\_price.isdigit():

IF payment\_price < booked\_halls[hall\_index + 4]:

DISPLAY f"That is not enough! The rate price per day is ${booked\_halls[hall\_index + 4]}!"

ELSE

BREAK

ELSE

DISPLAY "Please enter a valid number!"

DISPLAY "Booking information successfully updated! Here's your latest booking information:"

DISPLAY f"Booking {int(option)}"

DISPLAY f'Hall ID: {user\_info\_database[user\_index + 6 + int(option)]["hall\_id"]}'

DISPLAY f'Event Name: {user\_info\_database[user\_index + 6 + int(option)]["event\_name"]}'

DISPLAY f'Event Description: {user\_info\_database[user\_index + 6 + int(option)]["event\_description"]}'

DISPLAY f'Number of Pax: {user\_info\_database[user\_index + 6 + int(option)]["number\_of\_pax"]}'

DISPLAY f'Date: {user\_info\_database[user\_index + 6 + int(option)]["date"]}'

DISPLAY f'Time: {user\_info\_database[user\_index + 6 + int(option)]["time"]}'

DISPLAY f'Payment Price: ${user\_info\_database[user\_index + 6 + int(option)]["payment\_price"]}'

DISPLAY "Returning to view/edit/delete bookings page..."

view\_edit\_delete\_bookings(user\_index)

FUNCTION Deleting A Booking

DISPLAY "==Delete a booking=="

DISPLAY "Here is the list of your bookings by your event name")

DISPLAY "(If nothing shows up, that means you haven't made any bookings yet)"

SET x = 1

WHILE True:

REPEAT:

IF type(user\_info\_database[user\_index + 6 + x]) is in Dictionary

DISPLAY f"Booking {x}"

DISPLAY f'Event Name: {user\_info\_database[user\_index + 6 + x]["event\_name"]}'

DISPLAY ""

INCREMENT x

Else:

Break

UNTIL IndexError:

Break

WHILE True:

INPUT Deleting a Booking Number

IF len(option) == 0

view\_edit\_delete\_bookings(user\_index)

RETURN

ELSE IF option.isdigit()

REPEAT

IF type(user\_info\_database[user\_index + 6 + int(option)]) is in Dictionary

hall\_id = user\_info\_database[user\_index + 6 + int(option)]["hall\_id"]

hall\_index = booked\_halls.index(hall\_id)

available\_hall\_ids.append(booked\_halls[hall\_index])

available\_halls.append(booked\_halls[hall\_index])

available\_halls.append(booked\_halls[hall\_index + 1])

available\_halls.append(booked\_halls[hall\_index + 2])

available\_halls.append(booked\_halls[hall\_index + 3])

available\_halls.append(booked\_halls[hall\_index + 4])

user\_info\_database.remove(user\_info\_database[user\_index + 6 + int(option)])

booked\_hall\_ids.remove(booked\_halls[hall\_index])

SET i = 0

WHILE i < 5:

booked\_halls.remove(booked\_halls[hall\_index])

INCREMENT i

DISPLAY "Booking successfully deleted!"

DISPLAY "Returning to view/edit/delete bookings page..."

view\_edit\_delete\_bookings(user\_index)

BREAK

ELSE

DISPLAY "Please select a valid option!"

UNTIL IndexError:

DISPLAY "Please select a valid option!"

ELSE

DISPLAY "Please select a valid option!"

## **2.3 Pseudocode for admin interface**

FUNCTION login\_as\_admin()

PRINT “== Logging in as Admin==”

PRINT “Please enter username (Press enter to go back to main page): “

INPUT username

IF length(username) < 1 THEN

CALL main()

ELSE IF username <> ADMIN\_USERNAME THEN

PRINT “Invalid username!”

CALL login\_as\_admin()

ELSE

PRINT “Please enter password: “

IF password <> ADMIN\_PASSWORD THEN

PRINT “Incorrect password!”

CALL login\_as\_admin()

ELSE

CALL admin\_interface

ENDIF

ENDIF

ENDFUNCTION

PROCEDURE admin\_interface()

PRINT “==Welcome, Admin! What would you like to do?==”

PRINT “(1) Manage halls”

PRINT “(2) Manage bookings”

PRINT “(3) Manage users”

PRINT “(4) Logout”

WHILE True:

PRINT “Please pick an option by typing in the number”

INPUT option

IF option NOT IN “1234” OR length(option) <> 1 THEN

PRINT “Please enter a valid option”

ELSE IF option == “1” THEN

CALL hall\_management()

ELSE IF option == "2” THEN

CALL manage\_bookings()

ELSE IF option == "3” THEN

CALL manage\_users()

ELSE

PRINT “See you again, Admin!”

CALL main()

ENDIF

ENDWHILE

ENDPROCEDURE

PROCEDURE manage\_users()

PRINT “==User management==”

PRINT “What would like to do?”

PRINT "(1) List users by username”

PRINT "(2) View/Edit user information”

PRINT "(3) Delete user”

PRINT "(4) Go back”

WHILE TRUE

PRINT “Please pick an option by typing in the number”

INPUT option

IF option NOT IN “1234” OR length(option) <> 1 THEN

PRINT “Please enter a valid option”

ELSE IF option == “1” THEN

PRINT “Here is the list of users by username”

PRINT(username\_database)

CALL manage\_users()

ELSE IF option == "2” THEN

CALL edit\_user\_information()

ELSE IF option == "3” THEN

CALL delete\_users()

ELSE

CALL admin\_interface

ENDIF

ENDWHILE

ENDPROCEDURE

PROCEDURE edit\_user\_information()

WHILE TRUE

PRINT “Please enter username (Press enter to go back)”

INPUT username

IF length(username) < 1 THEN

manage\_users()

BREAK

ELSE IF username NOT IN username\_database THEN

PRINT “That user does not exist”

ELSE

user\_index = user\_info\_database.index(username)

BREAK

ENDIF

ENDWHILE

PRINT “==Viewing/Editing user info==”

PRINT “Current info”

PRINT “Username : “, STR(user\_info\_database[user\_index])

PRINT “Password:”, STR(user\_info\_database[user\_index + 1])

PRINT “First Name:”, STR(user\_info\_database[user\_index + 2])

PRINT “Last Name:”, STR(user\_info\_database[user\_index + 3])

PRINT “Date of birth:”, STR(user\_info\_database[user\_index + 4])

PRINT “Contact number:”, STR(user\_info\_database[user\_index] + 5})

PRINT “Email:”, STR(user\_info\_database[user\_index + 6])

PRINT “ Press enter to edit account information. Press q (or any other key) to go back”

INPUT response

IF length(response) <> 0 THEN

CALL manage\_users()

RETURN

ELSE

PASS

ENDIF

username = user\_info\_database[user\_index]

password = user\_info\_database[user\_index + 1]

email = user\_info\_database[user\_index + 6]

PRINT “Enter a new first name (Press enter to keep current)”

INPUT first\_name

IF length(first\_name) < 1 THEN

first\_name = user\_info\_database[user\_index + 2]

END IF

PRINT “Enter a new last name (Press enter to keep current)”

INPUT last\_name

IF length(last\_name) < 1 THEN

last\_name = user\_info\_database[user\_index + 3]

ENDIF

WHILE TRUE

PRINT “Enter your new date of birth (Press enter to keep current)”

INPUT date\_of\_birth

IF length(date\_of\_birth) < 1 THEN

date\_of\_birth = user\_info\_database[user\_index + 4]

BREAK

ELSE IF NOT valid\_date\_format(date\_of\_birth) THEN

PRINT “ Invalid date of birth! Please re-enter.”

ELSE

BREAK

ENDWHILE

WHILE TRUE

PRINT “Enter your new contact number (Press enter to keep current)”

INPUT contact\_number

IF length(contact\_number) < 1 THEN

contact\_number = user\_info\_database[user\_index + 4]

BREAK

ELSE IF NOT valid\_contact(contact\_number) THEN

PRINT “ Invalid contact number! Please re-enter.”

ELSE

BREAK

ENDWHILE

user\_info\_database[user\_index] = username

user\_info\_database[user\_index + 1] = password

user\_info\_database[user\_index + 2] = first\_name

user\_info\_database[user\_index + 3] = last\_name

user\_info\_database[user\_index + 4] = date\_of\_birth

user\_info\_database[user\_index + 5] = contact\_number

user\_info\_database[user\_index + 6] = email

PRINT “ Account successfully updated!”

PRINT “Here is the latest info for user {user\_info\_database[user\_index]}”

PRINT “Username : “, STR(user\_info\_database[user\_index])

PRINT “Password:”, STR(user\_info\_database[user\_index + 1])

PRINT “First Name:”, STR(user\_info\_database[user\_index + 2])

PRINT “Last Name:”, STR(user\_info\_database[user\_index + 3])

PRINT “Date of birth:”, STR(user\_info\_database[user\_index + 4])

PRINT “Contact number:”, STR(user\_info\_database[user\_index] + 5})

PRINT “Email:”, STR(user\_info\_database[user\_index + 6])

PRINT “Returning to user management page….”

CALL manage\_users

ENDPROCEDURE

FUNCTION delete\_user()

WHILE TRUE

PRINT “Please enter username (Press enter to go back)”

INPUT username

IF length(username) < 1 THEN

manage\_users()

ELSE IF username NOT IN username\_ database THEN

PRINT “This user does not exist”

ELSE

user\_index = user\_info\_database.index(username)

BREAK

ENDIF

ENDWHILE

TRY

IF type(user\_info\_database[user\_index + 7] IS DICT THEN

PRINT “This user has halls booked! Please remove them before deleting their account”

CALL admin\_interface

RETRUN

ENDIF

EXCEPT IndexError

PRINT “This user has not bookings. You can proceed to delete”

PASS

ENDTRY

WHILE TRUE

PRINT “Are you sure you want to delete this account? This cannot be undone! (Y/N)?”

INPUT answer

IF UPPER(answer) == “Y” THEN

username\_database.remove(user\_info\_database[user\_index])

i = 0

WHILE i < 7

user\_info\_database.remove(user\_info-database[user\_index + i])

INCREMENT i

ENDWHILE

PRINT “Account has been deleted successfully.”

PRINT “Returning to user management page…”

manage\_users

BREAK

ELSE IF UPPER(answer) == “N” THEN

PRINT “Returning to user management page…”

CALL manage\_users()

ELSE

PRINT “Please provide a valid response”

ENDIF

ENDWHILE

ENDWHILE

ENDFUNCTION

PROCEDURE hall\_management()

PRINT “==Hall Management==”

PRINT “(1) Create new hall”

PRINT “(2) View halls”

PRINT “(3) Edit a hall”

PRINT “(4) Delete a hall”

PRINT “(5) Go back”

WHILE TRUE

PRINT “Please pick an option by typing in the number”

INPUT option

IF option NOT IN “12345” OR length(option) <> 1 THEN

PRINT “Please enter a valid option”

ELSE IF option == “1” THEN

CALL create\_new\_hall()

ELSE IF option == "2” THEN

CALL view\_halls()

ELSE IF option == "3” THEN

CALL edit\_hall()

ELSE IF option == “4” THEN

CALL delete\_a\_hall()

ELSE

CALL admin\_interface

ENDIF

ENDWHILE

ENDPROCEDURE

FUNCTION create\_new\_hall()

PRINT “==New hall creation==”

WHILE TRUE

PRINT “Please enter a hall ID (Press enter to go back”)

INPUT hall\_id

IF length(hall\_id) < 1 THEN

hall\_management()

RETURN

ELSE IF (hall\_id IN available\_hall\_ids) OR (hall\_id IN booked\_hall\_ids) THEN

PRINT “That hall id is already taken. Please enter a different one”)

ELSE

BREAK

ENDIF

ENDWHILE

WHILE TRUE

PRINT “ Enter the name of the hall”

INPUT hall\_name

IF length(hall\_name) < 1 THEN

PRINT “ Please do not leave this empty”

ELSE

BREAK

ENDWHILE

WHILE TRUE

PRINT “ Enter the description of the hall”

INPUT hall\_description

IF length(hall\_description) < 1 THEN

PRINT “ Please do not leave this empty”

ELSE

BREAK

ENDWHILE

PRINT “Please enter the number of pax of the hall”

INPUT hall\_pax

WHILE NOT hall\_pax.IsDigits()

PRINT “Please enter a valid number”

INPUT hall\_pax

ENDWHILE

PRINT “Please enter the daily price rate of the hall: ($)”

INPUT hall\_price

WHILE NOT hall\_price.IsDigits()

PRINT “Please enter a valid number”

INPUT hall\_price

ENDWHILE

available\_hall\_ids.Append(hall\_id)

available\_halls.Append(hall\_id)

available\_halls.Append(hall\_name)

available\_halls.Append(hall\_description)

available\_halls.Append(hall\_pax)

available\_halls.Append(hall\_price)

PRINT “New hall successfully created!”

PRINT “Returning to hall management page…”

hall\_managemennt()

ENDFUNCTION

FUNCTION view\_halls()

PRINT “==Viewing hall information==”

PRINT “Available halls (by ID) : “, STR(available\_hall\_ids)

PRINT “Booked halls (by ID) : “, STR(booked\_hall\_ids)

WHILE TRUE

PRINT “Please enter the ID of the hall you want to view (Press enter to go back to main page)”

INPUT hall\_id

IF LENGTH(hall\_id) < 1 THEN

CALL hall\_management()

RETURN

ELSE IF hall\_id IN available\_hall\_ids THEN

hall\_index = available\_hall\_ids(hall\_id)

PRINT “Displaying hall info of “, hall\_id

PRINT “ ID: “, STR(available\_halls[hall\_index])

PRINT “Name: “, STR(available\_halls[hall\_index + 1])

PRINT “Description: “, STR(available\_halls[hall\_index + 2])

PRINT “Hall pax: “, STR(available\_halls[hall\_index + 3])

PRINT “Status : Available”

PRINT “Rate price per day: $“, STR(available\_halls[hall\_index + 4])

ELSE IF hall\_id IN booked\_hall\_ids THEN

hall\_index = available\_hall\_ids(hall\_id)

PRINT “Displaying hall info of “, hall\_id

PRINT “ ID: “, STR(available\_halls[hall\_index])

PRINT “Name: “, STR(available\_halls[hall\_index + 1])

PRINT “Description: “, STR(available\_halls[hall\_index + 2])

PRINT “Hall pax: “, STR(available\_halls[hall\_index + 3])

PRINT “Status : Booked”

PRINT “Rate price per day: $“, STR(available\_halls[hall\_index + 4])

ELSE

PRINT “That hall does not exist!”

ENDIF

ENDWHILE

ENDFUNCTION

FUNCTION edit\_hall()

PRINT “ ==Editing hall information==”

PRINT “Available halls (by ID) : “, STR(available\_hall\_ids)

PRINT “Booked halls (by ID) : “, STR(booked\_hall\_ids)

WHILE TRUE

PRINT “Please enter the ID of the hall you want to view (Press enter to go back to main page)”

INPUT hall\_id

IF LENGTH(hall\_id) < 1 THEN

CALL hall\_management()

RETURN

ELSE IF hall\_id IN available\_hall\_ids THEN

hall\_index = available\_halls.Index(hall\_id)

PRINT “Current Hall Info”

PRINT “ ID: “, STR(available\_halls[hall\_index])

PRINT “Name: “, STR(available\_halls[hall\_index + 1])

PRINT “Description: “, STR(available\_halls[hall\_index + 2])

PRINT “Hall pax: “, STR(available\_halls[hall\_index + 3])

PRINT “Status : Available”

PRINT “Rate price per day: $“, STR(available\_halls[hall\_index + 4])

PRINT “Press enter to edit hall information. Press q (or any other key) to go back”

INPUT response

IF LENGTH(response) <> 0 THEN

CALL hall\_management()

RETURN

ENDIF

PRINT “Please enter a new hall name (Press enter to keep current)”

INPUT hall\_name

IF LENGTH(hall\_name) < 1 THEN

hall\_name = available\_halls[hall\_index + 1]

ENDIF

PRINT “Please enter a new hall description (Press enter to keep current)”

INPUT hall\_description

IF LENGTH(hall\_description) < 1 THEN

hall\_description = available\_halls[hall\_index + 2]

ENDIF

WHILE TRUE

PRINT “Please enter the new number of pax of the hall (Press enter to keep current)”

INPUT hall\_pax

IF LENGTH(hall\_pax) < 1 THEN

hall\_pax = available\_halls[hall\_index + 1]

BREAK

ELSE IF hall\_pax.IsDigits

BREAK

ELSE

PRINT “Please enter a valid number”

ENDIF

ENDWHILE

WHILE TRUE

PRINT “Please enter a new daily price rate of the hall (Press enter to keep current)”

INPUT hall\_price

IF LENGTH(hall\_price) < 1 THEN

hall\_price = available\_halls[hall\_index + 1]

BREAK

ELSE IF hall\_price.IsDigits

BREAK

ELSE

PRINT “Please enter a valid number”

ENDIF

ENDWHILE

available\_halls[hall\_index + 1] = hall\_name

available\_halls[hall\_index + 2] = hall\_description

available\_halls[hall\_index + 3] = hall\_pax

available\_halls[hall\_index + 4] = hall\_price

PRINT “Hall successfully updated! Here is the latest hall info”

PRINT “ ID: “, STR(available\_halls[hall\_index])

PRINT “Name: “, STR(available\_halls[hall\_index + 1])

PRINT “Description: “, STR(available\_halls[hall\_index + 2])

PRINT “Hall pax: “, STR(available\_halls[hall\_index + 3])

PRINT “Status : Available”

PRINT “Rate price per day: $“, STR(available\_halls[hall\_index + 4])

PRINT “Returning to hall management page”

CALL hall\_management()

BREAK

ELSE IF hall\_id IN booked\_hall\_ids THEN

hall\_index = booked\_halls.Index(hall\_id)

PRINT “Current Hall Info”

PRINT “ ID: “, STR booked \_halls[hall\_index])

PRINT “Name: “, STR(booked\_halls[hall\_index + 1])

PRINT “Description: “, STR(booked\_halls[hall\_index + 2])

PRINT “Hall pax: “, STR(booked\_halls[hall\_index + 3])

PRINT “Status : Booked”

PRINT “Rate price per day: $“, STR(booked\_halls[hall\_index + 4])

PRINT “Press enter to edit hall information. Press q (or any other key) to go back”

INPUT response

IF LENGTH(response) <> 0 THEN

CALL hall\_management()

RETURN

ENDIF

PRINT “Please enter a new hall name (Press enter to keep current)”

INPUT hall\_name

IF LENGTH(hall\_name) < 1 THEN

hall\_name = booked\_halls[hall\_index + 1]

ENDIF

PRINT “Please enter a new hall description (Press enter to keep current)”

INPUT hall\_description

IF LENGTH(hall\_description) < 1 THEN

hall\_description = booked \_halls[hall\_index + 2]

ENDIF

WHILE TRUE

PRINT “Please enter the new number of pax of the hall (Press enter to keep current)”

INPUT hall\_pax

IF LENGTH(hall\_pax) < 1 THEN

hall\_pax = booked\_halls[hall\_index + 1]

BREAK

ELSE IF hall\_pax.IsDigits

BREAK

ELSE

PRINT “Please enter a valid number”

ENDIF

ENDWHILE

WHILE TRUE

PRINT “Please enter a new daily price rate of the hall (Press enter to keep current)”

INPUT hall\_price

IF LENGTH(hall\_price) < 1 THEN

hall\_price = booked\_halls[hall\_index + 1]

BREAK

ELSE IF hall\_price.IsDigits

BREAK

ELSE

PRINT “Please enter a valid number”

ENDIF

ENDWHILE

booked\_halls[hall\_index + 1] = hall\_name

booked\_halls[hall\_index + 2] = hall\_description

booked\_halls[hall\_index + 3] = hall\_pax

booked\_halls[hall\_index + 4] = hall\_price

PRINT “Hall successfully updated! Here is the latest hall info”

PRINT “ ID: “, STR(booked\_halls[hall\_index])

PRINT “Name: “, STR(booked\_halls[hall\_index + 1])

PRINT “Description: “, STR(booked\_halls[hall\_index + 2])

PRINT “Hall pax: “, STR(booked\_halls[hall\_index + 3])

PRINT “Status :Booked ”

PRINT “Rate price per day: $“, STR(booked\_halls[hall\_index + 4])

PRINT “Returning to hall management page”

CALL hall\_management()

BREAK

ELSE

PRINT “ That hall does not exist”

ENDWHILE

ENDFUNCTION

FUNCTION delete\_a\_hall()

PRINT “==Deleting a hall==”

PRINT “Note: You can only delete halls that have not been booked yet”

PRINT” If you want to delete a hall that is booked, you have to first remove the booking via the ‘Remove a booked hall from a user’ option under ‘Manage bookings’ ”

PRINT “Available halls (by ID):”, STR(available\_hall\_ids)

WHILE TRUE

PRINT “Please enter the ID of the hall you want to delete (Press enter to go back to main page)”

IF LENGTH(hall\_id) < 1 THEN

CALL hall\_management

RETRUN

ELSE IF hall\_id IN available\_hall\_ids THEN

hall\_index = available\_halls.Index(hall\_id)

BREAK

ELSE

PRINT “Please select a valid option”

ENDIF

available\_hall\_ids.Remove(available\_halls[index])

i = 0

WHILE i < 5

available\_halls.Remove(available\_halls[hall\_index + i])

INCREMENT i

ENDWHILE

PRINT “Hall successfully deleted!”

PRINT “Returning to hall management page…”

CALL hall\_management()

ENDWHILE

ENDFUNCTION

PROCEDURE manage\_bookings()

PRINT “==Booking Management==”

PRINT “(1) View booked halls”

PRINT “(2) Edit booked hall information of a user”

PRINT “(3) Remove a booked hall from a user ”

PRINT “(4) Go back”

WHILE TRUE

PRINT “Please pick an option by typing in the number”

INPUT option

IF option NOT IN “1234” OR length(option) <> 1 THEN

PRINT “Please enter a valid option”

ELSE IF option == “1” THEN

CALL view\_booked\_halls()

ELSE IF option == "2” THEN

CALL edit\_booked\_hal\_information()

ELSE IF option == "3” THEN

CALL remove\_booked\_hall()

ELSE

CALL admin\_interface

ENDIF

ENDWHILE

ENDPROCEDURE

PROCEDURE view\_booked\_halls()

PRINT “==Viewing booked halls==”

PRINT “Displaying user information of all booked halls now”

PRINT “List of booked halls by IDs :”, STR(booked\_hall\_ids)

PRINT “(If nothing shows up, that means no one has made nay bookings yet)”

FOR item IN username\_database

user\_index = user\_info\_database.Index(item)

x = 1

WHILE TRUE

TRY

IF TYPE(user\_info\_database[user\_index + 6 + x] IS DICT THEN

PRINT “Booking”, x

PRINT “Username:”, STR(item)

PRINT “Hall ID: “, STR(user\_info\_database[user\_index + 6 + x][“hall\_id”])

PRINT “Event Name: “, STR(user\_info\_database[user\_index + 6 + x][“event\_name”])

PRINT “Event Description: “, STR(user\_info\_database[user\_index + 6 + x][“event \_description”])

PRINT “Number of Pax: “, STR(user\_info\_database[user\_index + 6 + x][“ number\_of\_pax”])

PRINT “Date: “, STR(user\_info\_database[user\_index + 6 + x][“date”])

PRINT “Time: “, STR(user\_info\_database[user\_index + 6 + x][“time”])

PRINT “Payment Price :$ “, STR(user\_info\_database[user\_index + 6 + x][“payment\_price”])

PRINT “ “

ELSE

BREAK

ENDIF

EXCEPT IndexError

BREAK

ENDTRY

INCREMENT x

ENDWHILE

CALL manage\_bookings()

NEXT item

ENDPROCEDURE

FUNCTION edit\_booked\_hall\_information()

PRINT “==Editing booked hall information of a user==”

PRINT “Please enter username (Press enter to go back)”

INPUT username

IF LENGTH(username) < 1 THEN

CALL manage\_bookings()

RETURN

ELSE IF username NOT IN username\_database THEN

PRINT “That user does not exist”

CALL edit\_booked\_hall\_information()

ENDIF

user\_index = user\_info\_database.Index(username)

TRY

IF TYPE(user\_info\_database[user\_index + 7] IS DICT THEN

PASS

ELSE

PRINT “That user does not have any bookings!”

CALL edit\_booked\_hall\_information()

ENDIF

EXCEPT IndexError

BREAK

ENDTRY

ENDTRY

PRINT “Here is the list of bookings by”, STR(username)

x = 1

WHILE TRUE

TRY

IF TYPE(user\_info\_database[user\_index + 6 + x]) IS DICT THEN

PRINT “Booking”, x

PRINT “Event Name: “, STR(user\_info\_database[user\_index + 6 + x][“event\_name”])

PRINT “ “

INCREMENT x

ELSE

BREAK

ENDIF

EXCEPT IndexError

BREAK

ENDTRY

ENDWHILE

WHILE

PRINT “Select a booking to edit by typing in the booking number (Press enter to go back)”

INPUT option

IF LENGTH(option) == 0 THEN

CALL edit\_booked\_hall\_information()

BREAK

ELSE IF option.IsDigit() THEN

TRY

IF TYPE(user\_info\_database[user\_index + 6 + INT(option)]) IS DICT THEN

BREAK

ELSE

PRINT “Please select a valid option!”

ENDIF

EXCEPT IndexError

PRINT “Please select a valid option!”

ENDTRY

ELSE

PRINT “Please select a valid option!”

ENDIF

ENDWHILE

PRINT “You are now editing booking”, STR(option)

PRINT “Please enter a new event name (Press enter to keep current)”

INPUT event\_name

IF LENGTH(event\_name) < 1 THEN

event\_name = user\_info\_database[user\_index + 6 +INT(option)][“event\_name”]

ENDIF

PRINT “Please enter a new event description (Press enter to keep current)”

INPUT event\_description

IF LENGTH(event\_description) < 1 THEN

event\_description = user\_info\_database[user\_index + 6 +INT(option)][“event\_description”]

ENDIF

WHILE TRUE

PRINT “Please enter a new number of pax (Press enter to keep current)”

INPUT number\_of\_pax

IF LENGTH(number\_of\_pax) < 1 THEN

number\_of\_pax = user\_info\_database[user\_index + 6 +INT(option)][“ number\_of\_pax”]

BREAK

ENDIF

IF number\_of\_pax.IsDigits THEN

BREAK

ELSE

PRINT “Please enter a valid number”

ENDIF

ENDWHILE

WHILE TRUE

PRINT “Please enter a new date of rental (Press enter to keep current)”

INPUT date

IF LENGTH(date) < 1 THEN

date = user\_info\_database[user\_index + 6 +INT(option)][“date”]

BREAK

ELSE IF NOT valid\_date\_format(date) THEN

PRINT “Invalid date! Please re-enter.”

ELSE

BREAK

ENDIF

ENDWHILE

WHILE TRUE

PRINT “Please enter a new time of rental (Press enter to keep current)”

INPUT time

IF LENGTH(time) < 1 THEN

time = user\_info\_database[user\_index + 6 +INT(option)][“time”]

BREAK

ENDIF

ELSE IF NOT valtime(time) THEN

PRINT “Invalid time or incorrect format! Please re-enter.”

ELSE

BREAK

ENDIF

ENDWHILE

WHILE TRUE

PRINT “Please enter a new payment price (Press enter to keep current)”

INPUT payment\_price

IF LENGTH(payment\_price) < 1 THEN

payment\_price = user\_info\_database[user\_index + 6 +INT(option)][“ payment\_price”]

BREAK

ELSE IF payment\_price.IsDigits THEN

BREAK

ELSE

PRINT “Please enter a valid number”

ENDIF

ENDWHILE

user\_info\_database[user\_index + 6 + INT(option)][“event\_name”] = event\_name

user\_info\_database[user\_index + 6 + INT(option)][“event\_description”] = event\_description

user\_info\_database[user\_index + 6 + INT(option)][“number\_of\_pax”] = number\_of\_pax

user\_info\_database[user\_index + 6 + INT(option)][“date”] = date

user\_info\_database[user\_index + 6 + INT(option)][“time”] = time

user\_info\_database[user\_index + 6 + INT(option)][“payment\_price”] = payment\_price

PRINT “Booking information successfully updated! Here is the latest booking information for”, STR(username)

PRINT “Booking”, (option)

PRINT “Hall ID :”, STR(user\_info\_database[user\_index + 6 + int(option)][“hall\_id”])

PRINT “Event Name:”, STR(user\_info\_database[user\_index + 6 + int(option)][“event\_name”])

PRINT “Event Description:”, STR(user\_info\_database[user\_index + 6 + int(option)][“event\_description”])

PRINT “Number of Pax:”, STR(user\_info\_database[user\_index + 6 + int(option)][“number\_of\_pax”])

PRINT “Date :”, STR(user\_info\_database[user\_index + 6 + int(option)][“date”])

PRINT “Time:”, STR(user\_info\_database[user\_index + 6 + int(option)][“time”])

PRINT “Payment Price : $”, STR(user\_info\_database[user\_index + 6 + int(option)][“payment\_price”])

PRINT “Returning to booking management page…”

CALL manage\_bookings()

ENDFUNCTION

FUNCTION remove\_booked\_hall()

PRINT “==Removing a booked hall from a userr==”

PRINT “Please enter a username (Press enter to go back)”

INPUT username

IF LENGTH(username) < 1 THEN

CALL manage\_bookings()

RETURN

ELSE IF username NOT IN username\_database THEN

PRINT “That user does not exist!”

CALL remove\_booked\_hall()

END IF

user\_index = user\_info\_database.Index(username)

TRY

IF TYPE(user\_info\_database[user\_index + 7] IS DICT THEN

PASS

ELSE

PRINT “That user does not have any bookings!”

CALL remove\_booked\_hall()

ENDIF

EXCEPT IndexError

BREAK

ENDTRY

PRINT “Here is the list of bookings by”, STR(username)

x = 1

WHILE

TRY

IF TYPE(user\_info\_database[user\_index + 6 + x]) IS DICT THEN

PRINT “Booking”, x

PRINT “Event Name: ”, STR(user\_info\_database[user\_index + 6 + x][“event\_name”]

PRINT “ “

INCREMENT x

ELSE

BREAK

ENDIF

EXCEPT IndexError

BREAK

ENDTRY

ENDWHILE

WHILE TRUE

PRINT “Select a booking to remove by typing in the booking number (Press enter to go back)”

INPUT option

IF LENGTH(option) == 0 THEN

CALL remove\_booked\_hall()

BREAK

ELSE IF option.IsDigit() THEN

TRY

IF TYPE (user\_info)database[user\_index + 6 + INT(option)] IS DICT THEN

hall\_id = user\_info\_database[user\_index + 6 + INT(option)][“hall\_id”]

hall\_index = booked\_halls.Index(hall\_id)

available\_hall\_ids.Append(booked\_halls[hall\_index])

available\_halls.Append(booked\_halls[hall\_index])

available\_halls.Append(booked\_halls[hall\_index + 1 ])

available\_halls.Append(booked\_halls[hall\_index + 2 ])

available\_halls.Append(booked\_halls[hall\_index + 3 ])

available\_halls.Append(booked\_halls[hall\_index + 4 ])

user\_info\_database.Remove(user\_info\_database[user\_index + 6 + INT(option)])

booked\_hall\_ids.Remove(booked\_halls[hall\_index])

i = 0

WHILE i < 5

booked\_halls.Remove(booked\_halls[hall\_index])

INCREMENT i

ENDWHILE

PRINT “Booking successfully removed!”

PRINT “Returning to booking management page…”

CALL manage\_bookings()

ELSE

PRINT “Please select a valid option!”

ENDIF

EXCEPT IndexError

PRINT “Please select a valid option!”

ENDTRY

ELSE

PRINT “Please select a valid option!”

ENDIF

ENDWHILE

ENDFUNCTION

END

# **3.0 SOURCE CODE AND PROGRAM BREAKDOWN**

## **3.1 User Source Code**

### 3.1.1 Line 1-3

from datetime import datetime  
import re  
import json

* Imports external libraries such as **datetime**, **re** and **json**.

### 3.1.2 Line 4-5

ADMIN\_USERNAME = "Banana"  
ADMIN\_PASSWORD = "B4n4n4"

* This variables is used to store the admin login credentials.

### 3.1.3 Line 8-48

try:  
 with open('username\_database.txt', 'r') as f:  
 username\_database = json.load(f)  
except FileNotFoundError:  
 print('Could not run the code as the file "username\_database.txt" could not be found!')  
 exit()  
  
try:  
 with open('user\_info\_database.txt', 'r') as f:  
 user\_info\_database = json.load(f)  
except FileNotFoundError:  
 print('Could not run the code as the file "user\_info\_database.txt" could not be found!')  
 exit()  
  
try:  
 with open('available\_hall\_ids.txt', 'r') as f:  
 available\_hall\_ids = json.load(f)  
except FileNotFoundError:  
 print('Could not run the code as the file "available\_hall\_ids.txt" could not be found!')  
 exit()  
  
try:  
 with open('available\_halls.txt', 'r') as f:  
 available\_halls = json.load(f)  
except FileNotFoundError:  
 print('Could not run the code as the file "available\_halls.txt" could not be found!')  
 exit()  
  
try:  
 with open('booked\_hall\_ids.txt', 'r') as f:  
 booked\_hall\_ids = json.load(f)  
except FileNotFoundError:  
 print('Could not run the code as the file "booked\_hall\_ids.txt" could not be found!')  
 exit()  
  
try:  
 with open('booked\_halls.txt', 'r') as f:  
 booked\_halls = json.load(f)  
except FileNotFoundError:  
 print('Could not run the code as the file "booked\_halls.txt" could not be found!')  
 exit()

* This line of codes checks for database files in the folder
* If file is missing error message is sent and program will not run.

### 3.1.4 Line 51-63

def valid\_password(\_password):  
 lower, upper, number = 0, 0, 0  
 for c in \_password:  
 if 97 <= ord(c) <= 122:  
 lower += 1  
 if 65 <= ord(c) <= 90:  
 upper += 1  
 if 48 <= ord(c) <= 57:  
 number += 1  
 if lower < 2 or upper < 2 or number < 2 or len(\_password) < 8:  
 return False  
 else:  
 return True

* The function **valid\_password()** is used to validate whether a password contains at least 2 lowercase, 2 uppercase and 2 numbers. If criteria does not meet the code will return **False**.

### 3.1.5 Line 66-71

def valid\_date\_format(date\_string, date\_format="%d/%m/%Y"):  
 try:  
 datetime.strptime(date\_string, date\_format)  
 return True  
 except ValueError:  
 return False

* This function **valid\_date\_format()** uses the datetime library, which enables this code to work with date objects. The function itself is used to validate date format ensuring it is written in DD/MM/YYYY format. If criteria is not met function will return **False**.

### 3.1.6 Line 74-77

def valtime(\_time):  
 pattern = re.compile("^(0[1-9]|1[0-2]):[0-5][0-9] [APap][Mm]$")  
 if not pattern.match(\_time):  
 return False

* The **valtime()** function uses the re library or **RegEx** which stands for regular expressions. One of the uses of this library is to validate if a string matches a certain format.
* For example, in this function **re.compile** creates a format where the first character must be either 0 or 1, than a digit between 0 to 2, followed by “:”, next to it is a digit between 0 to 5, besides that a digit between 0 to 9, followed by a space and ends with “AM” or “PM” either in uppercase or lowercase.

### 3.1.7 Line 80-82

def valid\_email(\_email):  
 format = r'\b[A-Za-z0-9.%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'  
 return bool(re.fullmatch(format, \_email))

* The **valid\_email()** function uses the re library to validate email format. Ensures that anything before the domain name ends with “@.”.

### 3.1.8 Line 85-87

def valid\_contact(phone\_number):  
 pattern = re.compile(r'^[+\d-]+$')  
 return bool(re.fullmatch(pattern, phone\_number))

* The function **valid\_contact()** uses the re library to validate phone number format.

### 3.1.9 Line 90-130

def main():  
 print("==Welcome to Banana Halls! What would you like to do?==")  
 print("(1) Sign up as User")  
 print("(2) Login as User")  
 print("(3) Login as Admin")  
 print("(4) Quit")  
  
 while True:  
 option = input("Please pick an option by typing in the number: ")  
 if option not in "1234" or len(option) != 1:  
 print("Please enter a valid option!")  
 elif option == "1":  
 sign\_up()  
 break  
 elif option == "2":  
 login()  
 break  
 elif option == "3":  
 login\_as\_admin()  
 break  
 else:  
 with open('username\_database.txt', 'w') as f:  
 json.dump(username\_database, f)  
  
 with open('user\_info\_database.txt', 'w') as f:  
 json.dump(user\_info\_database, f)  
  
 with open('available\_hall\_ids.txt', 'w') as f:  
 json.dump(available\_hall\_ids, f)  
  
 with open('available\_halls.txt', 'w') as f:  
 json.dump(available\_halls, f)  
  
 with open('booked\_hall\_ids.txt', 'w') as f:  
 json.dump(booked\_hall\_ids, f)  
  
 with open('booked\_halls.txt', 'w') as f:  
 json.dump(booked\_halls, f)  
  
 print("Goodbye!")  
 exit()

The first few lines of the **main()** function displays all options available to the user in the main menu interface.

Following that, the rest of the **main()** function calls a function depending on which option the user picks;

* If **option == “1”**, the function **sign\_up()** is called
* If **option == “2”**, the function **login()** is called
* If **option == “3”**, the function **login\_as\_admin()** is called
* If **option == “4”**, the function stores all data variables collected from the user to text database files, and will call on the function **exit()** which will end the program.

### 3.1.10 Line 133-205

def sign\_up():  
 print("==New account creation==")  
 while True:  
 username = input("Enter a username (Press enter to go back to main page): ")  
 if len(username) < 1:  
 main()  
 return  
 elif username in username\_database:  
 print("That username already is already taken! Please enter a different username")  
 else:  
 break  
  
 print("Your password should contain at least 8 characters.")  
 print("Your password should also include at least of 2 numbers, 2 uppercase and 2 lowercase characters.")  
 password = input("Enter a password: ")  
 while not valid\_password(password):  
 print("Your password is not strong enough! Please follow the format above.")  
 password = input("Please re-enter your password: ")  
 valid\_password(password)  
  
 while True:  
 first\_name = input("Enter your first name: ")  
 if len(first\_name) < 1:  
 print("Please don't leave this empty!")  
 else:  
 break  
  
 while True:  
 last\_name = input("Enter your last name: ")  
 if len(last\_name) < 1:  
 print("Please don't leave this empty!")  
 else:  
 break  
  
 while True:  
 date\_of\_birth = input("Enter your date of birth (DD/MM/YYYY): ")  
 if not valid\_date\_format(date\_of\_birth):  
 print("Invalid date of birth! Please re-enter.")  
 else:  
 break  
  
 while True:  
 contact\_number = input("Enter your contact number: ")  
 if len(contact\_number) < 1:  
 print("Please don't leave this empty!")  
 else:  
 if not valid\_contact(contact\_number):  
 print("Invalid contact number! Please re-enter.")  
 else:  
 break

while True:  
 email = input("Enter your email: ")  
 if len(email) < 1:  
 print("Please don't leave this empty!")  
 else:  
 if not valid\_email(email):  
 print("Invalid email address! Please re-enter.")  
 else:  
 break

username\_database.append(username)  
 user\_info\_database.append(username)  
 user\_info\_database.append(password)  
 user\_info\_database.append(first\_name)  
 user\_info\_database.append(last\_name)  
 user\_info\_database.append(date\_of\_birth)  
 user\_info\_database.append(contact\_number)  
 user\_info\_database.append(email)  
  
 print("New user successfully created! Please login from the main page.")  
 print("Returning to main page...")  
 main()

* The **sign\_up()** function is used to collect user information such as; username, password, first name, last name, date of birth, contact number and email to create a user account. Excluding first name and last name, each input calls a function that has a validation process to ensure information is correct and in the right format.
* Data collected is appended into a list.
* Function **main()** is called at the end which returns user to the user menu interface.

### 3.1.11 Line 208-225

def login():  
 print("==Login to your account==")  
 username = input("Please enter username (Press enter to go back to main page): ")  
  
 if len(username) < 1:  
 main()  
 return  
 elif username not in username\_database:  
 print(f"That user does not exist!")  
 login()  
  
 user\_index = user\_info\_database.index(username)  
 password = input("Please enter password: ")  
 if password != user\_info\_database[user\_index + 1]:  
 print("Incorrect password!")  
 login()  
 else:  
 user\_interface(user\_index)

* The function **login()** compares the user’s username and password to the ones stored in the database. If no error is found such as **“That user does not exist!”** or **“Incorrect password!”** the function **user\_interface(user\_index)** is called.

### 3.1.12 Line 228-259

def user\_interface(user\_index):  
 print(f"==Welcome, {user\_info\_database[user\_index]}! What would you like to do?==")  
 print("(1) Make a new booking")  
 print("(2) View all halls")  
 print("(3) View/Edit/Delete bookings")  
 print("(4) View/Edit account into")  
 print("(5) Delete account")  
 print("(6) Logout")  
  
 while True:  
 option = input("Please pick an option by typing in the number: ")  
 if option not in "123456" or len(option) != 1:  
 print("Please enter a valid option!")  
 elif option == "1":  
 new\_booking(user\_index)  
 elif option == "2":  
 view\_all\_halls(user\_index)  
 elif option == "3":  
 view\_edit\_delete\_bookings(user\_index)  
 elif option == "4":  
 edit\_account\_info(user\_index)  
 elif option == "5":  
 try:  
 if type(user\_info\_database[user\_index + 7]) is dict:  
 print("You cannot delete your account with halls booked! Please delete your bookings before deleting your account.")  
 else:  
 delete\_account(user\_index)  
 except IndexError:  
 delete\_account(user\_index)  
 else:  
 print(f"See you again, {user\_info\_database[user\_index]}!")  
 main()

The **user\_interface(user\_index)** function provides users new options (1-6) users can access. Depending on which option is selected a different function is called.

* If **option == “1”**, the function **new\_booking(user\_index)** is called
* If **option == “2”**, the function **view\_all\_halls(user\_index)** is called
* If **option == “3”**, the function **view\_edit\_delte\_bookings(user\_index)** is called
* If **option == “4”**, the function **edit\_account\_info(user\_index)** is called
* If **option == “5”**, the function **delete\_account(user\_index)** is called

If user selects any other option, the function **main()** is called.

### 3.1.13 Line 262-360

def edit\_account\_info(user\_index):  
 print("==View/Edit Account Info==")  
 print("Current Info")  
 print(f"Username: {user\_info\_database[user\_index]}")  
 print(f"Password: {user\_info\_database[user\_index + 1]}")  
 print(f"First name: {user\_info\_database[user\_index + 2]}")  
 print(f"Last name: {user\_info\_database[user\_index + 3]}")  
 print(f"Date of birth: {user\_info\_database[user\_index + 4]}")  
 print(f"Contact number: {user\_info\_database[user\_index + 5]}")  
 print(f"Email: {user\_info\_database[user\_index + 6]}")  
  
 response = input("Press enter to edit account info. Press q (or any other key) to go back: ")  
 if len(response) != 0:  
 user\_interface(user\_index)  
 return  
 else:  
 pass  
  
 username\_database.remove(user\_info\_database[user\_index])  
 while True:  
 username = input("Enter your new username (Press enter to keep current): ")  
 if len(username) < 1:  
 username = user\_info\_database[user\_index]  
 break  
 elif username in username\_database:  
 print("That username already is already taken! Please enter a different username")  
 else:  
 break  
 username\_database.append(username)  
  
 print("Your password should contain at least 8 characters.")  
 print("Your password should also include at least of 2 numbers, 2 uppercase and 2 lowercase characters.")  
 password = input("Enter your new password (Press enter to keep current): ")  
 if len(password) < 1:  
 password = user\_info\_database[user\_index + 1]  
 else:  
 while not valid\_password(password):  
 print("Your password is not strong enough! Please follow the format above.")  
 password = input("Please re-enter your password: ")  
 valid\_password(password)  
  
 first\_name = input("Enter your new first name (Press enter to keep current): ")  
 if len(first\_name) < 1:  
 first\_name = user\_info\_database[user\_index + 2]  
  
 last\_name = input("Enter your new last name (Press enter to keep current): ")  
 if len(last\_name) < 1:  
 last\_name = user\_info\_database[user\_index + 3]  
  
 while True:  
 date\_of\_birth = input("Enter your new date of birth (Press enter to keep current) (DD/MM/YYYY): ")  
 if len(date\_of\_birth) < 1:  
 date\_of\_birth = user\_info\_database[user\_index + 4]  
 break  
 elif not valid\_date\_format(date\_of\_birth):  
 print("Invalid date of birth! Please re-enter.")  
 else:  
 break  
  
 while True:  
 contact\_number = input("Enter your new contact number (Press enter to keep current): ")  
 if len(contact\_number) < 1:  
 contact\_number = user\_info\_database[user\_index + 5]  
 break  
 else:  
 if not valid\_contact(contact\_number):  
 print("Invalid contact number! Please re-enter.")  
 else:  
 break  
  
 while True:  
 email = input("Enter your new email (Press enter to keep current): ")  
 if len(email) < 1:  
 email = user\_info\_database[user\_index + 6]  
 break  
 else:  
 if not valid\_email(email):  
 print("Invalid email address! Please re-enter.")  
 else:  
 break  
  
 user\_info\_database[user\_index] = username  
 user\_info\_database[user\_index + 1] = password  
 user\_info\_database[user\_index + 2] = first\_name  
 user\_info\_database[user\_index + 3] = last\_name  
 user\_info\_database[user\_index + 4] = date\_of\_birth  
 user\_info\_database[user\_index + 5] = contact\_number  
 user\_info\_database[user\_index + 6] = email  
  
 print("Account successfully updated! Here's your latest account info")  
 print(f"Username: {user\_info\_database[user\_index]}")  
 print(f"Password: {user\_info\_database[user\_index + 1]}")  
 print(f"First name: {user\_info\_database[user\_index + 2]}")  
 print(f"Last name: {user\_info\_database[user\_index + 3]}")  
 print(f"Date of birth: {user\_info\_database[user\_index + 4]}")  
 print(f"Contact number: {user\_info\_database[user\_index + 5]}")  
 print(f"Email: {user\_info\_database[user\_index + 6]}")  
 print("Returning to user interface...")  
 user\_interface(user\_index)

The function **edit\_account\_info(user\_index)** is used for the following:

* Displays the users current information from the database
* Allows users to enter new information or keep current information
* New information is validated
* Updates the database
* Displays the users updated information from the database
* Returns user to the user interface

### 3.1.14 Line 363-380

def delete\_account(user\_index):  
 while True:  
 answer = input("Are you sure you want to delete your account? This cannot be undone! (y/n): ")  
 if answer == "y":  
 username\_database.remove(user\_info\_database[user\_index])  
 i = 0  
 while i < 7:  
 user\_info\_database.remove(user\_info\_database[user\_index])  
 i += 1  
 print("Your account has been deleted successfully. Sorry to see you go!")  
 main()  
 break  
 elif answer == "n":  
 print("Returning to user interface...")  
 user\_interface(user\_index)  
 break  
 else:  
 print("Please provide a valid response!")

* The function **delete\_account(user\_index)** removes all information of the user from the database and returns user to the user interface.

### 3.1.15 Line 383-412

def view\_all\_halls(user\_index):  
 print("== Viewing hall information ==")  
 print(f"Available halls (by ID): {available\_hall\_ids}")  
 print(f"Booked halls (by ID): {booked\_hall\_ids}")  
  
 while True:  
 hall\_id = input("Please enter the ID of the hall you want to view (Press enter to go back to main page): ")  
 if len(hall\_id) < 1:  
 user\_interface(user\_index)  
 return  
 elif hall\_id in available\_hall\_ids:  
 hall\_index = available\_halls.index(hall\_id)  
 print(f"Displaying hall info of {hall\_id}")  
 print(f"ID: {available\_halls[hall\_index]}")  
 print(f"Name: {available\_halls[hall\_index + 1]}")  
 print(f"Description: {available\_halls[hall\_index + 2]}")  
 print(f"Hall pax: {available\_halls[hall\_index + 3]}")  
 print(f"Status: Available")  
 print(f"Rate price per day: ${available\_halls[hall\_index + 4]}")  
 elif hall\_id in booked\_hall\_ids:  
 hall\_index = booked\_halls.index(hall\_id)  
 print(f"Displaying hall info of {hall\_id}")  
 print(f"ID: {booked\_halls[hall\_index]}")  
 print(f"Name: {booked\_halls[hall\_index + 1]}")  
 print(f"Description: {booked\_halls[hall\_index + 2]}")  
 print(f"Hall pax: {booked\_halls[hall\_index + 3]}")  
 print(f"Status: Booked")  
 print(f"Rate price per day: ${booked\_halls[hall\_index + 4]}")  
 else:  
 print(f"That hall does not exist!")

* The function **view\_all\_halls(user\_index)** displays all available halls and all booked halls by the user from the database.

### 3.1.16 Line 415-506

def new\_booking(user\_index):  
 print("==Making a new booking==")  
 print(f"Available halls (by ID): {available\_hall\_ids}")  
 while True:  
 hall\_id = input("Enter a hall ID to view information about that hall (Press enter to go back): ")  
 if len(hall\_id) < 1:  
 user\_interface(user\_index)  
 exit()  
 elif hall\_id in available\_hall\_ids:  
 break  
 else:  
 print("That hall ID does not exist!")  
  
 hall\_index = available\_halls.index(hall\_id)  
 print("Hall Info")  
 print(f"ID: {available\_halls[hall\_index]}")  
 print(f"Name: {available\_halls[hall\_index + 1]}")  
 print(f"Description: {available\_halls[hall\_index + 2]}")  
 print(f"Hall pax: {available\_halls[hall\_index + 3]}")  
 print(f"Status: Available")  
 print(f"Rate price per day: ${available\_halls[hall\_index + 4]}")  
  
 response = input("Press enter to proceed with the booking, press q (or any other key) to go back: ")  
 if len(response) != 0:  
 new\_booking(user\_index)  
 return  
  
 while True:  
 event\_name = input("Please enter your event name: ")  
 if len(event\_name) < 1:  
 print("This cannot be empty!")  
 else:  
 break  
  
 while True:  
 event\_description = input("Please enter your event description: ")  
 if len(event\_description) < 1:  
 print("This cannot be empty!")  
 else:  
 break  
  
 while True:  
 number\_of\_pax = input("Please enter your new number of pax (Press enter to keep current): ")  
 if number\_of\_pax.isdigit():  
 if int(number\_of\_pax) > int(available\_halls[hall\_index + 3]):  
 print(f"That amount is too big! Maximum amount of pax is {available\_halls[hall\_index + 3]}")  
 else:  
 break  
 else:  
 print("Please enter a valid number!")  
  
 while True:  
 date = input("Please enter your date of rental (DD/MM/YYYY): ")  
 if not valid\_date\_format(date):  
 print("Invalid date! Please re-enter.")  
 else:  
 break  
  
 while True:  
 time = input("Please enter your time of rental (hh:mm am/pm): ")  
 if valtime(time) == False:  
 print("Invalid time or incorrect format! Please re-enter")  
 else:  
 break  
  
 while True:  
 payment\_price = input("Please enter your payment price: $")  
 if payment\_price.isdigit():  
 if int(payment\_price) < int(available\_halls[hall\_index + 4]):  
 print(f"That is not enough! The rate price per day is ${available\_halls[hall\_index + 4]}!")  
 else:  
 break  
 else:  
 print("Please enter a valid number!")  
  
 user\_info\_database.insert(user\_index + 7, {"hall\_id": available\_halls[hall\_index], "event\_name": event\_name, "event\_description": event\_description, "number\_of\_pax": number\_of\_pax, "date": date, "time": time, "payment\_price": payment\_price})  
 booked\_hall\_ids.append(available\_halls[hall\_index])  
 booked\_halls.append(available\_halls[hall\_index])  
 booked\_halls.append(available\_halls[hall\_index + 1])  
 booked\_halls.append(available\_halls[hall\_index + 2])  
 booked\_halls.append(available\_halls[hall\_index + 3])  
 booked\_halls.append(available\_halls[hall\_index + 4])  
  
 available\_hall\_ids.remove(available\_halls[hall\_index])  
 i = 0  
 while i < 5:  
 available\_halls.remove(available\_halls[hall\_index])  
 i += 1  
  
 print("Hall successfully booked!")  
 print("Returning to user interface...")  
 user\_interface(user\_index)

The function **new\_booking(user\_index)** allows the user to:

* Displays all available halls to the user
* User selects hall
* Hall information is shown, user may confirm hall selection to proceed
* Hall is selected
* Program requests information from user
* Information is validated
* Information is stored in database
* Booking is completed
* Function **user\_interface(user\_index)** is called and user returns to user menu interface

### 3.1.17 Line 509-527

def view\_edit\_delete\_bookings(user\_index):  
 print("==View/Edit/Delete Bookings==")  
 print("(1) View all bookings")  
 print("(2) Edit a booking")  
 print("(3) Delete a booking")  
 print("(4) Go back")  
  
 while True:  
 option = input("Please pick an option by typing in the number: ")  
 if option not in "1234" or len(option) != 1:  
 print("Please enter a valid option!")  
 elif option == "1":  
 view\_bookings(user\_index)  
 elif option == "2":  
 edit\_a\_booking(user\_index)  
 elif option == "3":  
 delete\_a\_booking(user\_index)  
 else:  
 user\_interface(user\_index)

The function **view\_edit\_delete\_bookings(user\_index)** provides users with new options based around the booked halls:

* If **option == “1”** function **view\_bookings(user\_index)** is called
* If **option == “2”** function **edit\_a\_booking(user\_index)** is called
* If **option == “3”** function **delete\_a\_booking(user\_index)** is called
* If **option == “4”** function **user\_interface(user\_index)** is called

### 3.1.18 Line 530-552

def view\_bookings(user\_index):  
 print("==Viewing all bookings==")  
 print("Here is the list of your bookings")  
 print("(If nothing shows up, that means you haven't made any bookings yet)")  
 x = 1  
 while True:  
 try:  
 if type(user\_info\_database[user\_index + 6 + x]) is dict:  
 print(f"Booking {x}")  
 print(f'Hall ID: {user\_info\_database[user\_index + 6 + x]["hall\_id"]}')  
 print(f'Event Name: {user\_info\_database[user\_index + 6 + x]["event\_name"]}')  
 print(f'Event Description: {user\_info\_database[user\_index + 6 + x]["event\_description"]}')  
 print(f'Number of Pax: {user\_info\_database[user\_index + 6 + x]["number\_of\_pax"]}')  
 print(f'Date: {user\_info\_database[user\_index + 6 + x]["date"]}')  
 print(f'Time: {user\_info\_database[user\_index + 6 + x]["time"]}')  
 print(f'Payment Price: ${user\_info\_database[user\_index + 6 + x]["payment\_price"]}')  
 print("")  
 x += 1  
 else:  
 break  
 except IndexError:  
 break  
 view\_edit\_delete\_bookings(user\_index)

* The function **view\_bookings(user\_index)** displays all of the user’s booked halls and their information from the database.
* If no halls were booked previously, the code will be skipped

### 3.1.19 Line 555-666

def edit\_a\_booking(user\_index):  
 print("==Edit a booking==")  
 print("Here is the list of your bookings by your event name")  
 print("(If nothing shows up, that means you haven't made any bookings yet)")  
 x = 1  
 while True:  
 try:  
 if type(user\_info\_database[user\_index + 6 + x]) is dict:  
 print(f"Booking {x}")  
 print(f'Event Name: {user\_info\_database[user\_index + 6 + x]["event\_name"]}')  
 print("")  
 x += 1  
 else:  
 break  
 except IndexError:  
 break  
  
 while True:  
 option = input("Select a booking to edit by typing in the booking number (Press enter to go back): ")  
 if len(option) == 0:  
 view\_edit\_delete\_bookings(user\_index)  
 return  
 elif option.isdigit():  
 try:  
 if type(user\_info\_database[user\_index + 6 + int(option)]) is dict:  
 break  
 else:  
 print("Please select a valid option!")  
 except IndexError:  
 print("Please select a valid option!")  
 else:  
 print("Please select a valid option!")  
  
 hall\_id = user\_info\_database[user\_index + 6 + int(option)]["hall\_id"]  
 hall\_index = booked\_halls.index(hall\_id)  
  
 print(f"You are now editing booking {int(option)}")  
 event\_name = input("Please enter your new event name (Press enter to keep current): ")  
 if len(event\_name) < 1:  
 event\_name = user\_info\_database[user\_index + 6 + int(option)]["event\_name"]  
  
 event\_description = input("Please enter your event description: ")  
 if len(event\_description) < 1:  
 event\_description = user\_info\_database[user\_index + 6 + int(option)]["event\_description"]  
  
 while True:  
 number\_of\_pax = input("Please enter your new number of pax (Press enter to keep current): ")  
 if len(number\_of\_pax) < 1:  
 number\_of\_pax = user\_info\_database[user\_index + 6 + int(option)]["number\_of\_pax"]  
 break  
 else:  
 if number\_of\_pax.isdigit():  
 if int(number\_of\_pax) > int(booked\_halls[hall\_index + 3]):  
 print(f"That amount is too big! Maximum amount of pax is {booked\_halls[hall\_index + 3]}")  
 else:  
 break  
 else:  
 print("Please enter a valid number!")  
  
 while True:  
 date = input("Please enter your new date of rental (Press enter to keep current) (DD/MM/YYYY): ")  
 if len(date) < 1:  
 date = user\_info\_database[user\_index + 6 + int(option)]["date"]  
 break  
 elif not valid\_date\_format(date):  
 print("Invalid date! Please re-enter.")  
 else:  
 break  
  
 while True:  
 time = input("Please enter your new time of rental (hh:mm am/pm): ")  
 if len(time) < 1:  
 time = user\_info\_database[user\_index + 6 + int(option)]["time"]  
 break  
 elif valtime(time) == False:  
 print("Invalid time or incorrect format! Please re-enter")  
 else:  
 break  
  
 while True:  
 payment\_price = input("Please enter your new payment price (Press enter to keep current): $")  
 if len(payment\_price) < 1:  
 payment\_price = user\_info\_database[user\_index + 6 + int(option)]["payment\_price"]  
 break  
 else:  
 if payment\_price.isdigit():  
 if int(payment\_price) < int(booked\_halls[hall\_index + 4]):  
 print(f"That is not enough! The rate price per day is ${booked\_halls[hall\_index + 4]}!")  
 else:  
 break  
 else:  
 print("Please enter a valid number!")  
  
 user\_info\_database[user\_index + 6 + int(option)]["event\_name"] = event\_name  
 user\_info\_database[user\_index + 6 + int(option)]["event\_description"] = event\_description  
 user\_info\_database[user\_index + 6 + int(option)]["number\_of\_pax"] = number\_of\_pax  
 user\_info\_database[user\_index + 6 + int(option)]["date"] = date  
 user\_info\_database[user\_index + 6 + int(option)]["time"] = time  
 user\_info\_database[user\_index + 6 + int(option)]["payment\_price"] = payment\_price  
  
 print("Booking information successfully updated! Here's your latest booking information:")  
 print(f"Booking {int(option)}")  
 print(f'Hall ID: {user\_info\_database[user\_index + 6 + int(option)]["hall\_id"]}')  
 print(f'Event Name: {user\_info\_database[user\_index + 6 + int(option)]["event\_name"]}')  
 print(f'Event Description: {user\_info\_database[user\_index + 6 + int(option)]["event\_description"]}')  
 print(f'Number of Pax: {user\_info\_database[user\_index + 6 + int(option)]["number\_of\_pax"]}')  
 print(f'Date: {user\_info\_database[user\_index + 6 + int(option)]["date"]}')  
 print(f'Time: {user\_info\_database[user\_index + 6 + int(option)]["time"]}')  
 print(f'Payment Price: ${user\_info\_database[user\_index + 6 + int(option)]["payment\_price"]}')  
  
 print("Returning to view/edit/delete bookings page...")  
 view\_edit\_delete\_bookings(user\_index)

The function **edit\_a\_booking(user\_index)** allows the user to:

* View all of user’s booked halls
* Selects a booked hall
* Display all current information from the database
* Able to change/add new information to the hall
* Updates the database
* Print the new updated information from the database
* Returns user to the view/edit/delete bookings interface

### 3.1.20 Line 669-721

def delete\_a\_booking(user\_index):  
 print("==Delete a booking==")  
 print("Here is the list of your bookings by your event name")  
 print("(If nothing shows up, that means you haven't made any bookings yet)")  
 x = 1  
 while True:  
 try:  
 if type(user\_info\_database[user\_index + 6 + x]) is dict:  
 print(f"Booking {x}")  
 print(f'Event Name: {user\_info\_database[user\_index + 6 + x]["event\_name"]}')  
 print("")  
 x += 1  
 else:  
 break  
 except IndexError:  
 break  
  
 while True:  
 option = input("Select a booking to delete by typing in the booking number (Press enter to go back): ")  
 if len(option) == 0:  
 view\_edit\_delete\_bookings(user\_index)  
 return  
 elif option.isdigit():  
 try:  
 if type(user\_info\_database[user\_index + 6 + int(option)]) is dict:  
 hall\_id = user\_info\_database[user\_index + 6 + int(option)]["hall\_id"]  
 hall\_index = booked\_halls.index(hall\_id)  
  
 available\_hall\_ids.append(booked\_halls[hall\_index])  
 available\_halls.append(booked\_halls[hall\_index])  
 available\_halls.append(booked\_halls[hall\_index + 1])  
 available\_halls.append(booked\_halls[hall\_index + 2])  
 available\_halls.append(booked\_halls[hall\_index + 3])  
 available\_halls.append(booked\_halls[hall\_index + 4])  
  
 user\_info\_database.remove(user\_info\_database[user\_index + 6 + int(option)])  
  
 booked\_hall\_ids.remove(booked\_halls[hall\_index])  
 i = 0  
 while i < 5:  
 booked\_halls.remove(booked\_halls[hall\_index])  
 i += 1  
  
 print("Booking successfully deleted!")  
 print("Returning to view/edit/delete bookings page...")  
 view\_edit\_delete\_bookings(user\_index)  
 break  
 else:  
 print("Please select a valid option!")  
 except IndexError:  
 print("Please select a valid option!")  
 else:  
 print("Please select a valid option!")

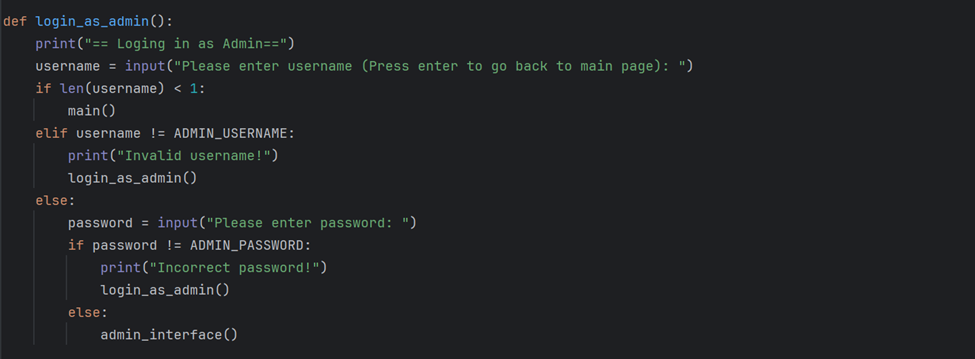
The function **delete\_a\_booking(user\_index)** allows the user to:

* View all of user’s bookings and their information from the database
* Select a booking
* Removes all information of the booking from the database
* Returns user to view/edit/delete bookings interface

## **3.2 Admin Source Code**

### 3.2.1 Line 724-738

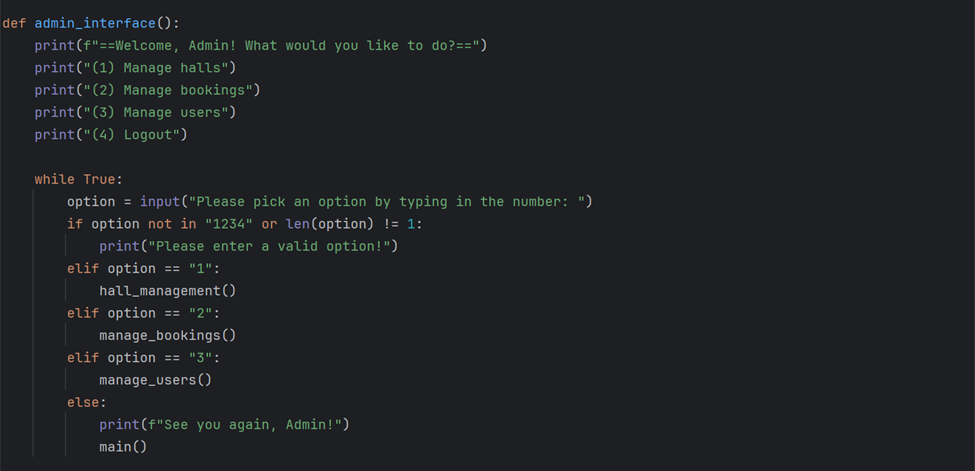
**log\_in\_as\_admin() Function**



* The function above will ask the admin to log in with the admin username and password.
* If both credentials are as the one stored, the admin will be directed to the admin interface.
* In case the credentials provided do match, the function calls itself.
* Before logging in, the admin can choose to go back to **main().**

### 3.2.2 Line 741-760

**admin\_interface() Procedure**



The menu above lists the functionalities of the admin :

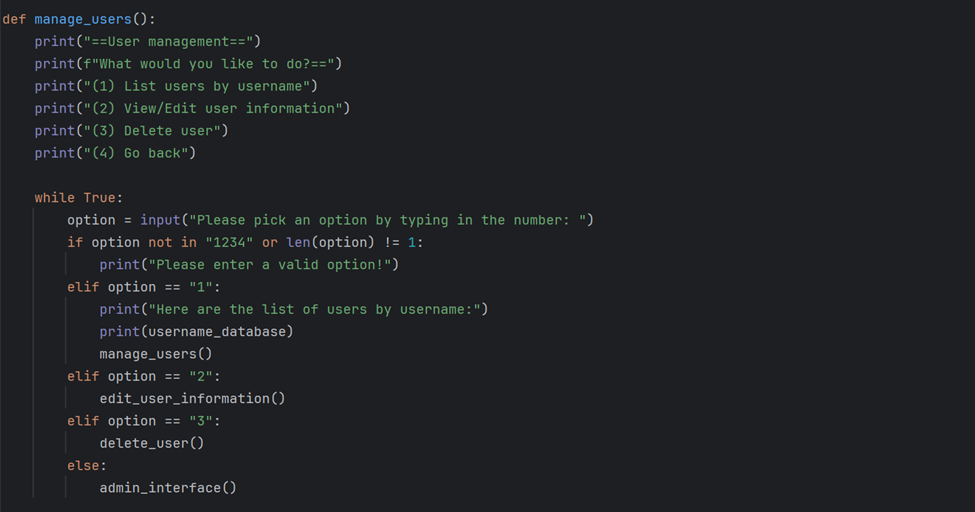
1. Manage Halls
2. Manage bookings
3. Manage Users

And the admin can choose what he/she wants to do or enter “4” to log out and return to the main page.

Only after successfully logging in, the admin can call the above functions related to the management of the Online Booking system.

### 3.2.3 Line 763-784

**manage\_users() Procedure**



The procedure above is executed when the admin has entered “3” in **admin\_interface().** The menu has options:

1. List user by username: Prints a list of usernames
2. View/Edit user information
3. Delete User

The admin can choose to edit or delete a user. After that, the admin will return to the **manage\_users()** interface and can further choose to edit or delete more users. They can also go back to the admin page by entering “4”.

### 3.2.4 Line 787-868

**edit\_user\_information() Procedure**

A computer screen shot of a program code

Description automatically generatedA computer screen with text on it

Description automatically generated

A computer screen shot of a program code

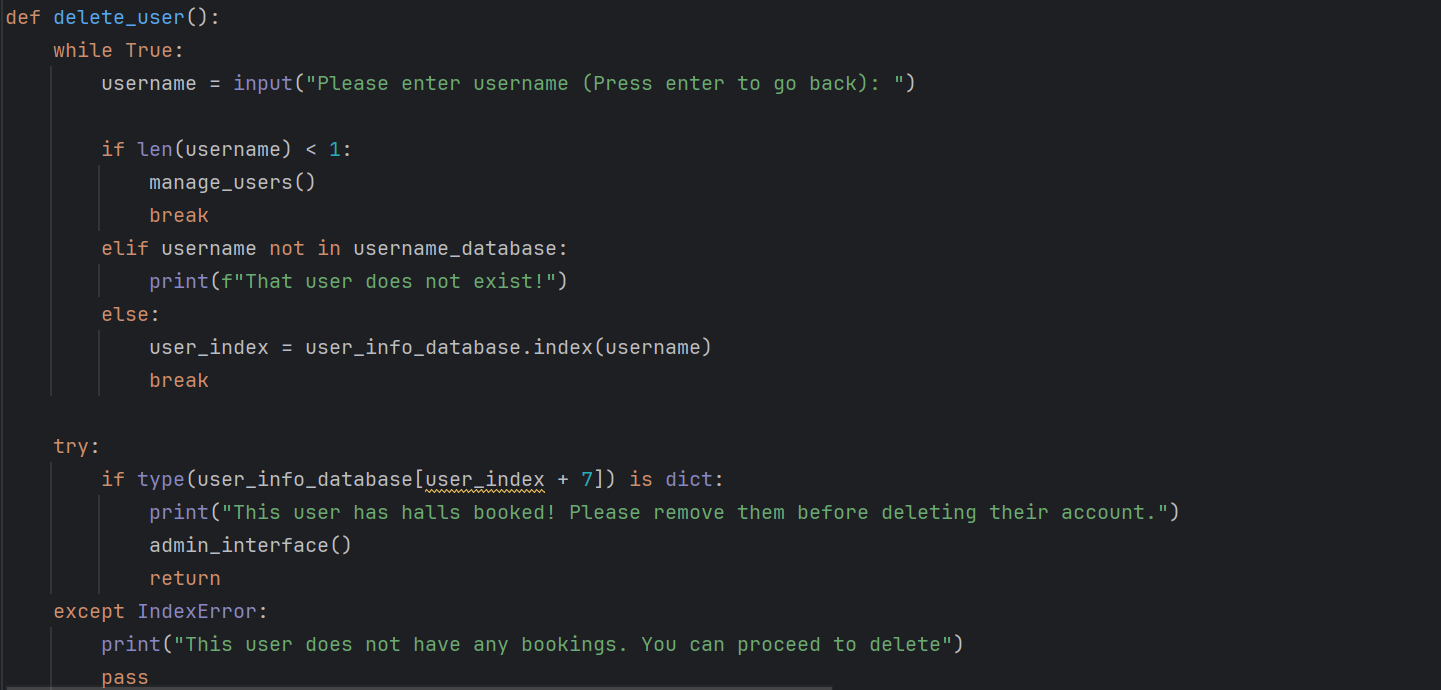
Description automatically generatedA screen shot of a computer code

Description automatically generated

* The procedure above prompts the admin to enter the username of the user they want to edit.
* It checks if the user exists and prompts the admin to re-enter if no such user exists.
* If the username exists, the admin will then enter a new first name and last name or they may choose to keep the current ones stored.
* Any new date of birth and contact number entered will be validated by functions **valid\_date\_format()** and **valid\_contact()** respectively.
* The record of the user is then updated and displayed.

### 3.2.5 Line 871-909

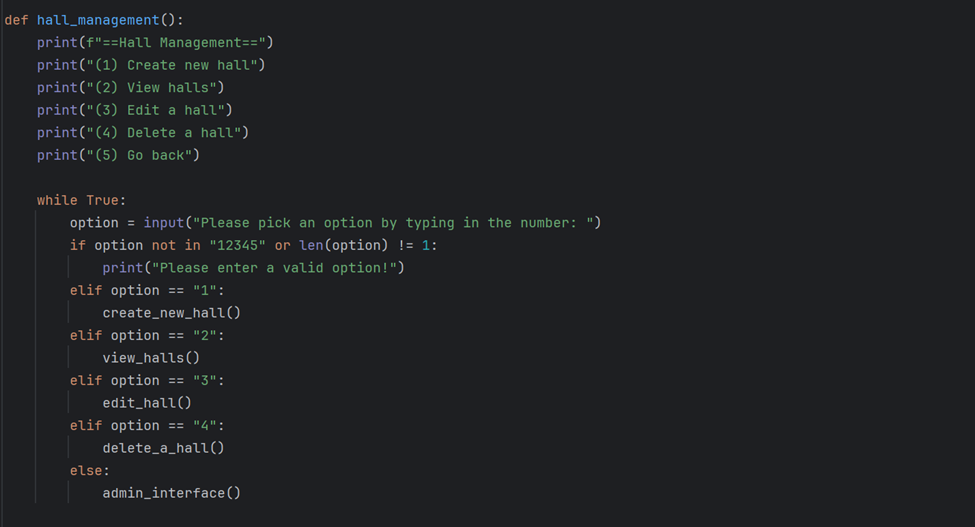
**delete\_user() Function**



* Inside the While True loop, it prompts the user to enter a username and checks the length of the input.
* Pressing enter allows the admin to go back by calling the **manage\_users()** function.
* If the username entered is not in **username\_database[]**, it prints a message stating that the user does not exist.
* Otherwise, it finds the index of the username in **user\_info\_database[]** and breaks out of the loop.
* It then checks whether the user has halls booked and the admin is notified.
* If the user has halls, it prints a message, returns to the admin interface where the admin must call **remove\_booked\_hall()** through **manage\_bookings()** in admin page.
* If the user doesn't have halls booked, the function enters a second loop where it asks the admin for confirmation to delete the account. If the admin enters 'y' (yes), it removes the user information from both **username\_database[]** and **user\_info\_database[]** using a For loop.
* It then prints a success message, returns to the user management page (**manage\_users()**). The admin can enter ‘n’ to return to the user management page without deleting the account.

### 3.2.6 Line 912-933

**hall\_management() Procedure**



The procedure above starts with a menu that lists out the functionalities of the admin concerning management of halls. It prompts the user to pick an option by typing in a number. The loop continues until the admin enters a valid input.

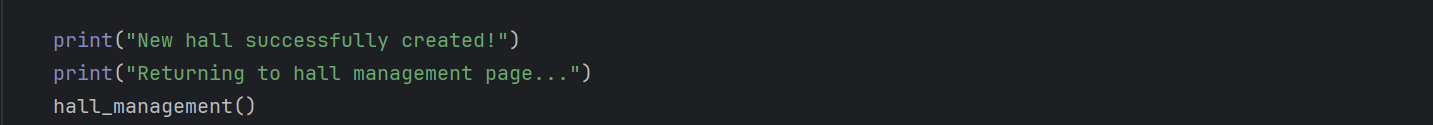
* + "1" to call the **create\_new\_hall()**.
  + "2" to call the **view\_halls()**.
  + "3" to call the **edit\_hall()**.
  + "4" to call the **delete\_a\_hall()**.
  + "5"to call the **admin\_interface()**.

### 3.2.7 Line 936-986

**create\_new\_hall() Function**

A computer screen with many colorful text

Description automatically generated with medium confidenceA computer screen shot of a black screen

Description automatically generated

When the admin enters “2” in **hall\_management(),** the **create\_new\_hall()** function is called. The admin is asked to input information about the hall they want to create including:

* Hall ID
* Hall Name
* Hall Description
* Hall capacity
* The daily price rate of the hall
* Validation process for each input ensures unique hall IDS, numeric values for price and capacity and no null values are created for name and description of halls.
* Then, the new hall is added to **available\_hall\_ids[]** and **available\_hall[]** and a message that the new hall has been successfully created is displayed.
* The function then calls **hall\_management**() to allow the further to choose a new option or to return to **admin\_interface()**.
* Before the admin logs out, the new records created will be added to the respective text files in the **main()** procedure.

### 3.2.8 Line 989-1018

**view\_halls() Function**

A computer screen shot of a program code

Description automatically generatedA computer code on a black background

Description automatically generated

* The **view\_halls()** function allows the admin to search for a hall and its information.
* In a While True loop, the function prompts the admin for a hall ID.
* If the hall has not yet been booked, it displays all the information on the hall with its Status set to ‘Available’ and if the hall id is found in list **booked\_hall\_ids[],** it displays “Booked” along with the hall information.
* If no hall ID is found, the function tells the user that the hall they are looking for does not exist and the admin can press enter instead of entering a hall ID to break the loop and go back to **hall\_management()** function.

### 3.2.9 Line 1021-1151

**edit\_hall() Function**

A computer screen shot of a program

Description automatically generatedA computer screen shot of a program code

Description automatically generated

A computer screen shot of a program code

Description automatically generatedA computer screen shot of a program code

Description automatically generatedA computer screen with many white and green text

Description automatically generated

A screen shot of a computer code

Description automatically generatedA computer screen shot of a program code

Description automatically generated

* The **edit\_hall()** function allows administrators to modify information about existing halls in the hall management system.
* It prints all available and booked halls by their respective IDs.
* In a while True loop, the function prompts the admin to enter the ID of the hall they want to update.
* The function will look up the hall ID entered and will not break until an existing ID is entered or if the user presses Enter to return to **hall\_management()** function.
* If the entered hall ID is found in the list of available halls, it displays the current information of the hall and prompts the user to press Enter to edit or any other key to go back.
* If the user chooses to edit, the function prompts an input for new information for the hall, validates it, updates the **available\_hall\_ids[]** and **available\_halls[].**
* If the hall is found in the list of booked halls, the relevant lists are then updated.
* The function then prints the updated information and returns to **hall management().**

### 3.2.10 Line 1154-1178

**delete\_a\_hall() Function**

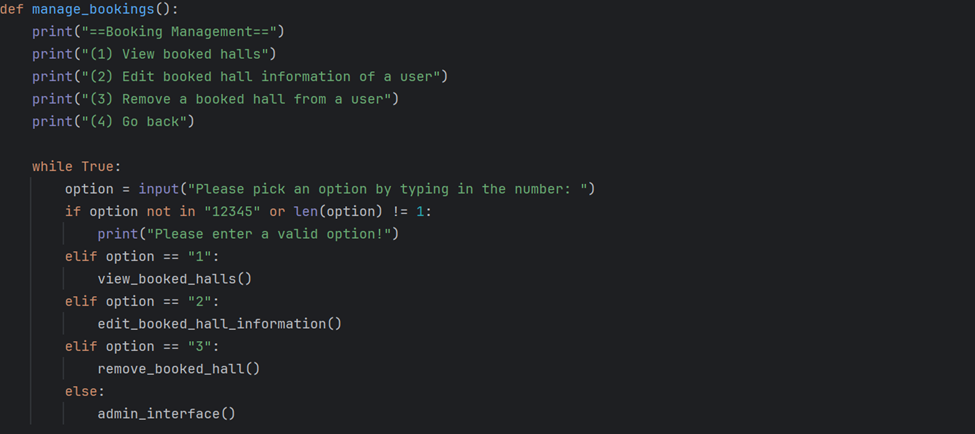
A computer screen shot of a program code

Description automatically generated

* The **delete\_a\_hall()** function allows to delete the current halls that have not been booked yet.
* If the admin wants to delete a booked hall, they must first delete the bookings made for that hall.
* In a while True loop, the admin is asked to input the hall ID they want to delete after displaying all halls that are in the **available\_hall\_ids[]** list.
* The admin can press Enter to go back to **hall\_management()** function.
* If the hall ID entered is found in the list of available halls, the function retrieves the index of that hall and uses it to delete the hall ID from **available\_hall\_ids[]** and the information stored for that hall in **available\_halls[].**
* The function will print a success message and return to **hall\_management()** function.

### 3.2.11 Line 1181-1199

**manage\_bookings() Procedure**



The **manage\_bookings()** function is called if the admin enters “2” in the admin\_interface() function. It is a menu driven function that gives access to more functions responsible for managing booked halls. It allows the admin to:

1. Search and view booked halls
2. Edit the booked hall information
3. Remove a booking from a user (it is required before the deleting a hall from the system)
4. Go back to the admin\_interface()

The while True loop is used to prompt the admin to enter a valid option.

### 3.2.12 Line 1202-1230

**view\_booked\_halls() Procedure**

A computer screen shot of a program code

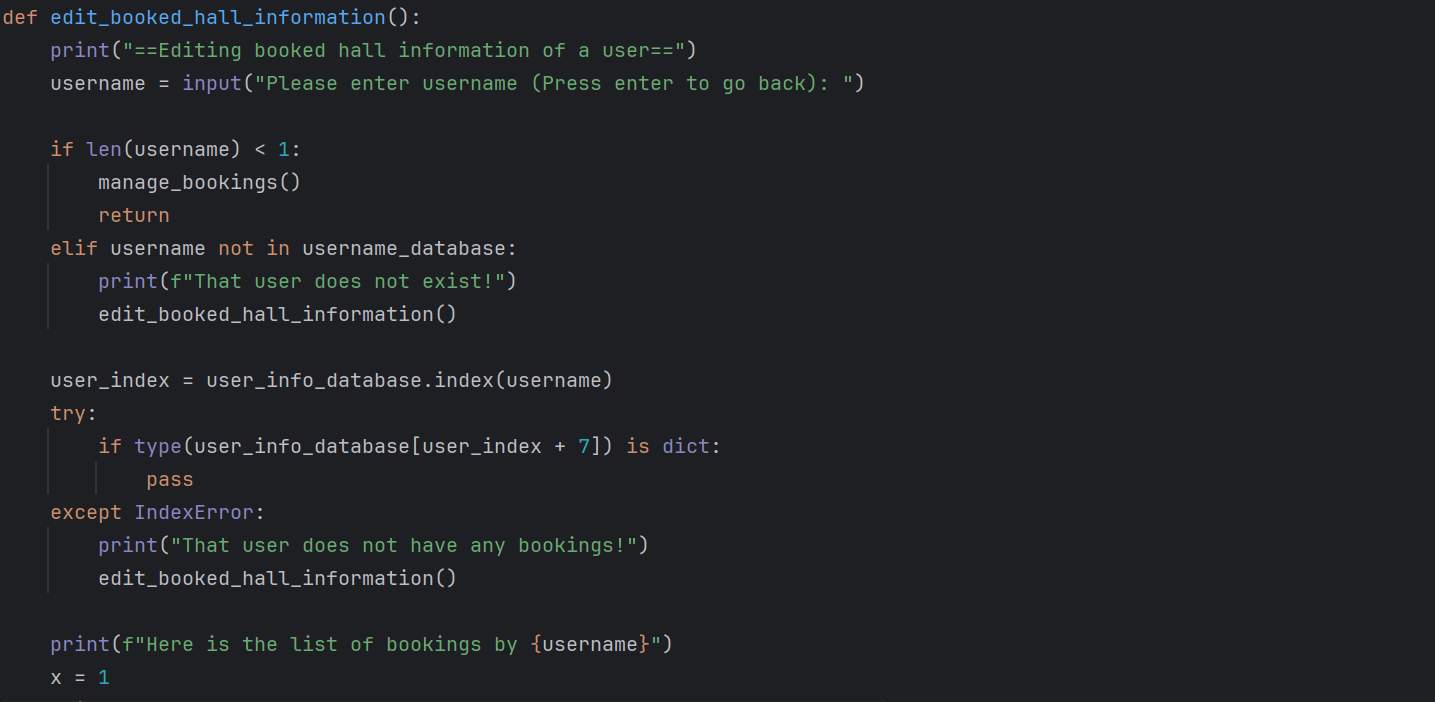
Description automatically generatedA black rectangle with white text

Description automatically generated

* The **view\_booked\_halls()** function will display all information on booked halls in a hall management system.
* It first prints the IDs of all booked halls.
* The function then enters a loop that iterates through each user in the **username\_database[]** until an IndexError occurs that is trapped by an exception handling.
* The IndexError means that there is no more bookings that can be retrieved for this user.
* For each user, it retrieves the index in the **user\_info\_database[]** and iterates through each booking stored as a dictionary starting from index 7.
* The loop will print information for each booking (if any), including the username, hall ID, event name, event description, number of pax, date, time, and payment price.
* After the loop, the function will call manage\_bookings() to return to booking management interface.

### 3.2.13 Line 1233-1350

**edit\_booked\_hall\_information() Function**

A computer screen shot of a code

Description automatically generated

A computer screen with text on it

Description automatically generated

A computer screen with text

Description automatically generatedA computer code on a black background

Description automatically generatedA computer screen with text

Description automatically generatedA screen shot of a computer code

Description automatically generated

A computer screen with colorful text

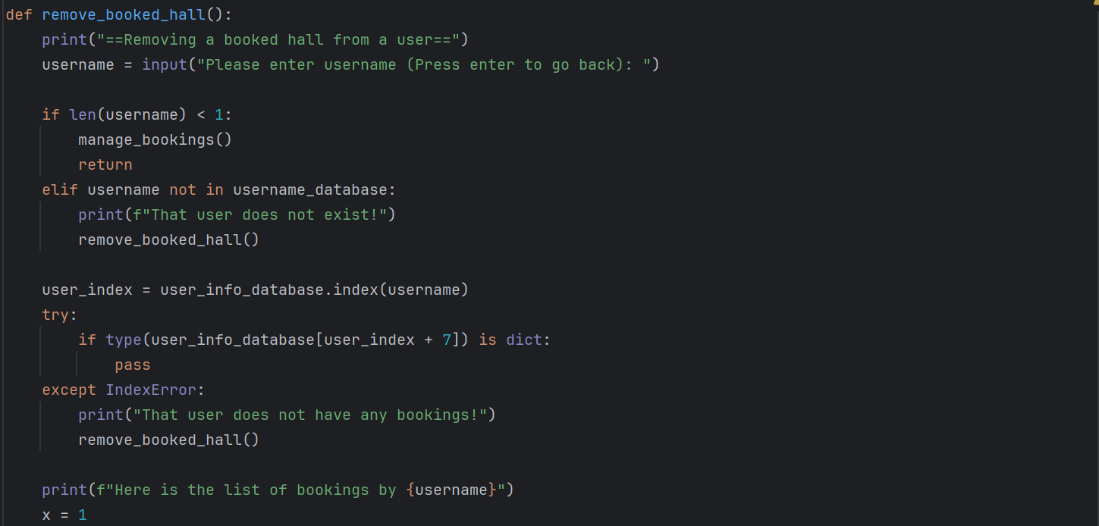
Description automatically generated

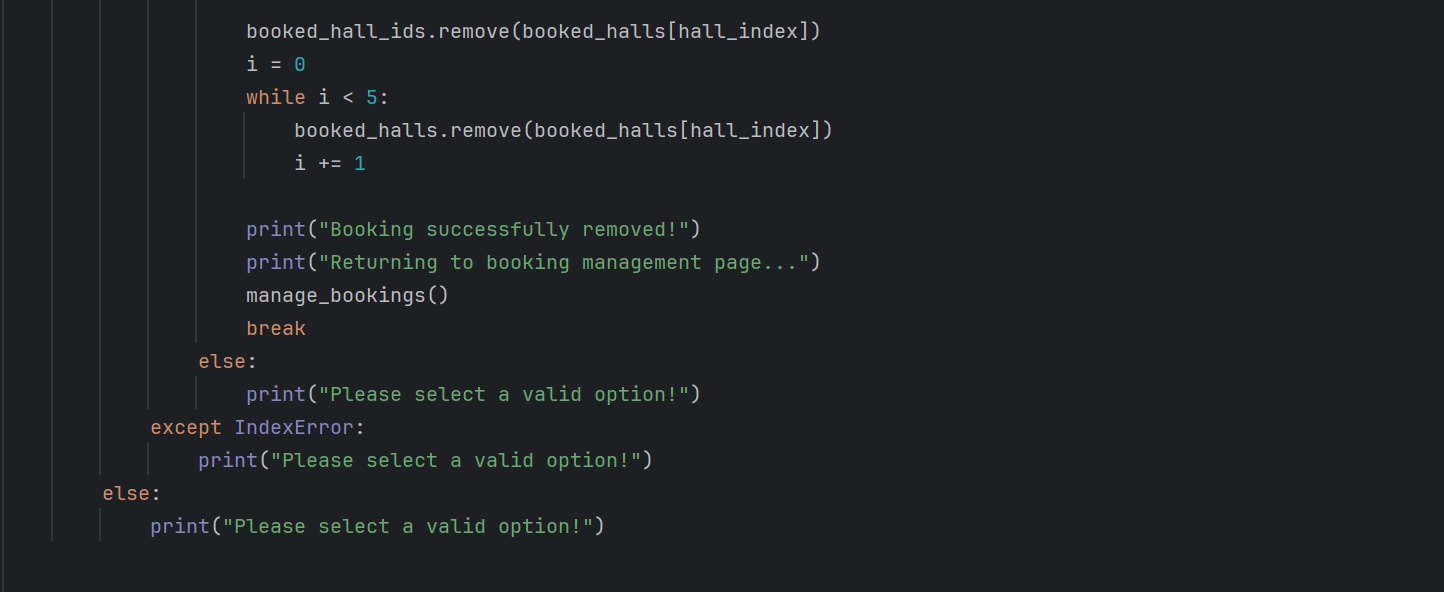
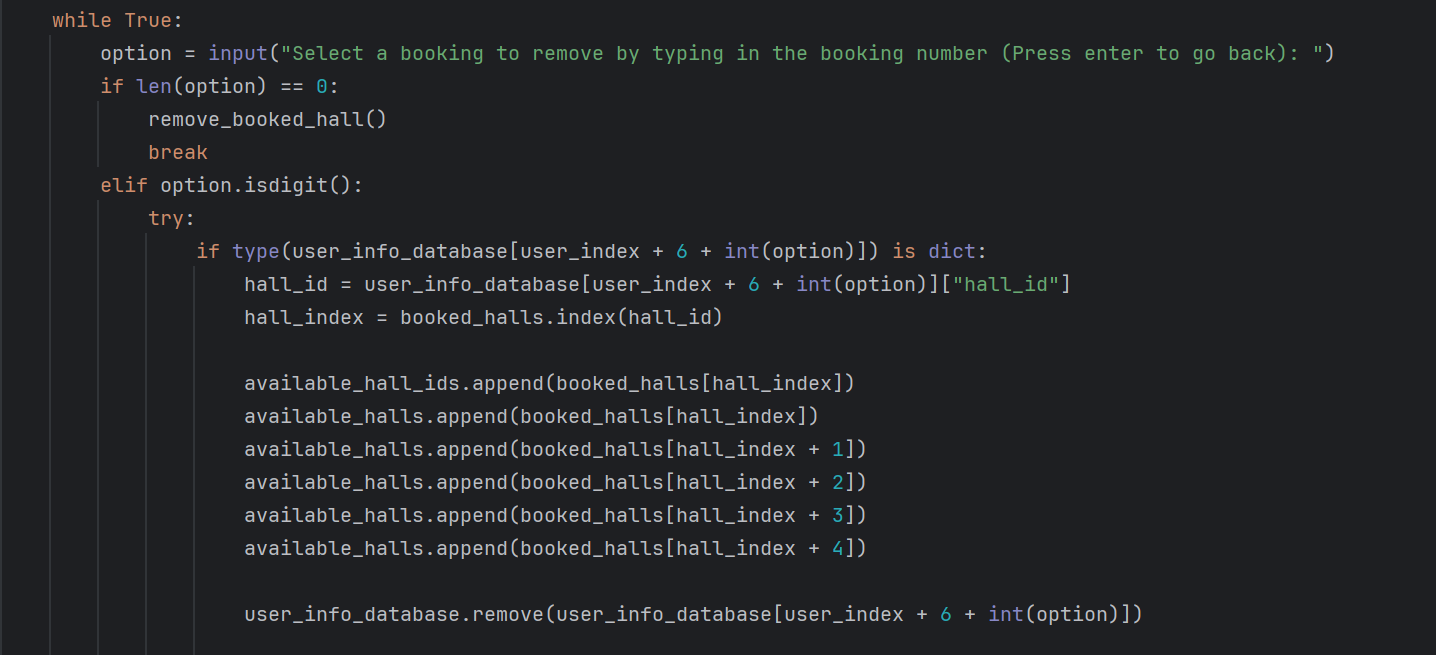
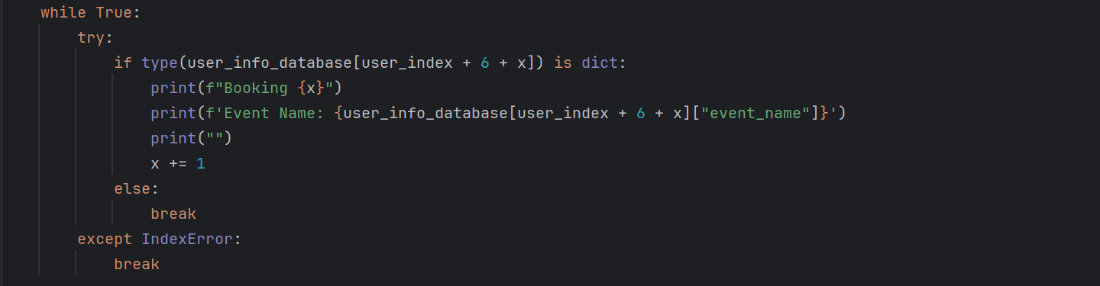
* The **edit\_booked\_hall\_information()** function will allow the admin to modify any information regarding a hall that has been booked by a user.
* It first prompts the admin to enter the username whose bookings they will edit.
* Pressing enter will call the **manage\_bookings()** function.
* The function will then look for the username in the **username\_database[]** and it is found, it will print a message to notify the admin that the user does not exist.
* The function calls itself to prompt for a valid username.
* For a valid username entered, it checks if that user has any booking by retrieving its index in the **user\_info\_database[]** and checking if there is a dictionary stored at index 7 in that list.
* It displays the bookings if found and prompts the admin to select a booking to edit by entering its number.
* Pressing enter will again call the **manage\_bookings()** function in case the admin chooses not to edit.
* For a valid booking entered, the function prompts for new information, validates it and updates the relevant fields **in user\_info\_database[].** Validate functions including **valid\_date\_format()** and **valtime()** are used to ensure a correct format for date and time.
* It then return to the booking management page.

-While True loops are used to ensure valid inputs are provided by the admin.

### 3.2.14 Line 1353-1421

**remove\_booked\_hall\_() Function**

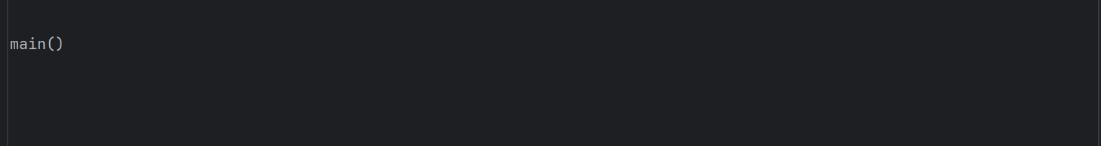




* The **remove\_booked\_hall()** function allows the admin to remove a booked hall from a user’s list of bookings.
* It prompts the user to enter a valid username.
* Pressing enter will call the **manage\_bookings()** function to return to booking management page.
* If the username entered does not exist, the admin is notified, and the function calls itself.
* The function then checks if this user has any bookings.
* If not, the function calls the **manage\_bookings()** function to return to the booking management page.
* Else, it prints all the bookings made by the user and prompts the admin to select a booking by entering the booking number.
* The admin can instead press enter to return to start of the function to choose another booking or to return to **manage\_bookings().**
* Using the booking number, the function retrieves the hall ID from **user\_info\_database[].**
* It then retrieves the hall index of that from **booked\_halls[]** and append the hall ID back to **available\_hall\_ids[] and available\_halls[].**
* The function then removes the booking information in **user\_info\_database[]** and the hall ID from **booked\_hall\_ids[] and booked\_halls[].**
* The function prints a successful message and returns to **manage\_bookings()** function.
* An Except IndexError is used when there are no bookings stored for that user.
* If and Else ifs and a while True loop are used to ensure the admin enters a valid option.

### 3.2.15 Line 1424

**Calling of** **main()**



# **4.0 SCREENSHOTS OF SAMPLE INPUT/OUTPUT AND EXPLANATION**

## **5.1 Sign up as User**

**A screen shot of a computer

Description automatically generated**

When first running the program, you’ll be greeted with the main menu interface. As a first-time user, you are to first sign up to create a user profile, thus type 1 and enter.

A black background with white text

Description automatically generated

User is than requested to enter a username.

A black background with white text

Description automatically generated

Note: If username is taken, a message will appear telling to enter a different username.

A computer screen with text and numbers

Description automatically generated

User is requested to fill up more information.

A screen shot of a computer

Description automatically generated

Note: If user enters an invalid or incorrect format data, system will request to re-enter data.

A screen shot of a computer

Description automatically generated

Once sign up is complete, a successful message will be displayed, and user is brought back to the main menu interface. Now, user may select the login option.

## **5.2 Login as User**

A black screen with white text

Description automatically generated

When logging in, user is requested to enter the username and password associated to the user’s account.

A screenshot of a computer screen

Description automatically generated

Once logged in, user is welcomed and given the option of selecting any of the displayed features. User may access the feature by typing the number associating with it. Since this is a new user, user should first start by “Make a new booking”.

A screenshot of a computer

Description automatically generated

When user first starts the hall booking process, the ID of available halls will be displayed to the user. The user may type in the hall ID to view information regarding the hall, user may proceed the booking process with the selected hall by pressing “enter”, if not user may press “q” to re-select different hall.

A screen shot of a computer

Description automatically generated

User must then fill up the following information.

A screenshot of a computer screen

Description automatically generated

Once completed, a success message is displayed, and user is brought back to the user menu interface.

A screenshot of a computer

Description automatically generated

User may select option 2 to view all available halls including the user’s booked hall(s) and information.

A screen shot of a computer

Description automatically generated

User may select option 3 to view/edit/delete bookings. User is granted access to new options from here.

A screenshot of a computer program

Description automatically generated

From here if user selects option 1, all bookings made by user will be displayed alongside the information.

A screenshot of a computer

Description automatically generated

If user wishes to edit a booking, user should select option 2. User may select which hall to edit by entering the booking number on top of the “Event Name”. Note: If nothing appears, it means no bookings have been made yet.

A screen shot of a computer

Description automatically generated

Once hall is selected, user may alter/edit hall information, if user refuses to change the information user may press “enter” to skip it.

A screenshot of a computer

Description automatically generated

A success message and the new hall details are displayed. User is than brought back to the view/edit/delete page.

A screen shot of a computer

Description automatically generated

Lastly user may delete a booking by selecting option 3. User must than enter the booking number of the hall to be deleted.

A screenshot of a computer

Description automatically generated

Once deleted a success message is displayed, and user is returned to the view/edit/delete menu.

A screen shot of a computer

Description automatically generated

User may leave this page by selecting option 4, user is returned to the user menu interface.

A screenshot of a computer

Description automatically generated

From the main menu, option 4 allows users to view their account info. From here users may edit their account by pressing “enter” or “q” to return to the user main interface.

A computer screen shot of a program

Description automatically generated

If user presses “enter”, current user info is displayed, user than may enter new information or skip it by pressing “enter”.

A screenshot of a computer

Description automatically generated

Once completed, a success message is displayed alongside the updated user info. User is then returned to user menu interface.



Option 5 allows users to delete their account by selecting either y or n.

A screenshot of a computer screen

Description automatically generated

If users decides to delete their account, account deletion success message is displayed and user is returned to main menu interface. Note: If user decides to cancel, user will just be returned to the main user interface.

A screenshot of a computer program

Description automatically generated

Option 6 allows user to log out of the system. Once logged out, a goodbye message is prompted and user is returned to the main menu interface.

## **5.3 Login as Admin**

A screen shot of a computer

Description automatically generated

When running the program, the admin is greeted by the main menu interface. The admin is required to login into the program. Type 3 (Login as admin) and enter.

A black background with white text

Description automatically generated

A black screen with white text

Description automatically generated

Upon the admin logging in, it is required for the admin to enter the correct username and password. If entered incorrectly, the program will be in a loop and will prompt the admin to type the correct username and password until it’s typed properly. Once done, the program will break and proceed to the admin interface.

A screenshot of a computer screen

Description automatically generated

The admin will be greeted by the admin Interface. The admin can choose whichever options they wish to proceed to by typing a number between 1 to 4 and select enter afterwards.

A screen shot of a computer program

Description automatically generated

The admin has the choice to either create a new hall, view halls, edit a hall or even delete a hall. The admin can also return to the admin Interface page by typing 5 (Go back) as a choice.

A screen shot of a computer

Description automatically generated

When creating a new hall, the admin must fill in details of the Hall ID, Name of the hall, the hall description, number of guests which are going to attend the event. The admin has the right to decide the price rate of the hall. Once these details are filled up by the admin, the new hall has been successfully created which prompt to go back to the hall management interface for the admin to pick other options if they choose to do so.

A screenshot of a computer program

Description automatically generated

When the admin views the hall information, the available halls and booked halls are listed. The admin can choose to view one of the available halls. For example, when the admin types h01, every information about that specific hall will be displayed for the admin to view them. Once the admin is done viewing the hall information, the admin can either view another available hall information or press enter to go back to the hall management main page.

A screenshot of a computer program

Description automatically generated

When the admin wants to edit a hall, the available halls and the booked halls will be displayed for the admin to view them and choose which specific hall they want to view and edit. For example, the admin wants to edit the hall information for h02, the current information of the entire h02 will be displayed. The admin could only edit the hall name and hall description for h02. Once the admin is done editing, the new edit of the hall information of h02 will be updated in the database and will be displayed in the program and it will automatically return to the hall management page.

A black background with many small colored lines

Description automatically generated with medium confidence

If the admin wants to delete a specific hall, the admin cannot delete the hall that has already been booked by the user beforehand. The admin can only delete the hall which hasn’t been booked yet by any user. The admin can delete the available halls listed in the database. For example, the admin can delete hall h03 which will be successfully deleted immediately since the admin is in charge. Once done with deleting the hall, the admin will be redirected to the hall management page.

A screenshot of a computer program

Description automatically generated

The admin is greeted by the admin interface. The admin can choose to manage bookings. If the admin chooses to manage bookings, the admin will be redirected to the booking management interface which the admin can either view booked halls, edit a booked hall information of a user, or even remove a booked hall of the user. The admin can also choose to go back to the admin interface page back again by entering 4 (Go Back).

A computer screen with white text

Description automatically generated

When the admin views booked halls, the entire information of the user’s booked hall will be displayed for the admin to view them ranging from Username to Payment Price. Once admin is done viewing, the program will redirect back to the booking management interface page.

A computer screen shot of a program

Description automatically generated

When the admin wants to edit a booked hall which is done by the user, the admin must sign in the user’s username to edit the booked hall. It will show the list of bookings made by that specific user. So, the admin can choose which bookings the admin wants to edit. For example, the admin edits Booking 1. The admin has to edit everything from A to Z, which is from the event name, event description, number of pax. Date and time of rental and payment price. Once every new information that has been edited by the admin is completed, the new information and data will be updated and saved to the database and the new information will be displayed for the admin to be viewed.

A screenshot of a computer program

Description automatically generated

When the admin chooses option 3 from the booking management interface page, the admin is redirected to removing the user’s booked hall. The admin will have to enter the user’s username to delete the user’s bookings. The booking number and the event name will be listed by the program for the admin to view it and delete them. Once booking is successfully removed, it will return to the booking management page where the admin can choose to stay on the same page or redirect to the admin interface page.

A screenshot of a computer program

Description automatically generated

The admin is greeted by the admin interface page again. This time, the admin can choose to select option 3 to manage users. If the admin chooses to manage users, the admin will be given a choice to either list users by username, view/edit user information, delete user’s information or go back to admin interface page by entering 4 (Go Back).

A black screen with white text

Description automatically generated

In the user management Page, the admin selects and chooses number 1, which will redirect to the list of users by username. In there, the list of usernames will be displayed for the admin to view it.

A screenshot of a computer program

Description automatically generated

When the admin wants to view or edit user information, every detail saved in the database earlier by the user is displayed in the program for the admin to view them. Except for the Username, Password and email which will be retained and unchanged, The First name, last name, date of birth, contact number can be altered, modified, and edited by the admin. This can usually be done if the admin wants to edit the user’s information or if the user forgot their username and password and they want to edit or update their new information. So, overall, the latest information which was edited will be displayed in the program and in the database to be saved. Once done, the admin will be redirected to the user management page.

A screenshot of a computer

Description automatically generated

The admin selects option 3 which is to delete the user’s account and information from the database. It is essential for the admin to type the specific user account they wish to delete. The program will give a heads up on whether they want to delete the user’s account and information or not by selecting either ‘y’ or ‘n’. If the admin chooses not to delete it by selecting ‘n’, then it would redirect the admin back to the user management page.

A screenshot of a computer program

Description automatically generated

The admin selects option 3 which is to delete the user’s account and information from the database. In this scenario as shown in the image above, the admin chooses to delete the user’s account and information which prompts the admin to select ‘y’. Once the admin selects ‘y’, the users account will be permanently deleted from the database. The admin also will be redirected to the user management page.

# **5.0 CONCLUSION**

Overall, writing python code for an online hall booking system was indeed a challenge. We encountered many difficulties when writing the code, such as bugs and limitations. We had struggled to figure out solutions for the bugs encountered. We had also battled to figure out alternative ways from the limitations that prevented our program from meeting its requirements. However, in the end, we overcame them with teamwork, research and critical thinking.

Through this group assignment, I have gained knowledge about python programming such as basic functions, types of data, looping statements, as well as conditional statements. This knowledge can be useful in helping us if we want to become software engineers in the future. It will enable us to design the most effective applications, such as web development, data analysis, machine learning, artificial intelligence, and automation.

# **6.0 REFERENCES**

Bondar, S. (2023, August 2). *How to Work with JSON Data in Python | Reintech media*. Reintech.io. <https://reintech.io/blog/how-to-work-with-json-data-in-python>

Galarnyk, M. (2023, January 27). *Guide to Python Dictionary and Dictionary Methods | Built In*. Builtin.com. <https://builtin.com/data-science/python-dictionary>

Ganchev, M. (2023, April 24). *List in Python*. 365 Data Science. <https://365datascience.com/tutorials/python-tutorials/lists-python/>

Klein, B. (2023, November 8). *30. Errors and Exception Handling | Python Tutorial*. Python-Course.eu. <https://python-course.eu/python-tutorial/errors-and-exception-handling.php>

Kumar, P. (2023, May 11). *Python return Statement - AskPython*. AskPython. <https://www.askpython.com/python/python-return-statement>

Mayo, M. (2023, August 1). *Mastering Regular Expressions with Python*. KDnuggets. <https://www.kdnuggets.com/2023/08/mastering-regular-expressions-python.html>

Motiso, D. (2022, December 13). *How To Write Pseudocode (Definition, Components and Pros)*. Indeed Career Guide. <https://www.indeed.com/career-advice/career-development/pseudocode>

Novotny, J. (2023, March 9). *How to Install and Import Modules in Python 3*. Linode Guides & Tutorials. <https://www.linode.com/docs/guides/installing-and-importing-modules-in-python-3/>

Ramuglia, G. (2023, September 11). *Creating Files in Python: Step-by-Step Tutorial*. Linux Dedicated Server Blog. <https://ioflood.com/blog/python-create-file/>

# **7.0 Workload Matrix**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Student 1 Name:**  **Demetrius Koh Zihin** | **Student 2 Name: Rooshda Fatimah Joomun** | **Student 3 Name: Shrinivas Hoh** | **Student 4**  **Name: Amal DJasreel** | **Total** |
| Design (Pseudocode or Flowchart) | **25%** | **25%** | **25%** | **25%** | **100%** |
| Coding / Implementation (Python code) | **25%** | **25%** | **25%** | **25%** | **100%** |
| Documentation | **25%** | **25%** | **25%** | **25%** | **100%** |
| Demonstration | **25%** | **25%** | **25%** | **25%** | **100%** |