ICT MINI PROJECT REPORT

By:
Mervyn Chiong Jia Rong
Pratyush Kumar Pandey
Lim Jing Da

Table of contents:

1. Algorithm Design:

The project was designed using Tkinter in Python. For file handling, json files were used.

Top level flowchart:

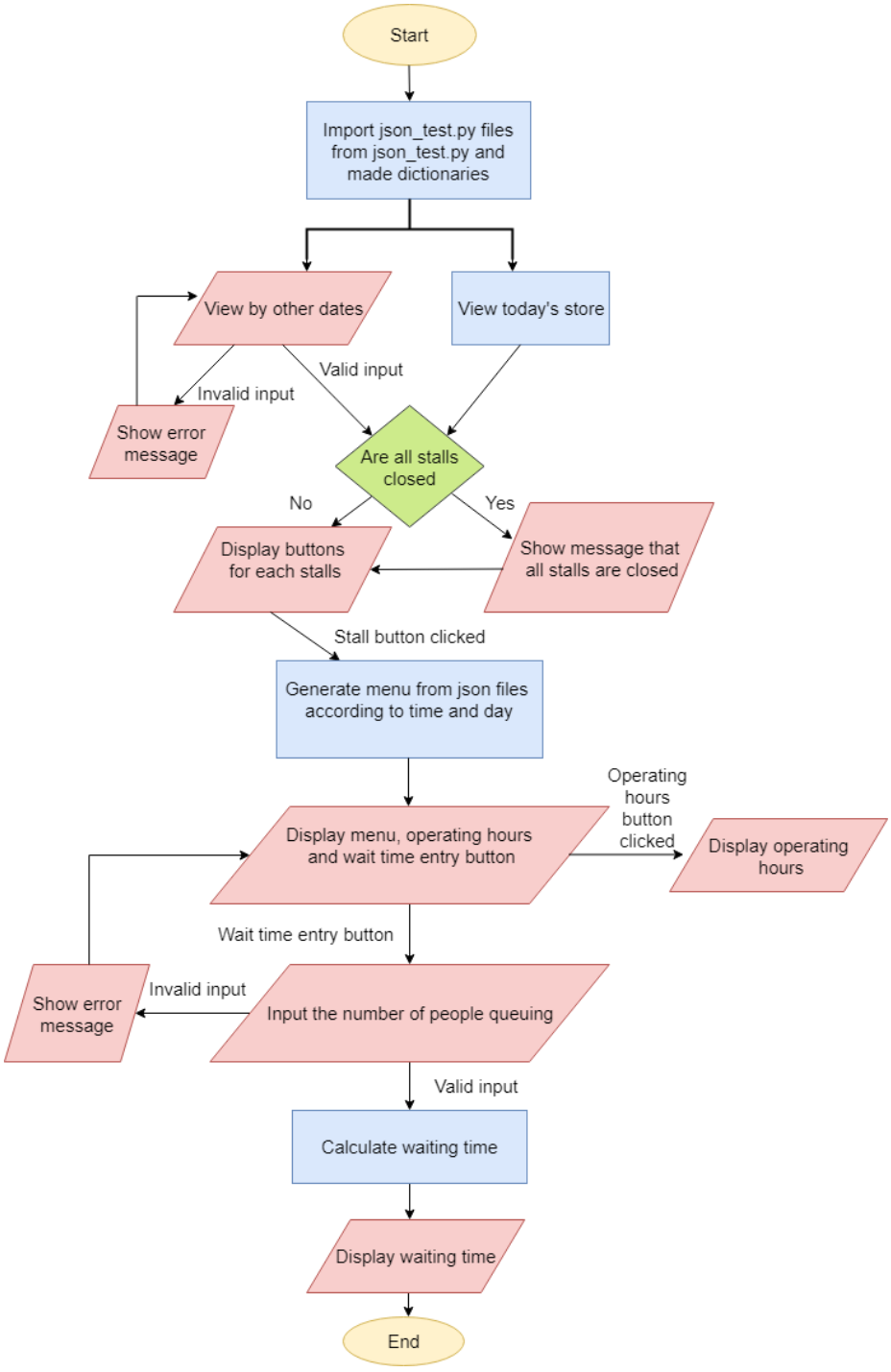


Fig. 1. The top level flowchart depicting the stages in the program

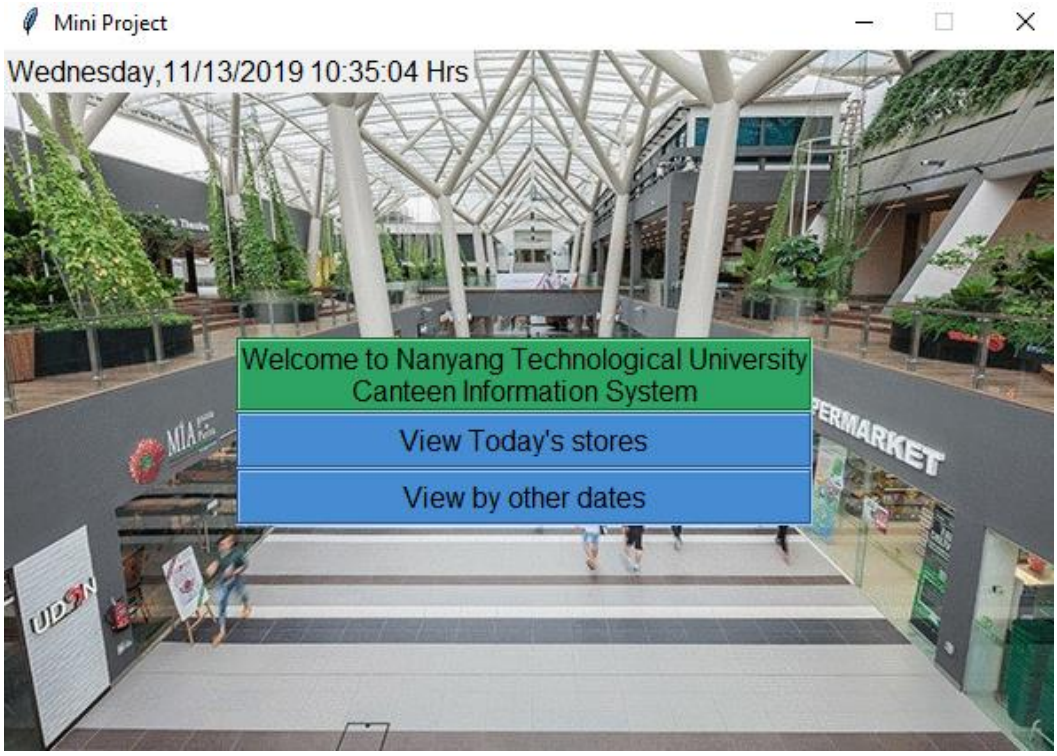


Fig. 2. Main page of the project showing the title, buttons and a digital clock

```
root = Tk()
bgimg = Image.open("back.gif")
photo1 = ImageTk.PhotoImage(bgimg)  # .resize((1000,1000), Image.ANTIALIAS))
w, h = photo1.width(), photo1.height()
root.geometry('{W}x{H}'.format(W=w, H=h))
root.resizable(False, False)
```

```
'''

Jing
The buttons for main Frame i.e button1 and button2 besides the title label was made and placed inside this main Frame
(frame)
'''
frame = Frame(LabelPhoto, width=100, height=100)
frame.place(x=130, y=160)
title = Label(frame, font=14, text="Welcome to Nanyang Technological University\nCanteen Information System",
              bg="#2DA463", fg="black", relief=GROOVE)
title.pack()
```

```
# button for custom date and time
button2 = Button(frame, font=12, text="View by other dates", bg="#458CD3", relief=GROOVE, command=lambda: openWindow2())
button2.pack(fill=X)
root.mainloop()

# 268 lines without comments
```

```
# button for current date and time
button1 = Button(frame, font=12, text="View Today's stores", bg="#458CD3", relief=GROOVE, command=lambda: openWindow1())
button1.pack(fill=X)
```

Fig.3. The code snippets for the main page

The aforementioned code snippets show the making of the main frame where a root has been defined and a label has been made to display the background image. After that all the buttons and the title label are packed in a frame which has been placed on the background image label.

Functions used:

tick()

```
'''

Pratyush
The tick() function is just used to keep the clock label ticking using the methods .after(...) and .config(...)
'''


def tick():
    timenow = (datetime.datetime.now()).strftime('%H:%M:%S')
    clock.config(text=timenow + ' Hrs')
    clock.after(200, tick)


tick()
```

Fig 4. The code snippet for tick() function

The aforementioned code snippet shows the tick() function used to keep the clock label ticking as time passes. It makes use of .after function to update the clock after every 200 milliseconds(keeping in mind the delay).

openWindow2()

```
def openWindow2():
    global root, frame
    frame.place_forget()
    f2 = Frame(LabelPhoto)
    f2.place(x=200, y=160)

    class dateandtime:
        def __init__(self, window):
            # making labels for hours and minutes
            labelhh = Label(window, font=14, text="Hours:\n(24 hr)", bg="#83574D", fg='white')
            labelmins = Label(window, font=14, text="Minutes:", bg='#83574D', fg='white')
            # using tk calendar for making DateEntry
            self.cal = DateEntry(f2, font=14, width=9, background='darkblue', foreground='white', borderwidth=1)
            self.cal.grid(row=5, column=1, columnspan=10)
            labelhh.grid(row=7, column=1)
            global yy, mm, dd, tt
            # making entries for hours and minutes
            self.hour = Entry(window, width=5, font=14)
            self.minutes = Entry(window, width=5, font=14)
            self.hour.grid(row=7, column=2)
            labelmins.grid(row=7, column=3)
            self.minutes.grid(row=7, column=4)

    user_data = dateandtime(f2)
```

Fig 5. The code snippet for openWindow2() function

The function openWindow2() functions when the button for custom date and time is clicked. Here, we use object oriented programming to store the user defined variables as instance variables of the object “user_data”. In this we make use of DateEntry class from tkcalendar for accepting the date from the user. Entry labels have been used to accept time in a 24 hour format. The function makes use of another nested function check().

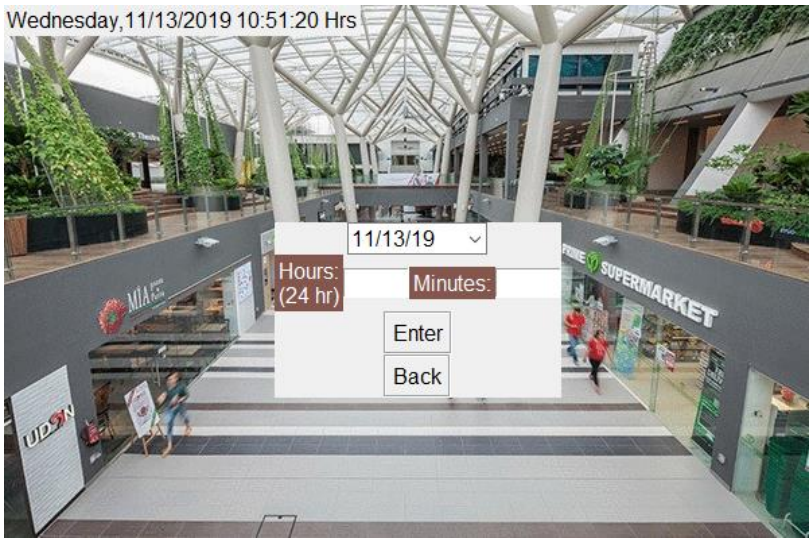


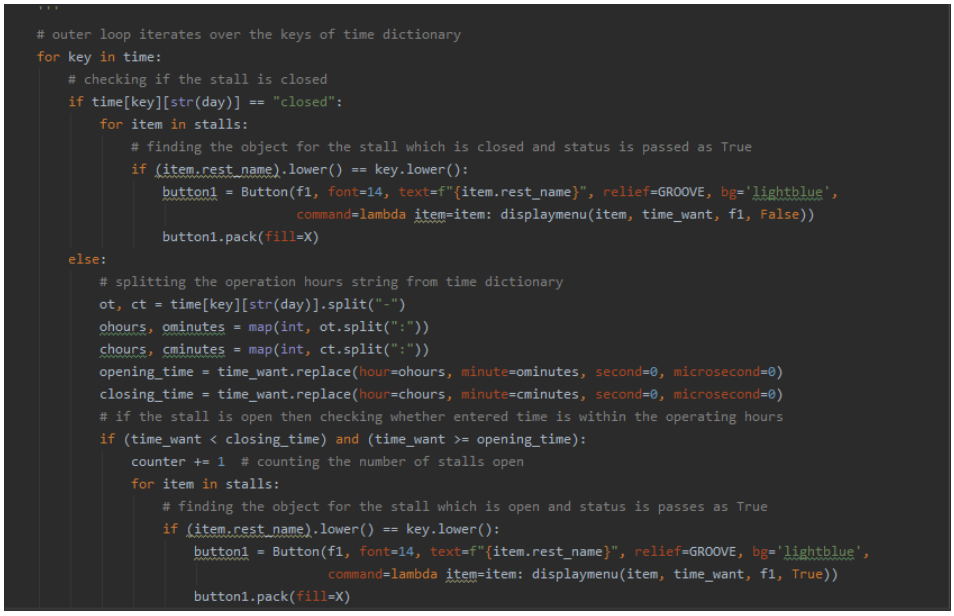
Fig 6. The custom date and time frame

openWindow1()

```
def openWindow1():
    global root, frame, weekdays
    frame.place_forget()
    f1 = Frame(LabelPhoto)
    f1.place(x=210, y=130)
    user_data = datetime.datetime.now()
    time_want = user_data
    day = time_want.weekday()
    chooseLabel = Label(f1, font=14, text='Please choose a store')
    chooseLabel.pack(fill=X)
    counter = 0
    '''

    Pratyush
    The loop design is such that the outer loop iterates over the keys of time dictionary that stores the operating
    hours of each stall and the inner loop iterates over stalls which is a list of objects of stall created in which the
    displaymenu(....) function is called. If the stall is closed then as status False is passed.
    '''
    # outer loop iterates over the keys of time dictionary
    for key in time:
        # checking if the stall is closed
        if time[key][str(day)] == "closed":
            for item in stalls:
```

Fig 7. The code snippet for frame declaration in openWindow1()

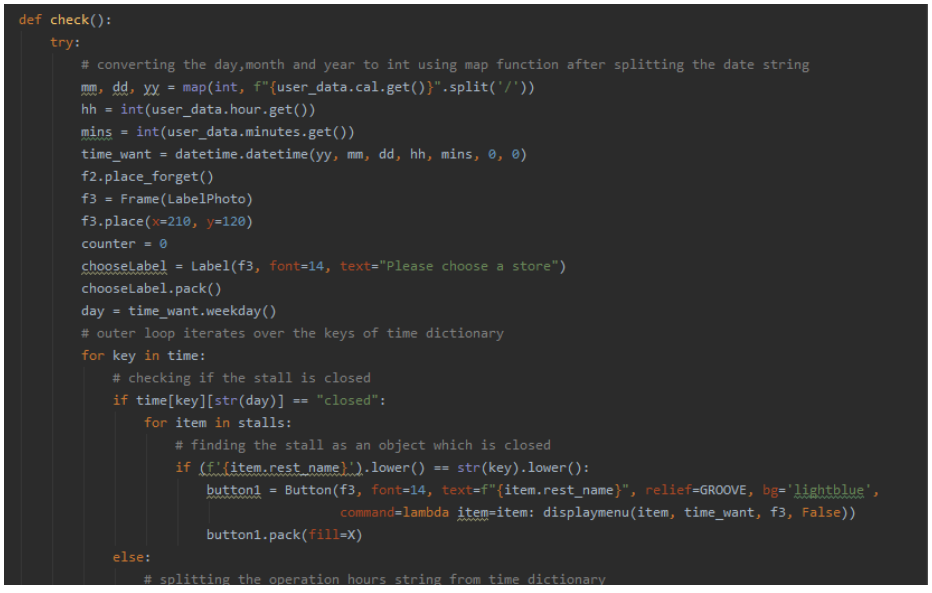


```
# outer loop iterates over the keys of time dictionary
for key in time:
    # checking if the stall is closed
    if time[key][str(day)] == "closed":
        for item in stalls:
            # finding the object for the stall which is closed and status is passed as True
            if (item.rest_name).lower() == key.lower():
                button1 = Button(f1, font=14, text=f"{item.rest_name}", relief=GROOVE, bg='lightblue',
                                 command=lambda item=item: displaymenu(item, time_want, f1, False))
                button1.pack(fill=X)
    else:
        # splitting the operation hours string from time dictionary
        ot, ct = time[key][str(day)].split("-")
        ohours, ominutes = map(int, ot.split(":"))
        chours, cminutes = map(int, ct.split(":"))
        opening_time = time_want.replace(hour=ohours, minute=ominutes, second=0, microsecond=0)
        closing_time = time_want.replace(hour=chours, minute=cminutes, second=0, microsecond=0)
        # if the stall is open then checking whether entered time is within the operating hours
        if (time_want < closing_time) and (time_want >= opening_time):
            counter += 1  # counting the number of stalls open
            for item in stalls:
                # finding the object for the stall which is open and status is passes as True
                if (item.rest_name).lower() == key.lower():
                    button1 = Button(f1, font=14, text=f"{item.rest_name}", relief=GROOVE, bg='lightblue',
                                     command=lambda item=item: displaymenu(item, time_want, f1, True))
                    button1.pack(fill=X)
```

Fig 8. The code snippet showing the loop design in openWindow1()

The function is called when the current date and time button is clicked. This function generates a frame to show the list of stalls and pass the status as “True” for the stalls that are open and “False” for the stalls which are closed through the loop design which requires two nested loop. The outer loop iterates over the keys of the time dictionary and inner loop(iterates over the list of objects created for each stall) checks for the operating hours.

check():



```
def check():
    try:
        # converting the day,month and year to int using map function after splitting the date string
        mm, dd, yy = map(int, f"{user_data.cal.get()}".split('/'))
        hh = int(user_data.hour.get())
        mins = int(user_data.minutes.get())
        time_want = datetime.datetime(yy, mm, dd, hh, mins, 0, 0)
        f2.place_forget()
        f3 = Frame(LabelPhoto)
        f3.place(x=210, y=120)
        counter = 0
        chooseLabel = Label(f3, font=14, text="Please choose a store")
        chooseLabel.pack()
        day = time_want.weekday()
        # outer loop iterates over the keys of time dictionary
        for key in time:
            # checking if the stall is closed
            if time[key][str(day)] == "closed":
                for item in stalls:
                    # finding the stall as an object which is closed
                    if (f'{item.rest_name}').lower() == str(key).lower():
                        button1 = Button(f3, font=14, text=f"{item.rest_name}", relief=GROOVE, bg='lightblue',
                                         command=lambda item=item: displaymenu(item, time_want, f3, False))
                        button1.pack(fill=X)
            else:
                # splitting the operation hours string from time dictionary
```

Fig 9. The code snippet for check() function

The check() function is called when “Enter” is pressed in the custom date and time frame. This function displays the list of stalls frame but this one works for custom date and time input. Besides, creating frame it has loop design which performs the same function as that of openWindow1().

displaymenu():

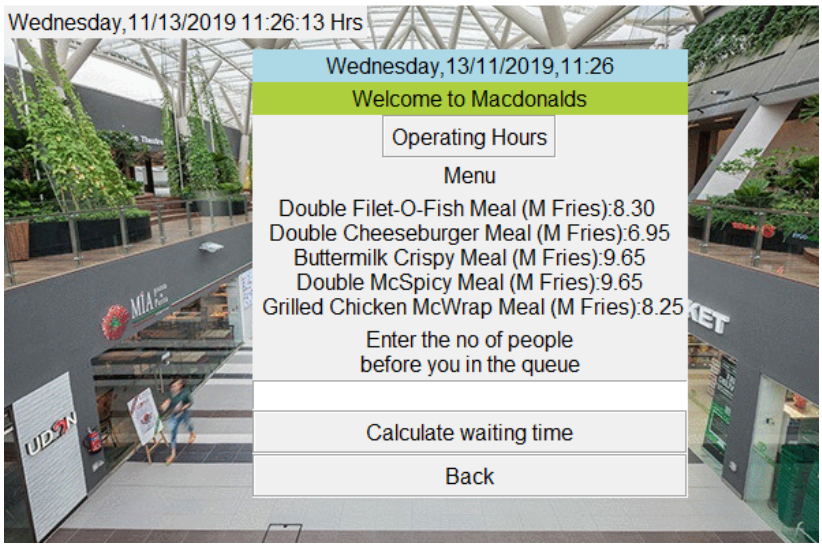


Fig 10. Display menu frame

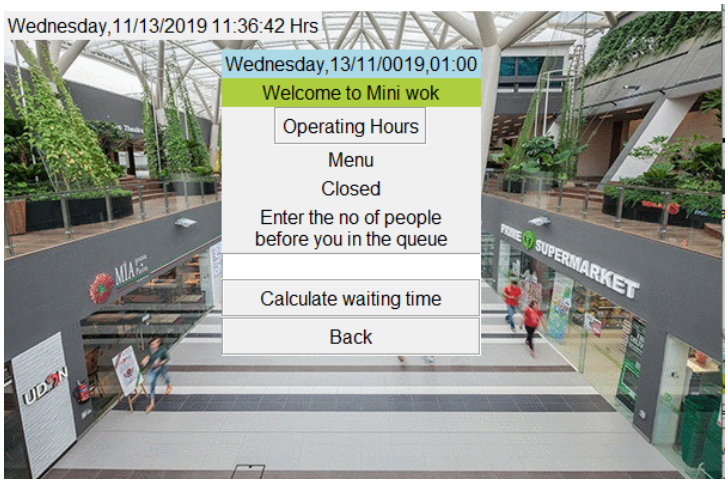


Fig 11. Display menu frame when the stall is closed

This function takes stall object, required time and the previously created frame (for the back button). It is called from openWindow1() or check() functions. Closed is displayed in the menu for stalls that are closed

```
def displaymenu(object, time_want, frame, status):
    global root, weekdays
    frame.place_forget()
    f1 = Frame(LabelPhoto)
    f1.place(x=180, y=30)
    print(object.rest_name)
    datelabel = Label(f1, font=14, text=f"{weekdays[int(time_want.weekday())]}," + time_want.strftime(
        "%d/%m/%Y") + ',' + time_want.strftime('%H:%M'), bg="lightblue")
    datelabel.pack(fill=X)
    label1 = Label(f1, font=14, text=f"Welcome to {object.rest_name}", bg='#AECF3D', fg='black')
    label1.pack(fill=X)
    operation = Button(f1, font=14, relief=GROOVE, text="Operating Hours", command=lambda: operating(object))
    operation.pack()
    weekday = time_want.weekday()
    menu = Label(f1, font=14, text="Menu")
    menu.pack(fill=X)
    '''

    Pratyush
    The loop checks for the time key of the weekday key in the nested dictionary for menus for each stall.The menu for
    each stall has been stored as a single string for the time key. Hence the loop first checks for the day and then the
    time and displays the menu accordingly
    '''
```

```
# if the stall is open then status is True
if status:
    for key2 in object.food[str(weekday)]:
        # splitting the different time frames and accessing the menu for that time frame when it is matched
        it, bt = key2.split("-")
        ih, im = map(int, it.split(":"))
        bh, bm = map(int, bt.split(":"))
        inner_time = time_want.replace(hour=ih, minute=im, second=0, microsecond=0)
        outer_time = time_want.replace(hour=bh, minute=bm, second=0, microsecond=0)
        # checking if the user time is within the time frame key of the menu
        if (time_want < outer_time) and (time_want >= inner_time):
            menulabel = Label(f1)
            menulabel.configure(font=14, text=f"{object.food[str(weekday)][key2]}")
            menulabel.pack(fill=X)
        else:
            continue
else:
    # if the stall is closed then closed is displayed in place of menu
    menulabel = Label(f1)
    menulabel.configure(font=14, text="Closed")
    menulabel.pack(fill=X)
LabelEntry = Label(f1, font=14, text="Enter the no of people\nbefore you in the queue")
LabelEntry.pack(fill=X)
try:
    no_of_persons = Entry(f1, font=14)
    no_of_persons.pack(fill=X)
```

Fig 12. The code snippet for displaymenu() function (for generating and displaying the menu)

The function employs a looping structure to generate the menu according to the day and time. This makes our code very dynamic as a stall can have different breakfast menus on different days or different menus on the same day. This is because the loop iterates over the menu dictionary of the stall which stores this information.

estimatewaittime():

This function is used to estimate the approximate wait time by taking the no of people in the queue and then estimate the wait time by assuming it to be 1 min 25 seconds per person. If the user input is greater than 42 then the user is advised to look for another stall with lesser waiting time.

```
def estimatewaittime(no_of_persons, frame):
    try:
        pax = int(no_of_persons.get())
        if pax < 0:
            raise ValueError
        Timesec = pax * 85
        Timemin = Timesec // 60
        Timesec = Timesec % 60
        if Timemin > 60:
            frame.place_forget()
            f1 = Frame(LabelPhoto)
            f1.place(x=210, y=160)
            approx_time2 = Label(f1, font=14, text="Waiting time is more than 1 hour\n Please look for another store")
            approx_time2.pack()
        else:
            frame.place_forget()
            f1 = Frame(LabelPhoto)
            f1.place(x=210, y=160)
            approx_time = Label(f1, font=14,
                                text=f"Approximate waiting time:     \n{Timemin} minutes and {Timesec} seconds")
            approx_time.pack()
        backbutton = Button(f1, font=14, relief=GROOVE, text="Back", command=lambda: back_button(f1, frame, 180, 30))
        backbutton.pack(fill=X)
    except:
        messagebox.showinfo("Warning", "Please enter positive integer value for no: of persons")
```

Fig 13. The code snippet for estimatewaittime() function (calculation of wait time)

operating():

This function displays the operating hours for the stall which is passed in as the parameter. It shows it in the form of a message box. It is called when the “operating hours” button is clicked.

```
def operating(object):
    string = ''
    for key in time[object.rest_name]:
        string = string + f"{weekdays[int(key)]}:{time[object.rest_name][key]} \n"
    messagebox.showinfo("Information", f"{string}")
```

Fig 14. Code snippet for operating() function (operating hours)

Outcome:

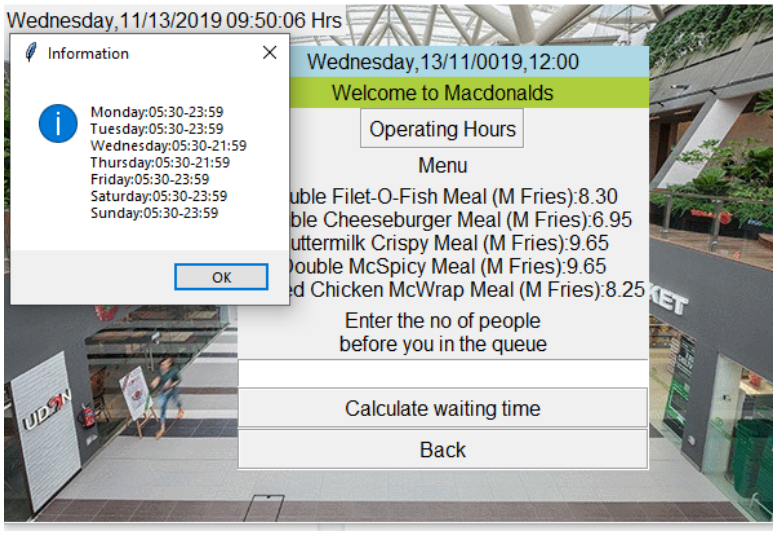


Fig 15. Message box for operating hours

back_button():

This function is called whenever the back button present in every frame is clicked. Using place_forget, the previous frame is removed and a new frame is raised on top.

```
def back_button(current, prev, X, Y):
    current.place_forget()
    prev.place(x=X, y=Y)
```

Fig 16. Code snippet for function for back button

Use of OOP:

Class Restaurants:

```
'''
Pratyush
The class Restaurants is a special class in the sense that it has parameterized constructor which assigns values to
instance variables rest_name and food. After class declaration instances of the class are created as the stalls.
'''



class Restaurants:
    def __init__(self, name, menu):
        self.rest_name = name  # instantiating variable rest_name
        self.food = menu  # instantiating variable food


'''
Pratyush
The following is the object declaration and passing of values for instance variables for rest_name and food
'''
soup = Restaurants("Soup", menu_for_soup)
mcdonalds = Restaurants("Macdonalds", menu_for_macdonalds)
miniwok = Restaurants("Mini wok", menu_for_miniwok)
chickenrice = Restaurants("Chicken rice", menu_for_chickenrice)
salad = Restaurants("Salad", menu_for_salad)
```

Fig 17. Code snippet demonstrating the use of classes

This is a class whose objects are all the different stalls. The instance variables food and menu are instantiated in a parametrised constructor. This makes the codes reusable and comprehensible.

File Handling:

```
def Stallinfos():
    try:
        with open(r"updatedtime.json", "r") as read_file:
            time = json.load(read_file)
        with open(r"souplinesdict.json", "r") as read_file:
            menu_for_soup = json.load(read_file)
        with open(r"saladlinesdict.json", "r") as read_file:
            menu_for_salad = json.load(read_file)
        with open(r"miniwokupdated.json", "r") as read_file:
            menu_for_miniwok = json.load(read_file)
        with open(r"macsupdated.json", "r") as read_file:
            menu_for_macdonalds = json.load(read_file)
        with open(r"chickenricelistdict.json", "r") as read_file:
            menu_for_chickenrice = json.load(read_file)
        return time,menu_for_soup,menu_for_salad,menu_for_miniwok,menu_for_macdonalds,menu_for_chickenrice

    except:
        print("File is not detected please change the pathway in the python code and try again")
        quit()
```

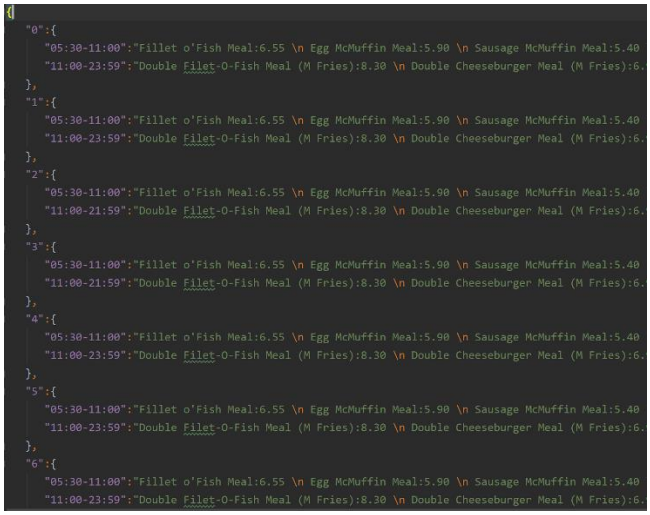


Fig 18. Json_test module and dictionaries inside the json files

The aforementioned code snippets show file handling that was done in a separate module: json_test.py. Using the function Stallinfos() the dictionaries for menu and operating hours was created. These menus had previously been stored in files with .json format.

2. Error Handling:

Faulty input in

a) No: of people queuing:

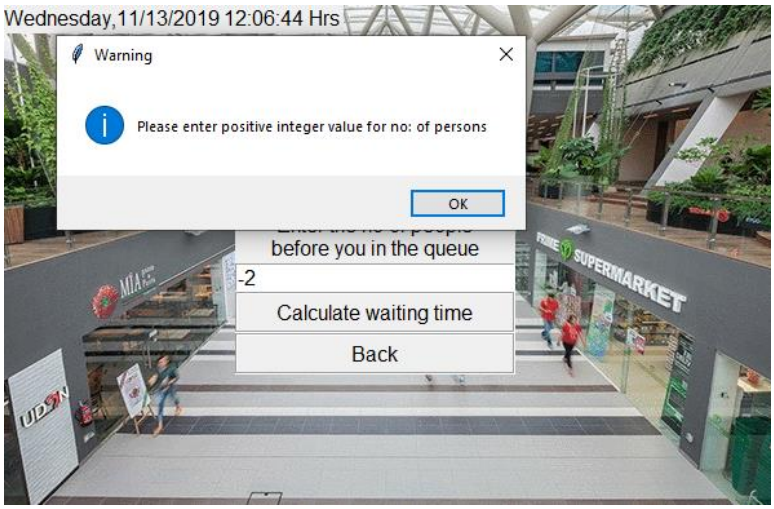


Fig 19. Frame showing the output for test case: -2 entered for pax queuing

In the function of estimatewaittime() a try and except block has been used to handle the ValueError exception that can arise due to faulty input like -2.

```
def estimatewaittime(no_of_persons, frame):
    try:
        pax = int(no_of_persons.get())
        if pax < 0:
            raise ValueError
        Timesec = pax * 85
        Timemin = Timesec // 60
        Timesec = Timesec % 60
        if Timemin > 60:
            frame.place_forget()
            f1 = Frame(LabelPhoto)
            f1.place(x=210, y=160)
            approx_time2 = Label(f1, font=14, text="Waiting time is more than 1 hour\n Please look for another store")
            approx_time2.pack()
        else:
            frame.place_forget()
            f1 = Frame(LabelPhoto)
            f1.place(x=210, y=160)
            approx_time = Label(f1, font=14,
                                text=f"Approximate waiting time:     \n{Timemin} minutes and {Timesec} seconds")
            approx_time.pack()
        backbutton = Button(f1, font=14, relief=GROOVE, text="Back", command=lambda: back_button(f1, frame, 180, 30))
        backbutton.pack(fill=X)
    except:
        messagebox.showinfo("Warning", "Please enter positive integer value for no: of persons")
```

Fig 20. Code snippet showing try except block in check()

b) Date and time:

```
def check():
    try:
        mm, dd, yy = map(int, f"{user_data.cal.get()}".split('/'))
        hh = int(user_data.hour.get())
        mins = int(user_data.minutes.get())
        time_want = datetime.datetime(yy, mm, dd, hh, mins, 0, 0)
        f2.place_forget()
        f3 = Frame(LabelPhoto)
        f3.place(x=210, y=120)
        counter = 0
        chooseLabel = Label(f3, font=14, text="Please choose a store")
        chooseLabel.pack()
        day = time_want.weekday()
        for key in time:
            if time[key][str(day)] == "closed":
                for item in stalls:
                    if item.rest_name == key:
                        button1 = Button(f3, font=14, text=f"{item.rest_name}", relief=GROOVE, bg='lightblue',
                                         command=lambda item=item: displaymenu(item, time_want, f3, False))
                        button1.pack(fill=X)
            else:
                ot, ct = time[key][str(day)].split("-")
                ohours, ominutes = map(int, ot.split(":"))
                chours, cminutes = map(int, ct.split(":"))
                opening_time = time_want.replace(hour=ohours, minute=ominutes, second=0, microsecond=0)
```

```
            ot, ct = time[key][str(day)].split("-")
            ohours, ominutes = map(int, ot.split(":"))
            chours, cminutes = map(int, ct.split(":"))
            opening_time = time_want.replace(hour=ohours, minute=ominutes, second=0, microsecond=0)
            closing_time = time_want.replace(hour=chours, minute=cminutes, second=0, microsecond=0)
            if (time_want < closing_time) and (time_want >= opening_time):
                counter += 1
                for item in range(len(stalls)):
                    if stalls[item].rest_name == key:
                        button1 = Button(f3, font=14, text=f"{stalls[item].rest_name}", relief=GROOVE,
                                         bg='lightblue',
                                         command=lambda item=item: displaymenu(stalls[item], time_want, f3,
                                                                               True))
                        button1.pack(fill=X)
            else:
                for item in stalls:
                    if item.rest_name == key:
                        button1 = Button(f3, font=14, text=f"{item.rest_name}", relief=GROOVE, bg='lightblue',
                                         command=lambda item=item: displaymenu(item, time_want, f3, False))
                        button1.pack(fill=X)
        backbutton = Button(f3, relief=GROOVE, font=14, text="Back", command=lambda: back_button(f3, f2, 200, 160))
        backbutton.pack(fill=X)
        if counter == 0:
            messagebox.showinfo("Information", " All stores are closed ")
    except:
        messagebox.showinfo("Warning", "Please enter valid date and time")
```

Fig 21. Code snippet showing try except block in estimatewaittime()

This code is to check if the input is of a real case. An example would be like 26:00 hrs which does not exist. If not, the except block is then called and request the user to redo said input. Other cases such as inputting a string type or a negative value, the exception block is also called.

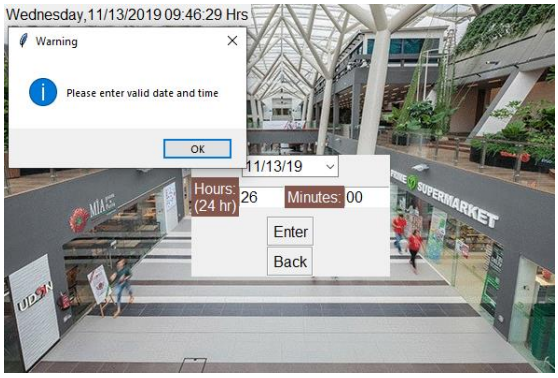


Fig 22. Frame showing the output for test case: 26 entered for Hours

c) File management:

```python
def Stallinfos():
    try:
        with open(r"updatedtime.json", "r") as read_file:
            time = json.load(read_file)
        with open(r"souplinesdict.json", "r") as read_file:
            menu_for_soup = json.load(read_file)
        with open(r"saladlinesdict.json", "r") as read_file:
            menu_for_salad = json.load(read_file)
        with open(r"miniwokupdated.json", "r") as read_file:
            menu_for_miniwok = json.load(read_file)
        with open(r"macsupdated.json", "r") as read_file:
            menu_for_macdonalds = json.load(read_file)
        with open(r"chickenricelistdict.json", "r") as read_file:
            menu_for_chickenrice = json.load(read_file)
        return time,menu_for_soup,menu_for_salad,menu_for_miniwok,menu_for_macdonalds,menu_for_chickenrice

    except:
        print("File is not detected please change the pathway in the python code and try again")
        quit()
```

Fig 23. Code for try except block while file handling

As file management can be error-prone, adding this try and except block (code snippet) will stop the whole process should the file be missing or if any other forms of error related to the file show up. Thereby, requesting the user to double-check the code and redo it should it be missing.

```
C:\Users\xjell\AppData\Local\Programs\Python\Python38-32\python.exe C:/Users/xjell/PycharmProjects/tkinter_stuff/tkinter/tkinter_works/json_test.py
File is not detected please change the pathway in the python code and try again

Process finished with exit code 0
```

Fig 24. Outcome for Invalid input: Test case

3. Reflections:

Pratyush:

This was my second group programming project. Previously, I had worked on front end programming through Dart and Android Studio. However, this was the first time I was using "*tkinter*". Since my contribution was mainly in front end, a lot of time was spent in learning how to use *tkinter*. Hence, this project was challenging as I was out of my comfort zone. Regardless, I faced a variety of problems like splitting the date and time. During the course of the project, learnt the implementation of functions like "*lambda"* and "*map*" and modules like "*functools*" and "*tkcalendar*". For file handling we were previously using excel files so I even learnt data frame manipulation using "*pandas*". The mini project is the place where I had to apply all the concepts we

learnt in LAMS like abstraction and divide and conquer. Overall, this was a great learning experience for me.

Jing:

This was my very first experience in programming a full code for a mini-project. With my inexperience in coding, this was like learning another language altogether by going through many online platforms. I personally went through most of the platforms from WxPython to Pygame and finally settling with Tkinter. During the front end design, I faced many challenges like we used for loops for making buttons for stalls. However, on clicking the buttons it displayed the menu only for one stall. Due to this we learnt the usage of partial from functool and use of x:x in lambda functions in Python. There were many other problems in front end design but we solved it as a team. In conclusion, even though the coding process was a tough and tedious one, it was still enjoyable while learning new things along the way.

Mervyn:

Firstly, this being my first ever programming group project, I did not realise the immense difficulty in collaboration with other programmers. During the course of this project, I worked mainly on the back end and faced problems in creating the database for the project like the restriction of json not supporting int datatype caused a lot of problems. In the back end, I also created the database and managed all the json files. With this I came to the realisation that using .txt files as a way to import data is very primitive and that there are always better ways in python. Such as using json module for example. Personally, this project worked towards my personal development and greatly enhanced my coding knowledge.

Word Count = 1198(Excluding Table of Contents,Title and Label for Image)