main.py

```
#Sleep Function prevents overload of information for user into the terminal (visually)
from time import sleep
import handling
import interface
import tools
#Defining initial variables
seats = {
    6:['_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_',],
    5:['_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_',],
    4:['_','_','_','_','_','_','_','_','_','_','_','_','_','_','_','_',],
    3:['_','_','_','_','_','_','_','_','_','_','_','_','_','_',],
    2:['_','_','_','_','_','_','_','_','_','_','_','_',],
    1:['_','_','_','_','_','_','_','_','_','_',],
    0:['_','_','_','_','_','_','_','_'],
}
prices = {1:100,2:80,3:70,4:70,5:60,6:40,7:20}
users = {}
program_status=True
#onload called before starting the main while loop to pull info from data.txt
users,seats = handling.onload(users,seats)
while program_status:
    user = tools.inp('str', ['a', 'n'], 'Please input the desired mode (a = admin, n = 
normal): ')
    if user == 'n':
        if tools.inp('str', ['y', 'n'], 'Are you a new user? (Y/N) ') == 'y':
            #current_user is used for many of the functions above in order to mutate the 
correct data/userdata
            current_user=tools.create_user(users)
            print(f'Welcome {current_user}!')
            sleep(1)
        else:
            while True:
                current_user = str(input('Please enter your username (not case or space 
sensitive):')).lower().replace(' ', '')
                sleep(1)
                if (current_user in [i for i in users.keys()]):
                    print(f'Welcome {current_user}!')
                    break
                print(f'"{current_user}" does not match any username!')
                continue
#This final while loop ensures that the user, regardles whether they are in admin or 
normal mode, can repeatedly access the menu until quit is selected
    while program_status:
        if user =='a':
            current_user=None
            program_status = tools.menu(user,tools.inp('str', ['1', '2', '3', '4'], '1. 
View Ticketing Status \n2. Cancel Ticket\n3. Reset\n4. 
Quit\n'),current_user,seats,users)
        else:
            program_status = tools.menu(user,tools.inp('str', ['1', '2', '3', '4'], '1. 
Book Ticket \n2. Cancel Ticket\n3. Show Ticket\n4. 
Quit\n'),current_user,seats,users)
```

tools.py

```python
from time import sleep
import handling
import interface
# This function facilitates exception handling for inputs, elements within input_options
must be strings
def inp(type, input_options=[], Input=None):
    while True:
        user_input = str(input(f'{Input}')).lower().replace(' ', '')
        sleep(1)
        if type == 'str':
            if (user_input in input_options):
                return user_input.lower()
            else:
                print(f'Please enter one of the following values: {input_options}')
        else:
            return (user_input.strip()).lower()
# This function is responsible for the menu interface which shows up, both in admin, and
normalmenu_option, ensuring the user ends up using the correct interface item
def menu(user_mode,menu_option,current_user,seats,users):
    if (menu_option) == '4':
            handling.offload(users)
            print('Goodbye! Thank you for using our services!')
            return False
    if user_mode == 'n':
        if (menu_option) == '1':
            interface.book_ticket(current_user, users, seats)
        elif (menu_option) == '2':
            interface.cancel_ticket(current_user, seats, users)
        elif (menu_option) == '3':
            interface.show_ticket(current_user, users)
    elif user_mode == 'a':
        if (menu_option) == '1':
            handling.update_display(seats)
        elif (menu_option) == '2':
            interface.cancel_ticket(None, seats, users)
        elif (menu_option) == '3':
            interface.reset(seats, users)
    return True
# This is used to create a new user in the users dict, and ensures that no two of the same
usernames can exist, as well as ensuring that whatever their username, it can be properly
stored and read again
def create_user(users=dict):
    while True:
        userName = inp('cu',Input='Please enter your preferred username (not case or space
sensitive): ')
        sleep(1)
        if (userName in users.keys()):
            print('This username already exists, Please try a new one')
        else:
            users[userName] = [[]]
            break
    return userName
```

interface.py

```python
from time import sleep
import handling
import tools
# I have created this new function, not present in milestone 1, to handle any exceptions
that may occur when handling seat/row inputs
def pick_seat(row_or_seat, seats=dict, row_num=None):
    while True:
        value = input(f'Please enter the {row_or_seat} number: ')
        try:
            if row_or_seat == 'row': test = seats[int(value)-1][0]
            elif row_or_seat == 'seat': test = seats[row_num-1][int(value)-1]
        except ValueError: # if the input value is not an integer the above int(value)
will raise a ValueError
            print(f'"{value}" is not an integer! please enter a valid interger.')
            continue
        except (IndexError, KeyError): # If the row does not exist it will raise a
KeyError, if the seat does not exist in the row it will raise a IndexError
            if row_or_seat == 'row': print(f'{value} is not a valid row, please enter a
row between 1 and 7!')
            elif row_or_seat == 'seat': print(f'There is no seat {value} in row {row_num},
please enter a valid seat number!')
            continue
        else:
            break
    return int(value) # we return the value which passes the given requirements as an
integer
# This function is called to book tickets, and loops until the user either quits or books
their ticket
def book_ticket(current_user, users=dict, seats=dict,
prices={1:100,2:80,3:70,4:70,5:60,6:40,7:20}):
    handling.update_display(seats)
    while True:
        row = pick_seat('row', seats)
        seat = pick_seat('seat', seats, row)
        #Checks whether the seat in question is available, no need to search the
users.values(), we can easily check using the seats dict
        if seats[row-1][seat-1] == 'x':
            print(f'row {row}, seat {seat} is already booked! Please select a valid seat')
            sleep(1)
            continue
        elif seats[row-1][seat-1] == '_':
            print(f'Total: £{prices[row]}')
            choice = tools.inp('str', ['1', '2'], '1. Confirm \n2. Cancel\n')
            if choice == '1':
                users[current_user][0].append([row-1, seat-1])
            seats = handling.update(seats, users)
            break
# I merged the cancel ticket function with the admin version, taking the current_user into
account to determine which should be called
def cancel_ticket(current_user,seats=dict, users=dict,
prices={1:100,2:80,3:70,4:70,5:60,6:40,7:20}):
    #copies the seats dict to ensure that we don't mutate the dict when displaying the
users booked seats
    tempC = seats.copy()
    if current_user:
        if users[current_user] == [[]]:
            print('You currently have no bookings\n')
            return None
        for booking in users[current_user][0]:
            tempC[booking[0]][booking[1]] = 'B'
        #Here we are simply displaying the users bookings
        counter = 0
        for i in tempC.values():
            temp_blank = [' ' for unused_variable in range(counter)]
            print(f"{''.join(temp_blank)}{' '.join(i)}")
            print()
            counter += 2
        #In this while loop we take 2 inputs from the user, ensuring they are cancelling
their own bookings, and mutate their data
        while True:
            row = pick_seat('row', seats)
            seat = pick_seat('seat', seats, row)
            if tempC[row-1][seat-1] == '_' or tempC[row-1][seat-1] == 'x':
                print(f'row {row}, seat {seat} is not booked by you! Please select a valid
seat')
                sleep(1)
                continue
            elif tempC[row-1][seat-1] == 'B':
                print(f'Row: {row}\nSeat: {seat}\nPrice: £{prices[row]}')
                choice = tools.inp('str',['y', 'n'],'Do you want to cancel? (Y/N)')
                if choice == 'y':
                    users[current_user][0].remove([row-1, seat-1])
                seats = handling.update(seats,users)
                break
    else:
        while True:
            handling.update_display(seats)
            row = pick_seat('row', seats)
            seat = pick_seat('seat', seats, row)
            if tempC[row-1][seat-1] == '_':
                print(f'row {row}, seat {seat} is not booked! Please select a valid seat')
                sleep(1)
                continue
            elif seats[row-1][seat-1] == 'x':
                print(f'Row: {row}\nSeat: {seat}\nPrice: £{prices[row]}')
                choice = tools.inp('str',['y', 'n'],'Do you want to cancel? (Y/N)')
                if choice == 'y':
                    for key in users.keys():
                        if [row-1, seat-1] in users[key][0]:
                            users[key][0].remove([row-1, seat-1])
                seats = handling.update(seats,users)
                break
# This function displayes the current users bookings, first checking whether the user has
any bookings, before proceeding
def show_ticket(current_user, users=dict, prices={1:100,2:80,3:70,4:70,5:60,6:40,7:20}):
    if users[current_user][0] == []:
        print('You currently have no bookings')
        print()
        return None
    for i in range(len(users[current_user][0])):
        #I placed the ------... on both sides (since it broke the tickets up better)
        print(f'- - - - - - - - - - - - - - - - - -\nTicket: {i+1}\nRow: 
{users[current_user][0][i][0]+1}\nSeat: {users[current_user][0][i][1]+1}\nPrice: 
£{prices[users[current_user][0][i][0]]}\n- - - - - - - - - - - - - - - - - -')
# This function resets all the bookings back to their default unbooked status
def reset(seats=dict, users=dict):
    choice = tools.inp('str', ['y', 'n'],'Are you sure you want to reset? (Y/N)')
    if choice == 'y':
        # Here we only reset the values and not the keys, this way the user's login 
username is still saved
        for key in users.keys():
                users[key][0] = []
    seats = handling.update(seats,users)
    return seats
```

handling.py

```
# This update function first sets all seats to empty ('_') to ensure no issues, then 
refreshes the seats based on the users dict
def update(seats=dict, users=dict):
    for row in seats.values():
        j = 0
        while j < len(row):
            row[j] = '_'
            j+=1
    for key in users.keys():
        for seat in users[key][0]:
            seats[seat[0]][seat[1]] = 'x'
    return seats
# This function, which was not in milestone 1, is implimented to ensure that the data in 
the txt files can be read
def can_it_be_read():
    try:
        with open('data.txt', 'r'):
            pass
    except:
        print('Data failed to load! Using backup data, please notify an admin if errors 
persist')
# If an exception is raised, backup.txt is returned to be used as the source of data, and 
data.txt is reconstruced later
        return 'backup.txt'
    else:
        return 'data.txt'
# This Function onloads the information from the txt, by enumurating, and 
using .split('|') which brakes the params into a list
def onload(users, seats):
    # Exception handling to ensure the programe runs if the data file cannot be accessed, 
a backup is used
```

```python
    data_file = open(can_it_be_read(), 'r')
    # This is executed right after can_it_be_read() to ensure that the backup file can be
accessed if nessessary.
    files_to_check = ['data.txt', 'backup.txt']
    for file in files_to_check:
        try:
# opening a file in 'x' creates a file, and raises an exception error if one already
exists
            with open(file, 'x'): # with open() ensures that the file is safely created
and closed after
                pass
        except FileExistsError:
            pass # although using pass is not always best practice, here we are merely
checking whether the file exists
    content = data_file.readlines()
    for unused_variable, line in enumerate(content):
        params = line.strip().split('|')
        temp_users_dict = {'user_name':params[0],'bookings':eval(params[1])}
        #Here i use the formated data to re-construct the users information in the users
dict
        users[temp_users_dict['user_name']] = [temp_users_dict['bookings']]
    data_file.close()
    seats = update(seats,users)
    return users,seats
# This function offloads/saves the user data to data.txt, each bit of data is seperated by
'|' in order to be split later
def offload(users=dict):
    data_file = open('data.txt','w')
    # creating/writing a backup file to ensure data remains even if data.txt cannot be
opened or is deleted
    back_up_file = open('backup.txt','w')
    for user_name in users.keys():
        data_file.write(f'{user_name}|{users[user_name][0]}\n')
        back_up_file.write(f'{user_name}|{users[user_name][0]}\n')
    data_file.close()
    back_up_file.close()
# this function is called to print the seats to the user in the format requested to be
displayed
def update_display(seats=dict):
    #Counter used to increment for blank space needed for the layout
    counter = 0
    for row in seats.values():
        #saves the the blank space for each individual row of seats
        temp_blank = [' ' for unused_variable in range(counter)]
        print(f"{''.join(temp_blank)}{' '.join(row)}")
        print()
        counter += 2
```