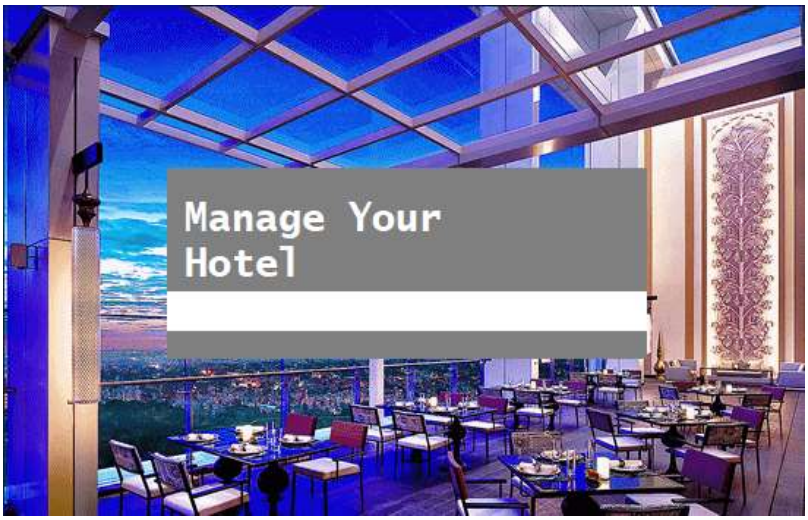
Project Report On

Hotel Management System



Submitted to

JAYSHREE PERIWAL HIGH SCHOOL

3, Chitrakoot Scheme

Jaipur

In partial fulfillment
of the requirements for
All India Senior School Certificate Examination 2021
Of
CENTRAL BOARD OF SECONDARY EDUCATION

Submitted by: ***Naman Gurawa*** (Roll No.: .....)
Prerak Lodha (Roll No.: .....)
Priyanshu Agarwal (Roll No.: .....)

Acknowledgement

We would like to thank everyone who helped us to accomplish this project.

Our sincere thanks to **respected teachers**, who helped us with their valuable suggestions and support throughout the development of the project.

We are highly thankful to our project guide **Ms. Himanshi Sharma** for providing guidance and support at every stage of the project.

We are extremely grateful to **Mrs. Jayshree Periwal, Director** and **Mrs. Madhu Maini, Principal** of JAYSHREE PERIWAL HIGH SCHOOL, JAIPUR, for providing us a very good computer lab, due to which this project became possible.

Naman Gurawa
Prerak Lodha
Priyanshu Agarwal

CERTIFICATE OF ORIGINALITY

This is to certify that the project entitled "**Hotel Management System**" submitted to **JAYSHREE PERIWAL HIGH SCHOOL** in partial fulfillment of the requirement for **All India Senior School Certificate Examination (AISSCE) 2021** of **CBSE**, is original work carried out by **Naman Gurawa, Prerak Lodha, Priyanshu Agarwal** under my guidance.

The matter embodied in this project is genuine work done by the students and has not been submitted of any course of study.

.....
Signature of the guide

Date: .....

Name: Ms. Himanshi Sharma

JAYSHREE PERIWAL HIGH SCHOOL

Jaipur

CONTENTS

OBJECTIVE AND SCOPE OF THE PROJECT

Objective:

The main objective of the program is to Computize management of hotel data.

This package is useful in maintaining Room Booking/Cancellation Data and Analyze the Sales of the Management.

Scope:

This project is developed for Hotels, Jaipur
Further, it can be easily customized for the use of any other company or firm.

This project will help them to create a working system into the latest concept of paper-less office.

PROBLEM DEFINITION

The project “Hotel Management System” shows a simple room bookings management in hotels. This project includes

The project stores and maintains Customer Name, Book ID, Date of Check-in and Check-out, etc. These details can be viewed, sorted and searched based upon different parameters as well as printed on paper using various reports available in the project (graphical representation).

Suitable assumptions can be made during implementation. A proper normalized database is to be maintained in the RDBMS and the front end is to be developed using advanced interface controls. User-friendly interface is to be generated.

As a developer, you are required to design the project and develop it as per user/customer/visitor needs. (Developer can also visit hotel to collect customer details and live data).

Life Cycle of the Project

System Development Life Cycle (SDLC)

The System Development Life Cycle (SDLC) is a set of activities that analysts, designers and users carry out to develop and implement an Information System.

The SDLC consists of the following activities.

Feasibility Study

↓

Requirement Definition (Analysis)

↓

Design (Database & Program)

↓

Development of Software

↓

Unit Testing

↓

System Testing

↓

Implementation

↓

Evaluation

↓

Maintenance

Details of Hardware and Software used

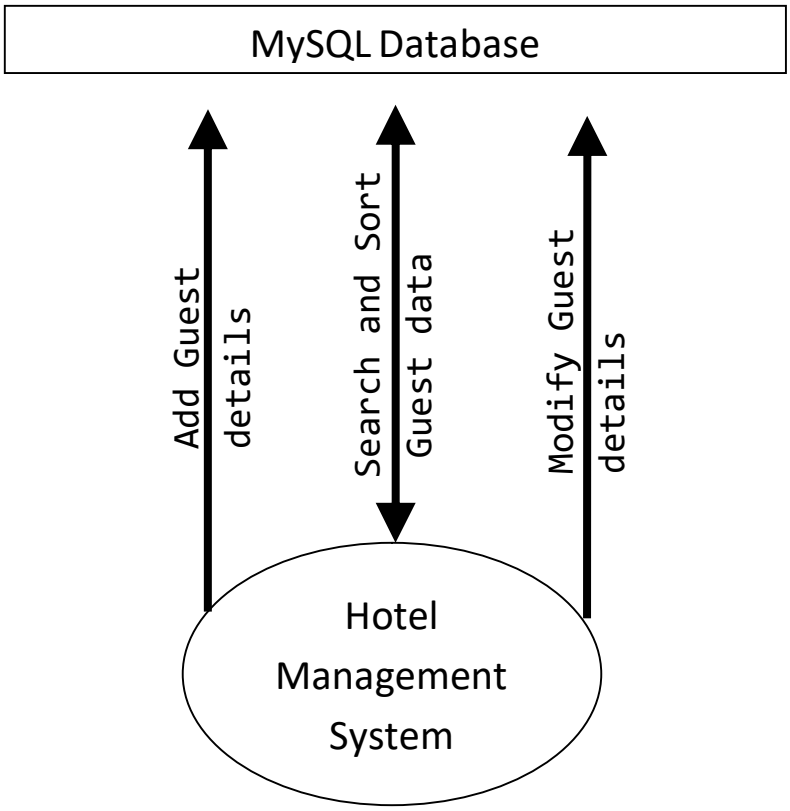
Hardware Specifications

Microprocessor (CPU)	: Intel Core i3 9350KF
Memory (RAM)	: 16 GB DDR3 2333 MHz
Virtual Memory	: 64-bit
Storage	: 1 TB SSD + 2 TB HDD
VDU	: VGA Display Port
Keyboard	: Logitech K200
Mouse	: Xmate Zorro Gaming Mouse
Printer	: Inkjet / Laser
GPU	: RTX 2080 Super 8GB GDDR6 256-bit

Software Specifications

Operating System	: Windows 10
Front-End Design	: Python 3.7.6
Back-End	: MySQL
Documentation	: Word (Office 365)
Version Control	: GIT

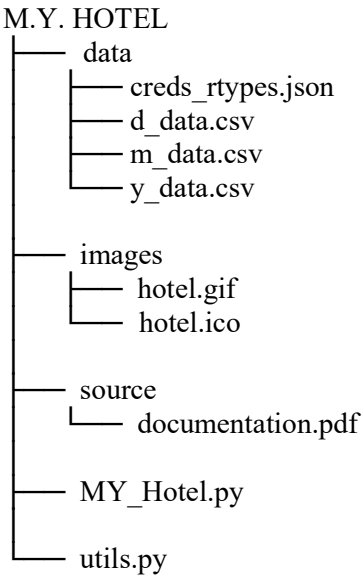
CONTEXT DIAGRAM



Administrator

SOURCE CODE FOR THE PROJECT

This project (M.Y. Hotel) has the following directory structure :



Starting with the description of `utils.py` :

This module contains all the "utilities" i.e. functions and constants that are required by the main project file.

1. MODULES USED

```
import random
import os
import importlib
import sys
from tkinter import Tk, Toplevel
import base64
```

2. CONSTANTS

```
src_desc = "HOW DOES THE SEARCH WORK:\n\nNOTE: All values are arranged in
ascending order(BOOK_ID)\n• All the searched"\
           " data is displayed above in the table\n\n• The radiobuttons on the
leftmost side give instant results" \
           " when clicked (though searching may take time). To clear the searched
data just click the radiobutton" \
           " 'None'(will not clear entries)\n\n• The entries (NAME, etc) search for
the value input" \
           "ted. They aren't case-sensitive\nNOTE: The more the number of entries
filled the less dilute will be the" \
           " result because the search is based on 'OR' method not 'AND'
method i.e. if BOOK_ID has an input of '98'" \
           " then all the the booking ids that have 98 in them will be shown and if
```

```
NAME field is also filled, say," \
           " with the value 'luc' then all the names having 'luc' in them will be
shown along with booking ids cont" \
           "aining '98'\n\n• Date search gives results for exact date of check in or
check out. The visible format is" \
           " mm/dd/yy \nNOTE: Date can also be inputted in yyyy/dd/mm format but
usage of the calendar drop down is" \
           " recommended (efficient)\n\n• Cancelling booking: Right click on any of
the above given rows of entries to"\
           " cancel booking or to simply remove data. The search has to be done again
to refresh the result display" \
           " area."
"Search description (How to search?)"

MONTHS =
{1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6: 'June', 7: 'July',
 8: 'August',
          9: 'September', 10: 'October', 11: 'November', 12: 'December'}
"Months dictionary"

sub_nums = ['₁', '₂', '₃', '₄', '₅', '₆', '₇', '₈', '₉', '₀']
"Subscript numbers required for too long name"

CRED_DICT = {
    "name": "",
    "password": "",
    "room_def": False,
    "sel_h_type": "custom",
    "log": "off",
    "hr_time": "2001-01-01",
    "currency": "Rupees",
    "h_type": {
        "small": ['12', '5', '0', '1', '1', '2', '2', '2'],
        "standard": ['50', '20', '10', '10', '4', '8', '8', '10'],
        "resort": ['135', '30', '30', '30', '15', '20', '20', '20'],
        "large": ['225', '40', '35', '40', '30', '50', '45', '35'],
        "custom": [str(0) for i in range(8)]
    },
    "tax": {"Security": "10", "Maintenance": "12", "Service": "14"},
    "h_r_price": {
        "small": ['500', '1000', '1500', '2000', '4000', '4500', '3500', '3000'],
        "standard": ['900', '1450', '1950', '3000', '5250', '6000', '4500', '3950'],
        "resort": ['1300', '2000', '3000', '4000', '6500', '8000', '7000', '5250'],
        "large":
['2000', '3500', '4800', '8000', '15000', '18000', '12000', '10000'],
        "custom": [str(0) for k in range(8)]
    },
    "notes": "This area is just to keep NOTES. They won't be removed even if user
changes page and would be"
             " saved when user exits."
}
"Default credentials dictionary"
```

3. FUNCTIONS

```
# SQL Connection and execution
def connect_to_sql():
    """Connects to database and creates tables if they don't exist."""
    import pymysql

    running = True
    while running:
        try:
            _connection = pymysql.Connect(host='localhost', user='root', passwd='',
database='hms__my_hotel')
            _cursor = _connection.cursor()
            _cursor.execute('''
            CREATE TABLE IF NOT EXISTS hotel_info(
            BOOK_ID VARCHAR(200) PRIMARY KEY,
            NAME VARCHAR(100),
            ROOM_NO INTEGER(4),
            DATE_OF_CIN DATE,
            DATE_OF_COUT DATE,
            ROOM_TYPE VARCHAR(20),
            PH_NO VARCHAR(20),
            PRICE DECIMAL(10,2),
            PAID_THRU VARCHAR(50));
            ''')
            _connection.commit()
            running = False
        except Warning or Exception as w:
            running = False
        except pymysql.err.Error:
            running = False


def create_sql_db():
    """Connecting database and creating database if it doesn't exist."""
    import pymysql

    running = True
    while running:
        try:
            connect = pymysql.Connect(host='localhost', user='root', passwd='',
database='hms__my_hotel')
            if connect is True:
                continue
            else:
                conn = pymysql.Connect(host='localhost', user='root', passwd='')
                curs = conn.cursor()
                curs.execute('CREATE DATABASE hms__my_hotel;')
                conn.commit()
                connect_to_sql()
        except pymysql.Error or Exception as i_e_:
            try:
                conn = pymysql.Connect(host='localhost', user='root', passwd='')
```

```
                curs = conn.cursor()
                curs.execute('CREATE DATABASE hms__my_hotel;')
                conn.commit()
                connect_to_sql()
                running = False
            except pymysql.Error or Exception as i_e:
                running = False
                connect_to_sql()
        except pymysql.err.DatabaseError:
            pass


def unique_id() -> int:
    """:returns: a unique ID number"""
    try:
        with open('./data/ids', 'r') as id_file:
            ids = id_file.read().split('|')

            while True:
                _id = random.randrange(999999999, 99999999999999)

                if str(_id) in ids:
                    pass
                else:
                    with open('./data/ids', 'a') as _id_file:
                        _id_file.write(str(_id)+'|')

                    break
        return _id

    except FileNotFoundError or Exception:
        with open('./data/ids', 'w') as id_file:
            _id = random.randrange(999999999, 99999999999999)
            id_file.write(str(_id))
            return _id


def install_necessary_modules(modname: str = None) -> None:
    """
    Function to install and import modules
    :param modname: name of module to be imported
    """
    try:
        # If module is already installed, try to import it
        importlib.import_module(modname)
        print(f"Importing {modname}")
    except ImportError:

        # Error if module is not installed
        if os.system('PIP --version') == 0:
            # No error from running PIP in the Command Window, therefore PIP.exe is
in the %PATH%
            os.system(f'PIP install {modname}')

        else:
```

```
            # Error, PIP.exe is NOT in the Path!!
            pip_location_attempt_1 = sys.executable.replace("python.exe", "")
+ "pip.exe"
            pip_location_attempt_2 = sys.executable.replace("python.exe", "")
+ "scripts/pip.exe"

            if os.path.exists(pip_location_attempt_1):
                # The Attempt #1 File exists!!!
                os.system(pip_location_attempt_1 + " install " + modname)

            elif os.path.exists(pip_location_attempt_2):
                # The Attempt #2 File exists!!!
                os.system(pip_location_attempt_2 + " install " + modname)

            else:
                # Neither Attempts found the PIP.exe file, So Fail...
                print('Fatal error: Can\'t find PIP.exe')


def center_window(win: Tk or Toplevel):
    """
    centers a tkinter window
    :param win: the main window or Toplevel window to center
    """
    win.update_idletasks()
    width = win.winfo_width()
    frm_width = win.winfo_rootx() - win.winfo_x()
    win_width = width + 2 * frm_width
    height = win.winfo_height()
    titlebar_height = win.winfo_rooty() - win.winfo_y()
    win_height = height + titlebar_height + frm_width
    x = win.winfo_screenwidth() // 2 - win_width // 2
    y = win.winfo_screenheight() // 2 - win_height // 2
    win.geometry('+{}+{}'.format(x, y))
    win.deiconify()



def create_gif():
    image_64_decode = base64.decodebytes(HOTEL_GIF)
    image_result = open('./images/hotel.gif', 'wb')
    image_result.write(image_64_decode)


def create_ico():
    image_64_decode = base64.decodebytes(HOTEL_ICO)
    image_result = open('./images/hotel.ico', 'wb')
    image_result.write(image_64_decode)


def check_file(file: str = None) -> bool:
    return True if os.path.isfile(file) else False


def check_and_create_file(file: str = None):
```

```
    if os.path.isfile(file):
        return True
    else:
        with open(file, 'w') as f:
            f.write('')
            f.close()


def check_dir(dirname: str = None) -> bool:
    return True if os.path.isdir(dirname) else False


def check_and_create_dir(dirname: str = None):
    if os.path.isdir(dirname):
        return True
    else:
        os.mkdir(dirname)
```

The module also contains base64 encoded binary string for `hotel.gif` and `hotel.ico` (in case user deletes them), which has not been included here because it has over 2700 lines of characters, which are not readable.

Now, the main program file `MY_Hotel.py`：

1. MODULES USED

```
# imported modules
from tkinter import (Tk, Toplevel, Entry, Radiobutton, Label, Button, Scrollbar,
Frame, Text, Menu, StringVar, IntVar,
                     END, YES, TOP, BOTTOM, RIGHT, LEFT, BOTH, N, S, W, CENTER,
VERTICAL, HORIZONTAL, EXTENDED, SUNKEN,
                     WORD, ALL, DISABLED)
from tkinter.scrolledtext import ScrolledText
from tkinter.ttk import Treeview, Style, Combobox, Separator
from tkinter.messagebox import (showerror, _show, askyesno, showinfo)
from os import path
import os
import json
import time
import datetime
from subprocess import Popen
from utils import (install_necessary_modules, MONTHS, CRED_DICT, center_window,
check_and_create_dir, unique_id,
                   check_and_create_file, create_gif, create_ico, create_sql_db,
check_file, src_desc, sub_nums,
                   random, base64, pymysql)

try:
    from PIL import Image, ImageTk
    import pandas
```

```
    from tkcalendar import DateEntry
    import matplotlib.pyplot as plt
except ImportError or ModuleNotFoundError:
    print('Some modules for functionality of the application have not been found.'
          '\nPlease wait till the modules are being downloaded...')
    try:
        install_necessary_modules('requests')
        install_necessary_modules('pymysql')
        install_necessary_modules('pandas')
        install_necessary_modules('matplotlib')
        install_necessary_modules('pillow')
        install_necessary_modules('tkcalendar')

        from PIL import Image, ImageTk
        import pymysql
        import pandas
        from tkcalendar import DateEntry
        import matplotlib.pyplot as plt

    except ModuleNotFoundError as e2:
        showerror('M.Y. Hotel', 'Cannot install required modules!')
```

2. CONSTANTS

```
# GLOBAL CONSTANTS
NORMAL = 'normal'
"Normal window mode with all the window decorators"

FIXED = 'fixed'
"Fixed window mode with resizing disabled"

ZOOMED = 'zoomed'
"Full screen window mode without decorators"

win_event = NORMAL  # default window mode is normal
"Current window mode variable"

room_t =
['Single', 'Double', 'Triple', 'Quad', 'Studio', 'Master', 'Junior', 'Connecting']
"Room types"

col_names = ['BOOK_ID', 'NAME', 'ROOM_NO', 'PH_NO']
"Column names for search"

DAY, MONTH, YEAR = 'day', 'month', 'year'
"paramters for the function `csv_sql`"
```

3. USEFUL FUNCTIONS

There are some functions which need to be defined before the main program starts. These functions are generally used for updating, declaring or creating something.

The first function here is used in creating graphs and is also used to update booking data in the csv files according to day, month and year :

```
def csv_sql(dtype: str = MONTH) -> pandas.DataFrame:
    """
    Function to create dataframe from MySQL table
    for creating graph.

    :param dtype: type in which date is required
    :returns: a pandas.DataFrame object
    """
    cursor.execute(
        'SELECT COUNT(*),DAY(DATE_OF_CIN),MONTH(DATE_OF_CIN),YEAR(DATE_OF_CIN) FROM
HOTEL_INFO GROUP BY MONTH'
        '(DATE_OF_CIN),YEAR(DATE_OF_CIN) ORDER BY
YEAR(DATE_OF_CIN),MONTH(DATE_OF_CIN),DAY(DATE_OF_CIN);'
    )

    data = cursor.fetchall()
    df_dict = {'DATE_TYPE': [], 'BOOKINGS': []}
    df = pandas.DataFrame()

    if dtype == DAY:
        for i in range(len(data)):

            if f'{data[i][1]} {MONTHS[int(data[i][2])]}, {data[i][3]}' in df_dict['DA
TE_TYPE']:
                df_dict['BOOKINGS'][df_dict['DATE_TYPE'].index(f'{data[i][1]} {MONTHS
[int(data[i][2])]},'
                                                               f' {data[i][3]}')] +=
data[i][0]

            else:
                df_dict['DATE_TYPE'].append(f'{data[i][1]} {MONTHS[int(data[i][2])]},
 {data[i][3]}')
                df_dict['BOOKINGS'].append(0)
                df_dict['BOOKINGS'][df_dict['DATE_TYPE'].index(f'{data[i][1]} {MONTHS
[int(data[i][2])]},'
                                                               f' {data[i][3]}')] +=
data[i][0]
        df = pandas.DataFrame(data=df_dict)
        df.to_csv('./data/d_data.csv', mode='w')

    elif dtype.lower() == MONTH:
        for i in range(len(data)):

            if f'{MONTHS[int(data[i][2])]}, {data[i][3]}' in df_dict['DATE_TYPE']:
                df_dict['BOOKINGS'][df_dict['DATE_TYPE'].index(f'{MONTHS[int(data[i][
2])]}, {data[i][3]}')] += \
                    data[i][0]

            else:
```

```
                df_dict['DATE_TYPE'].append(f'{MONTHS[int(data[i][2])]}, {data[i][3]}
')
                df_dict['BOOKINGS'].append(0)
                df_dict['BOOKINGS'][df_dict['DATE_TYPE'].index(f'{MONTHS[int(data[i][
2])]}, {data[i][3]}')] += \
                    data[i][0]
        df = pandas.DataFrame(data=df_dict)
        df.to_csv('./data/m_data.csv', mode='w')

    elif dtype.lower() == YEAR:
        for i in range(len(data)):

            if f'{data[i][3]}' in df_dict['DATE_TYPE']:
                df_dict['BOOKINGS'][df_dict['DATE_TYPE'].index(f'{data[i][3]}')] +=
data[i][0]

            else:
                df_dict['DATE_TYPE'].append(f'{data[i][3]}')
                df_dict['BOOKINGS'].append(0)
                df_dict['BOOKINGS'][df_dict['DATE_TYPE'].index(f'{data[i][3]}')] +=
data[i][0]
        df = pandas.DataFrame(data=df_dict)
        df.to_csv('./data/y_data.csv', mode='w')

    return df
```

The second function here is used to update credentials and other data values changed while the program being in use :

```
def write_to_json():
    """
    Updates the json creds_rtypes file with credentials
    and other data changed while program being in use.
    """
    cred_dict = creds_rtypes
    json_object = json.dumps(cred_dict, indent=4)

    with open('./data/creds_rtypes.json', 'w') as f:
        f.write(json_object)
    li = []

    for i in range(8):
        li.append(cred_dict['h_type'][cred_dict['sel_h_type']][i])
```

The last function creates the credentials JSON file - `creds_rtypes.json` :

```
def create_json():
    """Creates the JSON credentials file"""
    json_object = json.dumps(CRED_DICT, indent=4)
    with open('./data/creds_rtypes.json', 'w') as f:
        f.write(json_object)
```

4. MAIN PROGRAM

Now, the most important part of the project, the User-interface with several `class` declarations for the window and its frames.

The first and foremost declarations include the `Tk` class of the module – tkinter in python which has been defined as the subclass - `Logic` of `Tk` . (NOTE: `Tk` generates a window object.)
The rest of the frames such as `WelcomePage` , `LoginPage` , etc have been defined as separate classes uniquely. One may as well notice the use of `__slots__` in all the classes, which are used to replace the default usage of dictionaries in classes to save memory space.

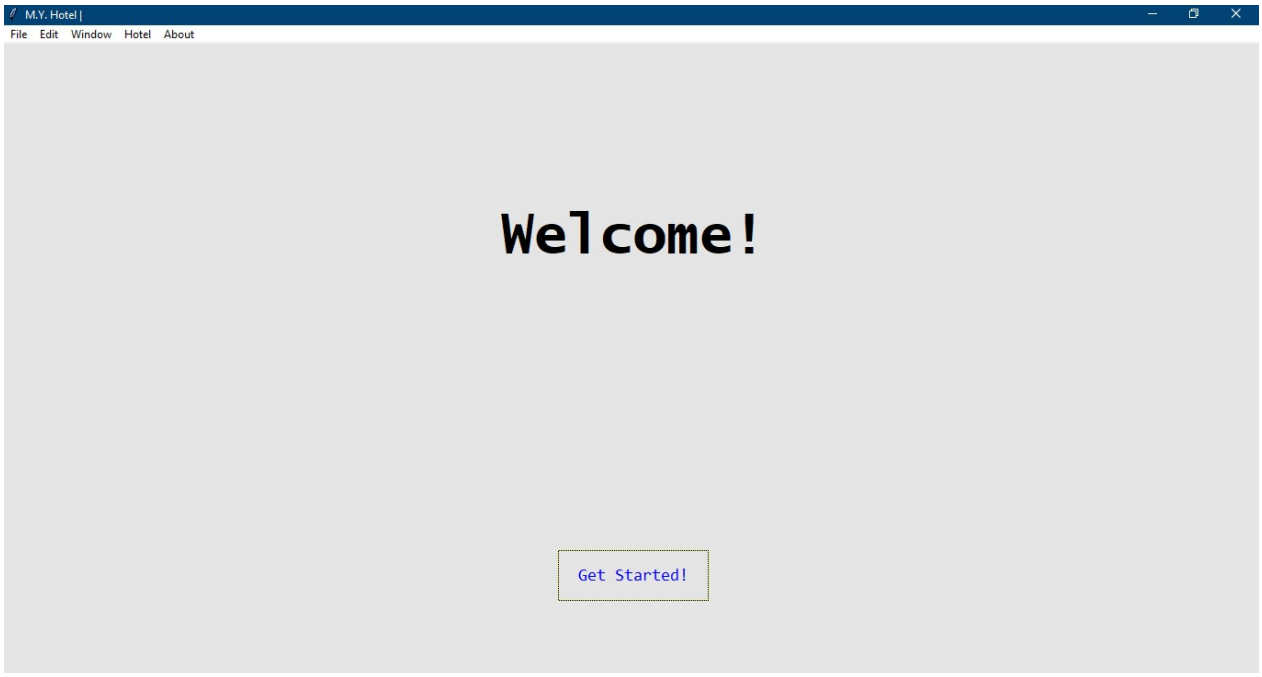
Working Code:

```
class Logic(Tk):
    """Tk class for window creation and window specific functions"""
    __slots__ = ['current_visible_frame', 'container', 'name_label']

    def __init__(self, *args, **kwargs):
        Tk.__init__(self, *args, **kwargs)

        # menu bar for application
        MainMenu(self)

        self.current_visible_frame = None

        # the main container frame which cycles through all other frames
        self.container = Frame(self, bg='gray90')
        self.container.pack(side=TOP, fill=BOTH, expand=True)

        # dictionary object for frames
        self.frames = {}
        for f in (WelcomePage, LoginPage, DataEntryPage, RoomTypesPage, BrowsePage, B
ookRoomPage):
            frame = f(self.container, self)  # looping through all the frames
            self.frames[f] = frame
            frame.grid(row=1, column=0, sticky='nsew')
            frame.config(bg='gray90')

        # initially displays welcome page
        self.show_frame(WelcomePage)
        self.current_visible_frame = WelcomePage
```

```
        self.name_label = Label(self, text=f'M.Y. Hotel
| {Hotel_Name} | {time.strftime("%A, %b %d, %Y")}',
                                font=('Consolas', 10, 'normal'),
justify=LEFT, fg='gray28', anchor=W,
                                width=int(self.winfo_screenwidth()))

        self.bind('<F11>', self.full_screen_toggle)

    def full_screen_toggle(self, event='none'):
        """Toggle full screen"""
        global win_event

        if win_event == NORMAL:
            self.focus_set()
            self.overrideredirect(1)
            self.overrideredirect(0)
            self.attributes('-fullscreen', True)

            self.name_label.place(x=int(self.winfo_width()) * 0.0001,
y=int(self.winfo_height()) * 0.975)

            self.update_bb()
            win_event = ZOOMED

        elif win_event == FIXED:
            self.geometry(f'{int(app.winfo_screenwidth() -
 40)}x{int(app.winfo_screenheight()) - 80}+2+8')
            self.resizable(False, False)
        else:
            self.full_screen_cancel()

    def full_screen_cancel(self, event='none'):
        """
        Cancels full screen
        :param event: passed by functions and binders
        """

        global win_event
        self.overrideredirect(0)
        self.attributes('-fullscreen', False)
        win_event = NORMAL

    def show_frame(self, context):
        """
        Shows next required frame or page
        :param context: name of frame
        """
        frame = self.frames[context]
        frame.tkraise()
        self.update_idletasks()

    def remove_frame(self, context):
        """
        Removes unwanted frames from background
        :param context: name of frame
```

```
        """
        frame = self.frames[context]
        frame.forget()
        frame.grid_forget()
        self.update_idletasks()

    def update_name(self, name: str, password: list):
        """
        Updates name of Hotel and password on signup
        :param name: hotel name
        :param password: password for sign-in
        """
        global Hotel_Name

        # Sets title, `Hotel_Name` to the user entered hotel name
        # and sets password variable in credentials dictionary.
        self.title(f"M.Y. Hotel | {name}")
        Hotel_Name = creds_rtypes['name'] = name
        creds_rtypes['password'] = password

        self.show_frame(LoginPage)
        self.current_visible_frame = LoginPage
        write_to_json()

    def update_bb(self):
        """Function to update credentials file and bottom label"""
        self.name_label.configure(text=f'M.Y. Hotel
| {Hotel_Name} | {time.strftime("%A, %b %d, %Y")} |'
                                       f' {time.strftime("%H:%M:%S %p")}')
        write_to_json()
        self.after(1000, self.update_bb)
```

Welcome Page:

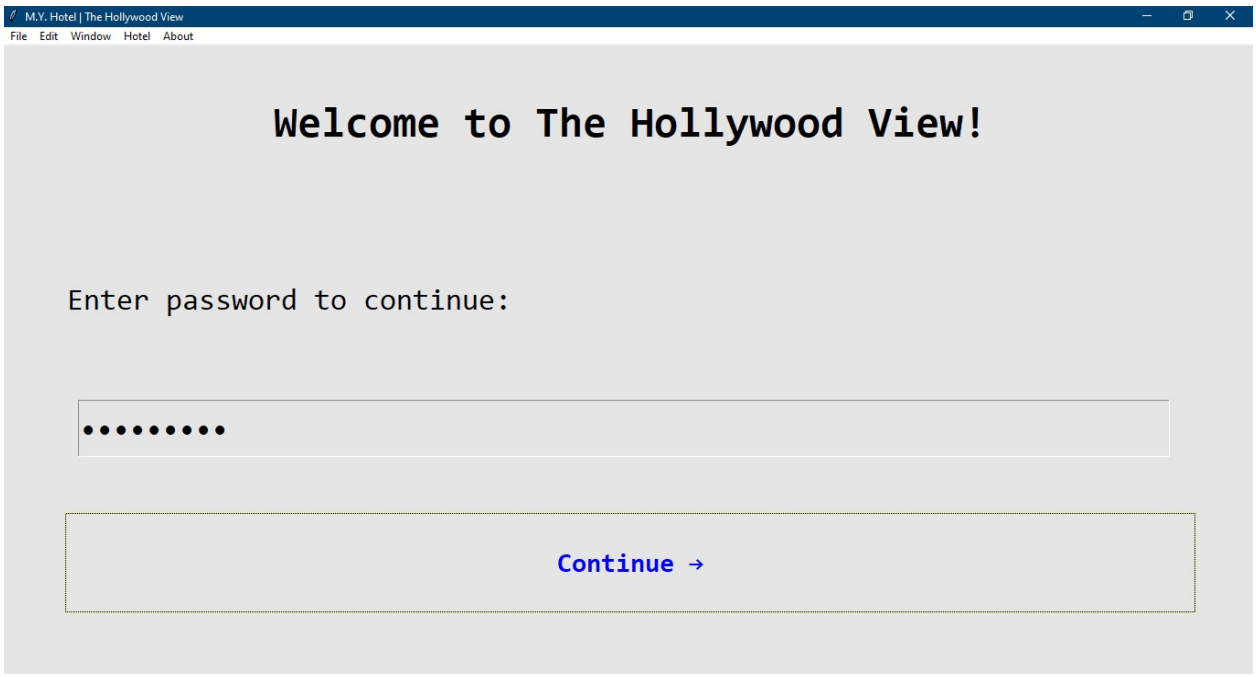


Working Code:

```
class WelcomePage(Frame):
    __slots__ = ['controller']

    def __init__(self, parent, controller):
        """
        Welcome Frame for the application
        :param parent: parent frame
        :param controller: parent window
        """
        Frame.__init__(self, parent)
        self.controller = controller

        welcome_label = Label(self, text='Welcome!',
width=int((self.winfo_screenwidth()) * 0.028), justify='center',
                              font=('Lucida Sans Typewriter', 45, 'bold'), bd=0,
relief='flat', bg='gray90',
                              height=int((self.winfo_screenheight()) * 0.009))
        welcome_label.pack(side=TOP, anchor=CENTER)
        welcome_label.grid_columnconfigure(0, weight=1)
        welcome_label.grid_rowconfigure(0, weight=1)

        self.update_idletasks()

        # Start with signing in or signing up (detected automatically)
        # user does not have to manually go to signup or sign-in page
        start_button = Button(self, text='Get Started!', width=16, height=2,
font=('Consolas', 14), bg='gray90',
                              command=lambda: self.frame_change(), justify='center',
bd=0, cursor='hand2', fg='blue')
        start_button.pack(side=BOTTOM,
```

```
anchor=CENTER, padx=int(self.winfo_reqheight()) * 0.1,
                        pady=int(self.winfo_reqwidth()) * 0.1)

    def frame_change(self):
        """Change frame to signin, or signup if credentials not found"""
        if len(creds_rtypes['name']) < 1:
            self.controller.remove_frame(WelcomePage)
            self.controller.current_visible_frame = DataEntryPage
            self.controller.show_frame(DataEntryPage)
        else:
            self.controller.current_visible_frame = LoginPage
            self.controller.show_frame(LoginPage)
```

Login Page:



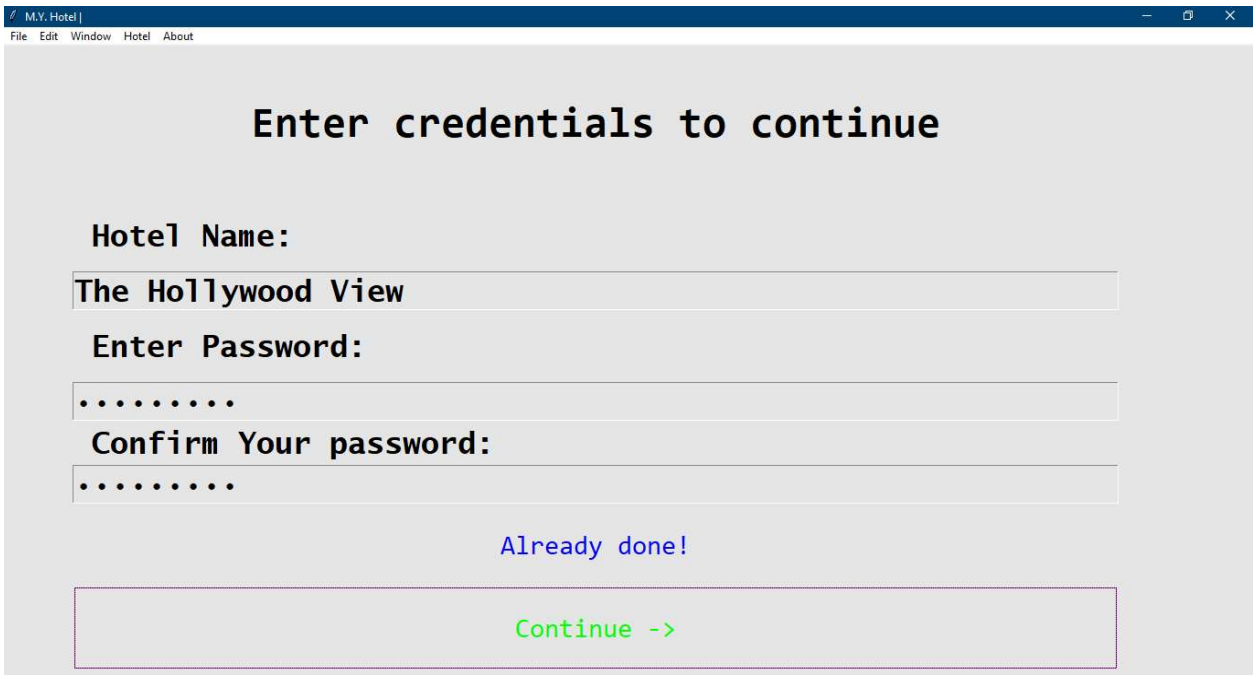
Working Code:

```
class LoginPage(Frame):
    __slots__ =
['controller', 'frame', 'name_label', 'pass_label', 'pass_entry', 'continue_BTN']

    def __init__(self, parent, controller):
        """
        Login page frame

        :param parent: parent frame
        :param controller: parent window
        """
        Frame.__init__(self, parent)
        self.controller = controller
```

```
        self.frame = Frame(self, width=1366, height=768, bd=0,
relief=SUNKEN, bg='gray90')
        self.name_label = Label(self.frame,
text=f'Welcome to {creds_rtypes["name"]}!', width=45, height=3, bd=0,
                                font=('Consolas', 35, 'bold'), bg='gray90')
        self.pass_label = Label(self.frame, text='Enter password to continue: ',
bd=0, height=2, bg='gray90',
                                font=('Consolas', 25), justify=LEFT)
        self.pass_entry = Entry(self.frame, show='•', width=65,
font=('Consolas', 25), bd=1, bg='gray90')
        self.continue_BTN = Button(self.frame, width=77, height=2,
text=f'Continue {chr(129130)}', font=("Consolas", 22, 'bold'),
                                   bd=0,
command=lambda: self.continue_it(), bg='gray90', fg='blue')

        self.frame.pack(anchor=N, side=TOP, expand=True)
        self.name_label.pack(side=TOP, anchor=N)
        self.pass_label.pack(anchor=W, padx=2, pady=70)
        self.pass_entry.pack(anchor=W, padx=15, ipadx=10, ipady=10)
        self.continue_BTN.pack(side=BOTTOM, anchor=S, pady=60, ipady=10)

        self.update()

    def convert_pwd(self) -> str:
        """:returns string decoded with password"""
        p = creds_rtypes.get('password').encode('utf-8')
        s = base64.b64decode(p)
        return str(s.decode('utf-8'))

    def continue_it(self):
        """Function to check password and continue if correct"""
        if self.pass_entry.get() == self.convert_pwd():
            if creds_rtypes['room_def'] is False:
                creds_rtypes["log"] = 'on'
                write_to_json()
                self.controller.current_visible_frame = RoomTypesPage
                self.controller.show_frame(RoomTypesPage)
            else:
                creds_rtypes["log"] = 'on'
                write_to_json()
                self.controller.current_visible_frame = BookRoomPage
                self.controller.show_frame(BookRoomPage)
        else:
            tk_mb.showerror('M.Y. Hotel', 'Error!\nEntered passwords do not match,
try again.')
            self.pass_entry.focus_force()

    def up_date(self):
        """Keeps the name of hotel updated"""
        self.name_label.configure(text=f'Welcome to {creds_rtypes["name"]}!')
        self.after(5000, self.up_date)
```

If the user hasn't signed-up (or if by any means the JSON file in the `data` folder is removed), the `DataEntryPage` will be shown.

DataEntryPage:



Working Code :

```
class DataEntryPage(Frame):
    __slots__ =
['controller', 'container_frame', 'title_label', 'name_label', 'name_entry', 'passwor
d_label',
                 'password_entry', 'pass_confirm_label', 'pass_confirm_entry', 'conti
nue_button']

    def __init__(self, parent, controller):
        """
        Class for entering credentials / signup
        :param parent: parent frame
        :param controller: parent window
        """
        Frame.__init__(self, parent)
        self.controller = controller

        self.container_frame = Frame(self, width=1366, height=768, bd=0,
relief=SUNKEN, bg='gray90')
        self.title_label = Label(self.container_frame, text='Enter credentials to
continue', width=44, height=3, bd=0,
                                 font=('Consolas', 35, 'bold'), bg='gray90')
        self.name_label = Label(self.container_frame, height=2, text='Hotel Name:',
```

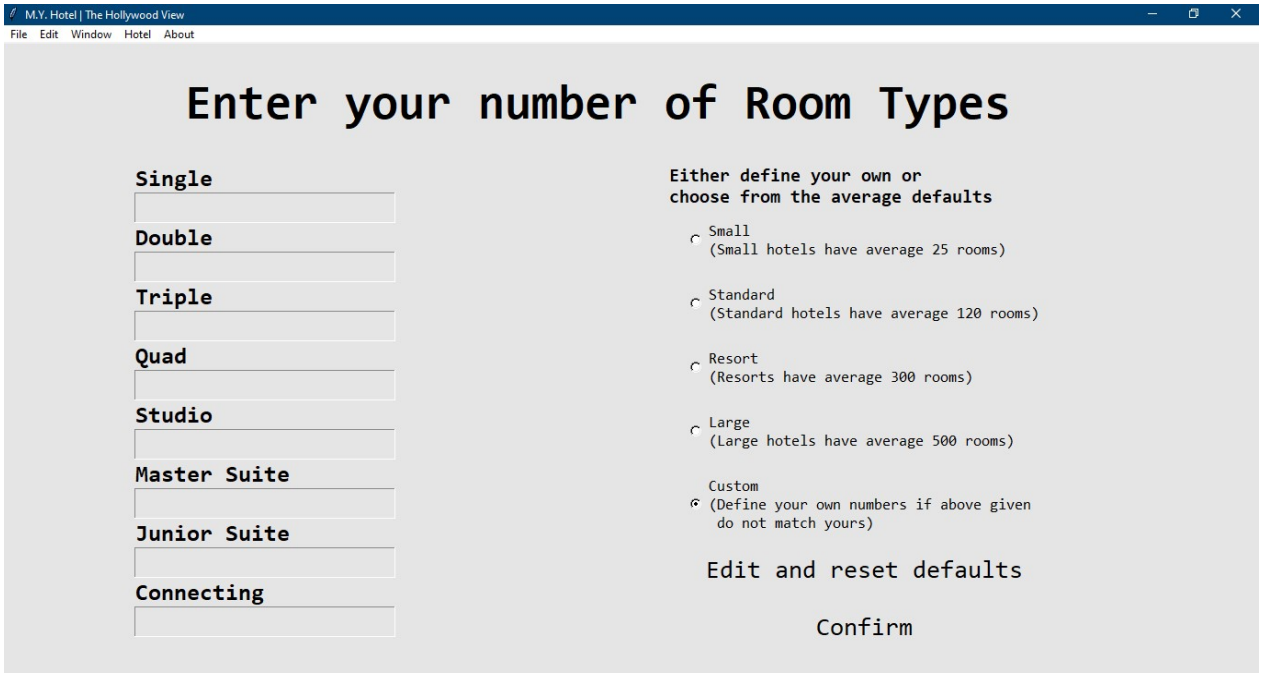
```
bd=0, bg='gray90',
                                font=('Lucida Sans Typewriter', 25, 'bold'),
justify=LEFT)
        self.name_entry = Entry(self.container_frame, width=55, bd=1, font=('Lucida
Sans Typewriter', 25, 'bold'),
                                bg='gray90')
        self.password_label = Label(self.container_frame, height=2, text='Enter
Password:', bd=0, bg='gray90',
                                    font=('Lucida Sans Typewriter', 25, 'bold'),
justify=LEFT)
        self.password_entry = Entry(self.container_frame, width=55, font=('Lucida
Sans Typewriter', 25, 'bold'),
                                    show='•', bd=1, bg='gray90')
        self.pass_confirm_label = Label(self.container_frame, text='Confirm Your
password:', bd=0, bg='gray90',
                                        font=('Lucida Sans Typewriter', 25, 'bold'),
justify=LEFT)
        self.pass_confirm_entry = Entry(self.container_frame, width=55, font=('Lucida
Sans Typewriter', 25, 'bold'),
                                        show='•', bd=1, bg='gray90')
        self.continue_button = Button(self.container_frame, width=71, height=2,
text='Continue ->', bd=0, bg='gray90',
                                      font=('Consolas', 22), fg='#00ff00',
command=lambda: self.assign_data())

        self.container_frame.pack(anchor=N, expand=True)
        self.title_label.pack(anchor=N, side=TOP, expand=True)
        self.name_label.pack(anchor=W, padx=75, ipadx=20)
        self.name_entry.pack(anchor=W, padx=75, ipadx=20)
        self.password_label.pack(anchor=W, padx=75, ipadx=20)
        self.password_entry.pack(anchor=W, padx=75, ipadx=20)
        self.pass_confirm_label.pack(anchor=W, padx=75, ipadx=20, ipady=4)
        self.pass_confirm_entry.pack(anchor=W, padx=75, ipadx=20)
        self.continue_button.pack(side=BOTTOM, anchor=S)

    def continue_with_data(self):
        """
        Function to further continue with, if conditions and data
        are satisfied by the previous function.
        """
        if len(creds_rtypes['name']) < 1:
            tk_mb.showerror('M.Y. Hotel', 'Credentials not found')
            return
        self.controller.update_name(creds_rtypes['name'], creds_rtypes['password'])
        if creds_rtypes['hr_time'] == '':
            pass
        else:
            creds_rtypes['hr_time'] = time.strftime('%Y-%m-%d')
        write_to_json()
        if creds_rtypes['room_def'] is False or creds_rtypes['sel_h_type']
== 'custom':
            self.controller.current_visible_frame = RoomTypesPage
            self.controller.show_frame(RoomTypesPage)
        else:
            self.controller.current_visible_frame = BookRoomPage
```

```
            self.controller.show_frame(BookRoomPage)

    def convert_pwd(self, pass_wd: str) -> str:
        """
        :param pass_wd: password to be encoded
        :returns base64 encoded pass_wd
        """
        enc_pw = base64.b64encode(pass_wd.encode('utf-8'))
        return enc_pw.decode('utf-8')

    def assign_data(self):
        """Function to check name, signup and update credentials"""
        if len(self.name_entry.get()) <= 3:
            name_warning = tk_mb._show('M.Y. Hotel', 'Too short for a name !\nKeep it
anyway?', 'warning', 'yesno')

            if name_warning == 'yes':
                self.controller.update_name(self.name_entry.get(), self.convert_pwd(s
elf.password_entry.get()))
            else:
                return

        if self.password_entry.get() == self.pass_confirm_entry.get():
            creds_rtypes['name'] = self.name_entry.get()
            creds_rtypes['password'] = self.convert_pwd(self.password_entry.get())
            self.continue_with_data()
        else:
            tk_mb.showerror('M.Y. Hotel', 'Passwords do not match!')
            return
```

To use any already existing data user might have to restart the application after adding the data folder named `data` to the folder where the application is.

After pressing Continue button, if the Number of Room Types are not defined the RoomTypesPage will be shown. By default, all the entries will be empty and hotel type will be set to Custom.

RoomTypesPage :



Working Code :

```
class RoomTypesPage(Frame):
    __slots__ =
['controller', 'win_frame', 'name_label', 'r_frame', 'l_frame', 'type_list', 'sv', 'v
',
                'hot_types', 'hot_label', 'confirm_button', 'edit_button']

    def __init__(self, parent, controller):
        """
        Class for defining room types
        :param parent: parent frame
        :param controller: parent window
        """
        Frame.__init__(self, parent)
        self.controller = controller

        if creds_rtypes['log'] == 'off':
            self.controller.current_visible_frame = LoginPage
            self.controller.show_frame(LoginPage)

        self.win_frame = Frame(self, bd=0, bg='gray90')
        self.name_label = Label(self.win_frame, text='Enter your number of Room
Types', bd=0, width=35, height=2,
                                font=('Consolas', 40, 'bold'), bg='gray90')
        self.r_frame = Frame(self.win_frame, bd=4, bg='gray90')
        self.l_frame = Frame(self.win_frame, bd=4, bg='gray90')

        self.win_frame.pack(anchor=N, expand=True)
```

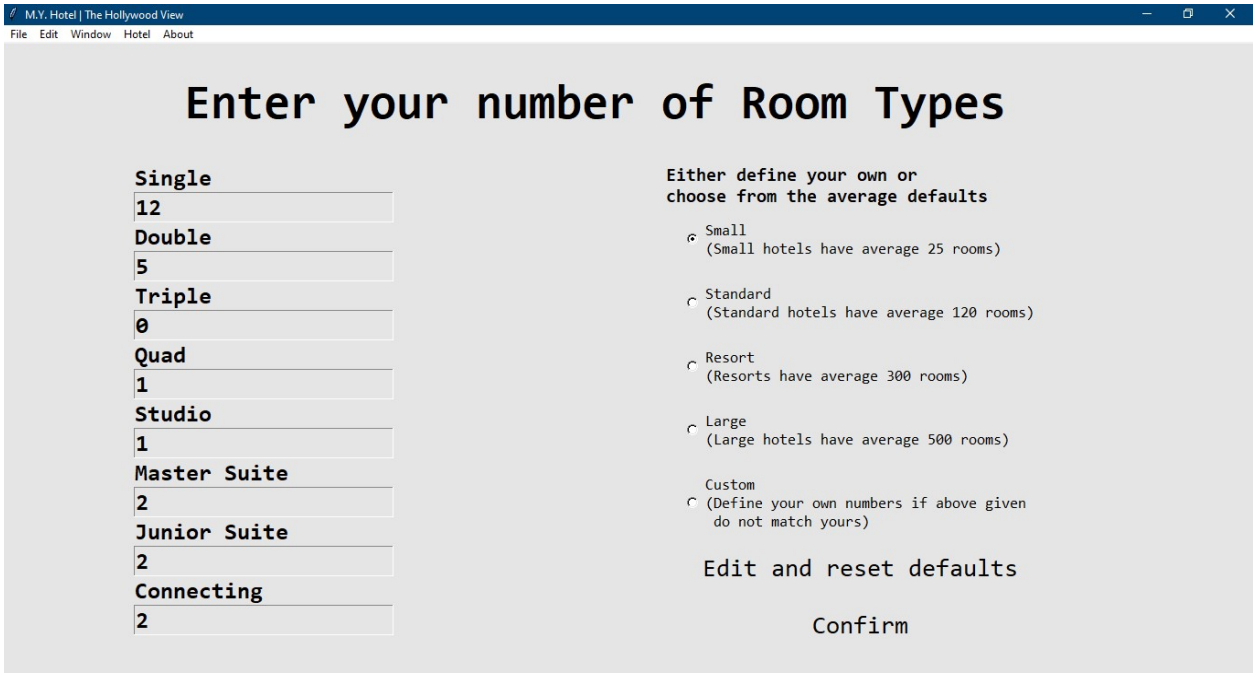
```
        self.name_label.pack(anchor=N)
        self.r_frame.pack(side=RIGHT, fill=BOTH)
        self.l_frame.pack(side=LEFT, fill=BOTH)

        self.type_list = ['Single', 'Double', 'Triple', 'Quad', 'Studio', 'Master
Suite', 'Junior Suite', 'Connecting']

        # String Variables for entries
        self.sv = list(StringVar() for i in range(8))

        # defining room type labels and entries
        for val, tex in enumerate(self.type_list):
            Label(self.l_frame, text=tex, bd=0,
font=('Consolas', 19, 'bold'), bg='gray90').pack(anchor=W)
            Entry(self.l_frame, bd=1,
font=('Consolas', 19, 'bold'), textvariable=self.sv[val], bg='gray90').pack(
                anchor=W)

        # Integer variable for radiocommands
        self.v = IntVar()
        self.v.set(4)

        # hotel types and small desciptwions for averages
        self.hot_types = ['Small\n(Small hotels have average 25 rooms)',
                          'Standard\n(Standard hotels have average 120 rooms)',
                          'Resort\n(Resorts have average 300 rooms)',
                          'Large\n(Large hotels have average 500 rooms)',
                          'Custom\n(Define your own numbers if above given\n do not
match yours)']
        self.hot_label = Label(self.r_frame, bd=0, font=('Consolas', 15, 'bold'),
justify=LEFT, bg='gray90',
                               text='Either define your own or \nchoose from the
average defaults')
        self.hot_label.pack(anchor=W)

        # radiobuttons for hotel types
        for value, t in enumerate(self.hot_types):
            Radiobutton(self.r_frame, text=t, padx=20, pady=10, variable=self.v,
command=self.r_choice, justify=LEFT,
                        bg='gray90', value=value,
font=('Consolas', 13)).pack(anchor=W)

        self.confirm_button = Button(self.r_frame, text='Confirm', width=25,
height=1, bd=0, font=('Consolas', 20),
                                     command=lambda: self.confirm(), bg='gray90')
        self.edit_button = Button(self.r_frame, text='Edit and reset defaults',
width=25, height=1, bd=0, bg='gray90',
                                  font=('Consolas', 20),
command=lambda: self.edit_reset())

        self.confirm_button.pack(side=BOTTOM, pady=10)
        self.edit_button.pack(side=BOTTOM)

    def confirm(self, event='none'):
        """
```

```
        Function to set the hotel type
        :param event: passed by functions and binders
        """
        if self.v.get() == 0:
            creds_rtypes['sel_h_type'] = 'small'

        elif self.v.get() == 1:
            creds_rtypes['sel_h_type'] = 'standard'

        elif self.v.get() == 2:
            creds_rtypes['sel_h_type'] = 'resort'

        elif self.v.get() == 3:
            creds_rtypes['sel_h_type'] = 'large'

        else:
            creds_rtypes['sel_h_type'] = 'custom'
            creds_rtypes['h_type']['custom'] =
list(self.sv[i].get() for i in range(8))

        # if all the room numbers are defined then 'room_def' is set to True
        creds_rtypes['room_def'] = True
        write_to_json()
        self.controller.current_visible_frame = BookRoomPage
        self.controller.show_frame(BookRoomPage)

    def r_choice(self):
        """Function for radio button choices"""

        def enter(t: str):
            """
            Function to set values in entries on radio button command
            :param t: hotel type
            """
            for i in range(8):
                self.sv[i].set(creds_rtypes['h_type'][t][i])

        if self.v.get() == 0:
            enter('small')
        elif self.v.get() == 1:
            enter('standard')
        elif self.v.get() == 2:
            enter('resort')
        elif self.v.get() == 3:
            enter('large')
        elif self.v.get() == 4:
            enter('custom')

    def edit_reset(self, event='none'):
        """
        Function to reset the default values of number of rooms in JSON file
        :param event: passed by functions and binders
        """
        if self.v.get() == 0:
            creds_rtypes['h_type']['small'] =
```

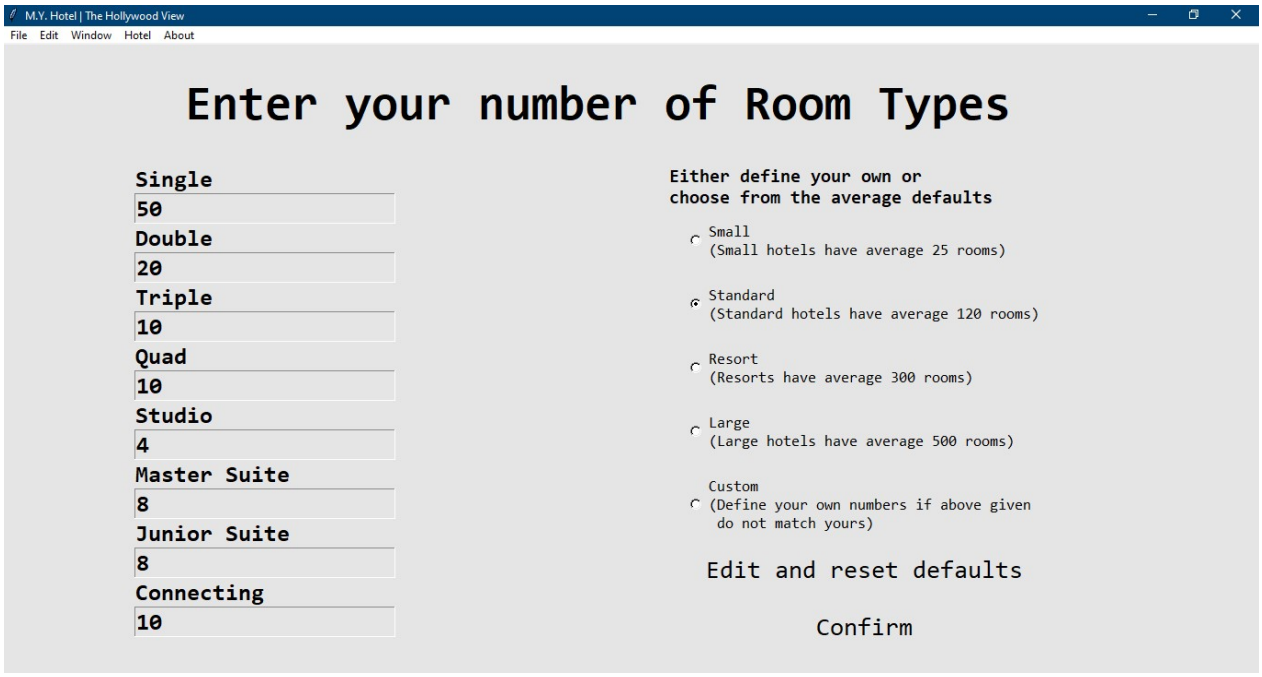
```
list(self.sv[i].get() for i in range(8))
            creds_rtypes['sel_h_type'] = 'small'

        elif self.v.get() == 1:
            creds_rtypes['h_type']['standard'] =
list(self.sv[i].get() for i in range(8))
            creds_rtypes['sel_h_type'] = 'standard'

        elif self.v.get() == 2:
            creds_rtypes['h_type']['resort'] =
list(self.sv[i].get() for i in range(8))
            creds_rtypes['sel_h_type'] = 'resort'

        elif self.v.get() == 3:
            creds_rtypes['h_type']['large'] =
list(self.sv[i].get() for i in range(8))
            creds_rtypes['sel_h_type'] = 'large'

        else:
            tk_mb.showerror('M.Y. Hotel', "You can only define the numbers.\n(Can't
reset in custom mode!)")
            return

        write_to_json()
```

Either the user can enter their own number of rooms in each entry field or the radiobuttons can be used to enter the number of rooms by the average, default number of rooms (taken through global surveys). Also the numbers can be changed any time later in the application.

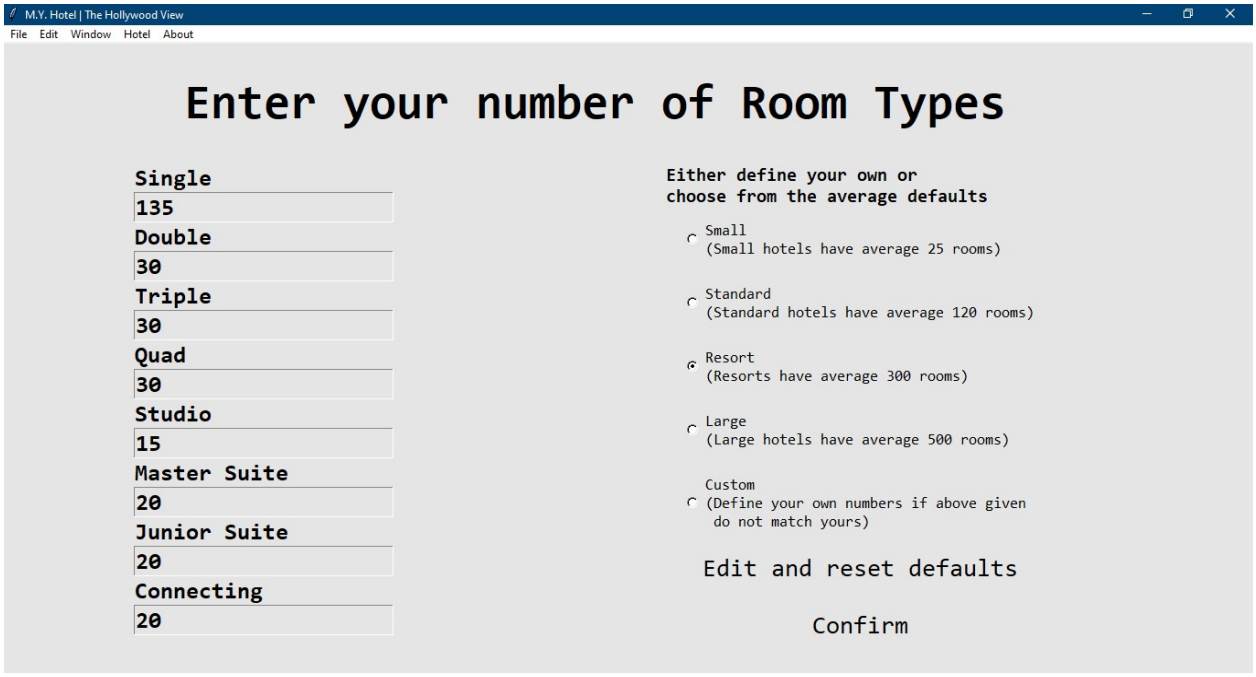
Small Hotels:



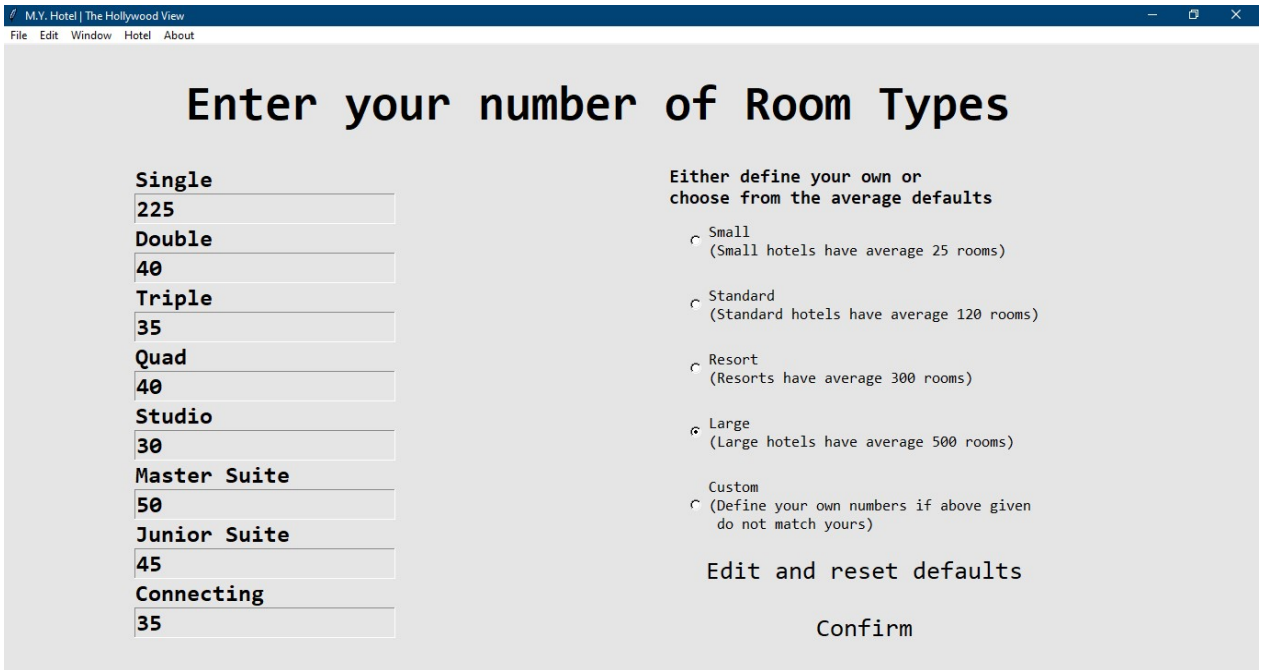
Standard Hotels:



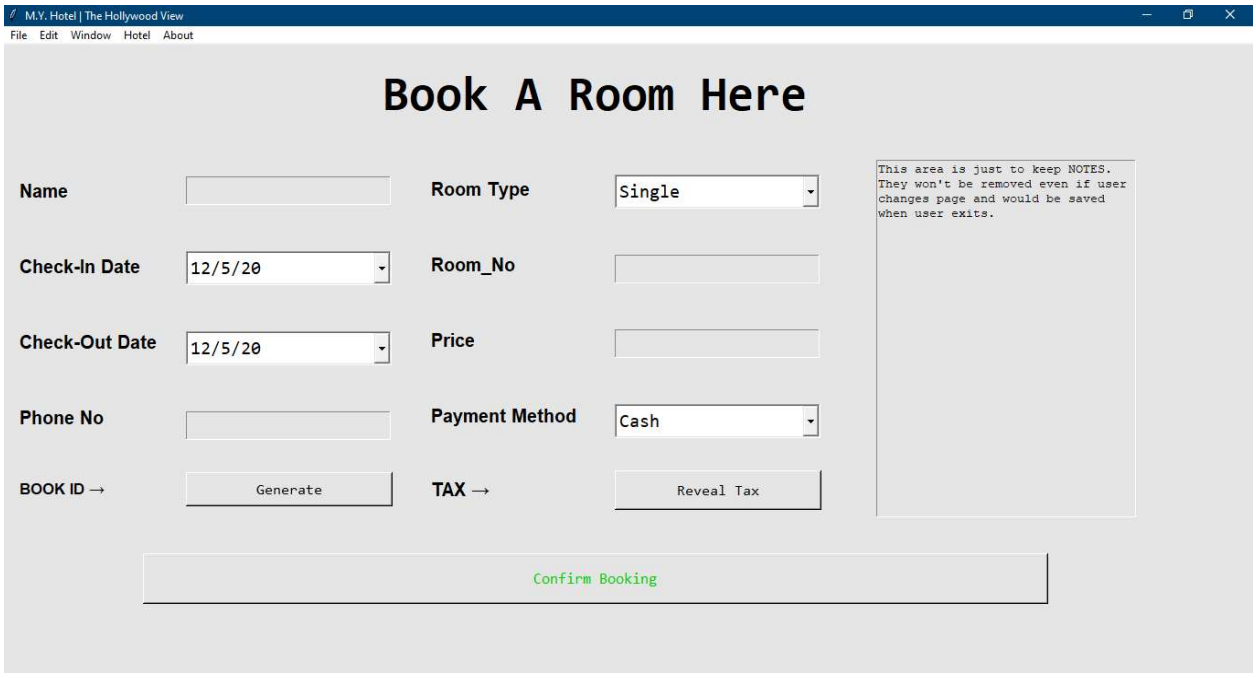
Resorts:



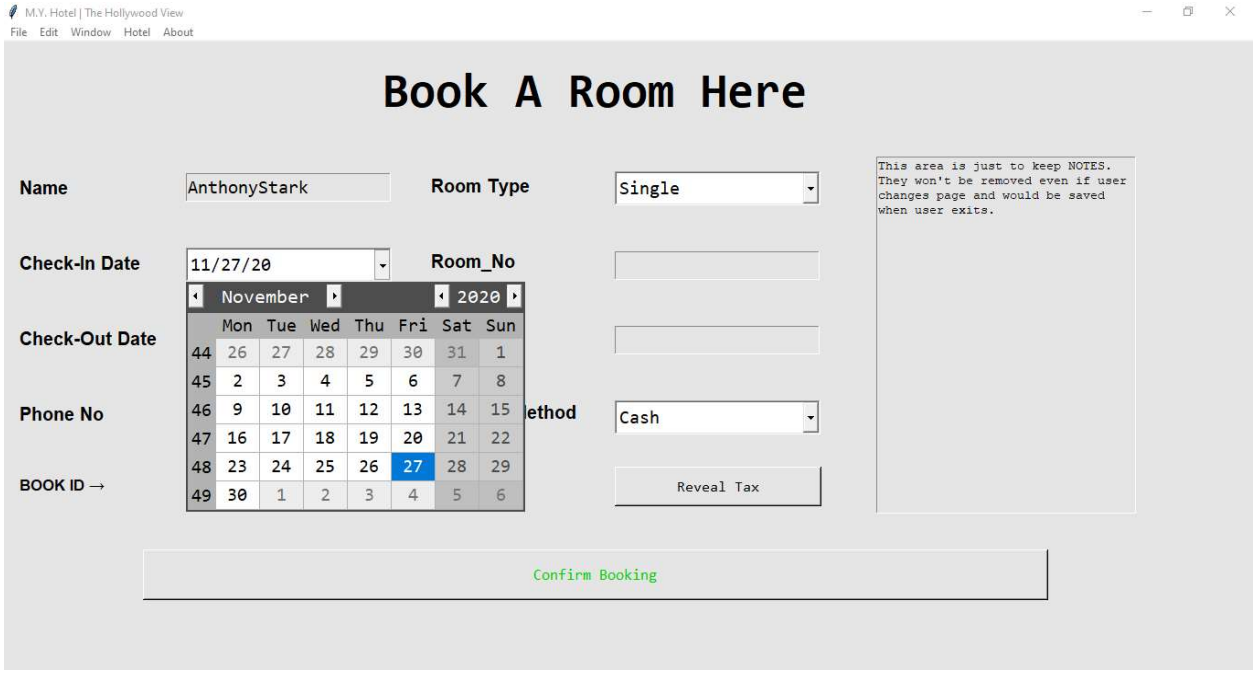
Large Hotels:



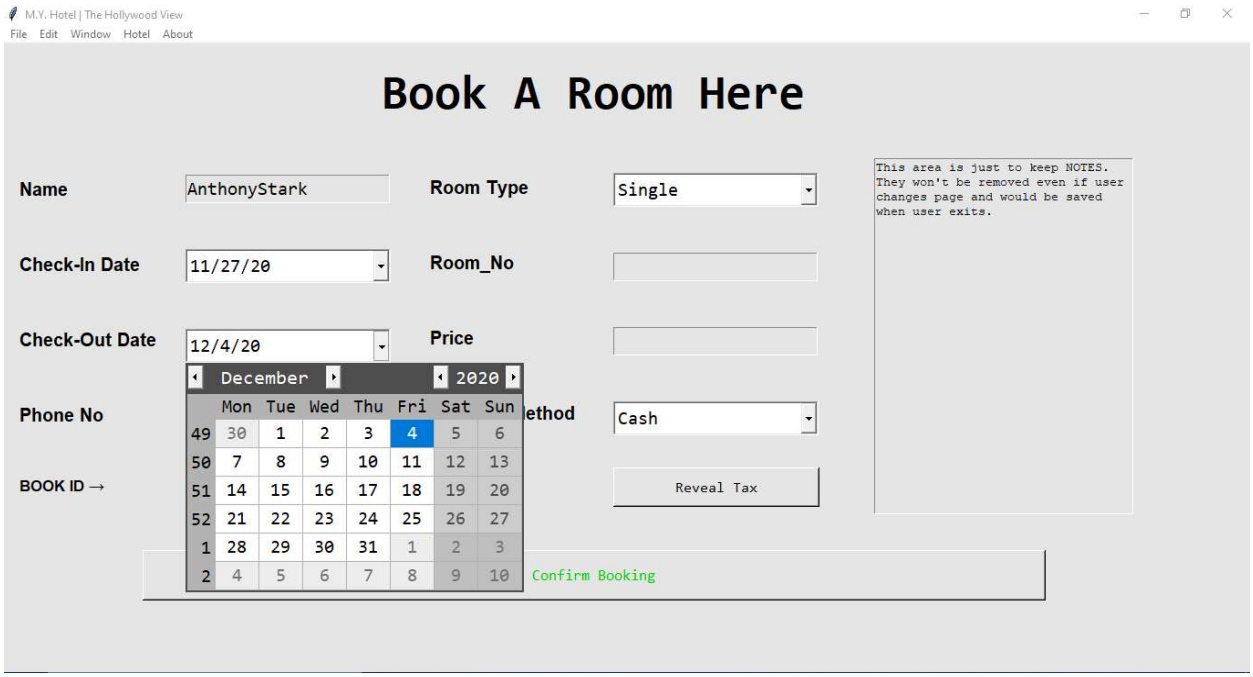
Booking rooms:



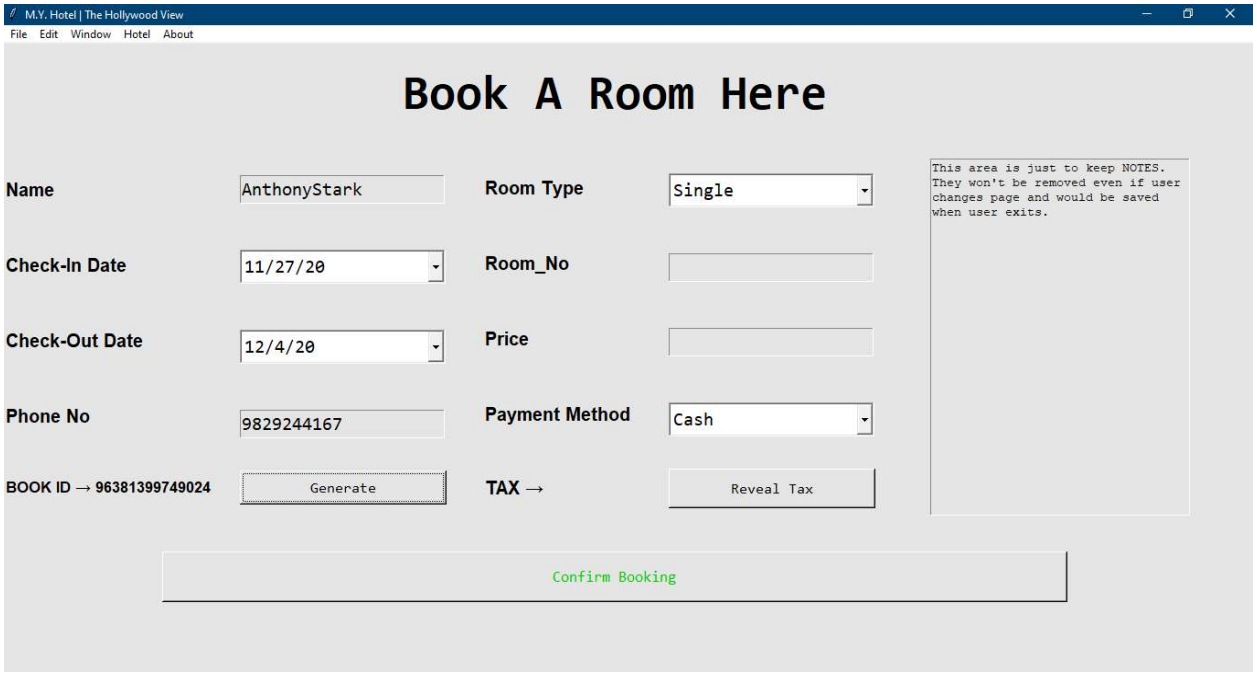
Calendar for Check-in date:



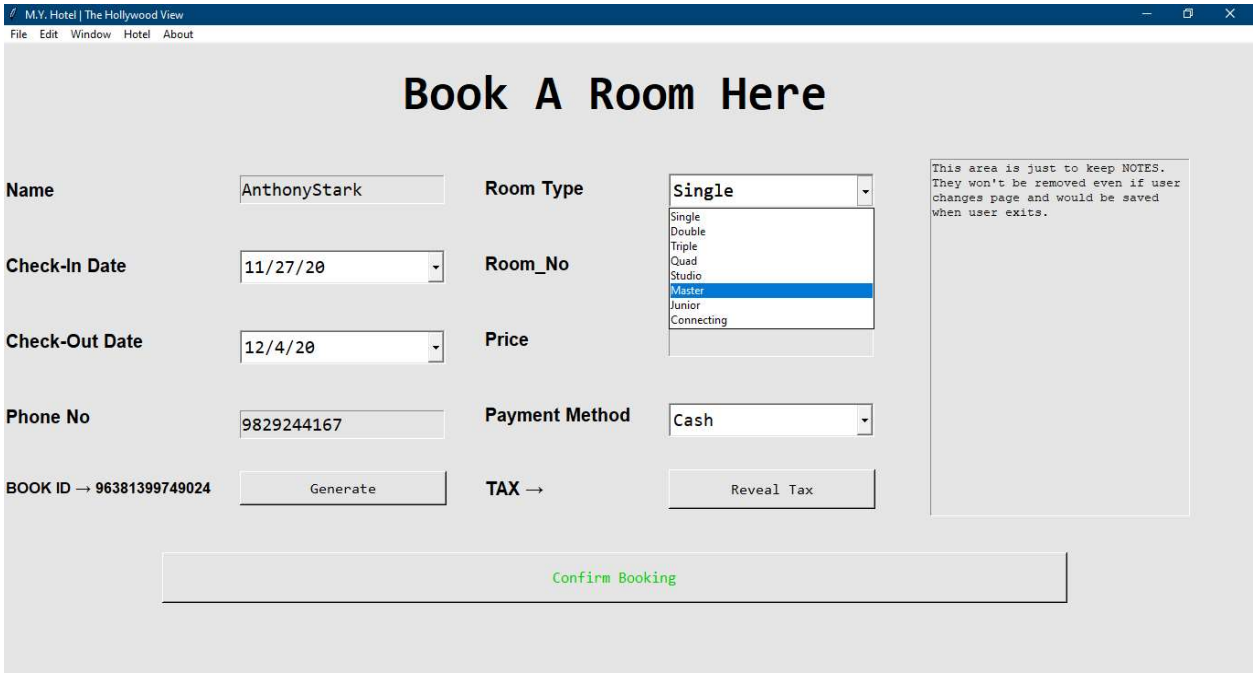
Calendar for Check-out date:



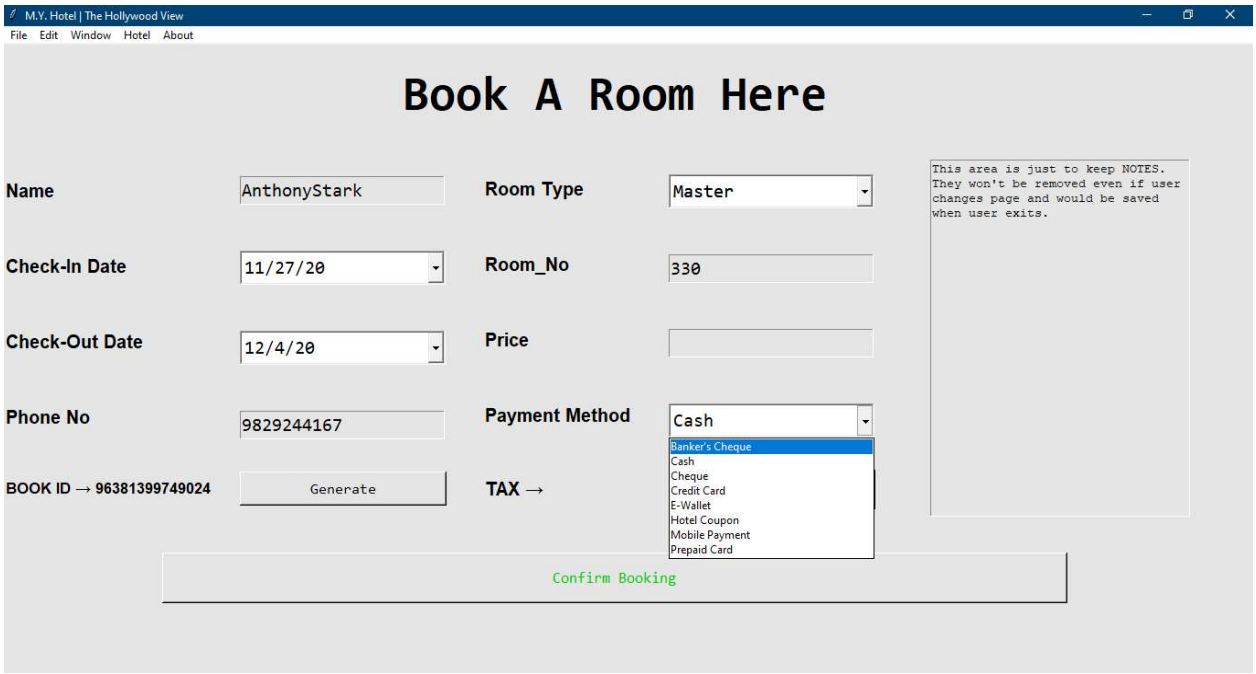
Generate Book ID:



Spinbox for Room Type selection:

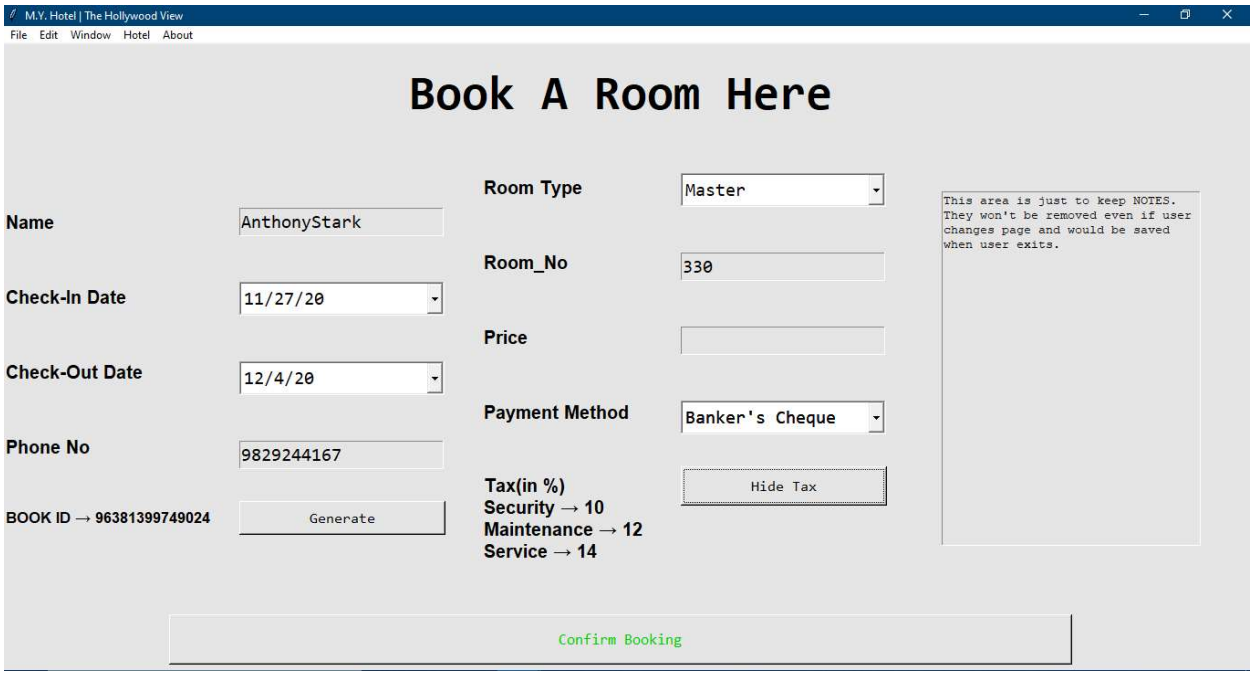


Spinbox for selecting Payment Method:

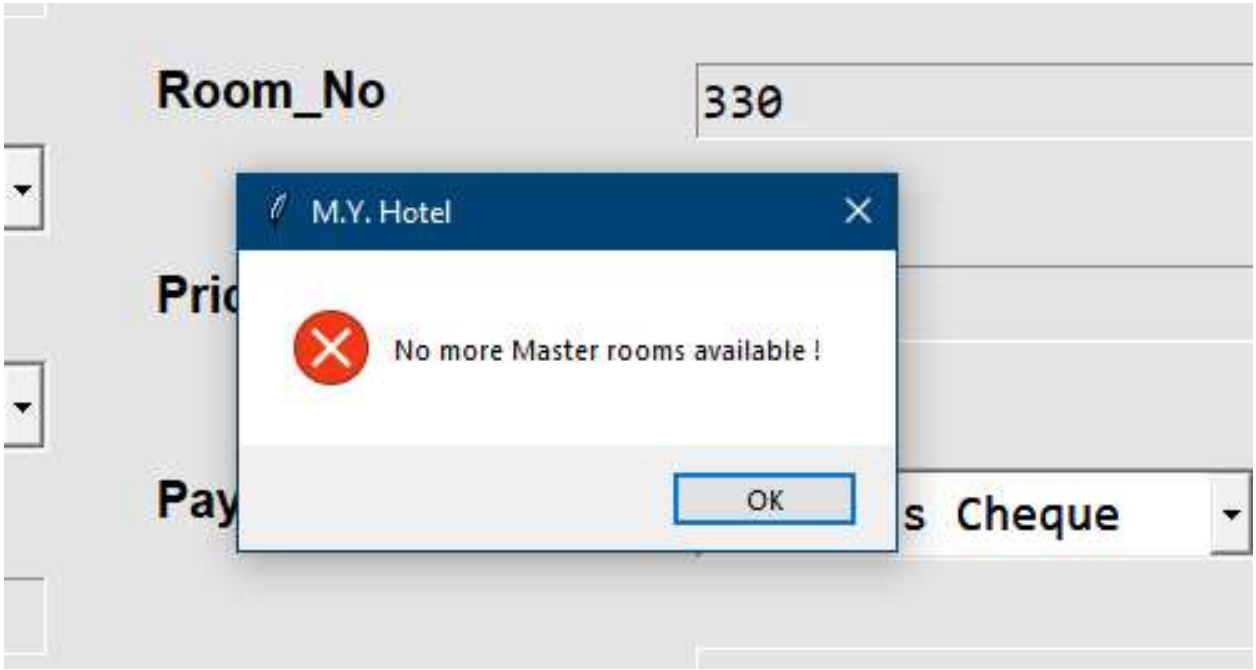


Show/Hide taxes:

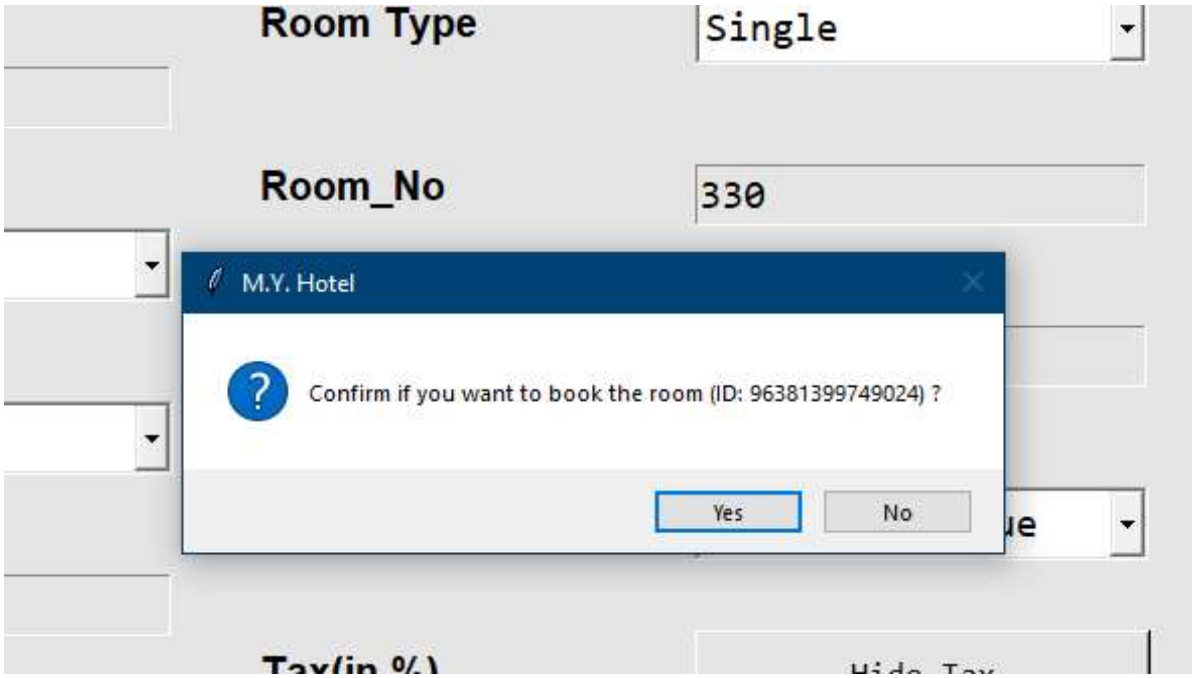
(NOTE: Reveal tax only when needed, as it may disrupt the UI, keeping it hidden is recommended unless the screen resolution is >= "1366x768")

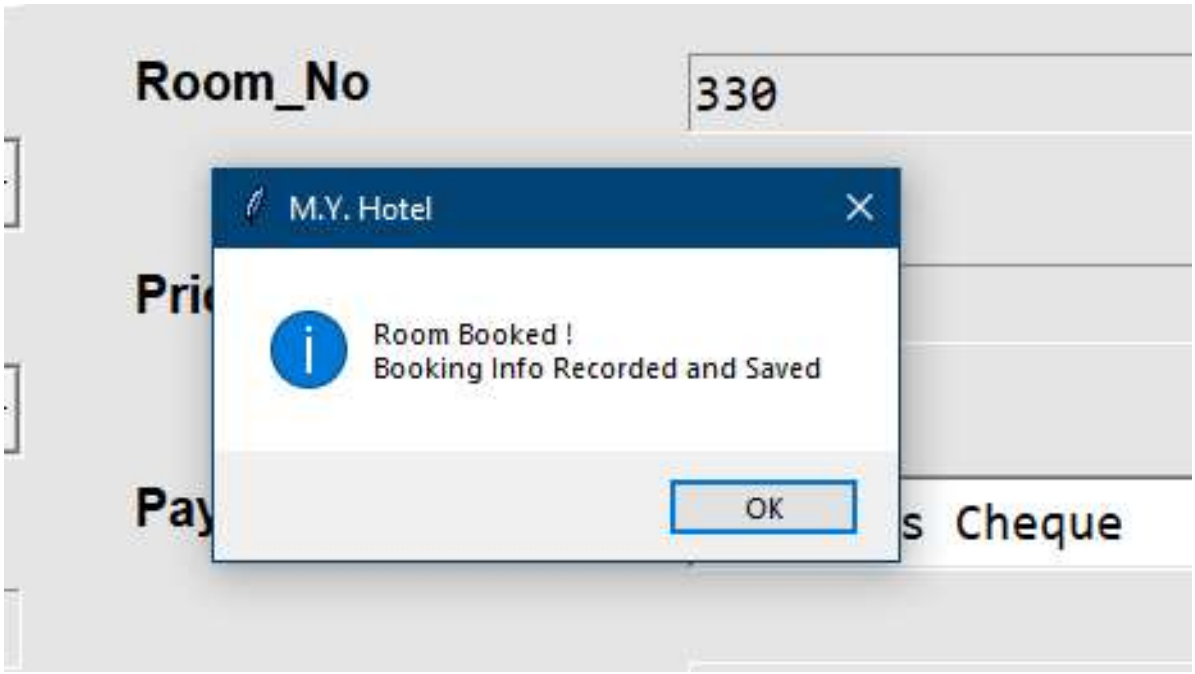


In the above case selected 'Room Type' is 'Master' of which all the rooms have been booked so this will display an error:



If any of the other fields have any kind of invalid input or are no longer available, error of similar kind will popup.
Now after changing the 'Room Type' to 'Single' and confirming booking there will be a final confirmation question and then the user will be informed if booking is done:





(Price will be automatically calculated)

Working Code:

```
class BookRoomPage(Frame):
    __slots__ =
['controller', 'frame', 'book_lbl', 'l_frame', 'l_l_frame', 'l_r_frame', 'receipt_fra
me',
                 'r_frame', 'r_l_frame', 'r_r_frame', 'name_vcmd', 'phone_vcmd', 'roo
m_vcmd', 'price_vcmd',
                 'r_v', 'pay_v', 'tax', 'note_txt', 'name_lbl', 'check_in_cal_lbl', '
check_out_cal_lbl',
                 'phone_lbl', 'book_id_lbl', 'room_type_lbl', 'rnum_lbl', 'price_lbl'
, 'tax_lbl',
                 'pay_method_lbl', 'name_entry', 'phone_entry', 'rnum_entry', 'price'
, 'check_in_cal',
                 'check_out_cal', 'book_id_gen_btn', 'rev_tax', 'book_btn', 'room_com
bo', 'pay_method', 'id']

    def __init__(self, parent, controller):
        """
        Class for entering data for booking Room
        :param parent: parent frame
        :param controller: parent window
        """
        Frame.__init__(self, parent)
        self.controller = controller

        if creds_rtypes['log'] == 'off':
            self.controller.current_visible_frame = LoginPage
            self.controller.show_frame(LoginPage)

        self.frame = Frame(self, bd=0, bg='gray90')
        self.book_lbl = Label(self.frame, text='Book A Room Here',
font=('Consolas', 40, 'bold'), justify=CENTER,
                              bd=1, bg='gray90')
        self.l_frame = Frame(self.frame, bd=0, bg='gray90')
        self.l_l_frame = Frame(self.l_frame, bd=0, bg='gray90')
        self.l_r_frame = Frame(self.l_frame, bd=0, bg='gray90')
        self.receipt_frame = Frame(self.frame, bd=0, bg='gray90')
        self.r_frame = Frame(self.frame, bd=0, bg='gray90')
        self.r_l_frame = Frame(self.r_frame, bd=0, bg='gray90')
        self.r_r_frame = Frame(self.r_frame, bd=0, bg='gray90')

        # ---------------  REGISTERED COMMANDS  ------------------ #
        self.name_vcmd = (self.l_r_frame.register(self.name_check))
        self.phone_vcmd = (self.frame.register(self.phone_check))
        self.room_vcmd = (self.r_r_frame.register(self.room_check))
        self.price_vcmd = (self.r_r_frame.register(self.price_check))
        self.r_v = StringVar()
        self.pay_v = StringVar()
        self.tax = list(creds_rtypes['tax'].values())[0]

        self.note_txt = Text(self.receipt_frame, bd=1, width=35, height=24,
```

```
bg='gray90', wrap=WORD)
        self.note_txt.insert('1.0', creds_rtypes['notes'])

        # ----------------  LABELS  ------------------- #
        self.name_lbl = Label(self.l_l_frame, text='Name', bd=1,
font=('Helvetica', 15, 'bold'), bg='gray90')
        self.check_in_cal_lbl = Label(self.l_l_frame, text='Check-In Date',
font=('Helvetica', 15, 'bold'), bd=1,
                                      bg='gray90')
        self.check_out_cal_lbl = Label(self.l_l_frame, text='Check-Out Date',
font=('Helvetica', 15, 'bold'), bd=1,
                                       bg='gray90')
        self.phone_lbl = Label(self.l_l_frame, bd=1, text='Phone No',
font=('Helvetica', 15, 'bold'), bg='gray90')
        self.book_id_lbl = Label(self.l_l_frame, text=f'BOOK ID {chr(129130)} ',
font=('Helvetica', 13, 'bold'),
                                 bd=1, bg='gray90')
        self.room_type_lbl = Label(self.r_l_frame, text='Room Type', bd=1,
font=('Helvetica', 15, 'bold'), bg='gray90')
        self.rnum_lbl = Label(self.r_l_frame, bd=1, text='Room_No',
font=('Helvetica', 15, 'bold'), bg='gray90')
        self.price_lbl = Label(self.r_l_frame, bd=1, text='Price',
font=('Helvetica', 15, 'bold'), bg='gray90')
        self.tax_lbl = Label(self.r_l_frame, text=f'TAX {chr(129130)} ', bd=1,
font=('Helvetica', 15, 'bold'),
                             bg='gray90')
        self.pay_method_lbl = Label(self.r_l_frame, text='Payment Method', bd=1,
font=('Helvetica', 15, 'bold'),
                                    bg='gray90')

        # ----------------  ENTRIES  ------------------- #
        self.name_entry = Entry(self.l_r_frame, bd=1, font=('Consolas', 15),
validatecommand=(self.name_vcmd, '%P'),
                                validate=ALL)
        # %P is used when the input is what all is allowed
        self.phone_entry = Entry(self.l_r_frame, bd=1, font=('Consolas', 15),
validatecommand=(self.phone_vcmd, '%P'),
                                 validate=ALL)
        self.rnum_entry = Entry(self.r_r_frame, bd=1, font=('Consolas', 15),
validatecommand=(self.room_vcmd, '%P'),
                                validate=ALL)
        self.price = Entry(self.r_r_frame, bd=1, font=('Consolas', 15),
validatecommand=(self.price_vcmd, '%P'),
                           validate=ALL)

        # ----------------  DATE ENTRIES  ------------------- #
        self.check_in_cal = DateEntry(self.l_r_frame, width=18,
day=int(time.strftime('%d')), font=('Consolas', 15),
                                      month=int(time.strftime('%m')),
year=int(time.strftime('%Y')))
        self.check_out_cal = DateEntry(self.l_r_frame, width=18,
day=int(time.strftime('%d')), font=('Consolas', 15),
                                       month=int(time.strftime('%m')),
year=int(time.strftime('%Y')))
```

```
        # ----------------  BUTTONS  ------------------- #
        self.book_id_gen_btn = Button(self.l_r_frame, width=24, text='Generate',
font=('Consolas', 12),
                                      command=lambda: self.gen_id())
        self.rev_tax = Button(self.r_r_frame, text='Reveal Tax',
font=('Consolas', 12), command=lambda: self.veal_tax(),
                              width=24)
        self.book_btn = Button(self, text='Confirm Booking', font=('Consolas', 13),
fg='green3', width=100,
                               command=lambda: self.book(), height=2)

        # ----------------  COMBOBOXES  ------------------- #
        self.room_combo = Combobox(self.r_r_frame, width=18, textvariable=self.r_v,
font=('Consolas', 15))
        self.room_combo['values'] =
['Single', 'Double', 'Triple', 'Quad', 'Studio', 'Master', 'Junior', 'Connecting']
        self.room_combo.current(0)
        self.pay_method = Combobox(self.r_r_frame, font=('Consolas', 15), width=18,
textvariable=self.pay_v)
        self.pay_method['values'] = ['Banker\'s Cheque', 'Cash', 'Cheque', 'Credit
Card', 'E-Wallet', 'Hotel Coupon',
                                     'Mobile Payment', 'Prepaid Card']
        self.pay_method.current(1)

        self.id = None
        self.up_date()

    def pack_all(self):
        """
        Packs (displays) all the widgets when frame is raised
        or focussed upon.
        """
        self.frame.pack(anchor=N, pady=10, ipadx=10)
        self.book_lbl.pack(anchor=N, pady=10, ipadx=10)
        self.l_frame.pack(side=LEFT, pady=10, ipadx=10)
        self.l_l_frame.pack(side=LEFT, anchor=N, pady=10, ipadx=5)
        self.l_r_frame.pack(side=RIGHT, anchor=N, pady=10, ipadx=10)
        self.receipt_frame.pack(side=RIGHT, pady=10, ipadx=40)
        self.r_frame.pack(side=RIGHT, pady=10, ipadx=10)
        self.r_l_frame.pack(side=LEFT, anchor=N, pady=10, ipadx=10)
        self.r_r_frame.pack(side=RIGHT, anchor=N, pady=10, ipadx=10)
        self.note_txt.pack()
        self.name_lbl.pack(anchor=W, pady=25, ipady=2)
        self.name_entry.pack(anchor=W, pady=25, ipady=2)
        self.check_in_cal_lbl.pack(anchor=W, pady=25, ipady=2)
        self.check_in_cal.pack(anchor=W, pady=25, ipady=2)
        self.check_out_cal_lbl.pack(anchor=W, pady=25, ipady=2)
        self.check_out_cal.pack(anchor=W, pady=25, ipady=2)
        self.phone_lbl.pack(anchor=W, pady=25, ipady=2)
        self.phone_entry.pack(anchor=W, pady=25, ipady=2)
        self.book_id_lbl.pack(anchor=W, pady=25)
        self.book_id_gen_btn.pack(anchor=W, pady=10, ipady=2)
        self.room_type_lbl.pack(anchor=W, pady=25, ipady=2)
        self.room_combo.pack(anchor=W, pady=25, ipady=2)
        self.rnum_lbl.pack(anchor=W, pady=25, ipady=2)
```

```
        self.rnum_entry.pack(anchor=W, pady=25, ipady=2)
        self.price_lbl.pack(anchor=W, pady=25, ipady=2)
        self.price.pack(anchor=W, pady=25, ipady=2)
        self.pay_method_lbl.pack(anchor=W, pady=25, ipady=2)
        self.pay_method.pack(anchor=W, pady=25, ipady=2)
        self.tax_lbl.pack(anchor=W, pady=25)
        self.rev_tax.pack(anchor=W, pady=10, ipady=5)
        self.book_btn.pack(anchor='n', ipadx=40)

    def unpack_all(self):
        """
        Unpacks (hides) all the widgets when focus is removed
        from the frame.
        """
        self.frame.pack_forget()
        self.book_lbl.pack_forget()
        self.l_frame.pack_forget()
        self.l_l_frame.pack_forget()
        self.l_r_frame.pack_forget()
        self.receipt_frame.pack_forget()
        self.r_frame.pack_forget()
        self.r_l_frame.pack_forget()
        self.r_r_frame.pack_forget()
        self.note_txt.pack_forget()
        self.name_lbl.pack_forget()
        self.name_entry.pack_forget()
        self.check_in_cal_lbl.pack_forget()
        self.check_in_cal.pack_forget()
        self.check_out_cal_lbl.pack_forget()
        self.check_out_cal.pack_forget()
        self.phone_lbl.pack_forget()
        self.phone_entry.pack_forget()
        self.book_id_lbl.pack_forget()
        self.book_id_gen_btn.pack_forget()
        self.room_type_lbl.pack_forget()
        self.room_combo.pack_forget()
        self.rnum_lbl.pack_forget()
        self.rnum_entry.pack_forget()
        self.price_lbl.pack_forget()
        self.price.pack_forget()
        self.pay_method_lbl.pack_forget()
        self.pay_method.pack_forget()
        self.tax_lbl.pack_forget()
        self.rev_tax.pack_forget()
        self.book_btn.pack_forget()

    def up_date(self):
        """Updates the entered notes to the credentials dictionary"""
        creds_rtypes['notes'] = self.note_txt.get('1.0', END)
        if self.controller.current_visible_frame == BookRoomPage:
            self.pack_all()
        else:
            self.unpack_all()
        self.after(150, self.up_date)
```

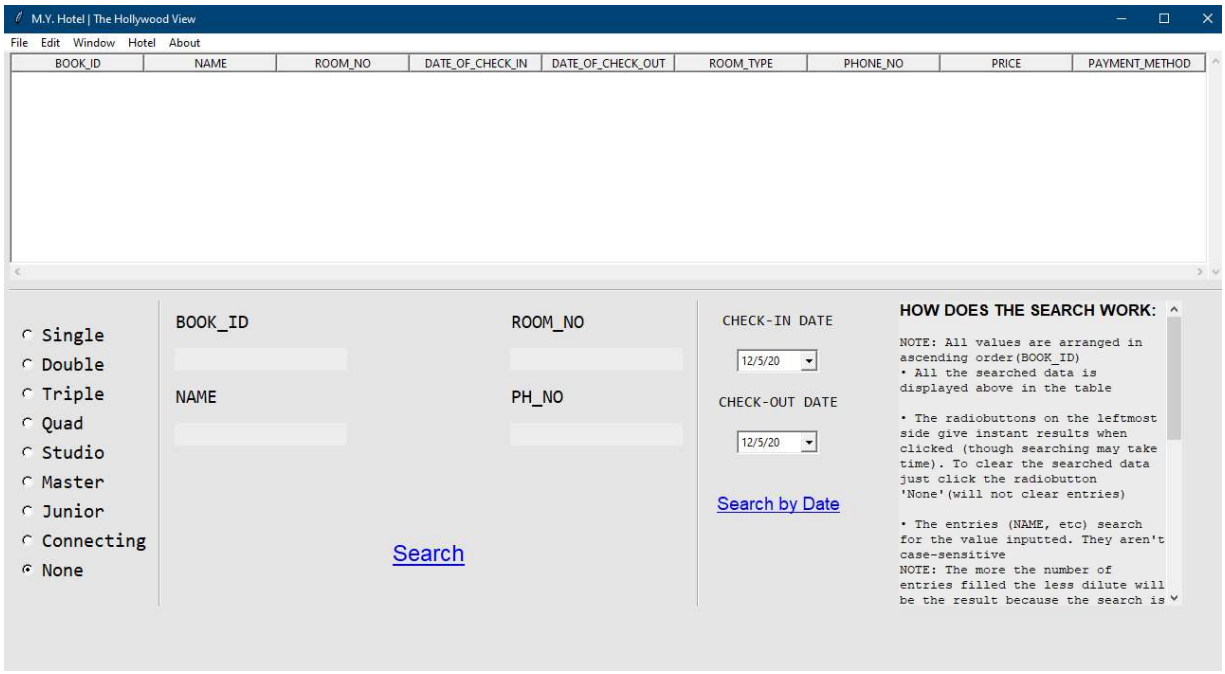
```
    def gen_id(self):
        """Enters the generated ID to booking ID label"""
        self.id = unique_id()
        self.book_id_lbl.config(text='')
        self.book_id_lbl.config(text=f'BOOK ID {chr(129130)} ' + str(self.id))

    def veal_tax(self):
        """Reveal taxes and their values"""
        keys = list(creds_rtypes["tax"].keys())
        vals = list(creds_rtypes["tax"].values())
        if self.rev_tax.cget('text') == 'Reveal Tax':
            self.tax_lbl.configure(text=f'Tax(in
%)\n{keys[0]} {chr(129130)} {vals[0]}\n{keys[1]} {chr(129130)}'
                                       f' {vals[1]}\n{keys[2]} {chr(129130)} '
                                       f'{vals[2]}', justify=LEFT)
            self.rev_tax.config(text='Hide Tax')
        else:
            self.tax_lbl.configure(text='Tax')
            self.rev_tax.config(text='Reveal Tax')

    def name_check(self, event):
        """
        Function to keep user from entering anything but alphabets
        :param event: passed by functions and binders
        """
        if str.isalpha(event) or event == '':
            return True
        else:
            return False

    def phone_check(self, event):
        """
        Function to keep user from entering anything but digits
        :param event: passed by functions and binders
        """
        if str.isdigit(event) or event == '':
            return True
        else:
            return False

    def room_check(self, event):
        """
        Function to keep user from entering anything but digits
        :param event: passed by functions and binders
        """
        if str.isdigit(event) or event == '':
            return True
        else:
            return False

    def price_check(self, event):
        """
        Function to keep user from entering anything but digits
        :param event: passed by functions and binders
        """
```

```
        if str.isdigit(event) or event == '' or event == '.':
            return True
        else:
            return False

    def confirm_booking(self):
        """Confirms booking and resets entry fields"""
        cursor.execute(f'INSERT INTO hotel_info VALUES("{self.id}",
"{self.name_entry.get()}", {self.rnum_entry.get()},'
                       f' "{str(self.check_in_cal.get_date())}",
"{str(self.check_out_cal.get_date())}",'
                       f' "{str(self.room_combo.get())}",
"{str(self.phone_entry.get())}", {str(self.price.get())}, '
                       f'"{str(self.pay_method.get())}");')
        connection.commit()
        showinfo('M.Y. Hotel', 'Room Booked !\nBooking Info Recorded and Saved')

        self.name_entry.delete(0, END)
        self.book_id_lbl.config(text=f'BOOK ID {chr(129130)}')
        self.phone_entry.delete(0, END)
        self.check_in_cal.set_date(datetime.date(year=int(time.strftime('%Y')),
month=int(time.strftime('%m')),
                                                 day=int(time.strftime('%d'))))
        self.check_out_cal.set_date(datetime.date(year=int(time.strftime('%Y')),
month=int(time.strftime('%m')),
                                                  day=int(time.strftime('%d'))))
        self.price.delete(0, END)
        self.rnum_entry.delete(0, END)

    def book(self):
        """Checks all entries and books rooms"""
        if len(self.name_entry.get()) < 2:
            showerror('M.Y. Hotel', 'Reconsider Name field !')
            return
        elif self.check_in_cal.get_date()
== self.check_out_cal.get_date() or self.check_out_cal.get_date() < \
                self.check_in_cal.get_date():
            showerror('M.Y. Hotel', 'Reconsider Check-in - Check-out Dates !\nNOTE:
You can\'t time travel OR jo'
                                    'in and leave on the same day!')
            return
        elif len(self.phone_entry.get()) < 7:
            showerror('M.Y. Hotel', 'Reconsider Phone No field !')
            return
        elif len(self.rnum_entry.get()) < 1:
            showerror('M.Y. Hotel', 'Room No not entered !')
            return
        elif self.id is None:
            showerror('M.Y. Hotel', 'BOOK ID not generated !')
            return
        cursor.execute('SELECT ROOM_NO FROM HOTEL_INFO;')
        data = cursor.fetchall()
        b_rooms = []
        for i in range(len(data)):
            b_rooms.append(data[i][0])
```

```
        if int(self.rnum_entry.get()) in b_rooms:  # if room already booked
            showerror('M.Y. Hotel', 'Room already occupied')
            return
        cursor.execute(f'SELECT ROOM_TYPE FROM HOTEL_INFO WHERE
ROOM_TYPE="{self.room_combo.get()}"')
        data = cursor.fetchall()
        num_booked = len(data)
        avail_rooms =
int(creds_rtypes['h_type'][creds_rtypes['sel_h_type']][room_t.index(self.room_combo.g
et())]) - \
            num_booked
        if avail_rooms < 1:
            showerror('M.Y. Hotel', f'No more {self.room_combo.get()} rooms available
!')
            return
        org_price =
int(creds_rtypes['h_r_price'][creds_rtypes['sel_h_type']][int(room_t.index(self.room_
combo.get()))])
        tax = (int(creds_rtypes['tax'][list(creds_rtypes['tax'].keys())[0]]) / 100)
* org_price + \
              (int(creds_rtypes['tax'][list(creds_rtypes['tax'].keys())[1]]) / 100)
* org_price + \
              (int(creds_rtypes['tax'][list(creds_rtypes['tax'].keys())[2]]) / 100)
* org_price
        self.price.insert(0, f'{int(round(org_price + tax, -1))}')

        def conf():
            """Confirmation function for booking room"""
            ans = askyesno('M.Y. Hotel', f'Confirm if you want to book the room
(ID: {self.id}) ?')
            if ans > 0:
                self.confirm_booking()
                pass
            else:
                return

        self.after(2000, conf)
```

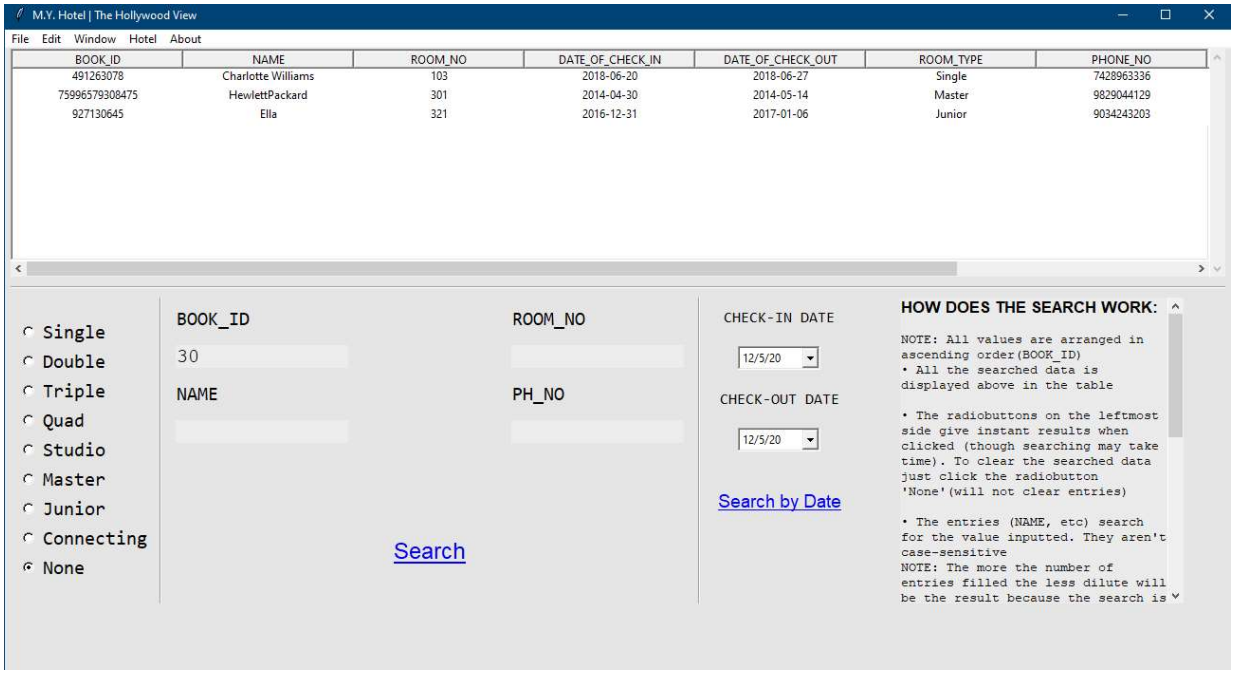
Places where specific type of input is required, like name only requires alphabets, no other type of input can be inserted i.e. numbers can't be inserted in a name.

Search / Cancellation Page

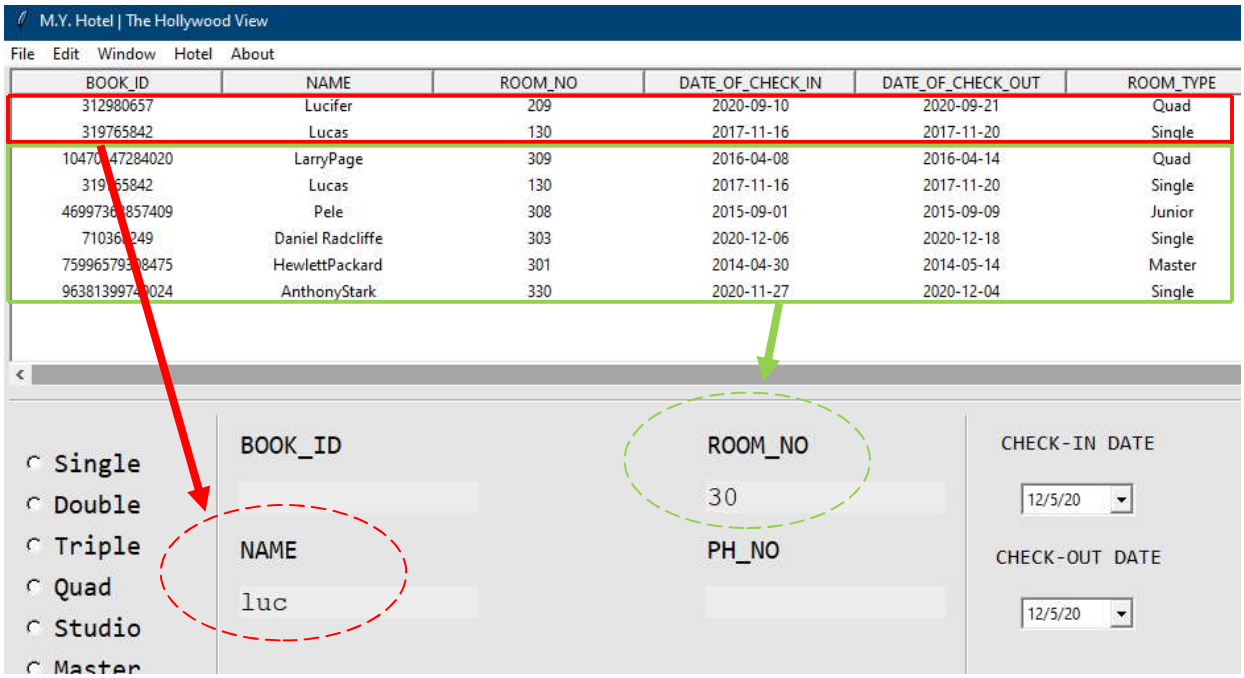


For searching there are three options:

1. Through Room Type Radiobuttons:
2. Through Book ID or Name or Room No or Phone No:
   (Results will be displayed even when more than one field is filled.)



(In the above search, Book IDs which have an occurrence of "30" in them will be displayed)

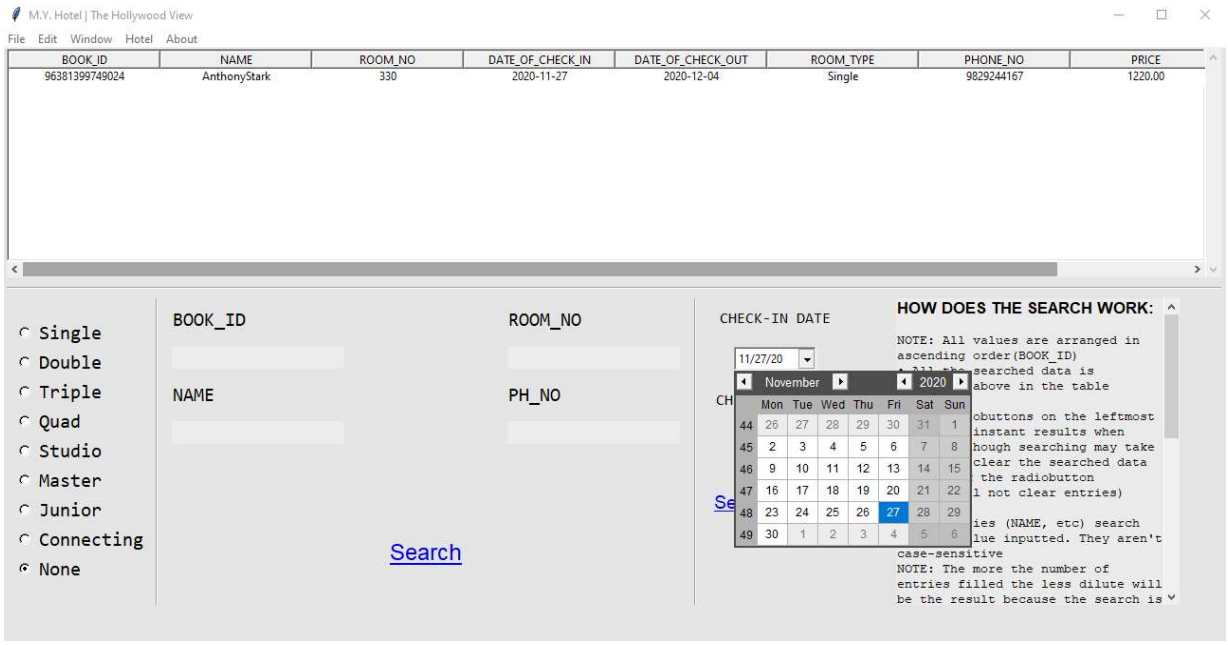


(In the above search, Names which have an occurrence of "Luc" , and Room No.s which have "30" in them will be displayed)

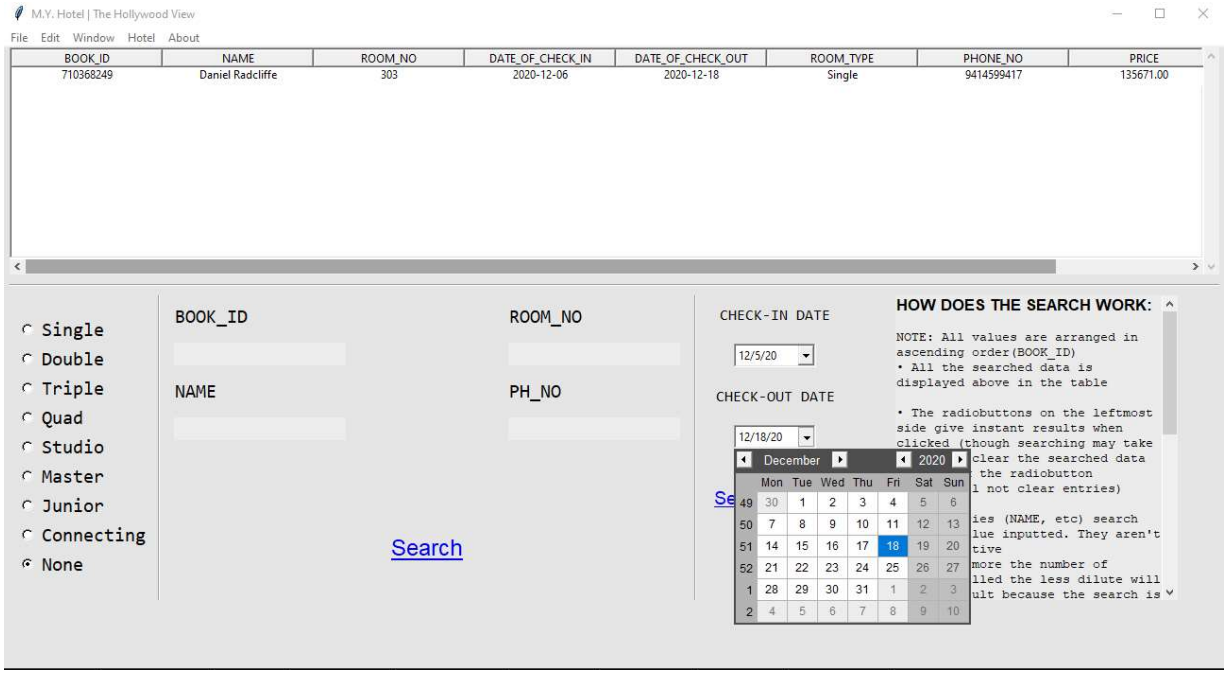
If multiple fields are filled the order in which results will be displayed is:

BOOK_ID → NAME → ROOM_NO → PH_NO

3. Through Check-in / Check-out dates:



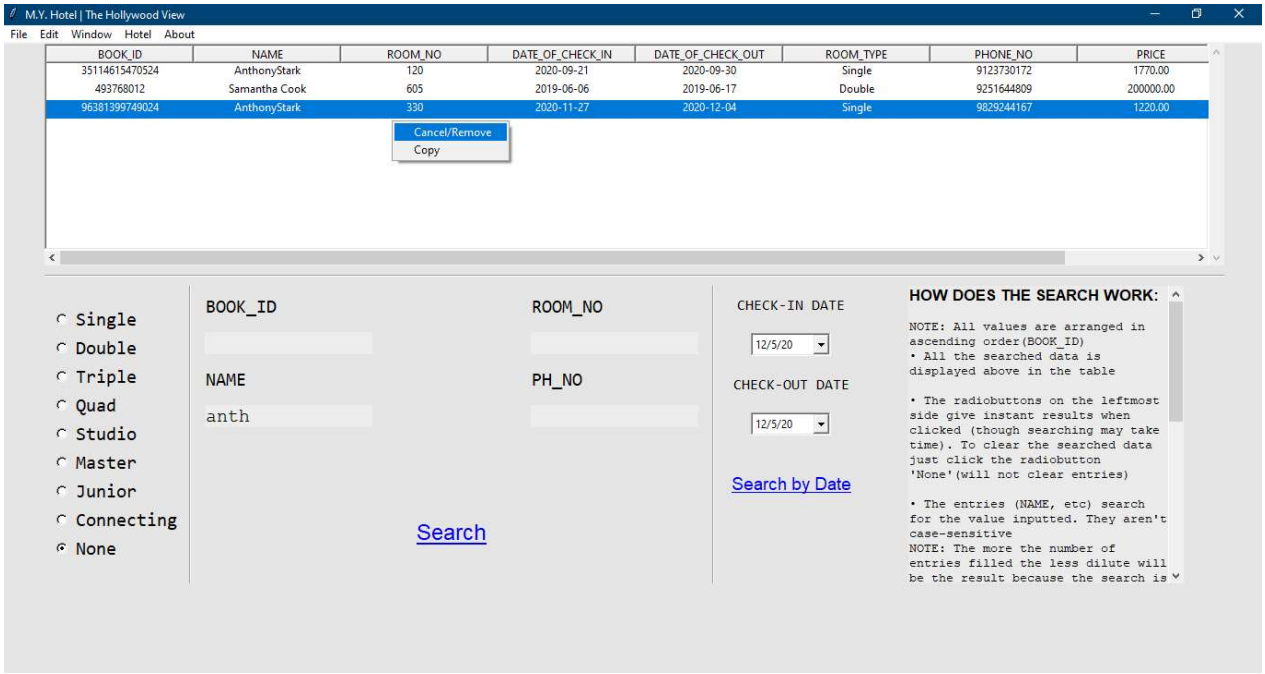
(In the above search, all the check-ins done on "27-11-2020" will be shown)



(In the above search, all the check-outs done on "18-12-2020" will be shown)

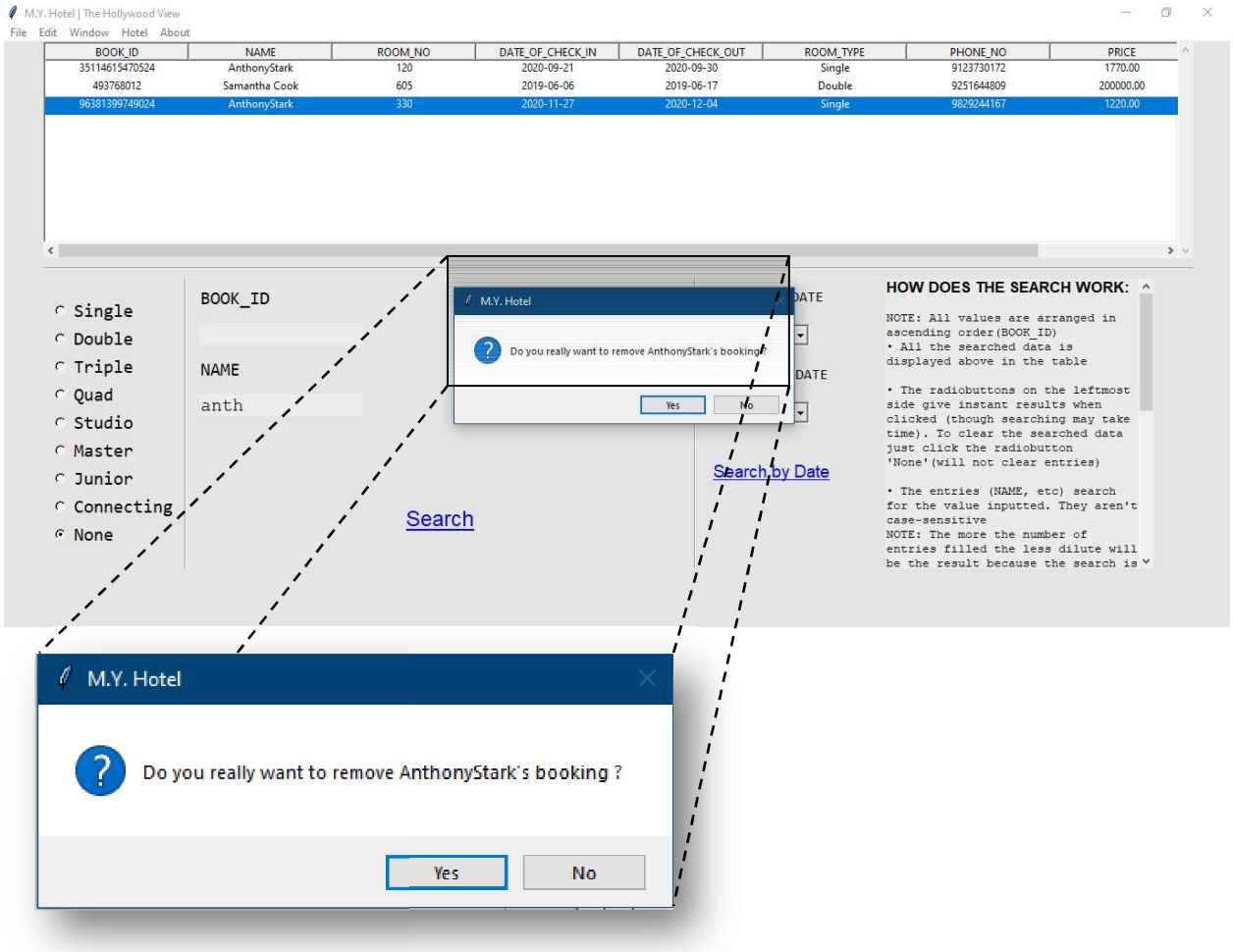
On selecting any row of the search results and right-clicking, a small menu will open up in which `Cancel/Remove` and `Copy` options will be available.

During multiple selection cancelling the bookings will result in cancelling the first selected row whereas all the rows will be copied in multiple selection.



Confirmation for booking removal will be asked. If user clicks on `Yes` , the booking will be removed. Selecting any radiobutton or simply clicking search again will refresh the search page and the cancelled booking will be removed.

A small description for `"How does the search work"` has also been provided (not editable), in the same page.




```
class BrowsePage(Frame):
    __slots__ =
['controller', 'frame', 'b_frame', 'scroll_b1', 'scroll_b2', 'b_list', 'sep', 'style'
,
                 'heading_list', 'index', 'iid', 'l_frame', 'sep1', 'l_mid_frame', 's
earch_label', 'r_mid_frame',
                 'sep2', 'r_frame', 'desc_frame', 'v1', 'v2', 'check_in_label', 'chec
k_in_cal', 'check_out_label',
                 'check_out_cal', 'r_menu', 'description_box', 'date_search_lbl']

    def __init__(self, parent, controller):
        """
        Class for browsing all the data for the selected room
        :param parent: parent frame
        :param controller: parent window
        """
        Frame.__init__(self, parent)
        self.controller = controller
        self.radio_list = []
        self.entry_list = []
        self.label_list = []
```

```
        if creds_rtypes['log'] == 'off':
            self.controller.current_visible_frame = LoginPage
            self.controller.show_frame(LoginPage)

        self.frame = Frame(self, bg='gray90')
        self.b_frame = Frame(self.frame, bd=0, bg='gray90')
        self.scroll_b1 = Scrollbar(self.b_frame)
        self.scroll_b2 = Scrollbar(self.b_frame, orient=HORIZONTAL)

        self.b_list = Treeview(self.b_frame, yscrollcommand=self.scroll_b1.set, xscro
llcommand=self.scroll_b2.set,
                               selectmode=EXTENDED)
        self.sep = Separator(self.frame, orient=HORIZONTAL)

        self.style = Style()
        self.style.configure("Treeview.Heading", font=('Consolas', 10, 'bold'))
        self.style.configure("Treeview.Column", font=('Courier New', 11))
        self.style.theme_use("winnative")

        self.b_list['columns'] = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
        self.b_list["show"] = "headings"
        self.scroll_b1.config(command=self.b_list.yview)
        self.scroll_b2.config(command=self.b_list.xview)

        self.heading_list =
['', 'BOOK_ID', 'NAME', 'ROOM_NO', 'DATE_OF_CHECK_IN', 'DATE_OF_CHECK_OUT', 'ROOM_TYP
E',
                             'PHONE_NO', 'PRICE', 'PAYMENT_METHOD']

        for i in range(len(self.heading_list)):
            self.b_list.heading(str(i), text=self.heading_list[i])
            self.b_list.column(str(i), width=140, anchor=N)

        self.index = self.iid = 0

        self.l_frame = Frame(self.frame, bd=0, bg='gray90')
        self.sep1 = Separator(self.frame, orient=VERTICAL)
        self.l_mid_frame = Frame(self.frame, bd=0, bg='gray90')
        self.search_label = Label(self.frame, bd=0, text='Search', width=10,
font=('Dialog Input', 18, 'underline'),
                                  fg='blue2', cursor='hand2', bg='gray90')
        self.r_mid_frame = Frame(self.frame, bd=0, bg='gray90')
        self.sep2 = Separator(self.frame, orient=VERTICAL)
        self.r_frame = Frame(self.frame, bd=0, bg='gray90')
        self.desc_frame = Frame(self.frame, bd=0, bg='gray90')

        self.v1 = IntVar()
        for value, tex in enumerate(room_t):
            self.radio_list.append(Radiobutton(self.l_frame, variable=self.v1, bd=0,
value=value, cursor='hand2',
                                               font=('Consolas', 15),
text=tex, bg='gray90',

command=lambda: self.rad_btn_cmd(int(self.v1.get())))))
```

```
        self.radio_list.append(Radiobutton(self.l_frame, variable=self.v1, bd=0,
value=8, font='Consolas 15',
                                           bg='gray90',
command=lambda: self.rad_btn_cmd(-1), text='None'))
        self.v1.set(8)

        self.v2 = list(StringVar() for i in range(4))
        for value, tex in enumerate(col_names):
            if value > 1:
                self.label_list.append(Label(self.r_mid_frame, text=tex,
font=('Consolas', 15), bd=0, bg='gray90'))
                self.entry_list.append(Entry(self.r_mid_frame, font=('Courier
New', 15), textvariable=self.v2[value],
                                             bd=0, bg='gray93', width=15))
            else:
                self.label_list.append(Label(self.l_mid_frame, text=tex,
font=('Consolas', 15), bd=0, bg='gray90'))
                self.entry_list.append(Entry(self.l_mid_frame, font=('Courier
New', 15), textvariable=self.v2[value],
                                             bd=0, bg='gray93', width=15))

        self.check_in_label = Label(self.r_frame, text='CHECK-IN DATE',
font=('Consolas', 12), bd=0, bg='gray90')
        self.check_in_cal = DateEntry(self.r_frame, width=10,
day=int(creds_rtypes['hr_time'].split('-')[0]),
                                      month=int(creds_rtypes['hr_time'].split('-
')[1]),
                                      year=int(creds_rtypes['hr_time'].split('-')[-
1]))
        self.check_out_label = Label(self.r_frame, text='CHECK-OUT DATE',
font=('Consolas', 12), bd=0, bg='gray90')
        self.check_out_cal = DateEntry(self.r_frame, width=10,
day=int(creds_rtypes['hr_time'].split('-')[0]),
                                       month=int(creds_rtypes['hr_time'].split('-
')[1]),
                                       year=int(creds_rtypes['hr_time'].split('-')[-
1]))
        self.date_search_lbl = Label(self.r_frame, font=('Dialog
Input', 14, 'underline'), bd=0, bg='gray90',
                                     text="Search by Date", fg='Blue2',
cursor='hand2')
        self.description_box = ScrolledText(self.desc_frame, width=35, height=20,
wrap=WORD, bd=0, cursor='arrow',
                                            bg='gray92')
        self.description_box.insert(END, src_desc)

        self.r_menu = Menu(self, tearoff=0)
        self.r_menu.add_command(label='Cancel/Remove', command=self.cancel_remove)
        self.r_menu.add_command(label='Copy', command=lambda: self.copy_event())

        for i in self.entry_list:
            i.bind('<Return>', self.search)
        self.search_label.bind('<ButtonRelease-1>', self.search)
        self.search_label.bind('<Enter>', lambda _=None: self.search_label.config(for
eground='orange'))
```

```
        self.search_label.bind('<Leave>', lambda _=None: self.search_label.config(for
eground='blue2'))
        self.check_in_cal.bind('<Return>', self.date_search)
        self.check_out_cal.bind('<Return>', self.date_search)
        self.date_search_lbl.bind('<ButtonRelease-1>', self.date_search)
        self.date_search_lbl.bind('<Enter>', lambda _=None: self.date_search_lbl.conf
ig(foreground='orange'))
        self.date_search_lbl.bind('<Leave>', lambda _=None: self.date_search_lbl.conf
ig(foreground='blue2'))
        self.b_list.bind('<ButtonRelease-1>', self.select_item)
        self.b_list.bind('<Double-1>', self.copy_event)
        self.b_list.bind('<Button-3>', self.r_popup)
        self.b_list.bind('<Button-2>', self.cancel_remove)
        self.b_list.bind('<App>', self.r_popup)
        self.description_box.bind("<Key>", lambda o: "break")
        self.description_box.bind('<Button-2>', lambda _: 'break')
        self.description_box.tag_add("start", "1.0", "1.25")
        self.description_box.tag_config("start", font=('Source Code
Pro', 13, 'bold'))

        self.up_date()

    def pack_all(self, event=None):
        self.frame.pack(anchor=N)
        self.b_frame.pack(side=TOP)
        self.scroll_b1.pack(side=RIGHT, fill=BOTH)
        self.scroll_b2.pack(side=BOTTOM, fill=BOTH)
        self.b_list.pack()
        self.sep.pack(side=TOP, fill=BOTH, pady=10)
        self.l_frame.pack(side=LEFT, padx=10)
        self.sep1.pack(side=LEFT, fill=BOTH)
        self.l_mid_frame.pack(anchor=N, side=LEFT, padx=5)
        self.search_label.pack(anchor=S, side=LEFT, ipady=40)
        self.r_mid_frame.pack(anchor=N, side=LEFT, padx=5)
        self.sep2.pack(side=LEFT, fill=BOTH)
        self.r_frame.pack(anchor=N, side=LEFT, padx=10)
        self.desc_frame.pack(anchor=N)
        for i in self.radio_list:
            i.pack(anchor=W)
        for i, j in zip(self.label_list, self.entry_list):
            i.pack(anchor=W, padx=10, pady=10)
            j.pack(anchor=W, padx=10, pady=5)

        self.check_in_label.pack(anchor=N, padx=10, pady=10)
        self.check_in_cal.pack(anchor=N, padx=10, pady=10)
        self.check_out_label.pack(anchor=S, padx=10, pady=10)
        self.check_out_cal.pack(anchor=S, padx=10, pady=10)
        self.date_search_lbl.pack(anchor=S, side=BOTTOM, ipady=30)
        self.description_box.pack(anchor=CENTER, fill=BOTH)

    def unpack_all(self, event=None):
        self.frame.pack_forget()
        self.b_frame.pack_forget()
        self.scroll_b1.pack_forget()
        self.scroll_b2.pack_forget()
```

```
        self.b_list.pack_forget()
        self.sep.pack_forget()
        self.l_frame.pack_forget()
        self.sep1.pack_forget()
        self.l_mid_frame.pack_forget()
        self.search_label.pack_forget()
        self.r_mid_frame.pack_forget()
        self.sep2.pack_forget()
        self.r_frame.pack_forget()
        self.desc_frame.pack_forget()
        for i in self.radio_list:
            i.pack_forget()
        for i, j in zip(self.label_list, self.entry_list):
            i.pack_forget()
            j.pack_forget()

        self.check_in_label.pack_forget()
        self.check_in_cal.pack_forget()
        self.check_out_label.pack_forget()
        self.check_out_cal.pack_forget()
        self.date_search_lbl.pack_forget()
        self.description_box.pack_forget()

    def up_date(self, event=None):
        if self.controller.current_visible_frame == BrowsePage:
            self.pack_all()
        else:
            self.unpack_all()
        self.after(100, self.up_date)

    def select_item(self, event=None) -> str:
        """:returns a list of the selected row"""
        cur_item = self.b_list.selection()
        item_str = ''
        for i in cur_item:
            item_list = self.b_list.item(int(i))['values']
            for j in range(len(item_list)):
                item_list[j] = str(item_list[j])
            item_str += ', '.join(item_list) + '\n'
        return item_str

    def cancel_remove(self, event=None):
        """
        Function to remove booking from MySQL
        :param event: passed by functions and binders
        """
        try:
            cur_sel_vals = list(self.select_item().split(', '))
            cur_sel_vals[8] = cur_sel_vals[8].rstrip('\n')
        except IndexError:
            return
        ans = askyesno('M.Y. Hotel', f'Do you really want to
remove {cur_sel_vals[1]}\'s booking ?')
        if ans < 1:
            return
```

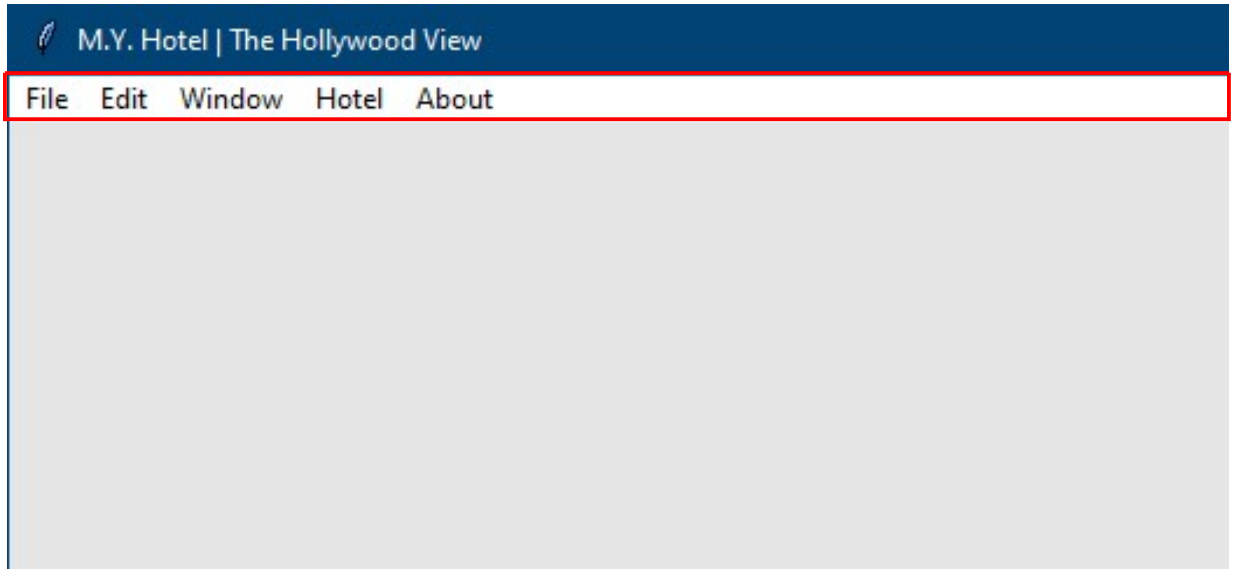
```
        cursor.execute(f'DELETE FROM HOTEL_INFO WHERE BOOK_ID="{cur_sel_vals[0]}" and
NAME="{cur_sel_vals[1]}" and'
                       f' ROOM_NO="{cur_sel_vals[2]}"'
                       f' and DATE_OF_CIN="{cur_sel_vals[3]}" and
DATE_OF_COUT="{cur_sel_vals[4]}" and ROOM_TYPE='
                       f'"{cur_sel_vals[5]}" and '
                       f'PH_NO="{cur_sel_vals[6]}" and PRICE="{cur_sel_vals[7]}" and
PAID_THRU="{cur_sel_vals[8]}"')
        connection.commit()

    def copy_event(self, event=None):
        """
        Copies the selected row to clipboard
        :param event: passed by functions and binders
        """
        self.clipboard_clear()
        c = self.select_item()
        self.clipboard_append(c)
        return

    def r_popup(self, event=None):
        """
        Menu for right click
        :param event: passed by functions and binders
        """
        try:
            self.r_menu.tk_popup(event.x_root, event.y_root, 0)
        finally:
            self.r_menu.grab_release()
        return

    def rad_btn_cmd(self, n: int):
        """Function to execute radio button commands"""
        if n == -1:
            self.b_list.delete(*self.b_list.get_children())
            return
        cursor.execute(f'SELECT * FROM hotel_info WHERE ROOM_TYPE="{room_t[n]}" OR
BOOK_ID="{str(self.v2[0].get())}" '
                       f'OR NAME="{str(self.v2[1].get())}" OR '
                       f'ROOM_NO="{str(self.v2[2].get())}" OR
PH_NO="{str(self.v2[3].get())}" OR '
                       f'DATE_OF_CIN="{str(self.check_in_cal.get_date())}" OR '
                       f'DATE_OF_COUT="{str(self.check_out_cal.get_date())}"')
        data = cursor.fetchall()
        self.b_list.delete(*self.b_list.get_children())
        for row in data:
            self.b_list.insert('', self.index, self.iid, values=row)
            self.index = self.iid = self.index + 1
        if self.length(data) > 16:
            for i in range(9):
                self.b_list.column(str(i), width=self.length(data) * 10, anchor=N)
        else:
            for i in range(9):
                self.b_list.column(str(i), width=160, anchor=N)
        return
```

```
    def search(self, event):
        """
        Function to search for SQL data using book ID, name, phone no, room no
        :param event: passed by functions and binders
        """
        cursor.execute(f'SELECT * FROM hotel_info WHERE BOOK_ID LIKE
"{str(self.v2[0].get())}" OR '
                       f'NAME="{str(self.v2[1].get())}" OR
ROOM_NO="{str(self.v2[2].get())}" OR '
                       f'PH_NO="{str(self.v2[3].get())}" OR
DATE_OF_CIN="{str(self.check_in_cal.get_date())}" OR '
                       f'DATE_OF_COUT="{str(self.check_out_cal.get_date())}"')
        self.b_list.delete(*self.b_list.get_children())
        data = []
        for i in range(len(col_names)):
            if len(self.v2[i].get()) == 0:
                pass
            else:
                unique_list = []
                cursor.execute(f'SELECT * FROM hotel_info WHERE {col_names[i]} LIKE
"%{str(self.v2[i].get())}%"')
                d = list(cursor.fetchall())
                for j in range(len(d)):
                    if len(d[j][0]) != 0 and d[j][0] in unique_list:
                        pass
                    else:
                        unique_list.append(d[j][0])
                        data.append(list(d[j]))
        for row in data:
            self.b_list.insert('', self.index, self.iid, values=row)
            self.index = self.iid = self.index + 1

        if self.length(data) > 16:
            for i in range(9):
                self.b_list.column(str(i), width=self.length(data) * 10, anchor=N)
        else:
            for i in range(9):
                self.b_list.column(str(i), width=160, anchor=N)

    def date_search(self, event):
        """
        Function to search for data using dates
        :param event: passed by functions and binders
        """
        cursor.execute(f'SELECT * FROM hotel_info WHERE
DATE_OF_CIN="{str(self.check_in_cal.get_date())}" OR '
                       f'DATE_OF_COUT="{str(self.check_out_cal.get_date())}"')
        data = cursor.fetchall()
        self.b_list.delete(*self.b_list.get_children())
        for row in data:
            self.b_list.insert('', self.index, self.iid, values=row)
            self.index = self.iid = self.index + 1
        if self.length(data) > 16:
            for i in range(9):
```

```
                self.b_list.column(str(i), width=self.length(data) * 10, anchor=N)
        else:
            for i in range(9):
                self.b_list.column(str(i), width=160, anchor=N)

    def length(self, dat) -> int:
        """
        Function to return the length of the longest word in SQL data
        :param dat: SQL data in form of tuples
        :returns: integer for maximum length
        """
        st = []
        for j in range(len(dat)):
            for k in range(8):
                st.append(len(str(dat[j][k])))
        return max(st) if len(st) is not 0 else 20
```

NOTE: Frames that contain large number of widgets have both `pack_all()` and `unpack_all()` functions in their classes (in this program, `BookRoomPage` and `BrowsePage`) to save memory space and for smooth functioning of the application.

MENUBAR :



Class defined for the menubar:

(NOTE: Every button on menubar has its own, separate function)

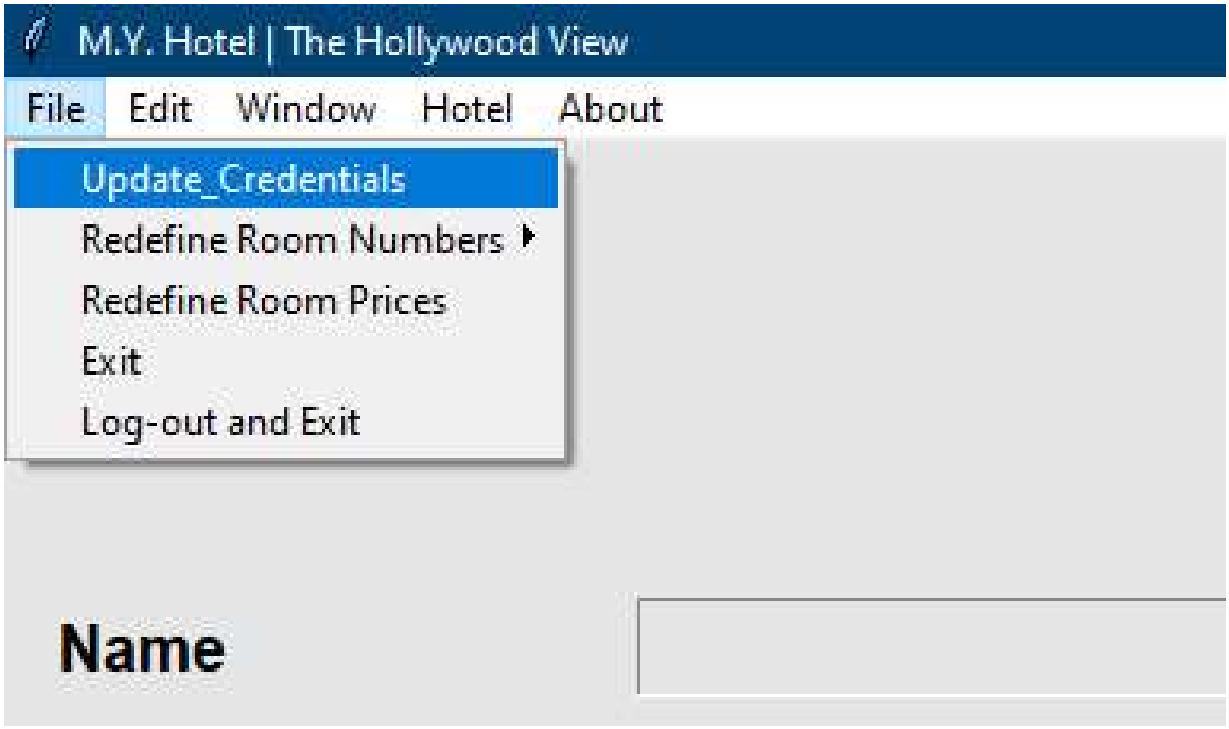
```
class MainMenu:
    __slots__ =
['master', 'file_menu', 'redefine_menu', 'edit_menu', 'window_menu', 'search_menu',
                 'sales_menu', 'about_menu']

    def __init__(self, master):
        """
        Menu class for 'M.Y. Hotel'
        :param master: parent window
        """
        self.master = master
        menu_bar = Menu(self.master)

        self.file_menu = Menu(menu_bar, tearoff=0)
        self.file_menu.add_command(label='Update_Credentials',
command=self.update_credentials)

        redefine_menu = Menu(menu_bar, tearoff=0)
        self.file_menu.add_cascade(label='Redefine Room Numbers',
menu=redefine_menu)
        redefine_menu.add_command(label='All', command=lambda: self.redef(0))
        redefine_menu.add_command(label='Single', command=lambda: self.redef(1))
        redefine_menu.add_command(label='Double', command=lambda: self.redef(2))
        redefine_menu.add_command(label='Triple', command=lambda: self.redef(3))
        redefine_menu.add_command(label='Quad', command=lambda: self.redef(4))
        redefine_menu.add_command(label='Studio', command=lambda: self.redef(5))
        redefine_menu.add_command(label='Master', command=lambda: self.redef(6))
        redefine_menu.add_command(label='Junior', command=lambda: self.redef(7))
```

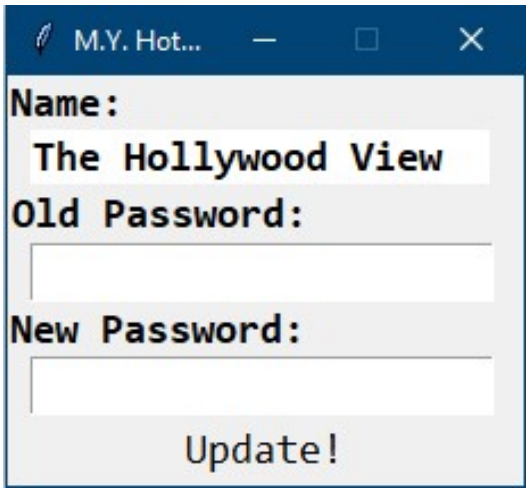
```
        redefine_menu.add_command(label='Connecting', command=lambda: self.redef(8))
        self.file_menu.add_command(label='Redefine Room Prices',
command=lambda: self.redef_prices())
        self.file_menu.add_command(label='Exit', command=self.exit)
        self.file_menu.add_command(label='Log-out and Exit', command=self.logoff)
        menu_bar.add_cascade(label='File', menu=self.file_menu)

        edit_menu = Menu(menu_bar, tearoff=0)
        edit_menu.add_command(label='Copy', accelerator="Ctrl+C",

command=lambda: menu_bar.focus_get().event_generate('<<Copy>>'))
        edit_menu.add_command(label='Cut', accelerator="Ctrl+X",

command=lambda: menu_bar.focus_get().event_generate('<<Cut>>'))
        edit_menu.add_command(label='Paste', accelerator="Ctrl+V",

command=lambda: menu_bar.focus_get().event_generate('<<Paste>>'))
        menu_bar.add_cascade(label='Edit', menu=edit_menu)

        window_menu = Menu(menu_bar, tearoff=0)
        window_menu.add_command(label='Full Screen', accelerator="F11",
command=lambda: self.full_screen())
        window_menu.add_command(label='Windowed', command=lambda: self.windowed())
        window_menu.add_command(label='Fixed Window',
command=lambda: self.fixed_window())
        menu_bar.add_cascade(label='Window', menu=window_menu)

        search_menu = Menu(menu_bar, tearoff=0)
        search_menu.add_command(label='Book a Room', accelerator="F4",
command=lambda: self.room_book())
        search_menu.add_command(label='Search/Cancellation', accelerator="F12",
command=lambda: self.past_cur_data())
        sales_menu = Menu(menu_bar, tearoff=0)
        search_menu.add_cascade(label='Check Sales', menu=sales_menu)
        sales_menu.add_command(label='Day Wise',
command=lambda: self.check_sales(DAY))
        sales_menu.add_command(label='Month Wise',
command=lambda: self.check_sales(MONTH))
        sales_menu.add_command(label='Year Wise',
command=lambda: self.check_sales(YEAR))
        menu_bar.add_cascade(label='Hotel', menu=search_menu)

        about_menu = Menu(menu_bar, tearoff=0)
        about_menu.add_command(label='About this software', accelerator='F1',
command=lambda: self.about())
        menu_bar.add_cascade(label='About', menu=about_menu)
        self.up_date()
        master.config(menu=menu_bar)
        # Bindings
        self.master.bind('<F4>', self.room_book)
        self.master.bind('<F12>', self.past_cur_data)
        self.master.bind('<F1>', self.about)
```

Now, the buttons that open menu are called cascades and in the above Menu class declaration there are 5 main cascades, namely – `'File'`, `'Edit'`, `'Window'`, `'Hotel'` and `'About'`, and 2 internal cascades - `'Redefine Room Numbers'` (cascade inside the `'File'` cascade), `'Check Sales'` (cascade inside the `'Hotel'` cascade).

(NOTE: ALL THE FUNCTIONS BELOW ARE PART OF `MainMenu` CLASS)

'File' cascade:



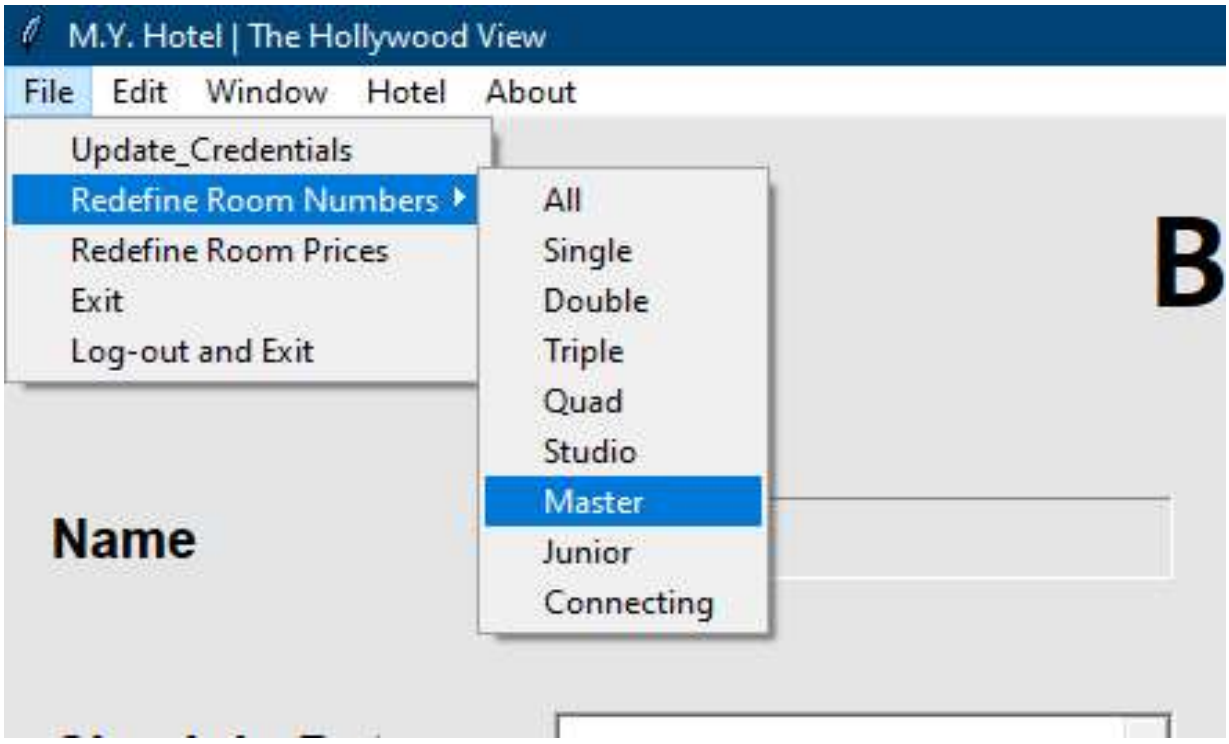


This function updates password and user name for login.

(NOTE: Name can be changed without filling both the password entry fields)

```
    def update_credentials(self):
        """Updates user credentials"""
        update_win = Toplevel(self.master)
        update_win.resizable(False, False)
        update_win.geometry(f'227x181')
        center_window(update_win)
        name_label = Label(update_win, text='Name:', bd=0,
font=('consolas', 14, 'bold'), justify=LEFT)
        old_pass_label = Label(update_win, text='Old Password:', bd=1,
font=('consolas', 14, 'bold'), justify=LEFT)
        new_pass_label = Label(update_win, text='New Password:', bd=0,
font=('Consolas', 14, 'bold'), justify=LEFT)
        name_entry = Entry(update_win, bd=0, font=('consolas', 14, 'bold'),
justify=LEFT)
        name_entry.insert(0, Hotel_Name)
        old_pass_entry = Entry(update_win, bd=1, font=('consolas', 14, 'bold'),
justify=LEFT, show='•')
        new_pass_entry = Entry(update_win, bd=1, font=('Consolas', 14, 'bold'),
justify=LEFT, show='•')
        update_button = Button(update_win, text='Update!', font=('Consolas', 14),
bd=0,
                               command=lambda: update(name_entry.get(),
old_pass_entry.get(), new_pass_entry.get()))

        name_label.pack(anchor=W)
        name_entry.pack(anchor=W, padx=10)
        old_pass_label.pack(anchor=W)
        old_pass_entry.pack(anchor=W, padx=10)
        new_pass_label.pack(anchor=W)
        new_pass_entry.pack(anchor=W, padx=10)
        update_button.pack(anchor=S)

        update_win.attributes('-topmost', 1)
        update_win.focus_force()

        def destroy(event):
            """
            Close window
            :param event: passed by functions and binders
            """
            update_win.destroy()

        update_win.bind('<Escape>', destroy)

        def update(name: str, old_pass: str, new_pass: str):
            """
            Updates name, checks old and new passwords
```

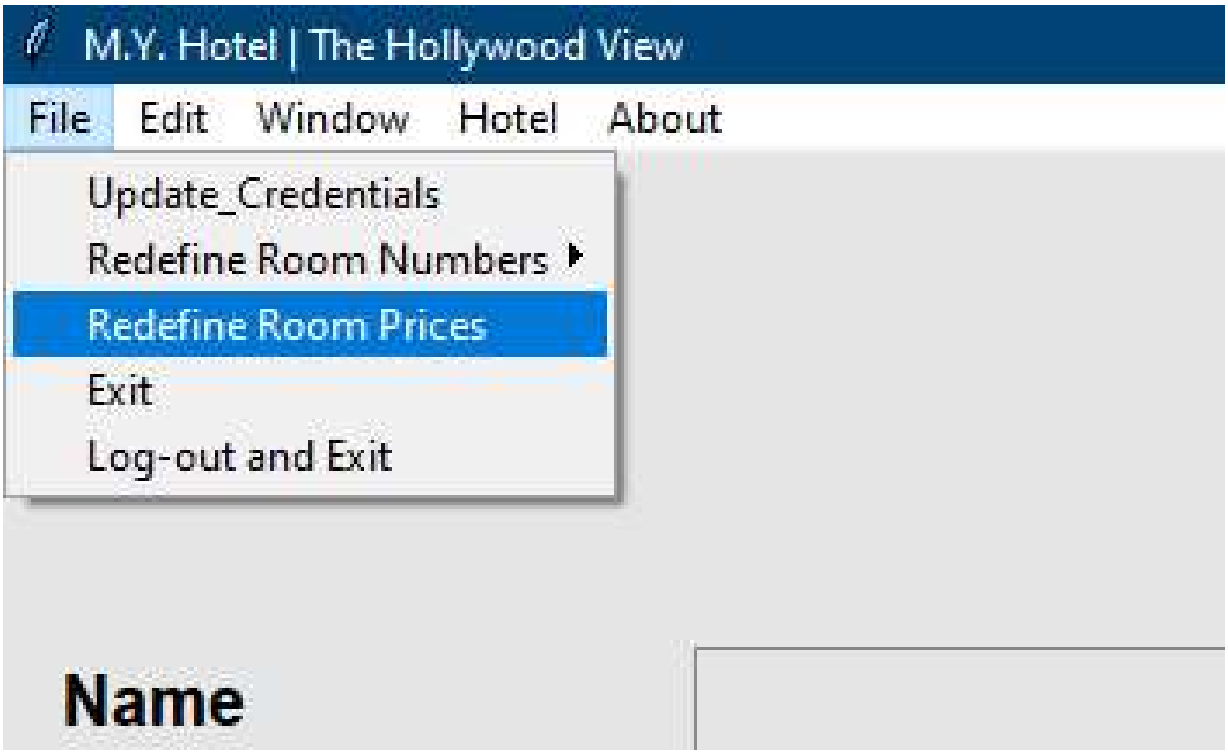
```
            :param name: name of hotel
            :param old_pass: old password
            :param new_pass: new password
            """
            creds_rtypes['name'] = name
            p = creds_rtypes.get('password').encode('utf-8')
            s = base64.b64decode(p)
            pd = str(s.decode('utf-8'))
            if len(old_pass) < 1 and len(new_pass) < 1:
                self.master.update_name(name, creds_rtypes['password'])
            elif len(old_pass) < 1 < len(creds_rtypes['password']) and len(new_pass)
> 0:
                showerror('M.Y. Hotel', 'Your old password field is empty')
                return
            else:
                if old_pass == pd:
                    enc_pd = base64.b64encode(new_pass.encode('utf-8'))
                    dec_pd = enc_pd.decode('utf-8')
                    self.master.update_name(name, dec_pd)
                    self.master.current_visible_frame = DataEntryPage
                    self.master.show_frame(LoginPage)
                else:
                    showerror('M.Y. Hotel', 'Your old password does not match!')
                    return
            update_win.destroy()
```

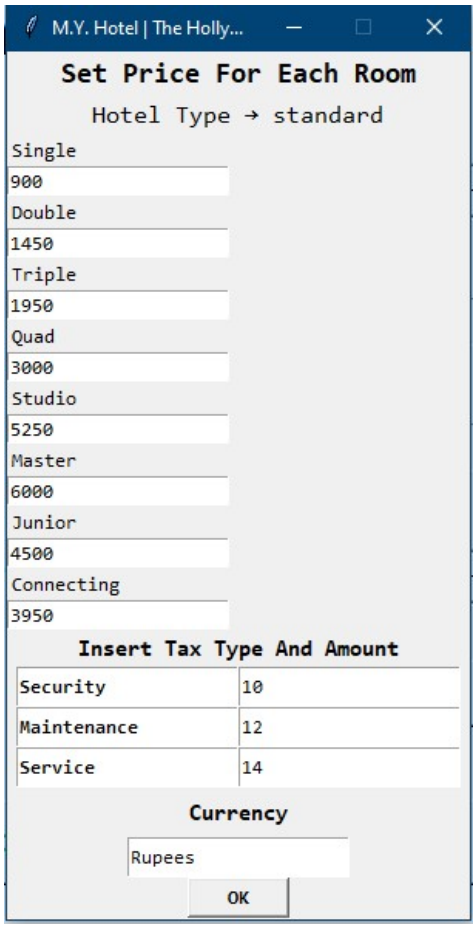


This function updates the number of rooms for every room type. If user clicks on `All` the 'Room Types ' entry page will be shown (mentioned above).

```
    def redef(self, n):
        """
        Redefine any particular room type or all of them
        :param n: integer for index of selected room type
        """
        if creds_rtypes['log'] == 'off':
            self.master.current_visible_frame = LoginPage
            self.master.show_frame(LoginPage)  # if logged off or user didn't log in
        else:
            if n == 0:
                self.master.current_visible_frame = RoomTypesPage
                self.master.show_frame(RoomTypesPage)
            else:
                def update(x):
                    """Updates the selected room type and exits Toplevel window"""
                    creds_rtypes['h_type'][creds_rtypes["sel_h_type"]][n - 1] = x
                    write_to_json()
                    re_win.destroy()

                re_win = Toplevel(self.master)
                re_win.geometry('187x81')
                center_window(re_win)
                re_win.resizable(False, False)
                re_label = Label(re_win,
text=f'Hotel Type {chr(129130)} ' + creds_rtypes["sel_h_type"], bd=0,
                                 justify=CENTER, font=('Times New Roman', 12))
                re_label.pack(anchor='nw')
                re_entry = Entry(re_win, bd=1, width=14, font=('Consolas', 18))
                re_entry.pack(anchor='nw')
                re_button = Button(re_win, text='Update', width=20, command=lambda:
update(re_entry.get()), bg='green',
                                   bd=0, font=('Times New Roman', 12))
                re_button.pack(anchor='nw', padx=1)
                re_win.focus_force()
```

```
            def destroy(event):
                """Closes window"""
                re_win.destroy()

            re_win.bind('<Escape>', destroy)
```





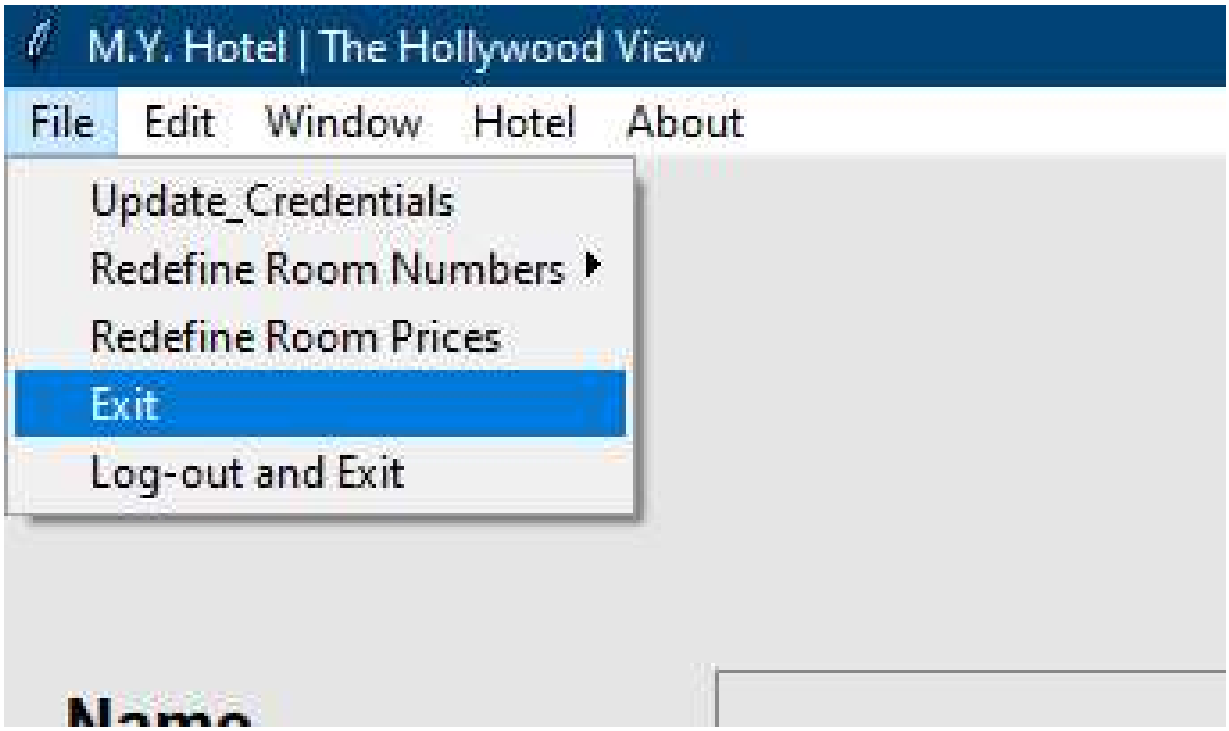
This function updates price of each room type as well as name of tax and their corresponding percentage.

(NOTE: Select only one type of currency throughout software usage, as it may disrupt the value of the price paid and taxes accordingly)

```
    def redef_prices(self):
        """Redefine rooms prices, taxes and currency"""
        price_win = Toplevel(self.master)
        price_win.geometry(f'300x560')
        center_window(price_win)
        price_win.resizable(False, False)
        set_lbl = Label(price_win, text='Set Price For Each Room', font='Consolas 14
bold')
        h_type_lbl = Label(price_win, text=f'Hotel
Type {chr(129130)} {creds_rtypes["sel_h_type"]}',
                           font=('Consolas', 12))

        str_var = list(StringVar() for i in range(len(room_t)))
        set_lbl.pack(anchor=N)
        h_type_lbl.pack(anchor=N)
```

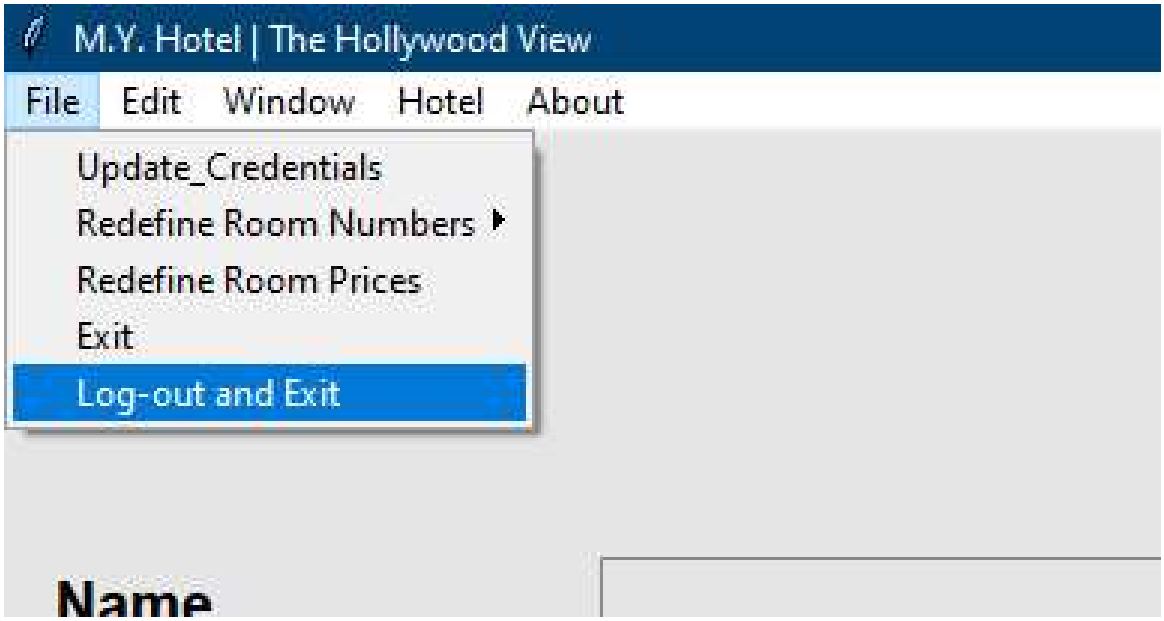
```
        for i in range(len(room_t)):
            Label(price_win, text=room_t[i], font=('Consolas', 10)).pack(anchor=W)
            Entry(price_win, textvariable=str_var[i],
font=('Consolas', 10)).pack(anchor=W)

        for i in range(len(room_t)):
            str_var[i].set(creds_rtypes['h_r_price'][creds_rtypes['sel_h_type']][i])

        tax_lbl = Label(price_win, text='Insert Tax Type And Amount',
font=('Consolas', 11, 'bold'))
        tax_lbl.pack()
        tax_frame = Frame(price_win)

        ttype1 = Entry(tax_frame, font=('Consolas', 9, 'bold'))
        ttype2 = Entry(tax_frame, font=('Consolas', 9, 'bold'))
        ttype3 = Entry(tax_frame, font=('Consolas', 9, 'bold'))
        _ttype1 = Entry(tax_frame, font=('Consolas', 9))
        _ttype2 = Entry(tax_frame, font=('Consolas', 9))
        _ttype3 = Entry(tax_frame, font=('Consolas', 9))

        ttype1.insert(0, list(creds_rtypes['tax'].keys())[0])
        ttype2.insert(0, list(creds_rtypes['tax'].keys())[1])
        ttype3.insert(0, list(creds_rtypes['tax'].keys())[2])
        _ttype1.insert(0, list(creds_rtypes['tax'].values())[0])
        _ttype2.insert(0, list(creds_rtypes['tax'].values())[1])
        _ttype3.insert(0, list(creds_rtypes['tax'].values())[2])

        ttype1.grid(row=0, column=0, ipady=4)
        _ttype1.grid(row=0, column=1, ipady=4)
        ttype2.grid(row=1, column=0, ipady=4)
        _ttype2.grid(row=1, column=1, ipady=4)
        ttype3.grid(row=2, column=0, ipady=4)
        _ttype3.grid(row=2, column=1, ipady=4)

        currency_lbl = Label(price_win, text='Currency',
font=('Consolas', 11, 'bold'))
        currency_entry = Entry(price_win, font=('Consolas', 9))
        currency_entry.insert(0, creds_rtypes['currency'])

        tax_frame.pack()
        currency_lbl.pack(pady=4)
        currency_entry.pack(ipady=4)

        def destroy(event):
            """Close the window"""
            price_win.destroy()

        def ok(event=None):
            """Function to update the redefined prices, etc"""
            p_list = []
            for i in range(len(room_t)):
                p_list.append(str_var[i].get())

            creds_rtypes['h_r_price'][creds_rtypes['sel_h_type']] = p_list
            creds_rtypes['tax'] = {}
```

```
            creds_rtypes['tax'][ttype1.get()] = _ttype1.get()  # update the tax type
and its value
            creds_rtypes['tax'][ttype2.get()] = _ttype2.get()  # Only three types of
taxes allowed
            creds_rtypes['tax'][ttype3.get()] = _ttype3.get()  # for references see
docs
            creds_rtypes['currency'] = currency_entry.get()    # updates currency
input

            write_to_json()
            price_win.destroy()

        ok_btn = Button(price_win, text='OK', font=('Consolas', 10, 'bold'), width=8,
command=ok)
        ok_btn.pack(anchor=S)
        price_win.bind('<Escape>', destroy)
        price_win.bind('<Return>', ok)
        price_win.focus_force()
        price_win.attributes('-topmost', 1)
```



Function to exit M.Y. Hotel.

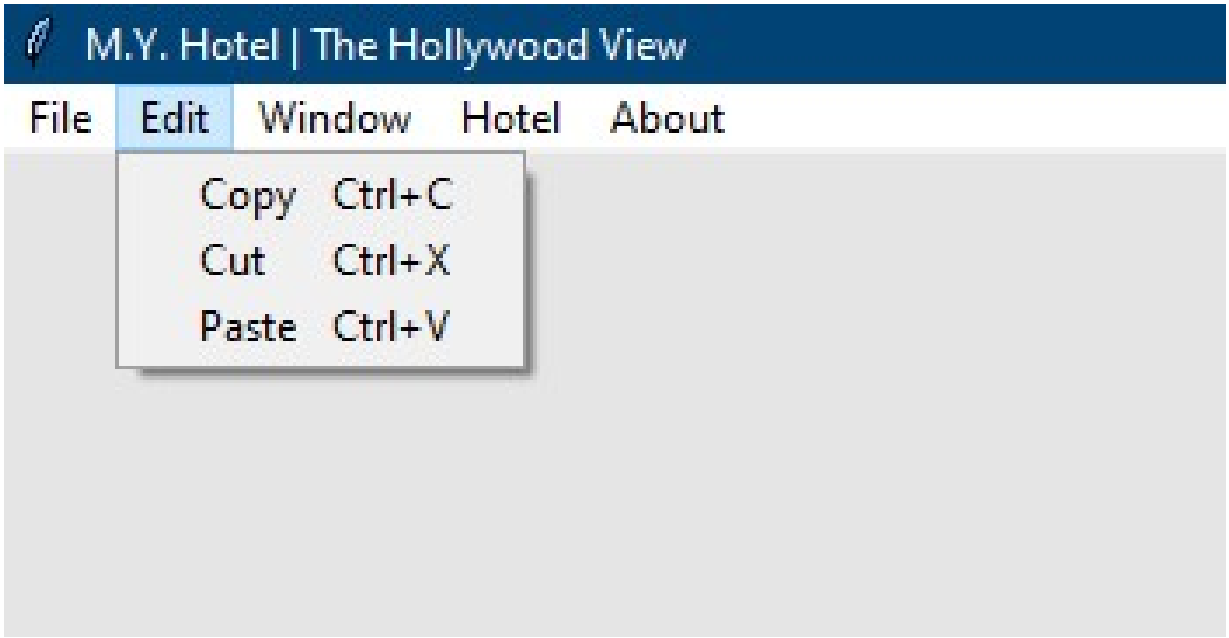
```
    def exit(self):
        """asks if user wants to exit"""
        ans = askyesno('M.Y. Hotel', 'Do you really want to exit?')
        if ans > 0:
            self.master.destroy()
        else:
            pass
```

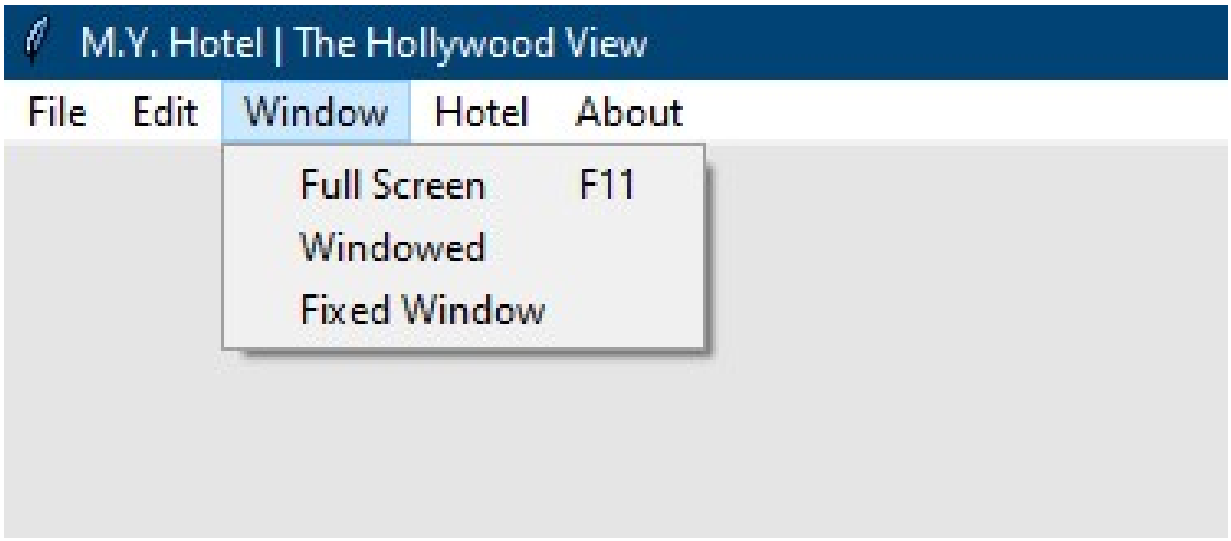


Function to log-out and exit M.Y. Hotel.

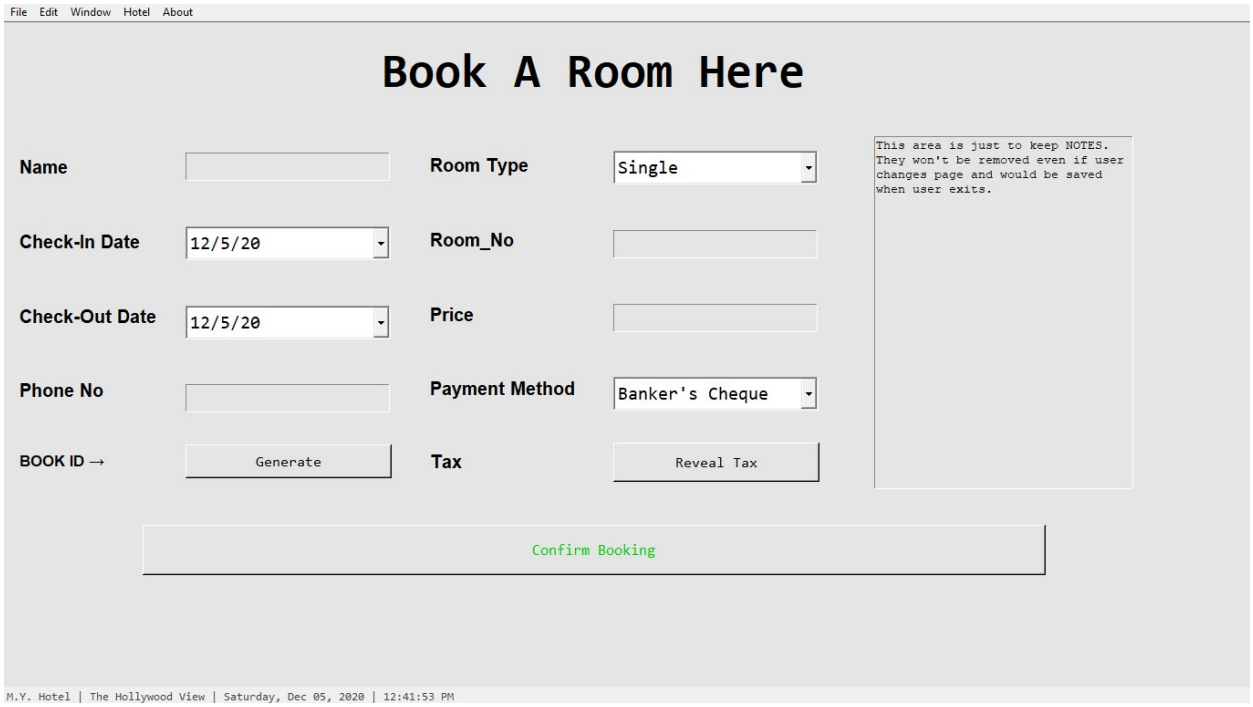
```
    def logoff(self):
        creds_rtypes['log'] = 'off'
        write_to_json()
        self.exit()
```

' Edit' cascade:

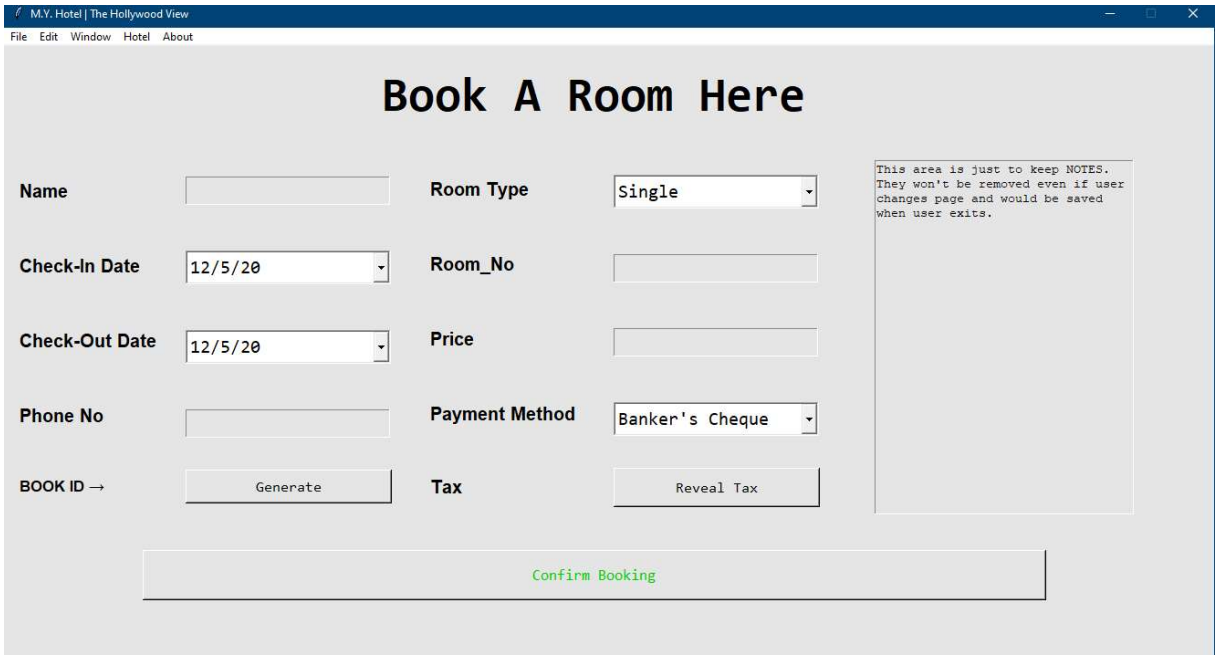




Window mode can be changed to 'fullscreen', 'fixed width' or just a 'regular window'.



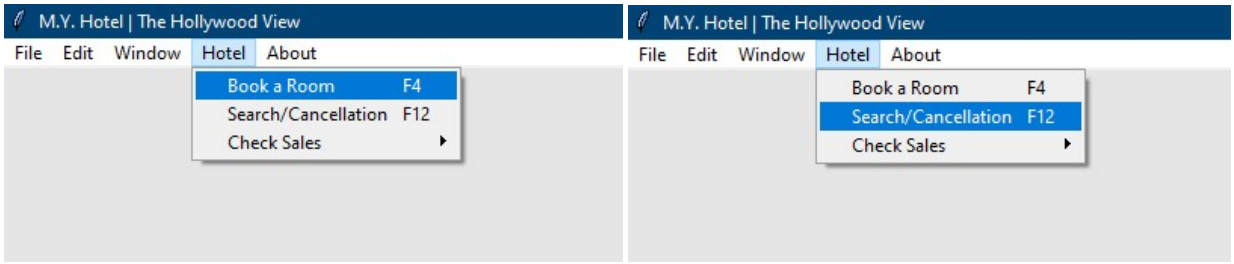
Fullscreen mode shows current time, date, day, name of hotel and name of program at the bottom.
(NOTE: No other window will be accessible in full screen mode)



In fixed width mode window can only be moved resizing is not allowed.

```
    def full_screen(self):
        """Toggle full screen window"""
        global win_event
        win_event = NORMAL
        self.master.full_screen_toggle()

    def windowed(self):
        """Makes resizable window"""
        global win_event
        win_event = ZOOMED
        self.master.full_screen_toggle()
        self.master.resizable(True, True)

    def fixed_window(self):
        """Makes window with resizing disabled """
        global win_event
        win_event = ZOOMED
        self.master.full_screen_toggle()
        win_event = FIXED
        self.master.full_screen_toggle()
```

'Hotel' cascade's:

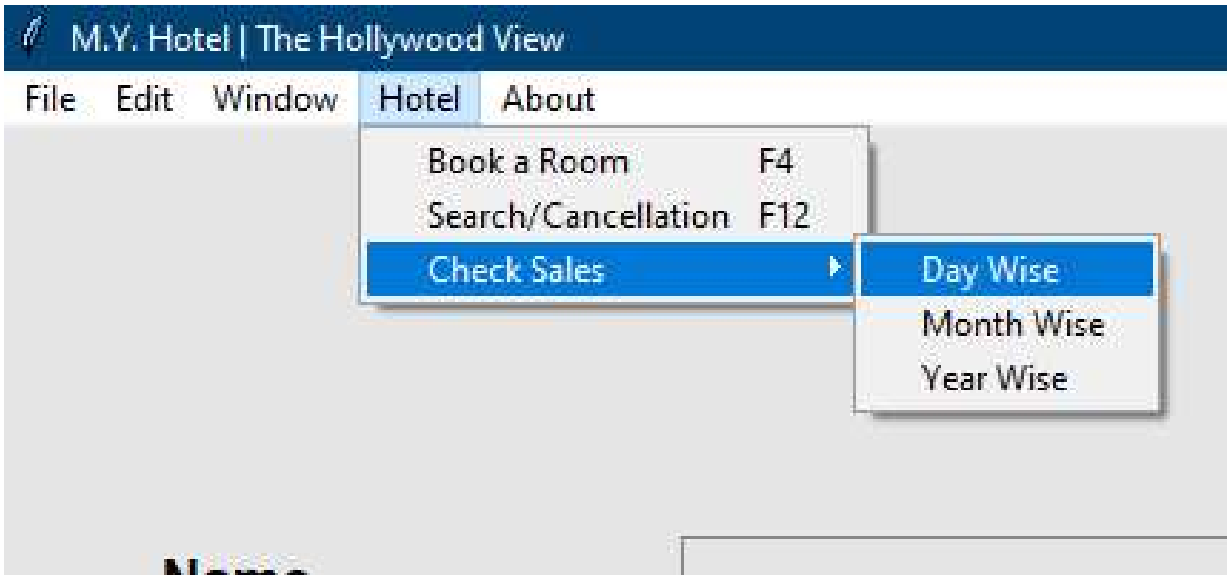


'Book a Room' and 'Search/Cancellation' buttons open the room booking and search page respectively.

```
    def room_book(self, event=None):
        """
        Book room
        :param event: passed by binders and functions
        """
        if creds_rtypes['log'] == 'off':
            self.master.current_visible_frame = LoginPage
            self.master.show_frame(LoginPage)
        else:
            self.master.current_visible_frame = BookRoomPage
            self.master.show_frame(BookRoomPage)

    def past_cur_data(self, event=None):
        """
        Search for past and present booking data
        :param event: passed by binders and functions
        """
        if creds_rtypes['log'] == 'off':
            self.master.current_visible_frame = LoginPage
            self.master.show_frame(LoginPage)
        else:
            self.master.current_visible_frame = BrowsePage
            self.master.show_frame(BrowsePage)
```

'Check Sales' cascade

This cascade's commands show graph of rooms booked day, month, and year wise.


M.Y. Hotel | The Hollywood View
File
Edit
Window
Hotel
About
Book a Room F4
Search/Cancellation F12
Check Sales
Day Wise
Month Wise
Year Wise

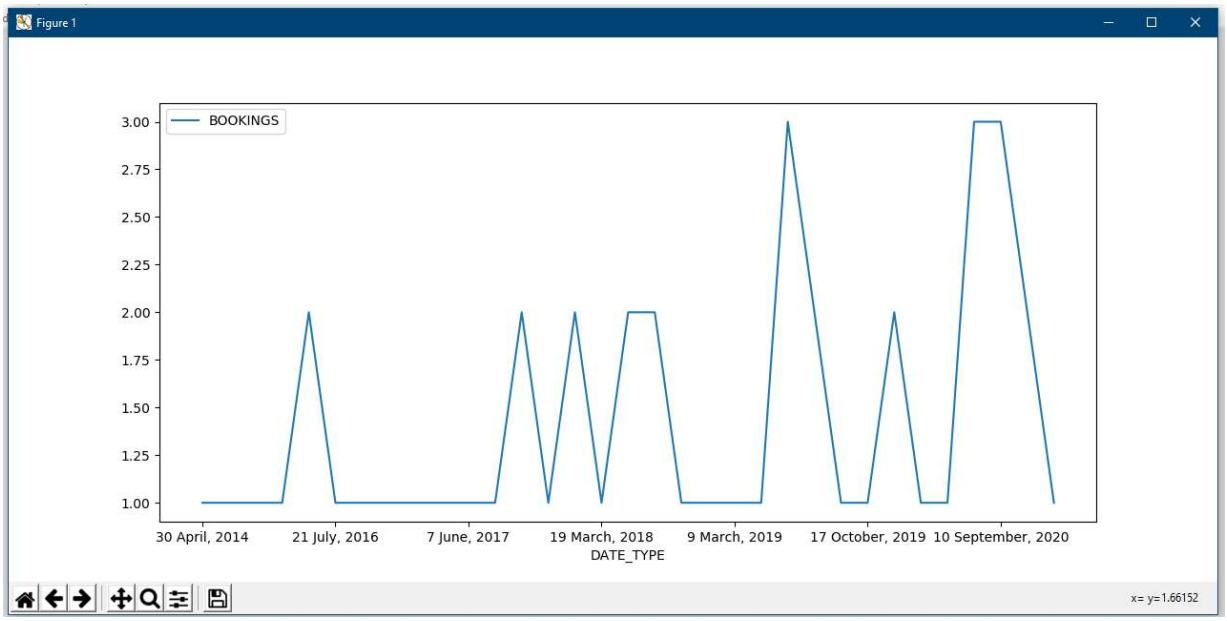
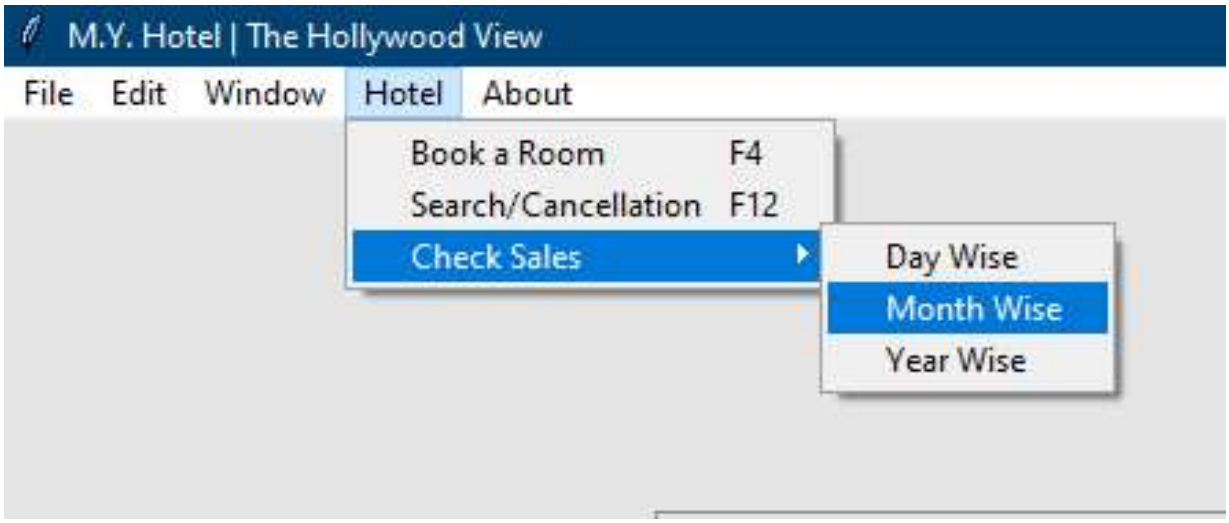


Figure 1
BOOKINGS
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
30 April, 2014
21 July, 2016
7 June, 2017
19 March, 2018
9 March, 2019
17 October, 2019
10 September, 2020
DATE_TYPE
x= y=1.66152


M.Y. Hotel | The Hollywood View
File
Edit
Window
Hotel
About
Book a Room F4
Search/Cancellation F12
Check Sales
Day Wise
Month Wise
Year Wise

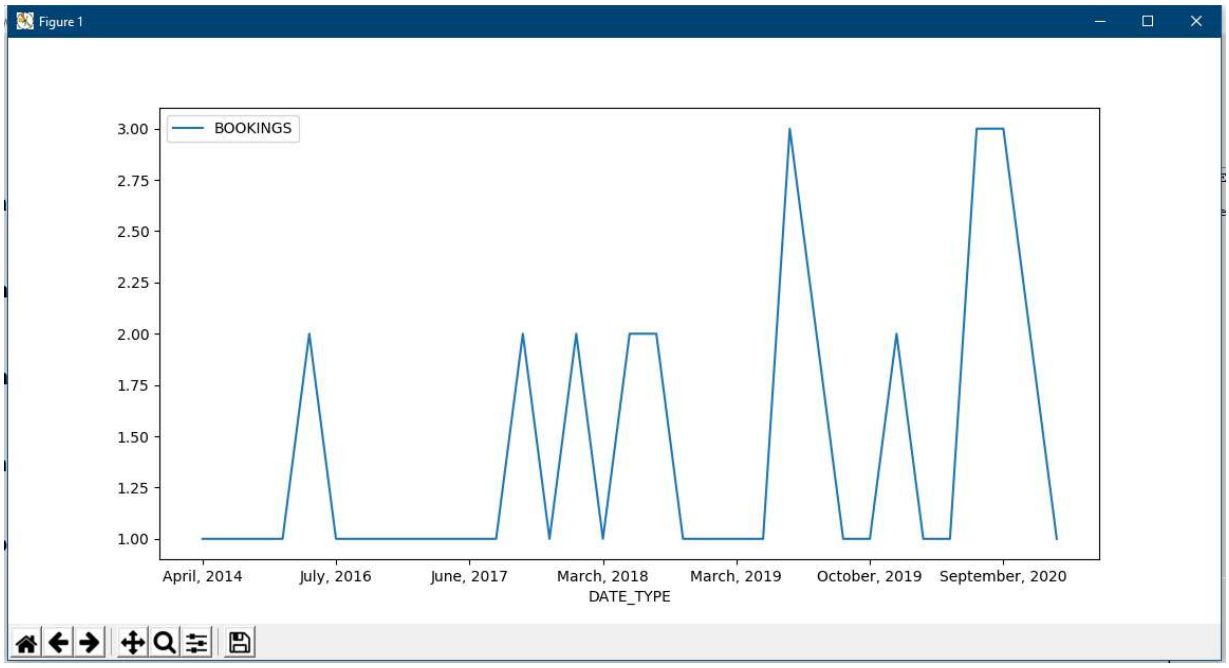
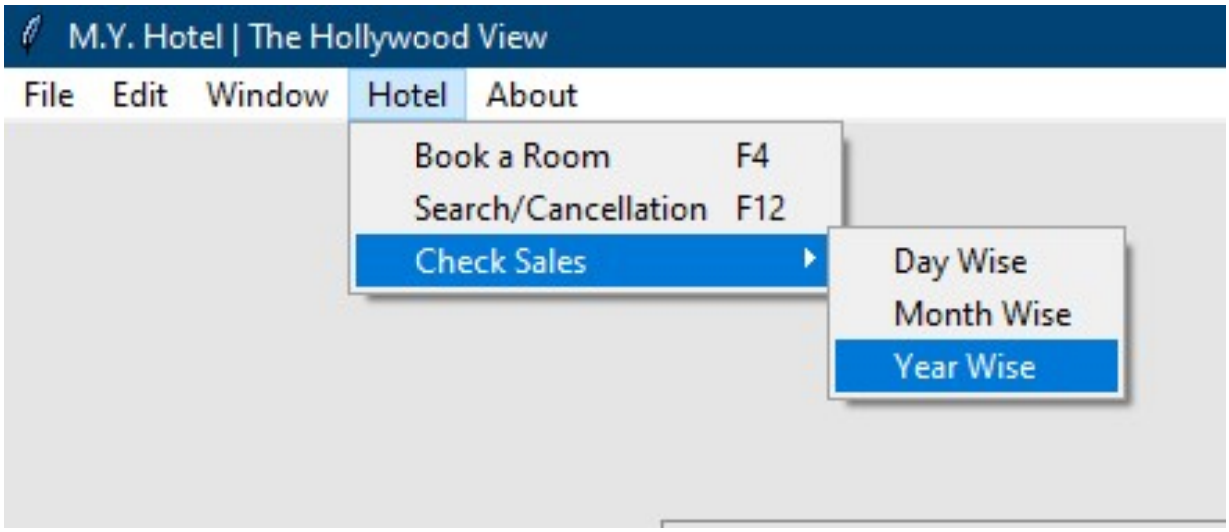
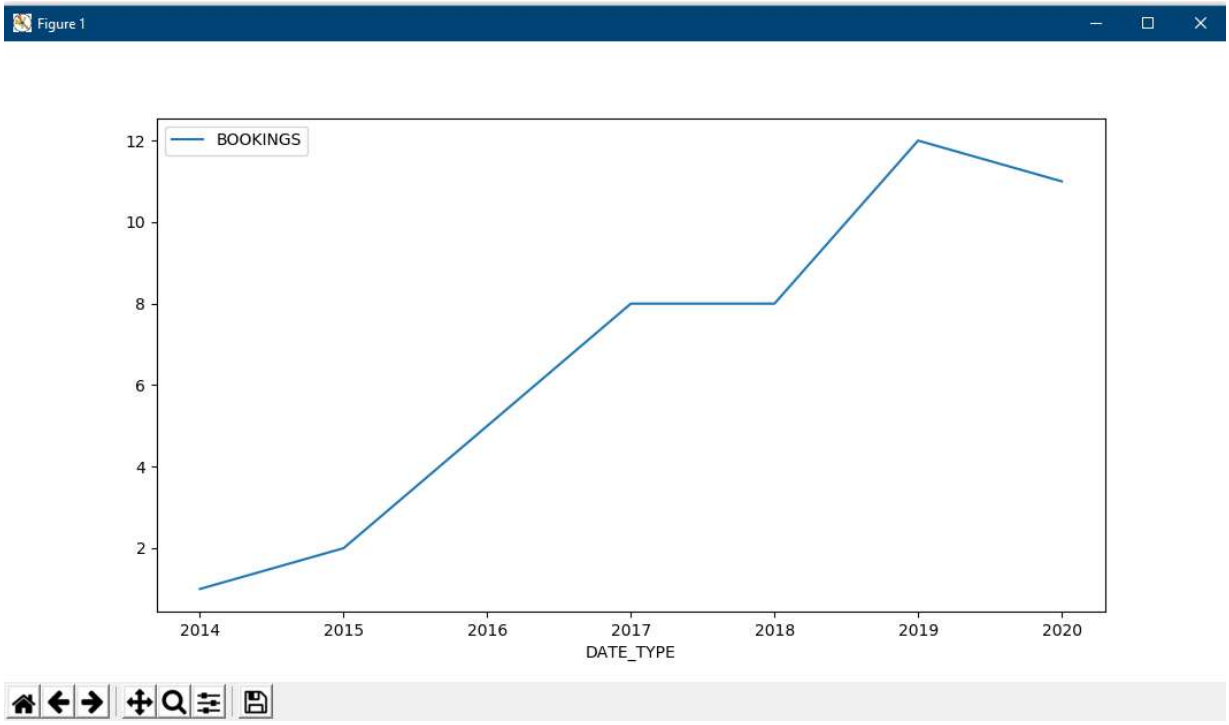


Figure 1
BOOKINGS
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
April, 2014
July, 2016
June, 2017
March, 2018
March, 2019
October, 2019
September, 2020
DATE_TYPE





(NOTE: Above graphs are based on 47 test entries, and since this isn't enough data the 'day wise' and 'month wise' graphs look similar)

(NOTE: Due to size limitations the graph can be viewed in intervals. To see more detailed data go to '%_data' in 'data' folder where % can be (d, m, y) according to {d=date, m=month, y=year})

```
    def check_sales(self, dtype: str = None):
        """
        Check sales in past
```

```
        :param dtype: date type based on day, month, or year
        """
        if creds_rtypes['log'] == 'off':
            self.master.current_visible_frame = LoginPage
            self.master.show_frame(LoginPage)
        csv_sql(dtype=dtype).plot(x='DATE_TYPE', y='BOOKINGS')
        plt.show()
```

'About' cascade

'About this software' command opens this documentation.

```
    def about(self, event='none'):
        """
        Opens documentation of this project
        :param event: passed by functions and binders
        """
        Popen([os.path.abspath('./source/documentation.pdf')], shell=True)
```

Other than the visible functions 'MainMenu' class has an `'up_date'` function which checks every 5 seconds whether the user is signed-in or not. If the user is signed in the `'Update_Credentials'` and `'Redefine Room Numbers'` functions will be accessible, otherwise they will be disabled.

```
    def up_date(self):
        """Function to keep the update and redefine menubar disabled until signup"""
        if creds_rtypes['name'] == '' and creds_rtypes['log']
== 'off' and creds_rtypes['room_def'] is False:
            self.file_menu.entryconfigure('Update_Credentials', state=DISABLED)
            self.file_menu.entryconfigure('Redefine Room Numbers', state=DISABLED)
            self.master.after(5000, self.up_date)
        else:
            self.file_menu.entryconfigure('Update_Credentials', state=NORMAL)
            self.file_menu.entryconfigure('Redefine Room Numbers', state=NORMAL)
            return
```

After all the above declaration of the main program, below given function shows the loading screen at start-up of the application and configures MySQL database.

```
def pre_load():
    """Shows the loading screen with Hotel name and loading bar"""
    load_win = Tk()
    load_win.overrideredirect(1)
    load_win.geometry(f'550x350')
    center_window(load_win)

    # path to background image file
    img_dir = path.join(path.dirname(__file__), 'images')
    image = Image.open(path.join(img_dir, 'hotel.gif'))
    # loading background image file
```

```
    bg_img = ImageTk.PhotoImage(image)
    # putting image into the window frame
    img_frm = Frame(load_win)
    img_lbl = Label(img_frm, image=bg_img)

    img_frm.pack(fill=BOTH, expand=YES)
    img_frm.pack_propagate(False)
    img_lbl.pack()
    # frame for hotel name and loading bar
    center_frm = Frame(img_frm)
    center_frm.place(relx=0.5, rely=0.5, anchor=CENTER)
    hotel_lbl = Label(center_frm, height=4, width=20, text='', font=('Lucida Sans
Typewriter', 20, 'bold'),
                      fg='white', bg='grey', justify='left')  # hotel label and
loading bar
    if len(creds_rtypes['name']) < 1:  # if name of hotel is not found initially name
of software will be shown
        hotel_lbl.configure(text=' Manage Your\n Hotel               \n')
    else:  # if name found hotel name will be displayed
        rem_num = ''
        cut_name = ''
        if len(creds_rtypes['name']) > 18:  # due to limited space if name of hotel
is greater than 18 characters
            for i in range(15):
                cut_name += creds_rtypes['name'][i]
            num = str(len(creds_rtypes['name']) - 13)  # name will be
displayed upto 13 characters
            for i in num:  #
                _i = int(i)  #
                rem_num += sub_nums[_i - 1]  # with number of not-shown letters
displayed at the end
            hotel_lbl.configure(text=' ' + cut_name + '...' + rem_num + ' \n')  # and
'...' just before the numbers
        else:
            spaces = 20 - len(creds_rtypes['name'])  # length of label has to remain
fixed so spaces are added
            hotel_lbl.configure(
                text=' ' + creds_rtypes['name'] + (' ' * spaces) + '\n')  # if length
of hotel name is less
    #
    than 18 characters

    load_win.wait_visibility()
    hotel_lbl.pack(fill=BOTH, expand=YES, ipadx=1)

    def loading_bar(s):
        hotel_lbl.configure(text=str(hotel_lbl.cget('text')) + s)

    increment = random.randint(2, 12)

    for i in range(increment):
        load_win.after(1100, lambda a=None: loading_bar('▓'))
        time.sleep(0.08)

    def selfdestruct():
```

```
        # creates database if not exists
        create_sql_db()
        for j in range(12 - increment):
            load_win.after(1100, lambda a=None: loading_bar('▉'))

        load_win.destroy()

    load_win.after(3000, selfdestruct)
    load_win.mainloop()
```

Finally the program starts executing from here. In this part of the code all the directories and their corresponding files, if do not exist – are checked and created, while the screen loads. Then the MySQL database connection is established and main window (declared as the `Logic` class above which is inherited from Tk class of tkinter) opens.

```
# START EXECUTING APPLICATION
check_and_create_dir('data')
if not check_file('./data/creds_rtypes.json'):
    create_json()
check_and_create_file('./data/d_data.csv')
check_and_create_file('./data/m_data.csv')
check_and_create_file('./data/y_data.csv')

check_and_create_dir('images')
if not check_file('./images/hotel.gif'):
    create_gif()

if not check_file('./images/hotel.ico'):
    create_ico()

with open('./data/creds_rtypes.json') as file:
    try:
        creds_rtypes = json.load(file)
        "Dictionary for all credentials data"

        Hotel_Name = creds_rtypes['name']
        "Name of hotel"
    except json.JSONDecodeError or FileNotFoundError:
        create_json()

# shows the loading window
pre_load()

connection = pymysql.Connect(host='localhost', user='root', passwd='',
database='hms__my_hotel')
"MySQL connection object for database"
cursor = connection.cursor()
"MySQL cursor object for the connection"

# Main window
app = Logic()
```

```
# creates database if not exists
app.title(f'M.Y. Hotel | {Hotel_Name}')
app.focus_force()
# geometry to match (somewhat) centered position
app.geometry(f'{int(app.winfo_screenwidth() - 20)}x{int(app.winfo_screenheight()) -
 80}+2+8')
app.mainloop()
# save the data and close database and JSON connections
write_to_json()
connection.close()
```

When the window is closed by the user, all the settings and data are stored and the connection to the database is closed.

DATA DICTIONARY

Database Name : "hms__my_hotel"
Table Name : "hotel_info"

"hotel_info" description:

FIELD	TYPE	SIZE	NULL	KEY	DEFAULT
BOOK_ID	VARCHAR	100	NO	PRIMARY	NULL
NAME	VARCHAR	100	YES		NULL
ROOM_NO	INT	4	YES		NULL
DATE_OF_CIN	DATE		YES		NULL
DATE_OF_COUT	DATE		YES		NULL
ROOM_TYPE	VARCHAR	20	YES		NULL
PH_NO	VARCHAR	20	YES		NULL
PRICE	DECIMAL	(10, 2)	YES		NULL
PAID_THRU	VARCHAR	50	YES		NULL

CSV Files for checking sales and plotting graphs have also been used:

Files: "d_data.csv": Contains number of bookings done on every single date

"m_data.csv": Contains number of bookings done every month

"y_data.csv": Contains number of bookings for every year