NEA

GreenBird Properties Database management system

Contents:

Analysis

Project Background:

Last summer I was doing work experience for my father's firm called greenbird which is a property renting firm that rents out flats to tenants. The work I did involved booking and checking out customers from different apartments, during which I had to record data into a record book as well as retrieve data. This data included flats occupied, flats not occupied, prices, length of stay etc.

In the process of recording the data, I found it especially difficult to read the data from the record book as it contained seemingly endless data with many rows repeated thus making it hard it identify the most recent value for a field for example I had trouble updating the flats occupied field as it was hard to identify its most recent value.

Seeing as there weren't a great number of flats under the company's portfolio it was understandable that there wasn't a more efficient method to how data was stored. This, however, was an unsustainable way of storing and using data which makes the firm prone to mistakes when booking customers.

Project Outline:

Seeing as the company was using a paper-based method of storing data, which meant that during business activity there are increased chances of mistakes happening, such as misreading when a booking ends or which apartments are available. For my project I am creating a GUI database management software that will make it easy to store, read and retrieve data onto and from a database.

Research:

Researching the Problem:

I organised a meeting with the manager to find out:

- How the current system works
- Problems with the current system
- How the new system should be defined
- Key features of the system

Interview with the manager:

Questions about data given and received:

Question 1: "what requests and criteria does the customer ask for when renting an apartment"

Answer 1: "Our apartments have adverts across different websites and so the customer already has an idea of what s in the apartment such as bathrooms and bedrooms. This leaves the customer's request to be about what flat they would like to book, how long they are planning to stay (monthly basis) and how many people are occupying the property."

Question 2: "what information do you ask for from the customer"

Answer 2: "we ask the customer for their basic information including name, email, phone number, birth date"

Question 3: "What information about the booking is given to the customer"

Answer 3: "we send the customer a message telling them when to check in and out

Questions about the current system:

Question 1: "how is data from the customer stored"

Answer 1: "I the manager retrieve all information from the customer usually via phone call or email, I then store the information about the customers booking in a record book

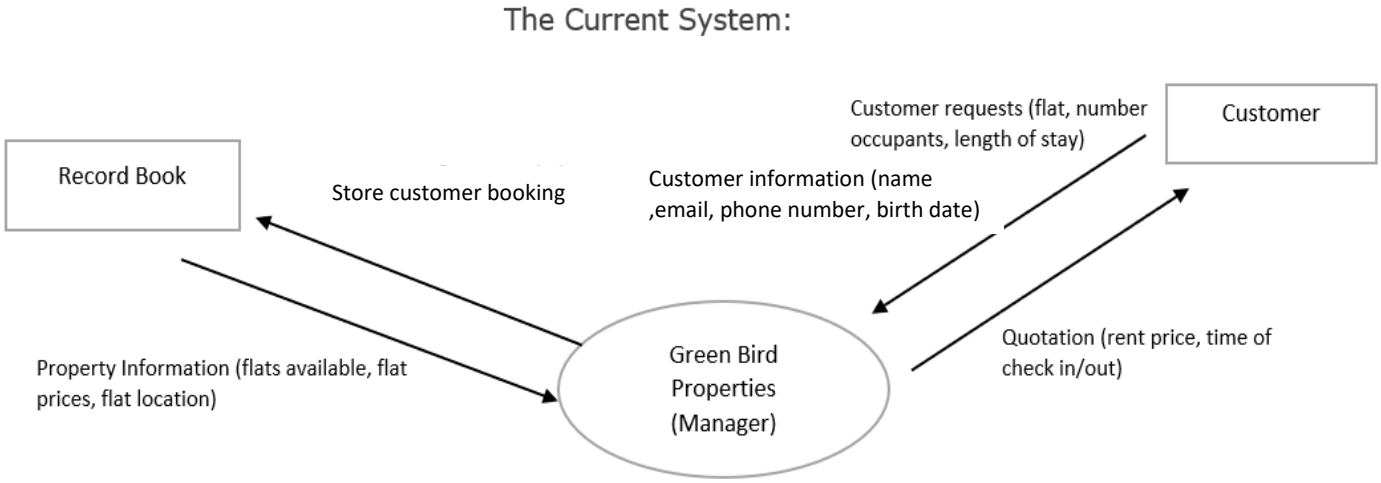
Question 2: “how is data about the apartment stored”

Question 2: “data about the apartment is also stored in a record book. This allows for checking that the apartment requested by the customer correlates with the one in our record books”

Question 3: “how is data about the customer booking retrieved and stored”

Answer 3: “data about the customer booking is also stored in the same record book. Some data needs to be calculated using other data. For example, calculating the end date would require calculation based on the start of the lease and the how many months the customer plans to stay. Once this data is calculated and stored, I give the information back to the customer by sending a message or email.”

This allowed me to create a data flow diagram to fully explain the current system:



Problems with the current system:

In the current system shown above I was able to identify the following problems:

Problem 1: Different rows will have same values for many fields

Why this is a problem: Because a new row is added each time data is recorded or updated, there will be many rows that have most fields containing the same values with only one difference. For example, if the manager decides to change the rent price for an apartment, a new row will be created identical to the row with the to-be changed price. Visually this can be confusing when retrieving that data and so the most updated record may not be chosen.

Problem 2: Hard to calculate some data that is based on other data

Why this is a problem: This is a problem because it could lead to the table having incorrect records. For example, an end date would have to be calculated based on the duration of stay in months(s). If this data is calculated wrong then there could confusion when checking out customers which could affect business reputation.

Problem 3: There is no measure to ensure incorrect data is not recorded.

Why this is a problem: As the current system is paper based, all data is entered manually. This leads it susceptible to human error. For example, when booking a customer an apartment already occupied may be recorded. If incorrect data is entered into the database this could ruin the integrity of the database as incorrect would then also be retrieved just like the problem of dynamic data.

Problem 4: Large amount of storage is required

Why this is a problem: This is because in a paper-based system data rows of data are never deleted and so as more bookings are recorded and updated more storage is needed to occupy them which is quite costly.

Problem 5: Easily Damaged

Why this is a problem: A paper-based system can be easily damaged by things such as water, without having a back up to rely on. It is also at risk of being misplaced

Why haven't these problems been considered by manager:

As mentioned in the project background, the business currently has five apartments under their portfolio which is seemingly manageable with a paper-based system. However, this doesn't mean that the problems with using this type of system won't affect the business and as the business grows it becomes more likely that these problems will become prevalent.

Computer Based System Vs Paper Based

Here is a general overview of why a new computerised system is advantageous

Computerised	**Paper-Based**
Can hold a vast amount of data	Limited by physical storage space available
Very fast to find a specific record	Can take a while to manually search through all of the records
Can easily search for a specific criterion e.g. "all of the people who live in Warwick"	Difficult to search for a specific criterion; every record would have to be manually looked at.
Can be used to analyse the data e.g. 'most popular selling item'	Very difficult to analyse the data
Data can be sorted into ascending or descending order on multiple criteria	Difficult to sort data on more than one criterion.
Can easily update or amend a record e.g. customer's address after moving to a new house	Changes have to be done manually. Records can look messy if scribbled out.
Records are stored safely; they are available when needed	Records can be lost or misfiled making it hard to find them
The database can be kept secure by use of passwords	The only security would be locking up the records.
Easy to make a back-up in case of data loss	Difficult to make a backup because every page/card would have to be re-written or photocopied. This means extra storage space is needed.

Researching the Solution:

What to look for in an efficient system?

- Tables in the database are in third normal form:

  A table in third normal form eliminates repeating values and splits the database into smaller tables which also reduces the storage size

- Dynamic data is updated automatically:

  Correlating to the problem with a paper-based system an efficient database will have the ability to automatically update data that is affected by uncontrollable factors.

- Data integrity is maintained when inserting, deleting or updating the table:

This means that when making changes to a table in a database, accuracy and consistency of all other data is maintained.

- Easy to query data to retrieve information:

  It must be easy to search the database and retrieve information based on given criteria

- Database can be retrieved if corrupted or lost:

  The database must be able to be stored in a back-up so in the event it gets corrupted it can easily be gotten back

Examples:

Fig 1

Rooms | Switchboard Items | Users

UserID	Last Name	First Name	User Name	User Type	Password	Click to Add
1	Person	Andrew	Andrew	Admin	*****	
2	Nunez	Fermin	Fermin	Normal User	*****	
3	Nunez	Alberto	Alberto	Normal User	*****	
4	Pech	Victor	Victor	Normal User	*****	
5	Huerta	Francisco	Francisco	Admin	*****	
6	Castor	Manuel	Manuel	Normal User	*****	
7	Nuñez	Jose Luis	Luis	Normal User	*****	
8	Generales	Pedidos	Pedidos	Normal User	*****	
(New)						

This is an example of a normalised computer-based database using Microsoft access.

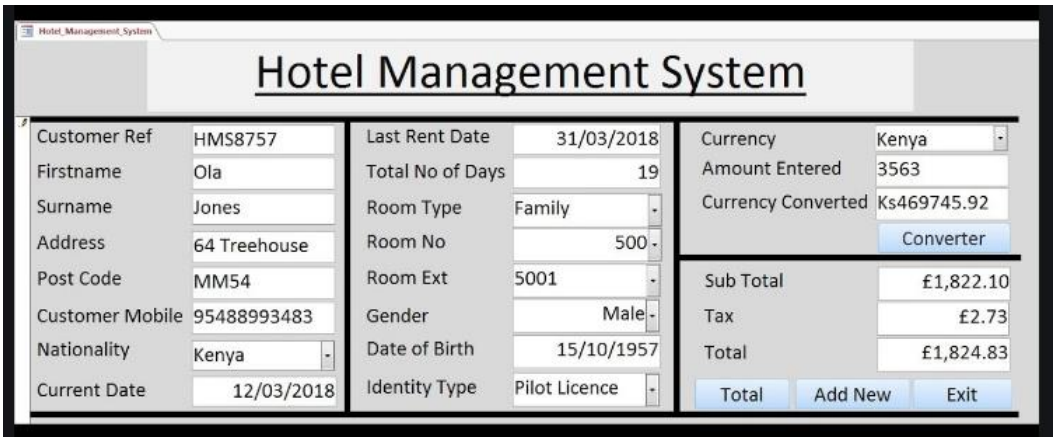


Fig 2

This is an example of a GUI system that shows how data would be entered into the database such as the first one above and so use of entry fields are necessary.

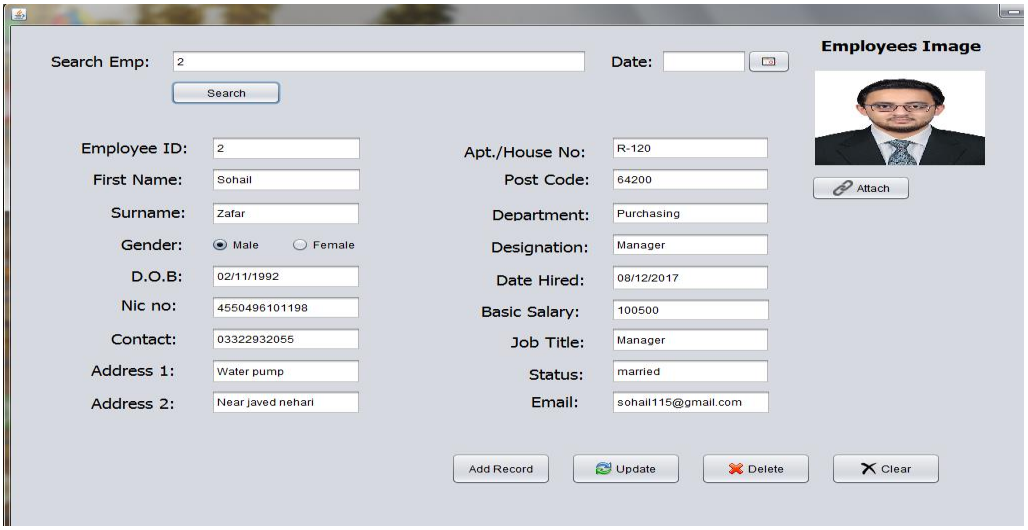


Fig 3

This is an example of an employee management system. Unlike the one above this software makes use of an update and delete function which I will incorporate into my program to add to the complexity. Furthermore it is easier to make sure data integrity is maintained when using these functions compared to the paper based system which is a problem with the current system

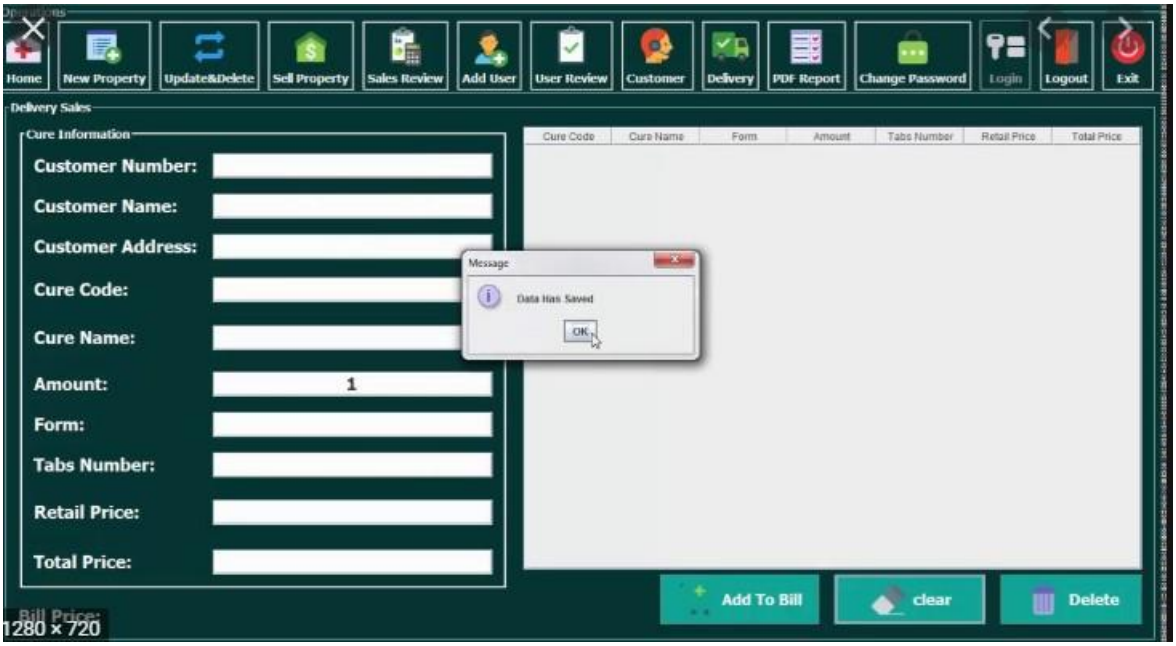


Fig 4

Unlike the previous example this system makes use of a display box (right). This makes viewing the data much easier compared to just reading it from an entry file

The End User, Proposal and Specific Objectives

The End User:

Ultimately my program will be used by the manager of the company and considering that she was currently using a paper-based system to record data, the program must be very easy to use as it is clear the manager isn't used to using computer software.

Specific Objectives:

1. The program must have a simplistic design in so that it is easily understandable by third party users such as the manager and potential subordinates. This can be proven by a positive response from the manager.
2. Program must have a menu
3. Program must connect to SQL and consequently make SQL tables
   - 3.1. Database tables must be easily manipulated
     - 3.1.1. Program must have an insert data function
     - 3.1.2. Program must have a delete data function
     - 3.1.3. Program must have an update data function
     - 3.1.4. Data can be entered using entry widgets
     - 3.1.5. Rows in the table must be able to be selected followed by automatic insert of field values into their respective entry widgets
       - 3.1.5.1. Functions that manipulate existing data should be able to be performed on rows selected from a display table

3.1.6. Program must have a function to search for data in a table based on given criteria

3.1.7. Program must have a function to display all data in the table

4. When inserting, updating and deleting data, data integrity must be maintained
   - 4.1. Databases must be normalised
   - 4.2. Program must prevent invalid data from being entered
   - 4.3. The program must calculate dynamic data automatically (as mentioned in the interview some data values need to be calculated)
5. Program must have a function to display customer booking information
   - 5.1. All customers should be displayed on a display table
     - 5.1.1. Customers on the display table should be able to be selected which will open a window displaying their information including the properties booked by that customer
     - 5.1.2. Properties booked by a customer should be able to be selected which will open a window showing information about that apartment
   - 5.2. All properties not in use should be displayed
6. Once a customer is booked into an apartment on the database, a document of the booking for the customer must be createds
7. Database must be portable

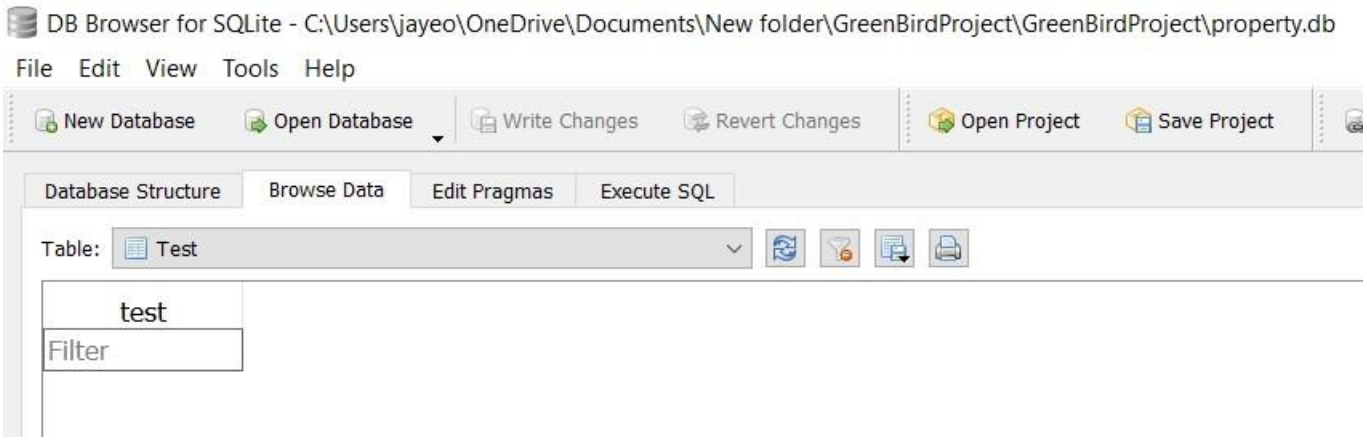
Prototype:

In order to check for the SQL connection and subsequent table creation I created a code that will connect to an sql database and create a table. This will be used extensively in the actual program and so a creating a prototype of its implementation was very useful.

Prototype code:

```
def Test():  #function to create the data base and its values
    con=sqlite3.connect('property.db') #connects to the property database or creates it if doesnt exist
    cursor=con.cursor()#alloes for database manipulation
    cursor.execute("PRAGMA foreign_keys = ON")



    cursor.execute('CREATE TABLE IF NOT EXISTS Test(\
            test text)')

    con.commit() #adds the table and fields to the database
    con.close() #closes the database
```

Proof of connection established and created table:



Design

Project Proposal/Aim:

I intend to create GUI database management software for the manager of GreenBird Properties which is an easier and more sustainable method of data management compared to their previous paper-based system. This database management software will allow the manager to book in a customer to an apartment, by storing information entered into a relational database through entry fields, as well as allow the manager to edit the property portfolio table which stores all information about the firm's properties. The program will have each function in a separate window: a window for booking in customers, editing the firm's portfolio and a window to check the status and information of customers and properties. Furthermore, after booking in a customer, a document of the booking will be created for use of the manager and the customer. The project will be programmed using object-oriented sPython 3.7 with the use of the tkinter module, sqlite3 and the datetime module.

System Design

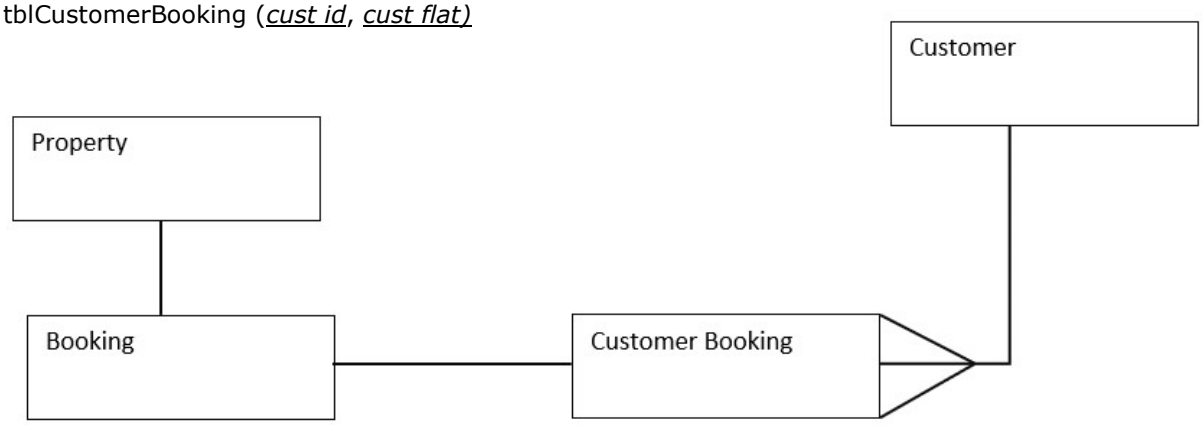
Database Design

tblProperty (idp, flat name, flat num, post code, town, city, bathrooms, bedrooms, rent price)

tblBookings (idb, duration, start date, end date, num of people, *flat name,* rent due date)

tblCustomer (idc, first name, surname, birth date, phone number, email)

tblCustomerBooking (*cust id*, *cust flat)*



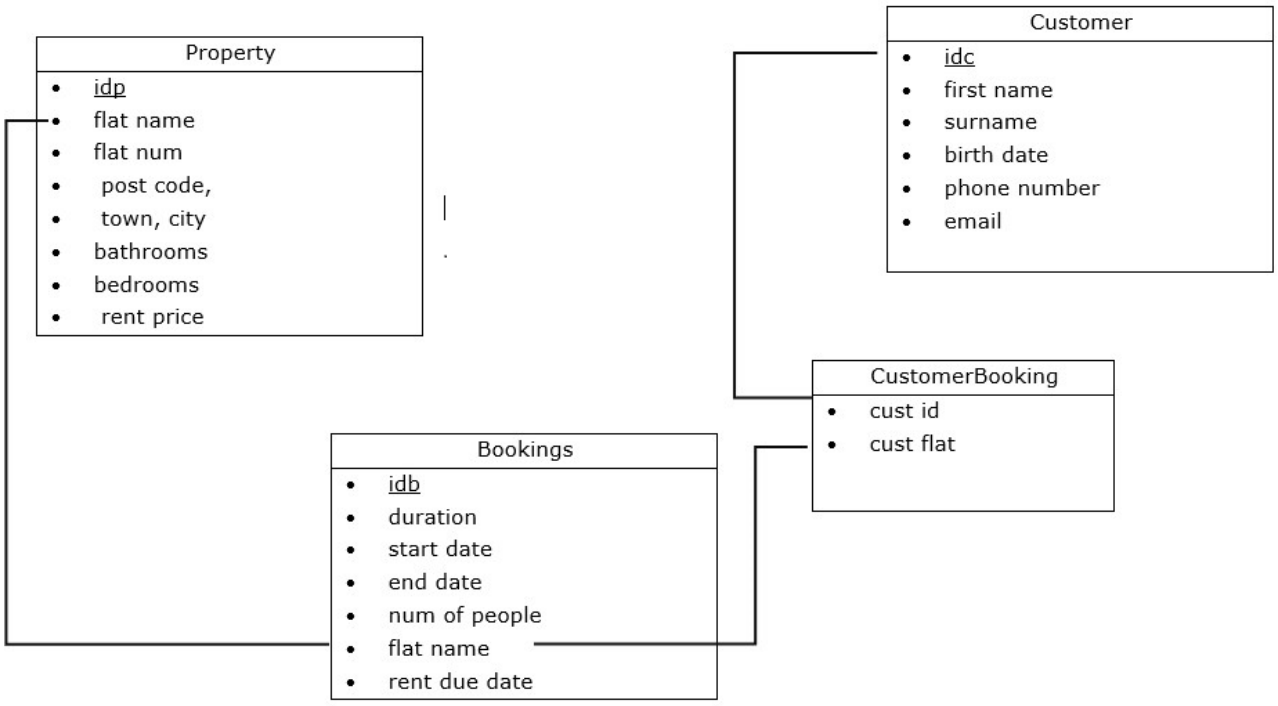


Table: Property		
Field	Key	Datatype
idp	Primary Key	Integer
flat_name	Foreign Key	Text
flat_num		Integer
post_code		Text
town		Text
city		Text
bathrooms		Text
bedrooms		Text
rent_price		Integer

Table: Bookings		
Field	Key	Datatype
idb	Primary Key	Integer
start_date		Text
end_date		text
num_of_people		Integer
flat_name	Foreign Key	Text
rent_due_date		Text

Table: Customer		
Field	Key	Datatype
idc	Primary Key	Integer
first_name		Text
surname		text
birth_date		texts
phone_number		Text
email		Text

Table: CustomerBookings		
Field	Key	Datatype
cust_id	Primary Key	Integer
cust_flat	Primary Key	Text

Key Queries That Will Be Used

1) Query to select the booking of a given customer

```
SELECT Bookings.*, Customer.first_name,Customer.surname FROM Bookings JOIN Customer \
 JOIN CustomerBooking ON Bookings.flat_name=CustomerBooking.cust_flat AND
Customer.idc=CustomerBooking.cust_id')
```

2) Query to delete a booking of a given customer

```
DELETE FROM Bookings WHERE EXISTS (SELECT CustomerBooking.cust_flat\
    FROM Property INNER JOIN (Customer INNER JOIN CustomerBooking ON Customer.idc =
CustomerBooking.cust_id) ON \
    Property.flat_name = CustomerBooking.cust_flat\
    WHERE Customer.first_name=? AND Customer.surname=? AND Customer.idc=? AND
Bookings.flat_name=Property.flat_name
```

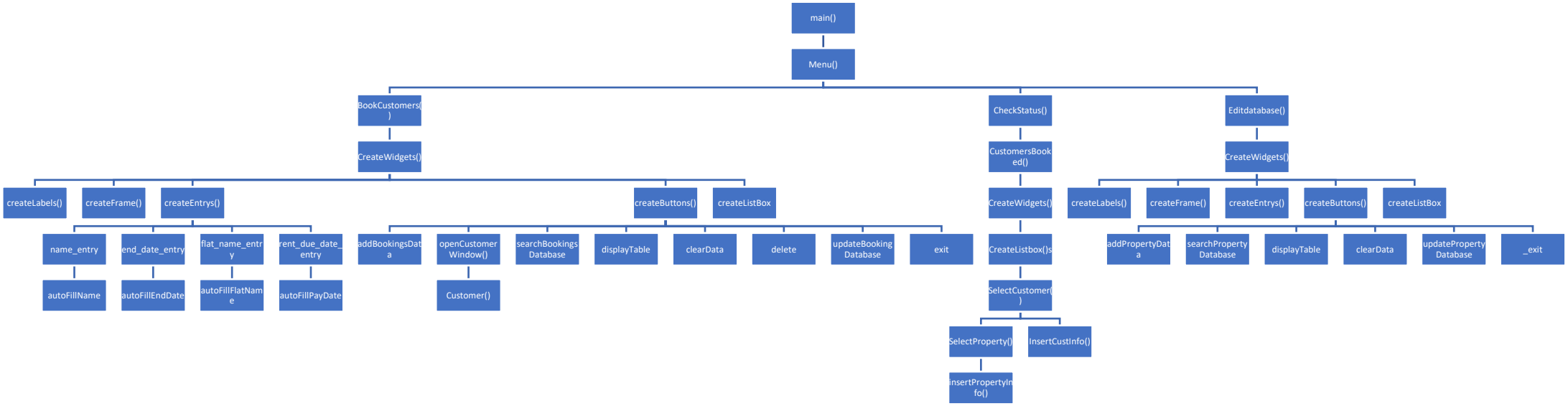
3) Query to get all the data of a customer

```
SELECT Bookings.flat_name FROM (Property INNER JOIN Bookings ON Property.flat_name =
Bookings.flat_name) \
    INNER JOIN (Customer INNER JOIN CustomerBooking ON Customer.idc =
CustomerBooking.cust_id) ON \
    Bookings.flat_name = CustomerBooking.cust_flat WHERE Customer.idc=? AND
Customer.first_name=? AND \
    Customer.surname=?
SELECT birth_date,phone_number,email FROM Customer WHERE first_name=? AND surname=?
```

Note: Continued in the technical solution

Hierarchy Diagram

Here is a hierarchy diagram of the program



Class Overview

Menu():

This class creates a window that acts as a menu. It has buttons that will call the other three main classes and thus open their windows (EditDatabase, BookCustomers, CheckStatus)

EditDatabase():

This is a class that creates an instance of the edit database window. This window is responsible for managing the data about the individual properties owned by GreenBird (the firm), e.g: flat_name, rent_price etc.

BookCustomers():

This window is responsible for managing data about a customer booking as well as making a customer booking

Customer():

This window is responsible for managing data about an individual customer

CheckStatus():

This class calls another class (CustomerBooking). This can be seen as redundant is in this fashion because of changes made causing its use to be relatively insignificant

CustomersBooked():

Creates a window that has a listbox that displays the current customers In the database. Customers can be clicked on which will then call the class CustomerInfo

CustomerInfo():

Creates a window that has a listbox displaying all customer information. Flats occupied by a customer can be clicked on which will call propertyInfo

PropertyInfo():

Creates a window that has a listbox displaying all information about a selected Property

Class Diagram

Below is a note to explain datatypes with the prefix tk

```
Menu()
-win~ Tk()
+clickcs
+clickbc
+clicked
+createMenu
+_exit
```

win is an instance of the tkinter module and thus creates a window

clickcs: stands for click check status this is a method that opens the check status window

clickbc: stands for click book customer this is a method that opens the book customer window

clicked: stands for click book customer this is a method that opens the edit database window

```
CheckStatus()
-check_status_window ~ tk()
+createWidgets
```

```
CustomersBooked()
-list_box_frame ~tk.Frame
-display_box ~ tk.Listbox
+createWidgets
+listboxCommand
+insertNameIntoListbox
+SelectCustomer
```

```
EditDatabase()
-edit_data_window~ tk()
-database_style~ ttk.style()

-main_frame
-data_frame
-data_entry_frame        ~ tk.Frame
-data_display_frame
-button_frame

-flat_name_entry
-flat_num_entry
-post_code_entry
-town_entry
-city_entry              ~ tk.Entry
-bathrooms_entry
-bedrooms_entry
-rent_price_entry

-add_datab
-search_datab
-display_datab
-clear_datab             ~ tk.Button
-delete_datab
-update_datab
-exitb

-display_box ~ tk.Listbox
+createStyle
+createFrame
+createLabels
+createEntrys
+createButtons
+listboxCommand
+createWidgets
+createListBox
+_exit
+clearData
+addPropertyData
+displayTable
+recordDisplayIndex
+delete
+deleteData
+getSqlCommand
+searchPropertyDatabase
+updatePropertyDatabase
+intEntryValidation
+validateName
+emptyEntryCheck
+validPostCodeCheck
+FlatNameAvailabiltyValidate
+databaseValid
```

```
BookCustomer()
-bookings_window~ tk()
-array ~ lists

-main_frame
-data_frame
-data_entry_frame        ~ tk.Frame
-data_display_frame
-button_frame

-name_entry
-duration_entry
-start_date_entry        ~ tk.Entry
-num_of_people_entry

-end_date_entry
-flat_name_entry
-rent_due_date_entry     ~tk.combo
                              -box

-add_customerb
-add_datab
-search_datab
-display_datab
-clear_datab             ~ tk.Button
-delete_datab
-update_datab
-exitb

-display_box ~ tk.Listbox
+createFrame
+createLabels
+createEntrys
+createButtons
+listboxCommand
+createWidgets
+bookingValid
+BookingAvailabiltyValidate
+validateDate
+emptyEntryCheck
+addBookingsData
+getSqlCommand
+searchBookingsDatabase
+createDocument
+delete
+autoFillFlatName
+getDateString
+stringDateToDatetime
+autoFillEndDate
+getPayDate
+autoFillPayDate
+getEndSentence
+openCustomerWindow
+splitName
+autoFillNames
+addCustBooking
+updateBookingDatabase
+displayTable
```

```
Customer()
-customer_window ~ tk()
-array ~ list

-main_frame
-data_frame
-data_entry_frame        ~ tk.Frame
-data_display_frame
-button_frame

-first_name_entry
-surname_entry
-birth_date_entry
-phone_number_entry
-email_entry             ~ tk.Entry

-add_datab
-search_datab
-display_datab
-clear_datab             ~ tk.Button
-delete_datab
-update_datab
-exitb

-display_box ~ tk.Listbox

+createStyle
+createFrame
+createLabels
+createEntrys
+createButtons
+listboxCommand
+createWidgets
+addCustomerData
+delete
+validEmailCheck
+validNameCheck
+validateDate
+validPhonenumber
+customerValid
+updateCustomerDatabase
+getSqlCommand
+searchCustomerDatabases
```

```
CustomersInfo()
-id ~ integer
-first_name ~ text
-surname ~ text
-customer_info_window ~tk()
-display_box ~ tk.listbox

+tupleToArray
+listboxCommand
+getCustInfo
+insert
+insertCustInfo
+SelectProperty
```

```
PropertyInfo()
-flat_name ~ text
-property_info_window ~ tk()
-list_box_frame ~ tk.Frame

+ createWidgets
+ getFlatInfo
+ insertPropertyInfo
```

Note:

tk.Frame: A frame is rectangular region on the screen. The frame widget is mainly used as a geometry master for other widgets, or to provide padding between other widgets.

tk.Label: The Label widget is a standard Tkinter widget used to display a text or image on the screen.

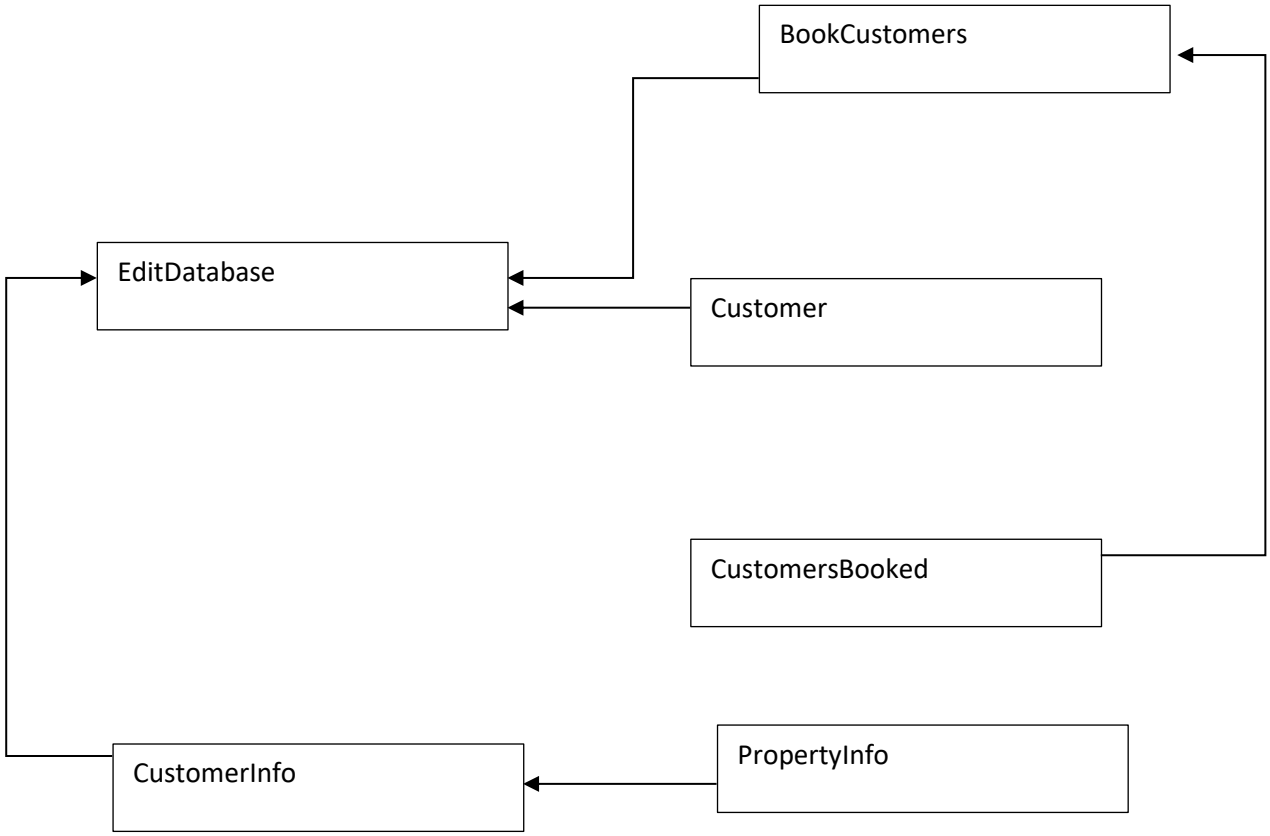
tk.Button: The Button widget is a standard Tkinter widget used to implement various kinds of buttons. You can associate a Python function or method with each button. When the button is pressed, Tkinter automatically calls that function or method.

tk(): instance of tkinter module, creates a root window which will hold all widgets

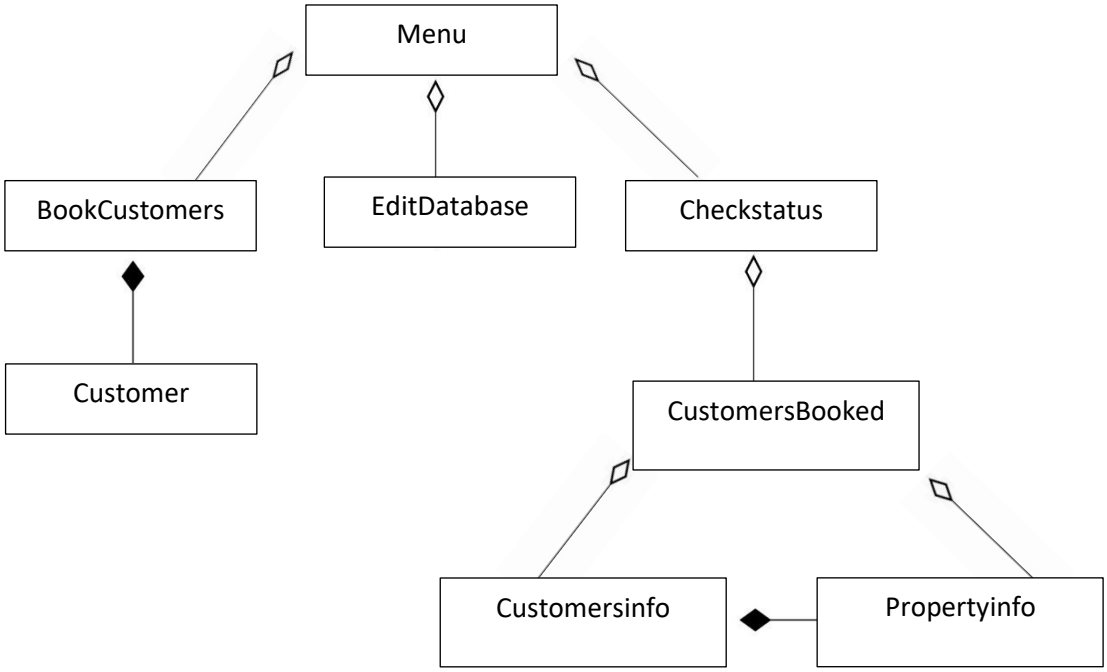
tk.Listbox: The Listbox widget is a standard Tkinter widget used to display a list of alternatives. The listbox can only contain text items

tk.entry: The Entry widget is a standard Tkinter widget used to enter or display a single line of text.

Inheritance Diagram

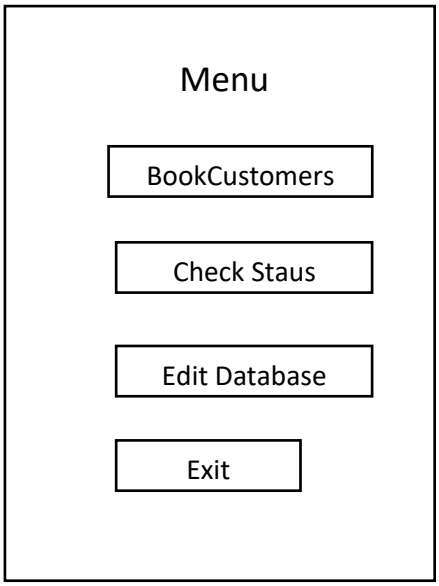


Association Diagram

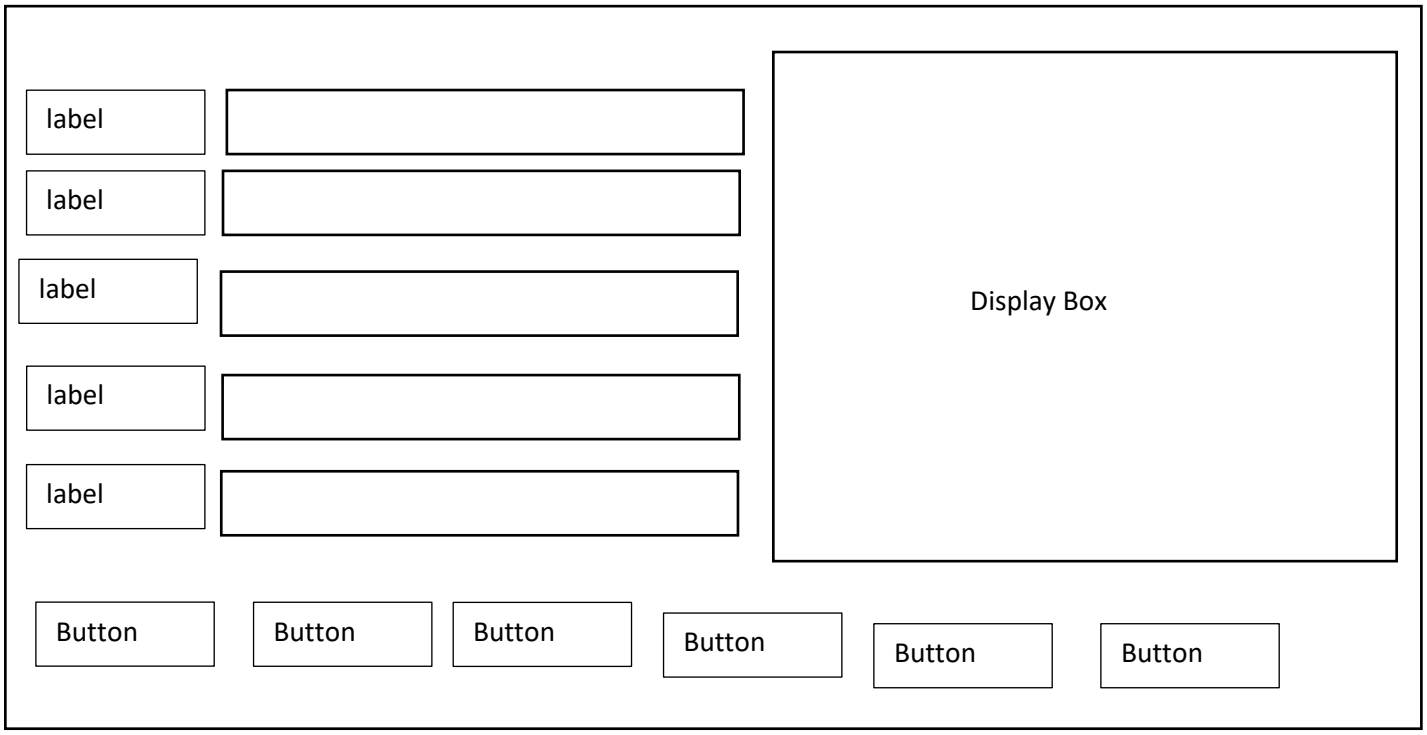


Screen Design

Design for menu



Design for database management windows



The above design will be used for all windows that involve managing a database, namely: Edit database(), Customer() and BookCustomer()

Design for display box windows: CustomerBooked() , CustomerInfo(),PropertyInfo()

CUSTOMER NAME

Flats Leased:
WINCHESTER

Customer info:
Birth Date: 14/04/2002
Phone Number: 093847364532
Email: HELO@GMAIL.COM

This display box will have text that can be selected and thus open new windows that display information

Technical Solution

File:GreenBirdProject.py

```
import tkinter as tk #imports the tkinter module which allow gui programming
from tkinter import ttk
import checkstatuswin as cs
import bookcustomerswin as bc
import editdatbwin as ed

from tkinter import messagebox as mbx

import bookingsdatabasesql as bdbsql
import customersql as cdbsql
import propertydatabasesql as pdbsql #imports the sql data base file

class Menu():

    def __init__(self):
        self.win=tk.Tk() #creates an instance of the tkinter module by calling its constructor
        self.win.title('GreenBird Properties')#creates the window title
        self.win.configure(bg='light grey')#sets the window background colour
        self.win.geometry('300x250')#sets the size of the window
        self.win.resizable(0,0)#set the window to a non resizble state


        pdbsql.createPropertyTable()# creates the property sql table
        bdbsql.createBookingTable() #creates the bookings sql table
        cdbsql.createCustomerTable() #creates the customer sql table
        cdbsql.createCustomerBookingTable()
        self.createMenu()#calls the createMenu window


    def _exit(self):
        close =mbx.askyesno('GreenBird Properties Database Management System',\
            'Confirm if you want to exit')#message box that asks yes or no to exit window
        if close >0:
            self.win.quit()
            self.win.destroy() #exits the window
            exit()

        return

    def clickcs(self):
        cs.CheckStatus(self.win)#calls the checkstatus class

    def clickbc(self):
        bc.BookCustomers(self.win) #calls the book customers class

    def clicked(self):
        ed.EditDatabase(self.win)#calls the edit database class

    def createMenu(self):

        menu_style=ttk.Style()
        menu_style.configure('My.TFrame',background ='light grey')#creats a style of frame to set
the background
        menu_style.configure('My.Tbutton',background='light grey')#sets a style for button to set
the background
        menu_style.configure('My.TLabel',background='light grey',font=('Calibri light',20))#sets
style for label sets background and font


        start_menu =ttk.Frame(self.win, style ='My.TFrame')#creates a frame widget that will contain
other widgets
```

```
        label=ttk.Label(start_menu,text='Menu:',style = 'My.TLabel') #creates a text label
        label.grid()#places the label in the specified location on the gui in grid form

        book_cutsomersb=ttk.Button(start_menu,text ='Book in Customers',\
            style ='My.TButton',command=self.clickbc)#creats a click button that performs the
command by callig the function it references
                                                        #calls function clickbc

        book_cutsomersb.grid()#places the button in the next available space on the grid

        check_statusb=ttk.Button(start_menu,text='Check Status',\
            style ='My.TButton',command=self.clickcs)#calls function clickcs
        check_statusb.grid()#



        edit_databaseb=ttk.Button(start_menu,text= 'Edit Database',\
            style ='My.TButton',command=self.clicked)#calls function clicked
        edit_databaseb.grid()

        exitb=ttk.Button(start_menu,text='Exit',command=self._exit,style ='My.TButton')#calls the
function exit
        exitb.grid()

        for i in start_menu.winfo_children(): # a for loop which loops through all widgets in the
startmenu frame
            i.grid_configure(pady=5,padx=50)#configures all the widgets and specifes the padding
along x and y


        start_menu.pack()#organises the position of the frame in the window


if __name__=='__main__':

    menu=Menu() #creates an instance of menu class
    menu.win.mainloop() #mainloop is a tkinter function that runs the tkinter module
```

Filen: editdatabwin.py

```
import tkinter as tk #imports the tkinter module which allow gui programming
from tkinter import ttk #imports a variant of the tk module that gives a more modern look for the
gui
from tkinter import scrolledtext as st #imports the scrolled text widget
import propertydatabasesql as pdbsql #imports the sql data base file
from tkinter import messagebox as mbx #imports a module that allows the use of pop up message boxes
import bookingsdatabasesql as bdbsql #imports the sql data base file
import re

class EditDatabase(): #edit database class created

    def __init__(self,win): # win from greenbird menu module is passed as a parameter for the main
parent window
        self.edit_data_window=tk.Toplevel(win)#creates a instance of the tkinter module in the form
of a new window
        self.edit_data_window.geometry('1090x420')#sets a default size for the window when opened
        self.edit_data_window.resizable(0,0)#makes the window non resizable
        self.createStyle() #calls the create style function
        self.createWidgets() #calls the create widgets function

        self.entry_ary=[self.flat_name_entry,self.flat_num_entry,\
        self.post_code_entry,self.town_entry,self.city_entry,self.bathrooms_entry,\
        self.bedrooms_entry,self.rent_price_entry]#creates an array that holds the variables of all
entries

    def createStyle(self):
        self.database_style=ttk.Style()#calls the module for creating a format style for widgets
        self.database_style.configure('MF.TLabelframe.Label',font =('calibri light',15)) #creates a
style for label
        self.database_style.configure('DEF.TLabelframe.Label',font =('calibri light',15))
        self.database_style.configure('DF.TLabelframe')
        self.database_style.configure('B.TButton',font =('calibri',13),height=3,width=16)
        self.database_style.configure('DEF.TLabel',font=('calibri light',14))
        self.database_style.configure('DEF.TEntry',font=('calibri',14))


    def createFrame(self):
        self.main_frame =ttk.LabelFrame(self.edit_data_window,text ='GreenBird Properties Database
Management')#creaetes a main frame that holds all frames
        self.main_frame.pack(anchor ='w',fill='both')


        self.data_frame=ttk.Frame(self.main_frame,style ='DF.TLabelframe')#creates a frame that
holdd the data widgets and frames
        self.data_frame.pack(side=tk.TOP,fill='x',pady=6)

        self.data_entry_frame=ttk.LabelFrame(self.data_frame,text ='Property Info:',style
='DEF.TLabelframe')#creates a labeled frame that will hold all data entries
        self.data_entry_frame.pack(side=tk.LEFT,anchor ='nw')


        self.data_display_frame=ttk.LabelFrame(self.data_frame,text='Property Details:',style
='DEF.TLabelframe')#creates a labeled frame that will hold all displayed data
        self.data_display_frame.pack(side=tk.RIGHT,anchor ='w')

        self.button_frame=ttk.Frame(self.main_frame)#creates a button frame that will hold all
buttons in the mainframe
        self.button_frame.pack(side=tk.BOTTOM,fill='x')
```

```
    def createLabels(self):

        flat_name_lbl=ttk.Label(self.data_entry_frame,text='Flat Name: ',style ='DEF.TLabel')
#creates a label
        flat_name_lbl.grid(row=0,column=0) #determines the position of a label on the grid

        flat_num_lbl=ttk.Label(self.data_entry_frame,text='Flat No.: ',style ='DEF.TLabel')
        flat_num_lbl.grid(row=1,column=0)

        post_code_lbl=ttk.Label(self.data_entry_frame,text='Post Code: ',style ='DEF.TLabel')
        post_code_lbl.grid(row=2,column=0)

        town_lbl=ttk.Label(self.data_entry_frame,text='Town: ',style ='DEF.TLabel')
        town_lbl.grid(row=3,column=0)

        city_lbl=ttk.Label(self.data_entry_frame,text='City: ',style ='DEF.TLabel')
        city_lbl.grid(row=4,column=0)

        bathrooms_lbl=ttk.Label(self.data_entry_frame,text='No. of Bathrooms: ',style ='DEF.TLabel')
        bathrooms_lbl.grid(row=5,column=0)

        bedrooms_lbl=ttk.Label(self.data_entry_frame,text='No. of Bedrooms: ',style ='DEF.TLabel')
        bedrooms_lbl.grid(row=6,column=0)

        rent_price_lbl=ttk.Label(self.data_entry_frame,text='Renting Price(£): ',style
='DEF.TLabel')
        rent_price_lbl.grid(row=7,column=0)


  def createEntrys(self):

        valid =self.data_entry_frame.register(self.intEntryValidation)#declares the function that
will be used as a validation for entries requiring integers
        flat_name=tk.StringVar()#declares the variable for the entry as a string
        flat_num=tk.IntVar() #declares the variable for the entry as a integer
        post_code=tk.StringVar()
        town=tk.StringVar()
        city=tk.StringVar()
        bathrooms=tk.StringVar()
        bedrooms=tk.StringVar()
        rent_price=tk.IntVar()


        self.flat_name_entry=ttk.Entry(self.data_entry_frame,textvariable =flat_name,width=40,font
=('calibri',15)) #creates a data entry widget
        self.flat_name_entry.grid(row=0,column=1)#positions the entry on the grid

        self.flat_num_entry=ttk.Entry(self.data_entry_frame,textvariable =flat_num,width=40,font
=('calibri',15),\
            validate ='key',validatecommand=(valid,'%P'))#creates a data entry widget and sets the
function valid as the validate command
        self.flat_num_entry.grid(row=1,column=1)

        self.post_code_entry=ttk.Entry(self.data_entry_frame,textvariable =post_code,width=40,font
=('calibri',15))
        self.post_code_entry.grid(row=2,column=1)

        self.town_entry=ttk.Entry(self.data_entry_frame,textvariable =town,width=40,font
=('calibri',15))
        self.town_entry.grid(row=3,column=1)


        self.city_entry=ttk.Entry(self.data_entry_frame,textvariable =city,width=40,font
=('calibri',15))
        self.city_entry.grid(row=4,column=1)
```

```
        self.bathrooms_entry=ttk.Entry(self.data_entry_frame,textvariable =bathrooms,width=40,font
=('calibri',15),\
            validate ='key',validatecommand=(valid,'%P'))
        self.bathrooms_entry.grid(row=5,column=1)

        self.bedrooms_entry=ttk.Entry(self.data_entry_frame,textvariable =bedrooms,width=40,font
=('calibri',15),\
            validate ='key',validatecommand=(valid,'%P'))
        self.bedrooms_entry.grid(row=6,column=1)


        self.rent_price_entry=ttk.Entry(self.data_entry_frame,textvariable =rent_price,width=40,font
=('calibri',15),\
            validate ='key',validatecommand=(valid,'%P'))
        self.rent_price_entry.grid(row=7,column=1)

        for i in self.data_entry_frame.winfo_children(): #creates a for loop for each widget in the
data entry frame
            i.grid_configure(pady=5,sticky='w') #configures the padding and position of each widget

    def createButtons(self):

        self.add_datab=ttk.Button(self.button_frame,text='Add New',command =self.addPropertyData)
#creates a button that when clicked performs the command addpropertydata
        self.add_datab.grid(row=0,column=0) #positions the button on the grid within its frame

        self.search_datab=ttk.Button(self.button_frame,text='Search',command
=self.searchPropertyDatabase)
        self.search_datab.grid(row=0,column=1)

        self.display_datab=ttk.Button(self.button_frame,text='Display',command =self.displayTable)
        self.display_datab.grid(row=0,column=2)

        self.clear_datab=ttk.Button(self.button_frame,text='Clear',command=
lambda:self.clearData(self.entry_ary))#lambda allows parameters to be passed into the command
        self.clear_datab.grid(row=0,column=3)

        self.delete_datab=ttk.Button(self.button_frame,text='Delete',command =self.delete)
        self.delete_datab.grid(row=0,column=4)

        self.update_datab=ttk.Button(self.button_frame,text='Update',command
=self.updatePropertyDatabase)
        self.update_datab.grid(row=0,column=5)

        self.exitb=ttk.Button(self.button_frame,text='Exit',command
=lambda:self._exit(self.edit_data_window))
        self.exitb.grid(row=0,column=6)

        for i in self.button_frame.winfo_children(): #creates a for loop for each widget in button
frame
            i.configure(style= 'B.TButton')#gives each button the style b.tbutton

    def listboxCommand(self):
        self.display_box.bind('<<ListboxSelect>>',\
            lambda event:self.recordDisplayIndex(event,self.entry_ary))#performs the command when
data is selected on the listbox

    def createWidgets(self):
#=========================================Frames=====================================================
============================
        self.createFrame()
        self.createLabels()
        self.createEntrys()
        self.createButtons()
```

```
        self.createListBox(self.data_display_frame,100,12)
        self.listboxCommand()
```

```
  def createListBox(self,frame,w,h):


        self.display_box=tk.Listbox(frame,width=w,height=h,font =('arial',15)) #creates a list box
that will display data and allow it to  be selected

        yscroll_bar=st.Scrollbar(frame,orient =tk.VERTICAL,command =self.display_box.yview) #creates
a vertical scrollbar

        xscroll_bar=st.Scrollbar(frame,orient =tk.HORIZONTAL,command =self.display_box.xview)
#creates a horizontal scrollbar

        self.display_box.configure(xscrollcommand =xscroll_bar.set)#sets the horizontal scroll bar
onto the list box

        self.display_box.configure(yscrollcommand =yscroll_bar.set)#sets the vertical scroll bar
onto the listbox

        yscroll_bar.pack(fill='y',side=tk.RIGHT)#puts the horizontal scroll bar onto the righ
        xscroll_bar.pack(fill='x',side=tk.BOTTOM) #puts the horizontal scroll bar onto the left

        self. display_box.pack(side=tk.TOP, fill='both', expand=1)#organis the list box the the top
of the window and expands within its frame

    def _exit(self,arg):
        exit =mbx.askyesno('GreenBird Properties Database Management System',\
            'Confirm if you want to exit')#message box that asks yes or no to exit window
        if exit >0:
            arg.destroy() #exits the window

        return

    def clearData(self,arg): #removes all data from the data entry widgets *
        for i in range(len(arg)):
            arg[i].delete(0,tk.END) #tk END refers to the end of the string and 0 the begining thus
removing the whole string

    def addPropertyData(self):

        valid,empty=self.databaseValid()#returns true or false for values valid and empty
        #adds the data entered in the entry into the property  database
        if valid and not empty:
            pdbsql.addPropertyData(self.flat_name_entry.get().upper(),\
            self.flat_num_entry.get(),self.post_code_entry.get().upper(),\
            self.town_entry.get().upper(),\
            self.city_entry.get().upper(),self.bathrooms_entry.get()\
            ,self.bedrooms_entry.get(),self.rent_price_entry.get()) #calls sql function to add to
       database
            self.displayTable()#displays the values on the database

    def displayTable(self,table='Property'):
        self.display_box.delete(0,tk.END)#empties the list box
        for i in pdbsql.viewData(table):#calls sql function that returns all data values as an array
```

```
            self.display_box.insert(tk.END,i) #the value returned from the sql function is then
printed on the display box

    def recordDisplayIndex(self,event,arg):#*
        #global self.sd#stands for search data
        try:
            search=self.display_box.curselection()[0] #returns an index that is used to specify the
location of the data on the listbox line i.e first line will be 0

            self.sd =self.display_box.get(search) #returns a list of data displayed on the list box

            for i in range(1,(len(arg)+1)):

                arg[i-1].delete(0,tk.END)
                arg[i-1].insert(tk.END,self.sd[i])#inserts each data in the list into entry


        except Exception as err:
            pass




    def delete(self):

        self.deleteData('Property','idp')#deletes the row selected

    def deleteData(self,table,sqlid):
        pdbsql.deleteData(self.sd[0],table,sqlid)#deletes data selected
        self.displayTable(table)#displays data on list box

    def getSqlCommand(self):#generates a sql command based on if values have been typed into entry
widgets

        sql_string='' #used to dynamiclly add sql command
        sql_variables=[] #will hold the value for each entry widget that has been typed into

        if self.flat_name_entry.get().isspace() or self.flat_name_entry.get()==='':
            pass
        else:
            sql_string+='flat_name=? '
            sql_variables.append(self.flat_name_entry.get().upper())

        if self.flat_num_entry.get().isspace() or self.flat_num_entry.get()==='':
            pass
        else:
            if sql_string==='':
                sql_string+='flat_num=? '
            else:
                sql_string+='AND flat_num=? '

            sql_variables.append(self.flat_num_entry.get())

        if self.post_code_entry.get().isspace() or self.post_code_entry.get()==='':
            pass
        else:
            if sql_string==='':
                sql_string+='post_code=? '
            else:
                sql_string+='AND post_code=? '
            sql_variables.append(self.post_code_entry.get().upper())

        if self.town_entry.get().isspace() or self.town_entry.get()==='':
            pass
        else:
```

```
            if sql_string==''':
                sql_string+='town=? '
            else:
                sql_string+='AND town=? '
            sql_variables.append(self.town_entry.get().upper())

        if self.city_entry.get().isspace() or self.city_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='city=? '
            else:
                sql_string+='AND city=? '
            sql_variables.append(self.city_entry.get().upper())


        if self.bathrooms_entry.get().isspace() or self.bathrooms_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='bathrooms=? '
            else:
                sql_string+='AND bathrooms=? '

            sql_variables.append(self.bathrooms_entry.get())


        if self.bedrooms_entry.get().isspace() or self.bedrooms_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='bedrooms=? '
            else:
                sql_string+='AND bedrooms=? '

            sql_variables.append(self.bedrooms_entry.get())

        if self.rent_price_entry.get().isspace() or self.rent_price_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='rent_price=? '
            else:
                sql_string+='AND rent_price=? '

            sql_variables.append(self.rent_price_entry.get())



        return sql_string,sql_variables


    def searchPropertyDatabase(self,):


        self.display_box.delete(0,tk.END)#empties display box

        sql_string,sql_variables=self.getSqlCommand()

        for i in pdbsql.searchPropertyData(sql_string,sql_variables):

            self.display_box.insert(tk.END,i,str('')) #looks for correlating data from entry and
inserts it into displaybox

    def updatePropertyDatabase(self):
```

```
        #updates data by inserting entry values into the the sql database
        pdbsql.updatePropertyData(self.sd[0],self.flat_name_entry.get().upper(),\
        self.flat_num_entry.get(),\
        self.post_code_entry.get().upper(),self.town_entry.get().upper(),\
        self.city_entry.get().upper(),self.bathrooms_entry.get(),\
        self.bedrooms_entry.get(),self.rent_price_entry.get())

        self.displayTable()


    def intEntryValidation(self,inp):


        if inp.isdigit(): #checks that data entered is only an integer
            return True
        elif inp is '': #or if data is embty
            return True
        else:
            return False

    def validateName(self,name):

        if not re.match(r"^[A-Za-z]+\s*[A-Za-z]*$",name): #format validator
            return False

        else:

            return True

    def emptyEntryCheck(self,entry_start_index):
        entrys=self.data_entry_frame.winfo_children()

        for i in range(entry_start_index,len(entrys)): #starts at 5 because widgets before that are
labels and we only want entry widgets
            if entrys[i].get() =='': #checks if an entry is empty
                empty=True
                return empty
            else:
                empty=False

        return empty

    def validPostCodeCheck(self):

        post_code=self.post_code_entry.get()
        if not re.match(r"^[A-Za-z]+[A-Za-z]*[0-9][0-9]\s*[0-9][A-Za-z][A-Za-z]",post_code):
            return False
        else:
            return True

    def FlatNameAvailabiltyValidate(self):

        flats=pdbsql.getAllFlats()#gets all flats
        if self.flat_name_entry.get().upper() in flats: #checks if the flat has been booked

            return False
        else:

            return True

    def databaseValid(self):
        valid=False
        empty=True

        empty=self.emptyEntryCheck(8)
```

```
        if empty:
            mbx.showerror('Error','Fill out all remaining fields!')
            return valid,empty


        if self.FlatNameAvailabiltyValidate():
            valid=True
        else:
            valid=False
            mbx.showerror('Error',f'{self.flat_name_entry.get()} is in the database already!')
            return valid,empty

        if self.validPostCodeCheck():
            valid=True
        else:
            valid=False
            mbx.showerror('Error',f'Incorrect format for post code!')
            return valid,empty

        if self.validateName(self.flat_name_entry.get()):
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for flat name!')
            return valid,empty

        if self.validateName(self.town_entry.get()):
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for town!')
            return valid,empty

        if self.validateName(self.city_entry.get()):
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for city!')
            return valid,empty

        return valid,empty
```

File: bookcustomerswin.py

```
import tkinter as tk #imports the tkinter module which allow gui programming
from tkinter import ttk
from editdatbwin import EditDatabase
from tkinter import scrolledtext as st
import bookingsdatabasesql as bdbsql
import propertydatabasesql as pdbsql
import customersql as cdbsql
from tkinter import messagebox as mbx
import customerwin as cr
import datetime
from docx import *
from docx.enum.text import WD_ALIGN_PARAGRAPH

class BookCustomers(EditDatabase): #inherits from editdatabase
    def __init__(self,win):
        self.win=win
        self.bookings_window=tk.Toplevel(win)#creates a instance of the tkinter module in the form
of a new window
        self.bookings_window.geometry('1090x440')#sets a default size for the window when opened
        self.bookings_window.resizable(0,0)
        self.createStyle()
        self.createWidgets()

        self.array=[self.name_entry,self.duration_entry,self.start_date_entry,\
        self.end_date_entry,self.num_of_people_entry,\
        self.flat_name_entry,self.rent_due_date_entry]#creates a list that holds all entrys


    def createFrame(self):

        main_frame =ttk.LabelFrame(self.bookings_window,text ='GreenBird Properties
Bookings')#creaetes a main frame that holds all frames
        main_frame.pack(anchor ='w',fill='both')


        self.data_frame=ttk.Frame(main_frame,style ='DF.TLabelframe')#creates a frame that holdd the
data widgets and frames
        self.data_frame.pack(side=tk.TOP,fill='x',pady=6)

        self.data_entry_frame=ttk.LabelFrame(self.data_frame,text ='Bookings:',style
='DEF.TLabelframe')#creates a labeled frame that will hold all data entries
        self.data_entry_frame.pack(side=tk.LEFT,anchor ='nw')


        self.data_display_frame=ttk.LabelFrame(self.data_frame,text='Available Bookings:',style
='DEF.TLabelframe')#creates a labeled frame that will hold all displayed data
        self.data_display_frame.pack(side=tk.RIGHT,anchor ='w')

        self.button_frame=ttk.Frame(main_frame)#creates a button frame that will hold all buttons in
the mainframe
        self.button_frame.pack(side=tk.BOTTOM,fill='x')

    def createLabels(self):

        #creates a label
        name_lbl=ttk.Label(self.data_entry_frame,text='Customer Name: ',style ='DEF.TLabel')
        name_lbl.grid(row=1,column=0)

        duration_lbl=ttk.Label(self.data_entry_frame,text='Duration (Months):',style ='DEF.TLabel')
        duration_lbl.grid(row=2,column=0)
```

```
        start_date_lbl=ttk.Label(self.data_entry_frame,text='Start Date (dd/mm/yyyy):',style
='DEF.TLabel')
        start_date_lbl.grid(row=3,column=0)

        end_date_lbl=ttk.Label(self.data_entry_frame,text='End Date:',style ='DEF.TLabel')
        end_date_lbl.grid(row=4,column=0)


        num_of_people_lbl=ttk.Label(self.data_entry_frame,text='Number of People: ',style
='DEF.TLabel')
        num_of_people_lbl.grid(row=5,column=0)

        flat_name_lbl=ttk.Label(self.data_entry_frame,text='Flat Name: ',style ='DEF.TLabel')
#creates a label
        flat_name_lbl.grid(row=6,column=0)

        rent_due_date_lbl=ttk.Label(self.data_entry_frame,text='Rent Due Date: ',style
='DEF.TLabel')
        rent_due_date_lbl.grid(row=7,column=0)

        tday=datetime.datetime.now() #uses the date time module to get current date
        date=self.getDateString(tday)#converts the date into a string

        date_lbl=ttk.Label(self.data_entry_frame,text = f'{date}')
        date_lbl.grid(row=8,column=0,sticky='w')

    def createEntrys(self):

        name=tk.StringVar()# declares variables types
        flat_name=tk.StringVar()
        duration=tk.IntVar()
        start_date=tk.StringVar()
        end_date=tk.StringVar()
        num_of_people=tk.IntVar()

        rent_due_date=tk.StringVar()
        phone_number=tk.StringVar()
        email=tk.StringVar()


        #creates entries
        valid =self.data_entry_frame.register(self.intEntryValidation) #same as the edit database
class

        self.name_entry=ttk.Combobox(self.data_entry_frame,textvariable =name,width=40,font
=('calibri',15),postcommand=self.autoFillNames)
        self.name_entry.grid(row=1,column=1)


        self.duration_entry=ttk.Entry(self.data_entry_frame,textvariable =duration,width=40,font
=('calibri',15),\
            validate ='key',validatecommand=(valid,'%P'))
        self.duration_entry.grid(row=2,column=1)

        self.start_date_entry=ttk.Entry(self.data_entry_frame,textvariable =start_date,width=40,font
=('calibri',15))
        self.start_date_entry.grid(row=3,column=1)

        self.end_date_entry=ttk.Combobox(self.data_entry_frame,textvariable =end_date,width=40,font
=('calibri',15),\
            postcommand= lambda: self.autoFillEndDate(self.start_date_entry.get()))
        self.end_date_entry.grid(row=4,column=1)
```

```
        self.num_of_people_entry=ttk.Entry(self.data_entry_frame,textvariable
=num_of_people,width=40,font =('calibri',15),\
            validate ='key',validatecommand=(valid,'%P'))
        self.num_of_people_entry.grid(row=5,column=1)

        self.flat_name_entry=ttk.Combobox(self.data_entry_frame,textvariable
=flat_name,width=40,font =('calibri',15)\
            ,postcommand= self.autoFillFlatName)
        self.flat_name_entry.grid(row=6,column=1)

        self.rent_due_date_entry=ttk.Combobox(self.data_entry_frame,textvariable
=rent_due_date,width=40,font =('calibri',15),\
            postcommand=self.autoFillPayDate)
        self.rent_due_date_entry.grid(row=7,column=1)

    def createWidgets(self):

        self.createFrame()
        self.createLabels()
        self.createEntrys()
        self.createListBox(self.data_display_frame,100,12)
        self.listboxCommand()
        self.createButtons()

        for i in self.data_entry_frame.winfo_children():
            i.grid_configure(pady=5,sticky='w')

    def listboxCommand(self):

        self.display_box.bind('<<ListboxSelect>>',lambda event
:self.recordDisplayIndex(event,self.array))

    def createButtons(self):

        #creates click buttons
        add_customerb=ttk.Button(self.data_entry_frame,text='Add New Customer',command
=self.openCustomerWindow)
        add_customerb.grid(row=0,column=0)

        add_datab=ttk.Button(self.button_frame,text='Book Customer',command =self.addBookingsData)
        add_datab.grid(row=0,column=0)

        search_datab=ttk.Button(self.button_frame,text='Search
Customer',command=self.searchBookingsDatabase)
        search_datab.grid(row=0,column=1)

        display_datab=ttk.Button(self.button_frame,text='Display
Customers',command=lambda:self.displayTable('Bookings'))
        display_datab.grid(row=0,column=2)


clear_datab=ttk.Button(self.button_frame,text='Clear',command=lambda:self.clearData(self.array))
        clear_datab.grid(row=0,column=3)

        delete_datab=ttk.Button(self.button_frame,text='Delete',command =self.delete)
        delete_datab.grid(row=0,column=4)

        self.update_datab=ttk.Button(self.button_frame,text='Update',command
=self.updateBookingDatabase)
        self.update_datab.grid(row=0,column=5)


exitb=ttk.Button(self.button_frame,text='Exit',command=lambda:self._exit(self.bookings_window))
        exitb.grid(row=0,column=6)
```

```
        for i in self.button_frame.winfo_children():
            i.configure(style= 'B.TButton')


    def bookingValid(self,update):

        valid=False
        empty=True

        empty=self.emptyEntryCheck(8)

        if empty:
            mbx.showerror('Error','Fill out all remaining fields!')
            return valid,empty

        if self.BookingAvailabiltyValidate(update):
            valid=True
        else:
            valid=False
            mbx.showerror('Error',f'{self.flat_name_entry.get()} has been leased already!')
            return valid,empty

        if self.validateDate(self.start_date_entry.get()):
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for date should be DD/MM/YYYY!')
            return valid,empty

        if self.validateDate(self.end_date_entry.get()):
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for date should be DD/MM/YYYY!')
            return valid,empty

        return valid,empty

    def BookingAvailabiltyValidate(self,update):

        flats=bdbsql.getAllFlats()#gets all flats
        if self.flat_name_entry.get() in flats and not update: #checks if the flat has been booked
update is true or false whether the booking validation is occurring when adding a new value or
updating a current one . if using the update function then update is true

            return False
        else:

            return True

    def validateDate(self,date_text):
        try:#attempts to convert a string to datetime, if successful then returns trues
            date_text=date_text.split('/')
            date=date_text[0]+'-'+date_text[1]+'-'+date_text[2]
            datetime.datetime.strptime(date, '%d-%m-%Y')
            return True
        except:

            return False

    def emptyEntryCheck(self,entry_start_index):
        entrys=self.data_entry_frame.winfo_children()

        for i in range(entry_start_index,len(entrys)-1): #starts at 5 because widgets before that
are labels and we only want entry widgets
```

```
            if entrys[i].get() ==''': #checks if an entry is empty
                empty=True
                return empty
            else:
                empty=False

        return empty

    def addBookingsData(self):

        valid,empty=self.bookingValid(False)


        if valid and not empty:
            bdbsql.addBookingsData(self.duration_entry.get(),self.start_date_entry.get(),\
                self.end_date_entry.get(),self.num_of_people_entry.get(),\
                self.flat_name_entry.get(),self.rent_due_date_entry.get()) #adds data from entrys
and puts in sql database
            self.addCustBooking()
            self.displayTable('Bookings')
            self.createDocument() #creates a document based on bookings

    def getSqlCommand(self):

        sql_string=''
        sql_variables=[]

        if self.name_entry.get().isspace() or self.name_entry.get()==''':
            pass
        else:
            first_name,surname=self.splitName(self.name_entry.get())
            sql_string+='first_name=? AND surname=? '

            sql_variables.append(first_name)
            sql_variables.append(surname)


        if self.duration_entry.get().isspace() or self.duration_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='duration=? '
            else:
                sql_string+='AND duration=? '

            sql_variables.append(self.duration_entry.get())

        if self.start_date_entry.get().isspace() or self.start_date_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='start_date=? '
            else:
                sql_string+='AND start_date=? '

            sql_variables.append(self.start_date_entry.get())

        if self.end_date_entry.get().isspace() or self.end_date_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='end_date=? '
            else:
                sql_string+='AND end_date=? '
            sql_variables.append(self.end_date_entry.get())
```

```
        if self.num_of_people_entry.get().isspace() or self.num_of_people_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='num_of_people=? '
            else:
                sql_string+='AND num_of_people=? '
            sql_variables.append(self.num_of_people_entry.get())

        if self.flat_name_entry.get().isspace() or self.flat_name_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='flat_name=? '
            else:
                sql_string+='AND flat_name=? '
            sql_variables.append(self.flat_name_entry.get())


        if self.rent_due_date_entry.get().isspace() or self.rent_due_date_entry.get()==''':
            pass
        else:
            if sql_string==''':
                sql_string+='rent_due_date=? '
            else:
                sql_string+='AND rent_due_date=? '

            sql_variables.append(self.rent_due_date_entry.get())


        return sql_string,sql_variables



    def searchBookingsDatabase(self):
        self.display_box.delete(0,tk.END)
        #uses entry values as search criteria

        sql_string,sql_variables=self.getSqlCommand()

        for i in bdbsql.searchBookingsData(sql_string,sql_variables):
            self.display_box.insert(tk.END,i) #inserts different data into display box based on
index

    def createDocument(self):

        location_list=pdbsql.getLocation(self.flat_name_entry.get()) #gets a list that has location
data of an apartment from sql database
        occpts=bdbsql.getNumOccpts(self.flat_name_entry.get()) #gets the number of occupants from
sql database
        doc=Document()#creates an instance of document module using docx
        pic=doc.add_picture('logo.png',width=shared.Cm(9.04),height=shared.Cm(5.69)) #adds picture
to the document
        lastpara=doc.paragraphs[-1]#puts the cursor at the last paragraph (in this case will be the
pic)
        lastpara.alignment=WD_ALIGN_PARAGRAPH.CENTER #center aligns the picture

        lease_declaration_paragraph=doc.add_paragraph(f"This is an agreement to sublet real property
        according to the terms specified below, \
        hereinafter known as the 'Agreement'. The Sublessor, known as Adrian Akinola agrees to
        sublet to Subtenant, known as ")
        lease_declaration_paragraph.add_run(self.name_entry.get()[1:])
        location_para=doc.add_paragraph("The location of the premises is located at ")

        for i in location_list: #prints out the whole location on the document
```

```
            location_para.add_run(f'\n {i}')


        duration=doc.add_paragraph('Lease Duration: ')
        duration.add_run('\n'+str(self.duration_entry.get()))
        duration.add_run(' months')
        duration.add_run('\nStart Date: ')
        duration.add_run(self.start_date_entry.get())
        duration.add_run('\nEnd Date: ')
        duration.add_run(self.end_date_entry.get())

        rent=doc.add_paragraph('The Rent is £')
        rent.add_run(str(bdbsql.getRentPrice(self.flat_name_entry.get())))
        rent.add_run(' to be paid on the ')
        rent.add_run(self.rent_due_date_entry.get())
        rent.add_run('.')

        num_of_people=doc.add_paragraph('No. Of Occupants  (including Subtenant): ')
        num_of_people.add_run(str(occpts))

        doc.save(f'Lease Agreement for {self.name_entry.get()} at
{self.flat_name_entry.get()}.docx')#saves the document

    def delete(self):


        self.deleteData('Bookings','idb')#deletes the row selected

    def autoFillFlatName(self):

        try:
            self.flat_name_entry['values']=''
            self.flat_name_entry.set('')
            available_flats=bdbsql.getAvailableFlats()

            for i in available_flats:
                self.flat_name_entry['values']=(*self.flat_name_entry['values'],i)

        except:
            pass

    def getDateString(self,date): #turns the date into a string

        mstr=(date.strftime('%m'))
        dstr=(date.strftime('%d'))
        ystr=(date.strftime('%Y'))
        datestr =dstr+'/'+mstr+'/'+ystr

        return datestr

    def stringDateToDatetime(self,date): #converts date string into a datetime variable to be
manipulated
        datestring=date
        datestring=datestring.split('/')
        date_as_time=datetime.datetime.strptime(datestring[2]+' '+datestring[1]+'
'+datestring[0],'%Y %m %d')

        return date_as_time

    def autoFillEndDate(self,date):
        try:
            self.end_date_entry['values']='' #sets combobox to empty
            self.end_date_entry.set('') #sets the entry in combobox to empty
            months=datetime.timedelta(days=28) # creates a value that can be used to add and
subtract to a date 28 days can be added to a date
```

```
            date_as_time=self.stringDateToDatetime(date) #convets date to datetime variable
            for i in range(int(self.duration_entry.get())):
                date_as_time+=months #adds 28 days depending on how many months

            date_as_time=self.getDateString(date_as_time) #converts it back to string
            self.end_date_entry['values']=date_as_time #puts it on combo box
        except:
            mbx.showerror('Error','Incorrect format for start date or duration!')

    def getPayDate(self,start_date): #creates a pay date for the rent
        ten_days=datetime.timedelta(days=10)
        five_days=datetime.timedelta(days=5)
        paydate=int((start_date + ten_days).strftime('%d'))
        if paydate >28:#if the start date plus 10 days is greater than the 28th of a month
            paydate=start_date+five_days
        else:
            paydate=start_date+ten_days





        return paydate

    def autoFillPayDate(self):
        try:
            self.rent_due_date_entry['values']='' #emptys the combobox
            self.rent_due_date_entry.set('') #clears current text on combobox

            date=self.stringDateToDatetime(self.start_date_entry.get()) #converts date from string
to datetime
            date=self.getPayDate(date) #retrieves the pay date
            date=date.strftime('%d') #gets the day from the date sting
            date=self.getEndSentence(date) #creates a sentence that is used to describe pay date (if
day is 24 then will create sentence 24th of each month)
            self.rent_due_date_entry['values']=(*self.rent_due_date_entry['values'],date)#puts the
whole sentence on the combobox list


        except Exception as err:
            print(err)

    def getEndSentence(self,day):
        import math
        day=int(day)
        ordinal = lambda n: "%d%s" % (n,"tsnrhtdd"[(math.floor(n//10)%10!=1)*(n%10<4)*n%10::4])
#gets an ordinal number for a day eg 24 returns 24th, 1 returns 1st
        day=ordinal(day)
        sentence = day +' of each month' ### add the ending month

        return sentence

    def openCustomerWindow(self):

        cr.Customer(self.win)

    def splitName(self,full_name):
        name=full_name.split(' ')
        first_name,surname=name[0],name[1]

        return first_name,surname

    def autoFillNames(self):
        try:
            self.name_entry['values']=''
```

```
            self.name_entry.set('')
            names=cdbsql.getCustomerName()


            for i in names:

                self.name_entry['values']=(*self.name_entry['values'],i[0])

        except Exception as err:
            print(err)




    def addCustBooking(self):
        cust_id=self.name_entry.get()[0]#need a way to get new id wiht same name
        cdbsql.addCustomerBooking(cust_id,self.flat_name_entry.get())#composite key


    def updateBookingDatabase(self):
        valid,empty=self.bookingValid(True)


        if valid and not empty:
            self.delete()
            self.addBookingsData()
        #updates data by inserting entry values into the the sql database



    def displayTable(self,table):######*
        self.display_box.delete(0,tk.END)#empties the list box
        for i in bdbsql.viewData():#calls sql function that returns all data values as an array
            self.display_box.insert(tk.END,i) #the value returned from the sql function is then
printed on the display box#
```

File:customerwin

```
import tkinter as tk #imports the tkinter module which allow gui programming
from tkinter import ttk
from editdatbwin import EditDatabase
import datetime
from tkinter import scrolledtext as st
import customersql as cdbsql
import re
from tkinter import messagebox as mbx #imports a module that allows the use of pop up message boxes


class Customer(EditDatabase): #inherits from editdatabase
    def __init__(self,win):
        self.customer_window=tk.Toplevel(win)#creates a instance of the tkinter module in the form
of a new window
        self.customer_window.geometry('900x370')#sets a default size for the window when opened
        self.customer_window.resizable(0,0)
        self.createStyle()
        self.createWidgets()

self.array=[self.first_name_entry,self.surname_entry,self.birth_date_entry,self.phone_number_entry,s
elf.email_entry]#creates a list that holds all entrys

    def createFrame(self):

        main_frame =ttk.LabelFrame(self.customer_window,text ='GreenBird Properties
Bookings')#creaetes a main frame that holds all frames
        main_frame.pack(anchor ='w',fill='both')


        #self.data_frame=ttk.Frame(main_frame,style ='DF.TLabelframe')#creates a frame that holdd
the data widgets and frames
        #self.data_frame.pack(side=tk.TOP,pady=6)
        self.data_display_frame=ttk.LabelFrame(main_frame,text='Customers:',style
='DEF.TLabelframe')#creates a labeled frame that will hold all displayed data
        self.data_display_frame.pack(side=tk.RIGHT,anchor ='nw')

        self.data_entry_frame=ttk.LabelFrame(main_frame,text ='Customer Details:',style
='DEF.TLabelframe')#creates a labeled frame that will hold all data entries
        self.data_entry_frame.pack(side=tk.TOP,anchor='nw')

        self.button_frame=ttk.Frame(main_frame)#creates a button frame that will hold all buttons in
the mainframe
        self.button_frame.pack(side=tk.LEFT)

    def createLabels(self):

        #creates a label
        first_name_lbl=ttk.Label(self.data_entry_frame,text='First Name: ',style ='DEF.TLabel')
        first_name_lbl.grid(row=0,column=0)

        surname_lbl=ttk.Label(self.data_entry_frame,text='Surname:',style ='DEF.TLabel')
        surname_lbl.grid(row=1,column=0)

        birth_date_lbl=ttk.Label(self.data_entry_frame,text='Date Of Birth (dd/mm/yyyy):',style
='DEF.TLabel')
        birth_date_lbl.grid(row=2,column=0)

        phone_number_lbl=ttk.Label(self.data_entry_frame,text='Phone Number:',style ='DEF.TLabel')
        phone_number_lbl.grid(row=3,column=0)


        email_lbl=ttk.Label(self.data_entry_frame,text='Email: ',style ='DEF.TLabel')
        email_lbl.grid(row=4,column=0)
```

```
    def createEntrys(self):

        first_name=tk.StringVar()# declares variables types
        surname=tk.StringVar()
        birth_date=tk.StringVar()
        phone_number=tk.StringVar()
        email=tk.StringVar()



        #creates entries

        self.first_name_entry=ttk.Entry(self.data_entry_frame,textvariable =first_name,width=25,font
=('calibri',15))
        self.first_name_entry.grid(row=0,column=1)


        self.surname_entry=ttk.Entry(self.data_entry_frame,textvariable =surname,width=25,font
=('calibri',15),)
        self.surname_entry.grid(row=1,column=1)

        self.birth_date_entry=ttk.Entry(self.data_entry_frame,textvariable =birth_date,width=25,font
=('calibri',15))
        self.birth_date_entry.grid(row=2,column=1)

        self.phone_number_entry=ttk.Entry(self.data_entry_frame,textvariable
=phone_number,width=25,font =('calibri',15))
        self.phone_number_entry.grid(row=3,column=1)


        self.email_entry=ttk.Entry(self.data_entry_frame,textvariable =email,width=25,font
=('calibri',15))
        self.email_entry.grid(row=4,column=1)

    def createButtons(self):

        #creates click buttons
        add_datab=ttk.Button(self.button_frame,text='Add Customer',command =self.addCustomerData)
        add_datab.grid(row=0,column=0)

        search_datab=ttk.Button(self.button_frame,text='Search
Customer',command=self.searchCustomerDatabase)
        search_datab.grid(row=0,column=1)

        display_datab=ttk.Button(self.button_frame,text='Display
Customers',command=lambda:self.displayTable('Customer'))
        display_datab.grid(row=1,column=0)


clear_datab=ttk.Button(self.button_frame,text='Clear',command=lambda:self.clearData(self.array))
        clear_datab.grid(row=1,column=1)

        delete_datab=ttk.Button(self.button_frame,text='Delete',command =self.delete)
        delete_datab.grid(row=2,column=0)

        update_datab=ttk.Button(self.button_frame,text='Update',command
=self.updateCustomerDatabase)
        update_datab.grid(row=2,column=1)



exitb=ttk.Button(self.button_frame,text='Exit',command=lambda:self._exit(self.customer_window))
        exitb.grid(row=3,column=0)
```

```
        for i in self.button_frame.winfo_children():
            i.configure(style= 'B.TButton')

    def createWidgets(self):

        self.createFrame()
        self.createLabels()
        self.createEntrys()
        self.createListBox(self.data_display_frame,35,10)
        self.listboxCommand()
        self.createButtons()

        for i in self.data_entry_frame.winfo_children():
            i.grid_configure(pady=5,sticky='w')

    def listboxCommand(self):


        self.display_box.bind('<<ListboxSelect>>',lambda event
:self.recordDisplayIndex(event,self.array))

    def addCustomerData(self):

        valid,empty=self.customerValid()

        if valid and not empty:


cdbsql.addCustomerData(self.first_name_entry.get().upper(),self.surname_entry.get().upper(),self.bir
th_date_entry.get(),\
                self.phone_number_entry.get(),self.email_entry.get().upper()) #adds data from entrys
and puts in sql database
            self.displayTable('Customer')

    def delete(self):



cdbsql.deleteBookingWithCustomer(self.first_name_entry.get(),self.surname_entry.get(),self.sd[0])
        self.deleteData('Customer','idc')#deletes the row selected


    def validEmailCheck(self):
        email=self.email_entry.get()
        if not re.match(r"^[A-Za-z0-9\.\+_-]+@[A-Za-z0-9\._-]+\.[a-zA-Z]+$", email):
            return False
        else:

            return True

    def validNameCheck(self):
        first_name=self.first_name_entry.get()
        surname=self.surname_entry.get()

        if not re.match(r"[A-Za-z]+$",first_name):
            return False
        elif not re.match(r"[A-Za-z]+$",surname):
            return False
        else:
            return True

    def validateDate(self,date_text):
        try:
            date_text=date_text.split('/')
            date=date_text[0]+'-'+date_text[1]+'-'+date_text[2]
```

```
            datetime.datetime.strptime(date, '%d-%m-%Y')
            return True
        except:

            return False

    def validPhonenumber(self):
        number=self.phone_number_entry.get()
        if not re.match(r"[0][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]",number):
            return False
        else:

            return True



    def customerValid(self):
        valid=False
        empty=True

        empty=self.emptyEntryCheck(5)
        if empty:
            mbx.showerror('Error','Fill out all remaining fields!')
            return valid,empty

        if self.validNameCheck():
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for first name or surname \nname should have no
special characters or white space!')
            return valid,empty

        if self.validateDate(self.birth_date_entry.get()):
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for birth date should be DD/MM/YYYY!')
            return valid,empty

        if self.validPhonenumber():
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for phone number!')
            return valid,empty

        if self.validEmailCheck():
            valid=True
        else:
            valid=False
            mbx.showerror('Error','Incorrect format for email!')
            return valid,empty


        return valid,empty

    def updateCustomerDatabase(self):
        #updates data by inserting entry values into the the sql database


cdbsql.updateCustomerData(self.sd[0],self.first_name_entry.get().upper(),self.surname_entry.get().up
per(),\

self.birth_date_entry.get(),self.phone_number_entry.get(),self.email_entry.get().upper())
```

```
        self.displayTable('Customer')


    def getSqlCommand(self):

        sql_string=''
        sql_variables=[]

        if self.first_name_entry.get().isspace() or self.first_name_entry.get()==='':
            pass
        else:
            sql_string+='first_name=? '
            sql_variables.append(self.first_name_entry.get().upper())

        if self.surname_entry.get().isspace() or self.surname_entry.get()==='':
            pass
        else:
            if sql_string==='':
                sql_string+='surname=? '
            else:
                sql_string+='AND surname=? '

            sql_variables.append(self.surname_entry.get().upper())

        if self.birth_date_entry.get().isspace() or self.birth_date_entry.get()==='':
            pass
        else:
            if sql_string==='':
                sql_string+='birth_date=? '
            else:
                sql_string+='AND birth_date=? '
            sql_variables.append(self.birth_date_entry.get())


        if self.phone_number_entry.get().isspace() or self.phone_number_entry.get()==='':
            pass
        else:
            if sql_string==='':
                sql_string+='phone_number=? '
            else:
                sql_string+='AND phone_number=? '
            sql_variables.append(self.phone_number_entry.get())

        if self.email_entry.get().isspace() or self.email_entry.get()==='':
            pass
        else:
            if sql_string==='':
                sql_string+='email=? '
            else:
                sql_string+='AND email=? '
            sql_variables.append(self.email_entry.get().upper())


        return sql_string,sql_variables

    def searchCustomerDatabase(self):
        self.display_box.delete(0,tk.END)
        #uses entry values as search criteria

        sql_string,sql_variables=self.getSqlCommand()

        for i in cdbsql.searchCustomerData(sql_string,sql_variables):
            self.display_box.insert(tk.END,i) #inserts different data into display box based on
index
```

File:checkstatuswin.py

```
import tkinter as tk #imports the tkinter module which allow gui programming
from tkinter import ttk
import customerbooked as cb


class CheckStatus():
    def __init__(self,win):
        self.check_status_window=tk.Toplevel(win)#creates a instance of the tkinter module in the
form of a new window
        self.check_status_window.geometry('500x500')#sets a default size for the window when opened
        self.createWidgets(self.check_status_window)


    def createWidgets(self,win):
        cb.CustomersBooked(win)
```

File:customerbooked.py

```
import tkinter as tk #imports the tkinter module which allow gui programming
from tkinter import ttk
from tkinter import scrolledtext as st

from customerinfowin import CustomerInfo
from bookcustomerswin import BookCustomers
from editdatbwin import EditDatabase
import customersql as cdbsql
import datetime

class CustomersBooked(BookCustomers):#inherits from book customers

    def __init__(self,win):
        self.win=win
        self.createStyle()
        self.createWidgets(win)
        self.insertNameIntoListbox()

    def createWidgets(self,win):
        self.list_box_frame=ttk.LabelFrame(win,text='Flats in Use',style='MF.TLabelframe')
        self.list_box_frame.pack()
        self.createListBox(self.list_box_frame,100,12)#creates list box)
        self.listboxCommand()



    def listboxCommand(self):
        self.display_box.bind('<<ListboxSelect>>',\
            lambda event:self.SelectCustomer(event))#performs the command when data is selected on
the listbox



    def insertNameIntoListbox(self): #inserts the newly created array with descriptive strings into
the list box
        name=cdbsql.getCustomerName()
        for i in range(len(name)):
            for j in range(len(name[i])):
                self.display_box.insert(tk.END,name[i][j])

            self.display_box.insert(tk.END,'')


    def SelectCustomer(self,event):#*
        global sd#stands for searched data
        try:
            search=self.display_box.curselection()[0] #returns an index that is used to specify the
location of the data on the listbox line i.e first line will be 0

            sd =self.display_box.get(search) #returns a list of data displayed on the list box
            name=sd.split(' ') #splits the clicked name from the display box
            CustomerInfo(self.win,name[0],name[1],name[2]) #customer info wil take in id, first name
and surname


        except Exception as err:
            print(err)

        return
```

File:customerinfowin.py

```
import tkinter as tk #imports the tkinter module which allow gui programming
from tkinter import ttk
from tkinter import scrolledtext as st
from editdatbwin import EditDatabase
import customersql as cdbsql
import bookingsdatabasesql as bdbsql
import propertydatabasesql as pdbsql

class CustomerInfo(EditDatabase):
    def __init__(self,win,id,first_name,surname):
        self.win=win#so that win can be used in other function
        self.id,self.first_name,self.surname=id,first_name,surname
        self.customer_info_window=tk.Toplevel(win)#creates a instance of the tkinter module in the
form of a new window
        self.customer_info_window.geometry('500x500')#sets a default size for the window when opened
        self.customer_info_window.resizable(0,0)
        self.createWidgets()
        self.insertCustInfo()


    def tupleToArray(self,table):
        new_table=[]
        for i in range(len(table)):###changes the 2d list format  from a list with tuples to a list
with arrays which then allows it to be edidted eg from [(x,y)] to [[x,y]]
            t=[]
            for j in range(len(table[i])):
                t.append(table[i][j])
            new_table.append(t)
        return new_table


    def createWidgets(self):

        name=self.first_name+' '+self.surname


self.list_box_frame=ttk.LabelFrame(self.customer_info_window,text=name,style='MF.TLabelframe')
        self.list_box_frame.pack()
        self.createListBox(self.list_box_frame,100,12)#creates list box)
        self.listboxCommand()


    def listboxCommand(self):
        self.display_box.bind('<<ListboxSelect>>',\
            lambda event:self.SelectProperty(event))#performs the command when data is selected on
the listbox


    def getCustInfo(self):
        cust_flats,cust_data=cdbsql.getCustomerData(self.id,self.first_name,self.surname)

        cust_flats=self.tupleToArray(cust_flats)
        cust_data=self.tupleToArray(cust_data)

        for i in range(len(cust_data)):
            cust_data[i][0]= 'Birth Date: '+cust_data[i][0]
            cust_data[i][1]= 'Phone Number: '+cust_data[i][1]
            cust_data[i][2]= 'Email: '+cust_data[i][2]

        return cust_flats,cust_data

    def insert(self,table):
```

```
        for i in range(len(table)):
            for j in range(len(table[i])):

                self.display_box.insert(tk.END,table[i][j])
        self.display_box.insert(tk.END,'')

    def insertCustInfo(self):

        cust_flats,cust_data=self.getCustInfo()
        cust_data[0].insert(0,'Customer info:')
        cust_flats.insert(0,['Flats Leased:'])

        self.insert(cust_flats)
        self.insert(cust_data)

    def SelectProperty(self,event):#*
        global sd#stands for searched data
        try:
            search=self.display_box.curselection()[0] #returns an index that is used to specify the
location of the data on the listbox line i.e first line will be 0

            sd =self.display_box.get(search) #returns a list of data displayed on the list box
            flat_name=sd
            flats=pdbsql.getAllFlats()
            if flat_name in flats:
                PropertyInfo(self.win,flat_name)


        except Exception as err:
            print(err)

        return



class PropertyInfo(CustomerInfo):
    def __init__(self,win,flat_name):
        self.flat_name=flat_name
        self.property_info_window=tk.Toplevel(win)#creates a instance of the tkinter module in the
form of a new window
        self.property_info_window.geometry('500x500')#sets a default size for the window when opened
        self.property_info_window.resizable(0,0)
        self.createWidgets()
        self.insertPropertyInfo()

    def createWidgets(self):



self.list_box_frame=ttk.LabelFrame(self.property_info_window,text=self.flat_name,style='MF.TLabelfra
me')
        self.list_box_frame.pack()
        self.createListBox(self.list_box_frame,100,12)#creates list box)


    def getFlatInfo(self):
        flats=pdbsql.getFlatInfo(self.flat_name)
        flats=self.tupleToArray(flats)

        for i in range(len(flats)):
            flats[i][1]= 'Flat Name: '+str(flats[i][1])
            flats[i][2]= 'Flat Num: '+str(flats[i][2])
            flats[i][3]= 'Post Code: '+str(flats[i][3])
            flats[i][4]= 'Town: '+str(flats[i][4])
            flats[i][5]= 'City: '+str(flats[i][5])
```

```
            flats[i][6]= 'Bathroom(s): '+str(flats[i][6])
            flats[i][7]= 'Bedroom(s): '+str(flats[i][7])
            flats[i][8]= 'Renting Price: £'+str(flats[i][8])



        return flats


    def insertPropertyInfo(self):
        flats=self.getFlatInfo()
        flats[0][0]='Property info:'
        self.insert(flats)
```

File:propertydatabasesql.py

```
import sqlite3


def createPropertyTable():#function to create the data base and its values
    con=sqlite3.connect('property.db')#connects to the property database or creates it if doesnt
exist

    cursor =con.cursor() #allows for database manipulation
    cursor.execute("PRAGMA foreign_keys = ON")

    #creates a table called property if doesnt exist and puts these values into the table
    cursor.execute(f'CREATE TABLE IF NOT EXISTS Property(\
            idp INTEGER PRIMARY KEY AUTOINCREMENT,\
            flat_name text UNIQUE,\
            flat_num integer,\
            post_code text,\
            town text,\
            city text,\
            bathrooms text,\
            bedrooms text,\
            rent_price integer)')
    con.commit()#adds the table and fields to the database
    con.close()#closes the database



def addPropertyData(flat_name,flat_num,post_code,town,city,bathrooms,bedrooms,rent_price):#function
to add the data entered into the data base
    con=sqlite3.connect('property.db')#connects to property database
    cursor =con.cursor()#initiates a cursor from sql which allows access and manipulation of data in
sql table and rows
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute(f'INSERT INTO Property
(flat_name,flat_num,post_code,town,city,bathrooms,bedrooms,rent_price) VALUES (?,?,?,?,?,?,?,?)',\
        (flat_name,flat_num,post_code,town,city,bathrooms,bedrooms,rent_price)) #adds each variable
to the fields in the database
    con.commit() #adds the data to the database



def viewData(table):
    con=sqlite3.connect('property.db')#connects to property database
    cursor =con.cursor()#initiates a cursor from sql which allows access and manipulation of data in
sql table and rows
    cursor.execute("PRAGMA foreign_keys = ON")
    cursor.execute(f'SELECT * FROM {table}') #the * means all and so this selects all rows from the
table property
    row=cursor.fetchall()#fetches and stores the row
    con.close()
    return row #the rows are returened to be used in the gui

def deleteData(id,table,sqlid):
    try:
        con=sqlite3.connect('property.db')
        cursor =con.cursor()
        cursor.execute("PRAGMA foreign_keys = ON")
        cursor.execute(f'DELETE FROM {table} WHERE {sqlid}=?',(id,))#a parameter id is passed into
the function to specify the data row to be deleted based on the primary key
        con.commit()
        con.close()
    except:
        pass
```

```
    return

def searchPropertyData(string,variable):
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")
    try:
        cursor.execute(f'SELECT * FROM Property WHERE {string}',\
            (variable)) #this looks for all rows in the table that are equal to the data

    except:
        pass

    #returns values that meet this criteria
    row=cursor.fetchall()
    con.close()


    return row

def updatePropertyData(id,flat_name,flat_num,post_code,town,city,bathrooms,bedrooms,rent_price):
    con=sqlite3.connect('property.db') #updates the row
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('UPDATE Property SET flat_name=?,flat_num=?,post_code=?,town=?,\
    city=?,bathrooms=?,bedrooms=?,rent_price=? WHERE idp=?',\
        (flat_name,flat_num,post_code,town,city,bathrooms,bedrooms,rent_price,id)) #updates the
fields using these data variables

    con.commit()
    con.close()


def getLocation(flat_name):
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets the location information from property
    cursor.execute('SELECT flat_name,flat_num,post_code,town,city FROM Property WHERE
flat_name=?',(flat_name,))
    location=cursor.fetchall()
    location=location[0]
    con.close()

    return location

def getFlatInfo(flat_name):


    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets the location information from property
    cursor.execute('SELECT * FROM Property WHERE flat_name=?',(flat_name,))
    flat_info=cursor.fetchall()

    return flat_info

def getAllFlats():
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
```

```
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets the flat name and status from the status table
    cursor.execute('SELECT flat_name FROM Property')
    flats=cursor.fetchall()
    con.close()

    flat_list=[]
    for i in flats:
        flat_list.append(i[0])

    return flat_list
```

File:bookingsdatabasesql

```
import sqlite3

import datetime


def createBookingTable():  #function to create the data base and its values
    con=sqlite3.connect('property.db') #connects to the property database or creates it if doesnt
exist
    cursor=con.cursor()#alloes for database manipulation
    cursor.execute("PRAGMA foreign_keys = ON")


    #creates a table called bookings if doesnt exist and then creates these fields into the table
    cursor.execute('CREATE TABLE IF NOT EXISTS Bookings(\
            idb integer PRIMARY KEY AUTOINCREMENT,\
            duration integer,\
            start_date text,\
            end_date text,\
            num_of_people integer,\
            flat_name text UNIQUE,\
            rent_due_date text,\
            FOREIGN KEY (flat_name) REFERENCES Property(flat_name) ON DELETE CASCADE ON UPDATE
CASCADE)')

    con.commit() #adds the table and fields to the database
    con.close() #closes the database


def addBookingsData(duration='',start_date='',end_date='',num_of_people='',\
    flat_name='',rent_due_date=''):#function to add the data entered into the data base
    con=sqlite3.connect('property.db')#connects to property database
    cursor =con.cursor()#initiates a cursor from sql which allows access and manipulation of data in
sql table and rows
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('INSERT INTO Bookings (duration,start_date,end_date,num_of_people,\
    flat_name,rent_due_date) VALUES (?,?,?,?,?,?)',(\
        duration,start_date,end_date,num_of_people,flat_name,rent_due_date))
    con.commit() #adds the data to the database

    con.close() ##closes the database

def searchBookingsData(string,variables):
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")
    #add for names
    try:
        cursor.execute(f'SELECT Bookings.*,Customer.first_name,Customer.surname FROM Bookings JOIN
Customer \
        JOIN CustomerBooking ON Bookings.flat_name=CustomerBooking.cust_flat AND
Customer.idc=CustomerBooking.cust_id WHERE {string}',\
            variables) #this looks for all rows in the table that are equal to the data

    except:
        pass
    #Create a string thaat generates sentence

    bookings=cursor.fetchall()#fetches and stores the row

    bookings_list=[]
    for i in range(len(bookings)):
        books=[]
```

```
        for j in range(len(bookings[i])):
            books.append(bookings[i][j])

        first_name=books.pop(-2)
        surname=books.pop(-1)
        name=first_name+' '+surname
        books.insert(1,name)
        bookings_list.append(books)

    con.close()

    return bookings_list #the rows are returened to be used in the gui



def updateBookingsData(id,duration,start_date,num_of_people,flat_name,rent_due_date):
    con=sqlite3.connect('property.db') #updates the row
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('UPDATE Bookings SET duration=?,start_date=?,num_of_people=?,\
    flat_name=?,rent_due_date=?,phone_number=?,email=? WHERE idb=?',\
        (duration,start_date,num_of_people,flat_name,rent_due_date,phone_number,email,id)) #updates
the fields using these data variables

    con.commit()
    con.close()

def getNumOccpts(flat_name):
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets the number of occupants
    cursor.execute('SELECT num_of_people FROM Bookings WHERE flat_name=?',(flat_name,))
    occpts=cursor.fetchall()
    occpts=occpts[0][0]
    con.close()

    return occpts

def getFlatName():
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets the flat name and status from the status table
    cursor.execute('SELECT flat_name,status FROM Property')
    flat=cursor.fetchall()
    con.close()

    return flat

def getAvailableFlats():
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('SELECT flat_name FROM Bookings')
    booked_flats=cursor.fetchall()

    cursor.execute('SELECT flat_name FROM Property')
    all_flats=cursor.fetchall()

    con.close()
```

```
    booked_flats_list=[]
    for i in booked_flats:
        booked_flats_list.append(i[0])

    all_flats_list=[]
    for i in all_flats:
        all_flats_list.append(i[0])

    available_flats=[]

    for i in range(len(all_flats_list)):
        if all_flats_list[i] not in booked_flats_list:

            available_flats.append(all_flats_list[i])


    return available_flats

def getRentPrice(flat_name): #gets the rent price based on flat name
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")
    cursor.execute('SELECT rent_price FROM Property WHERE flat_name=?',(flat_name,))
    price=cursor.fetchall()
    price=price[0][0]
    con.close()

    return price

def getAllFlats():
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets all flat names of those that have been booked
    cursor.execute('SELECT flat_name FROM Bookings')
    flats=cursor.fetchall()
    con.close()

    flat_list=[]
    for i in flats:
        flat_list.append(i[0])

    return flat_list


def viewData():
    con=sqlite3.connect('property.db')#connects to property database
    cursor =con.cursor()#initiates a cursor from sql which allows access and manipulation of data in
sql table and rows
    cursor.execute("PRAGMA foreign_keys = ON")
    cursor.execute(f'SELECT Bookings.*,Customer.first_name,Customer.surname FROM Bookings JOIN
Customer \
    JOIN CustomerBooking ON Bookings.flat_name=CustomerBooking.cust_flat AND
Customer.idc=CustomerBooking.cust_id') #the * means all and so this selects all rows from the table
property
    bookings=cursor.fetchall()#fetches and stores the row


    bookings_list=[]
    for i in range(len(bookings)):
        books=[]
```

```
        for j in range(len(bookings[i])):
            books.append(bookings[i][j])

        first_name=books.pop(-2)
        surname=books.pop(-1)
        name=first_name+' '+surname
        books.insert(1,name)
        bookings_list.append(books)

    con.close()

    return bookings_list #the rows are returened to be used in the gui
```

File:customersql

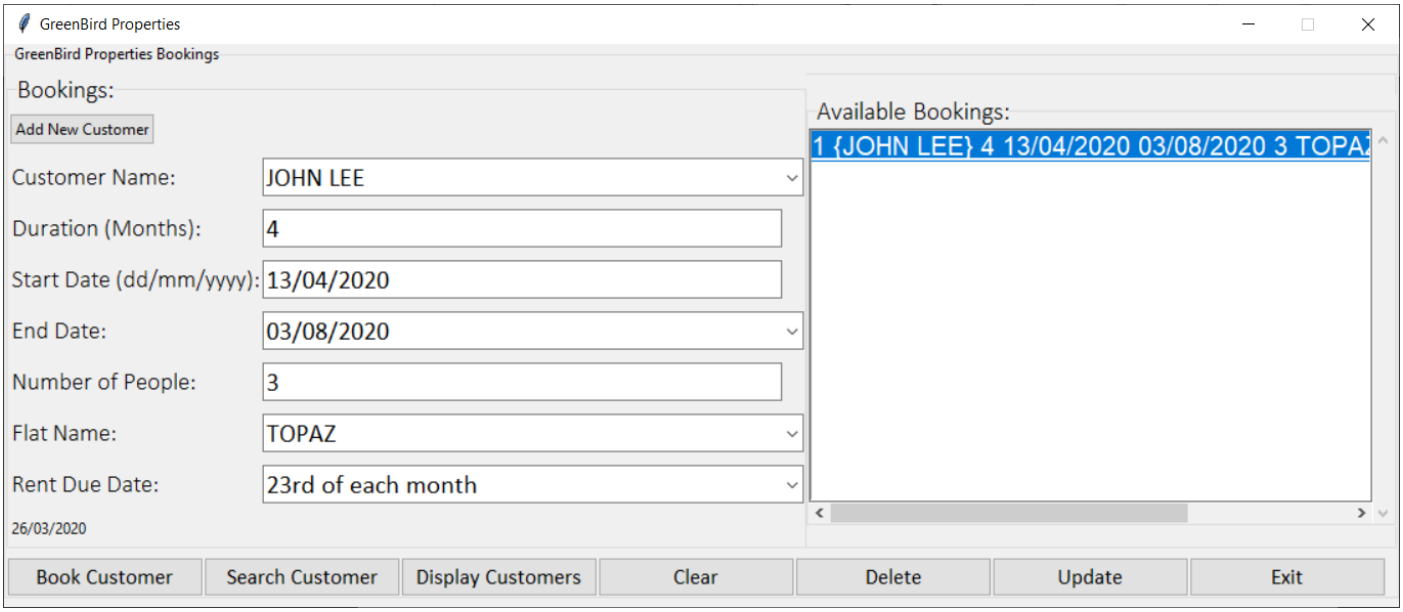
```
import sqlite3


def createCustomerTable():  #function to create the data base and its values
    con=sqlite3.connect('property.db') #connects to the property database or creates it if doesnt
exist
    cursor=con.cursor()#alloes for database manipulation
    cursor.execute("PRAGMA foreign_keys = ON")


    #creates a table called bookings if doesnt exist and then creates these fields into the table
    cursor.execute('CREATE TABLE IF NOT EXISTS Customer(\
            idc integer PRIMARY KEY AUTOINCREMENT,\
            first_name text,\
            surname text,\
            birth_date text,\
            phone_number text,\
            email text)')

    con.commit() #adds the table and fields to the database
    con.close() #closes the database

def addCustomerData(first_name='',surname='',birth_date='',phone_number='',email=''):#function to
add the data entered into the data base
    con=sqlite3.connect('property.db')#connects to property database
    cursor =con.cursor()#initiates a cursor from sql which allows access and manipulation of data in
sql table and rows
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('INSERT INTO Customer (first_name,surname,birth_date,phone_number,email) VALUES
(?,?,?,?,?)',(\
        first_name,surname,birth_date,phone_number,email))
    con.commit() #adds the data to the database

    con.close() ##closes the database

def createCustomerBookingTable():
    con=sqlite3.connect('property.db') #connects to the property database or creates it if doesnt
exist
    cursor=con.cursor()#alloes for database manipulation
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('CREATE TABLE IF NOT EXISTS CustomerBooking(\
            cust_id integer NOT NULL,\
            cust_flat text NOT NULL,\
            FOREIGN KEY (cust_id) REFERENCES Customer(idc) ON DELETE CASCADE,\
            FOREIGN KEY (cust_flat) REFERENCES Bookings(flat_name) ON DELETE CASCADE ON UPDATE
CASCADE,\
            PRIMARY KEY (cust_id,cust_flat) )')

    con.commit()
    con.close()

def addCustomerBooking(cust_id,cust_flat):
    con=sqlite3.connect('property.db')#connects to property database
    cursor =con.cursor()#initiates a cursor from sql which allows access and manipulation of data in
sql table and rows
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('INSERT INTO CustomerBooking (cust_id,cust_flat) VALUES
(?,?)',(cust_id,cust_flat))
    con.commit()
    con.close()
```

```
def getCustomerName():

    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets the flat name and status from the status table
    cursor.execute(f'SELECT idc,first_name,surname FROM Customer')
    name_list=cursor.fetchall()
    names=[]
    for i in name_list:
        name=[]
        name.append(str(i[0])+' '+i[1] +' '+i[2])

        names.append(name)


    con.close()
    return names

def getCustomerID(first_name,surname):
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    #gets the flat name and status from the status table
    cursor.execute('SELECT idc FROM Customer WHERE first_name=? AND
surname=?',(first_name,surname,))
    cust_id=cursor.fetchall()
    cust_id=cust_id[0][0]

    con.close()
    return cust_id

def getCustomerData(id,first_name,surname):
    con=sqlite3.connect('property.db')#connects to property database
    cursor =con.cursor()#initiates a cursor from sql which allows access and manipulation of data in
sql table and rows
    cursor.execute("PRAGMA foreign_keys = ON")
    cursor.execute('SELECT Bookings.flat_name FROM (Property INNER JOIN Bookings ON
Property.flat_name = Bookings.flat_name) \
    INNER JOIN (Customer INNER JOIN CustomerBooking ON Customer.idc = CustomerBooking.cust_id) ON \
    Bookings.flat_name = CustomerBooking.cust_flat WHERE Customer.idc=? AND Customer.first_name=?
AND \
    Customer.surname=?',(id,first_name,surname,))

    cust_flats=cursor.fetchall()

    cursor.execute('SELECT birth_date,phone_number,email FROM Customer WHERE first_name=? AND
surname=?',(first_name,surname,))
    cust_data=cursor.fetchall()
    con.close()
    return cust_flats,cust_data

def deleteBookingWithCustomer(first_name,surname,id):

    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")
    cursor.execute('DELETE FROM Bookings WHERE EXISTS (SELECT CustomerBooking.cust_flat\
    FROM Property INNER JOIN (Customer INNER JOIN CustomerBooking ON Customer.idc =
CustomerBooking.cust_id) ON \
    Property.flat_name = CustomerBooking.cust_flat\
```

```
    WHERE Customer.first_name=? AND Customer.surname=? AND Customer.idc=? AND
Bookings.flat_name=Property.flat_name)',(first_name,surname,id))

    con.commit()
    con.close()


def updateCustomerData(id,first_name,surname,birth_date,phone_number,email):
    con=sqlite3.connect('property.db') #updates the row
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")

    cursor.execute('UPDATE Customer SET first_name=?,surname=?,birth_date=?,phone_number=?,email=?
WHERE idc=?',\
        (first_name,surname,birth_date,phone_number,email,id)) #updates the fields using these data
variables

    con.commit()
    con.close()


def searchCustomerData(string,variable):
    con=sqlite3.connect('property.db')
    cursor =con.cursor()
    cursor.execute("PRAGMA foreign_keys = ON")
    try:
        cursor.execute(f'SELECT * FROM Customer WHERE {string}',\
            (variable)) #this looks for all rows in the table that are equal to the data

    except:
        pass

    #returns values that meet this criteria
    row=cursor.fetchall()
    con.close()


    return row
```

Testing

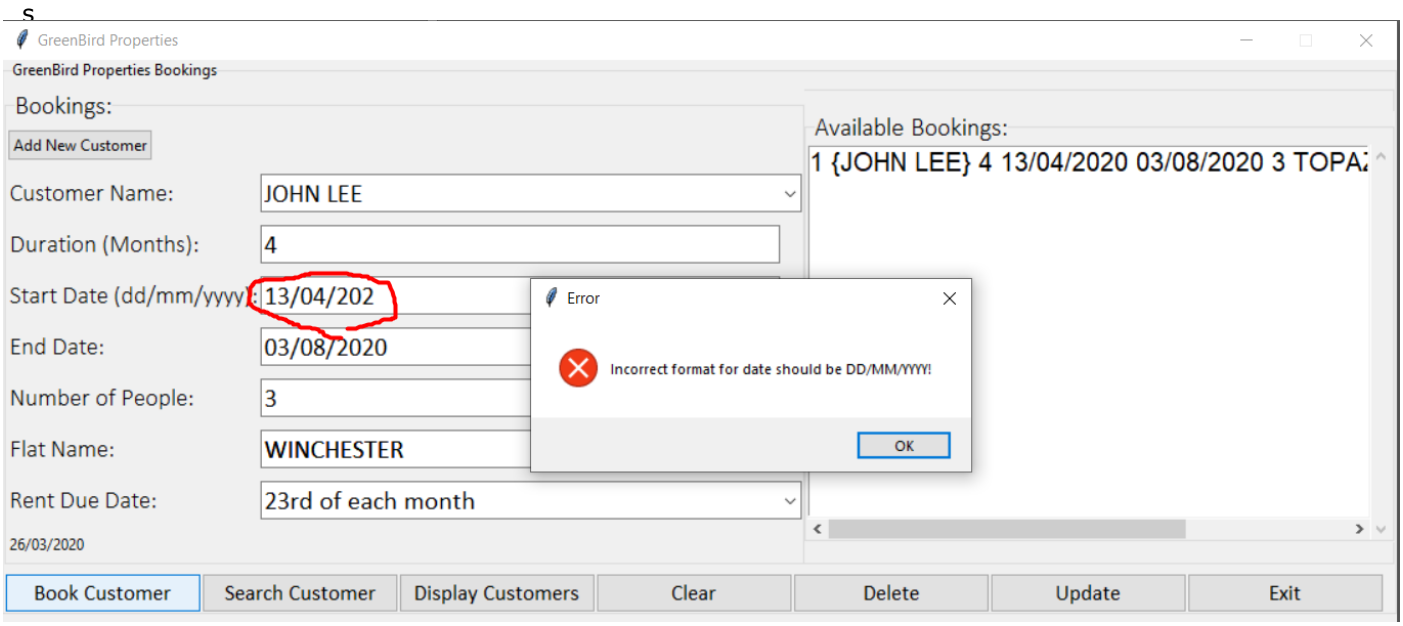
1. Data selected from a listbox automatically inputs value into entry widgets

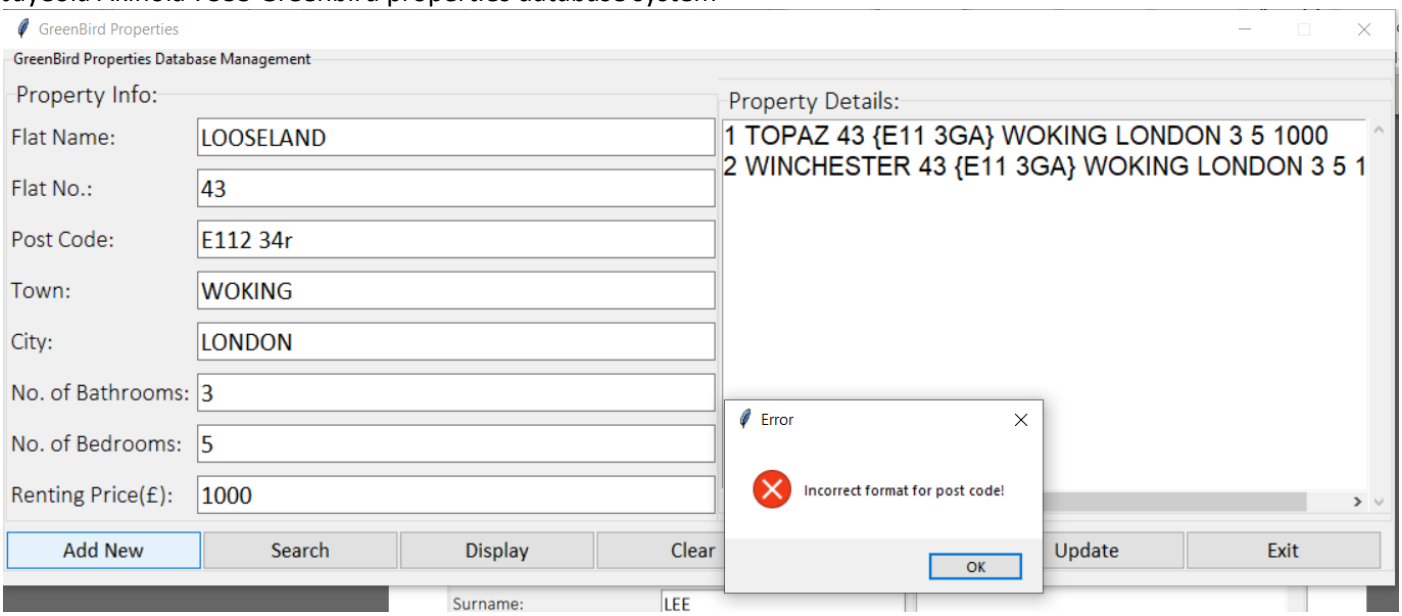


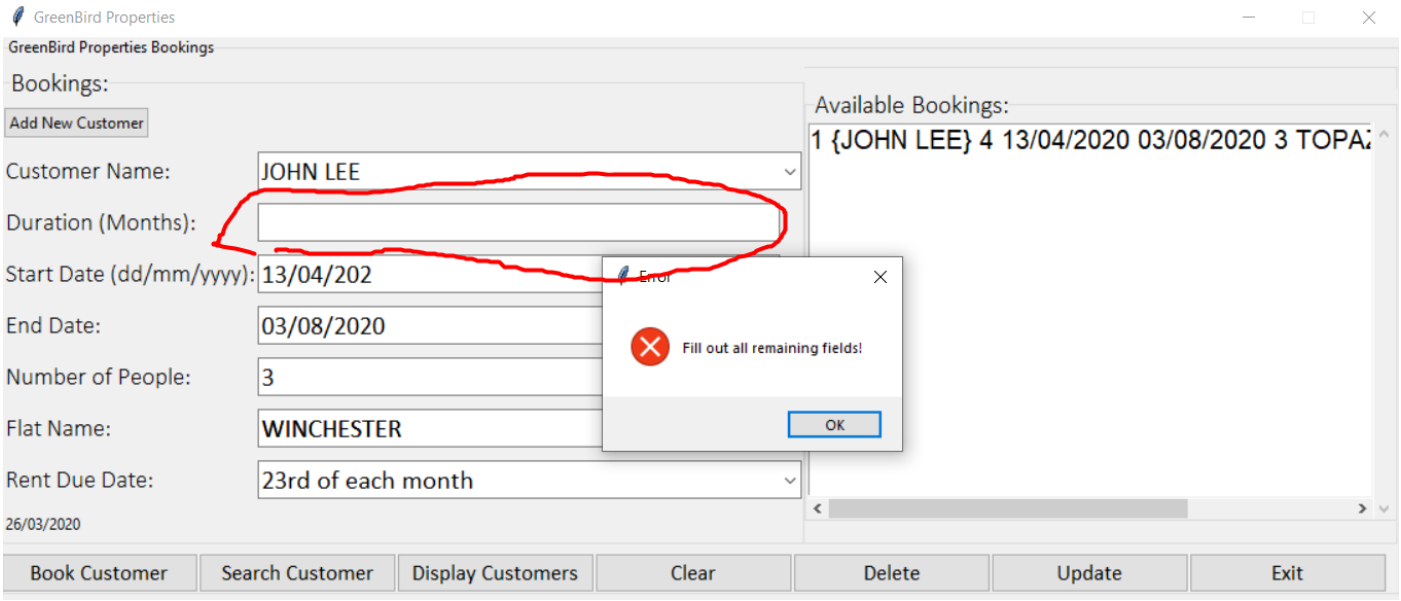
After selecting john lee from the display box, all data values relating to johns booking are inputted into the display box

2. Test for inputting invalid data

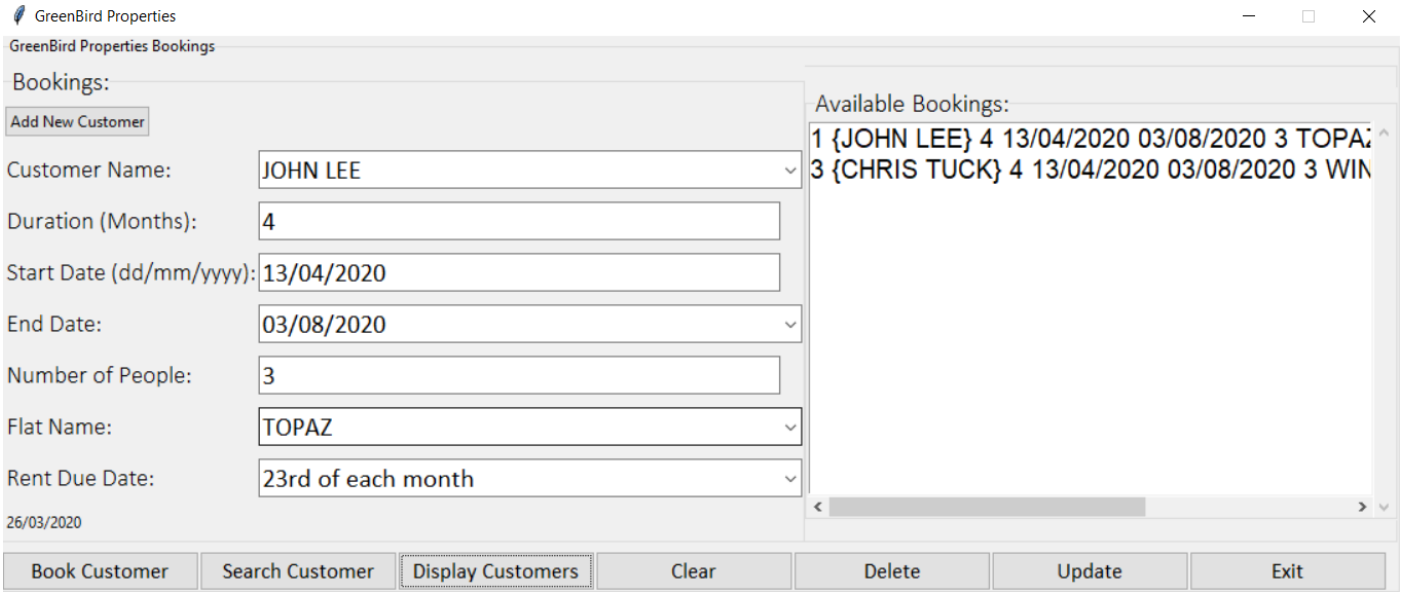
s

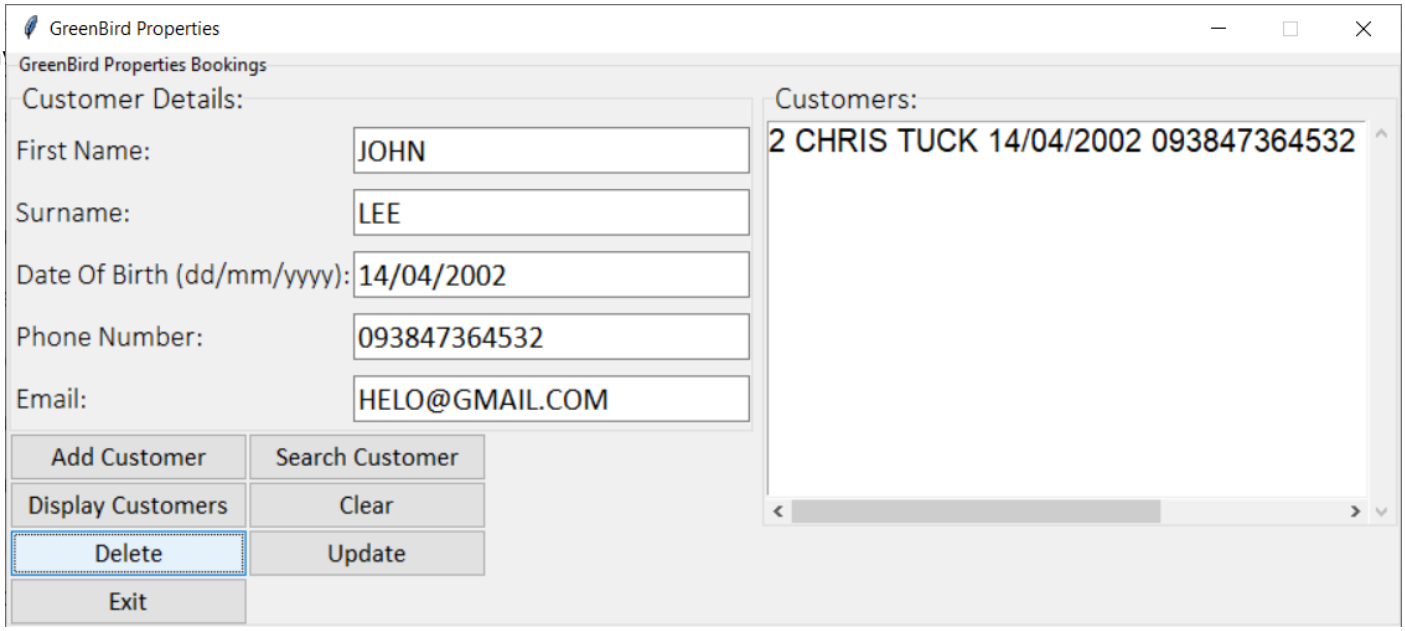


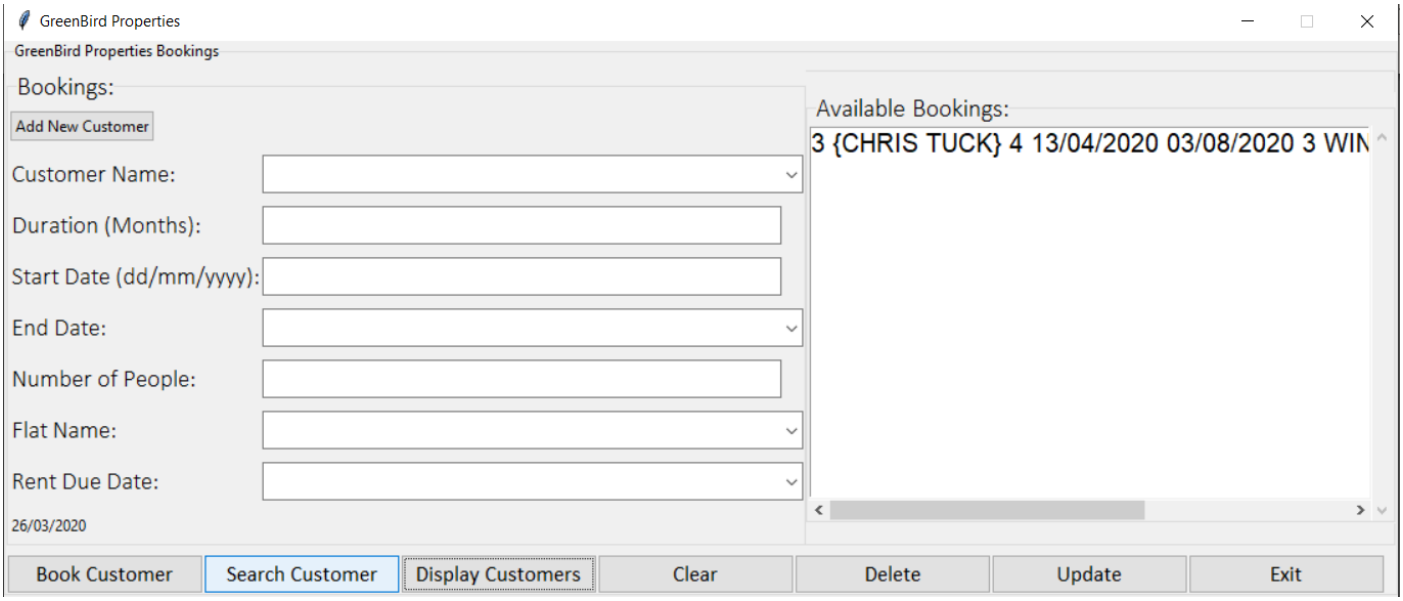




3. Test for maintaining data integrity

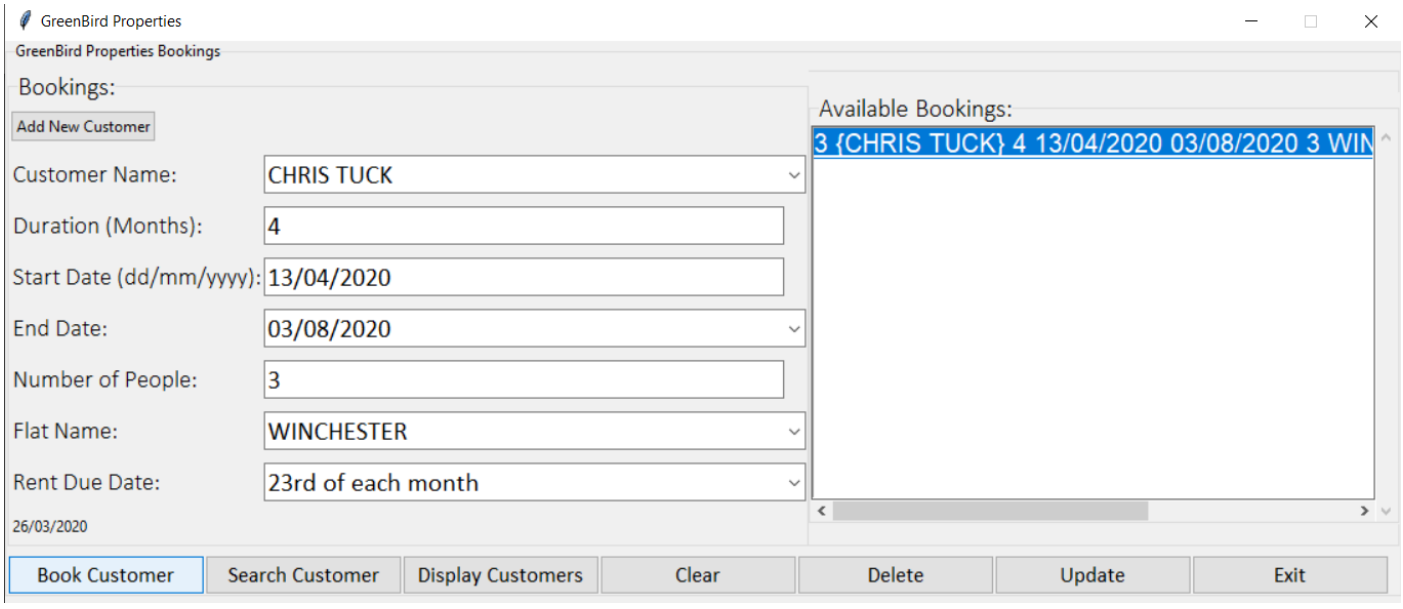






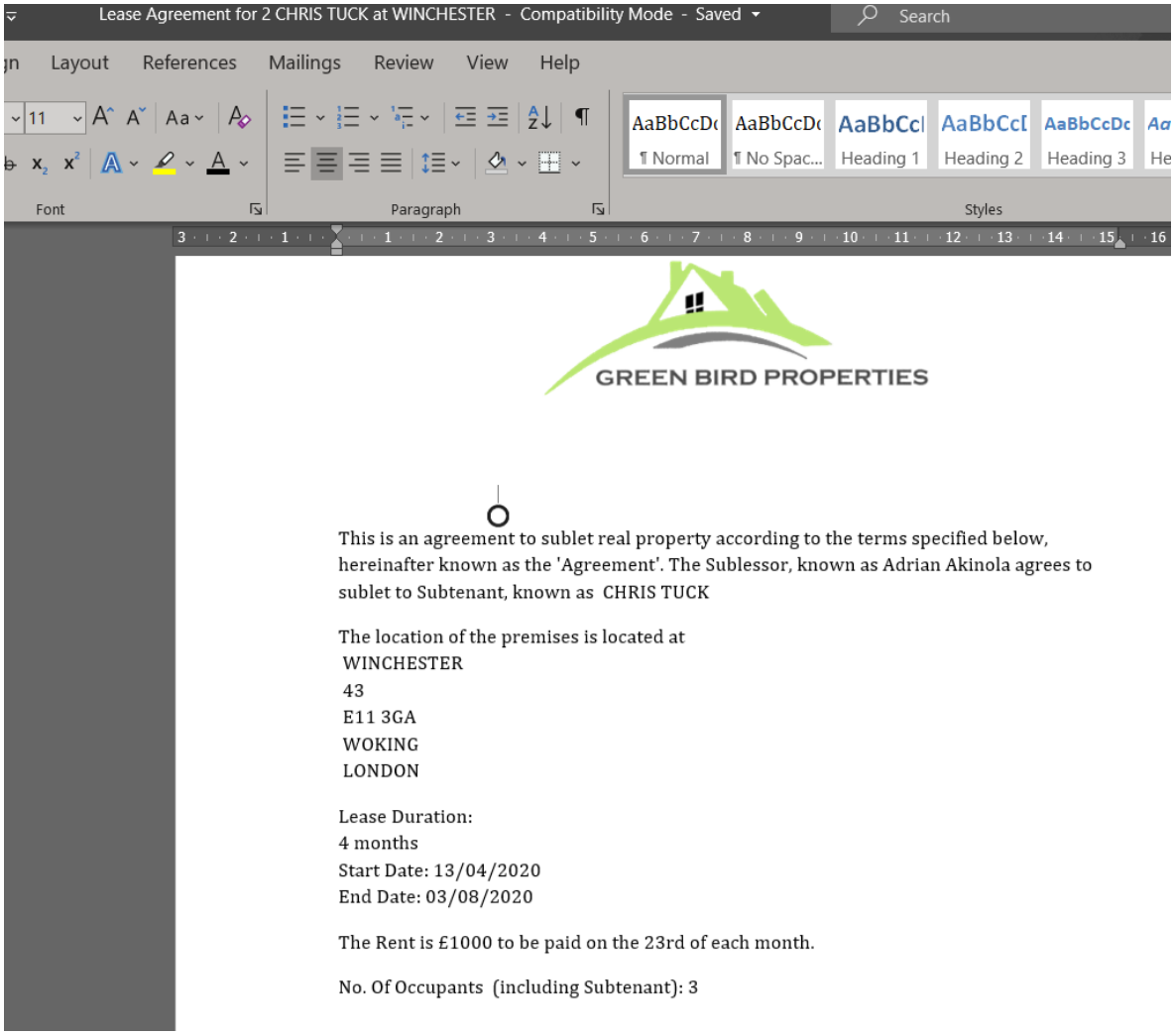
In the first image bothe chris tuck and john lee had bookings. The second image shows deleting of customer john lee (removing him from the customer table). Data integrity is maintained as when going back to customer booking, the apartment booked by that customer is now gone

4. Document creation

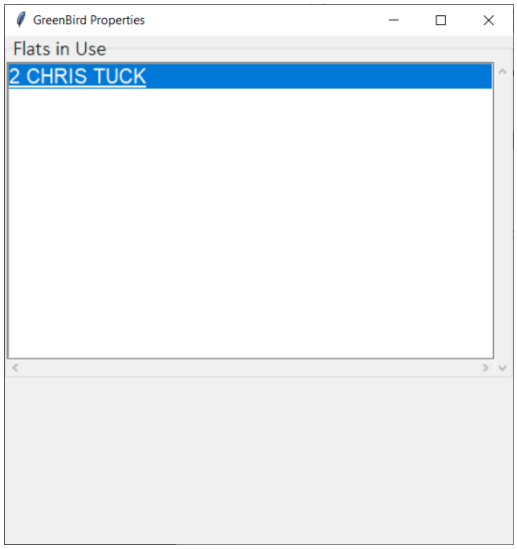


bookcustomerswin	25/03/2020 18:58	PY File
bookingsdatabasesql	25/03/2020 20:14	PY File
checkstatuswin	25/03/2020 19:00	PY File
customerbooked	25/03/2020 20:26	PY File
customerinfowin	25/03/2020 19:01	PY File
customersql	26/03/2020 13:39	PY File
customerwin	26/03/2020 09:38	PY File
editdatbwin	26/03/2020 08:56	PY File
flatsnotinuse	05/03/2020 15:33	PY File
flatsnotinusesql	19/02/2020 16:41	PY File
GreenBirdProject	18/03/2020 11:50	PY File
GreenBirdProject	04/08/2019 13:26	Python Project
GreenBirdProject.pyproj.user	01/11/2019 12:49	Per-User Project O...
GreenBirdProject.spec	30/08/2019 16:59	SPEC File
Lease Agreement for 1 JOHN LEE at TOPAZ	12/03/2020 14:09	Microsoft Word D...
Lease Agreement for 1 JOHN LEE at WINCHESTER	26/03/2020 13:40	Microsoft Word D...
Lease Agreement for 2 CHRIS TUCK at WINCHESTER	26/03/2020 13:40	Microsoft Word D...
logo	26/09/2019 15:10	PNG File
property	26/03/2020 13:42	DB File
propertydatabasesql	10/03/2020 15:36	PY File

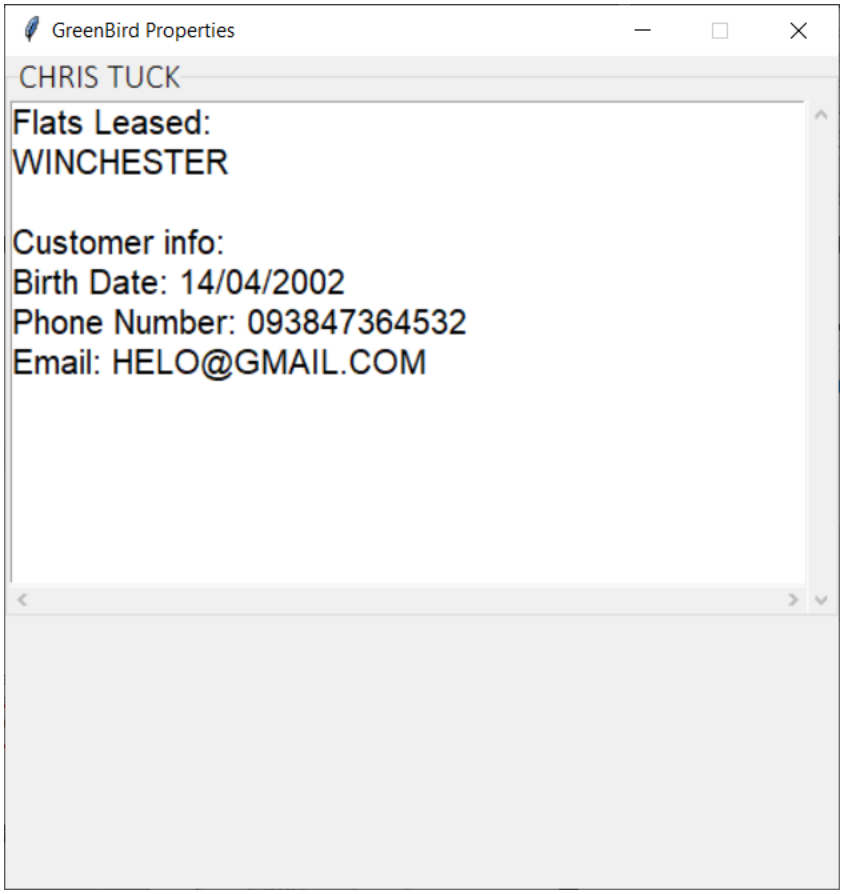
ss



5. Test for customer info display



After clicking customer name



Evaluation

Evaluation of system as a whole:

Overall the database management system has been successful in solving the problems with the current system, without creating significant new problems. The GUI program provides a simplistic design and thus easily understandable by third party end users. Furthermore complications with the program are minimised due to numerous validation checks, use of comboboxes that only provide valid data, and so even with its simplistic design, new users that try to use it cant make any mistakes that will ruin the integrity of the database, this also makes it easier for those users to understand how the software works. The program does well in mainting the data integrity of the relations and so no errors occur when adding, updating and deleting data. The program accomplishes in making work easier for the manager, it has in built functions such as the calculate end date and rent price which removes human error when recording these calculated values. However even though it makes work easier it potentially may act as a limitation if the manager has specific values for data that should be recorded but can't be because the program calculates its own values. An improvement would be to make the auto calculation an option instead thus its utility is determined by the manager. Furthermore when bookng customers the program works on the basis that customers only stay for a given number of months, this may not be the case but instead a certain amount of weeks. This could be improved by adding a weeks entry is well and adjust any necessary calculation. The program does well to use the booking information to create a document for the customer and manager. This makes work easier for the manager as no agreements need to be typed. However an improvement to this is to incorporate sending the email to the given customers.

How well are requirements met

Objective	Evaluation
1. The program must have a simplistic design in so that it is easily understandable by third party users such as the manager and potential subordinates. This can be proven by a positive response from the manager.	I feel that objective 1 has been met very well, as seen from screen shots in previous sections the program has a simplistic design that doesn't require prior knowledge of its function to understand. The use of large sized widgets(buttons , labels entrys) makes it easy on the eye and so all functions can be analysed by a user quicker and easier. However given that any new user to this program has no knowledge of its function, there is no guarantee that because the design is simplistic and widgets are eye catching, that it can be grasped quickly. An improvement would be to have notes that pop up when hovering over widgets that act as a form of help. Or a help function that displays what each button does
2.Program must have a menu	The program has an efficient menu that allows the user to click between three different windows. The menu has been implemented well with each click button stating exactly what it opens. The check status click button could be named something else that describes better what the button opens. Instead of check status it could be check customer informations. Check status as mentioned before is a redundant class and so its poor choice of name is understandable. Before the window was used to hold tabs that would say whether a flat was occupied or not. This got changed due to the introduction of the customer class where the customer is now an entity on its own that can have many flats and so a way of seeing whether a flat is occupied is through the customer that made the bookings

3.Program must connect to SQL and consequently make SQL tables	The program successfully connects to SQL using sqlite and thus allows the making of tables for the database. An improvement could be to use mysql instead of sqlite that way a form of login for the database can be used as a bette form of protection
3.1.Database tables must be easily manipulated	The programs three main data manipulation functions namely add data update data and delete are successfully implemented in the program. The user can easily click a record from the display box and apply a fuction or type in their own data and apply a function. What is very good about data maniputlation in the program is that after a function is used, the result is shown on the display table and so a user can easily keep track with the data their manipulating
4.When inserting, updating and deleting data, data integrity must be maintained	Data integrity is fully maintained within the system. This is due to the use of a normalised relations data base and so all data in each different table is coherent with each other when updating and deleting and inserting . This is also as a result of a successful prevention of invalid data being entered into the database. This as well as its simplistic design is what makes the program easy to use. When entering invalid data the use sees an error message which informs the user of how to enter the correct data. This means that all data in the database is correct and even someone with knowledge of the program can ruin the integrity of its Example: when entering numeric data into the database such as rent price, it is not possible ti enter a letter only numeric digits are allowed.
5.Program must have a function to display customer booking information	Using the display box when opening the check status window the customers in the database pop up. The program allows the user to easily select a customer which will then result in a pop window displaying the customers information, such as the flats booked, email etc. Furthermore by clicking on the flat booked another window opens up showing the description of given flat. This makes gives a better view to look up customer and flat information. However the check status window shows all customers even those who don't have apartments booked but are in the database. If amount of customers is a lot then it will be hard to select the customers that have apartments. An improvement would be to separate them into customers with and without apartments Moreover flat information can only be displayed by clicking on a customer that booked that flat. This is tedious and also not all flats can have their info displayed if not booked. An improvement would be to create a window for flats allowing them to be selected and thus show their info (objective 5.2)
6.Once a customer is booked into an apartment on the database, a document of the booking for the customer must be created	After booking in a customer the program successfully creates a document based on their information. This document comprises of the logo and the flat and customer information. This is very helpful for the manager using the program because it makes work much easier as there is no longer a need to type their own documents. The document created is good and comprises of the email of the customer however it doesn't send the email to the customer which means the manager will still have to do some for of work relating to the document. This can be improved by adding a function that sends it to the customer after clicking book customer
7.Database must be portable	The database is portable which is good meaning that it can be transported anywhere and back ups can be easily stored on external disks. However with improvements in technology, cloud storage has made portability less important and so the database may be more efficient if stored on a server rather than a computer

Independent feedback

The manager of greenbird properties took the program and had a test run during a business week. After the week she came back to me and was overall pleased. She was instantly able to understand the functions due to its simple design (objective 1) and so foud no trouble when adding data (3.1.1), deleting data (3.1.2), updating data 3.1.3 and searching for data (3.1.6). Furthermore she was extremely pleased with the end date calculation, she said that in the past errors have occurred when calculating how long a tenant stays. Moreover the document creation was the function she was most pleased with as it made the work load less and also potential spelling errors are now omitted

However an improvement she came across was the aforementioned sending the document to a customer via email. That is a function that she said would make the program more useful than it already is. Moreover another improvement is the automatic deleting of a booking once the booking is over in real time.

Evalution of feedback

The main improvement to the program is the adding of a method to send the email to customers after creating a document. This would be relatively easy to implement. The only difficulting being the implementation of the senders emai as my knowledge sending emails with python isn't too solid. Overall the feedback was positive and with minor improvents the program can become more efficient