import tkinter as tk

from tkinter import ttk, OptionMenu

import mysql.connector

from mysql.connector import Error

import json

from datetime import datetime

import smtplib

#pip install mysql

#pip3 install mysql-connector

#data list of customer

customer\_id\_list = []

customer\_name\_list = []

customer\_username\_list = []

customer\_pw\_list = []

customer\_gender\_list = []

customer\_phone\_list = []

customer\_email\_list = []

customer\_address\_list = []

customer\_createdDT\_list = []

#data list of provider

provider\_id\_list = []

provider\_name\_list = []

provider\_username\_list = []

provider\_pw\_list = []

provider\_intro\_list = []

provider\_gender\_list = []

provider\_phone\_list = []

provider\_email\_list = []

provider\_companyName\_list = []

provider\_companyPosition\_list = []

provider\_companyAddress\_list = []

provider\_status\_list = []

provider\_createdDT\_list = []

#data list of admin

admin\_id\_list = []

admin\_name\_list = []

admin\_username\_list = []

admin\_pw\_list = []

admin\_gender\_list = []

admin\_phone\_list = []

admin\_email\_list = []

admin\_createdDT\_list = []

#data list of cases

case\_id\_list = []

casefk\_customer\_id\_list = []

casefk\_provider\_id\_list = []

case\_category\_list = []

case\_description\_list = []

case\_address\_list = []

case\_district\_list = []

case\_area\_list = []

case\_price\_list = []

case\_status\_list = []

case\_createdDT\_list = []

#data list of cases\_application

casesApplyfk\_case\_id\_list = []

casesApplyfk\_provider\_id\_list = []

loginSuccess = False

def read\_mysql():

'''

read all mysql data into lists

'''

#clear data list of customer

customer\_id\_list.clear()

customer\_name\_list.clear()

customer\_username\_list.clear()

customer\_pw\_list.clear()

customer\_gender\_list.clear()

customer\_phone\_list.clear()

customer\_email\_list.clear()

customer\_address\_list.clear()

customer\_createdDT\_list.clear()

#clear data list of provider

provider\_id\_list.clear()

provider\_name\_list.clear()

provider\_username\_list.clear()

provider\_pw\_list.clear()

provider\_intro\_list.clear()

provider\_gender\_list.clear()

provider\_phone\_list.clear()

provider\_email\_list.clear()

provider\_companyName\_list.clear()

provider\_companyPosition\_list.clear()

provider\_companyAddress\_list.clear()

provider\_status\_list.clear()

provider\_createdDT\_list.clear()

#clear data list of admin

admin\_id\_list.clear()

admin\_name\_list.clear()

admin\_username\_list.clear()

admin\_pw\_list.clear()

admin\_gender\_list.clear()

admin\_phone\_list.clear()

admin\_email\_list.clear()

admin\_createdDT\_list.clear()

#clear data list of cases

case\_id\_list.clear()

casefk\_customer\_id\_list.clear()

casefk\_provider\_id\_list.clear()

case\_category\_list.clear()

case\_description\_list.clear()

case\_address\_list.clear()

case\_district\_list.clear()

case\_area\_list.clear()

case\_price\_list.clear()

case\_status\_list.clear()

case\_createdDT\_list.clear()

#clear data list of cases\_application

casesApplyfk\_case\_id\_list.clear()

casesApplyfk\_provider\_id\_list.clear()

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

sql = ("select \* from customer")

mycursor.execute(sql)

results = mycursor.fetchall()

for index in range(len(results)):

customer\_id\_list.append(results[index][0])

customer\_name\_list.append(results[index][1])

customer\_username\_list.append(results[index][2])

customer\_pw\_list.append(results[index][3])

customer\_gender\_list.append(results[index][4])

customer\_phone\_list.append(results[index][5])

customer\_email\_list.append(results[index][6])

customer\_address\_list.append(results[index][7])

customer\_createdDT\_list.append(results[index][8])

sql = ("select \* from provider")

mycursor.execute(sql)

results = mycursor.fetchall()

for index in range(len(results)):

provider\_id\_list.append(results[index][0])

provider\_name\_list.append(results[index][1])

provider\_username\_list.append(results[index][2])

provider\_pw\_list.append(results[index][3])

provider\_intro\_list.append(results[index][4])

provider\_gender\_list.append(results[index][5])

provider\_phone\_list.append(results[index][6])

provider\_email\_list.append(results[index][7])

provider\_companyName\_list.append(results[index][8])

provider\_companyPosition\_list.append(results[index][9])

provider\_companyAddress\_list.append(results[index][10])

provider\_status\_list.append(results[index][11])

provider\_createdDT\_list.append(results[index][12])

sql = ("select \* from admin")

mycursor.execute(sql)

results = mycursor.fetchall()

for index in range(len(results)):

admin\_id\_list.append(results[index][0])

admin\_name\_list.append(results[index][1])

admin\_username\_list.append(results[index][2])

admin\_pw\_list.append(results[index][3])

admin\_gender\_list.append(results[index][4])

admin\_phone\_list.append(results[index][5])

admin\_email\_list.append(results[index][6])

admin\_createdDT\_list.append(results[index][7])

sql = ("select \* from cases")

mycursor.execute(sql)

results = mycursor.fetchall()

for index in range(len(results)):

case\_id\_list.append(results[index][0])

casefk\_customer\_id\_list.append(results[index][1])

casefk\_provider\_id\_list.append(results[index][2])

case\_category\_list.append(results[index][3])

case\_description\_list.append(results[index][4])

case\_address\_list.append(results[index][5])

case\_district\_list.append(results[index][6])

case\_area\_list.append(results[index][7])

case\_price\_list.append(results[index][8])

case\_status\_list.append(results[index][9])

case\_createdDT\_list.append(results[index][10])

sql = ("select \* from cases\_application")

mycursor.execute(sql)

results = mycursor.fetchall()

for index in range(len(results)):

casesApplyfk\_case\_id\_list.append(results[index][1])

casesApplyfk\_provider\_id\_list.append(results[index][3])

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

class create\_dict(dict):

# \_\_init\_\_ function

def \_\_init\_\_(self):

self = dict()

# Function to add key:value

def add(self, key, value):

self[key] = value

class GuiOverlay:

def \_\_init\_\_(self, parent):

#set var

self.current\_userID = 0

self.current\_userType = ''

self.district0 = tk.BooleanVar()

self.district1 = tk.BooleanVar()

self.district2 = tk.BooleanVar()

self.district3 = tk.BooleanVar()

self.district4 = tk.BooleanVar()

self.district5 = tk.BooleanVar()

self.district6 = tk.BooleanVar()

self.district7 = tk.BooleanVar()

self.district8 = tk.BooleanVar()

self.district9 = tk.BooleanVar()

self.district10 = tk.BooleanVar()

self.district11 = tk.BooleanVar()

self.district12 = tk.BooleanVar()

self.district13 = tk.BooleanVar()

self.district14 = tk.BooleanVar()

self.district15 = tk.BooleanVar()

self.district16 = tk.BooleanVar()

self.district17 = tk.BooleanVar()

self.district0.set(True)

self.district1.set(True)

self.district2.set(True)

self.district3.set(True)

self.district4.set(True)

self.district5.set(True)

self.district6.set(True)

self.district7.set(True)

self.district8.set(True)

self.district9.set(True)

self.district10.set(True)

self.district11.set(True)

self.district12.set(True)

self.district13.set(True)

self.district14.set(True)

self.district15.set(True)

self.district16.set(True)

self.district17.set(True)

read\_mysql()

#formatting parent frame

parent.grid\_rowconfigure(1, weight = 1)

parent.grid\_columnconfigure(0, weight = 1)

#create main frames in parent frame

self.top\_frame = tk.Frame(parent, bg = 'cadetblue3', width = 450, height = 50, pady = 1)

center = tk.Frame(parent, bg = 'gray2', width = 50, height = 40, pady = 1)

btm\_frame = tk.Frame(parent, bg = 'white', width = 450, height = 45, pady = 1)

#display main frames in parent frame

self.top\_frame.grid(row = 0, sticky = 'ew')

center.grid(row = 1, sticky = 'nsew')

btm\_frame.grid(row = 2, sticky = 'ew')

#formatting top frame

self.top\_frame.grid\_rowconfigure(0, weight = 1)

self.top\_frame.grid\_columnconfigure(1, weight = 1)

#create widgets for top frame

bigTitle = tk.Label(self.top\_frame, text = 'ABC Interior Design & Decoration Agency', fg = 'white', bg = 'cadetblue3')

bigTitle.config(font=('Tw Cen MT', 13, 'bold'))

self.pageTitle = tk.Label(self.top\_frame)

#display widgets in top frame

myac = tk.Button(self.top\_frame, text="My Account", command=lambda: self.show\_page(self.plogin))

myac.grid(row = 0, column = 2)

bigTitle.grid(row = 0, column = 0, columnspan = 3)

#formatting center frame

center.grid\_rowconfigure(0, weight=1)

center.grid\_columnconfigure(1, weight=1)

#create center widgets

self.cl = tk.Frame(center, bg ='lightskyblue3', width = 90, height = 100)

self.cc = tk.Frame(center, bg = 'gray95', width = 100, height = 100)

cr = tk.Frame(center, bg ='lightskyblue3', width = 20, height = 100)

#display center widgets

self.cl.grid(row = 0, column = 0, sticky = 'nse')

self.cc.grid(row = 0, column = 1, sticky = 'nsew')

cr.grid(row = 0, column = 2, sticky = 'nsew')

#formatting bottom frame

btm\_frame.grid\_rowconfigure(0, weight = 1)

btm\_frame.grid\_columnconfigure(1, weight = 1)

#create widgets for bottom frame

exit = tk.Button(btm\_frame, text = 'Exit', width = 5, command = parent.destroy)

#display widgets in the bottom frame

exit.grid(row = 0, column = 2)

#build all frames

self.build\_cl()

self.build\_uploadCases()

self.build\_lookforCases()

self.build\_myAC()

self.build\_login()

self.build\_createAC()

self.pDetail = 0

#show home page

self.show\_page(self.pLOOK)

def add\_sep(self, r, c, widg):

'''

create empty label for formatting in grid

'''

tk.Label(widg).grid(row=r, column=c)

def check\_createAC(self):

'''

error checking for creating account

'''

#clear error messages

self.errorMsg0.destroy()

self.errorMsg1.destroy()

self.errorMsg2.destroy()

self.errorMsg3.destroy()

self.errorMsg4.destroy()

self.errorMsg5.destroy()

self.errorMsg6.destroy()

#clear account created label

self.createdLabel.destroy()

#set var

createACvalid = True

if self.position == 'Customer':

#check empty value

if not self.createACe0.get() or not self.createACe1.get() or not self.createACe2.get() or not self.createACe3.get() or not self.v0.get() or not self.createACe4.get():

self.errorMsg0 = tk.Label(self.pCreateCAC, text='Empty value!')

self.errorMsg0.grid(row=8, column=1,columnspan=3,sticky='w')

self.errorMsg0.config(fg='red')

createACvalid = False

#check name valid

nameValid = True

if self.createACe0.get():

for letter in self.createACe0.get():

if not letter.isalpha() and letter not in ' ':

nameValid = False

if len(self.createACe0.get()) > 30:

nameValid = False

if not nameValid:

self.errorMsg1 = tk.Label(self.pCreateCAC, text='Invalid name')

self.errorMsg1.grid(row=9, column=1,columnspan=3,sticky='w')

self.errorMsg1.config(fg='red')

createACvalid = False

#check username valid

usernameValid = True

if self.createACe1.get():

for letter in self.createACe1.get():

if len(self.createACe1.get()) < 6 or len(self.createACe1.get()) > 20 or (not letter.isalpha() and not letter.isdigit() and letter not in '\_'):

usernameValid = False

if not usernameValid:

self.errorMsg2 = tk.Label(self.pCreateCAC, text='Username should contain 6-20 characters, only in letters, numbers, underscore(\_)')

self.errorMsg2.grid(row=10,column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

#check username duplicate

for index in range(len(customer\_username\_list)):

if self.createACe1.get() == customer\_username\_list[index]:

self.errorMsg2 = tk.Label(self.pCreateCAC, text='Username exists')

self.errorMsg2.grid(row=10, column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

for index in range(len(provider\_username\_list)):

if self.createACe1.get() == provider\_username\_list[index]:

self.errorMsg2 = tk.Label(self.pCreateCAC, text='Username exists')

self.errorMsg2.grid(row=10, column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

for index in range(len(admin\_username\_list)):

if self.createACe1.get() == admin\_username\_list[index]:

self.errorMsg2 = tk.Label(self.pCreateCAC, text='Username exists')

self.errorMsg2.grid(row=10, column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

#check password valid

if self.createACe2.get():

if len(self.createACe2.get()) < 8 or len(self.createACe2.get()) > 20:

self.errorMsg3 = tk.Label(self.pCreateCAC, text='Password should contain 8-20 characters')

self.errorMsg3.grid(row=11, column=1,columnspan=3,sticky='w')

self.errorMsg3.config(fg='red')

createACvalid = False

#check phone valid

if self.createACe3.get():

if self.createACe3.get().isdigit():

if len(self.createACe3.get()) != 8:

self.errorMsg4 = tk.Label(self.pCreateCAC, text='Invalid phone no.')

self.errorMsg4.grid(row=12, column=1,columnspan=3,sticky='w')

self.errorMsg4.config(fg='red')

createACvalid = False

else:

#check phone duplicate

for index in range(len(customer\_phone\_list)):

if int(self.createACe3.get()) == customer\_phone\_list[index]:

self.errorMsg4 = tk.Label(self.pCreateCAC, text='Phone no. exists')

self.errorMsg4.grid(row=12, column=1,columnspan=3,sticky='w')

self.errorMsg4.config(fg='red')

createACvalid = False

else:

self.errorMsg4 = tk.Label(self.pCreateCAC, text='Invalid phone no.')

self.errorMsg4.grid(row=12, column=1,columnspan=3,sticky='w')

self.errorMsg4.config(fg='red')

createACvalid = False

#check email valid

if self.createACe4.get():

if '@'not in self.createACe4.get() or '.'not in self.createACe4.get() or len(self.createACe4.get()) > 40:

self.errorMsg5 = tk.Label(self.pCreateCAC, text='Invalid email')

self.errorMsg5.grid(row=13, column=1,columnspan=3,sticky='w')

self.errorMsg5.config(fg='red')

createACvalid = False

#check email duplicate

for index in range(len(customer\_email\_list)):

if self.createACe4.get() == customer\_email\_list[index]:

self.errorMsg5 = tk.Label(self.pCreateCAC, text='Email exists')

self.errorMsg5.grid(row=13, column=1,columnspan=3,sticky='w')

self.errorMsg5.config(fg='red')

createACvalid = False

elif self.position == 'Provider':

#check empty value

if not self.createACe0.get() or not self.createACe1.get() or not self.createACe2.get() or not self.v0.get() or not self.createACe4.get() or not self.createACe5.get():

self.errorMsg0 = tk.Label(self.pCreatePAC, text='Empty value!')

self.errorMsg0.grid(row=13, column=1,columnspan=3,sticky='w')

self.errorMsg0.config(fg='red')

createACvalid = False

#check name valid

nameValid = True

if self.createACe0.get():

for letter in self.createACe0.get():

if not letter.isalpha() and letter not in ' ':

nameValid = False

if len(self.createACe0.get()) > 30:

nameValid = False

if not nameValid:

self.errorMsg1 = tk.Label(self.pCreatePAC, text='Invalid name')

self.errorMsg1.grid(row=14, column=1,columnspan=3,sticky='w')

self.errorMsg1.config(fg='red')

createACvalid = False

#check username valid

usernameValid = True

if self.createACe1.get():

for letter in self.createACe1.get():

if len(self.createACe1.get()) < 6 or len(self.createACe1.get()) > 20 or (not letter.isalpha() and not letter.isdigit() and letter not in '\_'):

usernameValid = False

if not usernameValid:

self.errorMsg2 = tk.Label(self.pCreatePAC, text='Username should contain 6-20 characters, only in letters, numbers, underscore(\_)')

self.errorMsg2.grid(row=15, column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

#check username duplicate

for index in range(len(customer\_username\_list)):

if self.createACe1.get() == customer\_username\_list[index]:

self.errorMsg2 = tk.Label(self.pCreatePAC, text='Username exists')

self.errorMsg2.grid(row=15, column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

for index in range(len(provider\_username\_list)):

if self.createACe1.get() == provider\_username\_list[index]:

self.errorMsg2 = tk.Label(self.pCreatePAC, text='Username exists')

self.errorMsg2.grid(row=15, column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

for index in range(len(admin\_username\_list)):

if self.createACe1.get() == admin\_username\_list[index]:

self.errorMsg2 = tk.Label(self.pCreatePAC, text='Username exists')

self.errorMsg2.grid(row=15, column=1,columnspan=3,sticky='w')

self.errorMsg2.config(fg='red')

createACvalid = False

#check password valid

if self.createACe2.get():

if len(self.createACe2.get()) < 8 or len(self.createACe2.get()) > 20:

self.errorMsg3 = tk.Label(self.pCreatePAC, text='Password should contain 8-20 characters')

self.errorMsg3.grid(row=16, column=1,columnspan=3,sticky='w')

self.errorMsg3.config(fg='red')

createACvalid = False

#check phone valid

if self.createACe4.get():

if self.createACe4.get().isdigit():

if len(self.createACe4.get()) != 8:

self.errorMsg4 = tk.Label(self.pCreatePAC, text='Invalid phone no.')

self.errorMsg4.grid(row=17, column=1,columnspan=3,sticky='w')

self.errorMsg4.config(fg='red')

createACvalid = False

else:

#check phone duplicate

for index in range(len(provider\_phone\_list)):

if int(self.createACe4.get()) == provider\_phone\_list[index]:

self.errorMsg4 = tk.Label(self.pCreatePAC, text='Phone no. exists')

self.errorMsg4.grid(row=17, column=1,columnspan=3,sticky='w')

self.errorMsg4.config(fg='red')

createACvalid = False

else:

self.errorMsg4 = tk.Label(self.pCreatePAC, text='Invalid phone no.')

self.errorMsg4.grid(row=17, column=1,columnspan=3,sticky='w')

self.errorMsg4.config(fg='red')

createACvalid = False

#check email valid

if self.createACe5.get():

if '@'not in self.createACe5.get() or '.'not in self.createACe5.get() or len(self.createACe5.get()) > 40:

self.errorMsg5 = tk.Label(self.pCreatePAC, text='Invalid email')

self.errorMsg5.grid(row=18, column=1,columnspan=3,sticky='w')

self.errorMsg5.config(fg='red')

createACvalid = False

#check email duplicate

for index in range(len(provider\_email\_list)):

if self.createACe5.get() == provider\_email\_list[index]:

self.errorMsg5 = tk.Label(self.pCreatePAC, text='Email exists')

self.errorMsg5.grid(row=18, column=1,columnspan=3,sticky='w')

self.errorMsg5.config(fg='red')

createACvalid = False

#check company name, position, and address valid

if self.createACe6.get():

if not self.createACe7.get() or not self.createACe8.get():

self.errorMsg6 = tk.Label(self.pCreatePAC, text='You must enter your position and company address if you entered company name')

self.errorMsg6.grid(row=19, column=1,columnspan=3,sticky='w')

self.errorMsg6.config(fg='red')

createACvalid = False

else:

if self.createACe7.get() or self.createACe8.get():

self.errorMsg6 = tk.Label(self.pCreatePAC, text='You must enter your company name first')

self.errorMsg6.grid(row=19, column=1,columnspan=3,sticky='w')

self.errorMsg6.config(fg='red')

createACvalid = False

if createACvalid:

self.mysql\_createAC()

def mysql\_createAC(self):

'''

insert new account to mysql

'''

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

if self.position == 'Customer':

try:

#get user's gender

if self.v0.get() == 1:

gender = 'M'

else:

gender = 'F'

#get the datetime now

dt = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

mycursor = connection.cursor()

mycursor.execute("INSERT INTO customer(customer\_id, name, username, pw, gender, phone, email, address, created\_dt) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)", (len(customer\_id\_list)+1,self.createACe0.get(), self.createACe1.get(), self.createACe2.get(), gender, self.createACe3.get(), self.createACe4.get(), self.createACe5.get(), dt))

connection.commit()

#clear entries and radiobutton

self.createACe0.delete(0, 'end')

self.createACe1.delete(0, 'end')

self.createACe2.delete(0, 'end')

self.createACe3.delete(0, 'end')

self.createACe4.delete(0, 'end')

self.createACe5.delete(0, 'end')

self.v0.set(None)

#create and display label after sql successfully run

self.createdLabel = tk.Label(self.pCreateCAC, text = 'Account created')

self.createdLabel.grid(row = 20, column = 1, columnspan = 3, sticky = 'w')

read\_mysql()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

elif self.position == 'Provider':

try:

#get user's gender

if self.v0.get() == 1:

gender = 'M'

else:

gender = 'F'

#get the datetime now

dt = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

mycursor = connection.cursor()

mycursor.execute("INSERT INTO provider(provider\_id,name, username, pw, introduction, gender, phone, email, company\_name, company\_position, company\_address, status, created\_dt) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)", (len(provider\_id\_list)+1,self.createACe0.get(), self.createACe1.get(), self.createACe2.get(), self.createACe3.get(), gender, self.createACe4.get(), self.createACe5.get(), self.createACe6.get(), self.createACe7.get(), self.createACe8.get(), 'Verifying', dt))

connection.commit()

#clear entries and radiobutton

self.createACe0.delete(0, 'end')

self.createACe1.delete(0, 'end')

self.createACe2.delete(0, 'end')

self.createACe3.delete(0, 'end')

self.createACe4.delete(0, 'end')

self.createACe5.delete(0, 'end')

self.createACe6.delete(0, 'end')

self.createACe7.delete(0, 'end')

self.createACe8.delete(0, 'end')

self.v0.set(None)

#create and display label after inserting account

self.createdLabel = tk.Label(self.pCreatePAC, text = 'Account created, please wait for verification within 24 working hours to activate')

self.createdLabel.grid(row = 20, column = 1, columnspan = 3, sticky = 'w')

read\_mysql()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

def build\_createAC(self):

'''

build the frame to create account

'''

#create frame to create account

self.pCreateAC = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.pCreateAC.grid(row = 0, column = 0, sticky = 'nsew')

#set var

self.position = str()

#create widgets

Title = tk.Label(self.pCreateAC, text='Select your position')

b0 = tk.Button(self.pCreateAC, text='Customer', command = self.build\_createCAC)

b1 = tk.Button(self.pCreateAC, text='Provider', command = self.build\_createPAC)

#display widgets

Title.grid(row = 0, column = 0)

b0.grid(row = 1, column = 0)

b1.grid(row = 1, column = 1)

def build\_createCAC(self):

'''

build the frame to create customer account

'''

#create frame to create customer account

self.pCreateCAC = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.pCreateCAC.grid(row = 0, column = 0, sticky = 'nsew')

#formatting the frame

for e in range(10):

self.pCreateCAC.grid\_columnconfigure(e, weight = 1)

self.cc.grid\_columnconfigure(e, weight = 1)

#set var

self.position = 'Customer'

self.v0 = tk.IntVar()

#create error messages

self.errorMsg0 = tk.Label(self.pCreateCAC)

self.errorMsg1 = tk.Label(self.pCreateCAC)

self.errorMsg2 = tk.Label(self.pCreateCAC)

self.errorMsg3 = tk.Label(self.pCreateCAC)

self.errorMsg4 = tk.Label(self.pCreateCAC)

self.errorMsg5 = tk.Label(self.pCreateCAC)

self.errorMsg6 = tk.Label(self.pCreateCAC)

#create label to display when account created

self.createdLabel = tk.Label(self.pCreateCAC)

#create titles

l0 = tk.Label(self.pCreateCAC, text='Name\*')

l1 = tk.Label(self.pCreateCAC, text='Username\*')

l2 = tk.Label(self.pCreateCAC, text='Password\*')

l3 = tk.Label(self.pCreateCAC, text='Gender\*')

l4 = tk.Label(self.pCreateCAC, text='Phone\*')

l5 = tk.Label(self.pCreateCAC, text='Email\*')

l6 = tk.Label(self.pCreateCAC, text='Address')

#display titles

l0.grid(row = 0, column = 0)

l1.grid(row = 1, column = 0)

l2.grid(row = 2, column = 0)

l3.grid(row = 3, column = 0)

l4.grid(row = 4, column = 0)

l5.grid(row = 5, column = 0)

l6.grid(row = 6, column = 0)

#create entries and radiobuttons for input

self.createACe0 = tk.Entry(self.pCreateCAC)

self.createACe1 = tk.Entry(self.pCreateCAC)

self.createACe2 = tk.Entry(self.pCreateCAC, show='\*')

genderM = tk.Radiobutton(self.pCreateCAC, text='Male', var=self.v0, value=1)

genderF = tk.Radiobutton(self.pCreateCAC, text='Female', var=self.v0, value=2)

self.createACe3 = tk.Entry(self.pCreateCAC, width = 8)

self.createACe4 = tk.Entry(self.pCreateCAC, width = 25)

self.createACe5 = tk.Entry(self.pCreateCAC, width = 40)

#display entries and radiobuttons for input

self.createACe0.grid(row = 0, column = 1,columnspan=2, stick = 'w')

self.createACe1.grid(row = 1, column = 1,columnspan=2, stick = 'w')

self.createACe2.grid(row = 2, column = 1,columnspan=2, stick = 'w')

genderM.grid(row = 3, column = 1,columnspan=2, stick = 'w')

genderF.grid(row = 3, column = 2,columnspan=2, stick = 'w')

self.createACe3.grid(row = 4, column = 1,columnspan=2, stick = 'w')

self.createACe4.grid(row = 5, column = 1,columnspan=2, stick = 'w')

self.createACe5.grid(row = 6, column = 1,columnspan=2, stick = 'w')

#create and display button to submit

b0 = tk.Button(self.pCreateCAC, text='Create Account', command = self.check\_createAC)

b0.grid(row = 7, column = 2)

def build\_createPAC(self):

'''

build the frame to create provider account

'''

#create frame to create provider account

self.pCreatePAC = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.pCreatePAC.grid(row = 0, column = 0, sticky = 'nsew')

#formatting the frame

for e in range(10):

self.pCreatePAC.grid\_columnconfigure(e, weight = 1)

self.cc.grid\_columnconfigure(e, weight = 1)

#set var

self.position = 'Provider'

self.v0 = tk.IntVar()

#create error messages

self.errorMsg0 = tk.Label(self.pCreatePAC)

self.errorMsg1 = tk.Label(self.pCreatePAC)

self.errorMsg2 = tk.Label(self.pCreatePAC)

self.errorMsg3 = tk.Label(self.pCreatePAC)

self.errorMsg4 = tk.Label(self.pCreatePAC)

self.errorMsg5 = tk.Label(self.pCreatePAC)

self.errorMsg6 = tk.Label(self.pCreatePAC)

#create label to display when account created

self.createdLabel = tk.Label(self.pCreatePAC)

#create titles

l0 = tk.Label(self.pCreatePAC, text = 'Name\*')

l1 = tk.Label(self.pCreatePAC, text = 'Username\*')

l2 = tk.Label(self.pCreatePAC, text = 'Password\*')

l3 = tk.Label(self.pCreatePAC, text = 'Introduction')

l4 = tk.Label(self.pCreatePAC, text = 'Gender\*')

l5 = tk.Label(self.pCreatePAC, text = 'Phone\*')

l6 = tk.Label(self.pCreatePAC, text = 'Email\*')

l7 = tk.Label(self.pCreatePAC, text = 'Company name')

l8 = tk.Label(self.pCreatePAC, text = 'Company position')

l9 = tk.Label(self.pCreatePAC, text = 'Company address')

#display titles

l0.grid(row = 0, column = 0)

l1.grid(row = 1, column = 0)

l2.grid(row = 2, column = 0)

l3.grid(row = 3, column = 0)

l4.grid(row = 4, column = 0)

l5.grid(row = 5, column = 0)

l6.grid(row = 6, column = 0)

l7.grid(row = 7, column = 0)

l8.grid(row = 8, column = 0)

l9.grid(row = 9, column = 0)

#create entries and radiobuttons for input

self.createACe0 = tk.Entry(self.pCreatePAC)

self.createACe1 = tk.Entry(self.pCreatePAC)

self.createACe2 = tk.Entry(self.pCreatePAC, show='\*')

self.createACe3 = tk.Entry(self.pCreatePAC, width = 40)

genderM = tk.Radiobutton(self.pCreatePAC, text='Male', var=self.v0, value=1)

genderF = tk.Radiobutton(self.pCreatePAC, text='Female', var=self.v0, value=2)

self.createACe4 = tk.Entry(self.pCreatePAC, width = 8)

self.createACe5 = tk.Entry(self.pCreatePAC, width = 25)

self.createACe6 = tk.Entry(self.pCreatePAC, width = 30)

self.createACe7 = tk.Entry(self.pCreatePAC, width = 30)

self.createACe8 = tk.Entry(self.pCreatePAC, width = 40)

#display entries and radiobuttons for input

self.createACe0.grid(row = 0, column = 1,columnspan=2, stick = 'w')

self.createACe1.grid(row = 1, column = 1,columnspan=2, stick = 'w')

self.createACe2.grid(row = 2, column = 1,columnspan=2, stick = 'w')

self.createACe3.grid(row = 3, column = 1,columnspan=2, stick = 'w')

genderM.grid(row = 4, column = 1,columnspan=2, stick = 'w')

genderF.grid(row = 4, column = 2,columnspan=2, stick = 'w')

self.createACe4.grid(row = 5, column = 1,columnspan=2, stick = 'w')

self.createACe5.grid(row = 6, column = 1,columnspan=2, stick = 'w')

self.createACe6.grid(row = 7, column = 1,columnspan=2, stick = 'w')

self.createACe7.grid(row = 8, column = 1,columnspan=2, stick = 'w')

self.createACe8.grid(row = 9, column = 1,columnspan=2, stick = 'w')

#create and display button to submit

b0 = tk.Button(self.pCreatePAC, text='Create Account', command = self.check\_createAC)

b0.grid(row = 11, column = 2)

def logoutSuccess(self):

'''

commands after logout

'''

#set var

global loginSuccess

loginSuccess = False

#destroy logout button

self.logout.destroy()

#clear entries

self.usernameEntry.delete(0, 'end')

self.pwEntry.delete(0, 'end')

#recreate My Account button

myac = tk.Button(self.top\_frame, text="My Account", command=lambda: self.show\_page(self.plogin))

myac.grid(row = 0, column = 2)

#rebuild Upload Cases frame

self.pUpload.destroy()

self.build\_uploadCases()

self.show\_page(self.pLOOK)

def loginSuccess(self):

'''

commands after login success

'''

#set var

global loginSuccess

loginSuccess = True

#create Logout button

self.logout = tk.Button(self.top\_frame, text="Logout", command=lambda: self.logoutSuccess())

self.logout.grid(row = 1, column = 2)

#recreate My Account button

myac = tk.Button(self.top\_frame, text="My Account", command=lambda: self.show\_page(self.myAC))

myac.grid(row = 0, column = 2)

#rebuild My Account frame

self.build\_myAC()

#rebuild Upload Cases frame

self.pUpload.forget()

self.build\_uploadCases()

self.show\_page(self.myAC)

def build\_cl(self):

'''

build widgets in center left frame

'''

#create buttons

uploadCasesB = tk.Button(self.cl, text = "Upload Cases", command = lambda:self.show\_page(self.pUpload))

lookforCasesB = tk.Button(self.cl, text = "Look for Cases", command = lambda:self.show\_page(self.pLOOK))

#display buttons

uploadCasesB.grid(row = 0, column = 0, pady = 5)

lookforCasesB.grid(row = 1, column = 0)

def findUser(self, id, pw):

'''

check if login is valid

'''

read\_mysql()

#clear error message

self.errorMsg0.destroy()

#set var

global loginPressed

loginPressed = True

correctpw = tk.StringVar()

#check if username is in customer list

for index in range(len(customer\_username\_list)):

if id == customer\_username\_list[index]:

#set var

correctpw = customer\_pw\_list[index]

self.current\_userType = 'customer'

self.current\_userID = customer\_id\_list[index]

#check if username is in provider list

for index in range(len(provider\_username\_list)):

if id == provider\_username\_list[index] and provider\_status\_list[index] == 'Verified':

#set var

correctpw = provider\_pw\_list[index]

self.current\_userType = 'provider'

self.current\_userID = provider\_id\_list[index]

#check if username is in admin list

for index in range(len(admin\_username\_list)):

if id == admin\_username\_list[index]:

#set var

correctpw = admin\_pw\_list[index]

self.current\_userType = 'admin'

self.current\_userID = admin\_id\_list[index]

#check if password valid

if correctpw == pw:

self.loginSuccess()

else:

#clear password entry

self.pwEntry.delete(0, 'end')

self.errorMsg0 = tk.Label(self.plogin, text='''Username or password incorrect,

please try again.''')

self.errorMsg0.grid(row=5, column=1,columnspan=3,pady=3)

self.errorMsg0.config(fg='red')

def build\_login(self):

'''

build the frame to login

'''

#create Login frame

self.plogin = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.plogin.grid(row = 0, column = 0, sticky = 'nsew')

#create error message label

self.errorMsg0 = tk.Label(self.plogin)

#set var

idInput = tk.StringVar()

pwInput = tk.StringVar()

#create labels

l0 = tk.Label(self.plogin, text='Username')

l1 = tk.Label(self.plogin, text='Password')

#display labels

l0.grid(row=0, column=0,pady=3)

l1.grid(row=1, column=0,pady=3)

#create entries

self.usernameEntry = tk.Entry(self.plogin, width=25, textvariable=idInput)

self.pwEntry = tk.Entry(self.plogin, width=25, textvariable=pwInput, show='\*')

#display entries

self.usernameEntry.grid(row=0, column=1,columnspan=3,sticky='w',pady=3)

self.pwEntry.grid(row=1, column=1,columnspan=3,sticky='w',pady=3)

#create buttons

bCreateAC = tk.Button(self.plogin, text='Create Account', command = lambda:self.show\_page(self.pCreateAC))

bLogin = tk.Button(self.plogin, text='Login', command = lambda:self.findUser(idInput.get(), pwInput.get()))

#display buttons

bLogin.grid(row=3, column=1,columnspan=4,pady=3)

bCreateAC.grid(row=4, column=1,columnspan=4,pady=3)

#bind enter key to login

self.usernameEntry.bind('<Return>',lambda event:self.findUser(idInput.get(), pwInput.get()))

self.pwEntry.bind('<Return>',lambda event:self.findUser(idInput.get(), pwInput.get()))

def check\_uploadCases(self):

'''

error checking for uploading cases

'''

#clear error messages

self.errorMsg0.destroy()

self.errorMsg1.destroy()

#clear upload success label

self.uploadSuccessLabel.destroy()

#set var

uploadValid = True

#check empty value

if (not self.decoration\_input.get() and not self.design\_input.get()) or not self.v2.get() or not self.description\_input.get() or not self.address\_input.get() or not self.area\_input.get():

self.errorMsg0 = tk.Label(self.pUpload, text='Empty value!')

self.errorMsg0.grid(row=11, column=0)

self.errorMsg0.config(fg='red')

uploadValid = False

#check if area is positive integer

if self.area\_input.get():

if not self.area\_input.get().isdigit():

self.errorMsg1 = tk.Label(self.pUpload, text='Invalid area')

self.errorMsg1.grid(row=12, column=0)

self.errorMsg1.config(fg='red')

uploadValid = False

elif int(self.area\_input.get()) == 0:

self.errorMsg1 = tk.Label(self.pUpload, text='Invalid area!')

self.errorMsg1.grid(row=12, column=0)

self.errorMsg1.config(fg='red')

uploadValid = False

if uploadValid:

self.mysql\_uploadCases()

def mysql\_uploadCases(self):

'''

insert new cases to mysql

'''

#set var

district = str()

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

#get category

if self.decoration\_input.get() and not self.design\_input.get():

category = 'decoration'

elif self.design\_input.get() and not self.decoration\_input.get():

category = 'design'

else:

category = 'decoration,design'

#get district

if self.v2.get() == 1:

district = 'Islands'

elif self.v2.get() == 2:

district = 'Kwai Tsing'

elif self.v2.get() == 3:

district = 'North'

elif self.v2.get() == 4:

district = 'Sai Kung'

elif self.v2.get() == 5:

district = 'Sha Tin'

elif self.v2.get() == 6:

district = 'Tai Po'

elif self.v2.get() == 7:

district = 'Tsuen Wan'

elif self.v2.get() == 8:

district = 'Tuen Mun'

elif self.v2.get() == 9:

district = 'Yuen Long'

elif self.v2.get() == 10:

district = 'Kowloon City'

elif self.v2.get() == 11:

district = 'Kwun Tong'

elif self.v2.get() == 12:

district = 'Sham Shui Po'

elif self.v2.get() == 13:

district = 'Wong Tai Sin'

elif self.v2.get() == 14:

district = 'Yau Tsim Mong'

elif self.v2.get() == 15:

district = 'Central and Western'

elif self.v2.get() == 16:

district = 'Eastern'

elif self.v2.get() == 17:

district = 'Southern'

elif self.v2.get() == 18:

district = 'Wan Chai'

#get the datetime now

dt = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

mycursor = connection.cursor()

#get the last case ID

mycursor.execute("Select case\_id from cases order by case\_id desc limit 1")

last\_caseID = mycursor.fetchone()[0] + 1

mycursor.execute("INSERT INTO cases(case\_id, customer\_id, category, description, address, district, area, status, created\_dt) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)", (last\_caseID, self.current\_userID,category,self.description\_input.get(),self.address\_input.get(),district,self.area\_input.get(),'Verifying',dt))

connection.commit()

#create and display label after sql successfully run

l0 = tk.Label(self.pUpload, text = 'Request submitted, please wait for verification within 24 working hours to upload')

l0.grid(row = 11, column = 0,columnspan=5,sticky='w')

#clear uploaded input

self.decoration\_input.set(False)

self.design\_input.set(False)

self.pUploade0.delete(0, 'end')

self.pUploade1.delete(0, 'end')

self.v2.set(None)

self.pUploade2.delete(0, 'end')

#rebuild My Account

self.build\_myAC()

self.show\_page(self.pUpload)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

def build\_uploadCases(self):

'''

build the frame to upload cases

'''

#create Upload Cases frame

self.pUpload = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.pUpload.grid(row = 0, column = 0, sticky = 'nsew')

#formatting the frame

for e in range(10):

self.pUpload.grid\_columnconfigure(e, weight = 1)

self.cc.grid\_columnconfigure(e, weight = 1)

#create error message labels

self.errorMsg0 = tk.Label(self.pUpload)

self.errorMsg1 = tk.Label(self.pUpload)

#create upload success label

self.uploadSuccessLabel = tk.Label(self.pUpload)

#set var

self.design\_input = tk.BooleanVar()

self.decoration\_input = tk.BooleanVar()

self.v2 = tk.IntVar()

self.description\_input = tk.StringVar()

self.address\_input = tk.StringVar()

self.area\_input = tk.StringVar()

#create data title labels

l0 = tk.Label(self.pUpload, text='Category')

l1 = tk.Label(self.pUpload, text='Description')

l2 = tk.Label(self.pUpload, text='Address')

l3 = tk.Label(self.pUpload, text='District')

l4 = tk.Label(self.pUpload, text='Area (foot)')

#display data title labels

l0.grid(row=0, column=0, sticky='w')

l1.grid(row=1, column=0, sticky='w')

l2.grid(row=3, column=0, sticky='w')

l3.grid(row=5, column=0, sticky='w')

l4.grid(row=9, column=0, sticky='w', columnspan = 2)

#create check buttons for case category

pUploadcb0 = tk.Checkbutton(self.pUpload, text='Design', var = self.design\_input)

pUploadcb1 = tk.Checkbutton(self.pUpload, text='Decoration', var = self.decoration\_input)

#display check buttons for case category

pUploadcb0.grid(row=0, column=0, columnspan=2, padx=10)

pUploadcb1.grid(row=0, column=1, columnspan=2, padx=10)

#create radio buttons for case district

pUploadrb0 = tk.Radiobutton(self.pUpload, text='Islands', var=self.v2, value=1)

pUploadrb1 = tk.Radiobutton(self.pUpload, text='Kwai Tsing', var=self.v2, value=2)

pUploadrb2 = tk.Radiobutton(self.pUpload, text='North', var=self.v2, value=3)

pUploadrb3 = tk.Radiobutton(self.pUpload, text='Sai Kung', var=self.v2, value=4)

pUploadrb4 = tk.Radiobutton(self.pUpload, text='Sha Tin', var=self.v2, value=5)

pUploadrb5 = tk.Radiobutton(self.pUpload, text='Tai Po', var=self.v2, value=6)

pUploadrb6 = tk.Radiobutton(self.pUpload, text='Tsuen Wan', var=self.v2, value=7)

pUploadrb7 = tk.Radiobutton(self.pUpload, text='Tuen Mun', var=self.v2, value=8)

pUploadrb8 = tk.Radiobutton(self.pUpload, text='Yuen Long', var=self.v2, value=9)

pUploadrb9 = tk.Radiobutton(self.pUpload, text='Kowloon City', var=self.v2, value=10)

pUploadrb10 = tk.Radiobutton(self.pUpload, text='Kwun Tong', var=self.v2, value=11)

pUploadrb11 = tk.Radiobutton(self.pUpload, text='Sham Shui Po', var=self.v2, value=12)

pUploadrb12 = tk.Radiobutton(self.pUpload, text='Wong Tai Sin', var=self.v2, value=13)

pUploadrb13 = tk.Radiobutton(self.pUpload, text='Yau Tsim Mong', var=self.v2, value=14)

pUploadrb14 = tk.Radiobutton(self.pUpload, text='Central and Western', var=self.v2, value=15)

pUploadrb15 = tk.Radiobutton(self.pUpload, text='Eastern', var=self.v2, value=16)

pUploadrb16 = tk.Radiobutton(self.pUpload, text='Southern', var=self.v2, value=17)

pUploadrb17 = tk.Radiobutton(self.pUpload, text='Wan Chai', var=self.v2, value=18)

#display radio buttons for case district

pUploadrb0.grid(row=6, column=0,sticky='w')

pUploadrb1.grid(row=6, column=1,sticky='w')

pUploadrb2.grid(row=6, column=2,columnspan=7,sticky='w')

pUploadrb3.grid(row=6, column=3,columnspan=7,sticky='w')

pUploadrb4.grid(row=6, column=4,columnspan=7,sticky='w')

pUploadrb5.grid(row=6, column=5,columnspan=7,sticky='w')

pUploadrb6.grid(row=7, column=0,sticky='w')

pUploadrb7.grid(row=7, column=1,sticky='w')

pUploadrb8.grid(row=7, column=2,sticky='w')

pUploadrb9.grid(row=7, column=3,sticky='w')

pUploadrb10.grid(row=7, column=4,sticky='w')

pUploadrb11.grid(row=7, column=5,sticky='w')

pUploadrb12.grid(row=8, column=0,sticky='w')

pUploadrb13.grid(row=8, column=1,sticky='w')

pUploadrb14.grid(row=8, column=2,sticky='w')

pUploadrb15.grid(row=8, column=3,sticky='w')

pUploadrb16.grid(row=8, column=4,sticky='w')

pUploadrb17.grid(row=8, column=5,sticky='w')

#create entries

self.pUploade0 = tk.Entry(self.pUpload, width=40, textvariable=self.description\_input)

self.pUploade1 = tk.Entry(self.pUpload, width=30, textvariable=self.address\_input)

self.pUploade2 = tk.Entry(self.pUpload, width=6, textvariable=self.area\_input)

#display entries

self.pUploade0.grid(row=2, column=0, columnspan=5, sticky='w')

self.pUploade1.grid(row=4, column=0, columnspan=5, sticky='w')

self.pUploade2.grid(row=9, column=0, sticky='e')

#create submit button

pUploadb0 = tk.Button(self.pUpload, text='Submit', command = self.check\_uploadCases)

#check if user has login

if not loginSuccess:

self.loginWarning = tk.Label(self.pUpload, text = "Click 'My Account' to login first")

self.loginWarning.grid(row=10, column=1,columnspan=6)

self.loginWarning.config(fg='red')

#check if user is customer

elif self.current\_userType != 'customer':

self.loginWarning = tk.Label(self.pUpload, text = "Create a customer account to upload cases")

self.loginWarning.grid(row=10, column=1)

self.loginWarning.config(fg='red')

#display submit button only when user is 'customer'

else:

pUploadb0.grid(row=10, column=8, sticky='w')

self.pUpload.lower()

def check\_sortLookforCases(self):

'''

error checking and get sql for sorting

'''

#set var

sortValid = True

areaValid = False

orderby = tk.StringVar()

whereCategory = str()

whereDistrict = str()

whereArea = str()

districtDict = {}

#clear error messages

self.errorMsg0.destroy()

#get sql of select

select = "select \* from cases where status = 'Posted' and "

districts = ['Islands','Kwai Tsing','North','Sai Kung','Sha Tin','Tai Po','Tsuen Wan','Tuen Mun','Yuen Long','Kowloon City','Kwun Tong','Sham Shui Po','Wong Tai Sin','Yau Tsim Mong','Central and Western','Eastern','Southern','Wan Chai']

#collect district vars

districtsVar = [self.district0.get(),self.district1.get(),self.district2.get(),self.district3.get(),self.district4.get(),self.district5.get(),self.district6.get(),self.district7.get(),self.district8.get(),self.district9.get(),self.district10.get(),self.district11.get(),self.district12.get(),self.district13.get(),self.district14.get(),self.district15.get(),self.district16.get(),self.district17.get()]

#get sql form sorting districts

if self.district0.get() or self.district1.get() or self.district2.get() or self.district3.get() or self.district4.get() or self.district5.get() or self.district6.get() or self.district7.get() or self.district8.get() or self.district9.get() or self.district10.get() or self.district11.get() or self.district12.get() or self.district13.get() or self.district14.get() or self.district15.get() or self.district16.get() or self.district17.get():

#check if more than 1 district selected

if (self.district0.get() + self.district1.get() + self.district2.get() + self.district3.get() + self.district4.get() + self.district5.get() + self.district6.get() + self.district7.get() + self.district8.get() + self.district9.get() + self.district10.get() + self.district11.get() + self.district12.get() + self.district13.get() + self.district14.get() + self.district15.get() + self.district16.get() + self.district17.get()) > 1:

whereDistrict = '('

for e in range(len([i for i, x in enumerate(districtsVar) if x])):

if e < len([i for i, x in enumerate(districtsVar) if x])-1:

whereDistrict = whereDistrict + 'district = '+"'"+(districts[[i for i, x in enumerate(districtsVar) if x][e]])+"' or "

else:

whereDistrict = whereDistrict + 'district = '+"'"+(districts[[i for i, x in enumerate(districtsVar) if x][e]])+"')"

else:

whereDistrict = 'district = '+"'"+districts[[i for i, x in enumerate(districtsVar) if x][0]]+"'"

else:

whereDistrict = ''

#error checking for area input, and get sql

if self.areaMinInput.get() or self.areaMaxInput.get():

if not self.areaMinInput.get() or not self.areaMaxInput.get():

self.errorMsg0 = tk.Label(self.pLOOK, text='Invalid area')

self.errorMsg0.grid(row=0, column=11)

self.errorMsg0.config(fg='red')

sortValid = False

elif not self.areaMinInput.get().isdecimal() or not self.areaMaxInput.get().isdecimal():

self.errorMsg0 = tk.Label(self.pLOOK, text='Invalid area')

self.errorMsg0.grid(row=0, column=11)

self.errorMsg0.config(fg='red')

sortValid = False

elif int(self.areaMinInput.get()) < 0 or int(self.areaMaxInput.get()) >9999 or (int(self.areaMinInput.get()) > int(self.areaMaxInput.get())):

self.errorMsg0 = tk.Label(self.pLOOK, text='Invalid area')

self.errorMsg0.grid(row=0, column=11)

self.errorMsg0.config(fg='red')

sortValid = False

else:

areaValid = True

whereArea = 'area between ' + self.areaMinInput.get() + ' and ' + self.areaMaxInput.get()

#get sql form sorting category

if self.sortDesign.get() and not self.sortDecor.get():

whereCategory = "category = 'design'"

elif not self.sortDesign.get() and self.sortDecor.get():

whereCategory = "category = 'decoration'"

elif self.sortDesign.get() and self.sortDecor.get():

whereCategory = "category = 'decoration,design'"

else:

whereCategory = ''

#get sql of order by from sorting menu

if self.sortingChoice.get() == 'Case ID(asc)':

orderby = ''

elif self.sortingChoice.get() == 'Case ID(desc)':

orderby = ' order by case\_id desc'

elif self.sortingChoice.get() == 'Date(new to old)':

orderby = ' order by created\_dt desc'

elif self.sortingChoice.get() == 'Date(old to new)':

orderby = ' order by created\_dt asc'

#assign sql syntax

if whereCategory:

if whereDistrict:

whereDistrict = ' and ' + whereDistrict

if whereArea:

whereArea = ' and ' + whereArea

elif whereDistrict and whereArea:

whereArea = ' and ' + whereArea

if not(self.sortDesign.get() or self.sortDecor.get()) and not whereDistrict and not areaValid:

select = "select \* from cases where status = 'Posted'"

#combine sqls

where = whereCategory + whereDistrict + whereArea

sql = select + where + orderby

if sortValid:

self.mysql\_sortLookforCases(sql)

def mysql\_sortLookforCases(self, sql):

'''

display sorted cases with status 'Posted' from mysql

'''

#set var

dict = {}

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute(sql)

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#get total page no. and no. of data in last page

if caseNum > 15:

remainder = caseNum % 15

if remainder == 0:

pageNum = caseNum//15

else:

pageNum = caseNum//15 + 1

else:

pageNum = 1

remainder = caseNum

def nextPage(page):

'''

button to show next page

'''

#destroy previous data

for e in range(2,self.pLOOK.grid\_size()[1]):

l=list(self.pLOOK.grid\_slaves(row=e))

for w in l:

w.destroy()

self.show\_page(self.pLOOK)

#set var

page = page + 1

counter = 1

startValue = (page-1)\*15

endValue = page\*15

#display case data

for row in result[startValue: endValue]:

counter += 1

dict["caseID" + str(counter-2)] = row[0]

data\_label = tk.Label(self.pLOOK, text = row[0])

data\_label.grid(row = counter, column = 0, pady = 3)

data\_label = tk.Label(self.pLOOK, text = row[3])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.pLOOK, text = row[6])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.pLOOK, text = row[7])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.pLOOK, text = row[10])

data\_label.grid(row = counter, column = 4)

if page < pageNum:

#display Detail button

for e in range(15):

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

#create button for next page

if page != pageNum:

nextpage = tk.Button(self.pLOOK, text = 'Next page', command = lambda: nextPage(page))

nextpage.grid(row = 17, column = 7)

elif page == pageNum and caseNum != 0:

#display Detail button

if remainder != 0:

for e in range(remainder):

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

else:

#display Detail button

for e in range(15):

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

#create button for previous page

if page != 1:

previouspage = tk.Button(self.pLOOK, text = 'Previous page', command = lambda: previousPage(page))

previouspage.grid(row = 17, column = 6)

#display total result no. after sql successfully run

resultNum = tk.Label(self.pLOOK, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 8, pady = 3)

def previousPage(page):

'''

button to show previous page

'''

#destroy previous data

for e in range(2,self.pLOOK.grid\_size()[1]):

l=list(self.pLOOK.grid\_slaves(row=e))

for w in l:

w.destroy()

self.show\_page(self.pLOOK)

#set var

page = page - 1

counter = 1

startValue = (page-1)\*15

endValue = page\*15

#display case data

for row in result[startValue: endValue]:

counter += 1

dict["caseID" + str(counter-2)] = row[0]

data\_label = tk.Label(self.pLOOK, text = row[0])

data\_label.grid(row = counter, column = 0, pady = 3)

data\_label = tk.Label(self.pLOOK, text = row[3])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.pLOOK, text = row[6])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.pLOOK, text = row[7])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.pLOOK, text = row[10])

data\_label.grid(row = counter, column = 4)

#display Detail button

for e in range(15):

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

#create button for next page

nextpage = tk.Button(self.pLOOK, text = 'Next page', command = lambda: nextPage(page))

nextpage.grid(row = 17, column = 7)

#display total result no. after sql successfully run

resultNum = tk.Label(self.pLOOK, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 8, pady = 3)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.pLOOK, text = 'Previous page', command = lambda: previousPage(page))

previouspage.grid(row = 17, column = 6)

#set var

page = 0

nextPage(page)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

def build\_lookforCases(self):

'''

build the frame to look for cases

'''

#create Look for Cases frame

self.pLOOK = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.pLOOK.grid(row = 0, column = 0, sticky = 'nsew')

#formatting the frame

for e in range(10):

self.pLOOK.grid\_columnconfigure(e, weight = 1)

self.cc.grid\_columnconfigure(e, weight = 1)

#set var

self.sortingChoice = tk.StringVar()

self.DistrictChoice = tk.StringVar()

self.sortDesign = tk.BooleanVar()

self.sortDecor = tk.BooleanVar()

var = tk.StringVar()

var.set('District')

self.categorySort = tk.IntVar()

self.categorySort.set(0)

#create error messages

self.errorMsg0 = tk.Label(self.pLOOK)

#create sorting widgets

sort = tk.Label(self.pLOOK, text = 'Sorted by')

category = tk.Label(self.pLOOK, text = 'Category(Display only):')

design = tk.Checkbutton(self.pLOOK, text='Design', var = self.sortDesign)

decor = tk.Checkbutton(self.pLOOK, text='Decoration', var = self.sortDecor)

areaMin = tk.Label(self.pLOOK, text = 'Area(min[>=0]):')

self.areaMinInput = tk.Entry(self.pLOOK, width = 4)

areaMax = tk.Label(self.pLOOK, text = 'Area(max[<10000]):')

self.areaMaxInput = tk.Entry(self.pLOOK, width = 4)

#display sorting widgets

sort.grid(row = 0, column = 0, stick = 'e')

category.grid(row = 0, column = 2)

design.grid(row = 0, column = 3)

decor.grid(row = 0, column = 4)

areaMin.grid(row = 0, column = 5)

self.areaMinInput.grid(row = 0, column = 6)

areaMax.grid(row = 0, column = 7)

self.areaMaxInput.grid(row = 0, column = 8)

#create option menu

optionList = ('Case ID(asc)', 'Case ID(desc)', 'Date(new to old)', 'Date(old to new)')

districtMenu = tk.OptionMenu(self.pLOOK, variable = var, value = 'District')

self.sortingChoice.set(optionList[0])

sortingMenu = tk.OptionMenu(self.pLOOK, self.sortingChoice, \*optionList)

#create add\_checkbuttons in district menu

districtMenu['menu'].add\_checkbutton(label='Island', onvalue=True, offvalue=False, variable=self.district0)

districtMenu['menu'].add\_checkbutton(label='Kwai Tsing', onvalue=True, offvalue=False, variable=self.district1)

districtMenu['menu'].add\_checkbutton(label='North', onvalue=True, offvalue=False, variable=self.district2)

districtMenu['menu'].add\_checkbutton(label='Sai Kung', onvalue=True, offvalue=False, variable=self.district3)

districtMenu['menu'].add\_checkbutton(label='Sha Tin', onvalue=True, offvalue=False, variable=self.district4)

districtMenu['menu'].add\_checkbutton(label='Tai Po', onvalue=True, offvalue=False, variable=self.district5)

districtMenu['menu'].add\_checkbutton(label='Tsuen Wan', onvalue=True, offvalue=False, variable=self.district6)

districtMenu['menu'].add\_checkbutton(label='Tuen Mun', onvalue=True, offvalue=False, variable=self.district7)

districtMenu['menu'].add\_checkbutton(label='Yuen Long', onvalue=True, offvalue=False, variable=self.district8)

districtMenu['menu'].add\_checkbutton(label='Kowloon City', onvalue=True, offvalue=False, variable=self.district9)

districtMenu['menu'].add\_checkbutton(label='Kwun Tong', onvalue=True, offvalue=False, variable=self.district10)

districtMenu['menu'].add\_checkbutton(label='Sham Shui Po', onvalue=True, offvalue=False, variable=self.district11)

districtMenu['menu'].add\_checkbutton(label='Wong Tai Sin', onvalue=True, offvalue=False, variable=self.district12)

districtMenu['menu'].add\_checkbutton(label='Yau Tsim Mong', onvalue=True, offvalue=False, variable=self.district13)

districtMenu['menu'].add\_checkbutton(label='Central and Western', onvalue=True, offvalue=False, variable=self.district14)

districtMenu['menu'].add\_checkbutton(label='Eastern', onvalue=True, offvalue=False, variable=self.district15)

districtMenu['menu'].add\_checkbutton(label='Southern', onvalue=True, offvalue=False, variable=self.district16)

districtMenu['menu'].add\_checkbutton(label='Wan Chai', onvalue=True, offvalue=False, variable=self.district17)

#display menus

sortingMenu.grid(row = 0, column = 1, stick = 'w')

districtMenu.grid(row = 0, column = 9, stick = 'w')

#create and display sorting button

sortB = tk.Button(self.pLOOK, text = 'Sort', command = lambda: self.check\_sortLookforCases())

sortB.grid(row = 0, column = 10, stick = 'e')

#create data titles

l0 = tk.Label(self.pLOOK, text = 'Case ID')

l1 = tk.Label(self.pLOOK, text = 'Category')

l2 = tk.Label(self.pLOOK, text = 'District')

l3 = tk.Label(self.pLOOK, text = 'Area (foot)')

l4 = tk.Label(self.pLOOK, text = 'Created Date')

#display data titles

l0.grid(row = 1, column = 0)

l1.grid(row = 1, column = 1)

l2.grid(row = 1, column = 2)

l3.grid(row = 1, column = 3)

l4.grid(row = 1, column = 4)

#set var

dict = {}

#show all cases data first

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("select \* from cases where status = 'Posted'")

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#get total page no. and no. of data in last page

if caseNum > 15:

remainder = caseNum % 15

if remainder == 0:

pageNum = caseNum//15

else:

pageNum = caseNum//15 + 1

else:

pageNum = 1

remainder = caseNum

def nextPage(page):

'''

'''

#destroy previous data

for e in range(2,self.pLOOK.grid\_size()[1]):

l=list(self.pLOOK.grid\_slaves(row=e))

for w in l:

w.destroy()

self.show\_page(self.pLOOK)

#set var

page = page + 1

counter = 1

startValue = (page-1)\*15

endValue = page\*15

#display case data

for row in result[startValue: endValue]:

counter += 1

dict["caseID" + str(counter-2)] = row[0]

data\_label = tk.Label(self.pLOOK, text = row[0])

data\_label.grid(row = counter, column = 0, pady = 3)

data\_label = tk.Label(self.pLOOK, text = row[3])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.pLOOK, text = row[6])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.pLOOK, text = row[7])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.pLOOK, text = row[10])

data\_label.grid(row = counter, column = 4)

if page < pageNum:

#display Detail button

for e in range(15):

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

if page != pageNum:

#create button for next page

nextpage = tk.Button(self.pLOOK, text = 'Next page', command = lambda: nextPage(page))

nextpage.grid(row = 17, column = 7)

elif page == pageNum and caseNum != 0:

if remainder != 0:

#display Detail button

for e in range(remainder):

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

else:

for e in range(15):

#display Detail button

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.pLOOK, text = 'Previous page', command = lambda: previousPage(page))

previouspage.grid(row = 17, column = 6)

#display total result no. after sql successfully run

resultNum = tk.Label(self.pLOOK, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 8, pady = 3)

def previousPage(page):

#destroy previous data

for e in range(2,self.pLOOK.grid\_size()[1]):

l=list(self.pLOOK.grid\_slaves(row=e))

for w in l:

w.destroy()

self.show\_page(self.pLOOK)

#set var

page = page - 1

counter = 1

startValue = (page-1)\*15

endValue = page\*15

#display case data

for row in result[startValue: endValue]:

counter += 1

dict["caseID" + str(counter-2)] = row[0]

data\_label = tk.Label(self.pLOOK, text = row[0])

data\_label.grid(row = counter, column = 0, pady = 3)

data\_label = tk.Label(self.pLOOK, text = row[3])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.pLOOK, text = row[6])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.pLOOK, text = row[7])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.pLOOK, text = row[10])

data\_label.grid(row = counter, column = 4)

#display Detail button

for e in range(15):

dict["detailB" + str(e)] = tk.Button(self.pLOOK, text = 'Detail', command = lambda e=e:self.build\_lookforCases\_detail(dict["caseID" + str(e)]))

dict["detailB" + str(e)].grid(row = e+2, column = 5, padx = 10)

#create button for next page

nextpage = tk.Button(self.pLOOK, text = 'Next page', command = lambda: nextPage(page))

nextpage.grid(row = 17, column = 7)

#display total result no. after sql successfully run

resultNum = tk.Label(self.pLOOK, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 8, pady = 3)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.pLOOK, text = 'Previous page', command = lambda: previousPage(page))

previouspage.grid(row = 17, column = 6)

#set var

page = 0

nextPage(page)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

self.pLOOK.lower()

def check\_applyCases(self):

'''

error checking for applying applications

'''

#clear error messages

self.errorMsg0.destroy()

self.errorMsg1.destroy()

#clear apply success label

self.applySuccessLabel.destroy()

#set var

applyValid = True

#check empty value

if not self.description\_input.get() or not self.price\_input.get():

self.errorMsg0 = tk.Label(self.pDetail, text='Empty value!')

self.errorMsg0.grid(row=12, column=1)

self.errorMsg0.config(fg='red')

applyValid = False

#check if area is decimal

if self.price\_input.get() and not self.price\_input.get().isdecimal():

self.errorMsg1 = tk.Label(self.pDetail, text='Price must be integer!')

self.errorMsg1.grid(row=13, column=1)

self.errorMsg1.config(fg='red')

applyValid = False

if applyValid:

self.mysql\_applyCases()

def mysql\_applyCases(self):

'''

insert new cases to mysql

'''

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

#get the datetime now

dt = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

mycursor = connection.cursor()

#get the last cases application ID

mycursor.execute("Select casesApplication\_id from cases\_application order by casesApplication\_id desc limit 1")

last\_casesApplicationID = mycursor.fetchone()[0] + 1

mycursor.execute("INSERT INTO cases\_application(casesApplication\_id, case\_id, customer\_id, provider\_id, description, estimated\_price, admin\_status, status, apply\_dt) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)", (last\_casesApplicationID, self.applyCaseID,self.applyCustomerID,self.current\_userID,self.description\_input.get(),self.price\_input.get(),'Verifying','Waiting',dt))

connection.commit()

#clear widgets

self.pDetailDescription.delete(0, 'end')

self.pDetailPrice.delete(0, 'end')

self.applyB.destroy()

#create and display label after sql successfully run

self.applySuccessLabel = tk.Label(self.pDetail, text = '''Apply submitted,

please wait for verification within 24 working hours to upload''')

self.applySuccessLabel.grid(row = 13, column = 1, columnspan = 5, sticky = 'w')

#rebuild frames

read\_mysql()

self.build\_myAC()

self.show\_page(self.pDetail)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

def build\_lookforCases\_detail(self, caseID):

'''

build the frame to display cases details

'''

#create Look for Cases frame

self.pDetail = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.pDetail.grid(row = 0, column = 0, sticky = 'nsew')

#set var

self.applyCaseID = caseID

access = True

noError = False

#create data titles

l0 = tk.Label(self.pDetail, text = 'Case ID')

l1 = tk.Label(self.pDetail, text = 'Customer name')

l2 = tk.Label(self.pDetail, text = 'Category')

l3 = tk.Label(self.pDetail, text = 'Description')

l4 = tk.Label(self.pDetail, text = 'Address')

l5 = tk.Label(self.pDetail, text = 'District')

l6 = tk.Label(self.pDetail, text = 'Area (foot)')

l7 = tk.Label(self.pDetail, text = 'Since')

#display data titles

l0.grid(row = 0, column = 1)

l1.grid(row = 0, column = 2)

l2.grid(row = 2, column = 0)

l3.grid(row = 3, column = 0)

l4.grid(row = 4, column = 0)

l5.grid(row = 5, column = 0)

l6.grid(row = 6, column = 0)

l7.grid(row = 7, column = 0)

#create data labels

caseid = tk.Label(self.pDetail, text = self.applyCaseID)

self.applyCustomerID = casefk\_customer\_id\_list[caseID-1]

customerName = tk.Label(self.pDetail, text = customer\_name\_list[self.applyCustomerID-1])

category = tk.Label(self.pDetail, text = case\_category\_list[caseID-1])

description = tk.Label(self.pDetail, text = case\_description\_list[caseID-1])

address = tk.Label(self.pDetail, text = case\_address\_list[caseID-1])

district = tk.Label(self.pDetail, text = case\_district\_list[caseID-1])

area = tk.Label(self.pDetail, text = case\_area\_list[caseID-1])

dt = tk.Label(self.pDetail, text = case\_createdDT\_list[caseID-1])

#display data labels

caseid.grid(row = 1, column = 1)

customerName.grid(row = 1, column = 2)

category.grid(row = 2, column = 1)

description.grid(row = 3, column = 1)

address.grid(row = 4, column = 1)

district.grid(row = 5, column = 1)

area.grid(row = 6, column = 1)

dt.grid(row = 7, column = 1)

self.add\_sep(8, 0, self.pDetail)

if loginSuccess and self.current\_userType == 'provider':

for e in range(len(casesApplyfk\_case\_id\_list)):

if casesApplyfk\_case\_id\_list[e] == caseID and self.current\_userID == casesApplyfk\_provider\_id\_list[e]:

errorMsg0 = tk.Label(self.pDetail, text = 'You can only apply the same case once')

errorMsg0.grid(row=9, column=2)

errorMsg0.config(fg='red')

access = False

noError = True

if self.current\_userType == 'provider':

if access:

#create error messages

self.errorMsg0 = tk.Label(self.pDetail)

self.errorMsg1 = tk.Label(self.pDetail)

self.applySuccessLabel = tk.Label(self.pDetail)

#set var

self.description\_input = tk.StringVar()

self.price\_input = tk.StringVar()

#create widgets

title = tk.Label(self.pDetail, text = 'Application')

description = tk.Label(self.pDetail, text = 'Description')

price = tk.Label(self.pDetail, text = 'Estimated price(HKD)')

self.pDetailDescription = tk.Entry(self.pDetail, width=40, textvariable=self.description\_input)

self.pDetailPrice = tk.Entry(self.pDetail, width=30, textvariable=self.price\_input)

self.applyB = tk.Button(self.pDetail, text = 'Apply', command = lambda: self.check\_applyCases())

#display widgets

title.grid(row = 9, column = 0)

description.grid(row=10, column=0, columnspan=5, sticky='w')

price.grid(row=11, column=0, columnspan=5, sticky='w')

self.pDetailDescription.grid(row=10, column=1, columnspan=5, sticky='w')

self.pDetailPrice.grid(row=11, column=1, columnspan=5, sticky='w',padx=60)

self.applyB.grid(row=12, column=3)

noError = True

if self.current\_userType == 'customer' and self.current\_userID == self.applyCustomerID:

noError = True

if not noError:

errorMsg0 = tk.Label(self.pDetail, text = 'Login provider account to apply')

errorMsg0.grid(row=9, column=2)

errorMsg0.config(fg='red')

self.pDetail.lower()

self.show\_page(self.pDetail)

def build\_orderBuildingMaterial(self, ID):

'''

build the frame to order building materials

'''

def changeItemStatus(item, weightL, priceL, counter, row1, row2):

'''

change item status when selected different items

'''

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

#destroy previous data

weightL.destroy()

priceL.destroy()

mycursor = connection.cursor()

#TotalpriceAfterDiscount = ('%.2f')%(quantity\*discount)

mycursor.execute("SELECT weight\_g, price FROM bm\_quantity where item = '%s'"%(item))

itemData = mycursor.fetchone()

dict["weight" + str(counter)] = tk.Label(self.pOrderBM, text=itemData[0])

dict["weight" + str(counter)].grid(row=row1, column=1, sticky='w')

dict["price" + str(counter)] = tk.Label(self.pOrderBM, text=itemData[1])

dict["price" + str(counter)].grid(row=row2, column=1, sticky='w')

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

def checkSubmit(dict, ID):

'''

check if the submission is valid, then submit to the database

'''

#set var

global total, counter, row

submitValid = True

self.errorMsg0.destroy()

self.errorMsg1.destroy()

self.successMsg.destroy()

#check if the submission is valid

for e in range(total):

if dict["quantityVarBM" + str(e+1)].get().isdigit():

if int(dict["quantityVarBM" + str(e+1)].get()) <= 0:

submitValid = False

else:

submitValid = False

if not submitValid:

for e in range(total):

if not dict["quantityVarBM" + str(e+1)].get():

print(dict["quantityVarBM" + str(e+1)].get())

self.errorMsg0 = tk.Label(self.pOrderBM, text='Empty value!')

self.errorMsg0.grid(row=row, column=0, columnspan=3)

self.errorMsg0.config(fg='red')

else:

self.errorMsg1 = tk.Label(self.pOrderBM, text='Quantity must be positive integer!')

self.errorMsg1.grid(row=row+1, column=0, columnspan=3)

self.errorMsg1.config(fg='red')

else:

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

#get the datetime now

dt = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

#get the last no.

mycursor.execute("Select no from building\_material order by no desc limit 1")

num = mycursor.fetchone()[0] + 1

#get the last order id

mycursor.execute("Select order\_id from building\_material order by order\_id desc limit 1")

orderID = mycursor.fetchone()[0] + 1

#insert the order into database

for e in range(total):

mycursor.execute("Select price from bm\_quantity where item = '%s'"%(dict["orderChoiceBM" + str(e+1)].get()))

price = mycursor.fetchone()[0]

mycursor.execute("Insert into building\_material values (%s,%s,%s,%s,%s,%s,%s,%s,%s)", (num, ID, orderID, dict["orderChoiceBM" + str(e+1)].get(), dict["quantityVarBM" + str(e+1)].get(), price, int(dict["quantityVarBM" + str(e+1)].get())\*price, 'pending', dt))

#set var

num += 1

#clear entries

dict["quantityE" + str(e+1)].delete(0, 'end')

#output json

mycursor.execute("Select \* from building\_material where order\_id = %s order by no asc"%(orderID))

result = mycursor.fetchall()

mydict = create\_dict()

for row in result:

mydict.add(row[0],({"case\_id":row[1],"order\_id":row[2],"item":row[3],"quantity":row[4],"price":row[5],"total\_price":row[6],"status":row[7],"created\_dt":row[8]}))

orderBM\_json = json.dumps(mydict, indent=2, sort\_keys=True, default=str)

#rebuild pages

self.build\_myAC()

self.build\_orderBuildingMaterial(ID)

#display success message

self.successMsg = tk.Label(self.pOrderBM, text='Order successed, please wait for further notification.')

self.successMsg.grid(row=row+1, column=0, columnspan=3)

connection.commit()

#send email

try:

mainMsg = orderBM\_json

user = '3s.company3ss@gmail.com'

password = '3s11002299.'

receiver = "3s.company3ss@gmail.com"

subject = "ABC Interior Design & Decoration Agency | Order"

message = "Subject: {} \n\n{}".format(subject, mainMsg)

send\_to = ("{}".format(receiver))

mail = smtplib.SMTP\_SSL('smtp.gmail.com', 465)

mail.ehlo()

mail.login(user, password)

mail.sendmail(user, send\_to, message)

mail.close()

except Exception as x:

print("Error: unable to send email")

print(x)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

#create Upload Cases frame

self.pOrderBM = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.pOrderBM.grid(row = 0, column = 0, sticky = 'nsew')

#formatting the frame

for e in range(10):

self.pOrderBM.grid\_columnconfigure(e, weight = 1)

self.cc.grid\_columnconfigure(e, weight = 1)

#create error message labels

self.errorMsg0 = tk.Label(self.pOrderBM)

self.errorMsg1 = tk.Label(self.pOrderBM)

#create upload success label

self.successMsg = tk.Label(self.pOrderBM)

#set var

global total, counter, row

dict = {}

counter = 1

total = 1

row = 5

dict["quantityVarBM" + str(counter)] = tk.StringVar()

dict["orderChoiceBM" + str(counter)] = tk.StringVar()

#create widgets

l0 = tk.Label(self.pOrderBM, text='Case ID')

lID = tk.Label(self.pOrderBM, text=ID)

l1 = tk.Label(self.pOrderBM, text='Item')

l2 = tk.Label(self.pOrderBM, text='Weight(g)/one')

dict["weight" + str(counter)] = tk.Label(self.pOrderBM, text='1200')

l3 = tk.Label(self.pOrderBM, text='Quantity')

dict["quantityE" + str(counter)] = tk.Entry(self.pOrderBM, width=3, textvariable=dict["quantityVarBM" + str(counter)])

l4 = tk.Label(self.pOrderBM, text='Price')

dict["price" + str(counter)] = tk.Label(self.pOrderBM, text='10')

#display widgets

l0.grid(row=0, column=0, sticky='w')

lID.grid(row=0, column=1, sticky='w')

l1.grid(row=1, column=0, sticky='w')

l2.grid(row=2, column=0, sticky='w')

dict["weight" + str(counter)].grid(row=2, column=1, sticky='w')

l3.grid(row=3, column=0, sticky='w')

dict["quantityE" + str(counter)].grid(row=3, column=1, columnspan=5, sticky='w')

l4.grid(row=4, column=0, sticky='w')

dict["price" + str(counter)].grid(row=4, column=1, sticky='w')

#create option menu

optionList = ('Bricks', 'Cement', 'Concrete', 'Sand', 'Reinforced Concrete', 'Glass', 'Plastic', 'Wood', 'Tiles')

dict["orderChoiceBM" + str(counter)].set(optionList[0])

itemMenu = tk.OptionMenu(self.pOrderBM, dict["orderChoiceBM" + str(counter)], \*optionList, command=lambda item, counter=counter, row=row: changeItemStatus(item, dict["weight" + str(counter)], dict["price" + str(counter)], counter, row-3, row-1))

itemMenu.grid(row = 1, column = 1, stick = 'w')

#more material function

def moreMaterial\_function(optionList):

'''

add more materials

'''

def deleteItem(dict, counter\_temp):

'''

delete items

'''

#delete messages

self.errorMsg0.destroy()

self.errorMsg1.destroy()

self.successMsg.destroy()

global row, counter, total

dict["quantityVarBM" + str(counter\_temp)].set('')

dict["itemL" + str(counter\_temp)].destroy()

dict["itemMenu" + str(counter\_temp)].destroy()

dict["deleteB" + str(counter\_temp)].destroy()

dict["weightL" + str(counter\_temp)].destroy()

dict["weight" + str(counter\_temp)].destroy()

dict["quantityL" + str(counter\_temp)].destroy()

dict["quantityE" + str(counter\_temp)].destroy()

dict["priceL" + str(counter\_temp)].destroy()

dict["price" + str(counter\_temp)].destroy()

#reset counter

if total != counter\_temp:

for e in range(counter\_temp, total):

dict["quantityVarBM" + str(e)] = dict["quantityVarBM" + str(e+1)]

dict["itemL" + str(e)] = dict["itemL" + str(e+1)]

dict["itemL" + str(e)].grid(row=e\*5-4, column=0, sticky='w')

dict["itemMenu" + str(e)] = dict["itemMenu" + str(e+1)]

dict["itemMenu" + str(e)].grid(row = e\*5-4, column = 1, stick = 'w')

dict["deleteB" + str(e+1)].destroy()

dict["deleteB" + str(e)] = tk.Button(self.pOrderBM, text = 'Delete', command = lambda e=e: deleteItem(dict, e))

dict["deleteB" + str(e)].grid(row=e\*5-4, column=2, sticky='w')

dict["weightL" + str(e)] = dict["weightL" + str(e+1)]

dict["weightL" + str(e)].grid(row=e\*5-3, column=0, sticky='w')

dict["weight" + str(e)] = dict["weight" + str(e+1)]

dict["weight" + str(e)].grid(row=e\*5-3, column=1, sticky='w')

dict["quantityL" + str(e)] = dict["quantityL" + str(e+1)]

dict["quantityL" + str(e)].grid(row=e\*5-2, column=0, sticky='w')

dict["quantityE" + str(e)] = dict["quantityE" + str(e+1)]

dict["quantityE" + str(e)].grid(row=e\*5-2, column=1, columnspan=5, sticky='w')

dict["priceL" + str(e)] = dict["priceL" + str(e+1)]

dict["priceL" + str(e)].grid(row=e\*5-1, column=0, sticky='w')

dict["price" + str(e)] = dict["price" + str(e+1)]

dict["price" + str(e)].grid(row=e\*5-1, column=1, sticky='w')

#set var

counter = total-1

total -= 1

row -= 5

#relocate more material & submit buttons

moreB.grid(row=row+1, column=6, sticky='w')

submitB.grid(row=row+2, column=8, sticky='w')

#delete messages

self.errorMsg0.destroy()

self.errorMsg1.destroy()

self.successMsg.destroy()

#set var

global total, counter, row

counter += 1

total += 1

dict["quantityVarBM" + str(counter)] = tk.StringVar()

dict["orderChoiceBM" + str(counter)] = tk.StringVar()

#create widgets

dict["itemL" + str(counter)] = tk.Label(self.pOrderBM, text='Item')

dict["orderChoiceBM" + str(counter)].set(optionList[0])

dict["itemMenu" + str(counter)] = tk.OptionMenu(self.pOrderBM, dict["orderChoiceBM" + str(counter)], \*optionList, command=lambda item, counter=counter, row=row: changeItemStatus(item, dict["weight" + str(counter)], dict["price" + str(counter)], counter, row+2, row+4))

dict["deleteB" + str(counter)] = tk.Button(self.pOrderBM, text = 'Delete', command = lambda counter=counter: deleteItem(dict, counter))

dict["weightL" + str(counter)] = tk.Label(self.pOrderBM, text='Weight(g)/one')

dict["weight" + str(counter)] = tk.Label(self.pOrderBM, text='1200')

dict["quantityL" + str(counter)] = tk.Label(self.pOrderBM, text='Quantity')

dict["quantityE" + str(counter)] = tk.Entry(self.pOrderBM, width=3, textvariable=dict["quantityVarBM" + str(counter)])

dict["priceL" + str(counter)] = tk.Label(self.pOrderBM, text='Price')

dict["price" + str(counter)] = tk.Label(self.pOrderBM, text='10')

#display widgets

self.add\_sep(row, 0, self.pOrderBM)

dict["itemL" + str(counter)].grid(row=row+1, column=0, sticky='w')

dict["itemMenu" + str(counter)].grid(row = row+1, column = 1, stick = 'w')

dict["deleteB" + str(counter)].grid(row=row+1, column=2, sticky='w')

dict["weightL" + str(counter)].grid(row=row+2, column=0, sticky='w')

dict["weight" + str(counter)].grid(row=row+2, column=1, sticky='w')

dict["quantityL" + str(counter)].grid(row=row+3, column=0, sticky='w')

dict["quantityE" + str(counter)].grid(row=row+3, column=1, columnspan=5, sticky='w')

dict["priceL" + str(counter)].grid(row=row+4, column=0, sticky='w')

dict["price" + str(counter)].grid(row=row+4, column=1, sticky='w')

#relocate more material & submit buttons

moreB.grid(row=row+5, column=6, sticky='w')

submitB.grid(row=row+6, column=8, sticky='w')

#set var

row += 5

#more material button

moreB = tk.Button(self.pOrderBM, text='More materials', command = lambda : moreMaterial\_function(optionList))

moreB.grid(row=row, column=6, sticky='w')

#submit button

submitB = tk.Button(self.pOrderBM, text='Submit', command = lambda counter=counter: checkSubmit(dict, ID))

submitB.grid(row=row+1, column=8, sticky='w')

self.pOrderBM.lower()

self.show\_page(self.pOrderBM)

def check\_updatePrivacy(self):

'''

check input data to update user's personal information in Privacy is valid or not

'''

#clear error messages

self.errorMsg0.destroy()

self.errorMsg1.destroy()

self.errorMsg2.destroy()

self.errorMsg3.destroy()

self.errorMsg4.destroy()

self.errorMsg5.destroy()

#clear label 'saved'

self.savedLabel.destroy()

#set var

updateMyACvalid = True

if self.current\_userType == 'customer':

#check empty value

if not self.myACd0.get() or not self.myACd1.get() or not self.myACd2.get() or not self.myACd3.get():

self.errorMsg0 = tk.Label(self.myAC\_privacy, text='Empty value!')

self.errorMsg0.grid(row=9, column=1,columnspan =3, sticky='w')

self.errorMsg0.config(fg='red')

updateMyACvalid = False

#check name valid

nameValid = True

if self.myACd0.get():

for letter in self.myACd0.get():

if not letter.isalpha() and letter not in ' ':

nameValid = False

if len(self.myACd0.get()) > 30:

nameValid = False

if not nameValid:

self.errorMsg1 = tk.Label(self.myAC\_privacy, text='Invalid name')

self.errorMsg1.grid(row=10, column=1,columnspan =3, sticky='w')

self.errorMsg1.config(fg='red')

updateMyACvalid = False

#check username valid

usernameValid = True

if self.myACd1.get() and self.myACd1.get() != customer\_username\_list[self.current\_userID-1]:

for letter in self.myACd1.get():

if len(self.myACd1.get()) < 6 or len(self.myACd1.get()) > 20 or (not letter.isalpha() and not letter.isdecimal() and letter not in '\_'):

usernameValid = False

if not usernameValid:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username should contain 6-20 characters, only in letters, numbers, underscore(\_)')

self.errorMsg2.grid(row=11, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

#check username duplicate

for index in range(len(customer\_username\_list)):

if self.myACd1.get() == customer\_username\_list[index]:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username exists')

self.errorMsg2.grid(row=11, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

for index in range(len(provider\_username\_list)):

if self.myACd1.get() == provider\_username\_list[index]:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username exists')

self.errorMsg2.grid(row=11, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

for index in range(len(admin\_username\_list)):

if self.myACd1.get() == admin\_username\_list[index]:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username exists')

self.errorMsg2.grid(row=11, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

#check phone valid

if self.myACd2.get() and self.myACd2.get() != str(customer\_phone\_list[self.current\_userID-1]):

if self.myACd2.get().isdecimal():

if len(self.myACd2.get()) != 8:

self.errorMsg3 = tk.Label(self.myAC\_privacy, text='Invalid phone no.')

self.errorMsg3.grid(row=12, column=1,columnspan =3, sticky='w')

self.errorMsg3.config(fg='red')

updateMyACvalid = False

else:

#check phone duplicate

for index in range(len(customer\_phone\_list)):

if int(self.myACd2.get()) == customer\_phone\_list[index]:

self.errorMsg3 = tk.Label(self.myAC\_privacy, text='Phone no. exists')

self.errorMsg3.grid(row=12, column=1,columnspan =3, sticky='w')

self.errorMsg3.config(fg='red')

updateMyACvalid = False

else:

self.errorMsg3 = tk.Label(self.myAC\_privacy, text='Invalid phone no.')

self.errorMsg3.grid(row=12, column=1,columnspan =3, sticky='w')

self.errorMsg3.config(fg='red')

updateMyACvalid = False

#check email valid

if self.myACd3.get() and self.myACd3.get() != customer\_email\_list[self.current\_userID-1]:

if '@'not in self.myACd3.get() or '.'not in self.myACd3.get() or len(self.myACd3.get()) > 40:

self.errorMsg4 = tk.Label(self.myAC\_privacy, text='Invalid email')

self.errorMsg4.grid(row=13, column=1,columnspan =3, sticky='w')

self.errorMsg4.config(fg='red')

updateMyACvalid = False

#check email duplicate

for index in range(len(customer\_email\_list)):

if self.myACd3.get() == customer\_email\_list[index]:

self.errorMsg4 = tk.Label(self.myAC\_privacy, text='Email exists')

self.errorMsg4.grid(row=13, column=1,columnspan =3, sticky='w')

self.errorMsg4.config(fg='red')

updateMyACvalid = False

elif self.current\_userType == 'provider':

#check empty value

if not self.myACd0.get() or not self.myACd1.get() or not self.myACd3.get() or not self.myACd4.get():

self.errorMsg0 = tk.Label(self.myAC\_privacy, text='Empty value!')

self.errorMsg0.grid(row=13, column=1,columnspan =3, sticky='w')

self.errorMsg0.config(fg='red')

updateMyACvalid = False

#check name valid

nameValid = True

if self.myACd0.get():

for letter in self.myACd0.get():

if not letter.isalpha() and letter not in ' ':

nameValid = False

if len(self.myACd0.get()) > 30:

nameValid = False

if not nameValid:

self.errorMsg1 = tk.Label(self.myAC\_privacy, text='Invalid name')

self.errorMsg1.grid(row=14, column=1,columnspan =3, sticky='w')

self.errorMsg1.config(fg='red')

updateMyACvalid = False

#check username valid

usernameValid = True

if self.myACd1.get() and self.myACd1.get() != provider\_username\_list[self.current\_userID-1]:

for letter in self.myACd1.get():

if len(self.myACd1.get()) < 6 or len(self.myACd1.get()) > 20 or (not letter.isalpha() and not letter.isdecimal() and letter not in '\_'):

usernameValid = False

if not usernameValid:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username should contain 6-20 characters, only in letters, numbers, underscore(\_)')

self.errorMsg2.grid(row=15, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

#check username duplicate

for index in range(len(customer\_username\_list)):

if self.myACd1.get() == customer\_username\_list[index]:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username exists')

self.errorMsg2.grid(row=15, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

for index in range(len(provider\_username\_list)):

if self.myACd1.get() == provider\_username\_list[index]:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username exists')

self.errorMsg2.grid(row=15, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

for index in range(len(admin\_username\_list)):

if self.myACd1.get() == admin\_username\_list[index]:

self.errorMsg2 = tk.Label(self.myAC\_privacy, text='Username exists')

self.errorMsg2.grid(row=15, column=1,columnspan =3, sticky='w')

self.errorMsg2.config(fg='red')

updateMyACvalid = False

#check phone valid

if self.myACd3.get() and self.myACd3.get() != str(provider\_phone\_list[self.current\_userID-1]):

if self.myACd3.get().isdecimal():

if len(self.myACd3.get()) != 8:

self.errorMsg3 = tk.Label(self.myAC\_privacy, text='Invalid phone no.')

self.errorMsg3.grid(row=16, column=1,columnspan =3, sticky='w')

self.errorMsg3.config(fg='red')

updateMyACvalid = False

else:

#check phone duplicate

for index in range(len(provider\_phone\_list)):

if int(self.myACd3.get()) == provider\_phone\_list[index]:

self.errorMsg3 = tk.Label(self.myAC\_privacy, text='Phone no. exists')

self.errorMsg3.grid(row=16, column=1,columnspan =3, sticky='w')

self.errorMsg3.config(fg='red')

updateMyACvalid = False

else:

self.errorMsg3 = tk.Label(self.myAC\_privacy, text='Invalid phone no.')

self.errorMsg3.grid(row=16, column=1,columnspan =3, sticky='w')

self.errorMsg3.config(fg='red')

updateMyACvalid = False

#check email valid

if self.myACd4.get() and self.myACd4.get() != provider\_email\_list[self.current\_userID-1]:

if '@'not in self.myACd4.get() or '.'not in self.myACd4.get() or len(self.myACd4.get()) > 40:

self.errorMsg4 = tk.Label(self.myAC\_privacy, text='Invalid email')

self.errorMsg4.grid(row=17, column=1,columnspan =3, sticky='w')

self.errorMsg4.config(fg='red')

updateMyACvalid = False

#check email duplicate

for index in range(len(provider\_email\_list)):

if self.myACd4.get() == provider\_email\_list[index]:

self.errorMsg4 = tk.Label(self.myAC\_privacy, text='Email exists')

self.errorMsg4.grid(row=17, column=1,columnspan =3, sticky='w')

self.errorMsg4.config(fg='red')

updateMyACvalid = False

#check company name, position, and address valid

if self.myACd5.get():

if not self.myACd6.get() or not self.myACd7.get():

self.errorMsg5 = tk.Label(self.myAC\_privacy, text='You must enter your position and company address if you entered company name')

self.errorMsg5.grid(row=18, column=1,columnspan =3, sticky='w')

self.errorMsg5.config(fg='red')

updateMyACvalid = False

else:

if self.myACd6.get() or self.myACd7.get():

self.errorMsg5 = tk.Label(self.myAC\_privacy, text='You must enter your company name first')

self.errorMsg5.grid(row=18, column=1,columnspan =3, sticky='w')

self.errorMsg5.config(fg='red')

updateMyACvalid = False

if updateMyACvalid:

self.mysql\_updatePrivacy()

def mysql\_updatePrivacy(self):

'''

update user's personal information in Privacy to mysql

'''

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

if self.current\_userType == 'customer':

try:

#get user's gender

if self.v0.get() == 1:

gender = 'M'

else:

gender = 'F'

mycursor = connection.cursor()

mycursor.execute("Update customer set name = %s where customer\_id = %s", (self.myACd0.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update customer set username = %s where customer\_id = %s", (self.myACd1.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update customer set gender = %s where customer\_id = %s", (gender, self.current\_userID))

connection.commit()

mycursor.execute("Update customer set phone = %s where customer\_id = %s", (self.myACd2.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update customer set email = %s where customer\_id = %s", (self.myACd3.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update customer set address = %s where customer\_id = %s", (self.myACd4.get(), self.current\_userID))

connection.commit()

#create and display label after sql successfully run

self.savedLabel = tk.Label(self.myAC\_privacy, text = 'Saved')

self.savedLabel.grid(row = 8, column = 1)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

#rebuild frame

self.build\_lookforCases()

self.show\_page(self.myAC)

elif self.current\_userType == 'provider':

try:

#get user's gender

if self.v0.get() == 1:

gender = 'M'

else:

gender = 'F'

mycursor = connection.cursor()

mycursor.execute("Update provider set name = %s where provider\_id = %s", (self.myACd0.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update provider set username = %s where provider\_id = %s", (self.myACd1.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update provider set introduction = %s where provider\_id = %s", (self.myACd2.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update provider set gender = %s where provider\_id = %s", (gender, self.current\_userID))

connection.commit()

mycursor.execute("Update provider set phone = %s where provider\_id = %s", (self.myACd3.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update provider set email = %s where provider\_id = %s", (self.myACd4.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update provider set company\_name = %s where provider\_id = %s", (self.myACd5.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update provider set company\_position = %s where provider\_id = %s", (self.myACd6.get(), self.current\_userID))

connection.commit()

mycursor.execute("Update provider set company\_address = %s where provider\_id = %s", (self.myACd7.get(), self.current\_userID))

connection.commit()

self.savedLabel = tk.Label(self.myAC\_privacy, text = 'Saved')

self.savedLabel.grid(row = 11, column = 1)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

def mysql\_deleteUsers(self, table, ID, row):

'''

for admin to delete users that do not contain cases with status Processing

'''

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

if table == 'customer':

try:

mycursor = connection.cursor()

mycursor.execute("Delete from customer where customer\_id = %s", (ID, ))

mycursor.execute("Update cases set customer\_id = %s, status = %s where customer\_id = %s", (0, 'Cancelled', ID))

mycursor.execute("Update cases\_application set customer\_id = %s, admin\_status = %s, status = %s where customer\_id = %s", (0,'Declined','cCancelled',ID))

for e in range(ID+1, len(customer\_id\_list)+1):

mycursor.execute("Update customer set customer\_id = %s where customer\_id = %s", (e-1, e))

connection.commit()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

elif table == 'provider':

try:

mycursor = connection.cursor()

mycursor.execute("Delete from provider where provider\_id = %s", (ID, ))

mycursor.execute("Update cases set provider\_id = %s where provider\_id = %s", (0, ID))

mycursor.execute("Update cases\_application set provider\_id = %s, admin\_status = %s, status = %s where provider\_id = %s", (0,'Declined','pCancelled',ID))

for e in range(ID+1, len(provider\_id\_list)+1):

mycursor.execute("Update provider set provider\_id = %s where provider\_id = %s", (e-1, e))

connection.commit()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

#rebuild frames

self.build\_lookforCases()

self.build\_myAC()

def mysql\_deleteCases(self, ID, row):

'''

cancel cases or cases application

'''

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

if self.current\_userType == 'customer':

try:

mycursor = connection.cursor()

mycursor.execute("Update cases set status = %s where case\_id = %s", ('Cancelled', ID))

mycursor.execute("Update cases\_application set status = %s where case\_id = %s", ('cCancelled', ID))

connection.commit()

#rebuild frame

self.build\_lookforCases()

self.build\_myAC()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

elif self.current\_userType == 'provider':

try:

mycursor = connection.cursor()

mycursor.execute("Update cases\_application set status = %s where casesApplication\_id = %s", ('pCancelled', ID))

connection.commit()

#rebuild frame

self.build\_myAC()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

def mysql\_acceptCasesApply(self, ID, row):

'''

for customer to accept cases application

'''

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("Update cases\_application set status = %s where casesApplication\_id = %s", ('Accepted', ID))

connection.commit()

mycursor.execute("Select case\_id, provider\_id, estimated\_price from cases\_application where casesApplication\_id = %s", (ID, ))

list = mycursor.fetchone()

caseID = list[0]

providerID = list[1]

price = list[2]

mycursor.execute("Update cases\_application set status = %s where case\_id = %s and casesApplication\_id != %s", ('Declined', caseID, ID))

connection.commit()

mycursor.execute("Update cases set status = %s, provider\_id = %s, price = %s where case\_id = %s and provider\_id != %s", ('Processing', providerID, price, caseID, ID))

connection.commit()

#rebuild frame

self.build\_lookforCases()

self.build\_myAC()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

def mysql\_updateStatus(self, table, ID, original\_vars, option\_vars, adminOriginPrice, adminNewPrice):

'''

update changed status

'''

#set var

changedID = []

changed\_vars = []

changeValid = False

#get changed status and it's ID

if original\_vars != option\_vars:

for v in range(len(original\_vars)):

if original\_vars[v] != option\_vars[v].get():

changedID.append(ID[v])

changed\_vars.append(option\_vars[v].get())

changeValid = True

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

if table == 'admin\_cases':

#set var

changedPriceID = []

changedPrice = []

priceChangeValid = False

#get changed value

for e in range(int(len(adminNewPrice)/2)):

if adminNewPrice["input" + str(e)].get().isdigit():

if adminOriginPrice[e] != int(adminNewPrice["input" + str(e)].get()):

changedPriceID.append(adminNewPrice["caseID" + str(e)])

changedPrice.append(adminNewPrice["input" + str(e)].get())

priceChangeValid = True

#error checking for price input

else:

errorMsg0 = tk.Label(self.myAC\_cases, text = 'Price must be positive integer')

errorMsg0.grid(row=20, column=5,columnspan=3,sticky='w')

errorMsg0.config(fg='red')

if changeValid:

try:

mycursor = connection.cursor()

for e in range(len(changedID)):

mycursor.execute("Update cases set status = %s where case\_id = %s", (changed\_vars[e], changedID[e]))

connection.commit()

#rebuild frame

self.build\_lookforCases()

self.build\_myAC()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

if priceChangeValid:

try:

mycursor = connection.cursor()

for e in range(len(changedPriceID)):

mycursor.execute("Update cases set price = %s where case\_id = %s", (changedPrice[e], changedPriceID[e]))

connection.commit()

#rebuild frame

self.build\_myAC()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

elif table == 'admin\_case applications':

if changeValid:

try:

mycursor = connection.cursor()

for e in range(len(changedID)):

mycursor.execute("Update cases\_application set admin\_status = %s where casesApplication\_id = %s", (changed\_vars[e], changedID[e]))

connection.commit()

#rebuild frame

self.build\_myAC()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

elif table == 'admin\_providers':

if changeValid:

try:

mycursor = connection.cursor()

for e in range(len(changedID)):

mycursor.execute("Update provider set status = %s where provider\_id = %s", (changed\_vars[e], changedID[e]))

connection.commit()

#rebuild frame

self.build\_myAC()

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

def build\_myAC(self):

'''

build the frame to check user's account details

'''

read\_mysql()

#create My Account frame

self.myAC = tk.Frame(self.cc, bg = 'gray95', width = 100, height = 100)

self.myAC.grid(row = 0, column = 0, sticky = 'nsew')

#create and display notebook

self.nb = ttk.Notebook(self.myAC)

self.nb.pack(expand=1, fill="both")

#display customer data

if self.current\_userType == 'customer':

#create tabs

self.myAC\_myCases = ttk.Frame(self.nb)

self.myAC\_myCasesApplication = ttk.Frame(self.nb)

self.myAC\_privacy = ttk.Frame(self.nb)

#title of tabs

self.nb.add(self.myAC\_myCases, text='Cases')

self.nb.add(self.myAC\_myCasesApplication, text='Cases application')

self.nb.add(self.myAC\_privacy, text='Privacy')

#create error messages

self.errorMsg0 = tk.Label(self.myAC\_privacy)

self.errorMsg1 = tk.Label(self.myAC\_privacy)

self.errorMsg2 = tk.Label(self.myAC\_privacy)

self.errorMsg3 = tk.Label(self.myAC\_privacy)

self.errorMsg4 = tk.Label(self.myAC\_privacy)

self.errorMsg5 = tk.Label(self.myAC\_privacy)

#create label for 'saved'

self.savedLabel = tk.Label(self.myAC\_privacy)

#set var

self.v0 = tk.IntVar()

'''

My Cases

'''

#create data titles

l0 = tk.Label(self.myAC\_myCases, text = 'Case ID')

l1 = tk.Label(self.myAC\_myCases, text = 'Provider ID')

l2 = tk.Label(self.myAC\_myCases, text = 'Category')

l3 = tk.Label(self.myAC\_myCases, text = 'Description')

l4 = tk.Label(self.myAC\_myCases, text = 'Address')

l5 = tk.Label(self.myAC\_myCases, text = 'District')

l6 = tk.Label(self.myAC\_myCases, text = 'Area (foot)')

l7 = tk.Label(self.myAC\_myCases, text = 'Price (HKD)')

l8 = tk.Label(self.myAC\_myCases, text = 'Status')

l9 = tk.Label(self.myAC\_myCases, text = 'Since')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

l6.grid(row = 0, column = 6)

l7.grid(row = 0, column = 7)

l8.grid(row = 0, column = 8)

l9.grid(row = 0, column = 9)

#display all user's cases from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("SELECT \* FROM cases where customer\_id = %s and status != %s", (self.current\_userID, 'Cancelled'))

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#set var

customerCase\_dict = {}

counter = 0

for row in result:

if row[9] == 'Verifying' or row[9] == 'Posted':

customerCase\_dict["caseID" + str(counter)] = row[0]

else:

customerCase\_dict["caseID" + str(counter)] = 0

counter += 1

data\_label = tk.Label(self.myAC\_myCases, text = row[0])

data\_label.grid(row = counter, column = 0)

if row[2] == 0:

data\_label = tk.Label(self.myAC\_myCases, text = 'None')

else:

data\_label = tk.Label(self.myAC\_myCases, text = row[2])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_myCases, text = row[3])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_myCases, text = row[4])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_myCases, text = row[5])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_myCases, text = row[6])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_myCases, text = row[7])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_myCases, text = row[8])

data\_label.grid(row = counter, column = 7)

statusL = tk.Label(self.myAC\_myCases, text = row[9])

statusL.grid(row = counter, column = 8)

data\_label = tk.Label(self.myAC\_myCases, text = row[10])

data\_label.grid(row = counter, column = 9)

#create cancel button

for e in range(len(customerCase\_dict)):

if customerCase\_dict["caseID" + str(e)] != 0:

customerCase\_dict["cancelB" + str(e)] = tk.Button(self.myAC\_myCases, text = 'Cancel', command = lambda e=e:self.mysql\_deleteCases(customerCase\_dict["caseID" + str(e)], e+1))

customerCase\_dict["cancelB" + str(e)].grid(row = e+1, column = 10, padx = 10)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_myCases, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+1, column = 12, pady = 3)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Cases Application

'''

#create data titles

l0 = tk.Label(self.myAC\_myCasesApplication, text = 'Case ID')

l1 = tk.Label(self.myAC\_myCasesApplication, text = 'Provider ID')

l2 = tk.Label(self.myAC\_myCasesApplication, text = 'Description')

l3 = tk.Label(self.myAC\_myCasesApplication, text = 'Estimated price')

l4 = tk.Label(self.myAC\_myCasesApplication, text = 'Status')

l5 = tk.Label(self.myAC\_myCasesApplication, text = 'Apply date')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

#display all user's cases from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("SELECT \* FROM cases\_application where customer\_id = %s and admin\_status = %s and (status = %s or status = %s) order by case\_id, provider\_id", (self.current\_userID,'Accepted','Accepted','Waiting'))

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#set var

customerCaseApply\_dict = {}

counter = 0

for row in result:

if row[7] == 'Waiting':

customerCaseApply\_dict["caseApplicationID" + str(counter)] = row[0]

else:

customerCaseApply\_dict["caseApplicationID" + str(counter)] = 0

counter += 1

data\_label = tk.Label(self.myAC\_myCasesApplication, text = row[1])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_myCasesApplication, text = row[3])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_myCasesApplication, text = row[4])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_myCasesApplication, text = row[5])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_myCasesApplication, text = row[7])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_myCasesApplication, text = row[8])

data\_label.grid(row = counter, column = 5)

#create accept button

for e in range(len(customerCaseApply\_dict)):

if customerCaseApply\_dict["caseApplicationID" + str(e)] != 0:

customerCaseApply\_dict["acceptB" + str(e)] = tk.Button(self.myAC\_myCasesApplication, text = 'Accept', command = lambda e=e:self.mysql\_acceptCasesApply(customerCaseApply\_dict["caseApplicationID" + str(e)],e+1))

customerCaseApply\_dict["acceptB" + str(e)].grid(row = e+1, column = 6, padx = 10)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_myCasesApplication, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+1, column = 7, pady = 3)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Privacy

'''

#create data titles

l0 = tk.Label(self.myAC\_privacy, text = 'Customer ID')

l1 = tk.Label(self.myAC\_privacy, text = 'Name\*')

l2 = tk.Label(self.myAC\_privacy, text = 'Username\*')

l3 = tk.Label(self.myAC\_privacy, text = 'Gender')

l4 = tk.Label(self.myAC\_privacy, text = 'Phone\*')

l5 = tk.Label(self.myAC\_privacy, text = 'Email\*')

l6 = tk.Label(self.myAC\_privacy, text = 'Address')

l7 = tk.Label(self.myAC\_privacy, text = 'Account since')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 1, column = 0)

l2.grid(row = 2, column = 0)

l3.grid(row = 3, column = 0)

l4.grid(row = 4, column = 0)

l5.grid(row = 5, column = 0)

l6.grid(row = 6, column = 0)

l7.grid(row = 7, column = 0)

#create widgets

userID = tk.Label(self.myAC\_privacy, text = self.current\_userID)

self.myACd0 = tk.Entry(self.myAC\_privacy)

self.myACd1 = tk.Entry(self.myAC\_privacy)

self.myACd2 = tk.Entry(self.myAC\_privacy, width = 8)

self.myACd3 = tk.Entry(self.myAC\_privacy, width = 25)

self.myACd4 = tk.Entry(self.myAC\_privacy, width = 40)

dt = tk.Label(self.myAC\_privacy, text = customer\_createdDT\_list[self.current\_userID-1])

#insert user personal info

self.myACd0.insert(tk.END, customer\_name\_list[self.current\_userID-1])

self.myACd1.insert(tk.END, customer\_username\_list[self.current\_userID-1])

genderM = tk.Radiobutton(self.myAC\_privacy, text='Male', var=self.v0, value=1)

genderF = tk.Radiobutton(self.myAC\_privacy, text='Female', var=self.v0, value=2)

if customer\_gender\_list[self.current\_userID-1] == 'M':

self.v0.set(1)

else:

self.v0.set(2)

self.myACd2.insert(tk.END, customer\_phone\_list[self.current\_userID-1])

self.myACd3.insert(tk.END, customer\_email\_list[self.current\_userID-1])

self.myACd4.insert(tk.END, customer\_address\_list[self.current\_userID-1])

#display widgets

userID.grid(row = 0, column = 1,columnspan =2, sticky='w',padx =15)

self.myACd0.grid(row = 1, column = 1,columnspan =2, sticky='w')

self.myACd1.grid(row = 2, column = 1,columnspan =2, sticky='w')

genderM.grid(row = 3, column = 1,columnspan =2, sticky='w')

genderF.grid(row = 3, column = 2,columnspan =2, sticky='w')

self.myACd2.grid(row = 4, column = 1,columnspan =2, sticky='w')

self.myACd3.grid(row = 5, column = 1,columnspan =2, sticky='w')

self.myACd4.grid(row = 6, column = 1,columnspan =2, sticky='w')

dt.grid(row = 7, column = 1,columnspan =2)

#create save button

saveB = tk.Button(self.myAC\_privacy, text='Save', command = self.check\_updatePrivacy)

saveB.grid(row = 8, column = 2)

#display provider data

elif self.current\_userType == 'provider':

#create tabs

self.myAC\_myApplication = ttk.Frame(self.nb)

self.myAC\_myCases = ttk.Frame(self.nb)

self.myAC\_BM = ttk.Frame(self.nb)

self.myAC\_privacy = ttk.Frame(self.nb)

#title of tabs

self.nb.add(self.myAC\_myApplication, text='Application')

self.nb.add(self.myAC\_myCases, text='Cases')

self.nb.add(self.myAC\_BM, text='Building materials')

self.nb.add(self.myAC\_privacy, text='Privacy')

#create error messages

self.errorMsg0 = tk.Label(self.myAC\_privacy)

self.errorMsg1 = tk.Label(self.myAC\_privacy)

self.errorMsg2 = tk.Label(self.myAC\_privacy)

self.errorMsg3 = tk.Label(self.myAC\_privacy)

self.errorMsg4 = tk.Label(self.myAC\_privacy)

self.errorMsg5 = tk.Label(self.myAC\_privacy)

#create label for 'saved'

self.savedLabel = tk.Label(self.myAC\_privacy)

#set var

self.v0 = tk.IntVar()

'''

My Application

'''

#create data titles

l0 = tk.Label(self.myAC\_myApplication, text = 'Case ID')

l1 = tk.Label(self.myAC\_myApplication, text = 'Customer ID')

l2 = tk.Label(self.myAC\_myApplication, text = 'Description')

l3 = tk.Label(self.myAC\_myApplication, text = 'Estimated price')

l4 = tk.Label(self.myAC\_myApplication, text = 'Verification')

l5 = tk.Label(self.myAC\_myApplication, text = 'Case status')

l6 = tk.Label(self.myAC\_myApplication, text = 'Apply date')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

l6.grid(row = 0, column = 6)

#set var

dict = {}

#display all user's cases from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("SELECT \* FROM cases\_application where provider\_id = %s and status != 'pCancelled' order by case\_id asc", (self.current\_userID, ))

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#set var

counter = 0

for row in result:

if row[7] == 'Waiting':

dict["caseApplicationID" + str(counter)] = row[0]

else:

dict["caseApplicationID" + str(counter)] = 0

counter += 1

data\_label = tk.Label(self.myAC\_myApplication, text = row[1])

data\_label.grid(row = counter, column = 0)

if row[2] == 0:

data\_label = tk.Label(self.myAC\_myApplication, text = 'Account deleted')

else:

data\_label = tk.Label(self.myAC\_myApplication, text = row[2])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_myApplication, text = row[4])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_myApplication, text = row[5])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_myApplication, text = row[6])

data\_label.grid(row = counter, column = 4)

if row[7] == 'cCancelled':

data\_label = tk.Label(self.myAC\_myApplication, text = 'Case cancelled')

else:

data\_label = tk.Label(self.myAC\_myApplication, text = row[7])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_myApplication, text = row[8])

data\_label.grid(row = counter, column = 6)

#create cancel button

for e in range(len(dict)):

if dict["caseApplicationID" + str(e)] != 0:

dict["cancelB" + str(e)] = tk.Button(self.myAC\_myApplication, text = 'Cancel', command = lambda e=e:self.mysql\_deleteCases(dict["caseApplicationID" + str(e)], e+1))

dict["cancelB" + str(e)].grid(row = e+1, column = 7, padx = 10)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_myApplication, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+1, column = 8, pady = 3)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

My Cases

'''

#create data titles

l0 = tk.Label(self.myAC\_myCases, text = 'Case ID')

l1 = tk.Label(self.myAC\_myCases, text = 'Customer ID')

l2 = tk.Label(self.myAC\_myCases, text = 'Category')

l3 = tk.Label(self.myAC\_myCases, text = 'Description')

l4 = tk.Label(self.myAC\_myCases, text = 'Address')

l5 = tk.Label(self.myAC\_myCases, text = 'District')

l6 = tk.Label(self.myAC\_myCases, text = 'Area')

l7 = tk.Label(self.myAC\_myCases, text = 'Price')

l8 = tk.Label(self.myAC\_myCases, text = 'Status')

l9 = tk.Label(self.myAC\_myCases, text = 'Since')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

l6.grid(row = 0, column = 6)

l7.grid(row = 0, column = 7)

l8.grid(row = 0, column = 8)

l9.grid(row = 0, column = 9)

#display all user's cases from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("SELECT \* FROM cases where provider\_id = %s", (self.current\_userID, ))

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#set var

counter = 0

for row in result:

counter += 1

data\_label = tk.Label(self.myAC\_myCases, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_myCases, text = row[1])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_myCases, text = row[3])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_myCases, text = row[4])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_myCases, text = row[5])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_myCases, text = row[6])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_myCases, text = row[7])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_myCases, text = row[8])

data\_label.grid(row = counter, column = 7)

data\_label = tk.Label(self.myAC\_myCases, text = row[9])

data\_label.grid(row = counter, column = 8)

data\_label = tk.Label(self.myAC\_myCases, text = row[10])

data\_label.grid(row = counter, column = 9)

if row[9] != 'Done':

orderB = tk.Button(self.myAC\_myCases, text = 'Order materials', command = lambda row=row, counter=counter: self.build\_orderBuildingMaterial(row[0]))

orderB.grid(row = counter, column = 10)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_myCases, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+1, column = 10, pady = 3)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Building materials

'''

#create data titles

num = tk.Label(self.myAC\_BM, text='No.')

caseid = tk.Label(self.myAC\_BM, text='Case ID')

orderid = tk.Label(self.myAC\_BM, text='Order ID')

item = tk.Label(self.myAC\_BM, text='Item')

quantity = tk.Label(self.myAC\_BM, text='Quantity')

price = tk.Label(self.myAC\_BM, text='Price')

totalPrice = tk.Label(self.myAC\_BM, text='Total price')

status = tk.Label(self.myAC\_BM, text='Status')

since = tk.Label(self.myAC\_BM, text='Since')

#display data titles

row = 0

num.grid(row = row, column = 0)

caseid.grid(row = row, column = 1)

orderid.grid(row = row, column = 2)

item.grid(row = row, column = 3)

quantity.grid(row = row, column = 4)

price.grid(row = row, column = 5)

totalPrice.grid(row = row, column = 6)

status.grid(row = row, column = 7)

since.grid(row = row, column = 8)

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

#set var

results = []

mycursor = connection.cursor()

mycursor.execute("SELECT case\_id FROM cases WHERE provider\_id = %s ORDER BY case\_id asc"%(self.current\_userID))

caseIDList = mycursor.fetchall()

for id in caseIDList:

mycursor.execute("SELECT \* FROM building\_material WHERE case\_id = %s ORDER BY order\_id desc, status <> 'pending', status <> 'preparing', status <> 'delivering'"%(id[0]))

results += mycursor.fetchall()

counter = 1

for row in results:

#no.

numL = tk.Label(self.myAC\_BM, text = counter)

numL.grid(row = counter, column = 0)

#case id

caseidL = tk.Label(self.myAC\_BM, text = row[1])

caseidL.grid(row = counter, column = 1)

#order id

orderidL = tk.Label(self.myAC\_BM, text = row[2])

orderidL.grid(row = counter, column = 2)

#item

itemL = tk.Label(self.myAC\_BM, text = row[3])

itemL.grid(row = counter, column = 3)

#quantity

quantityL = tk.Label(self.myAC\_BM, text = row[4])

quantityL.grid(row = counter, column = 4)

#price

priceL = tk.Label(self.myAC\_BM, text = row[5])

priceL.grid(row = counter, column = 5)

#total price

totalPriceL = tk.Label(self.myAC\_BM, text = row[6])

totalPriceL.grid(row = counter, column = 6)

#status

statusL = tk.Label(self.myAC\_BM, text = row[7])

statusL.grid(row = counter, column = 7)

#since

sinceL = tk.Label(self.myAC\_BM, text = row[8])

sinceL.grid(row = counter, column = 8)

#formatting on sale frame

counter += 1

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Privacy

'''

#create data titles

l0 = tk.Label(self.myAC\_privacy, text = 'Provider ID')

l1 = tk.Label(self.myAC\_privacy, text = 'Name\*')

l2 = tk.Label(self.myAC\_privacy, text = 'Username\*')

l3 = tk.Label(self.myAC\_privacy, text = 'Introduction')

l4 = tk.Label(self.myAC\_privacy, text = 'Gender')

l5 = tk.Label(self.myAC\_privacy, text = 'Phone\*')

l6 = tk.Label(self.myAC\_privacy, text = 'Email\*')

l7 = tk.Label(self.myAC\_privacy, text = 'Company name')

l8 = tk.Label(self.myAC\_privacy, text = 'Company title')

l9 = tk.Label(self.myAC\_privacy, text = 'Company address')

l10 = tk.Label(self.myAC\_privacy, text = 'Account since')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 1, column = 0)

l2.grid(row = 2, column = 0)

l3.grid(row = 3, column = 0)

l4.grid(row = 4, column = 0)

l5.grid(row = 5, column = 0)

l6.grid(row = 6, column = 0)

l7.grid(row = 7, column = 0)

l8.grid(row = 8, column = 0)

l9.grid(row = 9, column = 0)

l10.grid(row = 10, column = 0)

#display user ID

userID = tk.Label(self.myAC\_privacy, text = self.current\_userID)

userID.grid(row = 0, column = 1,columnspan =2, sticky='w',padx =15)

#create widgets

self.myACd0 = tk.Entry(self.myAC\_privacy)

self.myACd1 = tk.Entry(self.myAC\_privacy)

self.myACd2 = tk.Entry(self.myAC\_privacy, width = 40)

self.myACd3 = tk.Entry(self.myAC\_privacy, width = 8)

self.myACd4 = tk.Entry(self.myAC\_privacy, width = 25)

self.myACd5 = tk.Entry(self.myAC\_privacy, width = 30)

self.myACd6 = tk.Entry(self.myAC\_privacy, width = 30)

self.myACd7 = tk.Entry(self.myAC\_privacy, width = 40)

dt = tk.Label(self.myAC\_privacy, text = provider\_createdDT\_list[self.current\_userID-1])

#insert user's personal info

self.myACd0.insert(tk.END, provider\_name\_list[self.current\_userID-1])

self.myACd1.insert(tk.END, provider\_username\_list[self.current\_userID-1])

self.myACd2.insert(tk.END, provider\_intro\_list[self.current\_userID-1])

genderM = tk.Radiobutton(self.myAC\_privacy, text='Male', var=self.v0, value=1)

genderF = tk.Radiobutton(self.myAC\_privacy, text='Female', var=self.v0, value=2)

if provider\_gender\_list[self.current\_userID-1] == 'M':

self.v0.set(1)

else:

self.v0.set(2)

self.myACd3.insert(tk.END, provider\_phone\_list[self.current\_userID-1])

self.myACd4.insert(tk.END, provider\_email\_list[self.current\_userID-1])

self.myACd5.insert(tk.END, provider\_companyName\_list[self.current\_userID-1])

self.myACd6.insert(tk.END, provider\_companyPosition\_list[self.current\_userID-1])

self.myACd7.insert(tk.END, provider\_companyAddress\_list[self.current\_userID-1])

#display widgets

self.myACd0.grid(row = 1, column = 1,columnspan =2, sticky='w')

self.myACd1.grid(row = 2, column = 1,columnspan =2, sticky='w')

self.myACd2.grid(row = 3, column = 1,columnspan =2, sticky='w')

genderM.grid(row = 4, column = 1,columnspan =2, sticky='w')

genderF.grid(row = 4, column = 2,columnspan =2, sticky='w')

self.myACd3.grid(row = 5, column = 1,columnspan =2, sticky='w')

self.myACd4.grid(row = 6, column = 1,columnspan =2, sticky='w')

self.myACd5.grid(row = 7, column = 1,columnspan =2, sticky='w')

self.myACd6.grid(row = 8, column = 1,columnspan =2, sticky='w')

self.myACd7.grid(row = 9, column = 1,columnspan =2, sticky='w')

dt.grid(row = 10, column = 1,columnspan =2)

#create save button

saveB = tk.Button(self.myAC\_privacy, text='Save', command = self.check\_updatePrivacy)

saveB.grid(row = 11, column = 2)

#display admin data

elif self.current\_userType == 'admin':

#create tabs

self.myAC\_cases = ttk.Frame(self.nb)

self.myAC\_applications = ttk.Frame(self.nb)

self.myAC\_customers = ttk.Frame(self.nb)

self.myAC\_providers = ttk.Frame(self.nb)

self.myAC\_privacy = ttk.Frame(self.nb)

#title of tabs

self.nb.add(self.myAC\_cases, text='Cases')

self.nb.add(self.myAC\_applications, text='Applications')

self.nb.add(self.myAC\_customers, text='Customers')

self.nb.add(self.myAC\_providers, text='Providers')

self.nb.add(self.myAC\_privacy, text='Privacy')

'''

Cases

'''

#create data titles

l0 = tk.Label(self.myAC\_cases, text = 'ID')

l1 = tk.Label(self.myAC\_cases, text = 'Customer ID')

l2 = tk.Label(self.myAC\_cases, text = 'Provider ID')

l3 = tk.Label(self.myAC\_cases, text = 'Category')

l4 = tk.Label(self.myAC\_cases, text = 'Description')

l5 = tk.Label(self.myAC\_cases, text = 'Address')

l6 = tk.Label(self.myAC\_cases, text = 'District')

l7 = tk.Label(self.myAC\_cases, text = 'Area')

l8 = tk.Label(self.myAC\_cases, text = 'Price')

l9 = tk.Label(self.myAC\_cases, text = 'Status')

l10 = tk.Label(self.myAC\_cases, text = 'Since')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

l6.grid(row = 0, column = 6)

l7.grid(row = 0, column = 7)

l8.grid(row = 0, column = 8)

l9.grid(row = 0, column = 9)

l10.grid(row = 0, column = 10)

#display all cases from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("SELECT \* FROM cases order by status <> 'Verifying', status <> 'Processing', status <> 'Posted', status <> 'Done',status <> 'Declined', case\_id asc")

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#get total page no. and no. of data in last page

if caseNum > 12:

remainder = caseNum % 12

if remainder == 0:

pageNum = caseNum//12

else:

pageNum = caseNum//12 + 1

else:

pageNum = 1

remainder = caseNum

def cases\_nextPage(result, caseNum, pageNum, page):

'''

button to show next page

'''

#destroy previous data

for e in range(1,self.myAC\_cases.grid\_size()[1]):

l=list(self.myAC\_cases.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page + 1

counter = 0

value = -1

startValue = (page-1)\*12

endValue = page\*12

#set dict & list

adminPricedict = {}

cases\_original\_vars = []

cases\_option\_vars = []

cases\_originalPrice = []

ID = []

#display case data

for row in result[startValue: endValue]:

counter += 1

self.statusChoice = tk.StringVar()

data\_label = tk.Label(self.myAC\_cases, text = row[0])

data\_label.grid(row = counter, column = 0)

if row[1] == 0:

data\_label = tk.Label(self.myAC\_cases, text = 'None')

else:

data\_label = tk.Label(self.myAC\_cases, text = row[1])

data\_label.grid(row = counter, column = 1)

if row[2] == 0:

data\_label = tk.Label(self.myAC\_cases, text = 'None')

else:

data\_label = tk.Label(self.myAC\_cases, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_cases, text = row[3])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_cases, text = row[4])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_cases, text = row[5])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_cases, text = row[6])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_cases, text = row[7])

data\_label.grid(row = counter, column = 7)

if row[9] == 'Processing':

value = value + 1

adminPricedict["caseID" + str(value)] = row[0]

cases\_originalPrice.append(row[8])

adminPricedict["input" + str(value)] = tk.Entry(self.myAC\_cases, width = 10)

adminPricedict["input" + str(value)].insert(tk.END, row[8])

adminPricedict["input" + str(value)].grid(row = counter, column = 8)

else:

data\_label = tk.Label(self.myAC\_cases, text = row[8])

data\_label.grid(row = counter, column = 8)

if row[9] != 'Cancelled':

ID.append(row[0])

if row[2] == 0:

#create option menu for unverified cases

optionList\_noProvider = ('Verifying', 'Posted', 'Declined')

omenu = tk.OptionMenu(self.myAC\_cases, self.statusChoice, \*optionList\_noProvider)

if row[9] == 'Verifying':

self.statusChoice.set(optionList\_noProvider[0])

elif row[9] == 'Posted':

self.statusChoice.set(optionList\_noProvider[1])

elif row[9] == 'Declined':

self.statusChoice.set(optionList\_noProvider[2])

else:

#create option menu verified cases

optionList\_yesProvider = ('Posted', 'Processing', 'Done', 'Declined')

omenu = tk.OptionMenu(self.myAC\_cases, self.statusChoice, \*optionList\_yesProvider)

if row[9] == 'Posted':

self.statusChoice.set(optionList\_yesProvider[0])

elif row[9] == 'Processing':

self.statusChoice.set(optionList\_yesProvider[1])

elif row[9] == 'Done':

self.statusChoice.set(optionList\_yesProvider[2])

elif row[9] == 'Declined':

self.statusChoice.set(optionList\_yesProvider[3])

omenu.grid(row = counter, column = 9)

#get original and new status

cases\_original\_vars.append(row[9])

cases\_option\_vars.append(self.statusChoice)

else:

data\_label = tk.Label(self.myAC\_cases, text = 'Cancelled')

data\_label.grid(row = counter, column = 9)

data\_label = tk.Label(self.myAC\_cases, text = row[10])

data\_label.grid(row = counter, column = 10)

if page != pageNum:

#create button for next page

nextpage = tk.Button(self.myAC\_cases, text = 'Next page', command = lambda: cases\_nextPage(result, caseNum, pageNum, page))

nextpage.grid(row = 14, column = 12)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_cases, text = 'Previous page', command = lambda: cases\_previousPage(result, caseNum, pageNum, page))

previouspage.grid(row = 14, column = 11)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_cases, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 13, pady = 3)

#create save button

save = tk.Button(self.myAC\_cases, text = 'Save', command = lambda: self.mysql\_updateStatus('admin\_cases',ID,cases\_original\_vars,cases\_option\_vars,cases\_originalPrice,adminPricedict))

save.grid(row = caseNum+3, column = 13)

def cases\_previousPage(result, caseNum, pageNum, page):

'''

button to show previous page

'''

#destroy previous data

for e in range(1,self.myAC\_cases.grid\_size()[1]):

l=list(self.myAC\_cases.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page - 1

counter = 0

value = -1

startValue = (page-1)\*12

endValue = page\*12

#set dict & list

adminPricedict = {}

cases\_original\_vars = []

cases\_option\_vars = []

cases\_originalPrice = []

ID = []

#display case data

for row in result[startValue: endValue]:

counter += 1

self.statusChoice = tk.StringVar()

if row[0] == 0:

data\_label = tk.Label(self.myAC\_cases, text = 'None')

else:

data\_label = tk.Label(self.myAC\_cases, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_cases, text = row[1])

data\_label.grid(row = counter, column = 1)

if row[2] == 0:

data\_label = tk.Label(self.myAC\_cases, text = 'None')

else:

data\_label = tk.Label(self.myAC\_cases, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_cases, text = row[3])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_cases, text = row[4])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_cases, text = row[5])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_cases, text = row[6])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_cases, text = row[7])

data\_label.grid(row = counter, column = 7)

if row[9] == 'Processing':

value = value + 1

adminPricedict["caseID" + str(value)] = row[0]

cases\_originalPrice.append(row[8])

adminPricedict["input" + str(value)] = tk.Entry(self.myAC\_cases, width = 10)

adminPricedict["input" + str(value)].insert(tk.END, row[8])

adminPricedict["input" + str(value)].grid(row = counter, column = 8)

else:

data\_label = tk.Label(self.myAC\_cases, text = row[8])

data\_label.grid(row = counter, column = 8)

if row[9] != 'Cancelled':

ID.append(row[0])

if row[2] == 0:

#create option menu for unverified cases

optionList\_noProvider = ('Verifying', 'Posted', 'Declined')

omenu = tk.OptionMenu(self.myAC\_cases, self.statusChoice, \*optionList\_noProvider)

if row[9] == 'Verifying':

self.statusChoice.set(optionList\_noProvider[0])

elif row[9] == 'Posted':

self.statusChoice.set(optionList\_noProvider[1])

elif row[9] == 'Declined':

self.statusChoice.set(optionList\_noProvider[2])

else:

#create option menu verified cases

optionList\_yesProvider = ('Posted', 'Processing', 'Done', 'Declined')

omenu = tk.OptionMenu(self.myAC\_cases, self.statusChoice, \*optionList\_yesProvider)

if row[9] == 'Posted':

self.statusChoice.set(optionList\_yesProvider[0])

elif row[9] == 'Processing':

self.statusChoice.set(optionList\_yesProvider[1])

elif row[9] == 'Done':

self.statusChoice.set(optionList\_yesProvider[2])

elif row[9] == 'Declined':

self.statusChoice.set(optionList\_yesProvider[3])

omenu.grid(row = counter, column = 9)

#get original and new status

cases\_original\_vars.append(row[9])

cases\_option\_vars.append(self.statusChoice)

else:

data\_label = tk.Label(self.myAC\_cases, text = 'Cancelled')

data\_label.grid(row = counter, column = 9)

data\_label = tk.Label(self.myAC\_cases, text = row[10])

data\_label.grid(row = counter, column = 10)

#create button for next page

nextpage = tk.Button(self.myAC\_cases, text = 'Next page', command = lambda: cases\_nextPage(result, caseNum, pageNum, page))

nextpage.grid(row = 14, column = 12)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_cases, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 13, pady = 3)

#create save button

save = tk.Button(self.myAC\_cases, text = 'Save', command = lambda: self.mysql\_updateStatus('admin\_cases',ID,cases\_original\_vars,cases\_option\_vars,cases\_originalPrice,adminPricedict))

save.grid(row = caseNum+3, column = 13)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_cases, text = 'Previous page', command = lambda: cases\_previousPage(result, caseNum, pageNum, page))

previouspage.grid(row = 14, column = 11)

#set var

page = 0

cases\_nextPage(result, caseNum, pageNum, page)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Applications

'''

#create data titles

l0 = tk.Label(self.myAC\_applications, text = 'ID')

l1 = tk.Label(self.myAC\_applications, text = 'Case ID')

l2 = tk.Label(self.myAC\_applications, text = 'Customer ID')

l3 = tk.Label(self.myAC\_applications, text = 'Provider ID')

l4 = tk.Label(self.myAC\_applications, text = 'Description')

l5 = tk.Label(self.myAC\_applications, text = 'Estimated price')

l6 = tk.Label(self.myAC\_applications, text = 'Verification')

l7 = tk.Label(self.myAC\_applications, text = 'Apply date')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

l6.grid(row = 0, column = 6)

l7.grid(row = 0, column = 7)

#display all cases from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute("SELECT \* FROM cases\_application where status != 'cCancelled' and status != 'pCancelled' order by admin\_status <> 'Verifying', admin\_status <> 'Accepted',casesApplication\_id asc")

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#get total page no. and no. of data in last page

if caseNum > 12:

remainder = caseNum % 12

if remainder == 0:

pageNum = caseNum//12

else:

pageNum = caseNum//12 + 1

else:

pageNum = 1

remainder = caseNum

def caseApply\_nextPage(result, caseNum, pageNum, page):

'''

button to show next page

'''

#destroy previous data

for e in range(1,self.myAC\_applications.grid\_size()[1]):

l=list(self.myAC\_applications.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page + 1

counter = 0

startValue = (page-1)\*12

endValue = page\*12

#set list

caseApply\_original\_vars = []

caseApply\_option\_vars = []

ID = []

#display case data

for row in result[startValue: endValue]:

counter += 1

self.statusChoice = tk.StringVar()

data\_label = tk.Label(self.myAC\_applications, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_applications, text = row[1])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_applications, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_applications, text = row[3])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_applications, text = row[4])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_applications, text = row[5])

data\_label.grid(row = counter, column = 5)

#create option menu

optionList = ('Accepted', 'Verifying', 'Declined')

omenu = tk.OptionMenu(self.myAC\_applications, self.statusChoice, \*optionList)

if row[6] == 'Accepted':

self.statusChoice.set(optionList[0])

elif row[6] == 'Verifying':

self.statusChoice.set(optionList[1])

elif row[6] == 'Declined':

self.statusChoice.set(optionList[2])

omenu.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_applications, text = row[8])

data\_label.grid(row = counter, column = 7)

#get original and new status

ID.append(row[0])

caseApply\_original\_vars.append(row[6])

caseApply\_option\_vars.append(self.statusChoice)

if page != pageNum:

#create button for next page

nextpage = tk.Button(self.myAC\_applications, text = 'Next page', command = lambda: caseApply\_nextPage(result, caseNum, pageNum, page))

nextpage.grid(row = 14, column = 9)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_applications, text = 'Previous page', command = lambda: caseApply\_previousPage(result, caseNum, pageNum, page))

previouspage.grid(row = 14, column = 8)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_applications, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 10, pady = 3)

#create save button

save = tk.Button(self.myAC\_applications, text = 'Save', command = lambda: self.mysql\_updateStatus('admin\_case applications', ID, caseApply\_original\_vars, caseApply\_option\_vars,0,0))

save.grid(row = caseNum+3, column = 10)

def caseApply\_previousPage(result, caseNum, pageNum, page):

'''

button to show previous page

'''

#destroy previous data

for e in range(1,self.myAC\_applications.grid\_size()[1]):

l=list(self.myAC\_applications.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page - 1

counter = 0

startValue = (page-1)\*12

endValue = page\*12

#set list

caseApply\_original\_vars = []

caseApply\_option\_vars = []

ID = []

#display case data

for row in result[startValue: endValue]:

counter += 1

self.statusChoice = tk.StringVar()

data\_label = tk.Label(self.myAC\_applications, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_applications, text = row[1])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_applications, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_applications, text = row[3])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_applications, text = row[4])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_applications, text = row[5])

data\_label.grid(row = counter, column = 5)

#create option menu

optionList = ('Accepted', 'Verifying', 'Declined')

omenu = tk.OptionMenu(self.myAC\_applications, self.statusChoice, \*optionList)

if row[6] == 'Accepted':

self.statusChoice.set(optionList[0])

elif row[6] == 'Verifying':

self.statusChoice.set(optionList[1])

elif row[6] == 'Declined':

self.statusChoice.set(optionList[2])

omenu.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_applications, text = row[8])

data\_label.grid(row = counter, column = 7)

#get original and new status

ID.append(row[0])

caseApply\_original\_vars.append(row[6])

caseApply\_option\_vars.append(self.statusChoice)

#create button for next page

nextpage = tk.Button(self.myAC\_applications, text = 'Next page', command = lambda: caseApply\_nextPage(result, caseNum, pageNum, page))

nextpage.grid(row = 14, column = 9)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_applications, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 10, pady = 3)

#create save button

save = tk.Button(self.myAC\_applications, text = 'Save', command = lambda: self.mysql\_updateStatus('admin\_case applications', ID, caseApply\_original\_vars, caseApply\_option\_vars,0,0))

save.grid(row = caseNum+3, column = 10)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_applications, text = 'Previous page', command = lambda: caseApply\_previousPage(result, caseNum, pageNum, page))

previouspage.grid(row = 14, column = 8)

#set var

page = 0

caseApply\_nextPage(result, caseNum, pageNum, page)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Customers

'''

#create data title labels

l0 = tk.Label(self.myAC\_customers, text = 'ID')

l1 = tk.Label(self.myAC\_customers, text = 'Name')

l2 = tk.Label(self.myAC\_customers, text = 'Username')

l3 = tk.Label(self.myAC\_customers, text = 'Gender')

l4 = tk.Label(self.myAC\_customers, text = 'Phone')

l5 = tk.Label(self.myAC\_customers, text = 'Email')

l6 = tk.Label(self.myAC\_customers, text = 'Address')

l7 = tk.Label(self.myAC\_customers, text = 'Since')

#display data title labels

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

l6.grid(row = 0, column = 6)

l7.grid(row = 0, column = 7)

#display all cases with status 'Posted' from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute(("SELECT customer\_id FROM cases WHERE status = %s"), ('Processing', ))

processingID = mycursor.fetchall()

mycursor.execute("SELECT \* FROM customer")

result = mycursor.fetchall()

caseNum = mycursor.rowcount

#get total page no. and no. of data in last page

if caseNum > 15:

remainder = caseNum % 15

if remainder == 0:

pageNum = caseNum//15

else:

pageNum = caseNum//15 + 1

else:

pageNum = 1

remainder = caseNum

def customers\_nextPage(result, caseNum, pageNum, page):

'''

button to show next page

'''

#destroy previous data

for e in range(1,self.myAC\_customers.grid\_size()[1]):

l=list(self.myAC\_customers.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page + 1

counter = 0

startValue = (page-1)\*15

endValue = page\*15

#display customer data

for row in result[startValue: endValue]:

deleteValid = True

counter += 1

#check if delete valid

for (e,) in processingID:

if int(e,) == row[0]:

deleteValid = False

data\_label = tk.Label(self.myAC\_customers, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_customers, text = row[1])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_customers, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_customers, text = row[4])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_customers, text = row[5])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_customers, text = row[6])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_customers, text = row[7])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_customers, text = row[8])

data\_label.grid(row = counter, column = 7)

#create delete button

if deleteValid:

deleteB = tk.Button(self.myAC\_customers, text = 'Delete', command = lambda row=row, counter=counter: self.mysql\_deleteUsers('customer', row[0], counter))

deleteB.grid(row = counter, column = 8, padx = 10)

if page != pageNum:

#create button for next page

nextpage = tk.Button(self.myAC\_customers, text = 'Next page', command = lambda: customers\_nextPage(result, caseNum, pageNum, page))

nextpage.grid(row = 17, column = 9)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_customers, text = 'Previous page', command = lambda: customers\_previousPage(result, caseNum, pageNum, page))

previouspage.grid(row = 17, column = 8)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_customers, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 10, pady = 3)

def customers\_previousPage(result, caseNum, pageNum, page):

'''

button to show previous page

'''

#destroy previous data

for e in range(1,self.myAC\_customers.grid\_size()[1]):

l=list(self.myAC\_customers.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page - 1

counter = 0

startValue = (page-1)\*15

endValue = page\*15

#display customer data

for row in result[startValue: endValue]:

deleteValid = True

counter += 1

#check if delete valid

for (e,) in processingID:

if int(e,) == row[0]:

deleteValid = False

data\_label = tk.Label(self.myAC\_customers, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_customers, text = row[1])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_customers, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_customers, text = row[4])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_customers, text = row[5])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_customers, text = row[6])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_customers, text = row[7])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_customers, text = row[8])

data\_label.grid(row = counter, column = 7)

#create delete button

if deleteValid:

deleteB = tk.Button(self.myAC\_customers, text = 'Delete', command = lambda row=row, counter=counter: self.mysql\_deleteUsers('customer', row[0], counter))

deleteB.grid(row = counter, column = 8, padx = 10)

#create button for next page

nextpage = tk.Button(self.myAC\_customers, text = 'Next page', command = lambda: customers\_nextPage(result, caseNum, pageNum, page))

nextpage.grid(row = 17, column = 9)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_customers, text = str(caseNum)+' results')

resultNum.grid(row = caseNum+2, column = 10, pady = 3)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_customers, text = 'Previous page', command = lambda: customers\_previousPage(result, caseNum, pageNum, page))

previouspage.grid(row = 17, column = 8)

#set var

page = 0

customers\_nextPage(result, caseNum, pageNum, page)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Providers

'''

l0 = tk.Label(self.myAC\_providers, text = 'ID')

l1 = tk.Label(self.myAC\_providers, text = 'Name')

l2 = tk.Label(self.myAC\_providers, text = 'Username')

l3 = tk.Label(self.myAC\_providers, text = 'Intro')

l4 = tk.Label(self.myAC\_providers, text = 'Gender')

l5 = tk.Label(self.myAC\_providers, text = 'Phone')

l6 = tk.Label(self.myAC\_providers, text = 'Email')

l7 = tk.Label(self.myAC\_providers, text = 'Company name')

l8 = tk.Label(self.myAC\_providers, text = 'Position')

l9 = tk.Label(self.myAC\_providers, text = 'Company address')

l10 = tk.Label(self.myAC\_providers, text = 'Status')

l11 = tk.Label(self.myAC\_providers, text = 'Since')

#display data title labels

l0.grid(row = 0, column = 0)

l1.grid(row = 0, column = 1)

l2.grid(row = 0, column = 2)

l3.grid(row = 0, column = 3)

l4.grid(row = 0, column = 4)

l5.grid(row = 0, column = 5)

l6.grid(row = 0, column = 6)

l7.grid(row = 0, column = 7)

l8.grid(row = 0, column = 8)

l9.grid(row = 0, column = 9)

l10.grid(row = 0, column = 10)

l11.grid(row = 0, column = 11)

#display all provider data from mysql

connection = mysql.connector.connect(host='localhost',

database='online\_shopping',

user='root',

password='')

try:

mycursor = connection.cursor()

mycursor.execute(("SELECT provider\_id FROM cases WHERE status = %s"), ('Processing', ))

processingID = mycursor.fetchall()

mycursor.execute("SELECT \* FROM provider order by status <> 'Verifying', provider\_id asc")

result = mycursor.fetchall()

providerNum = mycursor.rowcount

#get total page no. and no. of data in last page

if providerNum > 12:

remainder = providerNum % 12

if remainder == 0:

pageNum = providerNum//12

else:

pageNum = providerNum//12 + 1

else:

pageNum = 1

remainder = providerNum

def providers\_nextPage(result, providerNum, pageNum, page):

'''

button to show next page

'''

#destroy previous data

for e in range(1,self.myAC\_providers.grid\_size()[1]):

l=list(self.myAC\_providers.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page + 1

counter = 0

startValue = (page-1)\*12

endValue = page\*12

#set list

original\_vars = []

option\_vars = []

ID = []

#display case data

for row in result[startValue: endValue]:

deleteValid = True

counter += 1

self.statusChoice = tk.StringVar()

#check if delete valid

for (e,) in processingID:

if int(e,) == row[0]:

deleteValid = False

data\_label = tk.Label(self.myAC\_providers, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_providers, text = row[1])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_providers, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_providers, text = row[4])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_providers, text = row[5])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_providers, text = row[6])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_providers, text = row[7])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_providers, text = row[8])

data\_label.grid(row = counter, column = 7)

data\_label = tk.Label(self.myAC\_providers, text = row[9])

data\_label.grid(row = counter, column = 8)

data\_label = tk.Label(self.myAC\_providers, text = row[10])

data\_label.grid(row = counter, column = 9)

#create option menu

optionList = ('Verified', 'Verifying', 'Declined')

omenu = tk.OptionMenu(self.myAC\_providers, self.statusChoice, \*optionList)

if row[11] == 'Verified':

self.statusChoice.set(optionList[0])

elif row[11] == 'Verifying':

self.statusChoice.set(optionList[1])

elif row[11] == 'Declined':

self.statusChoice.set(optionList[2])

omenu.grid(row = counter, column = 10)

data\_label = tk.Label(self.myAC\_providers, text = row[12])

data\_label.grid(row = counter, column = 11)

#create delete button

if deleteValid:

deleteB = tk.Button(self.myAC\_providers, text = 'Delete', command = lambda row=row, counter=counter: self.mysql\_deleteUsers('provider', row[0], counter))

deleteB.grid(row = counter, column = 12, padx = 10)

#get original and new status

ID.append(row[0])

original\_vars.append(row[11])

option\_vars.append(self.statusChoice)

if page != pageNum:

#create button for next page

nextpage = tk.Button(self.myAC\_providers, text = 'Next page', command = lambda: providers\_nextPage(result, providerNum, pageNum, page))

nextpage.grid(row = 14, column = 11)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_providers, text = 'Previous page', command = lambda: providers\_previousPage(result, providerNum, pageNum, page))

previouspage.grid(row = 14, column = 10)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_providers, text = str(providerNum)+' results')

resultNum.grid(row = providerNum, column = 12, pady = 3)

#create save button

save = tk.Button(self.myAC\_providers, text = 'Save', command = lambda: self.mysql\_updateStatus('admin\_providers', ID, original\_vars, option\_vars,0,0))

save.grid(row = providerNum+1, column = 12)

def providers\_previousPage(result, providerNum, pageNum, page):

'''

button to show previous page

'''

#destroy previous data

for e in range(1,self.myAC\_providers.grid\_size()[1]):

l=list(self.myAC\_providers.grid\_slaves(row=e))

for w in l:

w.destroy()

#set var

page = page - 1

counter = 0

startValue = (page-1)\*12

endValue = page\*12

#set list

original\_vars = []

option\_vars = []

ID = []

#display case data

for row in result[startValue: endValue]:

deleteValid = True

counter += 1

self.statusChoice = tk.StringVar()

#check if delete valid

for (e,) in processingID:

if int(e,) == row[0]:

deleteValid = False

data\_label = tk.Label(self.myAC\_providers, text = row[0])

data\_label.grid(row = counter, column = 0)

data\_label = tk.Label(self.myAC\_providers, text = row[1])

data\_label.grid(row = counter, column = 1)

data\_label = tk.Label(self.myAC\_providers, text = row[2])

data\_label.grid(row = counter, column = 2)

data\_label = tk.Label(self.myAC\_providers, text = row[4])

data\_label.grid(row = counter, column = 3)

data\_label = tk.Label(self.myAC\_providers, text = row[5])

data\_label.grid(row = counter, column = 4)

data\_label = tk.Label(self.myAC\_providers, text = row[6])

data\_label.grid(row = counter, column = 5)

data\_label = tk.Label(self.myAC\_providers, text = row[7])

data\_label.grid(row = counter, column = 6)

data\_label = tk.Label(self.myAC\_providers, text = row[8])

data\_label.grid(row = counter, column = 7)

data\_label = tk.Label(self.myAC\_providers, text = row[9])

data\_label.grid(row = counter, column = 8)

data\_label = tk.Label(self.myAC\_providers, text = row[10])

data\_label.grid(row = counter, column = 9)

#create option menu

optionList = ('Verified', 'Verifying', 'Declined')

omenu = tk.OptionMenu(self.myAC\_providers, self.statusChoice, \*optionList)

if row[11] == 'Verified':

self.statusChoice.set(optionList[0])

elif row[11] == 'Verifying':

self.statusChoice.set(optionList[1])

elif row[11] == 'Declined':

self.statusChoice.set(optionList[2])

omenu.grid(row = counter, column = 10)

data\_label = tk.Label(self.myAC\_providers, text = row[12])

data\_label.grid(row = counter, column = 11)

if deleteValid:

deleteB = tk.Button(self.myAC\_providers, text = 'Delete', command = lambda row=row, counter=counter: self.mysql\_deleteUsers('provider', row[0], counter))

deleteB.grid(row = counter, column = 12, padx = 10)

#get original and new status

ID.append(row[0])

original\_vars.append(row[11])

option\_vars.append(self.statusChoice)

#create button for next page

nextpage = tk.Button(self.myAC\_providers, text = 'Next page', command = lambda: providers\_nextPage(result, providerNum, pageNum, page))

nextpage.grid(row = 14, column = 11)

#display total result no. after sql successfully run

resultNum = tk.Label(self.myAC\_providers, text = str(providerNum)+' results')

resultNum.grid(row = providerNum, column = 12, pady = 3)

#create save button

save = tk.Button(self.myAC\_providers, text = 'Save', command = lambda: self.mysql\_updateStatus('admin\_providers', ID, original\_vars, option\_vars,0,0))

save.grid(row = providerNum+1, column = 12)

if page != 1:

#create button for previous page

previouspage = tk.Button(self.myAC\_providers, text = 'Previous page', command = lambda: providers\_previousPage(result, providerNum, pageNum, page))

previouspage.grid(row = 14, column = 10)

#set var

page = 0

providers\_nextPage(result, providerNum, pageNum, page)

#return error message when sql failed to run

except Error as e:

print("Error reading data from MySQL table", e)

connection.close()

'''

Privacy

'''

#create data titles

l0 = tk.Label(self.myAC\_privacy, text = 'Admin ID')

l1 = tk.Label(self.myAC\_privacy, text = 'Name')

l2 = tk.Label(self.myAC\_privacy, text = 'Username')

l3 = tk.Label(self.myAC\_privacy, text = 'Gender')

l4 = tk.Label(self.myAC\_privacy, text = 'Phone')

l5 = tk.Label(self.myAC\_privacy, text = 'Email')

l6 = tk.Label(self.myAC\_privacy, text = 'Account since')

#display data titles

l0.grid(row = 0, column = 0)

l1.grid(row = 1, column = 0)

l2.grid(row = 2, column = 0)

l3.grid(row = 3, column = 0)

l4.grid(row = 4, column = 0)

l5.grid(row = 5, column = 0)

l6.grid(row = 6, column = 0)

#create data labels

userID = tk.Label(self.myAC\_privacy, text = self.current\_userID)

l0 = tk.Label(self.myAC\_privacy, text = admin\_name\_list[self.current\_userID-1])

l1 = tk.Label(self.myAC\_privacy, text = admin\_username\_list[self.current\_userID-1])

l2 = tk.Label(self.myAC\_privacy, text = admin\_gender\_list[self.current\_userID-1])

l3 = tk.Label(self.myAC\_privacy, text = admin\_phone\_list[self.current\_userID-1])

l4 = tk.Label(self.myAC\_privacy, text = admin\_email\_list[self.current\_userID-1])

dt = tk.Label(self.myAC\_privacy, text = admin\_createdDT\_list[self.current\_userID-1])

#display data labels

userID.grid(row = 0, column = 1)

l0.grid(row = 1, column = 1)

l1.grid(row = 2, column = 1)

l2.grid(row = 3, column = 1)

l3.grid(row = 4, column = 1)

l4.grid(row = 5, column = 1)

dt.grid(row = 6, column = 1)

self.pLOOK.lower()

def show\_page(self, page):

'''

show called frame

'''

self.pageTitle.destroy()

if page == self.pLOOK:

self.pageTitle = tk.Label(self.top\_frame, text = 'Look for Cases', bg = 'cadetblue3')

self.pageTitle.config(font=('Courier', 10, 'bold'))

self.pageTitle.grid(row = 1, column = 0, padx = 100)

elif page == self.pUpload:

self.pageTitle = tk.Label(self.top\_frame, text = 'Upload Cases', bg = 'cadetblue3')

self.pageTitle.config(font=('Courier', 10, 'bold'))

self.pageTitle.grid(row = 1, column = 0, padx = 100)

elif page == self.plogin:

self.pageTitle = tk.Label(self.top\_frame, text = 'Login', bg = 'cadetblue3')

self.pageTitle.config(font=('Courier', 10, 'bold'))

self.pageTitle.grid(row = 1, column = 0, padx = 100)

elif page == self.pCreateAC:

self.pageTitle = tk.Label(self.top\_frame, text = 'Create Account', bg = 'cadetblue3')

self.pageTitle.config(font=('Courier', 10, 'bold'))

self.pageTitle.grid(row = 1, column = 0, padx = 100)

elif page == self.myAC:

self.pageTitle = tk.Label(self.top\_frame, text = 'My Account', bg = 'cadetblue3')

self.pageTitle.config(font=('Courier', 10, 'bold'))

self.pageTitle.grid(row = 1, column = 0, padx = 100)

elif page == self.pDetail:

self.pageTitle = tk.Label(self.top\_frame, text = 'Look for Cases', bg = 'cadetblue3')

self.pageTitle.config(font=('Courier', 10, 'bold'))

self.pageTitle.grid(row = 1, column = 0, padx = 100)

elif page == self.pOrderBM:

self.pageTitle = tk.Label(self.top\_frame, text = 'Order building materials', bg = 'cadetblue3')

self.pageTitle.config(font=('Courier', 10, 'bold'))

self.pageTitle.grid(row = 1, column = 0, padx = 100)

page.lift()

def

root = tk.Tk()

root.geometry('1000x565')

root.title('ABC')

GuiOverlay(root)

root.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

main()