Acknowledgment

Our team would like to express our sincerest thanks to Professor Lam Hong Thanh. Thank you for establishing the circumstances for us to complete this project, allowing us to strengthen various abilities ranging from teamwork to analysis and problem-solving. We observe your dedication and devotion in every lesson we learn. You impart significant knowledge and experiences. We appreciated your classes and the time you spent assisting and advising us as we worked to master the exercises, particularly this assignment. This will be valuable knowledge and luggage for us to be able to step firmly in the future.

Programming techniques is a fascinating and efficient subject. Ensure that sufficient knowledge is provided concerning the practical demands of students. Despite our best efforts to use the knowledge we had been given during the project's implementation, we could not avoid making mistakes. Our team would like to receive your feedback and recommendations to improve the report so that next time we have a better report writing experience. We want to thank you again for all the fantastic things you've done for our group and other students in general. Your dedication to students has been, is, and will continue to be invaluable. Finally, the team would like to wish Ms. Thanh good health and success.

Table of contents

List of figures

1. **Introduction**
    1.1. **Overview**
        This Restaurant Billing System (RBS) project enables the user to maintain the restaurant's billing system and helps maintain the records. The RBS shall allow the restaurant to manage the orders easily with this program. It shall be able to update and adjust the order status and print the invoice after the successful payment.
        It is a user-friendly desktop-based system that efficiently takes care of the billing activities of the restaurant.

1.2. Statements of problem

Overcharging: In telecommunications, the billing system is a particularly delicate component that faces several issues, such as overcharging which frustrates consumers and users. This issue may be caused by the billing system's inaccuracy and the rating's inaccuracy, which is the rate assigned to each cell line.

Poor customer service: The majority of billing systems offer poor customer service, leaving little opportunity for client complaints or responses to their problems. As a result, a Report Generation system will be created for the user and management of the eBilling and Invoicing System. This system will provide both detailed and summary reports for analyzing computed data.

Security and Simplicity: Customer, Products, and Billing Generating, in other words, will automate the present manual bill generation system and maintain a searchable customer, charge database, and charge invoice, as well as preserve data security and user rights.

1.3. Purpose & Objectives

This project aims to develop an application that provides service to the user, collects order information, and generates invoices for each account.

The main project objectives are:

- To Illustrate and assess, from a business viewpoint, the entire billing system and the direct billing system-related capabilities.
- To create an emulation of a faster and more accurate billing system.
- To create a billing system that enables consumers to examine bill details.

1.4. Users

The user can be anyone either a customer or an employee. If the user is an employee, then he can make changes to the order like adjusting the quantities of foods and drinks in the bill. If the user is a customer, then he can only see the details of the chosen dishes and the total price and receive the electric invoice, not any other functions.

1.5. Functions requirements:

Designing a sales and product management system to ensure the following functions:

● Admin:

- Add/Delete/Update products
- Track goods in stock

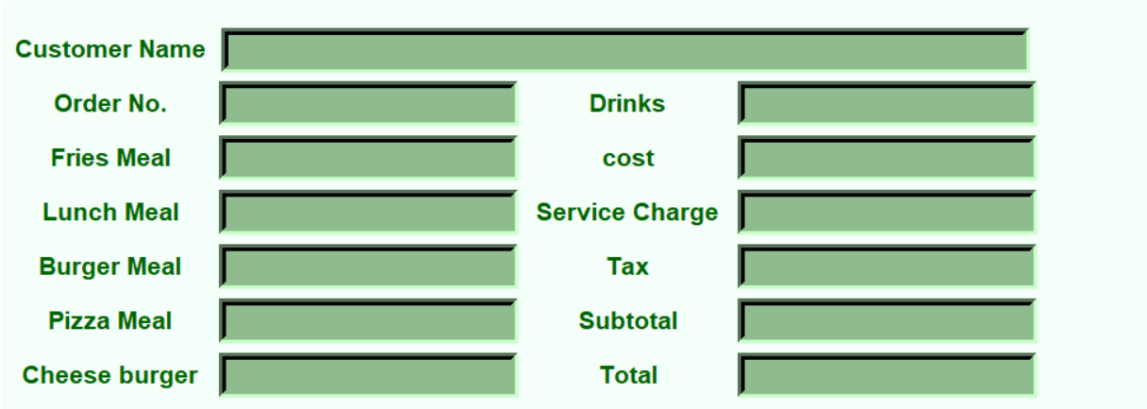
- Check and visualize sales data
- Check user data

● User:

- Check the information on available products
- Create an order

1.6. Input information

- **Customer name:** The username is taken from the user and fed into this. It is appropriately validated so that no mistake happens.
- **Order number:** The number of orders is automatically generated from 11111 to 99999
- **Input view:**




```
lblcus = Label(f1, font=( 'aria' ,16, 'bold' ),text="Customer Name",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblcus.grid(row=0,column=0)
txtcus = Entry(f1,font=('ariel' ,16,'bold') , textvariable=name,bd=6,insertwidth=4,bg="dark sea green" ,justify='left',width=56)
txtcus.place(x=186,y=5)

lblreference = Label(f1, font=( 'aria' ,16, 'bold' ),text="Order No.",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblreference.grid(row=1,column=0)
txtreference = Entry(f1,font=('ariel' ,16,'bold'), textvariable=rand , bd=6,insertwidth=4,bg="dark sea green" ,justify='right')
txtreference.grid(row=1,column=1)

lblfries = Label(f1, font=( 'aria' ,16, 'bold' ),text="Fries Meal",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblfries.grid(row=2,column=0)
txtfries = Entry(f1,font=('ariel' ,16,'bold'), textvariable=Fries , bd=6,insertwidth=4,bg="dark sea green" ,justify='right')
txtfries.grid(row=2,column=1)

lblLunch = Label(f1, font=( 'aria' ,16, 'bold' ),text="Lunch Meal",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblLunch.grid(row=3,column=0)
txtLunch = Entry(f1,font=('ariel' ,16,'bold'), textvariable=Lunch , bd=6,insertwidth=4,bg="dark sea green" ,justify='right')
txtLunch.grid(row=3,column=1)

lblburger = Label(f1, font=( 'aria' ,16, 'bold' ),text="Burger Meal",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblburger.grid(row=4,column=0)
txtburger = Entry(f1,font=('ariel' ,16,'bold'), textvariable=Burger , bd=6,insertwidth=4,bg="dark sea green" ,justify='right')
txtburger.grid(row=4,column=1)

lblFilet = Label(f1, font=( 'aria' ,16, 'bold' ),text="Pizza Meal",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblFilet.grid(row=5,column=0)
txtFilet = Entry(f1,font=('ariel' ,16,'bold'), textvariable=Filet , bd=6,insertwidth=4,bg="dark sea green" ,justify='right')
txtFilet.grid(row=5,column=1)

lblCheese_burger = Label(f1, font=( 'aria' ,16, 'bold' ),text="Cheese burger",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblCheese_burger.grid(row=6,column=0)
txtCheese_burger = Entry(f1,font=('ariel' ,16,'bold'), textvariable=Cheese_burger , bd=6,insertwidth=4,bg="dark sea green" ,justify='right')
txtCheese_burger.grid(row=6,column=1)
```

```
lblDrinks = Label(f1, font=( 'aria' ,16, 'bold' ),text="Drinks",fg="dark green",bd=10,anchor='w',bg="mint cream")
lblDrinks.grid(row=1,column=2)
txtDrinks = Entry(f1,font=('ariel' ,16,'bold'), textvariable=Drinks , bd=6,insertwidth=4,bg="dark sea green" ,justify='right')
txtDrinks.grid(row=1,column=3)
```

- **Menu:** The customer can choose what they want to have for their meals. They just basically give a specific amount of food they want and get the payment information on the right board.

- Fries meal
- Lunch meal
- Burger meal
- Pizza meal
- Cheeseburger
- Drinks

No.	Meals	Price	Qnty
1	Fries Meal	$25	0
2	Lunch Meal	$35	0
3	Burger Meal	$25	0
4	Pizza Meal	$25	0
5	Chese Burger	$25	0
6	Drinks	$25	0

```
#-----------TreeView-----------
my_tree=ttk.Treeview()
my_tree['columns']=("Name","Price","Qnty")
my_tree.column("#0",width=30,minwidth=25)
my_tree.column("Name",anchor=W, width=120)
my_tree.column("Price",anchor=CENTER, width=80)
my_tree.column("Qnty",anchor=CENTER,width=80)

my_tree.heading("#0",text="No.",anchor=CENTER)
my_tree.heading("Name",text="Meals",anchor=W)
my_tree.heading("Price",text="Price",anchor=CENTER)
my_tree.heading("Qnty",text="Qnty",anchor=CENTER)
my_tree.pack(padx=10,pady=15,fill=Y)
```

- **Invoice:**

No.	Order No.	Cus.Name	Status	Cost
	20886	Kha	Processing	$0.0
	21278	Phong	Processing	$203.5

```
my_bill=ttk.Treeview()
my_bill['columns']=("Order No.","Cus.Name","Status","Cost")
my_bill.column("#0",width=30,minwidth=25)
my_bill.column("Order No.",anchor=W, width=70)
my_bill.column("Cus.Name",anchor=W, width=110)
my_bill.column("Status",anchor=CENTER, width=100)
my_bill.column("Cost",anchor=CENTER,width=60)

my_bill.heading("#0",text="No.",anchor=CENTER)
my_bill.heading("Order No.",text="Order No.",anchor=W)
my_bill.heading("Cus.Name",text="Cus.Name",anchor=CENTER)
my_bill.heading("Status",text="Status",anchor=CENTER)
my_bill.heading("Cost",text="Cost",anchor=CENTER)
my_bill.pack(fill=Y)
```

2. Functions

2.1. Total

- **Interface**

- **Description**

```
btnTotal=Button(f1,padx=16,pady=8, bd=10 ,fg="black",font=('ariel' ,16,'bold'),width=10, text="TOTAL", bg="dark sea green",command=Ref)
btnTotal.grid(row=8, column=2)
```

```
def Ref():
    x=random.randint(12980, 50876)
    randomRef = str(x)
    rand.set(randomRef)
    cof =float(Fries.get())
    colunch= float(Lunch.get())
    cob= float(Burger.get())
    cofi= float(Filet.get())
    cochee= float(Cheese_burger.get())
    codr= float(Drinks.get())

    costoffries = cof*25
    costoflunch = colunch*40
    costofburger = cob*35
    costoffilet = cofi*50
    costofcheeseburger = cochee*50
    costofdrinks = codr*35

    costofmeal =str('%.2f'% (costoffries +  costoflunch + costofburger + costoffilet + costofcheeseburger + costofdrinks))
    PayTax=((costoffries +  costoflunch + costofburger + costoffilet +  costofcheeseburger + costofdrinks)*0.05)
    Totalcost=(costoffries +  costoflunch + costofburger + costoffilet  + costofcheeseburger + costofdrinks)
    Ser_Charge=((costoffries +  costoflunch + costofburger + costoffilet + costofcheeseburger + costofdrinks)*0.05)
    Service=str('%.2f'% Ser_Charge)
    OverAllCost=str( PayTax + Totalcost + Ser_Charge)
    PaidTax=str('%.2f'% PayTax)

    Service_Charge.set(Service)
    cost.set(costofmeal)
    Tax.set(PaidTax)
    Subtotal.set(costofmeal)
    Total.set(OverAllCost)
```

- **How it works:** When we click the button ‘Total’ it shall show the total price of the order later.

2.2. Print

- **Interface:**

- **Description:**

```
btnprint=Button(f1,padx=16,pady=8, bd=10 ,fg="black",font=('ariel' ,16,'bold'),width=10, text="PRINT", bg="dark sea green",command=printbill)
btnprint.grid(row=10, column=2)
```

```
def printbill():
    f=open(f"C:\\THC4\\bill {rand.get()}.txt","a+")
    line=f"Bill no.{rand.get()}\n"
    lines=f"Date:{localtime}\n"
    line0=f"Customer's Name:{name.get()}\n"+"\n"
    line1=f"Fries:          {Fries.get()}\n"
    line2=f"Lunch Meal:     {Lunch.get()}\n"
    line3=f"Burger:         {Burger.get()}\n"
    line4=f"Pizza Meal:     {Filet.get()}\n"
    line5=f"Chese Burger:   {Cheese_burger.get()}\n"
    line6=f"Drinks:         {Drinks.get()}\n"+"\n"
    line7="-------------------------\n"
    line8=f"Total:          ${Total.get()}\n"
    f.write(line+lines+line0+line1+line2+line3+line4+line5+line6+line7+line8)
    f.close()
    my_tree.insert(parent='',index='end',iid=0,text="1",values=("Fries Meal","$25",f"{Fries.get()}"))
    my_tree.insert(parent='',index='end',iid=1,text="2",values=("Lunch Meal","$35",f"{Lunch.get()}"))
    my_tree.insert(parent='',index='end',iid=2,text="3",values=("Burger Meal","$25",f"{Burger.get()}"))
    my_tree.insert(parent='',index='end',iid=3,text="4",valu                                    let.get()}"))
    my_tree.insert(parent='',index='end',iid=4,text="5",valu                                    Cheese_burger.get()}"))
    my_tree.insert(parent='',index='end',iid=5,text="6",valu                                    .get()}"))
    my_bill.insert(parent='',index='end',values=(rand.get(),name.get(),"Processing",f"${Total.get()}"))
```

```
name=StringVar()
Full name: restaurant.name
```

- **How it works:** This button allows the user to transmit the order information into the 1st board so that both the user and the staff can easily track and check the order information.

2.3. Update

- **Interface:**

- **Description:**

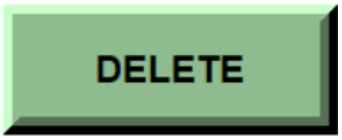
```
btnupdate=Button(f1,padx=16,pady=8, bd=10 ,fg="black",font=('ariel' ,16,'bold'),width=10, text="UPDATE", bg="dark sea green",command=update_item)
btnupdate.grid(row=8, column=3)
```

```
def update_item():
    selected = my_bill.focus()
    temp = my_bill.item(selected, 'values')
    sal_up = "Finished"
    my_bill.item(selected, values=(temp[0],temp[1], sal_up, temp[3]))
```

- **How it works:** After checking the information from the 1st board, the order will be in progress. When the order is finished, the employee can use this button to update the order status: from ‘processing’ to ‘finished’.

2.4. Delete

- **Interface:**



- **Description:**

```
btndel=Button(f1,padx=16,pady=8, bd=10 ,fg="black",font=('ariel' ,16,'bold'),width=10, text="DELETE", bg="dark sea green",command=deletebill)
btndel.grid(row=10, column=3)
```

```
def deletebill():
    i=my_bill.selection()
    my_bill.delete(i)
```

- **How it works:** If there is any fault when taking an order like the wrong customer’s name or the customer wants to adjust the order, the employee can use this button to delete that order from the display board

2.5. Reset

- **Interface:**

- **Description:**

```
btnreset=Button(f1,padx=16,pady=8, bd=10 ,fg="black",font=('ariel' ,16,'bold'),width=10, text="RESET", bg="dark sea green",command=reset)
btnreset.grid(row=8, column=1)
```

```
def reset():
    name.set("")
    rand.set("")
    Fries.set("")
    Lunch.set("")
    Burger.set("")
    Filet.set("")
    Subtotal.set("")
    Total.set("")
    Service_Charge.set("")
    Drinks.set("")
    Tax.set("")
    cost.set("")
    Cheese_burger.set("")
    my_tree.delete(*my_tree.get_children())
```

- **How it works:** After getting an order done, the employee takes another order for the customer; in this case, he can use this button to set everything again to the beginning.
-

2.6. Exit

- **Interface:**

- **Description:**

```
btnexit=Button(f1,padx=16,pady=8, bd=10 ,fg="black",font=('ariel' ,16,'bold'),width=10, text="EXIT", bg="dark sea green",command=qexit)
btnexit.grid(row=10, column=1)
```

```
def qexit():
    root.destroy()
```

- **How it works:** This button allows the user to leave the program.

3. Techniques

- **tkinter:**

+ Tk: Create window
+ Ttk: themed widget. Some widgets in Tkinter are supported to display different flower colors, the term is called theme. To use the theme, we import the ttk module.
+ Frame: Used to manage the widget
+ Button: to add buttons in the Python application.
+ manager classes to manage widgets such as pack(), grid() and place()
+ random.randint: This function returns a random integer among the specified integers.
+ datetime: to manipulate both date and time in python.

- **Time:**

+ Function time.asctime([tupletime]) Accepts a time-tuple and returns a string of 24 readable characters eg Mon Dec 11 18:07:14 2015
+ The function time.localtime([secs]) is similar to gmtime(), but it converts seconds to local time.
+ +function time.time() Returns time as a real number expressed in seconds since the epoch, in UTC
+ csv: processes CSV files to read and write and get data from specified columns.

4. Project evaluation

4.1. Having met requirements

- **User friendly:** The proposed system is user friendly because the retrieval and storing of data is fast and data is maintained efficiently. Moreover the graphical user interface is provided in the proposed system, which provides users to deal with the system very easily.
- **No or very little paperwork:** The proposed system either does not require paperwork or very little paperwork is required. All the data is immediately feted into the computer, and various bills and reports can be generated through computers. Since all the information is kept in a database, no organization's data can be destroyed. Moreover work becomes very easy because there is no need to keep data on paper.
- **Reports are easily generated:** Reports can be easily generated in a proposed system. So any type of report can be generated in a proposed system, which helps the managers in a decision-making activity.

4.2. Limitations

- **The inability to modify data:** The managing of colossal data effectively and efficiently for efficient results, storing the details of the consumers etc. in such a way that the database can be modified as not possible in the current system.
- **Manual operator control:** Manual operator control is there and leads to a lot of chaos and errors.
- **The inability of sharing the data:** Data cannot be shared in the existing system. This means that no two persons can use the same data in the existing system. Also, the two departments in an organization cannot interact with each other without the actual movement of data.

4.3. Further developments

- **Expand the database:** The proposed system is not fully equipped with the help of the current database to manage vast information. It could be that old records are outdated, and we can get access to these data by importing into a bigger database like the computing cloud.
- **Sharing the data is possible:** Data will be shared in the system. This means that two or more persons can use the same data in the existing system, provided that they have the right to access that data. Also the two or more departments in an organization can easily interact without the actual movement of data.
- **Visualize data:** It gives you a complete view of any outliers or invalidations that may be residing in your dataset. Statistical histograms and bar charts showing the completeness of attributes can be handy.
- **Automate Data Entry by Scanning Barcodes & QR Codes:** This can drastically reduce data entry time for a product since all that is needed to be done is scanning the barcode.

4.4. Conclusion

After all the hard work has been done for the electricity bill management system here. It is software that helps the user work with the billing cycles, pay bills, print the invoices, etc. This software reduces the amount of manual data entry and gives more significant efficiency. The User Interface is amicable and can be easily used by anyone. It also decreases the time taken to write details and other modules.

5. Latest version information

This section contains a cumulative list of Billing Restaurant System content changes on May 17, 2022.

1. Set '0' as the default quantity for each dish.

Description: The problem that we struggled with before this update was that, after resetting to the beginning, the quantity for each dish was empty (no number displayed). If the course is not ordered, the users do not have to enter the number ‘0’ from this version.

2. PDF file

Description: The Billing Restaurant System could only work with .txt files to generate electric invoices in the previous version. The proposed application can generate electric invoices in PDF file format from now on. Though the current PDF design is not attractive and user-friendly, we’re trying to fix it to make it more convenient and flexible, allowing the users to track and check the data information quickly.