

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“Jnanasangama”, Belagavi-590018, Karnataka



BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V.Puram, Bengaluru-560 004



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DATABASE MANAGEMENT SYSTEM MINI PROJECT
18CSL58

“COUNTER BILLING MANAGEMENT SYSTEM”

Submitted By

YAMUNA
1BI19CS190

for the academic year 2021-22

Department of Computer Science & Engineering
Bangalore Institute of Technology
K.R. Road, V.V.Puram, Bengaluru-560 004

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“Jnanasangama”, Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V.Puram, Bengaluru-560 004



Department of Computer Science & Engineering

Certificate

This is to certify that the implementation of **DBMS MINI PROJECT** entitled “*COUNTER BILLING MANAGEMENT SYSTEM*” has been successfully completed by **USN: 1BI19CS190**

NAME: YAMUNA of V semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in Computer Science & Engineering of the Visvesvaraya Technological University during the academic year 2021-2022

Lab Incharges:

Dr. Suneetha K R
Associate Professor
Dept. of CS&E
Bangalore Institute of Technology
Bengaluru

Dr. Girija J
Professor and Head
Department of CS&E
Bangalore Institute of Technology
Bengaluru

Examiners: 1)

2)

ABSTRACT

Counter billing management system plays effective role maintaining the records, no data is lost. Bills that have been deleted due to some uncertainty are also listed and admin can keep track of those bills. This makes the billing management runs smooth by removing the unnecessary detail from it.

It is very easy to use. It is user friendly. Less efforts are required to operate the system. Admin keeps track and manages all the records, bills of the management system. Checking of the items, employees, sales of the items and demand of the items can be got to know from this easily by retrieving the required data.

Counter billing management system saves time of both users and the customers. It generates bills, so customers have a reference to what they have bought at what price.

At its most basic description: billing software allows you to track the products and services your customers use, generate and send invoices with, and receive payments. However, some billing platforms are capable of much more. They can automate the repetitive tasks your finance team struggles with on a daily basis.

Creates a single database that will easily segregate a client purchases, relevant files along with a clean filter search option for effortless accessibility. Billing systems can create master data of parties and amounts. It also creates a backup either offline or online. Systems also provide easy data extraction for tax return filing and any other usage.

This system basically manages the counter bills generated from items. These bills are generated through the employees working. This also manages the deleted bills because of any wrong entries or cancellation of any item from a customer.

ACKNOWLEDGEMENT

The knowledge and satisfaction that accompany the successful completion of any task would be incomplete without mention of people who made it possible, whose guidance encouragement crowned my effort with success.

I would like to thank all and acknowledge the help I have received to carry out this Mini Project. I would like to convey my thanks to Head of Department Dr. GIRIJA J. for being kind enough to provide the necessary support to carry out the mini project.

I am most humbled to mention the enthusiastic influence provided by the lab in-charge Dr. Suneetha K, on the project for their ideas, time to time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I express my sincere gratitude to the friendly Science Department, BIT.

YUSRA NAHEED
1BI19CS192

Contents

1.		INTRODUCTION	
	1.1	INTRODUCTION	2
	1.2	PROBLEM STATEMENT	2
2.		BACK END DESIGN	
	2.1	CONCEPTUAL DESIGN	4
	2.2	LOGICAL DESIGN	5
	2.3	NORMALIZATION	6
3.		FRONT END DESIGN	
	3.1	SCREEN LAYOUT DESIGN	11
	3.2	CONNECTIVITY	13
4.		MAJOR MODULES	
	4.1	MODULES WITH DESCRIPTION	15
5.		IMPLEMENTATION	
	5.1	DATABASE CODE	17
	5.2	PYTHON CODE	21
6.		SNAPSHOTS	
	6.1	SNAPSHOTS	41
7.		APPLICATIONS	
	7.1	APPLICATIONS	48
8.		CONCLUSION	
	8.1	CONCLUSION	50

CHAPTER-1

INTRODUCTION

1. INTRODUCTION

1.1 INTRODUCTION

Billing Software is designed to handle time and billing tracking as well as invoicing customer for services and products. It helps the business owner's in a comprehensive manner to keep a track of multiple invoice and accounts just by few clicks on the mouse. It helps in managing chain of stores as well as multiple company billing system. It also provides recurring service and renting business billing solution.

At its most basic description: billing software allows you to track the products and services your customers use, generate and send invoices with, and receive payments. However, some billing platforms are capable of much more. They can automate the repetitive tasks your finance team struggles with on a daily basis.

Creates a single database that will easily segregate a client purchases, relevant files along with a clean filter search option for effortless accessibility. Billing systems can create master data of parties and amounts. It also creates a backup either offline or online. Systems also provide easy data extraction for tax return filing and any other usage.

This system basically manages the counter bills generated from items. These bills are generated through the employees working. This also manages the deleted bills because of any wrong entries or cancellation of any item from a customer.

1.2 PROBLEM STATEMENT:

To design and implement a counter billing management system for the purpose of generating and managing bills in an easier and faster way.

CHAPTER-2
BACK END DESIGN

2. BACK END

2.1 CONCEPTUAL DATABASE DESIGN

ER DIAGRAM

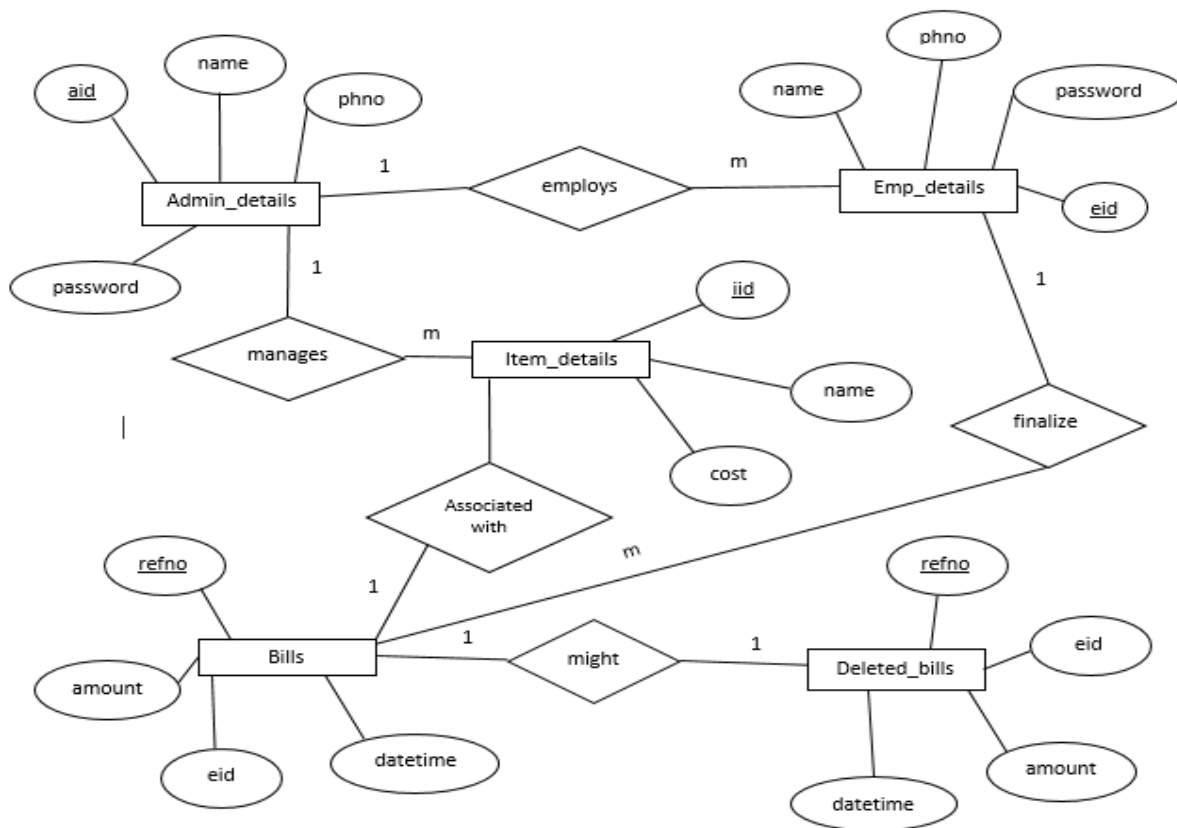


Fig 2.1.1 E. R Diagram

E. R diagram consists of 5 entities. There are no weak entities as every entity has its primary key. Each entity has one primary key and all attributes associated with the entities above are simple attributes.

2.2 LOGICAL DATABASE DESIGN

ER-MAPPING

Admin_details

<u>AID</u>	NAME	PASSWORD	PHNO
------------	------	----------	------

Emp_details

<u>EID</u>	NAME	PASSWORD	PHNO
------------	------	----------	------

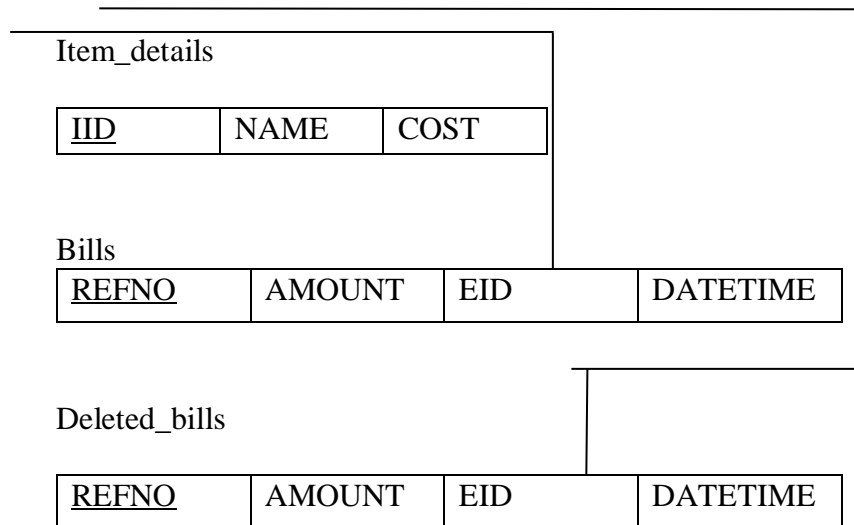


Fig 2.2.1 ER to Relational Mapping

ER to Relational Mapping consists of tables, these tables are created by following 7 steps algorithm tables are connected using foreign keys, in the above diagram manages loan, manages loan status tables are formed due to the M:N relation between two entities

2.3 NORMALIZATION

Database Normalization is a technique of organizing the data in database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like avoid insertion anomaly, update anomaly & deletion anomaly. It is a multi step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating Redundant(useless) data
- Ensuring data dependencies make sense i.e data is logically stored

2.3.1 : CONDITIONS FOR NORMALIZATION

First Normal Form (1NF):

As per First Normal Form, no two Rows of data must contain repeating group of information i.e. each set of columns must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

Second Normal Form (2NF):

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails Second normal form.

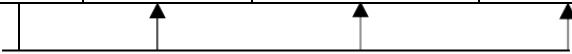
Third Normal Form (3NF):

Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this transitive functional dependency should be removed from the table and also the table must be in Second normal form.

NORMALISED TABLES

2.3.2 ADMIN_DETAILS

<u>AID</u>	NAME	PASSWORD	PHNO
------------	------	----------	------



1NF

This is satisfied as all attributes are atomic

2NF

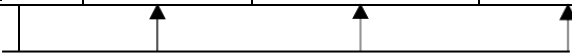
This is satisfied as there is no partial dependency

3NF

This is satisfied as there is no transitive dependency

2.3.3 EMP_DETAILS

<u>EID</u>	NAME	PASSWORD	PHNO
------------	------	----------	------



1NF

This is satisfied as all attributes are atomic

2NF


This is satisfied as there is no partial dependency

3NF

This is satisfied as there is no transitive dependency

2.3.4 ITEM_DETAILS

<u>IID</u>	NAME	COST
------------	------	------



COUNTER BILLING MANAGEMENT SYSTEM

1NF

This is satisfied as all attributes are atomic

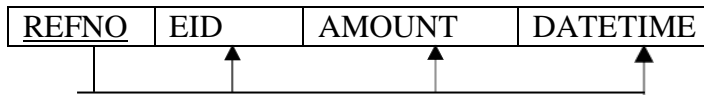
2NF

This is satisfied as there is no partial dependency

3NF

This is satisfied as there is no transitive dependency

2.3.5 BILLS



1NF

This is satisfied as all attributes are atomic

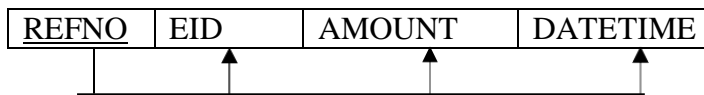
2NF

This is satisfied as there is no partial dependency

3NF

This is satisfied as there is no transitive dependency

2.3.6 DELETED_BILLS



1NF

This is satisfied as all attributes are atomic

2NF

This is satisfied as there is no partial dependency

3NF

This is satisfied as there is no transitive dependency

CHAPTER-3

FRONT END DESIGN

3.FRONT END DESIGN

3.1 SCREEN LAYOUT DESIGN

3.1 Tkinter

Tkinter is the standard GUI library for python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps:

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more of the widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Tk()

Tkinter is a python package which comes with many functions and methods that can be used to create an application, we create an instance of tkinter frame, i.e, Tk() . It helps to display the root window and manages all the other components of the tkinter application.

Ttk()

The basic idea of tkinter.ttk is to separate, to the extent possible, the code implementing a widget's behavior from the code implementing its appearance. Ttk comes with 18 widgets, twelve of which already existed in tkinter and other six are new. Using the Ttk widgets gives the application an improved look.

Widgets used:

Label - It is used to display text or image on the screen.

Button – It is used to add buttons to your application.

ComboBox – It contains a down arrow to select from list texts, graphics etc.

Entry – It is used to input single line text entry from user.

Frame – It is used as container to hold and organize the widgets.

Scrollbar – It is used to scroll down the contents. It provides a slide controller.

Menu – It is used to create all kinds of menu used by an application.

LabelFrame – It is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts

COUNTER BILLING MANAGEMENT SYSTEM

MessageBox – It is used to display message boxes in your applications.

Tkcalendar – It provides DateEntry and Calendar widgets for tkinter application. A DateEntry widget contains three fields that refer to general format of date as MM/DD/YY.

Treeview- It refers to hierarchical representation. When we have a relation between data in that case, we use treeview.

3.1 CONNECTIVITY

MySQL Connector/Python enables Python programs to access MySQL databases, using an API that is compliant with the Python database API specification v2.0 (PEP 249). It is written in pure Python and does not have any dependencies except for the Python Standard Library

We write the following code to connect to database by creating connection

```
import mysql.connector

mydb =
mysql.connector.conne
ct(host = "localhost",
user = "root",
password= "*****"
database= "created")
```

Then to perform queries we use a cursor. Cursor can be created using .cursor() method.

Cursor Object: The MySQLCursor of mysql-connector-python(and similar libraries) is used to execute statements to communicate with the MySQL database.

Using the methods of it you can execute SQL statements, fetch data from the result sets, call procedures. The following statement creates a cursor object.

```
mycursor = mydb.cursor()
```

CHAPTER-4

MAJOR MODULES

4. MAJOR MODULES

4.1 DESCRIPTION OF FUNCTIONALITIES

ADMIN MODULE: This module deals with the managing of employees, items and sales. This module plays a key role in handling the necessary operations in a counter billing system. The admin is responsible for the employees to be employed. This module performs various operations

- Admin details: Displays the details of the admin
- Employee details: Displays the list of employees working for the management system
- Sales: Shows sales of the items whose bills are generated
- Add user: Admin has been given the authority to add the users by providing their necessary details employee identification number, username, password and phone number
- Show items: Displays all the items that are being managed
- Add item: Admin has been given the authority to add the items by providing the necessary details about that particular item like item identification number, name and its cost price
- Delete user: Admin has been given the authority to delete a user from the management system
- Delete item: Admin manages the items for the management system. Admin can delete if required
- Change price: Admin can change the price of any item depending on the sales and demand if that item in the market thereby attract customers
- Deleted bill: There might occur scenario that a bill generated has to be deleted. Certain reasons are when a customer cancels any item from the generated bill or when some wrong entry is done in the bill. This tab lists the bills that are deleted.

USER MODULE: This module deals with generating and managing of bills. Here a bill generated can be deleted depending upon the scenario for the smooth flow in the management system without causing errors in the calculations. The various operation

- User details: Displays the details of the user basically employees
- Billing: Here is where the user/employee selects the item requested by the customer with its quantity and generates a final bill
- View bill: Once when the bill is finalized, then the user can view the generated bill in this tab
- Remove bill: For due to some reasons a bill generated has to be deleted or removed, it is done in this tab.

CHAPTER-5 IMPLEMENTATION

5. IMPLEMENTATION

5.1 DATABASE CODE

5.1.1 CREATION OF TABLES

```
CREATE TABLE EMP_DETAILS (
```

```
EID INT(10) PRIMARY KEY,
```

```
NAME VARCHAR(20),
```

```
PASSWORD VARCHAR(20),
```

```
PHNO INT(10) UNIQUE);
```

```
CREATE TABLE ADMIN_DETAILS (
```

```
AID INT(10) PRIMARY KEY,
```

```
NAME VARCHAR(20),
```

```
PASSWORD VARCHAR(20),
```

```
PHNO INT(10) UNIQUE);
```

```
CREATE TABLE ITEM_DETAILS (
```

```
IID INT(10) PRIMARY KEY,
```

```
NAME VARCHAR(20),
```

```
COST VARCHAR(20));
```

COUNTER BILLING MANAGEMENT SYSTEM

```
CREATE TABLE BILLS (  
    DATETIME VARCHAR(40),  
    EID INT(20),  
    AMOUNT DOUBLE,  
    REFNO INT(10));
```

```
CREATE TABLE DELETED_BILLS (  
    DATETIME VARCHAR(40),  
    EID INT(20),  
    AMOUNT DOUBLE,  
    REFNO INT(10));
```

5.1.2 TRIGGERS

```
DELIMITER &&

CREATE TRIGGER REMOVEBILLS
AFTER DELETE ON BILLS FOR
EACH ROW

BEGIN
IF NEW.REFNO=
'NULL' THEN
UPDATE DELETED
BILLS DB
SET REFNO = NEW.REFNO
WHERE DB.EID = OLD.EID AND

DB.AMOUNT=OLD.AMOUNT;

END IF;
END
```

Trigger is added to the BILLS table which is triggered when a bill is set to deleted, and refno, EId, total amount and date time is inserted into deleted_bills s table.

5.1.3 STORED PROCEDURE

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_adminlogin` (IN `username`  
VARCHAR(200), IN `adminpwd` VARCHAR(200)) BEGIN  
select Name, Aid, password from admin where Name=username and password=adminpwd;  
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_userlogin` (IN `userID`  
VARCHAR(200), IN `userpwd` VARCHAR(200)) BEGIN  
select Name, ,id, password from employee where Eid=userID and password=userpwd;  
END
```


5.2 PYTHON CODE

5.2.1 INITIAL SETUP

```
from tkinter import *
from data import *
from tkinter import ttk
from datetime import datetime
import random
#from PIL import ImageTk, Image
import os

selected_items = []
r = random.randint(10000, 999999)

root = Tk()
root.geometry("1000x500")
root.title("Billing System")
root.configure(bg="#212121")
adminB = Button(root, text="ADMIN", fg="white", bg="#1966ff", padx=16,
                pady=8, bd=5, width=10, command=admin_login)
userB = Button(root, text="USER", fg="white", bg="#1966ff", padx=16,
               pady=8, bd=5, width=10, command=user_login)
adminB.grid(row=0, column=1)
userB.grid(row=1, column=1)
#img = ImageTk.PhotoImage(Image.open(r"billing system\img\logo.jpg"))
#panel = Label(root, image=img)
#panel.photo = img
#panel.grid(row=2, column=0)
root.mainloop()
```

5.2.2 ADMIN

```
def admin_login():
    al = Toplevel()
    al.geometry("250x100")
    al.title("ADMIN LOGIN")
    al.configure(bg="#212121")
    l1 = Label(al, text="ENTER AID: ", fg="white", bg="#212121")
    e1 = Entry(al)
    l2 = Label(al, text="ENTER PASSWORD: ", fg="white", bg="#212121")
    e2 = Entry(al, show="*")
    sb = Button(al, text="LOGIN", fg="white", bg="#1966ff",
                command=lambda: admin_auth(e1.get(), e2.get()))
    l1.grid(row=1, column=0)
    e1.grid(row=1, column=1)
    l2.grid(row=2, column=0)
    e2.grid(row=2, column=1)
    sb.grid(row=3, column=1)

#Admin dashboard
def ad(username):
    ad = Toplevel()
    ad.geometry("900x700")
    ad.title("ADMIN DASHBOARD")
    ad.configure(bg="#212121")
    tab_main = ttk.Notebook(ad)
    t1 = admin_details(tab_main, username)
    t2 = list_emp(tab_main, username)
    t3 = sales(tab_main, username)
    t4 = add_user(tab_main, username)
    t5 = list_items(tab_main, username)
    t6 = add_item_to_db(tab_main, username)
    t7 = delete_user(tab_main, username)
    t8 = delete_item(tab_main, username)
    t9 = change_price_of_item(tab_main, username)
    t10 = deleted_bills(tab_main, username)
    tab_main.add(t1, text="ADMIN DETAILS")
    tab_main.add(t2, text="EMPLOYEE DETAILS")
    tab_main.add(t3, text="SALES")
    tab_main.add(t4, text="ADD USER")
    tab_main.add(t5, text="SHOW ITEMS")
    tab_main.add(t6, text="ADD ITEM")
    tab_main.add(t7, text="DELETE USER")
    tab_main.add(t8, text="DELETE ITEM")
```

COUNTER BILLING MANAGEMENT SYSTEM

```
tab_main.add(t9, text="CHANGE PRICE")
tab_main.add(t10, text="DELETED BILLS")
tab_main.pack(expand=1, fill='both')
```

#Admin authentication

```
def admin_auth(username, password):
    if username and password and username.isnumeric():
        if ck_details_admin(int(username), password):
            ad(username)
        else:
            popup("ERROR WRONG DETAILS")
    else:
        popup("ERROR WRONG DETAILS")
```

#Admin details

```
def admin_details(tab_main, username):
    t = Frame(tab_main, background="#212121")
    data = get_admin_details(int(username))
    lname = Label(t, text="NAME: "+data[1], fg="white", bg="#212121")
    lid = Label(t, text="ID: "+str(data[0]), fg="white", bg="#212121")
    lphno = Label(t, text="PHNO: "+str(data[2]), fg="white", bg="#212121")
    lname.config(font=("Courier", 44))
    lid.config(font=("Courier", 44))
    lphno.config(font=("Courier", 44))
    lname.pack()
    lid.pack()
    lphno.pack()
    return t
```

5.2.3 EMPLOYEE LOGIN

```
def user_login():
    ul = Toplevel()
    ul.geometry("250x100")
    ul.title("USER LOGIN")
    ul.configure(bg="#212121")
    l1 = Label(ul, text="ENTER EID: ", fg="white", bg="#212121")
    e1 = Entry(ul)
    l2 = Label(ul, text="ENTER PASSWORD: ", fg="white", bg="#212121")
    e2 = Entry(ul, show="*")
    sb = Button(ul, text="LOGIN", fg="white", bg="#1966ff",
                command=lambda: user_auth(e1.get(), e2.get()))
    l1.grid(row=0, column=0)
    e1.grid(row=0, column=1)
    l2.grid(row=1, column=0)
    e2.grid(row=1, column=1)
```

COUNTER BILLING MANAGEMENT SYSTEM

```
sb.grid(row=2, column=1)
```

#Employee authentication

```
def user_auth(username, password):
    if username and password and username.isnumeric():
        if ck_details_emp(int(username), password):
            ud(username)
        else:
            popup("ERROR WRONG DETAILS")
    else:
        popup("ERROR WRONG DETAILS")
```

#Employee dashboard

```
def ud(username):
    ud = Toplevel()
    ud.geometry("700x500")
    ud.title("USER DASHBOARD")
    ud.configure(bg="#212121")
    tab_main = ttk.Notebook(ud)
    t1 = user_details(tab_main, username)
    t2 = bill(tab_main, username)
    t3 = show_items(tab_main, username)
    t4 = remove_bill(tab_main, username)
    tab_main.add(t1, text="USER DETAILS")
    tab_main.add(t2, text="BILLING")
    tab_main.add(t3, text="VIEW BILL")
    tab_main.add(t4, text="REMOVE BILL")
    tab_main.pack(expand=1, fill='both')
```

#Employee details

```
def user_details(tab_main, username):
    t = Frame(tab_main, background="#212121")
    data = get_user_details(int(username))
    lname = Label(t, text="NAME: "+data[1], fg="white", bg="#212121")
    lid = Label(t, text="ID: "+str(data[0]), fg="white", bg="#212121")

    lphno = Label(t, text="PHNO: "+str(data[2]), fg="white", bg="#212121")
    lname.config(font=("Courier", 34))
    lid.config(font=("Courier", 34))
    lphno.config(font=("Courier", 34))
    lname.pack()
    lid.pack()
    lphno.pack()
    return t
```

Showing the list of employees

```
def s(tree):
    tree.delete(*tree.get_children())
```

COUNTER BILLING MANAGEMENT SYSTEM

```
rows = get_all_employees()
for i in rows:
    tree.insert("", 0, text=i[0], values=(i[1], i[3]))

# List employees
def list_emp(tab_main, username):
    t = Frame(tab_main, background="#212121")

    tree = ttk.Treeview(t)
    b = Button(t, text="SHOW", command=lambda: s(
        tree), fg="white", bg="#1966ff")
    tree["columns"] = ("one", "two")
    tree.heading("#0", text="ID")
    tree.heading("one", text="NAME")
    tree.heading("two", text="PHNO")
    tree.column("#0", anchor=CENTER)
    tree.column("one", anchor=CENTER)
    tree.column("two", anchor=CENTER)
    b.pack()
    tree.pack()
    return t
```

5.2.4 BILLING SECTION

```
def bill(tab_main, username):
    t = Frame(tab_main, background="#212121")
    rows = get_items()
    options = []
    for i in rows:
        options.append(i[1])
    clicked = StringVar()
    if(len(options)):
        clicked.set(options[0])
        cl = IntVar()
        cl.set(1)
        drop = OptionMenu(t, clicked, *options)
        l = Label(t, text="SELECT QUANTITY:", fg="white", bg="#212121")
        q = OptionMenu(t, cl, 1, 2, 3, 4, 5, 6, 7, 8, 9)
        b = Button(t, text="ADD", command=lambda: add_item(
            clicked.get(), cl.get()), fg="white", bg="#1966ff")
        reset = Button(t, text="RESET", command=lambda: resetf(),
            fg="white", bg="#1966ff")
        drop.pack()
        l.pack()
        q.pack()
```

COUNTER BILLING MANAGEMENT SYSTEM

```
b.pack()
reset.pack()
return t
```

BILLS

Function for resetting the bill

```
def resetf():
    global selected_items
    global r
    selected_items.clear()
    r = random.randint(10000, 999999)
```

Print/Show the bill

```
def get_bill(username):
    cost = 0
    items = []
    for i in selected_items:
        cost = cost + int(i[2])
        items.append(i[0])
    tax = round(cost * 0.2, 2)
    service = round(cost/99, 2)
    total_cost = round(cost+tax+service)
    t = Toplevel()
    t.geometry("350x700")
    t.title("BILL")
    t.configure(bg="#212121")
    date = Label(t, text="DATE: " +
        datetime.now().strftime("%d-%m-%Y"), fg="white", bg="#212121")
    time = Label(t, text="TIME: "+datetime.now().strftime("%H:%M"),
        fg="white", bg="#212121")
    ref = Label(t, text="REF NO: "+str(r), fg="white", bg="#212121")
    emp = Label(t, text="EMPLOYEE ID: "+str(username),
        fg="white", bg="#212121")
    costl = Label(t, text="COST: "+str(cost), fg="white", bg="#212121")
    taxl = Label(t, text="TAX: "+str(tax), fg="white", bg="#212121")
    servicel = Label(t, text="SERVICE CHARGES: " +
        str(service), fg="white", bg="#212121")
    total_costl = Label(t, text="TOTAL COST: " +
        str(total_cost), fg="white", bg="#212121")
    date.pack()
    time.pack()
    ref.pack()
    emp.pack()
    costl.pack()
    taxl.pack()
    servicel.pack()
    total_costl.pack()
    Label(t, text="ITEMS:", fg="white", bg="#212121").pack()
```

COUNTER BILLING MANAGEMENT SYSTEM

```
for it in range(len(selected_items)):
    Label(t, text=selected_items[it][0] + " ( X " + str(
        selected_items[it][1]) + " )", fg="white", bg="#212121").pack()
if total_cost:
    store(datetime.now(), username, total_cost, r)
return t

# Add items to the bill
def add_item(item, itemno):
    cost = get_cost(item)

    selected_items.append((item, itemno, cost))
    return

# Show the various items added to the bill when refresh is clicked
def two(tree):
    tree.delete(*tree.get_children())
    for row in selected_items:
        tree.insert("", 0, text=row[0], values=(row[1], row[2]))

# Show the various items added to the bill
def show_items(tab_main, username):
    t = Frame(tab_main, background="#212121")
    tree = ttk.Treeview(t)
    b = Button(t, text="REFRESH", command=lambda: two(
        tree), fg="white", bg="#1966ff")
    b2 = Button(t, text="SUBMIT", command=lambda: get_bill(
        username), fg="white", bg="#1966ff")
    reset = Button(t, text="RESET", command=lambda: resetf(),
        fg="white", bg="#1966ff")
    tree["columns"] = ("one", "two")
    tree.heading("#0", text="NAME")
    tree.heading("one", text="NO")
    tree.heading("two", text="COST")
    tree.column("#0", anchor=CENTER)
    tree.column("one", anchor=CENTER)
    tree.column("two", anchor=CENTER)
    b.pack()
    b2.pack()
    reset.pack()
    tree.pack()
    return t

# Remove bill
def remove_bill(tab_main, username):
```

COUNTER BILLING MANAGEMENT SYSTEM

```
t = Frame(tab_main, background="#212121")
l = Label(t, text="ENTER REFNO:", fg="white", bg="#212121")
e = Entry(t)
b = Button(t, text="DELETE", fg="white", bg="#1966ff",
           command=lambda: delete_bill(e.get()))
l.pack()
e.pack()
b.pack()
tree = ttk.Treeview(t)
tree["columns"] = ("two", "three")
tree.heading("#0", text="DATE/TIME")
```

```
tree.heading("two", text="AMOUNT")
tree.heading("three", text="REFNO")
tree.column("#0", anchor=CENTER)
tree.column("two", anchor=CENTER)
tree.column("three", anchor=CENTER)
b = Button(t, text="SHOW", command=lambda: sal_del(
    tree, username), fg="white", bg="#1966ff")
b.pack()
tree.pack()
return t
```

View deleted bills

```
def deleted_bills(tab_main, username):
    t = Frame(tab_main, background="#212121")
    tree = ttk.Treeview(t)
    b = Button(t, text="SHOW", command=lambda: saldellist(
        tree), fg="white", bg="#1966ff")
    tree["columns"] = ("one", "two", "three")
    tree.heading("#0", text="DATE/TIME")
    tree.heading("one", text="EMPLOYEE ID")
    tree.heading("two", text="AMOUNT")
    tree.heading("three", text="REFNO")
    tree.column("#0", anchor=CENTER)
    tree.column("one", anchor=CENTER)
    tree.column("two", anchor=CENTER)
    tree.column("three", anchor=CENTER)
    b.pack()
    tree.pack()
    return t
```

Check and delete bill

```
def delete_bill(ref):
    if ref and ref.isnumeric():
        delete_bill_db(int(ref))
        popup("DONE")
```


COUNTER BILLING MANAGEMENT SYSTEM

```
else:  
    popup("ENTER VALID DETAILS")
```

5.2.5 SALES

Showing the list of sales

```
def sal(tree):  
    tree.delete(*tree.get_children())  
    rows = get_all_sales()  
    for i in rows:  
        tree.insert("", 0, text=i[0], values=(i[1], i[2], i[3]))
```

List sales

```
def sales(tab_main, username):  
    t = Frame(tab_main, background="#212121")  
    tree = ttk.Treeview(t)  
    b = Button(t, text="SHOW", command=lambda: sal(  
        tree), fg="white", bg="#1966ff")  
    tree["columns"] = ("one", "two", "three")  
    tree.heading("#0", text="DATE/TIME")  
    tree.heading("one", text="EMPLOYEE ID")  
    tree.heading("two", text="AMOUNT")  
    tree.heading("three", text="REFNO")  
    tree.column("#0", anchor=CENTER)  
    tree.column("one", anchor=CENTER)  
    tree.column("two", anchor=CENTER)  
    tree.column("three", anchor=CENTER)  
    b.pack()  
    tree.pack()  
    return t
```

5.2.6 USER

Validate and add user

```
def ck_and_add(username, name, password, phno):  
    row = get_user_details(username)  
    if name and password and username.isnumeric() and phno.isnumeric() and row == None:  
        add_user_to_db(int(username), name, password, int(phno))  
        popup("DONE")  
    else:  
        popup("PLEASE ENTER VALID DETAILS")
```

Add user

```
def add_user(tab_main, username):
```

COUNTER BILLING MANAGEMENT SYSTEM

```
t = Frame(tab_main, background="#212121")
l1 = Label(t, text="Enter user id: ", fg="white", bg="#212121")
e1 = Entry(t)
l2 = Label(t, text="Enter name: ", fg="white", bg="#212121")
e2 = Entry(t)
l3 = Label(t, text="Enter user password: ", fg="white", bg="#212121")
e3 = Entry(t)
l4 = Label(t, text="Enter PHNO: ", fg="white", bg="#212121")
e4 = Entry(t)
b = Button(t, text="ADD", command=lambda: ck_and_add(e1.get(), e2.get(), e3.get(), e4.get()),
           fg="white", bg="#1966ff")
l1.grid(row=0, column=0)
e1.grid(row=0, column=1)
l2.grid(row=1, column=0)
e2.grid(row=1, column=1)
l3.grid(row=2, column=0)
e3.grid(row=2, column=1)
l4.grid(row=3, column=0)
e4.grid(row=3, column=1)
b.grid(row=4, column=1)
return t

# Check and delete user
def exec_del_user(uid):
    if uid.isnumeric():
        remove_user(int(uid))
        popup("DONE")
    else:
        popup("ENTER VALID DETAILS")

# Delete user
def delete_user(tab_main, username):
    t = Frame(tab_main, background="#212121")
    l1 = Label(t, text="Enter user id: ", fg="white", bg="#212121")
    e1 = Entry(t)
    b = Button(t, text="REMOVE", command=lambda: exec_del_user(
        e1.get()), fg="white", bg="#1966ff")
    l1.grid(row=0, column=0)
    e1.grid(row=0, column=1)
    b.grid(row=2, column=1)
    return t

# Showing the list of sales for specific user
def sal_del(tree, username):
    tree.delete(*tree.get_children())
    rows = get_all_sales_related(int(username))

    for i in rows:
        tree.insert("", 0, text=i[0], values=(i[2], i[3]))
```

5.2.7 ITEMS

Showing the list of items

```
def show_item_list(tree):
    tree.delete(*tree.get_children())
    rows = get_items()
    for i in rows:
        tree.insert("", 0, text=i[0], values=(i[1], i[2]))
```

List items

```
def list_items(tab_main, username):
    t = Frame(tab_main, background="#212121")
    tree = ttk.Treeview(t)
    b = Button(t, text="SHOW", command=lambda: show_item_list(
        tree), fg="white", bg="#1966ff")
    tree["columns"] = ("one", "two")
    tree.heading("#0", text="ID")
    tree.heading("one", text="NAME")
    tree.heading("two", text="COST")
    tree.column("#0", anchor=CENTER)
    tree.column("one", anchor=CENTER)
    tree.column("two", anchor=CENTER)
    b.pack()
    tree.pack()
    return t
```

Validate and add item

```
def ck_and_add_item(itemid, name, cost):
    row = ck_item_exists(itemid)
    if name and itemid.isnumeric() and cost.isnumeric() and row == None:
        add_item_to_db_data(int(itemid), name, int(cost))
        popup("DONE")
    else:
        popup("PLEASE ENTER VALID DETAILS")
```

Add items to database

```
def add_item_to_db(tab_main, username):
    t = Frame(tab_main, background="#212121")
    l1 = Label(t, text="Enter item id: ", fg="white", bg="#212121")
    e1 = Entry(t)
    l2 = Label(t, text="Enter name: ", fg="white", bg="#212121")
    e2 = Entry(t)
    l3 = Label(t, text="Enter cost: ", fg="white", bg="#212121")
    e3 = Entry(t)
    b = Button(t, text="ADD", command=lambda: ck_and_add_item(e1.get(), e2.get(), e3.get()),
```

COUNTER BILLING MANAGEMENT SYSTEM

```
        fg="white", bg="#1966ff")
l1.grid(row=0, column=0)
e1.grid(row=0, column=1)
l2.grid(row=1, column=0)
e2.grid(row=1, column=1)
l3.grid(row=2, column=0)
e3.grid(row=2, column=1)
b.grid(row=4, column=1)
return t

# Check and edit price
def ck_and_edit_price(item, price):
    if item and price.isnumeric():
        update_price(item, int(price))
        popup("DONE")
    else:
        popup("PLEASE ENTER VALID DETAILS")

# Change price of item
def change_price_of_item(tab_main, username):
    t = Frame(tab_main, background="#212121")
    rows = get_items()
    options = []
    for i in rows:
        options.append(i[1])
    clicked = StringVar()
    if(len(options)):
        clicked.set(options[0])
        drop = OptionMenu(t, clicked, *options)
        l = Label(t, text="ENTER NEW PRICE:", fg="white", bg="#212121")
        c = Entry(t)
        b = Button(t, text="UPDATE PRICE", command=lambda: ck_and_edit_price(
            clicked.get(), c.get()), fg="white", bg="#1966ff")
        drop.pack()
        l.pack()
        c.pack()
        b.pack()
    return t

# Check and delete item
def exec_del_item(iid):
    if iid.isnumeric():
        remove_item(int(iid))
        popup("DONE")
    else:
        popup("ENTER VALID DETAILS")
```

COUNTER BILLING MANAGEMENT SYSTEM

Delete item

```
def delete_item(tab_main, username):
    t = Frame(tab_main, background="#212121")
    l1 = Label(t, text="Enter item id: ", fg="white", bg="#212121")
    e1 = Entry(t)
    b = Button(t, text="REMOVE", command=lambda: exec_del_item(
        e1.get()), fg="white", bg="#1966ff")
    l1.grid(row=0, column=0)
    e1.grid(row=0, column=1)
    b.grid(row=2, column=1)
    return t
```

5.2.8 data.py

```
import mysql.connector
import datetime
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="*****",
    database="createdb",
    port=3306
)
cur = conn.cursor()

# xyz
def ck_details_emp(username, password):
    try:
        cur.execute(
            "SELECT * FROM EMP_DETAILS WHERE EID='{}' AND
PASSWORD='{}'".format(username, password))
        row = cur.fetchone()
        if row != None:
            return True
        else:
            return False
    except Exception:
        return False

# Auth admin
def ck_details_admin(username, password):
    try:
        cur.execute(
            "SELECT * FROM ADMIN_DETAILS WHERE AID='{}' AND
PASSWORD='{}'".format(username, password))
```

COUNTER BILLING MANAGEMENT SYSTEM

```
row = cur.fetchone()
if row != None:
    return True
else:
    return False
except Exception:
    return False
```

Get user details

```
def get_user_details(username):
    try:
        cur.execute(
            "SELECT EID, NAME, PHNO FROM EMP_DETAILS WHERE
EID='{ }'.format(username))
        row = cur.fetchone()
        return row
    except Exception:
        return False
```

Get admin details

```
def get_admin_details(username):
    try:
        cur.execute(
            "SELECT AID, NAME, PHNO FROM ADMIN_DETAILS WHERE
AID='{ }'.format(username))
        row = cur.fetchone()
        return row
    except Exception:
        return False
```

Get all item details

```
def get_items():
    try:
        cur.execute("SELECT * FROM ITEM_DETAILS ORDER BY IID ASC")
        rows = cur.fetchall()
        return rows
    except Exception:
        return False
```

Get cost of an item

```
def get_cost(item):
    try:
        cur.execute("SELECT * FROM ITEM_DETAILS WHERE NAME='{ }'.format(item))
        rows = cur.fetchone()
```

COUNTER BILLING MANAGEMENT SYSTEM

```
    return rows[2]
except Exception:
    return False
```

Store bill permanently

```
def store(datetimev, username, total_cost, ref):
    sql = "INSERT INTO BILLS VALUES ('{}', {}, {}, {})".format(
        datetimev.strftime("%d-%m-%Y %H:%M"), username, total_cost, ref)
    cur.execute(sql)
    conn.commit()
```

Get all employees

```
def get_all_employees():
    cur.execute("SELECT * FROM EMP_DETAILS ORDER BY EID ASC")
    rows = cur.fetchall()
    return rows
```

Get all sales

```
def get_all_sales():
    cur.execute("SELECT * FROM BILLS")
    rows = cur.fetchall()
    return rows
```

Add user

```
def add_user_to_db(username, name, password, phno):
    sql = "INSERT INTO EMP_DETAILS VALUES ({}, '{}', '{}', {})".format(
        username, name, password, phno)
    cur.execute(sql)
    conn.commit()
```

Check if item exists

```
def ck_item_exists(id):
    try:
        cur.execute("SELECT * FROM ITEM_DETAILS WHERE IID='{}'.format(id))
        row = cur.fetchone()
        return row
    except Exception:
        return False
```

Add item to database

```
def add_item_to_db_data(itemid, name, cost):
    try:
        sql = "INSERT INTO ITEM_DETAILS VALUES ({}, '{}', {})".format(
```

COUNTER BILLING MANAGEMENT SYSTEM

```
        itemid, name, cost)
    cur.execute(sql)
    conn.commit()
except Exception:
    return False
```

Remove user from database

```
def remove_user(username):
    try:
        sql = "DELETE FROM EMP_DETAILS WHERE EID={}".format(username)
        cur.execute(sql)
        conn.commit()
    except Exception:
        return False
```

Remove item from database

```
def remove_item(id):
    try:
        sql = "DELETE FROM ITEM_DETAILS WHERE IID={}".format(id)
        cur.execute(sql)
        conn.commit()
    except Exception:
        return False
```

Update price of an item

```
def update_price(item_name, cost):
    try:
        sql = "UPDATE ITEM_DETAILS SET COST={} WHERE NAME='{}'.format(
            cost, item_name)
        cur.execute(sql)
        conn.commit()
    except Exception:
        return False
```

Get all related sales

```
def get_all_sales_related(username):
    try:
        cur.execute("SELECT * FROM BILLS WHERE EID={}".format(username))
        rows = cur.fetchall()
        return rows
    except Exception:
        return False
```

Delete bill from database

COUNTER BILLING MANAGEMENT SYSTEM

```
def delete_bill_db(ref):
    try:
        sqlpre = "INSERT INTO DELETED_BILLS SELECT * FROM BILLS WHERE
REFNO={ }".format(
            ref)
        cur.execute(sqlpre)
        conn.commit()
        sql = "DELETE FROM BILLS WHERE REFNO={ }".format(ref)
        cur.execute(sql)
        conn.commit()
    except Exception:
        return False

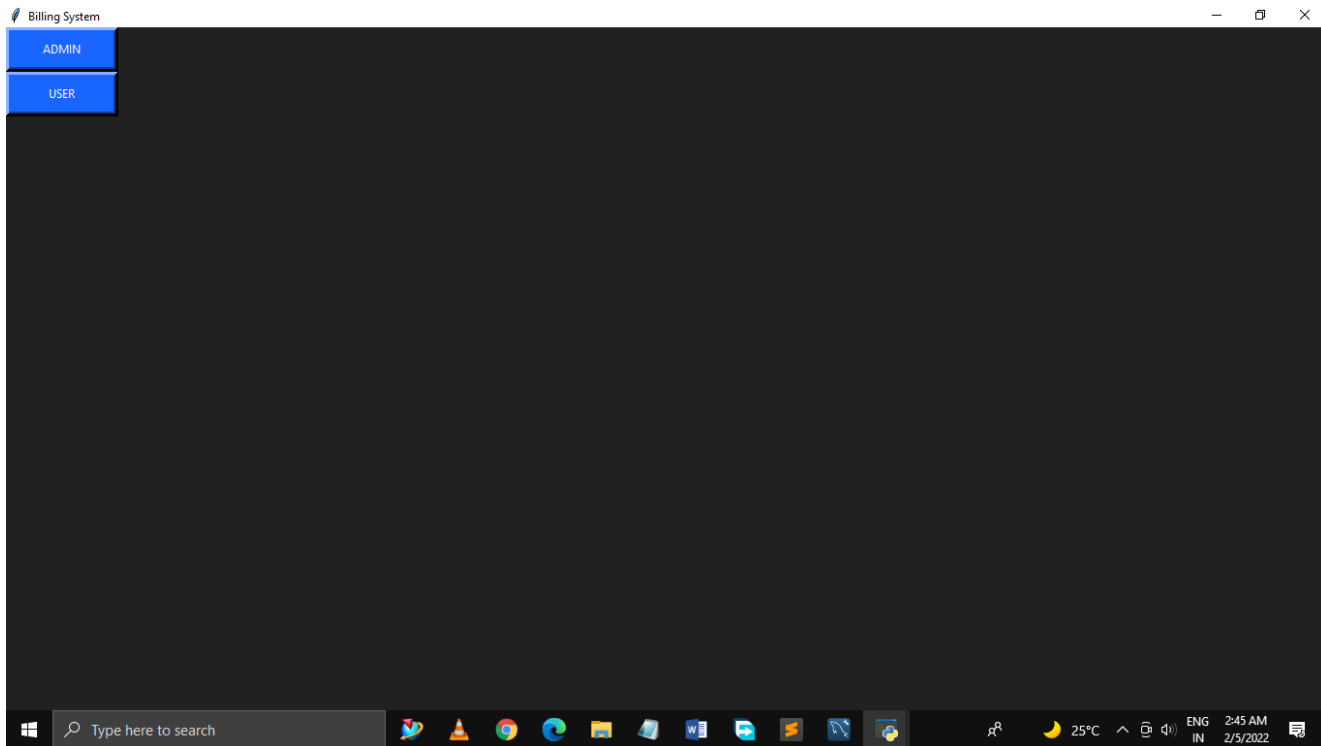
# Get all deleted bills
def get_all_deleted():
    try:
        sql = "SELECT * FROM DELETED_BILLS ORDER BY DATETIME ASC"
        cur.execute(sql)
        rows = cur.fetchall()
        return rows
    except Exception:
        return []
```

CHAPTER-6

SNAPSHOTS

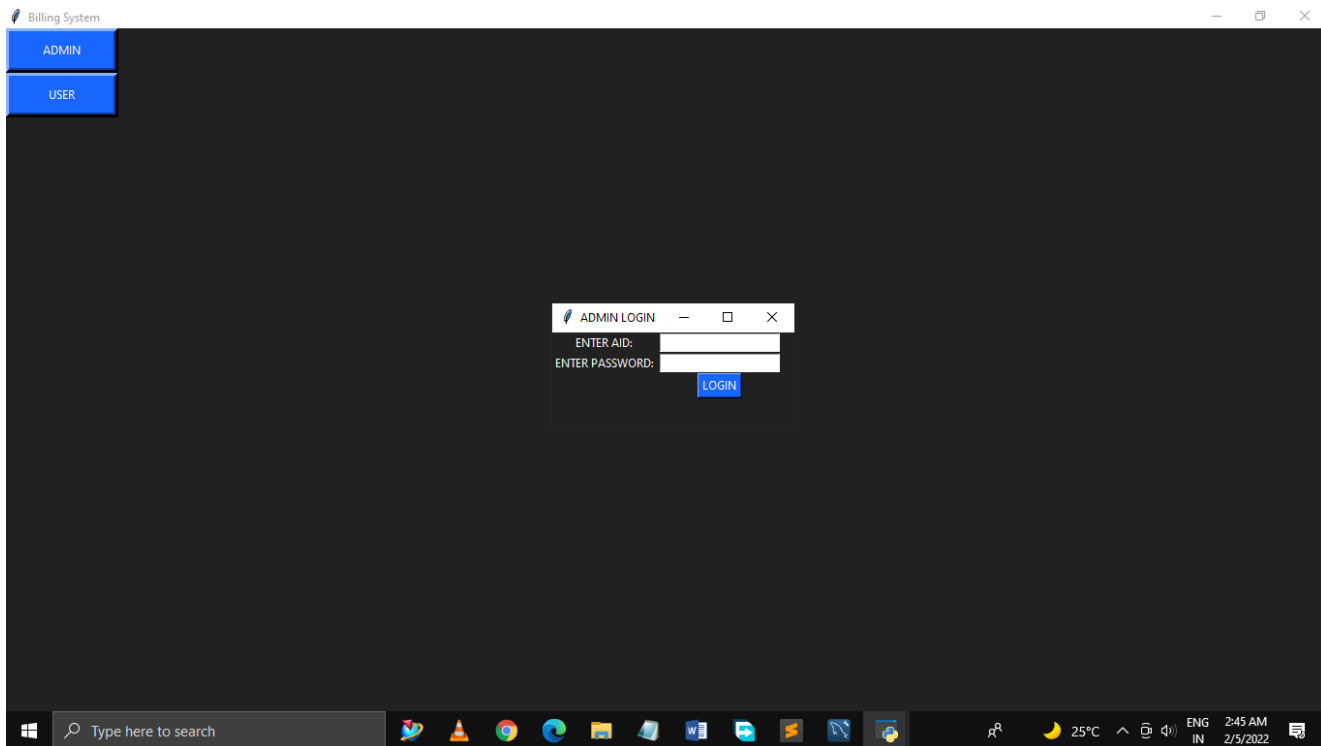
COUNTER BILLING MANAGEMENT SYSTEM

HOME LOGIN



6.1.1 SNAPSHOT 1

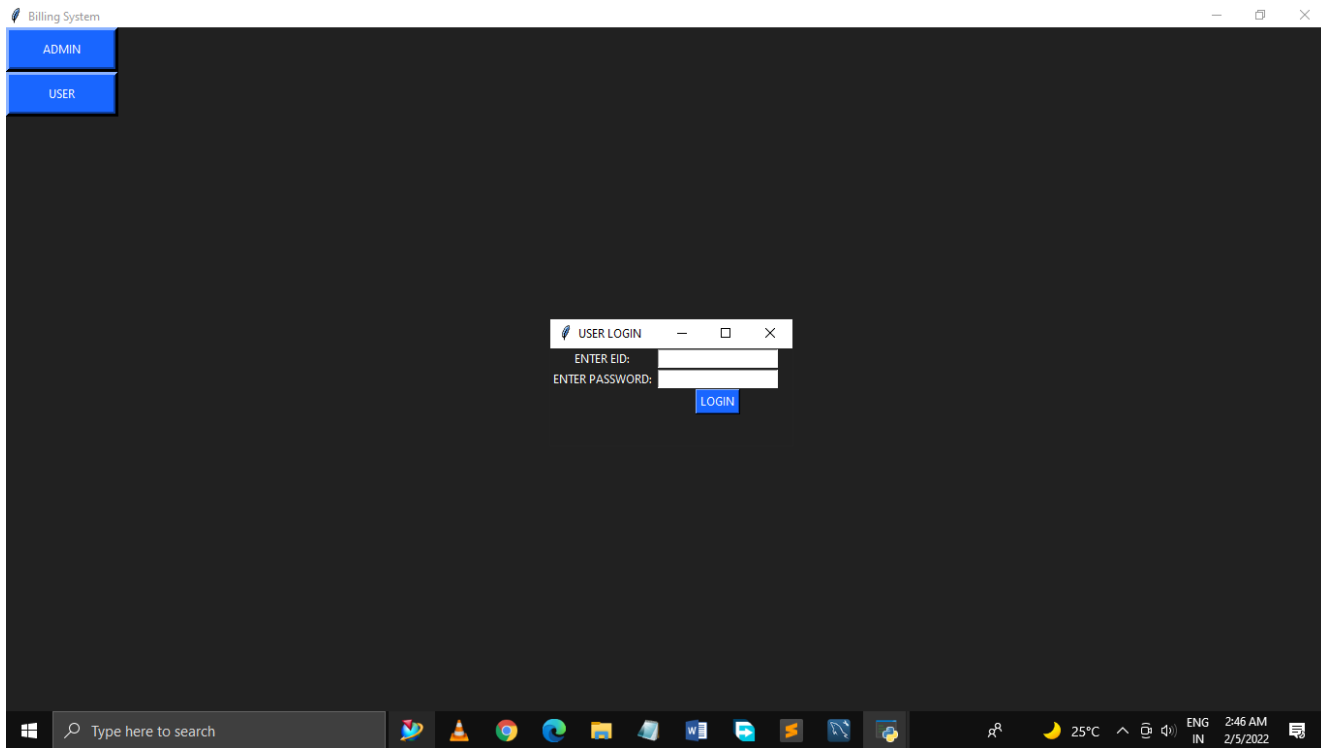
ADMIN LOGIN



6.1.2 SNAPSHOT 2

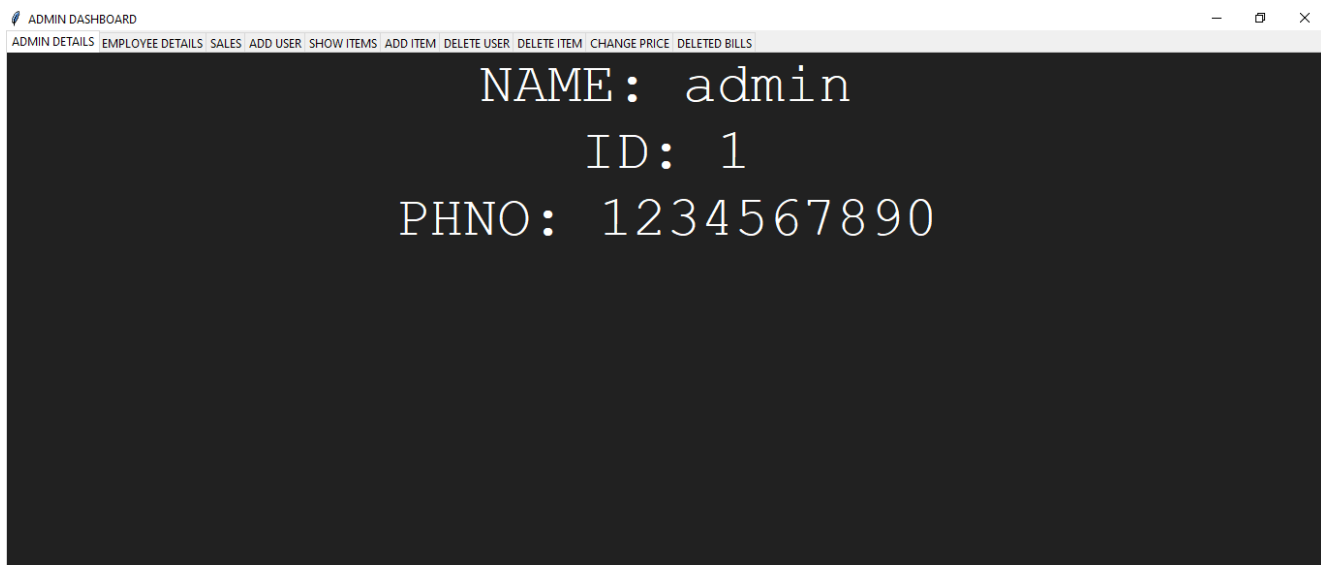
COUNTER BILLING MANAGEMENT SYSTEM

USER LOGIN



6.1.3 SNAPSHOT 3

ADMIN DASHBOARD



6.1.4 SNAPSHOT 4

COUNTER BILLING MANAGEMENT SYSTEM

EMPLOYEE DETAILS

The screenshot shows a web application titled 'ADMIN DASHBOARD'. The navigation bar includes links for 'ADMIN DETAILS', 'EMPLOYEE DETAILS', 'SALES', 'ADD USER', 'SHOW ITEMS', 'ADD ITEM', 'DELETE USER', 'DELETE ITEM', 'CHANGE PRICE', and 'DELETED BILLS'. The 'EMPLOYEE DETAILS' link is active. A 'SHOW' button is visible. Below the button, a table displays employee information:

ID	NAME	PHNO
13	mun	987654452
12	min	234567567
11	me	234567

The Windows taskbar at the bottom shows the search bar, taskbar icons, and system tray with the date 2/5/2022 and time 2:47 AM.

6.1.5 SNAPSHOT 5

USER DASHBOARD

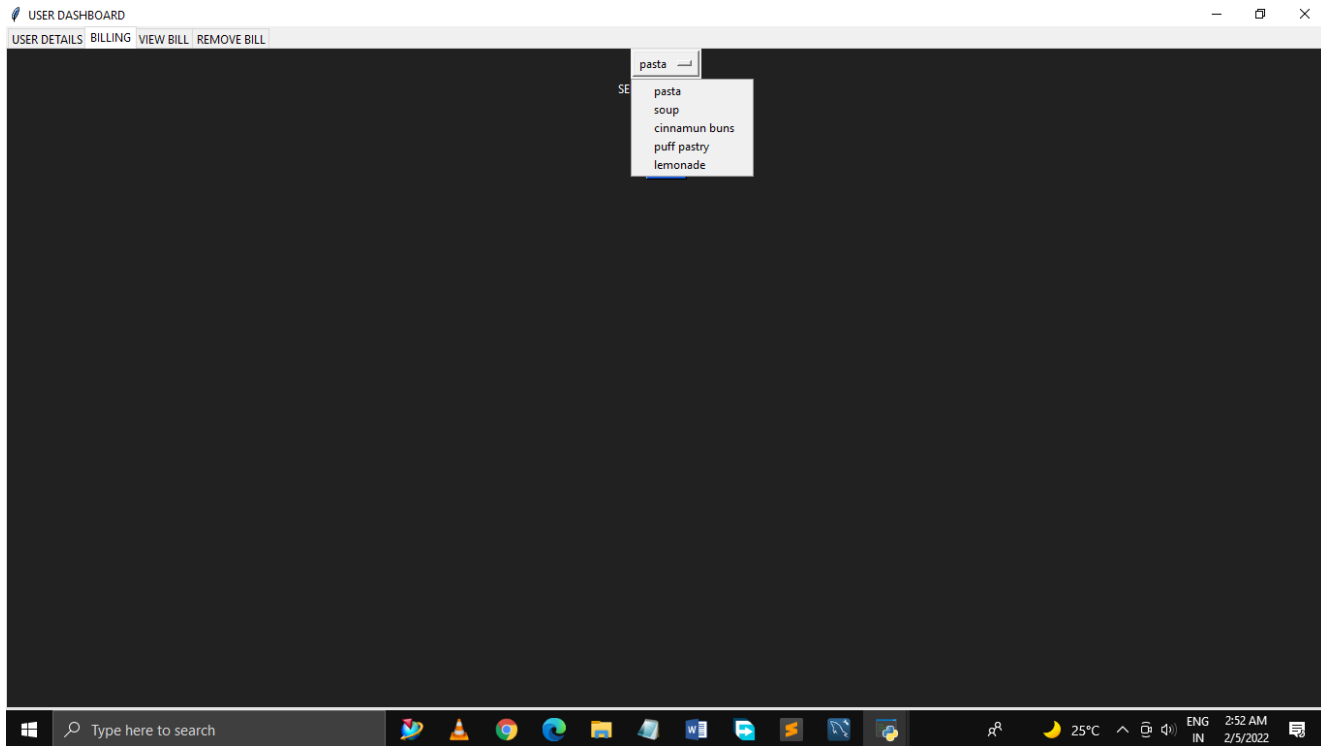
The screenshot shows a web application titled 'USER DASHBOARD'. The navigation bar includes links for 'USER DETAILS', 'BILLING', 'VIEW BILL', and 'REMOVE BILL'. The 'USER DETAILS' link is active. The main content area displays the following information:

NAME: mun
ID: 13
PHNO: 987654452

6.1.6 SNAPSHOT 6

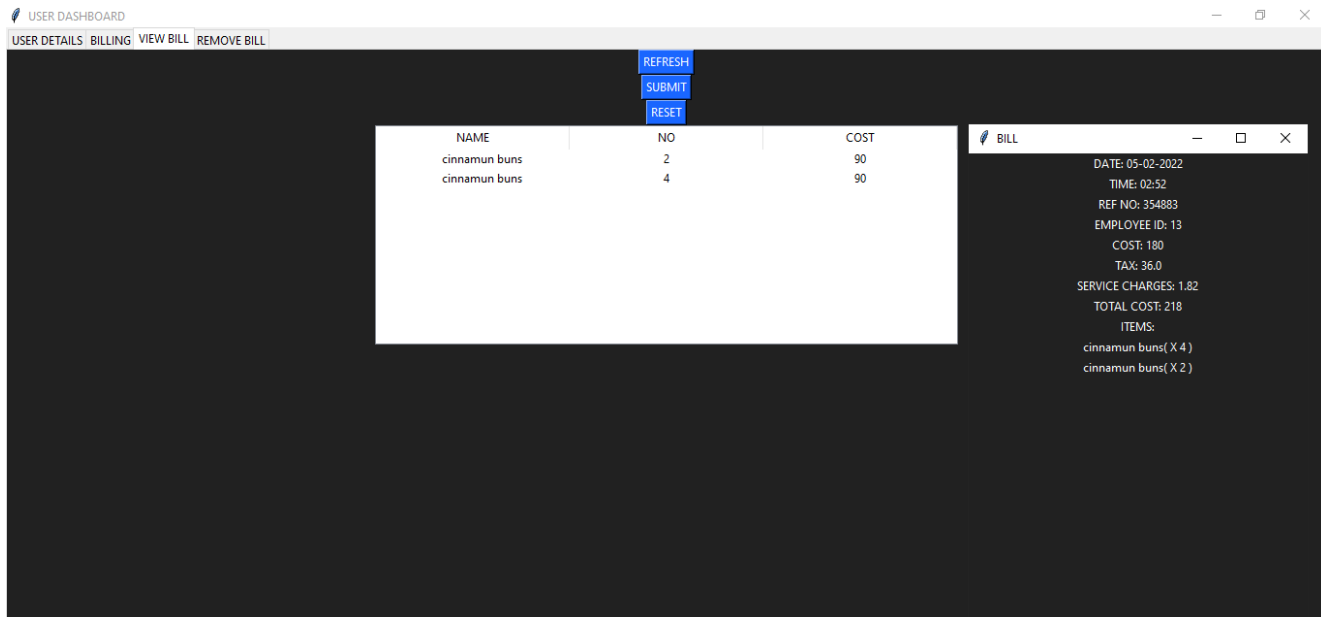
COUNTER BILLING MANAGEMENT SYSTEM

BILLING



6.1.7 SNAPSHOT 7

VEIWINING BILL



6.1.8 SNAPSHOT 8

COUNTER BILLING MANAGEMENT SYSTEM

REMOVING BILL

USER DASHBOARD

USER DETAILS BILLING VIEW BILL REMOVE BILL

ENTER REFNO:

354883

DELETE

SHOW

DATE/TIME	AMOUNT	REFNO
05-02-2022 02:53	218.0	354883

6.1.9 SNAPSHOT 9

CHANGING PRICE OF AN ITEM

ADMIN DASHBOARD

ADMIN DETAILS EMPLOYEE DETAILS SALES ADD USER SHOW ITEMS ADD ITEM DELETE USER DELETE ITEM CHANGE PRICE DELETED BILLS

lemonade

ENTER NEW PRICE:

55

UPDATE PRICE

6.1.10 SNAPSHOT 10

COUNTER BILLING MANAGEMENT SYSTEM

LISITNG DELETED BILLS

ADMIN DASHBOARD

- □ X

ADMIN DETAILS EMPLOYEE DETAILS SALES ADD USER SHOW ITEMS ADD ITEM DELETE USER DELETE ITEM CHANGE PRICE DELETED BILLS

SHOW

DATE/TIME	EMPLOYEE ID	AMOUNT	REFNO
05-02-2022 02:53	13	218.0	354883
05-02-2022 02:52	13	218.0	354883

6.1.11 SNAPSHOT 11

CHAPTER-7

APPLICATIONS

7. APPLICATIONS

7.1 APPLICATIONS COUNTER BILLING MANAGEMENT SYSYTEM

- Accounting
- Legal
- SAAS (software-as-a-service)
- Retail
- Manufacturing
- Recurring billing
- Invoicing capability
- Reports and notifications
- Data integration
- Company logo display on invoices
- Project billing
- Hourly billing
- Late payment reporting and alerts
- Hours per project and per client tracking
- Expense tracking and insertion in invoice
- Non-standard payment terms support
- Multiple tax rates support
- Multiple currency support
- Reports by client, project, and date
- Database encryption

CHAPTER-8

CONCLUSION

8. CONCLUSION

8.1 CONCLUSION

Counter billing management system plays effective role maintaining the records, no data is lost. Bills that have been deleted due to some uncertainty are also listed and admin can keep track of those bills. This makes the billing management runs smooth by removing the unnecessary detail from it.

It is very easy to use. It is user friendly. Less efforts are required to operate the system. Admin keeps track and manages all the records, bills of the management system. Checking of the items, employees, sales of the items and demand of the items can be got to know from this easily by retrieving the required data.

Counter billing management system saves time of both users and the customers. It generates bills, so customers have a reference to what they have bought at what price.

7. APPLICATIONS

8. CONCLUSION

8.1 CONCLUSION

This project serves as a wonderful alternative to brick-and-mortar financial institutions and gives a new dimension to the field of banking and loans. Loan Management System being an automated interface makes customer-employee communication much more efficient. This system reduces the heavy paper work needed to issue loans traditionally and makes the process efficient and transparent.

Developing this project came with enormous amount of learning, enabling me to acquire new skills as well as experimenting with my already acquired skills. It improved my skills in database management system helping me learn various perks of organized data.