A Project Report On

ELECTRICITY BILL MANAGEMENT SYSTEM

Submitted In Partial Fulfilment Of The

Requirements for the Award of the Degree Of

BACHELOR OF COMPUTER APPLICATION

By

ANKIT KUMAR

AJU/190534

Under The Guidance Of

MS. SNEHA KASHYAP

Internal Guide

DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

ARKA JAIN UNIVERSITY, JHARKHAND

JAMSHEDPUR

2019-2022

ELECTRICITY BILL MANAGEMENT SYSTEM

A Project Report

Submitted In Partial Fulfilment Of The

Requirements for the Award of the Degree Of

BACHELOR OF COMPUTER APPLICATION

By

ANKIT KUMAR

AJU/190534

Under The Guidance Of

MS. SNEHA KASHYAP

Internal Guide

DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

ARKA JAIN UNIVERSITY, JHARKHAND

JAMSHEDPUR

2019-2022

ARKA JAIN UNIVERSITY

JAMSHEDPUR, JHARKHAND
DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project entitled **ELECTRICITY BILL MANAGEMENT SYSTEM**, is bona-fide work of **ANKIT KUMAR** bearing Registration Number: **AJU/190534** submitted in partial fulfilment of the requirement for the award of degree in BACHELOR OF COMPUTER APPLICATION from ARKA JAIN UNIVERSITY, JHARKHAND

Internal Guide

Head of Department

University Seal

Date: 26/05/2022

accenture

Forage

Inspiring and empowering future professionals

Ankit Kumar

Developer Program

Certificate of Completion of Internship

January 8th, 2022 – May 2nd, 2022

Project: Electricity Bill Management System

Over the period of January 2022 - May 2022, Ankit Kumar has completed practical task modules in:

Define technical requirements
Design changes to an existing architecture
Scale on-premise system infrastructure to the cloud Reading and understanding code Attention to detail

Debugging algorithms
Unit testing
User Acceptance Testing - UAT
Security maturity assessment IAM policies and permissions

Securing the software development lifecycle (SDLC)
Shaping the Problem Data and privacy

Johnny Rosenby
Nordic Recruitment Director

Tom Brunskill CEO, Co-Founder of Forage

Enrolment Verification Code YMsRMaYwThyR3Cmo9 | User Verification Code fXtKneQdYW2KuoCXf | Issued by **Forage**

ABSTRACT

Electricity Billing System Project in Python is a software-based application developed in Python programming language. The project aims at serving the department of electricity by computerizing the billing system. It mainly focuses on the calculation of Units consumed during the specified time and the money to be paid to electricity offices. This computerized system will make the overall billing system easy, accessible, comfortable and effective for the consumers of this project.

The system excludes the need of maintaining paper electricity bill, administrator does not have to keep a manual track of the users. Thus, it saves human efforts and resources.

Tools like Python and My-SQL Server are hi-tech technologies which have been used to develop this project for high performance. Python being widely used as an OOP language gives the leverage in developing this projects because of its powerful packages.

MySQL is an open-source relational database management system used for storing data of the Discoms. The MySQL server provides a database management system with querying and connectivity capabilities, as well as the ability to have excellent data structure and integration with many different platforms

ACKNOWLEDGEMENT

It is a genuine pleasure to express my profound gratitude and deep regards to my Internal Guide "**MS. SNEHA KASHYAP**" and our HOD **"Prof. DR. ARVIND KUMAR PANDEY"** for their exemplary guidance, monitoring and constant encouragement. I would like to express my special thanks to **ARKA JAIN UNIVERSITY** who gave me the golden opportunity to do this wonderful project on the topic **"ELECTRICITY BILL MANAGEMENT SYSTEM"**, which helped me in doing a lot of Research and I came to know about so many new things.

Finally many thanks to my friends, who have helped and given me suggestions, supports and corrections throughout the project.

DECLARATION

I hereby declare that the Project entitled "ELECTRICITY BILL MANAGEMENT SYSTEM" done at ARKA JAIN UNIVERSITY has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

This project is done in partial fulfilment of the requirement for the award of degree of BACHELOR OF COMPUTER APPLICATION to be submitted as the final semester project as part of our curriculum

ANKIT KUMAR

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 12

1.1. OVERVIEW: 12

OBJECTIVE: 12

PURPOSE AND SCOPE: 13

CHAPTER 2: SURVEY OF TECHNOLOGIES 14

CHAPTER 3: SYSTEM ANALYSIS 15

3.1. SYSTEM ANALYSIS 15

3.2. EXIXTING SYSTEM 16

3.3. Proposed System 16

3.4. FEASIBILITY STUDY 17

3.4.1. Technical Feasibility 17

3.4.2. Economic Feasibility 17

3.4.3. Operational Feasibility 18

3.5. Hardware Requirements 18

3.6. Software Requirements 18

3.7. Conceptual Models 19

3.7.1. SDLC 19

3.8. Gantt Chart 21

3.9. Data Flow Diagrams (DFDs) 22

3.9.1. Level 0 DFD 22

3.9.2. Level 1 DFD 23

3.9.3. Level 2 DFD 25

CHAPTER 4: SYSTEM DESIGN 28

4.1. Basic Modules: 28

Functionality of Admin: 28

Functionality of Department Users of the Discoms: 28

4.2. Database Normalization 29

4.2.1. First Normal Form 29

4.2.2. Second Normal Form 29

4.2.3 Third Normal form 30

4.3. Data Dictionary 31

4.4. ER-Diagram 33

CODING DETAILS AND EFFICIENCY 34
LOGIN PAGE: 34
ADMIN HOMEPAGE 42
DEPT. USER HOMEPAGE 79
Testing Approach 100
Type of Testing 100
Test case 101
Use Case 104
Use Case Diagram 105
CHAPTER 6: RESULTS AND DISCUSSIONS 106
Admin Login 106
User Dept. Login Page 106
Admin Homepage 107
Dept. User Homepage 107
Add Dept. Users 108
Manage Dept. Users 108
Add Customers 109
Delete Customers 109
Bill Generation 110
Change Password 110
Search One Customer 111
Search List Of customers 111
Payment 112
CHAPTER 7: CONCLUSION & FUTURE SCOPE 113
7.1. CONCLUSION 113
7.2 FUTURE SCOPE 113
REFERENCES 114

LIST OF TABLES

Table 3. 1 Hardware Requirements 18
Table 3. 2 Software Requirements 18

Table 4. 1 Level 1 Normalized Table 29
Table 4. 2 Level 2 Normalized Form 29
Table 4. 3 Level 3 Normalized Form 30

Table3.1.1.Dept. User Database 31
Table3.1.2.Customer Database 31
Table3.1.3. Bill Database 31
Table3.1.4.Reading Database 32
Table3.1.5. Tariff Database 32
Table3.1.6 Payment Database 32

LIST OF FIGURES

Figure 3.1 Iterative Enhancement Model 20
Figure 3.2 Gantt chart 21
Figure 3.3 Level Zero DFD 22
Figure 3.4. Level 1 Admin DFD 23
Figure 3.5 Level One DFD Dept. User 24
Figure 3.6. Level 2 ADMIN DFD (2.0) 25
Figure 3.7. Level 2 ADMIN DFD (3.0) 26
Figure 3.8. Level 2 Admin DFD (5.0) 26
Figure 3.9. Level 2 Dept. User DFD (2.0) 27
Figure 3.10. Level 2 Dept. User DFD (4.0) 27

Figure 4.1. ER Diagram 33

CHAPTER 1: INTRODUCTION

1.1. OVERVIEW:

Our project entitled “Electricity Bill Management System” aims is to generate electricity bill with all the charges. Manual system that is employed is extremely laborious and quite inadequate. It only makes the process more difficult and hard.

The aim of our project is to develop a system that is meant to partially computerize the work performed in the Electricity Board like generating monthly electricity bill, record of consuming unit of energy, store record of the customer and previous unpaid record.

We used Tkinter (Python) as front end and Python package MySQL-connector as back end for developing our project.

My SQL Server is a powerful relational database application with which a desktop user can efficiently create and manipulate database systems. Access targets the desktop category and works best for individuals and workgroup

Managing megabytes of data for multi-user access to the same database.

OBJECTIVE:

- Easy maintenance of database and overall project.
- Adopting security measures, by maintaining the login of username and password.
- Reducing data redundancy.
- Reduce workload of staff.
- Reduce the delay in processing time.
- Provide user-friendliness in all possible ways.
- Store data in a centralized location to reduce redundancy and increase consistency.
- Digitalization of bill management
- Digital management of customers records
- Reduce dependency on papers

PURPOSE AND SCOPE:

The manual work processes was time consuming and hence slow. Following are the main drawbacks of the existing system:

- The basic and major drawbacks in the existing system are the speed of retrieval of data from files, which leads to delay.
- Maintenance of voluminous data is very cumbersome and laborious job.
- There are plenty of chances of duplicity of data and information.
- Updating is very tedious job.
- There is no central database from where one can get different statistical data at one place.
- Dependency on papers

The purpose of this project is to overcome these problems by automating the system. Automation of the data maintenance would reduce the manpower and will result in accurate data & above all increase the efficiency of the system.

CHAPTER 2: SURVEY OF TECHNOLOGIES

- **PYTHON**

Python being one of the most used high level language has been used to create this project. Python powerful packages eases the development of the project. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library

- **TKINTER**

Tkinter is a GUI development package used in python. Tkinter has been used for developing the GUI of this project for easy and interactive usage. Tkinter supports a range of Tcl/Tk versions, built either with or without thread support. The official Python binary release bundles Tcl/Tk 8.6 threaded. See the source code for the Tkinter module for more information about supported versions

- **MYSQL SERVER**

MYSQL database is used as a database for the storage and the retrieval of data. MySQL-connector package has been used for connecting to the database. MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database.

CHAPTER 3: SYSTEM ANALYSIS

3.1. SYSTEM ANALYSIS

System analysis is a process of gathering and interpreting facts, diagnosing problems and the information to recommend improvements on the system. It is a problem solving activity that requires intensive communication between the system users and system developers. System analysis or study is an important phase of any system development process. The system is studied to the minutest detail and analysed. The system analyst plays the role of the interrogator and dwells deep into the working of the present system. The system is viewed as a whole and the input to the system are identified. The outputs from the organizations are traced to the various processes. System analysis is concerned with becoming aware of the problem, identifying the relevant and decisional variables, analysing and synthesizing the various factors and determining an optimal or at least a satisfactory solution or program of action. A detailed study of the process must be made by various techniques like interviews, questionnaires etc. The data collected by these sources must be scrutinized to arrive to a conclusion. The conclusion is an understanding of how the system functions. This system is called the existing system. Now the existing system is subjected to close study and problem areas are identified. The designer now functions as a problem solver and tries to sort out the difficulties that the enterprise faces. The solutions are given as proposals. The proposal is then weighed with the existing system analytically and the best one is selected. The proposal is presented to the user for an endorsement by the user. The proposal is reviewed on user request and suitable changes are made. This is loop that ends as soon as the user is satisfied with proposal. Preliminary study is the process of gathering and interpreting facts, using the information for further studies on the system. Preliminary study is problem solving activity that requires intensive communication between the system users and system developers. It does various feasibility studies. In these studies a rough figure of the system activities can be obtained, from which the decision about the strategies to be followed for effective system study and analysis can be taken.

3.2. EXIXTING SYSTEM

In the existing system the bills are manages manually but in proposed system we have to computerize the bill management using this application.

- Lack of security of data.
- More man power.
- Time consuming.
- Consumes large volume of pare work.
- Needs manual calculations.
- No direct role for the higher officials
- Dependency on papers.

3.3. Proposed System

The aim of proposed system is to develop a system of improved facilities. The proposed system can overcome all the limitations of the existing system. The system provides proper security and reduces the manual work.

- Security of data.
- Ensure data accuracy's.
- Proper control of the higher officials.
- Minimize manual data entry.
- Minimum time needed for the various processing.
- Greater efficiency.
- Better service.
- User friendliness and interactive.

3.4. FEASIBILITY STUDY

Feasibility study is made to see if the project on completion will serve the purpose of the organization for the amount of work, effort and the time that spend on it. Feasibility study lets the developer foresee the future of the project and the usefulness. A feasibility study of a system proposal is according to its workability, which is the impact on the organization, ability to meet their user needs and effective use of resources. Thus when a new application is proposed it normally goes through a feasibility study .The document provide the feasibility of the project that is being designed and lists various areas that were considered very carefully during the feasibility study of this project such as Technical, Economic and Operational feasibilities. The following are its features:

3.4.1. Technical Feasibility

The system must be evaluated from the technical point of view first. The assessment of this feasibility must be based on an outline design of the system requirement in the terms of input, output, programs and procedures. Having identified an outline system, the investigation must go on to suggest the type of equipment, required method developing the system, of running the system once it has been designed.

3.4.2. Economic Feasibility

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require.
The following are some of the important financial questions asked during preliminary investigation:

- The costs conduct a full system investigation.
- The cost of the hardware and software.
- The benefits in the form of reduced costs or fewer costly errors.

Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also all the resources are already available, it give an indication of the system is economically possible for development.

3.4.3. Operational Feasibility

In this feasibility study it is determined whether there is need of well qualified operator or simple user. Is there need to train the operator or not? This project is supporting the Graphical User Interface; hence operating this project is so simple. Even a person who has a little knowledge of computer can easily handle this well. There is no need of trained operator.

3.5. Hardware Requirements

Name of Component	Specification
Processor	1.6 GHz or faster processor
RAM	2 GB of RAM (2.5 GB if running on a virtual machine)
Required hard disk drive	5400 RPM hard disk drive
Required hard disk space	10 GB of available hard disk space

Table 3. 1 Hardware Requirements

3.6. Software Requirements

Name of Component	Specification
Operating System	Windows 7 and above
Front End	Tkinter , python
Back End	Python
Database	My SQL Server 2021
Web Server	IIS Server

Table 3. 2 Software Requirements

3.7. Conceptual Models

Conceptual models differ in the degrees of freedom given to the user in accomplishing tasks and achieving goals. In some cases, the task flow is highly structured and linear, thus giving the user less degrees of freedom in terms of alternative ways of the interaction. In other cases, the task flow is unstructured, thus giving the user more degree of freedom in terms of alternatives in how to perform the interaction.

3.7.1. SDLC

In order to develop the project **"ELECTRICITY BILL MANAGEMENT SYSTEM"** we have adopted the **Iterative Enhancement Model** also known as Incremental Model. This model removes the shortcoming of waterfall model. Since many facts of this system are already known. It is not a new concept and hence no research is required. A working version can be easily created and hence the system can start working. Rest of the functionalities can be implemented in the next iteration and can be delivered later. As the requirement analysis is also not required. It not being a new technology risk involved is also less. So, one need not perform detailed risk analysis. If redevelopment Admin is less than development can be started with a smaller number of people and in next increments others can be involved. As this model combines the advantage of waterfall model and prototyping, clients are always aware of the product being delivered and can always suggest changes and enhancements and can get them implemented. As less amount of customercommunication is required one need not apply spiral model in which all types of analysis is done in detail. As the deadline is affordable one need not to for Rapid Application Development model.

Iterative enhancement model is useful when less manpower is available for software development and the release deadlines are specified. It is best suited for in house product development, where it is ensured that the user has something to start with. The complete product is divided into releases and the developer delivers the product release by release.

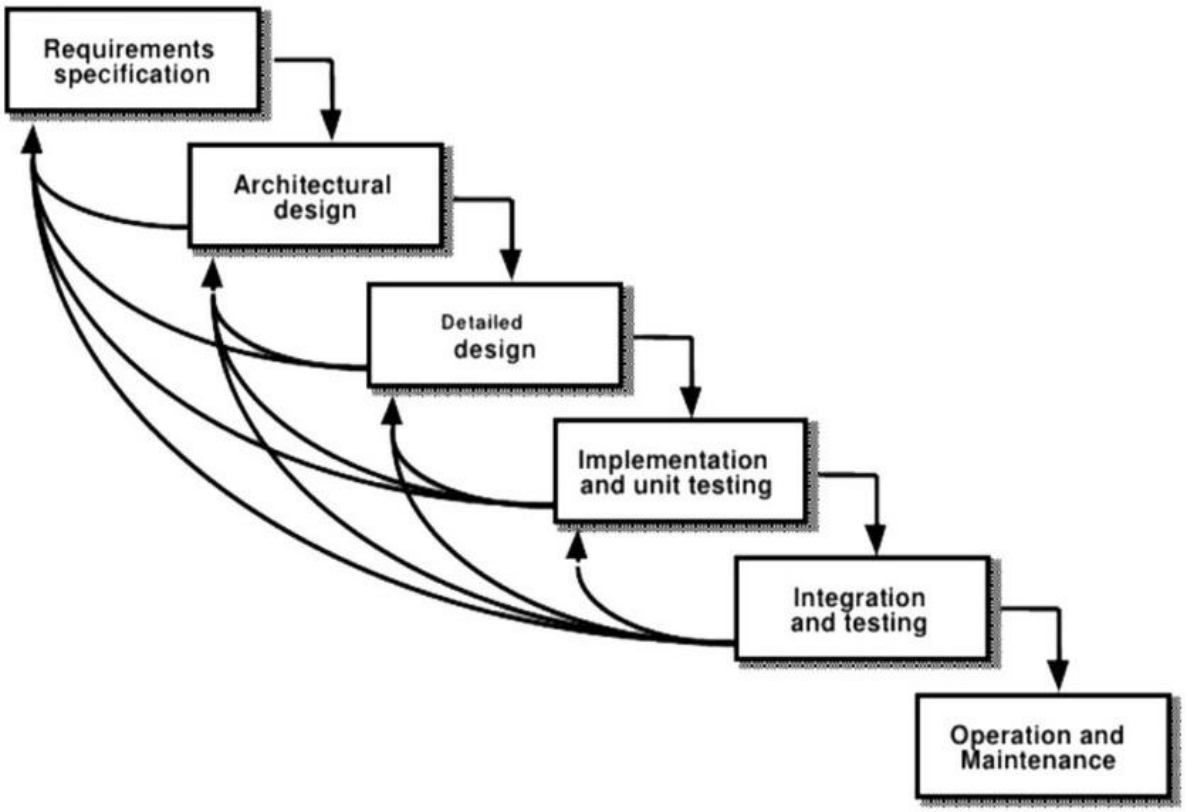


Figure 3.1 Iterative Enhancement Model

3.8. Gantt Chart

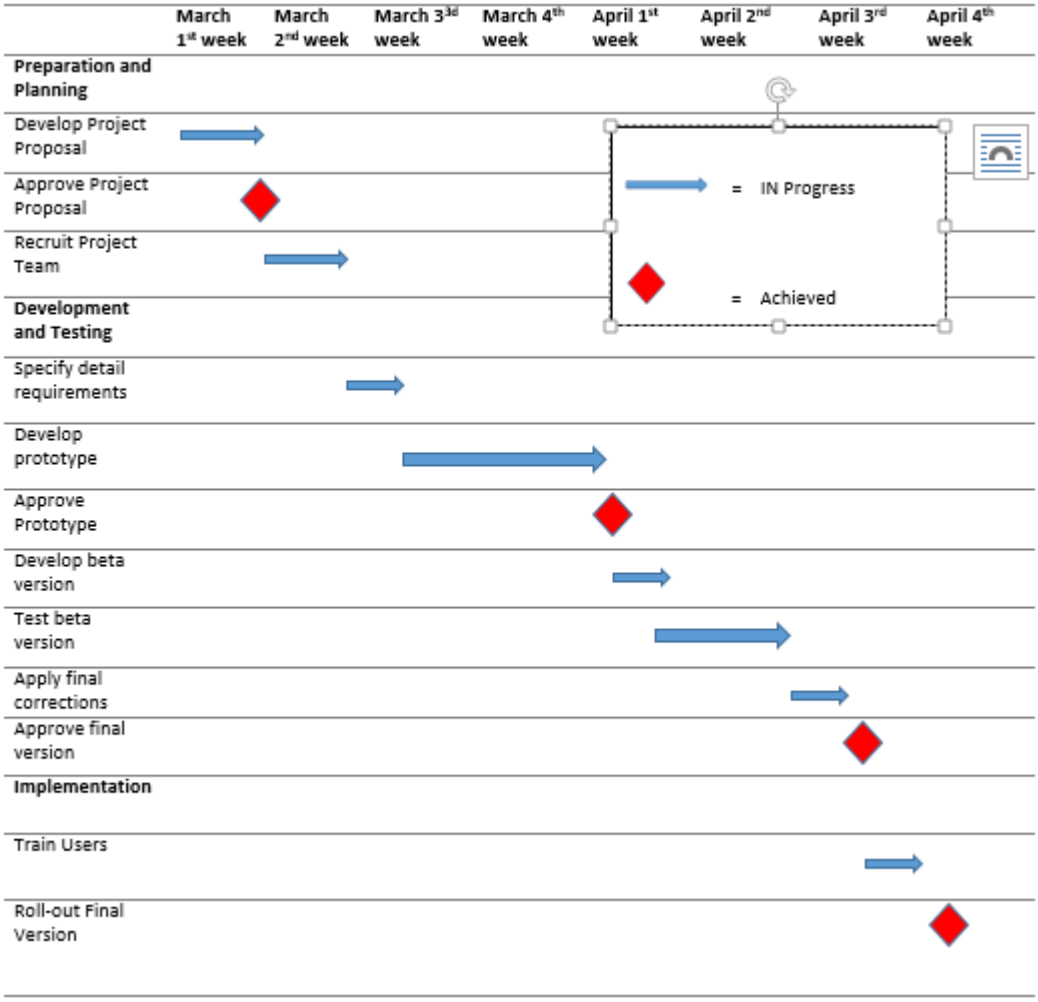


Figure 3.2 Gantt chart

3.9. Data Flow Diagrams (DFDs)

3.9.1. Level 0 DFD

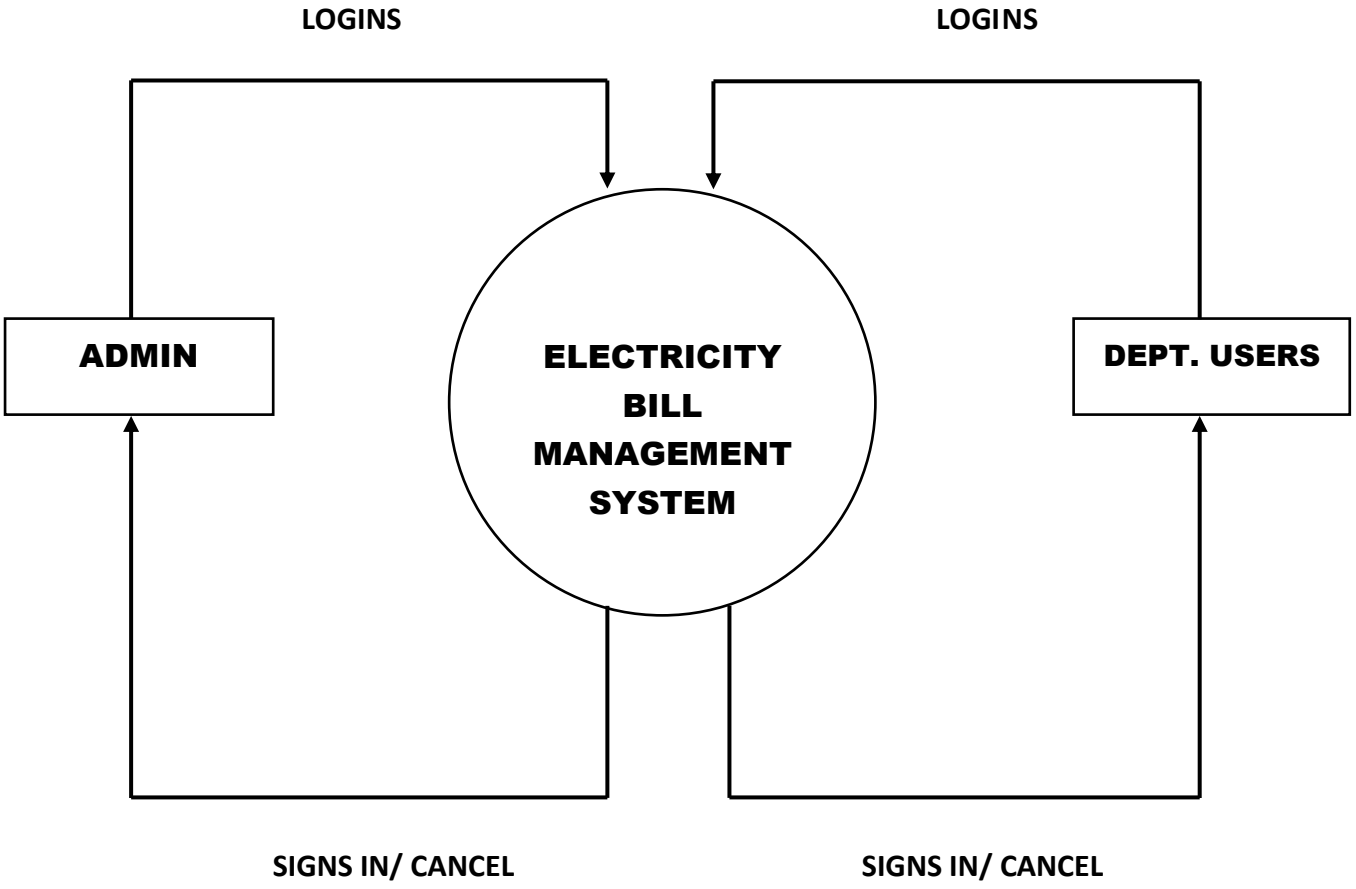


Figure 3.3 Level Zero DFD

3.9.2. Level 1 DFD

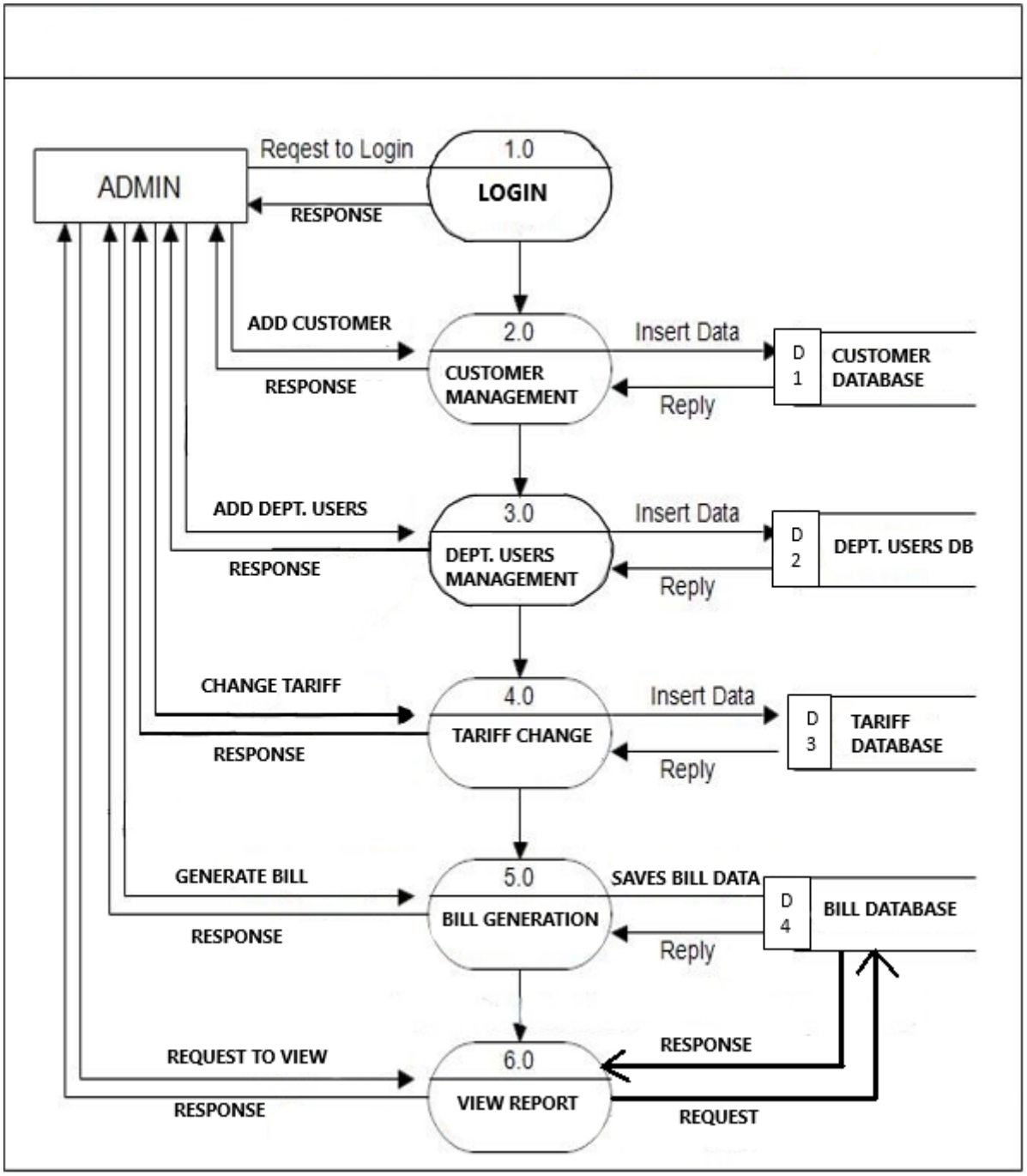


Figure 3.4. Level 1 Admin DFD

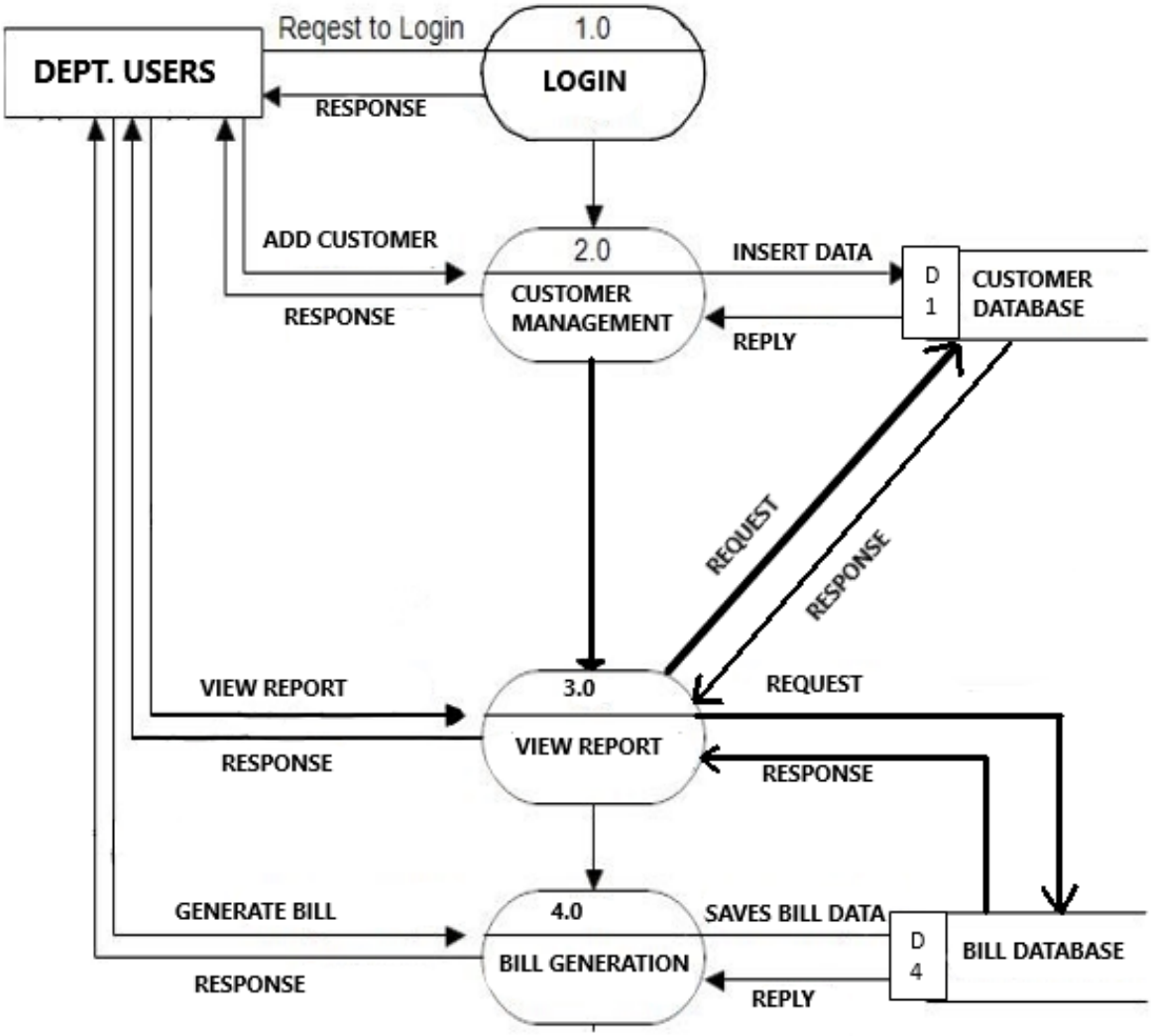


Figure 3.5 Level One DFD Dept. User

3.9.3. Level 2 DFD

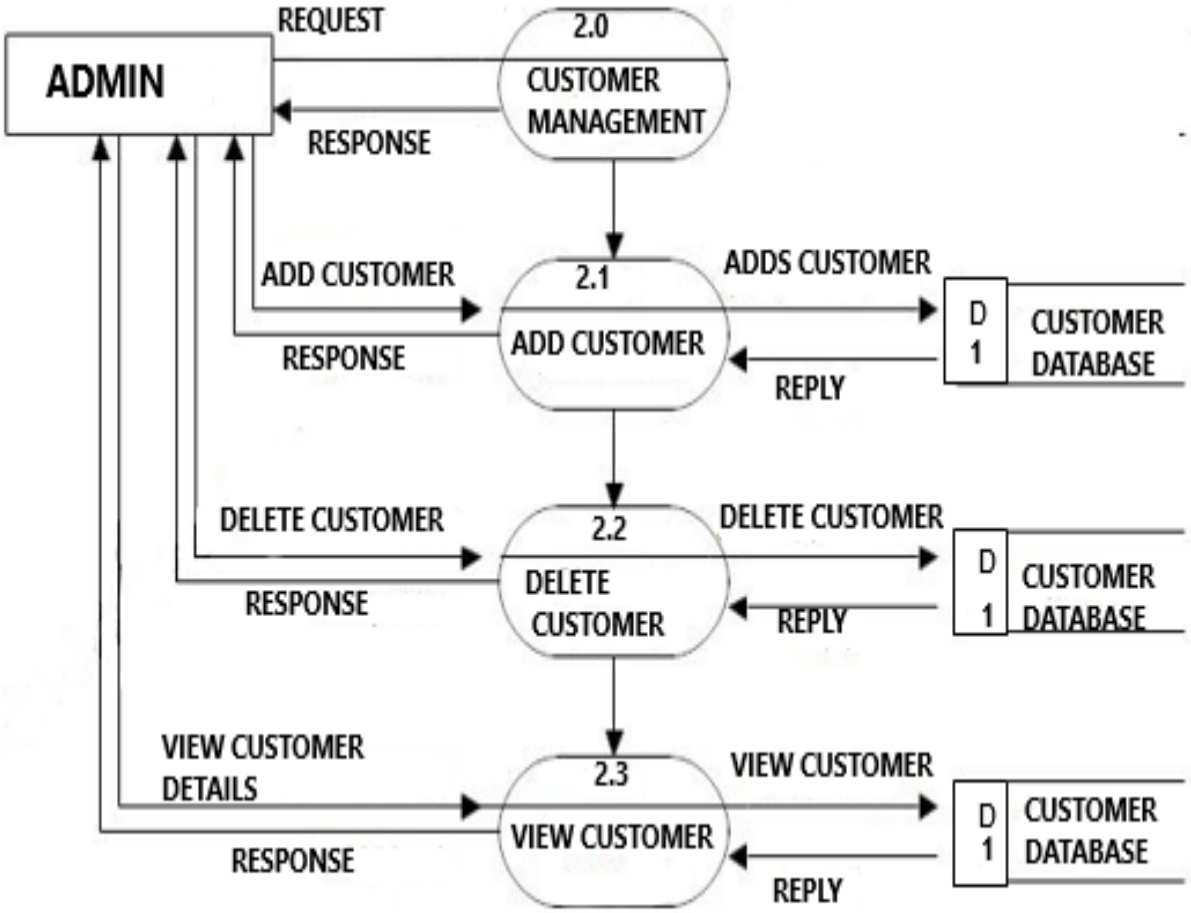


Figure 3.6. Level 2 ADMIN DFD (2.0)

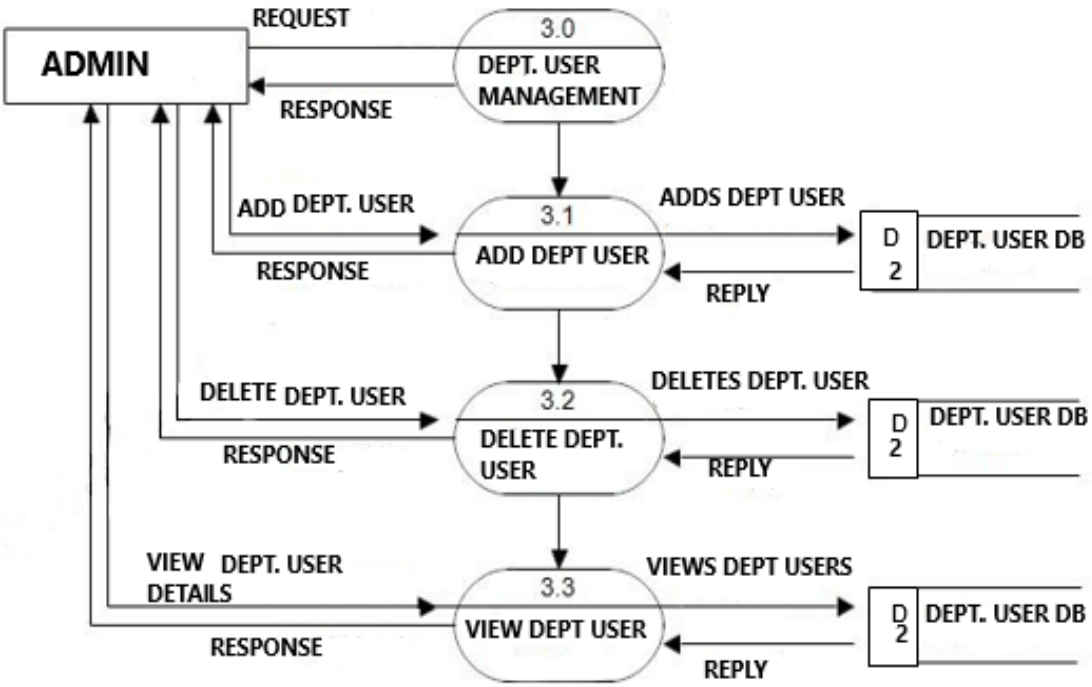


Figure 3.7. Level 2 ADMIN DFD (3.0)

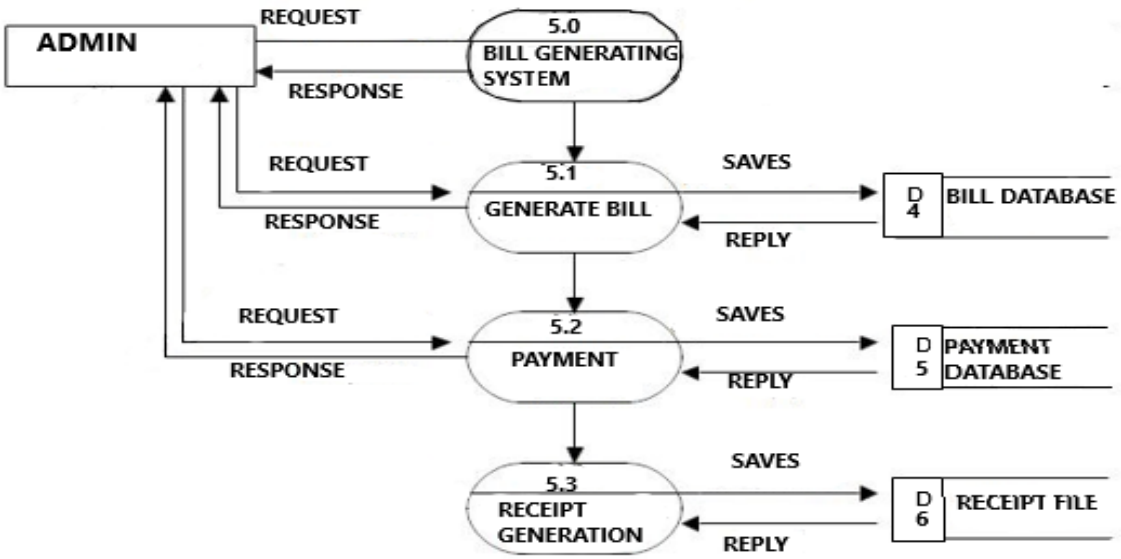


Figure 3.8. Level 2 Admin DFD (5.0)

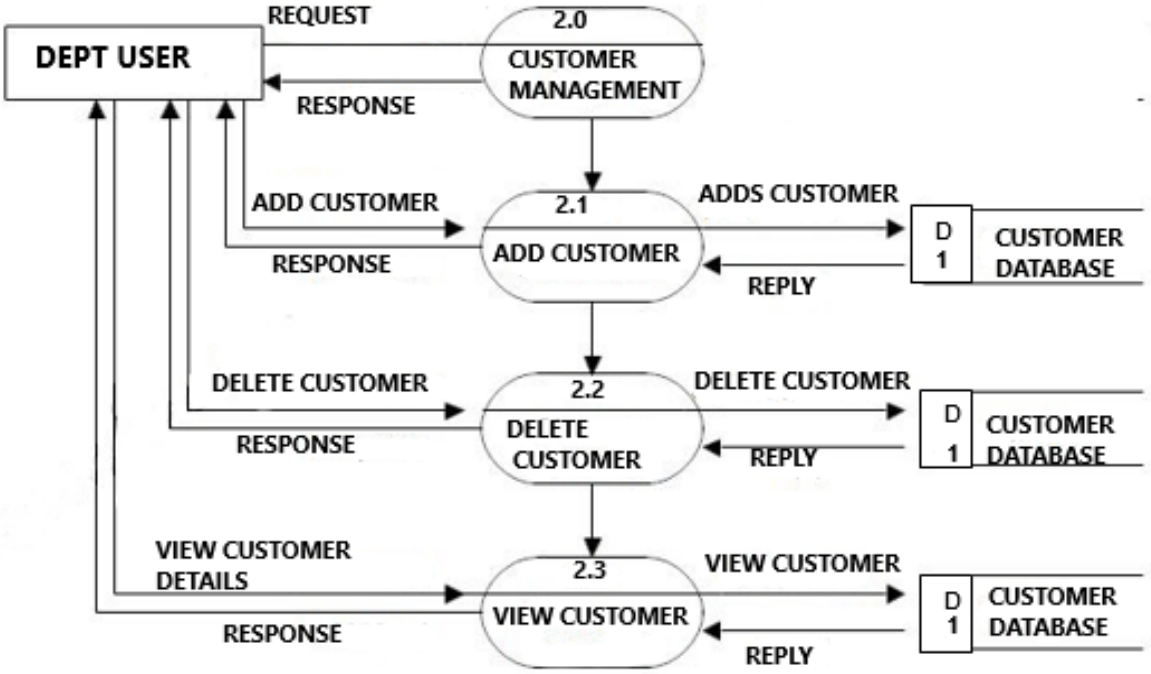


Figure 3.9. Level 2 Dept. User DFD (2.0)

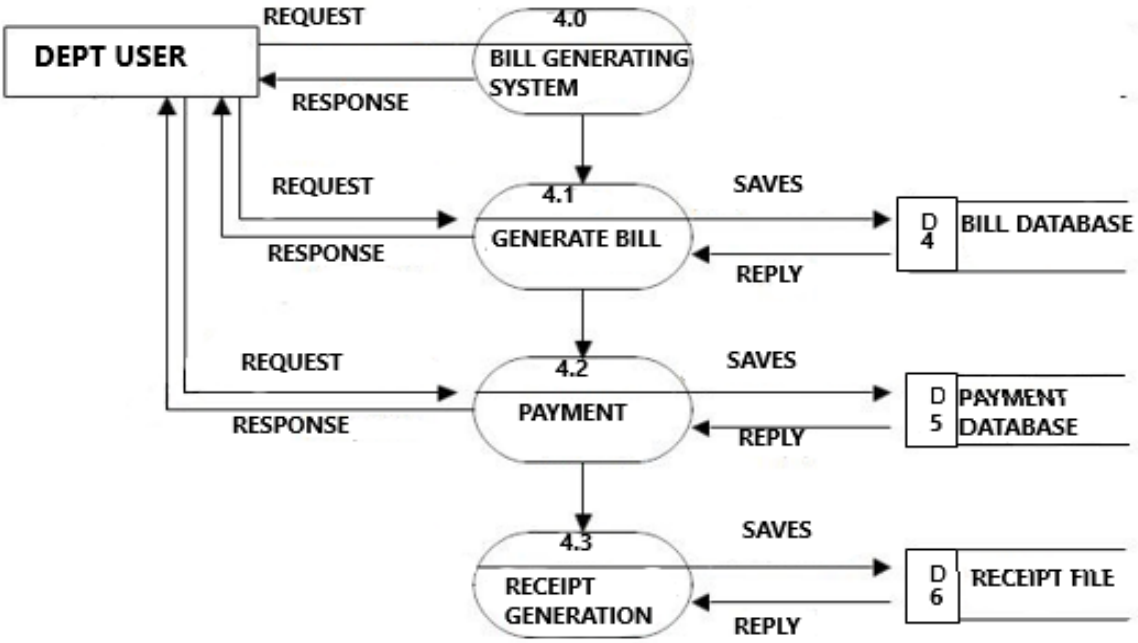


Figure 3.10. Level 2 Dept. User DFD (4.0)

CHAPTER 4: SYSTEM DESIGN

It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy the specific requirements. Before planning, you need to understand the old system thoroughly and determine how computers can best be used in order to operate efficiently.

System Design focuses on how to accomplish the objective of the system.

It mainly focuses on:

- Systems
- Processes
- Technology

4.1. Basic Modules:

In this society helping system there are two types of user one is **Admin and department employee**.

Functionality of Admin:

Admin can manage all the system by login with his username and password.

Admin manages all the functionalities for managing tariff rates etc. Admin can also add other department users, delete department users, and can view department users. Admin can also add customers, delete customers or can view customers. Admin can also generate bill.

Functionality of Department Users of the Discoms:

In our Electricity Bill Management System admin can add department user. Department users can login into their system by using valid credentials. After logging in they can perform various activities like adding customers, viewing customer's records or generating electricity bill.

4.2. Database Normalization

4.2.1. First Normal Form

Customer	
PK	CustomerID
	Name
	MeterNo
	Address
	Zipcode
	Division
	Tariff
	PhoneNo
	Email
	Billno
	Outstanding_amt
	reading

Table 4. 1 Level 1 Normalized Table

4.2.2. Second Normal Form

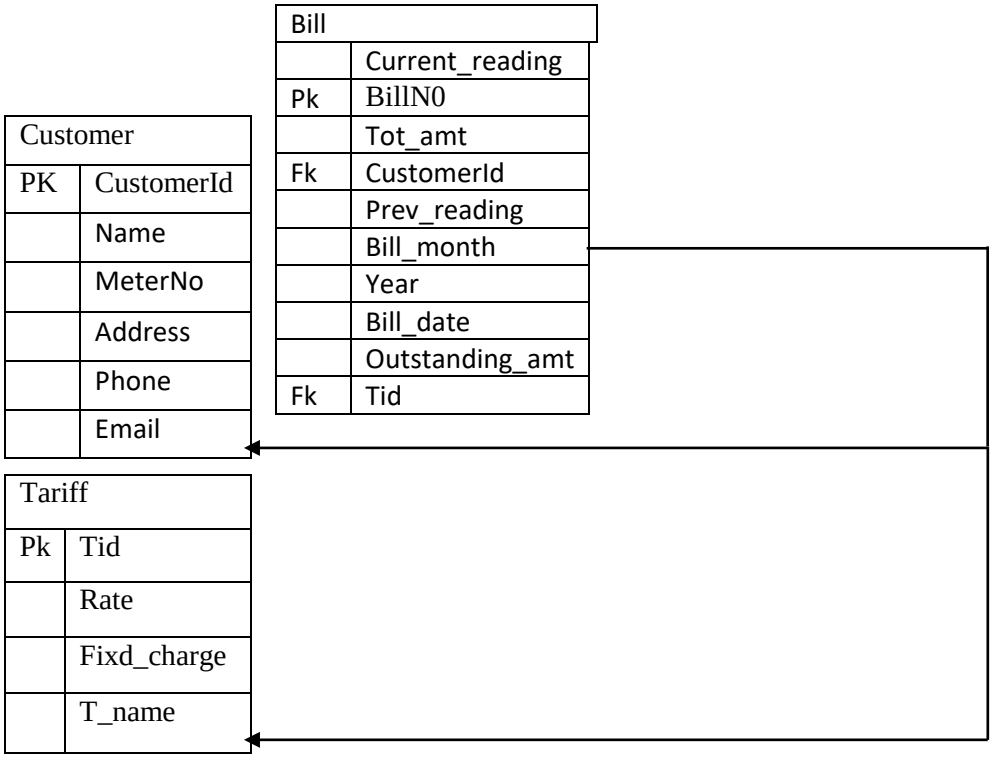


Table 4. 2 Level 2 Normalized Form

4.2.3 Third Normal form

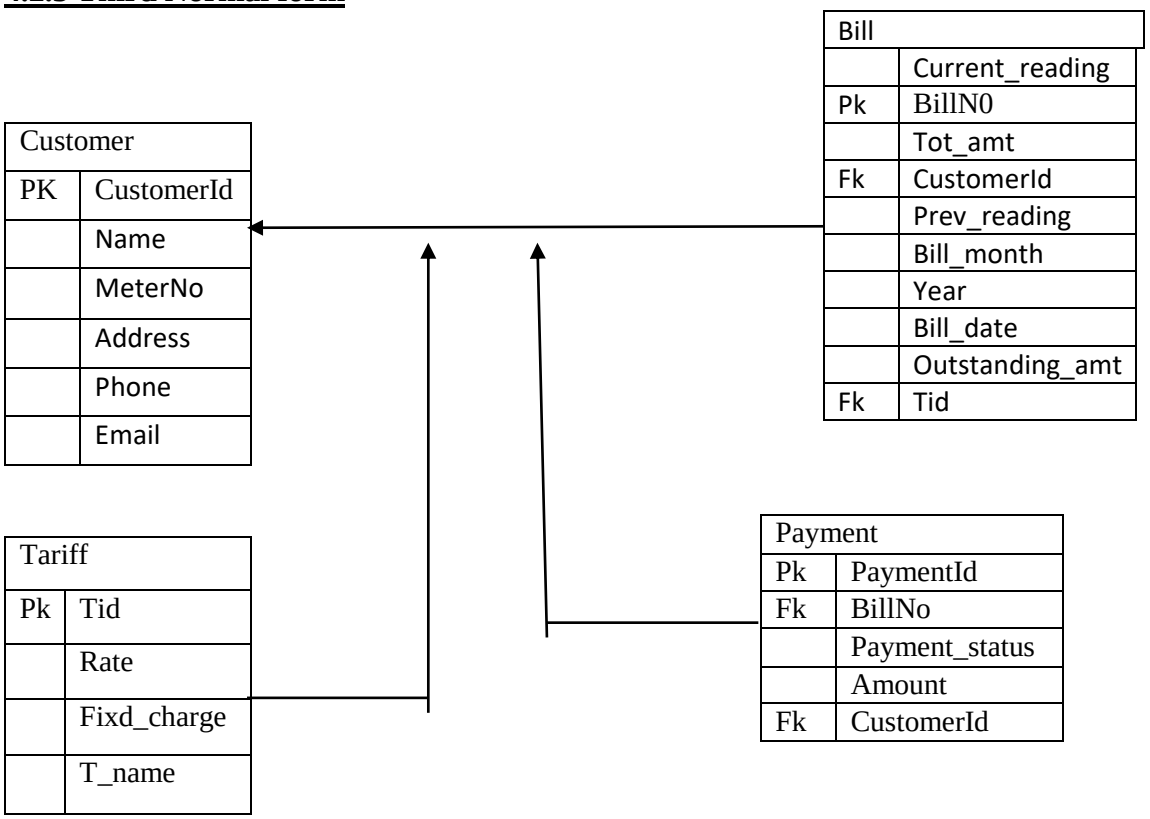


Table 4. 3 Level 3 Normalized Form

4.3. Data Dictionary

Table3.1.1.Dept. User Database

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
admin	varchar(50)	NO	UNI	NULL	
name	varchar(50)	NO		NULL	
password	varchar(20)	NO		NULL	
AdminNo	int	NO	PRI	NULL	auto_increment

Table3.1.2.Customer Database

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
Meterno	int	NO		NULL	
name	varchar(50)	NO		NULL	
zipcode	int	NO		NULL	
address	varchar(80)	NO		NULL	
Tariff	int	NO		NULL	
division	varchar(20)	NO		NULL	
Email	varchar(40)	YES		NULL	
phno	int	NO		NULL	
ConsumerNO	int	NO	PRI	NULL	auto_increment

Table3.1. 3. Bill Database

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
BillNo	int	NO	PRI	NULL	auto_increment
bill_date	date	YES		NULL	
tot_amt	int	YES		NULL	
con_no	int	YES		NULL	
current_reading	int	YES		NULL	
previous_reading	int	YES		NULL	
outstanding_amt	int	YES		NULL	

Table3.1. 4.Reading Database

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
R_id	int	NO	PRI	NULL	auto_increment
dob	date	YES		NULL	
con_no	int	YES		NULL	
reading	varchar(20)	YES		NULL	

Result Grid
Form Editor
Field Types

Table3.1.5. Tariff Database

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
T_id	int	NO	PRI	NULL	auto_increment
rate	int	YES		NULL	
fixed_charge	int	YES		NULL	
T_name	varchar(10)	YES		NULL	

Result Grid
Form Editor
Field Types

Table3.1.6 Payment Database

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
PaymentID	int	NO	PRI	NULL	auto_increment
Bill_no	int	NO		NULL	
Payment_status	varchar(10)	NO		NULL	
Amount	int	YES		NULL	
Con_no	int	NO		NULL	

4.4. ER-Diagram

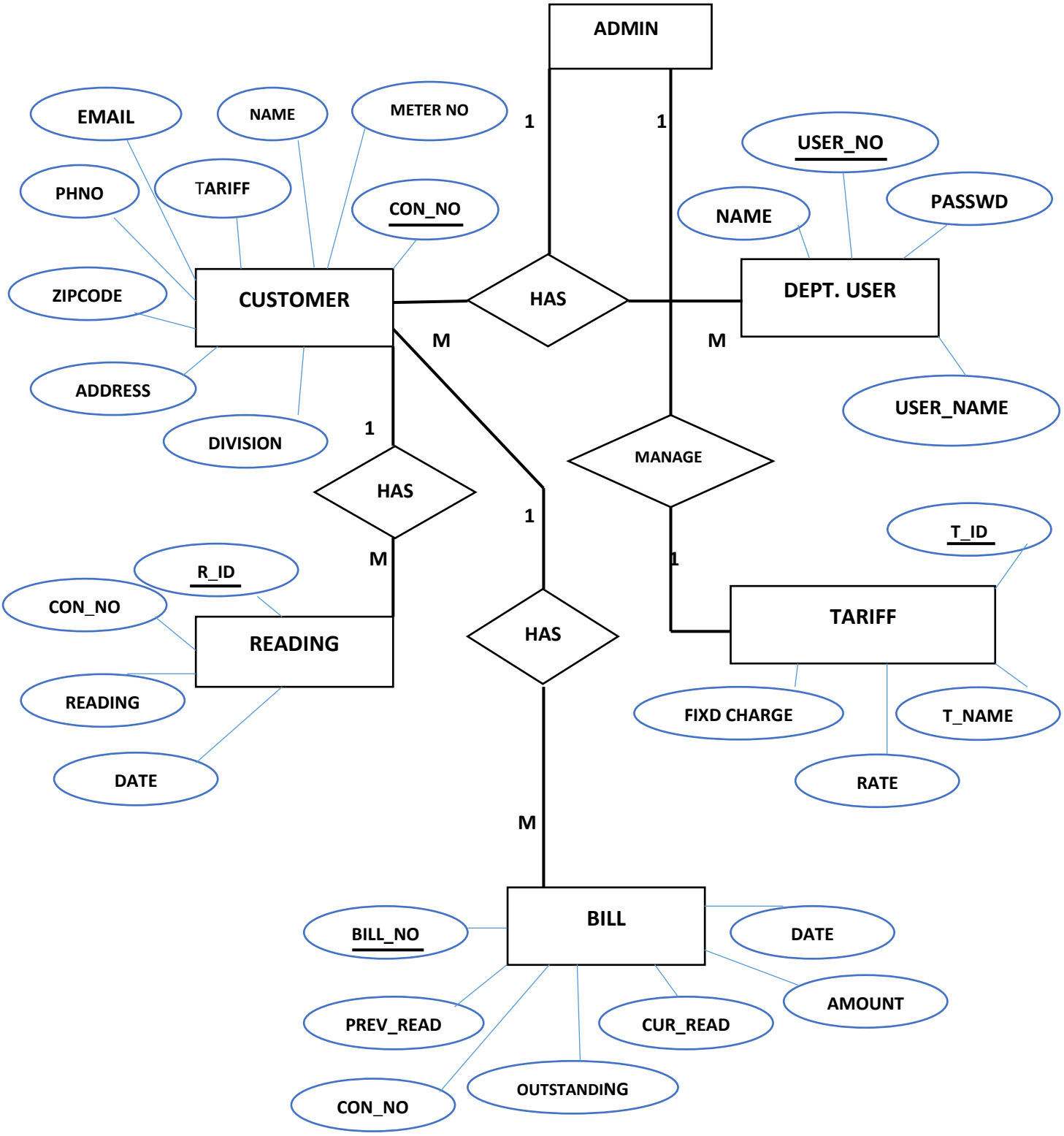


Figure 4.1. ER Diagram

CHAPTER 5: IMPLEMENTATION AND TESTING

Testing is vital for the success of any software, no system design is ever perfect. Testing is also carried in two phases, first phase is during the software engineering that is during the module creation, second phase is after the completion of software, and this is system testing which verifies that the whole set of programs hanged together.

CODING DETAILS AND EFFICIENCY

LOGIN PAGE:

```
from tkinter import ttk
from tkinter import messagebox
from homepagemaster import HomePageMaster
import mysql.connector
from homepage import *
global user_entry
global password_entry
global login
global sign_page
global admin
global name
global passwd
global user_entry2
global password_entry2
global user_name
global password

def tab_fun():
  global user_entry
  global password_entry
```

```
    global login
    global user_entry2
    global password_entry2


    login = Tk()
    login.geometry("540x360")
    login.title("LOGIN")
    login.iconphoto(True,
PhotoImage(file="C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\title1.png"))


    login_image = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\login.png")
    login_image = login_image.resize((200, 200), Image.ANTIALIAS)
    login_image = ImageTk.PhotoImage(login_image)
    my_ntbk = ttk.Notebook(login)
    my_ntbk.pack(pady=15)
    my_frame1 = Frame(my_ntbk, width=500, height=500)
    my_frame2 = Frame(my_ntbk, width=500, height=500)
    my_frame1.pack(fil="both", expand=1)
    my_frame2.pack(fil="both", expand=1)
    my_ntbk.add(my_frame1, text="Master Login")
    my_ntbk.add(my_frame2, text="Login")
    # login Image
    icon_label = Label(my_frame1, image=login_image)
    icon_label.place(x=40, y=100)
    # login Image
    icon_label = Label(my_frame2, image=login_image)
    icon_label.place(x=40, y=100)
    # Username
    username = Label(my_frame1, text="Username", font=("Helvetica", 12, "bold"), padx=10,
pady=5)
    username.place(x=250, y=100)
    user_entry = Entry(my_frame1, width=20, font=("Helvetica", 8))
    user_entry.place(x=350, y=107)
    # Password
```

```
    password = Label(my_frame1, text="Password", font=("Helvetica", 12, "bold"), padx=10, pady=5)

    password.place(x=250, y=170)

    password_entry = Entry(my_frame1, width=20, font=("Helvetica", 8))

    password_entry.place(x=350, y=177)

    login_button = Button(my_frame1, text="LOGIN", font=("Helvetica", 10, "bold"), padx=10, pady=5,

                          fg="white", bg="black", command=submit)

    login_button.place(x=300, y=230)

    signup_button = Button(my_frame1, text="ADD ADMIN", font=("Helvetica", 10, "bold"), padx=10, pady=5,

                           fg="white", bg="black", command=add_button)

    signup_button.place(x=380, y=230)

    # Login TAB

    username2 = Label(my_frame2, text="Username", font=("Helvetica", 12, "bold"), padx=10, pady=5)

    username2.place(x=250, y=100)

    user_entry2 = Entry(my_frame2, width=20, font=("Helvetica", 8))

    user_entry2.place(x=350, y=107)

    # Password

    password2 = Label(my_frame2, text="Password", font=("Helvetica", 12, "bold"), padx=10, pady=5)

    password2.place(x=250, y=170)

    password_entry2 = Entry(my_frame2, width=20, font=("Helvetica", 8))

    password_entry2.place(x=350, y=177)

    login_button2 = Button(my_frame2, text="LOGIN", font=("Helvetica", 10, "bold"), padx=10, pady=5,

                           fg="white", bg="black", command=submit_logintab)

    login_button2.place(x=350, y=230)

    login.mainloop()

def clear_page():

    global admin

    global name

    global passwd

    admin.delete(0, END)
```

```
    name.delete(0, END)
    passwd.delete(0, END)
def add_admin():
    global admin
    global name
    global passwd
    try:
        b = mysql.connector.connect(
            host="localhost",
            user="root",
            passwd="Iamankit@02",
            database="ebm"
        )
        cursor = db.cursor()
        cursor.execute("CREATE TABLE IF NOT EXISTS ADMIN(Admin VARCHAR(50),
Name VARCHAR(50), Password VARCHAR(20),"
                    "AdminNo INT AUTO_INCREMENT PRIMARY KEY)")


        sql_command = "INSERT INTO ADMIN(Admin, Name, Password)
VALUES(%s,%s,%s)"
        values = (admin.get(), name.get(),
                  passwd.get())
        cursor.execute(sql_command, values)


        db.commit()
        clear_page()
        messagebox.showinfo("Successful", "Submitted"
    except:
        messagebox.showerror("Error", "Invalid Entry")
def backtologin():
    global sign_page
    sign_page.destroy()
    tab_fun()
def add_button():
```

```
    global user_entry
    global password_entry
    global user_name
    global password
    global login
    if user_entry.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")
    elif password_entry.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")
    else:
        try:
            db = mysql.connector.connect(
                host="localhost",
                user="root",
                passwd="Iamankit@02",
                database="ebm"
            )
            mycursor = db.cursor()
            # submit_values = (user_entry.get(), password_entry.get())
            mycursor.execute(f'SELECT password from master where
adminname="{user_entry.get()}"')
            password = str(mycursor.fetchone())
            if password[2:-3] == password_entry.get():
                login.destroy()
                create_page()
            else:
                user_entry.delete(False, END)
                password_entry.delete(False, END)
                messagebox.showerror("Error", "Wrong Password Or Username")
        except:
            messagebox.showerror("ERROR", "Invalid Username")
def create_page():
    global sign_page
```

```
    global login
    global admin
    global name
    global passw
    sign_page = Tk()
    sign_page.geometry("480x540")
    sign_page.title("Create Admin")

    back = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\backcus.png")
    back = back.resize((480, 540), Image.ANTIALIAS)
    back = ImageTk.PhotoImage(back)

    mainlabel = Label(sign_page, image=back)
    mainlabel.place(x=0, y=0)

    title_label = Label(sign_page, text="ADD ADMINISTRATOR", font=("Helvetica", 30,
"bold"), fg="#2730e3", bg="black")
    title_label.grid(row=0, column=0, columnspan=2, padx=20, pady=30)
    # Fill options

    # consumer_label = Label(window, text="CONSUMER NO.", font=("Helvetica", 10,
"bold"), fg="White", bg="black")
    # consumer_label.grid(row=1, column=0, sticky="w", padx=20, pady=5)
    admin_label = Label(sign_page, text="ADMIN NAME", font=("Helvetica", 10, "bold"),
fg="White", bg="black")
    admin_label.grid(row=2, column=0, sticky="w", padx=20, pady=10)
    name_label = Label(sign_page, text="NAME", font=("Helvetica", 10, "bold"), fg="White",
bg="black")
    name_label.grid(row=3, column=0, sticky="w", padx=20, pady=10)
    passwd_label = Label(sign_page, text="PASSWORD", font=("Helvetica", 10, "bold"),
fg="White", bg="black")
    passwd_label.grid(row=4, column=0, sticky="w", padx=20, pady=10)
    # Entry boxes
    admin = Entry(sign_page, width=30)
    admin.grid(row=2, column=1, padx=20)
```

```
    name = Entry(sign_page, width=30)
    name.grid(row=3, column=1, padx=20
    passwd = Entry(sign_page, width=30)
    passwd.grid(row=4, column=1, padx=20)
    # Submit Button
    submit_button = Button(sign_page, text="SUBMIT", font=("Helvetica", 10, "bold"),
fg="black", bg="green",
                           command=add_admin)
    submit_button.grid(row=10, column=1, padx=20, pady=20, columnspan=1)
    # Clear Fields
    clear_button = Button(sign_page, text="CLEAR", font=("Helvetica", 10, "bold"), fg="black",
bg="grey",
                          command=clear_page)
    clear_button.grid(row=10, column=0, padx=20, columnspan=2)
    # Back Button
    back_button = Button(sign_page, text="<<BACK<<", font=("Helvetica", 10, "bold"),
fg="black", bg="grey",
                         command=backtologin)
    back_button.grid(row=10, column=1, sticky="e")
    sign_page.mainloop()
def submit():
    loal user_entry
    global password_entry
    global user_name
    global password
    global login
    if user_entry.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")
    elif password_entry.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")
    else
        try:
            db = mysql.connector.connect(
```

```
            host="localhost",
            user="root",
            passwd="Iamankit@02",
            database="ebm"
        )

        cursor = db.cursor()

        name_admin = user_entry.get()
        cursor.execute(f'SELECT password from master where adminname="{name_admin}"')
        password = str(cursor.fetchone())
        if password[2:-3] == password_entry.get():
            login.destroy()
            d = HomePageMaster(name_admin)
            d.mainhomepage()

        else:

            user_entry.delete(False, END)
            password_entry.delete(False, END)
            messagebox.showerror("Error", "Wrong Password Or Username")
    except EXCEPTION as e:
        messagebox.showerror("ERROR", "Invalid Username")
def submit_logintab():

    global user_entry2
    global password_entry2
    global login

    if user_entry2.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")

    elif password_entry2.get() == "":
```

```
        messagebox.showerror("ERROR", "Invalid Entry")
    else:
        try:
            db = mysql.connector.connect(
                host="localhost",
                user="root",
                passwd="Iamankit@02",
                database="ebm"
            )
            cursor = db.cursor()
            admin_name = user_entry2.get()
            cursor.execute(f'SELECT password from admin where admin="{admin_name}"')
            password = str(cursor.fetchone())

            if password[2:-3] == password_entry2.get():
                login.destroy()
                x = AdminPageclass(admin_name)
                x.AdminPage()

            else:
                user_entry2.delete(False, END)
                password_entry2.delete(False, END)
                messagebox.showerror("Error", "Wrong Password Or Username")
        except EXCEPTION as e:
            messagebox.showerror("ERROR", "Invalid Username")


tab_fun()
```

ADMIN HOMEPAGE

```
from tkinter import *
```

```
from PIL import Image, ImageTk
from tkinter import ttk
from tkinter import messagebox
import mysql.connector
import datetime
import webbrowser
class HomePageMaster:

    global label
    global my_frame4
    global my_listbox
    def __init__(self, adminname1):
        self.root = ''
        self.my_frame1 = ''
        self.my_frame2 = ''
        self.my_frame3 = ''
        self.my_frame4 = ""
        self.my_frame5 = ""
        self.my_frame6 = ""
        self.my_frame7 = ""
        self.my_frame8 = ""
        self.my_frame9 = ""
        self.my_frame10 = ""
        self.my_ntbk = ''
        self.adminname1 = adminname1
        self.old_passwd = ""
        self.new_passwd = ""
        self.renew_passwd = ""
        self.meterno = ""
        self.name = ""
        self.address = ""
        self.zipcode = ""
        self.division = ""
```

```
        self.phno = ""
        self.tariff = ""
        self.email = ""
        self.newadmin = ""
        self.newname = ""
        self.newpasswd = ""
        self.con = ""
        self.con_no = ""
        self.cur = ""
        self.view_list_entry = ""
        self.drop = ""
        self.consumer_no = ""
        self.con_delete = ""

    def close(self, pageno1):
        self.my_ntbk.hide(pageno1)

    def clear(self, page):
        if page == 1:
            self.old_passwd.delete(0, END)
            self.new_passwd.delete(0, END)
            self.renew_passwd.delete(0, END)
        elif page == 2
            self.meterno.delete(0, END)
            self.name.delete(0, END)
            self.address.delete(0, END)
            self.zipcode.delete(0, END)
            self.division.delete(0, END)
            self.phno.delete(0, END)
            self.tariff.delete(0, END)
            self.email.delete(0, END)
        else:
            self.newadmin.delete(0, END)
```

```
        self.newname.delete(0, END)
        self.newpasswd.delete(0, END)
    def mainhomepage(self):
        global my_listbox
        global disp_label
        global my_tree
        global label
        self.root = Tk()
        self.root.title("Electricity Bill Management | Homepage")
        self.root.geometry("840x540")
        self.root.config(bg="black")



        home_image =
Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\Icons\\label_image.png")
        home_image = home_image.resize((840, 540), Image.ANTIALIAS)
        home_image = ImageTk.PhotoImage(home_image)
        backimg = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\backcus.png")
        backimg = backimg.resize((840, 540), Image.ANTIALIAS)
        backimg = ImageTk.PhotoImage(backimg
        self.my_ntbk = ttk.Notebook(self.root)
        self.my_ntbk.pack()
        self.my_frame1 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame2 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame3 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame4 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame5 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame6 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame7 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame8 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame9 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame10 = Frame(self.my_ntbk, width=840, height=540)
        self.my_frame1.pack(fil="both", expand=1)
```

```
self.my_frame2.pack(fil="both", expand=1)
self.my_frame3.pack(fil="both", expand=1)
self.my_frame4.pack(fil="both", expand=1)
self.my_frame5.pack(fil="both", expand=1)
self.my_frame6.pack(fil="both", expand=1)
self.my_frame7.pack(fil="both", expand=1)
self.my_frame8.pack(fil="both", expand=1)
self.my_frame9.pack(fil="both", expand=1)
self.my_frame10.pack(fil="both", expand=1)


self.my_ntbk.add(self.my_frame1, text="Homepage")
self.my_ntbk.add(self.my_frame2, text="Change Password")
self.my_ntbk.add(self.my_frame3, text="Add Customer")
self.my_ntbk.add(self.my_frame4, text="Admin List")
self.my_ntbk.add(self.my_frame5, text="Add Admins")
self.my_ntbk.add(self.my_frame6, text="View Records")
self.my_ntbk.add(self.my_frame7, text="Generate Bill")
self.my_ntbk.add(self.my_frame8, text="Search One Customer")
self.my_ntbk.add(self.my_frame9, text="See List")
self.my_ntbk.add(self.my_frame10, text="Delete Customer")


home_label = Label(self.my_frame1, image=home_image)
home_label.place(x=0, y=0)
back_label2 = Label(self.my_frame2, image=backimg)
back_label2.place(x=0, y=0)
back_label2 = Label(self.my_frame3, image=home_image)
back_label2.place(x=0, y=0)
back_label3 = Label(self.my_frame4, image=backimg)
back_label3.place(x=0, y=0)
back_label3 = Label(self.my_frame5, image=home_image)
back_label3.place(x=0, y=0)
```

```
        back_label3 = Label(self.my_frame6, image=backimg)
        back_label3.place(x=0, y=0)
        back_label3 = Label(self.my_frame7, image=backimg)
        back_label3.place(x=0, y=0)
        back_label3 = Label(self.my_frame8, image=backimg)
        back_label3.place(x=0, y=0)
        back_label3 = Label(self.my_frame9, image=backimg)
        back_label3.place(x=0, y=0)
        back_label3 = Label(self.my_frame10, image=backimg)
        back_label3.place(x=0, y=0)
        my_menu = Menu(self.root, bg="black")
        self.root.config(menu=my_menu)
        manage_menu = Menu(my_menu, bg="black", fg="white", borderwidth=5)
        my_menu.add_cascade(label="Admins", menu=manage_menu)
        manage_menu.add_command(label="Manage", command=self.list)
        manage_menu.add_command(label="Add Admins", command=self.add_adminpage)
        chngpasswd_menu = Menu(my_menu, bg="black", fg="white")
        my_menu.add_cascade(label="Change Password", menu=chngpasswd_menu)
        chngpasswd_menu.add_command(label="Change password",
command=self.change_passwd)
        logout_menu = Menu(my_menu, bg="black", fg="white")
        my_menu.add_cascade(label="Exit", menu=logout_menu)
        logout_menu.add_command(label="Exit")
        self.my_ntbk.hide(1)
        self.my_ntbk.hide(2)
        self.my_ntbk.hide(3)
        self.my_ntbk.hide(4)
        self.my_ntbk.hide(5)
        self.my_ntbk.hide(6)
        self.my_ntbk.hide(7)
        self.my_ntbk.hide(8)
        self.my_ntbk.hide(9)
```

```
main_label = Label(self.my_frame2, text="Change Password", font=("Helvetica", 40, "bold"),
                   fg="white", bg="black", padx=20, pady=80)
main_label.grid(row=0, column=0, columnspan=2)

oldpasswd_label = Label(self.my_frame2, text="Old Password", font=("Helvetica", 10, "bold"),
                        fg="white", bg="black")
oldpasswd_label.grid(row=1, column=0)

newpasswd_label = Label(self.my_frame2, text="New Password", font=("Helvetica", 10, "bold"),
                        fg="white", bg="black")
newpasswd_label.grid(row=2, column=0)

renewpasswd_label = Label(self.my_frame2, text="ReEnter New Password", font=("Helvetica", 10, "bold"),
                          fg="white", bg="black")
renewpasswd_label.grid(row=3, column=0)

self.old_passwd = Entry(self.my_frame2, width=30)
self.old_passwd.grid(row=1, column=1, pady=10, padx=10)

self.new_passwd = Entry(self.my_frame2, width=30)
self.new_passwd.grid(row=2, column=1, pady=10, padx=10)

self.renew_passwd = Entry(self.my_frame2, width=30)
self.renew_passwd.grid(row=3, column=1, pady=10, padx=10)

submit_passwd = Button(self.my_frame2, text="Submit", font=("Helvetica", 10, "bold"),
                       fg="black", bg="green", command=self.submit_passwd)
submit_passwd.grid(row=4, column=0, columnspan=2, pady=20)

clear_passwd = Button(self.my_frame2, text="Clear", font=("Helvetica", 10, "bold"),
                      fg="black", bg="grey", command=lambda: self.clear(1))
```

```
clear_passwd.grid(row=4, column=1, pady=20)

close_passwd = Button(self.my_frame2, text="Close", font=("Helvetica", 10, "bold"),
                      fg="black", bg="red", command=lambda: self.close(1))
close_passwd.grid(row=4, column=2, pady=20)

cus_img = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\user.png")
cus_img = cus_img.resize((25, 25), Image.ANTIALIAS)
cus_img = ImageTk.PhotoImage(cus_img)

cus_button = Button(self.my_frame1, text="CUSTOMER", font=("Algerian", 18, "bold"),
                    padx=10, pady=5, fg="#2730e3", bg="black", image=cus_img,
compound="left",
                    command=self.add_customer)
cus_button.place(x=50, y=180)
# view records Button

record_img = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\record.png")
record_img = record_img.resize((25, 25), Image.ANTIALIAS)
record_img = ImageTk.PhotoImage(record_img)

view_button = Button(self.my_frame1, text="RECORDS", font=("Algerian", 18, "bold"),
                     padx=10, pady=5, fg="#2730e3", bg="black", image=record_img,
compound="left",
                     command=self.Records)
view_button.place(x=50, y=250)

# generate bill Button
generate_icon =
Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\generate.png")
generate_icon = generate_icon.resize((35, 35), Image.ANTIALIAS)
generate_icon = ImageTk.PhotoImage(generate_icon)
```

```
        generate_button = Button(self.my_frame1, text="GENERATE BILL", font=("Algerian",
18, "bold"),
                                 padx=10, pady=5, fg="#2730e3", bg="black", image=generate_icon,
compound="left",
                                 command=self.generate_bill)
        generate_button.place(x=50, y=320)
        # Delete Button


        delete_img =
Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\delete_icon.png")
        delete_img = delete_img.resize((25, 25), Image.ANTIALIAS)
        delete_img = ImageTk.PhotoImage(delete_img)


        delete_button = Button(self.my_frame1, text="Delete", font=("Algerian", 18, "bold"),
                               padx=10, pady=5, fg="#2730e3", bg="black", image=delete_img,
compound="left",
                               command=self.delete_customer)
        delete_button.place(x=250, y=180)


        # exit Button
        exit_icon = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\exiticon.png")
        exit_icon = exit_icon.resize((25, 25), Image.ANTIALIAS)
        exit_icon = ImageTk.PhotoImage(exit_icon)


        exit_button = Button(self.my_frame1, text="Exit", font=("Algerian", 18, "bold"),
                             padx=40, pady=5, fg="#e32727", bg="black", image=exit_icon,
compound="left",
                             command=self.root.destroy)
        exit_button.place(x=50, y=420)


##############################################################################
##########################################


        # Add Customer Page
```

```
title_label = Label(self.my_frame3, text="ADD DETAILS", font=("Helvetica", 30, "bold"), fg="#2730e3",
                    bg="black")
title_label.grid(row=0, column=0, columnspan=2, padx=20, pady=30)


meter_label = Label(self.my_frame3, text="METER NO.", font=("Helvetica", 10, "bold"), fg="White", bg="black")
meter_label.grid(row=2, column=0, sticky="w", padx=20, pady=10)


name_label = Label(self.my_frame3, text="NAME", font=("Helvetica", 10, "bold"), fg="White", bg="black")
name_label.grid(row=3, column=0, sticky="w", padx=20, pady=10)


address_label = Label(self.my_frame3, text="ADDRESS", font=("Helvetica", 10, "bold"), fg="White", bg="black")
address_label.grid(row=4, column=0, sticky="w", padx=20, pady=10)


zipcode_label = Label(self.my_frame3, text="ZIPCODE", font=("Helvetica", 10, "bold"), fg="White", bg="black")
zipcode_label.grid(row=5, column=0, sticky="w", padx=20, pady=10)


division_label = Label(self.my_frame3, text="DIVISION", font=("Helvetica", 10, "bold"), fg="White", bg="black")
division_label.grid(row=6, column=0, sticky="w", padx=20, pady=10)


phno_label = Label(self.my_frame3, text="PHONE NO.", font=("Helvetica", 10, "bold"), fg="White", bg="black")
phno_label.grid(row=7, column=0, sticky="w", padx=20, pady=10)


tariff_label = Label(self.my_frame3, text="TARIFF", font=("Helvetica", 10, "bold"), fg="White", bg="black")
tariff_label.grid(row=8, column=0, sticky="w", padx=20, pady=10)


email_label = Label(self.my_frame3, text="EMAIL", font=("Helvetica", 10, "bold"), fg="White", bg="black")
email_label.grid(row=9, column=0, sticky="w", padx=20, pady=10)
```

```
# Entry boxes

self.meterno = Entry(self.my_frame3, width=30)
self.meterno.grid(row=2, column=1, padx=20)

self.name = Entry(self.my_frame3, width=30)
self.name.grid(row=3, column=1, padx=20)

self.address = Entry(self.my_frame3, width=30)
self.address.grid(row=4, column=1, padx=20)

self.zipcode = Entry(self.my_frame3, width=30)
self.zipcode.grid(row=5, column=1, padx=20)

self.division = Entry(self.my_frame3, width=30)
self.division.grid(row=6, column=1, padx=20)

self.phno = Entry(self.my_frame3, width=30)
self.phno.grid(row=7, column=1, padx=20)

self.tariff = Entry(self.my_frame3, width=30)
self.tariff.grid(row=8, column=1, padx=20)

self.email = Entry(self.my_frame3, width=30)
self.email.grid(row=9, column=1, padx=20)

# Submit Button

submit_button = Button(self.my_frame3, text="SUBMIT", font=("Helvetica", 10, "bold"), fg="black", bg="green",
                       command=self.submit_customer_details)
submit_button.grid(row=10, column=1, padx=20, pady=20, columnspan=2)
```

```
        # Clear Fields

        clear_button = Button(self.my_frame3, text="CLEAR", font=("Helvetica", 10, "bold"),
fg="black", bg="grey",
                              command=lambda: self.clear(2))
        clear_button.grid(row=10, column=0, padx=20, columnspan=2)

        # Back Button
        close_button = Button(self.my_frame3, text="CLOSE", font=("Helvetica", 10, "bold"),
fg="black", bg="red",
                              command=lambda: self.close(2))
        close_button.grid(row=10, column=2, sticky="e")

    # Add Admins:

        title_label = Label(self.my_frame5, text="ADD ADMINISTRATOR", font=("Helvetica",
30, "bold"), fg="#2730e3",
                            bg="black")
        title_label.grid(row=0, column=0, columnspan=2, padx=20, pady=30)

        # Fill options

        # consumer_label = Label(window, text="CONSUMER NO.", font=("Helvetica", 10,
"bold"), fg="White", bg="black")
        # consumer_label.grid(row=1, column=0, sticky="w", padx=20, pady=5)

        admin_label = Label(self.my_frame5, text="ADMIN NAME", font=("Helvetica", 10,
"bold"), fg="White", bg="black")
        admin_label.grid(row=2, column=0, sticky="w", padx=20, pady=10)

        name_label = Label(self.my_frame5, text="NAME", font=("Helvetica", 10, "bold"),
fg="White", bg="black")
        name_label.grid(row=3, column=0, sticky="w", padx=20, pady=10)

        passwd_label = Label(self.my_frame5, text="PASSWORD", font=("Helvetica", 10,
"bold"), fg="White", bg="black")
```

```
passwd_label.grid(row=4, column=0, sticky="w", padx=20, pady=10)

# Entry boxes
self.newadmin = Entry(self.my_frame5, width=30)
self.newadmin.grid(row=2, column=1, padx=20)

self.newname = Entry(self.my_frame5, width=30)
self.newname.grid(row=3, column=1, padx=20)

self.newpasswd = Entry(self.my_frame5, width=30)
self.newpasswd.grid(row=4, column=1, padx=20)
# Submit Button

submit_button = Button(self.my_frame5, text="SUBMIT", font=("Helvetica", 10, "bold"), fg="black", bg="green",
                       command=self.save_admins)
submit_button.grid(row=10, column=1, padx=20, pady=20, columnspan=1)
# Clear Fields

clear_button = Button(self.my_frame5, text="Clear", font=("Helvetica", 10, "bold"), fg="black", bg="grey",
                      command=lambda: self.clear(3))
clea_button.grid(row=10, column=0, padx=20, columnspan=2)
# Back Button
back_button = Button(self.my_frame5, text="Close", font=("Helvetica", 10, "bold"), fg="black", bg="red",
                     command=lambda: self.close(4))
back_button.grid(row=10, column=1, sticky="e")

# Viewing Records
view_head_label = Label(self.my_frame6, text="SEARCH RECORD", font=("Helvetica", 30, "bold"), fg="blue",
                        bg="black", padx=20, pady=20)
view_head_label.grid(row=0, column=0)
```

```
        search_0ne_button = Button(self.my_frame6, text="Search One Customer",
font=("Helvetica", 15, "bold"),
                                fg="yellow", bg="black", padx=20, pady=10,
command=self.See_one_customer)
        search_0ne_button.place(x=150, y=200)


        search_list_button = Button(self.my_frame6, text="See Customer List", font=("Helvetica",
15, "bold"),
                                fg="yellow", bg="black", padx=20, pady=10, command=self.See_list)
        search_list_button.place(x=150, y=280)


        closesearch_button = Button(self.my_frame6, text=" X Close", font=("Helvetica", 10,
"bold"),
                                fg="black", bg="red", command=lambda: self.close(5))
        closesearch_button.place(x=680, y=50)


        disp_label = Label(self.my_frame8, text="", font=("Helvetica", 10, "bold"), fg="white",
bg="black")
        disp_label.place(x=120, y=220)
        # search One Customer


        view_head_label = Label(self.my_frame8, text="SEARCH CUSTOMER",
font=("Helvetica", 30, "bold"), fg="blue",
                              bg="black", padx=20, pady=20)
        view_head_label.grid(row=0, column=0)



        view_con_label = Label(self.my_frame8, text="Consumer No:", font=("Helvetica", 10,
"bold"), fg="white",
                            bg="black", padx=20, pady=20)
        view_con_label.grid(row=1, column=0)


        self.con = Entry(self.my_frame8, width=30)
        self.con.grid(row=1, column=2, padx=20)
```

```
        search_record_button = Button(self.my_frame8, text="Search", font=("Helvetica", 10,
"bold"), fg="black",
                                      bg="green", padx=30, command=self.search_one_customer)
        search_record_button.grid(row=3, column=0, columnspan=2, padx=20, pady=20)


        close_one_button = Button(self.my_frame8, text=" X Close", font=("Helvetica", 10,
"bold"),
                                  fg="black", bg="red", command=lambda: self.close(7))
        close_one_button.place(x=680, y=30)




        # search list of customers


        view_head_label = Label(self.my_frame9, text="SEARCH CUSTOMER",
font=("Helvetica", 30, "bold"), fg="blue",
                                bg="black", padx=20, pady=20)
        view_head_label.grid(row=0, column=0)


        view_list_label = Label(self.my_frame9, text="Search", font=("Helvetica", 12, "bold"),
fg="white",
                                bg="black")
        view_list_label.grid(row=1, column=0)


        self.view_list_entry = Entry(self.my_frame9, width=30)
        self.view_list_entry.grid(row=1, column=1, padx=10)


        self.drop = ttk.Combobox(self.my_frame9, values=["Search By..", "Division", "Name",
"Zipcode", "Tariff"])
        self.drop.current(0)
        self.drop.grid(row=1, column=2)


        search_consumer_button = Button(self.my_frame9, text="Search", font=("Helvetica", 10),
bg="green", fg="black",
                                        command=self.search_list)
        search_consumer_button.grid(row=2, column=0)
```

```
# my_tree = ttk.Treeview(self.my_frame9)
#
# my_tree['columns'] = ("NAME", "Consumer No", "Meter No")
#
# my_tree.column("#0", anchor=W, width=0)
# my_tree.column("NAME", anchor=W, width=120)
# my_tree.column("Consumer No", anchor=W, width=120)
# my_tree.column("Meter No", anchor=W, width=120)
#
# my_tree.heading("NAME", text="Name", anchor=W)
# my_tree.heading("Consumer No", text="Consumer No", anchor=W)
# my_tree.heading("Meter No", text="Meter No", anchor=W)
# my_tree.grid(row=3, column=0, padx=20, pady=20)

close_list_button = Button(self.my_frame9, text=" X Close", font=("Helvetica", 10,
"bold"),
                           fg="black", bg="red", command=lambda: self.close(8))
close_list_button.place(x=680, y=30)

# Generate Bill

generate_head_label = Label(self.my_frame7, text="BILL GENERATION",
font=("Helvetica", 30, "bold"), fg="blue",
                            bg="black", padx=20, pady=20)
generate_head_label.grid(row=0, column=0)

generate_label = Label(self.my_frame7, text="Consumer No:", font=("Helvetica", 10,
"bold"), fg="white",
                       bg="black", padx=20)
generate_label.grid(row=1, column=0, pady=10)

generate_label1 = Label(self.my_frame7, text="Current Reading", font=("Helvetica", 10,
"bold"), fg="white",
                        bg="black", padx=20)
```

```
generate_label1.grid(row=2, column=0, pady=10)


self.consumer_no = Entry(self.my_frame7, width=30)
self.consumer_no.grid(row=1, column=1)


self.cur = Entry(self.my_frame7, width=30)
self.cur.grid(row=2, column=1, padx=20, pady=10)


generate_bill_button = Button(self.my_frame7, text="Generate Bill", font=("Helvetica",
10, "bold"), fg="black",
                              bg="green", padx=30, command=self.generate_bill_fun)
generate_bill_button.grid(row=4, column=0, columnspan=2, padx=20, pady=20)


close_list_button = Button(self.my_frame7, text=" X Close", font=("Helvetica", 10,
"bold"),
                           fg="black", bg="red", command=lambda: self.close(6))
close_list_button.place(x=680, y=30)


view_head_label_delete = Label(self.my_frame10, text="DELETE CUSTOMER",
font=("Helvetica", 30, "bold"),
                               fg="blue", bg="black", padx=20, pady=20)
view_head_label_delete.grid(row=0, column=0)


view_con_label = Label(self.my_frame10, text="Consumer No:", font=("Helvetica", 10,
"bold"), fg="white",
                       bg="black", padx=20, pady=20)
view_con_label.grid(row=1, column=0)


self.con_delete = Entry(self.my_frame10, width=30)
self.con_delete.grid(row=1, column=2, padx=20)


search_record_button = Button(self.my_frame10, text="Delete", font=("Helvetica", 10,
"bold"), fg="black",
                              bg="green", padx=30, command=self.delete_customer_fun)
```

```
        search_record_button.grid(row=3, column=0, columnspan=2, padx=20, pady=20)


        close_one_button = Button(self.my_frame10, text=" X Close", font=("Helvetica", 10,
"bold"),
                                  fg="black", bg="red", command=lambda: self.close(9))
        close_one_button.place(x=680, y=30)
        self.root.mainloop()


    def change_passwd(self):
        self.my_ntbk.select(1)
    def generate_bill(self):
        self.my_ntbk.select(6)
    def add_adminpage(self):
        self.my_ntbk.select(4)
    def Records(self):
        self.my_ntbk.select(5)


    def See_one_customer(self):
        self.my_ntbk.select(7)


    def See_list(self):
        self.my_ntbk.select(8)




    def submit_passwd(self):
        oldPassword = self.old_passwd.get()
        newPassword = self.new_passwd.get()
        renewPassword = self.renew_passwd.get()
        print(oldPassword)
        if oldPassword == "":
            messagebox.showerror("ERROR", "Invalid Entry")
```

```
        elif newPassword == "":
            messagebox.showerror("ERROR", "Invalid Entry")
        elif renewPassword == "":
            messagebox.showerror("ERROR", "Invalid Entry")
        else:
            if newPassword != renewPassword:
                self.old_passwd.delete(False, END)
                self.new_passwd.delete(False, END)
                self.renew_passwd.delete(False, END)
                messagebox.showwarning("Warning", "Passwords Mismatch")
            else:
                try:
                    db = mysql.connector.connect(
                        host="localhost",
                        user="root",
                        passwd="Iamankit@02",
                        database="ebm"
                    )

                    cursor = db.cursor()

                    cursor.execute(f'SELECT password from admin where
admin="{self.adminname1}"')
                    password = str(cursor.fetchone())

                    if password[2:-3] == oldPassword:

                        cursor.execute(f'UPDATE admin SET password="{newPassword}" WHERE
admin="{self.adminname1}"')
                        db.commit()
                        messagebox.showinfo("SUBMITTED", "Successful")
                        self.my_ntbk.hide(1)
                    else:
                        self.old_passwd.delete(False, END)
```

```
                self.new_passwd.delete(False, END)
                self.renew_passwd.delete(False, END)
                messagebox.showerror("Error", "Wrong Password")

            except EXCEPTION as e:
                messagebox.showerror("ERROR", e)

    def add_customer(self):
        self.my_ntbk.select(2)

    def submit_customer_details(self):
        response = messagebox.askyesno("SUBMIT", "Are you sure")
        if response == 1:
            try:
                db = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )
                cursor = db.cursor()

                cursor.execute("CREATE TABLE IF NOT EXISTS customer(Meterno INT(10),
Name VARCHAR(50), Zipcode INT(8), "
                               "Address VARCHAR(80),Tariff INT(10), Division VARCHAR(20), Email
VARCHAR(40),"
                               " Phno INT(10),"
                               "ConsumerNO INT AUTO_INCREMENT PRIMARY KEY)")

                sql_command = "INSERT INTO customer(Meterno, name, zipcode, address, Tariff,
division, Email, phno) " \
                              "VALUES(%s, %s, %s, %s, %s, %s, %s, %s)"
                values = (int(self.meterno.get()), self.name.get(), int(self.zipcode.get()),
self.address.get(), int(self.tariff.get()),
                          self.division.get(), self.email.get(),
```

```
                int(self.phno.get()))

            cursor.execute(sql_command, values)
            db.commit()
            self.clear(2)
            messagebox.showinfo("Submit", "Saved")

        except:
            messagebox.showerror("Error", "Invalid Entry")
    else:
        pass

def save_admins(self):
    if self.newadmin.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")
    elif self.newname.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")

    elif self.newpasswd.get() == "":
        messagebox.showerror("ERROR", "Invalid Entry")

    else:

        response = messagebox.askyesno("Confirmation", "Are You Sure")

        if response == 1:
            try:
                db = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )
```

```
            cursor = db.cursor()
            cursor.execute(
                "CREATE TABLE IF NOT EXISTS ADMIN(Admin VARCHAR(50), Name
VARCHAR(50), Password VARCHAR(20),"
                "AdminNo INT AUTO_INCREMENT PRIMARY KEY)")
            sql_command = "INSERT INTO ADMIN(Admin, Name, Password)
VALUES(%s,%s,%s)"
            values = (self.newadmin.get(), self.newname.get(),
                      self.newpasswd.get())
            cursor.execute(sql_command, values)

            db.commit()
            self.clear(3)
            messagebox.showinfo("Successful", "Submitted")

        except:
            messagebox.showerror("Error", "Invalid Entry")
    else:
        pass

def see_detailsadmin(self):

    global my_listbox
    global label

    label.destroy()

    try:

        db = mysql.connector.connect(
            host="localhost",
            user="root",
            passwd="Iamankit@02",
            database="ebm"
```

```
        )


        cursor = db.cursor()
        n = my_listbox.get(ANCHOR)
        l = ["Name", "UserName", "Admin No"]

        cursor.execute(f'SELECT * from ADMIN WHERE name="{n}"')
        adminsearch = cursor.fetchall()

        text_variable = str()

        for i in adminsearch:

            text_variable = f' ADMIN NO : {i[3]}  \n\n NAME : {i[1]} \n\n USERID : {i[0]}'



    except :

        messagebox.showerror("ERROR", "!!ERROR!!")

def list(self):
    self.my_ntbk.select(3)
    self.list_admins()



def exit(self):
    global home
    var = messagebox.askyesno("EXIT", "Are You Sure")
    if var == 1:
        home.destroy()
    else:
        pass
```

```
def search_one_customer(self):
    global disp_label
    disp_label.place_forget()
    consumer_no = self.con.get()
    if consumer_no == "":
        messagebox.showerror("ERROR", "Invalid entry")
    elif consumer_no == " ":
        messagebox.showerror("ERROR", "Invalid entry")
    else:
        consumer_no = int(consumer_no)
        try:

            db = mysql.connector.connect(
                host="localhost",
                user="root",
                passwd="Iamankit@02",
                database="ebm"
            )
            cursor = db.cursor()
            cursor.execute(f'SELECT * from customer where consumerno={consumer_no}')
            search_result = cursor.fetchall()

            if search_result == []:
                messagebox.showwarning("Warning", "No Data Found")
            else:
                text = str()
                for result1 in search_result:
```

```
                text = f' NAME :                {result1[1]} \n\n CONSUMER NO :    {result1[8]}
\n\n METER NO :          {result1[0]} \n\n ' \
                    f'ADDRESS :           {result1[3]} \n\n ZIPCODE :              {result1[2]}
\n\n DIVISION :              {result1[5]} \n\n ' \
                    f'TARIFF :                {result1[4]} \n\n PHONE NO. :          {result1[7]}
\n\n EMAIL :                {result1[6]}'


            disp_label = Label(self.my_frame8, text=text, font=("Helvetica", 10, "bold"),
fg="white", bg="black",
                        justify=LEFT)
            disp_label.place(x=170, y=200)


        except:
            messagebox.showerror("ERROR", "!!ERROR!!")



    def search_list(self):
        global my_tree
        my_tree = ttk.Treeview(self.my_frame9)
        my_tree['columns'] = ("NAME", "Consumer No", "Meter No")
        my_tree.column("#0", anchor=W, width=0)
        my_tree.column("NAME", anchor=W, width=120)
        my_tree.column("Consumer No", anchor=W, width=120)
        my_tree.column("Meter No", anchor=W, width=120)
        my_tree.heading("NAME", text="Name", anchor=W)
        my_tree.heading("Consumer No", text="Consumer No", anchor=W)
        my_tree.heading("Meter No", text="Meter No", anchor=W)
        my_tree.grid(row=3, column=0, padx=20, pady=20)
        selected = self.drop.get()
        value = self.view_list_entry.get()
        if value == "":
            messagebox.showerror("ERROR", "Invalid Entry")
        elif value == " ":
            messagebox.showerror("ERROR", "Invalid Entry")
        else:
```

```
        if selected == "Search by...":
            messagebox.showwarning("WARNING", "Invalid Selection")
        elif selected == "Division":
            if value == "":
                messagebox.showerror("ERROR", "Invalid entry")
            elif value == " ":
                messagebox.showerror("ERROR", "Invalid entry")
            else:
                try:
                    db = mysql.connector.connect(
                        host="localhost",
                        user="root",
                        passwd="Iamankit@02",
                        database="ebm"
                    )
                    cursor = db.cursor()
                    cursor.execute(f'SELECT * from customer where division="{value}"')
                    search_result = cursor.fetchall()
                    if search_result == []:
                        messagebox.showwarning("Warning", "No Data Found")
                    else:
                        i = 0
                        for result1 in search_result:
                            my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                            i += 1
                except:
                    messagebox.showerror("ERROR", "!!ERROR!!")
        elif selected == "Name":
            if value == "":
                messagebox.showerror("ERROR", "Invalid entry")
            elif value == " ":
                messagebox.showerror("ERROR", "Invalid entry")
```

```
        else:
            try:
                db = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )

                cursor = db.cursor()
                cursor.execute(f'SELECT * from customer where Name="{value}"')
                search_result = cursor.fetchall()

                if search_result == []:
                    messagebox.showwarning("Warning", "No Data Found")
                else:
                    i = 0
                    for result1 in search_result:
                        my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                        i += 1
            except:
                messagebox.showerror("ERROR", "!!ERROR!!"
    elif selected == "Zipcode":
        if value == "":
            messagebox.showerror("ERROR", "Invalid entry"
        elif value == " ":
            messagebox.showerror("ERROR", "Invalid entry")
        else:
            value = int(value)
            try:
                db = mysql.connector.connect(
                    host="localhost",
```

```
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )
                cursor = db.cursor()
                cursor.execute(f'SELECT * from customer where Zipcode="{value}"')
                search_result = cursor.fetchall()
                if search_result == []:
                    messagebox.showwarning("Warning", "No Data Found")
                else:
                    i = 0
                    for result1 in search_result:
                        my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                        i += 1
            except:
                messagebox.showerror("ERROR", "!!ERROR!!")
        elif selected == "Tariff":
            if value == "":
                messagebox.showerror("ERROR", "Invalid entry")
            elif value == " ":
                messagebox.showerror("ERROR", "Invalid entry")
            else:
                value = int(value)
                try:
                    db = mysql.connector.connect(
                        host="localhost",
                        user="root",
                        passwd="Iamankit@02",
                        database="ebm"
                    )
                    cursor = db.cursor()
                    cursor.execute(f'SELECT * from customer where Tariff="{value}"')
```

```
                search_result = cursor.fetchall()
                if search_result == []:
                    messagebox.showwarning("Warning", "No Data Found")
                else:
                    i = 0
                    for result1 in search_result:
                        my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                        i += 1
            except:
                messagebox.showerror("ERROR", "!!ERROR!!")
        else:
            messagebox.showwarning("WARNING", "No Data Found")


    def generate_bill_fun(self):
        global d
        global root1
        global consumer
        global total
        global current_reading
        global tariff
        global curr_date
        global curr_month
        global curr_year
        global outstanding
        global prev_reading
        global rate
        global fixd_charge
        global phone
        global curr_month
        global net
        tariff = 0
        current_reading = self.cur.get()
```

```
consumer = self.consumer_no.get()
d = datetime.datetime.now()
curr_date = d.strftime("%x")
curr_month = d.strftime("%B")
curr_year = d.strftime("%Y")
curr_time = d.strftime("%X")
outstanding = 0
prev_reading = 0
rate = 0
fixd_charge = 0
phone = 0
division = ''
add = ''
name = ''
meterno = 00
net = 0
if current_reading == "":
    messagebox.showerror("ERROR", "Invalid Entry")

elif consumer == "":
    messagebox.showerror("ERROR", "Invalid Entry")

else:
    consumer = int(consumer)
    current_reading = int(current_reading)

    try:
        db = mysql.connector.connect(
            host="localhost",
            user="root",
            passwd="Iamankit@02",
            database="ebm"
        )
```

```
root1 = Tk()
root1.title('BILL')
root1.geometry("400x680")
my_canvas.create_window((0, 0), window=second_frame, anchor="nw"

cursor = db.cursor()
try:
    cursor.execute(f'SELECT * from customer where ConsumerNO={consumer}')
    search_result = cursor.fetchall()

    for data in search_result:
        tariff = data[4]
        phone = data[7]
        division = data[5]
        add = data[3]
        name = data[1]
        meterno = data[0]

except:
    messagebox.showwarning("Warning", "No Data found")
try:
    cursor.execute(f'SELECT * from tariff where T_id="{int(tariff)}"')
    tariff_result = cursor.fetchall()

    for i in tariff_result:
        fixd_charge = int(i[2])
        rate = int(i[1])
except:
    messagebox.showwarning("Warning", "No Data found")
try:
    cursor.execute(f'SELECT dob, reading as last FROM reading where
con_no={consumer} ORDER BY dob DESC LIMIT 1')
    reading_data = cursor.fetchall()
```

```
            for read in reading_data:
                prev_date = read[0]
                prev_reading = int(read[1])
        except:
            messagebox.showwarning("Warning", "No Data found")


        try:
             cursor.execute(f'SELECT out_amount as last FROM outstanding_amt where
con_no={consumer} ORDER BY date_reading DESC LIMIT 1')
             out = cursor.fetchall()
             for outamt in out:
                 outstanding = outamt[0]
        except:
            messagebox.showwarning("Warning", "No Data found")


        diff_reading = current_reading - prev_reading
        energy_charge = diff_reading * rate
        total = energy_charge + fixd_charge
        net = total + outstanding


        bill_content = f' Electricity Department \n\n\n Bill \n\n\n\n\n' \
                    f'Division : {division} \n\n Consumer No. : {consumer} \n\n Name : {name}
\n\n ' \
                    f'Address : {add} \n\n Phone No. : {phone}' \
                    f'\n\n\n --------Comercial Detaills-------- \n\n\n ' \
                    f'Tariff : {tariff} \n\n Load : 220V \n\n' \
                    f'--------Meter Details-------- \n\n\n' \
                    f'Meter No. : {meterno} \n\n ' \
                    f'--------Billing Parameter-------- \n\n\n' \
                    f'Bill Issue Date : {curr_date} \n\n Bill Month : {curr_month} \n\n Time :
{curr_time} \n\nDate Of Previous Reading' \
                    f' : {prev_date} \n\n Previous Reading : {prev_reading} \n\n Current
Reading : {current_reading} \n\n ' \
                    f'Difference : {diff_reading} \n\n ' \
```

```
                    f'--------Current Assessment--------\n\n\n' \
                    f'Energy Charge : {energy_charge} \n\n Fixed Charge = {fixd_charge} \n\n '
\
                    f'--------Net Demand-------- \n\n\n' \
                    f'Total Amount : {total} \n\n Outstanding Amt : {outstanding} \n\n Net
Payable: {net}'



        bill_label = Label(second_frame, text=bill_content, font=("Helvetica", 12, "bold"),
justify=LEFT)
        bill_label.grid(row=0, columnspan=2, column=0)
        bill_button = Button(second_frame, text="SAVE", font=("Helvetica", 12, "bold"),
bg="green", fg="black",
                        command=self.save_bill)
        bill_button.grid(row=1, column=0, padx=40)
        pay_save_button = Button(second_frame, text="PAY & SAVE", font=("Helvetica",
12, "bold"), bg="green",
                            fg="black", command=self.pay_and_save)
        pay_save_button.grid(row=1, column=1, padx=40)INSERT INTO reading(dob,
con_no, reading) VALUES
      except:
        messagebox.showerror("ERROR", "!!ERROR!!")


  def save_bill(self):
    global consumer
    global d
    global root1
    global current_reading
    global total
    global prev_reading
    global outstanding
    global curr_month
    global curr_year
    global net
    try:
```

```
        db = mysql.connector.connect(
            host="localhost",
            user="root",
            passwd="Iamankit@02",
            database="ebm"
        )
        cursor = db.cursor()
        sql_command = "INSERT INTO reading(dob, con_no, reading) VALUES(%s,%s,%s)"
        values = (d, consumer, current_reading)
        sql_command1 = "INSERT INTO bill(bill_date, tot_amt, con_no, current_reading, previous_reading, " \
                       "outstanding_amt, bill_month, year) VALUES(%s,%s,%s,%s,%s,%s, %s, %s)"
        values1 = (d, total, consumer, current_reading, prev_reading, outstanding, curr_month, int(curr_year))


        sql_command2 = "INSERT INTO outstanding_amt(con_no, date_reading, out_amount, tot_amt, out_month)" \
                       "values(%s,%s,%s,%s,%s)"
        values2 = (consumer, d, net, total, curr_month)
        cursor.execute(sql_command, values)
        cursor.execute(sql_command1, values1)
        cursor.execute(sql_command2, values2)
        db.commit()
        messagebox.showinfo("INFO", "Saved")
        root1.destroy()
    except EXCEPTION as ee:
        messagebox.showerror("ERROR", "Unable to Save")
        print(ee)
def pay_and_save(self):
    webbrowser.open("https://rzp.io/l/uqQgiTS")
def delete_admins(self):
    global my_listbox
    global label
    label.destroy()
```

```
    delete = my_listbox.get(ANCHOR)
    if delete == "":
        messagebox.showwarning("Warning", "Item not selected")
    else:
        response = messagebox.askyesno("Confirmation", "Are You Sure")
        print(response)
        if response == 0:
            print(response, "kumar")
            pass
        else:
            try:
                db = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )
                cursor = db.cursor()
                print(response, "ankit2")
                print(response, "ankit1")
                print(delete)
                print(type(delete))
                cursor.execute(f'Delete From admin where name="{delete}"')
                db.commit()
                messagebox.showinfo("Item Deleted", "Deleted")
            except:
                messagebox.showerror("ERROR", "!!ERROR!!")


def list_admins(self):
    global label
    global my_listbox
    label.destroy()
    try:
```

```
            db = mysql.connector.connect(
                host="localhost",
                user="root",
                passwd="Iamankit@02",
                database="ebm"
            )
            my_listbox = Listbox(self.my_frame4, bg="black", fg="white", width=50, height=200)
            my_listbox.grid(row=0, column=0)
            cursor = db.cursor()
            cursor.execute("SELECT Name Adminno from admin")
            admins = cursor.fetchall()
            for admin in admins:
                my_listbox.insert(END, admin[0]
        except :
            messagebox.showerror("ERROR", "!!ERROR!!")


        details_button = Button(self.my_frame4, text="See Details", font=("Helvetica", 10,
"bold"),
                                fg="black", bg="grey", command=self.see_detailsadmin)
        details_button.place(x=350, y=50)


        deleteadmin_button = Button(self.my_frame4, text="Delete", font=("Helvetica", 10,
"bold"),
                                    fg="black", bg="grey", command=self.delete_admins)
        deleteadmin_button.place(x=490, y=50)


        closeadmin_button = Button(self.my_frame4, text="Close", font=("Helvetica", 10, "bold"),
                                   fg="black", bg="red", command=lambda: self.close(3))
        closeadmin_button.place(x=560, y=50)
    def delete_customer(self):
        self.my_ntbk.select(9)
    def delete_customer_fun(self):
        consumer_no = self.con_delete.get()
```

```
if consumer_no == "":
    messagebox.showerror("ERROR", "Invalid entry")

elif consumer_no == " ":
    messagebox.showerror("ERROR", "Invalid entry")

else:
    consumer_no = int(consumer_no)

    try:

        db = mysql.connector.connect(
            host="localhost",
            user="root",
            passwd="Iamankit@02",
            database="ebm"
        )
        cursor = db.cursor()
        cursor.execute(f'Select * from customer where consumerno={consumer_no}')
        result = cursor.fetchall(
        for data in result:
            cons = data[8]
        cons = int(cons)
        response = messagebox.askyesno("Warning", "Are You Sure")
        if response == 0:
            pass
        else:
            cursor.execute(f'DELETE from customer where consumerno={cons}')
            db.commit()
            messagebox.showinfo("INFO", "Deleted!!")
    except:
        messagebox.showwarning("Warning", "No Data found")
```

DEPT. USER HOMEPAGE

```
from tkinter import *
from PIL import Image, ImageTk
from tkinter import messagebox
import mysql.connector
from tkinter import ttk
import webbrowser
import datetime
class AdminPageclass:
    def __init__(self, adminname1):
        self.root = ''
        self.my_frame1 = ''
        self.my_frame2 = ''
        self.my_frame3 = ''
        self.my_frame4 = ''
        self.my_frame5 = ''
        self.my_frame6 = ''
        self.my_frame7 = ''
        self.my_ntbk = ''
        self.adminname1 = adminname1
        self.old_passwd = ""
        self.new_passwd = ""
        self.renew_passwd = ""
        self.meterno = ""
        self.name = ""
        self.address = ""
        self.zipcode = ""
        self.division = ""
        self.phno = ""
        self.tariff = ""
        self.email = ""
    def close(self, pageno1):
        self.my_ntbk.hide(pageno1
```

```
def clear(self, page):
    if page == 1:
        self.old_passwd.delete(0, END)
        self.new_passwd.delete(0, END)
        self.renew_passwd.delete(0, END)
    else:
        self.meterno.delete(0, END)
        self.name.delete(0, END)
        self.address.delete(0, END)
        self.zipcode.delete(0, END)
        self.division.delete(0, END)
        self.phno.delete(0, END)
        self.tariff.delete(0, END)
        self.email.delete(0, END


def AdminPage(self):
    global disp_label
    self.root = Tk()
    self.root.title("Electricity Bill Management | Homepage")
    self.root.geometry("840x540")
    home_image = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\Icons\\label_image.png")
    home_image = home_image.resize((840, 540), Image.ANTIALIAS)
    home_image = ImageTk.PhotoImage(home_image)
    backimg = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\backcus.png")
    backimg = backimg.resize((840, 540), Image.ANTIALIAS)
    backimg = ImageTk.PhotoImage(backimg)
    self.my_ntbk = ttk.Notebook(self.root)
    self.my_ntbk.pack()
    self.my_frame1 = Frame(self.my_ntbk, width=840, height=540)
    self.my_frame2 = Frame(self.my_ntbk, width=840, height=540)
    self.my_frame3 = Frame(self.my_ntbk, width=840, height=540)
    self.my_frame4 = Frame(self.my_ntbk, width=840, height=540)
    self.my_frame5 = Frame(self.my_ntbk, width=840, height=540)
    self.my_frame6 = Frame(self.my_ntbk, width=840, height=540)
```

```
self.my_frame7 = Frame(self.my_ntbk, width=840, height=540
self.my_frame1.pack(fil="both", expand=1)
self.my_frame2.pack(fil="both", expand=1)
self.my_frame3.pack(fil="both", expand=1)
self.my_frame4.pack(fil="both", expand=1)
self.my_frame5.pack(fil="both", expand=1)
self.my_frame6.pack(fil="both", expand=1)
self.my_frame7.pack(fil="both", expand=1)
self.my_ntbk.add(self.my_frame1, text="Homepage")
self.my_ntbk.add(self.my_frame2, text="Change Password")
self.my_ntbk.add(self.my_frame3, text="Add Customer")
self.my_ntbk.add(self.my_frame4, text="Search Records")
self.my_ntbk.add(self.my_frame5, text="Search Customer")
self.my_ntbk.add(self.my_frame6, text="See List")
self.my_ntbk.add(self.my_frame7, text="Generate Bill")
home_label = Label(self.my_frame1, image=home_image)
home_label.place(x=0, y=0)
back_label2 = Label(self.my_frame2, image=backimg)
back_label2.place(x=0, y=0)

back_label2 = Label(self.my_frame3, image=backimg)
back_label2.place(x=0, y=0)

home_label = Label(self.my_frame4, image=home_image)
home_label.place(x=0, y=0)

back_label2 = Label(self.my_frame5, image=backimg)
back_label2.place(x=0, y=0)

back_label2 = Label(self.my_frame6, image=backimg)
back_label2.place(x=0, y=0)

back_label2 = Label(self.my_frame7, image=home_image)
back_label2.place(x=0, y=0)
```

```
my_menu = Menu(self.root)
self.root.config(menu=my_menu)

Manage_menu = Menu(my_menu)
my_menu.add_cascade(label="Change Password", menu=Manage_menu)
Manage_menu.add_command(label="Change password", command=self.change_passwd)
self.my_ntbk.hide(1)
self.my_ntbk.hide(2)
self.my_ntbk.hide(3)
self.my_ntbk.hide(4)
self.my_ntbk.hide(5)
self.my_ntbk.hide(6)

main_label = Label(self.my_frame2, text="Change Password", font=("Helvetica", 40, "bold"),
            fg="white", bg="black", padx=20, pady=80)
main_label.grid(row=0, column=0, columnspan=2)

oldpasswd_label = Label(self.my_frame2, text="Old Password", font=("Helvetica", 10, "bold"),
                fg="white", bg="black")
oldpasswd_label.grid(row=1, column=0)

newpasswd_label = Label(self.my_frame2, text="New Password", font=("Helvetica", 10, "bold"),
                fg="white", bg="black")
newpasswd_label.grid(row=2, column=0)

renewpasswd_label = Label(self.my_frame2, text="ReEnter New Password", font=("Helvetica", 10,
"bold"),
                  fg="white", bg="black")
renewpasswd_label.grid(row=3, column=0)

self.old_passwd = Entry(self.my_frame2, width=30)
self.old_passwd.grid(row=1, column=1, pady=10, padx=10)
```

```
    self.new_passwd = Entry(self.my_frame2, width=30)
    self.new_passwd.grid(row=2, column=1, pady=10, padx=10)

    self.renew_passwd = Entry(self.my_frame2, width=30)
    self.renew_passwd.grid(row=3, column=1, pady=10, padx=10)

    submit_passwd = Button(self.my_frame2, text="Submit", font=("Helvetica", 10, "bold"),
                    fg="black", bg="green", command=self.submit_passwd)
    submit_passwd.grid(row=4, column=0, columnspan=2, pady=20)

    clear_passwd = Button(self.my_frame2, text="Clear", font=("Helvetica", 10, "bold"),
                    fg="black", bg="grey", command=lambda: self.clear(1))
    clear_passwd.grid(row=4, column=1, pady=20)

    close_passwd = Button(self.my_frame2, text="Close", font=("Helvetica", 10, "bold"),
                    fg="black", bg="red", command=lambda: self.close(1))
    close_passwd.grid(row=4, column=2, pady=20)
# Main Page Options:
    cus_img = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\user.png")
    cus_img = cus_img.resize((25, 25), Image.ANTIALIAS)
    cus_img = ImageTk.PhotoImage(cus_img)
    cus_button = Button(self.my_frame1, text="CUSTOMER", font=("Algerian", 18, "bold"),
                padx=10, pady=5, fg="#2730e3", bg="black", image=cus_img, compound="left",
                command=self.add_customer)
    cus_button.place(x=50, y=180)
    # view records Button
    record_img = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\record.png")
    record_img = record_img.resize((25, 25), Image.ANTIALIAS)
    record_img = ImageTk.PhotoImage(record_img)
    view_button = Button(self.my_frame1, text="RECORDS", font=("Algerian", 18, "bold"),
                padx=10, pady=5, fg="#2730e3", bg="black", image=record_img, compound="left",
                command=self.Records)
    view_button.place(x=50, y=250)
```

```
        # generate bill Button
        generate_icon = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\generate.png")
        generate_icon = generate_icon.resize((35, 35), Image.ANTIALIAS)
        generate_icon = ImageTk.PhotoImage(generate_icon)
        generate_button = Button(self.my_frame1, text="GENERATE BILL", font=("Algerian", 18, "bold"),
                                 padx=10, pady=5, fg="#2730e3", bg="black", image=generate_icon,
compound="left"
                                 command=self.generate_bill)
        generate_button.place(x=50, y=320)
        # exit Button
        exit_icon = Image.open("C:\\Users\\dell\\PycharmProjects\\EBM\\icons\\exiticon.png")
        exit_icon = exit_icon.resize((25, 25), Image.ANTIALIAS)
        exit_icon = ImageTk.PhotoImage(exit_icon)
        exit_button = Button(self.my_frame1, text="Exit", font=("Algerian", 18, "bold"),
                             padx=40, pady=5, fg="#e32727", bg="black", image=exit_icon, compound="left",
                             command=self.root.destroy)
        exit_button.place(x=50, y=420)
    # Add Customer Page
        title_label = Label(self.my_frame3, text="ADD DETAILS", font=("Helvetica", 30, "bold"),
fg="#2730e3",
                            bg="black")
        title_label.grid(row=0, column=0, columnspan=2, padx=20, pady=30)
        # Fill options
        # consumer_label = Label(window, text="CONSUMER NO.", font=("Helvetica", 10, "bold"),
fg="White", bg="black")
        # consumer_label.grid(row=1, column=0, sticky="w", padx=20, pady=5)
        meter_label = Label(self.my_frame3, text="METER NO.", font=("Helvetica", 10, "bold"),
fg="White", bg="black")
        meter_label.grid(row=2, column=0, sticky="w", padx=20, pady=10)
        name_label = Label(self.my_frame3, text="NAME", font=("Helvetica", 10, "bold"), fg="White",
bg="black")
        name_label.grid(row=3, column=0, sticky="w", padx=20, pady=10)
        address_label = Label(self.my_frame3, text="ADDRESS", font=("Helvetica", 10, "bold"), fg="White",
bg="black")
        address_label.grid(row=4, column=0, sticky="w", padx=20, pady=10)
```

```
zipcode_label = Label(self.my_frame3, text="ZIPCODE", font=("Helvetica", 10, "bold"), fg="White", bg="black")

zipcode_label.grid(row=5, column=0, sticky="w", padx=20, pady=10

division_label = Label(self.my_frame3, text="DIVISION", font=("Helvetica", 10, "bold"), fg="White", bg="black")

division_label.grid(row=6, column=0, sticky="w", padx=20, pady=10)

phno_label = Label(self.my_frame3, text="PHONE NO.", font=("Helvetica", 10, "bold"), fg="White", bg="black")

phno_label.grid(row=7, column=0, sticky="w", padx=20, pady=10)

tariff_label = Label(self.my_frame3, text="TARIFF", font=("Helvetica", 10, "bold"), fg="White", bg="black")

tariff_lbel.grid(row=8, column=0, sticky="w", padx=20, pady=10

email_label = Label(self.my_frame3, text="EMAIL", font=("Helvetica", 10, "bold"), fg="White", bg="black")

email_label.grid(row=9, column=0, sticky="w", padx=20, pady=10)


# Entry boxe

self.meterno = Entry(self.my_frame3, width=30)

self.meterno.grid(row=2, column=1, padx=20)


self.name = Entry(self.my_frame3, width=30)

self.name.grid(row=3, column=1, padx=20)


self.address = Entry(self.my_frame3, width=30)

self.address.grid(row=4, column=1, padx=20)


self.zipcode = Entry(self.my_frame3, width=30)

self.zipcode.grid(row=5, column=1, padx=20)


self.division = Entry(self.my_frame3, width=30)

self.division.grid(row=6, column=1, padx=20)


self.phno = Entry(self.my_frame3, width=30)

self.phno.grid(row=7, column=1, padx=20)
```

```
    # Back Button
    close_button = Button(self.my_frame3, text="CLOSE", font=("Helvetica", 10, "bold"), fg="black",
bg="red",
                    command=lambda: self.close(2))



    # Viewing Records


    view_head_label = Label(self.my_frame4, text="SEARCH RECORD", font=("Helvetica", 30, "bold"),
fg="blue",
                     bg="black", padx=20, pady=20)
    view_head_label.grid(row=0, column=0)


    search_0ne_button = Button(self.my_frame4, text="Search One Customer", font=("Helvetica", 15,
"bold"),
                       fg="yellow", bg="black", padx=20, pady=10, command=self.See_one_customer)
    search_0ne_button.place(x=150, y=200)



    # search One Customer


    view_
", font=("Helvetica", 10, "bold"),
                       fg="black", bg="red", command=lambda: self.close(6))
        renewPassword = self.renew_passwd.get()
        if oldPassword == "":
            messagebox.showerror("ERROR", "Invalid Entry")
        elif newPassword == "":
            messagebox.showerror("ERROR", "Invalid Entry")


        elif renewPassword == "":
            messagebox.showerror("ERROR", "Invalid Entry")
        else:
            if newPassword != renewPassword:
                self.old_passwd.delete(False, END)
```

```
            self.new_passwd.delete(False, END)
            self.renew_passwd.delete(False, END)
            messagebox.showwarning("Warning", "Passwords Mismatch")
        else:
            try
                db = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )
                cursor = db.cursor()
                cursor.execute(f'SELECT password from admin where admin="{self.adminname1}"')
                password = str(cursor.fetchone())
                if password[2:-3] == oldPassword:
                    cursor.execute(f'UPDATE admin SET password="{newPassword}" WHERE
admin="{self.adminname1}"')
                    db.commit()
                    messagebox.showinfo("SUBMITTED", "Successful")
                    self.my_ntbk.hide(1)
                else:
                    self.old_passwd.delete(False, END)
                    self.new_passwd.delete(False, END)
                    self.renew_passwd.delete(False, END)
                    messagebox.showerror("Error", "Wrong Password")
            except EXCEPTION as e:
                messagebox.showerror("ERROR", e)
    except:
        messagebox.showerror("ERROR", "Error")
  def logout(self):
    self.root.destroy(
  def submit_customer_details(self):
    lis = [self.meterno, self.name, self.address, self.zipcode, self.division, self.phno, self.tariff,
self.email]
```

```
    flag = 0
    for i in lis:

        if i.get() == "":
            flag = 1
            break
    if flag == 1:
        messagebox.showerror("ERROR", "Invalid Entry")
    else:
        response = messagebox.askyesno("SUBMIT", "Are you sure")
        if response == 1:
            try:
                db = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )
                cursor = db.cursor()

                cursor.execute("CREATE TABLE IF NOT EXISTS customer(Meterno INT(10), Name
VARCHAR(50), Zipcode INT(8), "
                               "Address VARCHAR(80),Tariff INT(10), Division VARCHAR(20), Email VARCHAR(40),
Phno INT(10),"
                               "ConsumerNO INT AUTO_INCREMENT PRIMARY KEY)")

                sql_command = "INSERT INTO customer(Meterno, name, zipcode, address, Tariff, division,
Email, phno) " \
                              "VALUES(%s, %s, %s, %s, %s, %s, %s, %s)"
                values = (int(self.meterno.get()), self.name.get(), int(self.zipcode.get()), self.address.get(),
int(self.tariff.get()),
                          self.division.get(), self.email.get(), int(self.phno.get()))

                cursor.execute(sql_command, values)
                db.commit()
```

```
            self.clear(2)
            messagebox.showinfo("SAVE", "SUCCESS")
        except:
            messagebox.showerror("Error", "Invalid Entry")
    else:
        pass
def search_one_customer(self)
    global disp_label
    disp_label.place_forget()
    consumer_no = self.con.get()
    if consumer_no == "":
        messagebox.showerror("ERROR", "Invalid entry")
    elif consumer_no == " ":
        messagebox.showerror("ERROR", "Invalid entry")
    else:
        consumer_no = int(consumer_no)
        try:
            db = mysql.connector.connect(
                host="localhost",
                user="root",
                passwd="Iamankit@02",
                database="ebm"
            )
            cursor = db.cursor()
            cursor.execute(f'SELECT * from customer where consumerno={consumer_no}')
            search_result = cursor.fetchall()
            if search_result == []:
                messagebox.showwarning("Warning", "No Data Found")
            else:
                text = str()


                for result1 in search_result:
                    text = f' NAME :                {result1[1]} \n\n CONSUMER NO :    {result1[8]} \n\n METER NO
:          {result1[0]} \n\n ' \
```

```
                    f'ADDRESS :          {result1[3]} \n\n ZIPCODE :          {result1[2]} \n\n DIVISION :
{result1[5]} \n\n ' \
                    f'TARIFF :           {result1[4]} \n\n PHONE NO. :        {result1[7]} \n\n EMAIL :
{result1[6]}'


            disp_label = Label(self.my_frame5, text=text, font=("Helvetica", 10, "bold"), fg="white",
bg="black",
                               justify=LEFT)
            disp_label.place(x=170, y=200)


        except:
            messagebox.showerror("ERROR", "!!ERROR!!")
    def See_list(self):
        self.my_ntbk.select(5)
    def search_list(self):
        global my_tree
        my_tree = ttk.Treeview(self.my_frame6)
        my_tree['columns'] = ("NAME", "Consumer No", "Meter No")
        my_tree.column("#0", anchor=W, width=0)
        my_tree.column("NAME", anchor=W, width=120)
        my_tree.column("Consumer No", anchor=W, width=120)
        my_tree.column("Meter No", anchor=W, width=120)
        my_tree.heading("NAME", text="Name", anchor=W)
        my_tree.heading("Consumer No", text="Consumer No", anchor=W)
        my_tree.heading("Meter No", text="Meter No", anchor=W)
        my_tree.grid(row=3, column=0, padx=20, pady=20
        selected = self.drop.get()
        value = self.view_list_entry.get()
        if value == "":
            messagebox.showerror("ERROR", "Invalid Entry")
        elif value == " ":
            messagebox.showerror("ERROR", "Invalid Entry")
        else:
            if selected == "Search by...":
                messagebox.showwarning("WARNING", "Invalid Selection")
```

```
        elif selected == "Division"
            if value == "":
                messagebox.showerror("ERROR", "Invalid entry")
            elif value == " ":
                messagebox.showerror("ERROR", "Invalid entry")
            else:
                try:
                    db = mysql.connector.connect(
                        host="localhost",
                        user="root",
                        passwd="Iamankit@02",
                        database="ebm"
                    )
                    cursor = db.cursor()
                    cursor.execute(f'SELECT * from customer where division="{value}"')
                    search_result = cursor.fetchall()
                    if search_result == []:
                        messagebox.showwarning("Warning", "No Data Found")
                    else:
                        i = 0
                        for result1 in search_result:
                            my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                            i += 1
                except:
                    messagebox.showerror("ERROR", "!!ERROR!!")
        elif selected == "Name":
            if value == "":
                messagebox.showerror("ERROR", "Invalid entry")
            elif value == " ":
                messagebox.showerror("ERROR", "Invalid entry")
            else:
                try:
```

```
                db = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    passwd="Iamankit@02",
                    database="ebm"
                )
                cursor = db.cursor()
                cursor.execute(f'SELECT * from customer where Name="{value}"')
                search_result = cursor.fetchall()
                if search_result == []:
                    messagebox.showwarning("Warning", "No Data Found")
                else:
                    i = 0
                    for result1 in search_result:
                        my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                        i += 1
            except:
                messagebox.showerror("ERROR", "!!ERROR!!")
        elif selected == "Zipcode":
            if value == "":
                messagebox.showerror("ERROR", "Invalid entry")
            elif value == " ":
                messagebox.showerror("ERROR", "Invalid entry")
            else:
                value = int(value)
                try
                    db = mysql.connector.connect(
                        host="localhost",
                        user="root",
                        passwd="Iamankit@02",
                        database="ebm"
                    )
                    cursor = db.cursor()
```

```
                cursor.execute(f'SELECT * from customer where Zipcode="{value}"')
                search_result = cursor.fetchall()
                if search_result == []:
                    messagebox.showwarning("Warning", "No Data Found")
                else:


                  i = 0
                    for result1 in search_result:
                        my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                      i += 1
            except:
                messagebox.showerror("ERROR", "!!ERROR!!")
        elif selected == "Tariff":
            if value == "":
                messagebox.showerror("ERROR", "Invalid entry")
            elif value == " ":
                messagebox.showerror("ERROR", "Invalid entry"
            else:
                value = int(value)
                try:
                    db = mysql.connector.connect(
                        host="localhost",
                        user="root",
                        passwd="Iamankit@02",
                        database="ebm"
                    )
                    cursor = db.cursor()
                    cursor.execute(f'SELECT * from customer where Tariff="{value}"')
                    search_result = cursor.fetchall()
                    if search_result == []:
                        messagebox.showwarning("Warning", "No Data Found")
                    else:
                        i = 0
```

```
                for result1 in search_result:
                    my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
                    i += 1
            except:
                messagebox.showerror("ERROR", "!!ERROR!!")
        else:
            messagebox.showwarning("WARNING", "No Data Found")
    def generate_bill(self):
        self.my_ntbk.select(6)


    def generate_bill_fun(self):
        global d
        global consumer
        global total
        global current_reading
        global tariff
        global curr_date
        global curr_month
        global outstanding
        global prev_reading
        global rate
        global fixd_charge
        global phone
        global curr_year
        global net
        tariff = 0
        current_reading = self.cur.get()
        consumer = self.consumer_no.get()
        d = datetime.datetime.now()
        curr_date = d.strftime("%x")
        curr_month = d.strftime("%B")
        curr_year = d.strftime("%Y")
        cur_time = d.strftime("%X")
```

```
    outstanding = 0
    prev_reading = 0
    rate = 0
    fixd_charge = 0
    phone = 0
    division = ''
    add = ''
    name = ''
    meterno = 00
    net =
    if current_reading == "":
        messagebox.showerror("ERROR", "Invalid Entry")
    elif consumer == "":
        messagebox.showerror("ERROR", "Invalid Entry")
    else:
        consumer = int(consumer)
        current_reading = int(current_reading)

        try:
            db = mysql.connector.connect(
                host="localhost",
                user="root",
                passwd="Iamankit@02",
                database="ebm"
            )
                search_result = cursor.fetchall()
                if search_result == []:
                    messagebox.showwarning("Warning", "No Data Found")
                else:
                    i = 0
                    for result1 in search_result:
                        my_tree.insert(parent="", index='end', iid=i, values=(result1[1], result1[8],
result1[0]))
```

```
                i += 1
            except:
                messagebox.showerror("ERROR", "!!ERROR!!")
        else:
            messagebox.showwarning("WARNING", "No Data Found")
    def generate_bill(self):
        self.my_ntbk.select(6)


            # Create A Main Frame
            main_frame = Frame(root1)
            main_frame.pack(fill=BOTH, expand=1)
            # Create A Canvas
            my_canvas = Canvas(main_frame)
            my_canvas.pack(side=LEFT, fill=BOTH, expand=1)
            # Add A Scrollbar To The Canvas
            my_scrollbar = ttk.Scrollbar(main_frame, orient=VERTICAL, command=my_canvas.yview)
            my_scrollbar.pack(side=RIGHT, fill=Y)
            # Configure The Canvas
            my_canvas.configure(yscrollcommand=my_scrollbar.set)
            my_canvas.bind('<Configure>', lambda e:
my_canvas.configure(scrollregion=my_canvas.bbox("all")))
            # Create ANOTHER Frame INSIDE the Canvas
            second_frame = Frame(my_canvas)
            # Add that New frame To a Window In The Canvas
            my_canvas.create_window((0, 0), window=second_frame, anchor="nw")
            cursor = db.cursor()
            try:
                cursor.execute(f'SELECT * from customer where ConsumerNO={consumer}')
                search_result = cursor.fetchall()
                for data in search_result:
                    tariff = data[4]
                    phone = data[7]
                    division = data[5]
                    add = data[3]
```

```
            name = data[1]
            meterno = data[0]
        except:
            messagebox.showwarning("Warning", "No Data found")
        try
            cursor.execute(f'SELECT * from tariff where T_id="{int(tariff)}"')
            tariff_result = cursor.fetchall()
            for i in tariff_result:
                fixd_charge = int(i[2])
                rate = int(i[1])
        except:
            messagebox.showwarning("Warning", "No Data found"
        try:
            cursor.execute(f'SELECT dob, reading as last FROM reading where con_no={consumer}
ORDER BY dob DESC LIMIT 1')
            reading_data = cursor.fetchall()
            for read in reading_data:
                prev_date = read[0]
                prev_reading = int(read[1])
                print(prev_reading)
                print(prev_date)

        except:
            messagebox.showwarning("Warning", "No Data found")
        try:
             cursor.execute(f'SELECT out_amount as last FROM outstanding_amt where
con_no={consumer} ORDER BY date_reading DESC LIMIT 1')
             out = cursor.fetchall()
             for outamt in out:
                 outstanding = outamt[0]
        except:
            messagebox.showwarning("Warning", "No Data found")

        diff_reading = current_reading - prev_reading
```

```
            energy_charge = diff_reading * rate
            total = energy_charge + fixd_charge
            net = total + outstanding

            bill_content = f' Electricity Department \n\n\n Bill \n\n\n\n\n' \
                        f'Division : {division} \n\n Consumer No. : {consumer} \n\n Name : {name} \n\n ' \
                        f'Address : {add} \n\n Phone No. : {phone}' \
                        f'\n\n\n --------Comercial Detaills-------- \n\n\n ' \
                        f'Tariff : {tariff} \n\n Load : 220V \n\n' \
                        f'--------Meter Details-------- \n\n\n' \
                        f'Meter No. : {meterno} \n\n ' \
                        f'--------Billing Parameter-------- \n\n\n' \
                        f'Bill Issue Date : {curr_date} \n\n Bill Month : {curr_month} \n\n Time : {cur_time}
\n\n Date Of Previous Reading' \
                        f' : {prev_date} \n\n Previous Reading : {prev_reading} \n\n Current Reading :
{current_reading} \n\n ' \
                        f'Difference : {diff_reading} \n\n ' \
                        f'--------Current Assessment--------\n\n\n' \
                        f'Energy Charge : {energy_charge} \n\n Fixed Charge = {fixd_charge} \n\n ' \
                        f'--------Net Demand-------- \n\n\n' \
                        f'Total Amount : {total} \n\n Outstanding Amt : {outstanding} \n\n Net Payable: {net}'
            bill_label = Label(second_frame, text=bill_content, font=("Helvetica", 12, "bold"), justify=LEFT)
            bill_label.grid(row=0, columnspan=2, column=0)
            bill_button = Button(second_frame, text="SAVE", font=("Helvetica", 12, "bold"), bg="green",
fg="black",
                              command=self.save_bill)

            root1.mainloop()
        except EXCEPTION as e:
            messagebox.showerror("ERROR", "!!ERROR!!")
            print(e)
    def save_bill(self):
        global consumer
        global d
        global current_reading
```

```
        global total
        global prev_reading
        global outstanding
        global curr_month
        global curr_year
        global net

        try:
            db = mysql.connector.connect(
                host="localhost",
                user="root",
                passwd="Iamankit@02",
                database="ebm"
            )
            cursor = db.cursor()
            sql_command = "INSERT INTO reading(dob, con_no, reading) VALUES(%s,%s,%s)"
            values = (d, consumer, current_reading)
            sql_command1 = "INSERT INTO bill(bill_date, tot_amt, con_no, current_reading,
previous_reading, " \
                        "outstanding_amt, bill_month, year) VALUES(%s,%s,%s,%s,%s,%s, %s)"
            values1 = (d, total, consumer, current_reading, prev_reading, outstanding, curr_month,
curr_year)
            sql_command2 = "INSERT INTO outstanding_amt(con_no, date_reading, out_amount, tot_amt,
out_month)" \
                        "values(%s,%s,%s,%s,%s)"
            values2 = (consumer, d, net, total, curr_month)
            cursor.execute(sql_command, values)
            cursor.execute(sql_command1, values1)
            cursor.execute(sql_command2, values2)
            db.commit()
        except EXCEPTION as ee:
            messagebox.showerror("ERROR", "Unable to Save")
            print(ee)
    def pay_and_save(self):
        webbrowser.open("https://rzp.io/l/uqQgiTS")
```

Testing Approach

Software testing is a process used to identify the correctness, completeness and quality of developed computer software. It includes a set of activities conducted with the intent of finding errors in software so that it could be corrected before the product is released to the end users. In other word software testing is an activity to check that the software system is defect free.

Software testing is primarily a broad process that is composed of several interlinked processes. The primary objective of software testing is to measure software health along with its completeness in terms of core requirements. Software testing involves examining and checking software through different testing processes.

The objectives of these processes can include:

- **Completeness** - Verifying software completeness in regards to functional/business requirements
- **Errors Free** - Identifying technical bugs/errors and ensuring the software is error-free
- **Stability** - Assessing usability, performance, security, localization, compatibility and installation

This phase determines the error in the project. If there is any error then it must be removed before delivery of the project.

Type of Testing

For determining errors various types of test action are performed: -

Unit testing: - Unit testing focuses verification effort on the smallest unit of software design – the module. Using the detail design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors detected as a result is limited by the constrained scope established for unit testing. The unit test is always white box oriented, and the step can be conducted in parallel for multiple modules.

Unit testing is normally considered an adjunct to the coding step. After source level code has been developed, reviewed, and verified for correct syntax, unit test case design begins

Test case

Test case for Login Page.

Test Case Id -TC00I

Sr.No.	Input/ Action	Expected Result	Actual Result	Remark
1	Leave text/ Field empty.	Will show error message “Invalid Entry”	Error message “Invalid Entry”	Pass
2	Entered Invalid username or password.	Will show error message " “Invalid Username Or Password "	Error message " Invalid Username Or Password "	Pass
3	Entered Valid Username.	Opens The Page	Opens The page	Pass

Test case for adding customer

Test Case Id -TC002

Sr.No.	Input/Action	Expected Result	Actual Result	Remark
1	Leave text/ field empty	Will show error message ” Invalid Entry"	Error message “Invalid Entry"	Pass
2	Entered Valid data	Will accept the Data.	Data accepted	Pass

Test case for Report

Test Case Id -TC003

Sr.No.	Input/Action	Expected Result	Actual Result	Remark
1	Leave text/ field empty	Will show error message “0- Records Found"	Error message “0- Records Found"	Pass
2	Entered Valid data	Will accept the Data.	Data accepted	Pass

Test Case for Bill Generation

Test Case Id – TC004

Sr.No.	Input/Action	Expected Result	Actual Result	Remark
1.	Leave Text/ Field Empty/ Wrong Input	Will show message “Invalid Entry”	Error Message “Invalid Entry”	Pass
2.	Enter Valid Data	Will accept data and Generate Bill	Generates Bill	Pass

Test case for Payment

Test Case Id- TC005

Sr.No	Input/Action	Expected Result	Actual result	Remark
1.	Payment Failed Due to any reason.	Shows message “Payment Failed”	Error Message “ Payment Failed”	Pass
2.	On successful Payment	Shows Message “Payment Successful” Generates payment Id.	Generates Payment Id and updates the concerned database.	Pass

Integration Testing - A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

System Testing: **-** Software is only one element of a larger computer-based system. Ultimately, software is incorporated with other system elements (e.g. new hardware, information), and a series of system integration and validation tests are conducted. Steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.

A classics system testing problem is "finger pointing". This occurs when a defect is uncovered, and one system element developer blames another for the problem. The software engineer should anticipate potential interfacing problems and design error handling paths that test all information coming from other elements of the system, conduct a series of tests that simulate bad data or other potential errors at the software interface, record the results or tests to use as "evidence" if finger pointing does occur, participate in the planning and design of system test to ensure that software is adequately tested.

There are many types of system tests that are worthwhile for software-based systems:-

Usability Testing - Usability Testing is a type of testing done from an end-user's perspective to determine if the system is easily usable.

Functionality testing - Tests all functionalities of the software against the requirement.

Performance testing – Performance testing is designed to test the run-time performance of software within the context of an integrated system

Security testing – Security testing attempts to verify that protection mechanisms built into a system will protect it from improper penetration

Stress tests – Stress tests are designed to confront programs with abnormal situations

Use Case

A use case diagram is essentially a picture showing system behavior along with the key actors that interact with the system. The use case represents complete functionality. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box are known. Use Case elements are used to make test cases when performing the testing. The use case should contain all system activities that have significance to the users. A use case can be thought of as a collection of possible scenarios related to a particular goal, indeed. Use cases can be employed during several stages of software development, such as planning system requirements, validating design and testing software.

<u>Use case Diagram Objects</u>

Use case diagrams mostly consist of 3 objects: -

Actor - Actor is a use case diagram is any entity that performs a role in one given system. This could be a person, organization or an external system.

Use Case - A Use case represents a function or an action within the system. its drawn as an oval and named with the function.

System - System is used to define the define the scope of the use case and drawn as rectangle.

There are two functions: -

Include – This represents required. Symbol of this function is dashed arrow and arrow is labeled

----------->

with the keyword <<include>>

Extend – This represents optional and it is also shown with dashed arrow the arrow is labeled with the keyword <<extend>>

<----------

Use Case Diagram

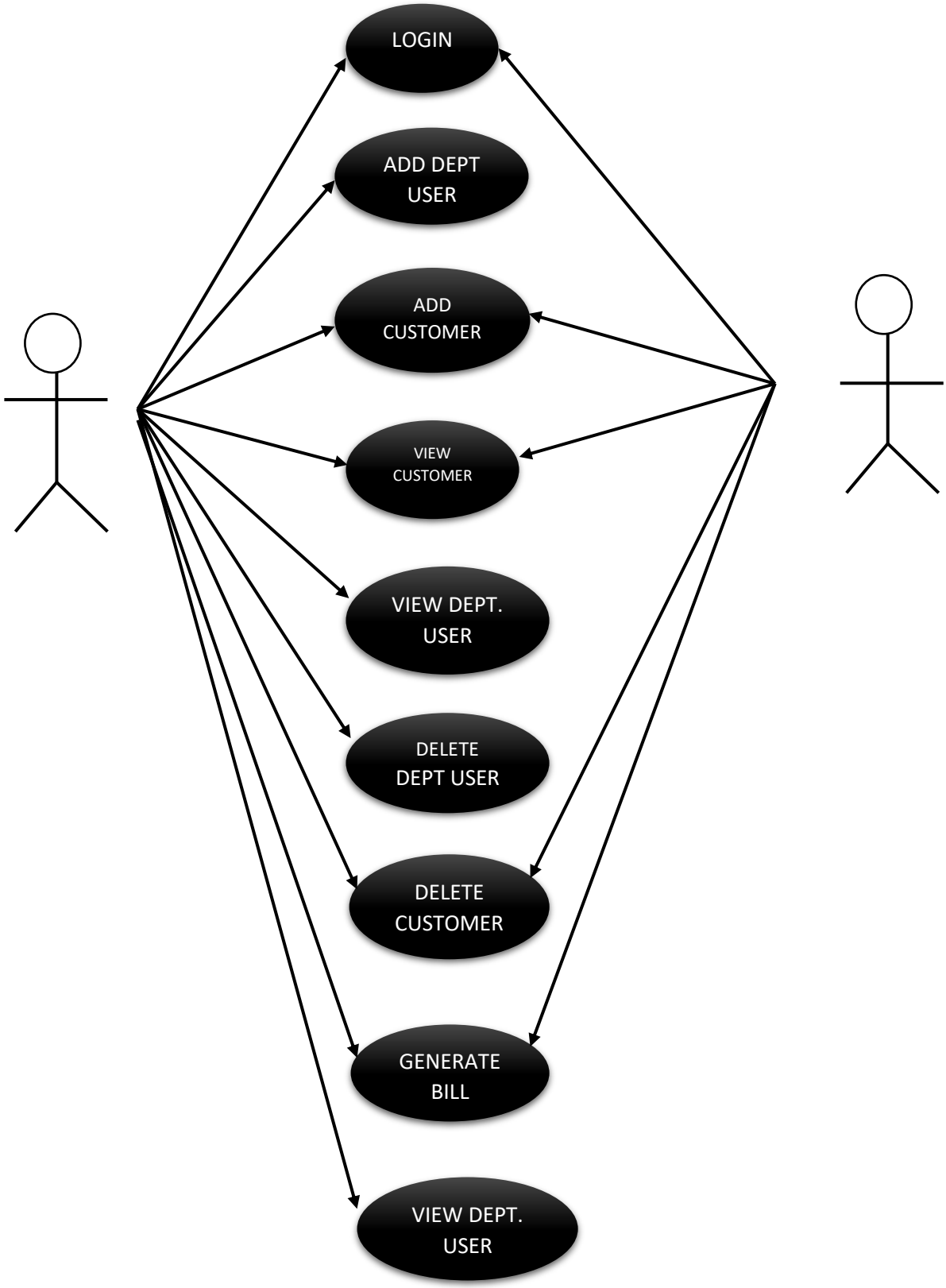
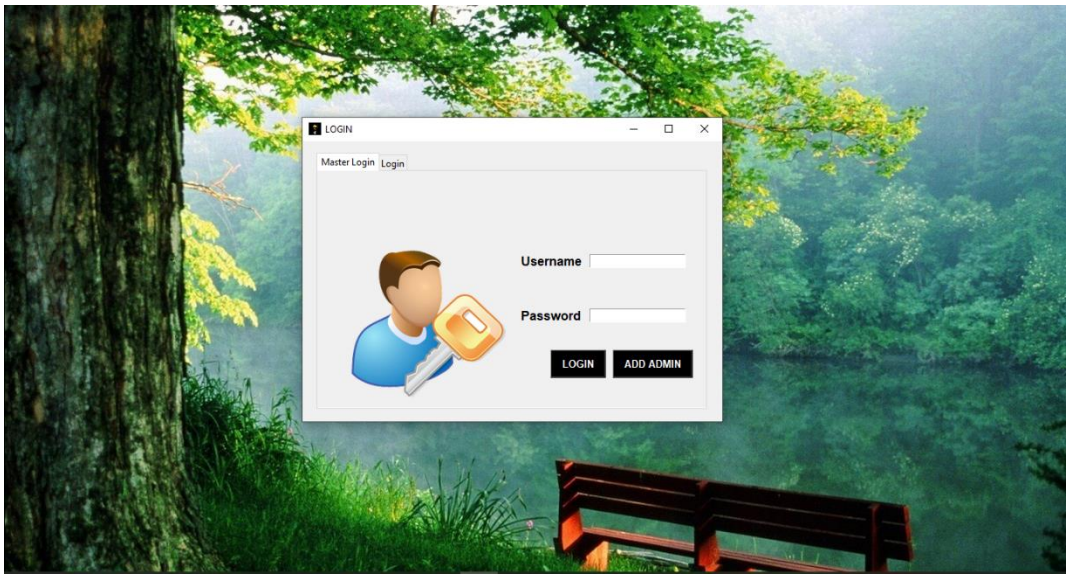


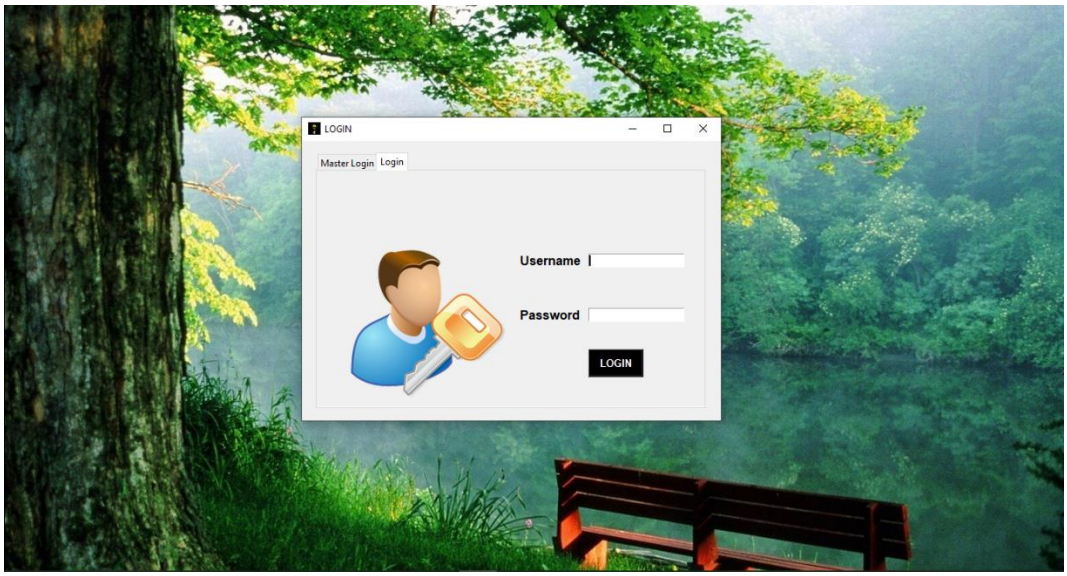
Figure 5: Use Case Diagram

CHAPTER 6: RESULTS AND DISCUSSIONS

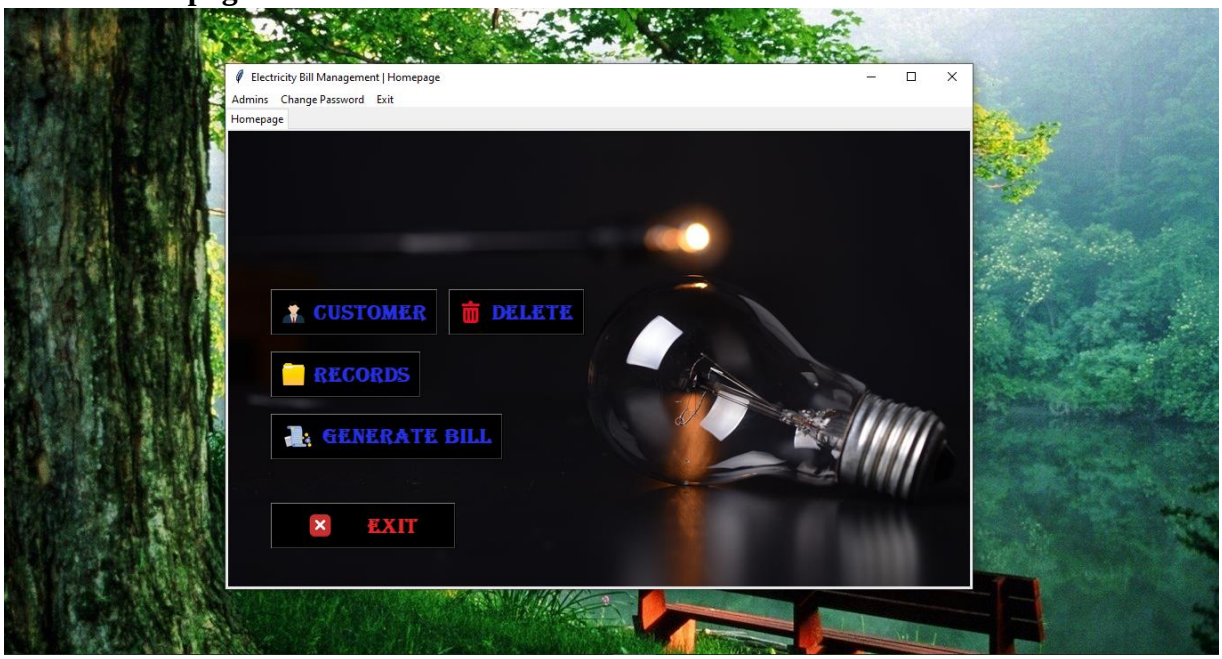
Admin Login



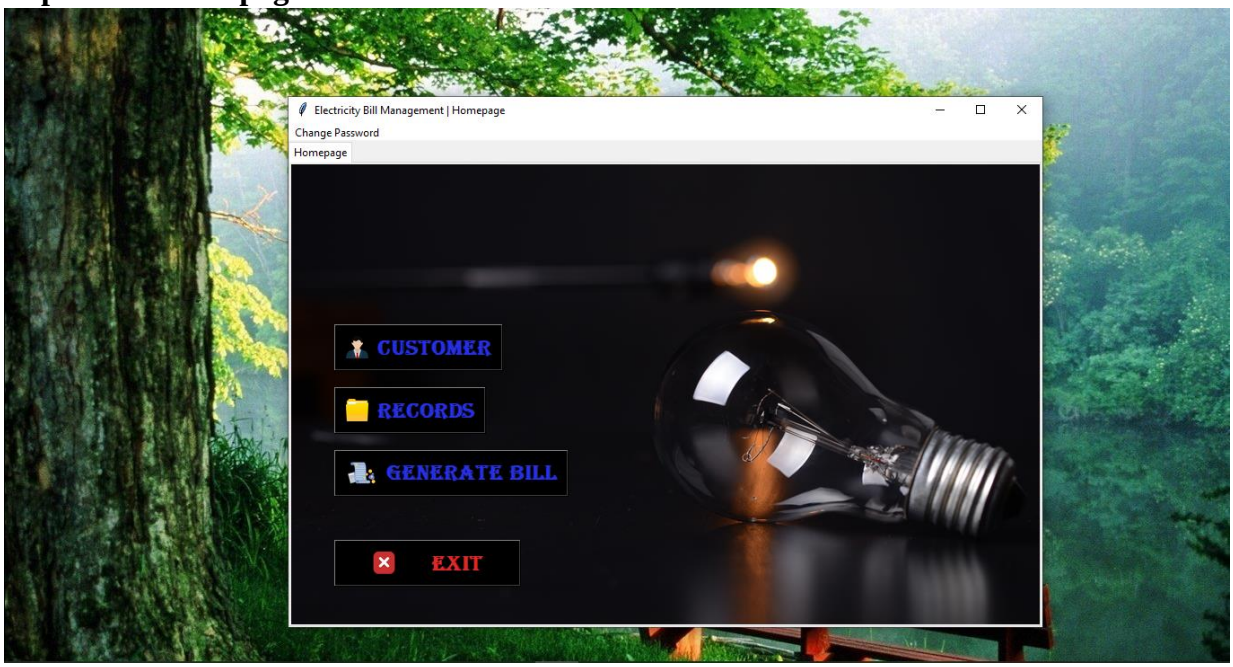
User Dept. Login Page



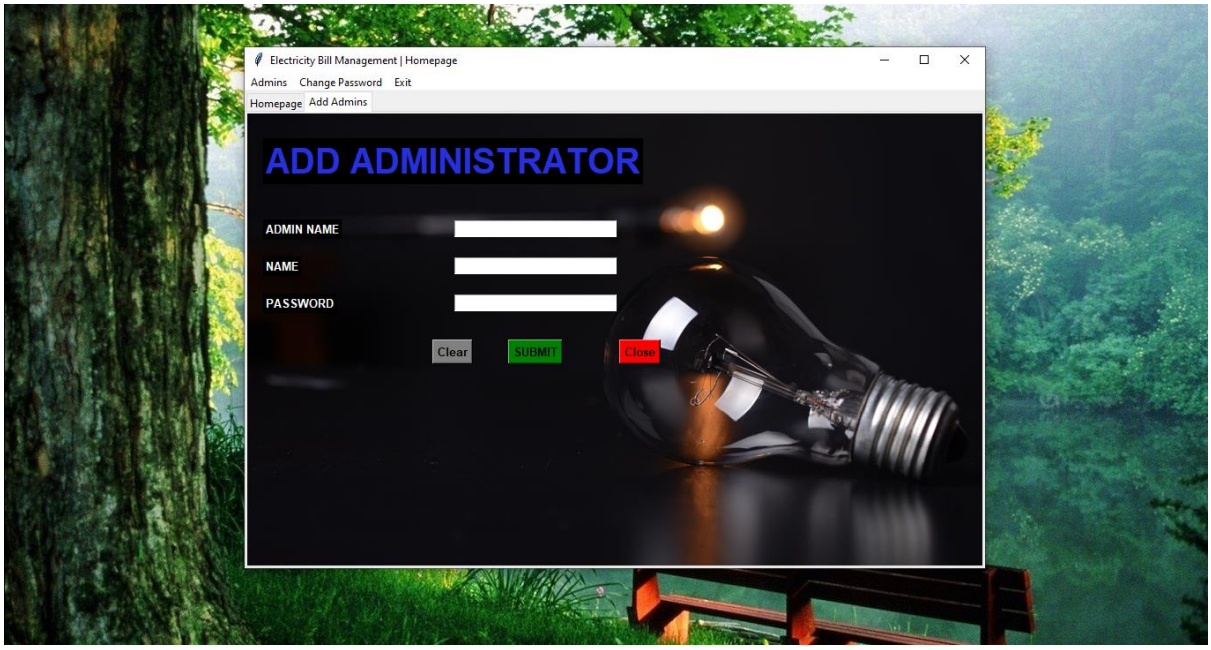
Admin Homepage



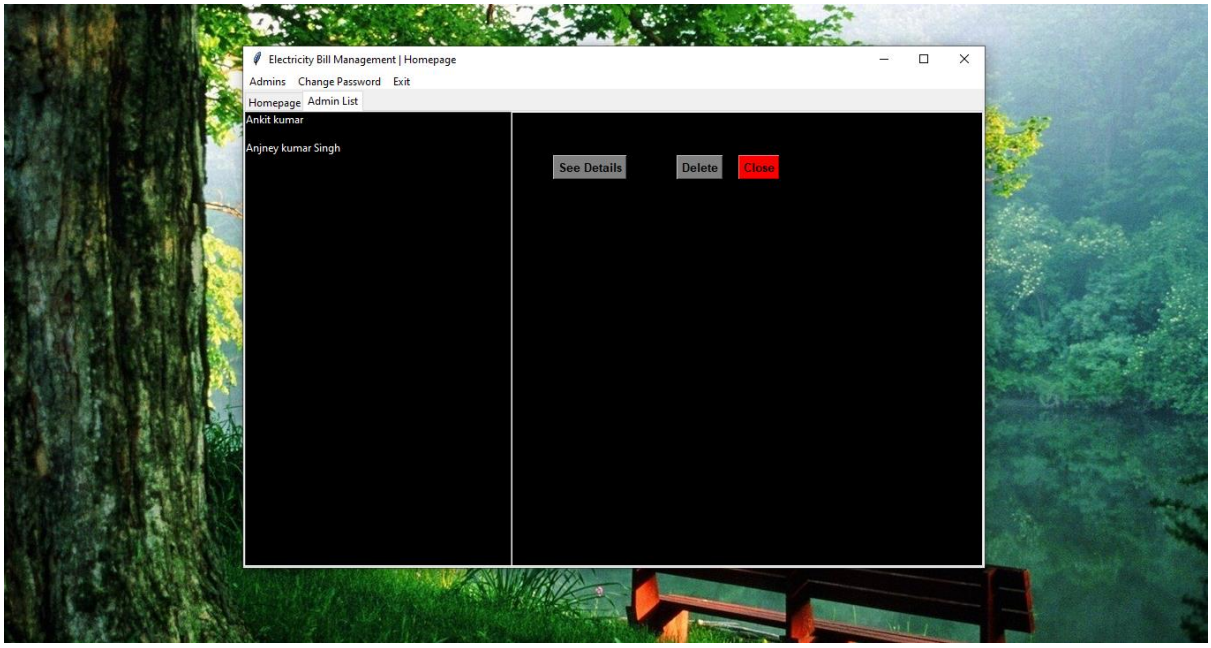
Dept. User Homepage



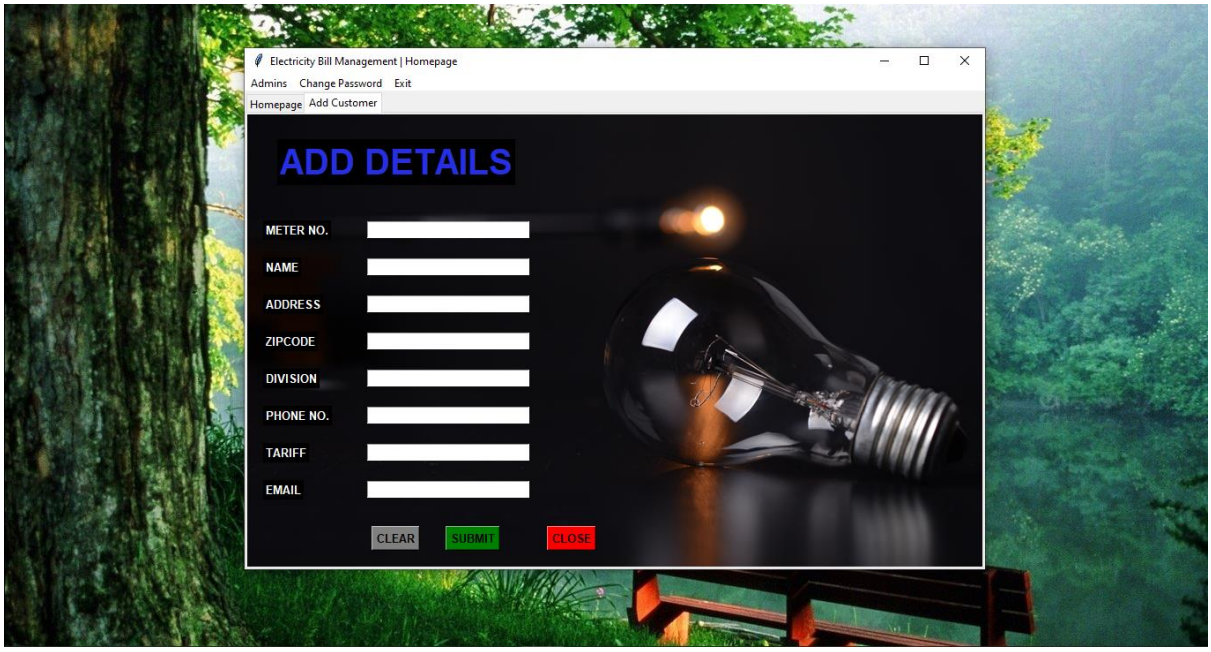
Add Dept. Users



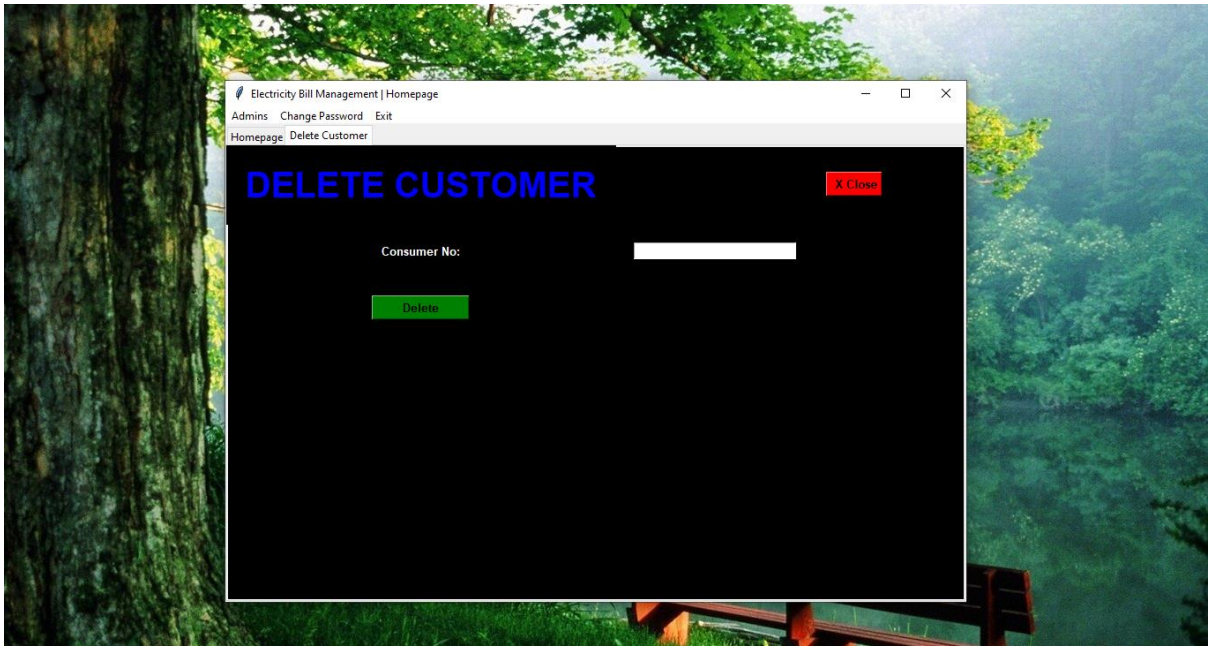
Manage Dept. Users



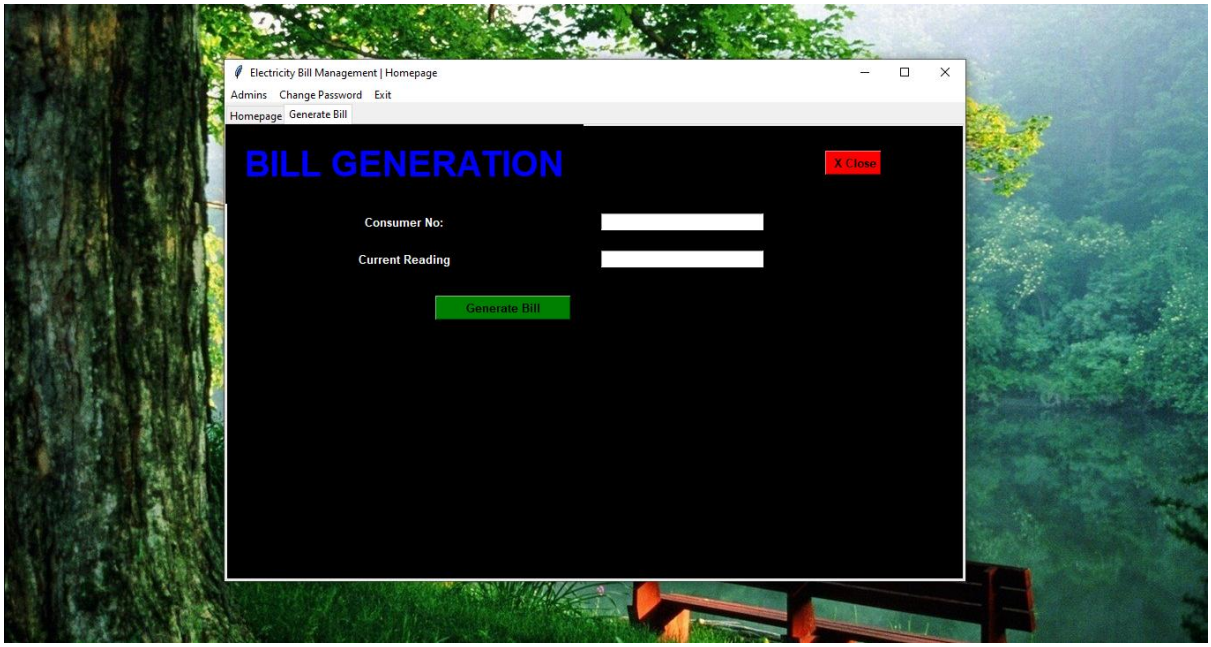
Add Customers



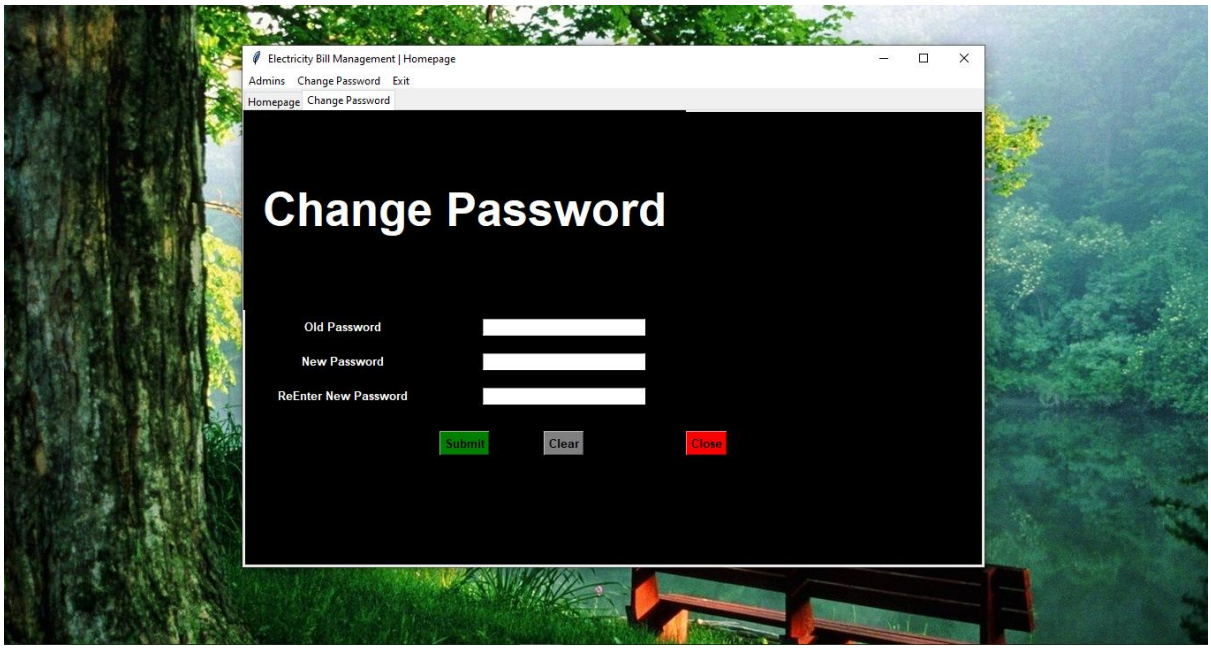
Delete Customers



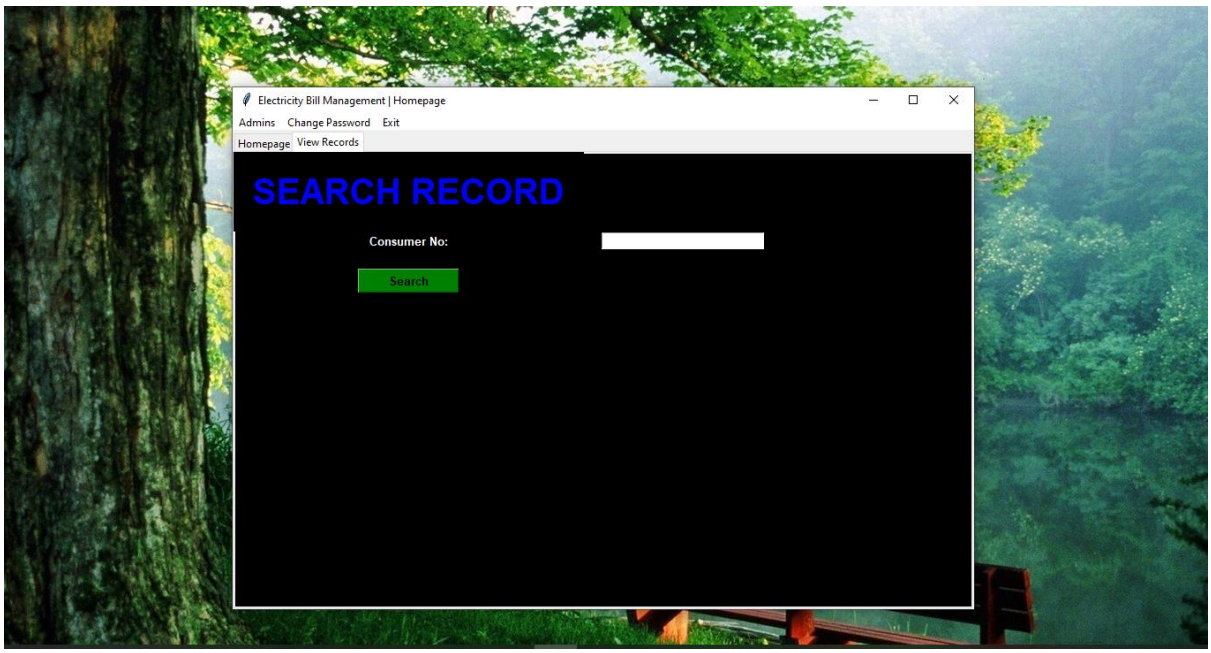
Bill Generation



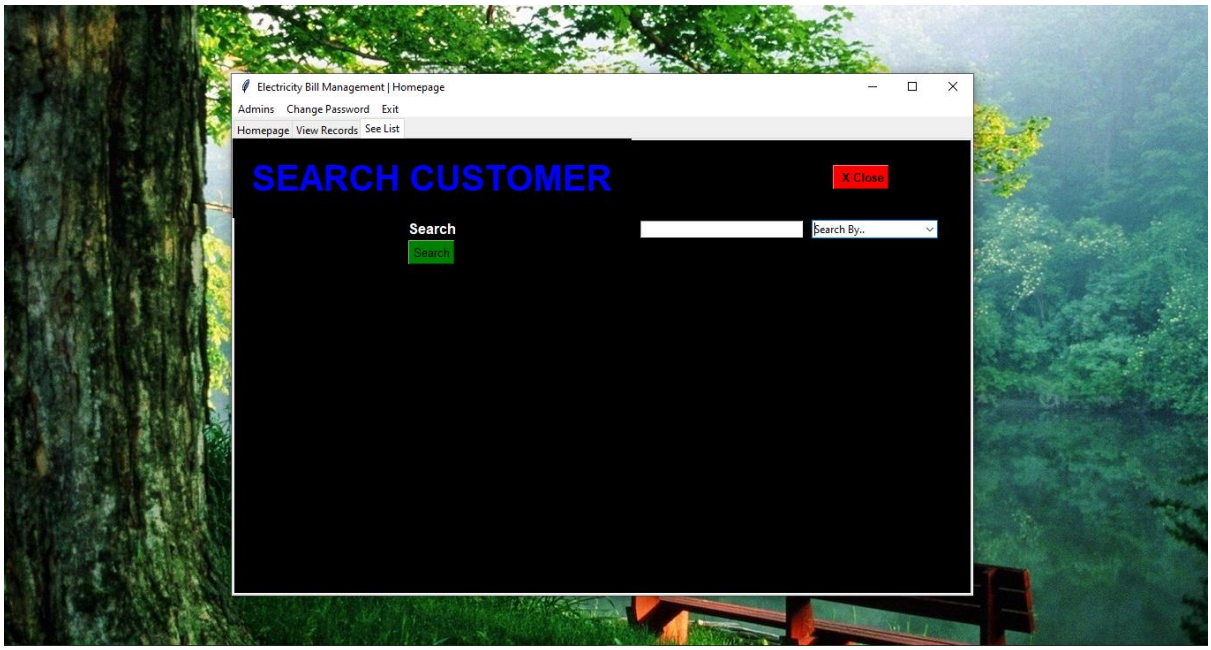
Change Password



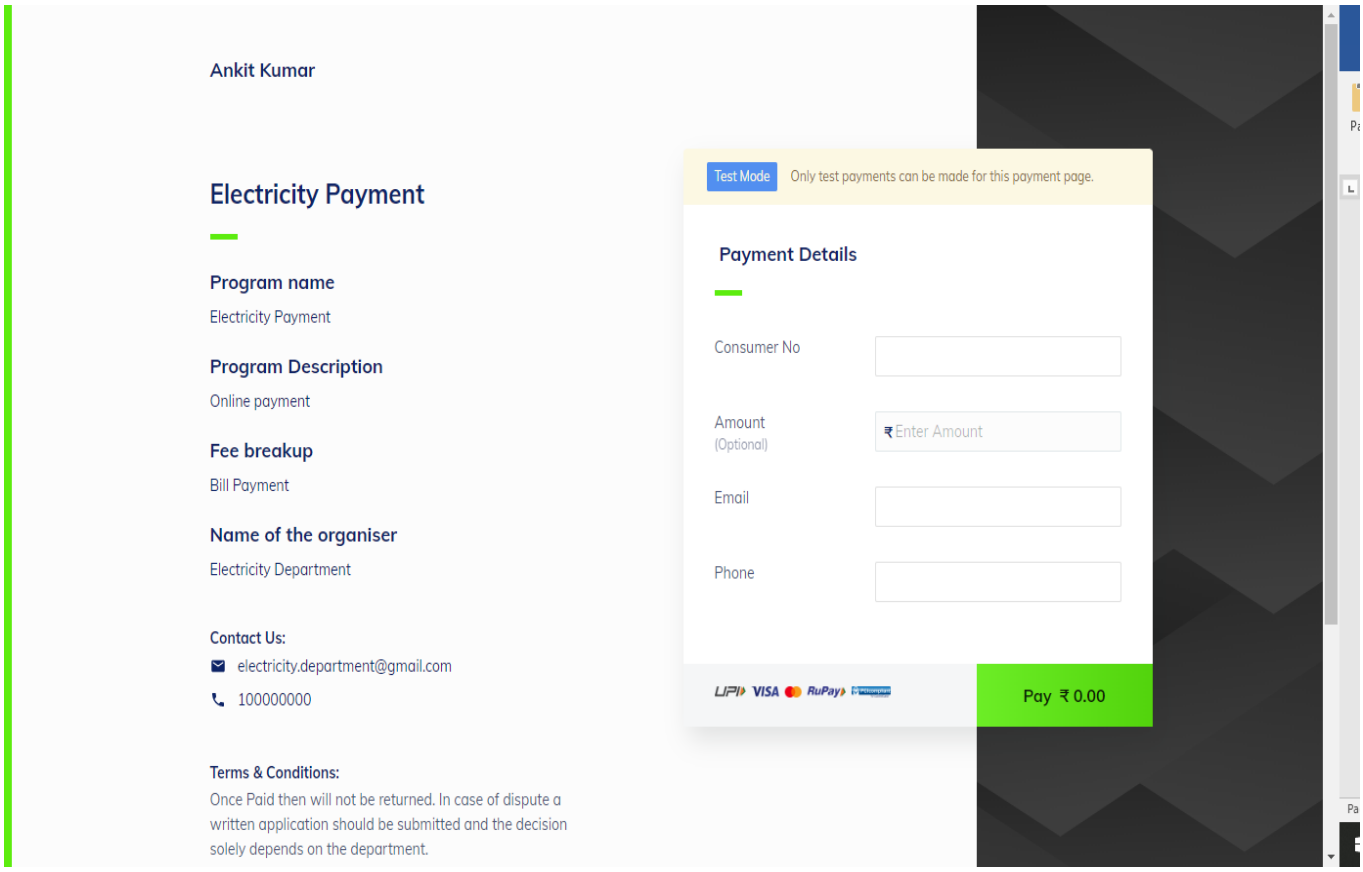
Search One Customer



Search List Of customers



Payment


Ankit Kumar
Electricity Payment
Program name
Electricity Payment
Program Description
Online payment
Fee breakup
Bill Payment
Name of the organiser
Electricity Department
Contact Us:
electricity.department@gmail.com
100000000
Terms & Conditions:
Once Paid then will not be returned. In case of dispute a written application should be submitted and the decision solely depends on the department.
Test Mode
Only test payments can be made for this payment page.
Payment Details
Consumer No
Amount
(Optional)
₹ Enter Amount
Email
Phone
Pay ₹ 0.00

CHAPTER 7: CONCLUSION & FUTURE SCOPE

7.1. CONCLUSION

The “ELECTRICITY BILL MANAGEMENT SYSTEM" has been computed successfully and was also tested successfully by taking "Test Cases". It is user friendly, and has required options, which can be utilized by the Electricity distributer to perform the desired operations.

The Software is developed using TKINTER (PYTHON) as front end, PYTHON as back end and MY- SQL Server as the database in windows environment.

The goals that are achieved by the software are:

- Simplification of the operations
- Less processing time and getting required information
- User friendly
- Portable and flexible for further enhancement

7.2 FUTURE SCOPE

We have left all the options open so that there is any futre requirements in the system by the electricity discoms for the enhancement of the system then it is possible to implement them.
Today, the market place is flooded with several E Management options for societies and their members to choose from. A variety of members are being offered worst services by the societies. In the last couple of years, the growth of IT Industry has been phenomenal as more have started discovering the benefits of using this platform. Therefore keeping in mind every requirement of a Electricity distributer we have built this system. In future, the UI can be made more intuitive for better user experience. And we will make this system live and provide Software as Service (SAS).

REFERENCES

The following reference has been used to develop the project "ELECTRICITY BILL MANAGEMENT SYSTEM":

Books: -

- SQL QuickStart Guide: The Simplified Beginner's Guide to Managing, Analyzing, and Manipulating Data With SQL
- Python Programming : A modular Approach by Sheetal Taneja & Naveen Kumar

Web Source: -

- https://www.google.com
- https://www.youtube.com
- https:/www.geeksforgeeks.com
- Notes From Arka Jain University
- MySQL. Documentation
- Python Documentation