

# **ELECTRICITY BILL MANAGEMENT SYSTEM**

**A MINI PROJECT REPORT**

**SUBMITTED BY**

LINGESH V K	221701032
KOTTESWARAN S	221701031
DWIJESH SREERAM S	221701014
DHANUSH M	221701013

In partial fulfillment for the award of the degree of

**BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND  
DESIGN**

**RAJALAKSHMI ENGINEERING COLLEGE**

**THANDALAM CHENNAI – 602105**



**ANNA UNIVERSITY: CHENNAI 600625**

## **BONAFIDE CERTIFICATE**

Certified that this project report “ELECTRICITY BILL MANAGEMENT SYSTEM” is the Bonafide work of “DHANUSH M (221701013), LINGESH V K (221701032), KOTTESWARAN S (221701031), DWIJESH SREERAM S (221701014)” who carried out the project work under my supervision.

### **SIGNATURE**

Mr.S.Uma Maheswara Roa M.E.,  
Professor and Head,  
Computer Science and Design,  
Rajalakshmi Engineering College,  
Thndalam, Chennai – 602105.

EXTERNAL EXAMINER

### **SIGNATURE**

Mr.Vijaykumar M.Tech.,  
Asst. Professor (SS),  
Computer Science and design,  
Rajalakshmi Engineering College,  
Thandalam, Chennai – 602105.

INTERNAL EXAMINAR

## **ACKNOWLEDGEMENT**

We are highly obliged in taking the opportunity to thank our Chairman Mr. S. Meganathan, Chairperson Dr.Thangam Meganathan and our Principal Dr.S.N.Murugesan for providing all the facilities which are required to carry out this project work.

We are ineffably indebted to our H.O.D MR.S.UMA MAHESWARA RAO M.E., for his conscientious guidance and encouragement to make this project a recognizable one.

We are extremely thankful to our faculty Mr.Vijaykumar M.Tech., for his valuable guidance and indefatigable support and extend our heartfelt thanks to all the teaching and non-teaching staff of Computer Science department who helped us directly or indirectly in the completion of this project successfully.

At last but not least gratitude goes to our friends who helped us compiling the project and finally to god who made all things possible.

Any omission in this brief acknowledgement doesn't mean lack of gratitude.

DHANUSH M 221701013

DWIJESH SREERAM S 221701014

LINGESH V K 221701032

KOTTESWARAN S 221701031

## **ABSTRACT**

Electricity Bill Management System is a web-based application designed to streamline and automate the process of managing electricity bills for both administrators and customers. Developed using a combination of technologies—Python, Flask, MySQL, HTML, CSS, and JavaScript, Node.js, HTML, CSS, and MySQL—this system offers a user-friendly interface for effective billing and consumption tracking. The platform allows administrators to manage customer accounts, monitor bill generation, and update rates, while customers can view their billing history, consumption details, and make payments. The system centralizes data storage, ensuring secure and efficient handling of billing information. This project aims to reduce administrative workload, improve data accuracy, and provide customers with easy access to their electricity usage and billing information in real-time..

## TABLE OF CONTENTS

	Page No.
<b>1. INTRODUCTION</b>	
1.1 INTRODUCTION	1
1.2 SCOPE OF THE WORK	
1.3 PROBLEM STATEMENT	
1.4 AIM AND OBJECTIVES OF THE PROJECT	
<b>2. SYSTEM SPECIFICATION</b>	
2.1 Hardware and software specifications	8
<b>3. SOFTWARE DESCRIPTION</b>	
3.1 Android Studios	9
3.1.1 Features	
<b>4. PROJECT DESCRIPTION</b>	
4.1 Module Description	11
4.2.1 Student	
4.2.2 Company	
<b>5. IMPLEMENTATION</b>	
5.1 Source code	12
5.2 Screen Shots	
5.3	
<b>6. CONCLUSION</b>	
REFERENCES	33
<b>7. BIBLIOGRAPHY</b>	34

# CHAPTER – 1

## INTRODUCTION

### 1.1. INTRODUCTION

The Electricity Bill Management System addresses the growing demand for an automated solution that allows customers and administrators to handle electricity billing efficiently. This system offers an intuitive interface where customers can view their electricity consumption, payment history, and bills, while administrators can manage customer information, set rates, generate bills, and analyze usage patterns. It improves transparency for customers and streamlines administrative tasks.

### 1.2. SCOPE OF THE WORK

The scope of this project includes developing a full-fledged web-based application that manages all aspects of electricity billing:

- Customer Module: Allows customers to register, view their monthly bills, check consumption, and make payments.
- Admin Module: Provides administrators with functionalities to manage customer data, set electricity rates, generate bills, and monitor payments.
- Data Management: Ensures secure storage and retrieval of customer and billing information in a centralized database.
- User Interface: Provides an intuitive and responsive user interface for both customer and admin modules.

### 1.3. PROBLEM STATEMENT

Managing electricity billing through manual or semi-automated methods is often time-consuming, inefficient, and prone to human error. This can result in billing inaccuracies, delayed payments, and reduced customer satisfaction. Utility companies need a centralized, automated system that can handle multiple processes

simultaneously—such as data management, bill generation, and payment tracking—to ensure seamless operations. The current systems often lack real-time data visibility for customers and may have inadequate security and data integrity.

#### 1.4 AIM AND OBJECTIVES OF THE PROJECT

**Aim:** To develop an automated Electricity Bill Management System that simplifies billing operations and provides real-time access to billing information for both administrators and customers.

**Objectives:**

1. Design a centralized, user-friendly web application that integrates customer and admin modules.
2. Implement secure data management for handling customer information, billing data, and payment history.
3. Enable administrators to manage rates, monitor consumption, generate bills, and track payments.
4. Provide customers with secure, real-time access to their monthly bills, consumption details, and payment options.
5. Enhance the accuracy, efficiency, and transparency of the billing process through automation and data integrity.

## **CHAPTER – 2**

### **SYSTEM SPECIFICATIONS**

#### **2.1 HARDWARE SPECIFICATIONS**

Processor	:	Intel i5
Memory Size	:	8GB (Minimum)
HDD	:	1 TB (Minimum)

#### **2.2 SOFTWARE SPECIFICATIONS**

Operating System	:	WINDOWS 10
Front – End	:	Python
Back - End	:	Sqlite3
Language	:	python,SQL



# CHAPTER - 3

## MODULE DESCRIPTION

The Electricity Bill Management System is a Python-based desktop application that facilitates the management of electricity bills for customers and administrators. The system is built using **Tkinter** for the GUI and **SQLite3** for database management. It provides functionality for both customers and admins to interact with the system, with features such as bill management, account management, and billing status updates.

### Modules:

#### 1. Database Module (sqlite3):

- This module is used for creating and managing the SQLite database that stores user data and bill information.
- **Tables:**
  - **users:** Stores user credentials (username, password, role—admin or customer).
  - **bills:** Stores bill details (bill ID, customer ID, usage, bill amount, payment status).
- **Key Operations:**
  - Creating, reading, updating, and deleting records for users and bills.

#### 2. GUI Module (Tkinter):

- This module is used to create a responsive and interactive graphical user interface for the system.
- **Main Features:**
  - **Admin Dashboard:** The admin can view, add, update, and delete bills.
  - **Customer Dashboard:** The customer can view their individual bills and check their payment status.
  - **Login/Signup:** Users (both admins and customers) can create an account or sign in to their existing account.

#### 3. Authentication and User Management Module:

- This module handles user authentication (login/signup) and ensures only authorized users can access specific features.
- Admin users have full control over managing bills, while customers can only view their bills.
- **Features:**
  - **Signup:** Users can register with a unique username and password.
  - **Login:** Users can log in with their credentials.

- **Role-based Access Control:** Differentiates between admin and customer functionalities.

#### 4. **Bill Management Module:**

- This module allows for adding, updating, viewing, and deleting bills.
- **Admin Functions:**
  - **Add Bill:** Admins can add new bills for customers, specifying usage, amount, and payment status.
  - **Update Bill:** Admins can modify an existing bill, including the usage and payment status.
  - **Delete Bill:** Admins can remove a bill from the system.
- **Customer Functions:**
  - **View Bills:** Customers can view a list of their past bills along with usage, amount, and payment status.

#### 5. **UI Customization and Features:**

- The application features a customized interface with background images, color schemes, and font styles to enhance user experience.
- Components include:
  - **Background Image:** A customizable background image displayed behind all interface elements.
  - **Font and Button Styles:** Custom fonts and button colors to make the UI more user-friendly and aesthetically pleasing.
  - **Error Handling:** Informative messages and error handling to guide users.

#### 6. **Error Handling and Validation Module:**

- This module ensures that users provide valid inputs for signup, login, and bill management.
- **Features:**
  - Checks for valid username/password on login.
  - Ensures bill fields like customer ID, bill amount, and usage are entered correctly before adding or updating bills.
  - Provides appropriate error messages when inputs are invalid or incomplete.

#### 7. **Message Box Module (from tkinter.messagebox):**

- This module is used for displaying information messages, warnings, and errors.
- **Key Operations:**
  - **Info Message:** Displays success messages (e.g., "Bill updated successfully").

- **Error Message:** Displays error messages in case of failed operations (e.g., "Invalid credentials").
- **Confirmation Message:** Used for confirming actions like deleting a bill.

## CHAPTER – 4

### SAMPLE CODING

```
import tkinter as tk
from tkinter import messagebox
import sqlite3
from tkinter import PhotoImage

# Database Setup
conn = sqlite3.connect('electricity_bills.db')
cursor = conn.cursor()

# Create required tables if they do not exist
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    role TEXT NOT NULL
)
""")
cursor.execute("""
CREATE TABLE IF NOT EXISTS customers (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    address TEXT,
    contact TEXT
)
""")
cursor.execute("""
CREATE TABLE IF NOT EXISTS bills (
    bill_id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```

        customer_id INTEGER,
        usage INTEGER,
        bill_amount REAL,
        status TEXT,
        FOREIGN KEY(customer_id) REFERENCES customers(id)
    )
    """)
conn.commit()

# Initialize Admin and some default data
cursor.execute("INSERT OR IGNORE INTO users (username, password, role) VALUES ('admin', 'admin123', 'admin')")
conn.commit()

# Tkinter App
class ElectricityBillApp:
    def __init__(self, root):
        self.root = root

        self.root.title("Electricity Bill Management System")
        self.root.geometry("800x600")
        self.root.configure(bg='#ADD8E6') # Light blue background

        # Background Image
        self.bg_image = PhotoImage(file="b.png") # Change this path as per your image location
        self.bg_label = tk.Label(self.root, image=self.bg_image)
        self.bg_label.place(relwidth=1, relheight=1)

        self.frame = tk.Frame(self.root, bg='#ADD8E6')
        self.frame.pack(pady=20)

        self.login_screen()

    def login_screen(self):

```

```

        for widget in self.frame.winfo_children():
            widget.destroy()

        tk.Label(self.frame, text="Login", font=("Arial", 20, 'bold'), bg='#ADD8E6', fg="white").grid(row=0,
column=0, columnspan=2)

        tk.Label(self.frame, text="Username:", bg='#ADD8E6', font=("Arial", 14)).grid(row=1, column=0,
pady=10)

        self.username_entry = tk.Entry(self.frame, font=("Arial", 14))

        self.username_entry.grid(row=1, column=1)

        tk.Label(self.frame, text="Password:", bg='#ADD8E6', font=("Arial", 14)).grid(row=2, column=0)

        self.password_entry = tk.Entry(self.frame, show="*", font=("Arial", 14))

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

        login_btn = tk.Button(self.frame, text="Login", font=("Arial", 14), command=self.login, bg="#4CAF50",
fg="white")

        login_btn.grid(row=3, column=0, columnspan=2, pady=20)

        signup_btn = tk.Button(self.frame, text="Sign Up", font=("Arial", 14), command=self.signup_screen,
bg="#FF5722", fg="white")

        signup_btn.grid(row=4, column=0, columnspan=2)

def login(self):

    username = self.username_entry.get()

    password = self.password_entry.get()

    cursor.execute("SELECT * FROM users WHERE username=? AND password=?", (username, password))

    user = cursor.fetchone()

    if user:

        role = user[3]

        if role == "admin":

            self.admin_dashboard()

```

```

        else:

            self.customer_dashboard(username)

    else:

        messagebox.showerror("Login Failed", "Invalid username or password")

def signup_screen(self):

    for widget in self.frame.winfo_children():

        widget.destroy()

    tk.Label(self.frame, text="Sign Up", font=("Arial", 20, 'bold'), bg='#ADD8E6', fg="white").grid(row=0,
column=0, columnspan=2)

    tk.Label(self.frame, text="Username:", bg='#ADD8E6', font=("Arial", 14)).grid(row=1, column=0,
pady=10)

    self.new_username_entry = tk.Entry(self.frame, font=("Arial", 14))
    self.new_username_entry.grid(row=1, column=1)

    tk.Label(self.frame, text="Password:", bg='#ADD8E6', font=("Arial", 14)).grid(row=2, column=0)
    self.new_password_entry = tk.Entry(self.frame, show="*", font=("Arial", 14))
    self.new_password_entry.grid(row=2, column=1)

    role_label = tk.Label(self.frame, text="Role (admin/customer):", bg='#ADD8E6', font=("Arial", 14))
    role_label.grid(row=3, column=0)
    self.role_var = tk.StringVar(value="customer")
    role_option = tk.OptionMenu(self.frame, self.role_var, "admin", "customer")
    role_option.grid(row=3, column=1)

    signup_btn = tk.Button(self.frame, text="Sign Up", font=("Arial", 14), command=self.create_user,
bg="#4CAF50", fg="white")
    signup_btn.grid(row=4, column=0, columnspan=2, pady=20)

def create_user(self):

    username = self.new_username_entry.get()
    password = self.new_password_entry.get()

```

```

role = self.role_var.get()

try:
    cursor.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)", (username,
password, role))
    conn.commit()
    messagebox.showinfo("Success", "Account created successfully")
    self.login_screen()
except sqlite3.IntegrityError:
    messagebox.showerror("Error", "Username already exists")

def admin_dashboard(self):
    for widget in self.frame.winfo_children():
        widget.destroy()

    tk.Label(self.frame, text="Admin Dashboard", font=("Arial", 20, 'bold'), bg='#ADD8E6',
fg="white").grid(row=0, column=0, columnspan=2)

    # Admin functionalities buttons

    add_bill_btn = tk.Button(self.frame, text="Add Bill", font=("Arial", 14), command=self.add_bill_screen,
bg="#2196F3", fg="white")
    add_bill_btn.grid(row=1, column=0, pady=10)

    update_bill_btn = tk.Button(self.frame, text="Update Bill", font=("Arial", 14),
command=self.update_bill_screen, bg="#FF9800", fg="white")
    update_bill_btn.grid(row=2, column=0, pady=10)

    delete_bill_btn = tk.Button(self.frame, text="Delete Bill", font=("Arial", 14),
command=self.delete_bill_screen, bg="#F44336", fg="white")
    delete_bill_btn.grid(row=3, column=0, pady=10)

    logout_btn = tk.Button(self.frame, text="Logout", font=("Arial", 14), command=self.login_screen,
bg="#FF5722", fg="white")
    logout_btn.grid(row=4, column=0, columnspan=2, pady=10)

```



```
def add_bill_screen(self):

    for widget in self.frame.winfo_children():

        widget.destroy()

    tk.Label(self.frame, text="Add Bill", font=("Arial", 20, 'bold'), bg='#ADD8E6', fg="white").grid(row=0,
column=0, columnspan=2)

    tk.Label(self.frame, text="Customer ID:", bg='#ADD8E6', font=("Arial", 14)).grid(row=1, column=0)
    customer_id_entry = tk.Entry(self.frame, font=("Arial", 14))
    customer_id_entry.grid(row=1, column=1)

    tk.Label(self.frame, text="Usage (units):", bg='#ADD8E6', font=("Arial", 14)).grid(row=2, column=0)
    usage_entry = tk.Entry(self.frame, font=("Arial", 14))
    usage_entry.grid(row=2, column=1)

    tk.Label(self.frame, text="Bill Amount:", bg='#ADD8E6', font=("Arial", 14)).grid(row=3, column=0)
    bill_amount_entry = tk.Entry(self.frame, font=("Arial", 14))
    bill_amount_entry.grid(row=3, column=1)

    tk.Label(self.frame, text="Status (paid/unpaid):", bg='#ADD8E6', font=("Arial", 14)).grid(row=4,
column=0)
    status_entry = tk.Entry(self.frame, font=("Arial", 14))
    status_entry.grid(row=4, column=1)

    add_btn = tk.Button(self.frame, text="Add Bill", font=("Arial", 14), command=lambda:
self.add_bill(customer_id_entry, usage_entry, bill_amount_entry, status_entry), bg="#4CAF50", fg="white")
    add_btn.grid(row=5, column=0, columnspan=2, pady=10)

def add_bill(self, customer_id_entry, usage_entry, bill_amount_entry, status_entry):

    customer_id = customer_id_entry.get()

    usage = usage_entry.get()

    bill_amount = bill_amount_entry.get()

    status = status_entry.get()
```

```

        cursor.execute("INSERT INTO bills (customer_id, usage, bill_amount, status) VALUES (?, ?, ?, ?)",
(customer_id, usage, bill_amount, status))

        conn.commit()

        messagebox.showinfo("Success", "Bill added successfully")

        self.admin_dashboard()

def update_bill_screen(self):

    for widget in self.frame.winfo_children():

        widget.destroy()

        tk.Label(self.frame, text="Update Bill", font=("Arial", 20, 'bold'), bg='#ADD8E6', fg="white").grid(row=0,
column=0, columnspan=2)

        tk.Label(self.frame, text="Bill ID:", bg='#ADD8E6', font=("Arial", 14)).grid(row=1, column=0)
        self.bill_id_entry = tk.Entry(self.frame, font=("Arial", 14))
        self.bill_id_entry.grid(row=1, column=1)

        tk.Label(self.frame, text="Customer ID:", bg='#ADD8E6', font=("Arial", 14)).grid(row=2, column=0)
        self.customer_id_entry = tk.Entry(self.frame, font=("Arial", 14))
        self.customer_id_entry.grid(row=2, column=1)

        tk.Label(self.frame, text="Usage (units):", bg='#ADD8E6', font=("Arial", 14)).grid(row=3, column=0)
        self.usage_entry = tk.Entry(self.frame, font=("Arial", 14))
        self.usage_entry.grid(row=3, column=1)

        tk.Label(self.frame, text="Bill Amount:", bg='#ADD8E6', font=("Arial", 14)).grid(row=4, column=0)
        self.bill_amount_entry = tk.Entry(self.frame, font=("Arial", 14))
        self.bill_amount_entry.grid(row=4, column=1)

        tk.Label(self.frame, text="Status (paid/unpaid):", bg='#ADD8E6', font=("Arial", 14)).grid(row=5,
column=0)
        self.status_entry = tk.Entry(self.frame, font=("Arial", 14))
        self.status_entry.grid(row=5, column=1)

```

```

        update_btn = tk.Button(self.frame, text="Update Bill", font=("Arial", 14), command=self.update_bill,
                                bg="#FF9800", fg="white")

        update_btn.grid(row=6, column=0, columnspan=2, pady=10)

    def update_bill(self):
        bill_id = self.bill_id_entry.get()
        customer_id = self.customer_id_entry.get()
        usage = self.usage_entry.get()
        bill_amount = self.bill_amount_entry.get()
        status = self.status_entry.get()

        cursor.execute("""
            UPDATE bills SET usage=?, bill_amount=?, status=? WHERE bill_id=? AND customer_id=?
            """, (usage, bill_amount, status, bill_id, customer_id))

        conn.commit()

        messagebox.showinfo("Success", "Bill updated successfully")

        self.admin_dashboard()

    def delete_bill_screen(self):

        for widget in self.frame.winfo_children():
            widget.destroy()

        tk.Label(self.frame, text="Delete Bill", font=("Arial", 20, 'bold'), bg='#ADD8E6', fg="white").grid(row=0,
                                                                 column=0, columnspan=2)

        tk.Label(self.frame, text="Bill ID:", bg='#ADD8E6', font=("Arial", 14)).grid(row=1, column=0)
        self.delete_bill_id_entry = tk.Entry(self.frame, font=("Arial", 14))
        self.delete_bill_id_entry.grid(row=1, column=1)

        tk.Label(self.frame, text="Customer ID:", bg='#ADD8E6', font=("Arial", 14)).grid(row=2, column=0)
        self.delete_customer_id_entry = tk.Entry(self.frame, font=("Arial", 14))
        self.delete_customer_id_entry.grid(row=2, column=1)

```

```
delete_btn = tk.Button(self.frame, text="Delete Bill", font=("Arial", 14), command=self.delete_bill,
bg="#F44336", fg="white")
```

```
delete_btn.grid(row=3, column=0, columnspan=2, pady=10)
```

```
def delete_bill(self):
```

```
    bill_id = self.delete_bill_id_entry.get()
```

```
    customer_id = self.delete_customer_id_entry.get()
```

```
    cursor.execute("DELETE FROM bills WHERE bill_id=? AND customer_id=?", (bill_id, customer_id))
```

```
    conn.commit()
```

```
    messagebox.showinfo("Success", "Bill deleted successfully")
```

```
    self.admin_dashboard()
```

```
def customer_dashboard(self, username):
```

```
    for widget in self.frame.winfo_children():
```

```
        widget.destroy()
```

```
    tk.Label(self.frame, text="Customer Dashboard", font=("Arial", 20, 'bold'), bg='#ADD8E6',
fg="white").grid(row=0, column=0, columnspan=2)
```

```
    cursor.execute("SELECT id FROM users WHERE username=?", (username,))
```

```
    user = cursor.fetchone()
```

```
    if user:
```

```
        customer_id = user[0]
```

```
        cursor.execute("SELECT * FROM bills WHERE customer_id=?", (customer_id,))
```

```
        bills = cursor.fetchall()
```

```
    if bills:
```

```
        row = 1
```

```
        for bill in bills:
```

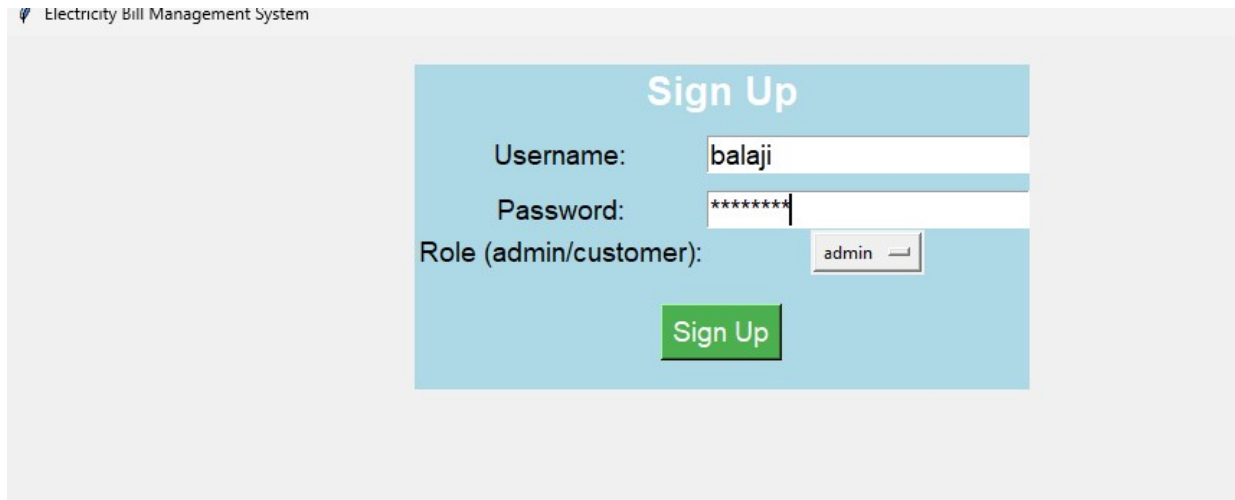
```
            bill_id, usage, bill_amount, status = bill[0], bill[2], bill[3], bill[4]
```

```
        tk.Label(self.frame, text=f"Bill ID: {bill_id}, Usage: {usage} units, Amount: ${bill_amount},  
        Status: {status}",  
                  bg='#ADD8E6', font=("Arial", 14)).grid(row=row, column=0, columnspan=2, pady=5)  
        row += 1  
    else:  
        tk.Label(self.frame, text="No bills found.", bg='#ADD8E6', font=("Arial", 14)).grid(row=1,  
        column=0, columnspan=2)  
  
        logout_btn = tk.Button(self.frame, text="Logout", font=("Arial", 14), command=self.login_screen,  
        bg="#FF5722", fg="white")  
        logout_btn.grid(row=row+1, column=0, columnspan=2, pady=10)  
  
root = tk.Tk()  
app = ElectricityBillApp(root)  
root.mainloop()
```

## CHAPTER - 5

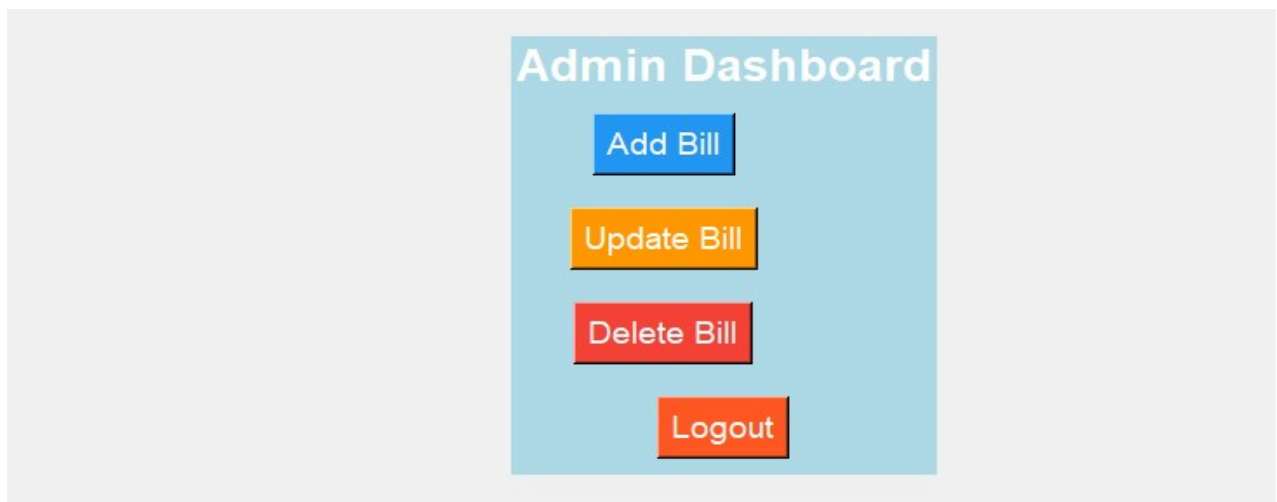
### SCREEN SHOTS

Fig 5.1 Admin signup and signin



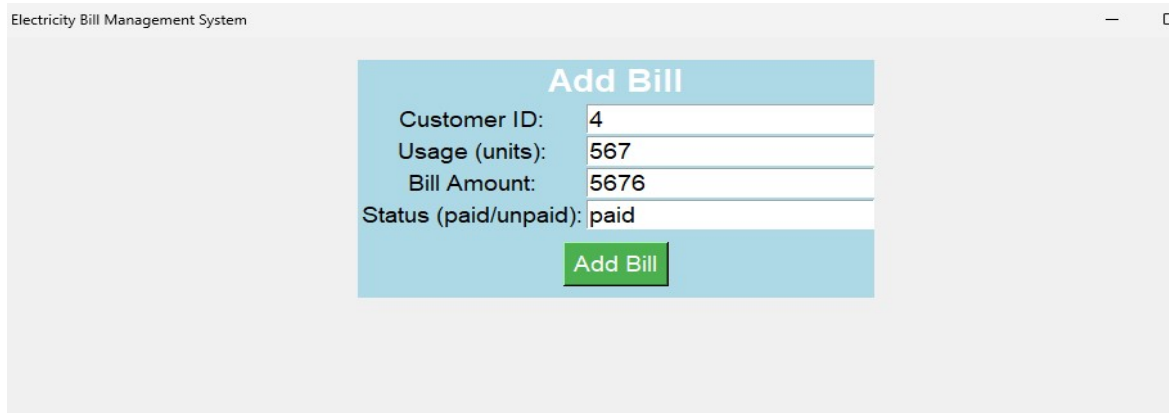
The screenshot shows the 'Sign Up' form within the 'Electricity Bill Management System'. The form is titled 'Sign Up' and contains three input fields: 'Username' with the value 'balaji', 'Password' with masked characters '\*\*\*\*\*', and 'Role (admin/customer)' with a dropdown menu currently showing 'admin'. A green 'Sign Up' button is positioned below the form fields.

Fig 5.2 Admin Dashboard



The screenshot displays the 'Admin Dashboard' with four action buttons arranged vertically: 'Add Bill' (blue), 'Update Bill' (orange), 'Delete Bill' (red), and 'Logout' (orange).

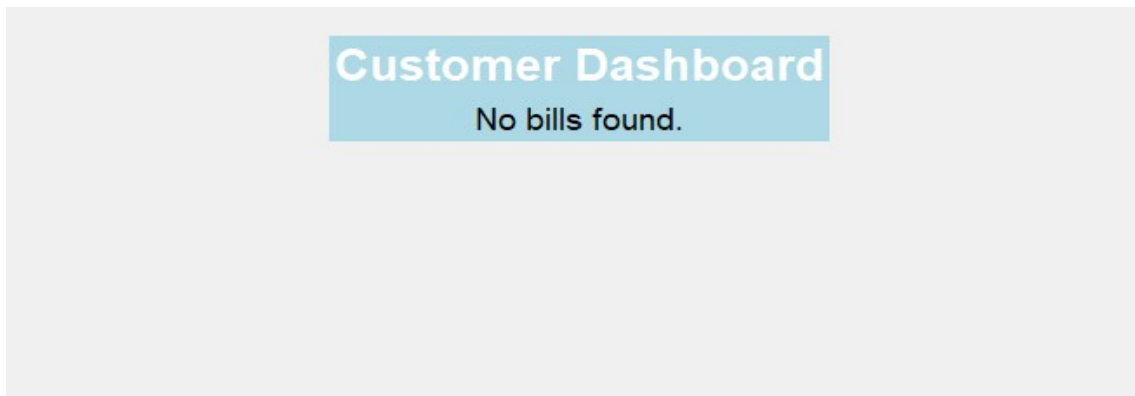
Fig 5.3 Add bill



The screenshot shows a web application window titled "Electricity Bill Management System". Inside, there is a light blue box with the heading "Add Bill". Below the heading, there are four input fields with labels: "Customer ID:" containing the value "4", "Usage (units):" containing "567", "Bill Amount:" containing "5676", and "Status (paid/unpaid):" containing "paid". At the bottom of this box is a green button labeled "Add Bill".

Add Bill	
Customer ID:	4
Usage (units):	567
Bill Amount:	5676
Status (paid/unpaid):	paid
<button>Add Bill</button>	

Fig 5.4 Customer view bills



The screenshot shows a web application window with a light blue box containing the heading "Customer Dashboard" and the text "No bills found." below it.

Customer Dashboard
No bills found.

## CHAPTER 6

### CONCLUSION AND FUTURE ENHANCEMENT

The Electricity Bill Management System developed using Python and SQLite3 provides a reliable and efficient solution for automating the electricity billing process. Through its modular design, the system effectively addresses the challenges of manual billing, such as time consumption, errors, and limited data access. By streamlining billing calculations, user authentication, payment tracking, and reporting, it improves accuracy and enhances the user experience for both customers and administrators.

With this system, administrators can efficiently manage customer data, update rates, and monitor payments, while customers can easily access their consumption details, view billing history, and make timely payments. The integration of a secure database ensures data integrity and makes it easy to store and retrieve information, enhancing both security and transparency.

Overall, this system meets the project's aim of creating an automated and userfriendly platform for electricity bill management. It simplifies operations, reduces errors, and ultimately leads to better customer satisfaction. Future enhancements could include more advanced analytics, integration with online payment gateways, and mobile compatibility, making the system even more comprehensive and accessible.



## CHAPTER – 7

### REFERENCES

1. <https://www.w3schools.com/sql/>
2. <https://www.tutorialspoint.com/sqlite/index.htm>
3. <https://www.wikipedia.org/>
4. <https://www.learnpython.org/>
5. <https://www.codecademy.com/learn/learn-python>