Water Management System Automation

Scripting Languages – CPSC 6210

Spring 2024

Professor: Soon – Ok Park

Abdullah Shaik

Mohammed Shabuddin

Swetha Patibandla

**Abstract:** In urban areas, most households depend on their local City Corporation or Water Board for water supply, which is provided according to predetermined schedules. However, any disruptions in this schedule, whether caused by technical issues or unforeseen circumstances, can lead to confusion and inconvenience for the residents. Our project addresses this challenge by automating the water supply scheduling process. Through a centralized database, users can easily access information regarding their sector's water supply schedule. Moreover, in the event of delays or changes, local officers can update the database, ensuring accurate and timely information dissemination. Additionally, the system provides features such as water bill information and contact details of the sector's water supply contractor, offering a comprehensive solution to household water supply management.

**Introduction:** Municipalities bear significant responsibilities, including the management of water resources, a vital component of urban infrastructure. The efficacy of a city's water distribution system hinges on various factors, including the availability of natural resources and the efficiency of infrastructure. Our project aims to enhance the efficiency of water management by developing a centralized database system and automating billing processes for consumer convenience. Traditionally, residents must manually track water supply schedules and calculate the next supply date, often leading to confusion and inefficiency.

This project aims to develop a Water Supply Management System using Python for backend operations and Tkinter for the Frontend User Interface. The system facilitates efficient management of water supply schedules for different areas within a city, focusing on timely delivery and effective communication with residents. By providing real-time information on water supply schedules, our system enables residents to prepare accordingly, minimizing water wastage. Moreover, by consolidating billing processes into a single online platform, we streamline payment procedures, eliminating the complexities associated with traditional methods.

**Existing Systems:** Currently, changes in water supply schedules are communicated through newspapers or local news channels, requiring residents to stay updated through various media channels. However, this approach could be better, as individuals may miss important updates, leading to inconvenience. Similarly, while digital payment options exist, they often need help with usability issues, complicating the billing process for consumers.

Our project addresses these shortcomings by providing a centralized platform for accessing accurate water supply information and simplifying billing procedures. By leveraging technology, we aim to enhance the efficiency and convenience of water management for urban residents.

**Technologies Used:**

Python: Utilized as the primary programming language for database connectivity and backend operations.

SQLite3: Integrated with the inbuilt SQLite3 module of Python for database management.

Tkinter: Employed as the GUI toolkit to develop the frontend interface for the application.

**Project Structure:**

**Backend Logic**

The backend logic involves the creation, manipulation, and deletion of data within the database. Key functionalities include:

* Defining database schema and table structures.
* Adding, viewing, and deleting records.
* Updating water supply dates dynamically based on user-defined parameters.

**Frontend Design**

The frontend interface is designed using Tkinter, offering a user-friendly environment for interaction. Features include:

* Creation of GUI elements such as buttons, text boxes, and TreeView for database representation.
* Integration with backend functionalities for seamless data manipulation.
* Enhanced user experience through intuitive design and layout.

**Code Snippets**

**Backend Operations**

* **Creating Officer Table**: Defines the schema and structure for the 'Officer' relation in the database.
* **Adding, Viewing, and Deleting Records**: Functions to perform CRUD operations on the database.
* **Updating Water Supply Dates**: Automates the process of updating supply dates based on predefined rules.

**Frontend Interface**

* **GUI Construction**: Design and layout of GUI elements using Tkinter.
* **Button Actions**: Implementation of button functionalities such as adding new records and resetting data.
* **TreeView Display**: Representation of database records in a TreeView format for easy viewing and management.

TheProject.py: 1759 lines of code

backend.py: 309 lines of code

In the presentation layer, TheProject.py file includes the main function and several classes:

**Check Class:** Sets up the main window of the application, including the title, dimensions, background color, and a frame within the window. It also includes buttons for admin login, customer login, and officer login, along with functions to redirect to the respective login screens.

**Window1 Class:** Creates the admin login screen with user input fields for username and password, along with buttons for login, reset, and exit. It also includes functions to handle the login system, reset input fields, and exit the application. After a successful login, it redirects to different screens based on the user's role (customer, officer, admin).

**Customer Class:** Manages customer-related functionality, including adding, displaying, deleting, and updating customer records. It also includes a function to handle the selection of a customer record and contains frames and widgets for the customer interface.

**Officer Class:** Manages officer-related functionality, including adding, displaying, deleting, and updating officer records. It also includes frames and widgets for the officer interface.

**Bills Class:** Manages billing information, including adding, displaying, deleting, and updating billing records. It also includes frames and widgets to display billing records using a Treeview widget.

**Locality Class:** Manages locality-related information, including adding, displaying, deleting, and updating locality records. It also includes frames and widgets for the locality interface.

**Reservoir Class:** Manages reservoir-related information, including adding, displaying, deleting, and updating reservoir records. It also includes frames and widgets for the reservoir interface.

**Window2 Class:** Creates the customer login screen with functions to fetch and display customer details based on the provided ID.

**Window3 Class:** Creates the officer login screen with functions to add, display, and reset officer records.

Theproject.py

|  |
| --- |
| from tkinter import\*  import tkinter.messagebox  from tkinter import ttk  import random  import time  import datetime  import pymysql  import tempfile, os  import backend  import sqlite3  my\_conn = sqlite3.connect('backend.db')  # Function to start the application  def main():  root = Tk() # Create a Tkinter root window  app = Check(root) # Create an instance of the Check class  root.mainloop() # Start the Tkinter event loop  ####~~~~~~~~~~Home Window~~~~~~~~~~####  class Check:  def \_\_init\_\_(self,master):  self.master = master  self.master.title("Home Page") # Set the title of the window  self.master.geometry('1920x1080+0+0') # Set the dimensions and position of the window  self.master.rowconfigure(0, weight=1)  self.master.columnconfigure(0, weight=1)  self.master.config(bg = 'powder blue') # Set the background color of the window  self.frame = Frame(self.master, bg = 'powder blue') # Create a frame within the window  self.frame.pack()  ###~~~Home Window Title~~~###    self.lblTitle = Label(self.frame, text = 'Home Page', font=('arial',50,'bold'), bg='powder blue', fg='black') # Create a label for the title  self.lblTitle.grid(row=0,column=0,columnspan=2,pady=40) # Place the label in the frame  ###~~~Buttons~~~###  self.LoginFrame = LabelFrame(self.frame, width=1000, height=600, font=('arial',20,'bold'), relief='ridge', bg='cadet blue', bd=20) # Create a labeled frame  self.LoginFrame.grid(row=2, column=0) # Place the labeled frame in the frame  self.btnCustomer = Button(self.LoginFrame, text = 'Customer', width = 17,font=('arial',20,'bold'),command = self.CustomerLogin) # Create a button for customer login  self.btnCustomer.grid(row=1,column=1,pady=20,padx=8) # Place the button in the labeled frame  self.btnAdmin = Button(self.LoginFrame, text = 'Admin', width = 17,font=('arial',20,'bold'), command = self.AdminLogin) # Create a button for admin login  self.btnAdmin.grid(row=2,column=1,pady=20,padx=8) # Place the button in the labeled frame  self.btnOfficer = Button(self.LoginFrame, text = 'Officer', width = 17,font=('arial',20,'bold'), command = self.OfficerLogin) # Create a button for officer login  self.btnOfficer.grid(row=3,column=1,pady=20,padx=8) # Place the button in the labeled frame  ###~~~Admin Login Redirect~~~###  def AdminLogin(self):  self.newWindow = Toplevel(self.master) # Create a new top-level window  self.app = Window1(self.newWindow) # Create an instance of the Window1 class. i.e., Redirect to Admin's Login Page    ###~~~Customer Login Redirect~~~###  def CustomerLogin(self):  self.newWindow = Toplevel(self.master) # Create a new top-level window  self.app = Window2(self.newWindow) # Create an instance of the Window2 class. i.e., Redirect to Customer's Login Page  ###~~~Officer Login Redirect~~~###  def OfficerLogin(self):  self.newWindow = Toplevel(self.master) # Create a new top-level window  self.app = Window3(self.newWindow) # Create an instance of the Window3 class. i.e., Redirect to Officer's Login Page    #~~~creates Window1 for Admin Login~~~###  class Window1:  def \_\_init\_\_(self,master):  self.master = master  self.master.title("Admin Login System")  self.master.geometry('1920x1080+0+0')  self.master.rowconfigure(0, weight=1)  self.master.columnconfigure(0, weight=1)  self.master.config(bg = 'powder blue')  self.frame = Frame(self.master, bg = 'powder blue')  self.frame.pack()  ###~~~User Input~~~###  self.Username = StringVar() # Variable to store the username  self.Password = StringVar() # Variable to store the password    ###~~~Window1 Title~~~###  self.lblTitle = Label(self.frame, text = 'Admin Login Page', font=('arial',50,'bold'), bg='powder blue', fg='black')  self.lblTitle.grid(row=0,column=0,columnspan=2,pady=40)  ###~~~Window1 Frames~~~###  self.LoginFrame1 = LabelFrame(self.frame, width=1350, height=600, font=('arial',20,'bold'), relief='ridge', bg='cadet blue', bd=20)  self.LoginFrame1.grid(row=1, column=0)  self.LoginFrame2 = LabelFrame(self.frame, width=1000, height=600, font=('arial',20,'bold'), relief='ridge', bg='cadet blue', bd=20)  self.LoginFrame2.grid(row=2, column=0)  ###~~~Window1 Labels~~~###  self.lblUsername=Label(self.LoginFrame1, text = 'Username',font=('arial',20,'bold'),bd=22,bg='cadet blue',fg='Cornsilk')  self.lblUsername.grid(row=0,column=0)  self.txtUsername=Entry(self.LoginFrame1,font=('arial',20,'bold'),textvariable=self.Username)  self.txtUsername.grid(row=0,column=1,padx=119)  self.lblPassword=Label(self.LoginFrame1, text = 'Password',font=('arial',20,'bold'),bd=22,bg='cadet blue',fg='Cornsilk')  self.lblPassword.grid(row=1,column=0)  self.txtPassword=Entry(self.LoginFrame1,font=('arial',20,'bold'),show='\*',textvariable=self.Password)  self.txtPassword.grid(row=1,column=1,columnspan=2,pady=30)  ###~~~Window1 Buttons~~~###  self.btnLogin = Button(self.LoginFrame2, text = 'Login', width = 17,font=('arial',20,'bold'), command = self.Login\_System)  self.btnLogin.grid(row=3,column=0,pady=20,padx=8)  self.btnReset = Button(self.LoginFrame2, text = 'Reset', width = 17,font=('arial',20,'bold'), command = self.Reset)  self.btnReset.grid(row=3,column=1,pady=20,padx=8)  self.btnExit = Button(self.LoginFrame2, text = 'Exit', width = 17,font=('arial',20,'bold'), command = self.iExit)  self.btnExit.grid(row=3,column=2,pady=20,padx=8)  ###~~~Redirecting Buttons after succesful login~~~###  self.Loginframe3 = Frame(self.frame,width=1000,height=200,bd=20,relief='ridge',bg='cadet blue')  self.Loginframe3.grid(row=4,column=0,pady=2, padx = 20)    self.btnCustomer = Button(self.Loginframe3, text = 'Customer',state = DISABLED, width = 12,font=('arial',20,'bold'), command = self.Customer)  self.btnCustomer.grid(row = 0,column = 0)  self.btnOfficer = Button(self.Loginframe3, text = 'Officer',state = DISABLED, width = 12, font=('arial',20,'bold'), command = self.Officer)  self.btnOfficer.grid(row = 0,column = 1, pady = 0, padx = 20)  self.btnBills = Button(self.Loginframe3, text = 'Bills',state = DISABLED, width = 12, font=('arial',20,'bold'), command = self.Bills)  self.btnBills.grid(row = 0,column = 2, pady = 0, padx = 20)  self.btnLocality = Button(self.Loginframe3, text = 'Locality',state = DISABLED, width = 12, font=('arial',20,'bold'), command = self.Locality)  self.btnLocality.grid(row = 0,column = 3, pady = 0, padx = 20)  self.btnReservoir = Button(self.Loginframe3, text = 'Reservoir',state = DISABLED, width = 12, font=('arial',20,'bold'), command = self.Reservoir)  self.btnReservoir.grid(row = 0,column = 4, pady = 0, padx = 20)    ###~~~Redirecting Functions~~~###  def Customer(self):  self.newWindow = Toplevel(self.master)  self.app = Customer(self.newWindow)  def Officer(self):  self.newWindow = Toplevel(self.master)  self.app = Officer(self.newWindow)  def Bills(self):  self.newWindow = Toplevel(self.master)  self.app = Bills(self.newWindow)  def Locality(self):  self.newWindow = Toplevel(self.master)  self.app = Locality(self.newWindow)  def Reservoir(self):  self.newWindow = Toplevel(self.master)  self.app = Reservoir(self.newWindow)    ###~~~Button Functions~~~###  def Login\_System(self):  u=(self.Username.get()) # Get the value of the username entered  p=(self.Password.get()) # Get the value of the password entered  if(u==str('Kevin')and p==str('Owens')): #The Username is set to : "Kevin" and the Password is set to : "Owens"  # If the credentials are correct, enable the below given buttons  self.btnCustomer.config(state = NORMAL)  self.btnOfficer.config(state = NORMAL)  self.btnBills.config(state = NORMAL)  self.btnLocality.config(state = NORMAL)  self.btnReservoir.config(state = NORMAL)  else:  tkinter.messagebox.askyesno("Error","The entered details are wrong") # Show an error message box if wrong credentials are entered  #Disable the below given buttons  self.btnUser.config(state = DISABLED)  self.btnOfficer.config(state = DISABLED)  self.Username.set("") # Clear the username entry field  self.Password.set("") # Clear the password entry field  self.txtUsername.focus() # Set the focus back to the username entry field  def Reset(self):  self.Username.set("") # Clear the value of the username  self.Password.set("") # Clear the value of the password  def iExit(self):  self.iExit = tkinter.messagebox.askyesno("Exit", "Confirm if you want to exit")  if self.iExit > 0: # If the user confirms the exit  self.master.destroy() # Destroy the main window and exit the application  else:  command=self.new\_window # Store the command to open a new window  return # Return from the function without taking any further action  ###~~~CUSTOMER Class~~~###  class Customer:  ###~~~CUSTOMER DB~~~###  def \_\_init\_\_(self, root):  # Initialize the Customer class with the given root window  self.root = root  blank\_space = " "  self.root.title(200 \* blank\_space + "Customer DB")  self.root.geometry("1920x1080+0+0")  self.root.rowconfigure(0, weight=1)  self.root.columnconfigure(0, weight=1)  id = StringVar() # Variable to store the customer ID  Name = StringVar() # Variable to store the customer name  Address = StringVar() # Variable to store the customer address  sector\_no = StringVar() # Variable to store the sector number  officer\_id = StringVar() # Variable to store the officer ID  reservoir\_id = StringVar() # Variable to store the reservoir ID  no\_of\_connection = StringVar() # Variable to store the number of connections  ###~~~CUSTOMER Functions~~~###  def iExit():  # Function to handle the exit button  iExit = tkinter.messagebox.askyesno("Exit", "Confirm if you want to exit")  if iExit>0:  root.destroy()  return  def iReset():  # Function to handle the reset button  self.txtid.delete(0, END) # Clear the customer ID entry field  self.txtName.delete(0, END) # Clear the customer name entry field  self.txtAddress.delete(0, END) # Clear the customer address entry field  self.cbosector\_no.current(0) # Reset the sector number dropdown to the default option  self.txtofficer\_id.delete(0, END) # Clear the officer ID entry field  self.txtreservoir\_id.delete(0, END) # Clear the reservoir ID entry field  self.txtno\_of\_connection.delete(0, END) # Clear the number of connections entry field    def addData():  # Function to add data to the customer database  if id.get() == "" or Name.get() == "" or Address.get() == "" or sector\_no.get() == "" or officer\_id.get() == "" or reservoir\_id.get() == "" or no\_of\_connection.get() == "":  tkinter.messagebox.askyesno("Error", "Please enter the correct Data")  else:  backend.addCustomer(  id.get(),  Name.get(),  Address.get(),  sector\_no.get(),  officer\_id.get(),  reservoir\_id.get(),  no\_of\_connection.get()  )  displayData()  super(self.customerlist, self).delete()  self.customerlist.insert(END,  (  id.get(),  Name.get(),  Address.get(),  sector\_no.get(),  officer\_id.get(),  reservoir\_id.get(),  no\_of\_connection.get()  ))    def displayData():  # Function to display data from the customer database  result = backend.viewCustomer()  if len(result)!=0:  self.customerlist.delete(\*self.customerlist.get\_children())  for row in result:  self.customerlist.insert('', END, values = row)  def deleteData():  # Function to delete data from the customer database  if(len(id.get())!= 0):  backend.delCustomer(sd[0])  iReset()  displayData()  tkinter.messagebox.showinfo("Delete", "Record successfully deleted")  def update():  # Function to update data in the customer database  if(len(id.get()) != 0):  backend.delCustomer(sd[0])  if(len(id.get()) != 0):  backend.addCustomer(id.get(), Name.get(), Address.get(), sector\_no.get(), officer\_id.get(), reservoir\_id.get(), no\_of\_connection.get())  displayData()  def customerREC(event):  # Function to handle selection of a customer record  global sd  iReset()  viewInfo = self.customerlist.focus()  learnerData = self.customerlist.item(viewInfo)  sd = learnerData['values']  self.txtid.insert(END,sd[0])  self.txtName.insert(END,sd[1])  self.txtAddress.insert(END,sd[2])  sector\_no.set(sd[3])  self.txtofficer\_id.insert(END,sd[4])  self.txtreservoir\_id.insert(END,sd[5])  self.txtno\_of\_connection.insert(END,sd[6])    ###~~~CUSTOMER Frames~~~####  # Creating the main frame for the customer interface  MainFrame = Frame(self.root, bd = 10, width = 1350, height = 700, relief = RIDGE, bg = "cadet blue")  MainFrame.grid()  # Frame for the buttons  ButtonFrame = Frame(MainFrame, bd = 5, width = 1340, height = 100, relief = RIDGE)  ButtonFrame.grid(row = 2, column = 0, pady = 8)  # Frame for the title  TitleFrame = Frame(MainFrame, bd = 7, width = 1340, height = 100, relief = RIDGE)  TitleFrame.grid(row = 0, column = 0)  # Frame for the top section  TopFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame.grid(row = 1, column = 0)  # Left frame within the top section  LeftFrame = Frame(TopFrame, bd = 5, width = 1340, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  LeftFrame.pack(side = LEFT)  # Frame for the widgets within the left frame  WidgetFrame = Frame(LeftFrame, bd = 5, width = 300, height = 180, padx = 2, pady = 4, relief = RIDGE)  WidgetFrame.pack(side = TOP, padx = 0, pady = 4)  # Right frame within the top section  RightFrame = Frame(TopFrame, bd = 5, width = 320, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  RightFrame.pack(side = RIGHT)  # Frame for the treeview within the right frame  TreeViewFrame = Frame(RightFrame, bd = 5, width = 310, height = 200, padx = 2, pady = 2, relief = RIDGE)  TreeViewFrame.pack(side = TOP)  ###~~~CUSTOMER Title~~~###  # Label for the title  self.lblTitle = Label(TitleFrame, font = ('arial', 56, 'bold'), text='Customer Database', bd = 7)  self.lblTitle.grid(row = 0, column = 0, padx =132)  ###~~~CUSTOMER Buttons~~~###  # Button for adding new data  self.btnAddNew = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Insert New" ,padx = 24, width = 8, height = 1, command = addData).grid(row = 0, column = 0, padx = 1)    # Button for displaying data  self.btnDisplay = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Display" ,padx = 24, width = 8, height = 1, command = displayData).grid(row = 0, column = 1, padx = 1)    # Button for deleting data  self.btnDelete = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Delete" ,padx = 24, width = 8, height = 1, command = deleteData).grid(row = 0, column = 2, padx = 1)    # Button for updating data  self.btnUpdate = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Update" ,padx = 24, width = 8, height = 1, command = update).grid(row = 0, column = 3, padx = 1)    # Button for resetting data  self.btnReset = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Reset" ,padx = 24, width = 8, height = 1, command = iReset).grid(row = 0, column = 4, padx = 1)    # Button for exiting the application  self.btnExit = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Exit" ,padx = 24, width = 8, height = 1, command = iExit).grid(row = 0, column = 5, padx = 1)  ###~~~CUSTOMER Widgets~~~###  # Creating label and grid for Customer ID  self.lblid = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Customer ID ', bd = 7, anchor='w', justify=LEFT)  self.lblid.grid(row=0,column=0,sticky =W,padx=5)  self.txtid = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = id)  self.txtid.grid(row=0, column=1)  # Creating label and grid for Customer Name  self.lblName = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Customer Name ', bd = 7, anchor='w', justify=LEFT)  self.lblName.grid(row=1,column=0,sticky =W,padx=5)  self.txtName = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Name)  self.txtName.grid(row=1, column=1)  # Creating label and grid for Customer Address  self.lblAddress = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Address ', bd = 7, anchor='w', justify=LEFT)  self.lblAddress.grid(row=2,column=0,sticky =W,padx=5)  self.txtAddress = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Address)  self.txtAddress.grid(row=2, column=1)  # Creating label and grid for Sector Number  self.lblsector\_no = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Sector No ', bd = 7, anchor='w', justify=LEFT)  self.lblsector\_no.grid(row=3,column=0,sticky =W,padx=5)  self.cbosector\_no = ttk.Combobox(WidgetFrame, width = 39, font = ('arial', 12, 'bold'), state = 'readonly', textvariable = sector\_no)  self.cbosector\_no['values'] = ('','1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13')  self.cbosector\_no.current(0)  self.cbosector\_no.grid(row = 3, column = 1)  # Creating label and grid for Officer ID  self.lblofficer\_id = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Officer ID', bd = 7, anchor='w', justify=LEFT)  self.lblofficer\_id.grid(row=4,column=0,sticky =W,padx=5)  self.txtofficer\_id = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = officer\_id)  self.txtofficer\_id.grid(row=4, column=1)  # Creating label and grid for Reservoir ID  self.lblreservoir\_id = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Reservoir ID', bd = 7, anchor='w', justify=LEFT)  self.lblreservoir\_id.grid(row=5,column=0,sticky =W,padx=5)  self.txtreservoir\_id = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = reservoir\_id)  self.txtreservoir\_id.grid(row=5, column=1)  # Creating label and grid for Number of Connections  self.lblno\_of\_connection = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'No. of connections', bd = 7, anchor='w', justify=LEFT)  self.lblno\_of\_connection.grid(row=6,column=0,sticky =W,padx=5)  self.txtno\_of\_connection = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = no\_of\_connection)  self.txtno\_of\_connection.grid(row=6, column=1)  ###~~~CUSTOMER Treeview~~~###  # Creating horizontal and vertical scrollbars for the treeview  scroll\_x = Scrollbar(TreeViewFrame, orient = HORIZONTAL)  scroll\_y = Scrollbar(TreeViewFrame, orient = VERTICAL)  # Creating the customerlist treeview with specified height, columns, and scrollbars  self.customerlist = ttk.Treeview(TreeViewFrame, height = 12, columns = ("id", "Name", "Address", "sector\_no", "officer\_id", "reservoir\_id", "no\_of\_connection"), xscrollcommand = scroll\_x.set,yscrollcommand = scroll\_y.set)  # Attaching the scrollbars to the treeview  scroll\_x.pack(side = BOTTOM, fill = X)  scroll\_y.pack(side = BOTTOM, fill = Y)  # Configuring the column headings of the treeview  self.customerlist.heading("id", text = "Customer ID")  self.customerlist.heading("Name", text = "Customer Name")  self.customerlist.heading("Address", text = "Customer Address")  self.customerlist.heading("sector\_no", text = "Sector No")  self.customerlist.heading("officer\_id", text = "Officer ID")  self.customerlist.heading("reservoir\_id", text = "Reservoir ID")  self.customerlist.heading("no\_of\_connection", text = "No. of conns.")  # Displaying only the column headings in the treeview  self.customerlist['show'] = 'headings'  # Configuring the width of each column in the treeview  self.customerlist.column("id", width = 90)  self.customerlist.column("Name", width = 130)  self.customerlist.column("Address", width = 130)  self.customerlist.column("sector\_no", width = 90)  self.customerlist.column("officer\_id", width = 90)  self.customerlist.column("reservoir\_id", width = 90)  self.customerlist.column("no\_of\_connection", width = 90)  # Packing the treeview to fill the available space and expand  self.customerlist.pack(fill = BOTH, expand = 1)  # Binding the customerREC function to the "<ButtonRelease-1>" event of the treeview  self.customerlist.bind("<ButtonRelease-1>", customerREC)  # Calling the displayData function to populate the treeview with data  displayData()  ###~~~OFFICER Class~~~###  class Officer:    ###~~~OFFICER DB~~~###  def \_\_init\_\_(self, root):  # Initialize the Officer class with the given root (main window) object  self.root = root  blank\_space = " "  self.root.title(200 \* blank\_space + "Officer DB") # Set the title of the main window  self.root.geometry("1920x1080+0+0") # Set the size and position of the main window  self.root.rowconfigure(0, weight=1)  self.root.columnconfigure(0, weight=1)  id = StringVar()  Name = StringVar()  sector\_no = StringVar()    ###~~~OFFICER Functions~~~###  def iExit():  # Function to handle the exit action when the user confirms to exit  iExit = tkinter.messagebox.askyesno("Exit", "Confirm if you want to exit")  if iExit>0:  root.destroy()  return  def iReset():  # Function to reset the input fields  self.txtid.delete(0, END)  self.txtName.delete(0, END)  self.cbosector\_no.current(0)  def addData():  # Function to reset the input fields  if id.get() == "" or Name.get() == "" or sector\_no.get() == "":  tkinter.messagebox.askyesno("Error", "Please enter the correct Data")  else:  backend.addOfficer(  id.get(),  Name.get(),  sector\_no.get()  )    displayData()    super(self.officerlist, self).delete()  self.officerlist.insert(END,  (  id.get(),  Name.get(),  sector\_no.get()  ))  def displayData():  # Function to display data in the officerlist  result = backend.viewOfficer()  if len(result)!=0:  self.officerlist.delete(\*self.officerlist.get\_children())  for row in result:  self.officerlist.insert('', END, values = row)  def deleteData():  # Function to delete selected data from the backend and update the display  if(len(id.get())!= 0):  backend.delOfficer(sd[0])  iReset()  displayData()  tkinter.messagebox.showinfo("Delete", "Record successfully deleted")  def update():  # Function to update data in the backend and update the display  if(len(id.get()) != 0):  backend.delOfficer(sd[0])  if(len(id.get()) != 0):  backend.addOfficer(id.get(), Name.get(), sector\_no.get())  displayData()  def officerREC(event):  # Function to handle the selection of an officer record in the officerlist  global sd  iReset() # Reset the input fields  viewInfo = self.officerlist.focus() # Get the focused item in the officerlist  learnerData = self.officerlist.item(viewInfo) # Retrieve the data of the selected item  sd = learnerData['values'] # Extract the values of the selected item and store them in sd  self.txtid.insert(END,sd[0]) # Insert the Officer ID value into the txtid Entry field  self.txtName.insert(END,sd[1]) # Insert the Officer Name value into the txtName Entry field  sector\_no.set(sd[2]) # Set the value of the sector\_no Combobox to the Sector No. value  ###~~~OFFICER Frames~~~###  # Create and configure various frames within the main window  MainFrame = Frame(self.root, bd = 10, width = 1350, height = 700, relief = RIDGE, bg = "cadet blue")  MainFrame.grid()  ButtonFrame = Frame(MainFrame, bd = 5, width = 1340, height = 100, relief = RIDGE)  ButtonFrame.grid(row = 2, column = 0, pady = 8)  TitleFrame = Frame(MainFrame, bd = 7, width = 1340, height = 100, relief = RIDGE)  TitleFrame.grid(row = 0, column = 0)  TopFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame.grid(row = 1, column = 0)  LeftFrame = Frame(TopFrame, bd = 5, width = 1340, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  LeftFrame.pack(side = LEFT)  WidgetFrame = Frame(LeftFrame, bd = 5, width = 300, height = 180, padx = 2, pady = 4, relief = RIDGE)  WidgetFrame.pack(side = TOP, padx = 0, pady = 4)  RightFrame = Frame(TopFrame, bd = 5, width = 320, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  RightFrame.pack(side = RIGHT)  TreeViewFrame = Frame(RightFrame, bd = 5, width = 310, height = 200, padx = 2, pady = 2, relief = RIDGE)  TreeViewFrame.pack(side = TOP)  ###~~~OFFICER Title~~~###  # Create and configure the title label for the Officer DB  self.lblTitle = Label(TitleFrame, font = ('arial', 56, 'bold'), text='Officer DB', bd = 7)  self.lblTitle.grid(row = 0, column = 0, padx =132)  ###~~~OFFICER Widgets~~~###  # Create and configure labels, entry fields, and combobox for Officer ID, Officer Name, and Sector No.  self.lblid = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Officer ID ', bd = 7, anchor='w', justify=LEFT)  self.lblid.grid(row=0,column=0,sticky =W,padx=5)  self.txtid = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = id)  self.txtid.grid(row=0, column=1)  self.lblName = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Officer Name ', bd = 7, anchor='w', justify=LEFT)  self.lblName.grid(row=1,column=0,sticky =W,padx=5)  self.txtName = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Name)  self.txtName.grid(row=1, column=1)  self.lblsector\_no = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Sector No.', bd = 7, anchor='w', justify=LEFT)  self.lblsector\_no.grid(row=2,column=0,sticky =W,padx=5)  self.cbosector\_no = ttk.Combobox(WidgetFrame, width = 39, font = ('arial', 12, 'bold'), state = 'readonly', textvariable = sector\_no)  self.cbosector\_no['values'] = ('','1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13')  self.cbosector\_no.current(0)  self.cbosector\_no.grid(row = 2, column = 1)  ###~~~OFFICER Treeview~~~###  # Create horizontal and vertical scrollbars for the treeview  scroll\_x = Scrollbar(TreeViewFrame, orient = HORIZONTAL)  scroll\_y = Scrollbar(TreeViewFrame, orient = VERTICAL)  # Create the treeview widget with specified columns and scrollbars  self.officerlist = ttk.Treeview(TreeViewFrame, height = 12, columns = ("id", "Name", "sector\_no"), xscrollcommand = scroll\_x.set,yscrollcommand = scroll\_y.set)  # Set the column headings for the treeview  scroll\_x.pack(side = BOTTOM, fill = X)  scroll\_y.pack(side = BOTTOM, fill = Y)  # Set the width of each column in the treeview  self.officerlist.heading("id", text = "Officer ID")  self.officerlist.heading("Name", text = "Officer Name")  self.officerlist.heading("sector\_no", text = "Sector No")  self.officerlist['show'] = 'headings'  self.officerlist.column("id", width = 70)  self.officerlist.column("Name", width = 150)  self.officerlist.column("sector\_no", width = 70)  self.officerlist.pack(fill = BOTH, expand = 1)  # Bind the click event to the officerREC function  self.officerlist.bind("<ButtonRelease-1>", officerREC)  # Display the data in the treeview  displayData()      ###~~~OFFICER Buttons~~~###  # Create and configure the insert new button  self.btnAddNew = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Insert New" ,padx = 24, width = 8, height = 1, command = addData).grid(row = 0, column = 0, padx = 1)    # Create and configure the display button  self.btnDisplay = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Display" ,padx = 24, width = 8, height = 1, command = displayData).grid(row = 0, column = 1, padx = 1)    # Create and configure the delete button  self.btnDelete = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Delete" ,padx = 24, width = 8, height = 1, command = deleteData).grid(row = 0, column = 2, padx = 1)    # Create and configure the update button  self.btnUpdate = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Update" ,padx = 24, width = 8, height = 1, command = update).grid(row = 0, column = 3, padx = 1)    # Create and configure the reset button  self.btnReset = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Reset" ,padx = 24, width = 8, height = 1, command = iReset).grid(row = 0, column = 4, padx = 1)    # Create and configure the exit button  self.btnExit = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Exit" ,padx = 24, width = 8, height = 1, command = iExit).grid(row = 0, column = 5, padx = 1)  ###~~~BILLS Class~~~###  class Bills:    ###~~~BILLS DB~~~###  def \_\_init\_\_(self, root):  # Initialize the Bills class with the root window  self.root = root  blank\_space = " "  self.root.title(200 \* blank\_space + "Billing DB")  self.root.geometry("1920x1080+0+0")  self.root.rowconfigure(0, weight=1)  self.root.columnconfigure(0, weight=1)  # Define StringVars for storing user inputs  customer\_id = StringVar()  id = StringVar()  Payments\_Due = StringVar()  due\_Date = StringVar()    ###~~~BILLS Functions~~~###  # Function to exit the program  def iExit():  iExit = tkinter.messagebox.askyesno("Exit", "Confirm if you want to exit")  if iExit>0:  root.destroy()  return  # Function to reset the input fields  def iReset():  self.txtid.delete(0, END)  self.txtcustomer\_id.delete(0, END)  self.txtPayments\_Due.delete(0, END)  self.txtdue\_Date.delete(0, END)  # Function to add data to the database  def addData():  if id.get() == "" or customer\_id.get() == "" or Payments\_Due.get() == "" or due\_Date.get == "":  tkinter.messagebox.askyesno("Error","Please enter the correct Data")  else:  backend.addBill(  id.get(),  customer\_id.get(),  Payments\_Due.get(),  due\_Date.get(),  )  displayData()  super(self.billinglist, self).delete()  self.billinglist.insert(END,  (  id.get(),  customer\_id.get(),  Payments\_Due.get(),  due\_Date.get(),  ))  # Function to display data from the database  def displayData():  result = backend.viewBill()  if len(result)!=0:  self.billinglist.delete(\*self.billinglist.get\_children())  for row in result:  self.billinglist.insert('', END, values = row)  # Function to delete data from the database  def deleteData():  if(len(id.get())!= 0):  backend.delBill(sd[0])  iReset()  displayData()  tkinter.messagebox.showinfo("Delete", "Record successfully deleted")  # Function to update data in the database  def update():  if(len(id.get()) != 0):  backend.delBill(sd[0])  if(len(id.get()) != 0):  backend.addBill(id.get(), customer\_id.get(), Payments\_Due.get(), due\_Date.get())  displayData()    # Function to handle the selection of a record in the billinglist  def billingREC(event):  global sd  iReset() # Reset the input fields  viewInfo = self.billinglist.focus() # Get the focused item in the billinglist  learnerData = self.billinglist.item(viewInfo) # Get the data of the focused item  sd = learnerData['values'] # Extract the values of the focused item  # Insert the values into the respective input fields  self.txtid.insert(END,sd[0])  self.txtcustomer\_id.insert(END,sd[1])  self.txtPayments\_Due.insert(END,sd[2])  self.txtdue\_Date.insert(END,sd[3])  ###~~~BILLS Frames~~~###  # Create and configure various frames within the main window  MainFrame = Frame(self.root, bd = 10, width = 1350, height = 700, relief = RIDGE, bg = "cadet blue")  MainFrame.grid()  ButtonFrame = Frame(MainFrame, bd = 5, width = 1340, height = 100, relief = RIDGE)  ButtonFrame.grid(row = 2, column = 0, pady = 8)  TitleFrame = Frame(MainFrame, bd = 7, width = 1340, height = 100, relief = RIDGE)  TitleFrame.grid(row = 0, column = 0)  TopFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame.grid(row = 1, column = 0)  LeftFrame = Frame(TopFrame, bd = 5, width = 1340, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  LeftFrame.pack(side = LEFT)  WidgetFrame = Frame(LeftFrame, bd = 5, width = 300, height = 180, padx = 2, pady = 4, relief = RIDGE)  WidgetFrame.pack(side = TOP, padx = 0, pady = 4)  RightFrame = Frame(TopFrame, bd = 5, width = 320, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  RightFrame.pack(side = RIGHT)  TreeViewFrame = Frame(RightFrame, bd = 5, width = 310, height = 200, padx = 2, pady = 2, relief = RIDGE)  TreeViewFrame.pack(side = TOP)  ###~~~BILLS Title~~~###  # Create and configure the title label for the billing window  self.lblTitle = Label(TitleFrame, font = ('arial', 56, 'bold'), text='Billing DB', bd = 7)  self.lblTitle.grid(row = 0, column = 0, padx =132)  ###~~~BILLS Buttons~~~###  # Create and configure the buttons for various operations in the billing window  self.btnAddNew = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Insert New" ,padx = 24, width = 8, height = 1, command = addData).grid(row = 0, column = 0, padx = 1)  self.btnDisplay = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Display" ,padx = 24, width = 8, height = 1, command = displayData).grid(row = 0, column = 1, padx = 1)  self.btnDelete = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Delete" ,padx = 24, width = 8, height = 1, command = deleteData).grid(row = 0, column = 2, padx = 1)  self.btnUpdate = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Update" ,padx = 24, width = 8, height = 1, command = update).grid(row = 0, column = 3, padx = 1)  self.btnReset = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Reset" ,padx = 24, width = 8, height = 1, command = iReset).grid(row = 0, column = 4, padx = 1)  self.btnExit = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Exit" ,padx = 24, width = 8, height = 1, command = iExit).grid(row = 0, column = 5, padx = 1)  ###~~~BILLS Labels~~~###  # Create and configure labels for the input fields  self.lblid = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Bill ID ', bd = 7, anchor='w', justify=LEFT)  self.lblid.grid(row=0,column=0,sticky =W,padx=5)  self.txtid = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = id)  self.txtid.grid(row=0, column=1)  self.lblcustomer\_id = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Customer ID ', bd = 7, anchor='w', justify=LEFT)  self.lblcustomer\_id.grid(row=1,column=0,sticky =W,padx=5)  self.txtcustomer\_id = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = customer\_id)  self.txtcustomer\_id.grid(row=1, column=1)  self.lblPayments\_Due = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Payment Due ', bd = 7, anchor='w', justify=LEFT)  self.lblPayments\_Due.grid(row=2,column=0,sticky =W,padx=5)  self.txtPayments\_Due = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Payments\_Due)  self.txtPayments\_Due.grid(row=2, column=1)  self.lbldue\_Date = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Due Date', bd = 7, anchor='w', justify=LEFT)  self.lbldue\_Date.grid(row=3,column=0,sticky =W,padx=5)  self.txtdue\_Date = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = due\_Date)  self.txtdue\_Date.grid(row=3, column=1)  ###~~~BILLS TreeView~~~###  # Create and configure a Treeview widget to display the billing records  scroll\_x = Scrollbar(TreeViewFrame, orient = HORIZONTAL)  scroll\_y = Scrollbar(TreeViewFrame, orient = VERTICAL)  self.billinglist = ttk.Treeview(TreeViewFrame, height = 12, columns = ("id", "customer\_id", "Payments\_Due", "due\_Date"), xscrollcommand = scroll\_x.set,yscrollcommand = scroll\_y.set)  scroll\_x.pack(side = BOTTOM, fill = X)  scroll\_y.pack(side = BOTTOM, fill = Y)  self.billinglist.heading("id", text = "Bill ID")  self.billinglist.heading("customer\_id", text = "Customer ID")  self.billinglist.heading("Payments\_Due", text = "Payemt Due")  self.billinglist.heading("due\_Date", text = "Due Date")  self.billinglist['show'] = 'headings'  self.billinglist.column("id", width = 90)  self.billinglist.column("customer\_id", width = 90)  self.billinglist.column("Payments\_Due", width = 90)  self.billinglist.column("due\_Date", width = 150)  self.billinglist.pack(fill = BOTH, expand = 1)  self.billinglist.bind("<ButtonRelease-1>", billingREC)  displayData()  ###~~~LOCALITY Class~~~###  class Locality:    ###~~~LOCALITY DB~~~###  # Define the constructor for the class, which takes a "root" argument  def \_\_init\_\_(self, root):  # Set the "root" attribute of the class to the "root" argument  self.root = root  # Set the title and geometry of the root window  blank\_space = " "  self.root.title(200 \* blank\_space + "Locality DB")  self.root.geometry("1920x1080+0+0")  self.root.rowconfigure(0, weight=1)  self.root.columnconfigure(0, weight=1)    # Define StringVars for storing user inputs in Locality window  sector\_no = StringVar()  Area\_Name = StringVar()  Water\_Supply\_Date = StringVar()  officer\_id = StringVar()  reservoir\_id = StringVar()    ###~~~LOCALITY Functions~~~###  # Function to handle the exit button click event  def iExit():  iExit = tkinter.messagebox.askyesno("Exit", "Confirm if you want to exit")  if iExit>0:  root.destroy()  return  # Function to reset the input fields  def iReset():  self.cbosector\_no.current(0)  self.txtWater\_Supply\_Date.delete(0, END)  self.txtofficer\_id.delete(0, END)  self.txtArea\_Name.delete(0, END)  self.txtreservoir\_id.delete(0, END)  # Function to add data to the backend and update the display  def addData():  if sector\_no.get() == "" or Area\_Name.get() == "" or Water\_Supply\_Date.get() == "" or officer\_id.get() == "" or reservoir\_id.get() == "":  tkinter.messagebox.askyesno("Error", "Please enter the correct Data")  else:  backend.addLocality(  sector\_no.get(),  Area\_Name.get(),  Water\_Supply\_Date.get(),  officer\_id.get(),  reservoir\_id.get()  )  displayData()  super(self.localitylist, self).delete()  self.localitylist.insert(END,  (  sector\_no.get(),  Area\_Name.get(),  Water\_Supply\_Date.get(),  officer\_id.get(),  reservoir\_id.get()  ))  # Function to display data in the locality list  def displayData():  result = backend.viewLocality()  if len(result)!=0:  self.localitylist.delete(\*self.localitylist.get\_children())  for row in result:  self.localitylist.insert('', END, values = row)  # Function to delete data from the backend and update the display  def deleteData():  if(len(sector\_no.get())!= 0):  backend.delLocality(sd[0])  iReset()  displayData()  tkinter.messagebox.showinfo("Delete", "Record successfully deleted")  # Function to update data in the backend and update the display  def update():  if(len(sector\_no.get()) != 0):  backend.delLocality(sd[0])  if(len(sector\_no.get()) != 0):  backend.addLocality(sector\_no.get(), Area\_Name.get(), Water\_Supply\_Date.get(), officer\_id.get(), reservoir\_id.get())  displayData()  def localityREC(event):  # Function to handle the locality table's selection event  global sd  iReset()  viewInfo = self.localitylist.focus()  learnerData = self.localitylist.item(viewInfo)  sd = learnerData['values']  sector\_no.set(sd[0])  self.txtArea\_Name.insert(END,sd[1])  self.txtWater\_Supply\_Date.insert(END,sd[2])  self.txtofficer\_id.insert(END,sd[3])  self.txtreservoir\_id.insert(END,sd[4])  ###~~~LOCALITY Frames~~~###  # Create the main frame with specified dimensions, border, relief, and background color  MainFrame = Frame(self.root, bd = 10, width = 1350, height = 700, relief = RIDGE, bg = "cadet blue")  MainFrame.grid()  # Create a sub-frame for buttons with specified dimensions, border, and relief, and place it in the main frame  ButtonFrame = Frame(MainFrame, bd = 5, width = 1340, height = 100, relief = RIDGE)  ButtonFrame.grid(row = 2, column = 0, pady = 8)  # Create a sub-frame for the title with specified dimensions, border, and relief, and place it in the main frame  TitleFrame = Frame(MainFrame, bd = 7, width = 1340, height = 100, relief = RIDGE)  TitleFrame.grid(row = 0, column = 0)  # Create a sub-frame for the content at the top with specified dimensions, border, and relief, and place it in the main frame  TopFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame.grid(row = 1, column = 0)  # Create a sub-frame for the content at the left side of the top frame with specified dimensions, border, background color, and relief, and place it to the left  LeftFrame = Frame(TopFrame, bd = 5, width = 1340, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  LeftFrame.pack(side = LEFT)  # Create a sub-frame for widgets inside the left frame with specified dimensions, border, padding, and relief, and place it at the top  WidgetFrame = Frame(LeftFrame, bd = 5, width = 300, height = 180, padx = 2, pady = 4, relief = RIDGE)  WidgetFrame.pack(side = TOP, padx = 0, pady = 4)  # Create a sub-frame for the content at the right side of the top frame with specified dimensions, border, background color, and relief, and place it to the right  RightFrame = Frame(TopFrame, bd = 5, width = 320, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  RightFrame.pack(side = RIGHT)  # Create a sub-frame for the tree view inside the right frame with specified dimensions, border, padding, and relief, and place it at the top  TreeViewFrame = Frame(RightFrame, bd = 5, width = 310, height = 200, padx = 2, pady = 2, relief = RIDGE)  TreeViewFrame.pack(side = TOP)  ###~~~LOCALITY Class~~~###  # Create a label for the title text inside the title frame with specified font, text, border, and position  self.lblTitle = Label(TitleFrame, font = ('arial', 56, 'bold'), text='Locality DB', bd = 7)  self.lblTitle.grid(row = 0, column = 0, padx =132)  ###~~~LOCALITY Button~~~###  # Create a button for adding new data with specified properties and command, and place it in the button frame  self.btnAddNew = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Insert New" ,padx = 24, width = 8, height = 1, command = addData).grid(row = 0, column = 0, padx = 1)    # Create a button for displaying data with specified properties and command, and place it in the button frame  self.btnDisplay = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Display" ,padx = 24, width = 8, height = 1, command = displayData).grid(row = 0, column = 1, padx = 1)    # Create a button for deleting data with specified properties and command, and place it in the button frame  self.btnDelete = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Delete" ,padx = 24, width = 8, height = 1, command = deleteData).grid(row = 0, column = 2, padx = 1)    # Create a button for updating data with specified properties and command, and place it in the button frame  self.btnUpdate = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Update" ,padx = 24, width = 8, height = 1, command = update).grid(row = 0, column = 3, padx = 1)    # Create a button for resetting data with specified properties and command, and place it in the button frame  self.btnReset = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Reset" ,padx = 24, width = 8, height = 1, command = iReset).grid(row = 0, column = 4, padx = 1)    # Create a button for exiting the program with specified properties and command, and place it in the button frame  self.btnExit = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Exit" ,padx = 24, width = 8, height = 1, command = iExit).grid(row = 0, column = 5, padx = 1)  ###~~~LOCALITY Buttons~~~###  # Create a label for the "Sector No." text with specified properties, and place it in the widget frame  self.lblsector\_no = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Sector No. ', bd = 7, anchor='w', justify=LEFT)  self.lblsector\_no.grid(row=0,column=0,sticky =W,padx=5)    # Create a combobox for selecting the sector number with specified properties, and place it in the widget frame  self.cbosector\_no = ttk.Combobox(WidgetFrame, width = 39, font = ('arial', 12, 'bold'), state = 'readonly', textvariable = sector\_no)  self.cbosector\_no['values'] = ('','1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13')  self.cbosector\_no.current(0)  self.cbosector\_no.grid(row = 0, column = 1)  # Create a label for the "Area Name" text with specified properties, and place it in the widget frame  self.lblArea\_Name = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Area Name ', bd = 7, anchor='w', justify=LEFT)  self.lblArea\_Name.grid(row=1,column=0,sticky =W,padx=5)    # Create an entry field for entering the area name with specified properties, and place it in the widget frame  self.txtArea\_Name = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Area\_Name)  self.txtArea\_Name.grid(row=1, column=1)  # Create a label for the "Water Supply Date" text with specified properties, and place it in the widget frame  self.lblWater\_Supply\_Date = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Water Supply Date ', bd = 7, anchor='w', justify=LEFT)  self.lblWater\_Supply\_Date.grid(row=2,column=0,sticky =W,padx=5)    # Create an entry field for entering the water supply date with specified properties, and place it in the widget frame  self.txtWater\_Supply\_Date = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Water\_Supply\_Date)  self.txtWater\_Supply\_Date.grid(row=2, column=1)  # Create a label for the "Officer ID" text with specified properties, and place it in the widget frame  self.lblofficer\_id = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Officer ID ', bd = 7, anchor='w', justify=LEFT)  self.lblofficer\_id.grid(row=3,column=0,sticky =W,padx=5)    # Create an entry field for entering the officer ID with specified properties, and place it in the widget frame  self.txtofficer\_id = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = officer\_id)  self.txtofficer\_id.grid(row=3, column=1)  # Create a label for the "Reservoir ID" text with specified properties, and place it in the widget frame  self.lblreservoir\_id = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Reservoir ID', bd = 7, anchor='w', justify=LEFT)  self.lblreservoir\_id.grid(row=4,column=0,sticky =W,padx=5)    # Create an entry field for entering the reservoir ID with specified properties, and place it in the widget frame  self.txtreservoir\_id = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = reservoir\_id)  self.txtreservoir\_id.grid(row=4, column=1)  ###~~~LOCALITY TreeView~~~###  # Create horizontal and vertical scrollbars for the locality treeview  scroll\_x = Scrollbar(TreeViewFrame, orient = HORIZONTAL)  scroll\_y = Scrollbar(TreeViewFrame, orient = VERTICAL)  # Create a treeview for displaying locality data with specified properties and scrollbars, and place it in the treeview frame  self.localitylist = ttk.Treeview(TreeViewFrame, height = 12, columns = ("sector\_no", "Area\_Name", "Water\_Supply\_Date", "officer\_id", "reservoir\_id"), xscrollcommand = scroll\_x.set,yscrollcommand = scroll\_y.set)  scroll\_x.pack(side = BOTTOM, fill = X)  scroll\_y.pack(side = BOTTOM, fill = Y)  # Set the headings for the columns in the locality treeview  self.localitylist.heading("sector\_no", text = "Sector No.")  self.localitylist.heading("Area\_Name", text = "Area Name")  self.localitylist.heading("Water\_Supply\_Date", text = "Water Supply Date")  self.localitylist.heading("officer\_id", text = "Officer ID")  self.localitylist.heading("reservoir\_id", text = "Reservoir ID")  # Display only the headings in the locality treeview  self.localitylist['show'] = 'headings'  # Set the width for each column in the locality treeview  self.localitylist.column("sector\_no", width = 90)  self.localitylist.column("Area\_Name", width = 150)  self.localitylist.column("Water\_Supply\_Date", width = 150)  self.localitylist.column("officer\_id", width = 90)  self.localitylist.column("reservoir\_id", width = 90)  # Pack and expand the locality treeview to fill the available space  self.localitylist.pack(fill = BOTH, expand = 1)  # Bind the "<ButtonRelease-1>" event to the localityREC function  self.localitylist.bind("<ButtonRelease-1>", localityREC)  # Call the displayData function to populate the locality treeview with data  displayData()  ###~~~RESERVOIR Class~~~###  class Reservoir:  ###~~~RESERVOIR DB~~~###  def \_\_init\_\_(self, root):  # Initialize the Reservoir class with the root window  self.root = root  blank\_space = " "  # Set the title and geometry of the root window  self.root.title(200 \* blank\_space + "Reservoir DB")  self.root.geometry("1920x1080+0+0")  self.root.rowconfigure(0, weight=1)  self.root.columnconfigure(0, weight=1)  # Create StringVar variables for storing the Reservoir ID, Name, and Water Level  id = StringVar()  Name = StringVar()  Water\_level = StringVar()    ###~~~RESERVOIR Functions~~~###  def iExit():  # Function to exit the application  iExit = tkinter.messagebox.askyesno("Exit", "Confirm if you want to exit")  if iExit>0:  root.destroy()  return  def iReset():  # Function to reset the input fields  self.txtid.delete(0, END)  self.txtName.delete(0, END)  self.txtWater\_level.delete(0, END)  def addData():  # Function to add data to the reservoir database  if id.get() == "" or Name.get() == "" or Water\_level.get() == "":  # Check if all input fields are filled  tkinter.messagebox.askyesno("Error", "Please enter the correct Data")  else:  # Call the backend function to add reservoir data  backend.addReservoir(  id.get(),  Name.get(),  Water\_level.get()  )  # Call the displayData function to update the data display  displayData()  # Delete the contents of the reservoir list treeview  super(self.reservoirlist, self).delete()  # Insert the new data into the reservoir list  self.reservoirlist.insert(END,  (  id.get(),  Name.get(),  Water\_level.get()  ))  def displayData():  # Function to display data from the reservoir database  result = backend.viewReservoir()  # Check if there is any data in the result  if len(result)!=0:  # Clear the existing data in the reservoir list treeview  self.reservoirlist.delete(\*self.reservoirlist.get\_children())  # Iterate over each row in the result and insert it into the reservoirlist  for row in result:  self.reservoirlist.insert('', END, values = row)  def deleteData():  # Function to delete data from the reservoir database  # Check if the id field is not empty  if(len(id.get())!= 0):  backend.delReservoir(sd[0]) # Delete the record corresponding to the id value from the backend  iReset() # Reset the input fields  displayData() # Update the data display  tkinter.messagebox.showinfo("Delete","Record successfully deleted") # Show a messagebox to indicate that the record was deleted successfully  def update():  # Function to update data in the reservoir database  # Check if the id field is not empty  if(len(id.get()) != 0):  backend.delReservoir(sd[0]) # Delete the existing record with the same id from the backend  if(len(id.get()) != 0):  backend.addReservoir(id.get(), Name.get(), Water\_level.get()) # Add the updated data to the reservoir database  displayData() # Update the data display    def ReservoirREC(event):  # Function to handle events related to the reservoir list treeview  global sd  iReset() # Reset the input fields  viewInfo = self.reservoirlist.focus() # Get the focused item in the reservoir list treeview  learnerData = self.reservoirlist.item(viewInfo) # Get the data associated with the focused item  sd = learnerData['values'] # Extract the values from the data dictionary and assign them to sd  # Insert the values from sd into the corresponding input fields  self.txtid.insert(END,sd[0])  self.txtName.insert(END,sd[1])  self.txtWater\_level.insert(END,sd[2])  ###~~~RESERVOIR Frames~~~###  # Creating the main frame for the application  MainFrame = Frame(self.root, bd = 10, width = 1350, height = 700, relief = RIDGE, bg = "cadet blue")  MainFrame.grid()  # Creating a frame for buttons  ButtonFrame = Frame(MainFrame, bd = 5, width = 1340, height = 100, relief = RIDGE)  ButtonFrame.grid(row = 2, column = 0, pady = 8)  # Creating a frame for the title  TitleFrame = Frame(MainFrame, bd = 7, width = 1340, height = 100, relief = RIDGE)  TitleFrame.grid(row = 0, column = 0)  # Creating a frame for the top section  TopFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame.grid(row = 1, column = 0)  # Creating a frame for the left section  LeftFrame = Frame(TopFrame, bd = 5, width = 1340, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  LeftFrame.pack(side = LEFT)  # Creating a frame for widgets  WidgetFrame = Frame(LeftFrame, bd = 5, width = 300, height = 180, padx = 2, pady = 4, relief = RIDGE)  WidgetFrame.pack(side = TOP, padx = 0, pady = 4)  # Creating a frame for the right section  RightFrame = Frame(TopFrame, bd = 5, width = 320, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  RightFrame.pack(side = RIGHT)  # Creating a frame for the tree view  TreeViewFrame = Frame(RightFrame, bd = 5, width = 310, height = 200, padx = 2, pady = 2, relief = RIDGE)  TreeViewFrame.pack(side = TOP)  ###~~~RESERVOIR Title~~~###  # Creating a label for the title  self.lblTitle = Label(TitleFrame, font = ('arial', 56, 'bold'), text='Reservoir DB', bd = 7)  self.lblTitle.grid(row = 0, column = 0, padx =132)  ###~~~RESERVOIR Labels~~~###  # Creating labels for reservoir information  self.lblid = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Reservoir ID ', bd = 7, anchor='w', justify=LEFT)  self.lblid.grid(row=0,column=0,sticky =W,padx=5)  self.txtid = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = id)  self.txtid.grid(row=0, column=1)  self.lblName = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Reservoir Name ', bd = 7, anchor='w', justify=LEFT)  self.lblName.grid(row=1,column=0,sticky =W,padx=5)  self.txtName = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Name)  self.txtName.grid(row=1, column=1)  self.lblWater\_level = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Water Level ', bd = 7, anchor='w', justify=LEFT)  self.lblWater\_level.grid(row=2,column=0,sticky =W,padx=5)  self.txtWater\_level = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Water\_level)  self.txtWater\_level.grid(row=2, column=1)  ###~~~RESERVOIR TreeView~~~###  # Creating horizontal and vertical scrollbars for the treeview  scroll\_x = Scrollbar(TreeViewFrame, orient = HORIZONTAL)  scroll\_y = Scrollbar(TreeViewFrame, orient = VERTICAL)  # Creating the treeview widget with specified columns and scroll commands  self.reservoirlist = ttk.Treeview(TreeViewFrame, height = 12, columns = ("id", "Name", "Water\_level"), xscrollcommand = scroll\_x.set,yscrollcommand = scroll\_y.set)  scroll\_x.pack(side=BOTTOM, fill=X) # Displaying the horizontal scrollbar at the bottom  scroll\_y.pack(side=BOTTOM, fill=Y) # Displaying the vertical scrollbar at the bottom  # Configuring column headings for the treeview  self.reservoirlist.heading("id", text = "Reservoir ID")  self.reservoirlist.heading("Name", text = "Reservoir Name")  self.reservoirlist.heading("Water\_level", text = "Water level")  self.reservoirlist['show'] = 'headings' # Showing only the column headings  self.reservoirlist.column("id", width=90) # Setting the width of the "id" column  self.reservoirlist.column("Name", width=150) # Setting the width of the "Name" column  self.reservoirlist.column("Water\_level", width=90) # Setting the width of the "Water\_level" column  self.reservoirlist.pack(fill=BOTH, expand=1) # Displaying the treeview and filling the available space  self.reservoirlist.bind("<ButtonRelease-1>", ReservoirREC) # Binding a function to the treeview's button release event  displayData() # Calling a function to display data in the treeview    ###~~~RESERVOIR Buttons~~~###  # Creating buttons with specified properties and commands  self.btnAddNew = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Insert New" ,padx = 24, width = 8, height = 1, command = addData).grid(row = 0, column = 0, padx = 1)  self.btnDisplay = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Display" ,padx = 24, width = 8, height = 1, command = displayData).grid(row = 0, column = 1, padx = 1)  self.btnDelete = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Delete" ,padx = 24, width = 8, height = 1, command = deleteData).grid(row = 0, column = 2, padx = 1)  self.btnUpdate = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Update" ,padx = 24, width = 8, height = 1, command = update).grid(row = 0, column = 3, padx = 1)  self.btnReset = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Reset" ,padx = 24, width = 8, height = 1, command = iReset).grid(row = 0, column = 4, padx = 1)  self.btnExit = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Exit" ,padx = 24, width = 8, height = 1, command = iExit).grid(row = 0, column = 5, padx = 1)  ###~~~creates Window2 for Customer Login~~~###  ## Class definition for Window2, which represents the Customer Login window  class Window2:  def \_\_init\_\_(self, root):  self.root = root  blank\_space = " "  # Set the title of the window with a long blank space  self.root.title(200 \* blank\_space + "USER")  # Set the size and position of the window  self.root.geometry("1920x1080+0+0")  # Configure the first row and first column to expand with the window  self.root.rowconfigure(0, weight=1)  self.root.columnconfigure(0, weight=1)    ###~~~Customer Login Frames~~~###  # Creating the main frame for the window  MainFrame = Frame(self.root, bd = 10, width = 1350, height = 700, relief = RIDGE, bg = "cadet blue")  MainFrame.grid()  # Creating sub-frames within the main frame  ButtonFrame = Frame(MainFrame, bd = 5, width = 1340, height = 100, relief = RIDGE)  ButtonFrame.grid(row = 2, column = 0, pady = 8)  TitleFrame = Frame(MainFrame, bd = 7, width = 1340, height = 100, relief = RIDGE)  TitleFrame.grid(row = 0, column = 0)  TopFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame.grid(row = 1, column = 0)  TopFrame1 = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame1.grid(row = 3, column = 0)  ###~~~Customer Login Frames~~~###  # Creating and displaying the title label  self.lblTitle = Label(TitleFrame, font = ('arial', 56, 'bold'), text='User Login', bd = 7)  self.lblTitle.grid(row = 0, column = 0, padx =132)  ###~~~Customer Label~~~###  # Creating and displaying the "Enter User ID" label  self.l1 = Label(TopFrame, font = ('arial', 20, 'bold'), text='Enter User ID: ', width=15, height = 1)  self.l1.grid(row=1,column=1)  # Creating and displaying the text box for user input  self.t1 = Text(TopFrame, font = ('arial', 20, 'bold'), height=1, width=10 ,bg='white')  self.t1.grid(row=1,column=2)  ###~~~Customer Login User Data and Labels~~~###  # Create StringVar variables to store user data  my\_str1 = StringVar()  my\_str2 = StringVar()  my\_str3 = StringVar()  my\_str4 = StringVar()  my\_str5 = StringVar()  my\_str6 = StringVar()  my\_str7 = StringVar()  my\_str8 = StringVar()  my\_str9 = StringVar()  # Create labels for displaying user data  # Label for ID  self.demo\_l2 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'ID', width=20,bd = 7, anchor='center')  self.demo\_l2.grid(row=3, column = 1, padx=10)  self.l2 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str1, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE )  self.l2.grid(row=3,column=2)    # Label for Name  self.demo\_l3 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'Name', width=20,bd = 7, anchor='center')  self.demo\_l3.grid(row=4, column = 1, padx=10)  self.l3 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str2, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE )  self.l3.grid(row=4,column=2)    # Label for Address  self.demo\_l4 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'Address', width=20,bd = 7, anchor='center')  self.demo\_l4.grid(row=5, column = 1, padx=10)  self.l4 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str3, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE )  self.l4.grid(row=5,column=2)    # Label for Sector No.  self.demo\_l5 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'Sector No.', width=20,bd = 7, anchor='center')  self.demo\_l5.grid(row=6, column = 1, padx=10)  self.l5 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str4, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE )  self.l5.grid(row=6,column=2)    # Label for Officer ID  self.demo\_l6 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'Officer ID', width=20,bd = 7, anchor='center')  self.demo\_l6.grid(row=7, column = 1, padx=10)  self.l6 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str5, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE )  self.l6.grid(row=7,column=2)    # Label for Reservoir ID  self.demo\_l7 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'Reservoir ID', width=20,bd = 7, anchor='center')  self.demo\_l7.grid(row=8, column = 1, padx=10)  self.l7 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str6, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE )  self.l7.grid(row=8,column=2)    # Label for No. of Connections  self.demo\_l8 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'No. of Connections', width=20,bd = 7, anchor='center')  self.demo\_l8.grid(row=9, column = 1, padx=10)  self.l8 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str7, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE)  self.l8.grid(row=9,column=2)  #Label for Water Supply Date  self.demo\_l9 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'Water Supply Date', width=20,bd = 7, anchor='center')  self.demo\_l9.grid(row=10, column = 1, padx=10)  self.l9 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str8, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE)  self.l9.grid(row=10,column=2)  #Label for Bill Due  self.demo\_l10 = Label(TopFrame1, font = ('arial', 20, 'bold'), text = 'Bill Due', width=20,bd = 7, anchor='center')  self.demo\_l10.grid(row=11, column = 1, padx=10)  self.l10 = Label(TopFrame1, font = ('arial', 20, 'bold'), textvariable=my\_str9, width=30, fg='purple', bd = 7, anchor='center',relief=RIDGE)  self.l10.grid(row=11,column=2)  # Set the StringVar variables to empty strings  my\_str1.set("")  my\_str2.set("")  my\_str3.set("")  my\_str4.set("")  my\_str5.set("")  my\_str6.set("")  my\_str7.set("")  my\_str8.set("")  my\_str9.set("")  ###~~~Customer Login Function~~~###  # Function to fetch and display customer details based on the provided ID  def my\_details(id):  try:  # SQL query to fetch customer details from the 'Customer' table using the provided ID  q="SELECT \* FROM Customer WHERE id= "+id  my\_cursor=my\_conn.execute(q)  data\_row=my\_cursor.fetchone()  # Update the StringVar variables with the fetched data  my\_str1.set(data\_row[0]) # ID  my\_str2.set(data\_row[1]) # Name  my\_str3.set(data\_row[2]) # Address  my\_str4.set(data\_row[3]) # Sector No.  my\_str5.set(data\_row[4]) # Officer ID  my\_str6.set(data\_row[5]) # Reservoir ID  my\_str7.set(data\_row[6]) # No. of Connections    # SQL query to fetch water supply date from the 'Locality' table based on the customer's sector  w="SELECT Locality.Water\_Supply\_Date FROM Locality, Customer WHERE Locality.sector\_no = Customer.sector\_no AND Customer.id = "+id  my\_cursor1=my\_conn.execute(w)  data\_row1=my\_cursor1.fetchone()  my\_str8.set(data\_row1) # Water Supply Date    # SQL query to fetch payment due from the 'Bills' table based on the customer's ID  e="SELECT Bills.Payments\_Due FROM Bills, Customer WHERE Bills.customer\_id = Customer.id AND Customer.id= "+id  my\_cursor2=my\_conn.execute(e)  data\_row2=my\_cursor2.fetchone()  my\_str9.set(data\_row2) # Bill Due  except sqlite3.Error as my\_error:  print("error: ",my\_error)  ###~~~Customer Login Button~~~###  self.btn = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Show Details" ,padx = 24, width = 8, height = 1, command = lambda: my\_details(self.t1.get('1.0',END))).grid(row = 0, column = 0, padx = 1)  ###~~~Creates Window3 for Officer Login~~~###  # Class definition for Window3, which represents the Officer Login window  class Window3:  def \_\_init\_\_(self, root):  self.root = root  blank\_space = " "  self.root.title(200 \* blank\_space + "Officer Details") # Set the title of the window  self.root.geometry("1920x1080+0+0") # Set the size and position of the window  self.root.rowconfigure(0, weight=1)  self.root.columnconfigure(0, weight=1)  # StringVar variables to store the input data  id = StringVar()  Name = StringVar()  Address = StringVar()  sector\_no = StringVar()  officer\_id = StringVar()  reservoir\_id = StringVar()  no\_of\_connection = StringVar()  ###~~~Officer Login Button~~~###  # Function to add data to the database  def addData():  if id.get() == "" or Name.get() == "" or Address.get() == "" or sector\_no.get() == "" or officer\_id.get() == "" or reservoir\_id.get() == "" or no\_of\_connection.get() == "":  tkinter.messagebox.askyesno("Error", "Please enter the correct Data")  else:  backend.addCustomer(  id.get(),  Name.get(),  Address.get(),  sector\_no.get(),  officer\_id.get(),  reservoir\_id.get(),  no\_of\_connection.get()  )  displayData()  super(self.OfficerCustomerList, self).delete()  self.OfficerCustomerList.insert(END,  (  id.get(),  Name.get(),  Address.get(),  sector\_no.get(),  officer\_id.get(),  reservoir\_id.get(),  no\_of\_connection.get()  ))  # Function to display data from the database  def displayData():  result = backend.viewCustomerFromOfficerID(officer\_id.get())  if len(result)!=0:  self.OfficerCustomerList.delete(\*self.OfficerCustomerList.get\_children())  for row in result:  self.OfficerCustomerList.insert('', END, values = row)  # Function to reset the input fields  def iReset():  self.txtid.delete(0, END)  self.txtName.delete(0, END)  self.txtAddress.delete(0, END)  self.cbosector\_no.current(0)  self.txtofficer\_id.delete(0, END)  self.txtreservoir\_id.delete(0, END)  self.txtno\_of\_connection.delete(0, END)    # Function to handle the selection of a record from the Officer Customer List tree view  def OfficerCustomerREC(event):  global sd  iReset()  viewInfo = self.OfficerCustomerList.focus()  learnerData = self.OfficerCustomerList.item(viewInfo)  sd = learnerData['values']  self.txtid.insert(END,sd[0])  self.txtName.insert(END,sd[1])  self.txtAddress.insert(END,sd[2])  sector\_no.set(sd[3])  self.txtofficer\_id.insert(END,sd[4])  self.txtreservoir\_id.insert(END,sd[5])  self.txtno\_of\_connection.insert(END,sd[6])  ###~~~Officer Frames~~~###  # Creating frames for the Officer Login window# Creating frames for the Officer Login window  MainFrame = Frame(self.root, bd = 10, width = 1350, height = 700, relief = RIDGE, bg = "cadet blue")  MainFrame.grid()  ButtonFrame = Frame(MainFrame, bd = 5, width = 1340, height = 100, relief = RIDGE)  ButtonFrame.grid(row = 3, column = 0, pady = 5)  TitleFrame = Frame(MainFrame, bd = 7, width = 1340, height = 100, relief = RIDGE)  TitleFrame.grid(row = 0, column = 0)  TopFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  TopFrame.grid(row = 4, column = 0)  LabelFrame = Frame(MainFrame, bd = 5, width = 1340, height = 500, relief = RIDGE)  LabelFrame.grid(row = 2, column = 0, pady = 8)  LeftFrame = Frame(TopFrame, bd = 5, width = 1340, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  LeftFrame.pack(side = LEFT)  WidgetFrame = Frame(LeftFrame, bd = 5, width = 300, height = 180, padx = 2, pady = 4, relief = RIDGE)  WidgetFrame.pack(side = TOP, padx = 0, pady = 4)  RightFrame = Frame(TopFrame, bd = 5, width = 320, height = 400, padx = 2, bg = "cadet blue", relief = RIDGE)  RightFrame.pack(side = RIGHT)  TreeViewFrame = Frame(RightFrame, bd = 5, width = 310, height = 200, padx = 2, pady = 2, relief = RIDGE)  TreeViewFrame.pack(side = TOP)  ###~~~Officer Login Title~~~###    # Create a label for the Officer Login title  self.lblTitle = Label(TitleFrame, font = ('arial', 56, 'bold'), text='Officer\'s Records', bd = 7)  self.lblTitle.grid(row = 0, column = 0, padx =132)    ###~~~Officer Login Button~~~###  # Create a button for displaying Officer's details  self.btnDisplay = Button(ButtonFrame, pady = 1, bd = 4, font = ('arial', 20, 'bold'), text = "Get Details" ,padx = 24, width = 8, height = 1, command = displayData).grid(row = 0, column = 0, padx = 1)  ###~~~Officer Login Label~~~###  # Create a label for entering the Officer ID  self.lblofficer\_id = Label(LabelFrame, font = ('arial',20,'bold'), text = 'Enter Officer ID:', width=15, height = 1, pady = 1, bd = 4,)  self.lblofficer\_id.grid(row=1,column=1)    # Create an entry field for the Officer ID  self.txtofficer\_id = Entry(LabelFrame, font = ('arial',20,'bold'), width=10 ,bg='white',bd=5, textvariable = officer\_id)  self.txtofficer\_id.grid(row=1, column=2)      # Creating widgets for Officer Login  ###~~~Customer ID Widget~~~###  # Create a label for Customer ID  self.lblid = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Customer ID ', bd = 7, anchor='w', justify=LEFT)  self.lblid.grid(row=0,column=0,sticky =W,padx=5)  # Create an entry field for Customer ID  self.txtid = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = id)  self.txtid.grid(row=0, column=1)  ###~~~Customer Name Widget~~~###  # Create a label for Customer Name  self.lblName = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Customer Name ', bd = 7, anchor='w', justify=LEFT)  self.lblName.grid(row=1,column=0,sticky =W,padx=5)    # Create an entry field for Customer Name  self.txtName = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Name)  self.txtName.grid(row=1, column=1)  ###~~~Address Widget~~~###  # Create a label for Address  self.lblAddress = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Address ', bd = 7, anchor='w', justify=LEFT)  self.lblAddress.grid(row=2,column=0,sticky =W,padx=5)    # Create an entry field for Address  self.txtAddress = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = Address)  self.txtAddress.grid(row=2, column=1)  ###~~~Sector No Widget~~~###  # Create a label for Sector No  self.lblsector\_no = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Sector No ', bd = 7, anchor='w', justify=LEFT)  self.lblsector\_no.grid(row=3,column=0,sticky =W,padx=5)  # Create a combobox for Sector No  self.cbosector\_no = ttk.Combobox(WidgetFrame, width = 39, font = ('arial', 12, 'bold'), state = 'readonly', textvariable = sector\_no)  self.cbosector\_no['values'] = ('','1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13')  self.cbosector\_no.current(0)  self.cbosector\_no.grid(row = 3, column = 1)  ###~~~Reservoir ID Widget~~~###  # Create a label for Reservoir ID  self.lblreservoir\_id = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'Reservoir ID', bd = 7, anchor='w', justify=LEFT)  self.lblreservoir\_id.grid(row=4,column=0,sticky =W,padx=5)    # Create an entry field for Reservoir ID  self.txtreservoir\_id = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = reservoir\_id)  self.txtreservoir\_id.grid(row=4, column=1)  ###~~~No. of Connections Widget~~~###  # Create a label for No. of Connections  self.lblno\_of\_connection = Label(WidgetFrame, font = ('arial',12,'bold'), text = 'No. of connections', bd = 7, anchor='w', justify=LEFT)  self.lblno\_of\_connection.grid(row=5,column=0,sticky =W,padx=5)    # Create an entry field for No. of Connections  self.txtno\_of\_connection = Entry(WidgetFrame, font = ('arial',12,'bold'), bd = 5, width = 40, justify = "left", textvariable = no\_of\_connection)  self.txtno\_of\_connection.grid(row=5, column=1)  ###~~~Officer Login TreeView~~~###  # Create horizontal scrollbar for the treeview  scroll\_x = Scrollbar(TreeViewFrame, orient = HORIZONTAL)  # Create vertical scrollbar for the treeview  scroll\_y = Scrollbar(TreeViewFrame, orient = VERTICAL)  # Create a TreeView widget with specified columns and scrollbars  self.OfficerCustomerList = ttk.Treeview(TreeViewFrame, height = 12, columns = ("id", "Name", "Address", "sector\_no", "officer\_id", "reservoir\_id", "no\_of\_connection"), xscrollcommand = scroll\_x.set,yscrollcommand = scroll\_y.set)  # Pack the scrollbars to the bottom of the TreeViewFrame  scroll\_x.pack(side = BOTTOM, fill = X)  scroll\_y.pack(side = BOTTOM, fill = Y)  # Set headings for each column  self.OfficerCustomerList.heading("id", text = "Customer ID")  self.OfficerCustomerList.heading("Name", text = "Customer Name")  self.OfficerCustomerList.heading("Address", text = "Customer Address")  self.OfficerCustomerList.heading("sector\_no", text = "Sector No")  self.OfficerCustomerList.heading("officer\_id", text = "Officer ID")  self.OfficerCustomerList.heading("reservoir\_id", text = "Reservoir ID")  self.OfficerCustomerList.heading("no\_of\_connection", text = "No. of conns.")  # Set 'show' option to display only the headings  self.OfficerCustomerList['show'] = 'headings'  # Set width for each column  self.OfficerCustomerList.column("id", width = 90)  self.OfficerCustomerList.column("Name", width = 200)  self.OfficerCustomerList.column("Address", width = 200)  self.OfficerCustomerList.column("sector\_no", width = 90)  self.OfficerCustomerList.column("officer\_id", width = 90)  self.OfficerCustomerList.column("reservoir\_id", width = 90)  self.OfficerCustomerList.column("no\_of\_connection", width = 90)  # Pack the OfficerCustomerList TreeView to fill and expand in the window  self.OfficerCustomerList.pack(fill = BOTH, expand = 1)  # Bind the ButtonRelease-1 event to OfficerCustomerREC function  self.OfficerCustomerList.bind("<ButtonRelease-1>", OfficerCustomerREC)  # Call the displayData function to populate the TreeView with dat  displayData()  # Check if the current module is being run directly  if \_\_name\_\_ == '\_\_main\_\_':  # Call the main function  main() |

**Backend**

A diagram of a company

Description automatically generated

**Schema**

A screenshot of a document

Description automatically generated

In the backend.py file, several functions handle database operations:

**officerData:** Creates the OFFICER table.

**addOfficer:** Adds an officer to the OFFICER table.

**viewOfficer:** Views all the officers from the OFFICER table.

**delOfficer:** Deletes an officer from the OFFICER table.

**officerView:** Views customers associated with a specific officer.

**reservoirData:** Adds a reservoir to the RESERVOIR table.

**delReservoir:** Deletes a reservoir from the RESERVOIR table.

**viewReservoir:** Views all the reservoirs from the RESERVOIR table.

**billData:** Creates the BILLS table.

**addBill:** Adds a bill to the BILLS table.

**delBill:** Deletes a bill from the BILLS table.

**viewBill:** Views all the bills

|  |
| --- |
| #Importing the necessary modules  import sqlite3  import dates as dates  from datetime import datetime as dt  import datetime as datetime  from tkinter import \*  #BACKEND  #Query to create the OFFICER table  def officerData():      con = sqlite3.connect("backend.db")      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("CREATE TABLE IF NOT EXISTS officer (id INTEGER PRIMARY KEY, Name text, sector\_no text, FOREIGN KEY('sector\_no') REFERENCES 'Locality'('sector\_no') ON UPDATE CASCADE ON DELETE CASCADE)")      con.commit()      con.close()  #Query to add an officer to the OFFICER table  def addOfficer(id, Name, sector\_no):      con = sqlite3.connect("backend.db")      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("INSERT INTO officer VALUES (?, ?, ?)", (id, Name, sector\_no))      con.commit()      con.close()  #Query to view all the officers from the OFFICER table  def viewOfficer():      con = sqlite3.connect("backend.db")      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("SELECT \* FROM officer")      rows = cur.fetchall()      con.close()      return rows  #Query to delete a officer from the OFFICER table  def delOfficer(id):      con = sqlite3.connect("backend.db")      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("DELETE FROM officer WHERE id=?",(id,))      con.commit()      con.close()  #Query to view customers associated with a specific officer  def officerView(officer\_id):      con = sqlite3.connect("backend.db")      cur = con.cursor()      cur.execute("SELECT \* FROM Customer WHERE officer.id = Customer.officer\_id AND Customer.officer\_id = ? ", (officer\_id,))      rows = cur.fetchall()      con.close()      return rows  #Query to create the RESERVOIR table  def reservoirData():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("CREATE TABLE IF NOT EXISTS Reservoir (id INTEGER PRIMARY KEY, Name text, Water\_level text)")      con.commit()      con.close()  #Query to add a reservoir to the RESERVOIR table  def addReservoir(id, Name, Water\_level):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("INSERT INTO Reservoir VALUES (?, ?, ?)", (id, Name, Water\_level))      con.commit()      con.close()  #Query to delete a reservoir from the RESERVOIR table  def delReservoir(id):      con = sqlite3.connect("backend.db")      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("DELETE FROM Reservoir WHERE id=?",(id,))      con.commit()      con.close()  #Query to view all the reservoirs from the RESERVOIR table  def viewReservoir():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("SELECT \* FROM Reservoir")      rows = cur.fetchall()      con.close()      return rows  #Query to create the BILLS table  def billData():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("CREATE TABLE IF NOT EXISTS Bills ('id' INTEGER PRIMARY KEY,  'customer\_id' text, 'Payments\_Due' text, 'due\_Date' text, FOREIGN KEY('customer\_id') REFERENCES 'Customer'('id') ON UPDATE CASCADE ON DELETE CASCADE)")      con.commit()      con.close()  #Query to add a bill to the BILLS table  def addBill(id, customer\_id, Payments\_Due, due\_Date):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("INSERT INTO Bills VALUES (?, ?, ?, ?)", (id, customer\_id, Payments\_Due, due\_Date))      con.commit()      con.close()  #Query to delete a bill from the BILLS table  def delBill(id):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("DELETE FROM Bills WHERE id=?",(id,))      con.commit()      con.close()  #Query to view all the bills from the BILLS table  def viewBill():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("SELECT \* FROM Bills")      rows = cur.fetchall()      con.close()      return rows  #Query to create the LOCALITY table  def localityData():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("CREATE TABLE IF NOT EXISTS Locality ('sector\_no' INTEGER PRIMARY KEY, 'Area\_Name' text, 'Water\_Supply\_Date' text, 'officer\_id' text, 'reservoir\_id' text, FOREIGN KEY('reservoir\_id') REFERENCES 'Reservoir'('id') ON UPDATE CASCADE ON DELETE CASCADE)")      con.commit()      con.close()  #Query to add an area to the LOCALITY table  def addLocality(sector\_no, Area\_Name, Water\_Supply\_Date, officer\_id, reservoir\_id):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("INSERT INTO Locality VALUES (?, ?, ?, ?, ?)", (sector\_no, Area\_Name, Water\_Supply\_Date, officer\_id, reservoir\_id))      con.commit()      con.close()  #Query to delete an area from the LOCALITY table  def delLocality(sector\_no):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("DELETE FROM Locality WHERE sector\_no=?",(sector\_no,))      con.commit()      con.close()  #Query to view all the areas from the LOCALITY table  def viewLocality():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("SELECT \* FROM Locality")      rows = cur.fetchall()      con.close()      return rows  #Query to create the CUSTOMER table  def customerData():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute('''CREATE TABLE IF NOT EXISTS Customer (id INTEGER PRIMARY KEY, Name text, Address text, sector\_no text, officer\_id text, reservoir\_id text, no\_of\_connection text, FOREIGN KEY("sector\_no") REFERENCES "Locality"("sector\_no") ON UPDATE CASCADE ON DELETE CASCADE, FOREIGN KEY("officer\_id") REFERENCES "Officer"("id") ON UPDATE CASCADE ON DELETE CASCADE, FOREIGN KEY("reservoir\_id") REFERENCES "Reservoir"("id") ON UPDATE CASCADE ON DELETE CASCADE)''')      con.commit()      cur.close()  #Query to add a Customer to the CUSTOMER table  def addCustomer(id, Name, Address, sector\_no, officer\_id, reservoir\_id, no\_of\_connection):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("INSERT INTO Customer VALUES (?, ?, ?, ?, ?, ?, ?)", (id, Name, Address, sector\_no, officer\_id, reservoir\_id, no\_of\_connection))      con.commit()      cur.close()  #Query to delete a Customer to the CUSTOMER table  def delCustomer(id):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("DELETE FROM Customer WHERE id = ?",(id,))      con.commit()      cur.close()  #Query to view all the Customers from the CUSTOMER table  def viewCustomer():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("SELECT \* FROM Customer")      rows = cur.fetchall()      con.close()      return rows  #Query to view the Customer(s) from the Officer ID  def viewCustomerFromOfficerID(officer\_id):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("SELECT \* FROM Customer WHERE officer\_id = ?",(officer\_id,))      rows = cur.fetchall()      con.close()      return rows  #Function to update the dates everyday  def updateDateEveryday():      seclist = []      nextdate = 4      d = 0      today = (datetime.date.today())      timedelta = (datetime.timedelta(days = nextdate))      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("SELECT Water\_Supply\_Date FROM Locality")      dAteS = cur.fetchall()      for every in dAteS:          EveryDate = every[0]          string\_date = changeToDate(EveryDate)          seclist.append(string\_date)      appended\_Date = today + timedelta      for all in seclist:          if all == datetime.date.today():              seclist[d] = appended\_Date          d=d+1      for all in seclist:          dst = changeToString(all)          cur.execute("UPDATE Locality SET Water\_Supply\_Date = ? WHERE TRUE",(dst,))      con.commit()      cur.close()  #To change the string to date format  def changeToDate(strng):      return dt.strptime(strng, '%d %B %Y').date()  #To change the date to string format  def changeToString(date):      return date.strftime('%d %B %Y')  #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~UPDATE QUERIES~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  #Query to update the OFFICER table  def updateOfficer():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("UPDATE Officer SET id = ? , Name = ? , sector\_no = ? ", (id, Name, sector\_no))      con.commit()      cur.close()  #Query to update the RESERVOIR table  def updateReservoir():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("UPDATE Reservoir SET id = ? , Name = ? , Water\_level = ? ", (id, Name, Water\_level))      con.commit()      cur.close()  #Query to update the BILL table  def updateBill():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("UPDATE Bills SET id = ? , Payments\_Due = ? , due\_Date = ? , customer\_id = ? ", (id, Payments\_Due, due\_Date, customer\_id))      con.commit()      cur.close()  #Query to update the LOCALITY table  def updateLocality():      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("UPDATE Locality SET sector\_no = ? , Area\_Name = ? , Water\_Supply\_Date = ? , officer\_id = ? , reservoir\_id = ? ", (sector\_no, Area\_Name, Water\_Supply\_Date, officer\_id, reservoir\_id))      con.commit()      cur.close()  #Query to update the CUSTOMER table  def updateCustomer(id):      con = sqlite3.connect('backend.db')      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("UPDATE Customer SET id = ? , Name = ? , sector\_no = ? , officer\_id = ? , reservoir\_id = ? , no\_of\_connection = ? WHERE id = ? ", (id, Name, Address, sector\_no, officer\_id, reservoir\_id, no\_of\_connection, id))      con.commit()      cur.close()  #Query to update the entire Officer Set  def dataOfficerUpdate(id="", Name="", sector\_no=""):      con=sqlite3.connect("backend.db")      cur = con.cursor()      cur.execute("PRAGMA foreign\_keys = ON")      cur.execute("UPDATE officer SET id=?, Name=?, sector\_no=?", (id, Name, sector\_no))      con.commit()      con.close()  #Calling the methods  officerData()  reservoirData()  billData()  localityData()  customerData()  updateDateEveryday() |

**Graphical User Interface**

A screenshot of a computer

Description automatically generated

A screenshot of a login screen

Description automatically generated

A screenshot of a login screen

Description automatically generated

A screenshot of a login page

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

**Conclusion and Recommendations**

**Conclusion**

The Water Supply Management System addresses the growing challenges of water resource management by streamlining supply schedules and improving communication with residents. By leveraging technology, the system aims to optimize resource utilization and ensure equitable distribution of water.

**Recommendations**

* **Integration of IoT**: Enhance the system by incorporating IoT devices for automated water supply management and consumption monitoring.
* **Smart Home Management**: Develop features for users to monitor daily water consumption and receive alerts via a mobile application.
* **Enhanced Communication**: Implement features to improve communication between water authorities and residents, ensuring timely notifications and updates.

**Future Enhancements**

* **Real-time Monitoring**: Integrate real-time monitoring capabilities to track water usage and detect anomalies.
* **Predictive Analysis**: Implement predictive algorithms to forecast water demand and optimize supply schedules.
* **Geospatial Visualization**: Incorporate geospatial visualization tools for better understanding and management of water distribution networks.

**References:**

Tkinter documentation: <https://docs.python.org/3/library/tkinter.html>

SQLite documentation: https://www.sqlite.org/docs.html

Python datetime module documentation: <https://docs.python.org/3/library/datetime.html>

SQLite foreign key constraints: https://www.sqlite.org/foreignkeys.html