# USER REGISTRATION AND AUTHENTICATION:

import tkinter as tk

from tkinter import messagebox import sqlite3

# Create the database and table if not exists conn = sqlite3.connect('user\_db.db')

c = conn.cursor()

# Create table if it doesn't exist c.execute('''

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT NOT NULL UNIQUE, password TEXT NOT NULL

) ''')

conn.commit()

# Function to register a user def register\_user():

username = entry\_username.get() password = entry\_password.get()

confirm\_password = entry\_confirm\_password.get()

if username == "" or password == "": messagebox.showerror("Error", "Please fill all fields") return

if password != confirm\_password: messagebox.showerror("Error", "Passwords do not match") return

# Insert into database try:

c.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, password))

conn.commit()

messagebox.showinfo("Success", "Registration Successful") entry\_username.delete(0, tk.END)

entry\_password.delete(0, tk.END) entry\_confirm\_password.delete(0, tk.END)

except sqlite3.IntegrityError:

messagebox.showerror("Error", "Username already exists")

# Function to authenticate user def authenticate\_user():

username = entry\_login\_username.get() password = entry\_login\_password.get()

if username == "" or password == "": messagebox.showerror("Error", "Please fill all fields") return

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

user = c.fetchone()

if user:

messagebox.showinfo("Success", "Login Successful") else:

messagebox.showerror("Error", "Invalid username or password")

# Create the main window window = tk.Tk()

window.title("User Registration and Login")

# Registration Form frame\_register = tk.Frame(window) frame\_register.pack(pady=10)

label\_username = tk.Label(frame\_register, text="Username:") label\_username.grid(row=0, column=0)

entry\_username = tk.Entry(frame\_register) entry\_username.grid(row=0, column=1)

label\_password = tk.Label(frame\_register, text="Password:") label\_password.grid(row=1, column=0)

entry\_password = tk.Entry(frame\_register, show="\*") entry\_password.grid(row=1, column=1)

label\_confirm\_password = tk.Label(frame\_register, text="Confirm Password:")

label\_confirm\_password.grid(row=2, column=0) entry\_confirm\_password = tk.Entry(frame\_register, show="\*") entry\_confirm\_password.grid(row=2, column=1)

button\_register = tk.Button(frame\_register, text="Register", command=register\_user)

button\_register.grid(row=3, columnspan=2)

# Login Form

frame\_login = tk.Frame(window) frame\_login.pack(pady=10)

label\_login\_username = tk.Label(frame\_login, text="Username:") label\_login\_username.grid(row=0, column=0) entry\_login\_username = tk.Entry(frame\_login) entry\_login\_username.grid(row=0, column=1)

label\_login\_password = tk.Label(frame\_login, text="Password:") label\_login\_password.grid(row=1, column=0) entry\_login\_password = tk.Entry(frame\_login, show="\*") entry\_login\_password.grid(row=1, column=1)

button\_login = tk.Button(frame\_login, text="Login", command=authenticate\_user)

button\_login.grid(row=2, columnspan=2)

# Run the Tkinter event loop window.mainloop()

# Close the database connection when done conn.close()

# TOLL CALCULATION

import tkinter as tk

from tkinter import messagebox

# Function to calculate toll fee def calculate\_toll():

try:

# Get the inputs from the user vehicle\_type = vehicle\_type\_var.get() distance = float(entry\_distance.get())

if distance <= 0:

messagebox.showerror("Error", "Please enter a valid distance") return

toll\_fee = 0

type")

# Calculate toll based on vehicle type if vehicle\_type == "Car":

toll\_fee = 0.1 \* distance # Car: $0.1 per mile elif vehicle\_type == "Truck":

toll\_fee = 0.2 \* distance # Truck: $0.2 per mile elif vehicle\_type == "Bus":

toll\_fee = 0.15 \* distance # Bus: $0.15 per mile else:

messagebox.showerror("Error", "Please select a valid vehicle return

# Show the result

label\_result.config(text=f"Total Toll Fee: ${toll\_fee:.2f}") except ValueError:

messagebox.showerror("Error", "Please enter a valid number for distance")

# Create the main window window = tk.Tk()

window.title("Toll Calculation System")

# Create the vehicle type label and options

label\_vehicle\_type = tk.Label(window, text="Select Vehicle Type:") label\_vehicle\_type.pack(pady=10)

vehicle\_type\_var = tk.StringVar(value="Car") # Default vehicle type is "Car"

vehicle\_type\_menu = tk.OptionMenu(window, vehicle\_type\_var, "Car", "Truck", "Bus")

vehicle\_type\_menu.pack(pady=10)

# Create the distance input field

label\_distance = tk.Label(window, text="Enter Distance (in miles):") label\_distance.pack(pady=10)

entry\_distance = tk.Entry(window) entry\_distance.pack(pady=10)

# Button to calculate toll

button\_calculate = tk.Button(window, text="Calculate Toll", command=calculate\_toll)

button\_calculate.pack(pady=10)

# Label to display the result

label\_result = tk.Label(window, text="Total Toll Fee: $0.00") label\_result.pack(pady=20)

# Run the Tkinter event loop

window.mainloop()

# Electronic Toll Collection (ETC)

import tkinter as tk

from tkinter import messagebox import sqlite3

# Database connection setup

conn = sqlite3.connect('etc\_system.db') c = conn.cursor()

# Create tables for storing vehicle and transaction data c.execute('''

CREATE TABLE IF NOT EXISTS vehicles (

id INTEGER PRIMARY KEY AUTOINCREMENT,

vehicle\_id TEXT NOT NULL UNIQUE, vehicle\_type TEXT NOT NULL, balance REAL NOT NULL

)

''')

c.execute('''

CREATE TABLE IF NOT EXISTS transactions ( transaction\_id INTEGER PRIMARY KEY AUTOINCREMENT, vehicle\_id TEXT NOT NULL,

toll\_fee REAL NOT NULL,

date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

) ''')

conn.commit()

# Function to register a new vehicle def register\_vehicle():

vehicle\_id = entry\_vehicle\_id.get() vehicle\_type = vehicle\_type\_var.get() try:

value.")

balance = float(entry\_balance.get()) if balance < 0:

messagebox.showerror("Error", "Balance must be a positive return

# Insert new vehicle into the database

c.execute("INSERT INTO vehicles (vehicle\_id, vehicle\_type, balance) VALUES (?, ?, ?)",

(vehicle\_id, vehicle\_type, balance)) conn.commit()

messagebox.showinfo("Success", "Vehicle registered successfully!")

# Clear fields after registration entry\_vehicle\_id.delete(0, tk.END) entry\_balance.delete(0, tk.END)

except ValueError:

messagebox.showerror("Error", "Please enter a valid balance.")

# Function to calculate toll fee based on vehicle type and distance def calculate\_toll():

vehicle\_id = entry\_vehicle\_id\_calc.get() try:

distance = float(entry\_distance.get()) if distance <= 0:

messagebox.showerror("Error", "Please enter a valid distance") return

# Fetch vehicle data from the database

c.execute("SELECT vehicle\_type, balance FROM vehicles WHERE vehicle\_id=?", (vehicle\_id,))

vehicle = c.fetchone()

if not vehicle:

messagebox.showerror("Error", "Vehicle not found. Please register first.")

return

vehicle\_type, current\_balance = vehicle

# Toll calculation based on vehicle type if vehicle\_type == "Car":

toll\_fee = 0.1 \* distance elif vehicle\_type == "Truck": toll\_fee = 0.2 \* distance

elif vehicle\_type == "Bus": toll\_fee = 0.15 \* distance

else:

messagebox.showerror("Error", "Invalid vehicle type.") return

toll.")

if current\_balance < toll\_fee:

messagebox.showerror("Error", "Insufficient balance for the

else:

# Deduct toll from the balance new\_balance = current\_balance - toll\_fee

c.execute("UPDATE vehicles SET balance=? WHERE vehicle\_id=?",

(new\_balance, vehicle\_id))

conn.commit()

# Record the transaction

c.execute("INSERT INTO transactions (vehicle\_id, toll\_fee) VALUES (?, ?)", (vehicle\_id, toll\_fee))

conn.commit()

messagebox.showinfo("Success", f"Toll fee of ${toll\_fee:.2f} has been deducted.\nNew balance: ${new\_balance:.2f}")

except ValueError:

messagebox.showerror("Error", "Please enter a valid distance.")

# Function to show transaction history def show\_transactions():

vehicle\_id = entry\_vehicle\_id\_trans.get()

c.execute("SELECT transaction\_id, toll\_fee, date FROM transactions WHERE vehicle\_id=?", (vehicle\_id,))

transactions = c.fetchall()

if not transactions:

messagebox.showinfo("No Transactions", "No transactions found for this vehicle.")

return

# Create a new window to display transaction history history\_window = tk.Toplevel(window) history\_window.title("Transaction History")

# Create labels for history table

header = tk.Label(history\_window, text="Transaction ID | Toll Fee | Date")

header.pack(pady=10)

# Display all transactions for trans in transactions:

trans\_label = tk.Label(history\_window, text=f"{trans[0]} |

${trans[1]:.2f} | {trans[2]}") trans\_label.pack()

# Create the main window window = tk.Tk()

window.title("Electronic Toll Collection (ETC) System")

# Vehicle Registration Section frame\_registration = tk.Frame(window) frame\_registration.pack(pady=20)

label\_vehicle\_id = tk.Label(frame\_registration, text="Vehicle ID:") label\_vehicle\_id.grid(row=0, column=0)

entry\_vehicle\_id = tk.Entry(frame\_registration) entry\_vehicle\_id.grid(row=0, column=1)

label\_vehicle\_type = tk.Label(frame\_registration, text="Select Vehicle Type:")

label\_vehicle\_type.grid(row=1, column=0)

vehicle\_type\_var = tk.StringVar(value="Car") # Default vehicle type is "Car"

vehicle\_type\_menu = tk.OptionMenu(frame\_registration, vehicle\_type\_var, "Car", "Truck", "Bus")

vehicle\_type\_menu.grid(row=1, column=1)

label\_balance = tk.Label(frame\_registration, text="Initial Balance ($):") label\_balance.grid(row=2, column=0)

entry\_balance = tk.Entry(frame\_registration) entry\_balance.grid(row=2, column=1)

button\_register = tk.Button(frame\_registration, text="Register Vehicle", command=register\_vehicle)

button\_register.grid(row=3, columnspan=2)

# Toll Calculation Section frame\_toll = tk.Frame(window) frame\_toll.pack(pady=20)

label\_vehicle\_id\_calc = tk.Label(frame\_toll, text="Enter Vehicle ID:") label\_vehicle\_id\_calc.grid(row=0, column=0)

entry\_vehicle\_id\_calc = tk.Entry(frame\_toll) entry\_vehicle\_id\_calc.grid(row=0, column=1)

label\_distance = tk.Label(frame\_toll, text="Enter Distance (in miles):") label\_distance.grid(row=1, column=0)

entry\_distance = tk.Entry(frame\_toll) entry\_distance.grid(row=1, column=1)

button\_calculate\_toll = tk.Button(frame\_toll, text="Calculate Toll", command=calculate\_toll)

button\_calculate\_toll.grid(row=2, columnspan=2)

# Transaction History Section frame\_history = tk.Frame(window) frame\_history.pack(pady=20)

label\_vehicle\_id\_trans = tk.Label(frame\_history, text="Enter Vehicle ID

for Transactions:") label\_vehicle\_id\_trans.grid(row=0, column=0) entry\_vehicle\_id\_trans = tk.Entry(frame\_history)

entry\_vehicle\_id\_trans.grid(row=0, column=1)

button\_show\_transactions = tk.Button(frame\_history, text="Show Transaction History", command=show\_transactions)

button\_show\_transactions.grid(row=1, columnspan=2)

# Run the Tkinter event loop

window.mainloop()

# Close the database connection when done

conn.close()

# Toll Booth Management

import tkinter as tk

from tkinter import messagebox

# Create the main application window root = tk.Tk()

root.title("Toll Booth Management System") root.geometry("600x400")

# Set the background image (you can change this to a valid image path) bg\_image = tk.PhotoImage(file="d:\clg\Sbackground.png") # Replace with your own image path

bg\_label = tk.Label(root, image=bg\_image) bg\_label.place(relwidth=1, relheight=1)

# Function to calculate toll tax based on vehicle type def calculate\_toll():

vehicle\_type = vehicle\_type\_entry.get()

if vehicle\_type.lower() == "car": toll\_fee = 50

elif vehicle\_type.lower() == "truck": toll\_fee = 100

elif vehicle\_type.lower() == "bus": toll\_fee = 80

else:

messagebox.showerror("Error", "Invalid vehicle type") return

messagebox.showinfo("Toll Fee", f"The toll fee for {vehicle\_type} is

${toll\_fee}")

# Create input fields and labels

vehicle\_type\_label = tk.Label(root, text="Enter Vehicle Type (Car/Truck/Bus):", bg="white", font=("Arial", 12)) vehicle\_type\_label.pack(pady=20)

vehicle\_type\_entry = tk.Entry(root, font=("Arial", 12)) vehicle\_type\_entry.pack(pady=10)

# Calculate toll button

calculate\_button = tk.Button(root, text="Calculate Toll", font=("Arial", 12), command=calculate\_toll)

calculate\_button.pack(pady=20)

# Run the main loop root.mainloop()

# Data Management and Storage

import tkinter as tk

from tkinter import messagebox, simpledialog import json

import os

# Initialize the root window root = tk.Tk()

root.title("Data Management and Storage Module") root.geometry("600x400")

# Initialize the data storage (using JSON for persistence) data\_file = "toll\_data.json"

# Function to load data from the file def load\_data():

if os.path.exists(data\_file):

with open(data\_file, "r") as file: return json.load(file)

else:

return []

# Function to save data to the file def save\_data(data):

with open(data\_file, "w") as file: json.dump(data, file)

# Initialize the vehicle data vehicle\_data = load\_data()

# Function to add a new record def add\_record():

vehicle\_type = simpledialog.askstring("Input", "Enter Vehicle Type (Car/Truck/Bus):", parent=root)

toll\_fee = simpledialog.askinteger("Input", "Enter Toll Fee:", parent=root)

if vehicle\_type and toll\_fee:

record = {"vehicle\_type": vehicle\_type, "toll\_fee": toll\_fee} vehicle\_data.append(record)

save\_data(vehicle\_data)

messagebox.showinfo("Success", "Record added successfully!") refresh\_data\_display()

# Function to display data in the listbox def refresh\_data\_display():

data\_listbox.delete(0, tk.END) for record in vehicle\_data:

data\_listbox.insert(tk.END, f"{record['vehicle\_type']} -

${record['toll\_fee']}")

# Function to delete a selected record def delete\_record():

try:

selected\_index = data\_listbox.curselection()[0] vehicle\_data.pop(selected\_index) save\_data(vehicle\_data)

refresh\_data\_display()

messagebox.showinfo("Success", "Record deleted successfully!") except IndexError:

messagebox.showwarning("Warning", "Please select a record to delete.")

# Function to clear all data def clear\_all\_data():

if messagebox.askyesno("Confirm", "Are you sure you want to clear all data?"):

vehicle\_data.clear() save\_data(vehicle\_data) refresh\_data\_display()

messagebox.showinfo("Success", "All data has been cleared!")

# UI Elements

add\_button = tk.Button(root, text="Add Record", font=("Arial", 12), command=add\_record)

add\_button.pack(pady=10)

delete\_button = tk.Button(root, text="Delete Record", font=("Arial", 12), command=delete\_record)

delete\_button.pack(pady=10)

clear\_button = tk.Button(root, text="Clear All Data", font=("Arial", 12), command=clear\_all\_data)

clear\_button.pack(pady=10)

# Listbox to display the vehicle data

data\_listbox = tk.Listbox(root, width=50, height=10, font=("Arial", 12)) data\_listbox.pack(pady=20)

# Refresh the list display with the current data refresh\_data\_display()

# Run the application root.mainloop()

# Reporting and Analytics

import tkinter as tk

from tkinter import messagebox, simpledialog import json

import os

# Initialize the root window root = tk.Tk()

root.title("Reporting and Analytics Module") root.geometry("600x400")

# Initialize the data storage (using JSON for persistence) data\_file = "toll\_data.json"

# Function to load data from the file def load\_data():

if os.path.exists(data\_file):

with open(data\_file, "r") as file: return json.load(file)

else:

return []

# Initialize the vehicle data vehicle\_data = load\_data()

# Function to calculate total revenue def calculate\_total\_revenue():

total\_revenue = sum(record['toll\_fee'] for record in vehicle\_data) messagebox.showinfo("Total Revenue", f"Total toll revenue:

${total\_revenue}")

# Function to calculate average toll fee def calculate\_average\_toll\_fee():

if vehicle\_data:

avg\_toll\_fee = sum(record['toll\_fee'] for record in vehicle\_data)

/ len(vehicle\_data)

messagebox.showinfo("Average Toll Fee", f"Average toll fee:

${avg\_toll\_fee:.2f}") else:

messagebox.showwarning("No Data", "No records available to calculate average.")

# Function to filter records by vehicle type def filter\_by\_vehicle\_type():

vehicle\_type = simpledialog.askstring("Input", "Enter Vehicle Type (Car/Truck/Bus):", parent=root)

if vehicle\_type:

filtered\_records = [record for record in vehicle\_data if record['vehicle\_type'].lower() == vehicle\_type.lower()]

if filtered\_records: display\_filtered\_data(filtered\_records)

else:

messagebox.showwarning("No Records", f"No records found for

{vehicle\_type}.")

# Function to display filtered data in a new window def display\_filtered\_data(filtered\_records):

filtered\_window = tk.Toplevel(root) filtered\_window.title(f"Filtered Data") filtered\_window.geometry("500x300")

listbox = tk.Listbox(filtered\_window, width=50, height=10, font=("Arial", 12))

listbox.pack(pady=20)

for record in filtered\_records:

listbox.insert(tk.END, f"{record['vehicle\_type']} -

${record['toll\_fee']}")

# UI Elements

total\_revenue\_button = tk.Button(root, text="Total Revenue", font=("Arial", 12), command=calculate\_total\_revenue) total\_revenue\_button.pack(pady=10)

average\_toll\_fee\_button = tk.Button(root, text="Average Toll Fee", font=("Arial", 12), command=calculate\_average\_toll\_fee) average\_toll\_fee\_button.pack(pady=10)

filter\_by\_vehicle\_button = tk.Button(root, text="Filter by Vehicle Type",

font=("Arial", 12), command=filter\_by\_vehicle\_type)

filter\_by\_vehicle\_button.pack(pady=10)

# Run the application

root.mainloop()

# System Administration

import tkinter as tk

from tkinter import messagebox, filedialog import shutil

import json import os

# Initialize the root window root = tk.Tk()

root.title("System Administration Module") root.geometry("600x400")

# Initialize the data storage (using JSON for persistence) data\_file = "toll\_data.json"

# Function to load data from the file def load\_data():

if os.path.exists(data\_file):

with open(data\_file, "r") as file: return json.load(file)

else:

return []

# Function to save data to the file def save\_data(data):

with open(data\_file, "w") as file: json.dump(data, file)

# Function to backup the data file def backup\_data():

if os.path.exists(data\_file): backup\_path =

filedialog.asksaveasfilename(defaultextension=".json", filetypes=[("JSON files", "\*.json")])

if backup\_path:

shutil.copy(data\_file, backup\_path) messagebox.showinfo("Backup Successful", f"Backup completed

successfully! Backup saved to: {backup\_path}") else:

messagebox.showwarning("No Data", "No data file found to backup.")

# Function to restore the data from a backup def restore\_data():

backup\_file = filedialog.askopenfilename(filetypes=[("JSON files", "\*.json")])

if backup\_file: try:

shutil.copy(backup\_file, data\_file) messagebox.showinfo("Restore Successful", f"Data restored

successfully from: {backup\_file}") except Exception as e:

messagebox.showerror("Restore Failed", f"Failed to restore

data: {e}")

# Function to clear all data def clear\_all\_data():

if messagebox.askyesno("Confirm", "Are you sure you want to clear all data? This cannot be undone!"):

if os.path.exists(data\_file): os.remove(data\_file)

messagebox.showinfo("Data Cleared", "All data has been cleared successfully.")

# UI Elements

backup\_button = tk.Button(root, text="Backup Data", font=("Arial", 12), command=backup\_data)

backup\_button.pack(pady=10)

restore\_button = tk.Button(root, text="Restore Data", font=("Arial", 12), command=restore\_data)

restore\_button.pack(pady=10)

clear\_button = tk.Button(root, text="Clear All Data", font=("Arial", 12),

command=clear\_all\_data)

clear\_button.pack(pady=10)

# Run the application

root.mainloop()