Report On

Digital Clock

Submitted in partial fulfillment of the requirements of the Course project in Semester IV of Second Year Computer Engineering

By

Shrikant Tathod (Roll No. 48)

Akshay Tiwari (Roll No. 53)

Harsh Upadhyay (Roll No. 54)

Supervisor

Ms. Sneha Mhatre

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

(2023-24)

Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the project entitled “Student Academic Management System” is a bonafide work of "Bhoomika Surve (Roll No. 45), Shreel Thakur (Roll No. 50), Abhishek Tiwari (Roll No. 52)" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester IV of Second Year Computer Engineering.

Ms.Sneha Mhatre
Mentor

Dr Megha Trivedi
Head of Department

Dr. H.V. Vankudre
Principal

ABSTRACT

This Python program implements a digital clock using the Tkinter library for the graphical user interface. It displays the current time in hours, minutes, and seconds, updating every second to reflect the real-time system clock. The program utilizes functions to retrieve the current time, formats it, and updates the display accordingly. Tkinter's **Label** widget is employed to showcase the time in a visually appealing manner. Overall, the digital clock provides a simple yet effective means of displaying real-time information to the user.

CONTENTS:

MODULE DESCRIPTION AND FLOWCHART:

Database Integration: The Student Surveillance System incorporates SQLite as its backend database, offering a lightweight and portable solution for data storage. The system establishes connections to the "students.db" database, which stores comprehensive information about students, their activities, and surveillance data. Within the database, three key tables are created:

1. students: This table holds student details such as roll number, name, gender, branch, academic year, address, and contact number.
2. activities: Manages records of student activities, including entry and exit timestamps, location, and any detected anomalies.
3. alerts: Tracks alerts generated by the system, documenting suspicious behavior, unauthorized access attempts, or other security-related incidents.

User Management: The system provides robust user management capabilities to ensure secure access and authentication:

- Registration: Users can securely register by creating a unique username-password combination, stored encrypted within the database.
- Authentication: The login mechanism verifies user credentials against entries in the users table, granting access upon successful validation.
- Session Management: Secure logout functionality is implemented to terminate active user sessions and prevent unauthorized access.

Surveillance Monitoring: The Student Surveillance System offers real-time monitoring of student activities, enabling administrators to ensure campus safety and security:

- Live Feed Analysis: Utilizes computer vision algorithms to analyze live video feeds from surveillance cameras, detecting and identifying individuals and their actions.
- Anomaly Detection: Detects anomalies in student behavior, such as loitering in restricted areas, overcrowding, or unusual movements, and generates alerts for immediate action.
- Activity Logging: Records all student activities, including entry and exit times, locations visited, and any flagged incidents, providing a comprehensive audit trail for security analysis.

Alert Management: The system features an alert management system to handle and respond to security-related incidents effectively:

- Alert Generation: Automatically generates alerts for detected anomalies or suspicious activities, providing detailed information and timestamps for each incident.
- Alert Notification: Alerts are promptly communicated to security personnel via email, SMS, or through the system's dashboard, ensuring swift response and resolution.
- Alert History: Maintains a log of all generated alerts, enabling administrators to review past incidents, track trends, and improve security protocols.

Graphical User Interface (GUI): The GUI of the Student Surveillance System is designed to be user-friendly and intuitive, facilitating seamless navigation and interaction:

- Interface Components: The GUI incorporates various elements such as menus, buttons, input fields, and status indicators, organized in a logical layout for ease of use.
- Visual Feedback: Provides visual feedback and notifications to users, such as alert pop-ups, status messages, and interactive maps displaying student movements.
- Customization Options: Offers customization options for interface themes, display preferences, and user permissions, allowing administrators to tailor the system to their specific requirements.

Connection Management and System Exit: The system includes functionality to manage database connections and ensure proper resource management:

- Connection Management: Establishes and maintains database connections securely, handling exceptions and errors gracefully to prevent data loss or corruption.
- System Exit: Provides a reliable mechanism for exiting the application, closing database connections, and saving any unsaved data or configurations to ensure data integrity and system stability.


Initialize Window
Create Labels (Time, Date, Day)
Create Alarm Entry
Create Set Button
Check Current Time and Alarm Time
Alarm Time Reached?
No
Main Loop Update Time, Date, and Day Labels
Yes
Play Alarm Sound
Show Alarm Message
Stop Alarm Sound
Reset Alarm Entry
Main Loop Update Time, Date, and Day Labels

SOFTWARE REQUIRED:

The Student Surveillance System requires the following software:

1. Python: The core programming language used to develop the system.
2. PyQt5: A Python binding for the Qt toolkit, used to create the graphical user interface (GUI) of the application.
3. SQLite: A lightweight and portable database management system used to store student and surveillance data.
4. Integrated Development Environment (IDE): Any Python IDE such as PyCharm, Visual Studio Code, or Spyder can be used for coding and debugging.
5. Operating System: The system should be compatible with Windows, macOS, or Linux distributions to run Python and the GUI application smoothly.

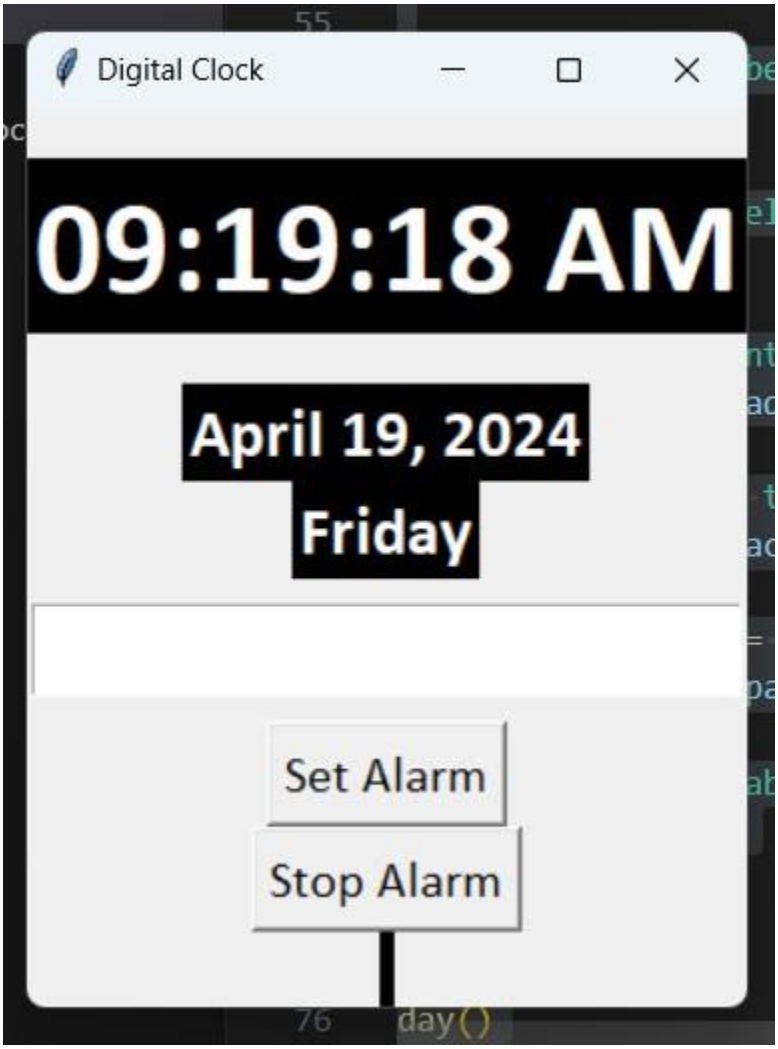
PROGRAM:

```
import tkinter as tk
from time import strftime
from tkinter import messagebox
import winsound

# Global variable to track if the alarm is ringing
alarm_ringing = False

def set_alarm():
    alarm_time = entry_alarm.get()
    if alarm_time:
        alarm_label.config(text=f"Alarm set for {alarm_time}")
    else:
        alarm_label.config(text="Please enter a valid time!")

def check_alarm():
    global alarm_ringing
    current_time = strftime('%H:%M')
    if current_time == entry_alarm.get() and not alarm_ringing:
        alarm_ringing = True
        play_alarm_sound()  # Play the alarm sound immediately
        messagebox.showinfo("Alarm", "Wake up!")
        alarm_label.config(text="")

def play_alarm_sound():
    global alarm_ringing
    alarm_ringing = True
    winsound.Beep(1000, 1000)  # Example: Beep sound for 1 second at 1000
Hz
    root.after(1000, play_alarm_sound)  # Repeat the alarm sound every
second

def stop_alarm():
    global alarm_ringing
    alarm_ringing = False
    winsound.PlaySound(None, winsound.SND_PURGE)  # Stop the alarm sound

def time():
    string = strftime('%H:%M:%S %p')
    label_time.config(text=string)
    label_time.after(1000, time)
    check_alarm()

def date():
    string = strftime('%B %d, %Y')
    label_date.config(text=string)

def day():
```

```
    string = strftime('%A')
    label_day.config(text=string)

root = tk.Tk()
root.title("Digital Clock")

label_time = tk.Label(root, font=('calibri', 40, 'bold'), bg='black',
fg='white')
label_time.pack(pady=20)

label_date = tk.Label(root, font=('calibri', 20, 'bold'), bg='black',
fg='white')
label_date.pack()

label_day = tk.Label(root, font=('calibri', 20, 'bold'), bg='black',
fg='white')
label_day.pack()

entry_alarm = tk.Entry(root, font=('calibri', 20))
entry_alarm.pack(pady=10)

set_alarm_button = tk.Button(root, text="Set Alarm", font=('calibri', 15),
command=set_alarm)
set_alarm_button.pack()

stop_alarm_button = tk.Button(root, text="Stop Alarm", font=('calibri',
15), command=stop_alarm)
stop_alarm_button.pack()

alarm_label = tk.Label(root, font=('calibri', 15), bg='black', fg='white')
alarm_label.pack()

time()
date()
day()

root.mainloop()
```

RESULTS AND CONCLUSION:

RESULTS:



CONCLUSION:

In conclusion, the provided flowchart outlines the logical flow of a digital clock program implemented in Python using the Tkinter library for the graphical user interface. The program initializes a window and creates labels for displaying the current time, date, and day of the week. It also includes an entry field for setting an alarm time and buttons for setting and stopping the alarm.

The main loop of the program continuously updates the current time label, checks if the alarm time has been reached, and updates the date and day labels accordingly. If the alarm time matches the current time, the program triggers an alarm sound, displays a message, and stops the alarm when dismissed.

Overall, the flowchart illustrates a clear sequence of steps involved in running the digital clock program, from initialization to ongoing time updates and alarm checks.