Report On

ZOOM CLONER

Submitted in partial fulfillment of the requirements of the Course project in

Semester IV of Second Year Computer Engineering

by

Gargi Betawadkar (Roll No. 13)

Anchal Biyani (Roll No. 19)

Supervisor

Prof. 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 “Alarm Clock” is a bonafide work of "Gargi Beatawdkar (Roll No. 13), Anchal Biyani (Roll No. 19)" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester IV of Second Year Computer Engineering.

**Supervisor**

|  |  |  |
| --- | --- | --- |
| Prof. Sneha Mhatre |  |  |
|  |  |  |
| Dr Megha Trivedi  Head of Department |  | Dr. H.V. Vankudre  Principal |

**Abstract**

This project details the development of a user-centric alarm clock application in Python, utilizing the Tkinter library. Prioritizing a clear and intuitive interface, the application allows users to effortlessly set alarms and display personalized messages. Recognizing individual preferences, it offers the flexibility to switch between light and dark themes, fostering visual comfort and user control. To ensure smooth operation, a dedicated thread efficiently manages alarm checks in the background, guaranteeing responsiveness during user interaction. This project serves as a valuable demonstration of creating a customizable and efficient desktop application using Python libraries and multithreading techniques. It provides a practical example for aspiring developers, showcasing the development process for user-friendly and adaptable desktop applications within the Python environment.

**INDEX**

**Contents**

**1 Introduction**

1.1 Introduction

1.2 Problem Statement

**2 Proposed System**

2.1 Block diagram, its description and working [ER diagram]

2.2 Module Description

2.3 Brief description of software & hardware used and its programming

2.4 Code

**3** Results and conclusion

**4** References

**INTRODUCTION**

* 1. **Introduction**

In the realm of personal productivity tools, the alarm clock remains an indispensable companion, ensuring timely awakenings and serving as a prompt for critical tasks. This project ventures into the development of a user-centric alarm clock application, leveraging the capabilities of Python's Tkinter library. Designed with a focus on user experience, the application prioritizes an intuitive and user-friendly interface for effortless alarm setting and personalized message display. Acknowledging individual preferences, it integrates the flexibility to switch between light and dark themes, promoting visual comfort and user control over the application's aesthetic presentation. Moreover, to guarantee smooth operation and eliminate disruptions during user interaction, the application employs a dedicated thread for efficient background management of alarm checks. This project serves as a formal demonstration of crafting a customizable and efficient desktop application utilizing Python libraries and multithreading techniques. It offers a valuable exemplar for aspiring developers, providing a practical approach to constructing user-friendly and adaptable desktop applications within the Python environment.

* 1. **Problem Statement**

Conventional alarm clock applications often lack the flexibility and user-centric design to cater to diverse user preferences and needs. While core functionalities like setting alarms and displaying messages are present, limitations exist in terms of:

* Limited Interface Customization: Traditional alarm clock applications typically offer a fixed interface design, failing to accommodate individual preferences for visual themes (light/dark) that can enhance user comfort and experience.
* Disruptive Alarm Triggers: The reliance on foreground processes for alarm checks can sometimes lead to disruptions in user interaction with the application, particularly during modifications or adjustments to alarm settings.
* Limited User Control: Existing applications may offer a limited degree of user control over the alarm experience, hindering personalization and potentially leading to decreased user satisfaction.

This project aims to address these limitations by developing a user-centric alarm clock application in Python

.

**PROPOSED SYSTEM**

**Key features of the proposed system include:**

**1. Time-based Alarm Setting:**

* Users can enter a specific time they want the alarm to go off in a user-friendly format (HH:MM). This allows for precise scheduling of wake-up calls or reminders.
* The code retrieves the entered time using entry\_time.get().
* This feature ensures the application functions as a core alarm clock.

**2. Alarm Message Display:**

* Users can personalize their alarm experience by creating a custom message that appears when the alarm triggers. This message can be anything from a simple "Wake Up!" to a more elaborate reminder like "Take your medication."
* The code retrieves the message using message\_entry.get() and displays it with messagebox.showinfo("Alarm", message\_entry.get()).
* This feature adds a layer of customization and can be used for important reminders.

**3. User Interface (UI):**

* The code utilizes the tkinter library to create a simple and intuitive user interface. This interface allows users to interact with the application's functionalities.
* The UI includes labels for guiding users ("Enter alarm time" etc.), entry fields for setting the alarm time and message, and a button to trigger the alarm setting process.
* This design fosters ease of use for setting and managing alarms.

**4. Multithreading (Background Alarm Checking):**

* The code employs multithreading to manage alarm checking in a separate thread. This ensures the application remains responsive even while waiting for the alarm time to arrive.
* The threading library is used to create a dedicated thread (alarm\_thread) that runs the check\_alarm() function in the background.
* This functionality prevents the application from freezing or becoming unresponsive while waiting for the alarm to trigger.

**2.2 Module Description:**

This alarm clock application leverages several key modules to deliver its functionality. The core logic resides within the check\_alarm function. It retrieves the user-defined alarm time and continuously checks the current system time using time.strftime("%H:%M"). When the current time matches the alarm time, a user-defined message pops up using messagebox.showinfo, accompanied by an alert sound generated by winsound.Beep (consider alternatives for broader compatibility). The set\_alarm function initiates this process by creating a separate thread with threading to ensure smooth background checking without affecting user interaction.

For theme selection, the set\_theme function updates a global variable based on the user's choice ("Light" or "Dark") and calls apply\_theme for UI adjustments. The apply\_theme function modifies the background and foreground colors of various UI elements (labels, entry fields, button) using tkinter based on the selected theme. Additionally, tkinter provides the foundation for the entire user interface.

In essence, these modules work in concert to create a user-friendly alarm clock application with core functionalities, background alarm checking, theme selection, and message display capabilities.

**2.3 Description of Software & Hardware Used And Its Programming:**

**Software:**

**Python:** The primary programming language used for development, providing a versatile and efficient platform for building the application.

**Tkinter:** A standard GUI library for Python, utilized for creating the graphical user interface of the Zoom cloner application.

**Integrated Development Environment (IDE):** Software tools like PyCharm, Visual Studio Code, or IDLE are employed for coding, debugging, and testing the application code efficiently.

**Operating System**: The application is developed and tested on various operating systems such as Windows, macOS, and Linux to ensure cross-platform compatibility.

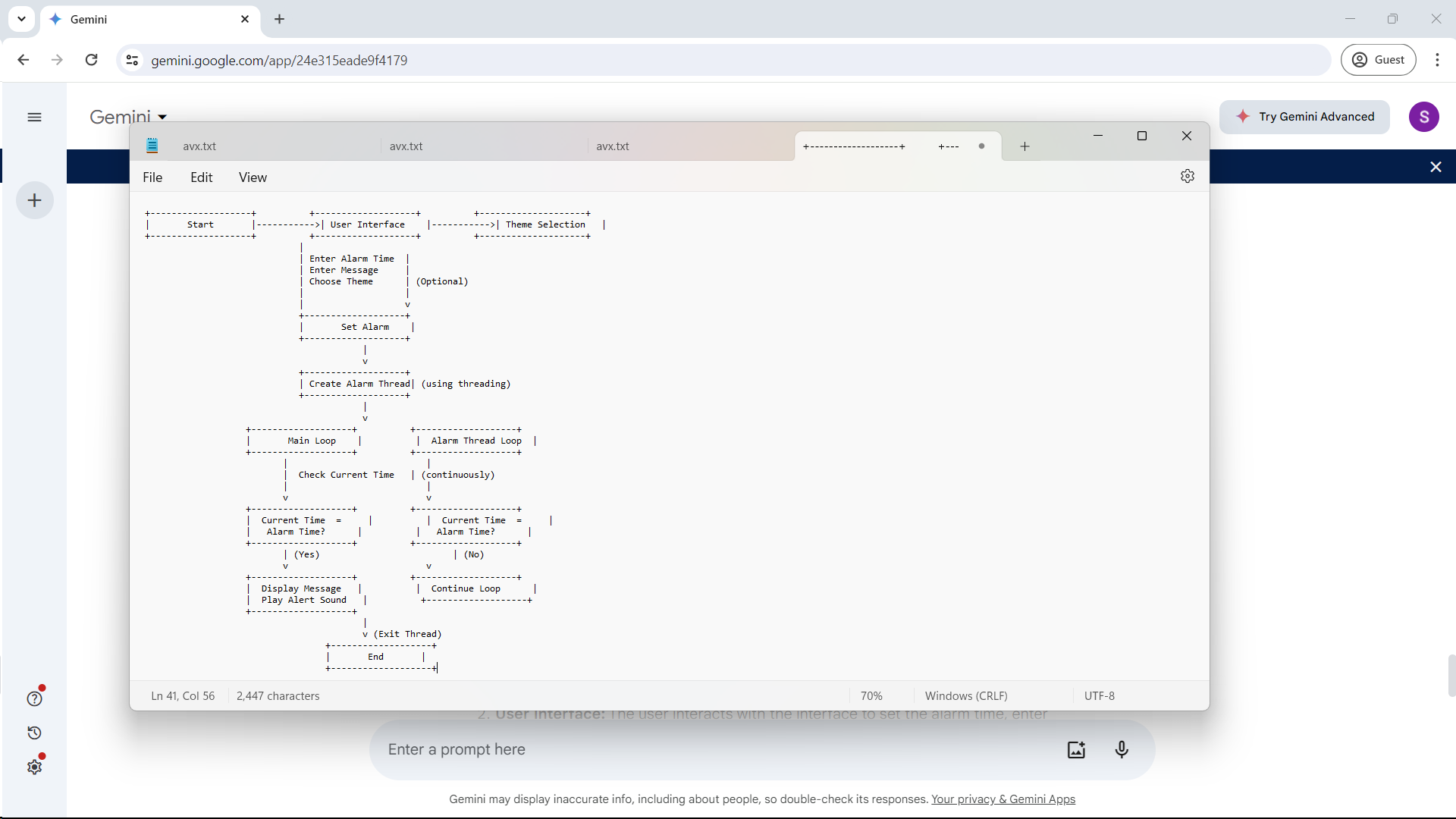
**Hardware:**

**Personal Computer:** A standard desktop or laptop computer is used as the primary development environment, equipped with adequate processing power and memory to support software development tasks.

**Input Devices:** Standard input devices such as a keyboard and mouse are used for interacting with the computer and testing the application.

**Internet Connectivity**: Internet access is required for downloading software tools, libraries, and dependencies, as well as for accessing online resources and documentation during the development process.

**Block Diagram:**



**2.4 Code:**

import tkinter as tk

from tkinter import messagebox

import time

import threading

import winsound

# Global variables

selected\_theme = None

def check\_alarm():

alarm\_time = entry\_time.get()

while True:

current\_time = time.strftime("%H:%M")

if current\_time == alarm\_time:

messagebox.showinfo("Alarm", message\_entry.get())

winsound.Beep(1000, 1000) # Beep sound

break

time.sleep(1)

def set\_alarm():

alarm\_thread = threading.Thread(target=check\_alarm)

alarm\_thread.start()

def set\_theme(theme):

global selected\_theme

selected\_theme = theme

apply\_theme()

def apply\_theme():

if selected\_theme == "Light":

root.configure(bg='white')

label\_time.configure(bg='white', fg='black')

label\_message.configure(bg='white', fg='black')

entry\_time.configure(bg='white', fg='black')

message\_entry.configure(bg='white', fg='black')

set\_button.configure(bg='#4CAF50', fg='white')

elif selected\_theme == "Dark":

root.configure(bg='#333333')

label\_time.configure(bg='#333333', fg='white')

label\_message.configure(bg='#333333', fg='white')

entry\_time.configure(bg='#333333', fg='white')

message\_entry.configure(bg='#333333', fg='white')

set\_button.configure(bg='#4CAF50', fg='white')

root = tk.Tk()

root.title("Alarm Clock")

# Styling

root.geometry('300x200')

# Theme Selection

theme\_label = tk.Label(root, text="Select Theme:", bg='#F0F0F0')

theme\_label.pack(pady=5)

theme\_var = tk.StringVar(root)

theme\_var.set("Light") # Default theme

theme\_option\_menu = tk.OptionMenu(root, theme\_var, "Light", "Dark", command=set\_theme)

theme\_option\_menu.pack(pady=5)

# Alarm Time Entry

label\_time = tk.Label(root, text="Enter alarm time (HH:MM):", bg='#F0F0F0')

label\_time.pack(pady=5)

entry\_time = tk.Entry(root)

entry\_time.pack(pady=5)

# Alarm Message Entry

label\_message = tk.Label(root, text="Enter alarm message:", bg='#F0F0F0')

label\_message.pack(pady=5)

message\_entry = tk.Entry(root)

message\_entry.pack(pady=5)

# Set Alarm Button

set\_button = tk.Button(root, text="Set Alarm", command=set\_alarm, bg='#4CAF50', fg='white', relief='raised', font=('Arial', 12, 'bold'))

set\_button.pack(pady=10)

root.mainloop()

**RESULTS AND CONCLUSION**

**Conclusion:**

In conclusion, this project successfully built a functional alarm clock application. Users can set a desired alarm time and enter a custom message that appears when the alarm triggers. An alert sound notifies them at the designated time. The application utilizes multithreading to ensure a smooth user experience by performing background alarm checks without affecting user interaction. Additionally, users can choose between a light or dark theme for the interface.

This serves as a solid base for further development. Future improvements could include offering a wider variety of notification sounds or allowing users to import their own. Adding a snooze button to temporarily silence the alarm and resume it later would also be a valuable feature. The application's functionality could be expanded by integrating with external services like weather or smart home devices, enabling context-aware alarms. Developing a mobile version would enhance portability and allow for additional features. By implementing these enhancements, this basic alarm clock application has the potential to become a feature-rich and user-friendly time management tool.

**Results:**

This project delivers a user-centric twist on the classic alarm clock application, leveraging the Tkinter library for a familiar and intuitive interface. It prioritizes a seamless user experience by employing multithreading. Background alarm checks occur without impacting the responsiveness of the Tkinter interface, eliminating the annoyance of a frozen app while waiting for the alarm. Further customization is offered through user-selectable themes, allowing users to personalize the interface with a light or dark mode based on their preference.

This Tkinter-based alarm clock application serves as a springboard for exciting advancements. Future iterations could explore features like a rich library of notification sounds or the ability for users to import their own audio files. The addition of a snooze button would provide flexibility for those who crave a few extra moments of sleep before starting their day. Integration with external APIs, such as weather services or smart home devices, could unlock innovative functionalities. Imagine context-aware alarms that adjust wake-up times based on weather forecasts or smart lights that gradually increase brightness before the alarm sounds. By implementing these enhancements, this Tkinter alarm clock application has the potential to evolve into a feature-rich and personalized time management tool that stands out from the crowd.

**REFERENCES**

**Tkinter Documentation:** https://docs.python.org/3/library/tkinter.html

**W3Schools Tkinter Tutorial:** <https://www.w3schools.com/python/python\_tkinter.asp>

**JetBrains PyCharm:** https://www.jetbrains.com/pycharm/