Smart Alarm Clock Project – Rpi Python Code Documentaion

Project Overview:
The Alarm Clock project is a Python-based application designed to allow users to set and manage alarms using a web interface. The application utilizes Flask, a micro web framework, for handling HTTP requests and providing a user interface. Users can set alarms by specifying the date, hour, minute, and optionally upload a custom ringtone. Alarms can be toggled on or off, and the system will trigger the alarms at the specified times.

Structure:

1. **Imports:**
    - Imports necessary Python modules and libraries for the application.
2. **Flask App Setup:**
    - Initializes a Flask application instance.
    - Sets up configurations for file uploads.
3. **Alarm Condition Check Function:**
    - Defines a function **alarm_condition_met()** to check if the current time matches the specified alarm time and date.
4. **Audio Streaming Functions:**
    - **start_audio_stream()**: Starts streaming audio from the phone to the Raspberry Pi via Bluetooth.
    - **stop_audio_stream(mac_address)**: Stops audio streaming and disconnects from the Bluetooth device.
    - **connect_device(mac_address)**: Connects to a Bluetooth device using its MAC address.
5. **Ringtone Playback Function:**
    - Defines a function **play_alarm_ringtone()** to play the alarm ringtone from memory.
6. **Flask Routes:**
    - Defines a route for handling both GET and POST requests to the root URL ("/").
    - Handles setting new alarms and toggling existing alarms based on form submissions.
    - Generates a webpage to display current and upcoming alarms.
7. **Alarm Handling Function:**
    - Defines a function **handle_alarms()** to manage the alarm logic.
    - Checks if alarm conditions are met and triggers alarm actions accordingly.

   - Handles audio streaming, ringtone playback, and Bluetooth device connections.

8. **Main Function:**
   - Starts the Flask server and the alarm handling thread in separate threads.

Line-by-Line Breakdown:

import os
import time
import subprocess
import threading from flask import Flask, request, render_template

- Imports necessary modules and libraries

app = Flask(__name__)

- Initializes a Flask application instance.

alarms = {}

- Initializes an empty dictionary to store alarm details.

UPLOAD_FOLDER = '/home/Edac145/Projects/alarmclock/alarm_set_v0.1/upload'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

- Sets the upload folder for file uploads in the Flask app.

def alarm_condition_met(alarm_hour, alarm_minute, alarm_date): # Function to check if the current time matches the alarm time and date

This function checks whether the current time matches the specified alarm time and date. It takes three parameters:

- **alarm_hour**: The hour of the alarm.
- **alarm_minute**: The minute of the alarm.
- **alarm_date**: The date of the alarm.

It uses the **time.localtime()** function to get the current time and date. Then it compares the current hour, minute, and date with the specified alarm time and date. If they match, it returns **True**; otherwise, it returns **False**.

def start_audio_stream(): # Function to start streaming audio from phone to Raspberry Pi via Bluetooth .

This function starts streaming audio from a phone to a Raspberry Pi via Bluetooth. It executes a subprocess using **subprocess.run()** to set the default audio sink to a Bluetooth device.

def stop_audio_stream(mac_address): # Function to stop audio streaming and disconnect from Bluetooth device

This function stops audio streaming and disconnects from a Bluetooth device. It takes one parameter:

- **mac_address**: The MAC address of the Bluetooth device to disconnect from.

It uses **subprocess.run()** to run the **bluetoothctl** command with the **disconnect** argument and the provided MAC address.

def connect_device(mac_address): # Function to connect to a Bluetooth device

This function connects to a Bluetooth device using its MAC address. It takes one parameter:

- **mac_address**: The MAC address of the Bluetooth device to connect to.

Similar to **stop_audio_stream()**, it uses **subprocess.run()** to run the **bluetoothctl** command with the **connect** argument and the provided MAC address.

def play_alarm_ringtone(ringtone_path, num_times, interval): # Function to play the alarm ringtone from memory

This function plays the alarm ringtone from memory. It takes three parameters:

- **ringtone_path**: The path to the alarm ringtone file.
- **num_times**: The number of times to play the ringtone.
- **interval**: The interval (in seconds) between each play of the ringtone.

It uses a loop to play the ringtone the specified number of times. Inside the loop, it executes a subprocess using **subprocess.run()** to play the ringtone file with **mpg123**. After each play, it pauses for the specified interval using **time.sleep()**.

```
@app.route('/', methods=['GET', 'POST'])

def index():

  # Function to handle requests to the root URL
  if request.method == 'POST':
    # Check if the form was submitted for setting new alarms
    if 'name' in request.form:
      # Handle setting new alarms
    else:  # Handle toggling alarms
      # Handle toggling alarms
  # Generate a page to display current and upcoming alarms
```

Defines a route for the root URL ("/"). When a request is made to this URL, the index() function is called.

Checks if the request method is POST.
If the request method is POST, it checks if the form was submitted for setting new alarms or for toggling existing alarms.

If the form was submitted for setting new alarms, it processes the form data and sets the alarm accordingly.

If the form was submitted for toggling alarms, it toggles the active state of the alarms based on the form data.

Finally, it generates a webpage to display the current and upcoming alarms.

def handle_alarms(): # Function to manage alarm logic

This function manages the alarm logic. It continuously checks if alarm conditions are met and triggers alarm actions accordingly. It handles audio streaming, ringtone playback, and Bluetooth device connections/disconnections.

def main(): # Main function to start Flask server and alarm handling

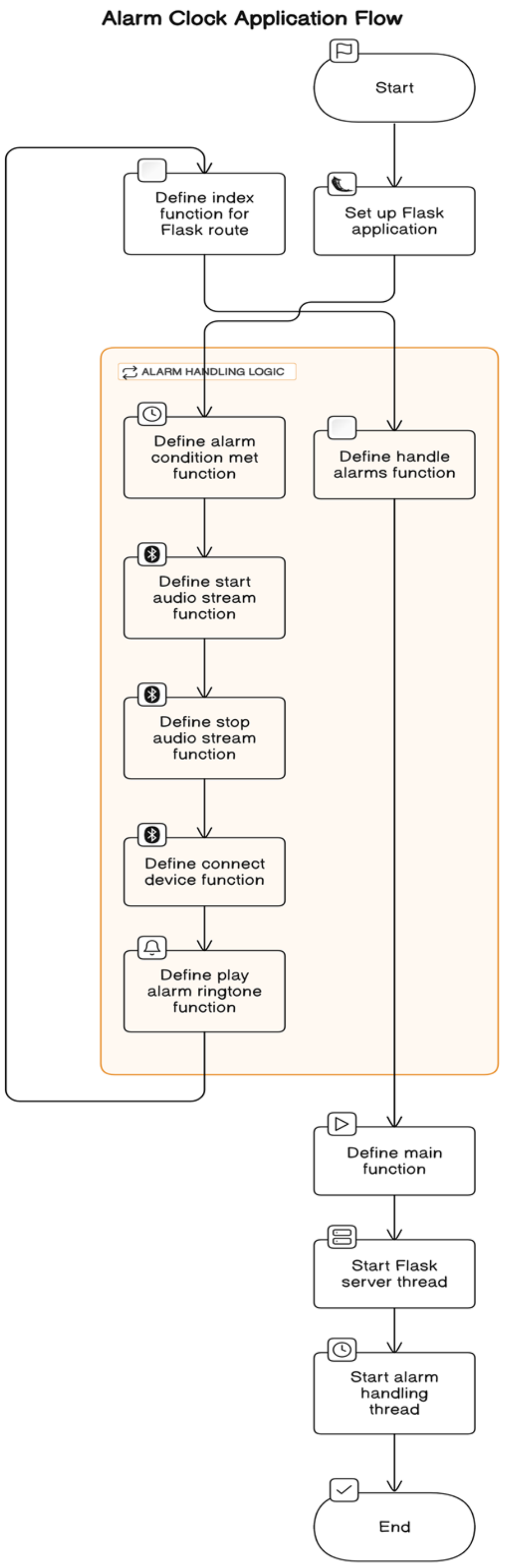
This function serves as the main entry point of the script. It starts the Flask server and the alarm handling thread in separate threads.

These functions collectively implement the alarm clock functionality with a web interface using Flask. Each function serves a specific purpose, contributing to the overall functionality of the application.

if __name__ == "__main__": main()

- Calls the main function if the script is run directly.

Flow chart:


Alarm Clock Application Flow
Start
Define index function for Flask route
Set up Flask application
ALARM HANDLING LOGIC
Define alarm condition met function
Define handle alarms function
Define start audio stream function
Define stop audio stream function
Define connect device function
Define play alarm ringtone function
Define main function
Start Flask server thread
Start alarm handling thread
End