**Activity Tracker**

Here is an detail description about **Activity Tracker** that monitors user interactions (mouse and keyboard activities), takes periodic screenshots, uploads activity logs and screenshots to AWS S3, and handles error like battery monitoring to pause tracking when the battery is low. It also uses a system tray icon for user interaction.

**Detailed Breakdown**

1. **Imports and Setup:**
   * Various modules are imported to handle tasks such as logging, GUI, system tray icon creation, mouse/keyboard tracking, AWS S3 interaction, and battery monitoring:
     + logging: For logging activities to a file.
     + pyautogui: For taking screenshots.
     + boto3: For interacting with AWS S3.
     + pynput: For listening to mouse and keyboard inputs.
     + pystray: For creating a system tray (taskbar) icon.
     + PIL (Python Imaging Library): For image manipulation (compressing screenshots).
     + psutil: For battery status monitoring.
     + tkinter: For GUI (e.g., configuration window).
2. **Global Variables:**
   * is\_active: A flag to track whether the user is active (e.g., moving the mouse or pressing keys).
   * **Thresholds for detecting "script-like" behavior** (unusual speed, straight-line mouse movement, or rapid key presses):
     + MOUSE\_SPEED\_THRESHOLD: Defines the maximum mouse speed considered "normal."
     + KEYPRESS\_THRESHOLD: Defines the minimum time between consecutive key presses for them to be considered rapid.
     + STRAIGHT\_LINE\_THRESHOLD: A tolerance for linear mouse movements that might indicate scripted activity.
   * last\_mouse\_pos and last\_mouse\_time: To calculate the speed of mouse movement.
   * last\_keypress\_time: To detect rapid key presses.
3. **System Tray Icon Creation:**
   * The function create\_icon\_image creates a simple blue square icon using the PIL library for the system tray.
   * The function on\_quit handles the quit operation for the system tray menu.
4. **Activity Logging:**
   * The log\_activity function logs user actions (mouse movements, clicks, key presses) with a timestamp to a log file (activity\_log.txt) and uploads the log to an AWS S3 bucket.
   * Activity types include:
     + Mouse Move, Mouse Click, Mouse Scroll
     + Key Pressed
   * Additionally, the script flags suspicious activities as "Scripted Activity Detected" based on threshold conditions.
5. **Mouse Activity Tracking:**
   * The on\_move, on\_click, and on\_scroll functions are triggered by mouse events:
     + on\_move: Calculates mouse speed and checks if the movement is too fast or too straight (indicating potential scripted behavior).
     + on\_click and on\_scroll: Log mouse clicks and scroll events.
6. **Keyboard Activity Tracking:**
   * The on\_press function tracks key presses and logs them. It also checks for rapid key presses by comparing the time between consecutive key presses.
7. **System Tray and Menu:**
   * The application creates a system tray icon with an option to quit the application. This is handled using the pystray library.
8. **ActivityTracker Class:**
   * This class handles screenshot capture, compression, and upload to AWS S3. It has the following responsibilities:
     + **capture\_and\_compress\_screenshot**: Takes a screenshot, compresses it, and uploads it to S3.
     + **compress\_image**: Compresses the screenshot to a smaller size for efficient upload.
     + **upload\_file\_to\_s3**: Uploads files (screenshots and logs) to AWS S3.
     + **upload\_log\_to\_s3**: Uploads the activity log to AWS S3.
     + **is\_connected\_to\_internet**: Checks if the device is connected to the internet before attempting uploads.
     + **capture\_screenshots\_periodically**: Periodically takes screenshots at the specified interval (screenshot\_interval).
     + **check\_battery\_and\_manage\_tracking**: Monitors the battery level and pauses or resumes activity tracking based on the battery percentage (pauses below 20%, resumes above 40%).
9. **Screenshot Capture and Upload:**
   * Screenshots are captured using pyautogui.screenshot(), then compressed and saved in memory using io.BytesIO (to avoid file I/O).
   * User can also enable and disable screenshot capture.
   * Screenshots are uploaded directly to AWS S3 using the upload\_file\_to\_s3 method, where they are stored under the screenshots/ folder with a timestamped name.
10. **Battery Monitoring:**
    * Using psutil.sensors\_battery(), the script monitors the battery percentage and automatically pauses or resumes activity tracking based on the battery level.
    * If the battery is below 20%, tracking is paused; if it's above 40%, tracking resumes.
11. **Running the Application:**
    * The start\_application function initializes the **ActivityTracker** object and starts several background threads to handle:
      + Activity tracking (mouse and keyboard events).
      + Periodic screenshot capture.
      + Battery monitoring.
    * The application also starts the **system tray icon** to allow the user to interact with the program and quit if needed.
12. **Multithreading:**
    * Threads are used to run background tasks (e.g., screenshot capture, battery monitoring) in parallel to the main application, ensuring the program remains responsive while performing these operations.
13. **User Interface (Tkinter GUI):**
    * The script also integrates a **Tkinter** window (referred to as ConfigWindow in the code), although the class for it (ConfigWindow) isn't fully defined in the provided code. This window would presumably allow the user to configure settings (like AWS S3 bucket name or screenshot intervals).
14. **Error Handling:**
    * The script includes error handling for various AWS S3 upload issues, such as missing credentials, network issues, or permission errors. It also handles network errors when uploading logs or screenshots, queuing the uploads if the device is offline.

**Code Flow:**

In the making of this project I have made 8 different files each file is numbered from 1 to 8. The each successive file is added some more functionalities with new function.

At the end of file I used threading for parallely execution of the function and at the most last part I execute the code. The basic workflow of each file is described below:

1. A Basic Code which makes the background and save the activity in System: This is the starting of the background application. I used pystray library for creating system tray. The agent then run into background and create log of all activity of mouse and keybord and save the log file to the current working directory.
2. Updated code which takes screenshot at proper interval and saved to cloud: Now I update the code as it will take screenshots at proper interval and save them to the cloud.
3. Saving both activity and screenshot to cloud: Extended code which can upload screenshot to the cloud as well as the activity\_log file.
4. Updating previous part with a user defined screeshot interval and saving it to the cloud: Next functionality is and now user can define his own interval of screenshots.
5. Updating the code for error handling like no internet connection: The agent will check the internet connection. If the connection not found then it will queued the data to upload.
6. Updated code for Compressing the image for efficient memory usage: To reduce the memory usage the agent will now compress the screenshot and then upload it to the cloud.
7. Updating the code for Low Battery detection: If the system has low battery then it will shut down all the activities and it will check at regular interval that if battery is charged then restart the agent.
8. Suspicious Activity Enhancement: The agent will now define the suspicious activity and flag it in the log file also a new feature is added that when the screenshot is uploaded to the cloud then it will delete from the system memory.
9. Activity\_log: At last I have added a sample Activity log of my system.

**Key Functionalities Summary:**

1. **Monitoring User Activity:** The script tracks mouse and keyboard events, detecting unusual patterns (e.g., rapid movements or scripted behaviors).
2. **System Tray Icon:** Allows user to interact with the application through a system tray icon.
3. **Screenshots:** Periodic screenshots are taken, also user can define the interval of the screenshot and then compressed, and uploaded to AWS S3.
4. **Activity Logging:** Logs user activities to a file in current working directory.
5. **Data Upload:** The data like screenshot and log file is uploading to the cloud at proper itterval.
6. **Error Handling and Resilience:** Code can also handle error like if there is no intenet connection it will queue the upload and retry after proper interval.
7. **Memory and Performance**: The agent is lightweight, with minimal impact on system resources also I used python inbuilt libraries as I can. In the agent memory optimization techniques is implemented to manage the application’s footprint, especially when handling large files or running for extended periods. Also when the activities are uploaded to the cloud then they are automatically deleted from the system.
8. **File Size Management:** The code will compress screenshots before uploading to reduce file size. This process will Implement automatically.
9. **Modular and Object-Oriented Design:** The application Structure is using a modular and object-oriented approach. The code is well-organized, with clear separation of concerns. Also the code is commented extensively to explain the purpose of each module and function.
10. **Battery Management:** Pauses tracking if the battery level is too low like >20 and resumes it when sufficient say <40.
11. **Time Zone Management:** The log file is storing the data with the proper time stamp.