Tehnical University of Cluj-Napoca

Faculty of Electronics , Telecommunications and Information Technology

CPU Real-Time Thermal Monitoring System

Computer Aided Graphics

Coordinator:

Conf. Dr. Ing. Mihaela CIRLUGEA

Student:

Cadar Alex-Dumitru

Group : 2021

2. Project Overview

Core Purpose :

This project is a **Real-Time CPU Thermal Monitor** built in MATLAB. Its main goal is to provide an easy-to-use dashboard that tracks the temperature of the processor as it happens. By connecting MATLAB to a local hardware server, the app transforms raw sensor data into clear, professional visual graphics.

How it Works :

The system connects to a **JSON web server** provided by Libre Hardware Monitor. It uses a specialized search function to find the "Core Average" temperature within a complex list of hardware sensors. Once the data is retrieved, the app updates two different types of visuals: a live Line Plot that shows the temperature history and a **Linear Gauge** for instant status checks.

User Interaction and Safety:

What makes this tool unique is its interactivity. Users are not just observers; they can actively change the **Alert Limit** and the **Refresh Speed** while the monitor is running. This is handled by a mathematical safety check: the app compares the live temperature to the user's limit. If the CPU gets too hot, the interface instantly

turns red to warn the user, ensuring the hardware stays within safe operating conditions.

3. Mathematical Model

The Safety Logic:

The monitoring system uses a basic comparison to decide if the computer is safe or at risk. This is based on the following relationship:

- **Normal State:** Current Temp < Limit (Interface remains blue)
- **Alert State:** Current Temp >= Limit (Interface turns red)

The user provides the **Limit** value through the GUI, and the system compares it to the live sensor data at every refresh cycle.

The Safety Index

To show how much of the "thermal budget" is being used, the app calculates a percentage using this formula:

$$Saf\,ety\ Index = \left(\frac{Current\ Temp}{Limit} \right) \times\ 100$$

If this index hits **100%** or higher, the program's logic triggers the visual alarm. This is a linear calculation that updates instantly whenever the user changes the limit in the edit box.

Statistical Calculations

As the program runs, it stores every temperature reading in a list. It then uses three standard formulas to fill the Statistics panel:

Maximum: The highest temperature recorded during the session.

Minimum: The lowest temperature recorded during the session.

Average: The arithmetic mean of all collected data:

$$Average = \frac{Sum\ of\ \ all\ readings}{Number\ of\ \ readings}$$

4. System Arhitecture

Data Flow Overview :

The architecture of this system follows a **Client-Server model**. The "Server" is the hardware monitoring software running on the PC, and the "Client" is the MATLAB application. Data travels through four main stages to reach the user's screen:

Hardware Sensors: Physical sensors (DTS) inside the CPU measure the temperature.

Libre Hardware Monitor: This software acts as a middleman. It reads the raw sensor data and publishes it as a structured JSON file on a local web server (Port 8085).

MATLAB Data Fetching: The MATLAB app uses a web-read command to download that JSON file every few milliseconds.

GUI Processing: The app parses the data, performs the mathematical calculations, and updates the graphs and statistics.

The Recursive Search Logic

Because the JSON file contains hundreds of sensors (GPU, Fans, RAM, etc.), the system uses a **Recursive Parsing Function**. Instead of looking for a fixed location, the function "walks" through the data tree, checking every "Child" node until it finds the specific text " Core Average. " This makes the architecture robust, even if the computer hardware changes, the app will still find the correct sensor.

Software Components

The application is built using a modular structure within a single MATLAB function:

The UI Manager : Handles the window, panels, and layout.

The Event Listeners : These wait for the user to push buttons (Start/Exit) or change settings (Limit/Speed).

The Monitoring Loop: A continuous while loop that runs in the background to keep the data fresh without freezing the interface.

5. GUI Features

The interface is designed with a **dark-theme industrial aesthetic** for high readability and low eye strain. It is divided into three functional zones:

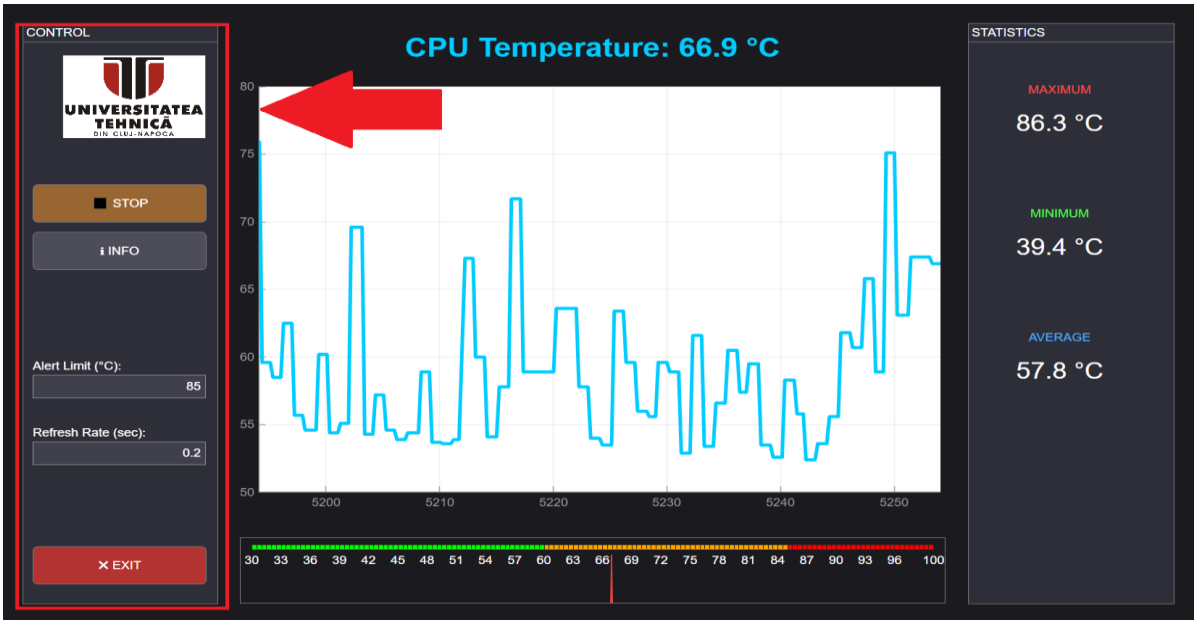
Control and Input Zone (Sidebar) : This area allows the user to interact with the system parameters.

Logo Display : A high-resolution JPG image providing hardware branding.

Alert Limit (Edit Box) : An interactive numeric field where the user sets the temperature threshold for the safety logic.

Refresh Rate (Edit Box) : Controls the speed of the data polling loop.

Action Buttons: Start, Info, and Exit buttons to manage the application state.

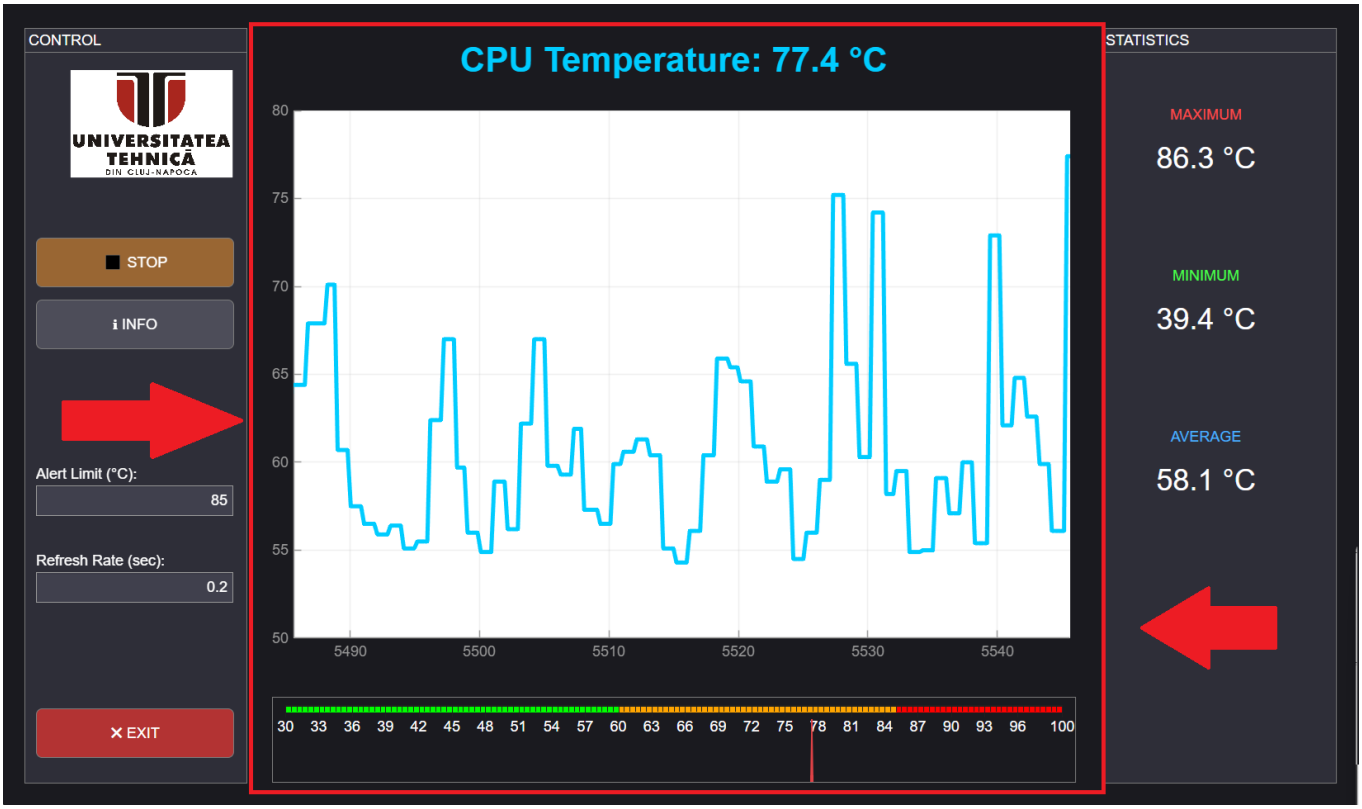


Visual Monitoring Zone (Main Center) the central area is dedicated to high-frequency visual updates.

Real-Time Plot: A sliding-window graph showing the temperature history for the last 60 seconds. The line color changes dynamically (Blue for safe, Red for alert).

Linear Gauge: Provides an analog-style representation of the current temperature for a quick status check.

Dynamic Header: A large bold label displaying the exact temperature value in real-time.

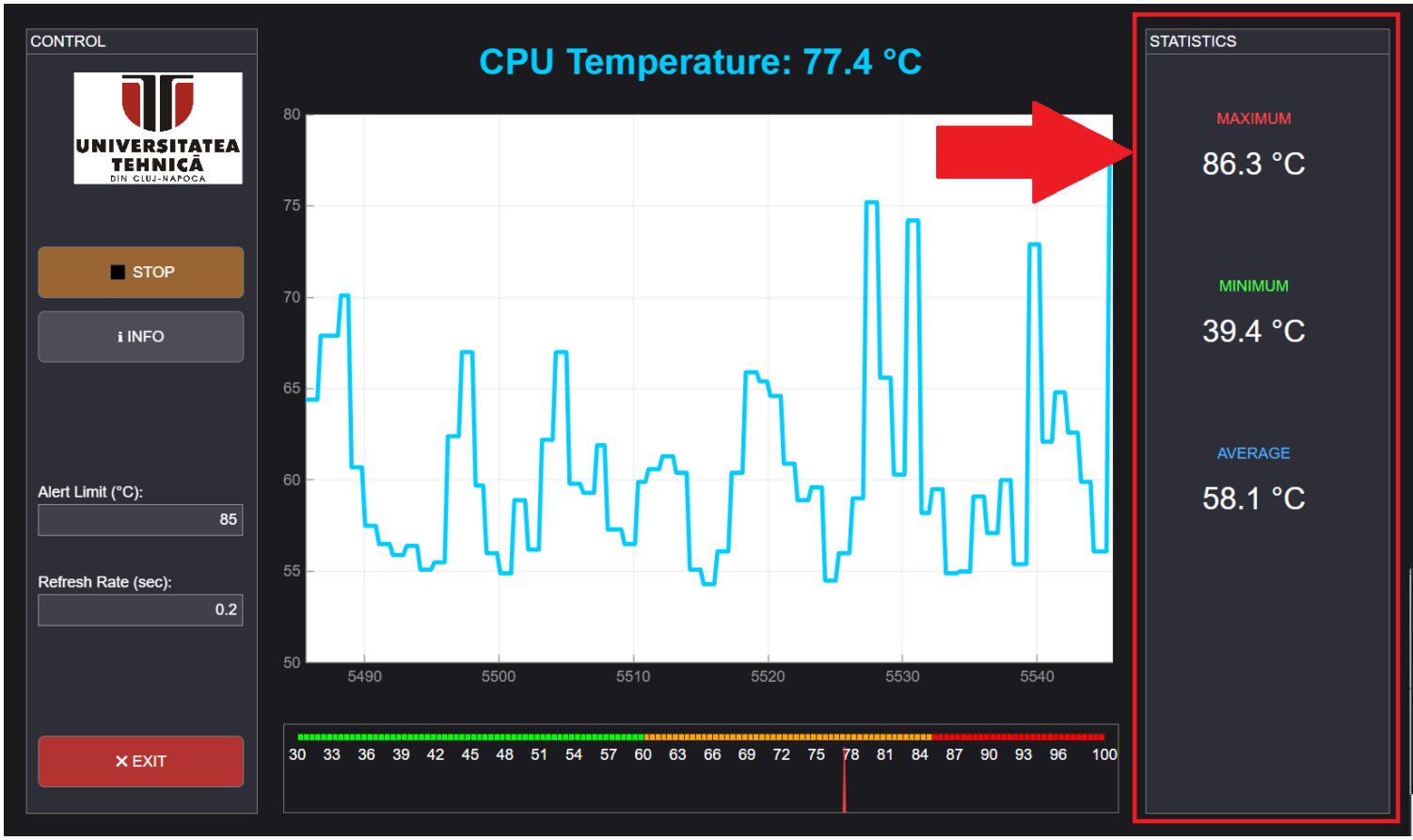


Statistical Analysis Zone (Right Panel): This panel displays aggregated data calculated during the active monitoring session.

Maximum: Displays the highest temperature peak reached.

Minimum: Displays the lowest temperature recorded.

Average: Shows the arithmetic mean of all readings, providing an overview of the thermal trend.

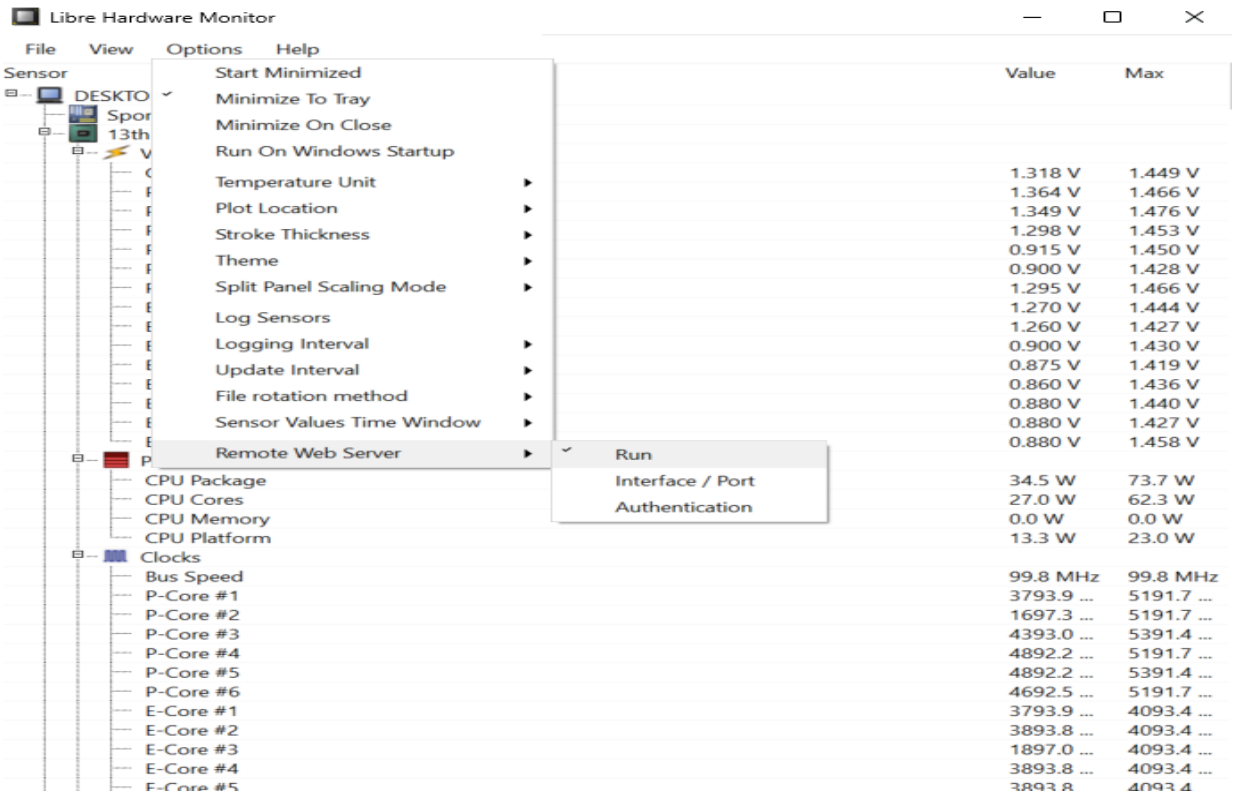


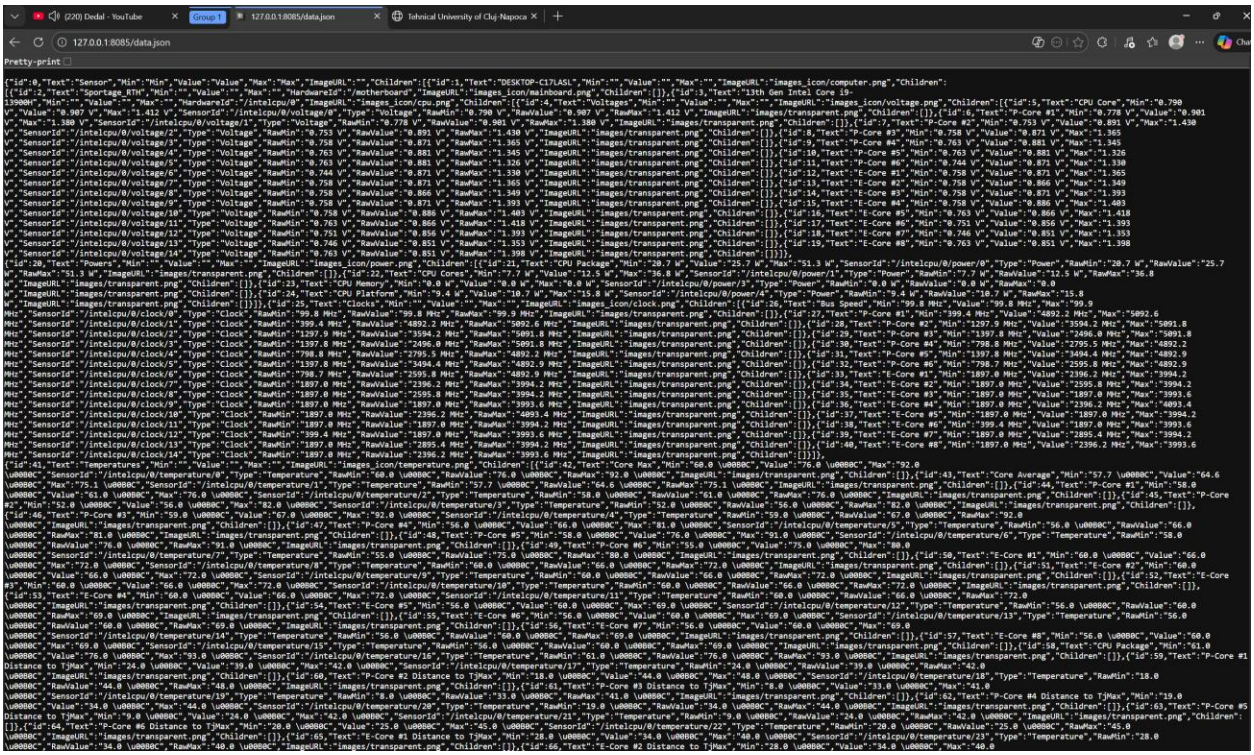
6. User Guide

To ensure the system functions correctly, the hardware server must be active before starting the MATLAB monitoring interface. Follow these steps:

Step 1: External Server Setup

1. Open the Libre Hardware Monitor application on your PC.
2. Go to the Options menu and select Remote Web Server.
3. Click on Run (ensure the default port is 8085). This starts the JSON data stream that MATLAB will read.
4. (Optional) You can check the json data by opening the following url on your preffered browser : **“ 127.0.0.1:8085/data.json ”**





Step 2: Launching the Interface

1. Open MATLAB and navigate to the project folder.
2. Ensure the file cpu_info.jpg is present in the same folder.
3. Run the script cpu_monitor_v6_final.m. The dark-themed dashboard will appear.

Step 3: Configuring Parameters

1. **Set Alert Limit:** Enter your desired safety temperature in the "Alert Limit" box (e.g., 80°C).
2. **Adjust Refresh Rate:** Enter the time interval for data updates in the "Refresh Rate" box. A value of 0.5/0.2 is recommended for smooth performance.

Step 4: Monitoring

1. Press the **START** button. The system will begin fetching data.
2. Watch the **Line Plot** for trends and the **Gauge** for instant status.
3. If the temperature exceeds your limit, the title and graph will turn **Red**.
4. Check the **Statistics** panel on the right to see the peak and average temperatures recorded during the session.

Step 5: Closing the App

1. Press the **STOP** button to pause the data collection.
2. Use the **EXIT** button to safely close the application window and clear the memory.

7. Future Improvements

While the current version is fully functional, there are several ways the system could be expanded in the future:

Data Archiving (Export to Excel)

A useful next step would be adding a "Save Report" button. This would allow the user to export all recorded temperatures and timestamps into an Excel or CSV file at the end of a session, making it easier to analyze the data later.

Dual Monitoring (CPU & GPU)

The system’s search logic is flexible and could be expanded to monitor the **Graphics Card (GPU)** as well. By adding a second graph and another data-fetching line, the user could see both the processor and video card temperatures side-by-side on the same dashboard.

Automatic Fan Control

The project could evolve from a monitoring tool into a control tool. In the future, the app could be programmed to automatically increase the computer's fan speeds whenever the temperature crosses a specific safety threshold.

8. References

1. Libre Hardware Monitor Project – Open Source Hardware Monitor
2. Json Data Parsing in MATLAB
3. MATLAB for Students volume I – Mihaela Cirlugea and Paul Farago
4. MATLAB for Students volume II – Mihaela Cirlugea and Paul Farago

5. Google AI Gemini

9. Source Code

```
function cpu_monitor()
% --- GUI WINDOW CONFIGURATION ---
% Initializes the main application window with a dark theme
fig = uifigure('Name', 'CPU Real-Time Thermal Monitoring System ', 'Color', [0.1 0.1 0.12], 'Position',
[100 100 1100 650]);
% --- UI LAYOUT PANELS ---
% Sidebar for user inputs and control buttons
sidePanel = uipanel(fig, 'Title', 'CONTROL', 'BackgroundColor', [0.18 0.18 0.22], 'ForegroundColor', 'w',
'Position', [20 20 180 610]);
% Right panel for displaying real-time aggregated metrics
statsPanel = uipanel(fig, 'Title', 'STATISTICS', 'BackgroundColor', [0.18 0.18 0.22], 'ForegroundColor',
'w', 'Position', [890 20 190 610]);
% --- IMAGE COMPONENT (JPG LOGO) ---
% Displays a hardware-related image to enhance the visual interface
axImg = uiaxes(sidePanel, 'Position', [5 470 175 110], 'BackgroundColor', [0.18 0.18 0.22]);
try
```

```
img = imread('cpu_info.jpg');
imshow(img, 'Parent', axImg);
axis(axImg, 'tight');
catch
% Fallback title if image file is missing
title(axImg, 'System Active', 'Color', [0.4 0.4 0.4]);
end
axis(axImg, 'off');
% --- INTERACTIVE PARAMETERS (USER MODIFIABLE) ---
% Thermal Alert Threshold: Triggers visual changes when reached
uilabel(sidePanel, 'Text', 'Alert Limit (°C):', 'Position', [10 240 160 20], 'FontColor', 'w');
editLimit = uieditfield(sidePanel, 'numeric', 'Position', [10 215 160 25], 'Value', 85, 'BackgroundColor',
[0.25 0.25 0.3], 'FontColor', 'w');
% Data Refresh Rate: Controls the sampling frequency of the JSON feed
uilabel(sidePanel, 'Text', 'Refresh Rate (sec):', 'Position', [10 170 160 20], 'FontColor', 'w');
editSpeed = uieditfield(sidePanel, 'numeric', 'Position', [10 145 160 25], 'Value', 0.5, 'Limits', [0.1
10], 'BackgroundColor', [0.25 0.25 0.3], 'FontColor', 'w');
% --- MAIN DISPLAY COMPONENTS ---
% Dynamic Title showing the current temperature
lblTitle = uilabel(fig, 'Text', 'CPU Temperature: -- °C', 'Position', [220 580 650 50], 'FontSize', 28,
'FontWeight', 'bold', 'FontColor', [0 0.8 1], 'HorizontalAlignment', 'center');
% Linear Gauge for a quick visual status check
gauge = uigauge(fig, 'linear', 'Position', [220 20 650 70], 'BackgroundColor', [0.1 0.1 0.12],
'FontColor', 'w');
gauge.Limits = [30 100];
gauge.ScaleColors = {'#00FF00', '#FFA500', '#FF0000'};
gauge.ScaleColorLimits = [30 60; 60 85; 85 100];
% --- STATISTICS LABELS ---
% Displays peak, lowest, and mean temperatures during the session
valMax = uilabel(statsPanel, 'Text', '-- °C', 'Position', [10 480 150 50], 'FontSize', 24, 'FontColor',
'w', 'HorizontalAlignment', 'center');
uilabel(statsPanel, 'Text', 'MAXIMUM', 'Position', [10 530 150 20], 'FontColor', [1 0.3 0.3],
'HorizontalAlignment', 'center');
valMin = uilabel(statsPanel, 'Text', '-- °C', 'Position', [10 350 150 50], 'FontSize', 24, 'FontColor',
'w', 'HorizontalAlignment', 'center');
uilabel(statsPanel, 'Text', 'MINIMUM', 'Position', [10 400 150 20], 'FontColor', [0.3 1 0.3],
'HorizontalAlignment', 'center');
valAvg = uilabel(statsPanel, 'Text', '-- °C', 'Position', [10 220 150 50], 'FontSize', 24, 'FontColor',
'w', 'HorizontalAlignment', 'center');
uilabel(statsPanel, 'Text', 'AVERAGE', 'Position', [10 270 150 20], 'FontColor', [0.3 0.7 1],
'HorizontalAlignment', 'center');
% --- MAIN REAL-TIME GRAPH ---
% Plotting area for the thermal evolution over time
ax = uiaxes(fig, 'Position', [220 120 650 450], 'BackgroundColor', [0.12 0.12 0.15], 'XColor', [0.6 0.6
0.6], 'YColor', [0.6 0.6 0.6]);
grid(ax, 'on'); hold(ax, 'on');
hLine = plot(ax, 0, 0, 'Color', [0 0.8 1], 'LineWidth', 2.5);
% --- CONTROL BUTTONS ---
btnStart = uibutton(sidePanel, 'text', '▶ START', 'Position', [10 400 160 40], 'BackgroundColor', [0.2
0.6 0.2], 'FontColor', 'w');
btnDoc = uibutton(sidePanel, 'text', 'DOCUMENTATION', 'Position', [10 350 160 40], 'BackgroundColor', [0.3
0.3 0.35], 'FontColor', 'w');
btnExit = uibutton(sidePanel, 'text', '✕ EXIT', 'Position', [10 20 160 40], 'BackgroundColor', [0.7 0.2
0.2], 'FontColor', 'w');
% --- STATE VARIABLES ---
isRunning = false; % Tracks monitoring status
allTemps = []; % Buffer to store all historical readings
% --- CALLBACK ASSIGNMENTS ---
btnExit.ButtonPushedFcn = @(src, event) delete(fig);
```

```
btnDoc.ButtonPushedFcn = @(src, event) showDoc();
btnStart.ButtonPushedFcn = @(src, event) toggleStart();
function showDoc()
% Defines the name of the documentation file
pdfFileName = 'Documentatie_CadarAlexDumitru.pdf';
% Checks if the file exists in the current folder before trying to open it
if exist(pdfFileName, 'file')
% Opens the PDF with the system's default viewer (e.g., Adobe, Edge, etc.)
winopen(pdfFileName);
else
% Shows an error message if the file is missing
uialert(fig, 'Documentation file not found in the project folder.', ...
'File Error', 'Icon', 'error');
end
end
function toggleStart()
% Manages the Start/Stop toggle logic and UI feedback
if isRunning
isRunning = false;
btnStart.Text = '▶ RESTART';
btnStart.BackgroundColor = [0.2 0.6 0.2];
else
isRunning = true;
btnStart.Text = '■ STOP';
btnStart.BackgroundColor = [0.6 0.4 0.2];
allTemps = []; % Clear history on new run
set(hLine, 'XData', [], 'YData', []);
startMonitoring();
end
end
function startMonitoring()
% Main loop for fetching and processing thermal data
url = 'http://127.0.0.1:8085/data.json';
opt = weboptions('Timeout', 2);
t_start = datetime('now');

while isRunning && ishandle(fig)
  try
     % Fetch data from Libre Hardware Monitor
     data = webread(url, opt);
     tempVal = getSpecificTemp(data, 'Core Average');
     if ~isempty(tempVal)
        t_now = datetime('now');
        elapsed = seconds(t_now - t_start);
        allTemps(end+1) = tempVal;

        % Update Graph (Plot)
        set(hLine, 'XData', [hLine.XData, elapsed], 'YData', [hLine.YData, tempVal]);
        % Automatic X-axis sliding window (last 60 seconds)
        if elapsed > 60, ax.XLim = [elapsed-60, elapsed]; end

        % Update Statistics (Calculated Values)
        valMax.Text = sprintf('%.1f °C', max(allTemps));
        valMin.Text = sprintf('%.1f °C', min(allTemps));
        valAvg.Text = sprintf('%.1f °C', mean(allTemps));

        % Update Header and Gauge display
        lblTitle.Text = sprintf('CPU Temperature: %.1f °C', tempVal);
        gauge.Value = tempVal;
```

```
        % THRESHOLD CHECK (Safety Index Logic)
        % Visual alert triggers if temperature exceeds user-defined limit
        if tempVal >= editLimit.Value
                lblTitle.FontColor = [1 0.3 0.3]; % Red alert
                hLine.Color = [1 0.3 0.3];
        else
                lblTitle.FontColor = [0 0.8 1]; % Cool blue status
                hLine.Color = [0 0.8 1];
           end
        end
catch
        % Silently catch connection drops to prevent script crashing
        end
        drawnow;
           % Wait for the next sampling cycle (Adjustable via GUI)
           pause(editSpeed.Value);
           end
        end
end
% --- RECURSIVE DATA PARSING FUNCTION ---
function val = getSpecificTemp(node, targetName)
   % Recursively searches through the JSON tree to find hardware sensor values
   val = [];
   % Base case: Check if current node is the target sensor
   if isfield(node, 'Text') && isfield(node, 'Value')
        if strcmp(node.Text, targetName) && contains(node.Value, '°C')
          str = extractBefore(node.Value, " ");
          val = str2double(strrep(str, ',', '.'));
          return;
        end
end
% Recursive case: Traverse child nodes in the JSON structure
if isfield(node, 'Children')
        c = node.Children;
                if iscell(c)
             for i = 1:length(c)
                val = getSpecificTemp(c{i}, targetName);
                if ~isempty(val), return; end
             end
        elseif isstruct(c)
           for i = 1:length(c)
              val = getSpecificTemp(c(i), targetName);
              if ~isempty(val), return; end
           end
       end
    end
end
```