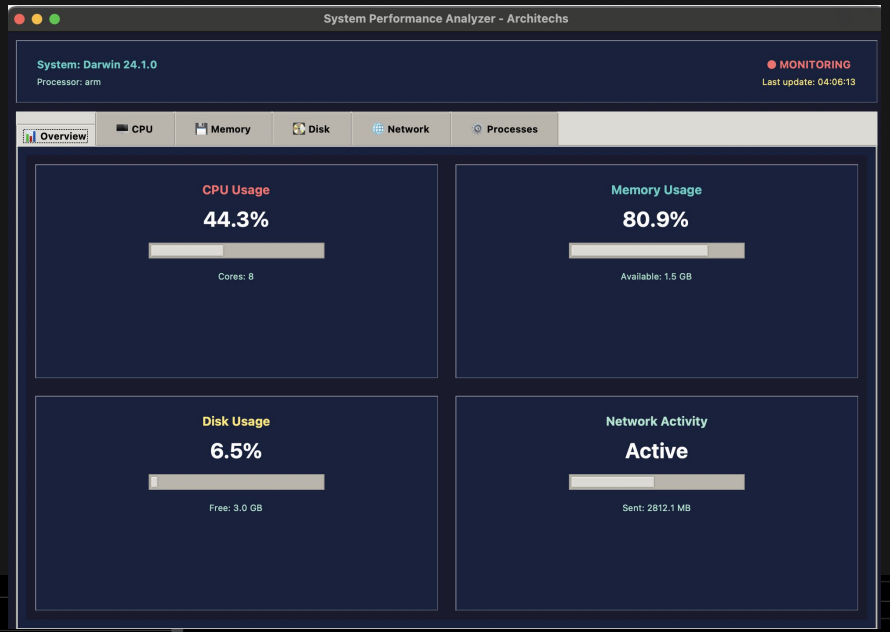
System Performance Analyzer

Team no- 250
Sub- SE-VI-T250
Team name: Architechs
Members: Harshit Jasuja, Yashika Dixit, Shivendra Srivastava

Objective

The primary goal of the System Performance Analyzer is to provide an intuitive, resource-efficient, and visually appealing GUI tool for **real-time monitoring of macOS system performance**. This includes tracking:

- CPU usage
- RAM consumption
- Disk utilization
- System specifications and OS metrics



Problem Statement & Proposed Solution

Problem Statement:

- Users often lack simple, open-source performance monitoring tools tailored for their specific interface.
- Native Activity Monitor is comprehensive but not always user-friendly for quick performance snapshots.
- Cross-platform tools often don’t scale well visually or functionally on macOS due to resolution and framework constraints.

Proposed Solution:

- A Python-based desktop application utilizing **`Tkinter`** and **`psutil`** for lightweight, efficient system tracking.
- Smooth **real-time graphing** using `matplotlib`, with alert popups for threshold breaches.
- Inclusion of modern UI/UX elements like splash screens and modular navigation for better user experience.

Updated Workflow / Architecture

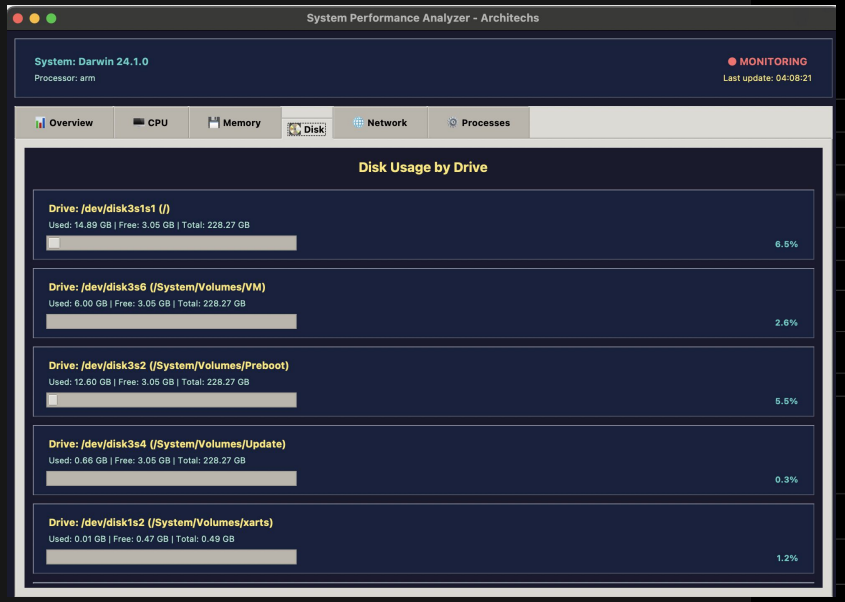
High-Level Architecture:

1. **User Interaction Layer:**
   - Graphical Interface via `tkinter`
   - Buttons, panels, graphs, and alert popups
2. **System Metrics Engine:**
   - `psutil` library gathers live data on CPU, memory, disk, battery, and processes
3. **Visualization Engine:**
   - `matplotlib` generates smooth, real-time graphs
   - Embeds charts directly within the Tkinter window using `FigureCanvasTkAgg`
4. **Thread Management:**
   - Threads ensure non-blocking updates to graphs and metrics without freezing the UI
5. **Utility Modules:**
   - Export log data to JSON
   - Theme switch (light/dark)
   - Splash screen integration on startup

```
User → GUI Interface → Metric Fetching → Visualization / Alert → Optional Export
```

Key Features

- **Splash Screen with Branding**
  Visually appealing startup screen with team and app identity.
- **Real-Time Resource Graphs**
  - CPU utilization over the last 60 seconds
  - Memory usage tracking
  - Disk space info
- **System Information Display**
  Fetch and display OS, processor, RAM, boot time, and system name.
- **Modular, Scalable GUI**
  Adapted specifically for macOS 13” screens.
- **Multithreading Support**
  For smooth GUI performance during heavy metric computation.
- **Alert System**
  Notifies users with popup dialogs on resource overuse.
- **Log Export**
  Export performance data to JSON for record keeping.



Technical Progress Overview

Module	Status	Description
Splash Screen	✅ Complete	Fully functional with branding
CPU Monitoring	✅ Complete	Graph + numeric display
Memory Monitoring	✅ Complete	Real-time updates
Disk Usage	✅ Complete	Storage available, used, and free
System Info Panel	✅ Complete	Platform, version, uptime
Thread Handling	✅ Complete	GUI does not freeze during updates
Alerts & Warnings	✅ Complete	Threshold based popup alerts
Export Data	✅ Complete	JSON format export

Code Snippet – Splash Screen

```
class SplashScreen:
    def __init__(self, root):
        self.splash = tk.Toplevel(root)
        self.splash.title("System Performance Analyzer")
        self.splash.geometry("500x300")
        self.splash.configure(bg="#1a1a2e")
```

Functionality:

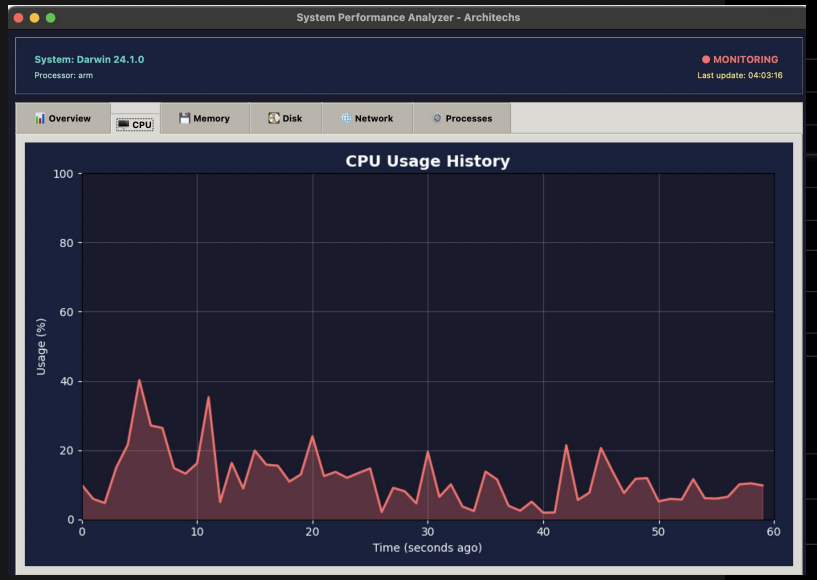
- Initializes a top-level window separate from the main app.
- Sets window dimensions and background theme.
- Adds branding during the app launch sequence.

Code Snippet – CPU Graphing

```
self.cpu_fig = Figure(figsize=(5, 2), dpi=100)
self.cpu_ax = self.cpu_fig.add_subplot(111)
self.cpu_data = deque([0]*60, maxlen=60)
self.cpu_canvas = FigureCanvasTkAgg(self.cpu_fig, master=self.main_frame)
```

Functionality:

- Initializes a matplotlib figure and axis for plotting CPU usage.
- Uses a deque to maintain a fixed-length rolling window of CPU data.
- Embeds the chart inside the tkinter GUI frame.



Role-wise Contributions

Team Member	Contribution Area
Harshit Jasuja	psutil system data integration, Threading and live data updates, Add process-level tracking, Additional alert thresholds (RAM, Disk)
Yashika Dixit	GUI design and layout , Splash screen and branding integration, Theme customization (light/dark mode)
Shivendra Srivastava	Real-time graph plotting with matplotlib, Alert system for high CPU usage, Packaging tool into .app (macOS)

Challenges Faced & Key Learnings

Challenges:

- Integrating live data updates without disrupting UI responsiveness.
- Adapting the interface to macOS-specific dimensions and display scaling.
- Managing background threads with tkinter's mainloop.

Key Learnings:

- Efficient use of Python libraries like `psutil` and `matplotlib`.
- GUI threading and synchronization concepts.
- Application modularization for easier maintenance.
- Coordinated teamwork and Git-based version control.

Future Enhancements & Scope

- Process-level monitoring: Show top memory- or CPU-consuming processes.
- Remote access: Allow stats syncing to a mobile or cloud dashboard.
- Theme customization: Add light/dark mode toggles for accessibility.
- macOS packaging: Convert Python script to native `.app` using `py2app`.
- System cleanup recommendations: Suggest user actions when system slows down.