Table of contents

Introduction

The System Monitoring Dashboard aims to develop a user-friendly Graphical User Interface (GUI) dashboard for monitoring various aspects of a Linux system. This project addresses the need for an easy-to-use tool that provides real-time insights into system performance, resource utilisation, and other critical metrics. The dashboard will be designed to assist system administrators, DevOps engineers, and other users in efficiently managing and troubleshooting Linux-based systems directly from the application.

1. Problem Statement

Monitoring Linux-based systems can be challenging due to the lack of user-friendly tools that provide comprehensive insights into system performance and resource utilisation. Existing monitoring solutions often require complex configurations and lack intuitive interfaces, making it difficult for users to identify and address system issues in a timely manner. There is a need for a centralised dashboard that simplifies system monitoring and provides real-time visibility into CPU usage, memory usage, battery usage, network traffic, and other key metrics.

2. Objectives

- Collect and display real-time data of system metrics.
- Provide visualisations such as charts, graphs, and tables.
- Provide ability to interfere processes
- Support customization options to allow users to configure monitoring preferences and dashboard layouts according to their needs.

3. Scope

The scope of the project includes:

- Developing a GUI dashboard using Gtop suitable for the terminal environment.
- Implementing backend functionalities for data collection, processing, and storage using Python.

4. Methodology

- Conducting research on relevant technologies and tools for creating GUI dashboard development.
- Designing the system architecture.
- Implementing frontend and backend components in parallel, with continuous integration and testing.
- Iteratively refining the dashboard interface based on user feedback and usability testing.
- Conducting thorough testing and optimization to ensure the dashboard meets performance and reliability requirements.

5. Requirements

The technical requirements for the project include:

- Frontend: PyQT5, CSS
- Backend: Python

6. Challenges and solution

6.1. Gtk Compatibility Issues on macOS

Our development faced significant hurdles with Gtk on macOS, including UI inconsistencies and performance degradation, jeopardizing the application's stability and user experience on Apple devices.

Solution: To address these issues, we switched to PyQt, which offers better macOS support, ensuring smoother performance and enhanced UI consistency. PyQt's rich features and strong community resources accelerated our development and improved the overall quality of the application on macOS platforms. As well as it works pretty good on Windows machines.

6.2. Performance Optimization in Real-Time Data Updates

Our development faced significant hurdles with maintaining efficient performance without consuming excessive system resources, especially crucial in a PyQt-based system monitoring dashboard that updates real-time data frequently. Frequent updates can strain the system, leading to increased CPU load and memory consumption, which in turn could negatively affect both the system being monitored and the dashboard's responsiveness and functionality.

Solution: To address these challenges, we optimized data collection and update intervals. We strategically set update intervals and applied data throttling to balance real-time accuracy with minimal performance overhead. This helped in reducing the unnecessary processing of data during periods of low system activity.

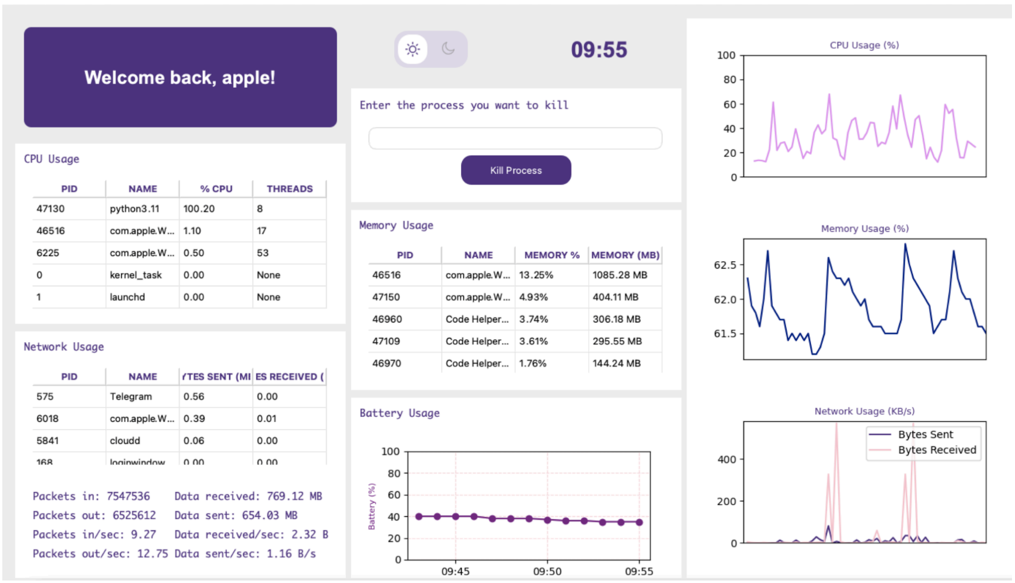
6.3. UI Design and Usability

Our project faced a big challenge in creating a user interface for our system monitoring dashboard. The interface needed to be easy to use, quick, and able to show a lot of data clearly. It was tough to make sure that all users could understand and use the dashboard easily, especially when displaying complex data like real-time CPU, memory, and network graphs.

Solution: To solve this design challenge, we followed well-known design rules and used the powerful tools provided by PyQt, which helped make the user interface better and easier to use. We also used Qt Quick to make the interface more dynamic and attractive. This made our dashboard not only work well but also be enjoyable and easy for users to interact with.

7. Outcomes

Overview of the System Monitoring Dashboard (Pic 1). It is a one page dashboard that displays information of CPU Usage, network usage, memory usage, battery usage and its graphs.



CPU Usage

PID	NAME	% CPU	THREADS
47130	python3.11	100.20	8
46516	com.apple.W...	1.10	17
6225	com.apple.W...	0.50	53
0	kernel_task	0.00	None
1	launchd	0.00	None

Network Usage

PID	NAME	'TES SENT (MI	ES RECEIVED (
575	Telegram	0.56	0.00
6018	com.apple.W...	0.39	0.01
5841	cloudd	0.06	0.00
168	loginwindow	0.00	0.00

```
Packets in: 7547536      Data received: 769.12 MB
Packets out: 6525612     Data sent: 654.03 MB
Packets in/sec: 9.27     Data received/sec: 2.32 B
Packets out/sec: 12.75 Data sent/sec: 1.16 B/s
```

Memory Usage

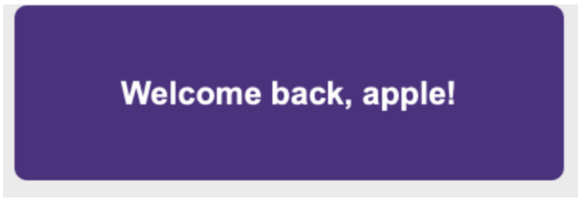
PID	NAME	MEMORY %	MEMORY (MB)
46516	com.apple.W...	13.25%	1085.28 MB
47150	com.apple.W...	4.93%	404.11 MB
46960	Code Helper...	3.74%	306.18 MB
47109	Code Helper...	3.61%	295.55 MB
46970	Code Helper...	1.76%	144.24 MB

Battery Usage

Pic 1. Overview of System Monitoring Dashboard

7.1. Welcome Widget

Welcome widget displays a greeting to current user. Application takes user name from the system and displays it. To get a user name, a library *getpass* is used.



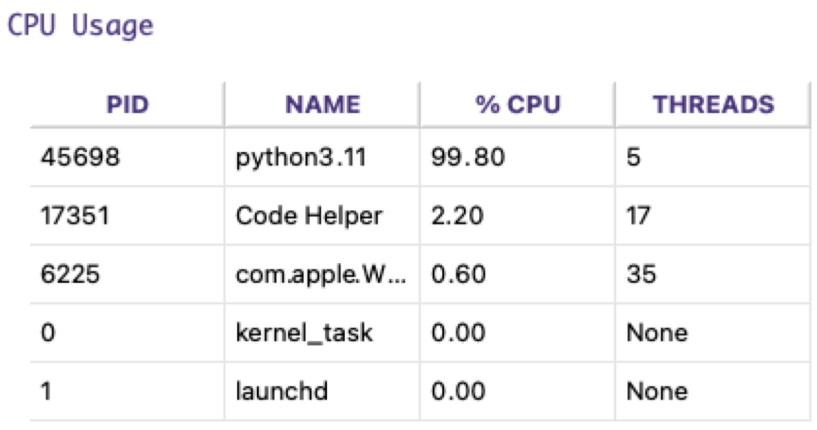
Pic 2. Welcome widget

7.2. CPU Usage Widget

The CPU Usage Table provides detailed information on the five processes that are currently consuming the most CPU resources on a system. This data can be accessed using the *psutil* library in Python. The table includes the following columns:

- **PID:** Process ID, which uniquely identifies each process running on the system.
- **Name:** The name of the process, providing an easy identification of the application or service.
- **%CPU:** The percentage of the system's CPU that the process is using. This helps in identifying processes that are consuming significant CPU resources.
- **Threads:** The number of threads within the process. A thread is the smallest unit of processing that can be scheduled by an operating system.

This table is instrumental for monitoring and managing system performance by pinpointing high-resource-consuming processes.



CPU Usage

PID	NAME	% CPU	THREADS
45698	python3.11	99.80	5
17351	Code Helper	2.20	17
6225	com.apple.W...	0.60	35
0	kernel_task	0.00	None
1	launchd	0.00	None

Pic 3. CPU usage widget

7.3. Network Usage Widget

The Network Usage widget displayed in the pic 4 is a useful tool for monitoring network activity of applications on a system. It lists the top five processes consuming network resources based on bytes sent and received. Here's a breakdown of the information provided in the widget:

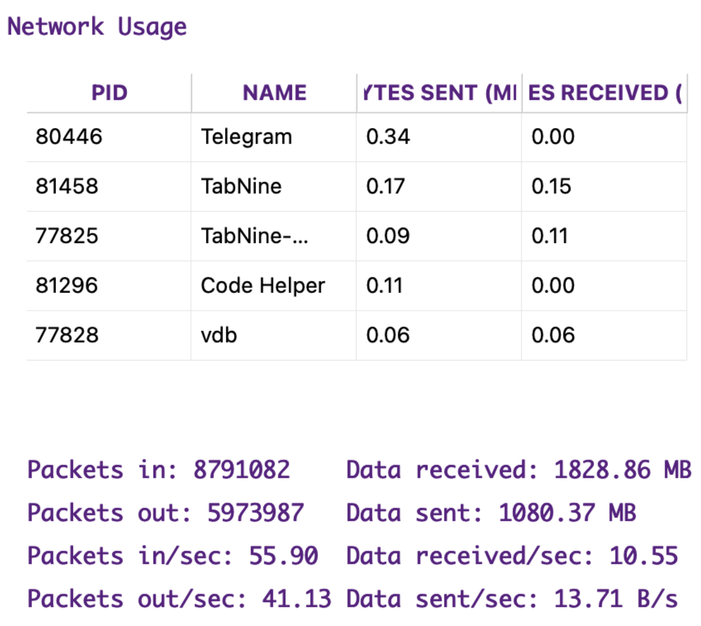
- **PID:** Process ID, which uniquely identifies each process running on the system.
- **Name:** The name of the process, providing an easy identification of the application or service.
- **Bytes Sent (MB):** Shows the total megabytes sent by each process. This can be used to detect processes that are uploading significant amounts of data.
- **Bytes Received (MB):** Displays the total megabytes received by each process. This is useful for identifying processes that are downloading substantial data.

Below the table, the widget provides aggregated network data:

- **Packets in:** Total number of packets received by the system.
- **Packets out:** Total number of packets sent by the system.
- **Data received (MB): Total megabytes of data received.**
- **Data sent (MB):** Total megabytes of data sent.
- **Packets in/sec:** The rate at which packets are received per second.
- **Packets out/sec:** The rate at which packets are sent per second.
- **Data received/sec (B/s):** Bytes per second received by the system.
- **Data sent/sec (B/s):** Bytes per second sent by the system.

This widget is particularly valuable for network administrators and users interested in monitoring and optimizing their network performance. By providing both per-process and total system network data, it helps in identifying high-traffic applications, assessing overall network load, and planning for network capacity.

This data can be accessed using the *psutil* library in Python.



Network Usage

PID	NAME	'TES SENT (MI	ES RECEIVED (
80446	Telegram	0.34	0.00
81458	TabNine	0.17	0.15
77825	TabNine-...	0.09	0.11
81296	Code Helper	0.11	0.00
77828	vdb	0.06	0.06

```
Packets in: 8791082     Data received: 1828.86 MB
Packets out: 5973987    Data sent: 1080.37 MB
Packets in/sec: 55.90   Data received/sec: 10.55
Packets out/sec: 41.13  Data sent/sec: 13.71 B/s
```

Pic 4. Network usage widget

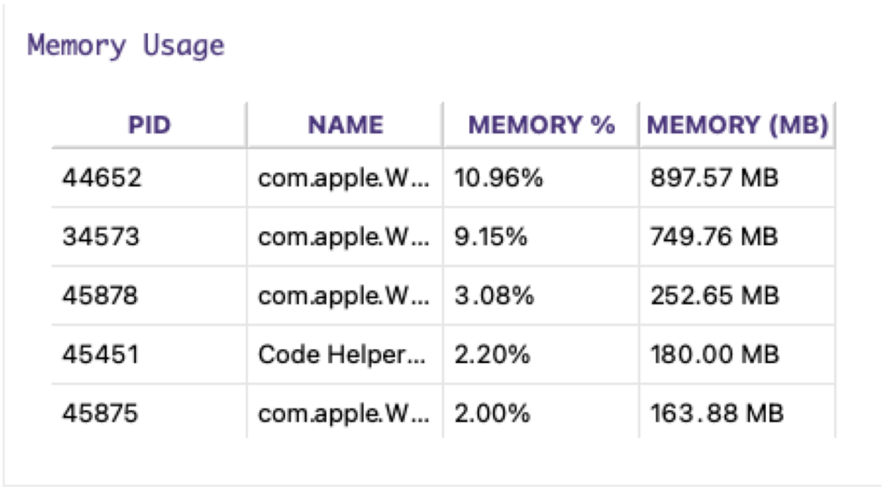
7.4. Memory Usage Widget

The Memory Usage Table provides a detailed overview of the processes consuming the most memory on the system, allowing for effective monitoring and management of system resources. The table includes the following columns:

- **PID:** Process ID, which uniquely identifies each process running on the system.

- **Name:** The name of the process, providing an easy identification of the application or service.
- **Memory %:** This column shows the percentage of the total available memory that each listed process is using. It helps in quickly identifying which processes are using a disproportionate amount of memory relative to the total system memory.
- **Memory (MB):** Displays the actual amount of memory, in megabytes, that each process is consuming. This metric is crucial for assessing the memory footprint of each process.

This table is invaluable for system administrators and users looking to optimize their systems or troubleshoot issues related to high memory consumption. By providing both percentage and absolute metrics, it offers a comprehensive view of memory usage that can guide decisions regarding application management, system upgrades, or configuration adjustments to enhance overall performance.



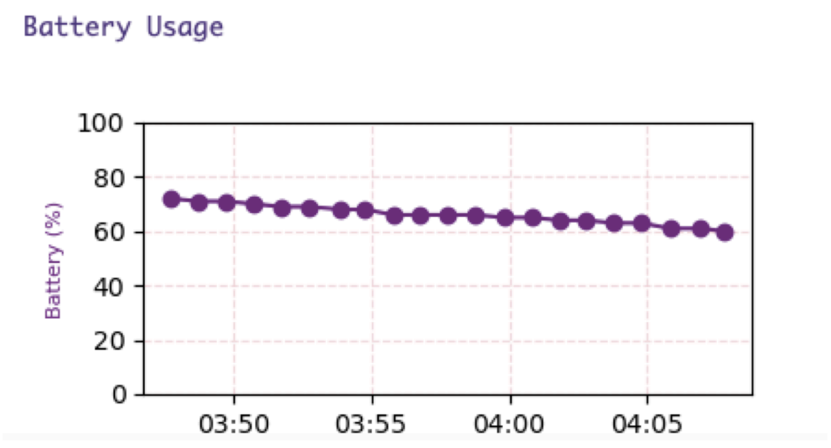
Memory Usage

PID	NAME	MEMORY %	MEMORY (MB)
44652	com.apple.W...	10.96%	897.57 MB
34573	com.apple.W...	9.15%	749.76 MB
45878	com.apple.W...	3.08%	252.65 MB
45451	Code Helper...	2.20%	180.00 MB
45875	com.apple.W...	2.00%	163.88 MB

Pic 5. Memory usage widget

7.5. Battery Usage

The Battery Usage feature in devices such as smartphones, tablets, and laptops is essential for monitoring and managing how apps and system processes consume battery power. This feature helps users understand their device's power consumption patterns and adjust settings or usage to optimize battery life. To get access of battery data, *psutil.sensors_battery()* is used.



Pic 5. Battery usage widget

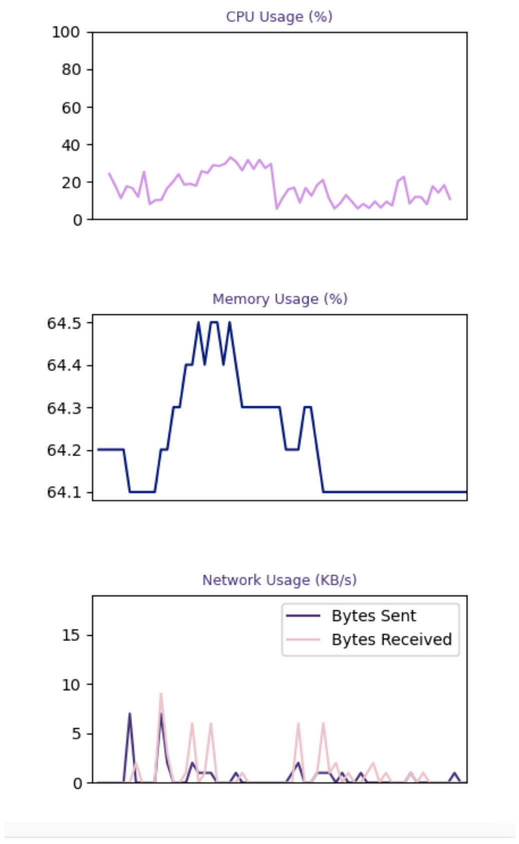
7.6. CPU, Memory and Network Usage Plots Widget

In this widget for CPU, Memory, and Network Usage, various system metrics are captured using the *psutil* library, a versatile tool for fetching process and system utilization information in Python. Here's a breakdown of how each metric is gathered:

- CPU Usage: The widget retrieves the CPU usage percentage using *psutil.cpu_percent()*. This function provides an instantaneous reading of the system's CPU utilization, giving users a snapshot of how much of the CPU's capacity is currently being used.

- Memory Usage: Memory consumption is monitored through *psutil.virtual_memory().percent*. This returns the percentage of total memory that is currently being used by the system. It is a crucial metric for understanding memory load and avoiding potential overflows that could lead to system slowdowns or crashes.
- Network Usage: Network traffic statistics are collected via *psutil.net_io_counters()*. This function yields network IO statistics, including bytes sent and received, providing a comprehensive view of the network activity. Such data is vital for monitoring incoming and outgoing traffic, helping to manage bandwidth and ensure network performance remains optimal.

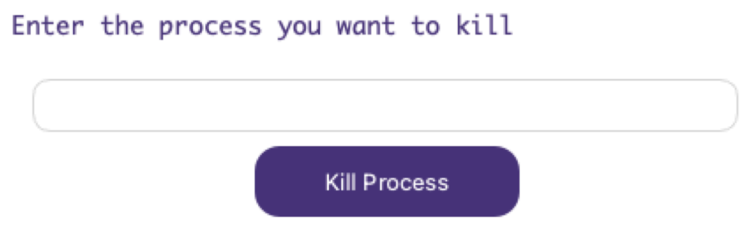
These metrics form the backbone of the widget, allowing for dynamic and real-time monitoring of system resources. By continuously updating these statistics, the widget provides administrators and users with timely data to make informed decisions about resource management, system performance optimization, and potential troubleshooting.



Pic 6. Graph for CPU, Memory and Network uasge

7.7. Kill Process Widget

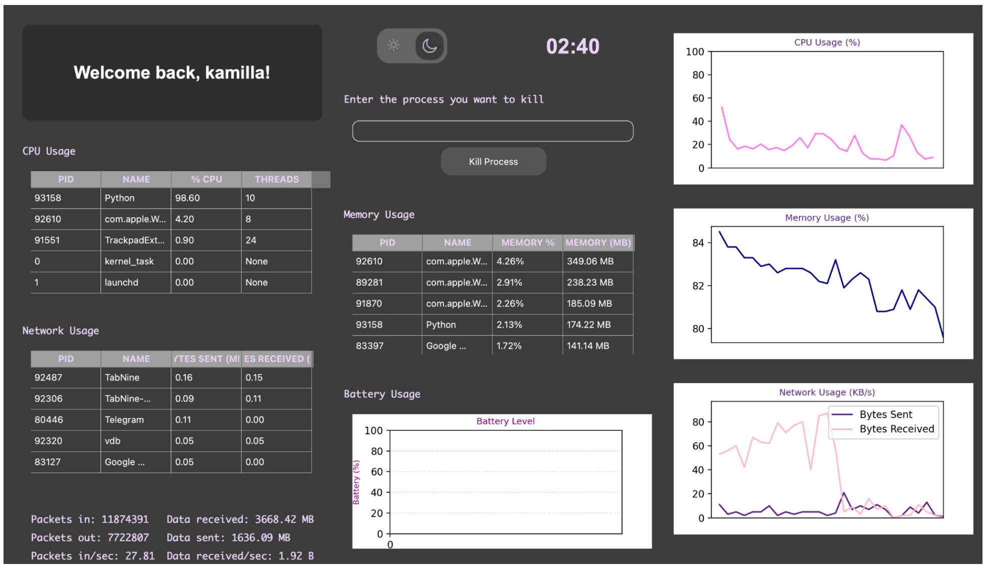
The Kill Process Widget is designed to facilitate the management of system resources by allowing users to terminate processes that are consuming excessive resources or behaving erratically. The primary function of this widget is to terminate processes. It sends a SIGTERM signal to the process identified by its PID (Process ID). SIGTERM is the default signal used to terminate a process gracefully, allowing it to release resources and save state if necessary before shutting down.



Pic 7. Kill process widget

7.8. Toggle Switch Widget

The toggle switch widget allows users to switch between dark mode and light mode in a user interface. Dark mode uses dark-colored icons, and user interface elements on a dark background, while light mode uses light-colored elements on a light background. The switch is a slider that users press and it moves from one side to another to change modes.



Welcome back, kamilla!

02:40

Enter the process you want to kill

CPU Usage

PID	NAME	% CPU	THREADS
93158	Python	98.60	10
92610	com.apple.W...	4.20	8
91551	TrackpadExt...	0.90	24
0	kernel_task	0.00	None
1	launchd	0.00	None

Memory Usage

PID	NAME	MEMORY %	MEMORY (MB)
92610	com.apple.W...	4.26%	349.06 MB
89281	com.apple.W...	2.91%	238.23 MB
91870	com.apple.W...	2.26%	185.09 MB
93158	Python	2.13%	174.22 MB
83397	Google ...	1.72%	141.14 MB

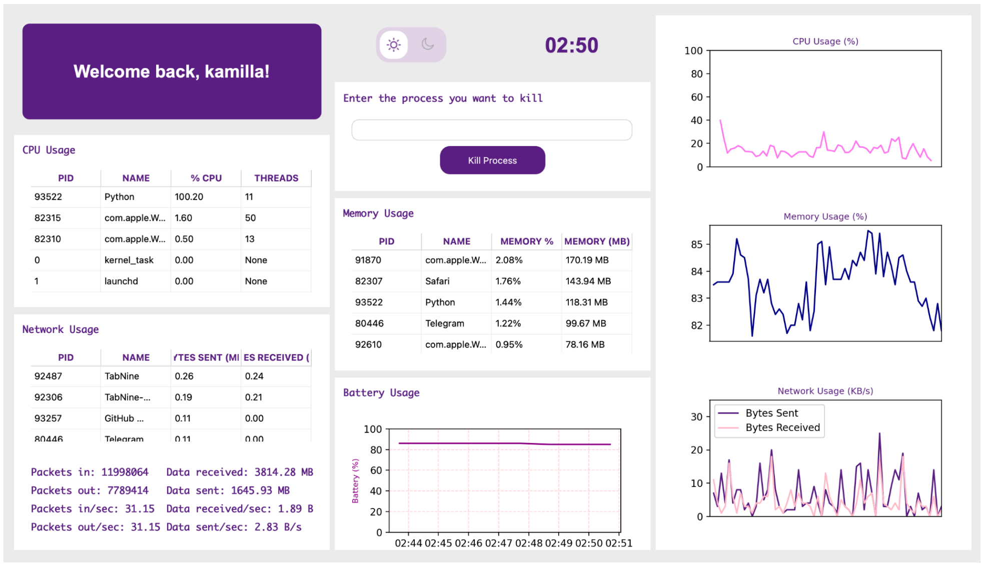
Network Usage

PID	NAME	'TES SENT (MI	ES RECEIVED (
92487	TabNine	0.16	0.15
92306	TabNine-...	0.09	0.11
80446	Telegram	0.11	0.00
92320	vdb	0.05	0.05
83127	Google ...	0.05	0.00

```
Packets in: 11874391    Data received: 3668.42 MB
Packets out: 7722807    Data sent: 1636.09 MB
Packets in/sec: 27.81   Data received/sec: 1.92 B
```

Battery Usage

Pic 8. Dark mode



Welcome back, kamilla!

02:50

Enter the process you want to kill

CPU Usage

PID	NAME	% CPU	THREADS
93522	Python	100.20	11
82315	com.apple.W...	1.60	50
82310	com.apple.W...	0.50	13
0	kernel_task	0.00	None
1	launchd	0.00	None

Memory Usage

PID	NAME	MEMORY %	MEMORY (MB)
91870	com.apple.W...	2.08%	170.19 MB
82307	Safari	1.76%	143.94 MB
93522	Python	1.44%	118.31 MB
80446	Telegram	1.22%	99.67 MB
92610	com.apple.W...	0.95%	78.16 MB

Network Usage

PID	NAME	'TES SENT (MI	ES RECEIVED (
92487	TabNine	0.26	0.24
92306	TabNine-...	0.19	0.21
93257	GitHub ...	0.11	0.00
80446	Telegram	0.11	0.00

```
Packets in: 11998064    Data received: 3814.28 MB
Packets out: 7789414    Data sent: 1645.93 MB
Packets in/sec: 31.15   Data received/sec: 1.89 B
Packets out/sec: 31.15 Data sent/sec: 2.83 B/s
```

Battery Usage

Pic 9. Light mode

7.9. Clock Widget

The Clock Widget is a simple yet essential tool that displays the current time as utilized by the system. It serves not only as a practical tool for users to keep track of time directly from their digital workspace but also integrates seamlessly into the user interface of the system or application.

Pic 10. Clock widget

8. Source Code

In order to access source code of the project about System Monitoring Dashboard, use following link https://github.com/cvmllv/sp_project

References

1. https://github.com/aksakalli/gtop?tab=readme-ov-file
2. https://dribbble.com/shots/16331784-IAAS-Dashboard
3. https://www.datapine.com/blog/monitoring-dashboard-templates/
4. https://medium.com/devops-dudes/monitoring-how-to-build-your-monitoring-dashboards-e11f89918dd1
5. https://psutil.readthedocs.io/en/latest/
6. https://doc.qt.io/qtforpython-6/
7. https://www.geeksforgeeks.org/qt-alignment-in-pyqt5/
8. https://habr.com/ru/companies/skillfactory/articles/599599/