1. Installation Instruction

1.1 Make sure Rust is installed on your system.

- On macOS: Open your terminal and run:

  curl --proto ' =https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

- On Windows: Download and run the Rust installer from rustup.rs. Complete the installation steps, then restart your terminal.

1.2 Clone this repository: $ git clone https://github.com/SawZiDunn/real-time-sys-monitor-rust.git

1.3 Go to the project directory: $ cd real-time-sys-monitor-rust

2. User Guide

2.1 Launching the Application

After installation, start the application by running the commands below on your Window or Mac Terminal by building and running the program.

Build the program: $ cargo build

Run the program: $ cargo run

2.2 Interface Overview

- **Dashboard**: Displays an overview of CPU, memory, disk, and network usage.
- **Processes**: Lists active processes with details on memory and CPU usage according to highest to lowest memory consumption.

2.3 Interacting with the Application

- **Start/Stop Monitoring**: Use the start/stop monitoring button to control monitoring process.
- **Refresh Rates**: The data refreshes periodically. You cannot adjust refresh intervals. However, you can adjust the interval for data logging.
- **Logging Data**: If file logging is enabled, historical data will be save to the json file for further analysis. If the file already exists, data will be appended, if not, a new json file will be created.

3. Technical Overview

3.1 Application Structure

- **Main Modules**:
  - system_monitor.rs: Core logic for monitoring CPU, memory, disk, and network statistics.
  - model.rs: Defines data structures and models used for handling system data.
  - utils.rs: Contains utility functions for data processing and formatting.

- main.rs: Initializes the application, sets up the GUI, and handles overall application flow.

3.2 Key Libraries

- **iced**: Used for building the GUI with a declarative approach, making it easy to manage dynamic data presentation.
- **sysinfo**: Provides data for system resources (CPU, memory, etc.) used in real-time monitoring.

3.3 Data Flow and Concurrency

- **Data Retrieval**: The application uses Rust's async capabilities to fetch system data at regular intervals without blocking the UI.
- **Concurrency**: Async tasks handle periodic data updates, leveraging Rust's lightweight task management.
- **File Logging**: If enabled, data is periodically saved to a file for historical tracking.

3.4 Future Improvements

Potential Improvements include:

- Adding graphical representations for data (e.g., charts).
- Enhanced error handling and modularity for code readability