Faculty of Engineering and Applied Science

SOFE 3200U Systems Programming

Final Project

Monitoring and Alerting System

Group Member 1

Name: Abdullah Ahmed

Student ID: 1008217099

Group Member 2

Name: Ronald Onyiorah

Student ID: 100887535

Group Member 3

Name: Malyka Sardar

Student ID:100752640

Group Member 4

Name: Mark Mikaeel

Student ID: 100830507

System Resources Monitoring Tool

Project Overview

Description:

This project is a Python-based application designed to monitor key system resources like CPU, memory, and disk usage. The tool lets users set custom thresholds for CPU and memory usage, and if the system exceeds these thresholds, the tool will send an alert via Slack. Additionally, we've incorporated stress testing for both CPU and memory, simulating system load while monitoring resource usage, making it a useful tool for anyone wanting to keep track of their system's health and performance.

Objective:

The goal of this project is to provide a simple yet effective solution for monitoring system resources. By setting custom thresholds for CPU and memory, the tool will automatically notify users through Slack if the system's usage exceeds these limits. It's perfect for system administrators or anyone interested in keeping their system running smoothly and efficiently.

Prerequisites

Software and Dependencies:

- Python 3.8 or higher: Ensure Python is installed on your system (version 3.8+).
- Docker: If you're using MacOS, you'll need Docker installed for environment setup.
- Python dependencies:
- Requests: This library is required for sending notifications via Slack.

Installation Process.

MacOS Setup:

- Navigate to the setup folder: `cd linux_environment_setup`

- Set the workspace for Linux: `./setMacOsWorkspace.sh`

- Activate the Python environment:

```
apt install python3-venv -y

python3 -m venv venv

source /workspace/venv/bin/activate
```

- Install the required requests library: `pip install requests`

Windows/Ubuntu Setup:

- Set up and activate Python environment:

```
apt install -y python3-venv

python3 -m venv myenv

source myenv/bin/activate
```

- Install the requests library: `pip install requests`

- Navigate to the project directory:

  `cd /path/to/project`

- Ensure the system resource script (*system_resources.sh*) is executable:

  `chmod +x backend/system_resources.sh`

- Run the project: To start the monitoring tool, run the following command:

  ```
  python3 frontend/main.py
  ```

Usage

How to Run the Script:

Once you've installed everything, simply run the following command in the terminal:

```
python3 frontend/main.py
```

This will launch the monitoring tool and prompt you to input a command.

Input/Output:

Input: You'll enter commands to monitor the system's CPU, memory, or disk usage, or to start continuous monitoring (*resources*).

Output: The tool will output the requested resource usage statistics, and if the thresholds you set are exceeded, alerts will be sent via Slack.

Examples:

- To monitor CPU usage once: `Enter a command: cpu`
- Output: `CPU Usage: 1.2%`
- To monitor all system resources continuously:

  ```
  Enter a command: resources

  Set CPU usage threshold (in %): 80

  Monitoring system resources with thresholds (CPU: 80%, Memory: 4
  Gi)...
  ```

Command-Line Arguments / Options

Here's a list of the available commands that can be used:

- **cpu**: Shows the current CPU usage once.
- **memory**: Shows the current memory usage once.
- **disk**: Shows the current disk usage once.
- **resources**: Continuously monitors CPU, memory, and disk usage, and alerts you if the thresholds are exceeded.
- **list**: Lists all available commands.
- **exit**: Exits the program.

Script Structure

Overview of the Code:

The project is structured into three main sections:

Main Program Flow: The *`main.py`* file drives the program by handling user input and controlling which functions are called.

System Resource Tracking: *`services.py`* collects system resource data, runs the shell script to gather usage information, and handles Slack notifications.

Stress Testing: The *`cpu_stress.py`* and *`memory_stress.py`* files simulate CPU and memory load, respectively, to test how the system handles under stress.

Key Functions:

`monitor_cpu_usage()`: Fetches and displays CPU usage once.

`monitor_memory_usage()`: Fetches and displays memory usage once.

`monitor_disk_usage()`: Fetches and displays disk usage once.

`monitor_system_resources()`: Continuously monitors the system resources, checks if thresholds are exceeded, and sends Slack alerts.

Slack Notification:

Whenever the CPU or memory usage exceeds the user-defined thresholds, the tool sends an alert to a Slack channel using the Slack API. The notification is triggered by the *send_slack_notification()* function in *services.py*.

Error Handling

Error Messages:

- **FileNotFoundError**: This happens if the log file can't be found or accessed. If that occurs, double-check that the script has permission to write to the log file. Also, confirm that the directory where the log file should be placed actually exists and is writable.
- **Invalid Command**: If you type a command the program doesn't recognize, it will let you know and politely ask you to enter one of the valid options. Think of it like the program saying, "Oops, that didn't work! Please try one of these instead."

Exit Codes:

- **0**: Everything worked just fine—no issues, all commands ran as expected.
- **1**: Something went wrong, and the program wasn't able to execute the command properly. This usually happens when there's a typo in the command or when the program can't access a resource it needs.

Testing

How to Test:

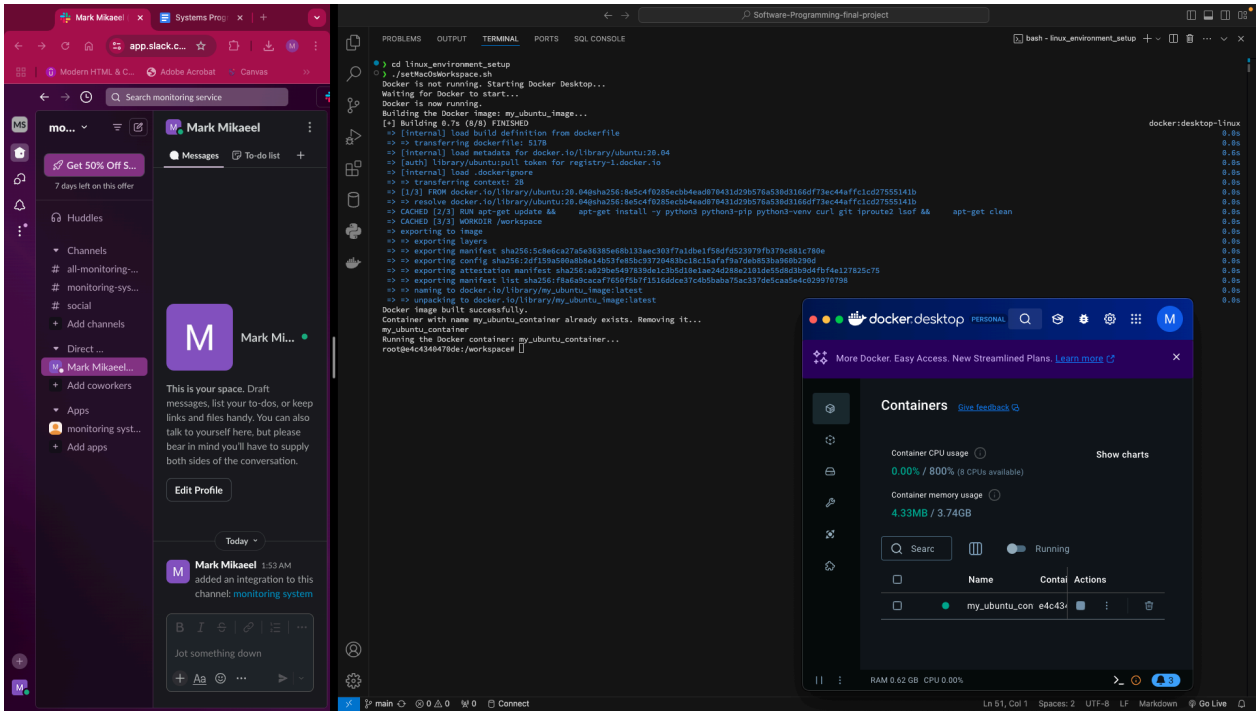
1. **Run the Program**: Start by running the tool with this command:

   ```
   python3 frontend/main.py
   ```

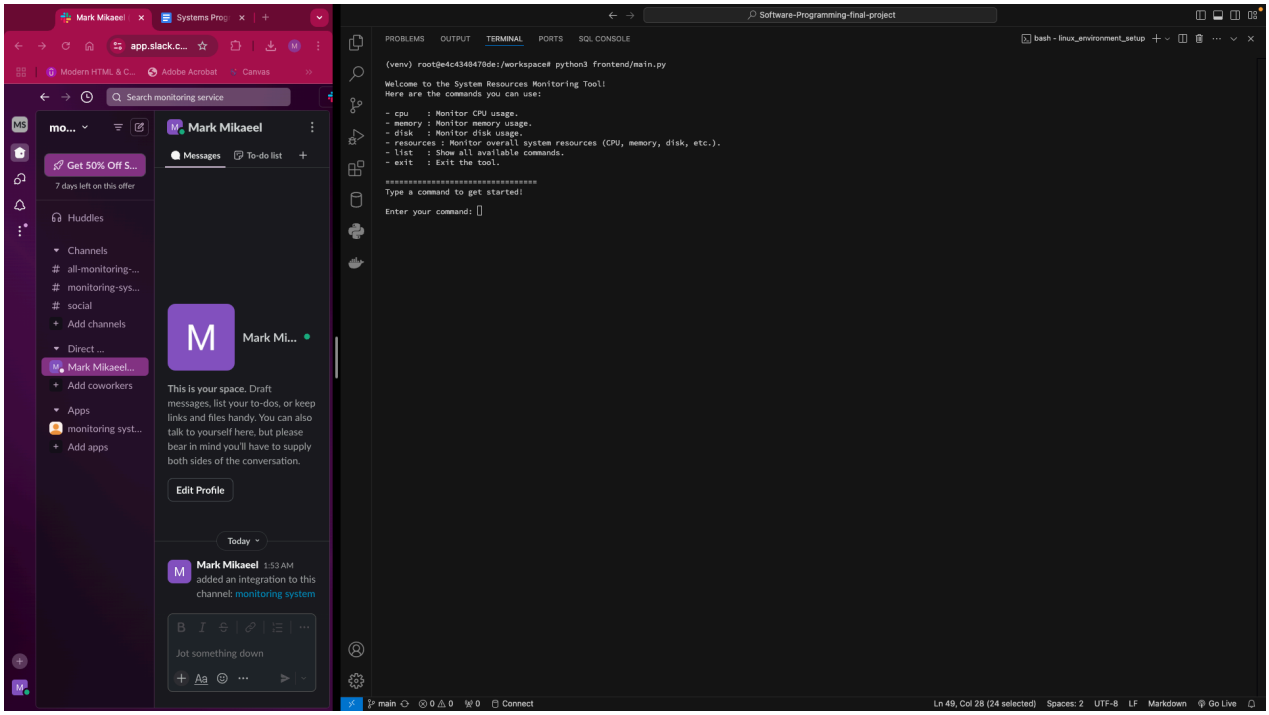
2. **Test the Commands**: Next, test the cpu, memory, and disk commands to check that they correctly show the resource usage.
3. **Test the resources Command**: Try out the resources command. Set different CPU and memory thresholds and see if the tool sends Slack notifications when the usage exceeds these limits. This is a good way to make sure everything is working, especially the alert system.

Screenshots of Testing

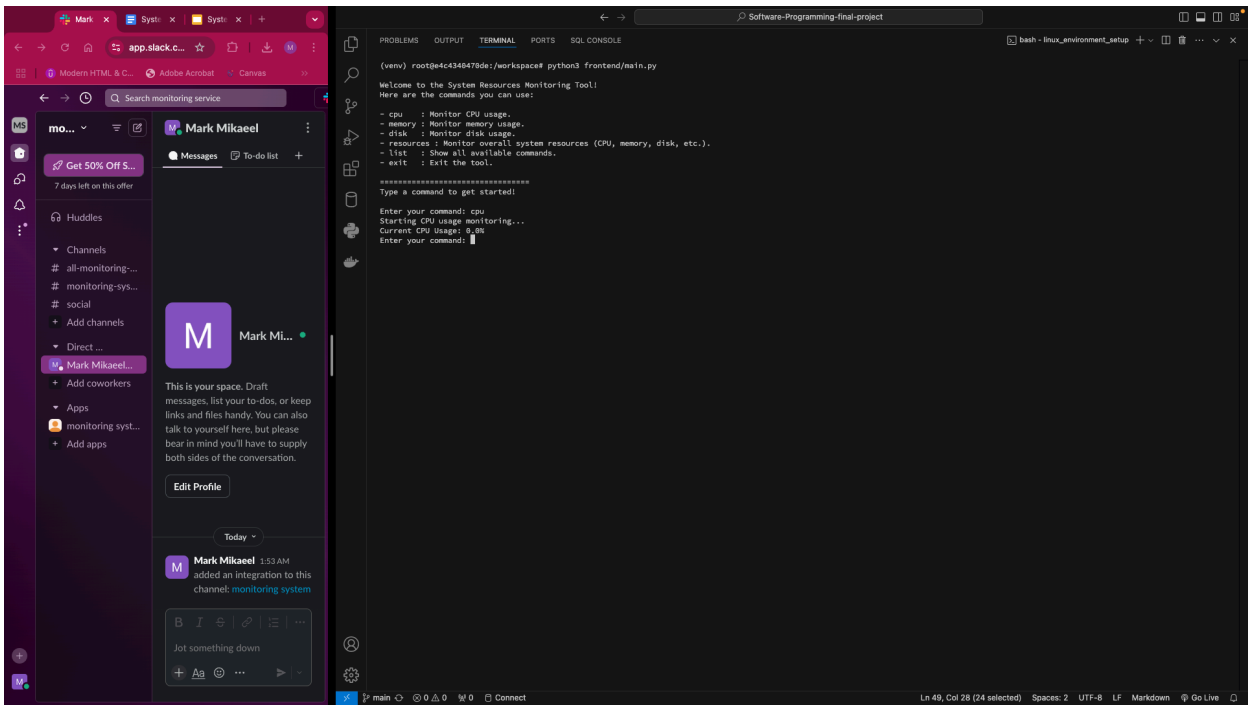
Script that launches Docker and docker linux container with python



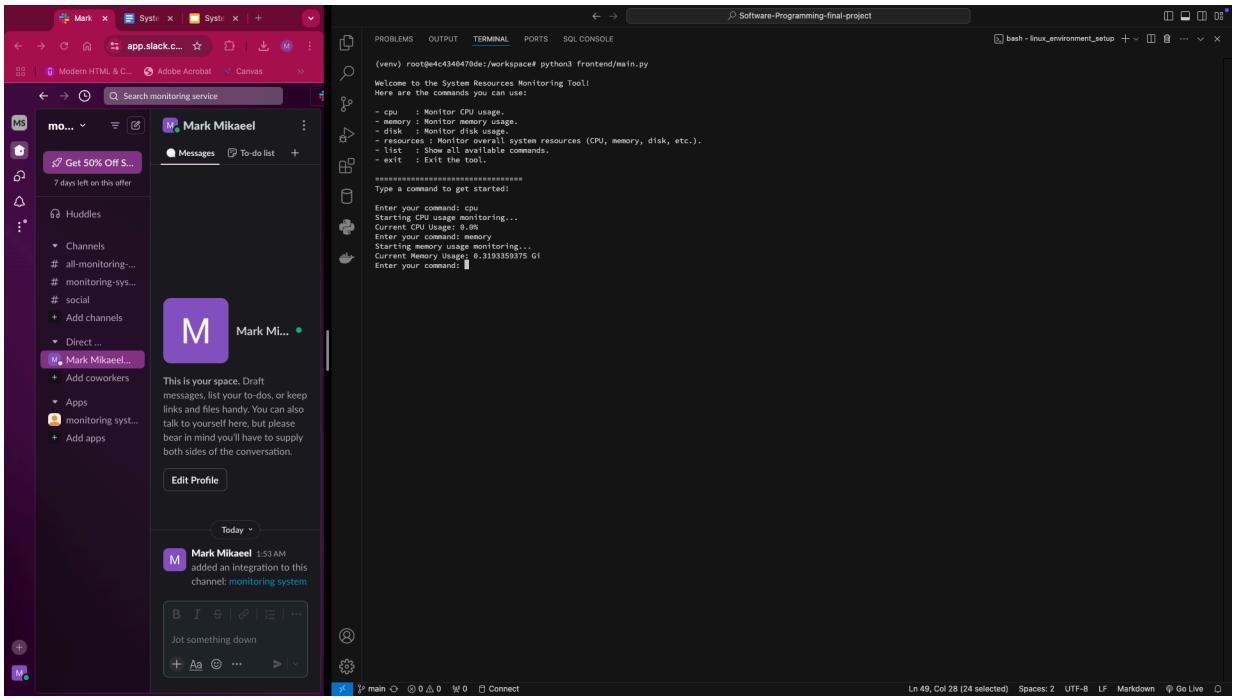
Welcome message



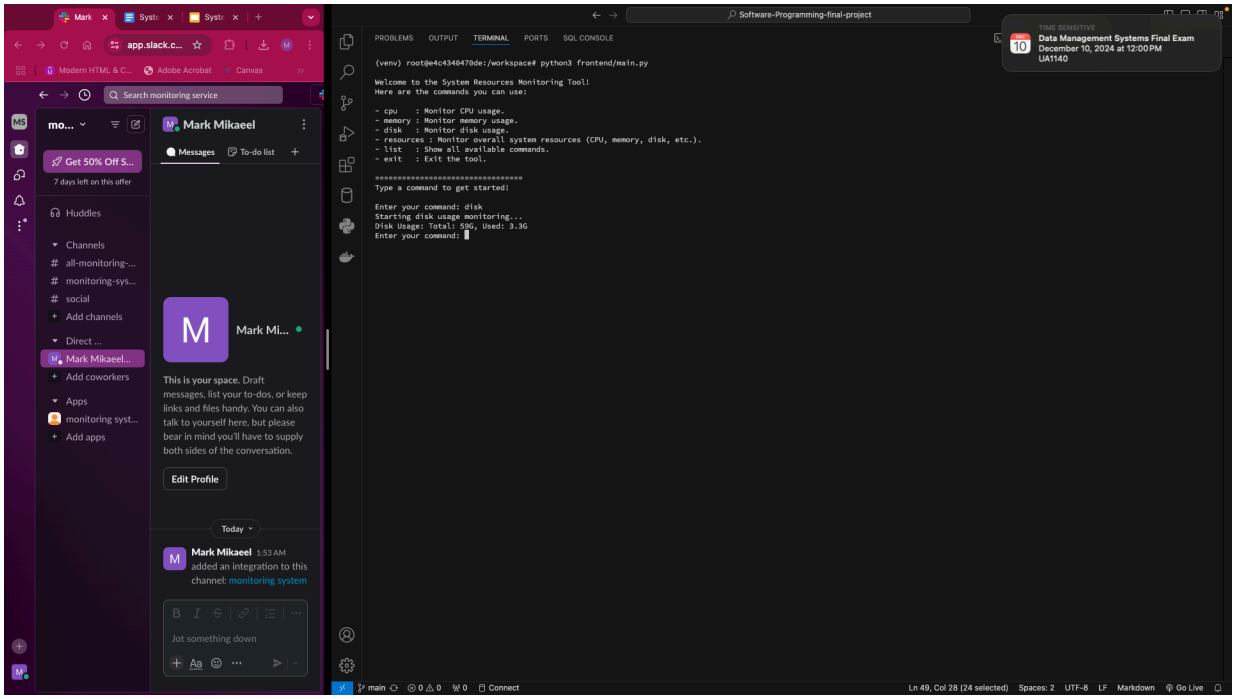
Show cpu usage



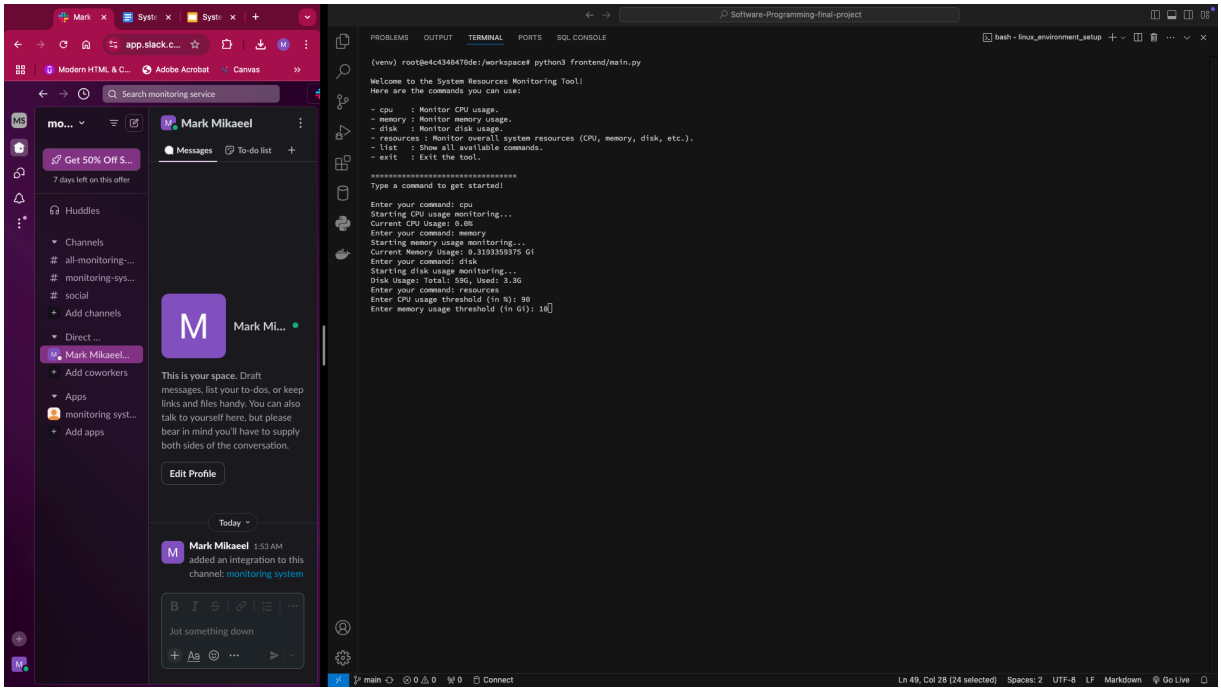
Show memory usage



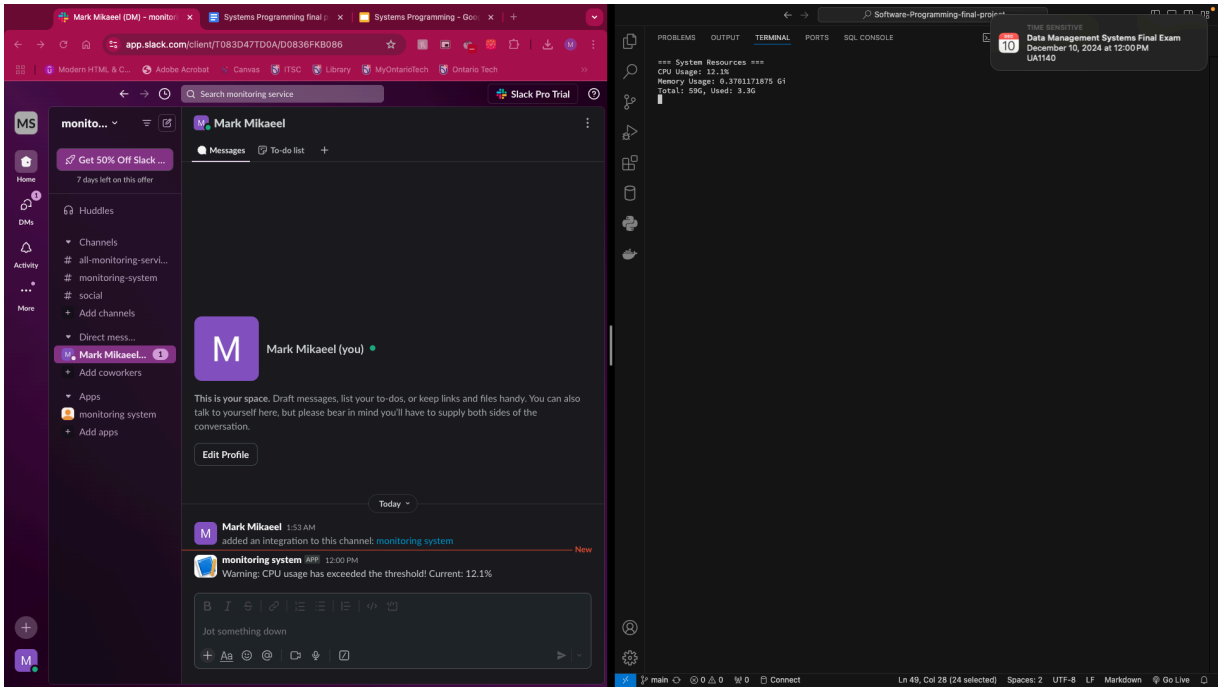
Show disk usage



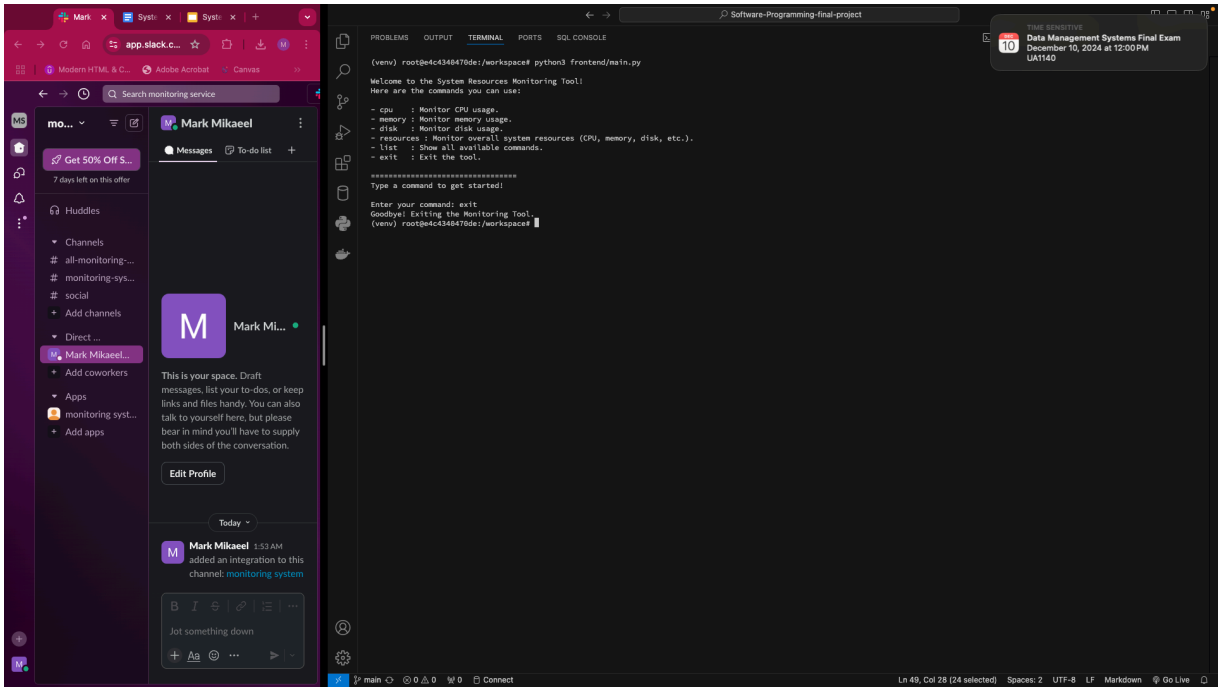
Prompt user to set thresholds



Show all resources in real time with slack notifications based on user set thresholds



Exit monitoring tool



Automated Testing:

At the moment, automated tests haven't been set up. So, you'll need to manually test each command to confirm everything is working as expected. Don't worry though—manual testing is pretty straightforward, and it'll help you ensure the program is running smoothly.

Additional Information

References:

- The official Python documentation for *subprocess* was extremely helpful for managing system processes and handling concurrency in the project.
- Slack's API documentation was essential for setting up the notifications feature.
- A variety of shell scripting tutorials were used to collect system resource data using common Linux commands like *top, free,* and *df.*