**Dynamic Resource Monitor: Project Report**

Submitted by: Yugandhar Patil

GitHub Repository - [github link](https://github.com/NidhishVyas/OS-Project)

**About the Project**

The Dynamic Resource Monitor is a real-time system monitoring application that combines backend metrics collection with frontend visualization to present critical system performance data. This project utilizes Python’s extensive libraries and React.js’s modern capabilities to deliver an intuitive and user-friendly experience.

**Objectives**

The primary objectives of this project were:

•To build a reliable monitoring tool that updates in real time.

•To present key system metrics in a clear and visually appealing format.

•To ensure the application is lightweight and responsive, minimizing its impact on system resources.

**How the group has managed to complete the project**

**Development Journey**

My work began with in-depth research on Python libraries such as psutil for collecting system data. Simultaneously, I explored how Flask APIs could communicate efficiently with a React frontend. Online forums, official documentation, and sample projects were invaluable resources during this phase.

**Backend Implementation**

As the primary contributor to the backend, my role was to fetch accurate system metrics and expose them through APIs. Key functionalities included:

•Calculating real-time disk and memory usage.

•Monitoring network speeds and uptime.

•Fetching and sorting top processes by resource consumption.

The psutil library was instrumental in gathering these metrics. Designing modular, reusable API endpoints was a key focus to ensure scalability and maintainability.

**Challenges and Solutions**

1.Backend Efficiency:

Fetching all system metrics simultaneously caused delays. By using asynchronous programming and optimizing data queries, I ensured that the backend could handle frequent updates efficiently.

2.CORS Errors:

During early integration tests, cross-origin requests were blocked. Adding the Flask-CORS package resolved this issue and allowed smooth API communication.

3.API Design Complexity:

Structuring the API responses to fit the frontend’s requirements took significant effort. I created modular endpoints, each focusing on specific metrics, to simplify frontend integration.

4.Data Visualization:

My teammate faced challenges rendering the data dynamically. I assisted by ensuring the API responses were consistent and included all necessary information for rendering.

**Personal Contributions**

**Backend Development**

I took ownership of the backend, focusing on:

•Fetching and processing system data using psutil.

•Implementing Flask routes to expose metrics via APIs.

•Ensuring data consistency and error handling during API calls.

**Teammate Contributions**

•Nidhish led the frontend development, building the React.js dashboard and integrating Chart.js for dynamic visualizations. His work ensured that the tool was not only functional but also visually engaging.

**References**

1.Flask documentation: https://flask.palletsprojects.com

2.Python psutil library: https://psutil.readthedocs.io

3.React.js documentation: https://reactjs.org/docs