



DoTTed, LLC.

"When you're in doubt, we figure it out"



SysMon+

PC Performance Monitoring for Windows

Development Team Design Document

MSCS710 Project

Spring 2024

Written By: Benjamin Fiore, Alec Lehmphul, and Nicholas Petrilli

Version History

Version	Authors	Description	Date
1.0.0	Nick Petrilli, Alec Lehmphul, Ben Fiore	Initial Version	2/3/2024
1.0.1	Nick Petrilli, Alec Lehmphul, Ben Fiore	“Final” Version	2/24/2024
1.0.2	Nick Petrilli, Alec Lehmphul, Ben Fiore	Final Version	5/5/2024

Table of Contents

Introduction	4
Objective and Approach	5
Approach	5
Development Methodology	5
Computing Resources	5
External Design	6
The “Glance” View	6
The “Detail” View	7
The “Process” View	8
Internal Design	10
Development Standard	10
Development Environment	10
Software Flow	11
Initialization	11
Metrics Collection	11
Data Aggregation	12
Database Storage	12
UI Data Extraction	14
Termination	14
Error Handling & Recovery	14
Testability	14
RESTful API Endpoints	15
Packaging	15
Security	16
Accessibility	16
Globalization	16
Risks and Dependencies	16
Definitions	17
References	18

Introduction

SysMon+ delivers an intuitive experience to access real-time system performance metrics for Windows x86-based laptops and desktops. *SysMon+* innovates the system performance monitoring application space by providing finer control over metric polling and a cleaner, easy-to-navigate and easy-to-customize user interface, all in a lightweight application that can be run on a wide range of systems and hardware combinations.

SysMon+ is compatible with all Windows 11/10 versions (except for ARM-based versions of Windows such as Windows 10 S), and provides users with metrics for a wide range of hardware and software components. Such components include:

- CPU: Processor name, processor current speed, processor max speed, number of cores, number of processes currently running, number of threads running, and overall utilization
- Memory: total memory installed, currently available/allocatable memory to system programs/processes, used memory, and overall utilization
- Disk: disk name, disk model, the current size of the swap file(s) in bytes, the current memory committed to the swap file(s) in bytes, the swap files utilization, total number of bytes read from the disk, total number of bytes written to the disk, the disk's read speed, disk write speed, and overall disk utilization
- Programs/processes*: Program/process name, process status, process' cpu percentage, process' memory usage in bytes, process' memory percentage, process disk speed, and process' disk percentage

** Programs/processes viewable to the user will only apply to actively running applications as determined by the SysMon+ application.*

SysMon+ provides access to these aforementioned system metrics through the use of a web-based user interface, which is accessible to the user through the web browser already installed onto their device. This helps *SysMon+* keep its application size low, while being able to be viewable on a wide range of devices with a wide range of screen ratios.

SysMon+ is fully runnable on the user's installed device, with no need to connect to the Internet for external servers. All data collected by *SysMon+* will be stored locally on the installed device's storage, providing maximum data privacy and security to the end user.

Objective and Approach

Approach

SysMon+ provides its secure, single-device metrics by storing all collected metrics in a database that is installed and configured on the end user's device (the installation and configuration of the database is handled by the installation of the application, requiring no manual database setup by the end user). When the user is ready to view the latest collected metrics, they can open the dashboard to see this data, as well as any data updates received by *SysMon+* in real time.

Development Methodology

SysMon+'s development will be carried out using the Agile approach.

Computing Resources

Computing resources required for the development of *SysMon+* are as follows:

- Sever: SQLite
- Frontend: Javascript, HTML5/CSS3
- Development PCs
 - Operating System: Windows
 - IntelliJ IDEA or VS Code
- Development Code
 - GitHub - Holds source code, manages version control, tracks defects and enhancements ("Issues")

Computing resources required for the user operation of *SysMon+* are as follows:

- Hardware
 - Operating System: Windows 10/11
 - CPU: No requirement
 - RAM: No requirement
 - HDD/SSD Space: *To be determined*
- Software: None (N/A)

External Design

The user interface for *SysMon+* will utilize the default web browser installed on the end user's device (NOTE: if no web browser is installed on the user's device and/or no web browser is detected on the user's device, then the installation will prompt the user to install a web browser before continuing installation).

To access the user interface, the batch file will automatically open the web page to the localhost:3000. Otherwise, users can manually enter the localhost:3000 URL to view it. Upon prompting the opening of the user interface, *SysMon+* will request the latest data collected from the local database. Once the data is retrieved, the graphical elements of the user interface will begin to load within the web browser.

The “Glance” View

Upon loading into the user interface, the user will be met with the “Glance” view, which will provide graphs of the current usage of each of the major hardware components on the end user's device (CPU, Memory/RAM, and Disk). The “Glance” view will also display some basic metrics for each hardware component, including how much of each hardware component is being utilized - this includes a color-coded backing of this utilization metric, which changes color based on how much of the hardware component is being utilized - as well as the top processes that are utilizing each hardware component. All line charts on the “Glance” View pages and “Detail” View pages are made with chart.js and react-chartjs-2.

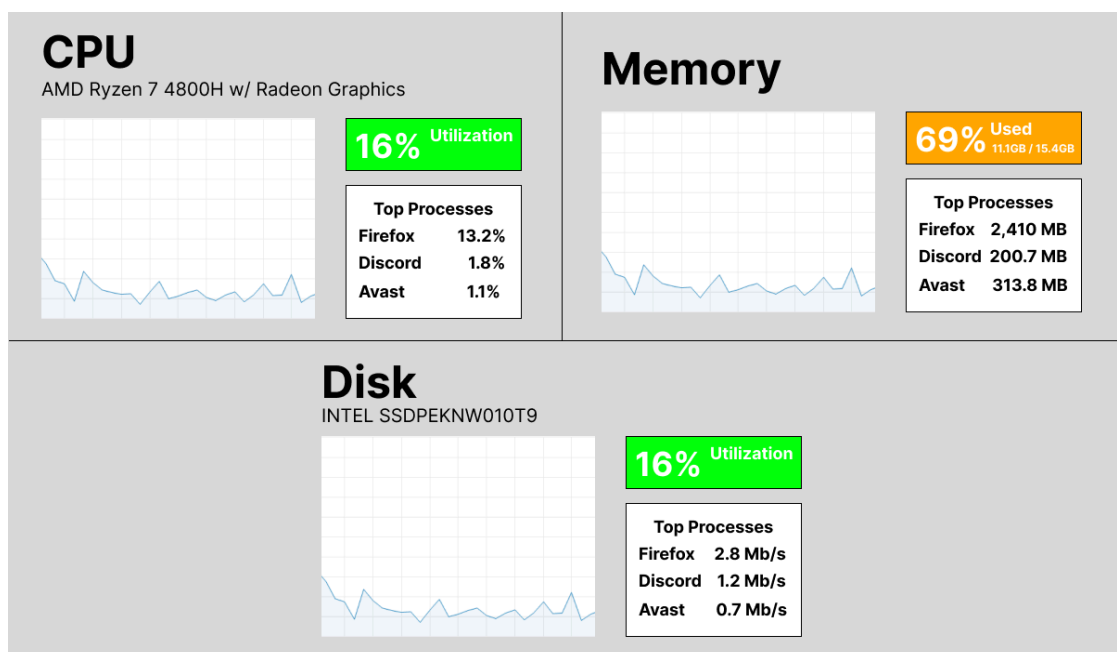


Figure A: The “Glance” View concept

Within the “Glance” view, the user can interact with different elements of the UI in order to access more detailed data of each hardware component.

The “Detail” View

If the user clicks on either the hardware component type (ie: CPU) or a hardware component’s graph, the user will enter the “Detail” view for that hardware component. Entering the “Detail” view for a hardware component will provide more information and detailed metrics for that given component. As seen in Figure B below, opening the “Detail” view for the CPU will allow the user to see a larger utilization graph (which will have more functionality than the smaller one on the “Glance” view), as well as the CPU’s current speed/frequency and its information.

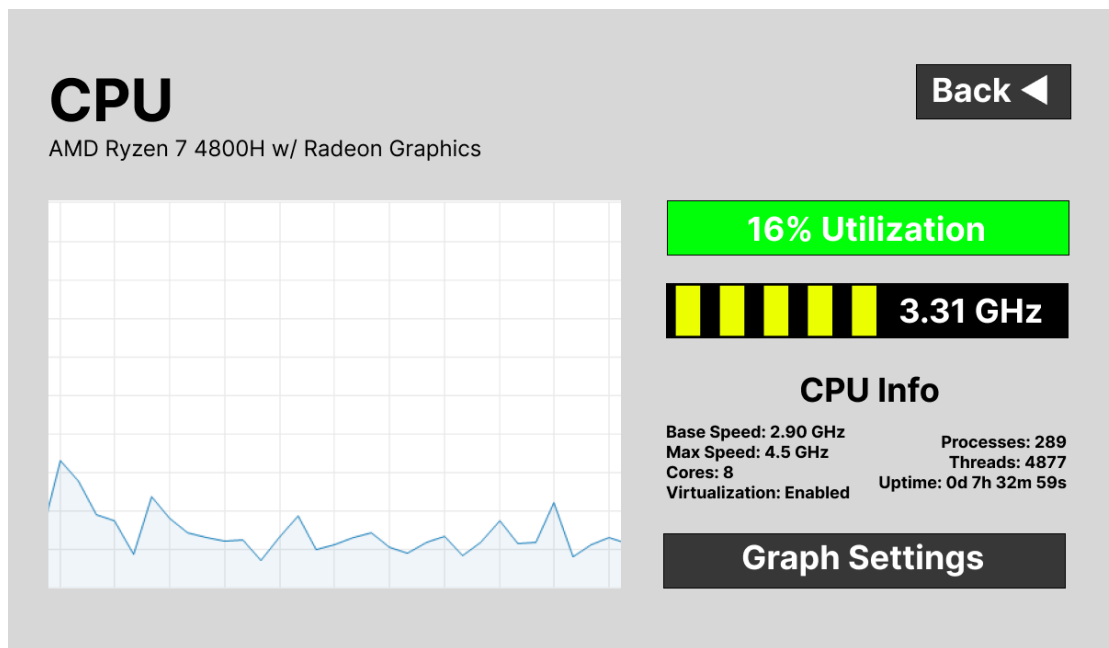


Figure B: The “Detail” View concept

The “Process” View

If the user clicks on the “Top Processes” section for any of the hardware components, the user will enter the “Process” view. Entering the “Process” view will allow the user to see the per-process/per-program usage of each of the hardware components on the system. When entering the “Process” view from any of the “Top Processes” tables on the “Glance” view, the “Process” view will automatically open in a sorted view for the hardware component that corresponds to the given “Top Processes” view. As seen in Figure C, below, opening the “Top Processes” view from the CPU

section of the “Glance” view will open a descending sort of the active processes/programs running on the user’s system per each process’s/program’s CPU usage.




Processes				
App Name	CPU 	RAM 	Disk 	
Firefox	12.2%	2,146.2 MB	0.0 Mb/s	
Avast Antivirus	0.8%	114.4 MB	0.0 Mb/s	
Microsoft Edge	0.8%	475.3 MB	0.0 Mb/s	
NVIDIA Ctrl Panel	0.0%	5.6 MB	0.0 Mb/s	
NVIDIA Container	0.0%	9.8 MB	0.0 Mb/s	
Runtime Broker	0.0%	19.7 MB	0.0 Mb/s	
AMD Software	0.0%	6.2 MB	0.0 Mb/s	
Notepad	0.0%	1.1 MB	0.0 Mb/s	
GNU Manipulation	0.0%	5.3 MB	0.0 Mb/s	
Office Click-N-Run	0.0%	4.0 MB	0.0 Mb/s	
NordVPN	0.0%	26.2 MB	0.0 Mb/s	
Gaming Services	0.0%	11.9 MB	0.0 Mb/s	
f.lux	0.0%	1.3 MB	0.0 Mb/s	

Figure C: The “Process” View concept

Internal Design

Development Standard

Development for *SysMon+* will follow the Agile development approach. The overall product objects are defined within the Objective and Approach section of this design document. *SysMon+* will continuously grow and evolve through numerous epics and phases, each involving tasks for designing, developing, and analyzing our software. The Agile Methodology emphasizes continuous growth, improvement, and collaboration which aligns with our team's professional and personal goals.

SysMon+ will be written using Java and the Spring framework for creating a backend API. This API will interact with an SQLite database which will store the countless process metrics outlined later in this design document. Information on these metrics can be found in the "Development" section. The frontend of this application will be a webpage utilizing HTML and JavaScript.

Hardware Resources

SysMon+ requires a machine running the Windows operating system. Since the backend will be packaged as an executable JAR, the user must also have the Java Virtual Machine (JVM) installed.

Development Environment

Compilers

SysMon+ requires Java 17 or JDK 17.

IDE

Tools and IDEs that will be useful when working with the *SysMon+* source code include:

- Backend - Java and Java Spring
 - ◆ IntelliJ IDEA Integrated Development Environment and VSCode
- API Testing
 - ◆ Postman
- Frontend - HTML / JavaScript / CSS
 - ◆ IntelliJ IDEA Integrated Development Environment and VSCode
- Database - SQLite

SysMon+ - PC Performance Monitoring for Windows

◆ SQLite Command Line Shell, SQLite Studio 3.4.4

Source Code Repository

The source code is hosted on GitHub at

<https://github.com/NickPetrilli/MSCS710-ProcessMonitor>.

Defect Tracker

GitHub Issues will be used as the product defect tracker. The following link contains the product's issues page, <https://github.com/NickPetrilli/MSCS710-ProcessMonitor/issues>.

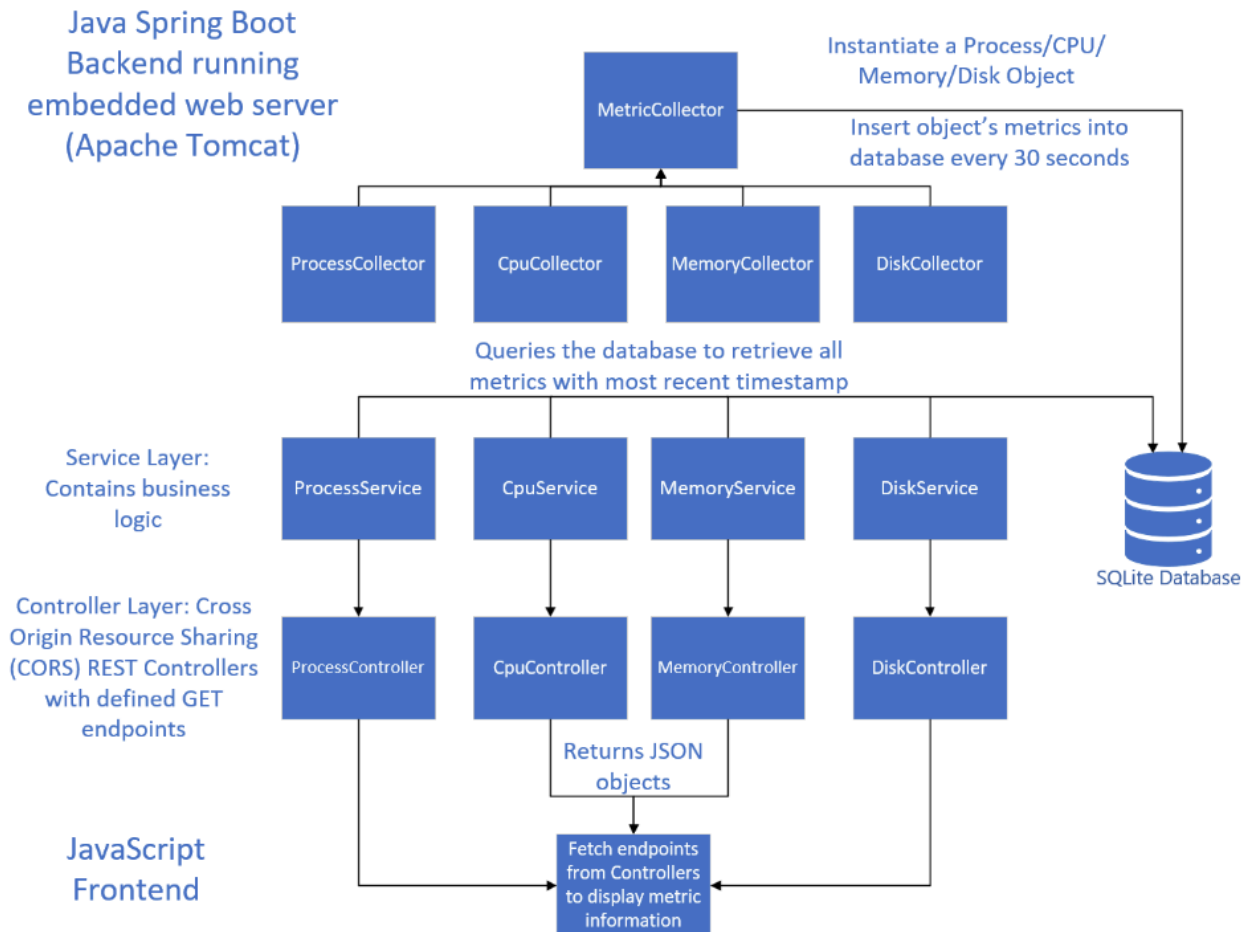
Build Process

Maven will be used as the project's build management and automation tool. All of the project's software dependencies can be found within the "pom.xml" file inside the GitHub repository. When the project is built, Maven pulls all the necessary JAR files and dependencies into the project. The React frontend is packaged into a "build" folder, which contains all Javascript code and necessary frontend dependencies.

Release Process

Download a .zip file on our GitHub. This zip file includes the packaged JAR file, the packaged React frontend (within the build folder), and two batch files that will run the application (provided that the necessary downloads are met).

Software Flow



Initialization

The main purpose of this application is to monitor and store system process information to allow a user to visualize their system's performance.

Once the software is downloaded, users can double click the runApplication.bat file to start the application. This batch file will open 3 command prompt windows: one to execute the JAR file to start the backend to collect metrics and store them in the database, another to install and build the React server to reach the frontend, and another that waits 15 seconds for the backend and frontend to boot up, and then will open up the web browser to view the React frontend.

Metrics Collection

The metrics to be collected are Process, CPU, Memory, Disk usage, which will be occurring every 30 seconds. We will be utilizing the OSHI ([Native] Operating System Hardware Information) library. The OSHI library provides a high level Java API for querying this information and abstracts the complexity of writing JNA (Java Native Access) code. CPU, Memory, and Disk metrics will consist of general information about said hardware. Process metrics will consist of general information (process id, name) as well as each individual process' cpu, memory, and disk usage. We will be using the latest version of OSHI, version 6.4.11, by including it as a dependency in our pom.xml file.

Database Storage

SysMon+ will store the end user's device metrics in the following SQLite tables. When collecting metrics every 10 seconds, our database tables will accumulate a lot of data, so we will be pruning the database every 5 minutes to delete records that are a week old to ensure adequate query performance for the frontend. All tables will be used for our "General View", containing the overall view of the system's utilization. The process table will be used for the "Process View" where metrics are listed on a per process basis. The other three tables (CPU, Memory, Disk) will be used for our "Detail View" to display more specific information about the hardware, and the totals of all the per process information.

Process Table

Name	Type	Null?	Default	Comments
<u>process_id</u>	INTEGER	N	N/A	Primary Key
<u>timestamp</u>	TEXT	N	N/A	Primary Key
name	TEXT	N	N/A	
status	TEXT	N	N/A	
cpuPercentage	REAL	N	N/A	
memUsageBytes	INTEGER	N	N/A	
memPercentage	REAL	N	N/A	
diskSpeed	REAL	N	N/A	
diskPercentage	REAL	N	N/A	

CPU Table

Name	Type	Null?	Default	Comments
<u>timestamp</u>	TEXT	N	N/A	Primary Key
name	TEXT	N	N/A	
speed	INTEGER	N	N/A	
maxSpeed	INTEGER	N	N/A	
cores	INTEGER	N	N/A	
processes	INTEGER	N	N/A	
threads	INTEGER	N	N/A	
utilization	REAL	N	N/A	

Memory Table

Name	Type	Null?	Default	Comments
<u>timestamp</u>	TEXT	N	N/A	Primary Key
totalMemory	INTEGER	N	N/A	
availableMemory	INTEGER	N	N/A	
usedMemory	INTEGER	N	N/A	
utilization	REAL	N	N/A	

Disk Table

Name	Type	Null?	Default	Comments
<u>timestamp</u>	TEXT	N	N/A	Primary Key

<u>name</u>	TEXT	N	N/A	Primary Key
model	TEXT	N	N/A	
swapTotal	INTEGER	N	N/A	
swapUsed	INTEGER	N	N/A	
swapUtilization	REAL	N	N/A	
totalReadBytes	INTEGER	N	N/A	
totalWriteBytes	INTEGER	N	N/A	
readSpeed	INTEGER	N	N/A	
writeSpeed	INTEGER	N	N/A	
utilization	REAL	N	N/A	

UI Data Extraction

The React frontend UI code will communicate with the backend RESTful API through GET requests to extract the necessary data and metrics. The JavaScript Fetch API will be utilized to communicate with the backend API's endpoints. The backend API returns a JSON object back to the frontend JavaScript code.

Termination

When the user closes the command prompt windows or aborts the service (via ctrl c), the backend Java Spring Boot and the React frontend will stop executing.

Error Handling & Recovery

The API will handle different errors and exceptions, returning different error response codes based. The frontend will respond to these different response codes. As our API only provides GET requests, all data will be fetched properly since the backend has already inserted the metrics. There shouldn't be any error responses because no data is being POSTed from the user to the server and only being fetched from the database in the correct format. In terms of handling these errors during our development, each of our classes log information and errors to our log file where we can fix anything that didn't work as intended.

Testability

To test the backend and the RESTful API, we will use Postman to send different requests to our API and make it function as intended. Also, utilizing Junit and Mockito, we have comprehensive test cases that ensure each endpoint returns a 200 OK status, returns JSON content and that the response body is not empty.

RESTful API Endpoints

The frontend will be interacting with a RESTful API built using Java Spring. Below are the following URL endpoints that we will be defining.

GET <http://localhost:8080/api/v1/cpu>

Returns the most recently stored CPU data from the database as a JSON object.

GET <http://localhost:8080/api/v1/cpu/top-processes>

Returns the top 3 processes based on CPU usage.

GET <http://localhost:8080/api/v1/cpu/processes>

Returns a list of all running processes sorted by cpu usage descending.

GET <http://localhost:8080/api/v1/cpu/processes-asc>

Returns a list of all running processes sorted by cpu usage ascending.

GET <http://localhost:8080/api/v1/cpu/chart>

Returns a list of cpu utilization to be plotted on the line chart.

GET <http://localhost:8080/api/v1/cpu/avg-util-5min>

Returns average cpu utilization in the past 5 minutes.

GET <http://localhost:8080/api/v1/cpu/avg-util-10min>

Returns average cpu utilization in the past 10 minutes.

GET <http://localhost:8080/api/v1/cpu/avg-util-15min>

Returns average cpu utilization in the past 15 minutes.

GET <http://localhost:8080/api/v1/cpu/avg-util-30min>

Returns average cpu utilization in the past 30 minutes.

GET <http://localhost:8080/api/v1/cpu/avg-util-1hour>

Returns average cpu utilization in the past 1 hour.

GET <http://localhost:8080/api/v1/cpu/avg-util-2hour>

Returns average cpu utilization in the past 2 hours.

- GET <http://localhost:8080/api/v1/cpu/avg-util-4hour>
Returns average cpu utilization in the past 4 hours.
- GET <http://localhost:8080/api/v1/cpu/avg-util-6hour>
Returns average cpu utilization in the past 6 hours.
- GET <http://localhost:8080/api/v1/cpu/avg-util-12hour>
Returns average cpu utilization in the past 12 hours.
- GET <http://localhost:8080/api/v1/cpu/avg-util-24hour>
Returns average cpu utilization in the past 24 hours.
- GET <http://localhost:8080/api/v1/memory>
Returns the most recently stored memory data from the database as a JSON object.
- GET <http://localhost:8080/api/v1/memory/top-processes>
Returns the top 3 processes based on memory usage.
- GET <http://localhost:8080/api/v1/memory/processes>
Returns a list of all running processes sorted by memory usage descending.
- GET <http://localhost:8080/api/v1/memory/processes-asc>
Returns a list of all running processes sorted by memory usage ascending.
- GET <http://localhost:8080/api/v1/memory/chart>
Returns a list of memory utilization to be plotted on the line chart.
- GET <http://localhost:8080/api/v1/memory/avg-util-5min>
Returns average memory utilization in the past 5 minutes.
- GET <http://localhost:8080/api/v1/memory/avg-util-10min>
Returns average memory utilization in the past 10 minutes.
- GET <http://localhost:8080/api/v1/memory/avg-util-15min>
Returns average memory utilization in the past 15 minutes.
- GET <http://localhost:8080/api/v1/memory/avg-util-30min>
Returns average memory utilization in the past 30 minutes.

- GET <http://localhost:8080/api/v1/memory/avg-util-1hour>
Returns average memory utilization in the past 1 hour.
- GET <http://localhost:8080/api/v1/memory/avg-util-2hour>
Returns average memory utilization in the past 2 hours.
- GET <http://localhost:8080/api/v1/memory/avg-util-4hour>
Returns average memory utilization in the past 4 hours.
- GET <http://localhost:8080/api/v1/memory/avg-util-6hour>
Returns average memory utilization in the past 6 hours.
- GET <http://localhost:8080/api/v1/memory/avg-util-12hour>
Returns average memory utilization in the past 12 hours.
- GET <http://localhost:8080/api/v1/memory/avg-util-24hour>
Returns average memory utilization in the past 24 hours.
- GET <http://localhost:8080/api/v1/disk>
Returns the most recent disk data stored in the database as a JSON object.
- GET <http://localhost:8080/api/v1/disk/top-processes>
Returns the top 3 processes based on disk usage.
- GET <http://localhost:8080/api/v1/disk/processes>
Returns a list of all running processes sorted by disk usage descending.
- GET <http://localhost:8080/api/v1/disk/processes-asc>
Returns a list of all running processes sorted by disk usage ascending.
- GET <http://localhost:8080/api/v1/disk/chart>
Returns a list of disk read and write speeds to be plotted on the line chart.
- GET <http://localhost:8080/api/v1/disk/avg-util-5min>
Returns average disk utilization in the past 5 minutes.
- GET <http://localhost:8080/api/v1/disk/avg-util-10min>
Returns average disk utilization in the past 10 minutes.

GET <http://localhost:8080/api/v1/disk/avg-util-15min>

Returns average disk utilization in the past 15 minutes.

GET <http://localhost:8080/api/v1/disk/avg-util-30min>

Returns average disk utilization in the past 30 minutes.

GET <http://localhost:8080/api/v1/disk/avg-util-1hour>

Returns average disk utilization in the past 1 hour.

GET <http://localhost:8080/api/v1/disk/avg-util-2hour>

Returns average disk utilization in the past 2 hours.

GET <http://localhost:8080/api/v1/disk/avg-util-4hour>

Returns average disk utilization in the past 4 hours.

GET <http://localhost:8080/api/v1/disk/avg-util-6hour>

Returns average disk utilization in the past 6 hours.

GET <http://localhost:8080/api/v1/disk/avg-util-12hour>

Returns average disk utilization in the past 12 hours.

GET <http://localhost:8080/api/v1/disk/avg-util-24hour>

Returns average disk utilization in the past 24 hours.

GET <http://localhost:8080/api/v1/process>

Returns a list of processes stored in the database at the most recent timestamp as a list of JSON objects.

GET <http://localhost:8080/api/v1/process/{name}>

Returns a list of processes stored in the database with the specified “name” field as a list of JSON objects.

Packaging

Java Spring Boot projects create a pom.xml file upon creation. This file defines information about the application such as dependencies and packaging type. Our application will be packaged in a JAR file (Java Archive), which encapsulates all Java code, project resources and metadata into one collection.

SysMon+ - PC Performance Monitoring for Windows

This JAR file that the project is packaged in will be an executable JAR file, in which the client's Windows system will wrap the Java application as a Windows service to start when the system boots up. This will allow for metrics collection immediately after start up, and not depending on the user starting it manually. For viewing the GUI, users will need to browse to the localhost:3000 to access the process information being queried from the local database server. We will create a batch file that is downloaded with the application that allows the user to create a desktop shortcut to easily open the React frontend.

Security

Our Java Spring Boot application uses an embedded web server called Apache Tomcat which runs on the local environment. Since we don't want to send sensitive process metrics across the Internet to be stored on a device other than the client's, the process metrics will always stay on a local server on the client's machine.

Accessibility

Our frontend GUI was specifically designed to be interacted with by any type of user. Users with less experience on the computer will be able to navigate through our UI as it is simple and easily accessible. The colors used in our application were picked to accommodate color-blind users, making sure the groups of colors are distinguishable from one another.

Globalization

Due to expense and time constraints, this project will only be implemented in English. Later versions will include translations for other languages.

Risks and Dependencies

Since this is a standalone project, there are no real external dependencies. However, there are internal dependencies such as our frontend developer depending on the backend developer to successfully collect the metrics and add to the database, and create endpoints to be able to query the data. Potential risks for this project could include procrastination or inability to learn the Java Spring Boot framework quickly enough and eventually not creating a project that we are proud of.

Definitions

REST or RESTful API

A REST (Representational State Transfer) API is an application programming interface that conforms to the constraints of REST architecture, allowing for interactions with RESTful web services. REST APIs are stateless, meaning that it does not remember or care about any previous interactions. REST APIs communicate using HTTP requests and responses (usually handled through JSON or some other format).

API Endpoint

A digital location, usually a URL, where an API receives requests about a specific resource on its server. The API endpoint acts as a point of contact between the API client and the API server.

Spring Framework

The Spring Framework is application framework and inversion of control container for the Java platform. The inversion of control or IoC container is the core of the Spring Framework. It creates and configures objects along with their dependencies, managing these objects' entire lifecycle with minimum input from the developer. In other words, the Spring Framework manages many components that make up your application for you, behind the scenes.

Spring Boot

Java Spring Boot is an open-source tool and framework used for creating microservices and web applications. In our case, we are using it for its web features in order to create a RESTful API. Refer to the "References" section below to find out more about Spring Boot directly from their website.

JNI

JNI is the Java Native Interface. It is the interface that allows Java code to interact with native written code (usually C/C++).

JNA

JNA is Java Native Access. Typically the JNI was used to interact with native code from Java and other C code, but JNA simplifies this process by allowing Java code to directly

call native code without the need for writing additional native code or dealing with the complexities of JNI. The JNA provides a more natural approach to accessing native libraries and facilitating the communication between Java and native code.

References

For more on the Spring Framework, refer to the link below.

<https://spring.io/projects/spring-framework>

For Spring Boot documentation and implementation, refer to the link below.

<https://spring.io/projects/spring-boot>

For SQLite documentation, refer to the link below.

<https://sqlite.org/docs.html>

For more on REST APIs, refer to the link below.

<https://www.redhat.com/en/topics/api/what-is-a-rest-api>

For more on API endpoints, refer to the link below.

<https://blog.postman.com/what-is-an-api-endpoint/>

For OSHI API documentation, refer to the link below.

<https://www.oshi.ooo/oshi-core-java11/apidocs/com.github.oshi/module-summary.html>

For JNI documentation and implementation, refer to the link below.

<https://www.baeldung.com/jni>

For JNA API documentation, refer to the link below.

[https://java-native-access.github.io/jna/4.2.1/overview-summary.html#:~:text=Java%20Native%20Access%20\(JNA\),additional%20JNI%20or%20native%20code.](https://java-native-access.github.io/jna/4.2.1/overview-summary.html#:~:text=Java%20Native%20Access%20(JNA),additional%20JNI%20or%20native%20code.)