Software Design Document:
Yet Another Metrics Collector (YAMeC)

C. Marcus DiMarco & Brendan Gibbons

MSCS710 Project

Spring 2025

Version	Description	Date
1.0.0	Initial version	Feb 8, 2025
1.1.0	Modified project purpose and refined goals, added process flow charts and UI mockups, added units for metrics collected in database	Feb 23, 2025
1.1.1	Named project to YAMeC, redefined database schema, added Figure 2, made function names appear in monospace text, split bulleted lists for readability, added sections on Accessibility and Internationalization	Feb 28, 2025

Table of Contents

1. Introduction

1.1. Purpose

This document outlines the software design for Yet Another Metrics Collector (YAMeC), a system metrics collector application. The application is designed to provide power users and system administrators with a lightweight and customizable solution for monitoring system performance and resource use over an extended period. System-level metrics (CPU usage, memory usage, etc.) are collected from the operating system using native C++ code and recorded them in a database for historical reference. Collected data is then viewable by the user using a web-based graphical user interface hosted on the local machine, with the application providing visual indicators of performance trends and insights on potential issues. The application will be built using Java, Maven, and Spring Boot, leveraging JNI for interfacing with the C++-based metric collection module.

1.2. Goals

- Accurate Metric Collection: Provide accurate and up-to-date system metrics.
- Metrics Timelining: Provide a historical timeline of system use and performance statistics.
- User-Friendly Web Interface: Offer a simple and intuitive web interface to view collected metrics.
- Modular Design: Employ a modular architecture to facilitate maintainability, testability, and future extensibility.
- Performance Efficiency: Minimize the performance impact of the metric collection process on the system.
- Intelligent Suggestions: Notes on current performance and resource issues and advise the user on potential actions to remedy them. Best practices are also suggested based on the status of the system.
- User Customizability: Allow the user to adjust behaviors of the application, such as how often it collects and prunes system metrics data.
- Cross-Platform Compatibility (Desired): Design the architecture to be adaptable to different operating systems (Linux, Windows, macOS), even if initial implementation focuses on one.

1.3. Target Audience

This document is intended for:

- Software developers involved in the implementation of the metrics collector application.
- Testers who will verify the application's functionality and performance.
- Operations and deployment teams who will be responsible for deploying and maintaining the application.
- Stakeholders interested in the technical design and architecture of the application.

2. Overall System Architecture

The application adopts a layered architecture, separating concerns and promoting modularity. The main components and their interactions are depicted in Figure 1:

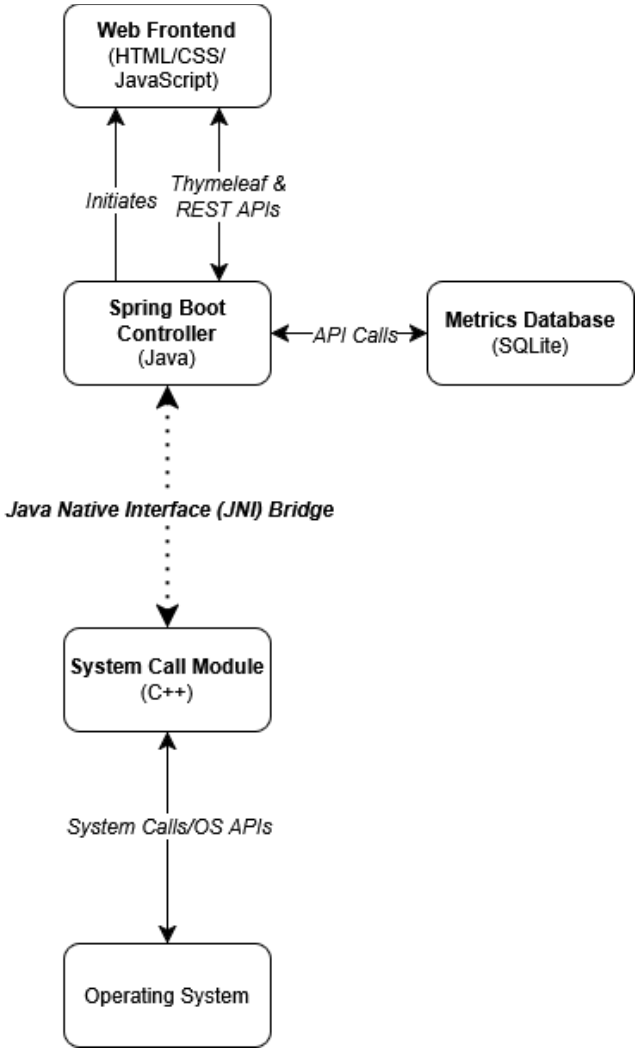

Figure 1 – Overall system architecture

2.1. Components:

- Web Frontend (User Interface): The client-side component, a standard web browser, used by users to access and view the collected metrics through the web application.
- Spring Boot Controller (metrics-collector-app): The core Java application.
    - Hosts the web application (using Thymeleaf).
    - Provides REST API for metric data.
    - Manages the overall application logic and workflow utilizing an MVC (Model, View, Controller) design pattern.
    - Interacts with the JNI Bridge to retrieve metrics.
- Metrics Database (SQLite): Provides long-term storage for historical system metrics
    - Spring Boot Controller interfaces with it via API calls
    - Web Frontend interfaces with it via calls to the Spring Boot Controller module
    - Refer to Section 4. Data Design for a more detailed discussion of the database schema
- JNI Bridge (metrics-collector-jni - JNI part in C++): A C++ layer acting as an intermediary between the Java application and the native C++ metrics module.
    - Provides JNI functions callable from Java.
    - Translates data between Java and C++ data types.
    - Calls functions in the C++ Metrics Module.
- C++ System Call Module (metrics-collector-jni – C++ part in system_metrics.cpp): Native C++ code responsible for collecting system metrics.
    - Utilizes OS-specific system calls or APIs to gather metric data (e.g., /proc/stat, `GetSystemTimes()`).
    - Focuses solely on metric collection logic, optimized for performance if needed.
- Operating System: The underlying operating system providing the data sources (files, APIs) for metric collection.

3. Module Design

3.1. yamec-app Module (Spring Boot Application)

3.1.1. Responsibilities:

- Web Application Hosting: Serve the web UI using an embedded web server (Apache Tomcat in Spring Boot).
- REST API Endpoint: Provide an API (/api/metrics) to expose metrics in JSON format.
- Data Presentation Logic: Prepare metric data for display in the web UI and API responses (View).
- JNI Interaction: Load the native library and call JNI methods to fetch metrics.
- Error Handling: Manage potential errors, especially related to JNI library loading and native method calls. This would include logging.

3.1.2. Key Components:

- com.gibbonsdimarco.yamec.app.MetricsCollectorApplication:
    - The main Spring Boot application class, responsible for application initialization.
- com.gibbonsdimarco.yamec.app.controller.MetricsController:
    - Handles HTTP requests.
    - Possible functions:
        - `getMetricsApi()`:
            - REST endpoint returning metrics as JSON (@ResponseBody).
        - `getMetricsPage()`:
            - Controller for the web UI, prepares data for the Thymeleaf template.
- com.gibbonsdimarco.yamec.app.service.MetricsService:
    - Abstraction layer for metric retrieval logic.
    - Possible functions:
        - `getSystemMetrics()`:
            - Calls JNI methods via SystemMetricsNative to get metrics. Processes and packages the metrics data into a Map.
- com.gibbonsdimarco.yamec.app.jni.SystemMetricsNative:
    - Java class declaring native methods that correspond to JNI functions in the yamec-jni module.
    - Handles loading the native library (`System.loadLibrary("metricsjni")`).

- Thymeleaf Templates (src/main/resources/templates/):
  - index.html:
    - Thymeleaf template to display metrics in a user-friendly HTML page.
- Static Resources (src/main/resources/static/):
  - css/style.css:
    - CSS for basic styling of the web UI.
  - js/app.js:
    - JS for fetching updates and adding dynamic behavior to the client.
- application.properties: Configuration file (port, application name, etc.).

3.1.3. Interfaces:

- REST API:
  - /api/metrics (GET) - Returns JSON payload of metrics.
- Web UI:
  - / (GET) - Serves the HTML page displaying metrics.
- JNI Interface:
  - Interaction with metrics-collector-jni via native methods defined in SystemMetricsNative.java.

3.2. yamec-jni Module (JNI and C++ Code)

3.2.1. Responsibilities:

- System Metric Collection:
  - Implement platform-specific logic to gather system metrics using C++ code.
- JNI Function Implementation:
  - Provide C++ JNI functions that Java can call.
- Data Conversion:
  - Convert data from C++ data types to Java-compatible data types within JNI functions.
- Native Library Building:
  - Compile C++ code into a shared library (.so or .dll).

3.2.2. Key Components:

- src/main/cpp/system_metrics.cpp:
  - Contains C++ functions for metric collection. Example functions may include: `get_cpu_usage()`, `get_total_memory()`.
  - Platform-dependent code will reside here, possibly using preprocessor directives (`#ifdef`, `#endif`) for different OS support.
- src/main/jni/SystemMetricsJNI.cpp:
  - C++ file containing JNI wrapper functions.
  - Example JNI functions:
    - `java_gibbonsdimarco_yamec_app_jni_SystemMetricsNative_getCpuUsage()`,
    - `java_gibbonsdimarco_yamec_app_jni_SystemMetricsNative_getTotalMemory()`.
  - These functions call corresponding C++ functions from system_metrics.cpp and handle JNI environment interaction and data type conversion.
- src/main/jni/SystemMetricsJNI.h (Optional - can be generated):
  - JNI header file defining function signatures for JNI implementations.
- pom.xml: Used by the Maven build manager to manage project settings and configurations, including dependencies, build settings, version, developer details, and so on.

3.2.3. Interfaces:

- JNI Interface:
  - Functions exposed to Java, defined in SystemMetricsNative.java and implemented in SystemMetricsJNI.cpp.
- C++ Function Interface:
  - Internal C++ functions within system_metrics.cpp module for metric collection, called by JNI functions.
- OS System APIs/Files:
  - Interaction with the operating system to retrieve metrics (e.g., system calls, reading files from /proc, Windows API calls).

4. Data Design

4.1. Metrics Collected

The application will initially collect the following system metrics:

- CPU Usage: Percentage of CPU time currently in use.
- Total Memory: Total system RAM available. (bytes)
- Used Memory/Free Memory (bytes)
- Disk Usage (Total, Used, Free for specific partitions) (bytes)
- Network Usage (bits sent/received on interfaces)

4.2. Data Structures

- C++ Code:
  - Native C++ data types will be used to collect metric values (e.g., double for CPU usage percentage, long long for memory in bytes).
- JNI Bridge:
  - Data will be converted between C++ native types and JNI types (e.g., jdouble, jlong).
- Java Application:
  - Metrics will be stored in java.util.Map within the MetricsService. The keys will be metric names (e.g., "cpuUsage", "totalMemory"), and values will be Java objects representing the metric values (e.g., Double, Long).
- JSON API Response:
  - Metrics exposed via the REST API will be formatted as a JSON object, mirroring the Map structure from the Java service layer.
- Thymeleaf Template Data:
  - The Map of metrics will be passed as a model attribute to the Thymeleaf template for rendering in the web UI.

4.3. Data Flow (Metric Retrieval)

1. Java code in MetricsService calls static native methods in SystemMetricsNative.java (e.g., `SystemMetricsNative.getCpuUsage()`).
2. This triggers a JNI call to the corresponding JNI function in SystemMetricsJNI.cpp (e.g., `java_com_gibbonsdimarco_yamec_app_jni_SystemMetricsNative_getCpuUsage()`).

3. The JNI function in C++ calls the appropriate C++ function in system_metrics.cpp (e.g., `get_cpu_usage()`).
4. The C++ function uses OS-specific methods to collect the metric data from the operating system.
5. The C++ function returns the metric value to the JNI function.
6. The JNI function converts the C++ data type to a JNI type and returns it back to the Java SystemMetricsNative class.
7. The Java MetricsService receives the metric value and stores it in the metrics Map.
8. The MetricsController retrieves the metrics Map and passes it to the web UI or returns it as JSON.

4.4. Database Design:

System metrics can be collected at both an application and a system level, depending on the type of metrics. For example, CPU usage time can be measured at an application level and a system level, whereas the CPU temperature can only be measured at a system level. These are all collected at the same time and are related to each other. So, to visualize this, all metrics connect to a single record with a shared timestamp in its own table.

Metrics can also be reduced in precision after a certain amount of time since they were collected. In this system, metrics are collected from the system every second. However, data is saved to the database in larger time intervals for the purpose of historical archiving without requiring an excess of database calls and causing poor performance. As a result, collected metrics all include timestamp and duration data. Further, to reduce storage overhead more, collected metrics can be condensed into longer timespans over time. These records are differentiated for the sake of data management by using a granularity label system. The assigned label also determines the duration of time new records of that granularity level should represent and the amount of time to wait until adjusting the data of that granularity level.

Metrics will be collected with association to the specific hardware components and not simply overall load across the entire system, which can be useful in diagnosing issues with a single device in a multi-device setup, such as when using a system with both Wi-Fi and ethernet support. Each device type has its own table, which each set of metrics collected being tied to those devices. Tables for devices and metrics collected are named after the device types, though for consistency and easier readability while programming, all table names and column names use camel-case, including any acronyms uses. For example, the table for CPU devices is named *Cpu*, and the tables for system and

application level CPU device metrics are called *CpuSystemMetrics* and *CpuApplicationMetrics*, respectively.

The relationships between the different tables of the database are illustrated in Figure 2 2 - Database Entity Relationship Diagram (No Attributes)Figure 2 2. The schema of each of the tables is described in the following subsections.

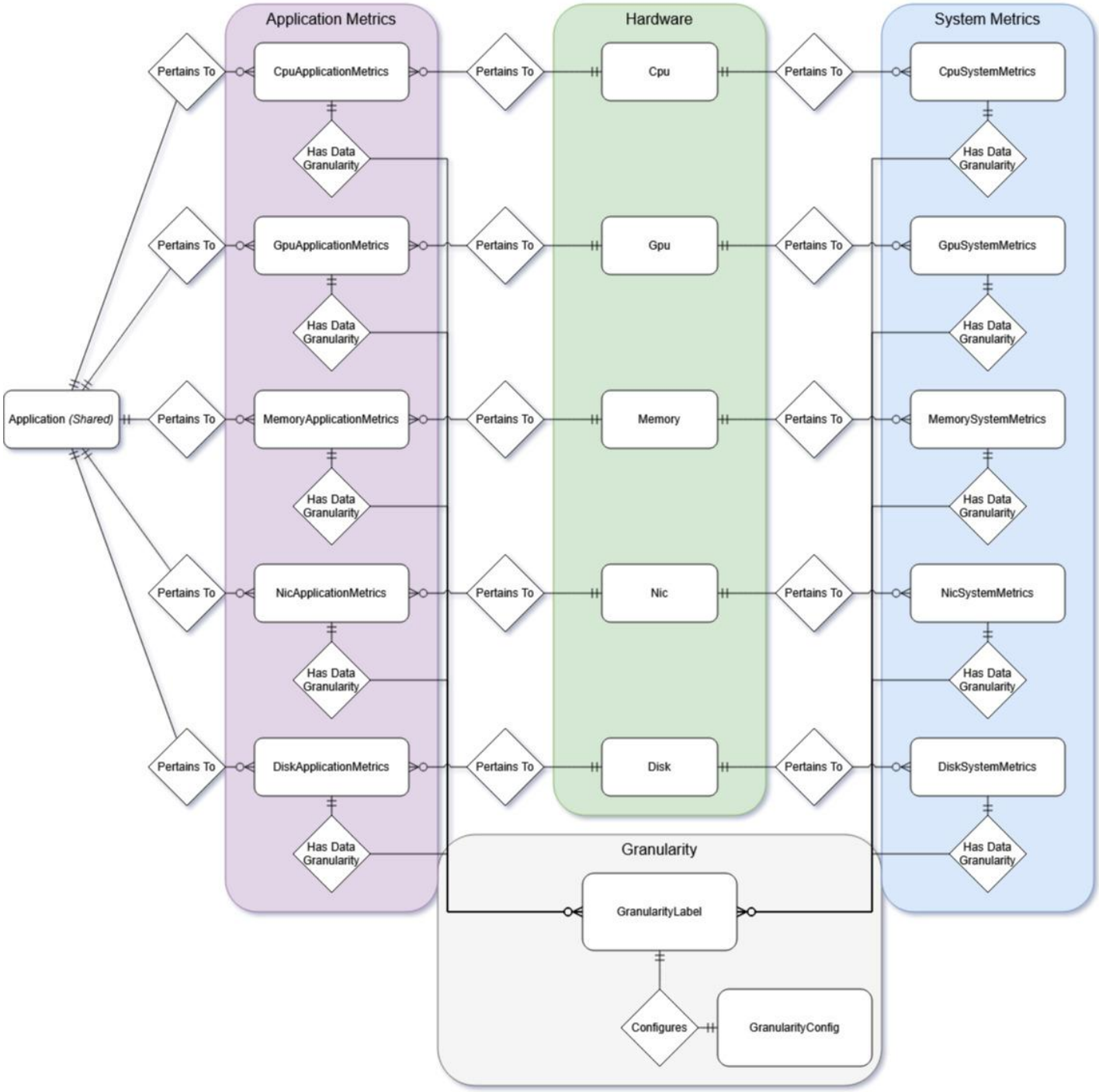

Figure 2 2 - Database Entity Relationship Diagram (No Attributes)

4.4.1. Application

- Records constant data about a specific process

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
Name	varchar(128)		The name of the application
Path	varchar(512)	Not Null	The location on the system which this application's executable is stored

4.4.2. Cpu

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
Name	varchar(255)		The product name (make and model) of the system processor
BaseSpeed	unsigned int		The standard clock speed in megahertz of the processor
Cores	unsigned int		The number of compute units/cores in the processor
LogicalProcessors	unsigned int		The number of compute threads the processor utilizes on a hardware level (hyperthreading/SMT)
Virtualization	tiny int (boolean)		Whether the processor supports virtualization technology or not
L1CacheSize	unsigned int		The capacity of L1 cache the processor in bytes
L2CacheSize	unsigned int		The capacity of L2 cache the processor in bytes
L3CacheSize	unsigned int		The capacity of L3 cache the processor in bytes

4.4.3. CpuApplicationMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Auto-increment Primary Key	
CpuId	unsigned int	Foreign Key (Cpu)	Links the metrics collected to a specific Cpu
ApplicationId	unsigned int	Foreign Key (Application)	Links the metrics collected to the specific application
Timestamp	timestamp		The starting time of the period which this metric represents
Duration	unsigned int		The number of seconds which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgUtilization	unsigned int		The average percentage of total CPU time used by the application within the measured timespan
MaxUtilization	unsigned int		The maximum percentage of total CPU time used by the application within the measured timespan
MinUtilization	unsigned int		The minimum amount of time the CPU is used by the application within the measured timespan

4.4.4. CpuSystemMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
CpuId	unsigned int	Foreign Key (Cpu)	Links the metrics collected to a specific Cpu
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgTemperature	unsigned decimal		The average temperature of the CPU in Celsius within the measured timespan
MaxTemperature	unsigned decimal		The maximum temperature of the CPU in Celsius within the measured timespan
MinTemperature	unsigned decimal		The minimum temperature of the CPU in Celsius within the measured timespan
AvgUtilization	unsigned int		The average percentage of total CPU time used by the entire system within the measured timespan
MaxUtilization	unsigned int		The maximum percentage of total CPU time used by the entire system within

		the measured timespan
MinUtilization	unsigned int	The minimum percentage of CPU time used by the entire system within the measured timespan

4.4.5. Gpu

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
Name	varchar(255)		The product name (make and model) of the system GPU
DedicatedMemory	unsigned int		The amount of memory in bytes dedicated to the GPU
SharedMemory	unsigned int		The amount of memory in bytes utilized by the GPU but shared with the rest of the system
GraphicsEngine	varchar(255)		The graphics engine version used by the GPU, by default (i.e., DirectX 12)

4.4.6. GpuApplicationMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
GpuId	unsigned int	Foreign Key (Gpu)	Links the metrics collected to a specific GPU
ApplicationId	unsigned int	Foreign Key (Application)	Links the metrics collected to the specific application
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgUtilization	unsigned int		The average percentage of total GPU time

		used by the application within the measured timespan
MaxUtilization	unsigned int	The maximum percentage of total GPU time used by the application within the measured timespan
MinUtilization	unsigned int	The minimum percentage of GPU time used by the application within the measured timespan
AvgDedicatedMemoryUse	unsigned int	The average amount of VRAM in bytes used by the application within the measured timespan
MaxDedicatedMemoryUse	unsigned int	The maximum amount of VRAM in bytes used by the application within the measured timespan
MinDedicatedMemoryUse	unsigned int	The minimum amount of VRAM in bytes used by the application within the measured timespan
AvgSharedMemoryUse	unsigned int	The average amount of shared graphics memory in bytes used by the application

			within the measured timespan
MaxSharedMemoryUse	unsigned int		The maximum amount of shared graphics memory in bytes used by the application within the measured timespan
MinSharedMemoryUse	unsigned int		The minimum amount of shared graphics memory in bytes used by the application within the measured timespan

4.4.7. GpuSystemMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
GpuId	unsigned int	Foreign Key (Gpu)	Links the metrics collected to a specific GPU
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
Temperature	unsigned decimal		The current temperature of the GPU in Celsius

AvgUtilization	unsigned int	The average percentage of total GPU time used by the entire system within the measured timespan
MaxUtilization	unsigned int	The maximum percentage of total GPU time used by the entire system within the measured timespan
MinUtilization	unsigned int	The minimum percentage of GPU time used by the entire system within the measured timespan
AvgDedicatedMemoryUse	unsigned int	The average amount of VRAM in bytes used by the entire system within the measured timespan
MaxDedicatedMemoryUse	unsigned int	The maximum amount of VRAM in bytes used by the entire system within the measured timespan
MinDedicatedMemoryUse	unsigned int	The minimum amount of VRAM in bytes used by the entire system within the measured timespan
AvgSharedMemoryUse	unsigned int	The average amount of shared

		graphics memory in bytes used by the entire system within the measured timespan
MaxSharedMemoryUse	unsigned int	The maximum amount of shared graphics memory in bytes used by the entire system within the measured timespan
MinSharedMemoryUse	unsigned int	The minimum amount of shared graphics memory in bytes used by the entire system within the measured timespan

4.4.8. Memory

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
Speed	unsigned int		The standard clock speed in transfers (bytes) per second of the memory
Formfactor	varchar(64)		The form factor or version of the memory (i.e., DDR5, LPDDR4, etc.)
Capacity	unsigned int		The amount of memory in bytes
SlotsUsed	unsigned int		The number of physical slots or channels currently in us

SlotsTotal	unsigned int	The number of physical slots or channels available (used or not) for memory
VirtualMemory	unsigned int	The amount of storage in bytes dedicated to page files and swap files, or virtual memory

4.4.9. MemoryApplicationMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
MemoryId	unsigned int	Foreign Key (Memory)	Links the metrics collected to a specific memory system
ApplicationId	unsigned int	Foreign Key (Application)	Links the metrics collected to the specific application
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgMemoryUse	unsigned int		The average amount of physical memory in bytes used by the application within the measured timespan
MaxMemoryUse	unsigned int		The maximum amount of

		physical memory in bytes used by the application within the measured timespan
MinMemoryUse	unsigned int	The minimum amount of physical memory in bytes used by the application within the measured timespan
AvgVirtualMemoryUse	unsigned int	The average amount of virtual memory in bytes dedicated to processes for this application within the measured timespan
MaxVirtualMemoryUse	unsigned int	The maximum amount of virtual memory in bytes dedicated to processes for this application within the measured timespan.
MinVirtualMemoryUse	unsigned int	The minimum amount of virtual memory in bytes dedicated to processes for this application within the measured timespan.

4.4.10. MemorySystemMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES

Id	unsigned int	Primary Key	
MemoryId	unsigned int	Foreign Key (Memory)	Links the metrics collected to a specific memory system
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgMemoryUse	unsigned int		The average amount of physical memory in bytes used by the entire system within the measured timespan
MaxMemoryUse	unsigned int		The maximum amount of physical memory in bytes used by the entire system within the measured timespan
MinMemoryUse	unsigned int		The minimum amount of physical memory in bytes used by the entire system within the measured timespan
AvgVirtualMemoryUse	unsigned int		The average amount of virtual memory in bytes dedicated to processes for this

		entire system within the measured timespan
MaxVirtualMemoryUse	unsigned int	The maximum amount of virtual memory in bytes dedicated to processes for this entire system within the measured timespan.
MinVirtualMemoryUse	unsigned int	The minimum amount of virtual memory in bytes dedicated to processes for this entire system within the measured timespan.

4.4.11. Disk

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
Name	varchar(255)		The product name (make and model) of the mass storage device
Type	varchar(64)		The formfactor or type of the mass storage device (i.e., SSD, HDD, etc.)
Capacity	unsigned int		The amount of storage in bytes
DriveLetters	varchar(64)		The letter(s) of the mass storage device in the system
ReadSpeed	unsigned int		The standard read speed in bytes per

			second of the mass storage device
WriteSpeed	unsigned int		The standard write speed in bytes per second of the mass storage device

4.4.12. DiskApplicationMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
DiskId	unsigned int	Foreign Key (Disk)	Links the metrics collected to a specific storage device
ApplicationId	unsigned int	Foreign Key (Application)	Links the metrics collected to the specific application
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgUtilization	unsigned int		The average percentage of time the disk is actively used by the application within the measured timespan
MaxUtilization	unsigned int		The maximum percentage of time the disk is actively used by the application within the

			measured timespan
MinUtilization	unsigned int		The minimum percentage of time the disk is actively used by the application within the measured timespan

4.4.13. DiskSystemMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
DiskId	unsigned int	Foreign Key (Disk)	Links the metrics collected to a specific storage device
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgUtilization	unsigned int		The average percentage of time the disk is actively used by the entire system within the measured timespan
MaxUtilization	unsigned int		The maximum percentage of time the disk is actively used by the entire system within the

		measured timespan
MinUtilization	unsigned int	The minimum percentage of time the disk is actively used by the entire system within the measured timespan
AvgResponseTime	unsigned int	The average response time in nanoseconds of the mass storage device within the measured timespan
MaxResponseTime	unsigned int	The maximum response time in nanoseconds of the mass storage device within the measured timespan
MinResponseTime	unsigned int	The minimum response time in nanoseconds of the mass storage device within the measured timespan

4.4.14. Nic

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
Name	varchar(255)		The product name (make and model) of the NIC
Type	varchar(64)		The type of network interface the device is (i.e., Wi-Fi 6e, Ethernet 10Gb, etc.)

4.4.15. NicApplicationMetrics

- Id (PK)
- NicId (FK)
- Connection Type (Protocol)
- Send Speed
- Receive Speed

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
NicId	unsigned int	Foreign Key (Nic)	Links the metrics collected to a specific storage device
ApplicationId	unsigned int	Foreign Key (Application)	Links the metrics collected to the specific application
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgSendBandwidth	unsigned int		The average bandwidth sent/outbound, in

		bits per second, from the specified NIC by the application within the measured timespan
MaxSendBandwidth	unsigned int	The maximum bandwidth sent/outbound, in bits per second, from the specified NIC by the application within the measured timespan
MinSendBandwidth	unsigned int	The minimum bandwidth sent/outbound, in bits per second, from the specified NIC by the application within the measured timespan
AvgReceiveBandwidth	unsigned int	The average bandwidth received/inbound, in bits per second, from the specified NIC by the application within the measured timespan
MaxReceiveBandwidth	unsigned int	The maximum bandwidth received/inbound, in bits per second, from the specified NIC by the application within the measured timespan
MinReceiveBandwidth	unsigned int	The minimum bandwidth

			received/inbound, in bits per second, from the specified NIC by the application within the measured timespan

4.4.16. NicSystemMetrics

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	
NicId	unsigned int	Foreign Key (Nic)	Links the metrics collected to a specific storage device
Timestamp	timestamp		The starting time of the period of data which this record represents
Duration	unsigned int		The number of seconds of data which this record represents
GranularityId	unsigned int	Foreign Key (Granularity)	The current granularity level
AvgSendBandwidth	unsigned int		The average bandwidth sent/outbound, in bits per second, from the specified NIC by the application within the measured timespan
MaxSendBandwidth	unsigned int		The maximum bandwidth sent/outbound, in bits per second, from the specified NIC by the application within

		the measured timespan
MinSendBandwidth	unsigned int	The minimum bandwidth sent/outbound, in bits per second, from the specified NIC by the application within the measured timespan
AvgReceiveBandwidth	unsigned int	The average bandwidth received/inbound, in bits per second, from the specified NIC by the application within the measured timespan
MaxReceiveBandwidth	unsigned int	The maximum bandwidth received/inbound, in bits per second, from the specified NIC by the application within the measured timespan
MinReceiveBandwidth	unsigned int	The minimum bandwidth received/inbound, in bits per second, from the specified NIC by the application within the measured timespan

4.4.17. Granularity

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
Id	unsigned int	Primary Key	

Label	varchar(255)		The name of the granularity level

4.4.18. GranularityConfig

COLUMN NAME	TYPE	CONSTRAINTS	NOTES
GranularityId	unsigned int	Primary Key Foreign Key (Granularity)	Each granularity level only has one configuration This schema separates the label from the user configurable state of the application
RecordTimespan	unsigned int	Not Null	The amount of time in seconds a record at this granularity level represents, by default
TimeToAge	unsigned int		The amount of time in seconds a record at this granularity level takes to age (its granularity decreases, or it is deleted)

5. User Interface Design

5.1. Web UI Description:

The web UI will be a simple, locally hosted web page accessible through a web browser. It will display the collected system metrics in a tabular format.

5.2. UI Elements:

- Page Title: "System Metrics"
- Metric Table:

    - Two columns: "Metric" and "Value".
    - Rows for each metric (CPU Usage, Total Memory, etc.).
    - Metric values will be displayed with appropriate units (e.g., "%" for CPU usage, "MB" or "GB" for memory).
- Error Display: If there are errors during metric collection (especially JNI related), an error message will be displayed prominently on the page.

5.3. UI Technology:

- Thymeleaf: Server-side templating engine to dynamically generate the HTML page with metric data.
- HTML: Structure of the web page.
- CSS: Basic styling for readability (e.g., table formatting, font styles).
- JavaScript will be added for dynamic updates (auto-refreshing metrics) or more interactive visualizations (charts). However, the initial design will focus on a simple static page refreshed by the user.

6. JNI Design

6.1. JNI Function Naming Convention:

JNI function names will follow the standard JNI naming convention:

Java_<package_name>_<class_name>_<method_name>

Example:

Java_com_gibbonsdimarco_yamec_app_jni_SystemMetricsNative_getCpuUsage

6.2. JNI Function Signatures:

JNI function signatures will map Java method parameter and return types to corresponding JNI types. For this application:

- Java methods will be static, native methods in SystemMetricsNative.java.
- Return types will be primitive types (double, long) or potentially strings for error messages.
- No arguments are expected for the initial metric retrieval methods. JNI functions will typically have JNIEnv* env and jclass clazz as arguments.

6.3. Data Type Conversion:

JNI layer will handle conversion between Java and C++ data types:

- double (Java) <--> jdouble (JNI) <--> double (C++)
- long (Java) <--> jlong (JNI) <--> long long (C++) (or appropriate C++ integer type)
- String handling (if needed for error messages or future text-based metrics) will involve JNI string functions (NewStringUTF, GetStringUTFChars, ReleaseStringUTFChars).

6.4. Error Handling in JNI:

- The C++ code will be designed to handle potential errors during system calls gracefully (e.g., checking return values, handling file access issues).
- If errors occur in the C++ code, the JNI functions should:
    - Return a special value (e.g., -1 for numeric metrics, or NULL if returning a String) or
    - Throw a Java exception (less preferred for performance in frequent metric retrieval but might be suitable for critical errors).
- Java code in MetricsService will check for error values or handle potential exceptions from JNI calls and display error messages in the web UI.

7. C++ Code Design (System Metrics Collection)

7.1. Metric Collection Methods:

The C++ code in system_metrics.cpp will utilize OS-specific methods for each metric. For example, on Windows, metrics can be collected utilizing the following C++ methods:

- CPU Usage: `GetSystemTimes` fills a buffer for the time the CPU is idle, the time spent running kernel code, and the time spent running user code. Utilization can be measured by polling it over some interval of time, taking the difference between the returned times, and calculating the percentage of time which the system is not idle.
- Total Memory: `GlobalMemoryStatusEx` fills a buffer value which indicates the status of the memory, including the percent of memory usage, the size and free space of the paging file, the size and free space of physical memory, and the size and free space of virtual memory

7.2. Performance Considerations:

- Metric collection functions in C++ should be designed to be efficient and minimize overhead.
- Minimize file I/O and system call overhead.
- Consider caching or sampling strategies if very frequent metric updates are required and performance becomes a concern (though for a simple application, this might not be necessary initially).

7.3. Platform Dependency Management:

- Use preprocessor directives (#ifdef, #elif, #endif) extensively in system_metrics.cpp to isolate platform-specific code sections.
- Create separate source files or folders for platform-specific implementations if the code difference is substantial.
- Build processes (Maven, build scripts) should be configured to compile the correct C++ source files based on the target operating system during native library compilation.

8. Non-Functional Requirements

- Performance: Metric collection should be fast and have minimal impact on overall system performance. Web UI should load quickly and be responsive.
- Reliability: The application should be robust and handle errors gracefully, especially in the JNI and native metric collection layers.
- Maintainability: Modular design and clear code structure will enhance maintainability.
- Security: For a locally hosted application, security is less critical, but best practices should still be followed (e.g., avoid storing sensitive information, be mindful of potential vulnerabilities in web UI dependencies if any are added in the future).
- Platform Support: Initial development may target a specific platform (e.g., Windows). Design should facilitate adding support for other platforms (Linux, macOS) in the future.

9. Technology Stack and Tools

- Programming Languages: Java, C++, HTML/CSS/JS

- Database Layer: SQLite
- Build Tool: Maven
- Framework: Spring Boot (for Java application), Thymeleaf (for web UI templating)
- JNI: Java Native Interface
- Operating Systems (Target): Initially Windows (with potential for Linux and macOS support)
- Development Tools:
    - IDE (IntelliJ IDEA)
    - JDK (Java Development Kit)
    - GCC/G++ (or platform-specific C++ compilers)
    - Maven
    - IzPack (.msi build tool)

10. Build and Deployment

10.1. Build Process:

1. Compile Native Library (yamec-jni):
    - Maven build in yamec-jni module will use exec-maven-plugin or a more integrated approach to compile C++ code into a shared library (.so or .dll). Platform-specific compilation settings will be configured (e.g., compiler flags, libraries to link).
    - The native library will be placed in yamec-jni/target/native/.
2. Package Spring Boot Application (yamec-app):
    - Maven build in yamec-app module will package the Java code, resources, and dependencies into an executable JAR file using spring-boot-maven-plugin.
    - Native Library Packaging: The native library from yamec-jni/target/native/ should be included in the Spring Boot JAR artifact. This could be done by copying it into the JAR structure using Maven plugins during the package phase, perhaps placing it within a native/ directory inside the JAR.
3. Parent POM (metrics-collector-parent): Orchestrates the build process for both modules. This will also contain the IzPack Maven plugin which will be responsible for building the final .msi for distribution.

10.2. Deployment Considerations:

- Running the Application: A user will run the .msi installer, which will unpack and install the .jar, .dlls, and check that the user has a suitable JRE installed on their machine and install one if not. Then, a shortcut will be added to their desktop and Programs.
    - To run the packaged JAR, the Java Virtual Machine needs to be able to find the native library.
        - java.library.path: Setting the java.library.path system property when launching the JAR is a common approach. This property needs to point to the directory where the native library resides (either externally on the filesystem or within the JAR).
        - Library Loading from JAR (Advanced): A more robust deployment approach would be to extract the native library from within the JAR to a temporary location at runtime and then load it. This involves more complex JAR manipulation and native library extraction logic.

- Platform-Specific Deployment (Future Enhancement): For different operating systems, different native libraries will be built (.so for Linux, .dll for Windows, .dylib for macOS). The deployment process needs to account for this, potentially requiring platform-specific packaging and distribution.

11. Accessibility

The web interface will make efforts to use high contrast and colorblind-safe colors where applicable. However, during initial development, this will be the limitation of accessibility efforts in the system.

12. Internationalization

The web interface will currently only be developed in US English.

13. Future Enhancements (Optional)

- Expanded Metric Set: Add more system metrics (disk I/O, network statistics).
- Metric History and Storage: Integrate an open-source monitoring toolkit (e.g., utilizing InfluxDB or Prometheus) to store historical metric data, enabling trend analysis and time-series visualizations.
- Advanced Web UI:
    - Real-time metric updates using WebSockets or Server-Sent Events (SSE).
    - Interactive charts and graphs for data visualization (using JavaScript charting libraries).
    - User configuration for selecting metrics to display and refresh intervals.
- Alerting: Implement alerting based on metric thresholds.
- Remote Monitoring: Enable remote access to the application and metric data with appropriate security measures (authentication, authorization).
- Containerization: Utilize Docker to ensure availability and portability of the application, regardless of user environment. This would ensure consistent packaging and configuration.

14. Open Issues and Risks

- JNI Complexity: JNI development can be complex and error-prone. Careful attention to memory management, data type conversion, and exception handling are required in the JNI layer.
- Platform Dependency: Developing and maintaining platform-specific C++ code for metric collection adds complexity. Thorough testing on target platforms is essential.
- Native Library Build Integration: Setting up a robust and cross-platform build process for the native library within the Maven build environment can be challenging.
- Error Handling in Native Code: Robust error handling in the C++ code is critical to prevent crashes or unexpected behavior.
- Security Considerations for Future Enhancements: If remote access or more advanced features are added, security will become a more significant concern.