# DoTTed, LLC.

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

# *SysMon+*

**PC Performance Monitoring for Windows**

**Test Approach 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 | Test Approach Document - Initial Version | 4/5/2024 |

# Table of Contents

**Objective** **4**
**API Tests** **5**
CPU Controller Tests 5
Memory Controller Tests 5
Disk Controller Tests 5
Process Controller Tests 6
**Database Tests** **7**
DataSourceConfig 7
DatabaseTest 7
CpuDatabaseTests 7
MemoryDatabaseTests 8
DiskDatabaseTests 8
ProcessDatabaseTests 8
**Frontend/UI Tests** **10**
**References** **11**

## Objective

*SysMon+* provides a secure, single-device metrics collection and visualization 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).

This document outlines how *SysMon+* will be tested to ensure that all fundamental components of the application - both backend and frontend - function as-intended. This document will detail what tests will be executed and what function(s) each test will check. Testing for the current version of *SysMon+* will include areas such as the application’s API connections, the application’s database, and the interactable user interface (UI).

## API Tests

Within our Spring Boot application we have four controllers that define our mappings to API endpoints. These endpoints handle our data metrics retrieval for the CPU, Memory, Disk and Processes. In order to test these endpoints, we are utilizing Spring Boot starter test and JUnit 5. Each class is annotated with @SpringBootTest, and each individual test is annotated with @Test. The Spring framework for testing allows the creation of a MockMvc to mock the Spring MVC infrastructure and simulate HTTP requests to our controller endpoints, without needing to deploy to a server. All tests expect a 200 OK response and expect the content type returned is JSON. Below are the tests that have been defined in the current test suite:

### CPU Controller Tests

- testGetCpuData() - Tests retrieving all cpu metrics for glance and detail views
- testGetCpuTopProcesses() - Tests retrieving cpu top processes for glance view
- testGetCpuProcessesDesc() - Tests retrieving process list sorted by cpu usage in descending order
- testGetCpuProcessesAsc() - Tests retrieving process list sorted by cpu usage in ascending order
- testGetCpuChartData() - Tests retrieving list of cpu utilization to display on chart

### Memory Controller Tests

- testGetMemoryData() - Tests retrieving all memory metrics for glance and detail views
- testGetMemoryTopProcesses() - Tests retrieving memory top processes for glance view
- testGetMemoryProcessesDesc() - Tests retrieving process list sorted by memory usage in descending order
- testGetMemoryProcessesAsc() - Tests retrieving process list sorted by memory usage in descending order
- testGetMemoryChartData() - Tests retrieving list of memory utilization to display on chart

### Disk Controller Tests

- testGetDiskData() - Tests retrieving all disk metrics for glance and detail views
- testGetDiskTopProcesses() - Tests retrieving disk top processes for glance view
- testGetDiskProcessesDesc() - Tests retrieving process list sorted by disk usage in descending order

- testGetDiskProcessesAsc() - Tests retrieving process list sorted by disk usage in ascending order
- testGetDiskChartData() - Tests retrieving list of disk read/write speed to display on chart

### Process Controller Tests

- testGetProcessData() - Tests retrieving list of all running processes

## Database Tests

In order to perform database tests, we created a new test database using the Spring Boot DriverManagerDataSource. This test database is only active when our “test” profile is active, and ensures a consistent environment for testing and isolation from production data. Using that created datasource, we can establish a connection to the test database and run necessary queries to test our application. Each test class is annotated with @SpringBootTest(classes = DataSourceConfig.class), which specifies the use of the test database, and each individual test is annotated with the JUnit @Test annotation.

We have defined three tests that are done on each database table, mimicking what is done in our application: testInsertAndReadData, testDeleteData, and testInsertAndReadMultipleData. The testInsertAndReadData tests that we can properly insert a record into our database tables, which is done when metrics are collected. The testDeleteData deletes data based on a timestamp, similar to how we prune our database tables. Lastly, the testInsertAndReadMultipleData is useful to ensure we can retrieve many records from the database tables at once, which is used to plot our utilization metrics on the charts.

### DataSourceConfig

- Class that creates data source bean for connecting to an SQLite database used for testing (TestDatabase.db)
- Annotated with @Bean and @Profile(“test”) - specifies that this bean configuration is only active when the “test” profile is active.

### DatabaseTest

- Interface that the below classes implement
- Defines 5 methods to implement:
  - setUp()
  - tearDown()
  - testInsertAndReadData()
  - testDeleteData()
  - testInsertAndReadMultipleData()

### CpuDatabaseTests

- @BeforeEach setUp() - establishes connection to test database, creates cpu table with corresponding columns. The @BeforeEach annotation means that the setUp function will run before each test case, ensuring the table has been created for queries.

- @AfterEach tearDown() - establishes connection to the test database, and drops the cpu table. The @AfterEach annotation ensures that the tearDown() method is run after each test case.
- testInsertAndReadData() - tests inserting a cpu record into the database  and then retrieves it, verifying it was inserted.
- testDeleteData() - tests deleting the record that was inserted.
- testInsertAndReadMultipleData() - tests inserting multiple cpu records and retrieving them all in a list.

### MemoryDatabaseTests

- @BeforeEach setUp() - establishes connection to test database, creates memory table with corresponding columns
- @AfterEach tearDown() - establishes connection to the test database, and drops the memory table.
- testInsertAndReadData() - tests inserting a memory record into the database and then retrieves it, verifying it was inserted.
- testDeleteData() - tests deleting the record that was inserted.
- testInsertAndReadMultipleData() - tests inserting multiple memory records and retrieving them all in a list.

### DiskDatabaseTests

- @BeforeEach setUp() - establishes connection to test database, creates disk table with corresponding columns
- @AfterEach tearDown() - establishes connection to the test database, and drops the disk table.
- testInsertAndReadData() - tests inserting a disk record into the database and then retrieves it, verifying it was inserted.
- testDeleteData() - tests deleting the record that was inserted.
- testInsertAndReadMultipleData() - tests inserting multiple disk records and retrieving them all in a list.

### ProcessDatabaseTests

- @BeforeEach setUp() - establishes connection to test database, creates process table with corresponding columns
- @AfterEach tearDown() - establishes connection to the test database, and drops the process table.
- testInsertAndReadData() - tests inserting a process record into the database and then retrieves it, verifying it was inserted.
- testDeleteData() - tests deleting the record that was inserted.

- testInsertAndReadMultipleData() - tests inserting multiple process records and retrieving them all in a list.

## Frontend/UI Tests

To ensure that all of *SysMon+*'s UI elements are functioning properly, a manual test will be conducted on all elements by a member of the development team. The tester will examine both the UI's presentation and performance to ensure that the frontend looks and functions as intended. Points of interest within the frontend testing for each page include:

Overall:

- All elements are presented neatly with correct spacing relative to their surrounding elements
- All elements correctly resize based on the size of the window the UI is contained within
- All elements display the correct data in the correct location(s)
- All graphs load with proper data and formatting
- All data points on all loaded graphs are interactable
- For pages that utilize a "Utilization" box, the "Utilization" box's color will change based on the received utilization for each hardware component (red = high, yellow = moderate, green = low)

Glance View / Homepage:

- All "mini" glance views for each monitored hardware component (CPU, Memory/RAM, Disk) are spaced evenly and do not crowd/overlap any other section
- All links to other pages within the UI (section headers, "Top Processes" tables) function properly
- The "Disk" selection menu only shows up for systems with more than one disk recognized by the system
- If a device has more than one disk, changing the disk within the "Disk" section will correctly update the section with the updated graph for said selected disk

Detail View:

- The "Back" button to the Glance view functions properly

Process View:

- The "Back" button to the Glance view functions properly
- All table section headers initiate the table to resort its list of processes based on the desired sort (hardware component + ascending or descending)

## References

For Spring Boot documentation and implementation, refer to the link below.
https://spring.io/projects/spring-boot

For Spring Boot testing documentation and implementation, refer to the link below.
https://docs.spring.io/spring-boot/docs/1.5.7.RELEASE/reference/html/boot-features-testing.html

For JUnit 5 testing documentation and user guide, refer to the link below.
https://junit.org/junit5/

For Apache Maven documentation, refer to the link below.
https://maven.apache.org/