Hanoi University o f Science and Technology

School of Information and Communications Technology

——————— o0o ———————

Project I

Secure coding in Java

A Windows/Linux System Monitor in Java

Supervisor: Nguyen Quoc Khanh

Group of students:

Full Name	**Student ID**
Dang Manh Cuong	20214949
Tran Minh Tuan	20214978

Hanoi, July 2023

Table of contents

Abstract

For a system to be secure, its software must also be secure. Engineering a secure application requires adhering to stringent guidelines to prevent potential vulnerabilities from surfacing. We have developed a Java system monitoring tool that can run on Windows and Linux. We find this an interesting topic of applying secure coding guidelines learned in the course. Security checks via code scanning using the SonarQube scanner did not detect any vulnerabilities in the source code, which we consider a promising result. In the process, we have also explored the flow to develop an application systematically with the help of source control. The project is a stepping stone for us to work with more security-conscious projects in the future.

About the project

The project is about creating a system monitoring application that works on two popular operating systems: Windows and Linux. The application is programmed in Java - one of the most popular programming languages. The program makes use of oshi - a system information library in Java - to provide the backend for displaying system information.

Task assignment

Member name	Contribution	Total percentage
Dang Manh Cuong	• Use case + UI design • Class design (60%) • Windows+Linux shared source code (50%) • Windows-specific code parts • QA testing • Slides (40%) • Report (70%)	55%
Tran Minh Tuan	• Class design (40%) • Windows+Linux shared source code (50%) • Linux-specific code parts • Slides (60%) • Report (30%)	45%

Chapter 1: Use Case description

The goal of the project is to create a system monitoring application in Java for both Windows & Linux. The application allows users to collect and display system information data. For the operating system, it displays basic information such as versioning or system time. The program supports displaying information from the CPU, memory, storage components and network components. For each type of resource, both real-time and static information are available from the interface. In addition, process information can also be viewed. The user can sort the list of processes by columns.
The use case diagram reflects clearly how the application can be utilized by the user. In the use case, the actor is the user. There are two main use cases; for the use case of displaying data, different types of data available are reflected as well. For process details, the use case of sorting is defined.

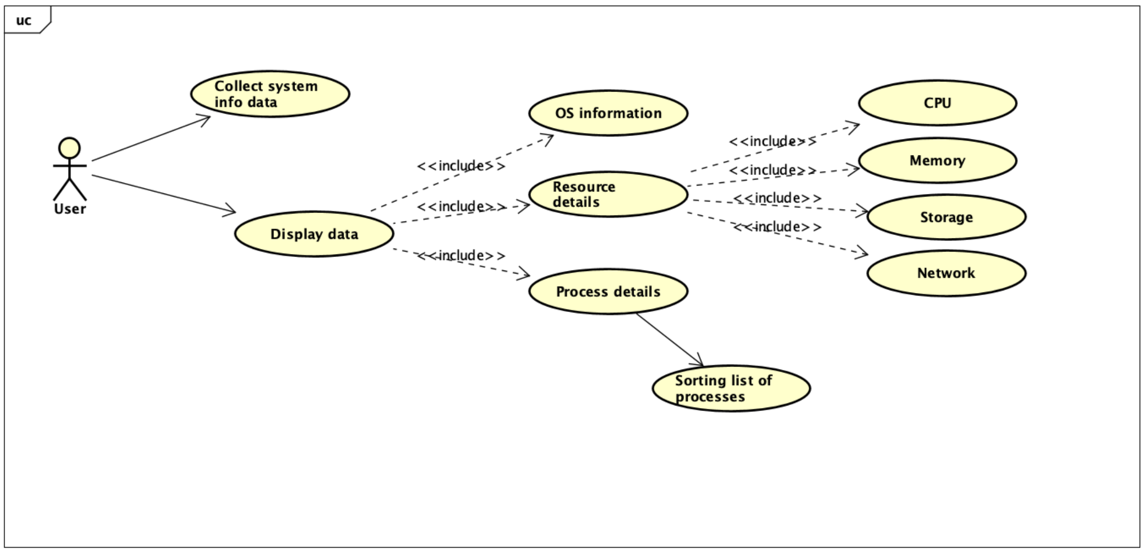
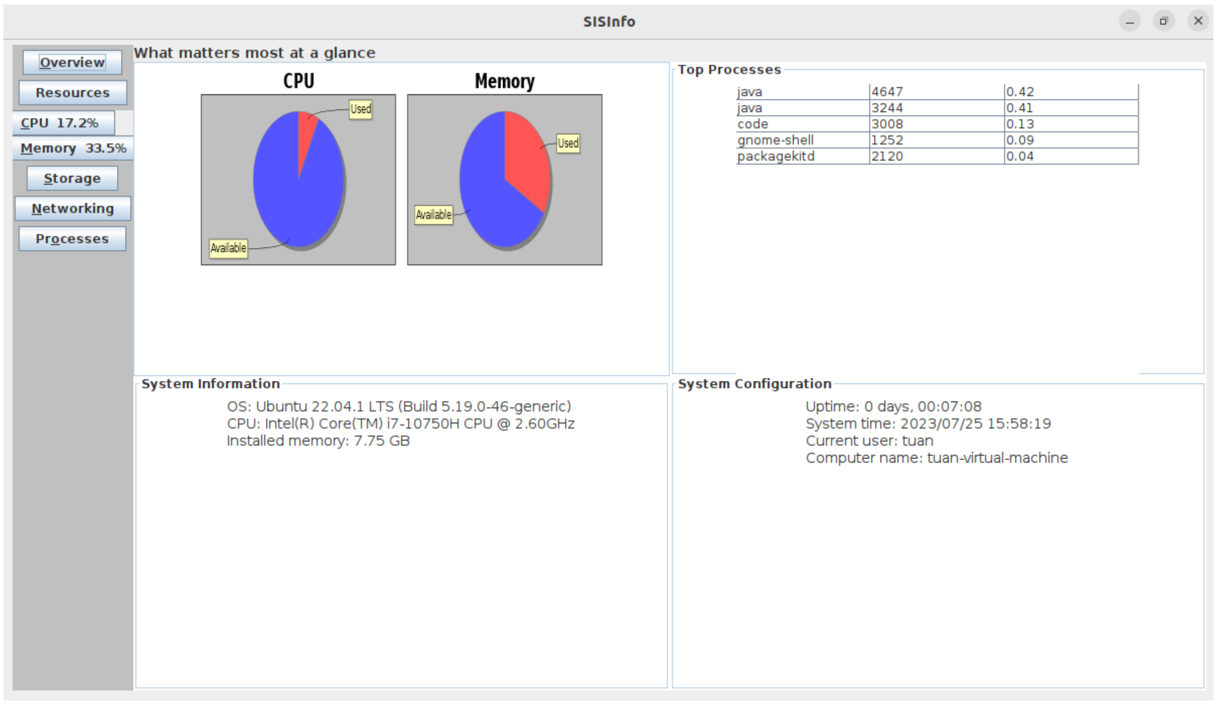


Figure 1: Use Case Diagram

Chapter 2: UI description

The user interface is designed to be simple, yet it is easy to understand the information at first glance.

First, upon opening the application, an overview panel displays basic information about the system. CPU and Memory usage, represented as two pie graphs, can be checked on the top left corner of the window, while the user can also look for which processes utilize the highest CPU load. In addition, static system information and configuration are available at the bottom of the window.



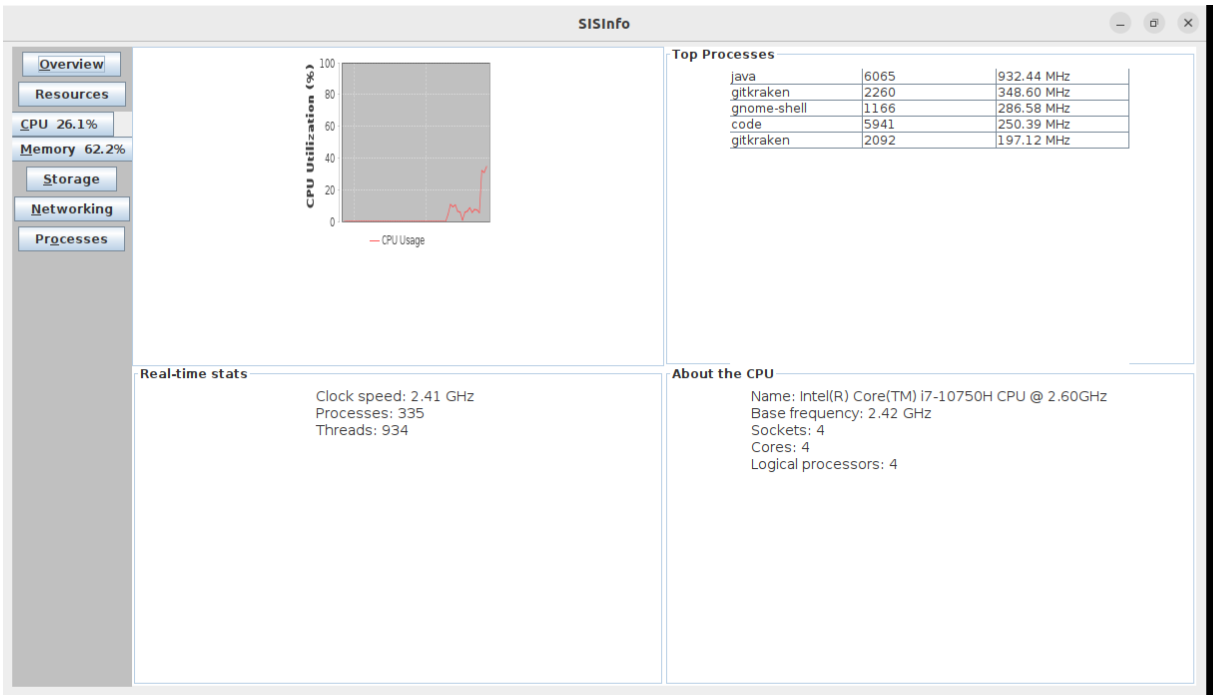
java	4647	0.42
java	3244	0.41
code	3008	0.13
gnome-shell	1252	0.09
packagekitd	2120	0.04

OS: Ubuntu 22.04.1 LTS (Build 5.19.0-46-generic)
CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
Installed memory: 7.75 GB

Uptime: 0 days, 00:07:08
System time: 2023/07/25 15:58:19
Current user: tuan
Computer name: tuan-virtual-machine

Figure 2: Overview panel

The user can look closer into the usage of every resource in the system. For CPU and Memory, a real-time line graph indicates resource usage in the last 5 minutes. Numeric real-time statistics are available for advanced users with even more data. Like the overview panel, there is a list of top processes utilizing the resource most. Finally, static information about the resource is displayed as well.



Top Processes

java	6065	932.44 MHz
gitkraken	2260	348.60 MHz
gnome-shell	1166	286.58 MHz
code	5941	250.39 MHz
gitkraken	2092	197.12 MHz

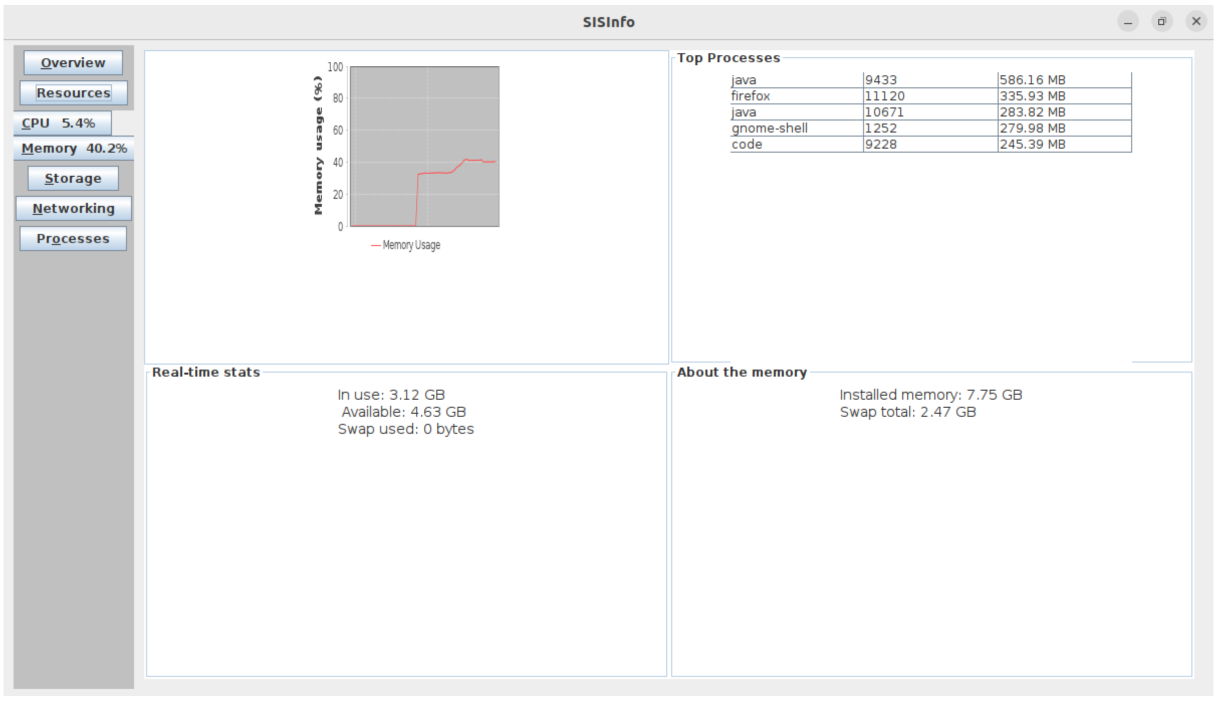
Real-time stats

Clock speed: 2.41 GHz
Processes: 335
Threads: 934

About the CPU

Name: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
Base frequency: 2.42 GHz
Sockets: 4
Cores: 4
Logical processors: 4

Figure 3: CPU panel



Top Processes

java	9433	586.16 MB
firefox	11120	335.93 MB
java	10671	283.82 MB
gnome-shell	1252	279.98 MB
code	9228	245.39 MB

Real-time stats

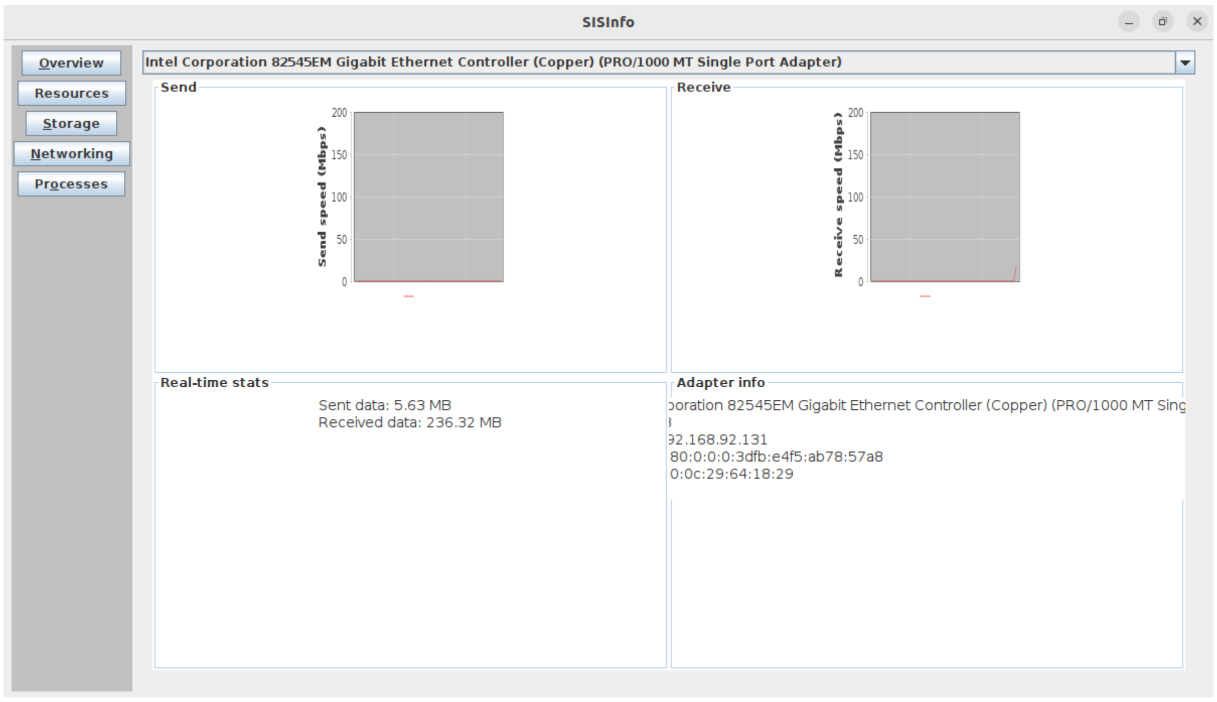
In use: 3.12 GB
Available: 4.63 GB
Swap used: 0 bytes

About the memory

Installed memory: 7.75 GB
Swap total: 2.47 GB

Figure 4: Memory panel

For network information, the program displays information for each adapter. Like other resources, static data and real-time information are available from the interface.

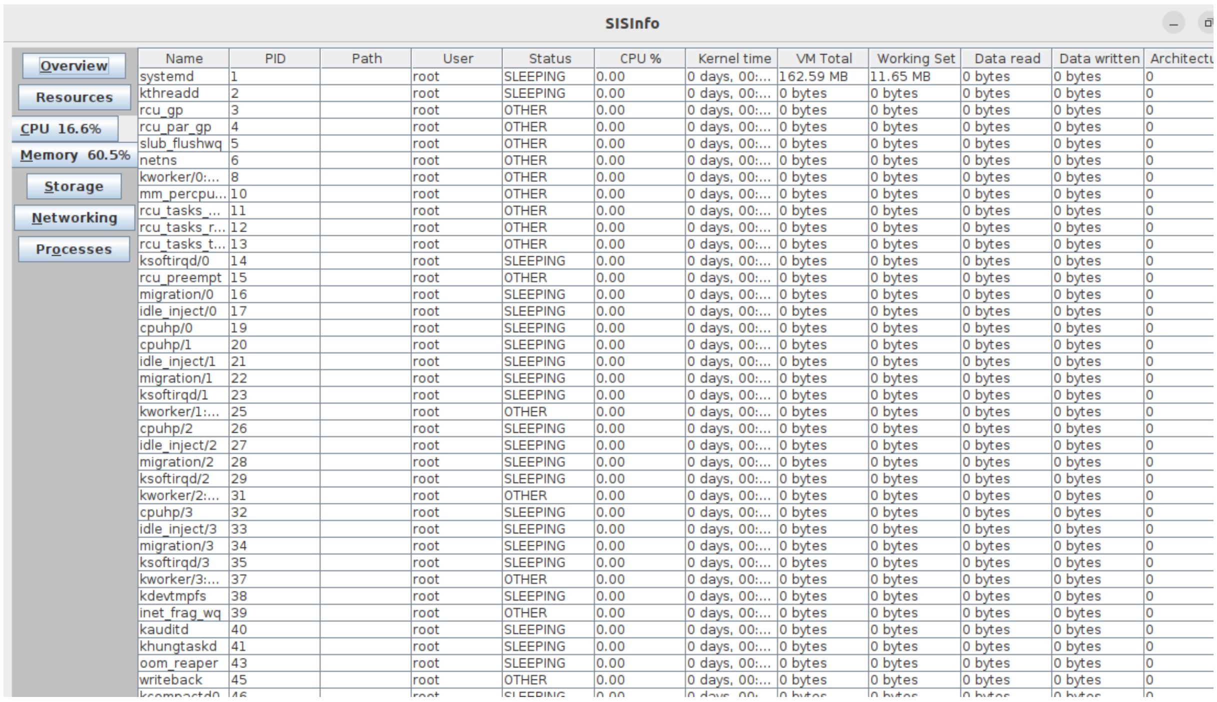


Sent data: 5.63 MB
Received data: 236.32 MB

poration 82545EM Gigabit Ethernet Controller (Copper) (PRO/1000 MT Sing
92.168.92.131
80:0:0:0:3dfb:e4f5:ab78:57a8
0:0c:29:64:18:29

Figure 5: Network panel

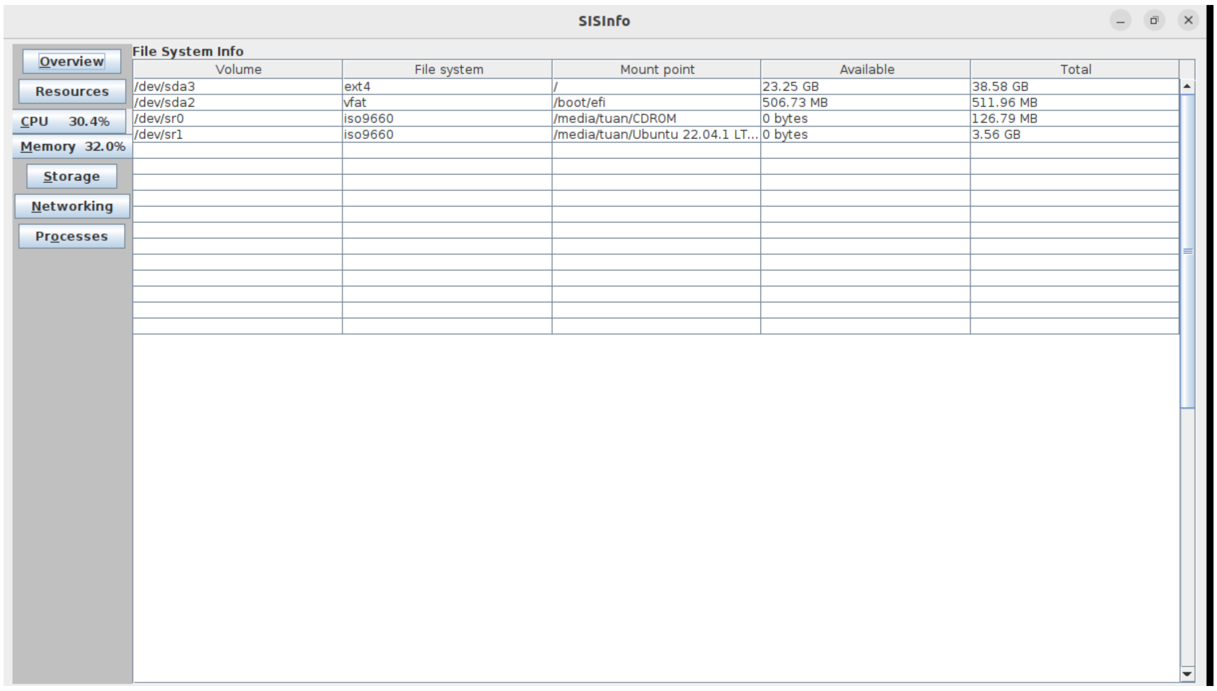
The user can check out the status of every process in the operating system. Depending on the user, each column can be sorted in ascending or descending order.



Name	PID	Path	User	Status	CPU %	Kernel time	VM Total	Working Set	Data read	Data written	Architectu
systemd	1		root	SLEEPING	0.00	0 days, 00:...	162.59 MB	11.65 MB	0 bytes	0 bytes	0
kthreadd	2		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
rcu_gp	3		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
rcu_par_gp	4		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
slub_flushwq	5		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
netns	6		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
kworker/0:...	8		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
mm_percpu...	10		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
rcu_tasks_...	11		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
rcu_tasks_r...	12		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
rcu_tasks_t...	13		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
ksoftirqd/0	14		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
rcu_preempt	15		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
migration/0	16		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
idle_inject/0	17		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
cpuhp/0	19		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
cpuhp/1	20		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
idle_inject/1	21		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
migration/1	22		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
ksoftirqd/1	23		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
kworker/1:...	25		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
cpuhp/2	26		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
idle_inject/2	27		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
migration/2	28		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
ksoftirqd/2	29		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
kworker/2:...	31		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
cpuhp/3	32		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
idle_inject/3	33		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
migration/3	34		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
ksoftirqd/3	35		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
kworker/3:...	37		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
kdevtmpfs	38		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
inet_frag_wq	39		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
kauditd	40		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
khungtaskd	41		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
oom_reaper	43		root	SLEEPING	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0
writeback	45		root	OTHER	0.00	0 days, 00:...	0 bytes	0 bytes	0 bytes	0 bytes	0

Figure 6: Process panel

When it comes to storage information, this is where there is a difference between Windows and Linux implementation. For Linux operating system, the information is limited to volume listing, along with file system, mount point and free space information for each volume.

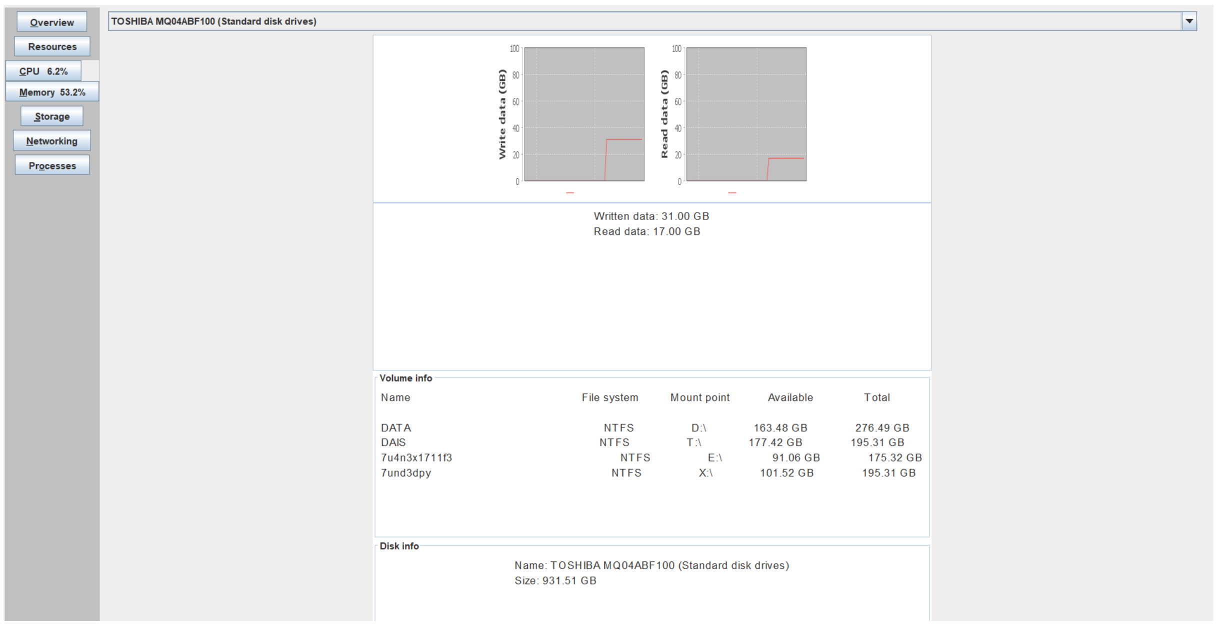


File System Info

Volume	File system	Mount point	Available	Total
/dev/sda3	ext4	/	23.25 GB	38.58 GB
/dev/sda2	vfat	/boot/efi	506.73 MB	511.96 MB
/dev/sr0	iso9660	/media/tuan/CDROM	0 bytes	126.79 MB
/dev/sr1	iso9660	/media/tuan/Ubuntu 22.04.1 LT...	0 bytes	3.56 GB

Figure 7: Storage panel on Linux

On the other hand, storage information on Windows systems is organized by disks. For each disk, volume information in each of them is displayed. In addition, static disk information and real-time data speed are available as well.



Written data: 31.00 GB
Read data: 17.00 GB

Volume info

Name	File system	Mount point	Available	Total
DATA	NTFS	D:\	163.48 GB	276.49 GB
DAIS	NTFS	T:\	177.42 GB	195.31 GB
7u4n3x1711f3	NTFS	E:\	91.06 GB	175.32 GB
7und3dpy	NTFS	X:\	101.52 GB	195.31 GB

Disk info

Name: TOSHIBA MQ04ABF100 (Standard disk drives)
Size: 931.51 GB

Figure 8: Storage panel on Windows

Chapter 3: Quality assurance testing

Software bugs can cause users unpleasant experiences and drive them away from using it. When taking security into account, security flaws may pose threats to the safety of the user. Therefore, software quality assurance (QA) is essential in any development process. A new application must undergo various testing phases to detect defects of any kind present.
In accordance with the project requirement, QA testing for this application centers around 2 types: functional testing and security testing.

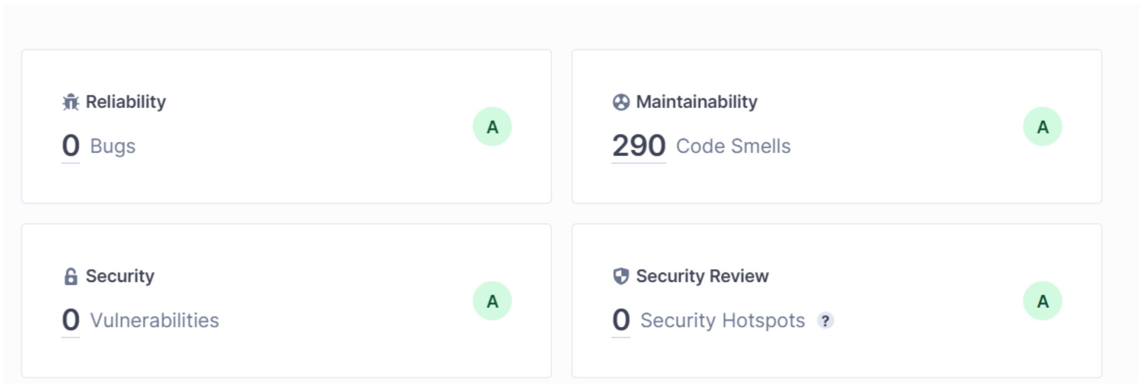
1. Functional Testing

Functional testing deals with the application's compliance with functional requirements and design specifications. This aspect focuses on the practical use of software from the user's point of view: its features, performance, ease of use, and absence of defects.
Please check out the next page for a table which describes the testing results.

Functional aspect	Test results	Rating
Features	The application supplies a good amount of information. Information is displayed correctly, and real-time stats are available for rapid response to system events	Good
Performance	The application may go unresponsive at a high data refresh rate (<= 1 second)	Average
Usability	The UI is resizable, and information is organized in grids - making it convenient for users to check for what they want. There is a vertical sidebar that helps users with navigating the application. However, the graphs are somewhat small. The process table is also cluttering	Acceptable

2. Security Testing

Security testing checks if the program complies with coding principles that maintain application security. The CERT Oracle Secure Coding Standard for Java is a well-established standard for secure coding in Java; therefore, it is used as the baseline against which the application is evaluated.
Due to the length of the standard, it is infeasible to assess the program manually; an automatic solution must be involved in this testing process. We opted for the SonarQube scanner, a popular code-scanning tool that detects bugs and security vulnerabilities. The scan was performed on both platforms' source code, producing two scan results. Below are the results:

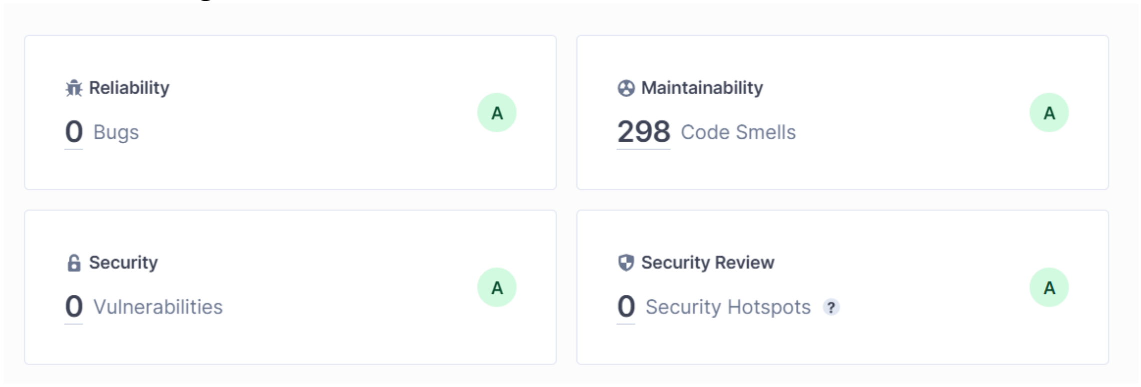

Reliability
0 Bugs
A

Maintainability
290 Code Smells
A

Security
0 Vulnerabilities
A

Security Review
0 Security Hotspots ?
A

Figure 9: SonarQube results for Windows source code


Reliability
0 Bugs
A

Maintainability
298 Code Smells
A

Security
0 Vulnerabilities
A

Security Review
0 Security Hotspots ?
A

Figure 10: SonarQube results for Linux source code

As shown in the results, the scanner did not detect any bugs or security vulnerabilities in any source code scanned. A high number of code smells were detected - around 300 on each platform. Nevertheless, most code smells are cosmetic and should not significantly impact the application's security. Overall, the application exhibited satisfactory behavior from the security point of view.

Conclusion

The project has provided us with a valuable opportunity to create a useful application with secure coding principles in mind. We have experienced the organized process of developing a program, from conceiving the design to source control using GitHub. In order to verify the application security, code scanning via SonarQube was carried out and produced good results. The application provides a good set of information of interest to any users who want to learn more about their system's inner workings. In addition, it may support users with diagnosing their system in the event the system misbehaves or runs slowly.

For future directions, we would like to apply secure coding principles in more challenging projects.

Reference

oshi: Native Operating System and Hardware Information on GitHub:
https://github.com/oshi/oshi
Creating a GUI With Swing, Java Tutorial, and Oracle:
https://docs.oracle.com/javase/tutorial/uiswing/
GitHub:
https://github.com/
SonarQube:
https://www.sonarsource.com/products/sonarqube/
CERT Oracle Coding Standard for Java:
https://wiki.sei.cmu.edu/confluence/display/java

Appendix A: Class diagram

The class diagram is static. It represents the static view of an application. The class diagram is used for visualizing, describing, and documenting different aspects of a system and constructing executable code of the software application. The application is designed with object-oriented principles in mind. The system, every hardware component, and the operating system are represented as objects in the diagram.

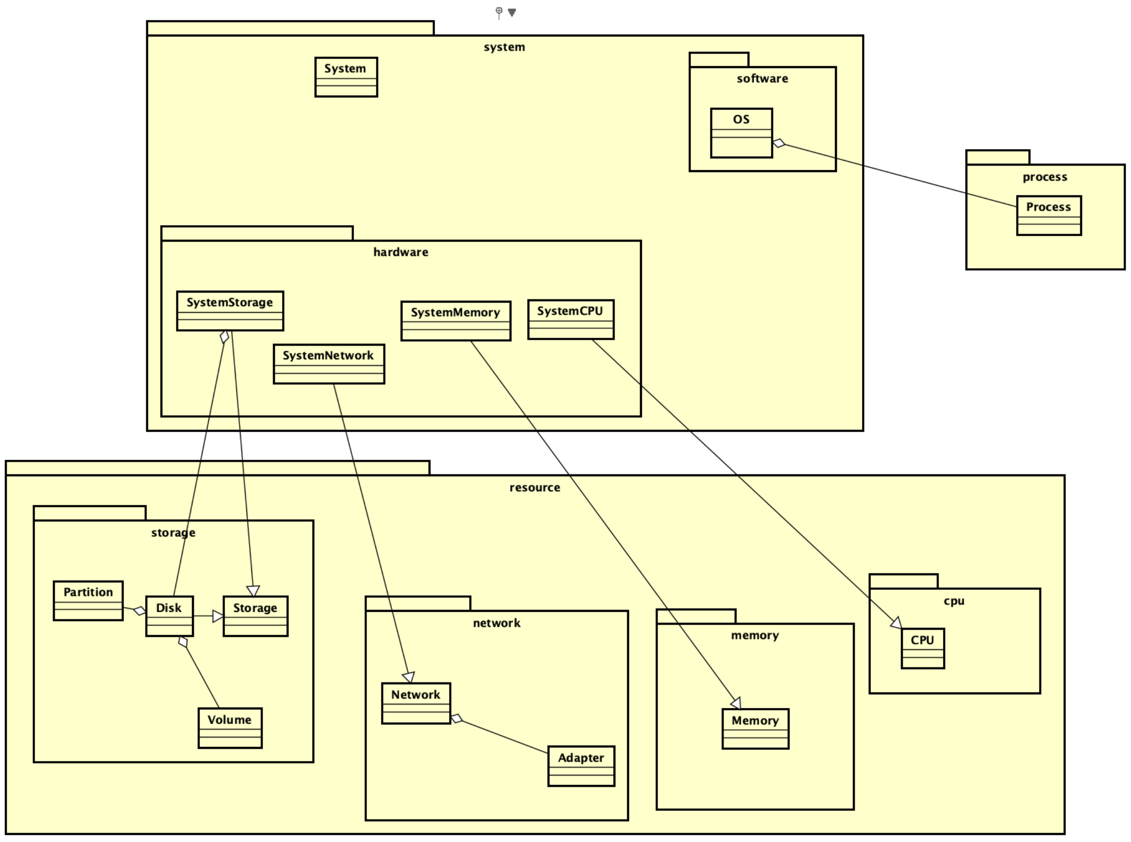


Figure 11: Class Diagram

Appendix B: Information collected

Name	Information	OS collected
CPU	1. Name 2. Architecture 3. Frequency 4. Number of Core 5. Several Logical Processors 6. Number of packages 7. List of cache 8. Utilization Percentage 9. Speed	Windows and Linux
Memory	1. Physical Memory In Use 2. Physical Memory Available 3. Usable Installed Memory 4. Speed 5. Total Swap 6. Memory Usage Percentage	Windows and Linux
Storage	- Disk Information 1. Total Data Written 2. Total Data Read 3. Name 4. Size - Volume Information 1. Name 2. File System 3. Mount Point 4. Available 5. Total	Windows
	1. Volume 2. File System 3. Mount Point 4. Available 5. Total	Linux

Network	1. Send Speed 2. Receive Speed 3. Name 4. identifier 5. IPv4 Address 6. IPv6 address 7. MAC Address 8. NDIS Interface Type	Windows and Linux
Process	1. Name 2. PID 3. Path 4. User 5. Status 6. CPU % 7. Kernel Time 8. VM Total 9. Working Set 10. Data Read 11. Data Written 12. Architecture	Windows and Linux
OS	1. Family 2. Version 3. Build 4. Uptime 5. Several Process 6. Number of Threads 7. User 8. Computer Name	Windows and Linux