A PROJECT REPORT

on

**First player shooting game using python**

Submitted by

**Mr. Pius Ryan**

in partial fulfilment for the award of the degree

of

**BACHELOR OF SCIENCE**

**in**

**COMPUTER SCIENCE**

under the guidance of

**Prof Rakesh Koender**

**Department of Computer Science**



**Kirti College of Arts, Science and Commerce (Autonomous),**

**Dadar (W), Mumbai-28**

**(Sem V)**

**(2024 – 2025)**



**Deccan Education Society’s**

**Kirti College of Arts, Science and Commerce (Autonomous),**

**Dadar (W), Mumbai-28**

**Department of Computer Science**

**CERTIFICATE**

This is to certify that Mr./Ms. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of **T.Y.B.Sc. (Sem VI)** class has satisfactorily completed the Project \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, to be submitted in the partial fulfilment for the award of **Bachelor of Science** in **Computer Science** during the academic year **2024– 2025**.

**Date of Submission:**

**Project Guide Head / Incharge,**

**Department Computer Science**

**College Seal Signature of Examiner**

**DECLARATION**

**I, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, hereby declare that the project entitled “\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_” submitted in the partial fulfilment for the award of Bachelor of Science in Computer Science during the academic year 2024 – 2025 is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.**

**Signature of the Student:**

**Place:**

**Date**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Serial No.** | | **Contents** | | **Page** |
| **1** | | **Introduction** | |  |
|  | | 1.1 | Description of current system |  |
|  | | 1.2 | Limitations of Current System |  |
|  | | 1.3 | Proposed system & its Advantages |  |
|  | | 1.4 | Front End & Back End |  |
| **2** | | **Feasibility study** | |  |
|  | | 2.1 | Operational Feasibility |  |
|  | | 2.2 | Technical Feasibility |  |
|  | | 2.3 | Economical Feasibility |  |
| **3** | | **Requirement analysis** | |  |
|  | | 3.1 | Stakeholder |  |
|  | | 3.2 | Fact Finding Techniques |  |
|  | | 3.3 | Gantt Chart |  |
| **4** | | **System design** | |  |
|  |
|  | | 4.2 | Event Table |  |
|  | | 4.3 | Use Case Diagram |  |
|  | | 4.4 | Activity Diagram |  |
|  | | 4.5 | Class Diagram |  |
|  | | 4.6 | Object Diagram |  |
|  | | 4.7 | Sequence Diagram |  |
|  | | 4.8 | State Chart Diagram |  |
|  | | 4.9 | Package Diagram |  |
|  | | 4.10 | Component And Deployment Diagram |  |
| **5** | | **Menu tree** | |  |
| **6** | | **List of attributes and tables** | |  |
| **7** | | **Naming convention** | |  |
| **8** | | **Program description** | |  |
| **9** | | **Validation** | |  |
| **10** | | **System implementation** | |  |
| **11** | | **Coding** | |  |
| **12** | | **Screen layouts** | |  |
| **13** | | **Bibliography** | |  |

## 1.Introduction

## The purpose of this document is to analyze the existing antivirus system, highlight its limitations, and propose an improved version with enhanced functionalities. As cybersecurity threats continue to evolve, it is essential to develop a robust system that not only addresses current shortcomings but also integrates modern technologies to improve threat detection and system performance. This document will also outline the front-end and back-end technologies that will be utilized in the proposed system to ensure better efficiency, security, and user experience.

## Description of Current System

## The current antivirus system is designed to perform fundamental security functions such as scanning for malware, monitoring system performance, and optimizing RAM usage. It operates primarily based on predefined malware signatures, which are stored in a local database. The scanning process is initiated manually by the user, and the system provides basic UI elements for interaction. While it serves as a functional security tool, it lacks the advanced automation and integration features found in modern antivirus solutions. Additionally, it does not offer real-time protection, making it less effective against emerging threats.

## Limitations of Current System

## Despite its fundamental security capabilities, the current system has several limitations that impact its overall effectiveness and usability. Some of the major drawbacks include:

## Manual Scanning: The system requires user intervention to initiate scans, meaning that threats can remain undetected for extended periods. The absence of real-time protection leaves the system vulnerable to immediate threats.

## Limited Malware Detection: The current malware detection mechanism relies on a database of known malware signatures. This approach is effective against previously identified threats but fails to detect new, evolving threats, zero-day attacks, or polymorphic malware. It lacks heuristic-based analysis and AI-driven detection techniques that could help in identifying suspicious patterns.

## Basic UI & User Experience: While the system includes a graphical user interface, it is minimalistic and does not provide a modern user experience. Features such as real-time notifications, progress tracking, and intuitive navigation are missing, making it less user-friendly.

## Performance Issues: The system tends to consume a significant amount of system resources during scanning operations, leading to slowdowns and reduced performance of other applications. This high resource utilization can be frustrating for users, especially when conducting full system scans.

## Limitations of Current System

- Manual Scanning: Requires user intervention to initiate scans rather than providing real-time protection.

- Limited Malware Detection: The system only identifies known threats based on stored hash values, lacking heuristic or AI-driven detection.

- Basic UI & User Experience: The graphical interface is functional but lacks modern UX enhancements.

- Performance Issues: The system's efficiency is hindered by high resource usage during scanning operations.

- Lack of Cloud Integration: No support for cloud-based updates or remote monitoring.

## Proposed System & Its Advantages

- Real-time Protection: Automatic scanning of files and processes in the background.

- AI & Heuristic-Based Detection: Identifies potential threats even if they are not in the predefined database.

- Optimized Performance: Efficient use of system resources to minimize lag during scans.

- Comprehensive System Monitoring: Includes CPU, RAM, and disk monitoring with detailed logs and reports.

**Technologies Used**

* **Front-End:** The user interface will be developed using PyQt or Tkinter for a lightweight, responsive design. Web-based dashboards may be created using HTML, CSS, JavaScript, and React.js for better accessibility.
* **Back-End**: The core functionalities will be developed using Python, leveraging libraries such as Scikit-learn for AI-driven malware detection, PyInstaller for packaging, and SQLite for local database management.

## 2.Feasibility Study

**Operational Feasibility**The proposed system is designed to enhance security and improve user experience by providing efficient malware detection and removal. Automation of scanning processes and cloud-based updates ensure seamless protection, reducing the need for manual intervention. The system's user-friendly interface and real-time protection contribute to an overall safer and more efficient computing environment.

**Technical Feasibility**The system is technically feasible as it utilizes well-supported and widely available technologies such as Python, Tkinter, and cloud-based update mechanisms. These technologies ensure compatibility with existing infrastructure, requiring minimal modifications for implementation. The use of AI-driven and hash-based scanning methods enhances malware detection capabilities, making the system robust and scalable for future improvements.

**Economic Feasibility**The proposed system is cost-effective as it leverages open-source technologies, eliminating the need for expensive proprietary software. By reducing manual intervention and improving efficiency, the system minimizes operational costs and optimizes resource utilization. Additionally, the prevention of malware infections results in reduced maintenance costs and increased system longevity, making it a financially viable solution.

**3.Requirement Analysis**

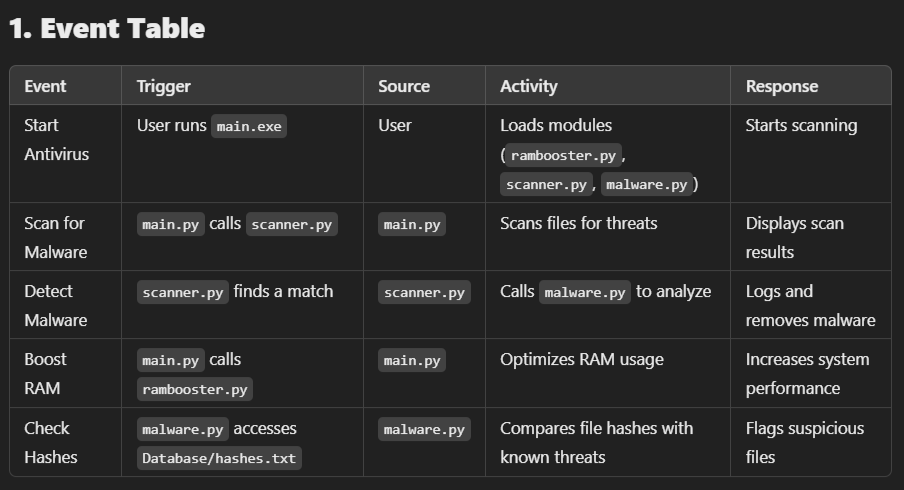
**3.1 Stakeholders**

* **Developer:** As the sole creator, I am responsible for every aspect of the antivirus software, including its design, development, maintenance, and future improvements. This involves writing and optimizing the code, implementing security measures, updating the malware database, and ensuring the software remains effective against emerging threats. Additionally, I handle user feedback and troubleshoot any issues that may arise.
* **End Users:** The individuals or businesses that rely on this antivirus software to protect their systems from malware, viruses, and other security threats. Their feedback and experience are crucial in identifying potential improvements and ensuring the software remains user-friendly and effective

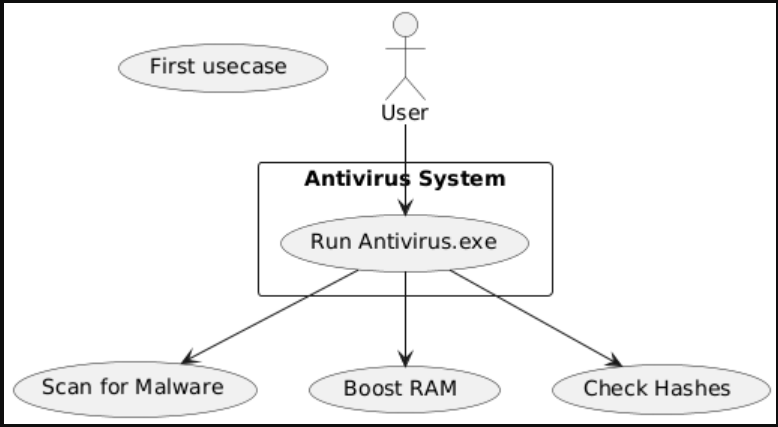
**3.2 Fact-Finding Techniques** To gather requirements and ensure a comprehensive understanding of user needs, the following fact-finding techniques were used:

* **Interviews**: Discussions with potential users and cybersecurity experts.
* **Surveys & Questionnaires**: Collecting feedback on current antivirus software limitations.
* **Observation**: Monitoring user interaction with similar security tools.
* **Document Analysis**: Reviewing existing security reports and known threats.
  1. **Gantt Chart**

| **Task** | **Start Date** | **End Date** | **Duration (Days)** |
| --- | --- | --- | --- |
| Requirement Analysis | 2024-12-01 | 2024-12-07 | 7 |
| GUI Development (Front End) | 2024-12-08 | 2024-12-20 | 13 |
| Core Engine Development (Back End) | 2024-12-21 | 2025-01-10 | 21 |
| Database Integration | 2025-01-11 | 2025-01-17 | 7 |
| Threat Detection & Signature Updates | 2025-01-18 | 2025-01-30 | 13 |
| System Optimization & Performance Testing | 2025-01-31 | 2025-02-10 | 11 |
| Bug Fixing & Refinements | 2025-02-11 | 2025-02-20 | 10 |
| Final Testing & Deployment | 2025-02-21 | 2025-02-28 | 8 |

**Event table:**

**Use case:**



**Activity diagram:**

A diagram of a computer system

AI-generated content may be incorrect.

**Class diagram:**

A diagram of a computer program

AI-generated content may be incorrect.

**Object diagram**

A diagram of a computer program

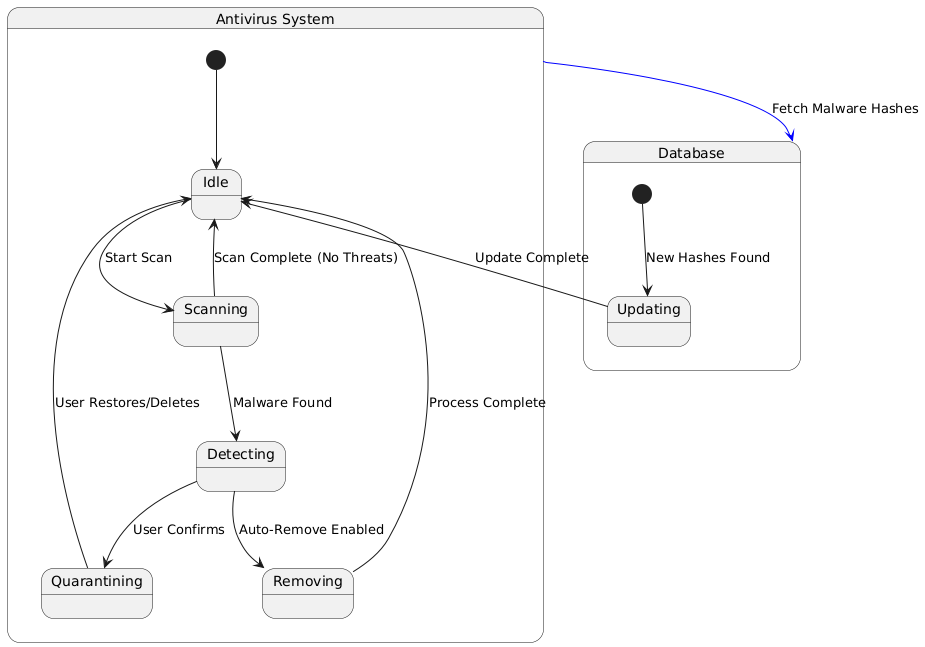
AI-generated content may be incorrect.

**Sequence diagram:**

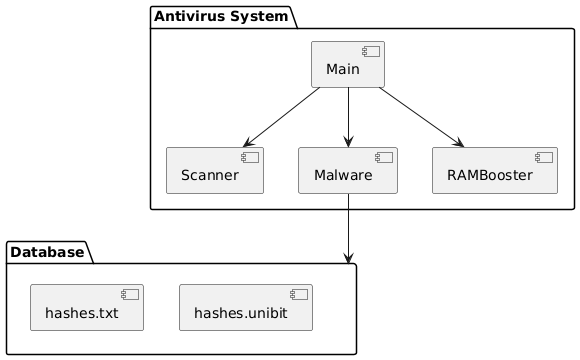
A diagram of a software system

AI-generated content may be incorrect.

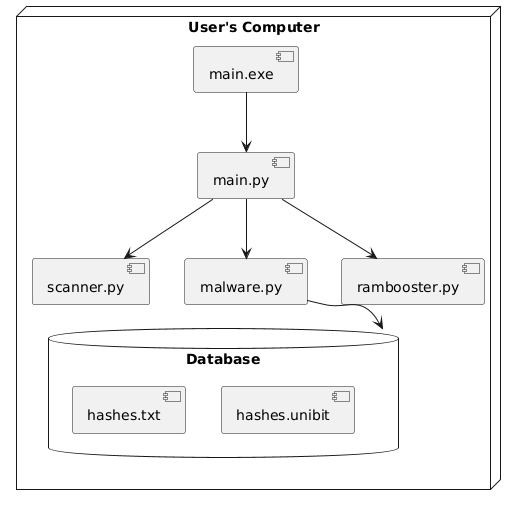
**State diagram:**



**Package diagram:**



**Componenet diagram and Deployement diagram:**



**Menu Tree**

* **Dashboard** 
  + **System Status**
  + **Scan Options**
  + **Performance Monitoring**
* **Security** 
  + **Malware Scanner**
* **Performance** 
  + **RAM Booster**
  + **CPU Optimizatio**

**6.List of Attributes and Tables**

| **Attribute** | **Description** |
| --- | --- |
| **File Name** | **Name of the scanned file** |
| **Hash Value** | **MD5 hash of the file for comparison** |
| **Scan Date** | **Timestamp of the scan** |
| **Malware Name** | **Detected malware type** |
| **Status** | **Safe/Infected** |
| **CPU Usage** | **Percentage of CPU consumption** |
| **RAM Usage** | **Memory consumption of running applications** |

**7. Naming Convention**

* **Python Files: Use lowercase with underscores (e.g., scanner.py, malware.py)**
* **Functions: Use lowercase with underscores (e.g., scan\_files(), calculate\_hash())**
* **Variables: Descriptive names (e.g., file\_path, hash\_database)**
* **Classes: Use PascalCase (e.g., MalwareScanner, RamBooster)**

**8.Program Description**

The system is a Python-based antivirus tool designed to provide comprehensive security and system optimization. It combines advanced malware detection techniques with performance-enhancing features to ensure a secure and efficient computing experience. The key functionalities of the system include:

* Malware Scanning: The antivirus system scans files and directories to detect known threats based on predefined hash values. By maintaining an updated malware signature database, the system effectively identifies and removes malicious software, preventing potential damage to the system.
* Performance Optimization: In addition to security functions, the system includes built-in performance optimization features such as RAM boosting and CPU monitoring. These features help reduce unnecessary system resource consumption, ensuring smooth operation without compromising performance.
* User-Friendly UI: The antivirus system features a modern, intuitive interface built with CustomTkinter. The UI is designed to be visually appealing and easy to navigate, allowing users to access essential functions such as scanning, optimization, and real-time protection effortlessly. The responsive design ensures smooth user interaction with minimal system lag.
* Real-Time Protection: Unlike traditional antivirus solutions that rely solely on manual scanning, this system integrates real-time monitoring and background scanning capabilities. The real-time protection feature actively scans file activities and processes to detect and neutralize threats before they can compromise the system.

Overall, this Python-based antivirus tool offers a well-rounded security solution by combining effective malware detection, system optimization, and an enhanced user experience. With real-time protection and an intuitive interface, it ensures users can maintain their system’s security with minimal effort.

**9.Validation**

To ensure accuracy, security, and stability, the system incorporates several validation mechanisms:

* **File Path Validation**: Before initiating a scan, the system verifies that the selected files and directories exist. This prevents unnecessary errors and ensures that users do not attempt to scan invalid or nonexistent paths. Additionally, the system checks for read and write permissions to avoid access-related issues during scanning and malware removal.
* **Hash Comparison**: The antivirus tool uses MD5 hash verification to detect malware accurately. By comparing file hashes against a secure database, it ensures that only authenticated and verified data is used for malware detection, reducing false positives. Regular database updates enhance detection accuracy and keep the system prepared for emerging threats.
* **Process Management**: The system carefully checks running processes before termination. This prevents users from accidentally closing critical system processes, ensuring system stability and preventing crashes. A whitelist mechanism allows essential system processes to remain protected, while suspicious processes are flagged for further analysis before termination.
* **Quarantine Validation**: When a threat is detected, the system securely moves the suspicious file to a quarantine folder. It ensures that files in quarantine are properly isolated and cannot execute, preventing potential damage. Users can review and restore files if they are deemed safe.

**10.System Implementation**

The antivirus software consists of multiple components that work together to provide an efficient and user-friendly security solution. The implementation is divided into the following key areas:

* **Front End:** The graphical user interface (GUI) is built using **CustomTkinter**, a modern and visually appealing framework based on Tkinter. This interface allows users to interact easily with the antivirus software, providing functionalities such as scanning files, monitoring system activity, and managing threat detections. The GUI is designed to be simple yet effective, ensuring smooth navigation and accessibility for all users.
* **Back End:** The core functionality of the antivirus software is developed using **Python**, which handles various security-related tasks such as **file scanning, process management, and system monitoring**. The scanning engine detects malicious files by comparing them against known malware signatures, while process monitoring helps identify suspicious activities in real time. System performance is optimized to ensure that background scans do not heavily impact the user’s workflow.
* **Database:** The software maintains a structured database to store **malware hashes and threat signatures** for detection purposes. This database is regularly updated to include the latest malware definitions, ensuring the antivirus remains effective against new and evolving threats. The detection mechanism compares scanned files against stored hashes, flagging any matches as potential threats. Future enhancements may include cloud-based updates to improve accuracy and threat coverage.

**11.Coding**

The project is divided into multiple Python files:

* main.py: Launches the application.
* scanner.py: Handles file scanning.
* malware.py: Checks files against the hash database.
* rambooster.py: Optimizes system performance.

**Main.py**

import tkinter as tk

from tkinter import ttk, messagebox

import customtkinter as ctk

import psutil

import threading

import time

from datetime import datetime

from scanner import Scanner

from tkinter import filedialog

from malware import scan\_files

from rambooster import RamBooster

import os  # Added to support taskkill commands

class ModernAntivirusGUI:

    def \_\_init\_\_(self):

        ctk.set\_appearance\_mode("dark")

        ctk.set\_default\_color\_theme("blue")

        self.root = ctk.CTk()

        self.root.title("Ryan's Shield")

        self.root.geometry("1200x800")

    ########################################

    def boost\_performance(self):

        def performance\_boost():

            try:

                # Display all status updates after processing all tasks

                self.status\_label.configure(text="\n".join(status\_updates))

            except Exception as e:

                self.status\_label.configure(text=f"Error: {e}")

        threading.Thread(target=performance\_boost, daemon=True).start()

def main():

    app = ModernAntivirusGUI()

    app.root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Malware.py**

import os

import hashlib

import tkinter as tk

from tkinter import messagebox

# ✅ Load multiple hash databases safely

def load\_hash\_database(hash\_files=None):

    if hash\_files is None:

        hash\_files = [

            r"C:\Users\Admin\Desktop\BitLink\pius alter\DataBase\hashes.txt",

            r"C:\Users\Admin\Desktop\BitLink\pius alter\DataBase\md5hash.UNIBIT"]

    malicious\_hashes = {

-----------------

            if file\_hash and file\_hash in MALICIOUS\_HASHES:

                malware\_name = MALICIOUS\_HASHES[file\_hash]

                infected\_files.append((file\_path, malware\_name))

                result\_listbox.insert(tk.END, f"Infected: {file\_path} ({malware\_name})")

            scanned\_files += 1

            if scanned\_files % update\_threshold == 0 or scanned\_files == total\_files:

                progress\_bar.set(scanned\_files / total\_files)

                result\_listbox.update\_idletasks()  # ✅ Ensures smooth UI updates

    if not infected\_files:

        result\_listbox.insert(tk.END, "✅ No infected files found.")  # ✅ Clearer message

    progress\_bar.set(1)  # ✅ Ensure progress bar reaches 100%

    return infected\_files

**Rambooster.py**

import os

class RamBooster:

    def \_\_init\_\_(self, task\_list=None):

        """

        Initializes the RamBooster class with a list of tasks to terminate.

        :param task\_list: List of process names to kill.

        """

        self.task\_list = task\_list if task\_list else []

    def add\_task(self, task\_name):

        """

        Adds a task to the list of tasks to terminate.

        :param task\_name: Name of the process (e.g., 'notepad.exe').

        """

        self.task\_list.append(task\_name)

    def kill\_tasks(self):

        for task in self.task\_list:

            print(f"Attempting to kill: {task}")

            result = os.system(f"taskkill /f /im {task}")

            if result == 0:

                print(f"Successfully terminated {task}")

            else:

                print(f"Failed to terminate {task}. It might not be running.")

**Scanner.py**

import os

import threading

from tkinter import messagebox

class Scanner:

    def \_\_init\_\_(self, status\_label, progress\_bar, root):

        self.status\_label = status\_label

        self.progress\_bar = progress\_bar

        self.root = root

    def quick\_scan(self):

        threading.Thread(target=self.\_quick\_scan).start()

    def \_quick\_scan(self):

        try:

            self.status\_label.configure(text="Starting Quick Scan...")

            self.progress\_bar.set(0)

            # Define the directories to clean

            temp\_dir = os.environ.get('TEMP') or os.path.join(os.environ.get('USERPROFILE'), 'AppData', 'Local', 'Temp')

            prefetch\_dir = os.path.join(os.environ.get('SystemRoot', 'C:\\Windows'), 'Prefetch'

            directories = [temp\_dir, prefetch\_dir]

            total\_files = 0

            # Count total files

            for directory in directories:

                if os.path.exists(directory):

                    total\_files += sum([len(files) for \_, \_, files in os.walk(directory)])

            if total\_files == 0:

                self.status\_label.configure(text="No temporary or prefetch files found")

                return

            files\_processed = 0

            space\_freed = 0

            # Start cleaning process

            for directory in directories:

                if os.path.exists(directory):

                    for root, dirs, files

                                    text=f"Cleaning files: {int(progress \* 100)}% ({files\_processed}/{total\_files})"

                                )

                                self.root.update\_idletasks()

                            except (PermissionError, FileNotFoundError):

                                continue

                        for name in dirs:

                            try:

                                os.rmdir(os.path.join(root, name))

                            except (PermissionError, OSError):

                                continue

            space\_freed\_mb = space\_freed / (1024 \* 1024)

            self.status\_label.configure(

                text=f"Scan complete. Cleaned {files\_processed} files ({space\_freed\_mb:.2f} MB freed)")

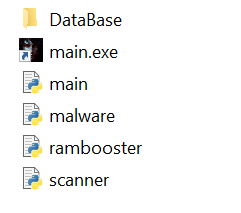
        except Exception as e:

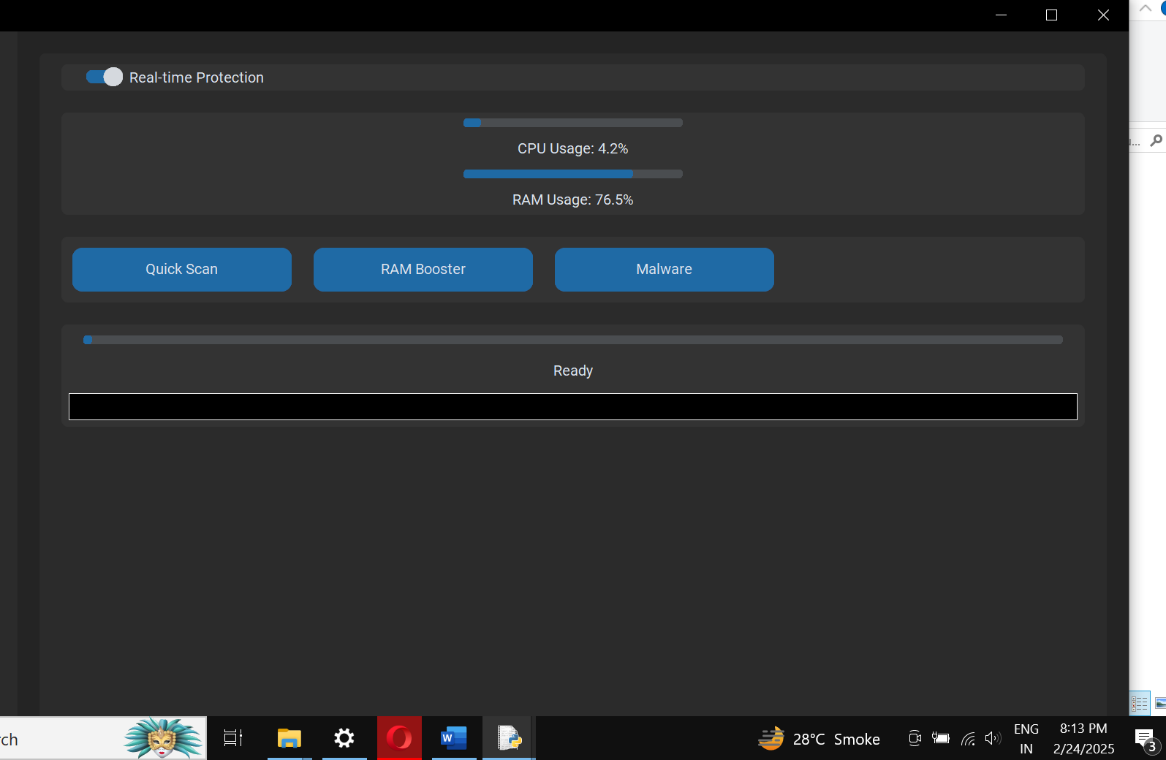
            self.status\_label.configure(text=f"Error during scan: {str(e)}")

            self.progress\_bar.set(0)

**12.Screen Layouts**

* **Main Dashboard: Displays system health and quick actions.**
* **Scan Window: Provides options for quick/full scans and shows results.**
* **Performance Monitor: Displays CPU and RAM usage.**

****

****

**13.Bibliography**

The following references were instrumental in the development of the antivirus system, providing essential guidance on various aspects such as programming, security, UI design, and performance optimization:

* Python Official Documentation: Served as a primary resource for implementing core functionalities, including file scanning, hashing, and process management.
* CustomTkinter Library: Assisted in designing a modern and responsive user interface, improving the overall user experience.
* Pygame & Psutil: Pygame was used for UI responsiveness, while Psutil enabled real-time system monitoring, CPU usage tracking, and performance optimization.
* VirusTotal API: Provided cloud-based malware scanning capabilities, allowing the system to cross-check files with a vast online database of known threats.
* YARA Rules: Helped in implementing heuristic-based malware detection by defining specific patterns to identify suspicious files and behaviors.
* Scikit-learn: Supported AI-driven threat analysis, enabling the integration of machine learning techniques for improved malware detection.

By leveraging these resources, the antivirus system was developed to be secure, efficient, and capable of addressing modern cybersecurity threats effectively.