Appendix:

T-Kinter Application:

```
import tkinter as tk from tkinter
import messagebox from tkinter
import filedialog import
subprocess import os import
logging import hashlib
from fpdf import FPDF
import time

# Global variable to track the last used tool
current_tool = "Default"

# Set up logging
log_path = os.path.join(os.path.expanduser("~"), "user_action_log.txt")
logging.basicConfig(filename=log_path,      level=logging.INFO,   format='%(asctime)s
%(message)s')

# Global variables to track mouse movement and log interval
last_log_time = time.time()
LOG_INTERVAL = 2 # Time interval in seconds (e.g., log every 2 seconds)


def log_user_action(action, details):
    """Log user actions to a file with throttling"""
    log_message = f"Action: {action}, Details: {details}"
    logging.info(log_message)


def log_mouse_move(event):
```

```
    """Throttle mouse move logging"""
    global last_log_time

    current_time = time.time()

    # Log only if the interval exceeds the defined LOG_INTERVAL (e.g., 2 seconds)
    if current_time - last_log_time >= LOG_INTERVAL:
        log_user_action('Mouse Moved', f"Mouse moved to position ({event.x}, {event.y})")
        last_log_time = current_time # Update the last log time


def log_mouse_click(event): """Log mouse click events""" log_user_action('Mouse
    Clicked', f"Mouse clicked at position ({event.x}, {event.y})")


# Helper function to run scripts def
run_script(script_name, tool_name):
    global current_tool
    current_tool = tool_name # Update the current tool name
    try:
                    log_user_action('Run Script', f"User started the script: {script_name} using tool:
{tool_name}") process =
        subprocess.Popen(
        ["python", script_name],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True
        )
        stdout, stderr = process.communicate()

        if process.returncode == 0:
```

```
            display_results(stdout)
        else:
            display_results(f"Error:\n{stderr}")
    except Exception as e:
        display_results(f"An unexpected error occurred: {str(e)}")
        log_user_action('Run Script Error', f"Error while running script {script_name}: {str(e)}")


def generate_sha256_hash(text):
    """Generate SHA256 hash of the given text"""
    return hashlib.sha256(text.encode()).hexdigest()


def save_results_and_log():
    """Save results to a PDF file with SHA256 hash and log data."""
    results_content = results_text.get("1.0", tk.END)
    sha256_hash = generate_sha256_hash(results_content)
    log_data = ""

    # Read the log file
    try:
        with open(log_path, "r") as log_file:
            log_data = log_file.read() except Exception as e:
    messagebox.showerror("Error", f"Could not read log file: {str(e)}")

    # Ask where to save the PDF
    save_path = filedialog.asksaveasfilename(defaultextension=".pdf", filetypes=[("PDF files",
"*.pdf")]) if save_path:
        try:
            # Create a PDF with the results and hash using fpdf
```

```
        pdf = FPDF()
        pdf.add_page()

        pdf.set_font("Arial", style='B', size=12) # Bold font for headings
        pdf.cell(200, 10, txt=f"Results for: {current_tool}", ln=True, align="C")
        pdf.cell(200, 10, txt=f"SHA256 Hash: {sha256_hash}", ln=True,
        align="L") pdf.set_font("Arial", size=12) # Set regular font for body
        pdf.cell(200, 10, txt="Results:", ln=True, align="L") pdf.multi_cell(200, 10,
        txt=results_content.strip())

        pdf.cell(200, 10, txt="Log Data:", ln=True, align="L")
        pdf.multi_cell(200, 10, txt=log_data.strip())

        pdf.output(save_path)

        messagebox.showinfo("Save Results", f"Results and Log saved to '{save_path}'")
    except Exception as e: messagebox.showerror("Error", f"Could not save the PDF:
    {str(e)}")


def display_results(content):
    results_text.delete("1.0", tk.END)
    results_text.insert(tk.END, content)


# Main GUI setup
root = tk.Tk()
root.title("Digital Forensic Tool")
root.geometry("800x500") # Adjusted window size to make it smaller
root.configure(bg="#2d2d34")
```

```
# Bind mouse events to log mouse movement and clicks
root.bind('<Motion>', log_mouse_move) # Logs mouse move events
root.bind('<Button-1>', log_mouse_click) # Logs left mouse button click events

# Header Section header_frame =
tk.Frame(root, bg="#1c1c22")
header_frame.pack(fill="x") title_label =
tk.Label( header_frame, text="Digital
Forensic Analysis Tool", font=("Helvetica",
20, "bold"), bg="#1c1c22", fg="#e6e6e6"
)
title_label.pack(pady=10)

# Description Section
desc_label = tk.Label(
  root,
  text=(
    "Delve into the Windows Registry for critical insights into user activity, "
    "software behavior, and forensic evidence. This tool empowers digital forensic analysts "
    "to extract and analyze registry data effectively."
  ),
  font=("Arial", 10),
  bg="#2d2d34",
  fg="#a5a5a5",
  justify="center",
  wraplength=800
)
desc_label.pack(pady=10)

# Content Frame: Tools on Left, Results on Right
```

```
content_frame = tk.Frame(root, bg="#2d2d34")
content_frame.pack(fill="both", expand=True, padx=15, pady=10)

# Tools Section (Left)
tools_frame = tk.Frame(content_frame, bg="#383842", width=250, relief="groove") # Adjusted width for left side tools_frame.pack(side="left", fill="y", padx=10, pady=10)
tools_frame.pack_propagate(False)

tools_label = tk.Label(
    tools_frame, text="Available
    Tools", font=("Helvetica", 16,
    "bold"), bg="#383842",
    fg="#f1f1f1"
)
tools_label.pack(anchor="w", padx=10, pady=10)

# Button Style
button_style = {
    'font': ("Arial", 12),
    'bg': "#4e5d94",
    'fg': "#ffffff",
    'activebackground': "#36457e",
    'activeforeground': "#ffffff",
    'relief': "flat",
    'padx': 10,
    'pady': 10
}

services = [
    ("Registry Viewer", "Take 2 - Noura.py"),
```

```
    ("Last Modified Tracker", "Take 5 - Noura.py"),
    ("Registry Exporter", "Take 9 - Noura.py"),
    ("Installed Software Viewer", "Take 10 - Noura.py"),
    ("Hardware Info Display", "Take 12 - Noura.py"),
    ("User Account Viewer", "Take 13- Noura.py"),
    ("Installed Apps Tracker", "Take 15 - Noura.py"),
]

for service_name, script_file in services:
    service_button = tk.Button( tools_frame, text=service_name,
        font=("Arial", 12), command=lambda s=script_file,
        t=service_name: run_script(s, t), bg="#4e5d94",
        fg="#ffffff",
        activebackground="#36457e",
        activeforeground="#ffffff",
        relief="flat",
        padx=10,
        pady=5
    )
    service_button.pack(pady=5, padx=10, fill="x")

# Results Section (Right) results_frame = tk.Frame(content_frame,
bg="#1c1c22", relief="groove") results_frame.pack(side="right", fill="both",
expand=True, padx=10, pady=10)

results_label = tk.Label(
    results_frame,
    text="Results",
    font=("Helvetica", 14),
    bg="#1c1c22", fg="#e6e6e6"
)
```

```
results_label.pack(anchor="w", padx=10, pady=10)

results_text = tk.Text(
    results_frame, height=15,
    wrap="word",
    font=("Courier", 10),
    relief="groove",
    borderwidth=2,
    bg="#1c1c22",
    fg="#f1f1f1",
    insertbackground="#ffffff"
)
results_text.pack(pady=10, padx=10, fill="both", expand=True)


# Single Button for Save and Download Log
save_button = tk.Button(
    results_frame,
    text="Save Results & Download Log",
    font=("Arial", 10),
    command=save_results_and_log,
    bg="#4e5d94",
    fg="#ffffff",
    activebackground="#36457e")

save_button.pack(side="bottom", pady=10)  # This places the button at the bottom of the results_frame
# Footer Section
footer_label = tk.Label(
    root,
    text="Developed by Digital Forensics Experts NOSH | All Rights Reserved 2024",
```

```
    font=("Arial", 8, "italic"),
    bg="#1c1c22",
    fg="#a5a5a5"
)
footer_label.pack(side="bottom", fill="x", pady=5)


def log_mouse_move(event):
    global last_log_time
    current_time = time.time()

    # Log only if 2 seconds have passed since the last log
    if current_time - last_log_time >= LOG_INTERVAL:
        log_user_action('Mouse Moved', f"Mouse moved to position ({event.x}, {event.y})")
        last_log_time = current_time # Update the last log time


# Start the GUI loop
root.mainloop()
```

Registry Viewer:

```
import winreg # Import the Windows Registry module


#This is the code to open and display each value the name and data, infromation in the registry
def display_registry_values(): try:
        # Specify the registry path
        registry_path = r"Software\Microsoft\Windows\CurrentVersion\Run"
        # Open the registry key key = winreg.OpenKey(winreg.HKEY_CURRENT_USER,
registry_path, 0, winreg.KEY_READ)

        print(f"Values under {registry_path}:\n")
```

```
        # Enumerate and display all values in the key
        i = 0 while
        True: try:
            # Enumerate values
            value_name, value_data, value_type = winreg.EnumValue(key, i)
          print(f"{value_name} : {value_data}") # Display value name and data i
          += 1 except OSError:
            break # No more values to read

        # Close the registry key winreg.CloseKey(key)

    except FileNotFoundError:
        print("Registry path not found.")
    except PermissionError: print("Permission denied: you may need
        administrator privileges.")
    except Exception as e: print(f"An
        error occurred: {e}")


# Run the function to display registry values display_registry_values()
```

Last Modified Tracker: **import winreg**

```
import datetime


def display_key_last_modified_time(registry_path):
    try:
        # Open the key and retrieve information
        key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, registry_path)
        info = winreg.QueryInfoKey(key)

        # info[2] contains the last modified time in 100-nanosecond intervals since Jan 1, 1601
                                last_modified_time = datetime.datetime(1601, 1, 1) +
datetime.timedelta(microseconds=info[2] // 10) print(f"The registry key [{registry_path}]

        was last modified on: {last_modified_time}") winreg.CloseKey(key)


    except Exception as e: print(f"An
        error occurred: {e}")


# Usage example
display_key_last_modified_time(r"Software\Microsoft\Windows\CurrentVersion")
```

Registry Exporter:

```
#To open and read from a file with the content of the registery

import winreg


def export_registry_to_report(registry_path, file_path):
  with open(file_path, 'w') as file:
    def write_key_info(base_key, path, indent=""):
      try:
        key = winreg.OpenKey(base_key, path)
        file.write(f"\n{indent}[{path}]\n")


        # Write values in the current key
        i = 0
        while True:
          try:
            value_name, value_data, value_type = winreg.EnumValue(key, i)
            file.write(f"{indent} {value_name} : {value_data}\n")
            i += 1
          except OSError:
            break # No more values
        # Recurse into subkeys
        i = 0
        while True:
          try:
            subkey_name = winreg.EnumKey(key, i) write_key_info(base_key,
            f"{path}\\{subkey_name}", indent + " ")
            i += 1
          except OSError:
            break # No more subkeys
```

```
            winreg.CloseKey(key)
        except Exception as e:
        pass

    write_key_info(winreg.HKEY_CURRENT_USER, registry_path)
    print(f"Report exported to {file_path}")

  # Open and display the content of the report
  with open(file_path, 'r') as file:
    content = file.read()
    print("\nRegistry Report
    Content:\n") print(content)


# Usage example
export_registry_to_report(r"Software\Microsoft\Windows\CurrentVersion",
"registry_report.txt")
```

Installed Software Viewer:

```
#To display the files
import winreg


def list_installed_software():
  software_list = []
  try:
                              key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE,
r"SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall")

    i = 0
    while True:
      try:
```

```
            subkey_name = winreg.EnumKey(key, i)
            subkey = winreg.OpenKey(key, subkey_name)
            try:
                display_name, _ = winreg.QueryValueEx(subkey,
            "DisplayName") software_list.append(display_name) except
            FileNotFoundError:
                pass # Skip if "DisplayName" is not present
            i += 1
        except OSError:
            break # No more subkeys

    winreg.CloseKey(key)

    print("Installed Software:")
    for software in software_list:
        print(software)
except Exception as e: print(f"An
    error occurred: {e}")

# Usage example
list_installed_software()
```

Hardware Info Display:

```
import winreg
```

```
def display_hardware_info():
  try:
    # Display CPU Information
                    cpu_key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE,
r"HARDWARE\DESCRIPTION\System\CentralProcessor\0") processor_name, _
  = winreg.QueryValueEx(cpu_key, "ProcessorNameString") print("CPU
  Information:") print(f" Processor Name: {processor_name}")
  winreg.CloseKey(cpu_key) except FileNotFoundError:
    print("CPU information registry key not found.") except
  Exception as e: print(f"An error occurred while fetching
  CPU info: {e}")

  try:
    # Display GPU Information (Example for NVIDIA GPU)
                    gpu_key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE,
r"SOFTWARE\Microsoft\DirectX") gpu_name, _ =
    winreg.QueryValueEx(gpu_key, "InstalledDisplayDrivers") print("\nGPU
    Information:") print(f" GPU Name: {gpu_name}")
    winreg.CloseKey(gpu_key) except FileNotFoundError:
    print("GPU information registry key not found.") except
  Exception as e: print(f"An error occurred while fetching
  GPU info: {e}")

# Usage example
display_hardware_info()
```

User Account Viewer:

```
import winreg

def display_user_accounts():
```

```
  try:
                                key     =     winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE,
r"SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList")


      print("User Accounts:")
      i = 0
      while True:
         try:
            sid = winreg.EnumKey(key, i) user_key = winreg.OpenKey(key, sid)
            profile_path, _ = winreg.QueryValueEx(user_key, "ProfileImagePath")
            print(f"SID: {sid}") print(f" Profile Path: {profile_path}")


            i += 1
         except OSError:
            break # No more accounts
      winreg.CloseKey(key)


  except Exception as e: print(f"An
      error occurred: {e}")


# Usage example
display_user_accounts()
```

Installed Apps Tracker:

```
import winreg
def display_installed_apps():
  try:
```

```
    registry_path = r"SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall"
    key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, registry_path)

    print("Installed Applications:")
    i = 0 while
    True: try:
            app_key_name    =    winreg.EnumKey(key,    i)
            app_key = winreg.OpenKey(key, app_key_name)

            try:
                app_name, _ = winreg.QueryValueEx(app_key, "DisplayName")
                install_date, _ = winreg.QueryValueEx(app_key, "InstallDate")
                print(f"App Name: {app_name}") print(f" Install Date:
                {install_date}")
            except FileNotFoundError: pass # Skip
                if information is missing

            i += 1 except
        OSError:
            break # No more applications

    winreg.CloseKey(key)


  except Exception as e: print(f"An
    error occurred: {e}")

# Usage
display_installed_apps()
```

Write me an e-mail that when I was trying to submit the form I had an issue with the education as it doesn't have the education of my university or the name of my university as an option to submit.