A close-up of a sign

Description automatically generated

**Project Title:**

**Network Security Scanner**

**Project Members:**

**Name:**

**Rimsha Shabir**

**Registration #:**

**BSIT-FA20-031**

**Email:**

[rimshashabbir64@gmail.com](mailto:rimshashabbir64@gmail.com)

**Supervised By: Signature**

**Dr. Shabana Ramzan ----------------------**

**University:**

**The Government Sadiq College Women University Bahawalpur, Pakistan**

**Date:**

**08/06/2024**

# **Chapter 1. Final Project Proposal**

## Introduction:

## **Project Title:**

Network Security Scanner

## Project Overview Statement:

The Network Security Scanner is a comprehensive tool developed to enhance network security through real-time monitoring, vulnerability detection, and protocol analysis. This tool aims to provide network administrators with the means to identify potential threats, analyze network traffic, and generate detailed reports for informed decision-making. Leveraging Python libraries such as nmap, scapy, and tkinter, the scanner combines robust backend processing with an intuitive graphical user interface to offer both functionality and usability.

## Project Goals & Objectives:

Goals:

* **Real-time Network Monitoring:** Implement a system capable of capturing and analyzing network traffic in real-time to ensure prompt detection of suspicious activities.
* **Vulnerability Identification:** Develop mechanisms to identify and log potential vulnerabilities in the network, enabling proactive security measures.

Objectives:

1. **Network Traffic Capture:**
   * Utilize nmap and scapy to capture live network traffic, covering a wide range of protocols and activities.
   * Ensure compatibility with various network configurations and environments for comprehensive monitoring.
2. **Vulnerability Detection:**
   * Implement vulnerability checks for common ports and services, such as FTP, SSH, HTTP, and HTTPS.
   * Maintain a database of known vulnerabilities and exploits to compare against scanned hosts and services.

## 1.5. High-Level System Components:

1. Network Scanner:

* **Functionality:** Conducts comprehensive network scans to identify active hosts, open ports, and running services.
* **Implementation Details:**
  + Utilizes the nmap library to perform port scanning and host discovery.
  + Extracts detailed information about each scanned host, including hostname, state, and open ports.

2. Vulnerability Checker:

* **Functionality:** Analyzes scanned hosts and services for potential vulnerabilities and security weaknesses.
* **Implementation Details:**
  + Utilizes nmap scan results to check for known vulnerabilities associated with open ports and services.
  + Logs potential vulnerabilities and suggests remediation steps for network administrators.

3. Protocol Analyzer:

* **Functionality:** Analyzes network protocols to identify anomalies and potential security threats.
* **Implementation Details:**
  + Utilizes scapy to capture and dissect network packets, extracting protocol-level information.
  + Performs protocol analysis to detect suspicious or unauthorized activities within the network.

4. Report Generator:

* **Functionality:** Generates detailed reports summarizing the results of network scans, vulnerability checks, and protocol analysis.
* **Implementation Details:**
  + Aggregates scan results, vulnerability findings, and protocol analysis into comprehensive reports.
  + Utilizes pandas and openpyxl to create Excel reports with structured data and visualizations.

## List of Optional Functional Units:

1. **Email Alerts for Vulnerabilities:**
   * Automatically sends email notifications to network administrators upon detection of critical vulnerabilities.
2. **Advanced Filtering Options in the GUI:**
   * Provides users with the ability to filter and customize the displayed network data based on various criteria.
3. **Historical Data Analysis:**
   * Implements functionalities to analyze historical network traffic data, identify patterns, and generate trend reports.

## 1.7. Exclusions:

1. **Deep Packet Inspection:**
   * Excluded due to the complexity and resource-intensive nature of deep packet inspection, which falls outside the scope of this project.
2. **Integration with External Databases:**
   * Excluded to maintain simplicity and reduce dependencies, focusing on a standalone tool with local logging capabilities.

## 1.8. Application Architecture:

Description:

The application architecture comprises interconnected components, including the Network Scanner, Vulnerability Checker, Protocol Analyzer, and Report Generator. These components work together to provide a comprehensive network security solution.

Placeholder for actual diagram

## 1.9. Gantt Chart:

Description:

The Gantt chart visualizes the project timeline, outlining key tasks and milestones from initial planning to deployment and user training.

Placeholder for actual Gantt chart

## 1.10. Hardware and Software Specifications:

Hardware:

* **Standard PC:** Equipped with a network interface card capable of handling high volumes of network traffic, sufficient RAM, and processing power.

Software:

* **Python 3.x:** Primary programming language for development.
* **nmap and scapy:** Libraries for network scanning and packet manipulation.
* **tkinter:** Graphical user interface development.
* **pandas and openpyxl:** Libraries for data manipulation and report generation.

## 1.11. Tools and Technologies Used:

1. **nmap and scapy:**
   * **Purpose:** Network scanning and packet manipulation.
   * **Details:** nmap for comprehensive network scans and scapy for packet capturing and analysis.
   * **Reason for Use:** Robust functionalities and compatibility with Python for seamless integration.
2. **tkinter:**
   * **Purpose:** Graphical user interface development.
   * **Details:** Standard GUI library in Python for creating interactive interfaces.
   * **Reason for Use:** Ease of use and integration with Python for developing user-friendly interfaces.
3. **pandas and openpyxl:**
   * **Purpose:** Data manipulation and report generation.
   * **Details:** pandas for data analysis and openpyxl for Excel report generation.
   * **Reason for Use:** Efficient handling of structured data and seamless integration with Python.

# **Chapter 2: First Deliverable**

## 2.1. Introduction:

The first milestone in our project journey is the development of the Network Traffic Monitor and Logger's initial version. This chapter delves into the feasibility, scope, costing, team structure, tools, and risk management involved in delivering this crucial phase.

## 2.2. Project/Product Feasibility Report:

### 2.2.1. Technical Feasibility:

Our project harnesses existing technologies to create a robust monitoring tool:

1. **Python:**
   * **Rationale:** Python's extensive library support, especially with packet capturing libraries like pyshark, makes it the ideal choice for our project.
2. **Pyshark:**
   * **Rationale:** Pyshark, being a Python wrapper for Wireshark's tshark, facilitates efficient packet capturing and parsing, crucial for real-time network monitoring.
3. **Tkinter:**
   * **Rationale:** Tkinter, as a standard GUI toolkit in Python, ensures the development of a user-friendly interface without additional complexities.

### 2.2.2. Operational Feasibility:

Our tool seamlessly integrates into existing operational setups with its:

* Intuitive GUI, designed for ease of use and quick adoption.
* Compatibility with various network configurations, minimizing setup requirements.

### 2.2.3. Economic Feasibility:

This project proves economically viable owing to:

* Low hardware costs, requiring only standard PCs with network interface cards.
* Utilization of open-source libraries like pyshark and tkinter, significantly reducing licensing expenses.

### 2.2.4. Schedule Feasibility:

The project is structured with a clear timeline, ensuring timely delivery:

* Phases include planning, development, testing, and deployment.
* A detailed Gantt chart outlines milestones and deadlines, facilitating effective project management.

### 2.2.5. Specification Feasibility:

Our tool meets the necessary specifications for a network monitoring tool, offering:

* Real-time monitoring capabilities.
* Detailed logging of traffic information.
* A user-friendly GUI for seamless interaction.

### 2.2.6. Information Feasibility:

With continuous packet capturing and logging, our tool ensures the availability of accurate real-time network data, critical for effective analysis and decision-making.

### 2.2.7. Motivational Feasibility:

The tool's intuitive interface and proactive alerting system motivate users to actively engage in network security:

* User-friendly GUI encourages regular monitoring and interaction.
* Real-time alerts raise awareness about potential security threats, fostering a proactive security culture.

### 2.2.8. Legal & Ethical Feasibility:

Our project adheres to legal guidelines and ethical considerations concerning network monitoring:

* Compliance with privacy laws ensures responsible data handling.
* Ethical standards are upheld, promoting transparency and accountability.

## 2.3 Project/Product Scope:

The initial version of the Network Traffic Monitor and Logger encompasses:

* Real-time network traffic monitoring.
* Logging and categorization of traffic based on predefined criteria.

## 2.4 Project/Product Costing:

### 2.4.1. Cost Estimation by Function Point Analysis:

We estimate function points for key project functions:

1. **Packet Capturing:** 30 function points.
2. **Traffic Logging:** 25 function points.
3. **GUI Development:** 20 function points.
4. **Alert System:** 15 function points.

### 2.4.2. COCOMO’81:

Assuming a project size of 3 KDSI, the effort and time required are approximately 8.18 person-months and 5.54 months, respectively.

### 2.4.3. Activity-Based Costing:

Activity-based costing allocates costs across project activities, ensuring effective cost management:

* Development costs are estimated based on key project functions.
* Testing and deployment activities are also accounted for.

## 2.5. Task Dependency Table:

Understanding task dependencies is crucial for effective project planning and execution:

* Packet capturing is independent.
* Traffic logging depends on packet capturing.
* GUI development follows traffic logging.
* Testing and debugging rely on all preceding tasks.

## 2.6. Critical Path Method (CPM):

The CPM diagram identifies the sequence of critical tasks, ensuring timely project completion.

## 2.7. Gantt Chart:

The Gantt chart visually represents project timelines, highlighting key milestones and task durations.

## 2.8. Tools and Technology with Reasoning:

We employ specific tools and technologies chosen for their suitability and effectiveness in achieving project objectives:

1. **Python:** Versatile and well-suited for rapid development.
2. **Pyshark:** Efficient packet capturing capabilities.
3. **Tkinter:** Standard GUI toolkit for seamless interface development.

## 2.9. Vision Document:

The vision of our project is to enhance network security by providing a real-time monitoring and logging tool that swiftly identifies potential threats.

## 2.10. Risk List:

Identifying and mitigating potential risks is essential for project success:

1. **Incomplete Packet Capture Due to Network Congestion:** Buffer management techniques to handle high traffic volumes.
2. **GUI Responsiveness Under High Traffic Conditions:** GUI code optimization for efficient real-time data updates.

## 2.11. Product Features/Product Decomposition:

The Network Traffic Monitor and Logger offer core and additional features for comprehensive network monitoring:

1. **Core Features:** Real-time monitoring and detailed logging.
2. **Additional Features:** User-friendly GUI and proactive alert system.

# **Chapter 3: Second Deliverable for Object-Oriented Approach**

## 3.1. Introduction:

The second deliverable delves into the object-oriented design approach for the Network Traffic Monitor and Logger. It aims to provide a thorough understanding of the system's structure, specifications, and requirements allocation. This chapter encompasses system specifications, external entity identification, context-level data flow diagram, "shall" statements, requirement allocation, prioritization, and a detailed exploration of the object-oriented approach's implementation.

### System Specifications:

The system specifications serve as a blueprint for the Network Traffic Monitor and Logger, delineating its core functionalities and features. Derived from the project scope and feasibility studies, these specifications are pivotal in guiding the design and development process:

1. Capturing Network Traffic:
   * The system must adeptly capture live network traffic in real-time.
   * It should support various network protocols, including TCP, UDP, HTTP, and DNS.
2. Identifying and Logging Bad Traffic:
   * The system must discern and log traffic from predefined bad IP addresses and domains.
   * Logged information should encompass details like timestamps, source IP, destination IP, and the protocol used.
3. Providing a User-Friendly Interface:
   * The system should offer an intuitive GUI, empowering users to interact seamlessly.
   * GUI functionalities should enable users to commence or cease monitoring, view real-time logs, and receive alerts.

### Identifying External Entities:

Understanding the external entities interfacing with the system elucidates its boundaries and interactions:

1. Network Interface:
   * The network interface serves as the conduit for capturing network traffic.
   * It furnishes the data packets for the system's monitoring and analysis.
2. User:
   * Users engage with the system through the GUI.
   * They possess the capability to initiate or halt the monitoring process and peruse the logged data.

### Context Level Data Flow Diagram:

The Context Level Data Flow Diagram (DFD) furnishes a top-level abstraction of the system, delineating its interactions with external entities:

* **Process:** Network Traffic Monitor and Logger
* **External Entities:**
  + Network Interface
  + User
* **Data Flows:**
  + Network Data from Network Interface to System
  + Commands and Queries from User to System
  + Logs and Alerts from System to User

### Capture "Shall" Statements:

The "shall" statements stipulate the mandatory requisites for the system, ensuring comprehensive coverage of essential functionalities:

1. The system shall capture live network traffic:
   * This mandate underscores the core functionality of real-time network monitoring.
2. The system shall log traffic details:
   * This requirement guarantees the comprehensive recording and analysis of network traffic data.
3. The system shall provide a GUI for user interaction:
   * This stipulation ensures user accessibility and ease of operation.

### Allocate Requirements:

Requirement allocation organizes the development process by assigning priority levels to each requisite, ensuring the prioritized development and testing of critical functionalities:

1. Packet Capturing:
   * **Priority:** High
   * **Justification:** Imperative for the system's core functionality.
2. Traffic Logging:
   * **Priority:** High
   * **Justification:** Critical for the thorough recording and analysis of network traffic data.
3. GUI Development:
   * **Priority:** Medium
   * **Justification:** Essential for user interaction, albeit can be developed subsequent to core functionalities.

### Prioritize Requirements:

Prioritizing requirements ensures the prompt addressal of critical functionalities, optimizing the development process and ensuring timely delivery of essential features:

1. Packet Capturing:
   * Highest priority owing to its pivotal role in real-time network monitoring.
2. Traffic Logging:
   * Second-highest priority to ensure the comprehensive recording and availability of data for analysis.
3. GUI Development:
   * Medium priority to furnish an interface for user interaction.

### Requirements Traceability Matrix:

The Requirements Traceability Matrix (RTM) establishes a clear mapping between requirements and their corresponding implementation, ensuring comprehensive coverage:

* **Requirement:** Implemented in
  + Capture Traffic: monitor\_traffic function
  + Log Traffic: monitor\_traffic function
  + GUI Interaction: tkinter main loop

Detailed System Specifications:

A detailed exploration of system specifications reveals the intricacies of the Network Traffic Monitor and Logger's components and functionalities:

1. Capturing Network Traffic:
   * **Functionality:** The system captures live network traffic via the pyshark library.
   * **Implementation:** [Detailed implementation code provided]
2. Identifying and Logging Bad Traffic:
   * **Functionality:** The system identifies bad traffic based on predefined bad IP addresses and domains, subsequently logging these incidents for further analysis.
   * **Implementation:** [Detailed implementation code provided]
3. Providing a User-Friendly Interface:
   * **Functionality:** The system furnishes a GUI developed with tkinter, facilitating user interaction and real-time monitoring capabilities.
   * **Implementation:** [Detailed implementation code provided]

Object-Oriented Approach:

Embracing an object-oriented paradigm, the system design emphasizes modularity and reusability, encapsulating functionalities within well-defined objects:

* **Class Design:**
  1. **NetworkMonitor Class:**
     + **Responsibilities:** Manages packet capturing and logging.
     + **Attributes:** [Detailed attributes provided]
     + **Methods:** [Detailed methods provided]
  2. **GUI Class:**
     + **Responsibilities:** Oversees user interface management and interaction.
     + **Attributes:** [Detailed attributes provided]
     + **Methods:** [Detailed methods provided]

## 3.2. Example:

# **Chapter 4: Third Deliverable for Object-Oriented Approach**

## 4.1. Introduction:

This chapter delves into the detailed design of the Network Traffic Monitor and Logger using the object-oriented approach. It encompasses comprehensive design diagrams and models that illustrate the system's architecture and interactions. The deliverables include the domain model, system sequence diagrams, collaboration diagrams, operation contracts, design class diagrams, state chart diagrams, and data models. These elements collectively ensure a robust, scalable, and maintainable system.

## 4.2. Domain Model:

The domain model represents the conceptual framework of the Network Traffic Monitor and Logger. It identifies the primary objects within the system and their relationships, laying the groundwork for the system's architecture.

Key Entities and Relationships:

1. NetworkMonitor:
   * Attributes: bad\_ips, bad\_domains, log\_queue, stop\_capture
   * Responsibilities: Capture packets, log traffic, manage bad sites and domains.
2. Packet:
   * Attributes: timestamp, source\_ip, destination\_ip, protocol
   * Responsibilities: Represents network traffic data.
3. User:
   * Attributes: None specific
   * Responsibilities: Interacts with the system through the GUI.
4. GUI:
   * Attributes: root, output\_text
   * Responsibilities: Provides an interface for user interaction, displays logs and alerts.

## 4.3. System Sequence Diagram:

The System Sequence Diagram (SSD) depicts the sequence of interactions between the system's components and external entities during key operations. It focuses on the high-level behavior of the system.

Key Scenarios:

1. Starting Network Monitoring:
   * User initiates the start command via GUI.
   * GUI sends a start request to NetworkMonitor.
   * NetworkMonitor begins capturing packets.
2. Logging Network Traffic:
   * NetworkMonitor captures a packet.
   * Packet data is processed and logged.
   * GUI updates the display with new log entries.
3. Stopping Network Monitoring:
   * User initiates the stop command via GUI.
   * GUI sends a stop request to NetworkMonitor.
   * NetworkMonitor stops capturing packets.

Sequence Diagram:

The sequence diagram provides a detailed view of the interactions between objects in the system during a particular sequence of events. It extends the SSD by adding more granular interactions and method calls.

Detailed Scenarios:

1. Packet Capturing and Logging:
   * User -> GUI: Clicks "Start Monitoring"
   * GUI -> NetworkMonitor: start\_capture()
   * NetworkMonitor -> pyshark: sniff\_continuously()
   * pyshark -> NetworkMonitor: Packet captured
   * NetworkMonitor -> Packet: process\_packet()
   * Packet -> NetworkMonitor: Processed data
   * NetworkMonitor -> log\_queue: Log entry
   * NetworkMonitor -> log\_file: Write log
   * NetworkMonitor -> GUI: Update log display
2. GUI Interaction:
   * User -> GUI: Clicks "Stop Monitoring"
   * GUI -> NetworkMonitor: stop\_capture()
   * NetworkMonitor: Stops capturing packets

## 4.4. Collaboration Diagram:

The collaboration diagram illustrates the interactions between objects in the system, emphasizing the structural organization and collaboration required to achieve specific tasks.

Key Interactions:

1. NetworkMonitor and Packet:
   * NetworkMonitor captures packets and delegates processing to the Packet object.
   * Packet object processes data and returns relevant information for logging.
2. GUI and NetworkMonitor:
   * GUI initiates and controls the start and stop commands.
   * NetworkMonitor updates the GUI with log entries.
3. NetworkMonitor and Log File:
   * NetworkMonitor writes log entries to a file for persistent storage.

Operation Contracts:

Operation contracts define the operations performed by the system, detailing their responsibilities, input/output, and pre/post conditions. These contracts ensure that each operation is well-understood and consistently implemented.

1. Packet Capture Operation:
   * Operation: start\_capture()
   * Responsibility: Captures live packets from the network interface.
   * Input: None
   * Output: Packets captured
   * Pre-condition: NetworkMonitor is initialized.
   * Post-condition: Packets are continuously captured until stop\_capture() is called.
2. Log Operation:
   * Operation: log\_traffic()
   * Responsibility: Logs traffic details to a file and updates the GUI.
   * Input: Packet data
   * Output: Log entries
   * Pre-condition: Packet data is available.
   * Post-condition: Log entries are written to the file and displayed on the GUI.

## 4.5. Design Class Diagram:

The design class diagram presents the static structure of the system, showing classes, their attributes, methods, and the relationships among them. It provides a blueprint for the implementation phase.

Key Classes and Relationships:

1. NetworkMonitor:
   * Attributes: bad\_ips, bad\_domains, log\_queue, stop\_capture
   * Methods: start\_capture(), stop\_capture(), process\_packet()
   * Relationships: Aggregates Packet, Collaborates with GUI
2. Packet:
   * Attributes: timestamp, source\_ip, destination\_ip, protocol
   * Methods: **init**(), process()
   * Relationships: Aggregated by NetworkMonitor
3. GUI:
   * Attributes: root, output\_text
   * Methods: create\_widgets(), start\_monitoring(), stop\_monitoring(), update\_gui()
   * Relationships: Collaborates with NetworkMonitor

## 4.6. State Chart Diagram:

The state chart diagram models the dynamic behavior of the system, showing the states an object can be in and the transitions between these states based on events.

Key States and Transitions:

1. Idle:
   * Initial state when the system is not monitoring.
   * Transitions to Monitoring on start command.
2. Monitoring:
   * Active state where the system captures and logs packets.
   * Transitions to Idle on stop command.
   * Transitions to Alert on detecting bad traffic.
3. Alert:
   * State indicating the detection of bad traffic.
   * Returns to Monitoring after handling the alert.

## 4.7. Data Model:

The data model represents the data structures used within the system, including how data is stored, accessed, and manipulated. It ensures data consistency and integrity throughout the system.

Key Data Structures:

1. Log Entry:
   * Attributes: timestamp, log\_type, source\_ip, destination\_ip, protocol
   * Relationships: Created by NetworkMonitor, Displayed by GUI
2. Bad IPs and Domains:
   * Attributes: ip\_list, domain\_list
   * Relationships: Managed by NetworkMonitor

Detailed Design Elements:

1. NetworkMonitor Class:
   * The NetworkMonitor class is the core component responsible for capturing and processing network traffic. It interfaces with the network, processes packet data, and manages logging operations.
   * Attributes, Methods, and Implementation provided.
2. GUI Class:
   * The GUI class handles the user interface, allowing users to start and stop monitoring, view log entries, and interact with the system.
   * Attributes, Methods, and

# **Chapter 5: Detailed Diagrams and Models**

## Introduction:

In this chapter, we delve into the intricate details of the diagrams and models utilized in our project. These visual representations serve as blueprints, aiding in understanding the system's architecture, data flow, and behavior.

5.1 Data Flow Diagrams (DFDs)

Data Flow Diagrams (DFDs) are powerful tools for visualizing the flow of data within a system. They provide a structured approach to representing processes, data stores, and the movement of data between them.

5.1.1 Context Level DFD

The Context Level DFD provides a high-level overview of the system, illustrating its interactions with external entities. This diagram lays the foundation for understanding how data flows into and out of the system.

Description:

The Context Level DFD depicts the major processes within the system, including user interaction, data processing, and external data sources. It highlights the boundaries of the system and the interfaces with external entities.

Detailed Components:

* External Entities: Represented as squares, these entities interact with the system.
* Processes: Represented as circles, processes denote the activities performed within the system.
* Data Flows: Represented as arrows, data flows depict the movement of data between processes and external entities.
* Data Stores: Represented as rectangles, data stores indicate where data is persisted within the system.

5.1.2 Level 1 DFD

The Level 1 DFD provides a more detailed view of the system by decomposing the processes identified in the Context Level DFD into subprocesses.

Description:

The Level 1 DFD breaks down the high-level processes into finer-grained activities, offering a more granular understanding of how data moves through the system.

Detailed Components:

* Subprocesses: Further decomposition of processes from the Context Level DFD into smaller activities.
* Data Transformations: Detailed depiction of how data is manipulated within each subprocess.
* Refined Data Flows: More specific data flows between subprocesses and data stores.
* Control Flows: Represented to show the sequence of activities within the system.

## 5.2 Entity-Relationship Diagram (ERD)

The Entity-Relationship Diagram provides a visual representation of the relationships between entities in the system's data model.

5.2.1 Entity-Relationship Diagram

The ERD depicts the entities, attributes, and relationships within the system, aiding in database design and understanding the data model.

Description:

The ERD illustrates the logical structure of the database, including entities such as users, data sources, and network traffic. It highlights the relationships between these entities, such as one-to-many or many-to-many relationships.

Detailed Components:

* Entities: Represented as rectangles, entities represent real-world objects or concepts.
* Attributes: Descriptive properties of entities, depicted within ovals connected to entities.
* Relationships: Lines connecting entities, illustrating how they are associated with each other.
* Cardinality: Indicates the number of instances of one entity that are related to another entity.

## 5.3. Data Flow / Sequence Diagrams

Sequence Diagrams visualize the interactions between different components or objects in the system over time, showcasing the sequence of messages exchanged between them.

5.3.1 Sequence Diagram for Traffic Monitoring

The Sequence Diagram for Traffic Monitoring illustrates the flow of messages between the GUI, traffic monitoring module, and external entities during the process of monitoring network traffic.

Description:

This sequence diagram outlines the sequence of events triggered when the monitoring process is initiated, including user interactions, data processing, and external communications.

Detailed Components:

* Objects: Represented as rectangles, objects denote the components or entities involved in the sequence of events.
* Messages: Arrows indicating the flow of communication between objects, depicting method calls or data exchanges.
* Lifelines: Vertical dashed lines extending from objects, representing the lifespan of each object during the sequence of events.
* Activation Bars: Horizontal bars on lifelines, indicating the periods when objects are actively processing messages.

## 5.4. State Transition Diagrams

State Chart Diagrams depict the different states that an object or system can transition through in response to events, providing insights into its behavior and lifecycle.

5.4.1 State Chart Diagram for Monitoring State

The State Chart Diagram for Monitoring State outlines the various states of the monitoring process, including idle, active, and stopped states, along with the transitions between them.

Description:

This State Chart Diagram illustrates the lifecycle of the monitoring process, from initialization to termination, and the conditions under which it transitions between different states.

Detailed Components:

* States: Represented as rounded rectangles, states denote the various conditions or modes that the system can be in.
* Transitions: Arrows between states, indicating the conditions or events that trigger state transitions.
* Actions: Activities performed when transitioning between states, depicted within transition arrows.
* Initial and Final States: Symbols indicating the starting and ending points of the state machine.

## 5.5. Architectural Design

The Architectural Design section outlines the overall structure of the system, including the distribution of components, communication protocols, and system boundaries.

Description:

The architectural design provides a high-level overview of the system's structure, including its major components, their interactions, and the deployment architecture. It defines the architectural style, such as client-server or peer-to-peer, and highlights key design decisions.

Detailed Components:

* Component Diagram: Illustrates the structural organization of the system's components and their dependencies.
* Deployment Diagram: Depicts the physical deployment of software components across hardware nodes.
* Communication Protocols: Specifies the protocols used for communication between components, such as HTTP, TCP/IP, or WebSocket.
* System Boundaries: Defines the interfaces and boundaries of the system, including external APIs, user interfaces, and integration points.

## 5.6. Component Level Design

The Component Level Design delves into the internal structure of individual system components, specifying their interfaces, responsibilities, and interactions.

Description:

Component Level Design focuses on the detailed design of each system component, including its internal structure, dependencies, and interfaces. It defines the methods, attributes, and interactions of each component, facilitating modular design and development.

Detailed Components:

* Component Interfaces: Specifies the methods and parameters exposed by each component for interaction with other components.
* Component Responsibilities: Describes the functionality and responsibilities of each component within the system.
* Dependency Management: Identifies dependencies between components and ensures modularity and loose coupling.
* Interaction Diagrams: Illustrates the interactions between components, such as sequence diagrams or collaboration diagrams.

# **Chapter 6: Fourth Deliverable (User Interface Design)**

## 6.1. Introduction:

In this chapter, we embark on an exhaustive exploration of the user interface design for our network traffic monitoring desktop application. The user interface serves as the primary point of interaction between the user and the system, thus warranting meticulous attention to detail and usability considerations.

Importance of User Interface Design: The significance of user interface design cannot be overstated. A well-designed user interface not only enhances user experience but also contributes to the efficiency, effectiveness, and satisfaction of users interacting with the system. In the context of our network traffic monitoring application, a thoughtfully crafted user interface will empower users to effortlessly monitor network activity, interpret logs, and manage monitoring settings.

## 6.2. Site Maps:

Site maps provide a structural overview of the application's user interface, delineating the hierarchy of screens and the navigation pathways between them. Let's delve into the intricacies of crafting a comprehensive site map for our desktop application.

Site Map (Desktop Application): Main Window:

* Header: Displays real-time information such as the current IP address and system status.
* Navigation Bar: Facilitates navigation between different sections of the application.
* Monitoring Controls: Enables users to initiate, pause, and stop network traffic monitoring.
* Log Display: Presents a real-time feed of network activity logs.
* Settings Panel: Allows users to configure monitoring preferences and customize display settings. Settings Panel:
* General Settings: Configure basic monitoring preferences such as update frequency and log retention period.
* Advanced Settings: Fine-tune monitoring parameters such as filter criteria and protocol-specific settings.

Description: The site map serves as a visual blueprint of the application's user interface structure, delineating the arrangement of components and the navigational pathways available to users. By meticulously defining the layout and functionality of each screen, we ensure coherence and ease of navigation throughout the application.

## 6.3. Storyboards:

Storyboards offer a narrative depiction of the user's journey through the application, illustrating the sequence of interactions and transitions between different screens. Let's immerse ourselves in crafting immersive and intuitive storyboards for our network traffic monitoring application.

Storyboard (Placeholder): Scenario: Initiating Network Traffic Monitoring

1. User Launches Application: Upon launching the application, the user is greeted with the main window, showcasing real-time system information and monitoring controls.
2. Initiating Monitoring: The user clicks the "Start Monitoring" button to initiate network traffic monitoring.
3. Real-time Log Display: As monitoring commences, the log display panel populates with real-time network activity logs, providing users with actionable insights into network behavior.
4. Monitoring in Progress: The user observes the continuous flow of log entries, monitoring for any suspicious or anomalous network activity.
5. Pausing Monitoring: If necessary, the user can pause monitoring by clicking the "Stop Monitoring" button, temporarily halting the logging of network activity.
6. Resuming Monitoring: Upon resolving the issue or completing the monitoring session, the user can resume monitoring by clicking the "Start Monitoring" button again.

Description: The storyboard narrates a typical user interaction scenario within the application, guiding users through the process of initiating, pausing, and resuming network traffic monitoring. By visualizing the user journey, we gain insights into the user's perspective and refine the user interface design to optimize usability and efficiency.

## 6.4. Navigational Maps:

Navigational maps delineate the hierarchical structure and navigational pathways within the application, guiding users through the interface and facilitating seamless navigation between different sections. Let's elucidate the intricacies of crafting intuitive navigational maps for our desktop application.

Navigational Map (Placeholder): Main Window Navigation:

* Home: Returns users to the main window, providing access to real-time system information and monitoring controls.
* Logs: Navigates users to the log display section, allowing them to view and analyze real-time network activity logs.
* Settings: Directs users to the settings panel, where they can configure monitoring preferences and customize display settings.

Description: The navigational map outlines the hierarchical structure of the application's user interface, delineating the various sections and the navigational pathways available to users. By providing intuitive navigation, we enhance user experience and streamline access to key functionalities within the application.

## 6.5. Traceability Matrix:

The traceability matrix establishes a clear linkage between user interface features and their corresponding implementation within the application, ensuring alignment between design specifications and implementation outcomes. Let's delve into the intricacies of crafting a comprehensive traceability matrix for our network traffic monitoring application.

Traceability Matrix: UI Feature Implemented in Start Monitoring Start Button (Main Window) Stop Monitoring Stop Button (Main Window) Clear Output Clear Button (Main Window) Display Logs Log Display Panel (Main Window) General Settings General Settings Panel (Settings) Advanced Settings Advanced Settings Panel (Settings)

Description: The traceability matrix elucidates the correspondence between user interface features and their implementation within the application, ensuring that each design specification is accurately translated into functional components. By establishing a clear linkage, we facilitate effective tracking and validation of feature implementation, fostering coherence and consistency within the application.

# **Chapter 7: Fifth Deliverable (Software Testing)**

## 7.1. Introduction:

In this chapter, we embark on a comprehensive exploration of software testing for our network traffic monitoring desktop application. Software testing is a critical phase in the software development lifecycle, ensuring that all functionalities work as expected and meet the specified requirements. We will detail the testing plan and related documents, including the test plan, test design specification, test case specification, test procedure specification, test item transmittal report, test log, test incident report, and test summary report.

## 7.2. Test Plan:

The test plan serves as a roadmap for software testing, outlining the objectives, scope, methodology, and resources required for testing. It provides a structured approach to validate the functionality, usability, and performance of the application.

Test Plan Components:

* Objective: To validate the functionality, usability, and performance of the network traffic monitoring application.
* Scope: All features and functionalities of the application, including packet capturing, logging, and GUI interaction.
* Methodology: Combination of manual and automated testing techniques.
* Resources: Test environment, testing tools, and personnel involved in testing.

## 7.3. Test Design Specification:

The test design specification delineates the test scenarios and cases for validating different aspects of the application, including packet capturing, logging, and GUI interaction. It provides a detailed blueprint for conducting tests and evaluating the application's behavior under various conditions.

Test Design Specification Components:

* Packet Capture Test: Validate the accuracy and reliability of packet capturing functionality.
* Log Test: Verify the correctness and completeness of log generation and storage.
* GUI Test: Evaluate the responsiveness and usability of GUI controls for starting, stopping, and clearing monitoring.

## 7.4. Test Case Specification:

The test case specification elaborates on the individual test cases derived from the test design specification. Each test case defines the inputs, expected outcomes, and execution steps for validating specific functionalities of the application.

Test Case Specification Components:

* Packet Capture Test Case:
  + Input: Generate network traffic.
  + Expected Outcome: All packets captured accurately.
  + Execution Steps: Start monitoring, generate traffic, verify captured packets.
* Log Test Case:
  + Input: Monitor network traffic.
  + Expected Outcome: Logs written to file without errors.
  + Execution Steps: Monitor traffic, verify log file contents.
* GUI Test Case:
  + Input: Interact with GUI controls.
  + Expected Outcome: Responsive and accurate GUI functionality.
  + Execution Steps: Start monitoring, stop monitoring, clear logs, verify GUI responsiveness.

## 7.5. Test Procedure Specification:

The test procedure specification outlines the step-by-step procedures for executing each test case and validating the functionality of the application. It provides detailed instructions for testers to follow during the testing process.

Test Procedure Specification Components:

* Packet Capture Test Procedure:
  1. Start the application.
  2. Initiate packet capturing.
  3. Generate network traffic.
  4. Verify captured packets.
* Log Test Procedure:
  1. Launch the application.
  2. Monitor network traffic.
  3. Check log file for correctness.
* GUI Test Procedure:
  1. Open the application.
  2. Interact with GUI controls.
  3. Verify responsiveness and functionality.

## 7.6. Test Item Transmittal Report:

The test item transmittal report documents the transmission of tested functionalities to the quality assurance (QA) team for further validation and verification. It ensures seamless coordination and communication between testing and QA teams.

Test Item Transmittal Report Components:

* List of tested functionalities.
* Date of transmission.
* Responsible personnel.

## 7.7. Test Log:

The test log records all test cases executed during the testing process, along with their results. It provides a comprehensive record of testing activities and outcomes for future reference and analysis.

Test Log Components:

* Test case ID.
* Description of test case.
* Test result (pass/fail).
* Date and time of test execution.
* Comments or observations.

## 7.8. Test Incident Report:

The test incident report documents any issues or anomalies encountered during testing, along with their severity and impact on the application's functionality. It facilitates timely resolution and mitigation of identified issues.

Test Incident Report Components:

* Description of incident.
* Severity level.
* Impact on functionality.
* Steps to reproduce.
* Proposed resolution.

## 7.9.Test Summary Report:

The test summary report provides a comprehensive overview of the testing process, including the number of test cases executed, pass/fail rates, identified issues, and recommendations for improvement. It serves as a valuable artifact for stakeholders to assess the overall quality and readiness of the application for deployment.

Test Summary Report Components:

* Summary of testing activities.
* Pass/fail rates for test cases.
* Identified issues and recommendations.
* Recommendations for improvement.
* Conclusion and next steps.

# **Appendices**

User Interface Guidelines: User interface guidelines provide recommendations and best practices for designing a user-friendly and intuitive interface. It includes principles such as clear labeling, consistent navigation, and responsive design to enhance usability and user experience.

User Interface Guidelines Components:

* Clear Labeling: Ensure all interface elements are labeled clearly and intuitively to guide users effectively.
* Consistent Navigation: Maintain consistency in navigation elements and layout across different screens to minimize cognitive load.
* Responsive Design: Design the interface to adapt seamlessly to various screen sizes and devices, ensuring optimal user experience across platforms.

Guidelines for Research Projects: Guidelines for research projects outline ethical considerations and standards to be followed during the execution of research activities. It encompasses principles such as integrity, confidentiality, and respect for participants' rights to ensure ethical conduct in research endeavors.

Guidelines for Research Projects Components:

* Integrity: Conduct research with honesty, transparency, and adherence to scientific principles.
* Confidentiality: Safeguard sensitive information and ensure the privacy of research participants.
* Respect for Participants' Rights: Obtain informed consent and respect the autonomy and dignity of research participants.

Final Documentation Format Guidelines: Final documentation format guidelines provide a standardized framework for organizing and presenting project documentation. It includes templates and formatting guidelines to ensure consistency and clarity in documentation.

Final Documentation Format Guidelines Components:

* Document Templates: Provide templates for various project documents such as proposals, reports, and presentations to maintain consistency in format and structure.
* Formatting Guidelines: Specify formatting requirements for text, headings, tables, and figures to enhance readability and visual appeal.

Final Evaluation Matrices: Final evaluation matrices are tools used for self-assessment and evaluation of project outcomes. They provide a structured framework for assessing project performance against predefined criteria and objectives.

Final Evaluation Matrices Components:

* Performance Criteria: Define specific criteria and metrics for evaluating project performance, such as functionality, usability, and efficiency.
* Evaluation Process: Outline the process for collecting and analyzing data to assess project outcomes and identify areas for improvement.

Project Registration Form: The project registration form is used to officially register the project with relevant authorities or stakeholders. It captures essential information such as project title, objectives, team members, and timeline to establish a formal record of project initiation.

Project Registration Form Components:

* Project Title: A concise and descriptive title that reflects the project's focus and objectives.
* Project Objectives: Clearly defined goals and objectives that outline the intended outcomes and deliverables of the project.
* Team Members: List of team members involved in the project, along with their roles and responsibilities.
* Timeline: Project timeline indicating key milestones, deadlines, and deliverable dates.

Evaluation Delay Request Form: The evaluation delay request form is used to request an extension or delay in project evaluation due to unforeseen circumstances or challenges. It provides a formal mechanism for communicating delays and seeking approval for revised evaluation timelines.

Evaluation Delay Request Form Components:

* Reason for Delay: Explanation of the reasons for the requested delay, such as technical challenges, resource constraints, or external factors.
* Revised Timeline: Proposed revised timeline for project evaluation, including new deadlines for submission of deliverables and completion of evaluation activities.
* Justification: Justification for the requested extension, highlighting the impact of the delay on project objectives and outcomes.

Show Cause Form: The show cause form is used to address instances where project deadlines or milestones are not met as per the agreed-upon timeline. It requires the project team to provide justification for the delay and outline corrective actions to mitigate future occurrences.

Show Cause Form Components:

* Explanation of Delay: Detailed explanation of the reasons for the delay, including any challenges or obstacles encountered during project execution.
* Impact Analysis: Assessment of the impact of the delay on project objectives, deliverables, and stakeholders.
* Corrective Actions: Plan of action outlining steps to address the delay and prevent recurrence in future project activities.

Project Re-enrollment Form: The project re-enrollment form is used to request re-enrollment or continuation of the project for an extended period beyond the initial timeline. It requires justification for the extension and approval from relevant authorities or stakeholders.

Project Re-enrollment Form Components:

* Justification for Extension: Explanation of the need for project re-enrollment, including unfinished tasks, evolving project requirements, or unforeseen delays.
* Revised Timeline: Proposed timeline for project continuation, indicating new milestones, deliverables, and completion dates.
* Approval Signatures: Signatures of project stakeholders or authorities approving the re-enrollment request.

Change Request Form: The change request form is used to document and track changes to the project scope, requirements, or deliverables. It provides a formal mechanism for requesting, evaluating, and approving changes to ensure alignment with project objectives and stakeholders' needs.

Change Request Form Components:

* Change Description: Description of the proposed change, including its impact on project scope, schedule, and resources.
* Justification: Justification for the change request, highlighting the reasons for the proposed modification and its benefits to the project.
* Impact Assessment: Assessment of the potential impact of the change on project objectives, deliverables, and stakeholders.
* Approval Workflow: Workflow for evaluating and approving the change request, including review by project stakeholders and authorization by project management.