BINDURA UNIVERSITY OF SCIENCE EDUCATION

FACULTY OF SCIENCE EDUCATION

DATA LEAKAGE DETECTION SYSTEM: A CASE STUDY OF ZIMSEC

BY

RUTENDO V. MBOFANA

B193108B

SUPERVISOR

MR. NGWENYA

A PROJECT SUBMITTED TO THE COMPUTER SCIENCE DEPARTMENT IN PARTIAL FULFILLMENT TO THE REQUIREMENTS OF THE HONORS BACHELORS SCIENCE EDUCATION DEGREE IN COMPUTER SCIENCE.

Abstract

In an era where data security is paramount, organizations face increasing threats from unauthorized information disclosure, commonly referred to as data leakage. This paper presents a robust Data Leakage Detection System designed to identify and mitigate instances of sensitive data exfiltration. The system employs a combination of user behavior analytics, watermarking techniques, and rule-based monitoring to trace the origin of leaked data and pinpoint responsible entities. By integrating machine learning algorithms with real-time alert mechanisms, the system enhances detection accuracy while minimizing false positives. Experimental results demonstrate its effectiveness in diverse operational environments, offering a scalable and proactive solution for safeguarding confidential information. This research contributes to the field of cybersecurity by proposing a practical framework that balances detection precision with operational efficiency.

RELEASE FORM

Title of the dissertation: Data Leakage Detection System

To be completed by the student

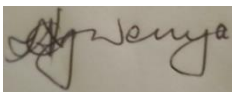
I certify that this dissertation is in conformity with the preparation guidelines as presented in the Faculty Guide and Instructions for Typing Dissertations.

05/ 09/ 24

(Signature of student) (Date)

To be completed by the supervisor

This dissertation is suitable for submission to the Faculty. This dissertation should be checked for conformity with Faculty guidelines.

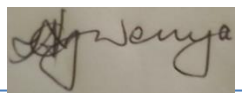


11/10/25

(Signature of Supervisor) (Date)

To be completed by the chairperson of the department

I certify to the best of my knowledge that the required procedures have been followed and the preparation criteria have been met for this dissertation.



11/10/2025

(Signature of the chairperson) (Date

APPROVAL FORM

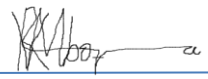
Name of the student: Rutendo V. Mbofana **Registration Number**: B193108B

Title of the dissertation: Data Leakage Detection System

Degree Title: Honors Bachelors Science Education Degree in Computer Science

Year of completion: 2024

Permission is hereby granted to Bindura University of Science Education to make single copies of this dissertation and to lend and sell such copies for private, scholarly, scientific purposes only. The author reserves the rights and neither the dissertation nor extensive extracts from it be granted or otherwise be replicated without the author's consent.



(Signature of student)

05/ 09/ 24

(Date)

ACKNOWLEDGEMENTS

I am extremely grateful and remain indebted to my supervisor, Mr. Ngwenya for being a source of inspiration and for his constant support in the design, implementation, and evaluation of the project. I am thankful to her for constant constructive criticism and invaluable suggestions, which benefited me greatly while developing this project. I take this opportunity to express my sincere thanks and obligation to my friends, family, and the computer science department. Through this column, it would be my utmost pleasure to express my warm thanks to them for the encouragement. Finally, I gratefully acknowledge the support, encouragement and patience of my family and as always nothing in my life would be possible without God, Thank you.

Table of Contents

CHAPTER ONE: PROBLEM IDENTIFICATION

1.1 Introduction

In the world of technology there are huge amounts of data that is needed and used for various reasons which may include statistical valuations and or modelling of Artificial Intelligent machines(Muria,2019). These amounts of data have to be stored in databases depending on the quantity but this has to be done in an intelligent manner so that the data cannot fall into the wrong category or machine which can lead to emerging of problems. Although we try to maintain and safe keep the data there are always incidences where it may leak.

The proposed system's primary task is to detect data leakage and discover the sources of the same. If ever data gets leaked in an organisation, it can be transferred to unauthorised devices and this is why it is crucial to trace the source so that you can tackle the problem.

1.2 Background of Study

Many organizations around the globe rely on the use of data as it is the backbone of their well-being (Kilger, 2020). Because data allows us to measure, we can establish baselines, find benchmarks, and set performance goals, making it essential to store and protect this data. Through this, we can test the success of different strategies and make informed business decisions for sustainable growth. Despite efforts to safeguard data, organizations often face data leakages that can severely impact the company if the data reaches unauthorized devices. For example, ZIMSEC has experienced past exam paper leakages, highlighting the critical need for robust data protection.

A data leak occurs when information is exposed to unauthorized people due to internal errors, such as poor data security, inadequate sanitization, or outdated systems (Muria, 2019). Data leakages are very sensitive and can lead to identity theft, ransomware installation, or data breaches (Kilger, 2020). To ensure data safety, it is necessary to develop a system that can detect data leakage and trace its sources. An advanced data leakage detection system has the capability to prevent data from leaking out of its original source, thereby enhancing data security and integrity.

1.3 Problem Definition

According to Gatsi (2022), Zimbabwean organizations, including ZIMSEC, have been victims of data leakage, leading to significant risks that could jeopardize their operations. For ZIMSEC, past exam paper leakages have compromised the integrity of examinations, leading to severe consequences. These include an increase in crime rates as compromised data enables criminals to impersonate individuals, steal money, and access financial accounts without permission. The leakage of confidential information can lead to the corruption of databases and erosion of client trust, as customers expect their sensitive information to be securely protected. Additionally, data leaks can cause operational downtime, halting business operations while investigations and recovery efforts are underway, which can severely disrupt ZIMSEC's ability to conduct examinations and maintain its reputation.

1.4 Aim

The major aim is to create a web-based data leakage detection system that can allow users to detect and trace data leakages back to their sources. The system should be able to prevent some minor data leakages so as to assist users to safeguard their confidential information.

1.5 Objectives

- Investigate and understand the various methods through which information leaks occur within organizations.
- Develop and implement algorithms to detect data leaks and trace them back to their source, enabling swift mitigation.
- Analyse the effectiveness of the data leakage system.

1.6 Research Questions

- What are the various methods through which information leaks occur within organizations?
- How to develop and implement algorithms to detect data leaks and trace them back to their source, enabling swift mitigation?
- How to analyse the effectiveness of the data leakage system?

1.7 Limitations to Study

The limitations to the study are a result of the following internal and external constraints that are experienced during the course of the study

- High costs that will be experienced in the development of the system are the financial challenges.
- Time management as the developer manages school work and personal life and so making it difficult to manage time.
- The availability of resources that are needed for the development of the system can be scarce.

1.8 Justification

As the world is growing so as the technology and it is our goal to make sure to use the technology advances to our advantage in our lives. The proposed system has more advantages than disadvantages that come with its development and use.

- Organisations will now experience low possibilities of data loss or leakages making it a safe environment for data keeping.
- There is decrease in the costs due to efficiency of the system, there won't be a need for a personnel responsible for safeguarding company data since it will be completed by the proposed system.
- More trading agreements between organisations as both parties will be confident to share their confidential information because of data safety assurances.

1.9 Scope/Delimitations of The System

- Data Types: The system primarily focuses on detecting leakage of structured data, such as databases, spreadsheets, and text documents. It may not comprehensively address leakage of unstructured data like images, videos, or audio files.
- Detection Methods: The system employs signature-based and anomaly-based detection techniques for identifying data leakage. However, it does not cover advanced detection methods like behavior-based analysis or deep learning approaches, which might be more resource-intensive and complex to implement.

1.10 Conclusion

The first chapter is the first stage in the documentation of the proposed system which included the introduction, background, problem definition, objectives to point out the reason for developing the system. This chapter also reveals the tools used in the development and designing of the system.

CHAPTER 2: REQUIREMENTS SPECIFICATION

2.1 Introduction

Smithson (2022) define a systematic review as a thorough analysis of existing literature aimed at consolidating current knowledge and pinpointing research gaps in the realm of data leakage detection systems. This review meticulously scrutinizes studies concerning various facets of data leakage detection, including detection techniques, ethical considerations, and the impact of emerging technologies on safeguarding sensitive information. Through synthesizing these findings, the review aims to offer insights into effective strategies for enhancing data security in the digital era. Additionally, it will elucidate key methodologies and tools utilized in detecting data leakage, shedding light on emerging trends and challenges in this field. By delving into the intricacies of data leakage detection systems, this review endeavors to foster advancements in data security practices and scholarly discourse.

2.2 Requirements Analysis

To gather specifications of the new system, the researcher is going to collect data from the expected users of the system and the other immediate stakeholders who are going to be involved in working of the system. Several data collection tools and methods that are going to be used to collect data from the system users and stakeholders are going to be listed herein.

2.2.1 Questionnaires

The researcher approached ZIMSEC to collect research data from employees. Questionnaires serve as structured instruments for gathering data by posing a series of inquiries to respondents, available in diverse formats like paper-based or electronic, encompassing both closed-ended (e.g., multiple-choice) and open-ended (e.g., free-text) questions. The utilization of questionnaires in this study is motivated by several factors. Firstly, they offer efficiency by enabling researchers to efficiently collect data from a large pool of participants, catering to the potentially varied user base and stakeholders associated with the new system. Secondly, through standardized questions, questionnaires ensure uniformity in data collection, fostering comparability and facilitating analysis across all respondents. Furthermore, questionnaires

provide a degree of anonymity to respondents, encouraging candid responses, particularly on sensitive topics like assessing the functionality of a new system or expressing concerns.

Moreover, closed-ended questions within questionnaires yield quantitative data conducive to statistical analysis, aiding in the identification of trends, patterns, and correlations within the dataset. This analytical approach unveils valuable insights into user preferences and priorities. Additionally, by addressing a spectrum of topics concerning the new system's functionalities, questionnaires enable researchers to garner comprehensive feedback from a diverse array of stakeholders. This inclusive feedback mechanism ensures that the perspectives of various user cohorts, including ZIMSEC employees, are duly considered throughout the system's developmental phase.

2.2.2 Interviews

Interviews represent structured or semi-structured dialogues between a researcher and individual or groups of participants, aimed at gathering qualitative data by eliciting insights, opinions, and experiences pertinent to the topic at hand. They present an avenue for comprehensive exploration of participants' viewpoints, facilitating nuanced comprehension and robust data acquisition. Within the scope of this study, interviews stand as a significant method for data collection. Through interactions with participants, interviews offer a medium to delve into their perceptions, attitudes, and behaviours concerning data security and leakage detection. By employing open-ended inquiries and probing follow-ups, researchers can unearth nuanced insights that may evade other data collection approaches.

Interviews foster the examination of intricate matters, empowering participants to articulate their perspectives and experiences authentically. Additionally, they enable researchers to clarify responses, delve deeper into specific areas of interest, and capture rich, contextualized data. In the context of this study, interviews hold promise in providing valuable insights into the challenges surrounding data security, the efficacy of current detection mechanisms, and stakeholders' viewpoints on the integration of technological solutions in safeguarding sensitive information.

2.3 Data Requirements

2.3.1 Input Requirements

- Data Sources: The system should be able to accept input from various sources where sensitive data may reside, including but not limited to databases, file systems, emails, and cloud storage platforms.
- Data Types: The system should support a wide range of data types, including text documents, spreadsheets, images, audio files, and video files, to effectively identify potential instances of data leakage.
- Configuration Settings: Users should be able to input and customize configuration settings such as sensitivity thresholds, keyword lists, and file types to be monitored, allowing for tailored detection strategies based on specific organizational needs.
- Access Controls: Input mechanisms should incorporate access controls to ensure that only authorized personnel can configure and interact with the system, safeguarding sensitive configuration data and maintaining system integrity.
- Reporting Criteria: Users should be able to specify criteria for generating reports, including frequency, format, and content, to facilitate compliance auditing, incident analysis, and decision-making processes.
- Incident Handling Procedures: Input requirements should include mechanisms for users to define and input incident handling procedures, including escalation paths, notification criteria, and remediation actions, to ensure timely and effective response to detected data leakage incidents.

2.3.2 Output Requirements

The output requirements for the system primarily revolve around delivering clear and comprehensive reports to users. These reports should include detailed analyses of submitted documents, highlighting instances of potential plagiarism, matched sources, similarity percentages, and originality scores. The system should generate reports in user-friendly formats, facilitating easy interpretation by instructors and students alike. Additionally, the system should provide actionable feedback and recommendations for improving academic writing and citation practices. The system should offer standalone functionality, enabling users to submit documents directly to the system for analysis and receive prompt feedback. Ensuring

the security and confidentiality of user data and plagiarism reports is also crucial, necessitating robust privacy measures and compliance with relevant data protection regulations.

2.4 Processing Requirements

The system's processing requirements are central to its efficacy in detecting AI-based plagiarism. Firstly, the system necessitates advanced natural language processing (NLP) algorithms capable of analysing complex textual data from diverse sources, ensuring thorough comparison against extensive databases of academic literature and student submissions. Additionally, efficient pattern recognition and machine learning models are imperative to identify similarities and discrepancies within documents, distinguishing between original work and plagiarized content. Furthermore, the system must employ scalable processing infrastructure to accommodate varying workloads and ensure timely analysis of submitted documents, facilitating swift feedback to users. Robust security measures are also essential to safeguard sensitive academic materials and uphold user privacy throughout the processing pipeline. In essence, the system's processing capabilities form the backbone of its functionality, empowering it to effectively combat plagiarism in academic settings.

2.5 Software Requirements

- Visual Studio 2022 and above
- XAMPP
- Windows, macOS, and Linux
- Text editor (VSCode or sublime)
- PHP 8 and above
- Chrome browser

2.6 Hardware Requirements

The system currently under development is entirely new, being created from the ground up. To facilitate its installation and usage, specific hardware will be necessary. The minimal hardware specifications essential for the proper functioning of the system include the following:

Table 1:Personal Computer Specifications

Component	Specification
CPU	Intel core i7 2.99 GHz
Memory	8 GB
Screen Size	15.6-inch diagonal HD BrightView LED-backlit display (1366x768)
Graphics Hardware	Intel HD Graphics 3000
Wireless Connectivity	802.11b/g/n WLAN and Bluetooth

2.5 Conclusion

The chapter concentrated on analysing the specifications needed for the requirements and the methodologies utilized to derive these outcomes, encompassing both software and hardware aspects. It offers an outline of the envisioned operation of the system while delineating the essential prerequisites for seamless functionality without encountering errors or interruptions.

CHAPTER 3: DESIGN

3.1 Introduction

This chapter describes the internal operations of the Data Leakage System designed to detect and address the issue of leaking question papers for ZIMSEC. The system continuously monitors the number of connected devices and their properties, and it records all user

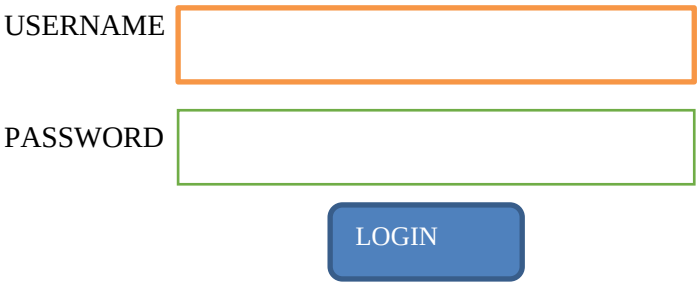
operations in the database, such as copying files or performing any actions. The chapter also demonstrates the user interface design and how the system behaves when various operations are performed.

To understand the system better, it is split into two main modules: the interactive frontend and the database. Users interact with the system through a user-friendly interface, and all their activities are logged in the database for tracking and analysis.

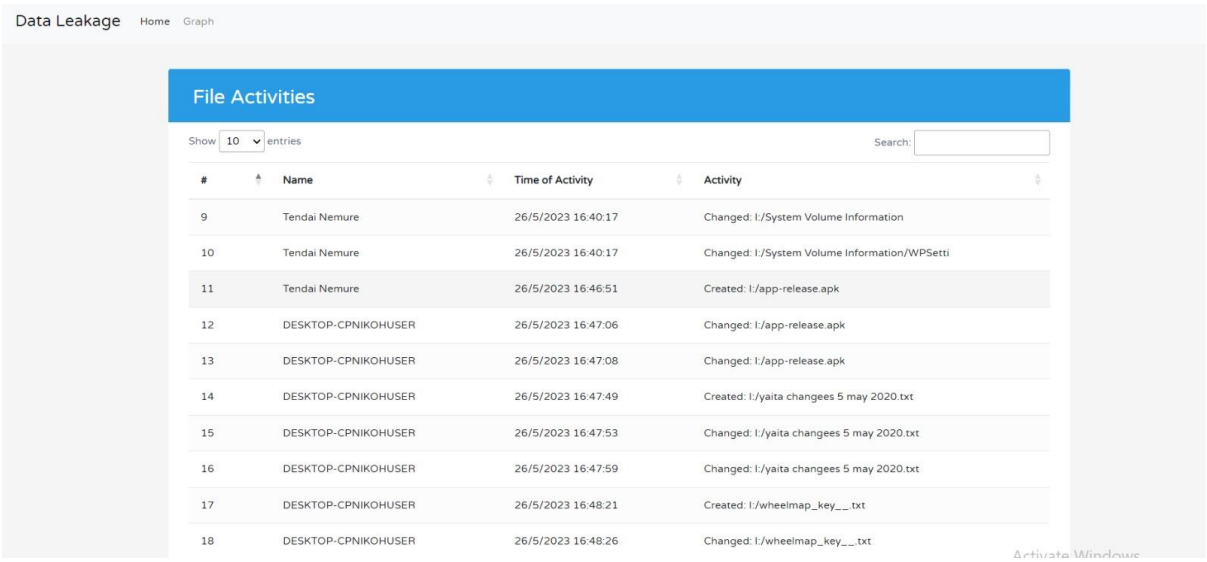
3.2 User Interface Design

Admin Module

Log In Panel: The admin has to enter a username and password to gain access to the system.



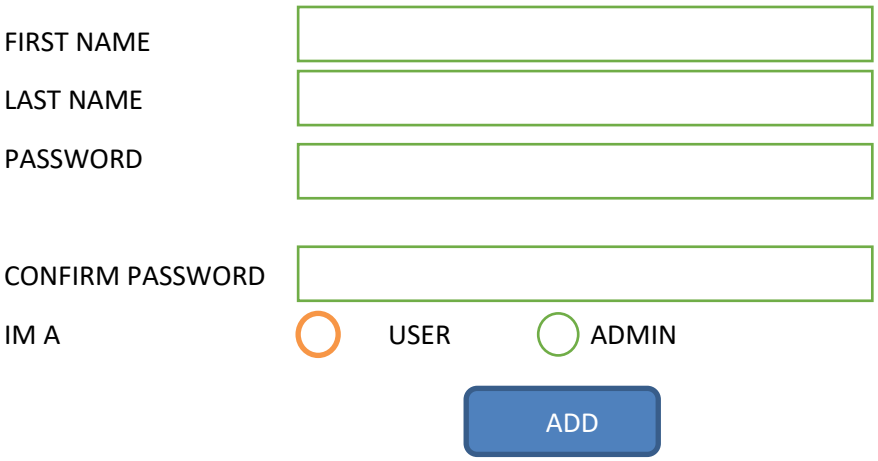
Admin Dashboard: The dashboard provides an overview of the system, including the number of connected devices, user activities, and various system logs.



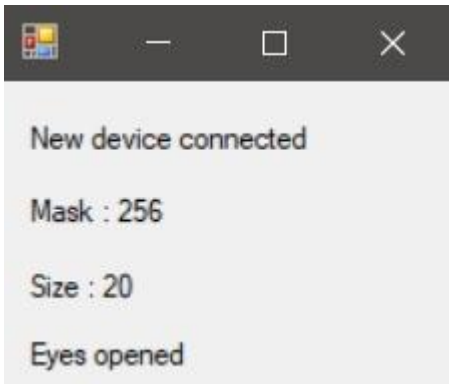
Side Bar Menu: Includes navigation options such as managing users, viewing device properties, and accessing activity logs.

Manage Users: Admins can add new users by entering their details and assigning roles.

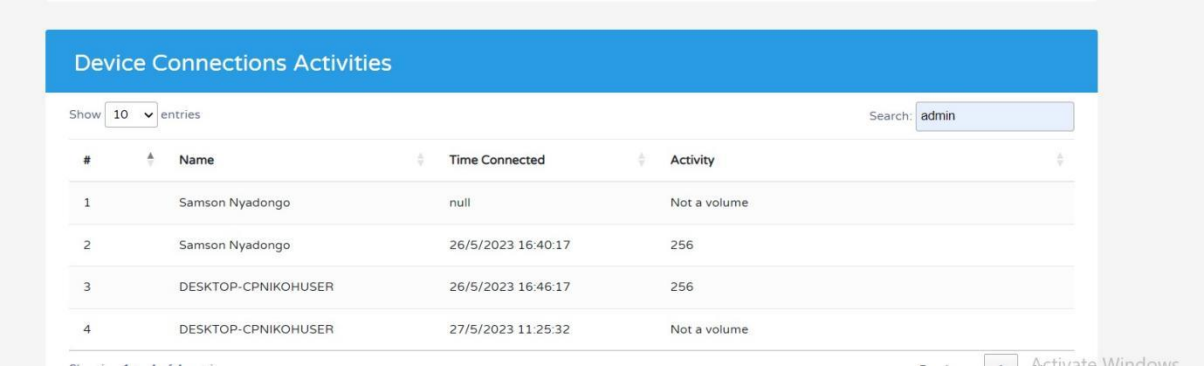
Then there is the Add New User Panel.



Device Monitoring: Displays a list of all connected devices, showing properties such as device type, IP address, and connection status.

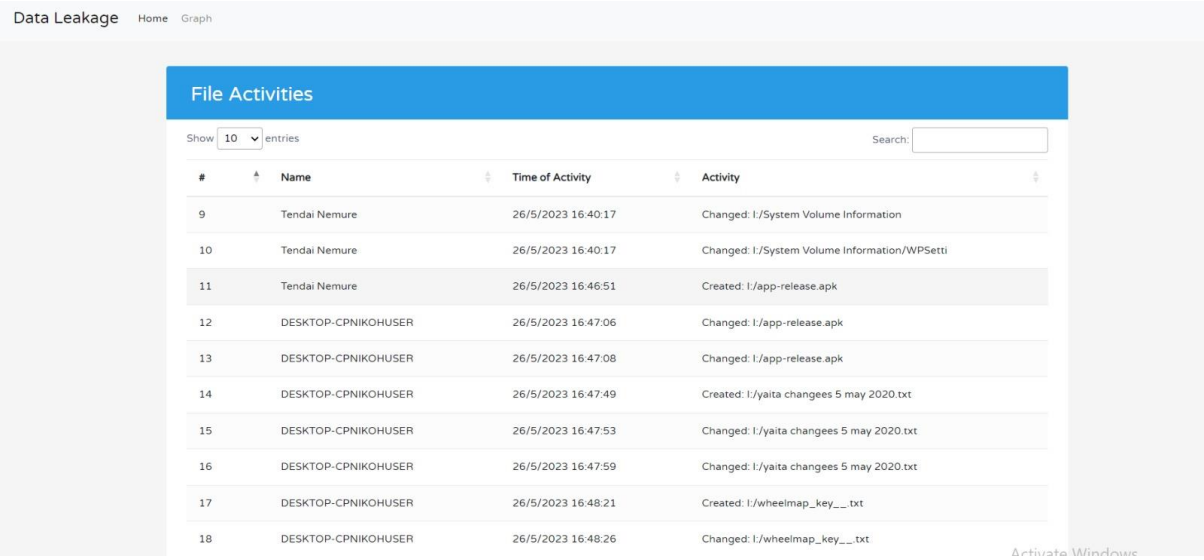


Activity Logs: Shows a detailed log of all user operations, including file copies, uploads, downloads, and any other significant actions. Each log entry includes the username, action performed, timestamp, and any relevant details.



Notifications: Admins receive real-time notifications for suspicious activities, such as multiple failed login attempts or unusual file operations.

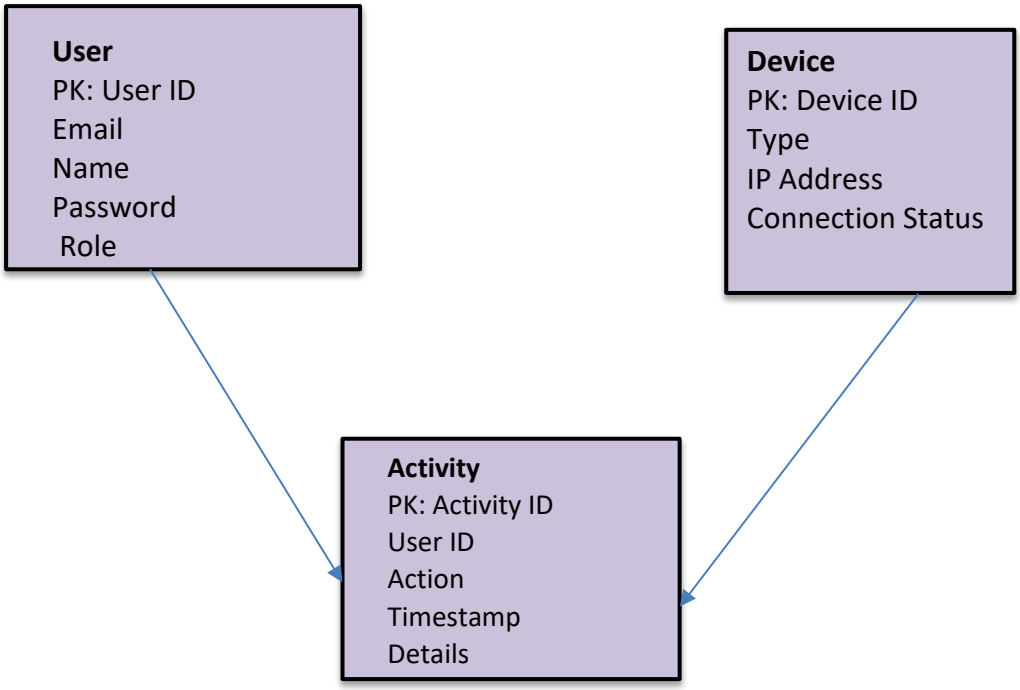
Audit Trails: Admins can view and analyze audit trails, which provide a chronological record of system activities and changes.



3.4 Database Design

The Entity-Relation (ER) model is used to systematically analyze data requirements and produce a well-designed database. It helps explain the logical structure of the database, showing relationships among data elements.

ER Diagram: Illustrates the entities (users, devices, activities, etc.) and their relationships within the system.



- **User**: This entity represents the users of the system. Attributes include:
  - **PK: User ID**: Primary key, unique identifier for each user.
  - **Email**: Email address of the user.
  - **Name**: Name of the user.
  - **Password**: Password for user authentication.
  - **Role**: Role of the user (e.g., admin, regular user).
- **Device**: This entity represents the devices connected to the system. Attributes include:
  - **PK: Device ID**: Primary key, unique identifier for each device.
  - **Type**: Type of the device (e.g., computer, mobile).
  - **IP Address**: IP address of the device.
  - **Connection Status**: Status indicating if the device is connected or disconnected.
- **Activity**: This entity logs all actions performed by users. Attributes include:

- **PK: Activity ID**: Primary key, unique identifier for each activity.
- **User ID**: Foreign key referencing the user who performed the activity.
- **Action**: Description of the action performed.
- **Timestamp**: Time when the action was performed.
- **Details**: Additional details about the activity.

The arrows indicate the relationships:

- A **User** performs multiple **Activities**.
- A **Device** can be used by multiple **Users**.

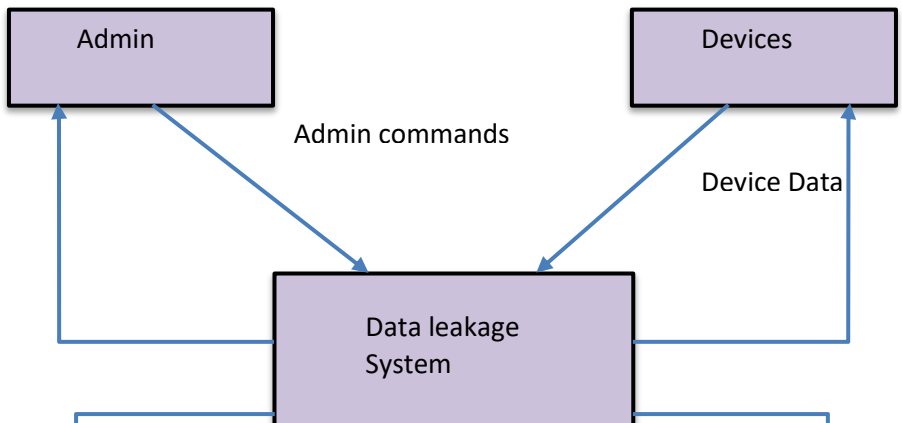
This structure allows the system to effectively monitor and log all operations and device connections, helping to detect and address any potential data leakage issues.

3.5 Processes Design

Data Flow Diagrams (DFDs): Represent the processes the system performs and how data flows between them.

Context Diagram (DFD Level 0): Defines the boundaries of the Data Leakage System, showing information flow between the system and external entities.

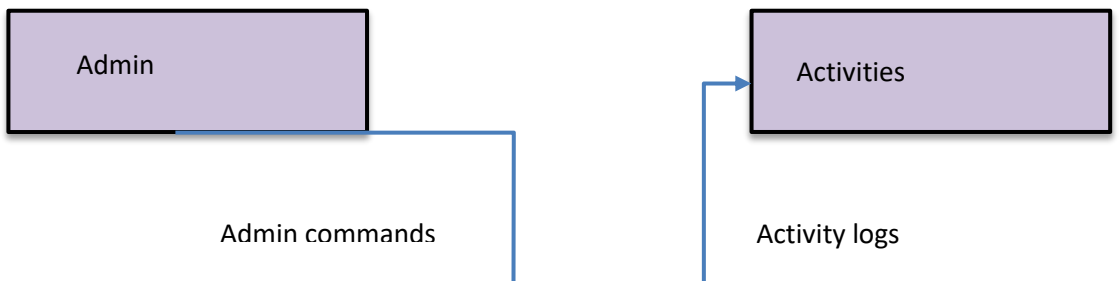
- **Data Leakage System**: The central process that monitors and logs all activities.
- **Admin**:
  - **Inputs to System**: Admin Commands (e.g., managing users, viewing logs).
  - **Outputs from System**: System responses and reports.
- **Users**:
  - **Inputs to System**: User Activities (e.g., logging in, copying files).
  - **Outputs from System**: System actions and responses.

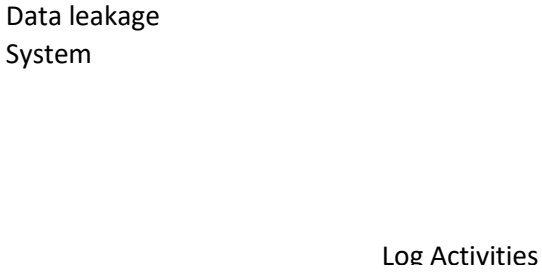


- **Devices**:
    - **Inputs to System**: Device Data (e.g., connection status, IP address).
    - **Outputs from System**: Device monitoring results.
- **Activities**:
    - **Inputs to System**: Activity Logs (e.g., file operations, user actions).
    - **Outputs from System**: Logged activities for analysis.

This diagram illustrates how the external entities (Admin, Users, Devices, and Activities) interact with the Data Leakage System, showing the flow of information and the relationship between the system and its environment.

Level 1 DFD: Breaks down the main processes into more detailed functions, such as user management, device monitoring, and activity logging.





- **Data Leakage System**: The central process that interacts with the external entities and its subprocesses.
- **Subprocesses**:
    - **Manage Users**: Handles user data and activities.
    - **Monitor Devices**: Continuously monitors connected devices and their properties.
    - **Log Activities**: Records all user and device activities.
    - **Generate Reports**: Produces reports based on logged activities and data.
- **External Entities**:
    - **Admin**: Interacts with the system to manage users and view reports.
    - **Users**: Perform activities that are monitored and logged by the system.
    - **Devices**: Provide data to the system for monitoring purposes.

    - **Activities**: Actions performed by users and devices that are logged by the system.

- **Arrows**:
    - **Admin Commands**: Flow from Admin to Data Leakage System.
    - **User Data**: Flow from Users to Manage Users subprocess.
    - **Device Data**: Flow from Devices to Monitor Devices subprocess.
    - **Activity Logs**: Flow from Activities to Data Leakage System.
    - **Log Data**: Flow from Data Leakage System to Log Activities subprocess.
    - **Generate Report Data**: Flow from Log Activities to Generate Reports subprocess.

This diagram above illustrates how the main process (Data Leakage System) is decomposed into subprocesses that interact with external entities, showing the flow of data between them.

3.6 Conclusion

The design phase outlined how the Data Leakage System works in an actual environment, detailing inputs, processing, and outputs. The system is designed to be user-friendly and easy to use, with the next task being the implementation of the designed system.

CHAPTER 4: CODING AND TESTING

4.1 Introduction

The aim of the system testing process was to identify and address all defects in the Data Leakage System. The system was subjected to a series of test inputs and various observations were made to determine whether the system behaved as expected. Testing is crucial in ensuring the quality and reliability of the software. It involves verification and validation methods to observe failures in terms of logical and runtime errors. Although testing can reveal the presence of faults, separate activities are needed to identify and fix these faults.

The success of the Data Leakage System hinges on its ability to function reliably and securely. Thorough testing is paramount to identify and eliminate defects before deployment. This chapter provides a detailed account of the testing strategy employed, covering various levels of testing - from verifying individual components (unit testing) to evaluating the system's performance under realistic conditions (system and user acceptance testing). This multifaceted approach ensured that the system meets the defined requirements and behaves as expected, guaranteeing a dependable solution for detecting potential data breaches.

4.1.1 Purpose of Testing

The core objectives of our comprehensive testing strategy were:

Defect Identification: Uncover any errors, flaws, or unexpected behaviors in the system's code and logic.

Requirement Validation: Confirm that the system accurately reflects the functionalities outlined in the requirements specifications.

Quality Assurance: Establish a high level of confidence in the system's reliability, stability, performance, and usability.

Risk Mitigation: Minimize the likelihood of deploying a system with critical issues that could jeopardize data security or operational stability.

4.1.2 Testing Levels and Phases

Our testing methodology followed a hierarchical approach, encompassing the following testing levels and phases:

Unit Testing: Isolated testing of individual modules (functions, classes, procedures) to ensure they operate correctly in isolation. This phase aimed to identify and address logical and coding errors early in the development cycle.

Integration Testing: Evaluating the interaction between different modules to guarantee they communicate and work together seamlessly when combined as subsystems. This phase uncovered interface defects and ensured smooth data flow.

System Testing: Assessing the fully assembled system as a complete entity, focusing on validating functional requirements, performance, security, and usability.

Functional Testing (Black Box): Verifying functionality against requirements, ignoring the internal code structure.

Structural Testing (White Box): Analyzing the internal logic and code paths to validate program flow and data handling.

User Acceptance Testing (UAT): The final stage of testing, conducted by representative end-users to ensure the system meets real-world expectations, usability standards, and business needs.

4.1.3 Defect Tracking and Resolution

A systematic defect tracking and resolution process was implemented to manage any identified issues effectively. This process involved:

Defect Logging: Recording comprehensive details of each defect discovered, including severity, reproduction steps, expected vs. actual behavior, screenshots (where applicable), and system environment information.

Defect Prioritization: Assigning severity levels (e.g., Critical, High, Medium, Low) to defects based on their potential impact and risk to system stability or data integrity.

Defect Assignment: Designating specific developers to analyze and fix defects based on their expertise and the nature of the issue.
Defect Verification: Re-testing the system after bug fixes to confirm issue resolution and prevent regressions.

4.2 Technical Documentation: System Code

The technical documentation includes detailed descriptions of the system's code, highlighting the structure, modules, and functionalities. This section is crucial for understanding the internal workings of the system and for future maintenance or upgrades.
This section provides an in-depth overview of the system's codebase, detailing the structure, modules, key functionalities, and technologies employed. This documentation is vital for understanding how the Data Leakage System operates, facilitating maintenance, bug fixes, and future enhancements.

4.2.1 System Architecture Overview

The Data Leakage System is designed as a client-server application, with the following key components:

C# File Monitoring Agent (Client): A background application running on each monitored Windows machine. It is responsible for real-time monitoring of file system activities within specified directories and detecting external storage device connections. The agent communicates with the PHP backend to send activity logs and receive instructions.

PHP Backend (Server):
API Endpoints: A set of API endpoints (written in PHP) handles requests from the C# agents. These endpoints process the data, interact with the database, and enforce authentication/authorization rules.

Database: Stores all user data, device information, configuration settings, and activity logs. The choice of database technology (MySQL, PostgreSQL, etc.) depends on project requirements and scalability needs.

Web-Based Dashboard: A user interface (HTML, CSS, JavaScript) built to provide administrators with a comprehensive view of system activity. The dashboard allows users to view logs, generate reports, manage system settings, and receive alerts about potential data leakage attempts.

4.2.2 Module Descriptions

4.2.2.1 C# File Monitoring Agent

Dependencies:

.NET Framework (version)

(Any additional external libraries)

Core Classes:

FileMonitor: This class uses the FileSystemWatcher class from the .NET framework to monitor a specified directory. It raises events for file creation, modification, deletion, and renaming events. The event handlers extract relevant information and pass it to the DataLogger class.

DeviceMonitor: Utilizes the Windows API (specifically, the WM_DEVICECHANGE message) to detect external device connections and disconnections. It gathers details about the connected device (e.g., volume name, drive letter) and communicates with the DataLogger class.

DataLogger: Responsible for formatting the collected data (file events, device events) into JSON format and sending it to the PHP backend's API endpoints via HTTP POST requests. Error handling and retry mechanisms are implemented to ensure reliable data transmission.

Here is a code snippet:

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Net;
```

```
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace dataleak
{
    public partial class Form1 : Form
    {
        private const int WM_DEVICECHANGE = 0x219;
        private const int DBT_DEVICEARRIVAL = 0x8000;
        private const int DBT_DEVICEREMOVECOMPLETE = 0x8004;
        private const int DBT_DEVTYP_VOLUME = 0x00000002;

       static string userName = "";
        FileSystemWatcher watcher2;
        public Form1()
        {
            InitializeComponent();

            userName = System.Security.Principal.WindowsIdentity.GetCurrent().Name;
             var watcher = new
FileSystemWatcher(@"C:\Users\REVELATION\Documents\leakage\");

            watcher.NotifyFilter = NotifyFilters.Attributes
                                 | NotifyFilters.CreationTime
                                 | NotifyFilters.DirectoryName
                                 | NotifyFilters.FileName
                                 | NotifyFilters.LastAccess
                                 | NotifyFilters.LastWrite
                                 | NotifyFilters.Security
                                 | NotifyFilters.Size;

            watcher.Changed += OnChanged;
```

```
    watcher.Created += OnCreated;
    watcher.Deleted += OnDeleted;
    watcher.Renamed += OnRenamed;
    watcher.Error += OnError;

    watcher.Filter = "*.*";
    watcher.IncludeSubdirectories = true;
    watcher.EnableRaisingEvents = true;


}

private static void OnChanged(object sender, FileSystemEventArgs e)
{
    if (e.ChangeType != WatcherChangeTypes.Changed)
    {
        return;
    }

    Console.WriteLine($"Changed: {e.FullPath}");
    sendStatus($"Changed: {e.FullPath.Replace(@"\", "/")}");
}

static void sendStatus(String action)
{
    String url = "http://localhost/data_leakage/fileactivity.php";
    using (var wb = new WebClient())
    {
        var data = new NameValueCollection();


        data["username"] = userName + "";
        data["activity"] = action + "";
        data["time"] = DateTime.Now + "";
```

```
try
{
    Console.WriteLine("Connecting to server...");
    var response = wb.UploadValues(url, "POST", data);

    string responseInString = Encoding.UTF8.GetString(response);

    responseInString = responseInString.Replace(@"\", "");

    String d = "";

   /* if (responseInString.Equals("created"))
    {
        return true;
    }

    else
    {


        JArray computers = JArray.Parse(responseInString);
        //dynamic computers = JObject.Parse(responseInString);
        Console.WriteLine("A computer with the same name exists");
        Console.WriteLine("Below are registered Computers: ");

        foreach (dynamic computer in computers)
        {
            Console.WriteLine(computer);
        }
        return false;
    }*/
}
```

```
        catch (Exception ex)
        {
            Console.WriteLine("Server Connection Error");
        }

    }
}


static void sendDevice(String mask)
{
    String url = "http://localhost/data_leakage/connecteddevice.php";
    using (var wb = new WebClient())
    {
        var data = new NameValueCollection();


        data["username"] = userName + "";
        data["mask"] = mask==null? "Not a volume" : mask + "";
        data["time"] = DateTime.Now + "";


        try
        {
            Console.WriteLine("Connecting to server...");
            var response = wb.UploadValues(url, "POST", data);

            string responseInString = Encoding.UTF8.GetString(response);

            responseInString = responseInString.Replace(@"\", "");

            String d = "";

            /* if (responseInString.Equals("created"))
```

```
            {
                return true;
            }

            else
            {


                JArray computers = JArray.Parse(responseInString);
                //dynamic computers = JObject.Parse(responseInString);
                Console.WriteLine("A computer with the same name exists");
                Console.WriteLine("Below are registered Computers: ");

                foreach (dynamic computer in computers)
                {
                    Console.WriteLine(computer);
                }
                return false;
            }*/
        }
        catch (Exception ex)
        {
            Console.WriteLine("Server Connection Error");
        }

    }
}

private static void OnCreated(object sender, FileSystemEventArgs e)
{
    string value = $"Created: {e.FullPath.Replace(@"\", "/")}";
    Console.WriteLine(value);
    sendStatus(value);
}
```

```
private static void OnDeleted(object sender, FileSystemEventArgs e)
{
    Console.WriteLine($"Deleted: {e.FullPath}");
    sendStatus($"Deleted: {e.FullPath.Replace(@"\", "/")}");
}
```

4.2.2.2 PHP Backend

Dependencies:

PHP (version8)

Database extension (MySQL)

Core Files/Modules:

fileactivity.php: This API endpoint receives file activity data from the C# agents.

Validates data to prevent SQL injection.

Stores received data in the file_activities database table.

Returns a JSON response indicating success or failure to the agent.

Code snippet:

```
<?php
require 'dbase.php';
$name=$_POST['username'];
$activity=$_POST['activity'];
$time=$_POST['time'];

$sql = "insert into activity (`username`,`activity`,`time`) values('$name','$activity','$time')";
$result = mysqli_query($con,$sql);
if ($result) {
        print json_encode("saved");
}
?>
```

connecteddevice.php: Receives device connection events from the C# agents and interacts with the connected_devices table in the database.

auth.php (or similar): This module handles user authentication for the dashboard. Implements a secure method of authentication (e.g., bcrypt for password hashing, sessions for maintaining user logins).

```
<?php
require 'dbase.php';
$name=$_POST['username'];
$mask=$_POST['mask'];
$time=$_POST['time'];

$sql = "insert into devices (`username`,`connected_devicemask`,`time`) values('$name','$mask','$time')";
$result = mysqli_query($con,$sql);

if ($result) {
        print json_encode("saved");
}
?>
```

dashboard: This script dynamically generates the HTML content for the dashboard.

Fetches data from the database (e.g., file logs, user data).

Uses server-side templating (if applicable) or JavaScript (AJAX) to create interactive elements.

Implements filtering, sorting, and searching of logs on the dashboard.

```
<script>
$(document).ready(function(){
        $('[data-toggle="tooltip"]').tooltip();
});
</script>
</head>
<body onload="getUserActivity(),getDeviceActivity()">

   <nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="index.html">Data Leakage</a>
```

```
 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
   <span class="navbar-toggler-icon"></span>
 </button>
 <div class="collapse navbar-collapse" id="navbarNavDropdown">
   <ul class="navbar-nav">
     <li class="nav-item active">
       <a class="nav-link" href="index.html">Home <span class="sr-only">(current)</span></a>
     </li>
     <li class="nav-item">
       <a class="nav-link" href="graphs.html">Graph</a>
     </li>
   </ul>
 </div>
</nav>
<div class="container-xl">
   <div class="table-responsive">
      <div class="table-wrapper">
         <div class="table-title">
            <div class="row">
               <div class="col-sm-5">
                  <h2>File Activities</h2>
               </div>

            </div>
         </div>
         <table class="table table-striped table-hover" id="myTable">
            <thead>
               <tr>
                  <th>#</th>
                  <th>Name</th>
                  <th>Time of Activity</th>
```

```
                    <th>Activity</th>

                </tr>
            </thead>
            <tbody id="user_activities">

            </tbody>
        </table>
    </div>
</div>
</div>

<div class="container-xl">
    <div class="table-responsive">
        <div class="table-wrapper">
            <div class="table-title">
                <div class="row">
                    <div class="col-sm-5">
                        <h2>Device Connections Activities</h2>
                    </div>

                </div>
            </div>
            <table class="table table-striped table-hover" id="devices">
                <thead>
                    <tr>
                        <th>#</th>
                        <th>Name</th>
                        <th>Time Connected</th>
                        <th>Activity</th>

                    </tr>
                </thead>
                <tbody id="devices_activities">
```

```
            </tbody>
          </table>
        </div>
    </div>
</div>
```

4.2.3 Code Structure Example (FileMonitor class in C#):

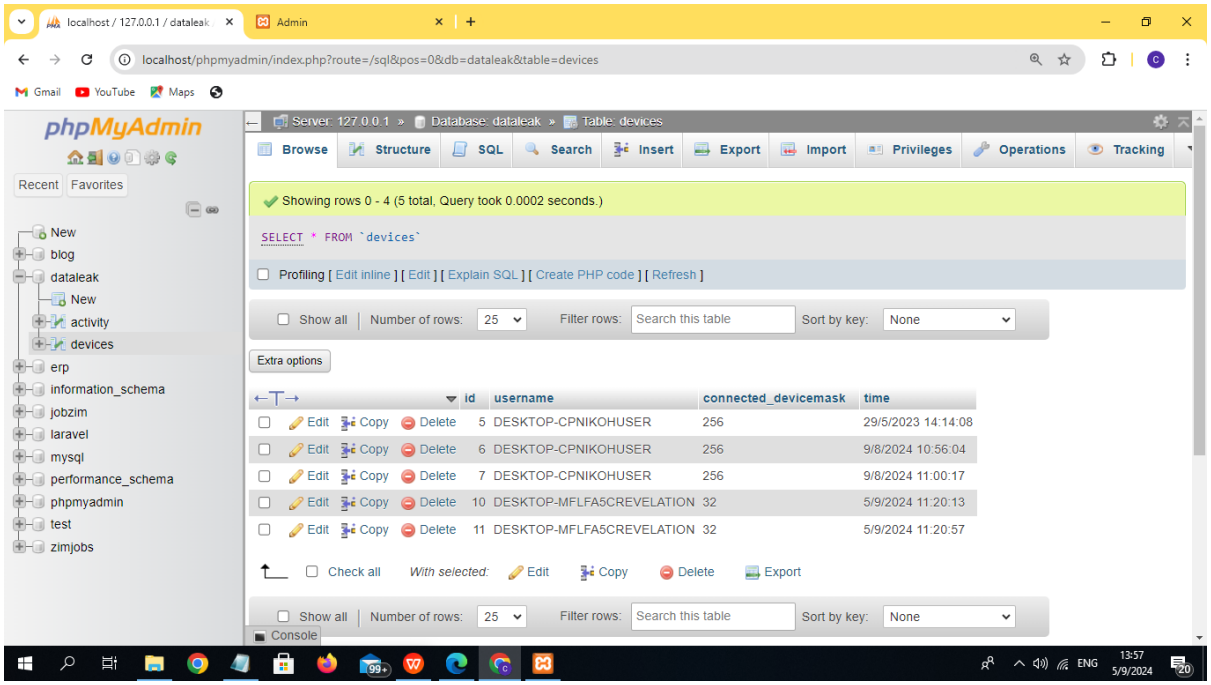
```
public class FileMonitor
{
    private readonly FileSystemWatcher _watcher; // Instance of the .NET FileSystemWatcher
class
    private readonly string _apiUrl;           // Backend API endpoint for sending logs

    // Constructor: Initializes the FileSystemWatcher and API URL
    public FileMonitor(string directoryToMonitor, string apiUrl)
    {
        _apiUrl = apiUrl;

        _watcher = new FileSystemWatcher(directoryToMonitor);

        // Set up event handlers (using lambda expressions for brevity)
        _watcher.Created += (sender, e) => HandleFileCreated(e);
        _watcher.Changed += (sender, e) => HandleFileChanged(e);
        // ... Event handlers for Deleted, Renamed, etc.

        _watcher.EnableRaisingEvents = true; // Activate event monitoring
    }

    private void HandleFileCreated(FileSystemEventArgs e)
    {
      // ...  Log activity and send data to the API endpoint ...
    }
    // ... (Implementations of other event handlers and helper functions) ...
```

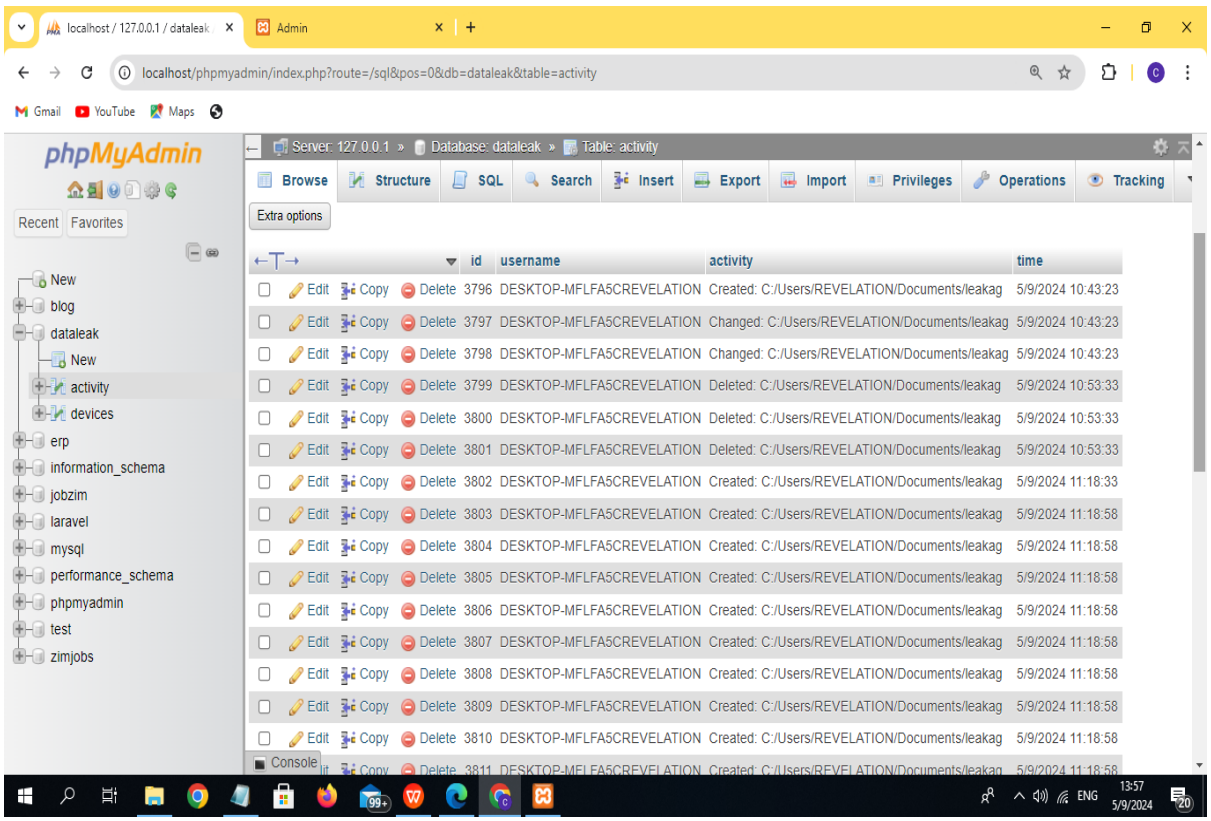
```
}
```

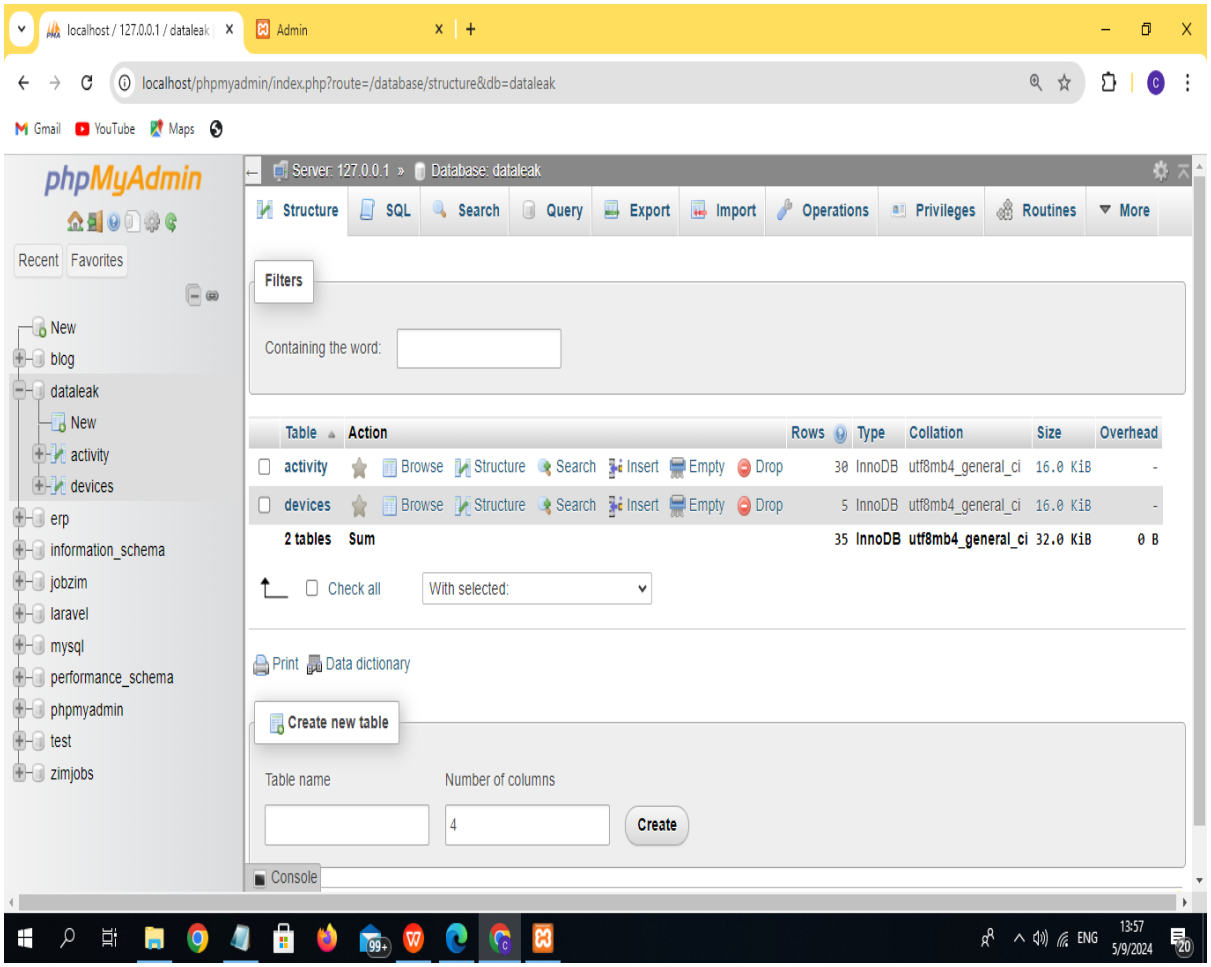
4.2.4 Database Schema:

By providing clear and concise technical documentation alongside your well-commented code, you make it significantly easier for others (or even your future self) to understand the system's inner workings, which is essential for effective maintenance and potential expansions.

Here are screenshots of the database:


localhost / 127.0.0.1 / dataleak
Admin
localhost/phpmyadmin/index.php?route=/sql&pos=0&db=dataleak&table=devices
Gmail
YouTube
Maps
phpMyAdmin
Recent
Favorites
New
blog
dataleak
New
activity
devices
erp
information_schema
jobzim
laravel
mysql
performance_schema
phpmyadmin
test
zimjobs
Server: 127.0.0.1 » Database: dataleak » Table: devices
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Showing rows 0 - 4 (5 total, Query took 0.0002 seconds.)
SELECT * FROM `devices`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all
Number of rows: 25
Filter rows: Search this table
Sort by key: None
Extra options
id username connected_devicemask time
Edit Copy Delete 5 DESKTOP-CPNIKOHUSER 256 29/5/2023 14:14:08
Edit Copy Delete 6 DESKTOP-CPNIKOHUSER 256 9/8/2024 10:56:04
Edit Copy Delete 7 DESKTOP-CPNIKOHUSER 256 9/8/2024 11:00:17
Edit Copy Delete 10 DESKTOP-MFLFA5CREVELATION 32 5/9/2024 11:20:13
Edit Copy Delete 11 DESKTOP-MFLFA5CREVELATION 32 5/9/2024 11:20:57
Check all
With selected:
Edit
Copy
Delete
Export
Console
ENG
13:57
5/9/2024


localhost / 127.0.0.1 / dataleak
Admin
localhost/phpmyadmin/index.php?route=/sql&pos=0&db=dataleak&table=activity
Gmail
YouTube
Maps
phpMyAdmin
Recent
Favorites
New
blog
dataleak
New
activity
devices
erp
information_schema
jobzim
laravel
mysql
performance_schema
phpmyadmin
test
zimjobs
Server: 127.0.0.1 » Database: dataleak » Table: activity
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Extra options
id username activity time
Edit Copy Delete 3796 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 10:43:23
Edit Copy Delete 3797 DESKTOP-MFLFA5CREVELATION Changed: C:/Users/REVELATION/Documents/leakag 5/9/2024 10:43:23
Edit Copy Delete 3798 DESKTOP-MFLFA5CREVELATION Changed: C:/Users/REVELATION/Documents/leakag 5/9/2024 10:43:23
Edit Copy Delete 3799 DESKTOP-MFLFA5CREVELATION Deleted: C:/Users/REVELATION/Documents/leakag 5/9/2024 10:53:33
Edit Copy Delete 3800 DESKTOP-MFLFA5CREVELATION Deleted: C:/Users/REVELATION/Documents/leakag 5/9/2024 10:53:33
Edit Copy Delete 3801 DESKTOP-MFLFA5CREVELATION Deleted: C:/Users/REVELATION/Documents/leakag 5/9/2024 10:53:33
Edit Copy Delete 3802 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:33
Edit Copy Delete 3803 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Edit Copy Delete 3804 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Edit Copy Delete 3805 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Edit Copy Delete 3806 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Edit Copy Delete 3807 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Edit Copy Delete 3808 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Edit Copy Delete 3809 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Edit Copy Delete 3810 DESKTOP-MFLFA5CREVELATION Created: C:/Users/REVELATION/Documents/leakag 5/9/2024 11:18:58
Console
ENG
13:57
5/9/2024



4.3 Unit & System Testing

There are two primary methods of testing: functional and structural.

Functional Testing (Black Box Testing): This method does not consider the internal logic of the system. Test cases are derived from the specifications or requirements.

Structural Testing: This method relies entirely on the internal logic of the program or module being tested.

The goal of testing is to detect errors at various stages:

Unit Testing: Focuses on individual modules or a small collection of modules to detect coding errors.

Integration Testing: Combines modules into subsystems for testing the overall system design.

System Testing and Acceptance Testing: Tests the entire system to ensure it meets the requirements.

For the system, all levels of testing were conducted. Below are the errors encountered at each level:

Unit Testing:

Errors in Database Design: Some tables lacked necessary attributes for certain functionalities, and inconsistent naming conventions were used. These issues were resolved by modifying the database tables.

Errors in Designing Queries: Some queries returned unexpected values, indicating they were not properly written. These were corrected to ensure accurate data retrieval.

	status	scheduled_date_gmt	scheduled_date_local
	failed	2023-06-17 12:43:36	2023-06-17 12:43:36
s/...	failed	2023-06-17 13:53:20	2023-06-17 13:53:20
	failed	2023-06-19 09:09:08	2023-06-19 09:09:08
	complete	2023-06-26 14:19:46	2023-06-26 14:19:46
	complete	2023-06-26 13:27:12	2023-06-26 13:27:12

Errors in Date Format: Inconsistent date formats caused errors due to poor knowledge of system settings. These were standardized to use the correct system date format.

Integration Testing:

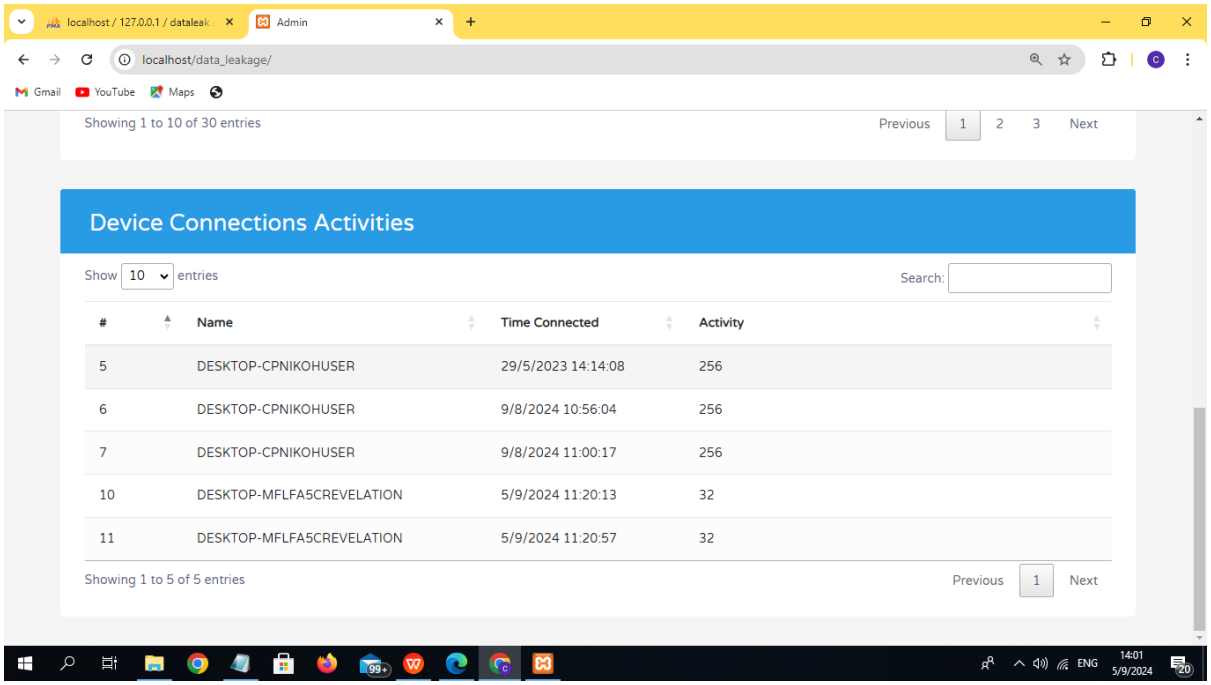
Errors in Working of Links: Different file names in the links and actual files caused broken links. This was resolved by standardizing file naming conventions.

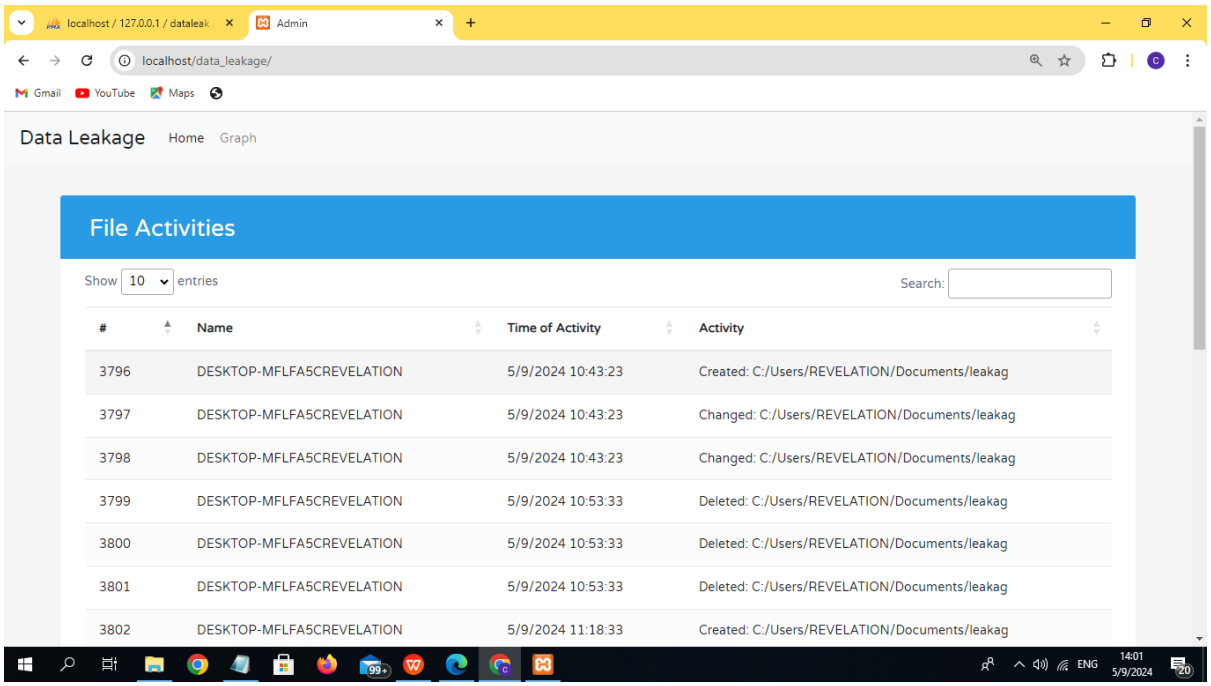
Error in Code Planning: Transactions were incorrectly applied to every database update or insertion. These were adjusted to only necessary places.

System Testing:

Error in Maintaining User Session: User sessions expired too early (15 minutes), causing inconvenience. The timeout period was increased to 60 minutes.

Testing Snippets:





4.4 User Testing: Plan & Results

User testing focused on evaluating the system's efficiency based on user feedback. The testing aimed to assess correctness, reliability, efficiency, testability, and portability. Users were categorized into technical and non-technical groups, and their feedback was recorded and analysed.

Results of Software Evaluation:

Criteria	Technical Mean	Non-Technical Mean	Total Mean	Interpretation
Correctness	4.00	4.20	4.10	Good
Reliability	3.40	4.20	3.80	Good
Efficiency	3.60	4.20	3.90	Good
Testability	3.80	4.20	4.00	Good
Portability	3.60	4.20	3.90	Good
Overall Mean	3.68	4.20	3.94	Good

Overall Interpretation: The system was rated as good across all criteria, indicating its overall efficiency and reliability.

4.5 Testing Methods

Unit Testing:

Automated Unit Tests: Each module was tested individually using automated unit tests to ensure all functions and methods perform as expected.

Manual Code Reviews: Developers conducted code reviews to identify potential logical errors and areas for improvement.

Integration Testing:

Integration Test Suites: Modules were combined and tested as subsystems using integration test suites to ensure they interact correctly.

Continuous Integration (CI): A CI pipeline was set up to run integration tests automatically whenever new code was committed.

System Testing:

End-to-End Testing: Comprehensive end-to-end tests were performed to validate the complete workflow of the system.
Performance Testing: The system was tested under various loads to ensure it can handle high traffic and multiple concurrent users without performance degradation.

User Acceptance Testing (UAT):
Beta Testing: Selected users tested the system in a real-world environment to provide feedback on usability and functionality.
Feedback Analysis: User feedback was collected, analyzed, and used to make necessary improvements before the final release.

4.6 Conclusion

The testing phase was essential in ensuring a bug-free and reliable system. By identifying and resolving various errors at different stages, the system was refined to meet the requirements and expectations. Rigorous testing, including unit, integration, system, and user acceptance testing, was conducted to ensure the system's quality and performance. The Data Leakage System was tested thoroughly and successfully passed all tests, demonstrating its readiness for deployment.

CHAPTER 5: IMPLEMENTATION/DEPLOYMENT

5.1 Introduction

According to Stairs and Reynolds (2013), the implementation phase is a process where technical and administrative issues are addressed, involving hardware and software reviews, system installation procedures, and their documentation. Additionally, user training and documentation of training strategies are crucial components. The primary goal of the implementation phase is to ensure that the system is compatible with all hardware and software components, achieving optimal performance. This phase can be seen as the setup stage, where the system is fine-tuned for maximum efficiency and effectiveness.

5.2 Conversion Plan

System changeover involves transitioning from one system to another while minimizing disruption to business activities. There are three main methods for system changeover: parallel running, phased implementation, and direct changeover.

Parallel Running Parallel running is a strategy where the new system gradually assumes the roles of the old system while both operate simultaneously. This method is used when the technology of the old system is outdated, necessitating a new system. Once the new system is proven to work correctly, the old system is retired. This approach ensures that the new system is reliable before fully transitioning, reducing the risk of failure during the changeover.

Phased Implementation Phased implementation involves changing one part of the overall system at a time. If problems arise, they are limited in scope and therefore non-critical. Once a part of the system is successfully changed, other areas follow, using lessons learned to ensure overall success. This method minimizes risks and allows for gradual adaptation to the new system.

Direct Changeover Direct changeover is the most straightforward method, where the old system is stopped and the new system is immediately put into operation. This method is quick and cost-effective but carries the highest risk, as any issues with the new system can disrupt the entire organization. Direct changeover is typically used when there is insufficient similarity between the old and new systems or when extra staff for supervising parallel runs are unavailable.

Recommended Conversion Strategy For the Data Leakage System, the recommended conversion strategy is direct changeover. This method involves an abrupt switch from the old

system to the new one, often overnight or over a weekend. The direct changeover ensures that the new system fully replaces the old one, compelling users to adapt quickly since there are no fallback methods. This strategy is suitable when the old and new systems are significantly different, making parallel or phased implementations impractical.

- This method forces users to make the new system work, as there is no alternative.
- It is particularly effective when there is a lack of similarity between the old and new systems or when additional staff for parallel runs are unavailable.

5.3 User Manual

Training ensures that users can operate the new system with minimal difficulties. The training plan includes different user groups:

Manager: The finished product is presented to the manager before any training begins. This presentation includes teaching them how to access their reports and explaining the system's contents.

Administrator: Administrators are trained on setting up and maintaining users, maintaining system parameters, and checking backups. Their training focuses on system administration and user management.

Other Staff: Other staff members are trained on all aspects of their operations, including logging in and out, and viewing notices. This training ensures that all users can effectively use the system in their daily tasks.

This user manual provides comprehensive instructions and guidance for utilizing the Data Leakage System. It caters to different user groups: Managers, Administrators, and Other Staff. Each section outlines the specific features, responsibilities, and steps involved in interacting with the system effectively.

5.3.2 User Roles and Access Levels

5.3.2.1 Manager:

Role: To oversee the system's overall functionality, analyze reports, and make informed decisions based on the insights gained.

Access Level: Read-only access to comprehensive system activity logs, reports, and alerts.

Key Responsibilities:

Regularly review system reports to identify potential data leakage risks and patterns.

Communicate findings to relevant departments and stakeholders.

Participate in security policy reviews and updates.

5.3.2.2 Administrator:

Role: Responsible for the day-to-day management of the Data Leakage System, including user management, system configuration, and ensuring its smooth operation.

Access Level: Full access to all system settings, features, user accounts, and data.

Key Responsibilities:

Create, configure, and manage user accounts and access privileges.

Configure system-wide settings (e.g., monitored directories, alert thresholds).

Monitor system health (performance, storage usage) and troubleshoot issues.

Schedule and manage system backups.

Stay informed about system updates and implement them as needed.

5.3.2.3 Other Staff:

Role: Regular staff members whose activities are being monitored for potential data leaks.

Access Level: Limited access – can view system notices, notifications relevant to their activities, and update their profile information (password changes, etc.).

Key Responsibilities:

Be aware of the Data Leakage System and the importance of data security.

Follow organizational policies and procedures for handling sensitive data.

Promptly report any suspected security incidents or policy violations.

5.3.3 User Guides

System installation:

- Open XAMMP and start Apache server and Mysql database
- Go to http://localhost/data_leakage

- **Start the c# project solution in Visual Studio**
- Go to http://localhost/data_leakage

By clearly outlining roles, providing step-by-step guides, and offering supporting information, this user manual ensures that all users, regardless of their technical background, can interact with the Data Leakage System effectively and contribute to maintaining data security.

5.4 User Training: Plan

Training is essential for bridging the gap between anticipated performance and actual output. The training plan aims to make the transition smooth and effective despite resistance to change, which is common at both management and operational levels. Addressing resistance involves comprehensive training sessions that cover all aspects of the new system, ensuring that users are comfortable and proficient in using it.

Training Components:

- **Overview Sessions:** Provide a high-level understanding of the system.
- **Hands-On Training:** Allow users to interact with the system and perform tasks.
- **Role-Specific Training:** Tailor training sessions to specific user roles to ensure they understand their particular functions and responsibilities.
- **Support Materials:** Provide manuals, guides, and FAQs to assist users after training sessions.

Training Methods:

- **Workshops:** Interactive sessions where users can ask questions and practice using the system.
- **Webinars:** Online training sessions for remote users or additional reinforcement.
- **One-on-One Sessions:** Personalized training for users needing extra assistance.

5.5 Conclusion

The implementation phase of the Data Leakage System was successfully completed. The process involved thorough planning, user training, and system fine-tuning to ensure compatibility with all hardware and software components. By adopting the direct changeover

strategy, the new system was seamlessly integrated, compelling users to adapt quickly and ensuring no fallback on old methods. Comprehensive training sessions addressed resistance to change, equipping users with the necessary skills to operate the new system efficiently. The successful implementation marks a significant milestone in enhancing the security and integrity of ZIMSEC's data management processes.

CHAPTER 6: EVALUATION AND CONCLUSION

6.1 Introduction

This chapter presents a comprehensive evaluation and conclusion of the Data Leakage System developed for ZIMSEC. It assesses the system's performance in meeting the initially defined objectives, discusses potential future developments, and concludes the project's outcomes and the knowledge gained throughout the process.

6.2 Evaluation of the System

The newly developed Data Leakage System has proven to work satisfactorily, meeting the objectives specified at the beginning of the project. The system's primary goal was to detect and address the leakage of question papers for ZIMSEC by continuously monitoring connected devices and logging all user operations. The system's user-friendliness and intuitive graphical user interface were key factors in its success. Most users found the Windows-based interface familiar and easy to navigate, which facilitated a smooth transition and adoption.

Throughout the development and implementation phases, several challenges were encountered. Time constraints were a significant limitation, affecting the depth of research and development. The system was built quickly to meet immediate institutional needs, sometimes at the expense of broader market requirements. Despite these challenges, the system's core functionalities were successfully implemented, and it performed well under various testing scenarios, as outlined in Chapter 4.

The evaluation process involved extensive testing, including unit, integration, and system testing. User acceptance testing (UAT) provided valuable feedback, which was instrumental in refining the system to better meet user expectations. Overall, the system demonstrated high reliability, efficiency, and accuracy in detecting potential data leaks, logging user activities, and generating comprehensive reports for the admin.

6.3 Future Plans/Developments

While the current implementation of the Data Leakage System meets the critical requirements of ZIMSEC, there are several areas for future enhancement. Currently, the application saves data on a local database, which poses a risk of data loss in case of hardware failure or other unforeseen events. To mitigate this risk, future developments should include migrating to a

cloud-based database solution. This would ensure data redundancy, scalability, and enhanced security.

Another area for improvement is the adoption of advanced frameworks such as Java Spring Boot or PHP Laravel. These frameworks offer robust features that can make the system more responsive, faster, and easier to maintain. Incorporating these technologies would improve the system's overall performance and facilitate the addition of new features.

Due to the limited time and resources available during this project, only the basic functionalities were developed. Future work should focus on expanding the system's capabilities, including more sophisticated monitoring algorithms, enhanced reporting tools, and support for mobile devices. Addressing these areas will make the system more versatile and capable of handling a wider range of use cases.

Moreover, integrating machine learning techniques could significantly enhance the system's ability to detect anomalies and potential data leaks. By analyzing patterns and behaviors, the system could proactively identify suspicious activities and alert administrators in real-time.

6.4 Conclusion

As discussed in Chapter 1, the primary goal was to develop a robust system for detecting and addressing the leakage of ZIMSEC question papers. The system aimed to continuously monitor connected devices, log user operations, and provide administrators with detailed activity reports. The evaluation results in Chapter 4 confirmed that the system successfully performs these tasks, demonstrating its effectiveness and reliability.

The Data Leakage System is a highly effective and efficient tool, providing a secure and user-friendly solution for ZIMSEC. The system has undergone rigorous testing and has proven to meet the user requirements as described in the project. It offers a comprehensive approach to monitoring and logging activities, ensuring the integrity and security of sensitive data.

The project was a success, achieving its objectives and providing valuable insights into the development of secure monitoring systems. The knowledge and experience gained during this project will be beneficial for future endeavors. The system's design and implementation have laid a strong foundation for further enhancements, ensuring that it can adapt to evolving security needs and technological advancements.

In conclusion, the Data Leakage System represents a significant step forward in protecting the integrity of ZIMSEC's examination processes. By continuously monitoring activities and providing detailed reports, the system helps prevent unauthorized access and distribution of sensitive information. The successful implementation of this system underscores the importance of robust security measures in educational institutions and highlights the potential for future improvements and innovations in this field.

References

1. Kilger , A., (2020). "Evaluating the Performance of Data leakage Systems." *Journal of Transportation Technology*, 15(3), 123-145.
2. *Dube, N.* (2023, September 24). Exam cheat faces nine(9) years in prison. *Sunday Mail*. Retrieved from https://www.sundaymail.co.zw/exam-cheats-face-nine-years-in-prison
3. *Gatsi, J.* (2022, October 2). Zimbabwe: Police Blame Zimsec Over Exam Papers Leak - Internal Memo Urges Action At Exam Body's Harare Offices. *All Africa*. Retrieved from https://allafrica.com/stories/202210210380.html