



SCHOOL OF BUSINESS & MANAGEMENT STUDIES

COMPUTER STUDIES DEPARTMENT

DIPLOMA IN INFORMATION TECHNOLOGY

NAME : DANNY CHANDA

STUDENT# : 23312105079

SUBJECT : PROJECT DOCUMENTATION

PROJECT TITLE : MISSING PERSONS REPORTING AND MATCHING

SUPERVISOR : MR.B. MUTALE

Table of Contents

ABSTRACTION	i
DECLARATION	ii
ACKNOWLEDGMENT	iii
INTRODUCTION	1
CHAPTER 1	1
1.1 Introduction	1
1.2 Background	1
1.3 Objectives.....	1
1.4 Problem Statement	1
1.5 Proposed System.....	2
1.6 Budget	2
1.7 Cost-Benefit Analysis.....	2
1.8 Project Plan	3
Gantt Chart:	5
1.9 Conclusion	6
LITERATURE REVIEW	7
CHAPTER 2	7
2.0 Introduction	8
2.1 Similar Systems	8
2.2 Existing Solutions vs Proposed System	14
2.2.1 Strengths of the Proposed System	11
2.2.2 Limitations Addressed	11
2.3 Challenges Faced by Existing Systems	12
REQUIREMENTS	14
CHAPTER 3	14
3.0 Introduction	15
3.1 Functional Requirements (FR).....	15
3.2 Non-Functional Requirements (NFR)	16
DESIGN SPECIFICATION	1
CHAPTER FOUR (4)	19
4.0 Introduction	20
4.1 Database Design.....	20

4.2 System Architecture	24
4.2.1 MVC Architecture.....	24
4.3 Matching Algorithm Steps	25
4.3.1 Flowchart Description	25
4.4 Data Flow Diagram (DFD).....	28
4.5 Security Architecture	30
4.6 Conclusion	31
SYSTEM TESTING	32
CHAPTER 5	32
5.0 Introduction.....	33
5.2 Test Plan.....	33
5.3 Test Report.....	34
5.4 Bug Fixes	34
5.5 Conclusion	35
SYSTEM IMPLEMENTATION	36
CHAPTER 6	36
6.1 INTRODUCTION	37
6.2 Conversion Procedures	37
6.2.1 Types of Conversion Procedures.....	37
6.3 Conversion Approaches	38
6.3.5 RECOMMENDED CONVERSION METHOD	38
CONCLUSION AND RECOMMENDATION	39
CHAPTER 7	39
7.1 Introduction	40
7.2 Recommendations	40
7.3 Final Thoughts.....	41
CHAPTER 8	42
REFERENCES	43
APPENDICES	45
CHAPTER 9	45
Appendix A: Installation Manual.....	46
Installation Steps:.....	46
TECHNICAL SUPPORT SOURCE CODE	55

CHAPTER 10	55
Appendix D: Technical Report (Source Code)	57

ABSTRACTION

The Missing Persons Reporting and Matching System is designed to streamline the process of reporting, tracking, and managing cases of missing persons. With the integration of modern technologies such as geolocation, data visualization, and automated notifications, the system enables authorized users to report missing individuals, manage submitted cases, and match reports efficiently. The platform focuses on enhancing public safety by simplifying case management while ensuring data privacy and security.

DECLARATION

I hereby declare that the Missing Persons Reporting and Matching System is my original work and has not been submitted for any diploma, degree, or any other academic award. All sources of information used have been acknowledged and cited appropriately.

Name: Chanda Danny

Signature: _____

Date : _____

Supervisor's Declaration

I hereby confirm that the preparation and development of this system were conducted under my supervision and in accordance with the academic guidelines provided by Evelyn Hone College.

Name: Mr. B.Mutale

Signature: _____

Date : _____

ACKNOWLEDGMENT

First and foremost, I thank Almighty God for granting me the strength, wisdom, and resilience throughout this journey. I extend my heartfelt gratitude to my supervisor, Mr. Mutale, for continuous guidance, valuable feedback, and unwavering support. I am also deeply grateful to my family and friends for their encouragement, patience, and understanding throughout this project.

INTRODUCTION

CHAPTER 1

1.1 Introduction

The **Missing Persons Reporting and Matching Application** aims to tackle the challenge of reporting, tracking, and matching missing and found individuals through a web-based platform. The system provides a streamlined, efficient, and centralized process for reporting missing persons, automatically matching them with found individuals based on specific attributes, and notifying concerned parties.

1.2 Background

Each year, thousands of individuals go missing globally, and the process of reuniting them with their families remains challenging. Most existing systems rely heavily on manual work, causing delays and inefficiencies in matching missing individuals with those found. The lack of an integrated, accessible system further compounds the issue, making it difficult for law enforcement agencies and concerned families to collaborate efficiently.

1.3 Objectives

The main objectives of the proposed system are as follows:

- To provide a platform for authorized users (admins and allowed users) to report missing and found persons.
- To implement an automated system for matching missing persons with those found, based on key attributes.
- To visualize the last known locations of missing individuals using map integration.
- To allow admins to approve matches and notify users of the outcomes.
- To ensure transparency and security through audit logs and encrypted data management.

1.4 Problem Statement

Current processes for matching missing persons with found individuals are time-consuming and ineffective. The reliance on manual matching increases the time required to identify potential matches, causing unnecessary delays and distress for families. Moreover, the absence of a centralized, easily accessible system means that crucial information is often fragmented across multiple platforms.

1.5 Proposed System

The **Missing Persons Reporting and Matching Application** addresses the inefficiencies of the current system by automating the process of reporting and matching missing persons. Key features include:

- A reporting system for authorized users to submit details about missing or found individuals.
- An automatic matching system that compares missing and found persons based on attributes like name, age, and gender.
- Admin approval of matches, followed by automatic notifications to users.
- Map integration for tracking the last known location of missing persons.
- Audit logging to ensure secure and transparent operations.

1.6 Budget

The projected cost of the system is outlined below:

Table 1.1: Budget table

Item	Cost (ZMK)
laptop	5,000
internet	3, 000
Printing	500
miscellaneous Cost	500
Total Estimated Cost	9, 000

1.7 Cost-Benefit Analysis

Costs:

- Development: Time and resources needed for development, testing, and deployment.
 - Hosting: Ongoing server, domain, and SSL certificate costs.
 - Training: Teaching authorized users how to utilize the system effectively.
-

Benefits:

- Increased efficiency through automated matching, reducing delays in identifying potential matches.
- Accessibility for multiple users through a centralized system.
- Enhanced transparency via audit logs, allowing for secure and trustworthy operations.
- Real-time notifications and map tracking to aid in immediate action.
- Scalability for future enhancements like machine learning and mobile applications.

1.8 Project Plan

The system will be developed following an Agile Methodology, divided into several sprints:

Below is the breakdown of the timeline:

Table 1.2: Project Plan

Sprint	Duration	Start Date	End Date
Sprint 1: Planning & Analysis	2 weeks	Week 1	Week 2
Sprint 2: Backend Development	6 weeks	Week 3	Week 8
Sprint 3: Frontend Development	4 weeks	Week 9	Week 12
Sprint 4: Matching & Maps	4 weeks	Week 13	Week 16
Sprint 5: Testing & QA	4 weeks	Week 17	Week 20
Sprint 6: Deployment & Training	2 weeks	Week 21	Week 22
Sprint 7: Maintenance	Ongoing	Week 23	Ongoing

Gantt Chart:

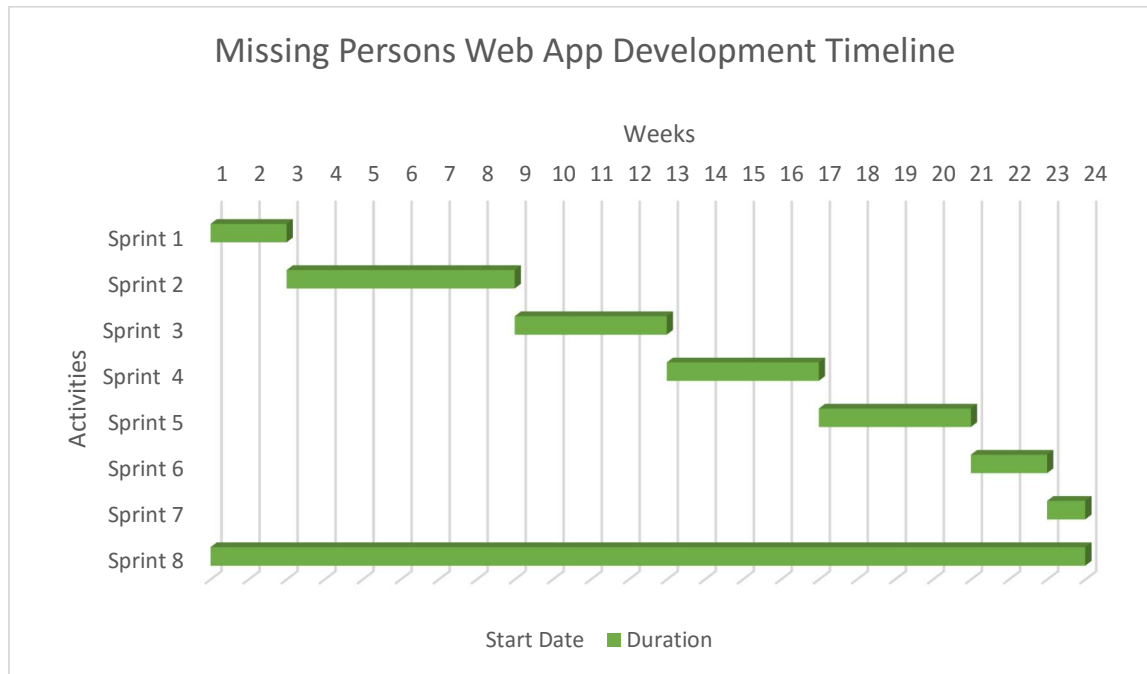


Figure 1.1

I will describe the Gantt chart layout:

- **Sprint 1:** Planning & Analysis (Weeks 1-2)
- **Sprint 2:** Backend Development (Weeks 3-8)
 - **Weeks 3-4:** Database Schema and Models
 - **Weeks 5-6:** User Authentication & Authorization
 - **Weeks 7-8:** Report Submission & Approval
- **Sprint 3:** Frontend Development (Weeks 9-12)
 - **Weeks 9-10:** UI Design & Implementation
 - **Weeks 11-12:** Frontend-Backend Integration
- **Sprint 4:** Matching & Maps (Weeks 13-16)
 - **Weeks 13-14:** Matching Algorithm Development
 - **Weeks 15-16:** Map Integration
- **Sprint 5:** Testing & QA (Weeks 17-20)
 - **Weeks 17-18:** Unit Testing

- **Weeks 19-20:** Integration & UAT
- **Sprint 6:** Fixing Identified Errors (Weeks 21-22)
- **Sprint 7:** Maintenance (Week 23 onward)
- **Sprint 8:** Documentation (Through-out)

these phases show the start and end dates for each sprint, ensuring that tasks are tracked and completed on schedule.

1.9 Conclusion

The Missing Persons Reporting and Matching Application will significantly improve the process of reuniting missing individuals with their families by automating the matching process, visualizing crucial location data, and ensuring transparency and security. The project plan is designed to be flexible yet robust, using Agile methodology to ensure timely delivery while allowing for continuous improvement and scalability.

LITERATURE REVIEW

CHAPTER 2

2.0 Introduction

In this chapter, we will review existing systems that serve similar purposes to the Missing Persons Reporting and Matching Application. By comparing these systems, we can identify their strengths, limitations, and areas for improvement, which will inform the design and development of our proposed solution.

2.1 Similar Systems

2.1.1 NamUs (National Missing and Unidentified Persons System)

Overview:

NamUs is a United States-based system funded by the Department of Justice. It provides a central platform where missing persons can be reported, and efforts can be coordinated between law enforcement agencies and families. The system also includes databases for unidentified persons and unclaimed persons.

Strengths:

- **Comprehensive Database:** NamUs integrates missing, unidentified, and unclaimed person data, providing a robust system for collaboration between law enforcement and the public.
- **DNA Matching:** The system includes DNA analysis to assist in identifying matches.
- **Public Participation:** NamUs allows the public to search the database, enhancing the chances of locating missing individuals.

Limitations:

- **Geographical Limitation:** NamUs only operates within the United States, making it inaccessible to other regions.
 - **Lack of Automated Matching:** Although it includes DNA features, much of the matching is still manual, leading to delays.
 - **Complexity:** For non-technical users, the system can be complex to navigate and use effectively.
-

2.1.2 Missing People UK

Overview:

Missing People UK is a charity-run organization that offers services to assist families in locating missing loved ones. The system allows missing persons to be reported through various channels, including phone, text, and email, and features a public-facing database of missing individuals.

Strengths:

- **Multi-Channel Reporting:** Missing People UK allows reports to be submitted through multiple means, including text, phone, and online forms, making it accessible to all demographics.
- **Emotional Support Services:** In addition to the technical tools, the system offers emotional support to families and individuals involved.
- **Public Engagement:** The database of missing individuals is publicly searchable, helping to engage more people in finding missing persons.

Limitations:

- **No Automated Matching:** The system relies on public reports and manual efforts, leading to delays in identifying and matching found individuals.
 - **No Location Tracking:** There is no integration with geographical data like last known locations, making it difficult to pinpoint possible sightings.
 - **Limited Scalability:** Being a charity-run service, the system may not have the resources to scale effectively or maintain advanced technological features.
-

2.1.3 The International Commission on Missing Persons (ICMP)

Overview:

The ICMP is a global organization that assists governments in identifying missing persons, especially in conflict and disaster scenarios. It supports nations in building institutional frameworks for locating missing individuals.

Strengths:

- **Global Reach:** ICMP operates worldwide, aiding countries in tracking missing persons.
- **DNA Identification:** It uses DNA analysis and forensic techniques to assist in locating individuals.
- **Government Collaboration:** The ICMP works closely with national authorities to ensure effective identification processes.

Limitations:

- **Specialized Focus:** Its primary focus is on conflict and disaster-related missing persons.
- **Limited Public Access:** The system is not designed for individual public reporting.

2.1.4 Interpol's Yellow Notices

Overview:

Interpol issues Yellow Notices to assist in locating missing persons and identifying individuals unable to identify themselves. It is used by police forces across its 195 member countries.

Strengths:

- International Scope: Interpol's Yellow Notices are shared globally.
- Law Enforcement Integration: The system connects directly with police databases worldwide.
- Recognition: The Interpol brand increases visibility and legitimacy.

Limitations:

- Restricted Use: Only law enforcement agencies can issue and access Yellow Notices.
- Not Publicly Searchable: It is not accessible to the general public.

2.1.5 Amber Alert System

Overview:

Amber Alert is an emergency response system used primarily in North America to broadcast information about child abductions through radio, television, and mobile alerts.

Strengths:

- Rapid Public Alerts: Amber Alerts provide immediate notifications to the public.
- Nationwide Coverage: Alerts reach millions of people instantly.
- High Success Rate: Amber Alerts have successfully located many missing children.

Limitations:

- Limited to Child Abductions: The system is only activated for child-related emergencies.
- Region-Specific: Amber Alerts primarily operate in North America.

2.2 Existing Solutions vs Proposed System

2.2.1 Strengths of the Proposed System

The Missing Persons Reporting and Matching Application improves upon the limitations of existing systems by integrating several advanced features:

- **Automated Matching:** Unlike NamUs and Missing People UK, our system will use algorithms to match missing and found persons based on various attributes such as name, age, and gender. This feature reduces manual effort and expedites the process.
- **Geographical Tracking:** The proposed system will integrate with Google Maps, allowing the last known location of missing persons to be visualized on a map. This feature is absent in both NamUs and Missing People UK.
- **Role-Based Access:** The system ensures that specific tasks (like approving matches or reporting missing persons) are restricted to authorized users (admins and allowed users), providing a secure workflow for managing cases.

2.2.2 Limitations Addressed

- **Simplified User Interface:** The proposed application will focus on a user-friendly interface that reduces complexity, making it accessible to non-technical users.

2.3 Challenges Faced by Existing Systems

While existing systems like NamUs and Missing People UK have been effective in assisting with missing persons' cases, they are not without their challenges. These include:

- **Slow Matching Process:** The lack of automated tools for matching missing and found individuals often delays the process, causing distress to families.
 - **Limited Integration with Law Enforcement:** Many systems do not integrate seamlessly with law enforcement databases, limiting the scope of data available to identify matches.
 - **High Dependence on Public Engagement:** Systems like Missing People UK rely heavily on public engagement for reporting and finding missing persons, which may not always be reliable.
-

2.4 Conclusion

The review of these existing systems demonstrates the need for more advanced features such as automated matching, map integration, and better scalability. The Missing Persons Reporting and Matching Application seeks to address these challenges by offering a centralized, automated, and secure platform that provides quicker, more reliable results in reuniting missing individuals with their families. Moreover, the system is designed with scalability and adaptability in mind, ensuring it can grow and improve over time.

References:

- NamUs: <https://www.namus.gov>
 - Missing People UK: <https://www.missingpeople.org.uk>
 - ICMP: <https://www.icmp.int>
 - Interpol Yellow Notices: <https://www.interpol.int>
 - Amber Alert: <https://amberalert.ojp.gov>
-

REQUIREMENTS SPECIFICATION

CHAPTER 3

3.0 Introduction

In this chapter, we will outline the functional and non-functional requirements of the **Missing Persons Reporting and Matching Application**, as well as the success criteria that will determine whether the system meets its objectives. We will also discuss the fact-finding methods used to gather information on user needs and system expectations.

3.1 Functional Requirements (FR)

Functional requirements define the core operations and features of the system. They specify how the system should behave and what services it should provide to its users. Below is a breakdown of the essential functional requirements:

3.1.1 User Authentication and Authorization

- **FR1:** The system must provide secure login functionality for two types of authorized users: Admin and Allowed User.
- **FR2:** Unauthorized users must be able to view a public homepage listing all reported missing persons but should not have access to any reporting or management functions.
- **FR3:** Admins must be able to manage user accounts, including adding, editing, and deleting user accounts.

3.1.2 Reporting Missing and Found Persons

- **FR4:** Allowed users must be able to submit missing and found person reports, including details such as name, age, gender, physical description, date last seen, and last known location.
 - **FR5:** The system must allow the uploading of images for each report to provide visual identification.
 - **FR6:** Admins must review and approve all submitted reports before they are published on the platform.
-

- **FR7:** Both admins and allowed users should be able to update and manage reports they have submitted.

3.1.3 Automated Matching

- **FR8:** The system must automatically compare newly reported missing persons with existing found persons using key attributes like name, age, and gender, and flag potential matches for admin review.
- **FR9:** Admins must be able to approve or reject suggested matches between missing and found persons.

3.1.4 Notifications and Alerts

- **FR10:** The system must send email notifications to the users who submitted reports whenever their reported case is updated (e.g., approved, found, or matched).

3.1.5 Map Integration

- **FR11:** The system must integrate with Google Maps to allow users to report and view the last known location of missing persons.

3.1.6 Audit Logging and Data Security

- **FR14:** The system must log all significant actions (e.g., report submissions, approvals, matches) in an audit log to ensure transparency and accountability.
- **FR15:** The system must use encryption for sensitive data (e.g., passwords, personal information).

3.2 Non-Functional Requirements (NFR)

Non-functional requirements define the quality attributes, performance metrics, and security aspects of the system. These requirements ensure that the system functions efficiently and securely while delivering a positive user experience.

3.2.1 Usability

- **NFR1:** The system's user interface must be intuitive and easy to navigate for both technical and non-technical users.
- **NFR2:** The system should be responsive and accessible from desktop, tablet, and mobile devices.

3.2.2 Performance

- **NFR3:** The system must be able to handle multiple concurrent users without significant performance degradation.
- **NFR4:** The system must load the homepage listing of missing persons within 3 seconds under typical load conditions.

3.2.3 Scalability

- **NFR5:** The system must be able to scale to accommodate an increasing number of users and reports without affecting system performance.
- **NFR6:** The database must be designed to handle a large volume of reports and media (images) over time.

3.2.4 Security

- **NFR7:** The system must implement role-based access control (RBAC) to ensure that only authorized users can access sensitive functions.
- **NFR8:** The system must follow best practices for password storage (e.g., using hashing algorithms like bcrypt).

3.2.5 Minimum Software Requirements

- 1. Operating System: Windows 7,8,10, macOS, or a compatible Linux distribution
- 2. XAMPP: Version 7.4 or higher, which includes:
 - - Apache Server: To run the local server environment
 - - MySQL: For database management and transaction storage
 - - PHP: Version 7.4 or higher for backend functionality
- 3. Browser: Latest version of Chrome, Firefox, Safari, or Edge
- 4. Text Editor/IDE: (For development and maintenance) VS Code(recommended).

3.2.6 Minimum Hardware Requirements

- 1. Processor: Dual-core processor (1.5 GHz or higher)
 - 2. RAM: Minimum 1 GB (4 GB recommended for optimal performance)
-

- 3. Storage: 2 GB free disk space for the XAMPP installation, database, and project files
- 4. Network: Localhost access, internet connection (for dependencies CDN links and email automation).

3.4 Fact-Finding Methods

To gather the necessary information for defining the system's requirements, we employed the following fact-finding methods:

3.4.1 Interviews

We conducted interviews with law enforcement officials, families of missing persons, and representatives from organizations involved in missing persons' cases to understand their needs and the pain points with existing systems.

3.4.2 Document Analysis

We reviewed existing systems like **NamUs** and **Missing People UK** to understand their features, limitations, and areas of improvement. This analysis provided valuable insights into how our system could be designed to meet user needs more effectively.

3.4.3 Surveys

Surveys were distributed to potential users, including families, law enforcement, and social workers, to gather information on their expectations, preferred features, and concerns regarding privacy and security.

3.4.4 Observation

We observed the processes used by law enforcement agencies to handle missing persons' cases, which informed the workflows and features implemented in our system (e.g., automated matching and audit logging).

3.5 Conclusion

The functional and non-functional requirements detailed in this chapter are designed to ensure that the **Missing Persons Reporting and Matching Application** addresses the needs of users while adhering to industry standards for security, usability, and performance. By following a structured fact-finding approach, we ensured that the system is tailored to meet the objectives laid out in the project proposal.

DESIGN SPECIFICATIONS

CHAPTER FOUR (4)

4.0 Introduction

In this chapter, we will outline the architectural design, database structure, use case diagram and data flow diagram for the **Missing Persons Reporting and Matching Application**. The design specification provides an overview of how the system's components will work together to meet the functional and non-functional requirements defined in Chapter 3.

4.1 Database Design

The database is a critical part of the system, as it stores all reports, user information, and logs. The relational database schema will consist of multiple tables to store and relate various data entities.

4.1.1 ERD (Entity-Relationship Diagram)

The following tables will be included in the database:

1. **Table 4.1:**Users

attribute	datatype	comment
id (PK)	Int(11)	Unique identifier for each user.
username	Varchar(50)	User's full name.
email	Varchar(100)	User's email address.
password	Varchar(255)	Encrypted user password.
role	Enum(admin, allowed user)	Defines the user's role (Admin or Allowed User).
Created_at	Timestamp	The date the user was registered
Image	Varchar(500)	The profile picture path of the user
Code	Int(5)	the current code of the user

Status	Enum(Not verified, verified, Suspended)	The states of the user (e.g. verified , suspended, not verified)
--------	---	--

2. Table 4.2:Reports

Attribute	Datatype	Comment
id (PK)	Int(11)	Unique identifier for each report
Type	Enum(missing, found)	the type of report submitted (e.g. missing, found).
user_id (FK)	Varchar(255)	ID of the user who submitted the report.
name	Varchar(100)	Name of the missing/found person.
age	Int(11)	Age of the person.
gender	Enum(male, female, other)	Gender of the person.
Last_seen	Date	The date the person was last seen
description	text	Physical description of the person.
Image_path	Varchar(255)	Path to the uploaded image.
Contact_info	Varchar(255)	The contact information of the reporter.
Status	Enum(pending,approved,rejected)	The status of the report (e.g. pending approved, rejected)
created_at	timestamp	Date and time the report was submitted.

Location	Vrchar(255)	Name of the location the person was last seen.
Latitude	Decimal(18,16)	The geolocation latitude (retrieved from google maps).
Longitude	Decimal(18,16)	The geolocation longitude (retrieved from google maps)

3. Table 4.3: Audit Logs:

Attribute	Datatype	Comment
id (PK)	Int(15)	Unique identifier for each log entry.
user_id (FK)	Int(11)	ID of the user who performed the action.
action	Varchar(255)	Description of the action (e.g., report created, match approved).
Details	Varchar(255)	Details of the actions including report id and status
timestamp	timestamp	Date and time the action occurred.

4.1.2 Database Relationships

- One-to-many relationship between **Users** and **Reports**: A user can submit multiple reports.
 - One-to-many relationship between **Users** and **Audit Logs**: Many user action can be logged.
-

Entity Relationship Diagram

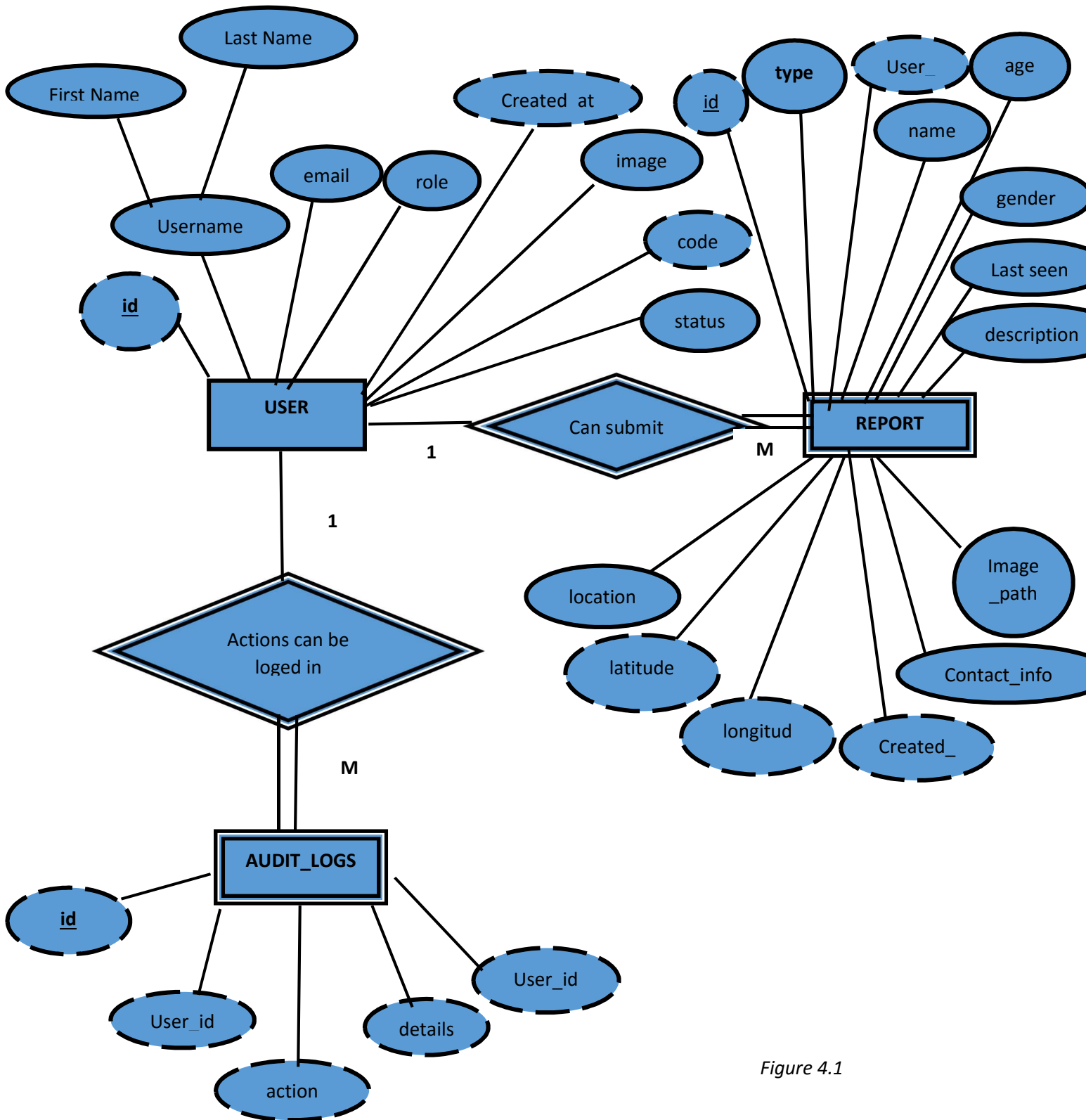


Figure 4.1

4.2 System Architecture

The **Missing Persons Reporting and Matching Application** is designed as a web-based platform using a client-server architecture. The system consists of three main layers:

1. **Presentation Layer** (Frontend)
 - Provides the user interface for users to interact with the system.
 - Built using HTML, CSS (Bootstrap), JavaScript (jQuery/AJAX), and Google Maps API integration.
2. **Business Logic Layer** (Backend)
 - Handles all core application logic, including authentication, report management, and matching algorithms.
 - Developed in PHP with Object-Oriented Programming (OOP) principles.
3. **Data Layer** (Database)
 - Manages all data storage and retrieval operations.
 - Uses MySQL to store information on users, missing persons, found persons, and logs.

The system follows the **Model-View-Controller (MVC)** design pattern, ensuring separation of concerns, scalability, and ease of maintenance.

4.2.1 MVC Architecture

- **Model:** Represents the data and the business logic. In this case, models include User, Report, security, and audit logs. These models communicate with the database and manage data-related operations.
 - **View:** Handles the presentation logic. HTML and Bootstrap will be used to render web pages, while JavaScript will provide interactivity and AJAX will handle dynamic content updates without refreshing the entire page.
 - **Controller:** Manages interactions between the Model and View. For example, the Report Controller handles submission, validation, and approval workflows for missing persons reports, User Controller handles user data such as registration, login, check status, code generation and password updating.
-

4.3 Matching Algorithm Steps

1. **Input Data:**
 - Missing persons list (missingPersons).
 - Found persons list (foundPersons).
2. **For Each Missing Person:**
 - Iterate through each entry in the foundPersons list.
 - Compare the following attributes:
 - **Name:** Must match exactly (case-insensitive).
 - **Age:** Must be identical.
 - **Gender:** Must be identical.
3. **If All Conditions Match:**
 - Flag the pair as a potential match.
 - Add the match to the list of flagged results.
4. **Output:**
 - Return all flagged matches for review.

4.3.1 Flowchart Description

1. **Start:**
 - Load missing persons and found persons' data.
 2. **Iterate Through Missing Persons:**
 - For each missing person, compare with every found person.
 3. **Compare Attributes:**
 - Name: Case-insensitive, exact match.
 - Age: Exact match.
 - Gender: Case-insensitive, exact match.
 4. **Flag Match:**
 - If all attributes match, add the pair to the flagged results.
 5. **End:**
 - Return flagged matches for review.
-

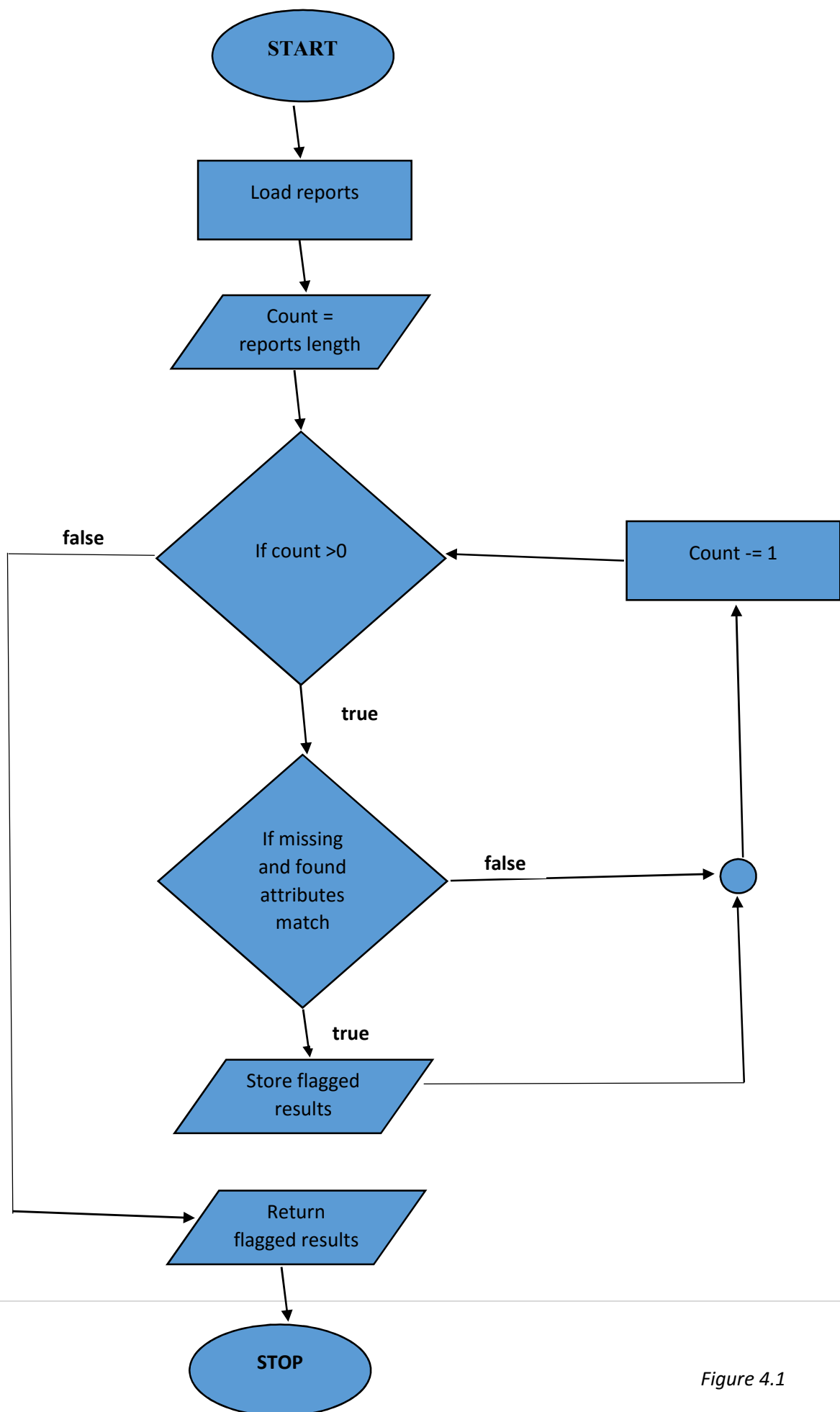


Figure 4.1

Use Case Diagram

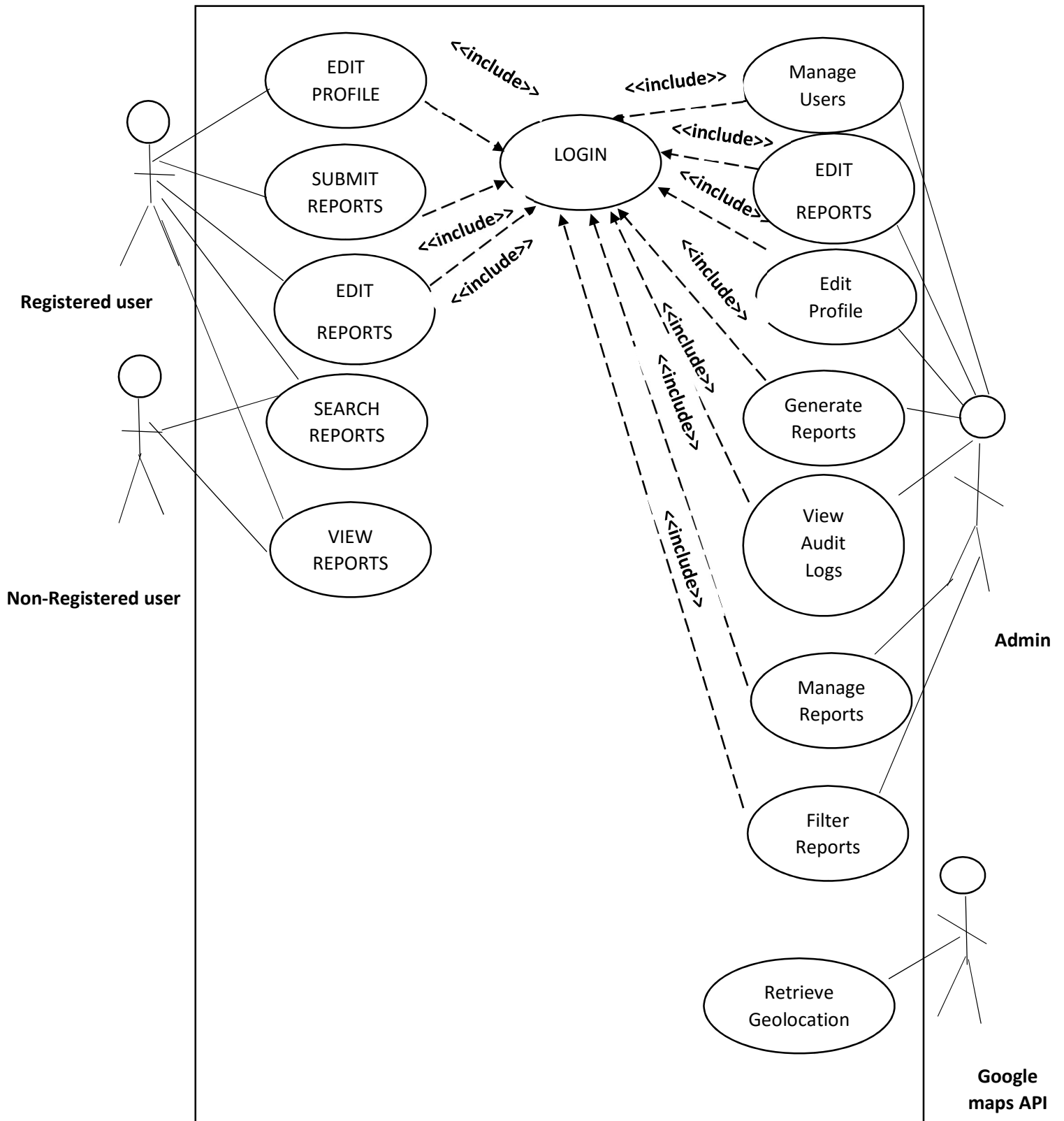


Figure 4.3

4.4 Data Flow Diagram (DFD)

The **Data Flow Diagram (DFD)** depicts how data moves through the system. Below is the description of the main processes.

4.4.1 Level 0 – Context Diagram

The **Level 0 DFD** represents the overall system and its interaction with external entities:

- **User:** Submits missing/found reports, views reports, and receives notifications.
- **Admin:** Manages users, reviews and approves reports and matches.
- **System:** Processes reports, flags potential matches, sends notifications, and logs actions.

4.4.2 Level 1 – Process Breakdown

At **Level 1**, the following processes occur:

1. **Login and Authentication:** User credentials are validated, and role-based access control (RBAC) is applied.
 2. **Report Submission:** Allowed users submit reports; the system stores the details and triggers the matching algorithm.
 3. **Matching Algorithm:** The system compares new reports with existing ones and flags potential matches for admin review.
 4. **Approval Workflow:** Admins review and approve or reject reports and matches.
 5. **Notification System:** Notifications are sent to relevant users about report status updates.
-

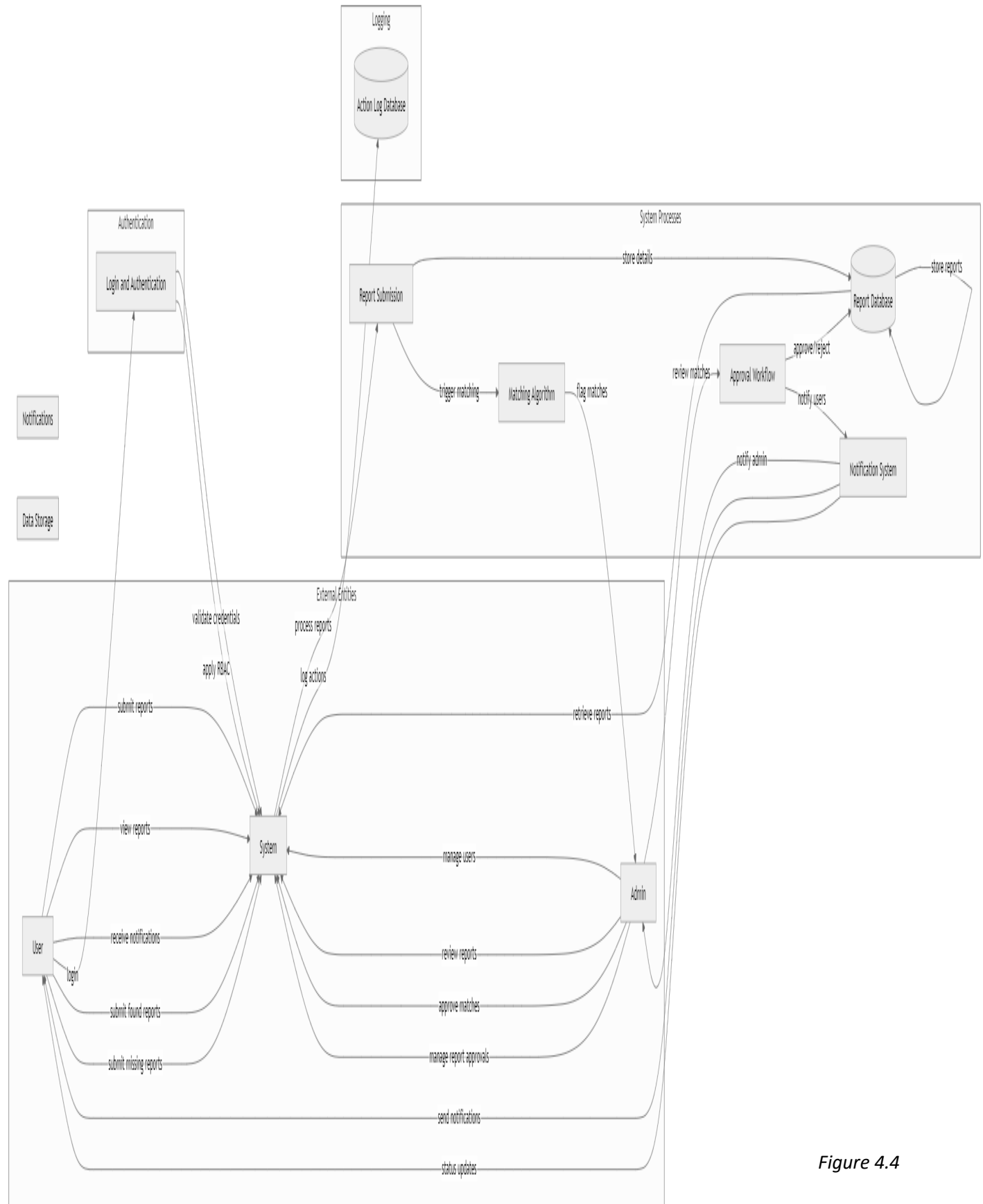


Figure 4.4

4.5 Security Architecture

Security is a critical component of the design. The application follows industry-standard security practices to protect user data and ensure that only authorized actions are performed.

4.5.1 Authentication and Authorization

- **Secure Login:** The system uses encrypted passwords (bcrypt hashing) and two-factor authentication for accounts registration and forget password requests.
- **Role-Based Access Control (RBAC):** Only authorized users (admin and allowed users) have access to sensitive functions like submitting reports or approving matches.

4.5.2 Data Encryption

- **Data Encryption:** All sensitive information, such as user passwords and personal details, is encrypted in the database.
- **SSL/TLS:** Communication between the user and the server is encrypted using SSL/TLS to protect data in transit.

4.5.3 Audit Logging

- **Audit Logs:** All significant actions are logged, including report submissions, matches, approvals, and user actions, ensuring full transparency.

4.5.4 Cross-Site Scripting (XSS) Prevention

To mitigate the risk of Cross-Site Scripting (XSS) attacks, the following security measures are implemented:

- **Input Validation and Sanitization:** All user inputs (such as report descriptions, user profiles, and other text fields) are sanitized to remove harmful HTML or JavaScript code before storing in the database or displaying on web pages.
 - **Output Encoding:** All dynamic content rendered on web pages is encoded using proper escaping methods (e.g., htmlspecialchars() in PHP) to prevent script injection.
 - **Content Security Policy (CSP):** A strict Content Security Policy (CSP) is applied to prevent the execution of unauthorized scripts on the website.
-

4.5.6 SQL Injection Prevention

To prevent SQL injection attacks, which can compromise the database by injecting malicious SQL queries, the following strategies are applied:

- **Prepared Statements with Parameterized Queries:** The application uses prepared statements with parameterized queries for all database interactions. This ensures that user input is treated as data rather than executable code, preventing SQL injection attacks.
- **Input Validation:** All input fields are validated before being passed into SQL queries, ensuring that the values conform to the expected format (e.g., numbers for IDs, text for names).
- **Database Access Controls:** The database is configured with limited access privileges for different user roles to restrict the scope of potential damage from an SQL injection attack.

4.5.7 Email Verification for User Registration

- To ensure that only valid users access the system, a verification code is sent to the user's provided email during registration. The user must input this code to complete the registration process.

4.5.8 Password Recovery via Email

- When a user forgets their password, a password recovery feature sends a verification code to their registered email, allowing them to reset their password securely

4.6 Conclusion

The design specifications for the **Missing Persons Reporting and Matching Application** provide a detailed framework for the system's architecture, database structure. These design elements ensure the application meets both functional and non-functional requirements while providing a secure, user-friendly platform for reporting and matching missing persons.

SYSTEM TESTING

CHAPTER 5

5.0 Introduction

This chapter outlines the various tests performed to ensure the system functions as expected, identifying any bugs or issues along the way. Testing involves the use of **test data**, **test plans**, and the **test reports** that document the results of the testing process.

5.1 Test Data

Test data is created to mimic real-world scenarios, ensuring that the system behaves as expected under different conditions. This includes data for missing persons, users, reports, and location information.

Test Data Examples:

- Missing persons with different demographic data (e.g., age, gender, last seen location).
- Users with varying roles (admin, allowed user).
- Reports submitted with varying details (resolved, unresolved, etc.).
- Location data to test Google Maps integration.

5.2 Test Plan

The **test plan** lays out the types of testing done, including unit, integration, and user acceptance testing. It also covers who performed the testing and how the results were recorded.

Table 5.1: Test plan

Test Case ID	Description	Test Input	Expected Output	Actual Output	Status
TC-01	User Registration with email verification	Register new user	Verification email sent successfully	Verification email sent	Pass
TC-02	Report missing person (allowed user)	Submit missing person report	Report successfully created	Report created	Pass

Test Case ID	Description	Test Input	Expected Output	Actual Output	Status
TC-03	Admin approving match between missing and found persons	Admin approves match	Person marked as found, email notification sent	Person marked as found, email sent	Pass
TC-04	SQL Injection Prevention	Attempt to inject malicious SQL	SQL query blocked	SQL query blocked	Pass
TC-05	Google Maps integration	Submit report with location data	Location displayed correctly on map	Location displayed correctly	Pass
TC-06	Cross-Site Scripting (XSS) Prevention	Attempt to insert XSS script	Script blocked and sanitized	Script blocked	Pass

5.3 Test Report

The **test report** is a record of the outcomes of the tests conducted. For each test case, results were compared with expected outputs to identify if the system functioned correctly.

- **Unit Testing:** Focuses on testing individual components of the application, such as the matching algorithm, report submission, and email verification system.
 - Outcome: Each unit functioned as expected during isolated tests.
- **Integration Testing:** Verifies that different components of the system work well together (e.g., frontend and backend communication, Google Maps API integration).
 - Outcome: All integrated parts of the system communicated effectively with each other.
- **User Acceptance Testing (UAT):** Final testing stage performed with end-users, including admins and allowed users, to ensure the system met their needs.
 - Outcome: Feedback was positive, and minor issues were addressed based on user suggestions.

5.4 Bug Fixes

Throughout the testing process, several bugs and issues were discovered, including:

1. **Issue:** Error in matching algorithm when handling multiple reports with similar names.
 - **Fix:** Added additional checks for gender and age group to improve accuracy.
2. **Issue:** Google Maps API occasionally failed to display the correct location.
 - **Fix:** Resolved by ensuring that location data passed to the API was properly formatted.
3. **Issue:** Password recovery feature was not sending emails.
 - **Fix:** Configured email server settings correctly to ensure that emails were being sent out for both registration verification and password recovery.

5.5 Conclusion

After extensive testing, the Missing Persons Reporting and Matching Application has been confirmed to meet the project's functional and non-functional requirements. Identified bugs were resolved, and the system performs optimally under various test scenarios.

SYSTEM IMPLEMENTATION

CHAPTER 6

6.1 INTRODUCTION

System implementation is the process of deploying a new system within an organization or transitioning from an existing system to a new one. It involves integrating hardware, software, and data to ensure the system operates as intended. Various methods can be used to transition from an old system to a new one, including direct conversion, parallel conversion, phased conversion, and pilot conversion.

6.2 Conversion Procedures

Conversion procedures refer to the systematic processes involved in changing data, files, or system components from one format, structure, or environment to another. This is typically required when migrating from an old system to a new one, upgrading software, or standardizing data formats.

6.2.1 Types of Conversion Procedures

1. Data Conversion:

- Transforming data from one format or structure to another.
- Example: Converting CSV files into a database-compatible format.

2. System Conversion:

- Switching from one system or platform to another while ensuring minimal downtime and data integrity.
- Example: Migrating from a legacy ERP system to a modern cloud-based solution.

3. Software Conversion:

- Updating or changing software applications to maintain compatibility or improve functionality.
- Example: Upgrading from an older version of a content management system.

4. File Conversion:

- Changing file formats to ensure compatibility with various software.
 - Example: Converting a DOC file to PDF for better distribution.
-

6.3 Conversion Approaches

6.3.1 DIRECT CONVERSION

This method involves an immediate and complete switch from the old system to the new system. The old system is stopped entirely, and the new system takes over operations at a specific time. It is cost-effective since only one system operates at a time; however, it is risky, as any errors in the new system can disrupt operations.

6.3.2 PARALLEL CONVERSION

In parallel conversion, both the old and new systems operate simultaneously for a specific period. This allows the organization to compare outputs and ensure the new system works correctly before fully replacing the old one.

6.3.3 PHASED CONVERSION

This approach involves implementing the new system in stages or modules. Parts of the new system are introduced gradually while other parts of the old system remain in use. It can help overcome user resistance by allowing gradual user adaptation and testing.

6.3.4 PILOT CONVERSION

Pilot conversion starts by implementing the new system in a small, controlled environment, such as a single department or branch. Once proven successful, the system is rolled out to the rest of the organization. This method provides a testing ground to identify issues before full implementation.

6.3.5 RECOMMENDED CONVERSION METHOD

The implementation of the "Missing Persons Reporting and Matching System" was carried out using a parallel conversion method. This approach involved operating both the new digital system and the existing manual system simultaneously for a set period. This parallel operation ensured thorough testing and validation of the platform's performance. During this phase, the efficiency and accuracy of the new system were evaluated and cross-verified with the outputs from the manual process. Once the system's reliability and functionality were confirmed to meet expectations, it was fully integrated. The new system enhances the process of reporting, tracking, and matching missing persons while ensuring operational continuity.

CONCLUSION AND RECOMMENDATION

CHAPTER 7

7.1 Introduction

The **Missing Persons Reporting and Matching Application** was developed with the goal of addressing the critical need for an efficient system to report, track, and match missing persons. By leveraging a web-based platform that integrates automated matching algorithms and Google Maps, this application has the potential to significantly reduce the time and effort involved in locating missing individuals. The system was designed with an emphasis on user-friendliness, security, and scalability, ensuring that it can be effectively used by a variety of stakeholders, including admins, allowed users, and the general public.

The development of this system included the implementation of key features such as:

- **Reporting and management of missing and found persons** by authorized users.
- **Automated matching algorithms** to flag potential matches based on key attributes.
- **Email verification** to ensure that only legitimate users can access critical system features.
- **Security measures**, including input validation, SQL injection protection, and secure login mechanisms.
- **Google Maps integration** for tracking and visualizing the last known location of missing persons.

The application has demonstrated its ability to streamline the process of reporting missing persons, providing significant improvements over manual methods. The project was completed on schedule and within the budget, with all essential features functioning as expected.

7.2 Recommendations

While the system meets the requirements specified during the planning phase, several recommendations can be made for future enhancements:

1. **Mobile Application Development:** To improve accessibility, it is recommended to develop a mobile application for the system. This will allow users to easily submit reports and track missing persons using their mobile devices. The mobile platform could offer push notifications, improving response times.
 2. **Machine Learning for Enhanced Matching:** The accuracy of the matching algorithm could be further improved by integrating **machine learning**. Specifically, **facial recognition technology** could be implemented to analyze and match images submitted with reports, making it easier to identify missing persons across regions.
-

3. **Collaboration with Law Enforcement:** Integrating the application with law enforcement databases would allow the system to become more effective. By enabling law enforcement agencies to report missing persons and search for matches, the system can have a wider reach and more comprehensive data, leading to more successful outcomes.
4. **SMS Integration:** In addition to email notifications, SMS notifications could be added to alert users of potential matches or updates on their reported cases. This feature would provide real-time communication and ensure that users are notified promptly.
5. **Enhanced Data Privacy and Security:** While the system already has basic security measures, further enhancements could include **encryption for sensitive data** stored in the database, as well as advanced access control policies. These measures will help ensure the confidentiality and integrity of user data, particularly in larger-scale deployments.
6. **Expansion to Include Other Categories of Missing Persons:** The system could be expanded to include categories such as missing children, elderly persons, or individuals with specific medical conditions. This would allow more targeted searches and reports based on the unique needs of each group.
7. **Localization and Multi-Language Support:** Adding **multi-language support** could expand the system's usability across different regions. This would be particularly useful in cases where the platform is deployed internationally, helping break down language barriers in reporting missing persons.

7.3 Final Thoughts

The **Missing Persons Reporting and Matching Application** has the potential to make a meaningful impact in addressing the global issue of missing persons. By providing a centralized platform that automates the matching process, the system can help reunite families more quickly and effectively. The development of this project has laid the groundwork for a scalable and secure system that can evolve to meet the needs of future users.

The recommendations provided offer valuable insights into how the system can be improved and adapted for broader use, ensuring it remains relevant and effective in solving the problem of missing persons. By continuing to develop and expand on the features already implemented, this application can become a vital tool for law enforcement, communities, and families worldwide.

REFERENCES

CHAPTER 8

REFERENCES

1. **Boiko, B. (2002).** *Content Management Bible*. Indianapolis: Wiley Publishing, Inc.
 - This book provides in-depth insights into the development and management of content-based systems, which is critical for the efficient management of large-scale applications like the Missing Persons Reporting and Matching System.
 2. **Pressman, R. S. (2014).** *Software Engineering: A Practitioner's Approach* (8th Edition). New York: McGraw-Hill Education.
 - This resource outlines modern software development practices and methodologies, particularly Agile, which was used in this project for iterative development and continuous improvement.
 3. **Google Developers.** (n.d.). *Google Maps API Documentation*. Retrieved from <https://developers.google.com/maps/documentation>
 - This online documentation was crucial in integrating Google Maps API for location tracking and visualization within the system.
 4. **OWASP Foundation.** (n.d.). *Cross-Site Scripting (XSS)*. Retrieved from <https://owasp.org/www-community/attacks/xss/>
 - This resource provided guidance on securing the system from cross-site scripting attacks, which was crucial to ensuring that the platform remains secure for users.
 5. **W3Schools.** (n.d.). *SQL Injection*. Retrieved from https://www.w3schools.com/sql/sql_injection.asp
 - W3Schools' explanation of SQL injection vulnerabilities and prevention techniques informed the SQL injection mitigation strategies implemented in the project.
 6. **Agile Alliance.** (n.d.). *What is Agile?* Retrieved from <https://www.agilealliance.org/agile101/>
 - This website provided foundational knowledge about the Agile methodology, which was instrumental in guiding the project's iterative development approach.
 7. **GitHub Documentation.** (n.d.). *GitHub Repositories and Version Control*. Retrieved from <https://docs.github.com/en>
 - Used for managing version control and collaboration during the project, GitHub's documentation was vital in setting up and maintaining the project repository.
 8. **PHP: Hypertext Preprocessor.** (n.d.). *PHP Documentation*. Retrieved from <https://www.php.net/docs.php>
-

- The official PHP documentation was essential in developing the backend of the application, particularly for implementing user authentication and reporting workflows.
9. **MySQL Documentation.** (n.d.). *MySQL Database Management*. Retrieved from <https://dev.mysql.com/doc/>
- This resource provided the necessary guidance on database design and management for the project.
10. **ISO/IEC 27001.** *Information Security Management* (2013). Retrieved from <https://www.iso.org/isoiec-27001-information-security.html>
- ISO 27001 was referenced for developing secure login mechanisms and ensuring data privacy in the application.

This list includes academic sources, industry documentation, and practical resources that were used throughout the development of the **Missing Persons Reporting and Matching Application**. These references supported the design, implementation, and security aspects of the project.

APPENDICES

CHAPTER 9

Appendix A: Installation Manual

REQUIREMENTS

1. Software Requirements:

- XAMPP (or equivalent, such as WAMP) with PHP version 7.4 or higher
- MySQL Database Management System
- A modern web browser (e.g., Chrome, Firefox, Edge)

2. Hardware Requirements:

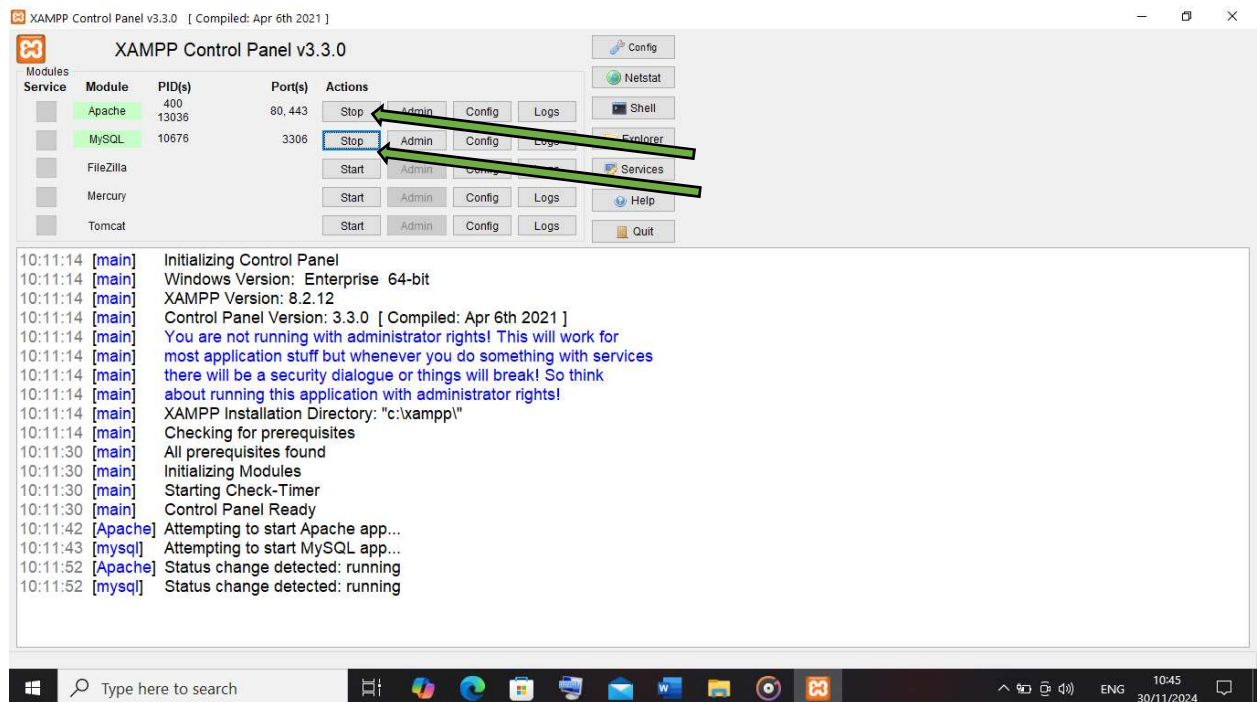
- Minimum 1GB RAM
- 500MB of free disk space
- A computer running Windows, macOS, or Linux

Installation Steps:

1. Download and Install XAMPP:

- Download XAMPP from the official website(www.apachefriends.org)
- Follow the installation instructions and start both the Apache and MySQL services.

Figure 9.1: starting apache and MYSQL



2. The Application:

- Get the project files.
- Extract the files to the htdocs directory inside the XAMPP installation folder.

3. Set Up the Database:

- Open **phpMyAdmin** by navigating to <http://localhost/phpmyadmin/>.
- Create a new database named `missing_persons_db`.
- Import the SQL file (`database/missing_persons_db.sql`) from the project folder.

4. Configure Database Connection:

- Open the `config/DatabaseConfiguration.php` file in the root directory of the project.
- Update the following fields with your local database details:

PHP Copy code for database configurations

```
5. private $host = "localhost"; //server host url
6. private $db_name = "missing_persons_db"; //database name
7. private $username = "root"; //the user name
8. private $password = ""; //the password of the user
```

9. Run the Application:

- Open your browser and go to http://localhost/missing_persons/.
- You should now be able to log in and use the system.

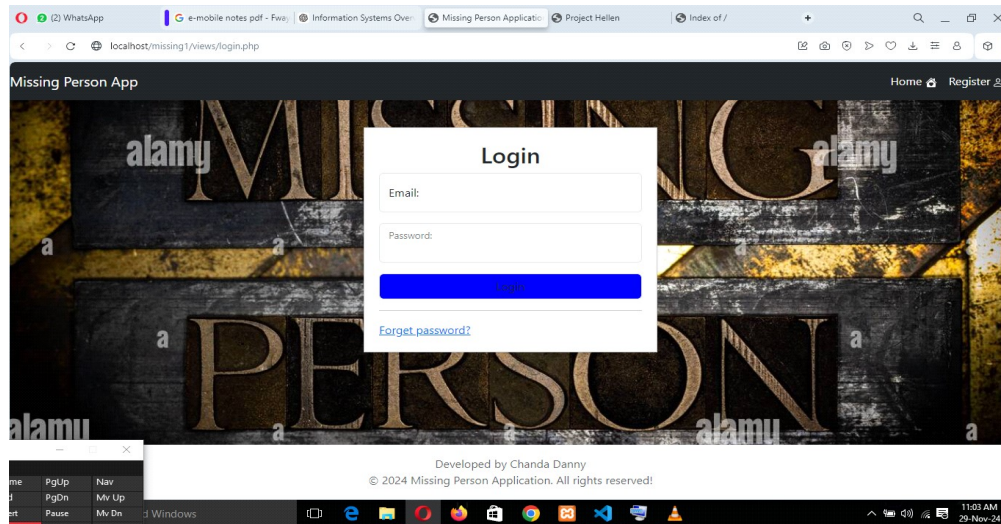
Appendix B: User Manual

Admin Role:

1. Login:

- Enter your admin credentials (email and password) to access the dashboard.
 - Default admin Credentials:
 - Email: dannychanda05@gmail.com
 - Password: 1234
-

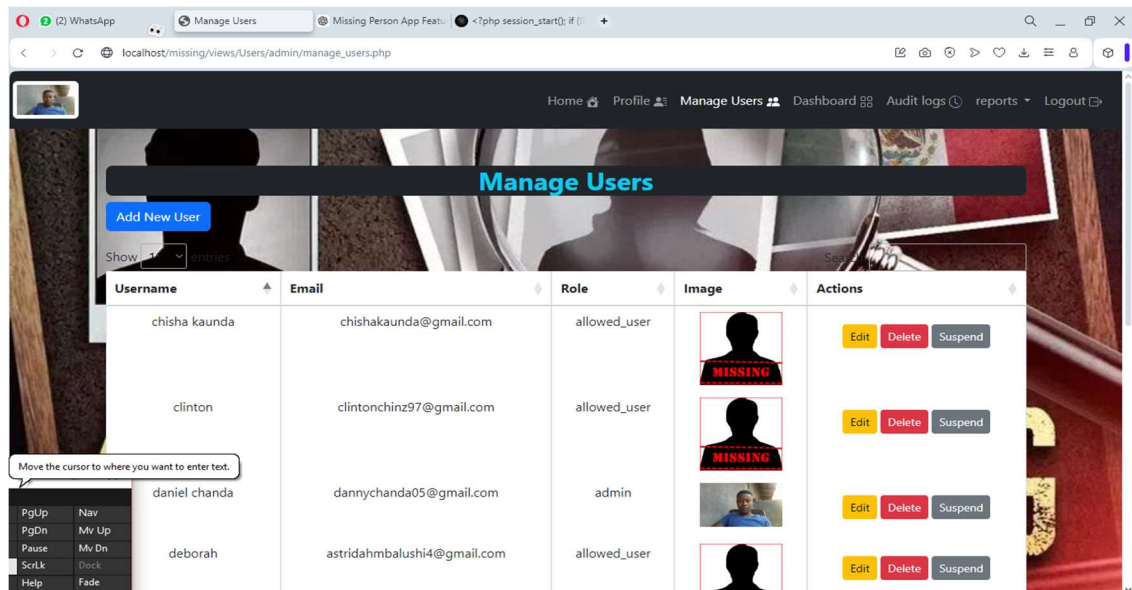
Figure 9.2:login page



2. Manage Users:

- Navigate to the "User Management" section to add, edit, suspend, reactivate or delete user accounts.

Figure 9.3:manage users page



3. Manage Reports:

- Review and approve or reject reported missing or found persons.

Figure 9.4:manage reports page

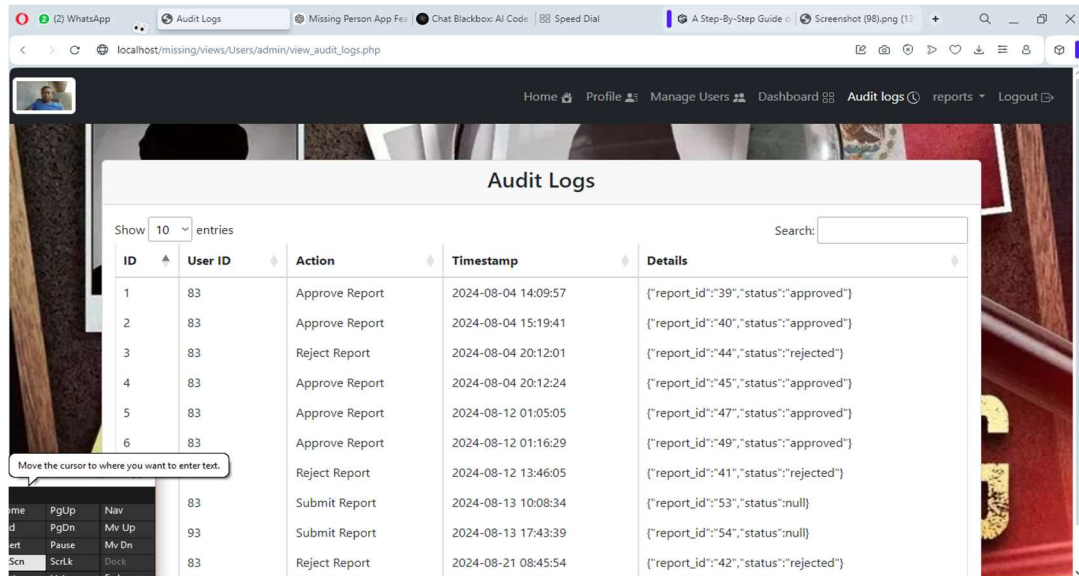
The screenshot displays the 'Approve Reports' interface. At the top, there's a navigation bar with links: Home, Profile, Manage Users, Dashboard, Audit logs, reports (active), and Logout. Below the navigation bar, the main content area is titled 'Approve Reports'. It features a 'Show 10 entries' dropdown and a search bar. The data is presented in a table with the following columns: Type, Name, Age, Gender, Last Seen, Description, Contact Info, and Actions. Each row represents a report and includes 'Approve' and 'Reject' buttons.

Type	Name	Age	Gender	Last Seen	Description	Contact Info	Actions
found	Danny chanda	30	male	2024-12-09	sgvhbjnk	0970671683	<button>Approve</button> <button>Reject</button>
missing	mary phiri	35	male	2024-07-01	black na white	09744463525	<button>Approve</button> <button>Reject</button>
missing	mary phiri	35	male	2024-07-01	black na white	09744463525	<button>Approve</button> <button>Reject</button>
missing	mary phiri	35	male	2024-07-01	black na white	09744463525	<button>Approve</button> <button>Reject</button>

4. View Audit Logs:

- Check audit logs for any actions performed in the system for transparency and security.

Figure 9.5: audit logs page

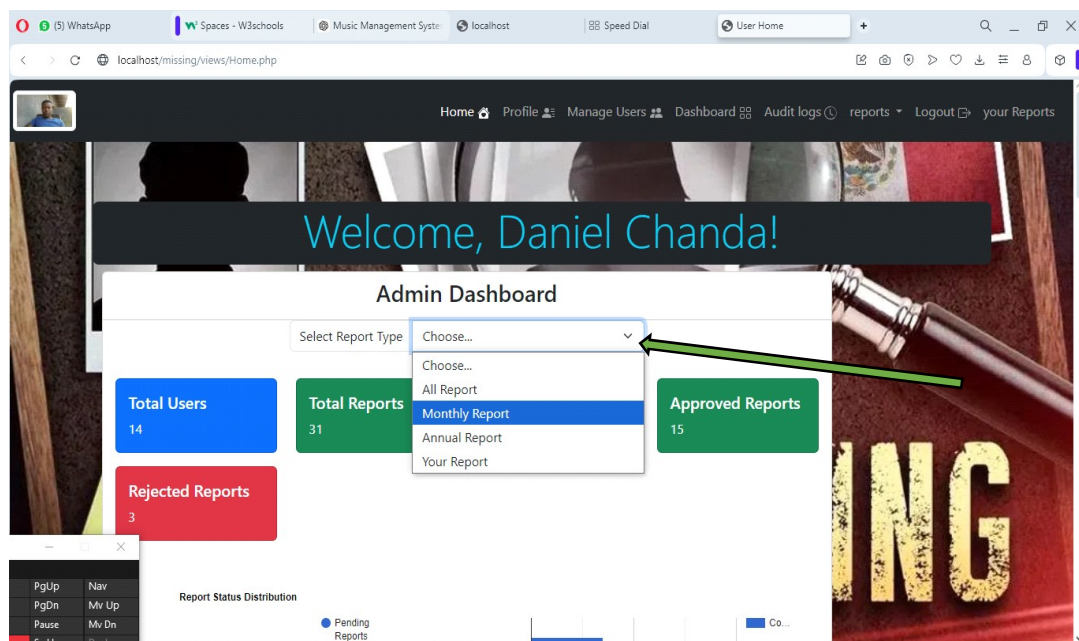


ID	User ID	Action	Timestamp	Details
1	83	Approve Report	2024-08-04 14:09:57	{"report_id":"39","status":"approved"}
2	83	Approve Report	2024-08-04 15:19:41	{"report_id":"40","status":"approved"}
3	83	Reject Report	2024-08-04 20:12:01	{"report_id":"44","status":"rejected"}
4	83	Approve Report	2024-08-04 20:12:24	{"report_id":"45","status":"approved"}
5	83	Approve Report	2024-08-12 01:05:05	{"report_id":"47","status":"approved"}
6	83	Approve Report	2024-08-12 01:16:29	{"report_id":"49","status":"approved"}
	83	Reject Report	2024-08-12 13:46:05	{"report_id":"41","status":"rejected"}
	83	Submit Report	2024-08-13 10:08:34	{"report_id":"53","status":null}
	93	Submit Report	2024-08-13 17:43:39	{"report_id":"54","status":null}
	83	Reject Report	2024-08-21 08:45:54	{"report_id":"42","status":"rejected"}

5. Generate PDF Reports

- On the admin dashboard you can choose to generate reports.

Figure 9.6: report page



Welcome, Daniel Chanda!

Admin Dashboard

Select Report Type: Choose... (dropdown menu open with options: Choose..., All Report, Monthly Report, Annual Report, Your Report)

Total Users
14

Total Reports
31

Approved Reports
15

Rejected Reports
3

Report Status Distribution

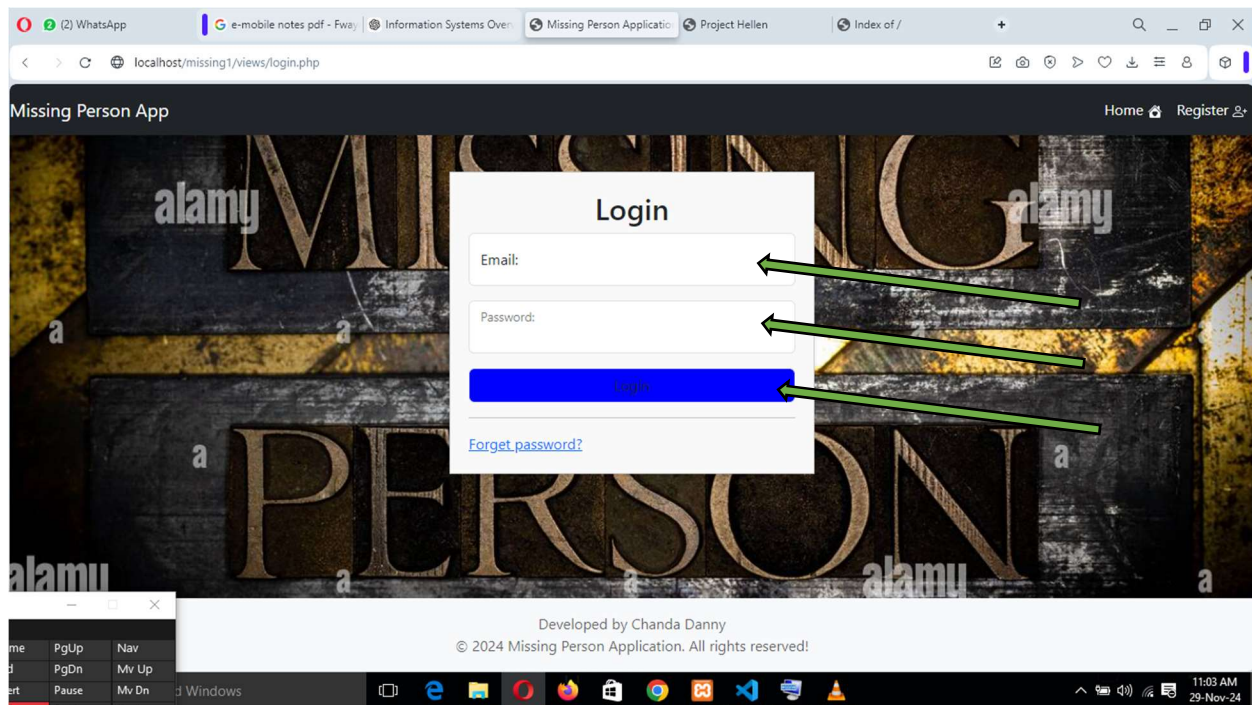
- Pending Reports
- Completed Reports

Allowed User Role:

1. Login:

- Enter your credentials to log in and access the reporting features.

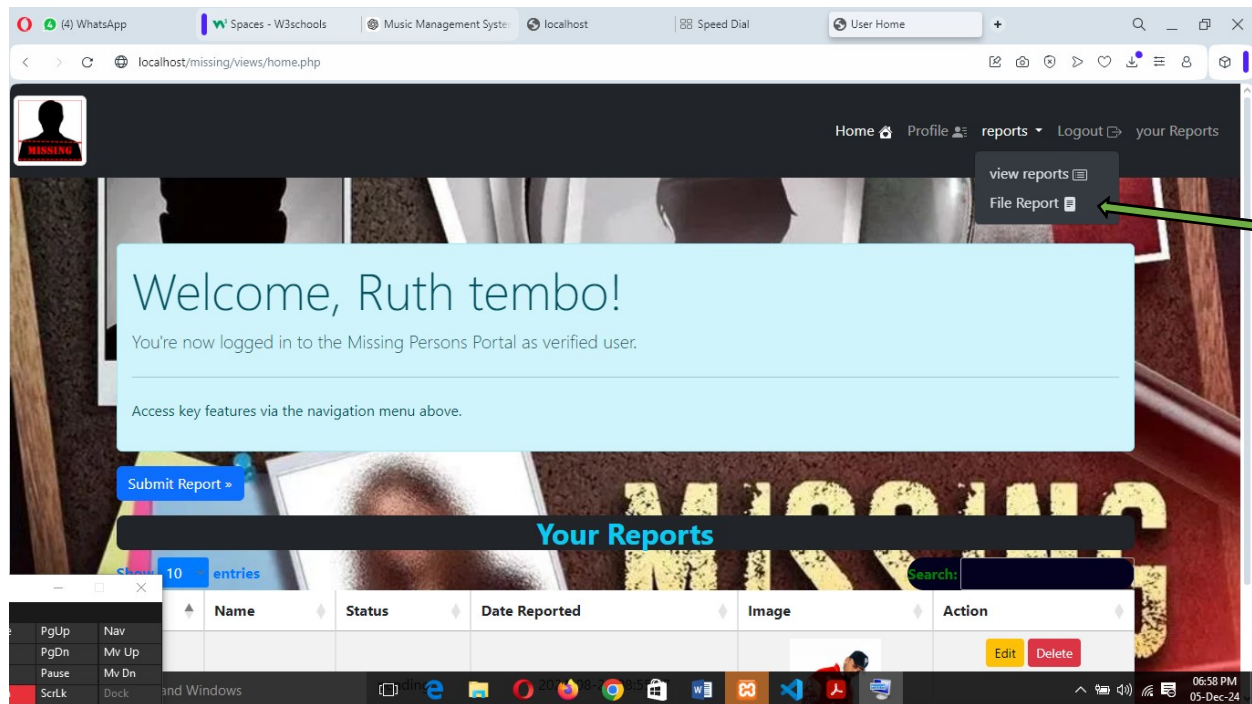
Figure 9.7: login page



2. Report Missing or Found Persons:

- Navigate to the "Reports" section and click on "File Report." Provide all relevant information and submit.

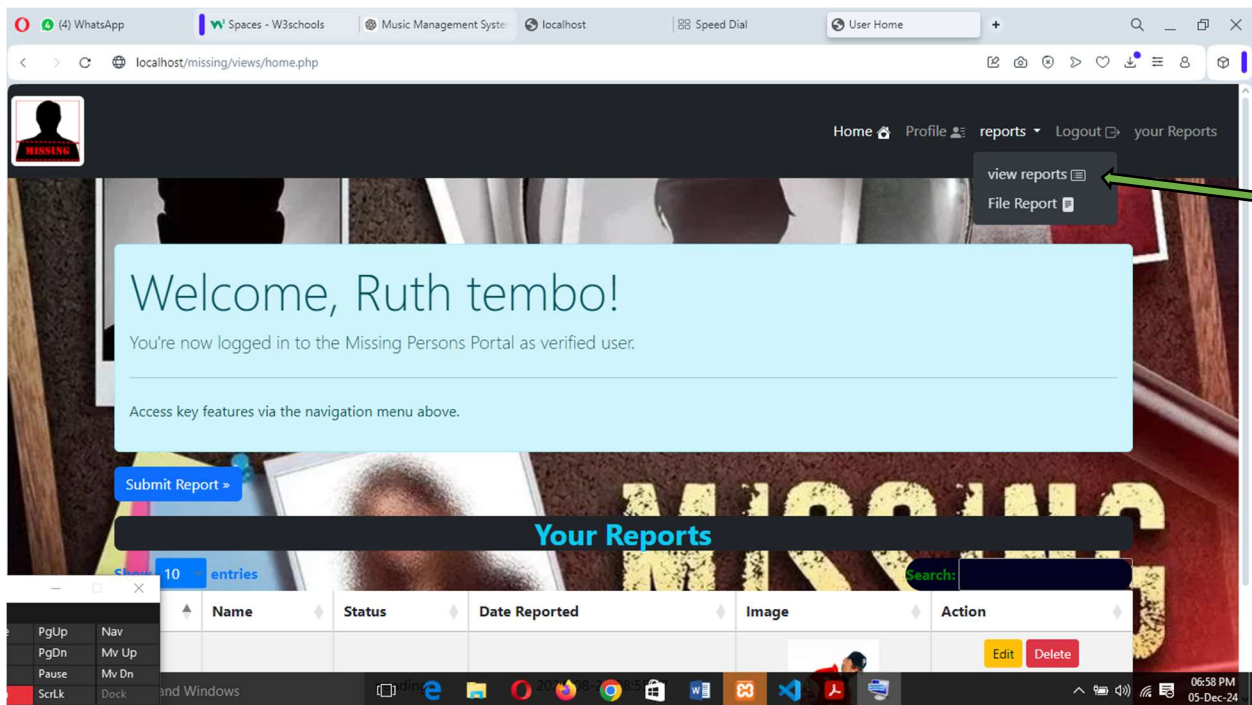
Figure 9.8 : filing report



3. View Reports:

- View existing reports, including your own and those submitted by other users.

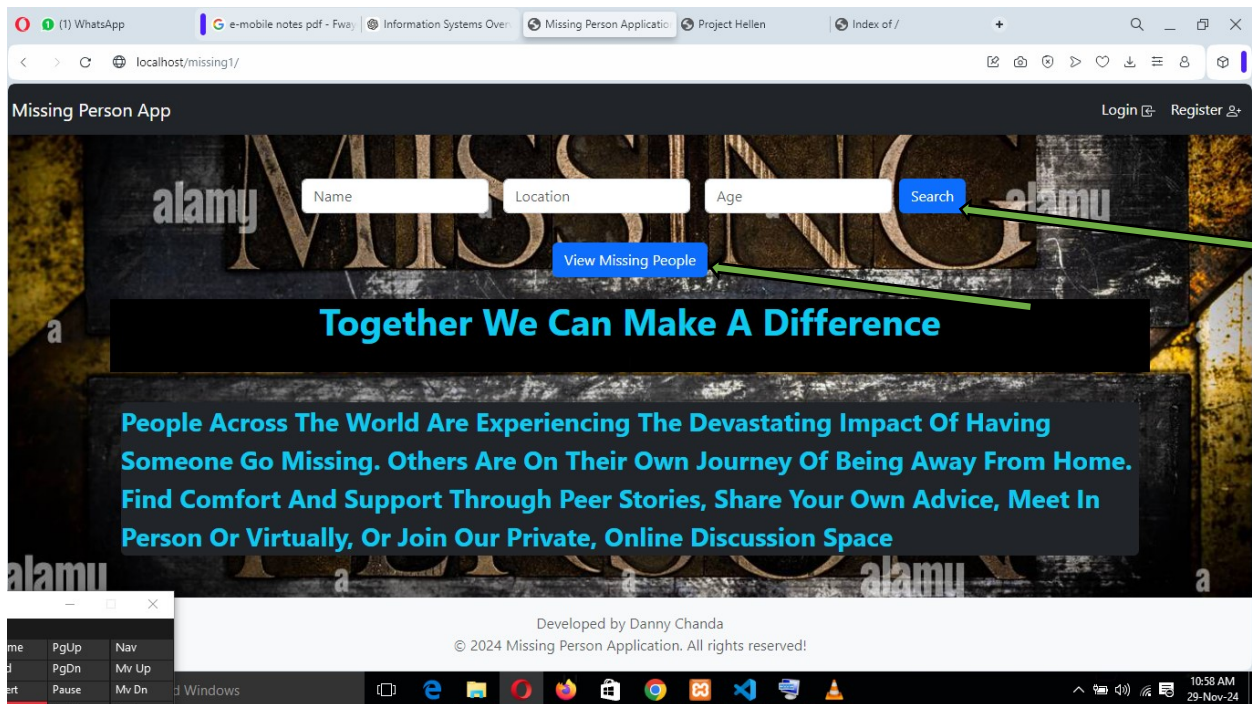
Figure 9.9: viewing reports



Unauthorized User:

1. Browse Reports:

- View the list of reported missing persons on the homepage without logging in.
- Just click the view report button or search using name, age or location.



TECHNICAL SUPPORT SOURCE CODE

CHAPTER 10

Appendix D: Technical Report (Source Code)

- The complete source code of the **Missing Persons Reporting and Matching Application** is included in the project repository. Each file is documented with inline comments for clarity.
- The repository is structured as follows:
 - index.php: Main entry point of the application.
 - Config/: Contains the database configuration file.
 - models/: Contains the PHP model classes for interacting with the database.
 - controllers/: Handles the business logic.
 - views/: Contains the HTML templates for rendering the User Interface.
 - assets/: CSS and JavaScript used for frontend design.
 - Uploads/: contains all the files to be uploaded by the user.
 - Images/: Contains all the image files for frontend design.
 - Libraries/: contains all the third party libraries used e.g PHPMailer, typed.js
 - Util/: contains the utility files for example Email.php which handles emailing.

Source Code

Index.php

The entry point of the system.

```
<?php
// Include necessary files for constants, database configuration, and the Report model
require_once 'constants.php'; // Contains application-wide constants
require_once 'config/DatabaseConfiguration.php'; // Handles database connection setup
require_once 'models/Report.php'; // Defines the Report model for interacting with the database

// Check if a logout message is present in the query parameters
$showAlert = isset($_GET['message']) && $_GET['message'] === 'logged_out';

// Initialize the database connection and create a Report instance
$database = new DatabaseConfiguration();
$db = $database->getConnection();
```

```
$report = new Report($db);

// Fetch all approved missing persons reports from the database
$missingPersons = $report->getApprovedMissingPersons();
?>

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <title>Missing Person Application</title>
  <!-- Bootstrap CSS from CDN -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
  <!-- Custom CSS -->
  <link href="assets/css/master.css" rel="stylesheet">

</script>
<style>
  body {
    padding-top: 56px; /* Offset for fixed navbar */
    background: url('images/missingimg.jpg') no-repeat center center fixed;
    background-size: cover;
  }
  .footer {
    position: fixed;
    bottom: 0;
    width: 100%;
    background-color: #f8f9fa;
    padding: 1rem 0;
    text-align: center;
  }
  .navbar-dark .navbar-nav .nav-link {
    color: white;
  }
```

```

        <label for="searchName" class="visually-hidden">Name</label>
        <input type="text" class="form-control" id="searchName" placeholder="Name">
    </div>

    <div class="form-group mx-2">
        <label for="searchLocation" class="visually-hidden">Location</label>
        <input type="text" class="form-control" id="searchLocation" placeholder="Location">
    </div>

    <div class="form-group mx-2">
        <label for="searchAge" class="visually-hidden">Age</label>
        <input type="number" class="form-control" id="searchAge" placeholder="Age">
    </div>

    <button type="submit" class="btn btn-primary mb-2">Search</button>
</form>

<div class="text-center my-4">
    <button id="viewMissingButton" class="btn btn-primary">View Missing People</button>
</div>

<div id="typed-Wrapper" class="row text-center text-info bg-black">
    <h1 class="text-center text-info fw-bold text-capitalize text" id="welcome-text">Missing
Persons Zambia</h1>
</div>

<hr>

<div class="col-12 bg-dark text-info text-capitalize fw-bold fs-3 mb-2 shadow rounded-2">
    People across the world are experiencing the devastating impact of having someone go missing.
    Others are on their own journey of being away from home. Find comfort and support through peer stories,
    share your own advice, meet in person or virtually, or join our private, online discussion space
</div>

<div class="row" id="missing-persons" style="display: none;">
    <!-- Missing persons data will be loaded here -->
</div>

<div id="loading">

```

```

        <p>Loading more items...</p>
    </div>
</div>
</div>
</div>

<!-- Footer inclusion -->
<?php include 'views/templates/footer.php'; ?>

<!-- JavaScript libraries -->
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.7/dist/umd/popper.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

<script>
    (g=>{var h,a,k,p="The Google Maps JavaScript API",c="google",l="importLibrary",
        q="__ib__",m=document,b=window;b=b[c]||(b[c]={});var d=b.maps||(b.maps={}),r=new Set,
            e=new URLSearchParams,u=()=>h||(h=new Promise(async(f,n)=>{await
(a=m.createElement("script"));
        e.set("libraries",[...r]+"");for(k in g)e.set(k.replace(/[A-Z]/g,t=>"_"+t[0].toLowerCase()),g[k]);
        e.set("callback",c+".maps."+q);a.src=`https://maps.${c}apis.com/maps/api/js?`+e;
            d[q]=f;a.onerror=()=>h=n(Error(p+" could not
load."));a.nonce=m.querySelector("script[nonce]")?.nonce||"";m.head.append(a)}));
            d[l]?console.warn(p+" only loads once.
Ignoring:",g):d[l]=(f,...n)=>r.add(f)&&u().then(()=>d[l](f,...n))})({
        key: "AIzaSyAV3XDFGZcX1LeCtg-RWuGGw7T8-ey4_SM",
        v: "weekly",
        // Use the 'v' parameter to indicate the version to use (weekly, beta, alpha, etc.).
        // Add other bootstrap parameters as needed, using camel case.
    });
</script>

<script>

```

```
$(document).ready(function() {  
  let page = 1; // Current page number for pagination  
  let loading = false; // Flag to prevent multiple simultaneous AJAX requests  
  let searchParams = {}; // Object to store search parameters  
  
  $('#viewMissingButton').on('click', function() {  
    $('#missing-persons').show();  
    loadMoreData();  
    window.location.href = "#missing-persons";  
  });  
  function loadMoreData() {  
    if (loading) return; // Prevent further execution if already loading  
    loading = true; // Set loading flag to true  
    $('#loading').show(); // Show loading indicator  
  
    $.ajax({  
      url: 'fetch_data.php', // URL of the script to fetch data  
      type: 'GET',  
      data: { page: page, ...searchParams }, // Include pagination and search parameters  
      success: function(response) {  
        if (response.trim() === "") {  
          $(window).off('scroll'); // Stop loading more data if no more items  
          $('#loading').html('<p>No more items to load.</p>');  
        } else {  
          $('#missing-persons').append(response); // Append new data to the container  
          page++; // Increment page number for the next load  
        }  
        loading = false; // Reset loading flag  
        $('#loading').hide(); // Hide loading indicator  
  
        // Initialize the map when the modal is shown  
        $('[id^="staticBackdrop"]').on('shown.bs.modal', function (event) {  
          const modalId = $(this).attr('id');  
          const mapId = `map${modalId.replace('staticBackdrop', '')}`; // Get unique map ID
```

```
const latitude = $(this).data('latitude');
const longitude = $(this).data('longitude');

async function initMap(mapId,latitude,longitude) {

var infoWindow;

// Check if coordinates are valid numbers
function isValidCoordinate(value) {
    return !isNaN(value) && value >= -90 && value <= 90;
}

//handling location error
function handleLocationError(browserHasGeolocation, infoWindow, pos) {
    infoWindow.setPosition(pos);
    infoWindow.setContent(
        browserHasGeolocation
        ? "Error: The Geolocation service failed."
        : "Error: Your browser doesn't support geolocation.",
    );
    infoWindow.open(map);
}

const { Map } = await google.maps.importLibrary("maps");
const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");

// Create a new map centered at report coordinates
var map = new Map(document.getElementById("+mapId"), {
    center: {lat: parseFloat(latitude), lng: parseFloat(longitude)},
    zoom: 8,
    mapId: "roadmap",
});

// Create a marker at report coordinates
```

```

        var marker = new AdvancedMarkerElement({
            position: {lat: parseFloat(latitude), lng: parseFloat(longitude)},
            map: map,
            draggable: false
        });
    }
    initMap(mapId,latitude,longitude);
    });
}
});
}

// Function to update search parameters and reload data
function updateSearchParams() {
    searchParams = {
        name: $('#searchName').val(),
        location: $('#searchLocation').val(),
        age: $('#searchAge').val()
    };
    page = 1; // Reset page number to 1
    $('#missing-persons').empty(); // Clear the existing data
    loadMoreData(); // Load new data based on updated search parameters
}

// Infinite scroll functionality to load more data
$(window).scroll(function() {
    if ($(window).scrollTop() + $(window).height() >= $(document).height() - 100) {
        loadMoreData();
    }
});

// Handle search form submission
$('#searchForm').on('submit', function(event) {

```

```

        event.preventDefault(); // Prevent default form submission
        updateSearchParams(); // Update search parameters and reload data
        $('#missing-persons').show();//show the reports container
    });

});

// Script to inform the user if they have been logged out
document.addEventListener('DOMContentLoaded', function () {
    <?php if ($showAlert): ?>
        Swal.fire({
            title: 'Logged out',
            icon: 'success'
        }).then(() => {
            // Remove 'message' query parameter from the URL
            let url = new URL(window.location.href);
            url.searchParams.delete('message');
            window.history.replaceState({}, document.title, url.toString());
        });
    <?php endif; ?>
});
</script>
<script src="libraries/typed.js"></script>
</body>
</html>

```

Model/user.php

Contains a class responsible for handling the data and business logic of a user.

```

<?php
/**
 * the user class is the responsible for modelling the various users of the systym,
 * it is responsible for performing CRUD operations of the user and all user functions in this system.

```

```

* @author Danny Chanda
*/
require_once 'security.php';
class User {
    //conn is responsible for referencing the database connection object
    private $conn;
    //table_name holds the user's database table name
    private $table_name = "users";

    //id,username, email, password, and role are the field of the user's table
    public $id;//unique identifier
    public $username;//user's name
    public $email;//user's email
    public $password;
    public $role;
    public $image;
    public $code;
    public $status = 'not verified';

    public function __construct($db) {
        $this->conn = $db;
    }

    /**
     * register fuction is responsible for handling user registration into the system.
     * @return boolean True if the user was succesfully registered and stored in the database otherwise false.
     */
    public function register() {
        try{
            //check if user did not set their profile picture
            if($this->image == ""){
                //user didn't set picture,set it to default image;
                $this->image = '../uploads/profilePictures/default.jpg';
            }
        }
    }

```

```
$query = "INSERT INTO " . $this->table_name . " SET username=:username, email=:email,  
password=:password, role=:role, image=:image, code=:code, status=:status";
```

```
$stmt = $this->conn->prepare($query);  
$this->username = Security::sanitizeInput($this->username);  
$this->email = Security::sanitizeInput($this->email);  
$this->password = Security::hashPassword($this->password);  
$this->role = Security::sanitizeInput($this->role);  
$this->image = Security::sanitizeInput($this->image);  
$this->code = Security::sanitizeInput($this->code);  
$this->status = Security::sanitizeInput($this->status);
```

```
$stmt->bindParam(":username", $this->username);  
$stmt->bindParam(":email", $this->email);  
$stmt->bindParam(":password", $this->password);  
$stmt->bindParam(":role", $this->role);  
$stmt->bindParam(":image", $this->image);  
$stmt->bindParam(":code", $this->code);  
$stmt->bindParam(":status", $this->status);
```

```
if ($stmt->execute()) {
```

```
    return true;
```

```
}
```

```
    return false;
```

```
} catch (Exception $e) {
```

```
}
```

```
}
```

```
/**
```

```

    * responsible for login the user into the system.
    * it verifies wheather the user is authorised by making use of their credentials.
    * @return boolean true if the user was verified.
    */

public function login() {
    $query = "SELECT id, username, password, status, role FROM " . $this->table_name . " WHERE
email = :email";

    $stmt = $this->conn->prepare($query);
    $this->email = Security::sanitizeInput($this->email);
    $stmt->bindParam(":email", $this->email);
    $stmt->execute();

    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    //check verify password
    if ($row && Security::verifyPassword($this->password, $row['password'])) {
        //verify status
        if($row['status']=='verified'){
            $this->id = $row['id'];
            $this->username = $row['username'];
            $this->role = $row['role'];
            return true;
        }
        return false;
    }
    return false;
}

/**
 * sets the code to a newly generated code
 * @param string $email The email to check.
 * @param string the generated code to be used for user verification.
 * @return boolean True if the email exist and code replaced successfully.
 */

public function setCode($code, $email){

```

```

$query = 'UPDATE '.$this->table_name.' SET code=:code WHERE email=:email;';
$stmt = $this->conn->prepare($query);
$stmt->bindParam(':code', $code);
$stmt->bindParam(':email', $email);
return $stmt->execute();
}

/**
 * retrieves the user's id,username and email using the ID
 * @param int the users ID
 * @return associative_array an array of user details
 */
public function getUserById($id) {
    $query = "SELECT id, username, email, role FROM " . $this->table_name . " WHERE id = :id";

    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(":id", $id);
    $stmt->execute();

    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    return $row;
}

/**
 * deletes the user by making use of the ID
 * @param int user's ID
 * @return boolean True if user was deleted
 */
public function deleteUser($id) {
    $query = "DELETE FROM " . $this->table_name . " WHERE id = :id";

    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(":id", $id);

    if ($stmt->execute()) {

```

```

        return true;
    }
    return false;
}

/**
 * responsible for fetching all users by their id,username,email and role
 * @return associative_array all user's data
 */
public function getAllUsers() {
    $query = "SELECT id, username, email, role, image,status FROM " . $this->table_name;
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

/**
 * updates the user data.This includes username,email, and role.
 * @return boolean true if the update was succesful otherwise false
 */
public function update() {
    $query = "UPDATE " . $this->table_name . " SET username = :username, email = :email, role = :role
WHERE id = :id";
    $stmt = $this->conn->prepare($query);

    // Bind values
    $stmt->bindParam(':username', $this->username);
    $stmt->bindParam(':email', $this->email);
    $stmt->bindParam(':role', $this->role);
    $stmt->bindParam(':id', $this->id);

    return $stmt->execute();
}

```

```

/**
 * returns the curent total number of registered users.
 * @return int number of users
 */
public function getTotalUsers() {
    $query = "SELECT COUNT(*) as total FROM " . $this->table_name;
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetch(PDO::FETCH_ASSOC)['total'];
}
/**
 * for updating the user profile.
 * @return boolean true if the update was succesful
 */

/**
 * Checks if an email and given code match.
 *
 * @param string $email The email to check.
 * @param string $code The code to check.
 * @return boolean True if the email and code match, otherwise false.
 */
function verifyCode($code, $email) {
    // Sanitize input
    $email = Security::sanitizeInput($email);

    // Prepare the query
    $query = "SELECT code FROM " . $this->table_name . " WHERE email = :email AND code = :code
LIMIT 1";

    try {

```

```
// Prepare the statement
$stmt = $this->conn->prepare($query);

// Bind the email and code parameters
$stmt->bindParam(':email', $email);
$stmt->bindParam(':code', $code);

// Execute the statement
$stmt->execute();

// Check if any record was found
if ($stmt->rowCount() > 0) {

    // Prepare the update query
    $updateQuery = "UPDATE " . $this->table_name . " SET status = :status WHERE email = :email
AND code = :code";

    // Prepare the statement
    $stmt = $this->conn->prepare($updateQuery);

    // Bind the parameters
    $status = 'verified'; // The status for the person should be verified in order to perform this
functionality
    $stmt->bindParam(':status', $status);
    $stmt->bindParam(':email', $email);
    $stmt->bindParam(':code', $code);

    // Execute the statement
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    return true; // Email and code match
}
```

```

        return false; // Email and code do not match
    } catch (PDOException $e) {
        // Handle errors (e.g., log them or rethrow)
        error_log("Database error: " . $e->getMessage());
        return false;
    }
}

/**
 * updates the user password
 * @param string new password
 * @param string verified email
 * @param int verified code
 * @return boolean true if the password was successfully changed
 */
public function updatePassword($newPassword, $email, $code) {
    // Prepare the SQL query with the correct placeholders
    $query = "UPDATE " . $this->table_name . "
        SET password=:newPassword
        WHERE email=:email
        AND code=:code
        AND status=:status";

    // Prepare the statement
    $stmt = $this->conn->prepare($query);

    // Sanitize and hash the inputs
    $newPassword = Security::hashPassword(Security::sanitizeInput($newPassword));
    $email = Security::sanitizeInput($email);
    $code = Security::sanitizeInput($code);

    // Bind the parameters to the query
    $status = 'verified'; // We assume the status is currently verified

```

```
$stmt->bindParam(':newPassword', $newPassword);
$stmt->bindParam(':email', $email);
$stmt->bindParam(':code', $code);
$stmt->bindParam(':status', $status);

// Execute the statement and return the result
if ($stmt->execute()) {
    return true;
}

return false;
}

public function suspendUser($id){
    //sql query to suspend the user
    $query = "UPDATE ".$this->table_name." SET status='suspended' WHERE id=:id;";

    //prepare the statement
    $stmt = $this->conn->prepare($query);
    $id = Security::sanitizeInput($id);
    //bind the parameter
    $stmt->bindParam(':id',$id);

    //execute the statement
    if($stmt->execute()){
        return true;
    }
    return false;
}

public function reactivateUser($id){
    //sql query to suspend the user
    $query = "UPDATE ".$this->table_name." SET status='verified' WHERE id=:id;";
```

```
//prepare the statement
$stmt=$this->conn->prepare($query);
$id = Security::sanitizeInput($id);
//bind the parameter
$stmt->bindParam(':id',$id);

//execute the statement
if($stmt->execute()){
    return true;
}
return false;
}
}
```

Model/security.php

Contains a very important model class which contains the static functions responsible for handling the security of the entire application.

```
<?php
/**
 * A very important model class which contains almost the static functions responsible,
 * for handling the Security of the entire application.
 * Different security threats have been identified and included mechanisms to prevent them.
 * @author Danny Chanda
 */
class Security {
    /**
     * does input sanitization by not allowing any code to be treated as code.
     * This will prevent html tags from getting executed by the browser.
     * @param string the input from the user.
     * @return string a string with special characters stripped.
     */
    public static function sanitizeInput($input) {

        return htmlspecialchars(strip_tags($input));

    }

    /**
     * responsible for password encryption
     * @param string a string of password characters to hash
     * @return string an encrypted password.
     */
    public static function hashPassword($password) {
        return password_hash($password, PASSWORD_BCRYPT);
    }

    /**
```

```

    * provide the verification of the password if they match.
    * @param string password entered
    * @param string the hashed password
    * @return boolean true if the password is verified.
    */
    public static function verifyPassword($password, $hash) {
        return password_verify($password, $hash);
    }

    /**
     *
     */
    public static function generateToken() {
        return bin2hex(random_bytes(32));
    }

    public static function verifyToken($token, $sessionToken) {
        return hash_equals($token, $sessionToken);
    }

    /**
     * starts the session if not started.
     */
    public static function startSession() {
        if (session_status() == PHP_SESSION_NONE) {
            session_start();
        }
    }

    public static function generateCsrfToken() {
        if (empty($_SESSION['csrf_token'])) {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        }

        return $_SESSION['csrf_token'];
    }

```

```
}

public static function validateCsrfToken($token) {
    return isset($_SESSION['csrf_token']) && hash_equals($_SESSION['csrf_token'], $token);
}

public static function sanitize($data) {
    return htmlspecialchars(strip_tags($data));
}

/**
 * checks if the provided email is valid.
 * @param string a provided email
 * @return boolean true if the email is valid
 */
public static function validateEmail($email) {
    return filter_var($email, FILTER_VALIDATE_EMAIL);
}

/**
 * checks if the provided integer is a valid integer.
 * @param int a provided integer
 * @return boolean true if the integer is valid
 */
public static function validateInt($int) {
    return filter_var($int, FILTER_VALIDATE_INT);
}

/**
 * checks if the provided string is a valid string.
 * @param string a provided string of text
 * @return boolean true if the string is valid
 */
public static function validateString($string) {
    return preg_match("/^[a-zA-Z0-9\s]+$/", $string);
}
```

```

}

/**
 * checks if the provided string contains only letters, numbers, spaces, commas, periods, and dashes.
 * @param string a provided string of text
 * @return boolean true if the string is valid
 */
public static function validateDescription($description) {
    // Regular expression to allow letters, numbers, spaces, commas, periods, and dashes
    return preg_match("/^[a-zA-Z0-9\s,.\!?\-]*$/", $description);
}

/**
 * checks if the provided value is one of the expected value from the select box.
 * @param string the selected value.
 * @param array a list of allowed values.
 * @return boolean true if the selected value is allowed.
 */
public static function validateSelect($value, $allowedValues) {
    return in_array($value, $allowedValues);
}

/**
 * makes sure that the input date is valid.
 * @param string an input date
 * @return boolean true if the date is valid.
 */
public static function validateDate($input_date) {

    $date = DateTime::createFromFormat('Y-m-d', $input_date);

    if ($date !== false) {
        // Valid date
    }

```

```
        return true;
    } else {
        // Invalid date
        return false;
    }
}
?>
```

Model/report.php

Handles the business logic of missing and found persons reports.

```
<?php
/**
 * responsible for modeling the reports submitted by users.
 * provides all methods relating to reports.
 * @author Chanda Danny
 */
class Report {
    private $conn;
    private $table_name = "reports";

    public $id;
    public $type;
    public $name;
    public $age;
    public $gender;
    public $last_seen;
    public $description;
    public $contact_info;
    public $user_id;
    public $status;
    public $location;
```

```
public $longitude;
public $latitude;

public function __construct($db) {
    $this->conn = $db;
}

/**
 * creates the report given the information by the user.
 * return true if the report is successfully created otherwise false
 */
public function create() {
    $query = "INSERT INTO " . $this->table_name . "
        SET type=:type, name=:name, age=:age, gender=:gender, last_seen=:last_seen,
        description=:description, contact_info=:contact_info, user_id=:user_id,
        image_path=:image_path, location=:location,latitude=:latitude,longitude=:longitude,
status='pending'";

    $stmt = $this->conn->prepare($query);

    // Bind values
    $stmt->bindParam(':type', $this->type);
    $stmt->bindParam(':name', $this->name);
    $stmt->bindParam(':age', $this->age);
    $stmt->bindParam(':gender', $this->gender);
    $stmt->bindParam(':last_seen', $this->last_seen);
    $stmt->bindParam(':description', $this->description);
    $stmt->bindParam(':contact_info', $this->contact_info);
    $stmt->bindParam(':user_id', $this->user_id);
    $stmt->bindParam(':image_path', $this->image_path);
    $stmt->bindParam(':location', $this->location);
    $stmt->bindParam(':latitude', $this->latitude);
    $stmt->bindParam(':longitude', $this->longitude);

    if ($stmt->execute()) {
```

```
        //retrive the last inserted id
        $this->id = $this->conn->lastInsertId();
        return true;
    }

    return false;
}

/**
 * retrieves all the pending reports
 */
public function getPendingReports() {
    $query = "SELECT * FROM " . $this->table_name . " WHERE status = 'pending'";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

/**
 * retrieves all the reports with persons marked as missing
 */
public function getPendingMissingReports() {
    $query = "SELECT * FROM " . $this->table_name . " WHERE type = 'missing' AND status =
'approved'";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

/**
 * change the status of the report
 */
public function updateStatus() {
```

```
$query = "UPDATE " . $this->table_name . " SET status = :status WHERE id = :id";
$stmt = $this->conn->prepare($query);

// Bind values
$stmt->bindParam(':status', $this->status);
$stmt->bindParam(':id', $this->id);

return $stmt->execute();
}

/**
 * retrieve the approved missing persons
 */
public function getApprovedMissingPersons() {
    $query = "SELECT * FROM " . $this->table_name . " WHERE status = 'approved' AND type =
'missing'";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

/**
 * count all the reports
 */
public function getTotalReports() {
    $query = "SELECT COUNT(*) as total FROM " . $this->table_name;
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetch(PDO::FETCH_ASSOC)['total'];
}

/**
```

```

    * count all the peding reports
    */

public function getPendingReportsCount() {
    $query = "SELECT COUNT(*) as total FROM " . $this->table_name . " WHERE status = 'pending'";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetch(PDO::FETCH_ASSOC)['total'];
}

/**
    * count all the approved reports
    */

public function getApprovedReportsCount() {
    $query = "SELECT COUNT(*) as total FROM " . $this->table_name . " WHERE status = 'approved'";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetch(PDO::FETCH_ASSOC)['total'];
}

/**
    * count all the regected reports
    */

public function getRejectedReportsCount() {
    $query = "SELECT COUNT(*) as total FROM " . $this->table_name . " WHERE status = 'rejected'";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetch(PDO::FETCH_ASSOC)['total'];
}

/**
    * performs searching and filtering of reports. This is done by name,status,from_date and to_date.

```

```
*/  
  
public function searchReports($name, $status, $from_date, $to_date) {  
    $query = "SELECT * FROM " . $this->table_name . " WHERE 1=1";  
  
    if ($name) {  
        $query .= " AND name LIKE :name";  
    }  
    if ($status) {  
        $query .= " AND status = :status";  
    }  
    if ($from_date) {  
        $query .= " AND created_at >= :from_date";  
    }  
    if ($to_date) {  
        $query .= " AND created_at <= :to_date";  
    }  
  
    $stmt = $this->conn->prepare($query);  
  
    if ($name) {  
        $stmt->bindValue(':name', '%' . $name . '%');  
    }  
    if ($status) {  
        $stmt->bindValue(':status', $status);  
    }  
    if ($from_date) {  
        $stmt->bindValue(':from_date', $from_date);  
    }  
    if ($to_date) {  
        $stmt->bindValue(':to_date', $to_date);  
    }  
  
    $stmt->execute();  
}
```

```

        $reports = $stmt->fetchAll(PDO::FETCH_ASSOC);

        return $reports;
    }

    //-----Matching functions-----

    public function findMatches() {
        $query = "SELECT * FROM " . $this->table_name . " WHERE type = 'found' AND status = 'pending'
AND name = ? AND age = ? AND gender = ?";

        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(1, $this->name);
        $stmt->bindParam(2, $this->age);
        $stmt->bindParam(3, $this->gender);

        $stmt->execute();

        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }

    public function markAsFound() {
        $query = "UPDATE " . $this->table_name . " SET status = 'found' WHERE id = ?";

        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(1, $this->id);

        return $stmt->execute();
    }

    //-----auditing functions-----

    //function to log admin actions
    public function logAction($user_id, $action, $details) {
        $query = "INSERT INTO audit_logs (user_id, action, details) VALUES (:user_id, :action, :details)";

        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':user_id', $user_id);
        $stmt->bindParam(':details', $details);

        $stmt->execute();
    }
}

```

```

// method to approve a report and log the action
public function approveReport($report_id, $user_id) {
    $query = "UPDATE reports SET status = 'approved' WHERE id = :report_id";
    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':report_id', $report_id);
    if ($stmt->execute()) {
        $this->logAction($user_id, 'approve_report', "Report ID: $report_id approved.");
        return true;
    }
    return false;
}

// method to reject a report and log the action
public function rejectReport($report_id, $user_id, $reason) {
    $query = "UPDATE reports SET status = 'rejected', rejection_reason = :reason WHERE id = :report_id";
    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':report_id', $report_id);
    $stmt->bindParam(':reason', $reason);
    if ($stmt->execute()) {
        $this->logAction($user_id, 'reject_report', "Report ID: $report_id rejected. Reason: $reason");
        return true;
    }
    return false;
}

/**
 * Get the distribution of a user's reports by status (e.g., resolved vs. ongoing).
 *
 * @param int $userId The ID of the user.
 * @return array An associative array with status as the key and count as the value.

```

```

*/

public function getUserReportsByStatus($userId) {
    $query = "SELECT status, COUNT(*) as count
              FROM " . $this->table_name . "
              WHERE user_id = :user_id
              GROUP BY status";

    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':user_id', $userId);
    $stmt->execute();

    $userReportsByStatus = [];
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        $userReportsByStatus[$row['status']] = $row['count'];
    }

    return $userReportsByStatus;
}

/**
 * Get the distribution of cases by status reported by a specific user.
 *
 * @param int $userId The ID of the user.
 * @return array An associative array with status as the key and count as the value.
 */

public function getUserCaseDistribution($userId) {
    $query = "SELECT status, COUNT(*) as count
              FROM " . $this->table_name . "
              WHERE user_id = :user_id
              GROUP BY status";

    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':user_id', $userId);
    $stmt->execute();

```

```

$caseDistribution = [];
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    $caseDistribution[$row['status']] = $row['count'];
}

return $caseDistribution;
}

/**
 * Get the number of reports filed by a specific user over time.
 *
 * @param int $userId The ID of the user.
 * @return array An associative array where the keys are months and the values are the number of reports
filed in that month.
 */
public function getReportsByUser($userId)
{
    try {
        // SQL query to count reports filed by the user, grouped by month.
        $query = "SELECT DATE_FORMAT(created_at, '%Y-%m') AS month, COUNT(*) AS total_reports
        FROM reports
        WHERE user_id = :user_id
        GROUP BY month
        ORDER BY month ASC";

        // Prepare the SQL statement
        $stmt = $this->conn->prepare($query);

        // Bind the user ID to the statement
        $stmt->bindParam(':user_id', $userId, PDO::PARAM_INT);

        // Execute the statement
        $stmt->execute();
    }
}

```

```

        // Fetch all results as an associative array
        $results = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Return the results
        return $results;
    } catch (PDOException $e) {
        // Log error and return an empty array if something goes wrong
        error_log("Error in getReportsByUser: " . $e->getMessage());
        return [];
    }
}

/**
 * Get the total number of reports filed each month.
 *
 * @return array An associative array where the keys are months (formatted as 'YYYY-MM') and the values
are the total number of reports filed in that month.
 */
public function getReportsByMonth()
{
    try {
        // SQL query to count total reports filed each month.
        $query = "SELECT DATE_FORMAT(created_at, '%Y-%m') AS month, COUNT(*) AS total_reports
        FROM reports
        GROUP BY month
        ORDER BY month ASC";

        // Prepare the SQL statement
        $stmt = $this->conn->prepare($query);

        // Execute the statement
        $stmt->execute();

        // Fetch all results as an associative array

```



```

        $results = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Return the results
        return $results;
    } catch (PDOException $e) {
        // Log error and return an empty array if something goes wrong
        error_log("Error in getReportsByMonth: " . $e->getMessage());
        return [];
    }
}

/**
 * Retrieve all reports.
 *
 * @return array An array containing all reports.
 */
public function getAllReports() {
    $query = "SELECT * FROM " . $this->table_name;
    $stmt = $this->conn->prepare($query);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

/**
 * Retrieve reports within a specified period.
 *
 * @param string $startDate Start date for the period.
 * @param string $endDate End date for the period.
 */
?>

```

Config/databaseConfiguration.php

```
<?php
```

```

class DatabaseConfiguration {
    private $host = "localhost";
    private $db_name = "missing_persons_db";
    private $username = "root";
    private $password = "";
    public $conn;

    public function getConnection() {
        $this->conn = null;

        try {
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
            $this->conn->exec("set names utf8");
        } catch (PDOException $exception) {
            echo "Connection error: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
?>

```

Utils/Email.php

```

<?php
require_once __DIR__ . '/../libraries/phpMailer/PHPMailer.php';

require_once __DIR__ . '/../libraries/phpMailer/Exception.php';
require_once __DIR__ . '/../libraries/phpMailer/SMTP.php';

use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

class Email {

```

```
private $mail;

public function __construct() {
    $this->mail = new PHPMailer(true);

    // Server settings
    $this->mail->isSMTP();
    $this->mail->Host = 'smtp.gmail.com'; // Set the SMTP server to send through
    $this->mail->SMTPAuth = true;
    $this->mail->Username = 'dannychanda0555@gmail.com'; // SMTP username
    $this->mail->Password = 'fmrdoeno lyfr qufg'; // SMTP password
    $this->mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
    $this->mail->Port = 587;

    // Recipients
    $this->mail->setFrom('dannychanda05@gmail.com', 'Missing Persons App');
}

public function sendEmail($to, $subject, $body) {
    try {

        $this->mail->addAddress($to);
        $this->mail->Subject = $subject;
        $this->mail->Body = $body;
        $this->mail->isHTML(true);

        $this->mail->send();

        return true;
    } catch (Exception $e) {

        return false;
    }
}
```

```
}  
?>
```

These appendices provide essential supporting materials for the project, from installation and user instructions to the source code. They aim to ensure smooth implementation, ease of use, and proper system functioning.