UNIVERSITY OF BUEA

REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

P.O. Box 63,

Buea, South West Region

CAMEROON

Tel: (237) 3332 21 34/3332 26 90

Fax: (237) 3332 22 72

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

ARCHIVAL AND RETRIEVAL OF MISSING OBJECTS USING IMAGE MATCHING ALGORITHMS AND ADVANCED MACHINE VISION TECHNIQUES

PHASE 2: REQUIREMENT ANALYSIS

By:

Group 6

Supervisor:

Dr Nkemeni Valery

University of Buea

2nd Semester 2023/2024 Academic Year

LIST OF PARTICIPANTS

NAMES	MATRICULE	Speciality
KOUETE TEPE KENNETH	FE21A220	Software
ZELEFACK MARIE	FE21A330	Software
EFUETAZOH ASONG RODERIC	FE21A179	Network
TAKO DIEUDONNEVOJONG	FE21A310	Software
BUINFUN MONIE JULIUS	FE21A154	Software

ABSTRACT

The system design phase of our project represents a critical transition from requirement analysis to detailed planning and blueprinting of the system architecture. This phase focuses on translating the collected requirements into structured and comprehensive design elements, ensuring that all components are well-defined and integrated to achieve the desired functionalities and performance.

This report presents a detailed overview of the system design, including the context diagram, use case diagram, sequence diagrams, class diagrams, and deployment diagram. The context diagram provides a high-level view of the system's interactions with external entities, establishing the system's boundaries and external dependencies. The use case diagram captures the primary functionalities of the system and the various actors involved, ensuring that all user interactions are comprehensively understood and documented.

The sequence diagrams further elaborate on these interactions by detailing the flow of information and processes over time for key use cases, providing a clear visualization of dynamic behaviour. The class diagrams offer a static view of the system’s structure, outlining the essential classes, their attributes, methods, and relationships, which form the backbone of the system's functionality. The deployment diagram illustrates the physical deployment of software components on hardware nodes, providing a clear understanding of the system's infrastructure and how different components interact in the production environment.

This design phase ensures that all aspects of the system are meticulously planned, facilitating a smooth transition to the implementation phase. It aims to ensure that the final product is robust, efficient, and scalable, meeting all specified requirements and laying a solid foundation for future enhancements and maintenance.

Table Of Content

TABLE OF FIGURES

INTRODUCTION

The system design phase for our project is crucial as it translates the requirements identified in the analysis phase into detailed architectural and design specifications. This phase involves creating several models to ensure a comprehensive understanding of the system's structure and interactions. We start with a context diagram to outline the system boundaries and external interactions. The use case diagram captures all essential functionalities and user interactions. Sequence diagrams detail the flow of information and processes for key operations, while class diagrams define the static structure of the system, including classes, attributes, methods, and relationships. Lastly, the deployment diagram illustrates the physical deployment of software components, ensuring efficient interaction in the production environment. This design phase ensures that the system is robust, scalable, and capable of meeting all requirements, providing a clear blueprint for the development team.

OBJECTIVES

The objective of this phase is to develop a comprehensive system design that translates requirements into detailed architectural and design specifications, ensuring clarity in system structure, interactions, and deployment for robust and scalable implementation.

SCOPE

This phase covers the creation of various design diagrams and specifications, including the context diagram, use case diagram, sequence diagram, class diagram, and deployment diagram. These diagrams will define the system architecture, highlight user interactions, outline the sequence of operations, structure the data model, and specify the deployment environment. The goal is to provide a clear blueprint for the development team, ensuring the system meets the identified requirements and functions effectively within its intended environment.

CONTEXT DIAGRAM

The context diagram provides a high-level overview of the system, showing the interaction between the system and external entities. This diagram will help to understand the boundaries of the system and the flow of data between the system and its external actors.

Components of the Context Diagram

- **External Entities:** Users, Admins, and External Services (e.g., notification services, mapping APIs).
- **System**: Lost and Found Item Management System (LFIMS).

External Entities

- **Users**: Individuals who use the system to report lost items, search for matches, and track items.
- **Admins**: System administrators who manage the database, verify claims, and provide support.
- **External Services**: Third-party services for notifications, maps, and possibly image processing.

High-Level Functions of LFIMS

- User Management
- Image Upload and Processing
- Search and Matching
- Notifications
- Support

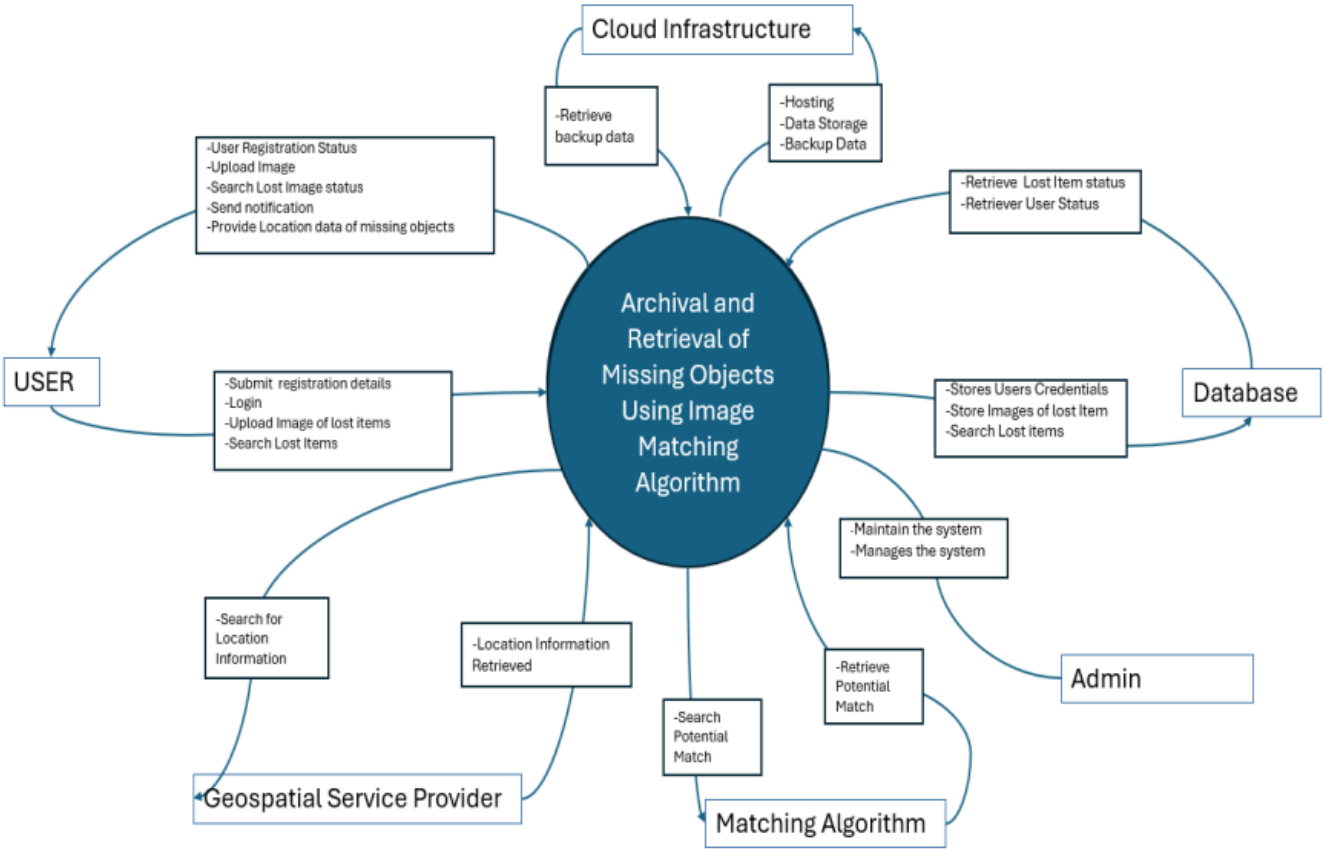

Figure 1: Context Diagram

Context Diagram Description

Users:

- **Inputs**: Upload images, search criteria, profile details.
- **Outputs**: Notifications, search results, tracking information, support responses.

Admins:

- **Inputs**: Verification information, system management commands.
- **Outputs**: Confirmations, alerts, support resolutions.

External Services:

- **Geospatial Location Service**: Provides geolocation data for tracking matched items.
- **Matching Algorithm**: Handles image and data matching processes.
- **Business Logic**: Contains the core logic for operations and workflows.
- **Database**: Stores all user data, images, metadata, and system logs.
- **Cloud Infrastructure**: Provides the necessary environment for deploying the application, ensuring scalability and reliability.

USECASE DIAGRAM

The use case diagram is a visual representation of the functional requirements of the system from the perspective of the actors interacting with it. It illustrates the various ways in which actors (such as users, administrators, and external systems) interact with the system to achieve specific goals or tasks. Each use case represents a specific functionality or feature of the system, while actors represent the roles or entities that initiate these interactions. The use case diagram provides a high-level overview of the system's behavior and helps stakeholders understand the system's functionality and the interactions between different components. In this section, we will present the actors involved in the system, identify the key use cases, and illustrate their relationships in the use case diagram.

Figure 2: use case Diagram.

Use case Description

1. **User Actor:**
    - **Register:** Users begin by registering with the system to gain access.
    - **Login:** After registration, users can log in to perform various tasks.
    - **Select Image:** Users can search for objects by selecting an image.
    - **Search Object:** Users can search for missing objects based on uploaded details.
    - **Upload Object:** Users can contribute information about missing objects to the system.
    - **Receive Notification:** If someone uploads information about an object that another user has searched for, the searching user receives a notification.
    - **Provide Feedback:** Users can give feedback on received notifications.
    - **Share on Social Media:** Users can share information about objects on social media platforms.
    - **Get Help:** Users can seek assistance if needed.
2. **Admin Actor:**
    - **Authentication:** Admins handle user authentication.
    - **Manage Users:** Admins manage user accounts.
    - **Monitor System:** Admins oversee overall system performance.
    - **Support Users:** Admins assist users with queries or issues.
    - **Send Notification:** Admins send out notifications as required.
3. **Geospatial Service Actor:**
    - **Access Geospatial Service:** Interaction with an external geospatial service for locating or mapping objects within the application.

Use Case	Precondition(s)	Postcondition(s)	Description
Register	User is not registered in the system.	User successfully registers and receives a unique identifier (e.g., user ID).	Users can register by providing necessary details (e.g., name, email, password). The system validates the information and creates a new user account.
Login	User is registered in the system.	User successfully logs in and gains access to their account.	Registered users can log in by providing valid credentials (e.g., username and password). The system verifies the credentials and grants access to the user's account.
Select Image	User is logged in.	User selects an image for searching missing objects.	Users can choose an image (e.g., a photo of a missing object) to initiate a search within the system.
Search Object	User is logged in.	System has relevant data about missing objects.	Users search for missing objects based on criteria (e.g., image, description). The system retrieves relevant information about the searched object (if available).
Upload Object	User is logged in.	User provides details about a missing object.	Users can contribute information about a missing object by uploading relevant details (e.g., description, image, location). The system stores this information for future reference.
Receive Notification	User has uploaded information about a missing object.	User receives a notification if another user uploads information about the same missing object.	When someone uploads details about an object that another user has searched for, the searching user receives a notification (e.g., email, in-app notification).
Provide Feedback	User has received a notification.	User provides feedback (positive/negative) on the notification.	After receiving a notification (e.g., about a found object), users can provide feedback (e.g., confirming the match or reporting inaccuracies).
Share on Social Media	User has found relevant information about a missing object.	User shares information about the missing object on social media platforms.	Users can share details about a missing object (e.g., a link to the object's profile) on social media channels (e.g., Twitter, Facebook). This helps spread awareness and potentially aids in locating the object.
Get Help	User encounters an issue or needs assistance.	User seeks help from the system	If users face any problems (e.g., technical issues, confusion), they can request assistance from the system.

		(e.g., through a support channel).	The system provides relevant support or guidance.
Authentication	Admin is logged in.	Admin performs authentication-related tasks (e.g., verifying user credentials).	Admins handle user authentication, ensuring that only authorized users can access the system.
Manage Users	Admin is logged in.	Admin manages user accounts (e.g., creating, updating, or deactivating accounts).	Admins have the authority to manage user accounts, including creating new accounts, updating user information, or deactivating accounts as needed.
Monitor System	Admin is logged in.	Admin monitors system performance (e.g., server health, response times, other system matrix).	Admins oversee the overall system performance, ensuring it runs smoothly and efficiently. They may monitor server metrics, logs, and other relevant data.
Support Users	Admin is logged in.	Admin assists users with queries, issues, or requests.	Admins provide support to users by addressing their questions, resolving issues, and offering guidance.
Send Notification	Admin identifies a relevant event (e.g., object match, system update).	Admin sends out notifications to relevant users (e.g., via email or in-app messages).	Admins can send notifications to users based on specific events (e.g., notifying a user about a found object, system maintenance, or updates).
Access Geospatial Service	System interacts with an external geospatial service.	System accesses geospatial data (e.g., coordinates, maps).	The system communicates with an external geospatial service (e.g., Google Maps API) to retrieve location-related information. This interaction allows users to locate missing objects geographically or visualize them on a map.

SEQUENCE DIAGRAM

In this section, we delve into the dynamic behaviour of the Lost Item Management System (LIMS) through sequence diagrams. Sequence diagrams provide a visual representation of the interactions between various components or actors within the system, illustrating the sequence of messages exchanged during a particular scenario or use case execution. By dissecting these interactions step by step, we gain deeper insights into the system's functionality and understand how different elements collaborate to fulfil user requests or system processes. Through the following sequence diagrams, we aim to elucidate the intricate workings of LIMS, highlighting key interactions and message flows that drive its operation.

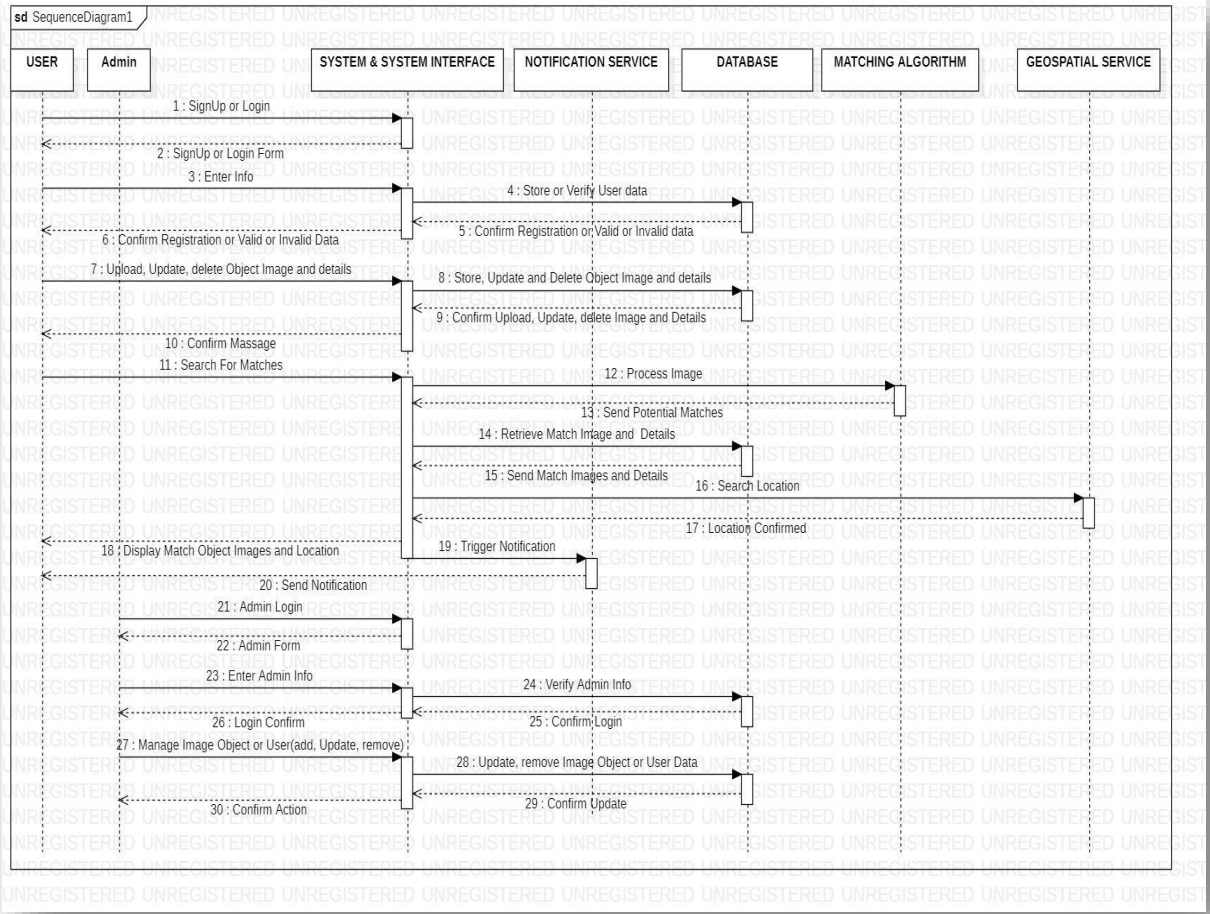

Figure 3: Sequence Diagram.

Sequence diagram Description

1. **Actors:**
    - **User:** Represents an individual interacting with the system.
    - **Admin:** Represents an administrative user responsible for managing system operations.
2. **Components:**
    - **System A/System Interface:** Represents the system or its interface through which users interact.
    - **Notification Service:** Handles notifications (e.g., email, in-app messages) to users.
    - **Database:** Stores relevant data (e.g., user information, object details).
    - **Matching Algorithm:** Performs object matching (e.g., finding lost objects based on descriptions or images).
    - **Geospatial Service:** Provides location-related data (e.g., coordinates, maps).
3. **Sequence of Actions:**
    - **1. Signup or Login:**
        - User or Admin initiates the interaction by signing up or logging in.
    - **2. Enter Info:**
        - User/Admin provides relevant information (e.g., registration details, login credentials).
    - **3. Submit Query:**
        - User submits a query (e.g., searching for a lost object).
    - **4. Store/Update/Delete Object Image and Details:**
        - Admin manages object data (e.g., adding, updating, or removing images and details).
    - **5. Confirm Match:**
        - System confirms a match (e.g., found object matches a user's query).
    - **6. Retrieve Match Image and Details:**
        - System retrieves details of the matched object.
    - **7. Send Match Images and Details:**
        - Notification service sends relevant information to the user.
    - **8. Search Location:**

- System searches for the location of the matched object.

- **9. Display Match Object Images and Location:**
  - User receives information about the found object's images and location.

4. **Notes:**
   - The sequence progresses vertically (top to bottom) to represent time.
   - Objects involved in the operation are listed horizontally (left to right) based on their participation in the message sequence.

CLASS DIAGRAM

In this section, we transition from exploring the dynamic behaviour of the Lost Item Management System (LIMS) to analysing its structural aspects through class diagrams. Class diagrams provide a static view of the system, illustrating the various classes, their attributes, methods, and relationships. By visually representing the system's architecture and organization of classes, we can comprehend how different components collaborate to achieve the system's functionality. Through the following class diagrams, we aim to elucidate the key classes and their associations within LIMS, offering a comprehensive understanding of its underlying structure and design principles.

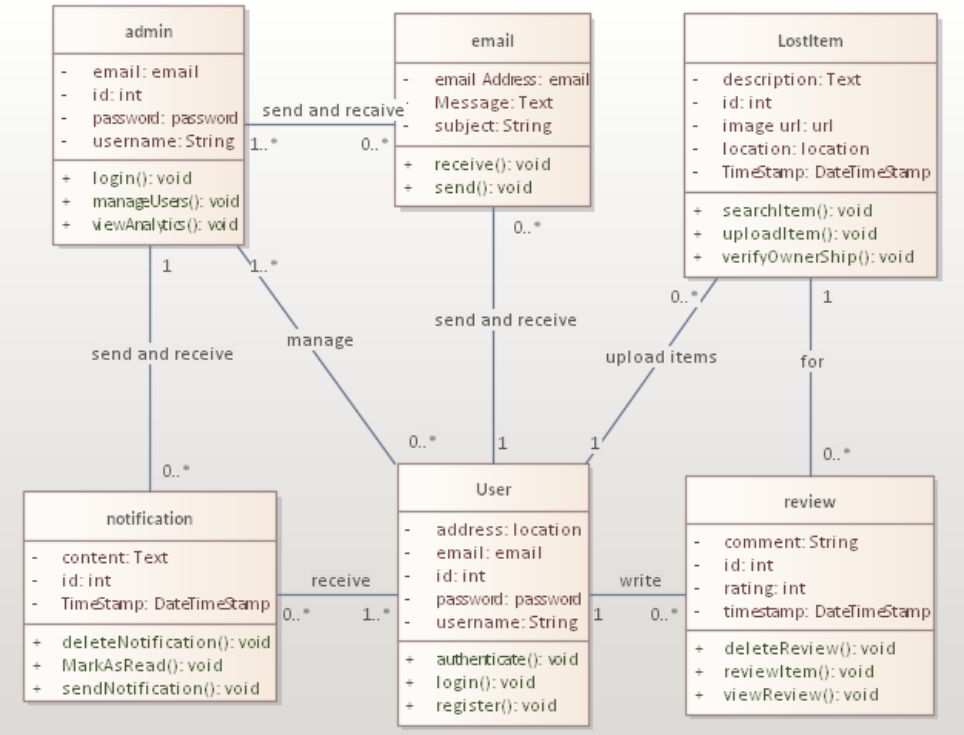

Figure 4: class Diagram

Class diagram Description

1. **User Class:**
    - **Attributes:**
        - id: int: Unique identifier for each user.
        - emailLocation: String: Stores the user's email address.
        - password: String: Stores the user's password.
    - **Methods:**
        - login(): void: Handles user login.
        - register(): void: Manages user registration.
        - updateProfile(): void: Allows users to update their profile information.
2. **Admin Class:**
    - **Attributes:**
        - id: int: Unique identifier for each admin.
        - emailLocation: String: Stores the admin's email address.
        - password: String: Stores the admin's password.
    - **Methods:**
        - authenticate(): void: Authenticates the admin's credentials.
        - manageUsers(): void: Admin-specific method for managing user accounts.
        - sendNotification(): void: Sends notifications to users.
3. **Review Class:**
    - **Attributes:**
        - id: int: Unique identifier for each review.
        - content: Text: Contains the content of the review.
        - rating: int: Represents the rating given by the user.
    - **Methods:**
        - deleteReview(): void: Allows users or admins to delete reviews.
        - viewReview(): void: Displays reviews for users or admins.
4. Email Class:
    - Attributes:
        - id: int: Represents a unique identifier for each email.
        - emailAddress: String: Stores the email address associated with the email.

        - password: String: Stores the password for email authentication.
    - Methods:
        - send(): void: Represents the action of sending an email.
        - receive(): void: Represents the action of receiving an email.

5. **Notification Class:**

    - **Attributes:**

        - id: int: Represents a unique identifier for each notification.
        - content: Text: Contains the content of the notification (e.g., message, alert).
        - timestamp: DateTimeStamp: Indicates when the notification was created.

    - **Methods:**

- sendNotification(): void: Handles the process of sending notifications to users.
- deleteNotification(): void: Allows users or administrators to delete notifications.

6. **LostItem Class:**

    - **Attributes:**
        - id: int: Represents a unique identifier for each lost item.
        - description: Text: Contains a description of the lost item (e.g., “red umbrella,” “black backpack”).
        - location: String: Stores information about where the item was lost (e.g., “Central Park,” “Subway Station”).
    - **Methods:**
        - searchItem(): void: Represents the action of searching for a lost item.
        - updateLocation(newLocation: String): void: Allows updating the location of the lost item.

DEPLOYMENT DIAGRAM

Welcome to the Deployment Diagram section, where we delve into the real-world setup of our software system. Just as architects design blueprints for buildings, deployment diagrams provide the infrastructure blueprint for our software. Here, we'll get a behind-the-scenes look at how our system's components are distributed across physical nodes, servers, and databases. This is where the virtual meets the physical, as we explore how our micro services, databases, and servers are deployed to ensure seamless communication and optimal performance. So, let's roll up our sleeves and dive into the nuts and bolts of our system's deployment, revealing the intricate web of connections that power our software ecosystem.

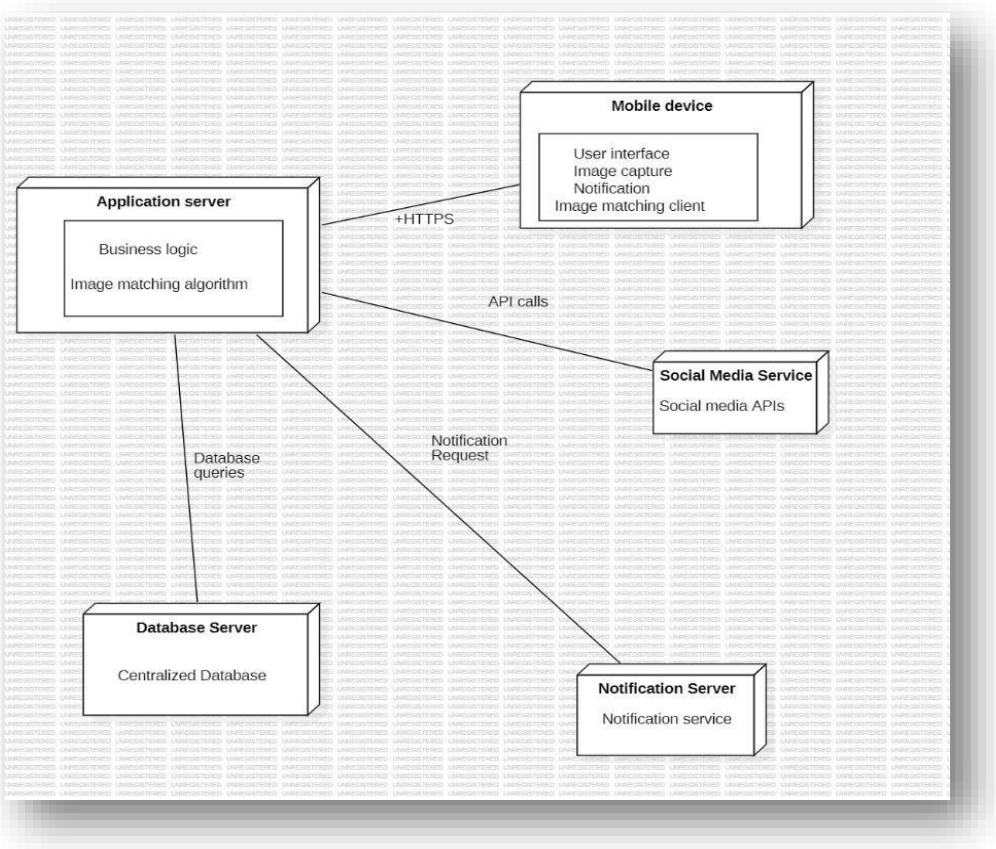

Figure 5: Deployment Diagram.

For the connection between the nodes,

- HTTPS: This connection represents secure communication between the mobile app and the application server. HTTPS ensures that data transferred between the client and server is encrypted and secure.
- Database queries: The application server queries the database server to store and retrieve images and metadata. This connection handles all database operations needed for the application to function.
- Notification Requests: The application server sends notification requests to the notification server. When a potential match is found, the application server instructs the notification server to send a push notification or email to the user. 18
- API Calls: The application server makes API calls to various social media platforms. This allows users to share information about their lost items on social media, engaging the community in the search process.

CONCLUSION.

The System Design phase marks a significant milestone in our journey towards developing a robust and user-centric software application. Throughout this phase, we've delved deep into the intricacies of our system, meticulously crafting its architecture, interactions, and components. From defining the system's context and boundaries to designing detailed class diagrams and sequence diagrams, we've laid the foundation for a scalable, efficient, and intuitive application. By embracing micro services architecture, we've modularized our system, enabling greater flexibility, scalability, and maintainability. Our class diagrams elegantly encapsulate the relationships and responsibilities of each component, fostering cohesion and modularity within our code base. As we transition to the next phase of development, armed with a comprehensive understanding of our system's design, we're poised to translate our vision into reality. With each line of code and architectural decision, we're one step closer to realizing a solution that not only meets the needs of our users but exceeds their expectations.

REFERENCES

Use Cases and Scenarios | Enterprise Architect User Guide (sparxsystems.com)

Unified Modeling Language (UML) description, UML diagram examples, tutorials and reference for all types of UML diagrams - use case diagrams, class, package, component, composite structure diagrams, deployments, activities, interactions, profiles, etc. (uml-diagrams.org)

Context Diagrams - GeeksforGeeks