**­­**

**REPUBLIQUE DU CAMEROON**

**PAIX-Travail-Patrie**

**MINISTRE DE L’ENSEIGNEMENT SUPERIEUR**

**FACULTE D’INGINERIE**

**ET TECHGNOLOGIE**

**REPUBLIC OF CAMEROON**

**Peace-Work-Fatherland**

**MINISTER OF HIGHER EDUCATION**

**FACULTY OF ENGINEERING**

**AND TECHNOLOGY**

**Database Design and Implementation of an Archival and Retrieval of Missing Object Application**

**COURSE TITLE: INTERNET PROGRAMMING AND MOBILE PROGRAMMING**

**COURSE CODE: 440**

**GROUP MEMBERS**

|  |  |
| --- | --- |
| **Name** | **Matricule** |
| NSEM CONFIDENT NJOCK | FE21A285 |
| NZEMTEJUH SYLVANUS | FE21A296 |
| NKWENKAM JENNIFER | FE21A279 |
| VTALAH DE GERAUD | FE21A329 |
| OLETIA AJONGAKUE SYNCLAIR | FE21A190 |

**COURSE FACILITATOR:** DR. Nkemeni Valery

Table of Contents

[Abstract 3](#_Toc169815390)

[INTRODUCTION 3](#_Toc169815391)

[LITERATURE REVIEW 4](#_Toc169815392)

[1. System Architecture 4](#_Toc169815393)

[2. Database Design 5](#_Toc169815394)

[I. Schema Design 5](#_Toc169815395)

[II. Reasons For Using Firebase 5](#_Toc169815396)

[III. Entity-Relationship (ER) Diagram 8](#_Toc169815397)

[Explanation Of Your ERD Diagram 8](#_Toc169815398)

[3. Firebase Implementation 10](#_Toc169815399)

[1. Data Synchronization 12](#_Toc169815400)

[4. Implementation Details 12](#_Toc169815401)

[1. Backend Implementation 12](#_Toc169815402)

[2. Challenges and Solutions 14](#_Toc169815403)

[5. Conclusion 14](#_Toc169815404)

[References 14](#_Toc169815405)

[Appendix 14](#_Toc169815406)

[5. Data Flow Diagram 15](#_Toc169815407)

# Abstract

This report details the database design and implementation of an application aimed at the archival and retrieval of missing objects using the Speeded Up Robust Features (SURF) algorithm for image matching. Leveraging Firebase Storage and Firebase Real-Time Database, this project aims to provide a seamless and efficient solution for tracking and finding lost items through user-uploaded images and associated metadata. The report covers the system architecture, database schema, algorithm integration, and implementation specifics, along with testing outcomes and future improvements.

# INTRODUCTION

The need for effective methods to track and retrieve missing objects has become increasingly important in our daily lives. Traditional methods are often time-consuming and inefficient. This project introduces a mobile application designed to assist users in locating lost items through an archival system that employs image matching algorithms, specifically SURF, to identify and retrieve objects based on user-uploaded images.

The primary objective of this project is to design and implement a robust database solution using Firebase services, ensuring efficient storage and retrieval of images and metadata. Additionally, the integration of the SURF algorithm aims to enhance the matching accuracy and retrieval speed.

# LITERATURE REVIEW

Numerous studies have explored various methods for object detection and retrieval, ranging from basic tagging systems to advanced machine learning algorithms. The SURF algorithm, known for its speed and accuracy in feature detection, has been widely used in computer vision applications. Firebase, with its real-time capabilities and ease of integration, presents an ideal backend solution for mobile applications requiring dynamic data handling.

Existing systems often rely on manual input and basic database management techniques. This project seeks to leverage the strengths of SURF and Firebase to create a more automated and efficient solution.

# System Architecture

The application architecture comprises three main components:

1. **Client-Side Application**: Built using XML, this component handles user interactions, including image uploads and search queries.
2. **Backend Services**: Powered by Firebase, the backend manages image storage, metadata storage, and real-time data synchronization.
3. **Image Matching Algorithm**: The SURF algorithm is implemented to process and match images based on feature detection.

The interaction between these components ensures a seamless user experience, from uploading images to retrieving matched results.

# Database Design

## Schema Design

The database design involves two main collections:

* **Firebase Storage:** For storing the actual images.
* **Firebase Real-Time Database:** For storing metadata associated with each image.

## Reasons For Using Firebase

Firebase is a popular choice for developing applications like the archival and retrieval of missing objects due to its comprehensive suite of tools and services designed to simplify and enhance app development. Here are several reasons why Firebase is particularly well-suited for such a system:

* **Real-time Database**
  + Real-time Updates: Firebase Realtime Database allows data to be synchronized across all clients in real-time. This means that any updates or changes to the database are instantly reflected in the app without the need for manual refreshes.
  + Scalability: It handles a large number of concurrent users and can scale automatically as the number of users grows, ensuring consistent performance.
* **Firebase Storage**
  + Easy File Storage and Retrieval: Firebase Storage is optimized for storing and retrieving user-generated content like images and videos. It provides a secure and robust solution for managing the images associated with the missing and found items.
  + Security Rules: You can define access controls on files using Firebase Security Rules, ensuring that only authorized users can view or upload files.
* **Authentication**
  + Secure User Authentication: Firebase Authentication provides easy-to-use SDKs and backend services to authenticate users using email, password, and various federated identity providers like Google, Facebook, and Twitter. This ensures that user data is secure and accessible only to authenticated users.
  + User Management: Firebase handles all aspects of user management, including password resets, email verifications, and account linking.
* Cloud Firestore
  + Flexible Data Model: Firestore, an alternative to Realtime Database, offers a more flexible and scalable NoSQL cloud database to store, sync, and query data for mobile and web apps.
  + Offline Support: Firestore provides robust offline support, allowing users to interact with the app and perform operations even when there is no internet connection. Changes are synced once the connection is restored.
* Cloud Functions
  + Serverless Functions: Firebase Cloud Functions allow you to run backend code in response to events triggered by Firebase features and HTTPS requests. This can be used to implement complex business logic, send notifications, and perform data transformations.
  + Automatic Scaling: Cloud Functions scale automatically based on the load, ensuring that your app can handle spikes in traffic without manual intervention.
* Cloud Messaging
  + Push Notifications: Firebase Cloud Messaging (FCM) enables you to send notifications to users to inform them about important events, such as matches for their lost items or updates on their posts.
  + Targeted Messaging: You can send targeted notifications to specific user segments based on their behavior and preferences.
* Analytics
  + Detailed Analytics: Firebase Analytics provides insights into user behavior and app performance. You can track events, user engagement, and retention, helping you to make informed decisions about app improvements.
  + Integration with Other Firebase Services: Analytics data can be used to trigger Cloud Functions, customize notifications, and more.
* Easy Integration
  + Seamless Integration: Firebase services are designed to work well together and are easily integrated into mobile and web applications. This reduces the development time and effort required to build a fully functional system.
  + Cross-Platform Support: Firebase supports iOS, Android, and web platforms, allowing you to build cross-platform apps with a consistent user experience.
* Cost-Effective
  + Free Tier: Firebase offers a generous free tier for many of its services, which is sufficient for small projects or initial development phases.
  + Pay-As-You-Go Pricing: Firebase uses a pay-as-you-go pricing model, allowing you to scale your costs with your usage. This is especially beneficial for startups and small businesses.
* Community and Support
  + Strong Community: Firebase has a large and active community of developers who contribute tutorials, libraries, and support, making it easier to find solutions and best practices.
  + Comprehensive Documentation: Firebase provides extensive documentation and support resources, helping developers to quickly learn and implement its features.

Overall, Firebase's comprehensive feature set, real-time capabilities, ease of integration, and scalability make it an ideal choice for developing an archival and retrieval system for missing objects.

## Entity-Relationship (ER) Diagram

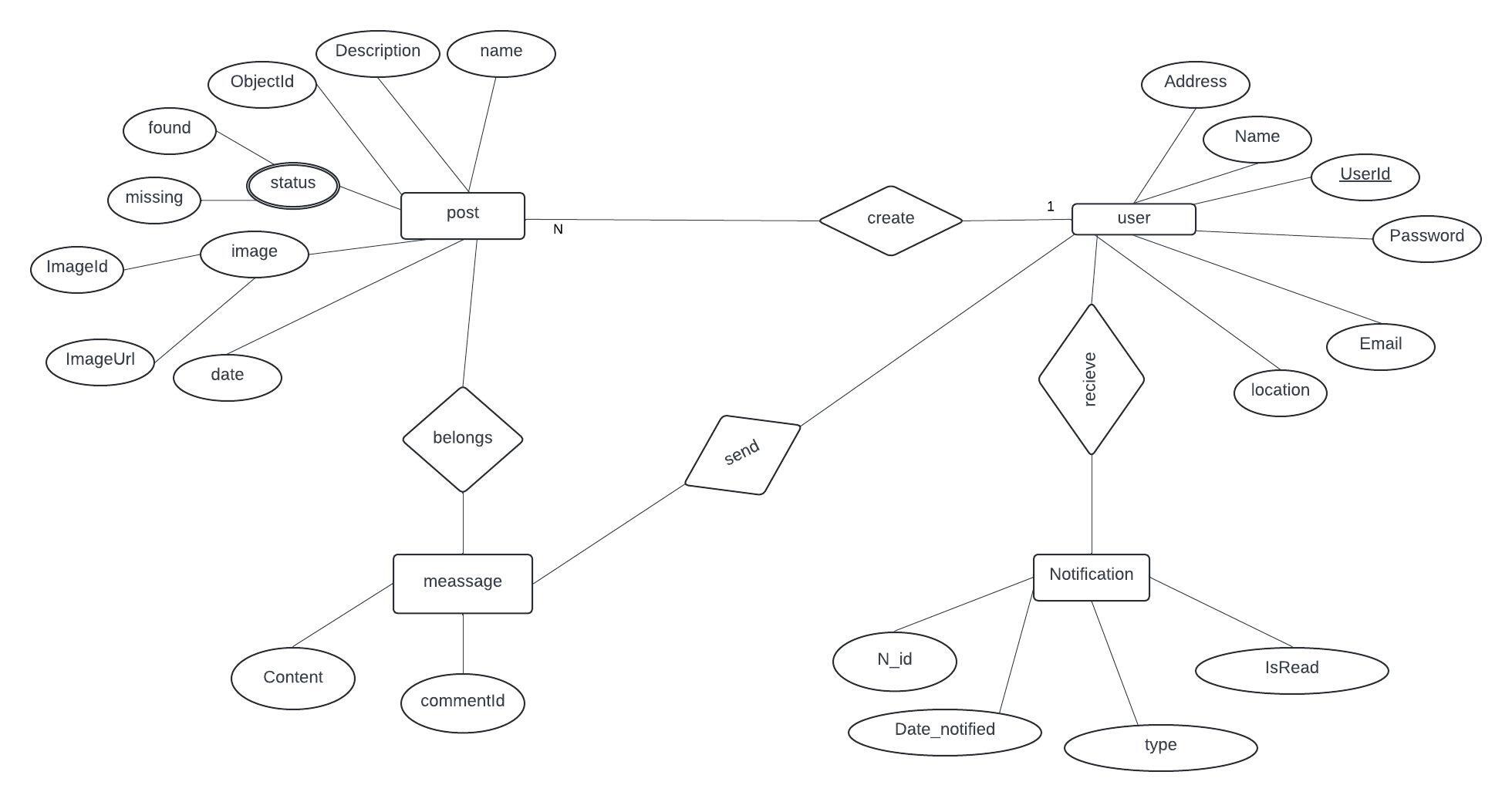


Figure 1. Entity Relationship diagram of an archival and retrieval of missing object app.

Explanation Of Your ERD Diagram:

**Entities and Attributes**

* **User**
  + **UserId**: The unique identifier for a user (Primary Key).
  + **Name**: The name of the user.
  + **Email**: The email address of the user.
  + **Password**: The password for the user's account.
  + **Address**: The address of the user.
  + **Location**: The current location of the user.
* **Post**
  + **ObjectId**: The unique identifier for an object (Primary Key).
  + **Description**: A description of the object.
  + **Status**: Indicates whether the object is found or missing.
  + **ImageId**: The unique identifier for an image.
  + **ImageUrl**: The URL of the image associated with the object.
  + **Date**: The date the post was created.
* **Message**
  + **CommentId**: The unique identifier for a message (Primary Key).
  + **Content**: The content of the message.
* **Notification**
  + **N\_id**: The unique identifier for a notification (Primary Key).
  + **Date\_notified**: The date the notification was sent.
  + **IsRead**: A boolean indicating whether the notification has been read.
  + **Type**: The type of notification.

**Relationships**

* **User and Post**:

A user can create many posts (one-to-many relationship). This is shown by the relationship "create" between User and Post, where one user can have multiple posts.

* **User and Notification**:

A user can receive many notifications (one-to-many relationship). This is shown by the relationship "receive" between User and Notification, where one user can receive multiple notifications.

* **Post and Message**:

A post can have many messages (one-to-many relationship). This is shown by the relationship "belongs" between Post and Message, where one post can have multiple messages.

* **User and Message:**

A user can send many messages (one-to-many relationship). This is shown by the relationship "send" between User and Message, where one user can send multiple messages.

**Explanation of Relationships**

* **Create (User-Post)**:

This relationship indicates that a user can create multiple posts. Each post contains details about a missing or found object, including its description, status, image, and date.

* **Receive (User-Notification)**: This relationship indicates that a user can receive multiple notifications. Each notification contains information such as the notification ID, date notified, read status, and type.
* **Send (User-Message):** This relationship indicates that a user can send multiple messages related to different posts. Each message contains content and a comment ID.
* **Belongs (Post-Message):** This relationship indicates that each post can have multiple messages associated with it, allowing users to comment or provide additional information about the post.

# Firebase Implementation

* Firebase Storage

Images are stored in a structured manner under a folder named "Images." Each image is named using a unique identifier to avoid conflicts.

* Firebase authentication

To store and authenticate user before they are granted access to our apps.

**Example Path:** `Images/imageID.jpg`

* Firebase Real-Time Database

Metadata for each image is stored in a JSON structure, which includes details such as the image URL, description, Status(found, missing), and upload date.­­­

Sample JSON Structure:

{

  "uploads": {

    "-O-jsqzj49Os95QLi1bi": {

      "date": "12/09/2024",

      "description": "this is a shoe",

      "imageUrl": "https://firebasestorage.googleapis.com/v0/b/findobject-64625.appspot.com/o/uploads%2F1718790347997.image%2Fjpeg?alt=media&token=0d64b22d-2a4a-4b52-b482-a90618d02bb9",

      "name": "shoes",

      "status": "Missing",

      "time": "14:33",

      "userId": "vUddPcTOIsSdHB84HyunwwFuftM2"

    },

    "-O-jxM5b6408ZTKaAk8D": {

      "date": "12/09/2024",

      "description": "this is a short sleeve shirt",

      "imageUrl": "https://firebasestorage.googleapis.com/v0/b/findobject-64625.appspot.com/o/uploads%2F1718791527803.image%2Fjpeg?alt=media&token=64fb8163-cf38-4721-995f-66e6d88fe8d1",

      "name": "dress",

      "status": "Found",

      "time": "13:45",

      "userId": "vUddPcTOIsSdHB84HyunwwFuftM2"

    },

    "found": {

      "-O-k2oGO2W2US3zPvhDW": {

        "date": "12/23/2023",

        "description": "vjghg",

        "imageUrl": "https://firebasestorage.googleapis.com/v0/b/findobject-64625.appspot.com/o/uploads%2F1718793220665.image%2Fjpeg?alt=media&token=cae0e75c-32ea-4d9d-98f3-3deed236d9f3",

        "name": "ggh",

        "status": "found",

        "time": "12:00",

        "userId": "vUddPcTOIsSdHB84HyunwwFuftM2"

      }

    },

    "missing": {

      "-O-k3zF58NlNsvzNqw8Y": {

        "date": "13/19/12",

        "description": "yxjjfj",

        "imageUrl": "https://firebasestorage.googleapis.com/v0/b/findobject-64625.appspot.com/o/uploads%2F1718793527494.image%2Fjpeg?alt=media&token=4cf2497b-6aba-4c41-a376-4dda1be5d64f",

        "name": "dudj",

        "status": "missing",

        "time": "12:50",

        "userId": "vUddPcTOIsSdHB84HyunwwFuftM2"

      }

    }

  }

}

## Data Synchronization

Data synchronization between Firebase Storage and Real-Time Database is achieved through Firebase functions that automatically update metadata upon image upload.

# Implementation Details

## Backend Implementation

Firebase functions are used to handle image uploads and metadata storage. Below are key code snippets:

1. Uploading Image to Firebase Storage:

private void uploadImageAndSaveData() {  
 if (imageUri != null) {  
 FirebaseStorage storage = FirebaseStorage.*getInstance*();  
 StorageReference storageReference = storage.getReference("uploads").child(System.*currentTimeMillis*() + ".jpg");  
  
 storageReference.putFile(imageUri).addOnSuccessListener(taskSnapshot ->  
 storageReference.getDownloadUrl().addOnSuccessListener(uri -> savePostData(uri.toString())))  
 .addOnFailureListener(e -> Toast.*makeText*(NewPostActivity.this, "Upload failed: " + e.getMessage(), Toast.*LENGTH\_SHORT*).show());  
 } else {  
 Toast.*makeText*(this, "No file selected", Toast.*LENGTH\_SHORT*).show();  
 }  
}

1. Storing Metadata in Firebase Real-Time Database:

FirebaseFirestore db = FirebaseFirestore.*getInstance*();  
  
 String name = editTextName.getText().toString().trim();  
 String date = editTextDate.getText().toString().trim();  
 String time = editTextTime.getText().toString().trim();  
 String description = editTextDescription.getText().toString().trim();  
 String status = ((RadioButton) findViewById(radioGroupStatus.getCheckedRadioButtonId())).getText().toString();  
  
 Map<String, Object> post = new HashMap<>();  
 post.put("name", name);  
 post.put("date", date);  
 post.put("time", time);  
 post.put("description", description);  
 post.put("status", status);  
 post.put("imageUrl", imageUrl);  
  
 db.collection("posts").add(post)  
 .addOnSuccessListener(documentReference -> Toast.*makeText*(NewPostActivity.this, "Post uploaded", Toast.*LENGTH\_SHORT*).show())  
 .addOnFailureListener(e -> Toast.*makeText*(NewPostActivity.this, "Error: " + e.getMessage(), Toast.*LENGTH\_SHORT*).show());  
}

1. API Integration

The app communicates with Firebase APIs to handle image uploads, metadata storage, and retrieval operations.

## Challenges and Solutions

Several challenges were encountered during the project:

* Data Synchronization: Ensuring real-time synchronization between Firebase services required careful implementation of Firebase functions.

# Conclusion

This project successfully demonstrates the feasibility of using Firebase and the SURF algorithm to create an efficient system for the archival and retrieval of missing objects. The implemented solution provides a user-friendly interface and robust backend, ensuring high performance and reliability. Future work includes exploring additional image matching algorithms and enhancing the system's scalability.

# References

1. Lizabeth Peterson. 2019, November, 04. Lost and Found app. <https://github.com/team-finders/lost-and-found-app>
2. Jayed Hossain J. 2019, May, 09. Lost and Found. https://www.slideshare.net/slideshow/lost-and-found-web-project/144729773

# Appendix

## Data Flow Diagram

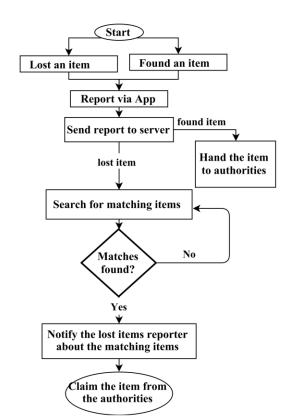


Figure 2. Lost and Found System Workflow

1. The flow diagram outlines a systematic approach to handling lost and found items using a mobile application.
2. Users report lost or found items via the app, and the server processes these reports.
3. Found items are handed to authorities, while lost items trigger a search for matches in the database.
4. If matches are found, the system notifies the user who lost the item, allowing them to claim their item from the authorities.

This process ensures that both lost and found items are efficiently reported, tracked, and retrieved, with the involvement of authorities to manage the physical handover of items.