

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 5: Database Design And Implementation

By:

Group 6

Supervisor:

Dr Nkemeni Valery
University of Buea

LIST OF PARTICIPANTS

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

Table Of Content

LIST OF PARTICIPANTS	i
INTRODUCTION.	1
OBJECTIVE	1
DATABASE DESIGN LIFECYCLE	2
SYSTEM REQUIREMENTS	2
Functional Requirements	2
Non-Functional Requirements.....	3
REQUIREMENTS ANALYSIS FOR ENTITIES AND RELATIONSHIP IDENTIFICATION	4
Summary of Requirements Related to Data Storage and Retrieval.....	4
Entity Set, Attributes, and Relationships Identification	4
Lost and Found items management.....	4
CONCEPTUAL DESING USING ER DIAGRAM	4
PHYSICAL DESIGN (COLLECTION STRUCTURE).	5
Collections.....	6
Collection Summary	8
INDEX DESIGN AND OPTIMIZATION	8
What is Indexing?.....	8
How Indexing Works?.....	8
Indexing Design Proper	9
DATABASE IMPLEMENTATION.....	10
Database Setup	11
Sign Up and Log In:	11
Create a New Cluster:	12
Deploy Cluster:	13
Creating The database:	13
Types of Collections.....	14
Standard Collection:	14
Capped Collection:	14
Cluster Index Collection:	14
Choosing an Appropriate Collection type.	14
Creating Indexes.	16
CONCLUSION.....	18
REFERNCES.....	19
GLOSSARY	20

INTRODUCTION.

In any complex application, a well-structured and efficiently managed database is crucial for ensuring data integrity, performance, and scalability. The database design and implementation phase of our lost and found items application aims to establish a robust foundation for handling the application's data requirements. Given the nature of our application, which involves user registrations, image uploads, real-time location tracking, notifications, and support services, the database must be designed to efficiently manage a diverse set of data types and relationships.

This section provides a comprehensive overview of the database design and implementation process. We will begin with a requirements analysis to identify the key data elements and their interrelationships. Following this, we will present the Entity-Relationship (ER) diagram, which visually represents the database structure. The ER diagram will be translated into a detailed schema definition, outlining the tables, fields, data types, and constraints.

Furthermore, we will discuss our indexing strategy to enhance query performance and the initial data population to set up essential records.

OBJECTIVE

The primary goal of this stage is to translate the system requirements into a coherent database structure that supports the application's functionalities seamlessly. This involves identifying key entities, defining their attributes, establishing relationships, and ensuring data normalization to eliminate redundancy and maintain consistency. Additionally, the database must be optimized for performance, with appropriate indexing strategies and security measures in place to protect sensitive user data.

WHY DESIGN ?

To reduce Redundancy.

Improve Data integrity

Ensure Completeness

DATABASE DESIGN LIFECYCLE

The database design lifecycle goes something like this:

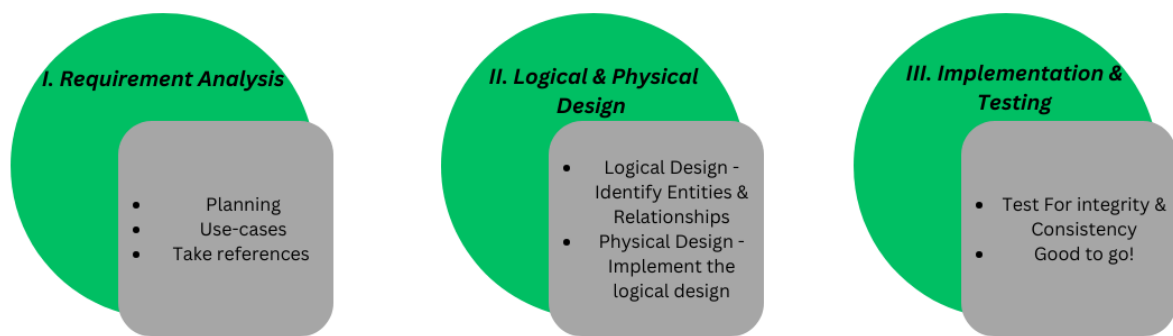


Figure 1: Database design LifeCycle.

SYSTEM REQUIREMENTS

To design and implement the database effectively, it's essential to clearly understand the system requirements. Here is a comprehensive list of requirements that the database should fulfil based on the various functionalities of the application:

Functional Requirements

1. User Management:

- Users can register with their personal information.
- Users can log in using their credentials.
- Users can update their profile information.
- Users can have different roles (e.g., admin, regular user).
- Users' activities are tracked for security and auditing.

2. Image Management:

- Users can upload images of lost items.

- Images can be captured via camera or selected from the gallery.
- Metadata (e.g., upload timestamp, user ID) is attached to images.
- Images are processed (e.g., compression, resizing) upon upload.

3. **Matching and Search:**

- Users can search for potential matches for lost items.
- The system uses advanced image matching algorithms for searches.
- Additional information can be provided to refine search results.
- Users can view and verify potential matches.
- Verified matches are updated with a "claimed" status.

4. **Notification System:**

- Users receive notifications (e.g., new matches, messages from admins).
- Notifications include details like content, date, and read/unread status.
- Users can mark notifications as read or delete them.

5. **Support System:**

- Users can access a help section for assistance.
- Users can submit support requests with details of their issues.
- Support requests are logged, categorized, and assigned to support agents.
- Users can communicate with support agents for issue resolution.

6. **Tracking System:**

- Users can track the location of matched items on a map.
- The system stores geolocation data for items.
- Users can view details about matched items on the map.

7. **Review and Verification:**

- Admins review potential matches for lost items.
- Admins verify ownership based on user-provided information.
- Verified items are marked as claimed, and notifications are sent to users.

8. **Email System:**

- The system sends emails to users (e.g., verification emails, notifications).
- Emails include details like subject, body, and sent timestamp.

Non-Functional Requirements

1. **Performance:**

- The database should handle a large number of concurrent users.
 - Image processing and search operations should be optimized for speed.
2. **Scalability:**
 - The database should be scalable to accommodate growing data volumes.
 - The system should support horizontal scaling.
 3. **Security:**
 - User credentials and sensitive data should be securely stored.
 - Data transmission should be encrypted.
 - Access control mechanisms should be in place to restrict unauthorized access.
 4. **Reliability:**
 - The database should ensure high availability.
 - Data integrity and consistency should be maintained at all times.
 5. **Usability:**
 - The system should provide a user-friendly interface for database interactions.
 - Error messages and feedback should be clear and helpful.
 6. **Maintainability:**
 - The database design should support easy maintenance and updates.
 - Documentation should be provided for database schemas and procedures.

REQUIREMENTS ANALYSIS FOR ENTITIES AND RELATIONSHIP IDENTIFICATION.

CONCEPTUAL DESIGN USING ER DIAGRAM

In this section, we will employ Entity-Relationship (ER) diagrams to visualize the conceptual design of our system. ER diagrams offer a clear and concise way to represent the essential entities, their attributes, and the relationships between them. By abstracting away implementation details, ER diagrams provide a high-level overview of the system's structure, facilitating communication and understanding among stakeholders.

Through ER diagrams, we will outline the core components of the system and illustrate how they interact with each other. This conceptual representation will serve as the foundation for further development, guiding the implementation of the system while allowing for flexibility

and scalability. The figure below shows the ER diagram of the system.

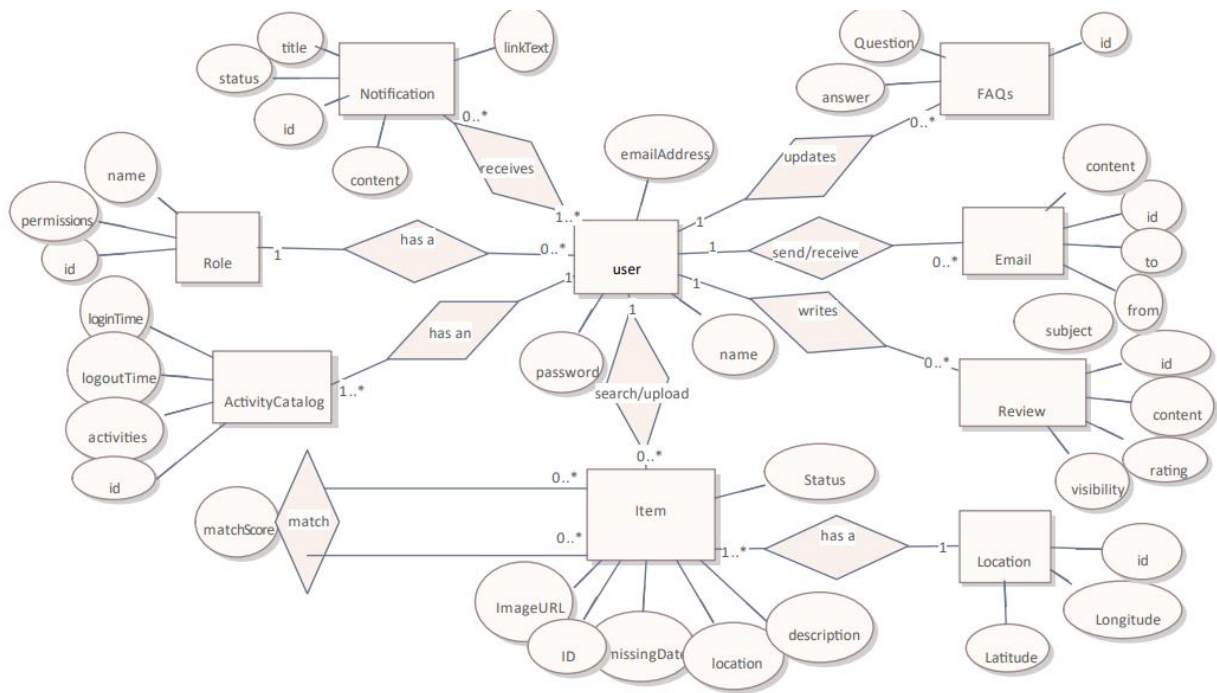


Figure 1: Entity Relationship Diagram

PHYSICAL DESIGN (COLLECTION STRUCTURE).

In this section, we transition from the conceptual design to the physical design of our database, tailored for MongoDB, a popular NoSQL database. Unlike traditional relational databases, MongoDB uses a flexible, document-oriented data model, which allows for the storage of data in JSON-like documents. This flexibility is particularly advantageous for applications with evolving requirements and complex, hierarchical data structures.

In the physical design phase, we will translate the entities, attributes, and relationships identified in our Entity-Relationship (ER) diagram into collections and documents. Each entity from the ER diagram will correspond to a collection, and the attributes of these entities will be represented as fields within these documents. Relationships between entities, which in relational databases are handled through foreign keys and join operations, will be managed using embedded documents and references.

This section will present the detailed NoSQL schema design for MongoDB, outlining the collections, their respective fields, and how relationships between different entities will be represented. This logical design serves as a blueprint for how data will be stored, organized, and accessed within MongoDB, ensuring the database structure supports the functionality and

requirements of our system while leveraging MongoDB's strengths in scalability, flexibility, and **performance**.

Collections

users

```
{
  "_id": ObjectId,
  "password": String,
  "name": String,
  "email": String,
  "role": {
    "roleId": ObjectId,
    "roleName": String,
    "permissions": [String]
  },
  "activityLog": [
    {
      "activityId": ObjectId,
      "loginTime": Date,
      "logoutTime": Date,
      "activities": String
    }
  ]
}
```

items

```
{
  "_id": ObjectId,
  "imageURL": String,
  "description": String,
  "location": String,
  "status": {
    "statusId": ObjectId,
    "statusDescription": String
  },
  "type": String, // "lost" or "found"
  "date": Date,
  "userId": ObjectId
}
```

matches

```
{
  "_id": ObjectId,
  "lostItemId": ObjectId,
  "foundItemId": ObjectId,
  "matchScore": Number,
}
```

```
"status": String // "pending claim", "claim approved", "further verification required"
}
```

Notifications

```
{
  "_id": ObjectId,
  "userId": ObjectId,
  "content": String,
  "status": String, // "read" or "unread"
  "title": String,
  "linkText": String
}
```

Reviews

```
{
  "_id": ObjectId,
  "userId": ObjectId,
  "content": String,
  "rating": Number,
  "visibility": Boolean
}
```

Locations

```
{
  "_id": ObjectId,
  "itemId": ObjectId,
  "latitude": Number,
  "longitude": Number,
  "timestamp": Date
}
```

Emails

```
{
  "_id": ObjectId,
  "to": String,
  "from": String,
  "subject": String,
  "message": String,
  "userId": ObjectId // for both sender and receiver if applicable
}
```

FAQs

```
{
  "_id": ObjectId,
  "question": String,
  "answer": String
}
```

Collection Summary

1. **users:** Stores user information, roles, and activity logs.
2. **items:** Stores details of lost and found items.
3. **matches:** Stores matching information between lost and found items.
4. **notifications:** Stores notifications sent to users.
5. **reviews:** Stores reviews submitted by users.
6. **locations:** Stores geolocation data for items.
7. **emails:** Stores emails sent and received by users.
8. **faqs:** Stores frequently asked questions and their answers.

INDEX DESIGN AND OPTIMIZATION

In this section, we will define and create necessary indexes to improve the performance of queries in our MongoDB database. The goal is to ensure that the indexes align with the most frequent query patterns and access paths. This strategic approach to indexing will enhance the overall efficiency and responsiveness of the system.

What is Indexing?

Indexing in the context of databases, including MongoDB, is a technique used to improve the performance of query operations. An index is a data structure that allows for fast retrieval of records from a database table or collection. Without indexes, a database must scan every document or record to find those that match a query, which can be slow and inefficient, especially with large datasets.

How Indexing Works?

Indexes work similarly to the index in a book, which helps you quickly locate information without having to read every page. In databases, an index stores the value of a specific field or set of fields in a structure that allows for quick lookup, such as a B-tree or hash table. When a query is executed, the database can use the index to find the requested data directly rather than performing a full collection scan.

Indexing Design Proper

User Management

Collection: users

- ✚ Index on Email
 - Reason: Unique identifier for login and user retrieval.
- ✚ Index on Role
 - Reason: Facilitates role-based queries for user filtering.
- ✚ Index on ActivityLog's loginTime
 - Reason: Efficient retrieval of user activity logs for auditing.

Lost and Found Items Management

Collection: items

- ✚ Index on ImageURL
 - Reason: Ensures uniqueness of URLs.
- ✚ Index on Status
 - Reason: Facilitates filtering based on item status.
- ✚ Index on Type and Date
 - Reason: Allows efficient filtering by item type and date.

Collection: matches

- ✚ Index on LostItemId and FoundItemId
 - Reason: Efficient retrieval of related lost and found items.
- ✚ Index on Status
 - Reason: Facilitates filtering based on match status.

Notification System

Collection: notifications

- ✚ Index on UserId
 - Reason: Efficient retrieval of notifications for a specific user.

✚ Index on Status

- Reason: Facilitates filtering based on read/unread status.

Review System

Collection: reviews

✚ Index on UserId

- Reason: Efficient retrieval of reviews written by a specific user.

✚ Index on Visibility

- Reason: Facilitates filtering based on review visibility status.

Tracking System

Collection: locations

✚ Index on ItemId

- Reason: Efficient retrieval of location data for a specific item.

✚ Geospatial Index on Latitude and Longitude

- Reason: Enables geospatial queries for location-based searches.

Support System

Collection: emails

✚ Index on UserId

- Reason: Efficient retrieval of emails associated with a specific user.

DATABASE IMPLEMENTATION.

The implementation phase marks a crucial step in the development process, where the theoretical database design is translated into a tangible and functional system. This phase is pivotal in bringing the envisioned database structure to life, ensuring that it meets the requirements and objectives outlined during the design phase.

Implementing the database involves converting the logical schema, defined through entity-relationship diagrams and data modelling, into a physical database with collections, fields,

indexes, and relationships. This transformation process is essential for laying the foundation of a robust and efficient database system capable of storing, managing, and retrieving data effectively.

The success of the implementation phase directly impacts the overall performance, reliability, and scalability of the database system. It is during this phase that various technical aspects, such as database setup, schema creation, index optimization, and security measures, are meticulously executed to ensure the system's functionality aligns with the project requirements.

Database Setup

Setting up the database environment involves several key steps to ensure the successful installation and configuration of the chosen database management system (DBMS). In our case, we'll focus on the setup process for **MongoDB**, a popular NoSQL database known for its flexibility and scalability. We took these steps to set up our database environment using MongoDB Atlas:

Sign Up and Log In:

Visit the MongoDB Atlas website and sign up for an account if you haven't already. Once you've signed up, log in to your MongoDB Atlas account using your credentials.

Log in to your account

Don't have an account? [Sign Up](#)



Or with email and password

Email Address

A text input field with a blue border. Inside the field, the email address 'kouete678@gmail.com' is typed. To the right of the text, there is a small green checkmark icon.

Figure 2: MongoDB sign-up page

Create a New Cluster:

After logging in, you'll be prompted to create a new cluster. A cluster is a group of MongoDB servers that work together to store your data. While in the clusters page, Click on ‘Create’ in the top-right corner of the screen.

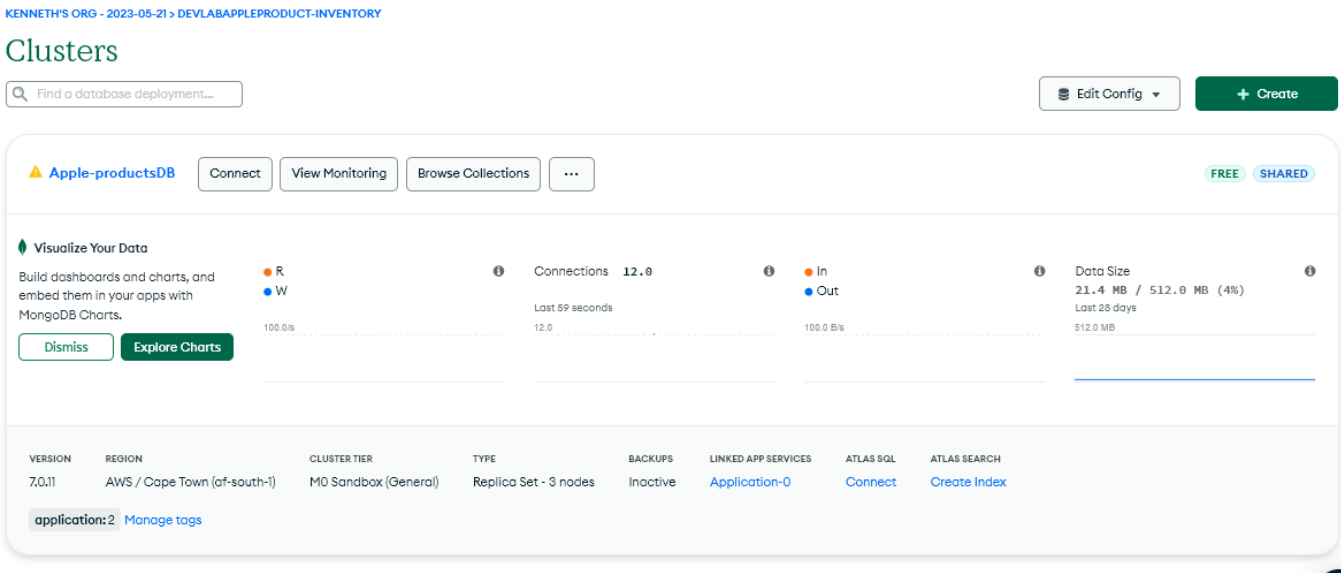


Figure 3: Create new cluster

Select Cloud Provider and Region: Choose your preferred cloud provider (e.g., AWS, Azure, Google Cloud) and the region where you want your cluster to be hosted. Consider factors like latency, compliance requirements, and data residency when selecting the region.

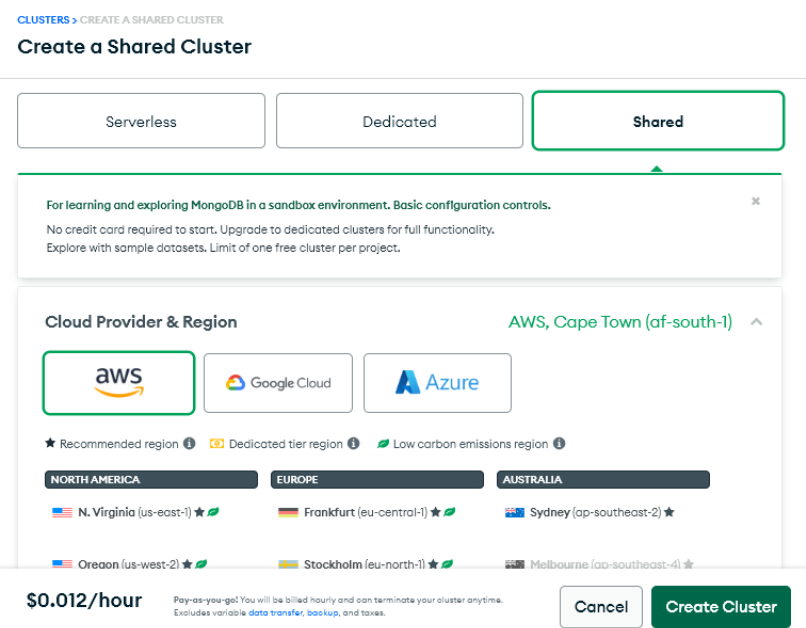


Figure 4: Cloud Provider Selection

Configure Cluster Settings:

Customize advanced settings for your cluster, such as cluster name, cluster tier, backup options, network access, and additional settings like encryption-at-rest and VPC peering.

Deploy Cluster:

Once you've configured your cluster settings, click the "Create Cluster" button to deploy your MongoDB cluster. MongoDB Atlas will provision the necessary resources on your chosen cloud provider and set up your cluster.

Creating The database:

after creating the cluster, we can now access it and create our Database. For this, we click on create new database.

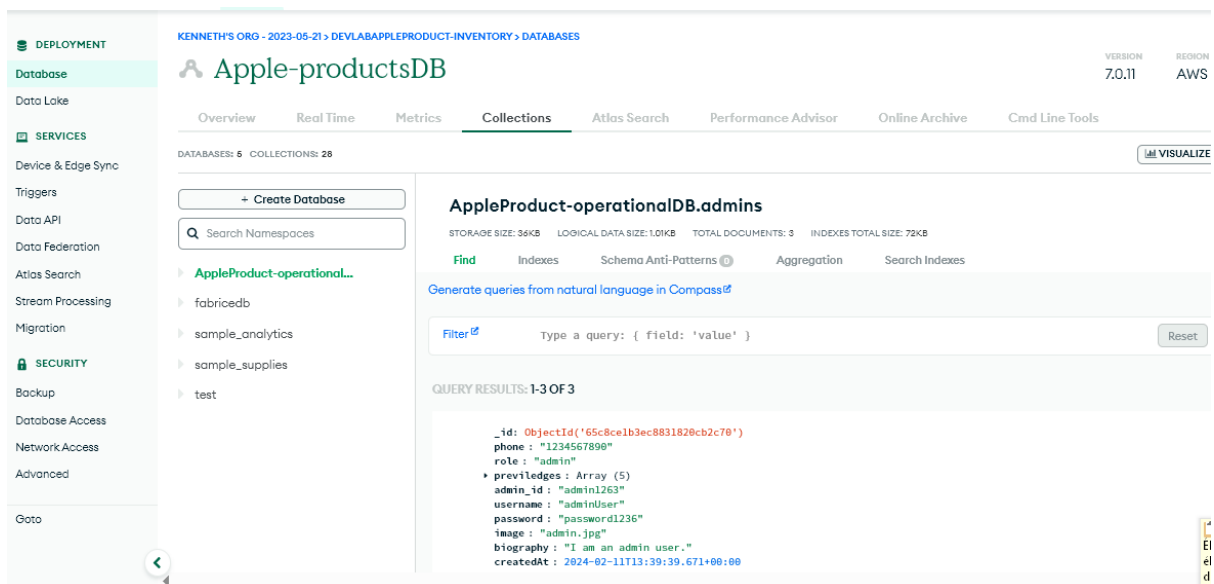


Figure 5: Create new database

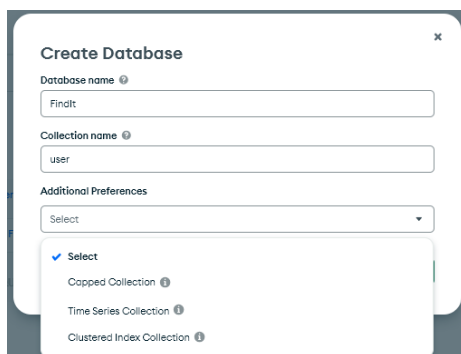


Figure 6: enter the information

Types of Collections

We observe 4 collection types:

Standard Collection:

A standard collection is the default collection type in MongoDB. It stores documents in an unordered manner and does not enforce any specific constraints or behaviours beyond basic MongoDB functionality.

Capped Collection:

A capped collection is a fixed-size collection that maintains insertion order and automatically removes the oldest documents when the collection reaches its maximum size.

Time Series Collection:

A time series collection is optimized for storing time-stamped data, such as sensor readings, telemetry data, or financial data indexed by time.

Cluster Index Collection:

A cluster index collection uses a clustered index to store data in a sorted order, which can improve query performance for range-based queries and data retrieval.

Choosing an Appropriate Collection type.

After analysing the data model of the application and considering the specific attributes and usage patterns of each collection, we have concluded that a standard collection would be the most suitable choice for our database implementation.

The decision to opt for standard collections is based on several factors:

1. **Data Characteristics:** The attributes within each collection do not exhibit characteristics that necessitate the use of capped collections, time series collections, or cluster index collections. The data is relatively static, with occasional updates, and does not have strict requirements for automatic data expiration or sorted storage.
2. **Query Patterns:** The query patterns for the application do not heavily rely on time-based queries or range-based queries that would benefit significantly from specialized

collection types. Most queries involve basic retrieval and filtering operations that can be efficiently handled by standard collections.

3. **Simplicity and Flexibility:** Standard collections offer simplicity and flexibility in data storage without imposing specific constraints or behaviours. This allows for easier management and maintenance of the database, especially in the absence of complex requirements that necessitate specialized collection types.
4. **Scalability:** Standard collections provide scalability and adaptability to future changes in data requirements or application functionality. They allow for seamless expansion and modification of the database schema as the application evolves over time.

Overall, the choice of standard collections aligns with the straightforward nature of the application's data model and query patterns, providing a pragmatic and efficient solution for storing and managing data in MongoDB Atlas.

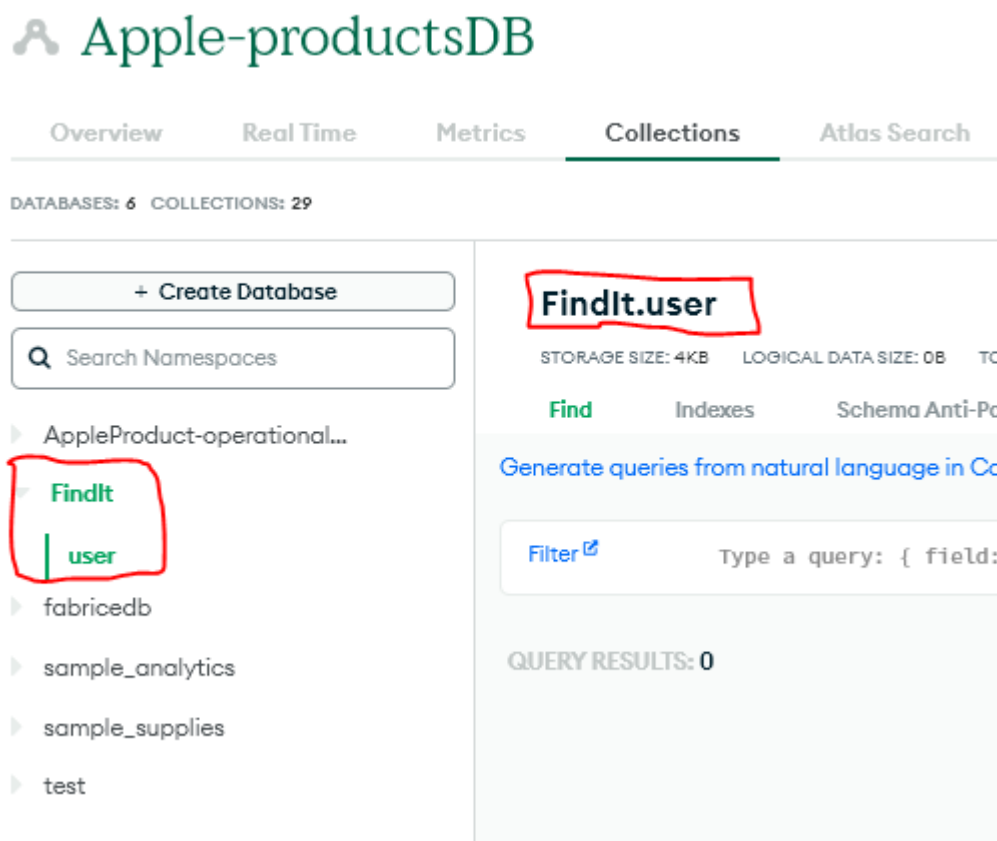


Figure 7: Database Created with user Collection

We then go for and create the other Collection.



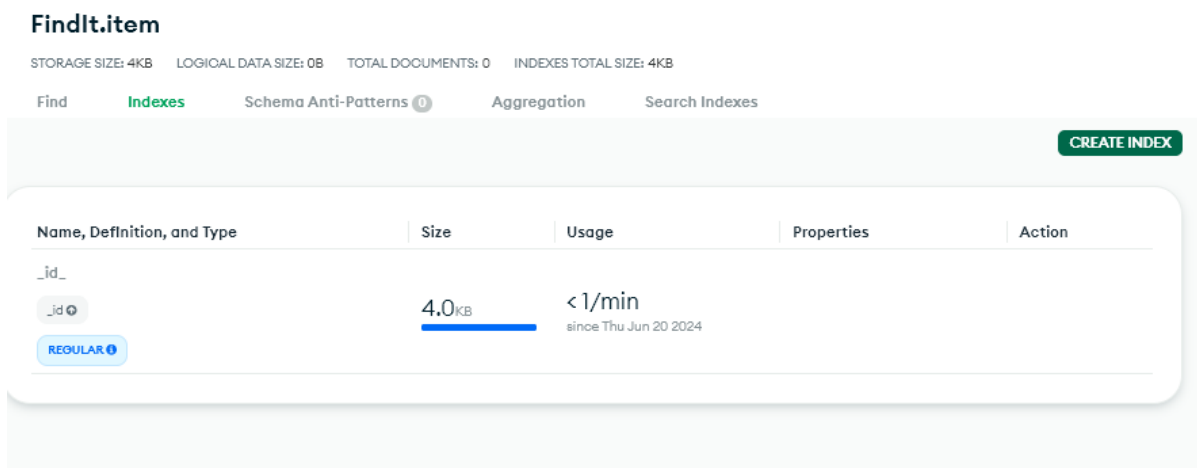
Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
FAQs	0	0B	0B	4KB	1	4KB	4KB
emails	0	0B	0B	4KB	1	4KB	4KB
item	0	0B	0B	4KB	1	4KB	4KB
locations	0	0B	0B	4KB	1	4KB	4KB
matches	0	0B	0B	4KB	1	4KB	4KB
notifications	0	0B	0B	4KB	1	4KB	4KB
reviews	0	0B	0B	4KB	1	4KB	4KB
user	0	0B	0B	4KB	1	4KB	4KB
fabricatedb	0	0B	0B	4KB	1	4KB	4KB

Figure 8. Collections created

Creating Indexes.

- Click on a particular collection where you want to create the index and click on indexes the click on ‘create index’

For example, let’s create the items collection indexes.



FindIt.item

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Find **Indexes** Schema Anti-Patterns ⓘ Aggregation Search Indexes

CREATE INDEX

Name, Definition, and Type	Size	Usage	Properties	Action
id { "_id" : 1 } REGULAR ⓘ	4.0KB	< 1/min since Thu Jun 20 2024		

Figure 9: Creating item’s indexes.

Click on ‘Create index’

FindIt.item

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Find **Indexes** Schema Anti-Patterns 0 Aggregation Search Indexes

Name, Definition, and Type	Size	Usage	Properties	Action
<div><div>_id</div><div><div>REGULAR</div></div></div>	4.0KB	< 1/min since Thu Jun 20 2024		
<div><div>imageURL_1_status_1</div><div><div>imageURL status</div><div>REGULAR</div></div></div>	4.0KB	< 1/min since Thu Jun 20 2024	<div>COMPOUND</div>	<div><div></div><div></div></div>

Figure 10. Index created.

We take the same procedure to create other indexes.

CONCLUSION

The database design and implementation phase of our lost and found items application is critical for establishing a solid foundation to support the system's functionalities effectively. By meticulously analysing the system requirements, identifying key entities, and defining their attributes and relationships, we have laid the groundwork for a well-structured and efficiently managed database.

The Entity-Relationship (ER) diagram serves as a visual representation of the database structure, guiding the translation of conceptual entities into concrete schema definitions. This ensures that the database accurately reflects the real-world entities and their interconnections, facilitating data integrity and consistency.

Furthermore, our indexing strategy aims to enhance query performance by strategically creating indexes on key fields, aligning with the most frequent query patterns and access paths. This approach ensures efficient data retrieval and responsiveness, optimizing the overall performance of the system.

As we move forward with the database implementation, including setting up the database environment, creating collections, defining indexes, and populating initial data, we remain committed to meeting the system's objectives of data integrity, performance, and scalability. With a robust database structure in place, we are well-equipped to support the application's operational requirements and deliver a seamless user experience.

REFERNCES.

<https://www.geeksforgeeks.org/database-design-in-dbms/>

<https://www.geeksforgeeks.org/nosql-database-design/>

https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

<https://www.mongodb.com/docs/atlas/support/>

GLOSSARY

- 1. Entity-Relationship (ER) Diagram:** A visual representation of the database structure that illustrates the entities, attributes, and relationships between entities in a system.
- 2. Database Design:** The process of defining the structure, organization, and relationships of data within a database system to meet the requirements of a specific application or system.
- 3. Database Implementation:** The phase of software development where the database design is translated into a physical database schema, including the creation of tables, indexes, and constraints.
- 4. Collection:** In MongoDB, a collection is a grouping of MongoDB documents. It is the equivalent of a table in relational databases.
- 5. Index:** A data structure that improves the speed of data retrieval operations on a database table or collection by allowing quick access to specific data based on indexed fields.
- 6. Capped Collection:** A fixed-size collection in MongoDB that maintains insertion order and automatically removes the oldest documents when the collection reaches its maximum size.
- 7. Cluster Index Collection:** A MongoDB collection that uses a clustered index to store data in a sorted order, which can improve query performance for range-based queries and data retrieval.
- 8. Time Series Collection:** A collection optimized for storing time-stamped data, such as sensor readings, telemetry data, or financial data indexed by time.
- 9. Entity:** A real-world object or concept that is represented in a database, typically as a table or collection, with its attributes and relationships.
- 10. Attribute:** A characteristic or property of an entity that describes its features or qualities, represented as columns in database tables or fields in MongoDB documents.
- 11. Relationship:** The association between entities in a database, representing how entities are connected or related to each other.
- 12. Indexing:** The process of creating indexes on database fields to improve query performance by enabling quick data retrieval based on indexed fields.
- 13. Data Normalization:** The process of organizing data in a database to eliminate redundancy and improve data integrity by reducing data duplication and dependency.
- 14. Data Redundancy:** The presence of duplicate or unnecessary data in a database, which can lead to inconsistencies and inefficiencies in data management.
- 15. Data Integrity:** The accuracy, consistency, and reliability of data stored in a database, ensuring that data remains valid and trustworthy throughout its lifecycle.
- 17. MongoDB Atlas:** A cloud-based database service provided by MongoDB that allows users to deploy, manage, and scale MongoDB databases in the cloud.