THE UNIVERSITY OF BUEA

P.O Box 63 Buea

Buea South West Region Cameroon

Tel: (237)233322760

REPUBLIC OF CAMEROON

PEACE -WORK-FATHERLAND

MINISTER OF HIGHER EDUCATION

FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER ENGINEERING

CEF474 : INTERNET AND MOBILE PROGRANMMING

DESIGN AND IMPLEMENTATION OF A MOBILE-BASED ARCHIVAL AND RETRIEVAL OF MISSING OBJECTS APPLICATION USING IMAGE MATCHING:

DATABASE DESIGN AND IMPLEMENTATION

COURSE INSTRUCTOR:

Dr NKEMENI VALERY

JUNE 2024

PRESENTED BY

NAME	MATRICULE
EKUNDIME GLEAN MAKOGE	FE21A433
DJOUMESSI LEKANE WENDY FORTUNE	FE21A173
INDAH RISCOBELLE MBAH	FE21A204
KEDJU PRECIOUS NGWE LEKUNZE	FE21A211
NGUEDIA JEATSA JOYCE GRACE	FE21A263

Table of Contents

I. Introduction

The efficient design and implementation of a database system is a critical component of modern software development. The primary goal is to create a system capable of handling large volumes of data, ensuring fast retrieval, maintaining transaction integrity, and supporting complex user interactions. This report covers the phases of requirement analysis, conceptual design, logical model, physical model, implementation, and testing, providing a comprehensive overview of the database development process.

II. Requirement Analysis

The requirement analysis phase is a fundamental step in the Database Development Life Cycle (DDLC). It involves systematically identifying, documenting, and analyzing the needs and expectations of stakeholders to ensure the successful design and implementation of a database system. In the context of our case study, the requirement analysis phase is particularly critical due to the complexity and interdependence of these components.

i. Functional Requirements Considered in the database

- **Data Storage:** The database must handle large volumes of document storage efficiently.
- **Data Retrieval:** Optimize for fast and accurate retrieval of documents.
- **Metadata Management:** Support metadata tagging for categorization and searchability.
- **Access Control:** Implement user roles and permissions in the database.
- **Version Control:** Store version histories within the database.
- **Transaction Records:** Maintain records of all payment transactions.
- **Chat Logs:** Store chat history for support interactions.
- **User Accounts:** Store user registration details, login credentials, and profile information.
- **Authentication Logs:** Maintain logs of authentication events for security auditing.

ii. Boundaries And Scope

a. Boundaries and scope

i. The system will handle document storage, retrieval, and management.
ii. The system will integrate with a payment gateway for processing transactions.
iii. The system will include a chatbox for real-time user support.
iv. The system will support user registration and authentication.

b. Parameters and Limitations

i. **User Access**

- Only authorized users can access sensitive documents.

ii. **Data Limits:**

- Maximum size of documents that can be uploaded.
- Maximum number of documents a user can store, depending on their subscription plan.

iii. **Performance:**

- The system should handle concurrent users efficiently.
- Ensure fast response times for document retrieval and payment processing.

iv. **Security:**

- Encrypt sensitive data in transit and at rest.
- Implement robust authentication and authorization mechanisms.
- Optionally, provide two-factor authentication for added security.

v. **Compliance:**

- Adhere to data protection regulations (e.g., GDPR).
- Ensure payment system compliance with PCI DSS.

III. Design Phase

2.1 Conceptual Design

The conceptual model identifies the high-level entities within the system without going into detail about their attributes or relationships

a. Entities

- User
- Object
- Category
- Transaction
- ChatSession
- Notification
- Feedback
- AuditLog

b. Relationships

- **User** and **Document**
  - One-to-Many (A user can upload multiple documents)
- **User** and **Transaction**
  - One-to-Many (A user can have multiple transactions)
- **User** and **ChatSession**
  - One-to-Many (A user can have multiple chat sessions)
- **User** and **Notification**
  - One-to-Many (A user can have multiple notifications)
- **ChatSession** and **Feedback**
  - One-to-One (A chat session can have one feedback entry)

- **User** and **Feedback**
    - One-to-Many (A user can provide multiple feedback entries)
- **User** and **AuditLog**
    - One-to-Many (A user can have multiple audit log entries)
- **Document** and **Category**
    - Many-to-One (A document can belong to one category)
- **ChatSession** and **User**
    - Many-to-One (A chat session can involve a user and an agent)

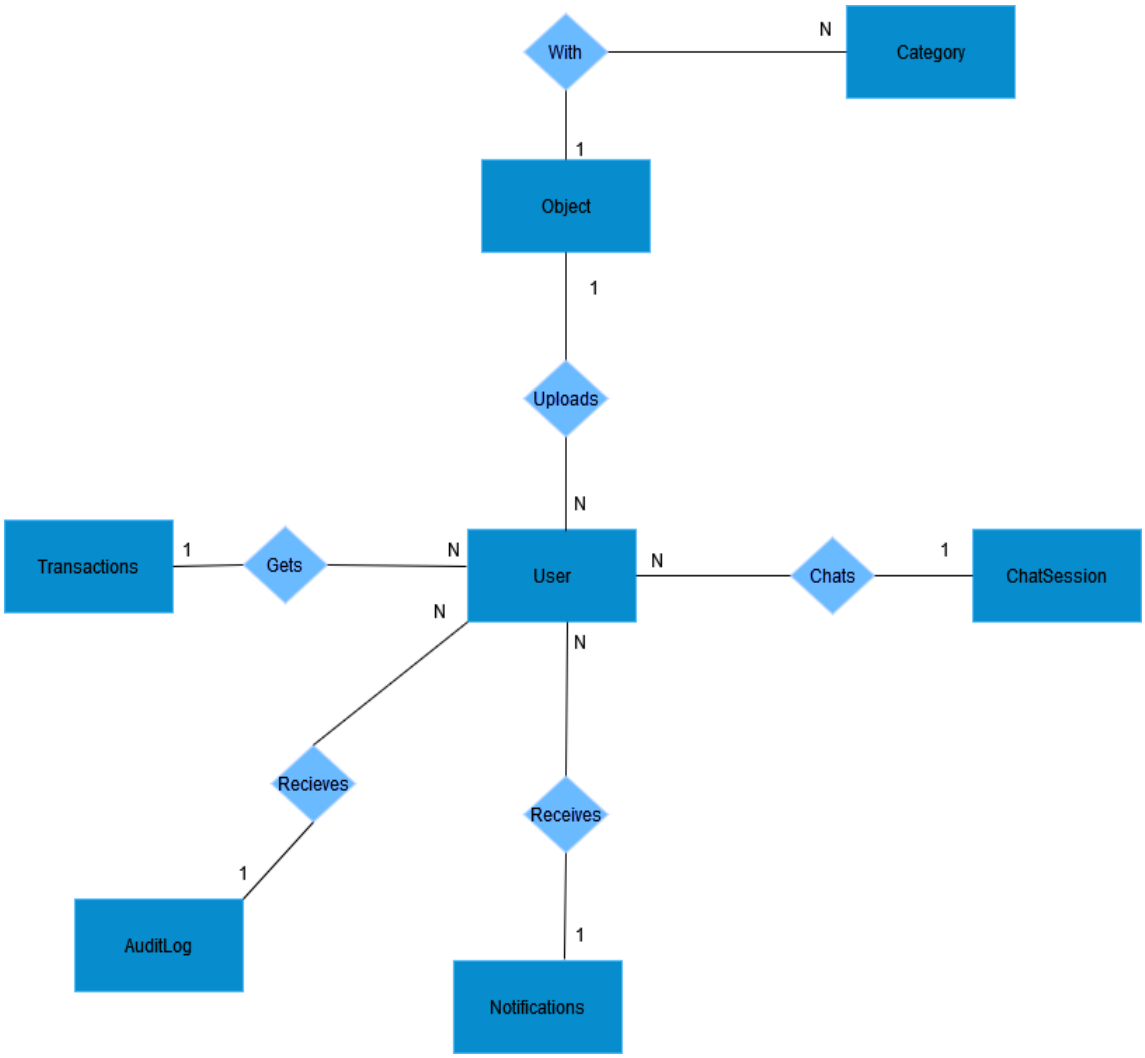


Fig 1.0 ER Diagram

2.2 Logical Models

The logical model expands the conceptual model by adding attributes, defining relationships, specifying primary keys, establishing foreign keys, and applying normalization techniques.

c. Entities And Attributes

i. User

- UserID (Primary Key)
- Username
- Email
- PasswordHash
- RegistrationDate
- ProfilePicture
- PhoneNumber

ii. Object

- ObjectID (Primary Key)
- CategoryID (Foreign Key)
- UserID (Foreign Key)
- Title
- Content
- Metadata
- UploadDate
- LastModified

iii. Entity: Transaction

- TransactionID (Primary Key)
- UserID (Foreign Key)
- Amount
- TransactionDate
- PaymentMethod
- Status (e.g., Completed, Pending, Failed)

iv. Entity: ChatSession

- ChatSessionID (Primary Key)
- FinderUserID (Foreign Key)
- FounderUserID (Foreign Key)
- StartTime
- EndTime
- ChatLog
- FeedbackID (Foreign Key)

i. Entity: Notification

- NotificationID (Primary Key)
- UserID (Foreign Key)
- Message
- IsRead
- CreatedAt

i. Entity: Feedback

- FeedbackID (Primary Key)

- UserID (Foreign Key)
- ChatSessionID (Foreign Key)
- Rating
- Comments
- CreatedAt

i. Entity: AuditLog

- AuditLogID (Primary Key)
- UserID (Foreign Key)
- Action
- Timestamp
- Details

i. Entity: Category

- CategoryID (Primary Key)
- CategoryName
- Description

d. Normalization:

- **First Normal Form (1NF):** Ensure each table has a primary key and no repeating groups.
- **Second Normal Form (2NF):** Ensure all non-key attributes are fully functionally dependent on the primary key.
- **Third Normal Form (3NF):** Ensure no transitive dependencies (i.e., non-key attributes depend only on the primary key).

Applying Normalization:

i. **User Table:**
   - Already in 1NF with primary key UserID.
   - No partial dependencies as UserID is the primary key, thus in 2NF.
   - No transitive dependencies, thus in 3NF.

ii. **Document Table:**
   - Already in 1NF with primary key DocumentID.
   - UserID and CategoryID fully determine other attributes, thus in 2NF.
   - No transitive dependencies, thus in 3NF.

iii. **Transaction Table:**
   - Already in 1NF with primary key TransactionID.
   - UserID fully determines other attributes, thus in 2NF.
   - No transitive dependencies, thus in 3NF.

iv. **ChatSession Table:**
   - Already in 1NF with primary key ChatSessionID.
   - UserID and AgentID fully determine other attributes, thus in 2NF.
   - No transitive dependencies, thus in 3NF.

v. **Notification Table:**
   - Already in 1NF with primary key NotificationID.
   - UserID fully determines other attributes, thus in 2NF.
   - No transitive dependencies, thus in 3NF.

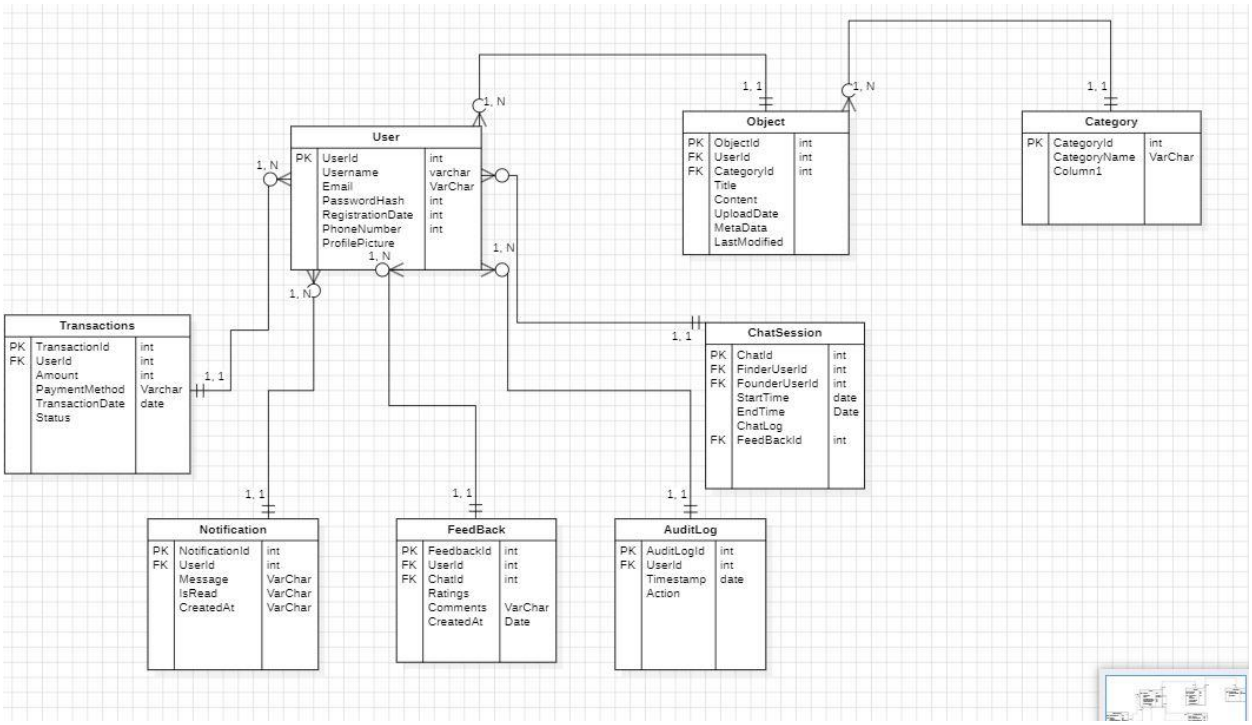
vi. **Feedback Table:**
   - Already in 1NF with primary key FeedbackID.
   - UserID and ChatSessionID fully determine other attributes, thus in 2NF.
   - No transitive dependencies, thus in 3NF.

vii. **AuditLog Table:**
   - Already in 1NF with primary key AuditLogID.
   - UserID fully determines other attributes, thus in 2NF.
   - No transitive dependencies, thus in 3NF.

viii. **Category Table:**

- Already in 1NF with primary key CategoryID.
- No partial dependencies as CategoryID is the primary key, thus in 2NF.
- No transitive dependencies, thus in 3NF.

Fig 2.0 Relational Schema



2.3 Physical Model

Implementation of Logical Design

The physical model translates the logical design into a structure suitable for MongoDB, a NoSQL document-oriented database.

Software Systems

The system will utilize MongoDB as the database management system, along with Node.js and Express.js for backend development, and Angular or React for frontend development.

DBMS

- **DBMS:** MongoDB (NoSQL document-oriented database)

Reasons for choosing Mongodb

1. Flexible Schema Design

NoSQL (MongoDB): Schemaless hence allowing for a flexible data model which is beneficial for our project requires frequent changes to the data structure, hence eliminating the need for complex schema migrations.

2. Document-Oriented Storage

NoSQL (MongoDB): Stores data in JSON-like documents, which can contain nested structures and arrays. This is ideal for storing complex data such as user profiles, documents with metadata, and chat logs.

3. Scalability

NoSQL (MongoDB): Designed for horizontal scalability, allowing the database to distribute data across multiple servers or clusters (sharding). This is crucial for handling large volumes of data and high traffic.

4. Performance

NoSQL (MongoDB): Optimized for high-throughput read and write operations. The ability to scale out horizontally ensures that performance remains high even under heavy loads.

5. Indexing and Querying

NoSQL (MongoDB): Offers rich indexing options and a powerful aggregation framework for efficient querying and data processing. Indexes can be created on any field, including nested fields within documents.

6. Handling Unstructured Data

NoSQL (MongoDB): Excels at handling unstructured or semi-structured data, making it a good fit for storing diverse data types without requiring a predefined schema.

7. Real-Time Data and Logging

- **NoSQL (MongoDB)**: Capable of handling real-time data streams and logging efficiently due to its flexible schema and high write throughput. This is particularly useful for storing chat logs and notifications.

Collections

Considerations for Physical and Practical Aspects of Database Design

- **Document Structure:** Use embedded documents or references (normalized data model) based on access patterns and query requirements.
- **Indexes:** Create indexes on fields used for querying (e.g., userID, categoryID) to improve query performance.
- **Sharding:** Plan for sharding if scalability requirements dictate distributing data across multiple MongoDB instances.
- **Backup and Recovery:** Implement regular backups using MongoDB tools and ensure backups are stored securely.
- **Security:** Configure MongoDB authentication and authorization to control access to databases and collections.
- **Scalability:** MongoDB's horizontal scalability allows for adding more nodes to handle increasing data volumes and user load.

2.4 Implementation

Putting the Designed Database into Action

i. **Database Creation**
ii. **Collections Creation**
iii. **Index Creation**.

Integration Testing Using Different Datasets

i. **Data Population:** Populate the MongoDB collections with test data to simulate different scenarios (e.g., user interactions, document uploads, payment transactions).
ii. **Functional Testing:** Conduct functional tests to ensure that CRUD operations work correctly for each collection.
iii. **Performance Testing:** Evaluate database performance under load using tools like Apache JMeter or custom scripts to measure response times and scalability.

Converting Data Format for Efficient Processing

i. **Data Conversion:** Ensure data is stored in MongoDB's BSON (Binary JSON) format, which is efficient for storage and retrieval operations.
ii. **Data Migration:** If migrating from a different database system, use MongoDB's import/export tools or custom scripts to convert data into BSON format.
iii. **Optimization:** Optimize queries and data access patterns based on MongoDB's document-oriented nature to leverage its strengths in handling nested data and dynamic schemas.

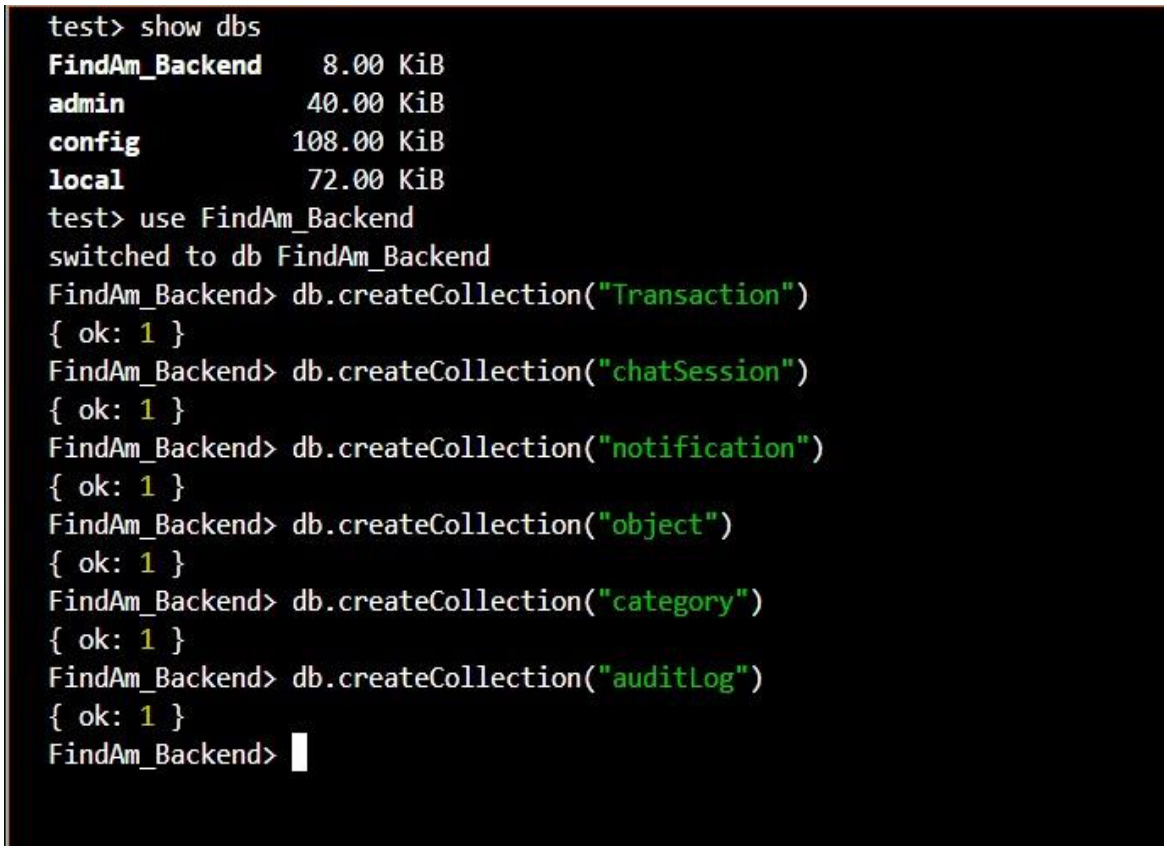
2.5 Testing

Examination of the Database for Errors

Testing the MongoDB database involves verifying its functionality, performance, and data integrity. Here's how the testing process can be structured:

1. **Functional Testing:**
    - **CRUD Operations:** Test create, read, update, and delete operations for each collection (e.g., users, documents, transactions) to ensure they behave as expected.

Create



```
FindAm_Backend> db.User.insertOne({_id: ObjectId(), userName: "precy_k", email: "precyk@gmail.com", p
asswordHash: "hashed_password", profilPicture: "path_to_profilePicture", phoneNumber: 671-234-567, re
gisterDate: new Date()})
{
  acknowledged: true,
  insertedId: ObjectId('667487a3d0c8ab82ad90defe')
}
```

Read

```
FindAm_Backend> db.User.find({userName:{$in:[ "precy_k"]}})
[
  {
    _id: ObjectId('667487a3d0c8ab82ad90defe'),
    userName: 'precy_k',
    email: 'precyk@gmail.com',
    passwordHash: 'hashed_password',
    profilPicture: 'path_to_profilePicture',
    phoneNumber: -130,
    registerDate: ISODate('2024-06-20T19:48:51.759Z')
  }
]
```

Update

```
FindAm_Backend> db.User.updateOne({userName: "precious"}, {$set:{email: "indah@gmail.com"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Delete

```
FindAm_Backend> db.User.deleteOne({userName: "precious"})
{ acknowledged: true, deletedCount: 1 }
```

- **Query Testing:** Execute various queries (e.g., find by userID, filter by status) to verify data retrieval accuracy.
- **Index Testing:** Validate that indexes are utilized effectively by reviewing query execution plans and performance metrics.

2. **Performance Testing:**

- **Load Testing:** We intend to simulate concurrent user interactions using tools like Apache JMeter to assess database performance under realistic load conditions.
- **Scalability Testing:** Evaluate database performance as data volume increases by gradually increasing the workload and monitoring response times.
- **Benchmarking:** Compare performance metrics (e.g., throughput, latency) against defined service level agreements (SLAs) to ensure they are met.

3. **Data Integrity Testing:**
    - **Validation Rules:** We ensured that data validation rules (e.g., required fields, data types) are enforced correctly during data insertion and updates.
    - **Referential Integrity:** We verified our references between collections (e.g., userID in transactions referencing users) are maintained properly.
    - **Error Handling:** We intend to test error scenarios (e.g., invalid queries, network failures) to ensure the database handles exceptions gracefully without data loss or corruption.

Primary Objective is to Validate the Database

The primary objective of testing is to validate that the MongoDB database meets functional requirements, performs efficiently under expected load, and maintains data integrity without errors or inconsistencies.

Quality Assurance Measures

To ensure high-quality and reliable database operation:

- **Code Reviews:** We intend conducting reviews of MongoDB queries and database interaction code to identify potential performance bottlenecks or security vulnerabilities.
- **Automated Testing:** We intend implementing automated unit tests and integration tests for database interactions to validate behavior across different scenarios.

- **Monitoring and Alerts:** Set up monitoring tools (e.g., MongoDB Atlas monitoring, CloudWatch) to monitor database performance metrics in real-time and configure alerts for anomalies.
- **Security Audits:** Perform regular security audits to identify and mitigate potential risks, including access control and data encryption.

IV. Conclusion

This comprehensive approach ensures a scalable, secure, and efficient database system that meets the diverse needs of the archival and retrieval system.

V. References

https://www.mongodb.com/docs/atlas/getting-started/

https://docs.staruml.io/working-with-additional-diagrams/entity-relationship-diagram

https://www.youtube.com/watch?v=c2M-rlkkT5o

Database Design - 2nd Edition by Adrienne Watt, Nelson Eng

The Morgan Kaufmann Series in Data Management Systems Series Editor: Jim Gray, Microsoft Research