UNIVERSITY OF BUEA

****

**DEPARTMENT: Computer Engineering**

**SUB DEPARTEMENT: Software Engineering**

**COURSE: CEF 440 Mobile Programming and Internet Programming**

**Design of a Database for Road Sign And a Road State Mobile Notification Application**

PRESENTED BY

|  |  |
| --- | --- |
| FE21A292 | NYENTY PAUL EGBE |
| FE21A274 | NKAFU VANIC ASONG |
| FE21A238 | METIEGE OTILI-FAVOUR MECHANE |
| FE21A156 | CHAM KANG NIBA |
| FE21A141 | ASONGLEFAC ZINGWA LIONEL |

### Supervisor: Dr. Nkemene Valery

**Table of Contents**

[1. Introduction 3](#__RefHeading___Toc4836_4116143598)

[2. Requirements 3](#__RefHeading___Toc4838_4116143598)

[3. Schema Design 4](#__RefHeading___Toc4840_4116143598)

[3.1. Users Collection 4](#__RefHeading___Toc4842_4116143598)

[3.2. Posts Collection 4](#__RefHeading___Toc4844_4116143598)

[3.3. Comments Collection 5](#__RefHeading___Toc4846_4116143598)

[3.4. Notifications Collection 5](#__RefHeading___Toc4848_4116143598)

[4. Set Up Indexes 5](#__RefHeading___Toc4850_4116143598)

[5. Handle Relationships 5](#__RefHeading___Toc4852_4116143598)

[6. Data Validation and Schema Design 5](#__RefHeading___Toc4854_4116143598)

[7. Handling Offline Data 6](#__RefHeading___Toc4856_4116143598)

[8. Scalability and Performance 6](#__RefHeading___Toc4858_4116143598)

[9. Security 6](#__RefHeading___Toc4860_4116143598)

[10. Example MongoDB Schema Design 6](#__RefHeading___Toc4862_4116143598)

[11. Final Steps 7](#__RefHeading___Toc4864_4116143598)

[12. Conclusion: 7](#__RefHeading___Toc4866_4116143598)

[13. References: 8](#__RefHeading___Toc4868_4116143598)

### 1. Introduction

Designing a MongoDB database for a mobile app involves several key steps, including understanding the app's requirements, designing an appropriate schema, and setting up the database for scalability and performance.

### 2. Requirements

* **User Management**:

User profiles, authentication, and authorization to be store.

Firebase Authentication will be used.

* **App Features**:

Specific functionalities like posts, comments, likes, notifications, and search.

MongoDB collections will be used to organize related data.

* **Data Relationships**:

Define how different data entities(users, posts, comments, likes) relate to each other.

One-to-many, many-to-one, many-to-many, etc.

Appropriate modeling techniques was used like embedding, referencing based on relationships. Also queries were optimized for fetching data by:

1) Creating indexes to support related data.

e.g db.posts.createIndex({ author\_name: 1 });

2) Limiting the number of query results

e.g db.posts.find().sort({ timestamp: -1 }).limit(10);

3) Using projections

db.posts.find({}, { timestamp: 1, title: 1, author: 1 }).sort({ timestamp: -1 });

4) Using $hint to select a particular index for performance testing.

e.g db.posts.find().hint({ timestamp: 1 });

- Only necessary fields will be returned to minimize data transfer.

- The fields needed will be specified.

* **Performance**:

To improve performance read/write operations were optimized. This can be achieved by using indexes, limiting query results, using projections, and $hint.

Also sharding(horizontal participationing) for large datasets will be used.

* **Scalability**: Handling increasing data and user load.

Data will be distributed across multiple MongoDB instances or clusters.

Performance will be monitored and adjusted accordingly.

* **Offline Capabilities**:

Implementing local data storage for offline usage.

Syncing data when the device is reconnects with the server.

Tools like MongoDB Realm Sync for seamless offline-first experience.

### 3. Schema Design

MongoDB is a NoSQL database and supports flexible schema design. Here are some common patterns:

#### 3.1. Users Collection

Each user document stores user-related information.

json

{

"\_id": "user\_id",

"username": "johndoe",

"email": "johndoe@example.com",

"password": "hashed\_password",

"profilePicture": "url\_to\_image",

"createdAt": "timestamp",

"updatedAt": "timestamp"

}

### 3.2. Posts Collection

Stores posts made by users.

json

{

"\_id": "post\_id",

"userId": "user\_id",

"content": "This is a post",

"media": ["url\_to\_image\_or\_video"],

"createdAt": "timestamp",

"updatedAt": "timestamp",

"likes": ["user\_id\_1", "user\_id\_2"],

"comments": [

{

"userId": "user\_id",

"comment": "Nice post!",

"createdAt": "timestamp"

}

]

}

### 3.3. Comments Collection

Alternatively, comments can be stored in a separate collection if they are expected to be numerous.

json

{

"\_id": "comment\_id",

"postId": "post\_id",

"userId": "user\_id",

"comment": "Nice post!",

"createdAt": "timestamp"

}

### 3.4. Notifications Collection

Stores notifications for users.

json

{

"\_id": "notification\_id",

"userId": "user\_id",

"type": "like/comment/follow",

"message": "User X liked your post",

"isRead": false,

"createdAt": "timestamp"

}

### 4. Set Up Indexes

Indexes are crucial for performance. Common indexes include:

* **Users**: username, email (for authentication and user lookup).
* **Posts**: userId, createdAt (for fetching user posts and sorting).
* **Comments**: postId, createdAt (for fetching comments of a post).

### 5. Handle Relationships

MongoDB's document model allows you to embed or reference related data.

* **Embedding**: Suitable for data that is frequently accessed together (e.g., embedding comments in posts).
* **Referencing**: Suitable for data that is accessed independently (e.g., referencing userId in posts).

### 6. Data Validation and Schema Design

Use MongoDB's schema validation to enforce data integrity.

json

{

"$jsonSchema": {

"bsonType": "object",

"required": ["username", "email", "password"],

"properties": {

"username": {

"bsonType": "string",

"description": "must be a string and is required"

},

"email": {

"bsonType": "string",

"description": "must be a string and is required"

},

"password": {

"bsonType": "string",

"description": "must be a string and is required"

}

}

}

}

### 7. Handling Offline Data

Use libraries like [Realm](https://realm.io/) or [PouchDB](https://pouchdb.com/) that can sync with MongoDB for offline-first capabilities.

### 8. Scalability and Performance

* **Sharding**: For large-scale applications, consider sharding to distribute data across multiple servers.
* **Replication**: Use replication for high availability.
* **Performance Monitoring**: Regularly monitor performance and optimize queries and indexes.

### 9. Security

* **Authentication**: Ensure secure authentication mechanisms.
* **Authorization**: Implement role-based access control.
* **Data Encryption**: Encrypt sensitive data both in transit and at rest.

### 10. Example MongoDB Schema Design

json

{

"users": {

"\_id": "ObjectId",

"username": "string",

"email": "string",

"password": "string",

"profilePicture": "string",

"createdAt": "date",

"updatedAt": "date"

},

"posts": {

"\_id": "ObjectId",

"userId": "ObjectId",

"content": "string",

"media": ["string"],

"createdAt": "date",

"updatedAt": "date",

"likes": ["ObjectId"],

"comments": [

{

"userId": "ObjectId",

"comment": "string",

"createdAt": "date"

}

]

},

"comments": {

"\_id": "ObjectId",

"postId": "ObjectId",

"userId": "ObjectId",

"comment": "string",

"createdAt": "date"

},

"notifications": {

"\_id": "ObjectId",

"userId": "ObjectId",

"type": "string",

"message": "string",

"isRead": "boolean",

"createdAt": "date"

}

}

### 11. Final Steps

* **Implementation**: Use MongoDB drivers to interact with the database.
* **Testing**: Thoroughly test the schema and data interactions.
* **Maintenance**: Regularly update the schema and indexes based on app usage patterns and performance analysis.

### 12. Conclusion:

The MongoDB database design is robust for the mobile app, ensuring it meets the app's requirements and scales effectively as your user base grows.

### 13. References:

<https://medium.com/@ya0285981/connecting-mongodb-in-your-android-app-using-java-9b273ae5ebd4>

<https://www.mongodb.com/resources/databases/android-databases>

<https://www.mongodb.com/resources/databases/database-for-mobile-apps>

8