NoSQL - Assessment

Introduction

This week you will be designing the database for the Food Order App in MongoDb. You will create collections, insert documents and retrieve records from the database.

All the create and insert operations in the database should be performed by executing the script.js file given in **Script_To_Populate_Database** folder.

All the select operations should be performed using the **Mongo Shell**. You can also use the MONGOSH shell attached to MongoDB Compass at the bottom.

Housekeeping points

- This is a minimal example and may not follow some standard practices.
- We focus on the main flow, and not much error handling.

DESIGN OF COLLECTIONS

- **Users** collection should have documents with the following fields -
    i. _id: A unique identifier for the document.
    ii. username: Represents the username of a user.
    iii. fullname: Represents the full name of a user.
    iv. email: Represents the email address of a user.
    v. password: Represents the password of a user.
    vi. isAdmin: Indicates whether the user has admin privileges or not.
    vii. isActive: Indicates whether the user's account is active or not.
    viii. createdTs: Represents the date and time when the document was created.
    ix. updatedTs: Represents the date and time when the document was last updated.
- **UserSessions** collection should have documents with the following fields -
    i. _id: A unique identifier for the document
    ii. userId: Represents the id of the user
    iii. sessionToken: Represents the session token generated for the user.
    iv. isActive: Indicates whether the session is currently active or not.
    v. createdTs: Represents the date and time when the document was created.
    vi. updatedTs: Represents the date and time when the document was last updated.
- **Categories** collection should have documents with the following fields -
    i. _id: A unique identifier for the category document.
    ii. name: Represents the name of a category.
    iii. description: Represents the description of a category.
    iv. image: Represents the URL of the image of a category.
- **Cuisines** collection should have documents with the following fields -
    i. _id: A unique identifier for the cuisine document.

ii. name: Represents the name of a cuisine.
iii. description: Represents the description of a cuisine.
iv. image: Represents the URL of the image of a cuisine.

- **Restaurants** collection should have documents with the following fields -
  i. _id: A unique identifier for the restaurant document.
  ii. name: Represents the name of a restaurant.
  iii. address: Represents the address of a restaurant.
  iv. image: Represents the URL of the image of a cuisine.
- **Fooditems** collection should have documents with the following fields -
  i. _id: A unique identifier for the fooditem document.
  ii. name: Represents the name of a fooditem.
  iii. description: Represents the description of a fooditem.
  iv. image: Represents the URL of the image of a fooditem.
  v. categoryId: Represents the id of the category to which the fooditem belongs.
  vi. cuisineId: Represents the id of the cuisine to which the fooditem belongs.
  vii. isVeg: Represents a value indicating the fooditem is vegetarian or not.
  viii. isActive: Represents a value indicating the fooditem is active or not.
  ix. createdTs: Represents the date and time when the document was created.
  x. updatedTs: Represents the date and time when the document was last updated.
- **Menus** collection should have documents with the following fields -
  i. _id: A unique identifier for the menu document.
  ii. restaurantId: Represents the id of the restaurant offering the menu.
  iii. isActive: Represents a value indicating the menu is active or not.
  iv. createdTs: Represents the date and time when the document was created.
  v. updatedTs: Represents the date and time when the document was last updated.
- **MenuItems** collection should have documents with the following fields -
  i. _id: A unique identifier for the menuItem document.
  ii. menuId: Represents the id of the menu document to which the menuItem belongs
  iii. fooditemId: Represents the id of the fooditem which is offered in the menu.
  iv. isActive: Represents a value indicating the menuItem is active or not.
  v. createdTs: Represents the date and time when the document was created.
  vi. updatedTs: Represents the date and time when the document was last updated.
- **Carts** collections should have documents with the following fields -
  i. _id: A unique identifier for the cart document.
  ii. userId: Represents the id of the user who has created the cart.
  iii. restaurantId: Represents the id of the restaurant from where the fooditems were added into the cart.
  iv. orderTotalPrice: Represents the total amount of price of all the fooditems placed in the cart.
  v. isActive: Represents a value indicating the cart is active or not.

vi. createdTs: Represents the date and time when the document was created.
vii. updatedTs: Represents the date and time when the document was last updated.

- **CartItems** collection should have documents with the following fields -
  i. _id: A unique identifier for the cartItem document.
  ii. cartId: Represents the id of the cart to which the cartItem belongs
  iii. fooditemId: Represents the id of the fooditem which is added to the cart.
  iv. fooditemPrice: Represents the price of the fooditem which is added to the cart.
- **ShippingDetails** collection should have documents with the following fields -
  i. _id: A unique identifier for the shippingDetail document.
  ii. shippingAddress: Represents the address where the order needs to be shipped.
  iii. emailId: Represents the email address of the user who has placed the order.
  iv. phoneNo: Represents the phone number of the user who has placed the order.
  v. isActive: Represents a value indicating the shippingDetail is active or not.
  vi. createdTs: Represents the date and time when the document was created.
  vii. updatedTs: Represents the date and time when the document was last updated.
- **Orders** collection should have documents with the following fields -
  i. _id: A unique identifier for the order document.
  ii. userId: Represents the id of the user who has created the cart.
  iii. restaurantId: Represents the id of the restaurant from where the fooditems were added into the cart.
  iv. orderTotalPrice: Represents the total amount of price of all the fooditems placed in the cart.
  v. shippingDetailsId: Represents the id of the shippingDetails for the order.
  vi. orderStatus: Represents the status of the order.
  vii. isActive: Represents a value indicating the cart is active or not.
  viii. createdTs: Represents the date and time when the document was created.
  ix. updatedTs: Represents the date and time when the document was last updated.
- **OrderItems** collection should have documents with the following fields -
  i. _id: A unique identifier for the orderItem document.
  ii. orderId: Represents the id of the order to which the orderItem belongs
  iii. fooditemId: Represents the id of the fooditem which is placed in the order.
  iv. fooditemPrice: Represents the price of the fooditem which is placed in the order.
  v. unitsPurchased: Represents the number of fooditems placed in the order.
  vi. isActive: Represents a value indicating the orderItem is active or not.
  vii. createdTs: Represents the date and time when the document was created.
  viii. updatedTs: Represents the date and time when the document was last updated.

Problem Statement

Your task is as follows -

1. **Create Database and Collections (15 Points)**

- Create a database named **FoodOrderAppDb** in your MongoDB Compass(using localhost connection) or MongoDb Atlas cloud account.
- Create **users, userSessions, categories, cuisines, restaurants, food items, menus, menuitems, carts, cartitems, shippingDetails, orders and orderitems collections**. Further, we need to insert data in the collections. We have already created a script using mongoose library to create various schemas and models. Also, we have added code to insert in these collections. For creating collections and inserting documents kindly follow the steps given below:
    1. Make sure that NodeJS is installed on the system.
    2. Open the Script_To_Populate_Database folder in VSCode.
    3. Open the terminal.
    4. First run the command **npm install** to install node_modules and all the dependencies.(for example, mongoose library)
    5. Next, put your MongoDB Connection String in the mongoose.connect() method.
    6. Finally, run the command **node script.js** to create all the 13 collections and insert documents in these collections from the json files present in the **data** folder.
    7. You can go to MongoDB Compass or MongoDB Atlas to see if the collections are created with documents inserted in them.
    8. Now you can take screenshots of documents entered in each collection and paste in the respective placeholders provided in the **NoSQL Assessment_For Noncoders.docx** file provided.

2. Retrieve records (45 Points)

- Retrieve the following records -

i. All distinct category names **(2 Points)**

ii. All vegetarian fooditems **(3 Points)**

iii. Retrieve the name and address of each of the restaurants. **(4 Points)**

iv. All orders with orderTotalPrice within the range of $75 to $150. **(4 Points)**

v. All orders which were created after 10th May 2023. **(4 Points)**

vi. All fooditems belonging to Italian cuisine. **(4 Points)**

vii. Get the userName and email of the user with fullname "John Doe". **(4 Points)**

viii. Retrieve details of the order with the lowest orderTotalPrice. **(4 Points)**

ix. Get the count of menu items for fooditem “Veg Pizza”. **(8 Points)**

x. Get the count of orders for each restaurant. **(8 Points)**

Program Organization

- You will be getting a word document file named **NoSQL Assessment_For Noncoders.docx** in which you need to place the screenshots of the collections with documents inserted as per Tasks 1 mentioned above.
- You also need to write your queries in the **NoSQL Assessment_For Nonoders.docx** as per the Task 2 mentioned above.

Evaluation Rubric

Total Project Points: 60

- Correctness:
  Correctness of implementation
  - Problem statement - point 1 : **15 Points**
  - Problem statement - point 2 : **45 Points**

Program Instructions

- You need to submit the **NoSQL Assessment_For Noncoders.docx** file containing all the screenshots and the queries.
- Project will not be evaluated if the file is not submitted in the expected format.