COMP 3133 – Full Stack Development – Lab 5

MongoDB & Mongoose

Developer Note:

1. Try and use the Robo 3T Studio to work on MongoDB queries
   https://studio3t.com/download/
2. Alternative to Robo 3T is VSCode plug-in cosmosdb

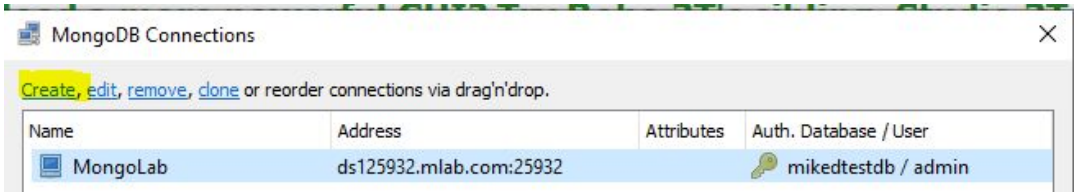
   https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-cosmosdb
3. Working on queries directly in the node application and outputting Json response is acceptable also.
4. Save the queries in a javascript file for submission ie. ex.js

References:

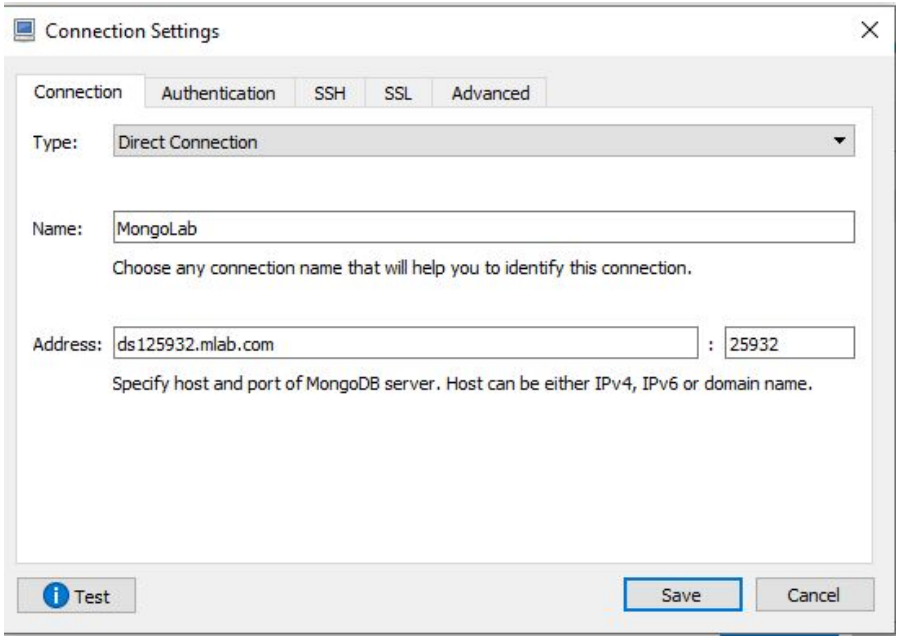
- https://www.tutorialspoint.com/mongodb/mongodb_query_document.htm

Robo 3T Studio Setup:

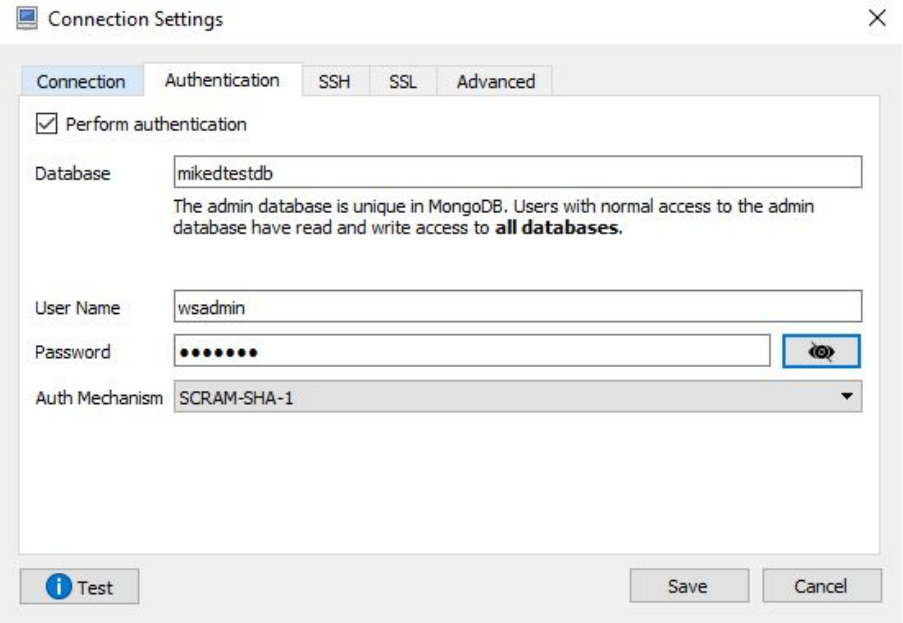
1. Launch Robo 3T Studio application from start menu. If it is not installed, then run the install from the following location: https://studio3t.com/download/
2. File menu >> Connection and open the MongoDB Connection interface



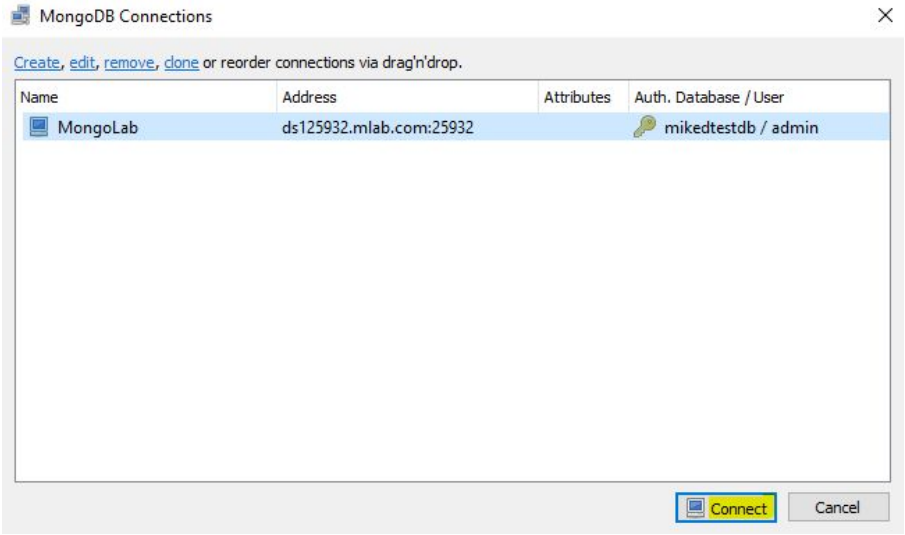
3. Click the create link and build the new connection from your Mongo Atlas address



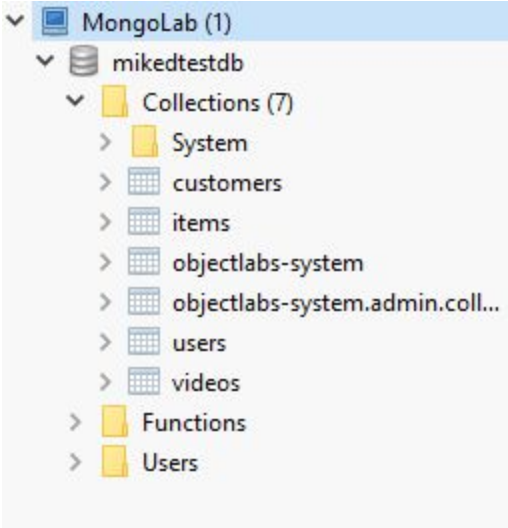
4. Click the Authentication tab and enter the connection authentication details. Click test to test the connection, when successful then save and close the connection windows.



5. Click the Connect button to open the connection



6. Verify the collections in the Mongo Atlas MongoDB.

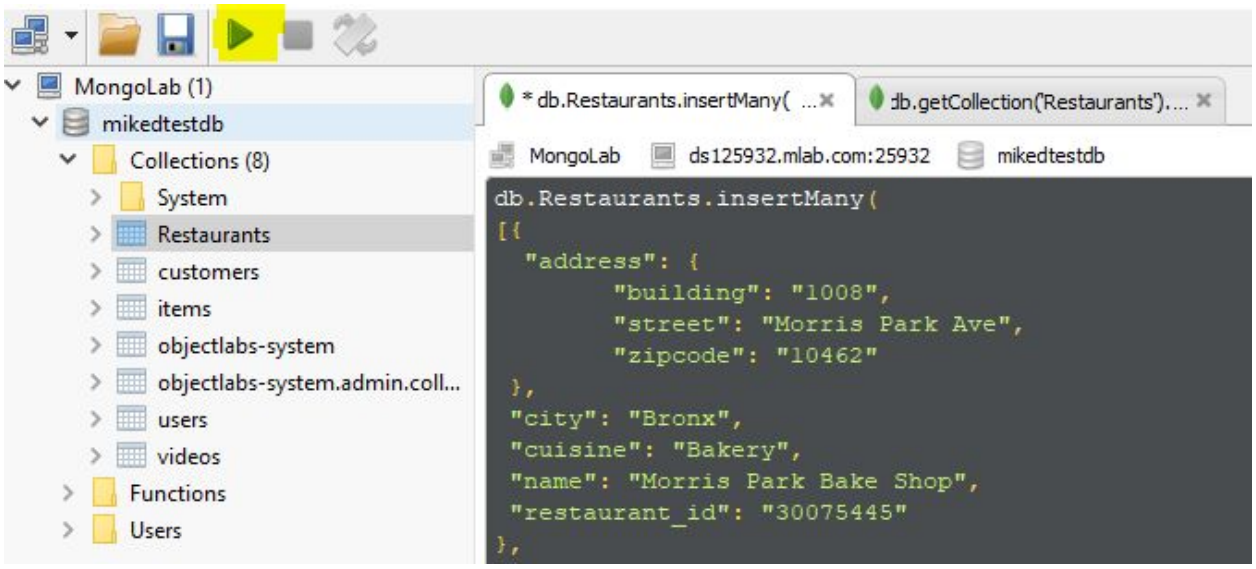


7. Select and right-click a collection to View Documents. This will open up a new panel to inspect the collection data.

Exercise 1: Creating Collections and Documents

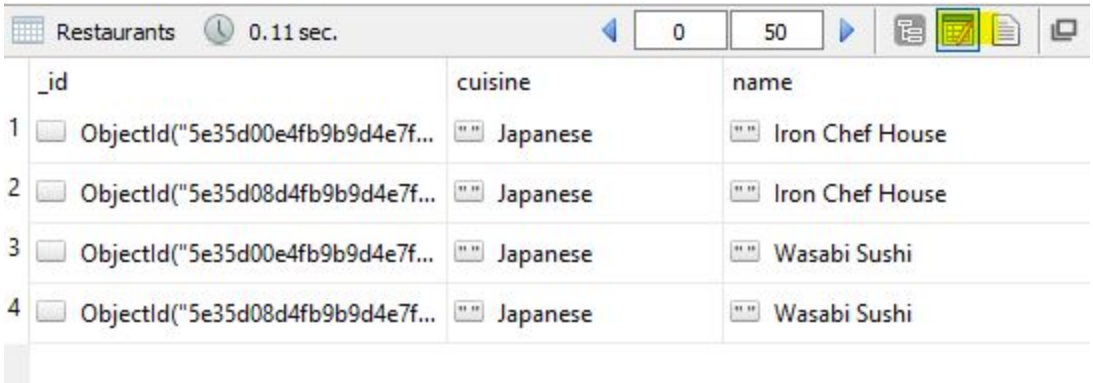
1. Use the seed data script found here.
   https://drive.google.com/open?id=13u4Kx1cPonjGj6y6imyH0DFuxt9ECWRE

   Use the MongoDb **dbo.collection.insert()** command to insert into the Restaurant collection and paste it into the query panel. Then click the execute button (found in the upper left toolbar) to run the script.



2. In the same query panel workspace use the MongoDb **db.collection.find()** command to view the documents.

3. Change the result set from tree view to tabular view, by click the following button on right toolbar.



Exercise 2: Projections, Query and Sorting

There are three main components of the MongoDB query:

1. Filter — Query
2. Select Columns (Fields) — Projections
3. Sort

1. Type the following **query** to filter the result set to only return the Japanese cuisine. Then execute the query.

```
db.getCollection('Restaurants').find(
     { "cuisine": "Japanese" } // query
     )
```

2. Using **projections** to select we can select which columns to either include '1' or exclude '0' in the query.

```
db.getCollection('Restaurants').find(
     { "cuisine": "Japanese" }, // query
     { "cuisine":1, "name":1}   // field projection
    )
```

3. We can sort the collection result set by using the **cursor.sort()** method. Use '1' for Ascending Order and '-1' for Descending Order.

4. **Write a query that will do the following**

   1. Filter on 'Japanese'' cuisine using the **$eq** logical operator

   2. Include the id, cuisines, name and city, resturant_id columns.

   3. Sort the restaurant_id in Ascending Order.

   Below is the expected result set.

_id	city	cuisine	name	restaurant_id
ObjectId("5e35e8594fb9b9...	Brooklyn	Japanese	Iron Chef House	40397699
ObjectId("5e35e8594fb9b9...	Brooklyn	Japanese	Wasabi Sushi	40398000

Exercise 3: Logical and Comparison Operators

https://docs.mongodb.com/manual/reference/operator/query-comparison/
https://docs.mongodb.com/manual/reference/operator/query-logical/

1. We can use the **'$eq' operator** to be more explicit in our query for cuisine. Using comparison operators we can use both **'$eq' equal** operator and the **'$ne' not equal** operator.

```
db.getCollection('Restaurants').find(
      { "cuisine": {$eq: "Delicatessen"} }, // query
      { "_id": 0, "cuisine":1, "name":1, "city": 1}   // field projection
    ).sort({ "name": -1 })
```

2. Write a query that uses the **$and** logical query operator, **$eq** and **$ne** comparison query operators.

   The query must return the following:

   1. All cuisines that are **equal** to Delicatessen **and** the city is **not equal** to Brooklyn
   2. The selected columns must include cuisines, name and city, but exclude id
   3. The sorting order must be Ascending Order on the name

   The following query will return the following result:

city	cuisine	name
Queens	Delicatessen	Big Tony's Sandwich Buffet
Queens	Delicatessen	The Godfather Panini Express

Exercise 4: Multiple Operators

Use the $and operator to create multiple conditions.
https://docs.mongodb.com/manual/reference/operator/query/and/

Build a query that does the following:

- Using the **$in** operator filter the **cuisines** that are "Bakery", "Chicken", "Hamburgers", "American"
- Using the **$ne** operator filter out the documents that have **city** "Brooklyn"
- Using the **$gt** operator include only documents that have **restaurant_Id** greater than 4000000
- Exclude columns _id. Include cuisine, name, city, restaurtant_id

- Sort Descending on **restaurant_id**

The following query will return the following result:

city	cuisine	name	restaurant_id
Bronx	Chicken	Mom's Fried Chicken	40382900
Staten Island	Hamburgers	Joeys Burgers	40397555
Queens	American	Brunos on the Boulevard	40397678

Exercise 5: Filtering on Array Columns

We can filter on the nested array data in the following way:

```
db.getCollection('Restaurants').find(
     { "address.building": {$eq: "1008" } }, // query
     { "restaurant_id": 1, "address.street":1, "name":1, "city": 1}
    ).sort({ "restaurant_id": 1 })
```

To correctly, output the address street we need to flatten out the array via aggregation. This is something we will learn in this week's lecture.

We can search wildcard values using the dollar $ operator in the following way

```
{name: /Deli$/}
```

Write a query that does the following:

Returns a result set where the name is contains "Thai", or the address street contains "Street" or the the address zip code equals 17988