Full Stack Development III – Lab 7

- MongoDb + Mongoose

Developer Note:

Download the **lab starter code** from this location.

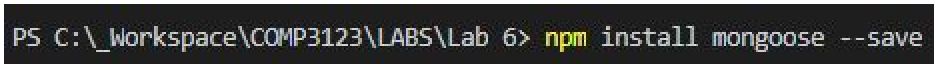
https://drive.google.com/open?id=1aOEqjEA1YploZ6U-b3ga2TNCPdtAZq4r

The **Restaurant sample data** from Lab 6 can be found at the following location

https://drive.google.com/open?id=1ECndE4JXBc0Rlfk9QEOVctaFgnDY-rc0

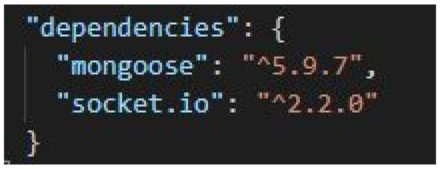
Exercise #1 – MongoDb and Mongoose

1. Using **npm install Mongoose** to connect to MongoDb



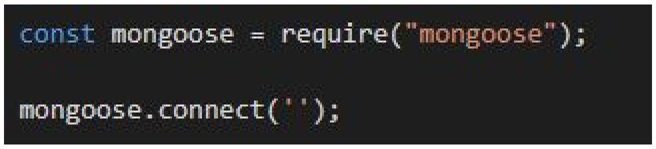
```
PS C:\_Workspace\COMP3123\LABS\Lab 6> npm install mongoose --save
```

2. Verify in your package.json file that Mongoose is a dependency.



```
"dependencies": {
  "mongoose": "^5.9.7",
  "socket.io": "^2.2.0"
}
```

3. In the **server.js** file and write the following code:



```
const mongoose = require("mongoose");

mongoose.connect('');
```

Leave the Mongoose connection string empty for now. We will populate it in the next step.

4. **Mongoose Connection to MongoDb**

   There are two options here:

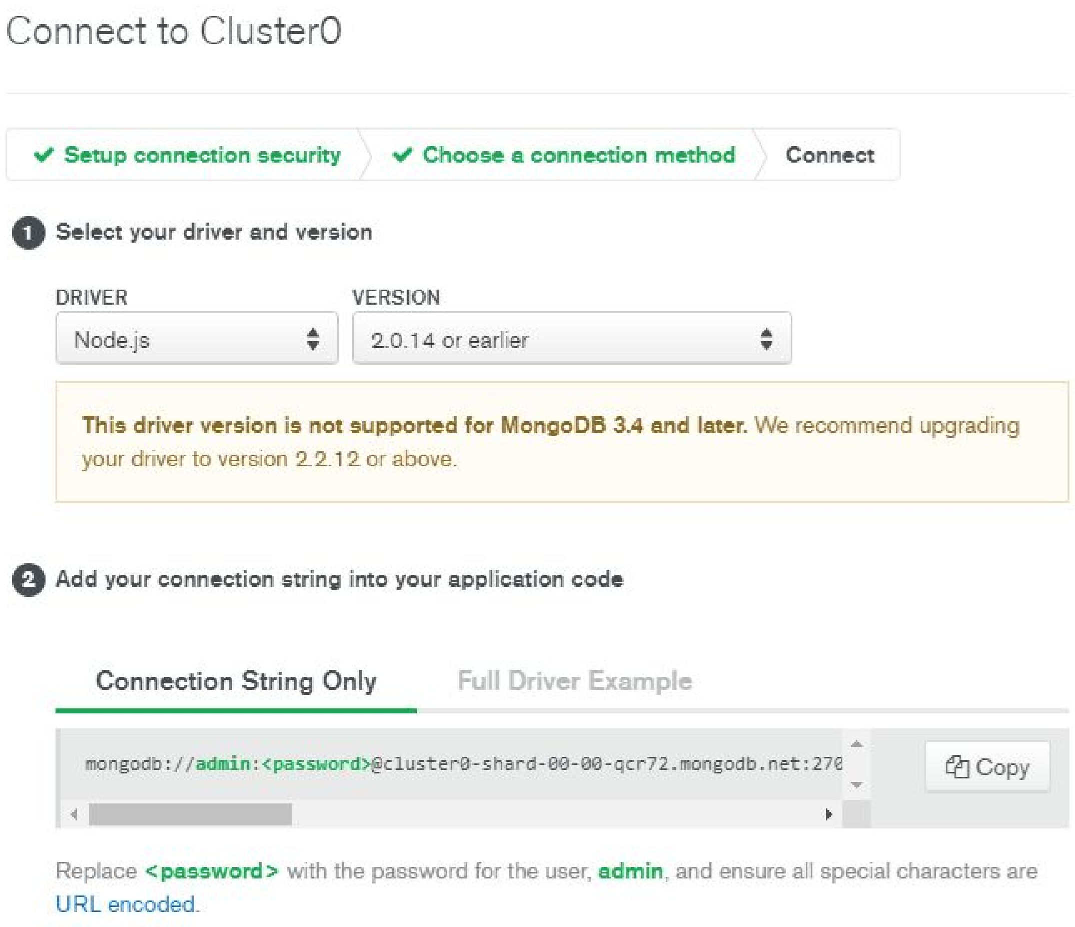
   - Use your local MongoDb connection string

```
const connectionString =
"mongodb://localhost:27017/?readPreference=primary&appname=MongoDB%20Compass&ssl=false";
```

- Use your Mongo Atlas Cloud connection string (example below)

```
'mongodb+srv://admin:<password>@cluster0-6fwsa.mongodb.net/test?retryWrites=true&w=majority'
```

- You need to log into your account and click **Connection**. Then select **Connect to Application.** Then select **Node Version 2.1.4** and **COPY** the connection string**.**



Connect to Cluster0

1 Select your driver and version

DRIVER: Node.js
VERSION: 2.0.14 or earlier

This driver version is not supported for MongoDB 3.4 and later. We recommend upgrading your driver to version 2.2.12 or above.

2 Add your connection string into your application code

```
mongodb://admin:<password>@cluster0-shard-00-00-qcr72.mongodb.net:270
```

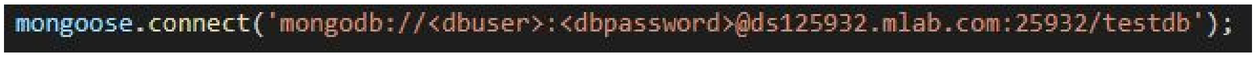
Replace <password> with the password for the user, admin, and ensure all special characters are URL encoded.

5. **Copy Mongoose Connection**

- Copy the following code in your **server.js file** before the **node http.createServer**

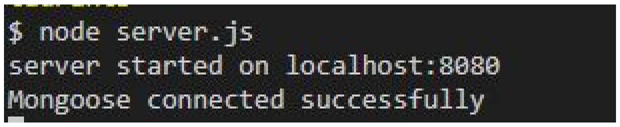
```
const connectionString =  "";

mongoose

  .connect(connectionString, {  useNewUrlParser: true } )

  .then( () => { console.log("Mongoose connected successfully "); },
    error => { console.log("Mongoose could not connected to database
: " + error);  }

  );
```

- Copy the MongoDB URI and paste it in the app.js Change the **dbuser** and **dbpassword (this is only required for the Mongo Atlas connection string)**.



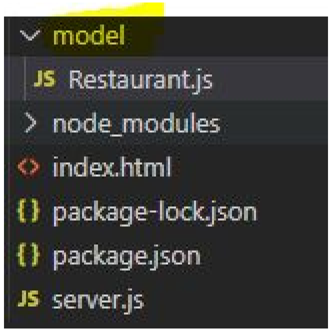
```
mongoose.connect('mongodb://<dbuser>:<dbpassword>@ds125932.mlab.com:25932/testdb');
```

6. Run the **server.js** file at the **command line** using Node to verify the connection is working successfully.



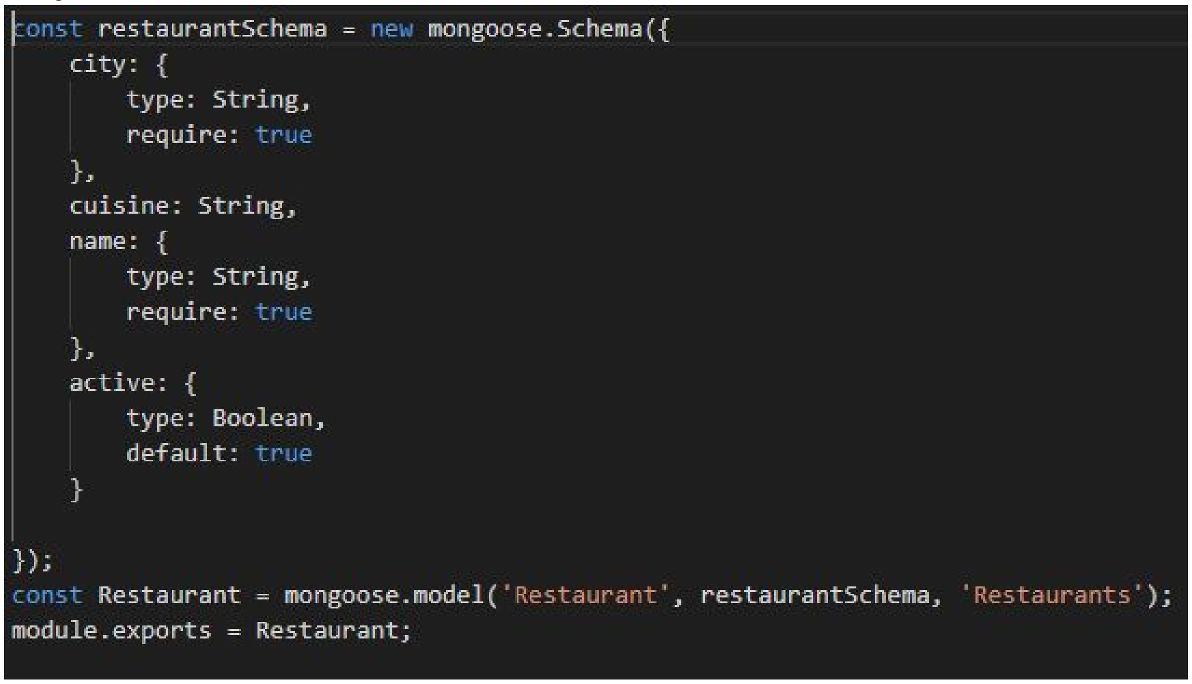
```
$ node server.js
server started on localhost:8080
Mongoose connected successfully
```

7. Create a folder named model in your project structure and then add file named Restaurant.js



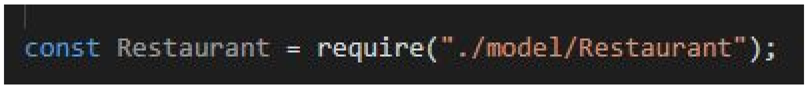
```
∨ model
   Restaurant.js
> node_modules
  index.html
  package-lock.json
  package.json
  server.js
```

8. Write the following schema code into the Restaurant.js file and export the new Restaurant Mongoose model



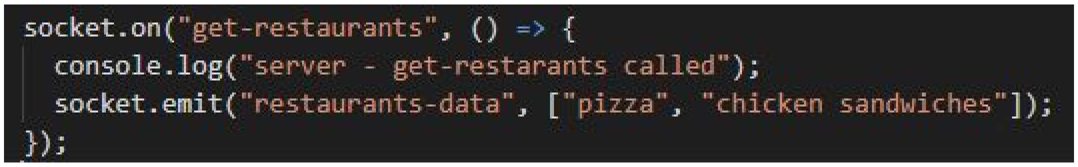
```
const restaurantSchema = new mongoose.Schema({
    city: {
        type: String,
        require: true
    },
    cuisine: String,
    name: {
        type: String,
        require: true
    },
    active: {
        type: Boolean,
        default: true
    }

});
const Restaurant = mongoose.model('Restaurant', restaurantSchema, 'Restaurants');
module.exports = Restaurant;
```

9. Back in the **server.js** import the Restaurant Mongoose model using require.



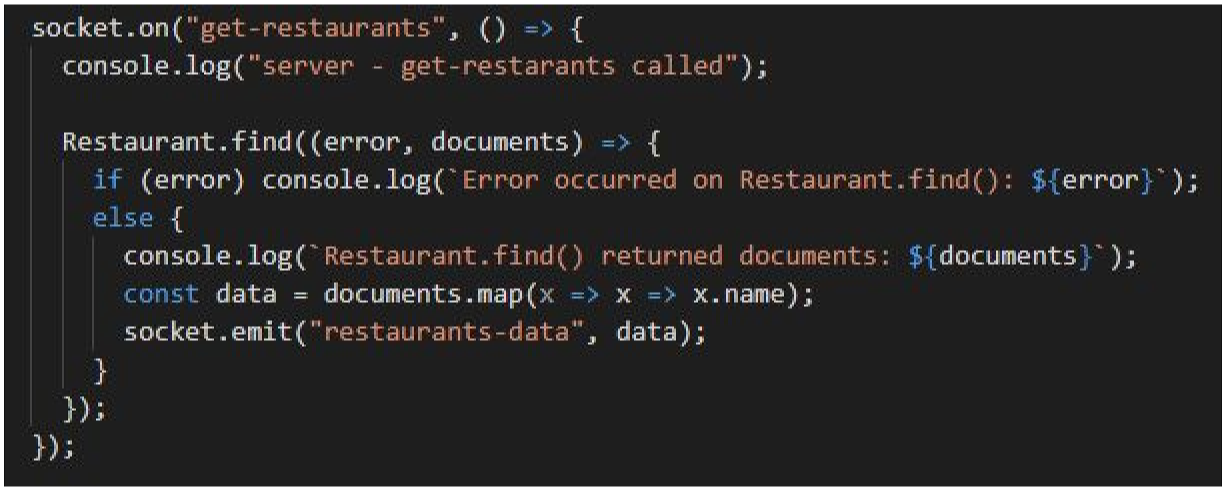
```
const Restaurant = require("./model/Restaurant");
```

10. In the **server.js** modify the following socket.io event handler



```
socket.on("get-restaurants", () => {
  console.log("server - get-restarants called");
  socket.emit("restaurants-data", ["pizza", "chicken sandwiches"]);
});
```

Change the code to use Mongoose.find to retrieve the Restaurant collections and return the documents to the client.



```
socket.on("get-restaurants", () => {
  console.log("server - get-restarants called");

  Restaurant.find((error, documents) => {
    if (error) console.log(`Error occurred on Restaurant.find(): ${error}`);
    else {
      console.log(`Restaurant.find() returned documents: ${documents}`);
      const data = documents.map(x => x => x.name);
      socket.emit("restaurants-data", data);
    }
  });
});
```

Task 1:Get Orders

Hint: you may need to manaully add data to your Order collection in the IDE Mongo Compas, Robot3T or Mongo Atlas to return data here. Otherwise, do task 2 in parallel, so you have data to query and return

- In **Model** folder of the application
    - Create an Order schema that contains an orderId, item and customer_name
    - Export the Order model
- In **server.js** implement the following:
    - Import the **Order model** using **require**
    - Add a socket event listener for **'get-orders'** and provide a callback function that does the following
        1. outputs to the console that the event was triggered
        2. Return the collection of Orders using **Mongoose.find()**
        3. **Emit** the Order data back to the client using a named event **'order-data'**
- In client file **index.html** implement the following:
    - wire up the Get Orders button to call the **addOrders** function and emit the event **'get-orders' to the server**
    - add event listener for **'order-data',** that will console.log the order data objects to the console

Task 2: Add New Order

- In **server.js** implement the following:

    - Add a socket event listener for **‘add-order’** and provide a callback function that does the following

        1. outputs to the console that the event was triggered
        2. Adds an Order using Mongoose.create() or Mongoose.save()
           Note: you can hard code the static data here for name in the server.js
        3. **Emit** an event back to the client using a named event **‘order-data-added**’ to notify the order was added.

- In client file **index.html** implement the following:

    - wire up the Add New Order button to call the **addOrders** function and emit the event **‘add-order’ to the server**
    - add event listener for **‘add-order-data’,** that will console.log message that **‘Order was added by server’**

Challenge:

- In **server.js** modify the Restaurant query to use query logical operators to filter and return only the records where the city is **Queens** and the cuisine is **Delicatessen. The only return the name and cuisine** to the Client caller (hint: use the map function to tranfrom the collection and return as JSON)