CMPS 356 Enterprise Application Development - Spring 2020
Lab 10 – Mongo DB

Objective

The objective of this lab is

- Practice reading and writing to a MongoDB Database using mongoose library
- Use mongoose to create document schema and model
- Use mongoose to read/write MongoDB documents to implement CRUD operations
- Practice MongoDB aggregation queries

Overview

This Lab is based on Lab 9 Banking App and Bookstore App. You are required to implement MongoDB repositories for both applications. DB repositories you will implement and deliver the same functionality as the file-based repositories provided in the base solution.

The tasks for this Lab are:

- Implement and test the Banking App database schema and repository methods.
- Implement and test the Bookstore App database schema and repository methods.

Project Setup

1. Download "Lab10-MongoDB" from the GitHub Repo and copy it to your repository.
2. Ensure that your **WebStorm** JavaScript language is set to **ECMAScript 6** and **Node.js Core** Libraries are enabled.
3. Make sure you have MongoDB installed [https://www.mongodb.com/download-center/community]. During the installation also install MongoDB Compass to get a graphical tool to access MongoDB databases [https://www.mongodb.com/products/compass]
4. If you face any issues follow this installation guide https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/
5. The project should be organized as follows:
    - **public** folder contains HTML pages, templates, CSS and client-side JavaScript
    - **data** folder has JSON files to be used in this lab.
    - **repositories** folder contains the repository classes.
    - **models** folder contains the document schemas.
    - **services** folder contains the services.

PART A - Banking App

Open the **BankingApp** on Webstorm and follow the steps below.

I. Connecting to MongoDB Database Using Mongoose

1. Open the terminal and start MongoDB server using **mongod**
2. Connecting to the MongoDb using mongoose
    - Install the mongoose package using the npm *[**npm** install **mongoos**]*
    - Open app.js and import **mongoose** package
    - Use mongoose to connect to the database (if the database does not exist then it will be auto created)

```
mongoose.connect('mongodb://localhost/BankDB');
```

If connecting fails, try using 127.0.0.1 instead of localhost.

II. Creating the Database Schemas and Models

The class diagram below shows the entities of the Banking App.

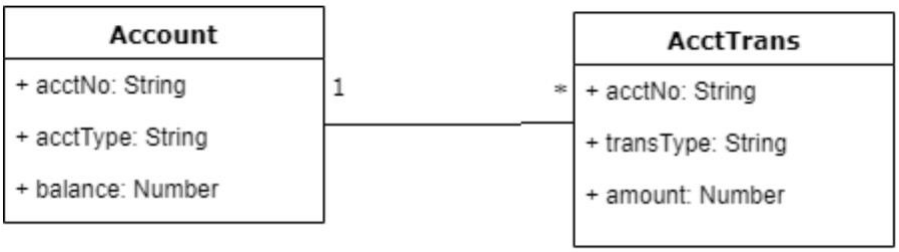

Figure 1 Banking Entities Diagram

1. Create a new folder and name it ***models***
2. Inside the ***models*** folder create two files and name them "**account.js**" and "**account-trans.js**" Inside **account.js** create **accountSchema** having the properties shown in the class diagram. Note that all fields are required. The balance should have a custom validation error "Balance is a required property".
3. Add a virtual property **minBalance** that returns 1000 if the account type is Saving or null otherwise.
4. Add a virtual property **monthlyFee** that returns 15 if the account type is Current or null otherwise.
5. Create and export a Model named **Account** based on the **accountSchema**
6. Inside **account-trans.js**. create **accountTransSchema** having the properties shown in the class diagram. Note that all fields are required. The **transType** could be either Debit or Credit. The **acctNo** should be a reference to the **Account** model.
7. Create and export a Model named **AcctTrans** based on the **accountTransSchema**
8. Open the **account-repository.js** and import both **Account** and the **AcctTrans** Models.
9. Implement all the repository methods using the **Account** and the **AcctTrans** Models. Make sure that you implement a method to load the **accounts.js** data to MongoDB Accounts collection.
10. Implement the **getAcctsTotalBalance** repository method using an aggregation query.
11. Add a Web API to make the **getAcctsTotalBalance** accessible at *'/reports/accts-summary'*
12. Test each method using **Mocha** or **Postman**.

PART B – Book Store App – Deadline Next week before the lab

In part B you should implement the entity schemas and the DB repository for the **BookStore App**. The DB repository should implement the same functionality as the file-based repository provided in the base solution. NOTE : **You should test your implementation as you progress and document your testing .**

1. Open the **Book Store App**. Change app.js to connect to "**BooksDB**" MongoDB database.
2. Create four Models [Book, Author, Borrower, and Borrowing]. The schema of these models should be derived based on the json data files provided in the base solution. Hints:
   - Book.authors property should be an array of refences to the Author model.
   - Author.books property should be an array of references to the Book Model.
   - Borrowing.bookId property should be a reference to the Book Model.
   - Borrowing.borrowerId should be a reference to the Borrower Model.

Make sure the Book, Borrower, Borrowing and Author Model schema properties are validated with custom validation. Example, the book title, author, …. are required properties.

3. In the **books-repository.js** import both models **Author, Book, Borrowing, Borrower** models and implement all the methods in books-repository.js using those models.
   Make sure that you implement a method to load the provided json data to MongoDB.

4. Add the needed repository and service methods to implement the following reports:
   a. Books Summary report: return the books counts, average page count per book category.
   b. Top 3 borrowers with the total number of books they have borrowed.
   c. Top 3 borrowed books and the number of times they have been borrowed.
   d. Borrowing summary: Summary of borrowings by book category. The report should return the total number of borrowings per book category.

   Test these reports using Postman. No need to provide a UI for them.

NOTE: All the query should be done on the Database. You should NOT do any filtering or aggregation on the client-side using JavaScript. You should use the database queries capabilities to implement all the aggregation, filtering needed for your solution.

Further details about MongoDB query operators is available at https://docs.mongodb.org/manual/reference/operator/query/ . You may use **Compass or Robo 3T** to interact with MongoDB database.

You need **to** test your implementation as you progress and document your testing. After you complete the lab, fill in the ***TestingDoc-Grading-Sheet.docx*** and save it inside **Assessment** folder. Sync your repository to push your work to GitHub.