CMPS 350 Web Development Fundamentals Spring 2022
Lab 10 – Data Management Using MongoDB

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. You are required to implement a MongoDB repository to deliver the same functionality as the file-based repository provided in the base solution.

The tasks for this Lab are:

- Implement and test the Banking 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 these installation guides:
https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/
https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-windows/

5. The project should be organized as follows:

- **public** directory contains HTML pages, templates, CSS and client-side JavaScript
- **data** directory has JSON files to be used in this lab.
- **repository** directory contains the repository classes.
- **model** directory contains the document schemas.
- **service** directory contains the services.

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 **mongoose**]*
    - 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.

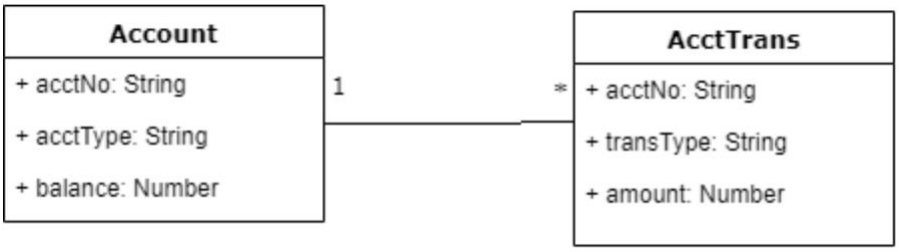

Figure 1 Banking Entities Diagram

1. Create a new directory and name it ***model***
2. Inside the ***model*** directory 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/accounts/balance'*
12. Test each method using **Mocha** or **Postman**.