

# WEB422 Assignment 1

## Submission Deadline:

Friday, May 29<sup>th</sup> @ 11:59pm

## Assessment Weight:

5% of your final course Grade

## Objective:

This first assignment will help students obtain the sample data loaded in MongoDB Atlas for the WEB422 course as well as to create (and publish) a simple Web API to work with the data.

## Specification:

### Step 1: Loading the "Sample Data" in MongoDB Atlas

The first step for this assignment is to create a new "Project" in your existing MongoDB Atlas account (if you have deleted your account from last semester, please revisit the documentation here from WEB322 - <https://web322.ca/notes/week08>).

Assuming that you have an account in MongoDB Atlas, please follow the instructions located below (from the WEB422 website) to create a new "Project", "Cluster" and "Load the Sample Dataset".

MongoDB Sample Data Instructions - <https://web422.ca/notes/mongodb-sample-data>

### Step 2: Building a Web API

Once you have completed the guide (Step 1), and have the data loaded in a new Project within your MongoDB Atlas account, we must build and publish a Web API to enable code on the client-side to work with the data.

To get started:

- First create a folder (ie: "salesAPI") for your project somewhere on your machine. Next, download the Assignment 1 boilerplate files from here:

<https://ict.senecacollege.ca/~patrick.crawford/shared/summer-2020/web422/A1/A1.zip>

- Once A1.zip has been downloaded, extract the files and add them to your newly created "salesAPI" folder.
- Open this folder in Visual Studio Code (which should now contain server.js and the "modules" folder) and perform the usual tasks for creating a web server from scratch in Visual Studio Code (ie: "**npm init**", followed by the "**npm install**" tasks for this project, such as "**express**", "**cors**" and "**body-parser**").
- Finally, initialize an empty Git repository for this folder using the command "git init"

Viewing / Modifying Existing Files:

### **modules/data-service.js**

This file (located in the "modules" directory) does not need to be modified at all. It exists to provide the 6 functions required by our Web API for this particular (sales) dataset, ie:

- **initialize()**: Establish a connection with the MongoDB server and initialize the "Sale" model with the "sales" collection
- **addNewSale(data)**: Create a new sale in the collection using the object passed in the "data" parameter
- **getAllSales(page, perPage)**: Return an array of all sales for a specific page (sorted by **saleDate**), given the number of items per page. For *example*, if **page** is **2** and **perPage** is **5**, then this function would return a sorted list of sales (by **saleDate**), containing items **6 – 10**. This will help us to deal with the large amount of data in this dataset and make paging easier to implement in the UI later.
- **getSaleById(Id)**: Return a single sale object whose "\_id" value matches the "Id" parameter
- **updateSaleById(data,Id)**: Overwrite an existing sale whose "\_id" value matches the "Id" parameter, using the object passed in the "data" parameter.
- **deleteSaleById(Id)**: Delete an existing sale whose "\_id" value matches the "Id" parameter

### **modules/salesSchema.js**

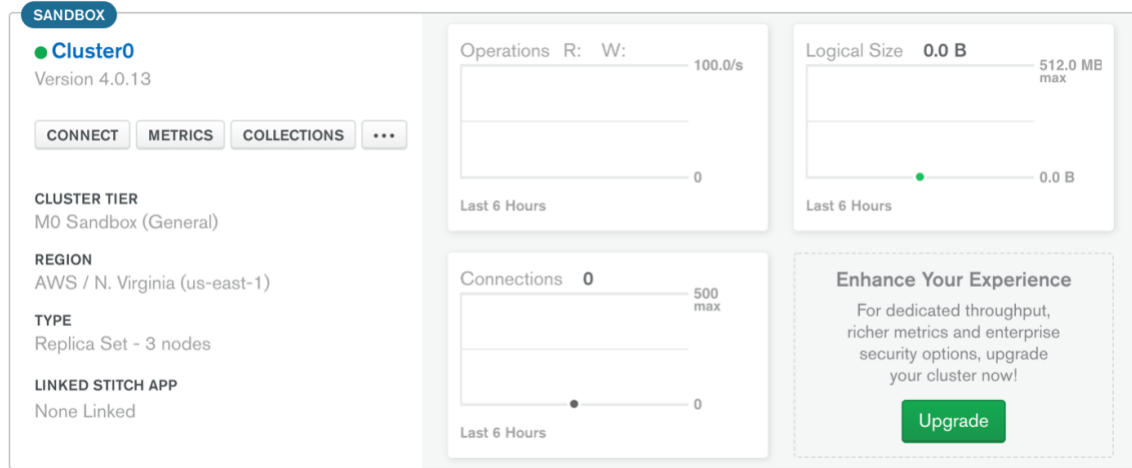
This file (located in the "modules" directory) does not need to be modified at all. It exists to provide the schema for this particular (sales) dataset

### **server.js**

Here is where the bulk of the work needs to be done. We must update it to add the 6 required routes (listed below) as well as to provide our dataService module with a valid MongoDB connection string.

### **Obtain the connection string**

- Ensure that you're looking at the overview for your newly created Cluster (within your newly created Project) that contains the sample data.



- Next, click the "CONNECT" button and grab the connection string using the "Connect Your Application" button. **NOTE:** If you have not yet created a user for this database, or whitelisted the ip: 0.0.0.0/0, please proceed to do this first.
- Once you have your connection string, it should look *something like this*:  
**mongodb+srv://userName:<password>@cluster0-abc0d.mongodb.net/test?retryWrites=true&w=majority**
- Next, replace the entire string <password> with your password for this cluster (do not include the < & > characters)
- Finally, replace the text **test** with the database name: **sample\_supplies** and add the updated connection string to line 6 of your server.js file (as a parameter to your dataService() function call)

## Add the routes

The next piece that needs to be completed before we have a functioning Web API is to actually define the routes (listed Below). **Note:** All routes must return JSON formatted data. If plain text is to be returned, it must be sent in an object with property "message", ie: {message: "new sale successfully added"}. Do not forget to return an error message if there was a problem.

- **POST /api/sales**

This route uses the body of the request to add a new "Sale" document to the collection and return a success / fail message to the client.

- **GET /api/sales**

This route must accept the numeric query parameters "page" and "perPage", ie:  
/api/sales?page=1&perPage=5. It will use these values to return all "Sales" objects for a specific "page" to the client.

- **GET /api/sales**

This route must accept a numeric route parameter that represents the id of the desired sale object, ie:  
/api/sales/5bd761dcae323e45a93ccfe8. It will use this parameter to return a specific "Sale" object to the client.

- **PUT /api/sales**

This route must accept a numeric route parameter that represents the `_id` of the desired sale object, ie: `/api/sales/5bd761dcae323e45a93ccfe8` as well as read the contents of the `request body`. It will use these values to update a specific "Sale" document in the collection and return a success / fail message to the client.

- **DELETE /api/sales**

This route must accept a numeric route parameter that represents the `_id` of the desired sale object, ie: `/api/sales/5bd761dcae323e45a93ccfe8`. It will use this value to delete a specific "Sale" document from the collection and return a success / fail message to the client.

### Step 3: Pushing to Heroku

Once you are satisfied with your application, deploy it to Heroku:

- Ensure that you have checked in your latest code using **git** (from within Visual Studio Code)
- Open the integrated terminal in Visual Studio Code
- Log in to your Heroku account using the command **heroku login**
- Create a new app on Heroku using the command **heroku create**
- Push your code to Heroku using the command **git push heroku master**

**IMPORTANT NOTE:** Since we are using an "**unverified**" **free** account on Heroku, we are limited to only **5 apps**, so if you have created 5 apps already, you must delete one (or verify your account with a credit card).

### Assignment Submission:

1. Add the following declaration at the top of your `server.js` file

```
/*
*****
* WEB422 – Assignment 1
* I declare that this assignment is my own work in accordance with Seneca Academic Policy.
* No part of this assignment has been copied manually or electronically from any other source
* (including web sites) or distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
* Heroku Link: _____
*
*****
*/
```

2. Compress (.zip) the files in your Visual Studio working directory (this is the folder that you opened in Visual Studio to create your client side code)

### Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- Submitted assignments **must** run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.