COMP 2406 B - Fall 2022
Assignment #4

MongoDB and OpenStack Deployment

Due Monday, November 28, 23:59. No late assignments will be accepted.

Submit a single zip file called "YourName-a4.zip". Do not submit node_modules folder.

This assignment has 100 marks. If your solution does not use MongoDB, you will receive 0 marks.

Assignment Background

In this assignment, you will use a database to store the office supply list and provide the functionality to allow users to add new items and modify the supplies currently stored in the database. Use MongoDB to achieve persistence of data between server restarts. Additionally, you will need to deploy your application to OpenStack.

For this assignment, you are expected to use the Express module, the template engines, the MongoDB module, as well as the additional tools we have learned about so far in the course. Your submission must NOT contain any **.html** files, but you can include **.css** files for styling.

Database Initializer (20 points)

To start this assignment, download the **A3-data.zip** file from Brightspace that was used for the previous assignment. This zip file contains the "vendors" directory with 3 vendor files (**grand.json**, **indigo.json**, and **staples.json**. Create the **database-initializer.js** file, which you can use to create the initial database for the assignment (or re-create it if you break the database during testing). Initialize a database called "**a4**" with a collection called "**supplies**". Read information from the vendor .json files, and for each office supply item, create a corresponding document in the collection "**supplies**". Besides the provided properties, such as `name`, `description`, `stock`, and `price`, each supply document should include `vendor`, `category`, and `rating`. The `vendor` and `category` values should match the corresponding values in the provided .json files. For example, the item with id 0 from **staples.json** must be initialized with the following document:

```
{
    _id: new ObjectId("6373f323f47de92346964890"),
    name: "Printer Paper",
    category: "Paper",
    vendor: "Staples",
    price: 5.5,
    stock: 3,
    rating: [-1, -1],
    description: "odio semper cursus. Integer mollis."
}
```

Notice you do not need to provide the `_id` value - it will be provided by MongoDB. The `rating` field will initially be set to `[-1, -1]`.

Initialize your project using **npm init**. Ensure the MongoDB Node.js driver is installed in your assignment directory by running **npm install mongodb**. Run the **database-initializer.js** file using Node.js while the MongoDB daemon is also running on your computer, and it will initialize a database called "**a4**" with a collection called "**supplies**". Now you can start your server.

Navigation Header (5 marks)

Each page on your web application should have a navigation header containing, at minimum, the following:

- link to the **home** page (/),
- link to the **supply items** page (/items),
- link to the **add supply items** page (/additem).
- link to the **bulk** page (/bulk)

Home Page (5 marks)

This page must contain the headers mentioned above. The page should also have a welcome message and outline the main functionality of your web application.

Supply Items Page (10 marks)

This page must contain the name of each supply item in the database, and each name in the list should be a link to that item's specific URL (i.e., **`/items/thatItemsUniqueID`**). The id (**`thatItemsUniqueID`**) should match the `_id` value provided by MongoDB.

Item's Page (15 marks)

The page must list all the information about the item stored in the database except the unique id. All the information on this page should be editable. The page must display the **average** rating of this item (based on all user ratings) or say "none" if the item did not receive any ratings. To implement this, notice that the rating value is an array of two values.

The user must not be allowed to edit the `rating` value directly but only add their rating as an integer between 0 and 5.

For inspiration and code hints, please refer to the **product-crud-example** presented in class and other code examples.

The user input must be verified on the client. You need to check that the price and the stock are a number, and the rating is an integer x, $0 \leq x \leq 5$. One way of doing this is by using **regex** to accept only numbers and, possibly, a dot (.), and the **Number()** method to convert a string value to a number.

The page must contain two buttons:

- **Update** – to update the item. When clicked, the client should verify the provided information. If at least one of the provided values is not valid, alert the user about the mistake. Otherwise, send the modified item data to the server (this should be a **PUT** request to **`/items/thatItemsUniqueID`**). The server should update the document in the database.
- **Delete** – to delete the item. The item document should be removed from the database. Send a **DELETE** request to **`/items/thatItemsUniqueID.`**

For both buttons, the server must communicate the result of the update/delete operation to the client. The client should display an alert to the user indicating whether the operation was successful.

Add New Supply Item Page (10 marks)

This page must provide the functionality to add a new supply item to the database by making a **POST** request to the resource **`/items`** containing a JSON encoding of a new item. The page must allow the user to enter the `name`, `category`, `vendor`, `price`, `stock`, and `description` for a new item. The **POST** request must be handled through client-side JavaScript (i.e., making an **XMLHttpRequest**). When the server's response indicates that the new item has been added, the browser should be redirected to the page to view the newly added item.

Bulk Page (20 marks)

This page must provide the functionality to perform queries and bulk updates on the database. The user should be able to enter the **vendor**, **category**, **stock**, and **sales percentage**. The documents in the database will be filtered according to the vendor's name, supply category, and item stock. The user does not have to enter all the values. For example, if only the vendor's name is specified, the documents will only be filtered according to the vendor's name.

The page must contain two buttons:

- **Add Stock** – to increase the stock of matching items. When clicked, the client should verify that the provided stock information is a positive integer. Otherwise, ask the user to update the information. Among all the supply items that match the provided vendor and category, we choose only those with a stock smaller than the one provided by the user and increase their stock by 5. This should be implemented using a **PUT** request to **/addstock**.
- **Sale** – to put matching items on sale. Verify that the user provided a **sales percentage** value and that it is a positive integer ≤ 100. Reduce the price for all the items, matching vendor/category values, by the specified percentage. This should be implemented using a **PUT** request to **/sale**.

For both buttons, the server should update the matching documents in the database and notify the client about the outcome. If the update was successful, redirect to the **Supply Items Page**. Otherwise, inform the user about the error.

Server Requirements

Your server API will be responsible for supporting the routes outlined below.

1. **`GET /`** – Should provide a response containing the HTML for the home page.
2. **`GET /items`** – the server must respond with an HTML page containing a list of all office supply items stored in the database.
3. **`GET /items/`*`:itemID`*** – This parameterized route should respond with HTML for the **Item's Page** explained above.
4. **`GET /additem`** – Should respond with an HTML page (**Add New Supply Item**) which provides functionality to add a new supply item to the database by making a **POST** request to the resource **`/items`**.
5. **`POST /items`** – Accepts a JSON encoding of a new item. The server can assume that the client sent valid information and will not run validation, which, in general, is a bad idea. The server must add the default rating values to the received object and add the document to the database. If the insert operation were successful, the server should send a response with a 201 status code ("created"). Otherwise, send a 500 code.
6. **`PUT /items/`*`:itemID`*** – This route will accept a JSON body matching the item data format shown on the first page. Your server does not need to verify the structure/integrity of the data – you can assume that it is following the proper format. The server should update the corresponding document (with the id ***`itemID`***) in the database and then send a (201) response to confirm the updates have been made. If there was an error, then send a 500 code.
7. **`DELETE /items/`*`:itemID`*** – The server should remove the document with the id ***`itemID`*** from the database. If the operation was successful, respond with a 201 code. Otherwise, respond with a 500 code.
8. **`PUT /addstock`** – This route will accept a JSON body containing vendor, category, and stock information. You do not need to verify the structure/integrity of the data. The server should increase the stock by 5 for all the documents matching vendor and category values whose current stock is smaller than the provided in JSON. If the update was successful, send a 201 status code. Otherwise, send a 500 code.
9. **`PUT /sale`** – This route will accept a JSON body containing vendor, category, and sales percentage. You do not need to verify the structure/integrity of the data. The server should reduce the price for all the items, matching vendor/category values, by the specified percentage. If the update was successful, send a 201 status code. Otherwise, send a 500 code.

Deployment to OpenStack (15 points)

You must deploy your solution to Assignment 4 on an OpenStack instance. Use the project called "COMP2406B-F22-assignment-4". If you are unfamiliar with OpenStack, you can read about it on the OpenStack support page. The support page also provides tutorials for creating an instance, connecting to an instance, and transferring files to your instance. Before the deadline for Assignment 4, you should have your web application installed and initialized on your OpenStack instance. To prevent any possible issues (e.g., recovering from a server crash during grading), you should also include instructions for initializing and running your server in the README file you submit with your code on Brightspace.

IMPORTANT: make sure you are in the "COMP2406B-F22-assignment-4" project when they choose to launch an instance to complete the assignment. Multiple projects are present in the drop-down menu (top left of the dashboard). You must select the correct one.

If you do not see the "COMP2406B-F22-assignment-4" project, you need to update your scs account as explained here.

Bonus (5 points)

For 5 bonus points retrieve five items with the highest rating. Let us know if you implemented the bonus on the home page of your app. The mark for the assignment cannot go over 100 points.

Academic Integrity

Submitting not original work for marks is a serious offence. We use special software to detect plagiarism. The consequences of plagiarism vary from grade penalties to expulsion, based on the severity of the offence.

You may:

- Discuss general approaches with course staff and your classmates,
- Show your web page, but not your code,
- Use a search engine / the internet to look up basic HTML, CSS, and JavaScript syntax.

You may not:

- Send or otherwise share your solution code or code snippets with classmates,
- Use code not written by you,
- Use a search engine / the internet to look up approaches to the assignment,
- Use code from previous iterations of the course unless it was solely written by you,
- Use the internet to find source code or videos that give solutions to the assignment.

If you ever have any questions about what is or is not allowable regarding academic integrity, please do not hesitate to reach out to course staff. We will be happy to answer. Sometimes it is difficult to determine the exact line, but if you cross it, the punishment is severe and out of our hands. Any student caught violating academic integrity, whether intentionally or not, will be reported to the Dean and be penalized. Please see Carleton University's Academic Integrity page.

Submit Your Work

To receive marks for your assignment, you must deploy it on OpenStack and submit your files to Brightspace. To submit your assignment to Brightspace, you must **zip** all your files and submit them to the Assignment 4 submission page. Do not submit the **node_modules** directory; otherwise, you will receive a **deduction of 10 points**. You should use NPM to manage dependencies and submit the necessary **package.json** file. You must provide a **README.txt** file explaining the steps required to initialize/run your submission. Also, let us know whether you implemented the bonus part.

Name your .zip file "YourName-a4.zip". Make sure you download your .zip file and check its contents after submitting it. If your .zip file is missing files or corrupt, you will lose marks. Your .zip file should contain all the resources required for your assignment to run.