COMP 2406 B - Fall 2022
Assignment #3

Express Server

Due Monday, November 14, 23:59. No late assignments will be accepted.

Submit a single zip file called "YourName-a3.zip".

This assignment has 100 marks. There is an opportunity to earn 5 bonus points, but the final mark for the assignment cannot exceed 100. The bonus points are not transferable to other assignments.

Assignment Background

In the previous two assignments, we have developed a server to serve vendor data, accept orders, add new office supplies items, and record/serve vendor stats. In this assignment, you will develop a server capable of managing vendor data that could be used for those previous components. The server will be responsible for storing vendor data for any number of vendors. The server will also provide the functionality to allow a user to add new vendors and modify the information/supplies of vendors currently stored on the server. For this assignment, you are expected to use the **Express** module, the template engines, as well as the additional tools we have learned about so far in the course.

Provided files

To start this assignment, download the **A3-data.zip** file from Brightspace. This contains a "**vendors**" directory with 3 vendor files (**grand.json**, **indigo.json**, and **staples.json**), the same as Assignment 2. The data for the provided vendors should be loaded into your server when it starts. Additional vendors that are added through your API do not need to be stored in files. In the next assignment, we will use MongoDB to achieve persistence of data between server restarts.

Server Requirements (85 marks)

Your server API will be responsible for supporting the routes outlined below. Additionally, a number of these routes must be capable of providing HTML or JSON data in response to a client's request, depending on the **content type** specified in the request's **Accept** header. This will allow your API to be more flexible, as it can provide the HTML contents for a browser or the JSON data for programmatic access. All HTML pages sent by the server must contain a **header** that includes links to the resources:

- **`/`**
- **`/vendors`**
- **`/addvendor`**

Assignment 3 – due Monday, November 14, 23:59

All your HTML content should be created using **template engines**. Your code must NOT contain any **.html** files.

Summary of routes the server must support:

1. **`GET /`** – Should provide a response containing the HTML for the home page. This must contain the headers mentioned above. The page should also contain a welcome message. This route does not need to support requests for JSON data.
2. **`GET /vendors`** – If the client is requesting HTML, the server must respond with an HTML page containing a list of all vendors stored on the server. This page must contain the name of each vendor in a list, and each name in the list should be a link to that vendor's specific URL (i.e., **`/vendors/thatVendorsUniqueID`**).
   If the client requests JSON data, the server should send back a JSON response with the following format (where **`array`** represents an array containing each of the vendor IDs):
   `{"vendors" : array }`
3. **`GET /addvendor`** – Should respond with an HTML page which provides functionality to add a new vendor to the server by making a **POST** request to the resource **`/vendors`**. The page must allow the user to enter the **name**, **delivery fee**, and **minimum order** for a new vendor. The **POST** request must be handled through client-side JavaScript (i.e., making an **XMLHttpRequest**). Details of the structure of the request are given in the specification for **`POST /vendors`** below. When the response from the server is received indicating that the vendor has been added, the browser should be redirected to the page to view the newly created vendor.
4. **`POST /vendors`** – Accepts a JSON encoding of a new vendor with the following format (where italics represent some value for that new vendor): **`{"name":`** ***`vendorName`*****`,`** **`"delivery_fee":`** ***`deliveryFee`*****`,`** **`"min_order":`** ***`minOrder`*****`}`** The handling of this request must verify that the correct fields exist in the body and add the new vendor to the server's data. If at least one of the fields is empty or missing from the received JSON encoding, then send a **400** response (bad request error). Initially, the office supply list of the new vendor should be blank. The new vendor data does not need to be stored in a file (i.e., it can be stored in RAM). Each vendor on the server must have a unique ID, and the handling of the request should ensure that a new ID is assigned to the new vendor that does not conflict with any other vendor's ID. The response sent by the server should contain the JSON representation of the newly created vendor.
5. **`GET /vendors/`*****`:vendorID`*** – This parameterized route should respond with either HTML or JSON data, depending on the value of the request's **Accept** header.
   If the request is for JSON data, the entire vendor object with the given ID (i.e., the vendor with ID = ***`:vendorID`***) can be stringified and sent in response.
   If the request is for HTML, your server must respond with a page that supports the following:
   a. The page must show the vendor's name, delivery fee, and minimum order.
   b. The page must show the entire supply list of the vendor, divided into categories. Each supply item must be shown within the correct category and show the ID, name, description, price, and stock of the item.

c. The user must be able to modify the name, delivery fee, and minimum order of the vendor. A straightforward way to support this is to place the information in text fields.

d. [for up to 5 BONUS points] The user must be able to add a new category to the supply list by specifying a category name (i.e., in a text field) and clicking a provided **Add Category** button. It should not be possible to add duplicate category names to a supply list. When the category is added to the supply list, the page should update to display this new category. Note that the data does NOT need to be updated on the server immediately. The changes will be local to the client until the **Save** button (described below) is pressed.

e. The user must be able to add a new supply item to the office supplies by specifying its name, description, price, stock, and category (similar to Assignment 2). A drop-down list should be used to select the category containing all the vendor's supply list categories. This drop-down list must be updated whenever a new category is added. When a new item is added to the supply list, the page should be updated to display the new item under the proper category. You must ensure that all supply list items continue to have unique IDs. You do NOT need to ensure that the names of items are unique. Note that the data does NOT need to be updated on the server immediately. The changes will be local to the client until the **Save** button (described below) is pressed and the server updates its state.

f. A **Save** button must be provided. When clicked, the modified vendor data should be sent to the server. The server should update the information stored for that vendor and respond to confirm success. When a successful response is received by the client, it should display an alert to the user indicating the changes have been saved on the server.

6. **PUT `/vendors/`*`:vendorID`*** – This route will accept a JSON body matching the format of the data used so far in the course. 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 representation of the vendor with the given ID (i.e., the vendor with ID = ***`:vendorID`***) and may then send a blank response to confirm the changes have been made. If the provided ID does not match any of the vendors on the server, a 404 response should be sent.

Code Quality and Documentation (15 marks)

Your code should be well-written and easy to understand. This includes providing clear documentation explaining the purpose and function of pieces of your code. You should use good variable/function names that make your code easier to read. You should do your best to avoid unnecessary computation and ensure that your code runs smoothly throughout the operation.

You should also include a README.txt file that explains any design decisions that you made, precise instructions detailing how to run your server, as well as any additional instructions that may be helpful to the TA.

Assignment 3 – due Monday, November 14, 23:59

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 submit your assignment, you must **zip** all your files and submit them to the Assignment 3 submission on Brightspace. Do not submit the **node_modules** directory; otherwise, you will receive a **deduction of 10 points**. You should use NPM to manage your project's dependencies and submit the necessary **package.json** file so TAs can install your assignment dependencies using the command **npm install**. You must provide a **README.txt** file explaining the steps required to build/run your submission. Also, let us know whether or not you implemented part 5d.

Name your .zip file “YourName-a3.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. The TA should be able to start your **server.js** server and interact with your app after typing **http://localhost:3000** in a browser without any additional setup.