Assignment 3

Express Restaurant Manager

Submit a single zip file called **assignment3.zip**.
This assignment has 100 marks.
You should read the marking scheme posted on cuLearn for details.

Assignment Background

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

To start this assignment, download the A3-Data.zip file from cuLearn. This contains a restaurant directory with 3 restaurant files, similar to assignment #2. The main difference in these files is that each restaurant now has an ID property. This can be used to reference the unique restaurants on the server. The data for the provided restaurants should be loaded into your server when it starts. Additional restaurants 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.

You should not submit the `node_modules` directory with this assignment. Instead, 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 should still provide a README.txt file explaining the steps required to build/run your submission.

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: `/`, `/restaurants`, and `/addrestaurant`.

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 `/restaurants` – If the client is requesting HTML, the server must respond with an HTML page containing a list of all restaurants stored on the server. This page must contain the name of each restaurant in a list, and each name in the list should be a link to that restaurant's specific URL (i.e., /restaurants/*thatRestaurantsUniqueID*). 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 restaurant IDs): {"restaurants" : *array* }
3. GET `/addrestaurant` – Should respond with an HTML page which provides functionality to add a new restaurant to the server by making a POST request to the resource `/restaurants`. The page must allow the user to enter the name, delivery fee, and minimum order for a new restaurant. 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 `/restaurants` below. When the response from the server is received indicating that the restaurant has been added, the browser should be redirected to the page to view the newly created restaurant.
4. POST `/restaurants` – Accepts a JSON encoding of a new restaurant with the following format (where italics represent some value for that new restaurant): {"name": *restaurantName*, "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 restaurant to the server's data. Initially, the menu of the new restaurant should be blank. The new restaurant data does not need to be stored in a file (i.e., it can be stored in RAM). Each restaurant 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 restaurant that does not conflict with any other restaurant's ID. The response sent by the server should contain the JSON representation of the newly created restaurant.

5. GET `/restaurants/:restID` – This parameterized route should respond with either HTML or JSON data, depending on the value of the Accept header of the request. If the request is for JSON data, the entire restaurant object with the given ID (i.e., the restaurant with ID = *:restID*) 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 restaurant's name, delivery fee, and minimum order.
    b. The page must show the entire menu of the restaurant, divided into categories. Each menu item must be shown within the correct category and show the ID, name, description, and price of the item.
    c. The user must be able to modify the name, delivery fee, and minimum order of the restaurant. A straightforward way to support this is to place the information in text fields.
    d. The user must be able to add a new category to the menu 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 menu. When the category is added to the menu, 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 menu item to the menu by specifying its name, description, price, and category. To select the category, a drop-down list should be used that contains all the restaurant's menu categories. This drop-down list must be updated whenever a new category is added. When the new item is added to the menu, the page should be updated to display the new item under the proper category. You must ensure that all menu items continue to have unique IDs. You do NOT need to ensure that 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 restaurant data should be sent to the server. The server should update the information stored for that restaurant 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 `/restaurants/:restID` – This route will accept a JSON body matching the format of the data used so far in the course. You 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 restaurant with the given ID (i.e., the restaurant with ID = *:restID*) and may then send a blank response to confirm the changes have been made. If the provided ID does not match any of the restaurants 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 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.**

Recap

Your zip file should contain all the resources required for your assignment to be installed and run by the TA.
Submit your **assignment3.zip** file to cuLearn.
Make sure you download the zip after submitting and verify the file contents.
