Homework 6 - Node

In this assignment, you will use Node.js and Express to develop a Web API that provides services related to data stored in a MongoDB database.

This assignment continues the “pet store” theme from previous assignments. In this case, you will build a Web API that allows applications to get information about the pets at the store and the pet toys that the store sells.

In completing this assignment, you will:

- Learn how to set up Node, Express, Mongo, and related packages
- Apply what you have learned about developing a Node Express app and using various objects and functions
- Implement JavaScript queries using Mongoose to retrieve data from a MongoDB database
- Create a server-side Web application that reads data from an incoming HTTP request and sends back JSON data in an HTTP response

Debugging/Error Note:

If you run into errors/bugs/don't understand the output that Codio is giving you, please post in the Discussion Forum and a TA will assist you! Please do NOT email Codio as they will not review any errors you are getting.

Getting Started

To complete this assignment, you will need to set up a development environment that uses Node, Express, Mongoose, and MongoDB. We recommend that you do this on your local computer as follows; you will likely need sudo/root access for each of these steps:

1. Install Node.js locally by downloading it from https://nodejs.org/en/download/>

2. From the Terminal, Command Prompt, etc. update Node Package Manager by typing the command: **`npm install npm –g`**

3. Create a new folder or directory for your project, then navigate to it using Terminal, Command Prompt, etc.

4. Initialize the project by typing the command: **`npm init`**

5. Install Express by typing the command: **`npm install express --save`**

6. Install Mongoose by typing the command: **`npm install mongoose --save`**

7. If you would like to install MongoDB locally, download the "Community Server" version from https://www.mongodb.com/download-center#community >and follow the instructions for installing it and running it; alternatively, you can create an account to use a cloud service such as MongoDB Atlas (https://www.mongodb.com/cloud/atlas>); either way, be sure to note the hostname and port number of where your database instance is running

Once you've set up your environment, download the three files you will need for this assignment:

- >right-click **this link** and save "index.js" to your computer
- >right-click **this link** and save "Animal.js" to your computer
- >right-click **this link** and save "Toy.js" to your computer

Copy all three files> to the directory that is the root of your Node Express project, i.e. the directory where you ran “npm init”.

Having trouble setting up your environment?

You can do this assignment using the Codio grading platform, which we have configured with Node, Express, Mongoose, and MongoDB.

Click the "BEGIN SUBMISSION" button at the bottom of the screen, which will take you to Codio.

Once you're there, use the "Filetree" on the left side of the screen to expand the "dev" folder, then click "Instructions.md", which will open up the file that includes instructions for completing this assignment on Codio.

The index.js file is the main file for your Node Express app. It includes the necessary packages, sets up the web server to listen on port 3000, and defines a route that sends back a simple JSON object for all HTTP requests.

To ensure that your setup works correctly, use Terminal or Command Prompt to navigate to the root directory of your Node Express project and type the command:
`node index.js`

You should see the message “Listening on port 3000”.

Then use your web browser to access **http://localhost:3000**> and you should see a JSON object that reads `{"msg":"It works!"}` in the browser.

Animal.js and Toy.js define the schema that you will need for your Mongo database. You may modify the host/port/database configuration that is passed to `mongoose.connect`> for your particular environment, but do not change the Schema definitions as these will be used by the grader when you submit.

Activity

In this assignment you will implement four Web APIs using Node and Express to handle the HTTP requests and responses, and Mongoose to handle the interaction with the MongoDB database.

The specifications of the APIs are provided below. We recommend that you attempt them in the order in which they are described. Note that in all but the first instance, your code will need to construct the JSON object that is sent back in the response, as opposed to just sending back the data that comes out of MongoDB.

`/findToy`

Parameters:

- **`id`: >the ID of the Toy to find**

Example usage: **`/findToy?id=1234`**

Description:

- This API finds and returns the Toy in the "toys" collection with the ID that matches the `id`> parameter. It should return the entire Toy document/object, including all properties that are stored in the database.

- If the `id`> parameter is unspecified, or if there is no Toy that has a matching ID, this API should return an empty object.

`/findAnimals`

Parameters:

- **`species`: >the species of the Animals to find**
- **`trait`: >one of the traits of the Animals to find**
- **`gender`: >the gender of the Animals to find**

Example usage: **`/findAnimals?species=Dog&trait=loyal&gender=female`**

Description:

- This API finds all Animals in the "animals" collection that have a species and gender that match the `species`> and `gender`> parameters, respectively, and for which one of the Animal's traits matches the `trait`> parameter. All matches should be complete matches, not partial matches using regular expressions, for instance.

- The return value is an array of objects representing each Animal, but the object must *only*> include the Animal's name, species, breed, gender, and age.

- If the `species`>, `trait`>, or `gender`> parameter is unspecified, it should be ignored in the search, i.e. the API should consider Animals regardless of their values for the unspecified parameter(s)

- However, if the `species`>, `trait`>, and `gender`> parameters are all unspecified, then the API should return an empty object.

- Likewise, if there are no Animals that match all of the specified parameters, the API should return an empty object.

Examples: Consider the following collection of Animals:

name	species	Breed	gender	age	traits
>“Cooper”	>“Dog”	>“Catahoula”	>“male”	>11	>[“crazy”, “funny”]
>“Lola”	>“Dog”	>“Beagle”	>“female”	>5	>[“loyal”, “friendly”]
>“Garfield”	>“Cat”	>“Tabby”	>“male”	>39	>[“lazy”, “hungry”]
>“Felix”	>“Cat”	>“Tuxedo”	>“male”	>98	>[“funny”, “loyal”]

Example #1

Request: **`/findAnimals?species=Dog&trait=loyal&gender=female`**

Response:
```
[{"name":"Lola","species":"Dog","breed":"Beagle","gender":"fem
ale","age":5}]
```

Example #2

Request: **`/findAnimals?trait=funny`**

Response:
```
[{"name":"Cooper","species":"Dog","breed":"Catahoula","gender"
:"male","age":11},
{"name":"Felix","species":"Cat","breed":"Tuxedo","gender":"mal
e","age":98}]
```

`/animalsYoungerThan`

Parameters:

- **`age`: >the maximum age (exclusive) of the Animals to find**

Example usage: **`/animalsYoungerThan?age=12`**

Description:

- This API finds all Animals in the “animals” collection that have an age that is less than (but not equal to!) the `age`> parameter.

- The return value is an object that has two properties:
  - “count”: the number of Animals whose age is less than the `age`> parameter.
  - “names”: an array containing the names of the Animals whose age is less than the `age`> parameter (these can be arranged in any order in the array)
- If there are no Animals that have an age less than the `age`> parameter, then the API should return an object that has a “count” property set to 0, but no “names” property
- If the `age`> parameter is unspecified or non-numeric, then the API should return an empty object

`/calculatePrice`

Parameters:

- **`id[i]` : >the ID of the *i*>th Toy to include in the calculation**
- **`qty[i]` : >the quantity of the *i*>th Toy to include in the calculation**

Example usage: **`/calculatePrice?id[0]=123&qty[0]=2&id[1]=456&qty[1]=3`**

Description:

- This API calculates the total price of purchasing the specified quantities of the Toys with the corresponding IDs, using the Toys’ price from the database.
- For each *i*, >this API finds the Toy with the ID equal to the `id[i]`> parameter and determines the subtotal for that Toy by multiplying its price by the specified quantity `qty[i]`>. It then uses the subtotals for all Toys to calculate the total price.
- The return value is an object that has two properties:
  - “totalPrice”: the calculated total for all Toys.
  - “items”: an array containing objects that hold information about the Toys that are included; for each Toy, there should be an object with these three properties:
    - “item”: the Toy’s ID, as specified in the query
    - “qty”: the quantity of the Toy, as specified in the query

  - "subtotal": the Toy's price multiplied by the quantity

- If an `id`> parameter does not correspond to the ID of a Toy in the database, then that ID and the corresponding quantity should be ignored

- If a `qty`> parameter is less than one or non-numeric, then it and the corresponding `id`> parameter should be ignored

- If the same `id`> parameter is specified more than once, then the total quantity for that Toy should be considered as the sum of all corresponding `qty`> parameters. However, if any such `qty`> parameter is less than one or non-numeric, that parameter should be ignored.

- If all `id`>/`qty`> parameters are to be ignored because of the above, then the API should return an object with "totalPrice" set to 0 and "items" set to an empty array

- The API should return an empty object if:

  - There are no query parameters specified in the request

  - The number of `id`> parameters does not match the number of `qty`> parameters

Examples: Consider the following collection of Toys:

ID	Name	Price
>"123"	>"Dog chew toy"	>10.99
>"456"	>"Dog pillow"	>25.99

Example #1

Request: **`/calculatePrice?id[0]=123&qty[0]=2&id[1]=456&qty[1]=3`**

Response: `{"items":[{"item":"123","qty":"2","subtotal":21.98},{"item":"456","qty":"3","subtotal":77.97}],"totalPrice":99.95}`

Example #2

Request: **`/calculatePrice?id[0]=123&qty[0]=1&id[1]=xxxx&qty[1]=3`**

Response: `{"items":[{"item":"123","qty":"1","subtotal":10.99}],> "totalPrice":10.99}`

Example #3

Request: **`/calculatePrice?id[0]=abc&qty[0]=1&id[1]=456&qty[1]=dog`**

Response: `{"items":[],"totalPrice":0}`

What about...?

You may encounter other cases that are not addressed in this document. You may handle those cases in any manner you choose (or ignore them entirely!) since they will not be considered for grading in this assignment.

One more important note:

As mentioned above, please do not change the Schema definitions in Animal.js or Toy.js, as these will be used by the automatic grader. You may, however, change the host/port/database configuration, since the grader will replace these with its own settings.

Helpful Hints

Review the past few lessons to see the syntax for setting up routes in your Express app and for using Mongoose to perform queries on your MongoDB database.

If you start your Node Express app and immediately see an error message such as:

```
MongoError: failed to connect to server [localhost:27017] on first connect [MongoError: connect ECONNREFUSED 127.0.0.1:27017]
```

This means that either your MongoDB database is not running or that you have specified the incorrect host/port combination.

For the `/animalsYoungerThan` API, you can do a “less than” search by specifying a query object in the form **`{`*`field`*`:{$lt:`*`value`*`}}`**. See https://docs.mongodb.com/manual/reference/operator/query/lt/ for more information.

For the `/calculatePrice` API, note that a URL query string such as

```
?id[0]=123&qty[0]=2&id[1]=456&qty[1]=3
```

will result in the HTTP Request object's "query" property having the properties "id" and "qty", both of which are arrays. You can then use those arrays to find the IDs and quantities that you need. You may want to consider using some of the ES6 data structures we saw last week, such as Maps and Sets, to relate IDs to their corresponding quantities.

Last, and perhaps most importantly, for the `/calculatePrice` API, try to avoid performing multiple queries on one request, e.g. calling `Toy.find` in a loop for each ID, but rather figure out how to construct a single query object that only calls `Toy.find` once. Since the callback method that is invoked as a result of the query is asynchronous, you may get unexpected results if you try to use `Toy.find` in a loop.

Before You Submit

Please be sure that:

- You have not modified the Schema definitions in Animal.js or Toy.js
- You have not created any additional files and all of your code is in index.js

Assessment

Your submission will be assessed using automatic grading scripts that will check that the API produces the correct return value for various inputs. Your score is determined by the percentage of these tests that "pass," i.e. that produce the correct output for the specified input.

To submit your assignment, click the "BEGIN SUBMISSION" button below and follow the instructions in the Codio Readme file. You only need to upload **index.js** for grading.

Homework #6 Submission (External resource)

(100.0 / 100.0 points)

This link will take you to Codio so that you may submit this assignment.

BEGIN SUBMISSION

Learn About Verified Certificates