ASSIGNMENT 3:
USING DATABASE VIA NODE.JS

RELEASED: **23 NOV 2022**
DUE: **15 DEC 2022 23:59**

SYNOPSIS

You are going to store and manipulate some data in a database. Build a RESTful API to serve HTTP responses for ***creating***, ***retrieving***, ***updating***, and ***deleting*** data.

BASIC SETUP

During *Lab 7*, you have learnt to set up a basic web server using ***Node.js*** and ***Express***. In *Lab 8*, you have further set up a ***MongoDB*** database on the MongoDB Atlas Cloud, and access it using ***Mongoose*** in Node.js.

DATABASE SCHEMA (20%)

Schemas and models in Mongoose help to confine the types of data to be stored. These two schemas are needed in this assignment:

Event

- **`eventId`**: number, required, unique
- **`name`**: string, required
- **`loc`**: *ObjectId → Location documents**
- **`quota`**: number

Location

- **`locId`**: number, required, unique
- **`name`**: string, required
- **`quota`**: number

** The Event schema has been defined in Lab 8, but some amendment is needed to accommodate the reference to documents in the* ***locations*** *collection. You should carefully read and understand the lecture slides on "****Documents in another collection****" for MongoDB.*

GET AND POST (30%)

(Done in Lab 8 but improvement needed) Your Express server should respond to these requests:

- *Q1:* **`GET http://`***server address***`/ev/`***event ID*

When the server receives this request, look up the event with the given event ID from the database. Then print the event *name*, *location ID*, *location name*, *event quota* on separate lines, as shown in the illustration on the right. The location quota is *not* necessary. All field names and string values should be quoted with double quotes. Extra spacing is allowed, and brackets/commas do no need to be in separate lines.

```
{
"eventId": 123,
"name": "CSCI2720 lab",
"loc":
{
"locId": 1,
"name": "SHB924"
},
"quota": 2
}
```

If the given event ID is not found, output an understandable message in the response body with status code **`404`**. All the response should be sent as HTTP responses with content type `text/plain`.

- *Q2:* **`POST http://`***server address***`/ev`**

When the server receives this request, it should use the parameters submitted in the HTTP request body to create a new event in the database. A simple HTML form like this can be used: *http://www.cse.cuhk.edu.hk/~chuckjee/2720lab8/form.html*

However, instead of letting the user decide the event ID, your code should look into the database to find the current maximum event ID, e.g. ***x***. Assign ***x+1*** as the new event ID. The form should ask the user for the location ID only but not the location name. A lookup should be done to check if the location quota is larger than or equal to the new event quota. If not, the event should not be created, and an error message should be responded to the user with status code **`406`**. *(Note: There is no need to check other events at the same location.)*

If the event is created successfully, respond to the user with the URL of the created event. Use HTTP status code **`201`**.

DELETE (10%)

We allow users to delete an event using this request:

- *Q3:* **`DELETE http://`***server address*`/ev/`***event ID***

If the event ID is found, the event should be removed from the database. Send a response with status code **`204`**, and nothing in the body. If not found, show a simple error message in the body with status code **`404`**.

You may utilize the software ***Postman*** (*https://www.postman.com*) for making **`DELETE`** requests to your Express server during your development. Of course, you may also build user pages to send it using JavaScript. This JS is not part of the assignment submission.

SOME MORE QUERIES (20%)

The following requests needs to be handled:

- *Q4:* **`GET http://`***server address*`/ev`**

List all the events available, with details formatted similar to the first illustration on the right.

- *Q5:* **`GET http://`***server address*`/lo/`***location ID***

Show the details for this location ID, with details formatted similar to the second illustration on the right. Respond with an error message if the location ID is not found with status code **`404`**.

- *Q6:* **`GET http://`***server address*`/lo`**

List all locations available. Show details similarly as for multiple events.

- *Q7:* **`GET http://`***server address*`/ev?q=`***number***

List all the events with quota of at least this number. Show details similarly as for multiple events. Respond with an empty array **`[ ]`** if there is none.

```
[
{
"eventId": 123,
"name": "CSCI2720 lab",
"loc":
{
"locId": 1,
"name": "SHB924"
},
"quota": 2
}
,
{
"eventId": 124,
"name": "CSCI2720 lab 2",
"loc":
{
"locId": 1,
"name": "SHB924"
},
"quota": 2
}
]
```

```
{
"locId": 1,
"name": "SHB924",
"quota": 100
}
```

UPDATING WITH PUT (20%)

Implement a simple HTML user interface for ***updating*** event information. Assume that the user always type an ***existing event ID*** to load the event information. The current data from the database should be shown to the user, so that they can decide what to edit, in an HTML form. Then update the database with the user data on submission, using this request with JavaScript:

- *Q8:* **`PUT http://`***server address*`/ev/`***event ID***

The event should always be updated successfully. Respond to the user with the event details as in the **`GET`** response. You may combine this HTML form with the POST form, or use a new HTML file. Checking of event and location quota is not necessary here.

FORMATS AND ASSUMPTIONS

Please observe these for all the response made by Express in this assignment:

1. In this assignment, since the response body do not always conform to standard JSON formats, e.g., for error messages, please always use `text/plain` as the `Content-Type`.
2. The JSON-like objects and arrays should be readable by standard JavaScript `JSON.parse()` if all newline characters are removed. The syntax of quotes is tricky.
3. Use the cors middleware for Express to enable submission from local HTML forms.
   ```
   const cors = require('cors'); app.use(cors());
   ```
4. Unless otherwise stated, the HTTP status code would be **`200 OK`** by default. **`201 Created`** is used for responding to successful POST requests, and **`204 No Content`** is used for responding to successful DELETE requests.
5. For development, you may feel free to populate your database with testing data.
6. For grading, graders will use another set of data in their database. Assume that there is always at least *one location* and *one event* in the database when grading. All locations linked from events are valid in the database.
7. Assume that the input data type is always correct, i.e., only numbers will be input for number fields. All numbers are positive numbers > 0. When grading, no fields are blank.
8. Assume that user always enters a valid and existing location ID when creating/updating events.
9. Events/locations with missing quota will not be tested.

10. The Node server should start at port 3000. Assume the access URL starts with `http://localhost:3000`
11. *We will not test for situations not mentioned in this document.*

CHALLENGE REQUIREMENTS (20%)

For ESTR2106 students: This part is compulsory. This assignment has a full score at 120%.

For CSCI2720 students: If you finish this part correctly, a 0.5% bonus will be given to the "Assignment" course assessment, which could go beyond its original 30% of assessments.

At "/" (root path), show a short paragraph to briefly describe the work here, with ~100 words.

1. In every `GET` queries above (Q1, Q4–7), add an extra field `"query_no"`, which indicates the total count of `GET` queries for this user on this browser, including requests giving errors. Show it only in the response when there is no error. The count should be reset if there is no further query in 5 minutes.
2. In the same Node.js/Express implementation, add a GraphQL API at the endpoint `/graphql` for these queries for data fetching only, accessible via GET or POST:
   `events`, `event(id:5)` , `locations`, `location(id:5)` *where the id 5 is only an example*

FEATURES AND MODULES

There is no cosmetic requirement for this assignment. You are welcome to implement extra styles and features at your own ability, when complying to requirements above. Besides ***Express*** and ***Mongoose***, you can use any Node.js module at your discretion, but ***please only use one JS file.***

SUBMISSION

You may test your work using ***Google Chrome*** (almost-latest versions) and the software ***Postman***. Please utilize the ***Developer Tools*** for debugging of your code. For grading, we may use other programmatic interfaces.

Plagiarism is heavily penalized. Do not attempt to share any code other than those given in labs. Please read this article carefully: *http://www.cuhk.edu.hk/policy/academichonesty*

Include your full name and student ID in ***all code files*** using comments, together with the required ***code header for honesty declaration***. Check that the file names and formats are correct. Zip your

files, name it as `(SID)_asg3.zip`, e.g., *`1155000001_asg3.zip`*, and submit it on the course site on Blackboard. ***Only these files should be included:***

- 1 `.js` file
- A number of HTML files, named appropriately to show their use
- `package.json` and `package-lock.json` from Node.js

DO NOT submit the **`node_modules`** ***folder.***

There is a late penalty of 10% in the first 72 hours of lateness. In the 72 hours after that, the penalty is 30%. The work will not be graded after more than 144 hours of lateness.

REFERENCE POINTS

Not every feature of the frameworks/libraries have been covered in class. Check the docs for details!

Express: *https://expressjs.com/en/4x/api.html* Mongoose: *https://mongoosejs.com/docs/api.html*