COMP3322 Modern Technologies on World Wide Web

Assignment Five

Total 12 points

Deadline: 17:00 May 2, 2023

Overview

Write an express.js program and name it **index.js**. This program provides the API to get/add weather data from/to a MongoDB server.

Objectives

1. A learning activity to support ILO 1 and ILO 2.
2. To practice how to use Node, Express, MongoDB, and Mongoose to create a simple REST API.

Specification

Assume you are using the MongoDB server running on **the course's node.js docker container** with the service name ***mongodb*** listening to the default port 27017.

The name of the weather data database is **weather**, and the name of the collection is **wrecords**. The collection consists of HK weather data and each document contains 6 fields (not including the _id field), which are: date, meanT, maxT, minT, humidity, and rain.

	date	**meanT**	**maxT**	**minT**	**humidity**	**rain**
	the date of the weather record (YYYYMMDD)	the average temperature of that date	the maximum temperature of that date (°C)	the minimum temperature of that date (°C)	the average humidity of that date (%)	the average rainfall of that date (mm)
Example 1	20221116	24.1	25.8	23.2	80	0
Example 2	20220904	28.4	29.6	26.7	81	8.6

You can download 2022 weather data (HKO_2022.csv) from the course's Moodle site. **Import the data to the weather database for the tests.**

Use the following framework for developing your program. You can download a template file (template.txt) from the course's Moodle site.

index.js

```
const express = require('express')
const app = express();




    /* Implement the logic here */
```

```
// error handler
app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.json({'error': err.message});
});

app.listen(8000, () => {
  console.log('Weather app listening on port 8000!')
});
```

TASK A

Use the command **mongoimport** to import the CSV file to the MongoDB server. Here are the steps to import the data to your docker's mongodb server.

1. Use Windows Explorer or Mac Finder to go to the data/db folder (which is inside the Node-dev folder).
2. Copy the HKO_2022.csv file there.
3. Access the docker desktop and open a terminal for the c33322-mongo container.
4. In the terminal, type this command:

```
mongoimport -d=weather -c=wrecords --type=csv --headerline
 --columnsHaveTypes --file=HKO_2022.csv
```

Write the code to set up a connection to the MongoDB server **using Mongoose**.
Use the following schema to access the database.

```
Schema {
  date: String,
  meanT: Number,
  maxT: Number,
  minT: Number,
  humidity: Number,
  rain: Number
}
```

Write the code to **monitor** the database connection and **terminate** the program **if lost** the database connection.

TASK B

Write **an endpoint** to handle all POST requests to the URI

http://localhost:8000/weather/YYYY/MM/DD

where YYYY stands for the year, MM stands for the month, and DD stands for the date. For example, **to add** the weather data for the date 2023-1-14 (14th January 2023), the browser sends this request: `POST /weather/2023/1/14 HTTP/1.1` or `POST /weather/2023/01/14 HTTP/1.1` to the server. The request message carries the weather data for 14th January 2023 in a **JSON string** in the message body.

```
'{"meanT":22.7,"maxT":24.7,"minT":20,"humidity":90,"rain":3.4}'
```

In response, the server returns a JSON string and an appropriate HTTP status code to the client, which reflects the completion status of the POST request.

Situations:

1. When the system checks that the path represents an incorrect date, it returns a JSON string `'{"error":"not a valid year/month/date"}'` with the HTTP status code **400**, e.g., 202/12/23, 22/106/07, 2022/November/17, 2023/2/30, 2023/03/33, etc.
2. When the system checks that the path represents a correct date, but the system already has a record for that date, it returns a JSON string `'{"error":"find an existing record. Cannot override!!"}'` with the HTTP status code **403**.
3. When the system checks that the path represents a correct date and does not have an existing record for that date, it adds this record to the database and returns a JSON string `'{"okay":"record added"}'` with the HTTP status code **200**.
4. When the program experiences an error (e.g., database issue, query error, etc), it returns the HTTP status code **500** with a JSON string `'{"error":"system error"}'` or `'{"error":$message}'`, where $message stands for the error message of that event.

TASK C

Write **another endpoint** to handle all GET requests to the URI

http://localhost:8000/weather/YYYY/MM/DD

for retrieving the weather data of the specified date. For example, to get the weather data for the date 2023-2-4 (4th February 2023), the browser sends this request: `GET /weather/2023/2/4 HTTP/1.1` or `GET /weather/2023/02/04 HTTP/1.1` to the server.

In response, the server returns a JSON string and an appropriate HTTP status code to the client, which reflects the completion status of the GET request.

Situations:

1. When the system checks that the path represents an incorrect date, it returns a JSON string `'{"error":"not a valid year/month/date"}'` with the HTTP status code **400**.
2. When the system checks that the path represents a correct date, but the system cannot find a record for that date, it returns a JSON string `'{"error":"not found"}'` with the HTTP status code **404**. For example, a request `GET /weather/2024/12/4 HTTP/1.1` for a future date.
3. When the system checks that the path represents a correct date and successfully retrieves the record for that date, it returns a JSON string containing the weather data with the HTTP status code **200**, e.g., `'{"Year":2023,"Month":2,"Date":4,"Avg Temp":14.4,"Max Temp":18.5,"Min Temp":11.9,"Humidity":69,"Rainfall":0}'`.
4. When the program experiences an error (e.g., database issue, query error, etc), it returns the HTTP status code **500** with a JSON string `'{"error":"system error"}'` or `'{"error":$message}'`, where $message stands for the error message of that event.

TASK D

Write **another endpoint** to handle all GET **requests** to the URIs

http://localhost:8000/weather/temp/YYYY/MM
http://localhost:8000/weather/humi/YYYY/MM
http://localhost:8000/weather/rain/YYYY/MM

for retrieving the temperature or humidity or rainfall **summary data** for the month YYYY-MM. For

example, to get the summary temperature data for the month of 2022/11, the browser sends this request: `GET /weather/temp/2022/11 HTTP/1.1` to the server. Similarly, the browser sends the requests: `GET /weather/humi/2022/11 HTTP/1.1` and `GET /weather/rain/2022/11 HTTP/1.1` to the server for getting the summary humidity and rainfall data respectively.

In response, the server returns a JSON string and an appropriate HTTP status code to the client, which reflects the completion status of the GET request.

Situations:

1. When the system checks that /YYYY/MM represents an incorrect year or month, it returns a JSON string `'{"error":"not a valid year/month"}'` with the HTTP status code **400**.
2. When the system checks that /YYYY/MM represents correct the year and month, but the system cannot find any data associated with that month YYYY-MM, it returns a JSON string `'{"error":"not found"}'` with the HTTP status code **404**. For example, a request `GET /weather/temp/2024/11 HTTP/1.1` for a future month.
3. When the system checks that /YYYY/MM represents correct the year and month, and it successfully retrieves all records for that month YYYY-MM, it returns a JSON string with the HTTP status code **200**. Depending on the path, the returned JSON string contains a different data set.
   For the path /weather/temp/YYYY/MM, e.g., /weather/temp/2022/11, the system returns the average temperature of that month (by **taking the mean of the average temperature** of each date in that month), the maximum temperature of that month (by finding the maximum temperature in all records of that month), and the minimum temperature of that month (by finding the minimum temperature in all records of that month). Here is the result JSON string for 2022-11:
   `'{"Year":2022,"Month":11,"Avg Temp":23.4,"Max Temp":28.6,"Min Temp":18.3}'`
   For the path /weather/humi/YYYY/MM, e.g., /weather/humi/2022/11, the system returns the average humidity of that month, the maximum humidity of that month, and the minimum humidity of that month. Here is the result JSON string for 2022-11:
   `'{"Year":2022,"Month":11,"Avg Humidity":82.67,"Max Humidity":93,"Min Humidity":64}'`
   For the path /weather/rain/YYYY/MM, e.g., /weather/rain/2022/11, the system returns the average rainfall of that month and the maximum daily rainfall of that month. Here is the result JSON string for 2022-11:
   `'{"Year":2022,"Month":11,"Avg Rainfall":4.36,"Max Daily Rainfall":58.1}'`
   If the data set does not have the data for the whole month, the system just calculates the average/maximum/minimum according to the available data.
4. When the program experiences an error (e.g., database issue, query error, etc), it returns the HTTP status code **500** with a JSON string `'{"error":"system error"}'` or `'{"error":$message}'`, where $message stands for the error message of that event.

TASK E

Write **an endpoint** to intercept all other request types and paths, which are not defined in previous tasks. Return a JSON string with the HTTP status code **400**. For example, for the request `GET /weather/humidity HTTP/1.1`, we get the response `'{"error":"Cannot GET /weather/humidity"}'`; for the request `DELETE /weather/2022/12/3 HTTP/1.1`, we get the response `'{"error":"Cannot DELETE /weather/2022/12/3"}'`.

Resources

You are provided with the following files.

- `template.txt` – the framework for the index.js file.
- `HKO_2022.csv` – 2022 weather data set.
- `HKO_2023_01.txt` – some data from January 2023; this data set is for your testing.
- `curl-cmd.txt` – contains example curl commands for accessing/testing the API.

Testing platform

We shall run the server program in the node-dev container set and use curl and Firefox to test the API.

Submission

Please finish this assignment before 17:00 May 2, 2023 Tuesday. Submit the following files:

1. A JSON file – use mongoexport to export the whole wrecords collection from the weather database.
   Similar to the mongoimport command, you have to open a terminal at the data/db folder and type the following command:
   ```
   mongoexport -d=weather -c=wrecords --jsonArray --sort='{date: 1}' --out=3035111999.json
   ```
   Replace 3035111999 with your student ID and upload this JSON file.
2. The complete index.js program and other required files.
3. The package.json file.

Grading Policy

Points	Criteria
2.0	Task A ▪ Database set up, import the data set, and export the data set. ▪ The program can connect and access the MongoDB database. ▪ The program can detect that the database connection is broken.
3.0	Task B ▪ Correctly handle the POST request for adding a new record to the database ▪ Error handling
3.0	Task C ▪ Correctly handle the GET request for retrieving the data for the date YYYY/MM/DD ▪ Error handling
3.0	Task D ▪ Correct handle the GET request for retrieving the summary temperature/humidity/rainfall data for the month YYYY/MM ▪ Error handling
1.0	Task E ▪ Error handling of all unknown methods and paths
-4.0	Using any external libraries.

Plagiarism

Plagiarism is a very serious academic offence. Students should understand what constitutes plagiarism, the consequences of committing an offence of plagiarism, and how to avoid it. ***Please***

note that we may request you to explain to us how your program is functioning as well as we may also make use of software tools to detect software plagiarism.