COMP3322 Modern Technologies on World Wide Web

Assignment Four

Total 14 points

Deadline: 23:59 May 3, 2025

Overview

Create an Express.js program named index.js that serves APIs for users to retrieve data from a MongoDB server containing daily statistics about passengers who have travelled across Hong Kong control points.

Objectives

1. A learning activity to support ILO 1 and ILO 2.
2. To practice using Node, Express, MongoDB, and Mongoose to create a simple RESTful Web Service.

Specification

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

The "**DailyFlow**" database contains daily statistics on passenger travel across control points in Hong Kong. The data, sourced from the Hong Kong Government Immigration Department, includes daily inbound and outbound trips since 2021, categorized by type of traveller such as Hong Kong Residents, Mainland Visitors, and Other Visitors.

https://data.gov.hk/en-data/dataset/hk-immd-set5-statistics-daily-passenger-traffic

The daily data in the "DailyFlow" database is generated by a data analytics program and is stored in a collection named "**daylog**". Each document in the collection contains six fields: _id, Date, Flow, Local, Mainland, and Others.

Date	Flow	Local	Mainland	Others
1/1/2021	Arrival	732	45	10
1/1/2021	Departure	3446	245	32
1/2/2021	Arrival	942	78	12
1/2/2021	Departure	3903	283	35
1/3/2021	Arrival	1770	42	41
1/3/2021	Departure	4692	373	58
1/4/2021	Arrival	1922	68	16
1/4/2021	Departure	4209	417	47
1/5/2021	Arrival	1323	74	22
1/5/2021	Departure	2381	148	58
1/6/2021	Arrival	1376	64	20
1/6/2021	Departure	2116	133	53

Note: (1) For each date, there are two rows (documents) in the collection. One row represents the incoming (arrival) counts and the other represents the outgoing (departure) counts. (2) The 'Date' field is in the format of m/d/yyyy.

To test your API, we provide the daily dataset containing data from Jan 2021 to Mar 2025. You can download the dataset (**daylog.csv**) from the course's Moodle site. **Import the data to the DailyFlow database for testing purposes.**

We provide a framework for you to develop your program. You can download the template file (template.txt) from the course's Moodle site.

index.js

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




     /* Implement the logic here */















app.listen(8080, () => {
  console.log('ASS4 App listening on port 8080!')
});
```

Task A

Use the command **mongoimport** to import the CSV file into MongoDB. 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 daylog.csv file to there.
3. Access the docker desktop and open a terminal on the **c3322-mongo container**.
4. In the terminal, type this command (in one line):

```
mongoimport -d=DailyFlow -c=daylog --type=csv --headerline --file=daylog.csv
```

Write the code to set up a connection to the MongoDB server **using Mongoose**.
Write the code in your Express app to (i) set up a connection to the MongoDB database using **Mongoose** for accessing the data, and (ii) set up a schema and a model for accessing the "daylog"

collection. Write the code that **monitors** the database connection and **terminates** the program if the connection to the database **is lost**.

Task B

Write an express routing handler to handle all **GET requests** for retrieving the raw data of the passenger flow for a specific date or a range of dates starting from a specific date. The returned data should be formatted as a **JSON string**.

For example, to retrieve the raw data for 30/12/2024, the client should send a GET request to `http://localhost:8080/HKPassenger/v1/data/2024/12/30.` This request returns the following JSON string:

```
[{"Date":"12/30/2024","Flow":"Arrival","Local":391739,"Mainland":89865,"Others":45696},{"Da
te":"12/30/2024","Flow":"Departure","Local":283917,"Mainland":85993,"Others":44449}]
```

To obtain the raw data for a range of dates, the request specifies the starting date with a query string that gives the number of days to retrieve. For example, a GET request to `http://localhost:8080/HKPassenger/v1/data/2024/12/30?num=5` returns the raw data from 30/12/2024 to 3/1/2025. This is the returned JSON string:

```
[{"Date":"12/30/2024","Flow":"Arrival","Local":391739,"Mainland":89865,"Others":45696},{"Da
te":"12/30/2024","Flow":"Departure","Local":283917,"Mainland":85993,"Others":44449},{"Date"
:"12/31/2024","Flow":"Arrival","Local":317739,"Mainland":119631,"Others":36735},{"Date":"12
/31/2024","Flow":"Departure","Local":301109,"Mainland":71413,"Others":32210},{"Date":"1/1/2
025","Flow":"Arrival","Local":393197,"Mainland":106290,"Others":29353},{"Date":"1/1/2025","
Flow":"Departure","Local":221050,"Mainland":151652,"Others":47462},{"Date":"1/2/2025","Flow
":"Arrival","Local":287942,"Mainland":74518,"Others":38535},{"Date":"1/2/2025","Flow":"Depa
rture","Local":247949,"Mainland":72837,"Others":48722},{"Date":"1/3/2025","Flow":"Arrival",
"Local":277067,"Mainland":82626,"Others":38619},{"Date":"1/3/2025","Flow":"Departure","Loca
l":294583,"Mainland":75817,"Others":42695}].
```

This routing handler performs the following actions:

- Only responds to the GET requests.
- To retrieve raw data for a specific date, the system connects to the database to retrieve all fields (excluding the _id field) and returns both the “Arrival” and “Departure” records for that date. **Always** returns the Arrival record first and followed by the Departure record.
- Do the same for retrieving raw data for a range of dates starting from a specific date. If the request extends beyond the last date in the database, the API should return the available raw data up to the last date it has to fulfill the request.
- Send the data as a single JSON string to the client.
- If the request specifies a date that exceeds the last date in the database, the API should return an empty JSON array.

This routing handler performs the following checking.

- If a request contains a year value outside the range of 2021 to 2025, the API should respond with an error JSON string and an HTTP status code of **400**.
  `{"error":"Wrong year input - must be a number between 2021 - 2025."}`
- If a request carries a Month value not in the range between 1 to 12, the API should respond with an error JSON string and an HTTP status code of **400**.

`{"error":"Wrong month input - must be a number between 1 - 12."}`

- If a request carries a date value not in the range between 1 to 31, the API should respond with an error JSON string and an HTTP status code of **400**.
  `{"error":"Wrong date input - must be a number between 1 - 31."}`
- If a request carries a date which is not a valid calendar date, e.g., 30/2/2023, the API should respond with an error JSON string and an HTTP status code of **400**.
  `{"error":"30/2/2023 is not a valid calendar date!"}`
- If a request includes a num query string that is either not a number or is a number less than 1, the API should respond with an error JSON string and an HTTP status code of **400**.
  `{"error":"Wrong query string num - must be a number greater than zero"}`
- [**Optional**] If the program experiences database issues, it should return an error JSON string with an HTTP status code of **500**.
  `{"error":"Experiencing database error!!"}`

TASK C

Write an express routing handler to handle **all GET requests** for retrieving aggregated passenger traffic data of different groups, filtered by a specific year or a specific month within a given year. For example, if a client wishes to access the aggregated **monthly** data for local residents in August 2024, they will send a GET request to
`http://localhost:8080/HKPassenger/v1/aggregate/local/2024/8`.
To retrieve the **yearly** aggregated data for **mainland** visitors in **2024**, the client would send the GET request to
`http://localhost:8080/HKPassenger/v1/aggregate/mainland/2024/`.
To retrieve the **yearly** aggregated data for **all groups** of cross-border passengers in **2023**, the client would send the GET request to
`http://localhost:8080/HKPassenger/v1/aggregate/all/2023/`.

This express routing handler supports fetching aggregated data for the passenger groups categorized as **'local', 'mainland', 'others', or 'all'** within any specific year from 2021-2025, with the option to specify a particular month.

The returned data should be formatted as a **JSON string** and ordered by month for yearly aggregated data or by day for monthly aggregated data.

Here is an example JSON string returned for the request
`http://localhost:8080/HKPassenger/v1/aggregate/local/2024/8`

```
[{"Date":"8/1/2024","Local":-20781},{"Date":"8/2/2024","Local":-66303},
{"Date":"8/3/2024","Local":-124934},{"Date":"8/4/2024","Local":143651},
{"Date":"8/5/2024","Local":46081},{"Date":"8/6/2024","Local":19070},
{"Date":"8/7/2024","Local":3305},{"Date":"8/8/2024","Local":-7833},
{"Date":"8/9/2024","Local":-60304},{"Date":"8/10/2024","Local":-122332},
{"Date":"8/11/2024","Local":162303},{"Date":"8/12/2024","Local":63863},
{"Date":"8/13/2024","Local":31829},{"Date":"8/14/2024","Local":10960},
{"Date":"8/15/2024","Local":-2919},{"Date":"8/16/2024","Local":-53868},
{"Date":"8/17/2024","Local":-100075},{"Date":"8/18/2024","Local":139598},
{"Date":"8/19/2024","Local":46160},{"Date":"8/20/2024","Local":21155},
{"Date":"8/21/2024","Local":6550},{"Date":"8/22/2024","Local":-5462},
{"Date":"8/23/2024","Local":-58546},{"Date":"8/24/2024","Local":-129487},
{"Date":"8/25/2024","Local":162539},{"Date":"8/26/2024","Local":65056},
{"Date":"8/27/2024","Local":28114},{"Date":"8/28/2024","Local":16668},
{"Date":"8/29/2024","Local":16700},{"Date":"8/30/2024","Local":-34767},
{"Date":"8/31/2024","Local":-57221}]
```

Note: The values for the Local group are determined by subtracting the number of departures from the number of arrivals of the local residents. If the resulting value in the data is negative, it indicates that more local residents departed from Hong Kong than arrived, and a positive value indicates the opposite.

Here is another example JSON string returned for the request
`http://localhost:8080/HKPassenger/v1/aggregate/`**`mainland`**`/`**`2024`**

```
[{"Month":"1/2024","Mainland":-39934},{"Month":"2/2024","Mainland":-56425},
{"Month":"3/2024","Mainland":-23109},{"Month":"4/2024","Mainland":64497},
{"Month":"5/2024","Mainland":-35804},{"Month":"6/2024","Mainland":7399},
{"Month":"7/2024","Mainland":46778},{"Month":"8/2024","Mainland":-37006},
{"Month":"9/2024","Mainland":31834},{"Month":"10/2024","Mainland":-23843},
{"Month":"11/2024","Mainland":18086},{"Month":"12/2024","Mainland":33070}]
```

The last example shows the returned JSON string for the request
`http://localhost:8080/HKPassenger/v1/aggregate/`**`all`**`/`**`2023`**`/`

```
[{"Month":"1/2023","Local":-729,"Mainland":11988,"Others":-12204,"Total":-945},
{"Month":"2/2023","Local":32516,"Mainland":34161,"Others":18439,"Total":85116},
{"Month":"3/2023","Local":-173144,"Mainland":7078,"Others":2744,"Total":-163322},
{"Month":"4/2023","Local":-62206,"Mainland":117839,"Others":15015,"Total":70648},
{"Month":"5/2023","Local":149674,"Mainland":-115064,"Others":-
15796,"Total":18814},
{"Month":"6/2023","Local":-196818,"Mainland":22308,"Others":14393,"Total":-
160117},
{"Month":"7/2023","Local":-74082,"Mainland":42460,"Others":-5692,"Total":-37314},
{"Month":"8/2023","Local":292382,"Mainland":-67134,"Others":-5189,"Total":220059},
{"Month":"9/2023","Local":-435613,"Mainland":109507,"Others":19204,"Total":-
306902},
{"Month":"10/2023","Local":479231,"Mainland":-91595,"Others":7420,"Total":395056},
{"Month":"11/2023","Local":-7504,"Mainland":6670,"Others":8050,"Total":7216},
{"Month":"12/2023","Local":-388803,"Mainland":93356,"Others":50141,"Total":-
245306}]
```

This routing handler performs the following actions:

- Only respond to the GET requests.
- To retrieve the yearly aggregated data for a specific group (Local, Mainland, or Others) within a certain year, first extract all relevant documents for that year from the database. Next, aggregate the data for the specified group for each month, and return the aggregated results for that group. If the request specifies the group 'all', the returned JSON string should include the aggregated data for the Local, Mainland, and Others groups, as well as the total aggregated data combining all three groups for each month.
- To retrieve the monthly aggregated data for a specific group (Local, Mainland, or Others) for a particular month and year, you should first collect all relevant documents from the database that match the specified month and year. Then, perform data aggregation for the selected group for each day of that month. The results become the aggregated data for the specified

group. If the request is for the group 'all', the returned JSON string should include the aggregated data for the Local, Mainland, and Others groups, as well as the total aggregated data combining all three groups for each day of the specified month.

- If the request specifies a month that extends beyond the last date in the database, the API should return an empty JSON array.
- Send the data as a single JSON string to the client.

This routing handler performs the following checking:

- If the request carries an incorrectly specified group, year, or month in the path, the handler should return an error JSON string with an HTTP status code of **400**. For examples,
```
{"error":"Cannot GET /HKPassenger/v1/local/2024/12"}
{"error":"Cannot GET /HKPassenger/v1/aggregate/overseas/2024/12"}
{"error":"Cannot GET /HKPassenger/v1/aggregate/all/2026"}
{"error":"Cannot GET /HKPassenger/v1/aggregate/others/2023/13"}
```

Note: Instead of explicitly identifying the specific part of the path that fails validation (as in Task B), this routing handler employs a different approach to handle incorrect paths.

- [**Optional**] If the program experiences database issues, it should return an error JSON string with an HTTP status code of **500**.
```
 {"error":"Experiencing database error!!"}
```

TASK D

Write an express routing handler to handle **POST requests** for adding new records to the MongoDB database. To request adding new records, the client would send a POST request to the URL `http://localhost:8080/HKPassenger/v1/data/`

The request includes a JSON string in the message body containing a set of **arrival and departure** data. For example, below is a JSON string provided to you for testing.
```
{
  "4/1/2025": [{"Flow":"Arrival","Local":247539,"Mainland":65342,"Others":30822},
       {"Flow":"Departure","Local":266275,"Mainland":70115,"Others":32814}],
  "4/2/2025": [{"Flow":"Arrival","Local":247677,"Mainland":65969,"Others":35066},
       {"Flow":"Departure","Local":298934,"Mainland":67121,"Others":30211}],
  "4/3/2025": [{"Flow":"Arrival","Local":253669,"Mainland":79434,"Others":38954},
       {"Flow":"Departure","Local":514166,"Mainland":76712,"Others":30454}]
}
```
You can assume that the input JSON string always includes the arrival and departure data for each specified date.

This routing handler performs the following actions:

- Only response to the POST requests.
- If no existing records match the input date, add the arrival and departure data to the database.
- After processing all input, the program sends a JSON string reporting the status of the transaction for each input date. For example, this is the response for successfully adding the above data to the database.
```
{
  "status": {
    "4/1/2025": "Added two records to the database",
    "4/2/2025": "Added two records to the database",
    "4/3/2025": "Added two records to the database"
  }
}
```

This routing handler performs the following checking:

- If the POST request does not carry data, the program should send an error JSON string with the HTTP status code of **400**. For example, `{"error":"POST request - missing data."}`.
- If matching records for an input date already exist in the database, the program should reject the transaction and report the status in the returned JSON string. For example, this is the response for running the above request again:
  ```
  {
    "status": {
      "4/1/2025": "Records existed; cannot override",
      "4/2/2025": "Records existed; cannot override",
      "4/3/2025": "Records existed; cannot override"
    }
  }
  ```
- For each input date, the program should verify that it is a valid calendar date before adding the data to the database. If it is an invalid date, report the reject status in the returned JSON string. For example,
  ```
  {
    "status": {
      "4/44/2025": "Wrong date format or invalid date",
      "4/5/2025": "Added two records to the database"
    }
  }
  ```
- [**Optional**] If the program experiences database issues, it should return an error JSON string with an HTTP status code of **500**.
  `{"error":"Experiencing database error!!"}`

To test this service, use the curl command to send a POST request with the JSON data to the Express server. You can download a few files from the course's Moodle site for testing.

TASK E

Instead of using the built-in Express error handler, **write an express routing handler** to intercept all request types and paths, which are not defined in previous tasks.

Return a JSON string with the HTTP status code of **400**. For example, for the request `POST /HKPassenger/local/2023/10`, we get the response `{"error":"Cannot POST /HKPassenger/local/2023/10"}`; for the request `GET /HKPassenger/v1/data/`, we get the response `'{"error":"Cannot GET /HKPassenger/v1/data/"}`.

Resources

You are provided with the following files.

- `template.txt` – the framework for the index.js file.

- `daylog.csv` – the dataset between 2021 Jan – 2025 Mar.
- `newdailylog-0*.json` – four JSON files for testing Task D.

Testing platform

We shall run the server program in the Node-dev container set, and use **Curl and Chrome** to test the APIs. You can use either Express 4.x or 5.x to implement the APIs.

Submission

Please finish this assignment before 23:59 on May 3, 2025 Saturday. Submit the following files:

1. The complete index.js program.
2. The **package.json** file for your express program.

Grading Policy

Points	Criteria
1.0	Task A ▪ 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 GET requests to retrieve raw data for a specific date or a range of dates starting from a specific date, and return a JSON string in the required format. ▪ Handle all errors
5.0	Task C ▪ Correctly handle GET requests to retrieve yearly or monthly aggregated data for a specific year or a specific month, and return a JSON string in the required format. ▪ Handle all errors
4.0	Task D ▪ Correctly handle POST requests to add new records to the collection, and return a JSON string in the required format. ▪ Handle all errors
1.0	Task E ▪ Error handling of all undefined methods and paths
-4.0	Using any external libraries that are not covered in the course.
-2.0	Not include the package.json file in the submission

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.***

Using GenAI

You are allowed to use Generative AI to explore various coding techniques and examples to complete your assignment. Feel free to divide the work into subtasks and employ GenAI to generate code snippets for better understanding. If you are using GenAI, you SHOULD include a report to clearly state the use of GenAI, including:

- The GenAI models you used.
- The prompts/questions you had with the GenAI and the responses. If you have adopted those code snippets, you SHOULD clearly indicate to us.

Should we suspect plagiarism in your submission, it would be unacceptable to justify this by stating you used GenAI for coding guidance and its output merely mirrors that of other students. While GenAI can be a valuable tool for exploring different techniques, we often need to apply adjustments to the provided code snippet to tailor it to our specific tasks.