Assignment 1

Submission Requirements:

1. Using the provided starter code, complete the code for the server.js file.
2. When you are finished:
    a. Delete the node_modules/ folder
    b. Rename the project to `A1_FIRSTNAME.js`. Replace `FIRSTNAME` with your name, example: `A1_PAOLO.js`
3. Create a zip file containing the entire project folder:
    a. Windows: Right click on the project folder > Send To > Compressed (zip) file
    b. Mac: Right click on the project folder > Compress
4. Submit the zip file to the course website.

Implementation Guidelines:

Your overall implementation must reflect the coding practices demonstrated in class. Key points include:

- When declaring variables, use let and const
- When declaring functions, use the const arrow function syntax described in class
- You **may** use the array shortcut functions: map(), reduce(), find(), forEach(), and filter()

Academic Integrity:

This is an individual assessment. While it is normal to want to study in a group, there is a fine line between helpful group discussion and academic dishonesty. It's fine to talk through general strategies or high-level approaches, but you must write your own code and avoid sharing or copying implementation details. If studying with others leads you to submit solutions that look nearly identical, then very likely you've crossed the boundary of what is acceptable group discussion.

When creating your assignment, please note the following:

What is allowed:

- Using course materials or internet resources to refresh your memory on basic Javascript syntax or concepts

Not permitted:

- Reposting any part of the assessment to online forums or homework help websites
- Contract plagiarism: Purchasing a solution, or completing a solution for compensation
- Sharing or receiving source code, references, or assistance from others

Usage of Artificial Intelligence (AI) tools:

- Using an AI to generate code on your behalf is NOT permitted.
- Asking an AI to solve a "bug" or "error" for you is NOT permitted. There is a fine line between asking an AI to explain to you the **meaning** of an error, and asking it for a solution to your error.
- The assignment must reflect your own understanding of the course material. Your submission should be based on what you have learned and can apply, not on what an AI system produces

Problem Description

In this assignment, you will develop a server side application that simulates a dog walking service. The system pairs walkers with dogs, tracks completed walk sessions, and collects feedback from owners.

Starter code has been provided on the course webpage. You must begin with this starter code and build your solution on top of it.

The application uses lists of data to represent dogs, walkers, and a walk history **Using these lists, implement the following endpoints:**

Endpoint	Description
`/walk/[dog id]/with/[walker id]`	Matches a walker with an appropriate dog. When a pairing occurs, the walk should be added to the history list as a completed session.
`/ratings/increase/[session id]`	For the given walk session, update the current rating by 1.
`/ratings/get/[walker id]`	Retrieves the average rating for the specified walker.

A detailed description of these endpoints is provided in the ***Endpoint Descriptions*** *section of this assignment.*

Once you have implemented these endpoints, test the endpoints using the links in the project's index.html file. You may add additional testing links as needed.

Description of Lists

The app manages 3 lists of data, provided to you in the assignment starter code's **/modules/data.js** file:

1. **Dogs List:** Stores dogs needing walks. Each dog record includes an id, name, breed, and owner details. Owners have a name and city

2. **Walkers List:** Keeps profiles of registered dog walkers. Each profile includes an id, name, city, and the walker's preferred breed. If a walker doesn't have a breed preference, this property is set to null. (Note: in programming, `null` usually means: "Does not exist")

3. **History List:** Tracks completed walking sessions. Each session record contains a unique id, the walker's id, the dog's id, and a rating from 1 to 5.

In the starter code, you may modify the objects as needed to test your code. However, the overall structure of the objects should remain the same.

Endpoint Descriptions:

Using the provided lists of data, implement the app's endpoints. Here's a description of each endpoint

1. `/walk/[dog id]/with/[walker id]`

Endpoint accepts two route parameters: a dog id and walker id. You may assume valid values will always be provided.

Using this data, the endpoint must check if the specified walker is eligible to walk the given dog, according to these rules:

- Both the walker and the dog must be located in the same city.
- If the walker has a preferred breed, the dog's breed must match that preference.
- If the walker has no preferred breed (null), they can walk any dog in the same city.

If the criteria are met, the endpoint must:

1. Create a new walk session and add it to the history list. By default, the rating should be set to 1.
2. Assign a unique session id. The session id must be 1 greater than the most recent walk session. For example, if the most recent walk session is 25, then the new session id should be 26.
3. Respond with a success message and the assigned session id.

If the walker does **not** meet the criteria, the endpoint should respond with an error message specifying why the walk could not be scheduled (e.g., city mismatch, breed mismatch).

2.`/ratings/increase/[session id]`

This endpoint accepts a completed walk session id as a route parameter. You may assume a valid session ID is always provided.

Using this information, the endpoint must:

1. Search the walk history for a session matching the provided ID.
2. If the session exists, update its rating by 1. The maximum possible rating is 5.
3. Responds with a success message confirming the new rating has been recorded.

if the rating value is already at 5, the endpoint should output an error message explaining the issue.

3.`/ratings/get/[walker id]`

The endpoint accepts a walker id as a route parameter. You may assume a valid walker id will always be provided.

Using this id, the endpoint must:

1. Search the history list for all walk sessions completed by the walker with the provided id
2. Compute the average rating for the walker, according to these rules:
    a. The average should only be calculated if the walker has at least 4 recorded ratings.

    b. When there are four or more ratings, calculate the average after excluding the single highest and the single lowest rating. This helps provide a fairer assessment by removing outliers. For example, if the walker’s ratings are: 2,2,3,5,5, then the final rating is computed as the average of: 2,3,5. (average = 3.33333)
    c. If there are fewer than four ratings, the average rating defaults to 0 to indicate there is insufficient data.
3. Respond with a message containing the average rating.

END OF ASSESSMENT