MICROSERVICE FOR HANDLING PLANETARY DATA OF OUR SOLAR SYSTEM (API DESIGN DOCUMENT)

Description

The microservice created handles data of the planets in our solar system. The purpose of the service is to update and retrieve these data. The details of the data used are described on the *Database Schema* section of the design document. Since the application is going to be consumed by another application, only the REST API was created without any front-end presentation feauture. In addition to updating and retrieving data, the service calculates the distance between two planets and also - as an extra capability - it can add and remove a planet from the database.

Development Specifics

For the development of this microservice, a REST API was built using mainly Typescript with the help of ExpressJS and MongoDB for the database. A NoSQL database was preferred in contrast to a relational database for expansion purposes. The database is hosted on MongoDB Atlas on AWS. The framework that handles the connection between the service and the database is Mongoose.

Dependencies :
- typescript
- yarn
- ExpressJS
- concurrently
- nodemon
- mongoose
- body-parser

Database Schema

The database created (planets) contains a collection (planet_data) that stores details for each planet. The structure of the database is as follows :

planet_data

Field	Type	Units	Example
_id	ObjectId	-	5f4a4a35bacd9608286e3d7f
name	String	-	Earth
size	Double	E (Earths)	1
distance_to_sun	Double	AU (Astronomical Units)	1
rings	Int32	-	0
orbital_speed	Double	km/s	29.79
number_of_moons	Int32	-	1

As it is obvious, the details include the requested (name, size, distance to the sun), but also where added several others as extra information (number of rings the planet has, mean orbital speed and number of moons orbiting the planet). The measurement units of its detail are provided on the table above.

Additional Database Notes

- As fore mentioned, the database is hosted on MongoDB Atlas and is accessible using the connection url :

  mongodb+srv://cluster0.to72o.mongodb.net/planets.

- The cluster on which the database is hosted has two registered users. The main admin user is {username : root, password: root}. The other user is a guest that can only read any database on the cluster {username: guest, password: guest}. The purpose of the guest user is for the re-purpose of the service to be used by a non-technical team.

- The url provided has to be altered if connection is being made directly from an application and not from a mongo shell. The url for application connection is:

  mongodb+srv://<username>:<password>@cluster0.to72o.mongodb.net/planets?retryWrites=true&w=majority.

- The collection hosting every planet data is not capped for expansion purposes.

Requests / Response Structure (API Calls)

allPlanets (GET)
https://localhost:3000/planets

- Description: The service responds with a json containing all documents and theirs fields which are stored in the collection.

getPlanet (GET)
https://localhost:3000/planet/{id}

- Description: The service responds with a json containing all fields of the document with the id given in the request.

getDistance (GET)
https://localhost:3000/distance/{id1}-{id2}

- Description: The service responds with a string which represents the subtraction result of planet-id1 distance to sun from the planet-id2 distance to sun. For the simplicity, the result is the absolute value of the subtraction without subtracting the larger from the smaller value. Also it assumed that the distance from the sun of each planet is calculated on an ideal linear line from the core of each planet. The result is in AU units.

addPlanet (PUT)
https://localhost:3000/planet

- Description: The request sends a json containing the required fields of the document. The service responds with the json of the newly added planet.

deletePlanet (DELETE)
https://localhost:3000/planet/{id}

- Description: The service responds a success message for the deletion of the planet.

updatePlanet(POST)
https://localhost:3000/planet/{id}

- Description: The service responds a success message for the update of the planet.

Additional API Calls Notes

- The getDistance GET Request calculates the distance between the two planets using a number variable and then casts that value into a string variable, which in return is send as a response.

- Testing the functionality of API calls was done using Postman.

Expansion Capabilities

The service can be easily expanded to contain further API calls, as well as changes to the database can be easily done due to the nature of the NoSQL database used.

Deployment

It is easy to deploy the service on a big cloud platform. For example, to install on an AWS EC2, after gaining SSH access, someone has to install NodeJS and yarn, then clone the repository and finally install the dependencies. After configuring the security options to have public access and then the app is ready.