Lab 7.1 - Implement a Data Aggregating Service

The labs-1 folder contains the following files:

- `package.json`
- `boat-service.js`
- `brand-service.js`
- `validate.js`

The `package.json` file has the following content:

```
{
  "name": "labs-1",
  "version": "1.0.0",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "echo \"Error: start script not specified\" && exit 1"
  },
  "keywords": [],
  "license": "UNLICENSED"
}
```

Create a service that is started when the **`npm start`** command is executed that consumes two other HTTP services.

The service must bind to a port number defined by the **`PORT`** environment variable.

The services are provided with mock data in this project folder as `boat-service.js` and `brand-service.js`.

When started, each mock service outputs a port. This output can be used to set the **`BOAT_SERVICE_PORT`** and **`BRAND_SERVICE_PORT`** environment variables when starting the aggregating service.

For instance if the port of the Boat service is 3333 and the port of the Brand service is 3334 the server can be started like so:

```
PORT=3000 BOAT_SERVICE_PORT=3333 BRAND_SERVICE_PORT=3334 npm start
```

Be sure to use the **`BOAT_SERVICE_PORT`** and **`BRAND_SERVICE_PORT`** environment variables in the service to get the relevant port for each service. For example, the values of these environment variables could be loaded into the service implementation like so:

```
const {
  BOAT_SERVICE_PORT,
  BRAND_SERVICE_PORT
} = process.env
```

To make a request to the Brand service:
`http://localhost:[BRAND_SERVICE_PORT]/[id]`

The Boat service responds with JSON data in the following format:

```
{
  "id": Number,
  "brand": Number,
  "color": String
}
```

A request to the Boat service: **`http://localhost:[BOAT_SERVICE_PORT]/[id]`**
The **`id`** and **`brand`** properties will only ever be Integers.

The Brand service responds with JSON data in the following format:

```
{
  "id": Number,
  "name": String
}
```


The **`brand`** property of the Boat service output corresponds to the **`id`** of the Brand service entities.

Create a service which accepts GET requests at **`http://localhost:[PORT]/[id]`**. Use the incoming **`id`** from the GET request to make a request to the Boat service and use data from the Boat service to make a request to the Brand service to retrieve associated brand data.

Combine the information from the two responses into a JSON payload and send that as a response. The JSON payload should have the following form:

```
{
  "id": Number,
  "color": String,
  "brand": String
}
```

The aggregating service should handle various scenarios in the following ways:

- For a normal successful request, respond with a 200 status code and the JSON payload of combined data (`{id, color, brand}`) as the response body. The `Content-Type` header must be `application/json`.
- Respond with a 404 status code if either service responds with a 404 status,
- If either service is not available, respond with a 500 status code with any response body.
- If either service responds with a non-200 status code then respond with a 500 status code with any response body
- If a request is made to the aggregating service with an ID that is not a valid integer, respond with a 400 status code, the response body is unimportant and can be anything.
- If either service responds with a 4XX status code that is not a 400 or 404 (401-403, 405-499) status code, then respond with a 500 status code with any response body.
- Be sure that if an upstream service is not available, that the service responds within 1250ms.

The validator code for this exercise starts the services automatically. Make sure that the services are not running and then run the following command in the labs-1 folder to check the implementation:

```
node validate.js
```

If the aggregating service is successfully implemented this should result in the following output: