Lab 5.1 - Implement a RESTful JSON GET

The labs-1 folder contains the following files:

- `model.js`
- `package.json`
- `validate.js`

The **`start`** field of the `package.json` file looks as follows:

```
    "start": "echo \"Error: start script not specified\" && exit 1",
```

The `model.js` file contains the following:

```
'use strict'

module.exports = {
  boat: boatModel()
}

function boatModel () {
  const db = {
    1: { brand: 'Chaparral', color: 'red' },
    2: { brand: 'Chaparral', color: 'blue' }
  }

  return {
    uid,
    create,
    read,
  update,
  delete: del
}

function uid () {
  return Object.keys(db)
    .sort((a, b) => a - b)
    .map(Number)
    .filter((n) => !isNaN(n))
    .pop() + 1 + ''
}

function create (id, data, cb) {
  if (db.hasOwnProperty(id)) {
    const err = Error('resource exists')
    err.code = 'E_RESOURCE_EXISTS'
    setImmediate(() => cb(err))
    return
  }
  db[id] = data
  setImmediate(() => cb(null, id))
}

function read (id, cb) {
  if (!(db.hasOwnProperty(id))) {
    const err = Error('not found')
    err.code = 'E_NOT_FOUND'
    setImmediate(() => cb(err))
    return
  }
  setImmediate(() => cb(null, db[id]))
}

function update (id, data, cb) {
  if (!(db.hasOwnProperty(id))) {
    const err = Error('not found')
    err.code = 'E_NOT_FOUND'
    setImmediate(() => cb(err))
    return
  }
  db[id] = data
  setImmediate(() => cb())
  }

  function del (id, cb) {
    if (!(db.hasOwnProperty(id))) {
      const err = Error('not found')
      err.code = 'E_NOT_FOUND'
      setImmediate(() => cb(err))
      return
    }
    delete db[id]
    setImmediate(() => cb())
  }
}
```

Use either Fastify or Express to implement a RESTful HTTP server so that when the command **`npm start`** is executed, a server is started that listens on **`process.env.PORT`**.

If implementing in Fastify, remember that running **`npm init fastify -- --integrate`** in the labs-1 folder will set up the project **`npm start`** is executed the server will automatically listen on **`process.env.PORT`**.

The server should support a **`GET`** request to a single route: **`/boat/{id}`** where **`{id}`** is a placeholder for any given ID - for instance **`/boat/2`**.

The **`GET /boat/{id}`** route should respond with a JSON payload. The route should also respond with the correct headers for a JSON response (**`Content-Type: application/json`**).

The server should only support this GET route. That means that any other routes or any other verbs should be handled according to the HTTP specification. Thankfully Express and Fastify will do most of this for us.

The following cases must be successfully handled:

- A successful request should respond with a **`200`** status code. Express and Fastify do this automatically.
- The response should have the correct mime type header. In this case we need to make sure the **`Content-Type`** header is set to **`application/json`**.
- A GET request to a route that does not exist should respond with a **`404`** status code. Fastify does this automatically and the typical Express configuration also handles this by default.

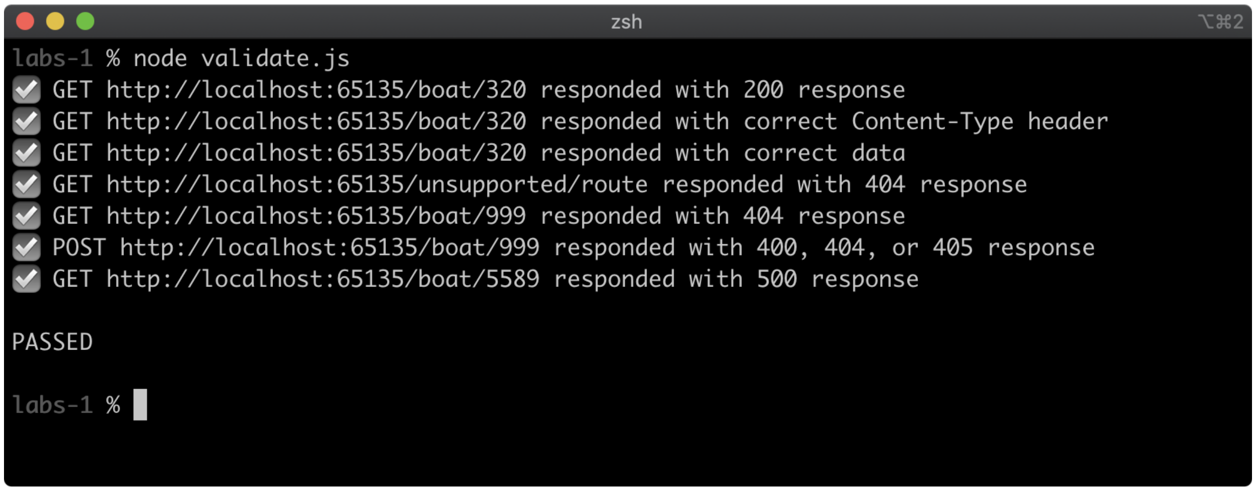
- If a given boat ID isn't found in the model the server should respond with a **`404`** status code. The response body can contain anything, but it's important that the response status is set to 404.
- Unexpected errors in the model should cause the server to respond with a **`500`** status code. This means that if the **`read`** method of the model passed an Error object to the callback that was unexpected or unrecognized, that error needs to be propagated to the framework we're using in some way so that the framework can automatically generate a 500 response.
- In the HTTP specification there is some ambiguity over how to handle unsupported HTTP methods. Any HTTP method other than **`GET`** should be responded to with either a **`400`**, **`404`** or **`405`** status code. Again Fastify and Express will respond to unsupported methods with one of these status codes.

Do not edit the **`model.js`** file, it will be overwritten by the validation process anyway. The **`model.js`** file is deliberately noisy, providing methods that we don't need for this exercise. This reflects the philosophical approach of the certification to provide occasionally messy API's to integrate with in order to better reflect real-world scenarios.

Once the server has been implemented, the following command can be executed to validate the implementation:

```
node validate.js
```

When correctly implemented the result of this command should be as follows:



```
labs-1 % node validate.js
✓ GET http://localhost:65135/boat/320 responded with 200 response
✓ GET http://localhost:65135/boat/320 responded with correct Content-Type header
✓ GET http://localhost:65135/boat/320 responded with correct data
✓ GET http://localhost:65135/unsupported/route responded with 404 response
✓ GET http://localhost:65135/boat/999 responded with 404 response
✓ POST http://localhost:65135/boat/999 responded with 400, 404, or 405 response
✓ GET http://localhost:65135/boat/5589 responded with 500 response

PASSED

labs-1 %
```