Individual Exercise

Add similar endpoints for the patients

Using what you learned from the first 3 exercises, set up 3 endpoints for the Patients resource:

- GET /patients
- GET /patients/:id
- POST /patients

Set up Postman requests for all three.

Get a list of visits by doctor id, patient id, or both

Goal: Set up an endpoint to retrieve a list of visits, filtered by doctor id, patient id or both ids.

A visit is an object that contains the doctor id, patient id and a date. We can implement getting a list of visits based on either doctor or patient id, or both.

We can do this using a query string in our route. If there is no query string, we can return the full list of visits.

We'll be using these methods:

- Request.query

1. In `src/index.js`, set up the route. Let's try returning all of the visits in our data file.

```
app.get("/api/v1/visits", (req, res) => {
  console.log(req.query);
  return res.json(data.visits);
});
```

2. Test the route on Postman to see what is returned by `req.query` in your terminal. Your URL on Postman should look something like `localhost:3000/api/v1/visits?doctorid=1&patientid=1`. You can edit the query parameters through the Postman UI (or toggle them to be included or not).

3. Implement the logic to find the correct visits.

```
app.get("/api/v1/visits", (req, res) => {
  const { doctorid, patientid } = req.query;
  let visits = data.visits;

  if (doctorid) {
    visits = visits.filter(
      (visit) => visit.doctorid === parseInt(doctorid, 10)
    );
  }

  if (patientid) {
    visits = visits.filter(
      (visit) => visit.patientid === parseInt(patientid, 10)
    );
  }

  return res.json(visits);
});
```

Swagger Documentation

Goal: Set up Swagger documentation.

We provide a starter `swagger.json` file for you in the starter repo. Let's set it up to show our documentation.

1. Install the package for `swagger-ui-express`. This module allows you to serve auto-generated swagger-ui generated API docs from express, based on a swagger.json file. The result is living documentation for your API hosted from your API server via a route. Documentation.

   ```
   npm install swagger-ui-express
   ```

2. In `src/index.js`, import the 2 modules you need.

   ```
   import swaggerUi from "swagger-ui-express";
   import swaggerDocument from "../swagger.json";
   ```

3. Add a route to show the documentation.

   ```
   app.use("/api/v1/docs", swaggerUi.serve, swaggerUi.setup(swaggerDocument));
   ```

4. Navigate to that route to see an interactive Swagger documentation.

5. Read up on the Swagger Documentation structure in their documentation. Using that structure, add API documentation for the patient routes.

Update the Swagger docs

1. Update your Swagger documentation to account for the `/patients` routes.

Approach 2

Auto-generate Swagger UI based on documentation in code

This approach uses the express-swagger-generator package.

You use comments in the code to write the documentation. For example:

```
/**
 * @typedef Patient
 * @property {string} id.required - Normally an auto increment number stored as string
 * @property {string} name.required - Full name
 */

/**
 * Get a single patient by ID
 * @route GET /patients/{id}
 * @group Patients
 * @param {string} id.path.required - Patient ID
 * @returns {Patient.model} 200 - Patient with requested ID
 * @returns {string} 404 - Requested patient not found
 * @produces application/json
 */
```

To set up swagger, add this to `src/index.js`

```
import expressSwaggerGenerator from "express-swagger-generator";
const host = `localhost:${port}`;
const basePath = "/"; // The forward slash is important!

// Options for the Swagger generator tool
const options = {
  // The root document object for the API specification
  // More info here: https://github.com/OAI/OpenAPI-
Specification/blob/master/versions/2.0.md#schema
  swaggerDefinition: {
    info: {
      title: "Health Insurance API",
      description: "This is Web service for patients, doctors and visits.",
      version: "1.0.0",
    },
    host: host,
    basePath: basePath,
    produces: ["application/json"],
    schemes: ["http", "https"],
  },
  basedir: __dirname, // Absolute path to the app
  files: ["./routes/**/*.js"], // Relative path to the API routes folder to find the
documentation
};

// Initialize express-swagger-generator and inject it into the express app
expressSwaggerGenerator(app)(options);
```

Implement error handling for the GET /visits route

We did the happy path above. Take some time to implement error handling for when:

- Doctorid param is not a number
- Patientid param is not a number

Code organization: separate routes

Right now, we have endpoints for doctors, patients and visits all in one file in `src/index.js`. To organize our code, we can split these up into 3 high-level routes (one for each).

Use the following to set up a `routes` folder and refactor your code to be more organized. Keep in mind that refactoring means nothing should change behaviour-wise in your code.

- Router
- express.Router
- A Guide to Routing - the last 2 sections on `app.route()` and `express.Router` are most relevant.

Refactoring your code

Extract the validations and logic within a route to separate class.