Introduction

In this assignment, you will use the MERN stack to write a Single Page Application (SPA) that tracks exercises completed by the user. You will use React for the front-end UI app. You will write a REST API using Node and Express for the back-end web service. You will use MongoDB for persistence.

This is the "Portfolio Assignment" for the course. This means that you are allowed to post the entirety of the assignment publicly (e.g., Github, personal website, etc.) after the term ends. You can use this as an opportunity to showcase your work to potential employers.

Be sure to periodically review Assignment 9: Tips, FAQs, Corrections threadLinks to an external site. in the Ed discussion board (pinned at the top)

Learning Outcomes

- What is the lifecycle of a React component? (Module 9 MLO 1)
  What are the useEffect and useHistory React hooks? (Module 9 MLO 2)
  How can we deploy a React app created using create-react-app (CRA) (Module 9 MLO 3)?
  What is the Fetch API? (Module 9 MLO 4)
  Why do we need to lift up state in some apps? (Module 9 MLO 5)?

Data for the App

You will store the data in MongoDB in a collection named `exercises`. Each document in the collection must have the following properties (i.e., all properties are required):

Property	Data Type	Comments
name	String	The name of the exercise
reps	Number	The number of times the exercise was performed
weight	Number	The weight of the weights used for the exercise
unit	String	The unit of measurement of the weight. Only values allowed are `kgs` and `lbs`
date	String	The date the exercise was performed. Specified as MM-DD-YY, e.g., 07-30-21.

REST API Web Service

You must implement a REST API that supports CRUD operations by implementing the following 4 endpoints:

1. Create using POST /exercises

Request

- The request body will be a JSON object with the 5 properties listed in the data model.
- The POST request will have no path parameters.

Request Validation

You can assume that the request body is a JSON object. However, your code must validate the request body. If the request body is valid then a new document must be created and the "Success" response (described below) must be sent. If the request body is invalid then the "Failure" response (described below) must be sent. Here are the requirements for the request body to be valid

- The body must contain all 5 properties (in other words, if any of the 5 properties is missing, the request is invalid).
- The `name` property must be a string containing at least one character (i.e., it can't be empty or a null string).
- The `reps` property must be an integer greater than 0.
- The `weight` property must be an integer greater than 0.
- The `unit` property must be either the string `kgs` or the string `lbs`.
- The `date` property must be a string in the format `MM-DD-YY`, where MM, DD and YY are 2-digit integers.
    - To validate this property, you can write your own code or just use a function we have provided in the section "Hints and Suggestions for the REST API."
    - Optional: You can enhance validation to ensure that the date is correct for the given month and year, assuming that the year is in this century
        - E.g., 12-31-21 and 02-29-20 are valid, but 13-32-21 and 02-30-20 are invalid
    - But any additional validation beyond the function we have provided is not required.

Response

- Success: If the request is valid then a new document must be created and the following response must be sent

        - Body: A JSON object with all the properties of the document including the unique ID value generated by MongoDB.
        - Content-type: `application/json`.
        - Status code: 201.
- Failure: If the request body is invalid then the following response must be sent
    - Body: A JSON object `{ Error: "Invalid request"}`
    - Content-type: `application/json`.
    - Status code: 400.

2. Read using GET /exercises

Request

- No path parameter.
- No request body (you don't need to validate this).

Response

- Body: A JSON array containing the entire collection.
    - If the collection is empty, the response will be an empty array
    - Each document in the collection must be a JSON object with all the properties of the document including the ID.
- Content-type: `application/json`.
- Status code: 200.

3. GET using GET /exercises/:_id

Request

- The path parameter will contain the ID of the document.
- No request body (you don't need to validate this).

Response

- Success: If a document exists with the specified ID, the following response must be sent
    - Body: A JSON object with all the properties of the document including the unique ID value.
    - Content-type: `application/json`.
    - Status code 200.
- Failure: If no document exists with the specified ID, the following response must be sent
    - Body: A JSON object `{ Error: "Not found"}`
    - Content-type: `application/json`.

    - Status code: 404.

4. Update using PUT /exercises/:_id

Request

- The request body will be a JSON object with all the 5 properties listed in the data model.
- The `date` property will be in the format `MM-DD-YY`, e.g., `06-24-21`.
- The path parameter will contain the ID of a document.

Response

- Success: If the request body is valid and a document exists with the specified ID, then the document must be updated and the following response must be sent
    - Body: A JSON object with all the properties of the updated document including the ID.
    - Content-type: `application/json`.
    - Status code: 200.
- Failure: If the request body is invalid then the following response must be sent
    - Body: A JSON object `{ Error: "Invalid request"}`
    - Content-type: `application/json`.
    - Status code: 400.
- Failure: If no document exists with the specified ID, the following response must be sent
    - Body: A JSON object `{ Error: "Not found"}`
    - Content-type: `application/json`.
    - Status code: 404.
- Note: first check the validity of the request body and if it is invalid, return the response with status code 400. Only look for the existence of the document if the request body is valid.

5. DELETE using DELETE /exercises/:_id

Request

- The path parameter will contain the ID of the document.
- There will not be a request body (you don't need to validate this).

Response

- Success: If a document exists with the specified ID, it must be deleted and the following response must be sent

        - Body: No response body
        - Content-type: Not applicable
        - Status code: 204.
- Failure: If no document exists with the specified ID, the following response must be sent
    - Body: A JSON object `{ Error: "Not found"}`
    - Content-type: `application/json`.
    - Status code: 404.

Technical Requirements

Separate Model Code from Controller Code

- Your model code must be separate from your controller code.
    - You cannot import the `mongoose` package in your controller code.
    - You cannot import the `express` packages in your model code.
- Your model code can be in as many files as you want.
- Similarly, your controller code can be in as many files as you want.
- However, the model code and the controller code must be in separate files.

Use a .env file

- Your REST API must use a `.env` file with 2 variables
    - PORT with value 3000, which is the port on which the Express server for the REST API will receive HTTP requests, and
    - `MONGODB_CONNECT_STRING` with the connect string that must be used to connect to the MongoDB server.
        - When testing your program we will change the value of this string to the MongoDB server we want to use for testing.

Use ES Modules

- Your REST API code must use ES modules (i.e., you cannot use Common JS modules)

Hints & Suggestions for the REST API

- You can write the code to validate the request body from scratch, or write it using the package express-validatorLinks to an external site., whichever you prefer.
- To validate the date property, you can write your own code or you can use the following function

```
/**
*
* @param {string} date
* Return true if the date format is MM-DD-YY where MM, DD and YY are 2 digit
integers
*/
function isDateValid(date) {
    // Test using a regular expression.
    // To learn about regular expressions see Chapter 6 of the text book
    const format = /^\d\d-\d\d-\d\d$/;
    return format.test(date);
}
```

- The Mongoose function updateOneLinks to an external site. returns a promise that resolves to an object with information about the update operation.
    - The value of the property `matchedCount` is 0 if no document matched the filter passed to `udpateOne`.
    - You can use this to determine if a document with the specified ID exists or not.
- The Mongoose function findByIdLinks to an external site. returns a promise.
    - If a document is found matching the ID then the promise resolves to that document.
    - Otherwise the promise resolves to a null value.

React UI

The UI must have the following 3 pages:

1. Home Page.
2. Edit Exercise Page.
3. Create Exercise Page.

Home Page

- This page is rendered when the app starts up.
- The page must display the data for all the exercises stored in MongoDB.
- The page must get the data by calling the endpoint `GET /exercises` in the REST API.
- The data must be displayed in an HTML table.
- Each row must display the 5 properties listed in the data model. The ID value must not be displayed.
- In addition to the data, each row must include 2 icons from the React Icons libraryLinks to an external site., one to support deleting the row and the other for updating the row.

        - You can choose any suitable icon from the library that clearly indicates the correct use of clicking on it.
        - Clicking on the delete icon must immediately delete the row by calling the endpoint `DELETE /exercises/:id` in the REST API.
        - Clicking on the edit icon must take the user to the Edit Exercise Page.
- You must implement a React component for the table and another for the row. You can implement as many React components beyond these two as you want, e.g., the row itself may contain other React components.
- This page must include a way for the user to go to the Create Exercise Page.
    - It is your choice how you present this functionality as long as it is clear how the user can go to that page.
    - For example, you can provide a link or an icon with informational text.

Edit Exercise Page

- This page will allow the user to edit the specific exercise for which the user clicked the edit icon.
- The controls to edit the exercise must be pre-populated with the existing data for that row.
- You must provide a button that
    - Saves the updated exercise by calling the endpoint `PUT /exercises/:id` in the REST API, and
        - If the update is successful (i.e., the response status code is 200), then
            - Shows an alert to the user with a message about the update being successful, and
            - Automatically takes the user back to the Home page.
        - If the update is unsuccessful (i.e., the response status is not 200), then
            - Shows an alert to the user with a message about the update having failed, and
            - Automatically takes the user back to the Home page.

Create Exercise Page

- This page will allow the user to add a new exercise to the database.
- You must provide input controls for the user to enter the 5 required properties.
- You must provide a button that:

        - Saves this new exercise by calling the endpoint `POST /exercises` in the REST API, and
            - If the creation is successful (i.e., the response status code is 201), then
                - Shows an alert to the user with a message about the exercise being created, and
                - Automatically takes the user back to the Home page.
            - If the creation is unsuccessful (i.e., the response status is not 201), then
                - Shows an alert to the user with a message about the exercise creation having failed, and
                - Automatically takes the user back to the Home page.

Technical Requirements

Function-Based Components

- Your React components must be function-based. You are not allowed to define class-based components.

Use the React Functionality Discussed in the Course

- You cannot use any React functionality we didn't cover in the course without getting instructor approval.
- In particular, you cannot use frameworks like Next.js.

Use a .env file & set the proxy property

- Your React app must use a `.env` file with a variable `PORT=8000` which is the port on which the Express server for the React app will receive HTTP requests.
- You must add the `proxy` property to the React app's `package.json` file as discussed in the exploration.

CSS

Update and add rules to the existing `App.css` file that resides in the `/src` folder. *Note that specifying black, white, and Times New Roman font are not allowed (because they are already the defaults).*

- Global page design:

    - Add a `body` rule in the first line of the `App.css` file that defines the font-family, background-color, color, margin, and padding for the app.
- Table
    - Add `tr th` and `tr td` rules to update borders, color, and padding.
- Form
    - Add `input`, and `button` rules that include the same font-family as the `body`
        - This is needed because the form elements do not inherit the font-family from `body` by default
- Note: you are allowed to add additional rules beyond the required rules listed above.

Design Features

- You must use a `<select>` element to provide the options for selecting the value of units in the Edit Exercise Page and the Create Exercise Page.
- You need to add semantic page layout tags in the `App.js` file, including at least the following:
    - The `<header>` tag will include a heading level 1 `<h1>` tag to specify the app's name and a paragraph `<p>` that describes it.
    - The `<footer>` tag will include the student's name in a copyright statement: © year first last.
    - These tags must show up on all 3 pages.
- All 3 pages must have links to the Home Page and the Create Exercise Page using a React component named `Navigation`.
    - The component `Navigation` must use the <nav> tagLinks to an external site. and the Link componentLinks to an external site. from `react-router-dom` to have links to the Home Page and the Create Exercise Page.
    - Add this component in `App.js` before the routes.