Assignment 6 - Full Stack MERN App

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.

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 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`.
  - You can assume that all the properties are valid.
  - The POST request will have no path parameters.
- Response

    - A JSON object with all the properties of the document including the unique ID value generated by MongoDB.
    - The content-type of the response must be set to `application/json`.
    - The status code must be 201.

2. Read using GET /exercises

- Request
    - No request body and no path parameters.
- Response
    - A JSON array containing the entire collection.
    - Each document in the collection must be a JSON object with all the properties of the document including the ID.
    - The content-type of the response must be set to `application/json`.
    - The status code must be 200.

3. 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`.
    - You can assume that all the properties are valid.
    - The path parameter will contain the ID of a document. You can assume that a document exists with this ID.
- Response
    - A JSON object with all the properties of the updated document including the ID.
    - The content-type of the response must be set to `application/json`.
    - The status code must be 200.

4. DELETE using DELETE /exercises/:id

- Request
    - The path parameter will contain the ID of the document. You can assume that a document exists with this ID.
    - The DELETE request will not have a body.
- Response
    - The status code must be 204.

Notes on the REST API

- You can assume that the requests will have valid data.
- In case of any error, the response status code must be `500` and the response body must have a JSON object with information about the error. This information can simply be the exception in the code that caused the error.
- Optional: Although we will not test this, you can send back response status code `404` (instead of `500`) if the update or delete request has an unknown ID.

React UI

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

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,
    - Shows an alert to the user with a message about the update being successful, 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,
    - Shows an alert to the user with a message about the creation being successful, and
    - Automatically takes the user back to the Home page.

Global 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.
- Your app must have a Navigation component and provide global navigation for the Home Page and 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 `<main>` tag will include the `<Route>` tags to import each page's content.

    - The `<footer>` tag will include the student's name in a copyright statement: © year first last.