Assignment 9 - Full Stack MERN App

Start Assignment

- Due Friday by 11:59pm
- Points 68
- Submitting a file upload
- Available May 26 at 8:30am - Jun 7 at 11:59pm

Introduction

In this assignment, you will use the MERN stack to write a Single Page Application (SPA) that tracks exercises completed by a user. You will use React for the frontend UI app. You will write a REST API using Node and Express for the backend 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: FAQs and Tips thread in the Ed discussion board (to be added in a few days)

> **Note:** In addition to functional requirements, this assignment also has certain technical requirements. Be sure to read the complete assignment and to follow the technical requirements. Even if your apps meet all the functional requirements, you will lose points if the technical requirements are not met. Similarly, we provide instructions on what files you must submit in the section "What To Turn In?" If you don't follow those directions, you will lose points for that as well.

Learning Outcomes

- What is the lifecycle of a React component? (Module 9 MLO 1)
- What are the `useEffect` and `useNavigate` React hooks? (Module 9 MLO 2)
- How can a React app downloaded from one server send requests to a REST API running on a different server? (Module 9 MLO 3)
- What is the Fetch API? (Module 9 MLO 4)
- Why do we need to lift up state in some React apps? (Module 9 MLO 5)?

REST API Web Service

Write a REST API that models exercises and provides CRUD operations listed in the section "CRUD Operations."

Starter Code

We have provided you starter code for the REST API in the file **exercises_rest.zip (https://canvas.oregonstate.edu/courses/1999527/files/112152472?wrap=1)** **(https://canvas.oregonstate.edu/courses/1999527/files/112152472/download?download_frd=1)** .

- Download and unzip this file.
- Update the file `.env` to put in the value of MongoDB connect string for your MongoDB instance.
- Then run `npm install` (once) and `npm start` to start the app.
- Note that the REST API will listen on port 3000.
- You must write code in the files `exercises_model.mjs` and `exercises_controller.mjs` to implement the REST API.
- You can use the starter code we have provided to you in these 2 files or change this starter code as you want. However, make sure your code follows the requirements listed in the section "Technical Requirements."
- Other than updating the value of the variable `MONGODB_CONNECT_STRING` in the file `.env`, do not change anything in the other files we have provided you with in the starter zip file. Also, do not add any new files to your REST API code.

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 the properties are required):

Property	**Data Type**	**Comments**
name	String	The name of the exercise.
reps	Number	The number of times the exercise was performed. The value must be an integer greater than 0.
weight	Number	The weight of the weights used for the exercise. The value must be an integer greater than 0.
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.
_id	Object	Unique ID. Automatically assigned by MongoDB when a document is created.

		Internally MongoDB creates an instance of the class `ObjectId`. In HTTP requests and responses, its string representation is used.

CRUD Operations

You must implement a REST API that supports CRUD operations by implementing the following 5 endpoints:

1. Create using POST /exercises

Request

- The request body will be a JSON object.
- 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 the following 5 properties.
  - If any of the 5 properties is missing, the request is invalid.
  - If the request contains any property other than these 5 properties, again 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., 01-31-24 and 02-29-24 are valid, but 01-32-24 and 02-29-23 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 **(not an array)** with all the properties of the document including the ID.
    - 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.
- The path parameter will contain the ID of a document.

Request Validation

You can assume that the request body will be a JSON object. However, your code must perform the same validation on the request body that is described for POST requests.

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 for the REST API

Only Use Express Functionality Discussed in the Course

- You must not use any functionality of Express that has not been discussed in the course unless you get approval from an instructor. You can post on Ed to ask the instructors for approval.
  - As an example, you cannot use Express Router() function in your code.

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. The controller code must not directly call any Mongoose code but can only interact with the database by calling functions defined by you in the model code.
  - You cannot import the `express` packages in your model code. You cannot pass any Express objects (e.g., request and response) to functions in your model. You cannot export the model class (by default, `Exercise`) from your model.

Use the Variables in the .env file Provided to You

- Your REST API code must use the `.env` file we have provided you with. This file has 2 variables
  - `PORT`
    - This is the port on which the Express server for the REST API will receive HTTP requests.
    - It's value in the file is 3000. Do not change this value.
  - `MONGODB_CONNECT_STRING`
    - Your code must use the value of this variable to connect to the MongoDB server.
    - Update the value to the connect string of the MongoDB server you want to use.
    - When testing your program, we will change the value of this variable 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).

Use async/await for Asynchronous Programming

- Your code must use the `async` and `await` syntax for asynchronous programming and cannot use the `.then()` syntax.
- The route handler functions in the controller will be asynchronous. You must wrap these functions inside `asyncHandler` to prevent your Express server from crashing due to any uncaught exception. If needed, review this topic in **Exploration — Writing Asynchronous Code (https://canvas.oregonstate.edu/courses/1999527/pages/exploration-writing-asynchronous-code)**.

Response Body Must Match the Specification

- According to the specification some of the successful responses return a **JSON object**, while `GET /exercises` returns a **JSON array**. Your implementation must match the specification. You will lose points for any mismatch with the expected response.

Failure Messages in the Response Body Must Match the Specification

- According to the specification there are just 2 distinct JSON objects that can be returned as the body in case of failures. These objects are `{"Error": "Not found"}` and `{"Error": "Invalid request"}`.
- Your REST API implementation must return these objects exactly as these are specified, i.e., the name of the property, the value of the property, and their case must match the specification. You will lose points for any mismatches.

Hints & Suggestions for the REST API

- The REST API for this assignment is similar to the REST API you implemented in Assignment 7.
- You can write the code to validate the request body from scratch, or write it using the package **express-validator (https://express-validator.github.io/docs/)**, 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);
}
```

React App

Implement a React App with the following 3 pages which are described in detail below:

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

Starter Code

We have provided you a zip file **exercises_react.zip (https://canvas.oregonstate.edu/courses/1999527/files/112154332?wrap=1) (https://canvas.oregonstate.edu/courses/1999527/files/112154332/download?download_frd=1)**, which has starter code for your React app.

- Instead of creating a new React app, download the zip file, unzip it into a directory. Then run `npm install` in the directory where you have unzipped the file to install the dependencies, and then `npm run dev` to start the app.
- The zip file has the boiler-plate code for a React app with the following changes
  - The `package.json` file has been updated to include the dependencies `react-router-dom` and `react-icons`.

- Do not change this file.
- The file `vite.config.js` has been updated to add a `proxy` property so that requests sent by the React app to the URL `/exercises` will be sent to the REST API running on port 3000 (as discussed in **Exploration — Implementing a Full-Stack MERN App - Part 1 (https://canvas.oregonstate.edu/courses/1999527/pages/exploration-implementing-a-full-stack-mern-app-part-1)**).
  - Do not change this file.

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 library (https://react-icons.github.io/react-icons/)** , 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 exercise.
- 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 for the React App

Use the Starter React App Provided By Use

- You must use the starter app we have provided you with. This React app listens on port 5173.
- Do not change anything in the file `package.json` unless you get approval from an instructor. You can post on Ed to ask the instructors for approval.

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. For example, you cannot use frameworks like Next.js.
- You also cannot use React Redux.
- You are allowed to use the `useContext` hook. However, the use of this hook is not required.
- You must pre-populate the Edit Exercise page using either the technique of "lifting up the state" described in the exploration or the `useContext` hook. The Edit Exercise page must not send an HTTP request to populate itself.

Use the fetch API & Don't Use Axios

- You must use the Fetch API for sending requests to the REST API.
- You cannot use Axios or another library/package for this purpose.

.jsx Extension for Component Files

- All the files with code for components must have the extension `.jsx` (and not have `.js`).

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.*

- Global page design:
  - Add a `body` rule 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 for font-family.
- 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.jsx` 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> tag (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/nav)** and the **Link component (https://v5.reactrouter.com/web/api/Link)** from `react-router-dom` to have links to the Home Page and the Create Exercise Page.

Hints & Suggestions for the React App

- The React app for this assignment is very similar to the `movie-ui` React app implemented in Module 9. Follow along with the videos in that module to code the `movie-ui` React app and that will help you implement the React app for this assignment.

What to Turn In?

REST API

- Submit the two files `exercises_model.mjs` and `exercises_controller.mjs` with the required functionality implemented in these files.
  - Don't change the names of the files.

- This means that you should not add your ONID to the file name.
- Add a comment with your first name and last name in each file.
- Upload the files separately, i.e., don't put them in a zip file.

- When you resubmit a file in Canvas, Canvas can attach a suffix to the file, e.g., the file name may become `exercises_model-1.js`. Don't worry about this name change, as no points will be deducted because of this.

React App

- For the React app submit a zip file with the development build.
    - When we start the React app, the server must listen on port 5173.
- The zip files must be named `youronid_react.zip` where `youronid` must be replaced by your own ONID.
    - E.g., if `chaudhrn` was submitting the assignment, the file must be named `chaudhrn_react.zip`.
    - When you resubmit a file in Canvas, Canvas can attach a suffix to the file, e.g., the file name may become `chaudhrn_react-1.zip`. Don't worry about this name change as no points will be deducted because of this.
- You must include all the code for your application, **except** the `node_modules` directory.
    - We will deduct 3 points if the zip file includes the `node_modules` directory.
- The grader will go the root directory of the React app, run `npm install` and then `npm run dev` to start the React app and test it.

Videos of a Sample Solution

Here are two videos showing testing a sample solution for our full-stack MERN app.

The first video shows testing the REST API. In the video, we test the REST API by sending requests using the VS Code REST Client. The requests we send are available in the file **a9-test-requests.http (https://canvas.oregonstate.edu/courses/1999527/files/112151116?wrap=1)** **(https://canvas.oregonstate.edu/courses/1999527/files/112151116/download?download_frd=1)** .