Lab 4

New Attempt

Due Oct 31 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip

CS-554 Lab 4

Metropolitan Museum of Art API

For this lab, you will create a Metropolitan Museum of Art API Single Page Application using React.

You will be creating a Single Page Application using React that implements the following routes using **Vite (https://vitejs.dev)** and **react router 6 (https://reactrouter.com/en/main)**.

Your React application will be making Axios calls to the **Metropolitan Museum of Art API (https://metmuseum.github.io)** for data.

The base URL is **https://collectionapi.metmuseum.org/public/collection/v1/**

So for example, if you were querying all Objects, the URL should be

https://collectionapi.metmuseum.org/public/collection/v1/objects (https://collectionapi.metmuseum.org/public/collection/v1/objects)

We will be using 3 of the endpoints from this API:

- **Objects (https://metmuseum.github.io/#objects)** **:** A listing of all valid Object IDs available for access. : **https://collectionapi.metmuseum.org/public/collection/v1/objects (https://collectionapi.metmuseum.org/public/collection/v1/objects)**
- **Object (https://metmuseum.github.io/#object)** **:** A record for an object, containing all open access data about that object, including its image (if the image is available under Open Access): **https://collectionapi.metmuseum.org/public/collection/v1/objects/[ObjectId] (https://collectionapi.metmuseum.org/public/collection/v1/objects/100)**. **https://collectionapi.metmuseum.org/public/collection/v1/objects/100 (https://collectionapi.metmuseum.org/public/collection/v1/objects/100)** for example, get's the object with the ID of 100.
- **Search (https://metmuseum.github.io/#search)** **: Note: This end point will only be used if you implement the search for extra credit.** A listing of all Object IDs for objects that contain the search query within the object's data: **https://collectionapi.metmuseum.org/public/collection/v1/search?q=Mona%20Lisa (https://collectionapi.metmuseum.org/public/collection/v1/search?q=Mona%20Lisa)** There are a number of parameters we can use, but for search, we will just search the data.

It is important to note that Objects and Search ONLY return the Object ID's of the item. You would then have to query the Object endpoint to get the data for each object id returned in the Objects or Search list of object ID's:

For example, when we search "Mona Lisa" we get the following response:

{ "total": 27, "objectIDs": [489409, 816432, 371658, 334332, 425573, 263907, 736844, 284685, 425572, 826812, 482133, 425583, 425571, 425565, 436910, 435765, 488484, 871830, 484216, 337494, 488711, 436658, 436896, 438389, 339116, 339130, 338125] }

We would then have to map through this and call the Object endpoint, passing in each ObjectID returned from the search results.

Pages

/

The root directory of your application will be a simple page explaining the purpose of your site (to talk about The Metropolitan Museum of Art, the API, perhaps your favorite art works!). Be creative with this one!

This page will have a `<Link>` to the Collection(the list of art returned by the API) Listing (`/collection/page/1`),

/collection/page/:page

You will utilize **this endpoint (https://metmuseum.github.io/#objects)** from the API to get the data for this route.

This route will render a paginated list of Objects from the API. It will use the :page param to determine what page to request from the API. If you are on page 1, you will show a button to go to the next page. If you are on page 2+, you will show a next button and a previous button, that will take you to the previous page. If you are on the last page, you will show a previous button. If the Page does not contain any more events in the list, the SPA will redirect to a 404 page. If the :page param is invalid, the SPA will redirect to a 400 page.

The route may also accept a query string of departmentIds, which will limit the responses to only get objects from a specific department. For example, /collection/page/1?departmentIds=11 will get the first page of the list of all European paintings (which is the department with departmentIds=11). You can see a list of the valid departments from the API: **https://collectionapi.metmuseum.org/public/collection/v1/departments (https://collectionapi.metmuseum.org/public/collection/v1/departments)**

/collection/:id

You will utilize **this endpoint (https://metmuseum.github.io/#object)** from the API to get the data for this route.

This route will show details about a single Object. **If the Object does not exist, the SPA will redirect to a 404 page. If the :id is invalid, the SPA will redirect to a 400 page. This must work for EVERY object in the API**

Pagination

The minimum you must provide for a pagination UI:

- **If you are on page 1, you will show a button to go to the *next* page, you should NOT show a previous button if you're on the first page.**
- **If you are on page 2+, you will show a *next* button and a *previous* button, that will take you to the next and previous pages respectfully.**
- **If you are on the last page, you will show a button to go to the *previous* page, you should NOT show the next button if on the last page**

The API makes it very easy to calculate the number of pages that you will have. You will use the following fields to calculate the number of pages based on the data in the API.

For example, When we query the Metropolitan Museum of Art API's objects endpoint you will notice that the JSON return object has the total parameter. `total` shows the total number of objects in the list. We will have **50 objects maximum per page**. Using these fields, we can easily calculate how many pages we will have in the list.

Again, it is SUPER important when working with an API that you read the documentation and understand how the API works and understand the schema of the data returned by the API. So please, please read the documentation and analyze the data returned.

HTML, Look, and Content

Be creative with this assignment!

I do, however, recommend using **Material-UI** (https://mui.com) to make your life easier if you need any UI elements.

As for the data that you chose to display on the details pages, that is up to you but there are some restrictions. You should show as much of the data as possible. For example, the /collection/page/:page route should show objects with their picture (if they have one), their title, the artist's display name, and the object's date. You should also link to its individual page in /collection/:id. Similarly, /collection/:id should show objects with their picture (if they have one), their title, the artist's display name, the artist's display bio, the artist's gender, the object's date, their department, their medium, their classification, their culture, their dimensions, and some additional fields. Some Objects in the data do not have all the fields. If there is a field you are displaying on the front-end, make sure that the data has that field present, if not, display "N/A" or something similar. Do NOT just dump raw JSON to the react components, points will be deducted if you do! Depending on how much data you display and how far you go with it, you may get extra points for going the extra mile and displaying as much data as possible.

Extra Credit

If you add search functionality and it works correctly, you will get 10 points extra. If you implement this, you MUST also paginate through the results if there are many results. I would suggest showing 20 results on each "page" of the search results.

General Requirements

1. Remember to submit your `package.json` file but **not** your `node_modules` folder.
2. Remember to have fun with the content.
3. Remember to practice usage of async / await!