

Lab 4

[New Attempt](#)

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

CS-554 Lab 4

React Marvel API






For this lab, you will create a Marvel API Single Page Application using React. You will be using your Redis lab as the backend for this application. You will create a three new routes for your Express/Redis backend and will be implementing the routes you created for Lab 2 into your React application.

Important Note: For your React Application to make Axios calls to your Express/Redis application, we need to either allow our Express application to receive Cross Origin Requests or we will need to add a proxy to our React Application. Both methods are listed below (You only have to do one or the other)

1. You will need to install the cors package to your Express/Redis application. This is a middleware that allows you to make Cross Origin Requests. You can require this package in your app.js and in your app.js you register the middleware:

```
const cors = require('cors');
app.use(cors());
```

2. Another option is to add a proxy to your React application's package.json: <https://create-react-app.dev/docs/proxying-api-requests-in-development/>  [\(https://create-react-app.dev/docs/proxying-api-requests-in-development/\)](https://create-react-app.dev/docs/proxying-api-requests-in-development/)

As a reminder, you will be using the [Marvel API](https://developer.marvel.com/)  [\(https://developer.marvel.com/\)](https://developer.marvel.com/). You will need to register and sign up for an API key. You will not be able to make requests to the API without signing up and getting an [API key](https://developer.marvel.com/account/)  [\(https://developer.marvel.com/account/\)](https://developer.marvel.com/account/). You will use the [Characters](https://gateway.marvel.com/v1/public/characters?ts=1592417963445&apikey=a8f9ccf932bf29fd379ef00e11668673&hash=f061194023791a1593a0ea861a27da67)  [. Comics](https://gateway.marvel.com/v1/public/characters?ts=1592417963445&apikey=a8f9ccf932bf29fd379ef00e11668673&hash=f061194023791a1593a0ea861a27da67)  [. and Stories](https://gateway.marvel.com/v1/public/comics?ts=1592417963445&apikey=a8f9ccf932bf29fd379ef00e11668673&hash=f061194023791a1593a0ea861a27da67)  [. listings. Please look at the data returned so you know the schema of the data and the objects it returns \(the links to Characters, Comics and Stories above work but using my API key.](https://gateway.marvel.com/v1/public/stories?ts=1592417963445&apikey=a8f9ccf932bf29fd379ef00e11668673&hash=f061194023791a1593a0ea861a27da67)

DO NOT use my API key. Please register for your own. You will need to compose the URL with your API key, a ts (time stamp) and a hash. You will be using ALL the routes you did for lab 2 with the addition to 3 new paginated routes.

As a reminder, you can use the following code to construct the URL. (this example is for characters, you would do the same for stories and comics) you can read more about AUTHORIZING AND SIGNING REQUESTS from the link below

<https://developer.marvel.com/documentation/authorization>  <https://developer.marvel.com/documentation/authorization>

```
const md5 = require('blueimp-md5');
const publickey = 'your_public_key(API KEY) from Marvel dev portal';
const privatekey = 'your private key from Marvel dev portal';
const ts = new Date().getTime();
const stringToHash = ts + privatekey + publickey;
const hash = md5(stringToHash);
const baseUrl = 'https://gateway.marvel.com:443/v1/public/characters';
const url = baseUrl + '?ts=' + ts + '&apikey=' + publickey + '&hash=' + hash;
```

Additional Routes Your Express/Redis Backend Needs:

You will be adding the following routes to your express/redis application. You will paginate through the lists of Characters, Comics, and Stories. The Marvel 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 Marvel API's character list (these properties are also available for comics and stories) endpoint you will notice the following properties in the JSON return object. **total** This field contains the total number of characters. **offset** This shows the offset where we are in the list. It will depend on how many characters you choose to show per page. **limit** This allows us to select how many characters we want per page, the default is 20. Using these fields, we can easily calculate how many pages we will have in the list. For example, if we keep the limit the default of 20, then we would take the total for characters, which is 1562 (at time of writing this assignment, more characters could be added/removed from the API at any time), and divide that by the limit of 20. $1562/20 = 78.1$ pages of characters. We then can use the offset property to tell where we are in the list. For example, the first page has an offset of 0, if we keep the default limit of 20, and then move to the next page, the offset will be 20 for page 2, for page 3, the offset would be 40 and so on.... You can supply the offset and limit values as URL parameters to the API's request endpoint.

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.

/api/characters/page/:pagenum

This route will do two things:

- 1) Check if the page has a cache entry in redis. If so, render the result for a page from the cache.
- 2) If the page is not cached, you will query the data from the Marvel API for the the requested page and fail the request if they are not found, or send JSON returned from the API for that particular page and cache the results for that page.

/api/comics/page/:pagenum

This route will do two things:

- 1) Check if the page has a cache entry in redis. If so, render the result for a page from the cache.
- 2) If the page is not cached, you will query the data from the Marvel API for the the requested page and fail the request if they are not found, or send JSON returned from the API for that particular page and cache the results for that page.

/api/stories/page/:pagenum



This route will do two things:

- 1) Check if the page has a cache entry in redis. If so, render the result for a page from the cache.
- 2) If the page is not cached, you will query the data from the Marvel API for the the requested page and fail the request if they are not found, or send JSON returned from the API for that particular page and cache the results for that page.

So for all of these three routes, when you make an Axios request from your React Application for a specific page, you will first check to see if you have the list of characters/comics/stories cached in Redis for that page. If you do, your express server will respond with the JSON data from the cache for that page. If that page is not in cache, you will then query the API for that page, respond with the data the API returned and then cache the data for that page in Redis.

React Components/Routes

You will be creating a Single Page Application using React that implements the following routes and makes Axios requests to your Express server.

You will be using [create-react-app](https://github.com/facebookincubator/create-react-app)  (<https://github.com/facebookincubator/create-react-app>) and [react router 6](https://reactrouter.com/docs/en/v6/getting-started/overview)  (<https://reactrouter.com/docs/en/v6/getting-started/overview>) to create your React application.

/

The root directory of your application will be a simple page explaining the purpose of your site (to talk about Marvel, the API etc..).

This page will have a [<Link>](#) to the Characters Listing (</characters/page/1>), The Comics Listing (</comics/page/1>), and the Stories Listing (</stories/page/1>)

/characters/page/:page

This route will render a paginated list of Marvel Characters from your Express/Redis Application. It will use the :page param to determine what page to request from our server. 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 the Page does not contain any more Characters in the list, the SPA will redirect to a 404 page. In this component, you will also have a link to the /characters/history component in this component.**

/characters/:id

This route will show details about a single Character. **If the Character does not exist, the SPA will redirect to a 404 page. This must work for EVERY character in the API**

/characters/history

This route will query your </api/characters/history> route on your Express/Redis Application and will display your list of the 20 most recently accessed characters. You should display the data for each character in your history list and render them into elements. Do NOT just display the raw JSON data.

/comics/page/:page

This route will render a paginated list of comics. It will use the `:page` param to determine what page to request from our server. 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 the Page does not contain any more comics in the list, the SPA will redirect to a 404 page.**

`/comics/:id`

This route will show details about a single comic. **If the comic does not exist, the SPA will redirect to a 404 page. This must work for EVERY comic in the API**

`/stories/page/:page`

This route will render a paginated list of stories. 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 the Page does not contain any more stories in the list, the SPA will redirect to a 404 page.**

`/stories/:id`

This route will show details about a single story. **If the series does not exist, the SPA will redirect to a 404 page. This must work for EVERY story 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.
- 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 only show the previous button.

Pagination HINT: Look at the data.. You can also do the pagination by using the following fields in the data: `"offset": 0, "limit": 20, "total": 1493, "count": 20`

You can supply the limit and offset values as URL parameters to the API's request URL. **AGAIN, READ the API documentation for this.**

HTML, Look, and Content

Besides the specified settings, as long as your HTML is valid and your page runs passes a [tota11y](http://khan.github.io/tota11y/)  (<http://khan.github.io/tota11y/>) test, the general look and feel is up to you. If you want to update it, you may; if you do not, that is fine.

As for the data that you chose to display on the details pages, that is up to you. You should show as much of the data as possible. For example, a single character has comics associated with it. It would be nice if you add links to the comics details page(in your application) for a comic that a character appears in, also perhaps link to the characters that are in a comic on the comic details page. You should at LEAST show the image for the character/comic/story, the name of the Character/Comic/Story and some additional fields. 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 either to characters, comics or stories you will get 5 points extra for each that you implement so you have the potential to get 15 points extra credit. **You do NOT have to cache any of the search results. You can either query the API directly from your React application to make this work (easier way to do it), or create additional routes on your Express/Redis application to make it work.**

General Requirements

1. Remember to submit your `package.json` file but **not** your `node_modules` folder.
2. Remember to run HTML Validator and tota11y tests!
3. Remember to have fun with the content.
4. Remember to practice usage of async / await!