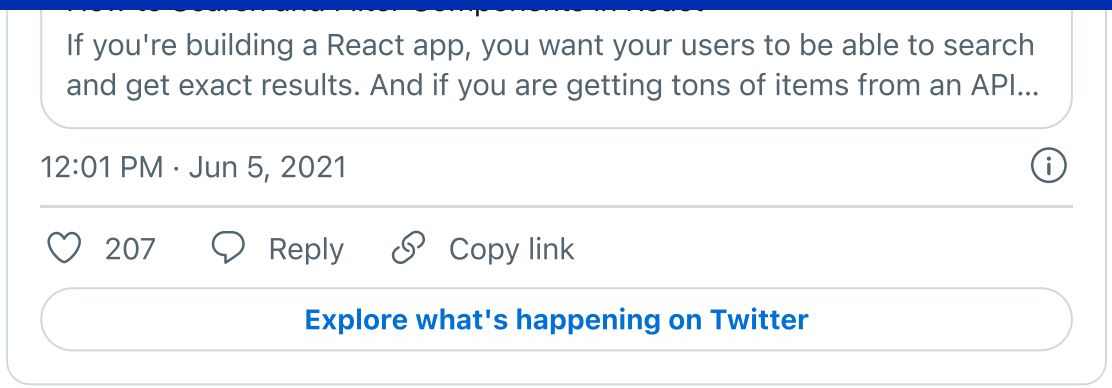
JUNE 7, 2022 / #REACT

# React Components – How to Create a Search, Filter, and Pagination Component in React

Spruce Emmanuel

I wrote the article "How to Search and Filter Components in React" exactly one year ago.

> **freeCodeCamp.org** ✓
> @freeCodeCamp · **Follow**
>
> If you're building a React app, you want your users to be able to search & get exact results.
>
> And if you're getting data from an API, they need to be able to find things easily.
>
> Here @sprucekhalifa shows you how to search & filter components in React.



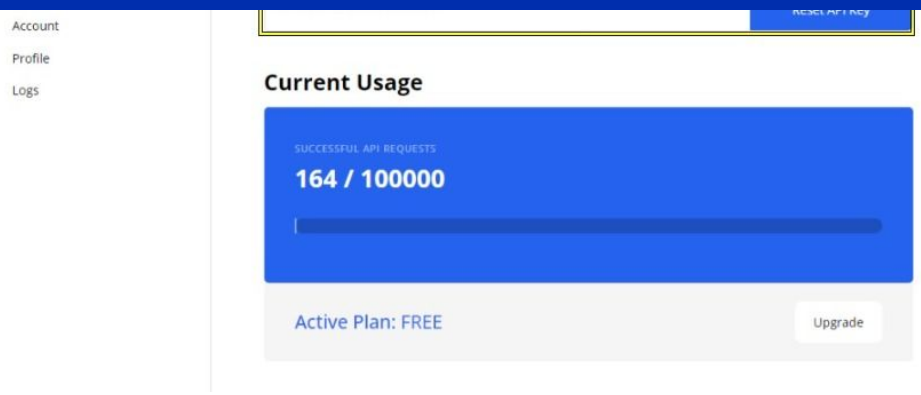

Since then, a lot has changed. The API we used for the tutorial has stopped working, so in this article, we'll recreate our previous examples while also introducing pagination to our component.

This tutorial makes use of React.js, so you'll need a basic understanding of React and JavaScript to follow along. This tutorial also assumes that you've read the previous article, How to Search and Filter Components in React.

# Getting Started

To get the country data for this tutorial, we will use the CountryAPI from CountryAPI.io.

We'll need an API key to use the API. To get your API key, go to CountryAPI and create an account. Your API key should appear on your dashboard:




Next we'll go create a new React app with Create React App. To do that run the following in your terminal:

```
# Run this to use npm
npx create-react-app search-app
# Or run this to use yarn
yarn create react-app search-app
cd my-app
npm start
# Or with yarn
yarn start
```

As usual, for live preview we are going to be using Codepen to display all our examples.

# How to Fetch the Data

To get our data, we must make a GET call to the `https://countryapi.io/api/all` endpoint while supplying our API key. In the **src** > **App.js** file of the React app that we created previously, delete all existing code and replace it with the following:

```
import { useState, useEffect } from "react";
import "./App.css";

function App() {
```

```
  useEffect(() => {
    const request_headers = new Headers();
    const api_key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx";
    request_headers.append("Authorization", `Bearer ${api_key}`);
    request_headers.append("Content-Type", "application/json");

    const request_options = {
      method: "GET",
      headers: request_headers,
    };

    fetch("https://countryapi.io/api/all", request_options)
      .then((res) => res.json())
      .then(
        (result) => {
          setLoaded(true);
          setItems(result);
        },
        (error) => {
          setLoaded(true);
          setError(error);
        }
      );
  }, []);

  console.log(items)

  if (error) {
    return <>{error.message}</>;
  } else if (!loaded) {
    return <>loading...</>;
  } else {
     return (
     //
    );
  }
 }

export default App;
```

All we did above was use the JavaScript fetch API to make the GET request to our endpoint and then store the returned data in the `items` state using `setState(result)`.

## How to display the data

To make a list, we'll need to generate an array of objects from the values in our returned object. Open **src > App.js** file and add the following code:

```
import { useState, useEffect } from "react";
import "./App.css";

function App() {
  const [error, setError] = useState(null);
  const [loaded, setLoaded] = useState(false);
  const [items, setItems] = useState([]);

  useEffect(() => {
    // fetch data
  }, []);

  const data = Object.values(items);

  if (error) {
    return <>{error.message}</>;
  } else if (!loaded) {
    return <>loading...</>;
  } else {
    return (
      <div className="wrapper">
        <ul className="card-grid">
          {data.map((item) => (
            <li key={item.alpha3Code}>
              <article className="card">
                <div className="card-image">
                  <img src={item.flag.large} alt={item.name} />
                </div>
                <div className="card-content">
                  <h2 className="card-name">{item.name}</h2>
                  // other card content
              </article>
            </li>
          ))}
        </ul>
      </div>
    );
  }
}

export default App;
```

With some added CSS, the example above looks like the preview below:


# How to Create the Search Component

First thing is to create an input field where users can enter their search query. Open **src** > **App.js** and make the following edit:

```
...

function App() {
 const [query, setQuery] = useState("");

  if (error) {
    return <>{error.message}</>;
  } else if (!loaded) {
    return <>loading...</>;
  } else {
    return (
      <div className="wrapper">
        <div className="search-wrapper">
          <label htmlFor="search-form">
            <input
              type="search"
              name="search-form"
              id="search-form"
```

```
            <span className="sr-only">Search countries here</span>
          </label>
        </div>

        ...
      </div>
    );
  }
}

export default App;
```

Above we created an input field and, using the `onChange` event handler, anytime the value of the input field changes we set the value to `query` using the `useState` hook.

Next we'll need to use that `query` to filter the data returned from our API.

```
const search_parameters = Object.keys(Object.assign({}, ...data));

 function search(data) {
    return items.filter(
      (item) =>
        search_parameters.some((parameter) =>
          item[parameter].toString().toLowerCase().includes(query)
        )
    );
  }
```

Let's break this down a bit. First we created a function `search()` which takes in our `data` as an argument. Combining the `Array.filter()` and `Array.some()` methods we checked if any of our Search Parameters include the value of our query `includes(query)`.

Of course we can hardcode our search parameters:

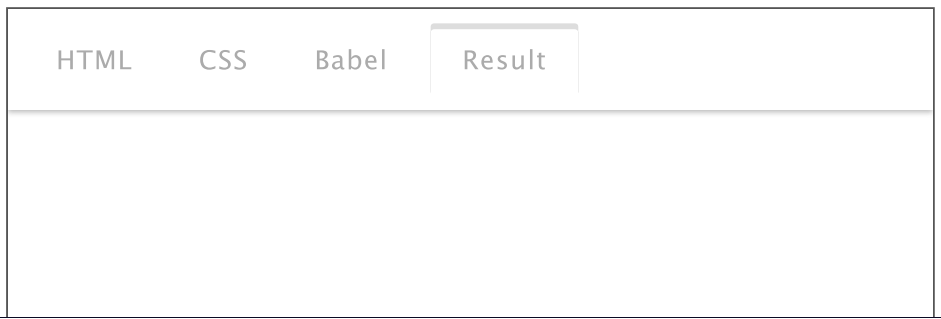
And while this is the fastest way, it is not future proof (data returned from an API may change) – I found this out the hard way. So instead of hardcoding we can get the search parameters from the `data` returned from the API.

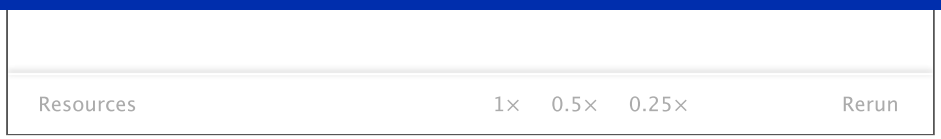
```
const search_parameters = Object.keys(Object.assign({}, ...data));
```

Lastly we need to use the new data returned from the `search(data)` function to build our country lists. Open the **App.js** file and edit the list we created earlier:

```
...
{search(data).map((item) => (
 <li key={item.alpha3Code}>
  <article className="card">
   <div className="card-image">
    <img src={item.flag.large} alt={item.name} />
     </div>
   <div className="card-content">
    <h2 className="card-name">{item.name}</h2>
    ...
   </div>
   </article>
 </li>
))}
```

With the search functionality added, the live preview now looks like this:






## How to Create the Filter Component

A filter is often used to group data by a specific keyword. In this example we want to group data by regions.

Again, instead of hardcoding this, we can get the regions from the data:

```
const filter_items = [...new Set(data.map((item) => item.region))];
```

After the search input field we created, add the the filter UI – its an HTML `select` input field:

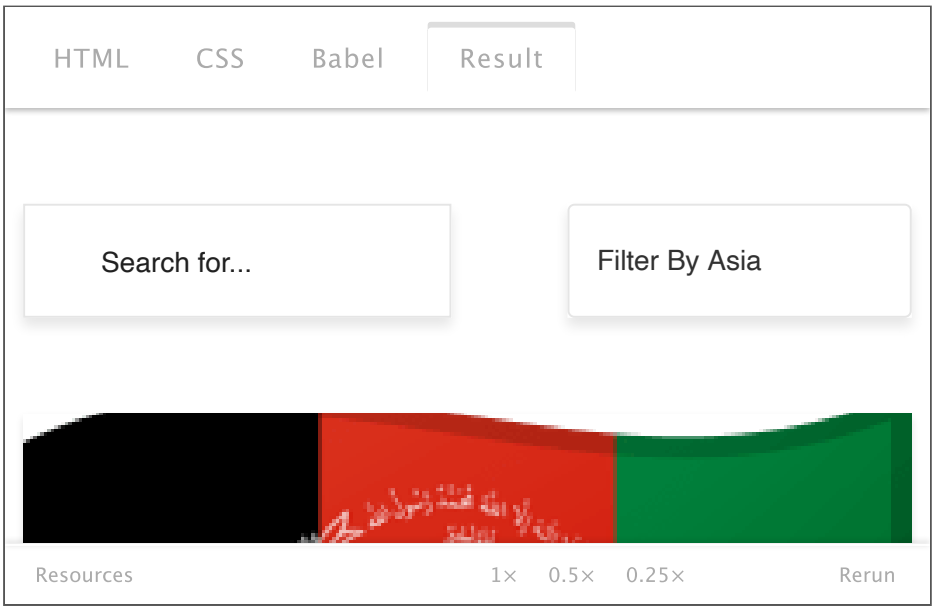
```
...
const [filter, setFilter] = useState("");
...

<div className="select">
 <select
  onChange={(e) => setFilter(e.target.value)}
  className="custom-select"
  aria-label="Filter Countries By Region">
  <option value="">Filter By Region</option>
  {filter_items.map((item) => (
   <option value={item}>Filter By {item}</option>
  ))}
</select>
<span className="focus"></span>
</div>
```

To add the filter, we have to modify the `search(data)` function. So instead of returning only the data we searched for, we are now returning the data

```
function search(items) {
    return items.filter(
      (item) =>
        item.region.includes(filter) &&
        search_parameters.some((parameter) =>
          item[parameter].toString().toLowerCase().includes(query)
        )
    );
  }
```

And there you have it – we can now filter our countries by region. The live preview and full code can be found on Codepen:




# How to Create the Pagination Component

Pagination doesn't just reduce the lists of items displayed on a web page – it also increases the app's performance since users only have to download few resources when page is loading.

To create pagination for our countries data, we'll start by specifying the number of items we want to display in a `useState`:

```
const [paginate, setpaginate] = useState(8);
```

Next let's use our `paginate` value to update our countries list:

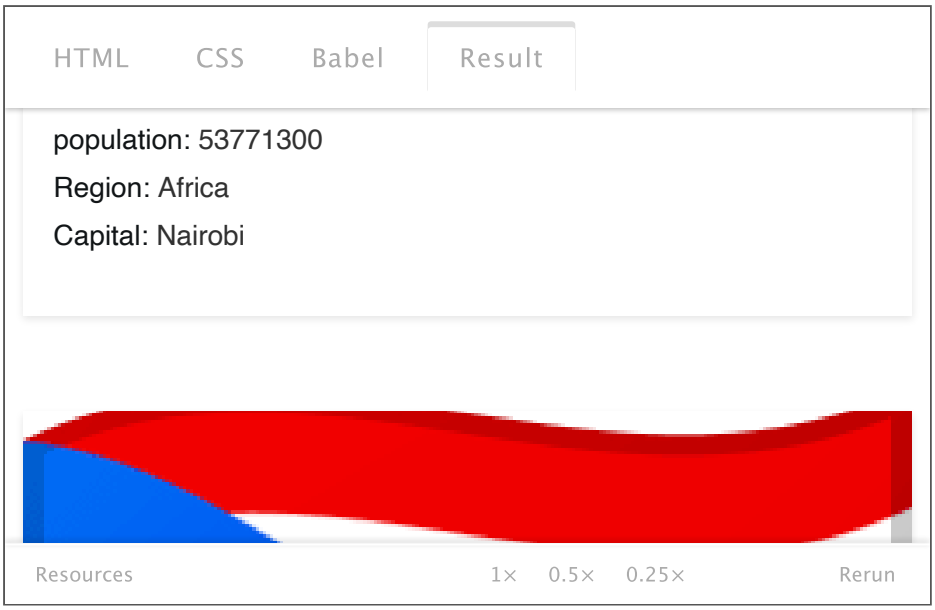
```
...
{search(data)
 .slice(0, paginate)
 .map((item) => (
 <li key={item.alpha3Code}>
  <article className="card">
   <div className="card-image">
    <img src={item.flag.large} alt={item.name} />
     </div>
   <div className="card-content">
    <h2 className="card-name">{item.name}</h2>
    ...
   </div>
   </article>
 </li>
))}
```

Next we need to create a function where we update this state anytime we call it:

```
const load_more = (event) => {
  setpaginate((prevValue) => prevValue + 8);
};
```

Lastly, let's create a button that will call the `load_more` function when clicked:

```
<button onClick={load_more}>Load More</button>
```




# Conclusion

In this article we looked at how to implement search, filter and pagination functionality in React by building a real world app using the CountryAPI from CountryAPI.io.

If you created something wonderful with this, please feel free to tweet about it and tag me @sprucekhalifa. And don't forget to hit the follow button.

Happy coding!

---

**Spruce Emmanuel**

Web Developer && Technical Writer

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.
Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

**You can make a tax-deductible donation here.**

**Trending Guides**

- Discord Won't Open
- Learn Java Courses
- SQL Like Statement
- File Explorer Error
- Python Find in List
- Functions in Python
- Python Reverse List
- Create a Table in SQL
- List Index Out of Range
- Python String.Replace()
- How to Open Task Manager
- Design Thinking Explained
- Learn Programming Courses
- C vs C++
- Python JSON
- What is a PC?
- What is Coding?
- Text Box in HTML
- Meta Tag in HTML
- Append in Python
- Python Not Equal
- Linux awk Command
- JS String Contains
- Default Constructor in Java
- Stuck Win 10 Hard Drive
- Color Codes for Grey Palette