# **Building a Pokemon search application**

How to get started with using React by building an application for searching Pokemon by using [Poke API](https://pokeapi.co/docs/v2#pokemon) and making it publically accessible via Github Pages.

## **Prerequisites**

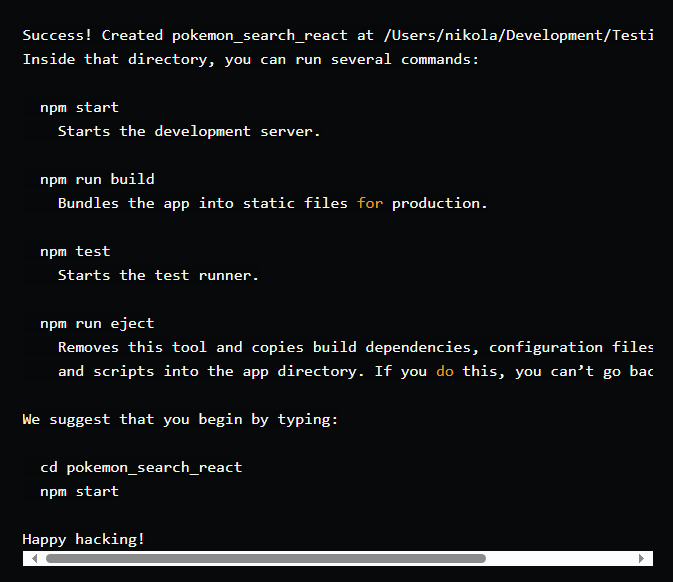
Make sure that you have the following tools installed:

* Node.js - here's a free but outdated [step by step guide/book](https://leanpub.com/meantodo) for both Windows and Mac. Or, really, just go to the [main website](https://nodejs.org/) and download the executable for your machine
* Git - here's a fun [getting started](https://nikola-breznjak.com/blog/miscellaneou/learn-git-fast-job-depends/) tutorial in case you're new to it

## **Start a new app with create-react-app**

* To start the project, run the following command (in your Terminal): npx create-react-app pokemon\_search\_react

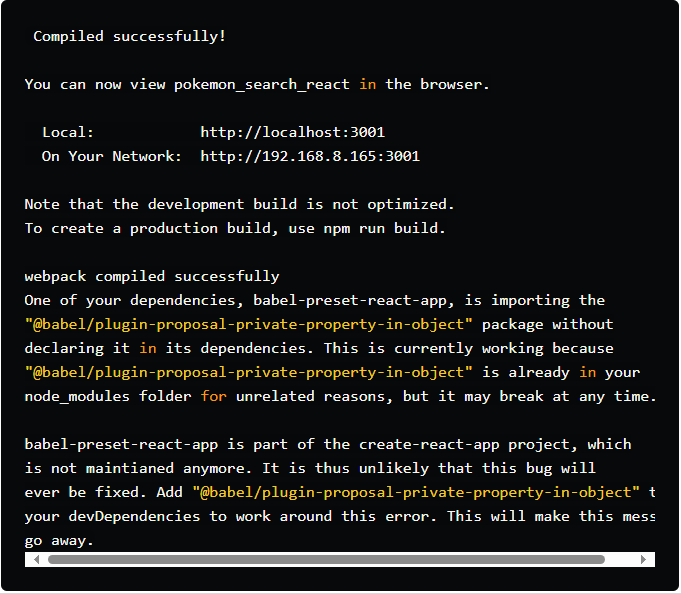
The output of that command will be similar to the one below:



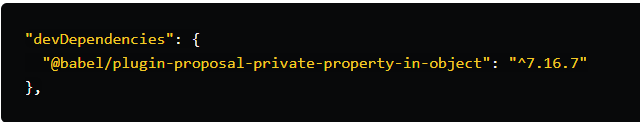
## **Running our scaffolded project**



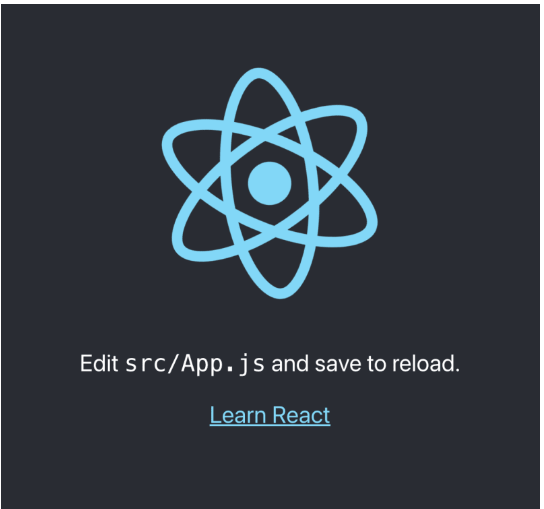
You should get this something similar to this output:



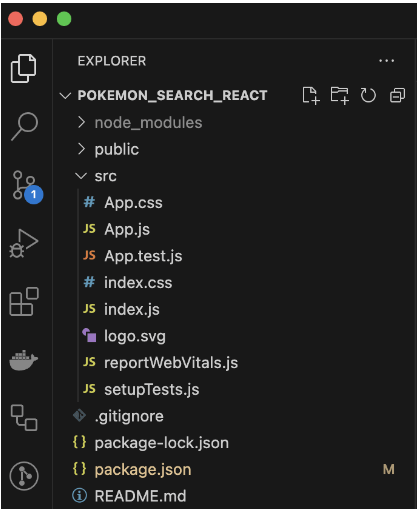
⚠️ If you're bothered by the seeming error-like output, then just add the following code to your package.json file:



You should see the following page in your browser if you open [http://localhost:3000](http://localhost:3001/)



## **Folder structure**



## **Add content**

if you search for the string Learn React, you'll see the string is within the App.js file. This file contains the following code:



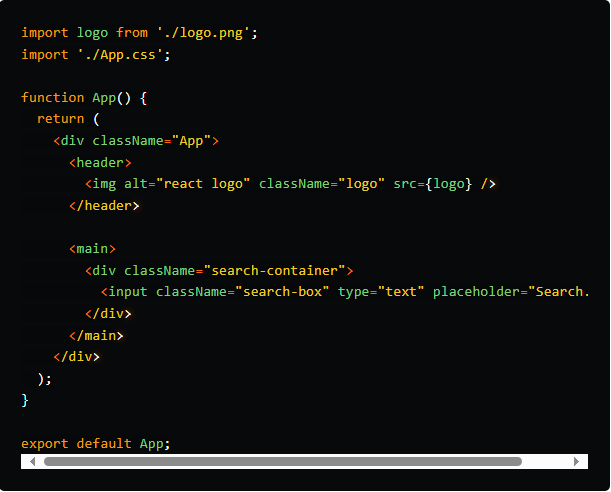
change that to Welcome to Pokemon Search.

Save that, you'll see the change reflected immediately in your browser.

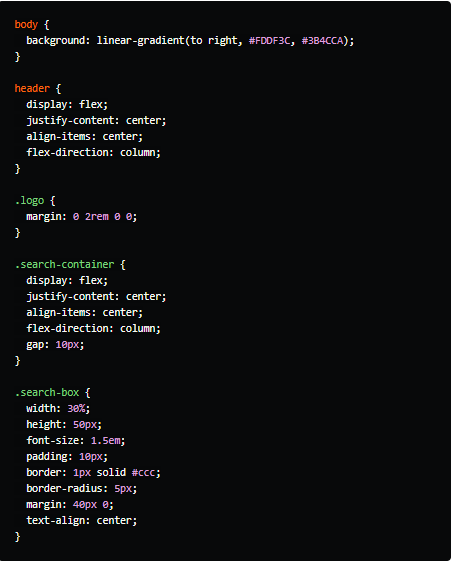
## **Add some style**

Borrow the image from [Wikipedia](https://en.wikipedia.org/wiki/Pok%C3%A9mon) and save it in the src folder as **logo.png**, added a search input, and styled it a bit to match it with the Pokemon style.

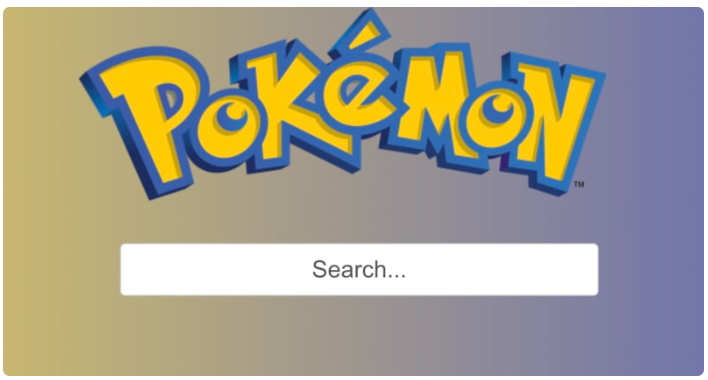
Here's the final code that I came up with in App.js:



And the CSS in the App.css file:



The way this looks now is as follows:



## **Pokemon API**

To fetch some data from the PokeAPI and show it in our app.

Google search for pokemon api and open the [first link](https://pokeapi.co/), you'll also get to the [documentation](https://pokeapi.co/docs/v2) for their API.

By looking at the docs, you can find that the API endpoint that lists all the available Pokemon is this: <https://pokeapi.co/api/v2/pokemon>. There's a total of 1302 records.

Now, fetching all this data every time you load your app would be a bad idea from the performance standpoint, but also from their [fair use](https://pokeapi.co/docs/v2#fairuse) policy.

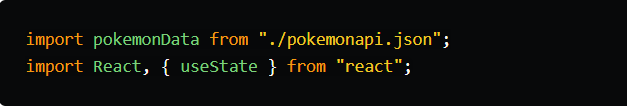
So, what we will do is open [this link](https://pokeapi.co/api/v2/pokemon?limit=1320) which will fetch all 1320 Pokemon endpoints. The output looks like this:



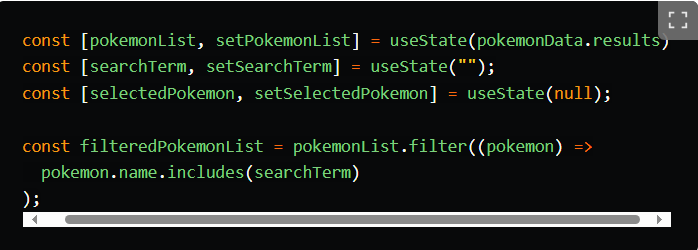
Now, save this output (just Command + S or Ctrl + s if you're on Windows) to a local file that we'll name pokemonapi.json and place it in the src/ folder.

## **Show the data**

Now we'll want to list all the Pokemon that we have in the pokemonapi.json file. To do that, we'll add some code at the top of the App.js file that looks like this:



And then, inside the function App():



filteredPokemonList function returns the list of pokemon that we loaded from the pokemonapi.json file and filters them by what we enter in the input box.

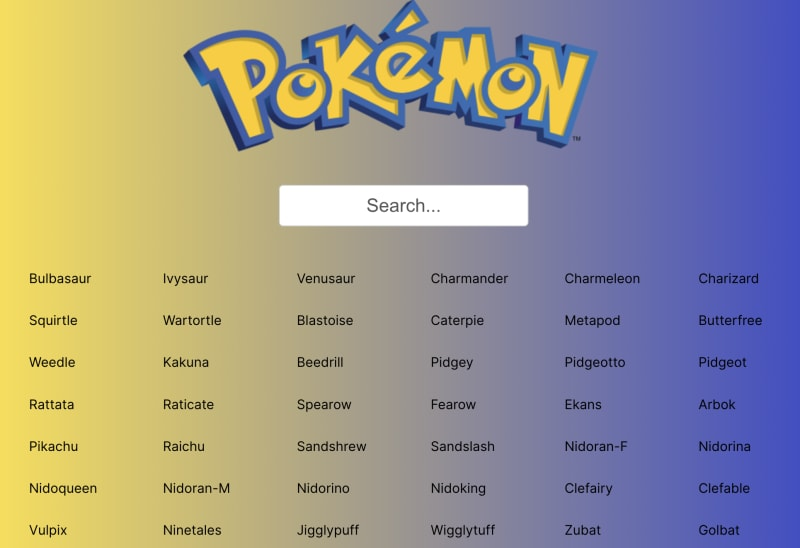
We'll add this portion of JSX just below the search-container div:



we'll add a bit of CSS:



Doing this, you should have your app look like this:

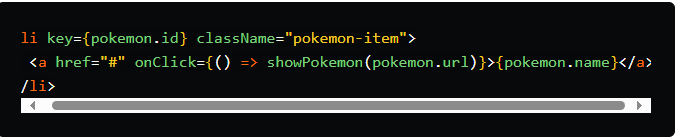


## **Get additional details about each Pokemon**

If you open any of the URLs that you see in the pokemonapi.json file, you will get an output like this:

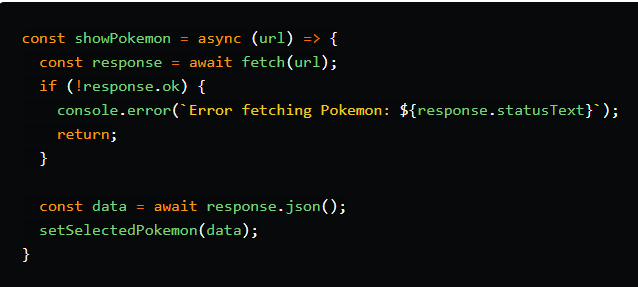


To show some additional information about a certain Pokemon (once clicked on its name in the list), we'll update the a tag inside the li:

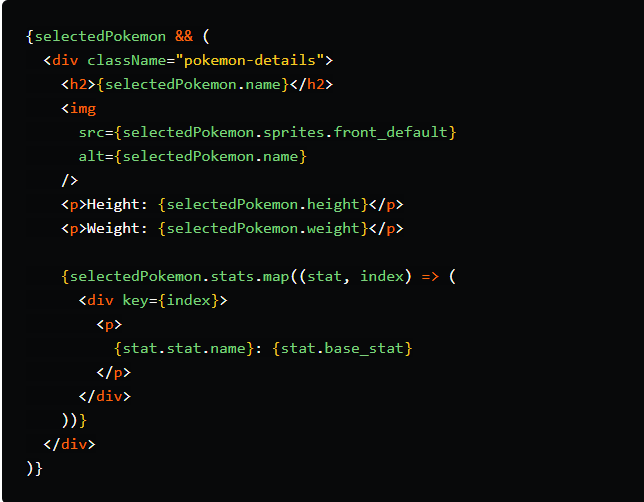


On the click event we attached the showPokemon function (by using onClick) and passed it the URL that we get in the main API call (loaded from the JSON file).

Now we should define this function:



To show the selected Pokemon, we'll add a bit of JSX between the search-container div and the ul:



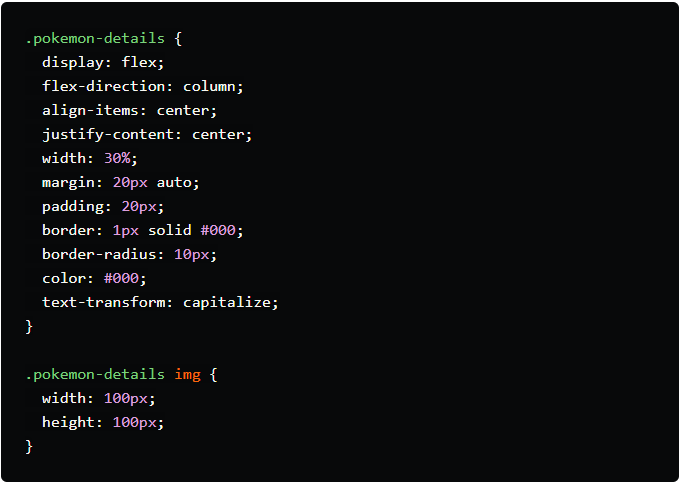
We're using selectedPokemon && so that the line {{ selectedPokemon.name }} doesn't throw an error for trying to access a property that doesn't exist on the object selectedPokemon (which is initialized as null in the beginning: const [selectedPokemon, setSelectedPokemon] = useState(null);).

BTW, useState is a so-called React Hook that essentially enables you to update a certain variable.

Anyways, back on point; we're using map to loop through all the elements of the stats array and output the name and base\_stat properties

.

To make it look nicer, we'll use a bit of CSS:



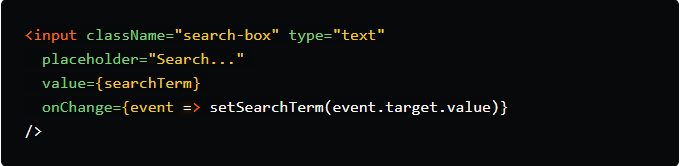
## **Two-way data binding**

Right now if you test your app and type something in the search box, it will not filter the list of Pokemon.

You may remember that in Vue.js, we achieved this by using the v-model. However, in React, there isn't a direct equivalent to Vue's v-model for two-way data binding.

In React we can achieve a similar effect by using a combination of state and event handlers.

To do this, we'll update the input tag to this:



In this code, searchTerm is a state variable that is set as the value of the input field. The onChange event handler updates searchTerm whenever the user types something into the input field. This creates a two-way data binding effect similar to Vue's v-model.

## **All the code**

here's the full listing of App.js file:

import logo from "./logo.png";

import "./App.css";

import pokemonData from "./pokemonapi.json";

import React, { useState } from "react";

function App() {

const [pokemonList, setPokemonList] = useState(pokemonData.results);

const [searchTerm, setSearchTerm] = useState("");

const [selectedPokemon, setSelectedPokemon] = useState(null);

const filteredPokemonList = pokemonList.filter((pokemon) =>

pokemon.name.includes(searchTerm)

);

const showPokemon = async (url) => {

const response = await fetch(url);

if (!response.ok) {

console.error(`Error fetching Pokemon: ${response.statusText}`);

return;

}

const data = await response.json();

setSelectedPokemon(data);

};

return (

<div className="App">

<header>

<img alt="react logo" className="logo" src={logo} />

</header>

<main>

<div className="search-container">

<input className="search-box" type="text" placeholder="Search..."

value={searchTerm}

onChange={event => setSearchTerm(event.target.value)}

/>

</div>

{selectedPokemon && (

<div className="pokemon-details">

<h2>{selectedPokemon.name}</h2>

<img

src={selectedPokemon.sprites.front\_default}

alt={selectedPokemon.name}

/>

<p>Height: {selectedPokemon.height}</p>

<p>Weight: {selectedPokemon.weight}</p>

{selectedPokemon.stats.map((stat, index) => (

<div key={index}>

<p>

{stat.stat.name}: {stat.base\_stat}

</p>

</div>

))}

</div>

)}

<ul>

{filteredPokemonList.map((pokemon) => (

<li key={pokemon.id} className="pokemon-item">

<a href="#" onClick={() => showPokemon(pokemon.url)}>

{pokemon.name}

</a>

</li>

))}

</ul>

</main>

</div>

);

}

export default App;

And here's the full contents of the App.css file:

body {

background: linear-gradient(to right, #FDDF3C, #3B4CCA);

}

header {

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

}

.logo {

margin: 0 2rem 0 0;

}

.search-container {

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

gap: 10px;

}

.search-box {

width: 30%;

height: 50px;

font-size: 1.5em;

padding: 10px;

border: 1px solid #ccc;

border-radius: 5px;

margin: 40px 0;

text-align: center;

}

.pokemon-item {

float: left;

margin: 10px;

}

.pokemon-item a {

color: #000;

text-decoration: none;

font-size: 16px;

transition: color 0.3s ease;

text-transform: capitalize;

}

.pokemon-item a:hover {

color: #3B4CCA;

}

ul {

display: grid;

grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));

gap: 5px;

list-style: none;

}

.pokemon-details {

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

width: 30%;

margin: 20px auto;

padding: 20px;

border: 1px solid #000;

border-radius: 10px;

color: #000;

text-transform: capitalize;

}

.pokemon-details img {

width: 100px;

height: 100px;

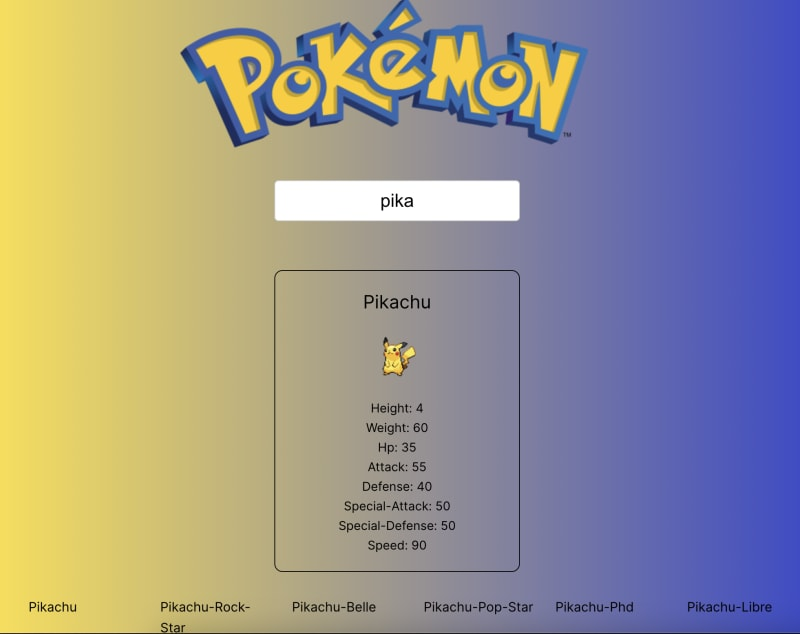
}

.pokemon-details p {

margin: 5px;

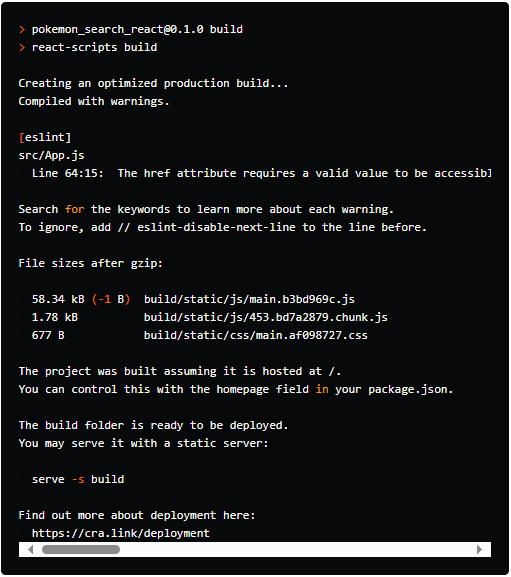
}

At this point, we can search for some Pokemon (my son's favorite is, ofc, Pikachu), and if we click on it, we'll get this:



## **Deployment**

If you'd like to host this on your web server, then first run npm run build and you'll get an output similar to this:



Now all you have to do is take the contents of the build folder (you may remember that in Vue that folder was called dist) and 'paste' it on your static web server.

If you don't have a server of your own, then Vite has an extensive [description](https://vitejs.dev/guide/static-deploy) for deploying your static pages to many popular services like Github Pages, Netlify, Vercel, Surge, etc.

You can deploy to Github Pages in under 2 minutes by following their [documentation](https://pages.github.com/).

Just for brevity sake the steps are as follows:

* create a new public Github repository and name it username.github.io, where username is your username on GitHub.
* clone the repo with git clone https://github.com/username/username.github.io
* inside the folder copy the contents of the build folder
* commit and push the changes:



Now your site will be visible online at [https://username.github.io](https://username.github.io/) (again, where username is your Github username)

You can see my deployment live [here](https://hitman666.github.io/pokemon-search-react/).

⚠️ If you'll have multiple folders in Github pages, then before using the npm run build command, you'll want to set the homepage variable in the package.json file to that folder. Example: "homepage": "/pokemon-search-react".