Rest Countries API Function

Content

This is a full-stack program. I used **React + NodeJS + Express** as the architecture of this project.

https://restcountries.com/#rest-countries

This URL provides all the APIs for processing country information, this contains detailed information about each country and various APIs for processing the data.

For this project, I have selected a few of these APIs. which are used to return the properties of each country, which includes the common name, the continent, the population, the area, and the Google Map geographic location.

Especially for processing the Google Map API. We input a shortUrl and we must use Backend Server to convert shortUrl into long Url.

There are only 1 file in backend and 5 files in frontend is this project. Here is the structure of this program.
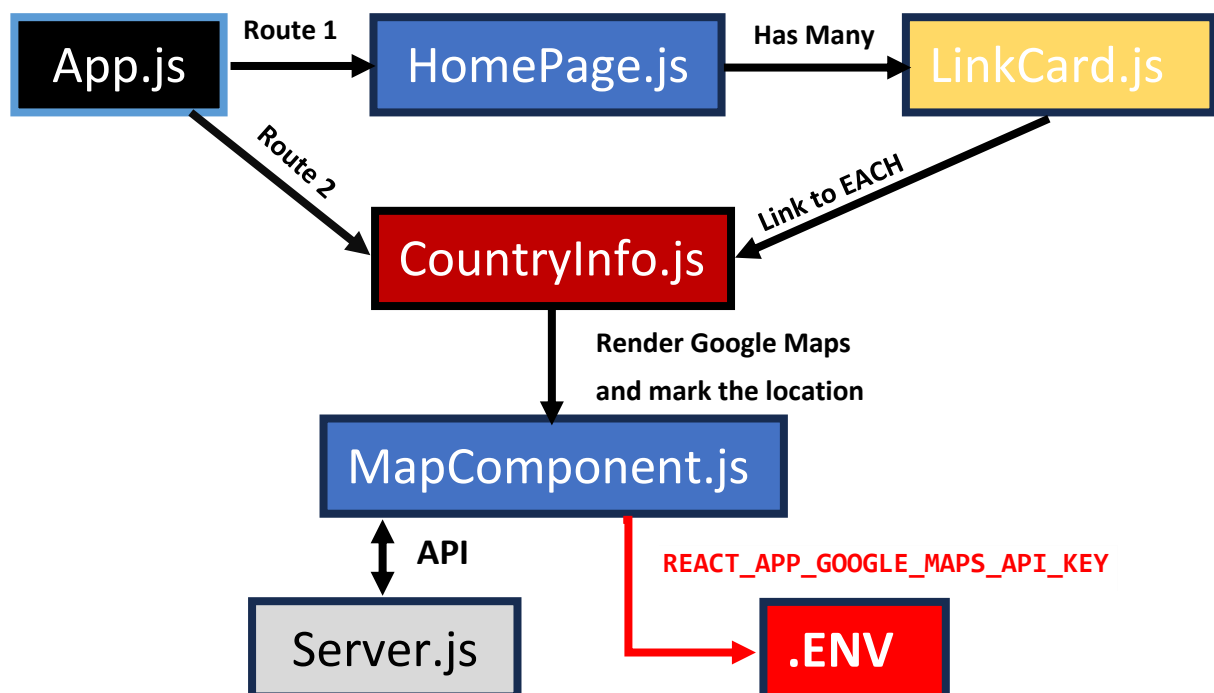
Backend:

- **Server.js**

Frontend:

- **App.js**
- **HomePage.js**
- **LinkCard.js**
- **CountryInfo.js**
- **MapComponent.js**

I draw a picture to explain the relationship between each component.

```
App.js  --Route 1-->  HomePage.js  --Has Many-->  LinkCard.js
  |                                                     |
  --Route 2-->                              Link to EACH
                 CountryInfo.js  <------------------
                        |
                        | Render Google Maps
                        | and mark the location
                        v
                 MapComponent.js
                   |          \
                   | API       \  REACT_APP_GOOGLE_MAPS_API_KEY
                   v            v
                Server.js      .ENV
```

Then I will explain how we connect Google Map API with React.

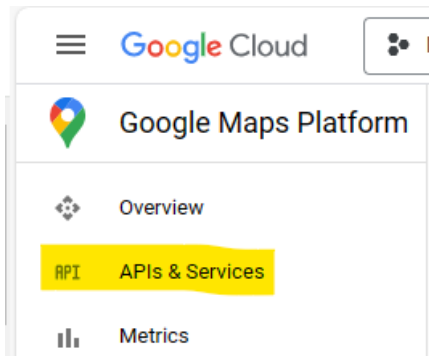**There are two files that we need to check carefully –server.js and MapComponent.js.**

How to get Google Map API?

1) **Register a Google Account**
2) **Go into the website**
   https://console.cloud.google.com/google/maps-apis/discover
3) **Click API and Services on the left.**

4) **Select the country.**

5) **Account type should be individual. Type in to your personal information and bank card information, then "START FREE"**

6) Create a new project

Select the project

Search for projects and folders

🔍

7) Enable APIS and Services

╋ ENABLE APIS AND SERVICES

8) Select Maps JavaScript API, then enable it.

JS

Maps JavaScript API
Google

Maps for your website

9) That is the Google Map API key, copy and save it.

Get started with Google Maps Platform

You're set up and ready to start developing! Here are the API keys you'll need for your implementation. The API key can be found in the Credentials section.

Your API key
...yA-Zb8x............2c4   ⧉

☑ Create budget alerts to help me stay informed about my spending and let me   ❷
know when I'm about to exceed my $200 Google Maps monthly credit
threshold

GO TO GOOGLE MAPS PLATFORM

10)   Don't set any restrictions.

Hide your Google Map API Key

That API key is confidential, remember not to show it to anyone else!!

1) We created a `.env` file in the folder – frontend.

This file stores the **secret information!**

2) We use process.env to get information from ` .env ` file

If we want to get information from **.env** file, the variable should be starts with **"REACT_APP".** Check **Map.Component.js**, line 21

`process.env.REACT_APP_GOOGLE_MAPS_API_KEY`

3) **Create `.gitignore` file to avoid uploading sensitive information on Github when you deploy your project**

MapComponent.js

```javascript
import React,{useEffect, useState} from 'react';
import { GoogleMap, LoadScriptNext, Marker } from '@react-google-maps/api';

const containerStyle = {
  width: '400px',
  height: '400px'
};

function MyMapComponent (props) {
  const [latLng, setLatLng] = useState(null);

  useEffect(() => {
    fetch(`https://rest-countries-api-backend.vercel.app/api/resolve-map-url?url=${props.mapUrl}`)
        .then(response => response.json())
        .then(data => setLatLng(data))
        .catch(error => console.error('Error:', error));
  }, [props.mapUrl]);


  return(
    <LoadScriptNext googleMapsApiKey={"AIzaSyAlqrv7wVOdkifjOOqKrTEIAsso8yAoe0U"}>
      {latLng && (
        <GoogleMap mapContainerStyle={containerStyle} center={latLng} zoom={5}>
          <Marker position={latLng} />
        </GoogleMap>
      )}
    </LoadScriptNext>
  );
}

export default MyMapComponent;
```

`LoadScriptNext googleMapsApiKey={...}`

This React component is used for loading the Google Maps. **LoadScriptNext** is a high-level component that loads the Google Maps library. You need to provide it with a valid **Google Map API key**. **LoadScriptNext** is used to ensure that Google Maps scripts are loaded and initialized before rendering map-related components.

`{latLng && (...`

This is a JavaScript **conditional rendering expression**.

It checks whether the latLng variable exists (or is a "true value"). **If latLng has a value** (e.g. an object containing latitude and longitude), then **the bracketed components are rendered**. **If latLng is null or undefined, nothing is rendered.**

This is a common pattern used to avoid rendering when the data is not yet ready.

`<GoogleMap mapContainerStyle={containerStyle} center={latLng} zoom={5}>`

GoogleMap component is used to display a Google Map instance on the page.

1) mapContainerStyle: define the style of the map container, such as width and height.

2) center: set the center point of the map, here the value of latLng variable is used. It contains objects for latitude (lat) and longitude (lng).

3) zoom: set the initial zoom level of the map. The higher the zoom level, the smaller the map display.

`<Marker position={latLng} />`

The Marker component is used to **place a marker on the Google Map**. Its position property specifies the position of the marker, again using the **latLng** variable.

server.js

```javascript
const express = require('express');
const request = require('request');
const cors = require('cors');

const app = express();
app.use(cors());

app.get('/api/resolve-map-url/', (req, res) => {
    const shortUrl = req.query.url;

    request({ url: shortUrl, followRedirect: false }, (err, response, body) =>
{
        if (err) {
            return res.status(500).send('Error occurred');
        }

        const longUrl = response.headers.location || shortUrl;
        const regex = /@([0-9.-]+),([0-9.-]+),/;
        const matches = longUrl.match(regex);


        if (matches && matches.length >= 3) {
            const latLng = { lat: parseFloat(matches[1]), lng:
parseFloat(matches[2]) };
            res.json(latLng);
        } else {
            res.status(404).send('Coordinates not found');
        }
    });
});

// Test the Vercel
app.get("/", (req, res) => {
    res.send("You succeeded to deploy backend to Vercel!");
});

const port = 5000;
app.listen(port, () => console.log(`Server running on port ${port}`));
```

`req.query`

In the Express framework, **req.query** is an object that gets the parameters of the query parameter. The query parameter is part of the URL that follows the path and **starts with ?** The format is usually a **key=value** pair, with multiple key-value pairs **separated by &**. Such as http://example.com/api/items?color=blue&size=large

Query Parameter is **color=blue&size=large**, then
   req.query.color = "blue"
   req.query.size ="large"

The front-end passes in
`fetch(``https://rest-countries-api-backend.vercel.app/api/resolve-map-url?url=${props.mapUrl}``)`
Then the backend will handle:
   Query string is `url=${props.mapUrl}`
   req.query.url = `${props.mapUrl}`

`request`

Request is a function in Node.js for making HTTP requests. An external API is called here for the purpose of **converting short URLs to long URLs.**

Entering a **shortUrl** request here will result in a response **response.headers.location**

`followRedirect: false`

Disable HTTP redirects.

HTTP redirection is a method of responding to a web request in which the server tells the client to go to another URL than the one originally requested. Redirects are very common in web development and are used in a variety of scenarios, such as web page relocation, short-link serving, load balancing….

If you visit a web page, the server may return a **301-status code** and a new URL, in which case your browser will automatically jump to the new URL.

The Request function does not automatically follow the new URL when it encounters an HTTP redirect (commonly used for short link services.) **This means that even if the shortUrl points to another address, the request will not automatically go to that address.**

`shortUrl -> longUrl`

There is an website that you can try to convert shortUrl into longUrl： https://urlex.org/

For example, if we cope with the location of China.

`shortUrl:` https://goo.gl/maps/p9qC6vgiFRRXzvGi7

longUrl:

https://www.google.com/maps/place/China/@34.4137724,86.0453361,4z/data=!3m1!4b1!4m5!3m4!1s0x31508e64e5c642c1:0x951daa7c349f366f!8m2!3d35.86166!4d104.195397?ucbcb=1

REGEX

```
const regex = /@([0-9.-]+),([0-9.-]+),/;
```

1）begin with **@** , because the format of google Map longUrl is**@latitude, longtitude, zoom**

2）**([0-9.-]+)**

At least one of the following three options

 1] digit 0 to 9

 2] DOT .

 3] Minus-

3）**Capture Groups – in the bracket ()**

/(pattern1) (pattern2)/

The first capture group captures pattern1 and the second capture group captures pattern2

4）match

string.match(regexp);

If **match** is used with **capture group**, then an **array** of is returned.

Such as the code in the project
```
const matches = longUrl.match(regex);
```

`matches` is an array

  matches[0] : The **entire matched string**, including the **@** and **two coordinates.**

  matches[1]: The match result of the first **capture group -- latitude.**

  matches[2]   The match result of the first **capture group -- longitude.**

parseFloat

Convert String into Float

HomePage.js

26-letter search link

```javascript
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import LinkCard from './LinkCard';

function HomePage(){
    const [data, setData] = useState([]);
    const [selectedLetter, setSelectedLetter] = useState('');

    useEffect(() => {
      axios.get('https://restcountries.com/v3.1/all')
        .then(response => {
          setData(response.data);
          console.log(response.data);
        })
        .catch(error => console.error('Error fetching data:', error));
    }, []);

    function handleLetterClick (letter) {
      setSelectedLetter(letter);
    }

    function FilterCountriesByLetter(selectedLetter){
      return function (countryObject) {
        if(selectedLetter === ''){
          return true;

        }
        else{
          return countryObject.name.common
          .normalize("NFD").replace(/[\u0300-\u036f]/g, "")
          .startsWith(selectedLetter);
        }
      }
    }

    return(
      <div>
        <h1>Bonuce Insights Challenge</h1>
        <h3>
          Selected Letter: <span style =
{{color:'blue'}}> {selectedLetter} </span>
        </h3>
```

```
      <h3>Select by the Capital:</h3>
      <div>
        {/* Show All Countries */}
        <button onClick={() => handleLetterClick('')}>All</button>
        {/* A - Z */}
        {Array.from({ length: 26 }, (_, i) => String.fromCharCode(65 +
i)).map(letter => (
          <button key={letter} onClick={() => handleLetterClick(letter)}>
            {letter}
          </button>
        ))}
      </div>

      <div style={styles.container}>
        {data.filter(FilterCountriesByLetter(selectedLetter))
          .map((item, index) =>
            <LinkCard key={index} data={item} />
          )}
      </div>
    </div>
  );
}

const styles = {
  container: {
      display: 'flex',
      flexDirection: 'row',
      flexWrap: 'wrap',
      justifyContent: 'center',
  }
};

export default HomePage;
```

This code snippet is for **generating a list of alphabetical buttons from A to Z in React**. It uses JavaScript's **Array.from** method and **map function** to realize**.

```
{Array.from({ length: 26 }, (_, i)
    => String.fromCharCode(65 + i)).map(letter => (
        <button key={letter} onClick={() => handleLetterClick(letter)}>
            {letter}
        </button>
    )
}
```

1) Array.from({ length: 26 }, (_, i) => String.fromCharCode(65 + i)):

Array.from: This method is used to create a new array instance from an iterable object.

{ length: 26 }: This defines **an array with 26 element**, because the English alphabet has 26 letters.

(_, i) => String.fromCharCode(65 + i): This is an **arrow function** which is used to **fill in an array**. This function operates on e**ach element of the array.**

The _ (underscore) is a reference to the current element, but it won't be used here, so the element here is used to indicate that the argument is ignored.

i is the index of the current element.

String.fromCharCode(65 + i): This expression is used to **generate letters from A to Z. The ASCII code for the uppercase letter A is 65, B is 66**, and so on. So **65 + i** here **generates a sequence of characters from 65 (A) to 90 (Z).**

2) .map(letter => ...):

letter is the **current element (current letter)** in the array.

For **each letter in the array,** the **map function** returns **a React <button> component.**

key={letter}: When you render a list of elements in React, each element should have **a unique key attribute**. The key here is each letter itself.

onClick={() => handleLetterClick(letter)}: This is a **click event handler**. **When the button is clicked**, it calls the **handleLetterClick function**, **passing the current letter as an argument to the function.**

In one word, the map function dynamically creates a list of 26 buttons, each representing a capital letter of the English alphabet. Clicking on these buttons triggers an event handler that filters the country by first letter based on the letter clicked.

```
function handleLetterClick (letter) {

    setSelectedLetter(letter);
}
```

With a keystroke, the selected letter is changed and saved in React Hook.

13

```
    function FilterCountriesByLetter(selectedLetter){
      return function (countryObject) {
        if(selectedLetter === ''){
          return true;
        }
        else{
          return countryObject.name.common
          .normalize("NFD").replace(/[\u0300-\u036f]/g, "")
          .startsWith(selectedLetter);
        }
      }
    }


    <div style={styles.container}>
      {data.filter(FilterCountriesByLetter(selectedLetter))
        .map((item, index) =>
          <LinkCard key={index} data={item} />
      )}
    </div>
```

Filter Function

First, this function return an array of **countryObject, then traverse each element.**

Then a **conditional expression** is returned.

For each element, the country Object will only be returned out if the conditional expression is true.

If the conditional expression is false, then the country Object will not be returned.

StartWith(letter) is used here to determine if the string has a first letter as required.

Convert Special Characters into Letters

```
normalize("NFD").replace(/[\u0300-\u036f]/g, "")
```

.normalize("NFD")

.normalize("NFD") is a string method in JavaScript that **converts a string to its Normalization Form Decomposed.** This is useful when dealing with internationalized characters that c**ontain accent marks and other type modifiers.**

When a character (such as "Å" or "é") can be represented in Unicode by **a combination of the base character and one or more modifiers**, **canonical decomposition** breaks down these combined characters into their base components.

`.replace(/[\u0300-\u036f]/g, "")`

/[\u0300-\u036f]/g   is a regular expression that matches accents and other modifiers in Unicode.
replace method replaces all characters matching this range with the empty string (removes them).

This code snippet first converts the string to its canonical decomposed form, breaking all compound characters into basic characters and modifiers. It then removes all modifiers, leaving only the basic characters.

Suppose we have the string **"Åland".** In Unicode, **"Å"** can be decomposed into **"A"** and **a circle** superscript.
"Åland" is processed by .normalize("NFD") to become "A∘land" (where "∘" represents a circle superscript).
Then .replace(/[\u0300-\u036f]/g, "") removes **"∘"** and the final result becomes "Aland".