

Rest Countries API Function

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.

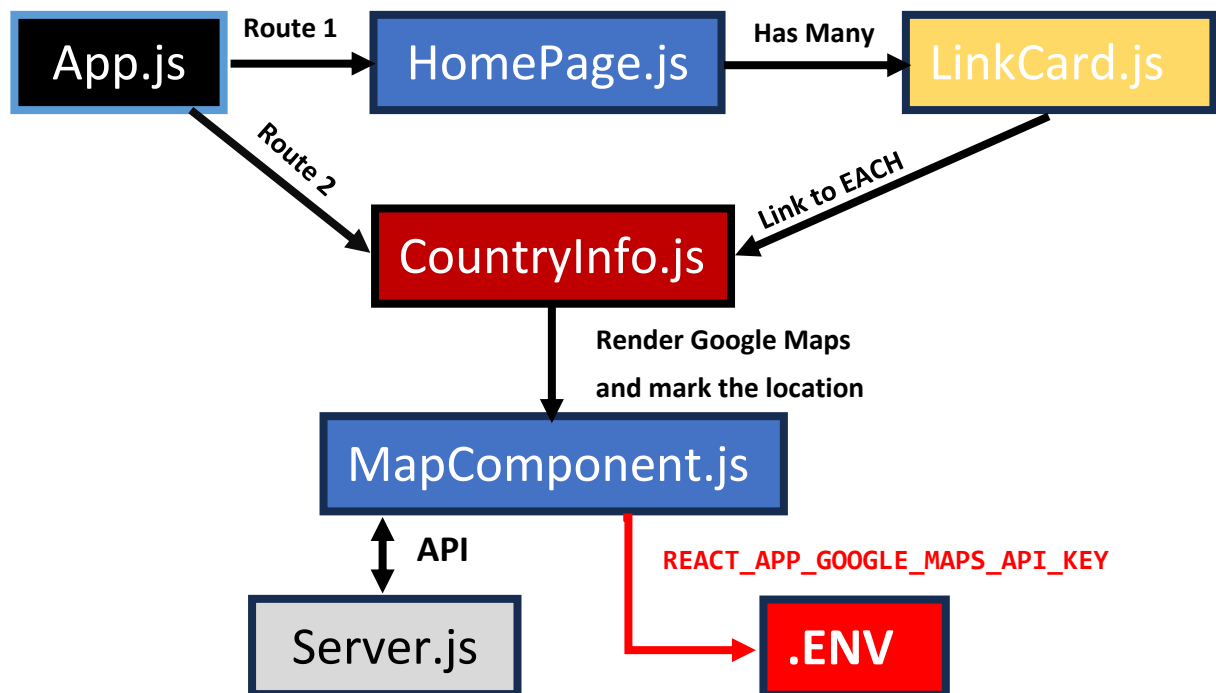
Backend:

- **Server.js**

Frontend:

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

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



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.

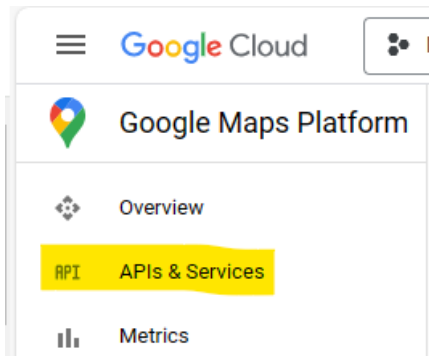
How to get Google Map API?

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.

Step 1 of 2 Account Information






[SWITCH ACCOUNT](#)

Country

By using this application, you agree to the [Google Cloud Platform](#), [Supplemental Free Trial](#), and [any applicable services and APIs Terms of Service](#).

[AGREE & CONTINUE](#)

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

 Account type  

Individual

[START FREE](#)

6) Create a new project

Select the project



CREATE A NEW PROJECT

Search for projects and folders



7) Enable APIs and Services

[+ ENABLE APIS AND SERVICES](#)

8) Select Maps JavaScript API, then enable it.



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 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.

```
frontend > .env
1  REACT_APP_GOOGLE_MAPS_API_KEY = [redacted]
```

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`

```
14
15   return(
16     <LoadScriptNext googleMapsApiKey={process.env.REACT_APP_GOOGLE_MAPS_API_KEY}>
17       {latLng && (
18         <GoogleMap mapContainerStyle={containerStyle} center={latLng} zoom={5}>
19           <Marker position={latLng} />
20         </GoogleMap>
21       )}
22     </LoadScriptNext>
23   );
24 }
```

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

```
frontend > .gitignore
1  .env
```

MapComponent.js

```
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={"AIzaSyAlqrv7wV0dkifj00qKrTEIAsso8yAoe0U"}>
      {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

```
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, longitude, 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(regex);
```

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