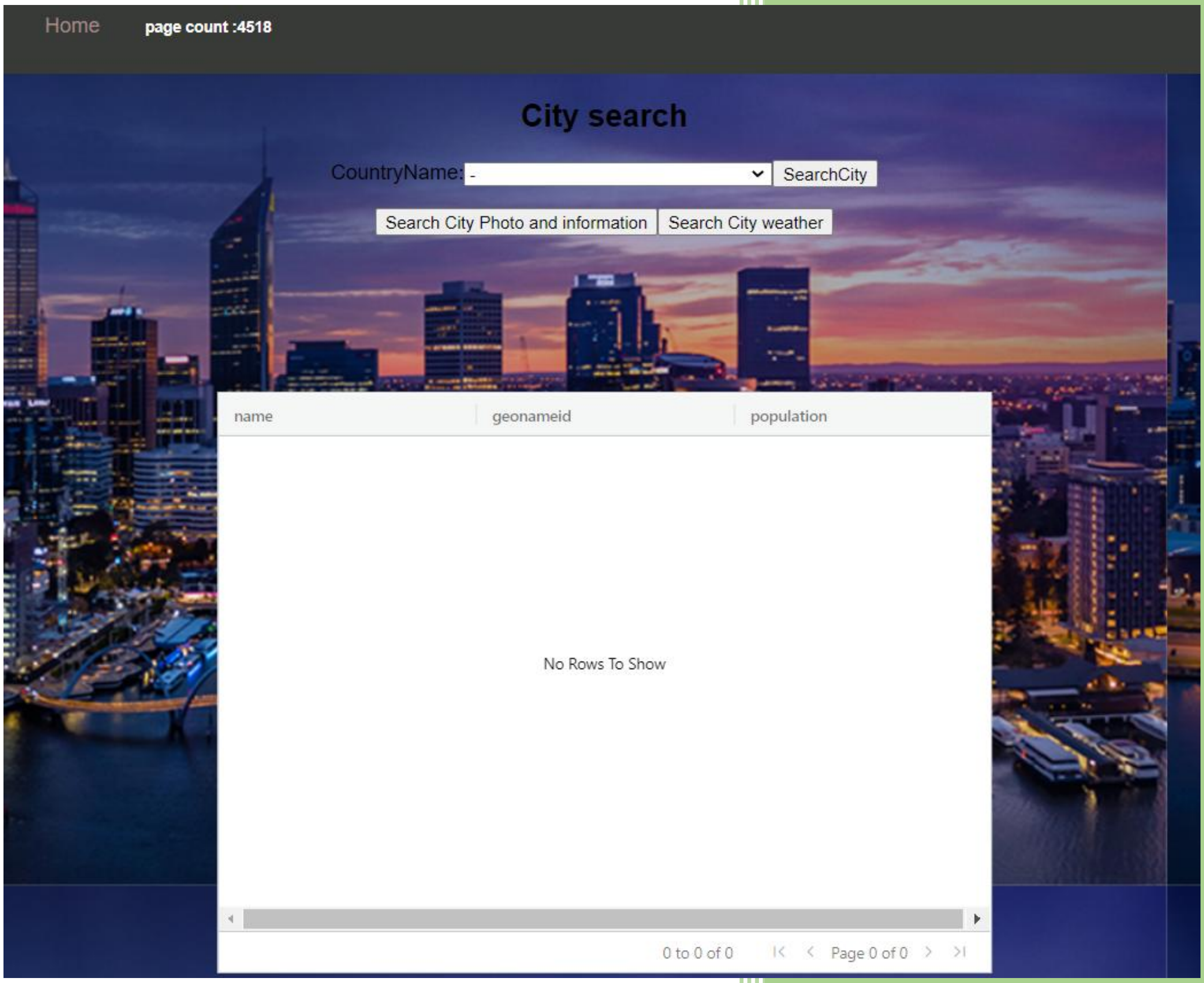
2022

<city search>



CAB432 Assignment 1

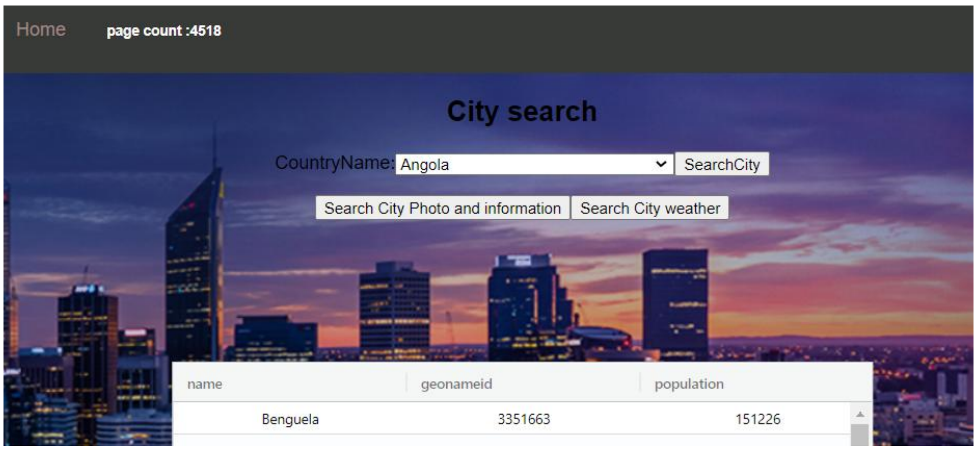
<Haobo Jin>
<N10642536>
9/21/2022

Contents

Introduction

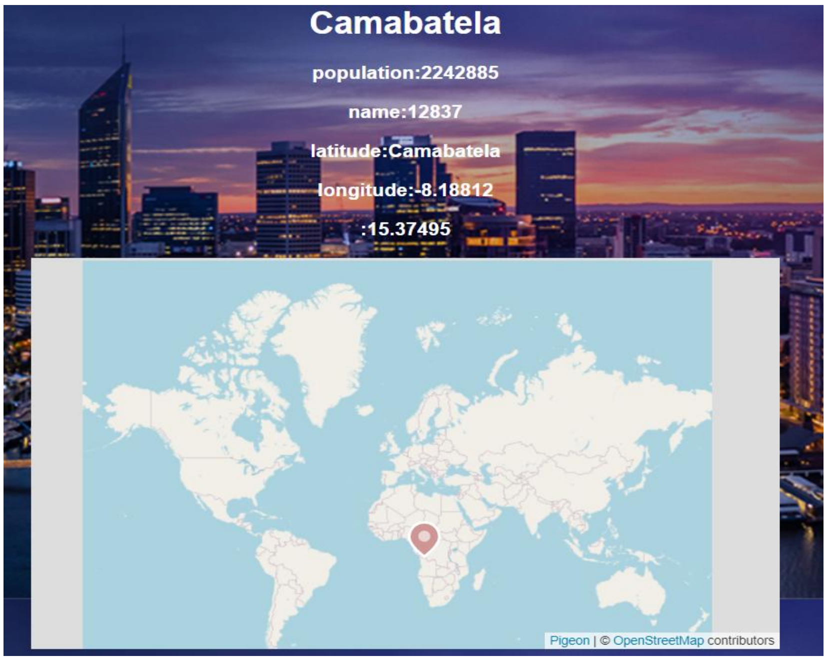
Mashup Purpose & description

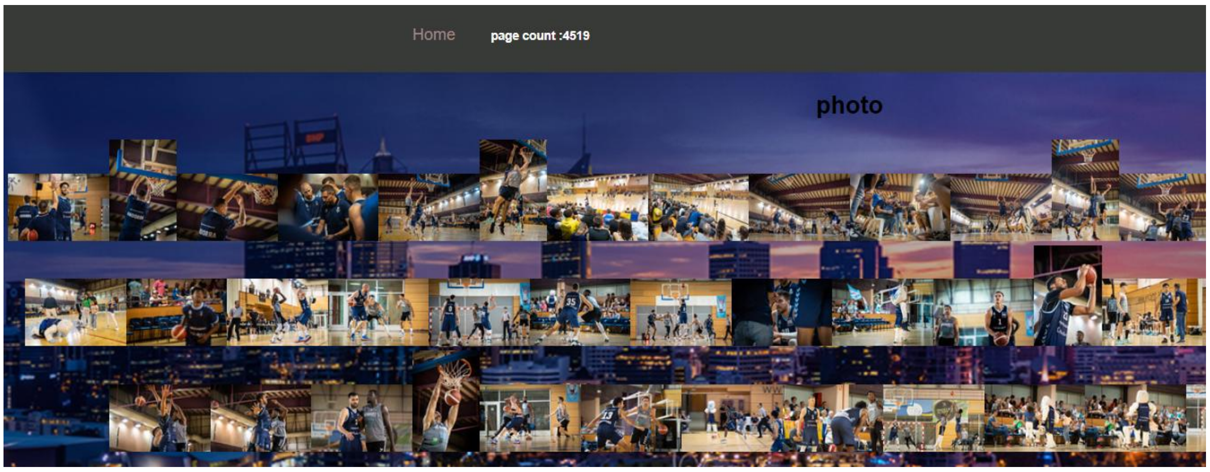
This application is called City Search, it is an application that mashup countries ,cities and its weather and photos. The purpose of making this application is to allow the user to get detailed information about the destination and the weather conditions and to have more knowledge about the destination. It is an application that mashup countries ,cities and its weather and photos API. The application has the following four main functions, 1. obtain information about the city which the country belongs, 2. obtain detailed information about a specific city and display the location of the city in the form of a map. 3. obtain photographs about the city 4. obtain weather data about a specific area and display the maximum and minimum temperature of the area in a graph.
As shown in the image below, the user selects the country in the drop down bar, gets information about the specific city and clicks on the button to get the city's geographical location, area photos and weather data.



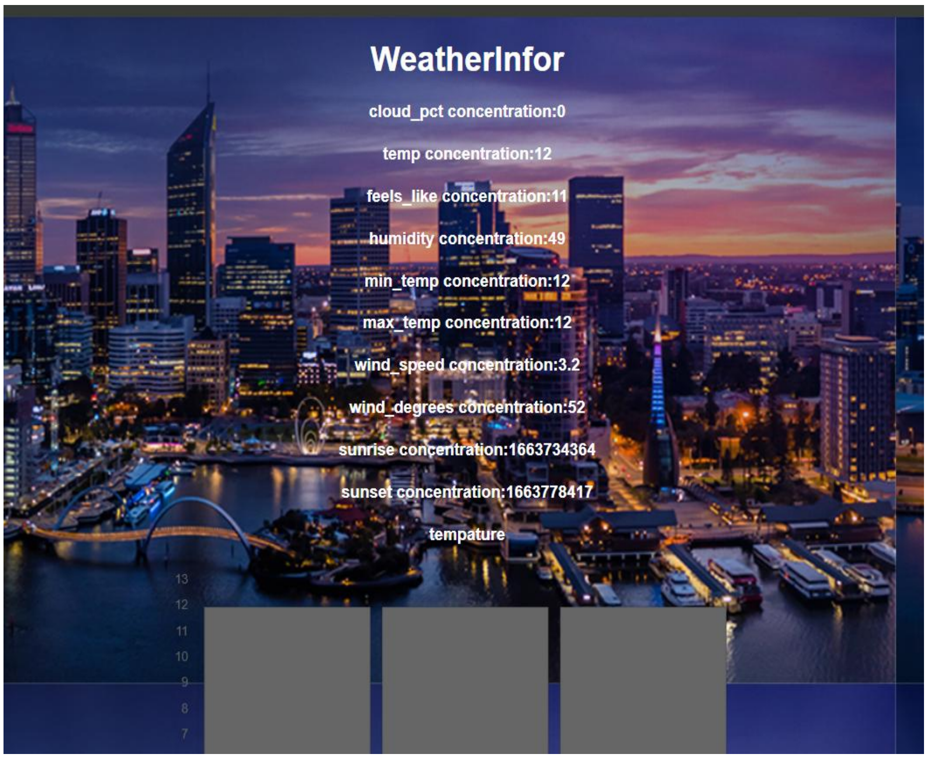
name	geonameid	population
Benguela	3351663	151226

Show city information and photos





Get weather information


WeatherInfor

cloud_pct concentration:0

temp concentration:12

feels_like concentration:11

humidity concentration:49

min_temp concentration:12

max_temp concentration:12

wind_speed concentration:3.2

wind_degrees concentration:52

sunrise concentration:1663734364

sunset concentration:1663778417

Services used

Battuta API

Return all country name and the country as a json

Endpoint: http://battuta.medunes.net/api/country/all/

Docs: http://battuta.medunes.net/

World Geo Data API

Returns all city information for a specific country, such as geonameid, population, latitude, longitude, etc.

Endpoint: https://world-geo-data.p.rapidapi.com/countries/GB/cities

Docs: https://rapidapi.com/natkapral/api/world-geo-data/

Weather by API
Return to get the latest weather data for specific cities

Endpoint: https://weather-by-api-ninjas.p.rapidapi.com/v1/weather

Docs: https://rapidapi.com/apininjas/api/weather-by-api-ninjas/

Flikr API
Returns a bunch of images and photos matching a specified query - may extend to the Flickr website

to see more information.

Endpoint: https://api.flickr.com/services

Docs: https://www.flickr.com/services/api/misc.urls.html

Persistence Service
DynamoDB

Used to store Table and data, connect with node's express, get or change data in table

Docs:https://ap-southeast2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#item-explorer?initialTagKey=&table= 10642536

Docker

Could deploy the program as an image to docker and then pull the image down in aws' virtual machine and run it as a countainer.

Docs :https://hub.docker.com/repository/docker/hulalalalala/10642536

Mashup Use Cases and Services

Get city information and photos

As a	Passengers
I want	Get the geographical location of a specific city and mark it on the map
So that	I can get a better idea of the location of the city.

As a	Passengers
I want	Access to city-specific data, relevant photos
So that	I can determine whether to visit the city or not.

I get all the country names and code data from the Battuta API and display it in the drop down bar. after the user selects a specific country, the code for that country is passed into the World Geo Data API. world Geo Data API returns information about the specified city, displays this information on the react grid .When the user clicks on the grid, the system gets the name of the city selected by the user and displays the information about the city, displaying the latitude and longitude of the city on a map. In addition the use case uses the flikr API to get a photo of the city .

Get city weather and temperature charts

As a	Passengers

I want	Get information about the weather in a particular city at this time.
So that	Determine what I need to pack for my visit to the city

As a	Passengers
I want	Get the maximum and minimum temperatures for a specific city, displayed in a graph
So that	A better visualisation of the local temperature

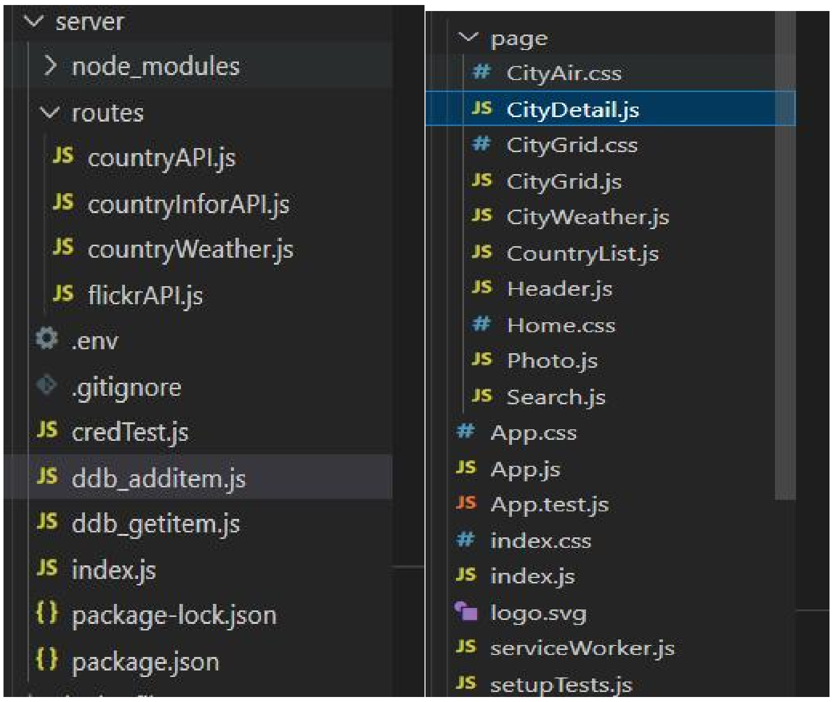
Similar to the use case, the name and code data for all countries is obtained from the Battuta API, and the country-specific code is passed into the World Geo Data API, specifying city-specific information. The name of that city is then entered into the Weather by API, which returns recent weather information for that city. The weather information is displayed on the page and the temperature information is displayed via a graph.

Technical breakdown

This is a deeper discussion of the architecture, the technology used in creating the mashup, any issues encountered, and overall, how you implemented the project.

Architecture and Data Flow

The code is split into front-end and back-end, with the front-end using the react framework and the back-end using node to compile.

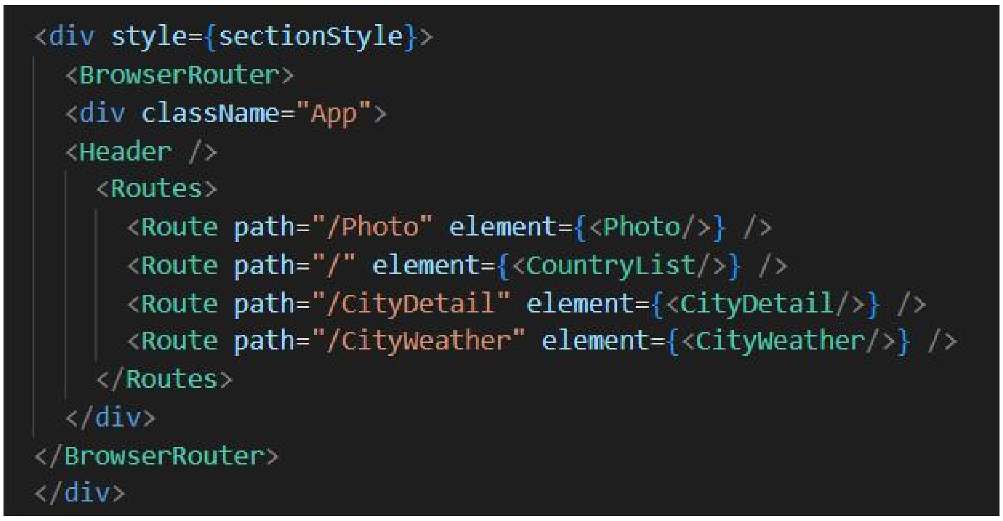


```
server
  node_modules
  routes
    countryAPI.js
    countryInforAPI.js
    countryWeather.js
    flickrAPI.js
  .env
  .gitignore
  credTest.js
  ddb_additem.js
  ddb_getitem.js
  index.js
  package-lock.json
  package.json
```

```
page
  CityAir.css
  CityDetail.js
  CityGrid.css
  CityGrid.js
  CityWeather.js
  CountryList.js
  Header.js
  Home.css
  Photo.js
  Search.js
App.css
App.js
App.test.js
index.css
index.js
logo.svg
serviceWorker.js
setupTests.js
```

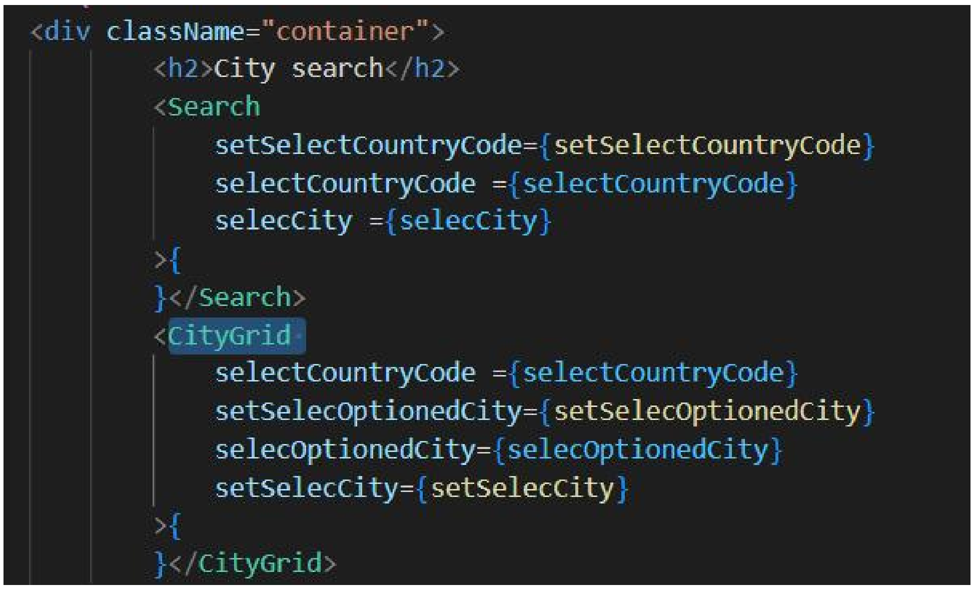
The back end mainly has the route folder which holds four js route files that are used to fetch data from the api and return it as json format. There is also a ddb_additem.js routing file which is used to connect to dynamoDB and fetch and change the page count data. Finally, the index.js file is used to load and configure all the routes.

The front end mainly has the page folder, the app.js file and the index.js file. page holds the react code for each page, which is loaded to app.js via the router, and finally index renders the app.



```
<div style={sectionStyle}>
  <BrowserRouter>
  <div className="App">
  <Header />
    <Routes>
      <Route path="/Photo" element={<Photo/>} />
      <Route path="/" element={<CountryList/>} />
      <Route path="/CityDetail" element={<CityDetail/>} />
      <Route path="/CityWeather" element={<CityWeather/>} />
    </Routes>
  </div>
</BrowserRouter>
</div>
```

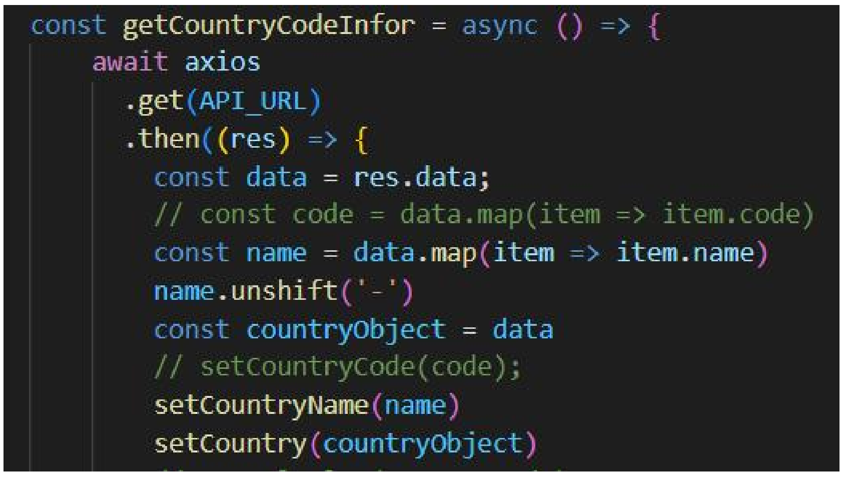
It is worth mentioning that the countryList router also load two components, search and CityGrid



```
<div className="container">
        <h2>City search</h2>
        <Search
            setSelectCountryCode={setSelectCountryCode}
            selectCountryCode ={selectCountryCode}
            selecCity ={selecCity}
        >{
        }</Search>
        <CityGrid
            selectCountryCode ={selectCountryCode}
            setSelecOptionedCity={setSelecOptionedCity}
            selecOptionedCity={selecOptionedCity}
            setSelecCity={setSelecCity}
        >{
        }</CityGrid>
```

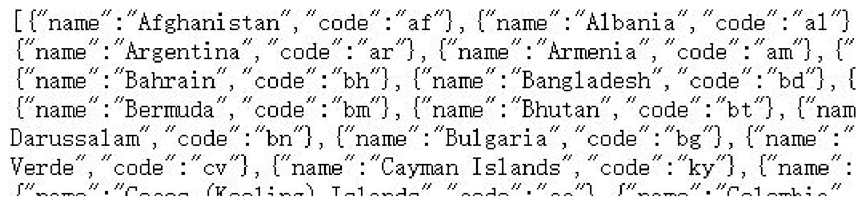
When the user logs in on port localHost:3000, the search router will be rendered by useEffect function in search.js.

The useEffect function accesses the port and extracts countries data from the back-end node file countryAPI.



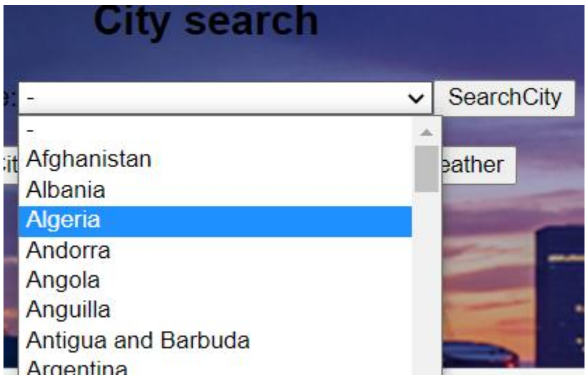
```
const getCountryCodeInfor = async () => {
    await axios
      .get(API_URL)
      .then((res) => {
        const data = res.data;
        // const code = data.map(item => item.code)
        const name = data.map(item => item.name)
        name.unshift('-')
        const countryObject = data
        // setCountryCode(code);
        setCountryName(name)
        setCountry(countryObject)
```

/country port visit http://battuta.medunes.net/api/country/all/ to get all the country names and codes and return them as json.

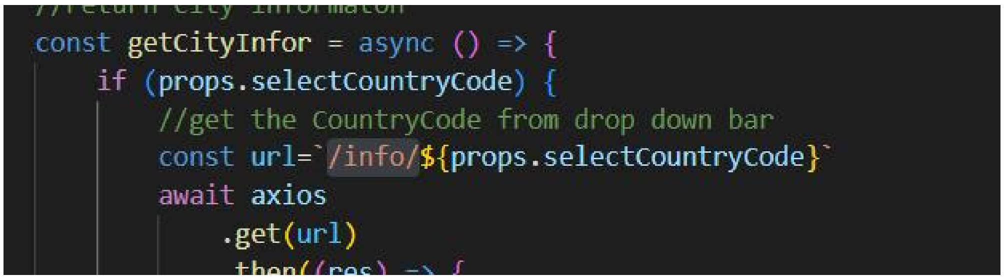


```
[{"name":"Afghanistan","code":"af"},{"name":"Albania","code":"al"}
{"name":"Argentina","code":"ar"},{"name":"Armenia","code":"am"},{"
{"name":"Bahrain","code":"bh"},{"name":"Bangladesh","code":"bd"},{
{"name":"Bermuda","code":"bm"},{"name":"Bhutan","code":"bt"},{"nam
Darussalam","code":"bn"},{"name":"Bulgaria","code":"bg"},{"name":"
Verde","code":"cv"},{"name":"Cayman Islands","code":"ky"},{"name":
```

React displays the name of the country obtained in the drop-down bar.

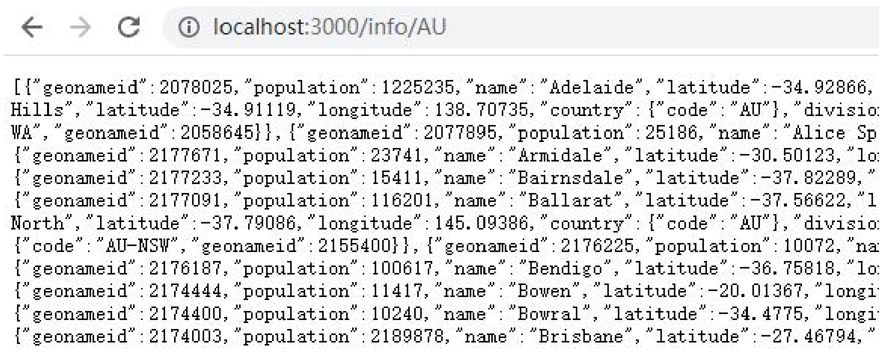


The city selected by the user will be passed into CityGrid.js via prop and useState function, and then the back-end port /info/{region} will be accessed via axios.



```
  const getCityInfor = async () => {
      if (props.selectCountryCode) {
          //get the CountryCode from drop down bar
          const url=`/info/${props.selectCountryCode}`
          await axios
              .get(url)
```

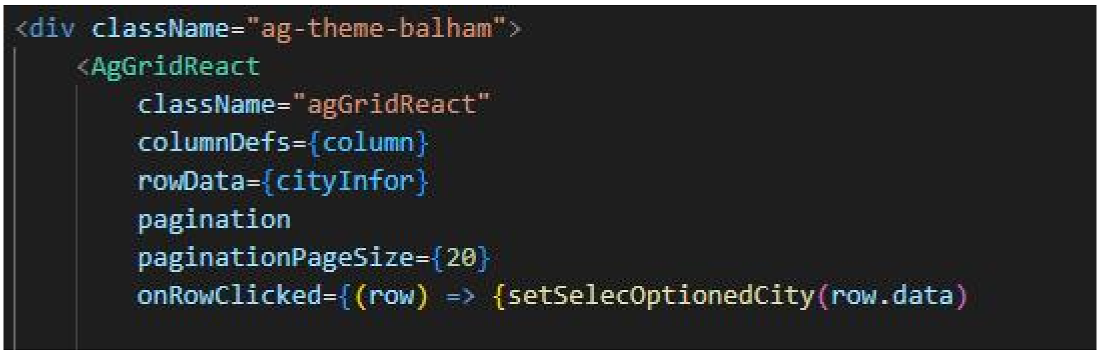
The /info/{region} port of the CountryInfoAPI extracts the region from the route and accesses the https://world-geo-data.p.rapidapi.com/countries/region/cities' port to retrieve data for all cities under a specific country, including geonameid,, population, name, latitude, longitude, etc. and return it as a json format



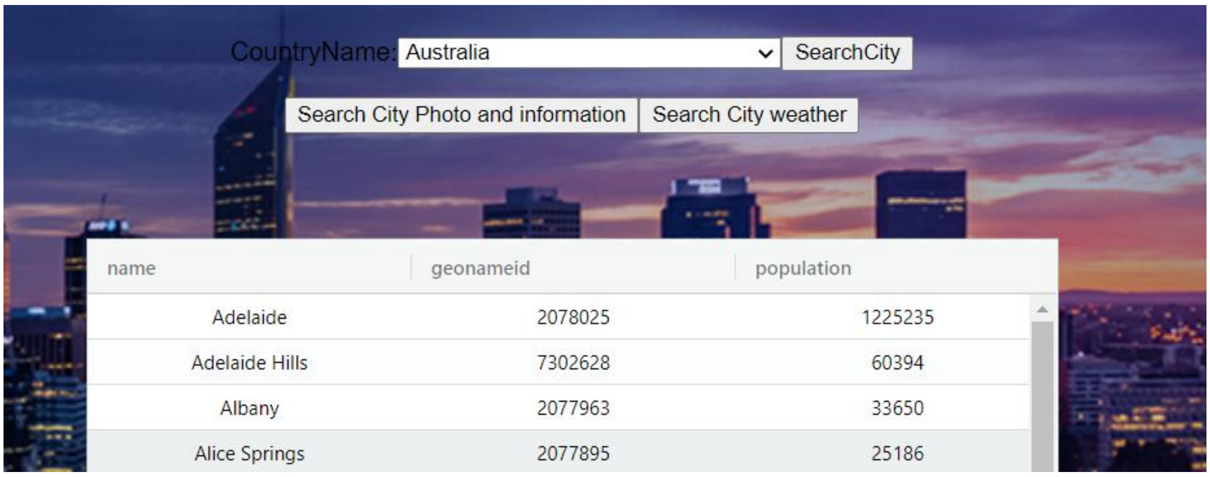
localhost:3000/info/AU

```
[{"geonameid":2078025,"population":1225235,"name":"Adelaide","latitude":-34.92866,
Hills","latitude":-34.91119,"longitude":138.70735,"country":{"code":"AU"},"divisio
WA","geonameid":2058645}},{"geonameid":2077895,"population":25186,"name":"Alice Sp
{"geonameid":2177671,"population":23741,"name":"Armidale","latitude":-30.50123,"lo
{"geonameid":2177233,"population":15411,"name":"Bairnsdale","latitude":-37.82289,"
{"geonameid":2177091,"population":116201,"name":"Ballarat","latitude":-37.56622,"l
North","latitude":-37.79086,"longitude":145.09386,"country":{"code":"AU"},"divisio
{"code":"AU-NSW","geonameid":2155400}},{"geonameid":2176225,"population":10072,"na
{"geonameid":2176187,"population":100617,"name":"Bendigo","latitude":-36.75818,"lo
{"geonameid":2174444,"population":11417,"name":"Bowen","latitude":-20.01367,"longi
{"geonameid":2174400,"population":10240,"name":"Bowral","latitude":-34.4775,"longi
{"geonameid":2174003,"population":2189878,"name":"Brisbane","latitude":-27.46794,"
```

React gets this data and displays it in the grid.

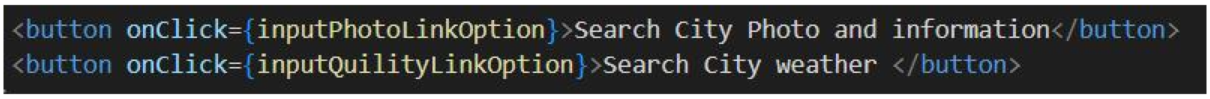


```
<div className="ag-theme-balham">
    <AgGridReact
        className="agGridReact"
        columnDefs={column}
        rowData={cityInfor}
        pagination
        paginationPageSize={20}
        onRowClicked={(row) => {setSelecOptionedCity(row.data)
```

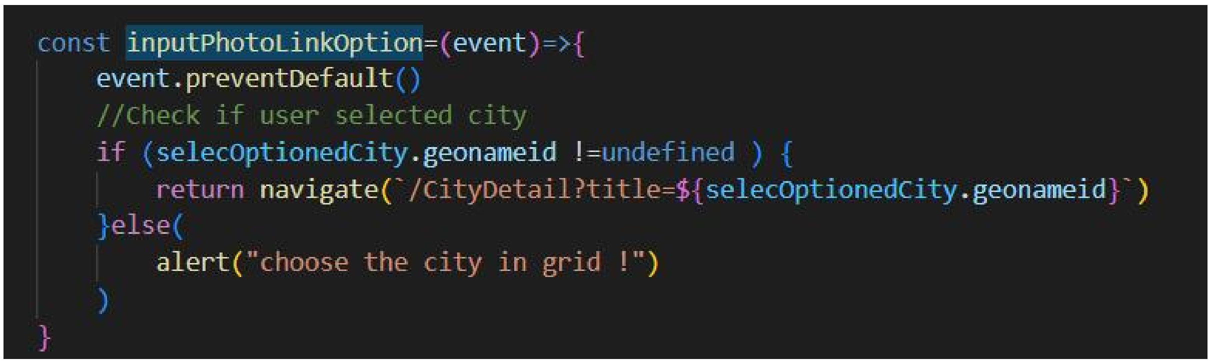


name	geonameid	population
Adelaide	2078025	1225235
Adelaide Hills	7302628	60394
Albany	2077963	33650
Alice Springs	2077895	25186

When the user clicks on a line in the grid, react stores the object data for that city in selecOptionedCity by usestate function. When the user clicks on the search city photo and information button, the page jumps to /CityDetail?title=${selecOptionedCity.geonameid} port. Then the axios middleware in CityDetail.js is called, accessing the /info/city/{ID} port in CountryInfoAPI.js.

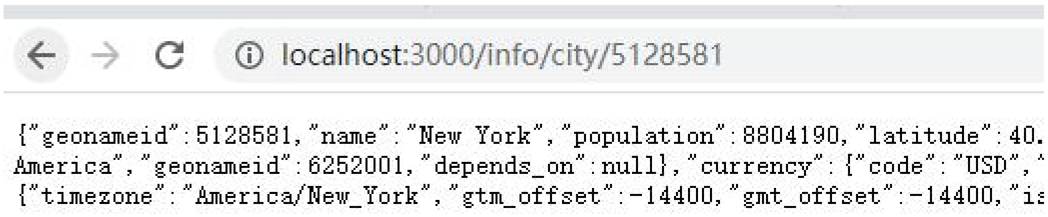


```
<button onClick={inputPhotoLinkOption}>Search City Photo and information</button>
<button onClick={inputQuilityLinkOption}>Search City weather </button>
```



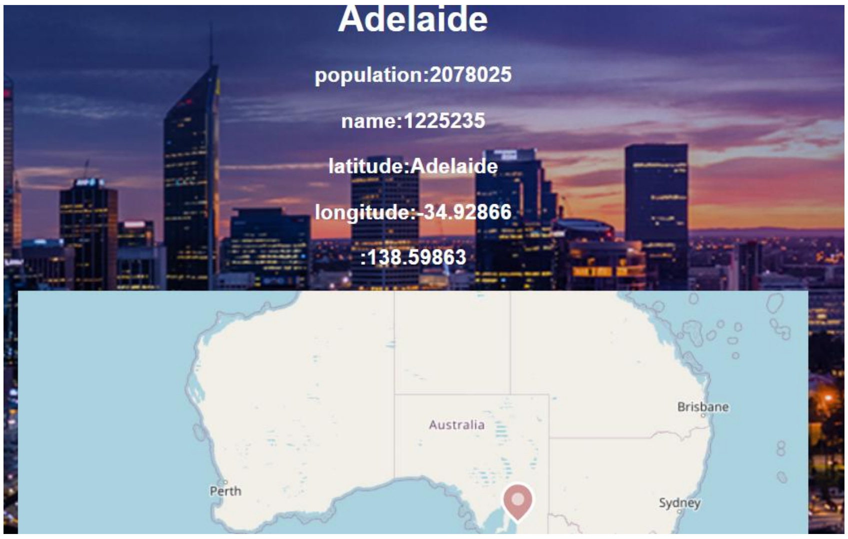
```
const inputPhotoLinkOption=(event)=>{
    event.preventDefault()
    //Check if user selected city
    if (selecOptionedCity.geonameid !=undefined ) {
        return navigate(`/CityDetail?title=${selecOptionedCity.geonameid}`)
    }else(
        alert("choose the city in grid !")
    )
}
```

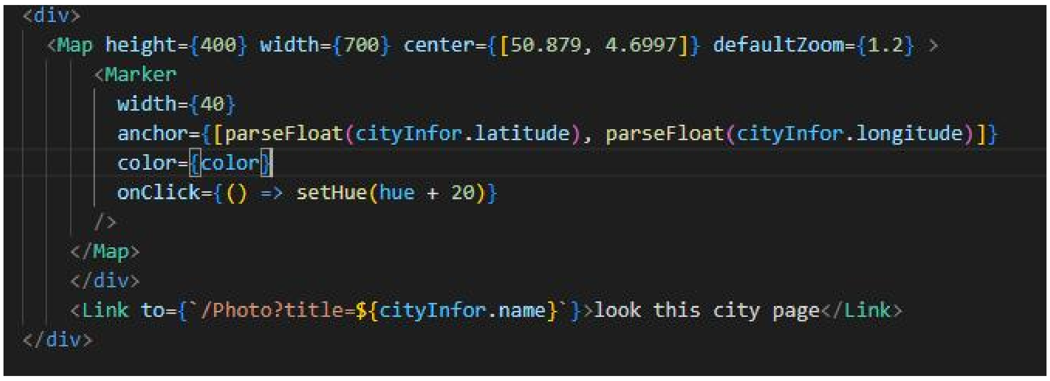
This port gets the city id and passes the city id into the https://world-geo-data.p.rapidapi.com/cities/{ID}` port, which returns the details of the particular city.



```
{"geonameid":5128581,"name":"New York","population":8804190,"latitude":40.
America","geonameid":6252001,"depends_on":null},"currency":{"code":"USD","
{"timezone":"America/New_York","gtm_offset":-14400,"gmt_offset":-14400,"is
```

Get the data in react, store it in the cityInfor of usestate and display the data in a list, then display the latitude and longitude via the map component.

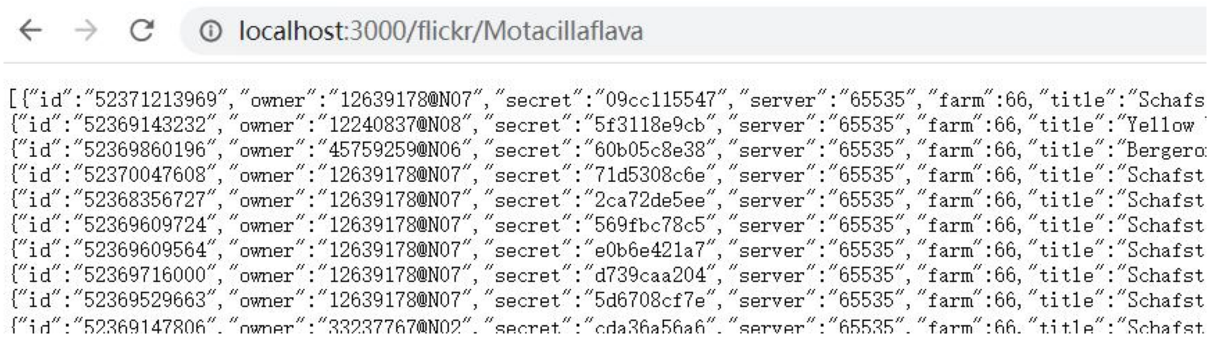




```
<div>
  <Map height={400} width={700} center={[50.879, 4.6997]} defaultZoom={1.2} >
      <Marker
        width={40}
        anchor={[parseFloat(cityInfor.latitude), parseFloat(cityInfor.longitude)]}
        color={color}
        onClick={() => setHue(hue + 20)}
      />
    </Map>
    </div>
    <Link to={`/Photo?title=${cityInfor.name}`}>look this city page</Link>
</div>
```

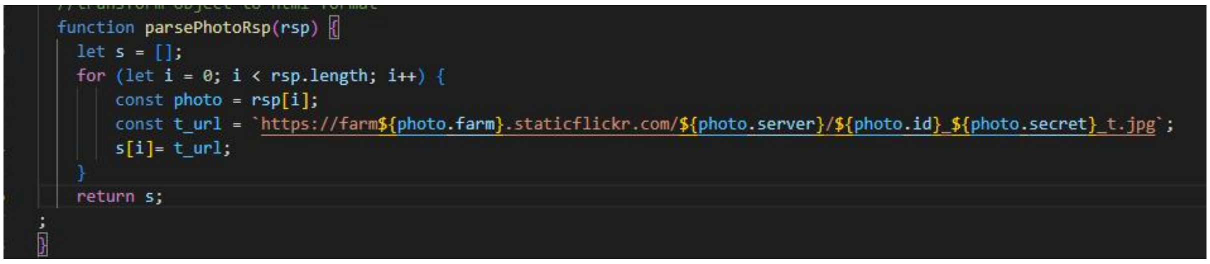
Clicking on the *look at this page* link at the bottom will redirect to the /Photo?title=${cityInfor.name} port of page.js and use the axios middleware to access the /flickr/${query} port of flickrAPI.js on the back end to access the FlickAPI

The FlickAPI gets the parameters of the query, visits https://flickr.photos.search port and returns the photo information related to a specific city, in the form of json.



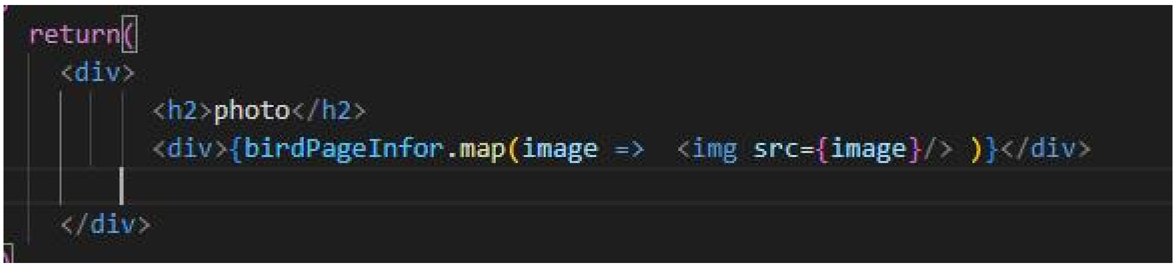
```
localhost:3000/flickr/Motacillaflava

[{"id":"52371213969","owner":"12639178@N07","secret":"09cc115547","server":"65535","farm":66,"title":"Schafs
{"id":"52369143232","owner":"12240837@N08","secret":"5f3118e9cb","server":"65535","farm":66,"title":"Yellow
{"id":"52369860196","owner":"45759259@N06","secret":"60b05c8e38","server":"65535","farm":66,"title":"Bergero
{"id":"52370047608","owner":"12639178@N07","secret":"71d5308c6e","server":"65535","farm":66,"title":"Schafst
{"id":"52368356727","owner":"12639178@N07","secret":"2ca72de5ee","server":"65535","farm":66,"title":"Schafst
{"id":"52369609724","owner":"12639178@N07","secret":"569fbc78c5","server":"65535","farm":66,"title":"Schafst
{"id":"52369609564","owner":"12639178@N07","secret":"e0b6e421a7","server":"65535","farm":66,"title":"Schafst
{"id":"52369716000","owner":"12639178@N07","secret":"d739caa204","server":"65535","farm":66,"title":"Schafst
{"id":"52369529663","owner":"12639178@N07","secret":"5d6708cf7e","server":"65535","farm":66,"title":"Schafst
{"id":"52369147806","owner":"33237767@N02","secret":"cda36a56a6","server":"65535","farm":66,"title":"Schafst
```

In react, the /flickr/${query} port is accessed to get the back-end data, which is then transformed into linked form by the parsePhotoRSP function.



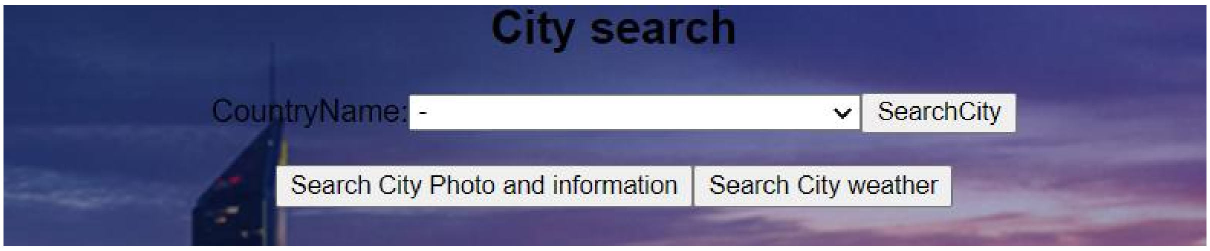
```
function parsePhotoRsp(rsp) {
  let s = [];
  for (let i = 0; i < rsp.length; i++) {
      const photo = rsp[i];
      const t_url = `https://farm${photo.farm}.staticflickr.com/${photo.server}/${photo.id}_${photo.secret}_t.jpg`;
      s[i]= t_url;
  }
  return s;
;
}
```

Finally, the image is transformed into html format and displayed in return.

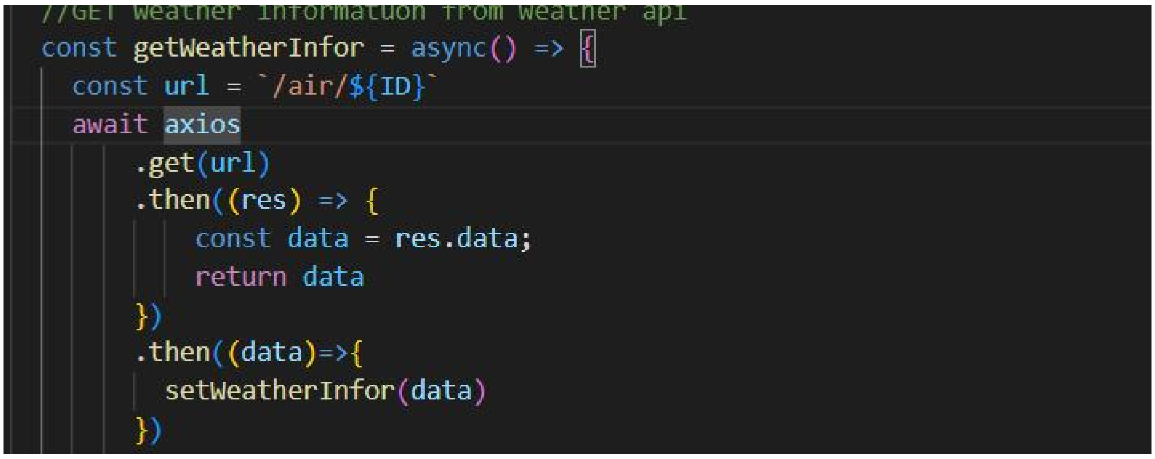


```
return(
  <div>
      <h2>photo</h2>
      <div>{birdPageInfor.map(image =>  <img src={image}/> )}</div>

  </div>
```

If we click on Search City Weather button on the home page, we will be redirected to the /CityWeather?title=${selecOptionedCity.name} page.

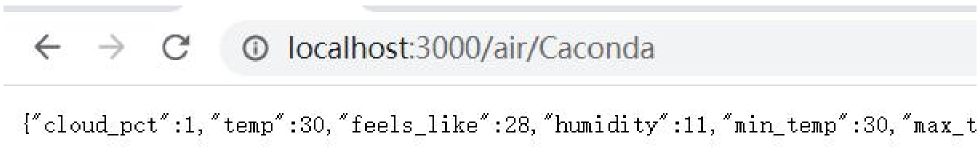


Access port port/air/{ID} with axios to access the CountryWeatherAPI



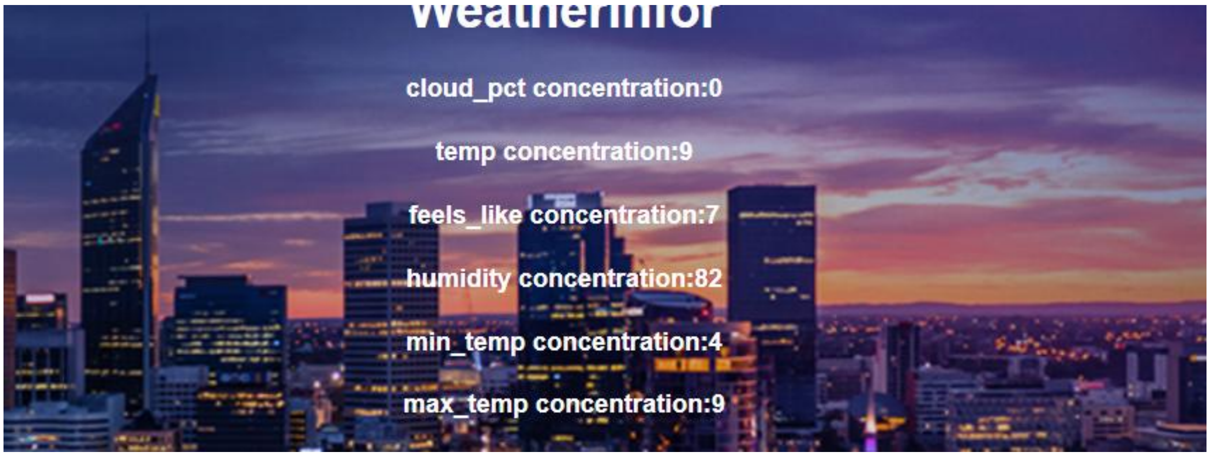
```
//GET weather information from weather api
const getWeatherInfor = async() => {
  const url = `/air/${ID}`
  await axios
      .get(url)
      .then((res) => {
        const data = res.data;
        return data
      })
      .then((data)=>{
        setWeatherInfor(data)
      })
```

The CountryWeatherAPI gets the parameters of the query, puts the query parameters into params, passes them to axios, and then visits https://weather-by-api-ninjas.p.rapidapi.com/v1/weather port to get the weather data for a specific city, returning it as json and return it as json.



localhost:3000/air/Caconda

```
{"cloud_pct":1,"temp":30,"feels_like":28,"humidity":11,"min_temp":30,"max_t
```



WeatherInfor

cloud_pct concentration:0

temp concentration:9

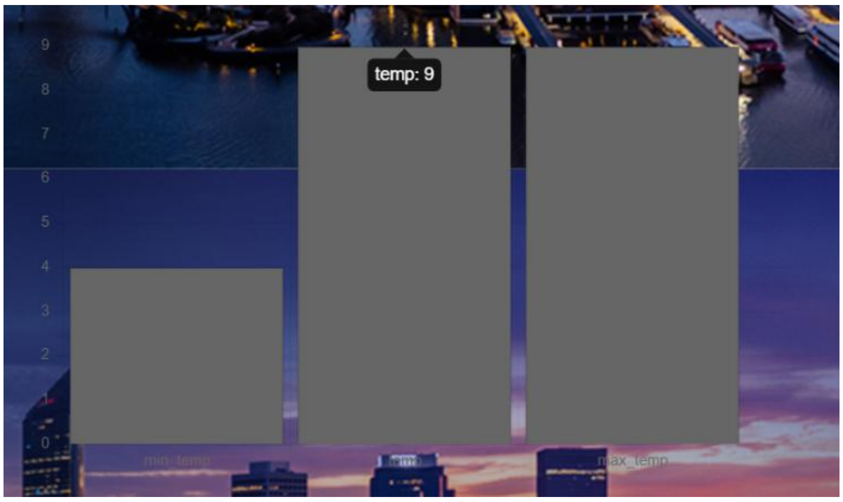
feels_like concentration:7

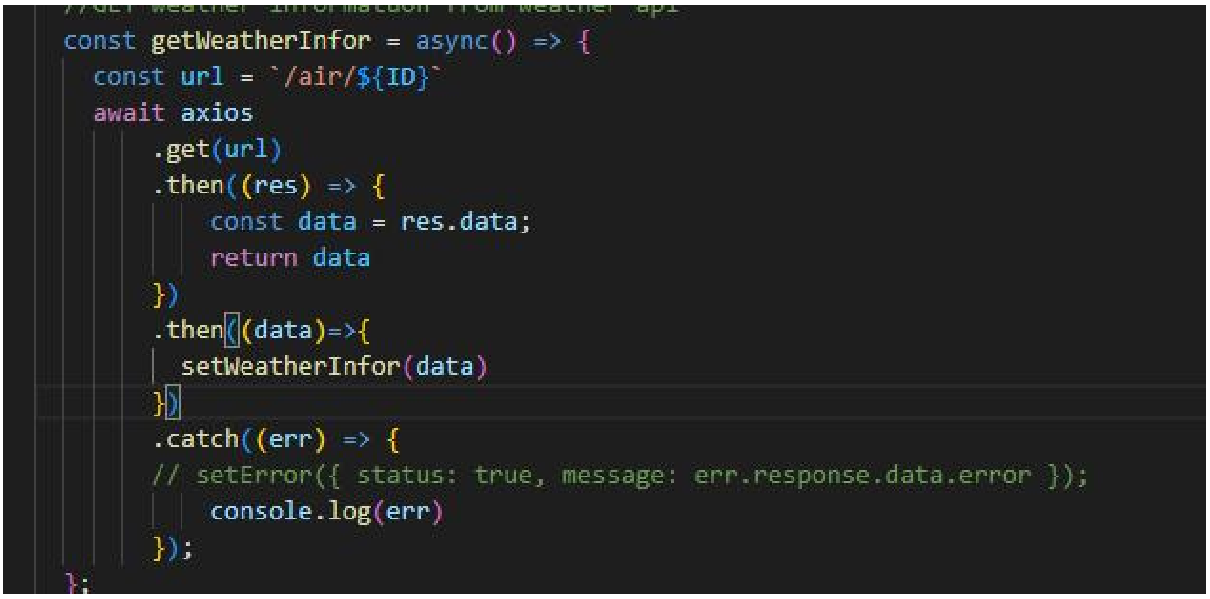
humidity concentration:82

min_temp concentration:4

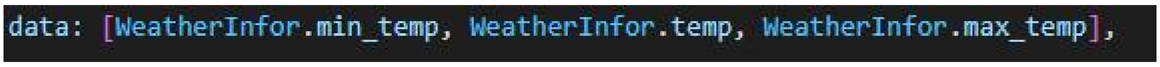
max_temp concentration:9

Get the data in react and display it in the list and pass the min_temp, temp, max_temp values of the data to the Barchart component to display it in the chart

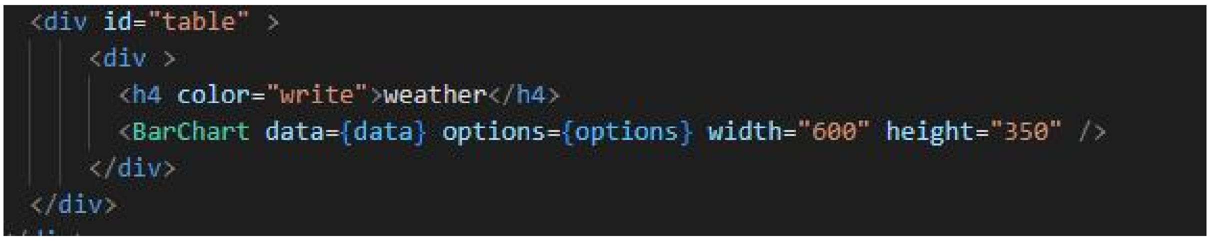




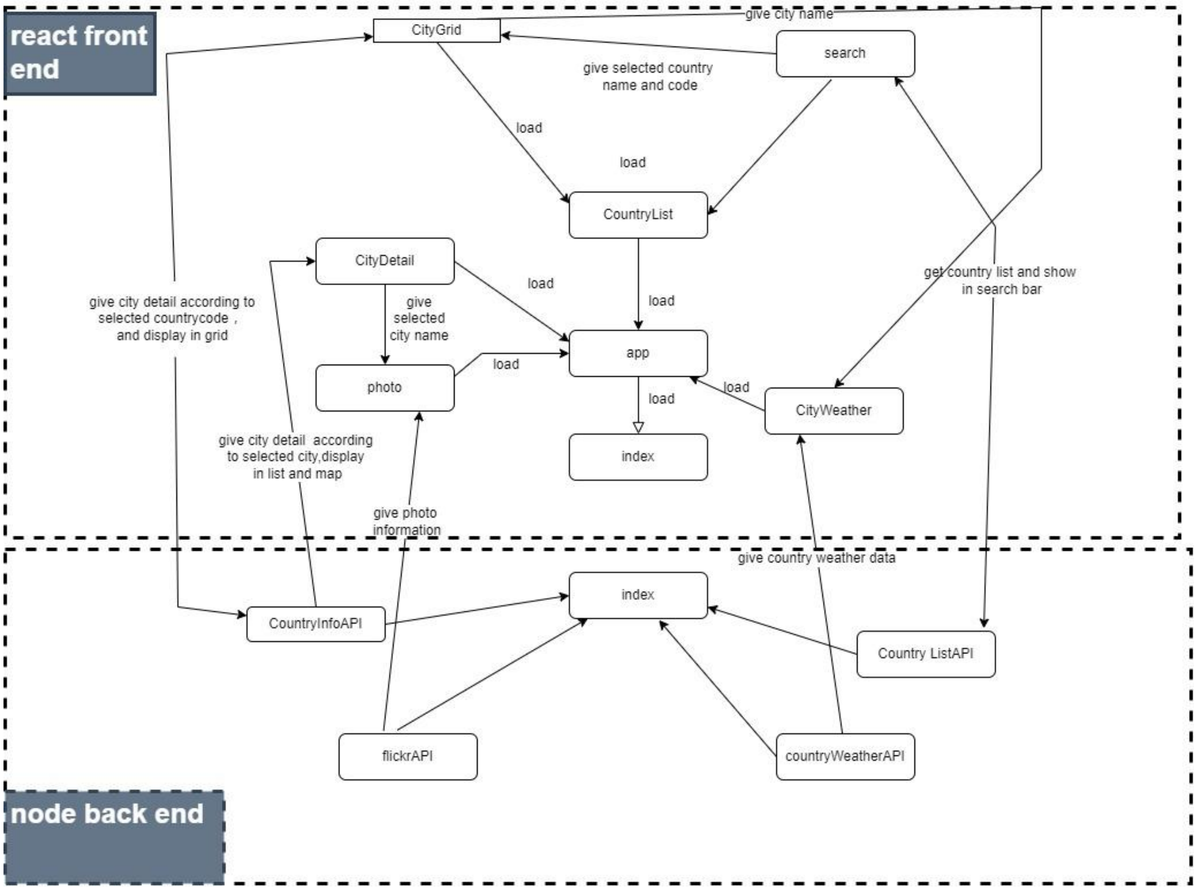
```
//get weather information from weather api
const getWeatherInfor = async() => {
  const url = `/air/${ID}`
  await axios
      .get(url)
      .then((res) => {
          const data = res.data;
          return data
      })
      .then((data)=>{
        setWeatherInfor(data)
      })
      .catch((err) => {
      // setError({ status: true, message: err.response.data.error });
          console.log(err)
      });
};
```



```
data: [WeatherInfor.min_temp, WeatherInfor.temp, WeatherInfor.max_temp],
```



```
  <div id="table" >
      <div >
        <h4 color="write">weather</h4>
        <BarChart data={data} options={options} width="600" height="350" />
      </div>
  </div>
```



react front end
CityGrid
give city name
search
give selected country name and code
load
load
CountryList
CityDetail
load
give selected city name
load
give city detail according to selected countrycode , and display in grid
get country list and show in search bar
app
photo
load
load
load
CityWeather
index
give city detail according to selected city,display in list and map
give photo information
give country weather data
index
CountryInfoAPI
Country ListAPI
flickrAPI
countryWeatherAPI
node back end

Deployment and the Use of Docker

Screenshot of Dockerfile is in appendix

The first step is to complete the dockerfile file and put in the environment variables and API key.

The second step, docker build -t, turns the final program into an docker image and deploys it to the docker of my device.

Step 3: Using docker tag and docker push, pass the image to dockerhub

Step 4: Use docker pull in AWS to load the image into the dockerhub

Step 5: Type docker run in aws VM terminal, put the environment variables and key value on the command line and run the program.

Test plan

Task	Exceted Outcome	Result
Home page shows all content	Type node index in terminal and the page pops up as normal	Pass
Show country information	The drop-down bar shows all country names	Pass
Search all cities	Select the country and click on the search button, the city information is displayed in the grid	Pass
The country selected in the drop-down field is not empty	Select "-" in the drop down bar, a warning pops up	Pass
Grid can choose row	Click on one of the grid rows to darken that row	Pass
Once row have been selected , click on search city photo information to be redirected to another page.	Click on one of the grid rows and click on search city photo information to jump to a new page	Pass
Once row have been selected ,click on the search city weather button to be redirected to another page.	Click on one of the grid rows and click on the search city weather button to jump to a new page	Pass
Clicking on search city photo information without selecting grid does not jump to the page	Without clicking on grid, click on search city photo information and a warning pops up	Pass
Page count can display data	Page count followed by a number	Pass
Page count number changed after page refresh	After refreshing the page, the Page count number is added by one	Pass

Show city detail	Click on search city photo information and a new page with data is displayed	Pass
Show map	Click on search city photo information and a new page with a map and mark is displayed	Pass
Show images related to a specific city	Click on show page link to jump to the new link and display the image	Pass
Show weather information for a specific city	Click on the search city weather button to jump to a new page showing the weather information	Pass
Charts showing temperatures in specific cities	New page with graphical representation	Pass

Difficulties / Exclusions / unresolved & persistent errors /

The first one was that the api was not working and the program was unable to read data from the interface and reported an error. This was later solved by replacing the api with a new one and modifying the back-end code.

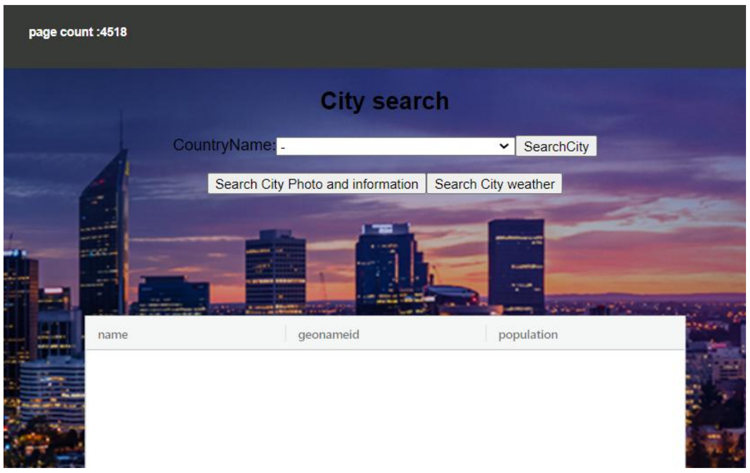
The second is a port problem. After the docker configuration, the front-end middle-ware axios kept reporting errors when connecting to all api, but later found that the reason was that the url was written as localHost:/xx/ when fetching the access port. The problem was solved after changing fetch to axios and localHost:/xx/ to /xx/.

Extensions (Optional)

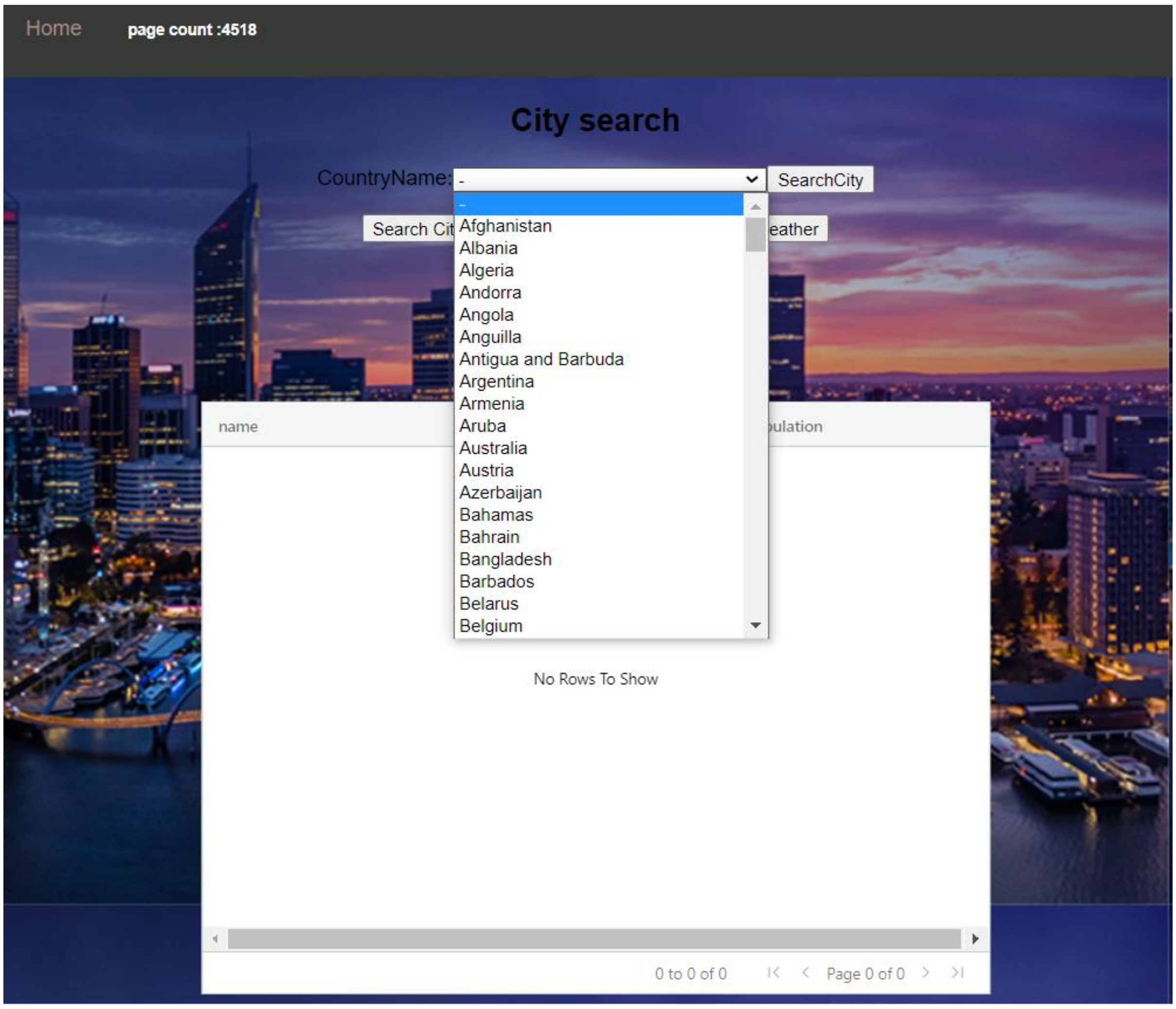
In this section, you can tell us if you wish to how you might extend your app and make it better. that you didn't have time to tell us about.

User guide

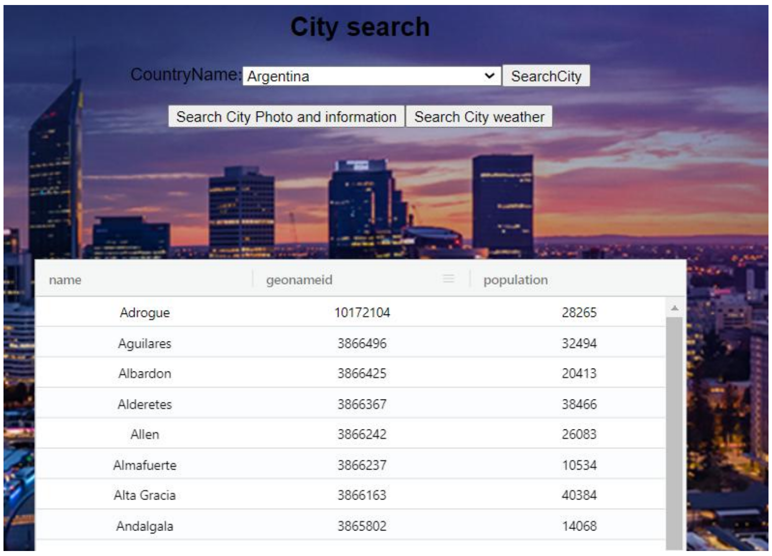
Visit localHost:3000 and go to the main screen. The page count in the top left corner shows the number of accesses



Click on the drop down list to show all available country names

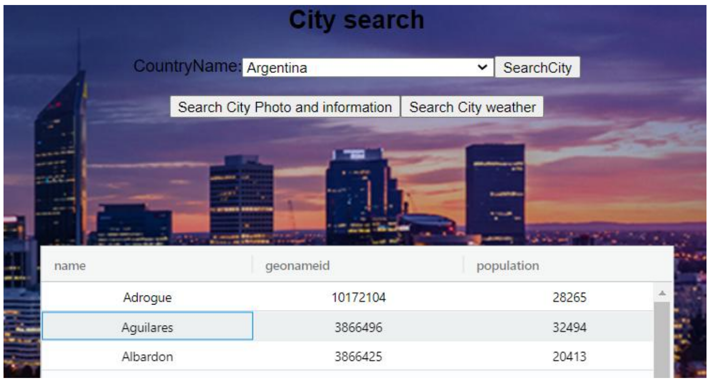


Select country and click on the search city button, all cities in the selected country will be displayed on the grid.



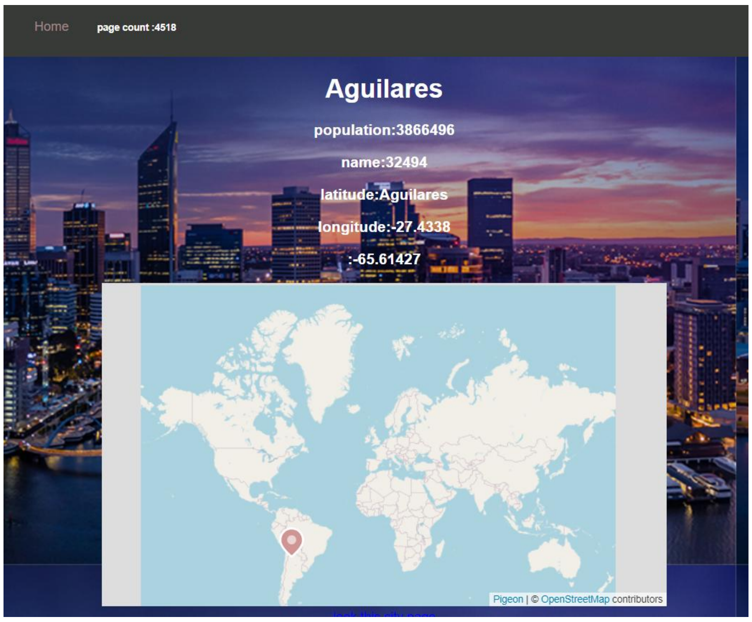
name	geonameid	population
Adrogue	10172104	28265
Aguilares	3866496	32494
Albardon	3866425	20413
Alderetes	3866367	38466
Allen	3866242	26083
Almafuerte	3866237	10534
Alta Gracia	3866163	40384
Andalgala	3865802	14068

Click on any row of the grid and click on the search city photo and information button.



name	geonameid	population
Adrogue	10172104	28265
Aguilares	3866496	32494
Albardon	3866425	20413

Go to the city information page.



Aguilares

population:3866496

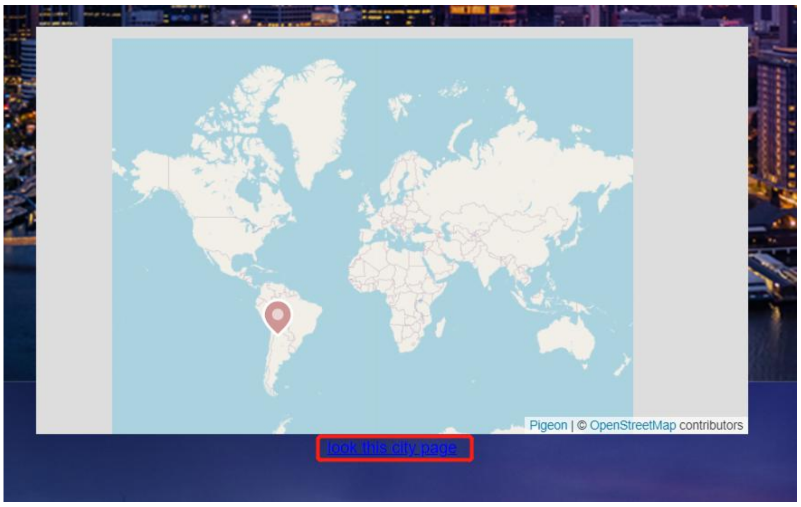
name:32494

latitude:Aguilares

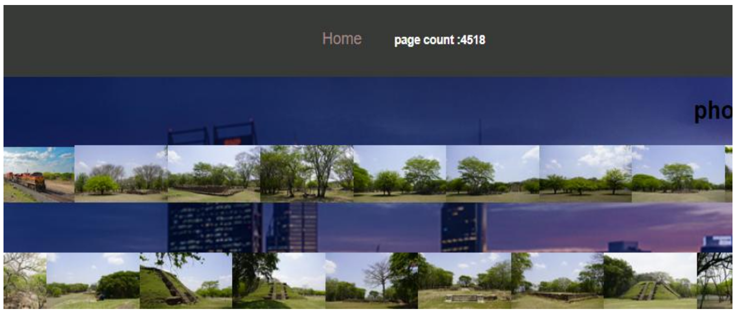
longitude:-27.4338

:-65.61427

Click on the link at the bottom

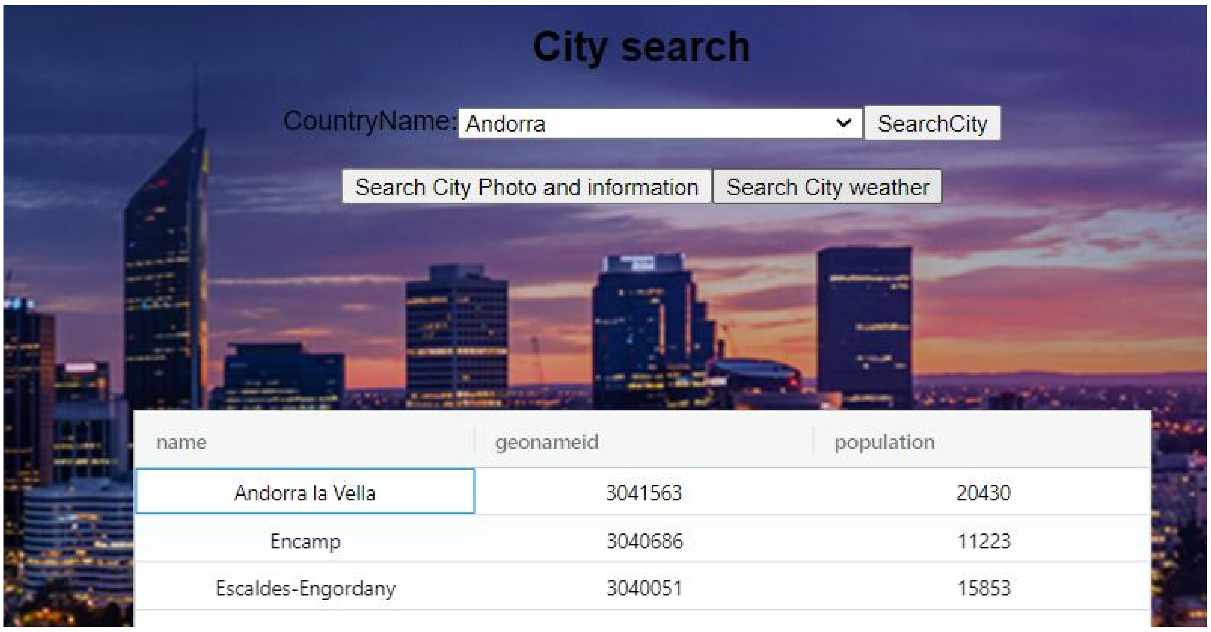


Jump to a new page showing images related to that city.



Click on home to return to the main screen

Click on the search city weather button.


City search

name	geonameid	population
Andorra la Vella	3041563	20430
Encamp	3040686	11223
Escaldes-Engordany	3040051	15853

Jump to the city weather page, which displays weather information and temperature charts for that city.

Analysis

In this section we ask you look at your application and to analyse it in response to a couple of prompts we supply below. The marking is based on the quality of There are two questions. In each case, there is an overall question, and then a series of bullet points that help you respond. A good answer can be no more than a couple of sentences in response to each of them. Say it quickly and get on with the next one. This exercise is comparatively straightforward but there is a corresponding task in This exercise is comparatively straightforward but there is a corresponding task in Assignment 2 which will be far more difficult.

Question 1: Vendor Lock-in

1.Weather by API-Ninjas API that returns Weather data for a specified city for scenarios such as Weather forecasts. Changing the weather API is not difficult. It provides very limited functionality, and only returns a small amount of weather data for a particular city, which is fairly common. If the weather API is replaced, developers can replace it with a more powerful weather api, as long as the new API returns the same or more weather-related data, but it comes at a price. It competes with the Visual Crossing Weather API, which provides specific city Weather information, including some that can be visualized, but is a paid API. absence of the Weather by API-Ninjas API is too single-minded and has returned too little data.

The Buttuta API returns code and name for all countries. It is primarily applicable to the context of querying certain information for a particular country. It's very easy to replace, and any API that returns country and country code can replace it.If the BUTTUTA API is out of service, could choose any API alternative that returns all the country data by simply extracting the name and code data in the back-end code using the map function. Code data. Its competitors are the Countries Cities API and others.the CountryInfo API is a paid API that returns the country list. The absence of the BUTTUTA API is due to the API's single functionality and the limited number of times it can be accessed per month.

The World Geo Data API is very powerful and can return detailed information about a particular country, city, or administrative area. This applies primarily to locale-related scenarios. It is difficult to replace because it returns very detailed data and can return city and country information separately. To replace the API, a developer would need to find two or three locale-specific api to fill in for the service it originally provided, which can lead to extremely complex back-end code.Its competitors are the Countries Cities API , but it returns more data than the Countries Cities API. The absence of the World Geo Data API makes it a paid API with a daily limit on access.

The Flickr API returns images related to query parameters. It's very versatile. It's hard to replace because there are fewer api with similar capabilities. If we want to replace the API, we need to find an API that does the same thing for image search, but so far I haven't found one that does the same thing.

2.we can transform the persistence service from Dynamo DB to SQL Server by first using SQL Server to access the AWS DynamoDB API, using a library called BOTO3 in SQL Server, then writing a Python script that accesses DynamoDB, and then writing a wrapper, this will execute our Python script and output the results as a normal result set, eventually putting the results into an SQL table.

*Several problems were encountered. The first was that we can not use external scripts in SQL Server, so we need to configure the environment parameter,'*exec sp_configure 'external scripts enabled' *. The second is a problem with using the INSERT wrapper, so we need to program the INSERT wrapper.(Derek Colley,2018)*

Question 2: Container Deployment

In your response, you should consider the following prompts:

Q1

- Disadvantages of container-based deployment:

  1. Increasing complexity: as the number of containers and applications increases, so does the complexity. Managing so many containers in a production environment is a challenging task.

  2. Native Linux support: Most Container Technologies, such as Docker, are based on Linux containers, which are a little more cumbersome to run in a Microsoft environment than in a native Linux environment, and daily use brings complexity.
- I don't deploy applications directly on virtual machines

Because a lot of effort goes into the deployment environment, we don't need to set up a new one to use Docker. Just download the Docker image and run it on a different server.

Second of all. Deploying programs directly on a virtual machine can cause a number of problems due to inconsistencies between the development and test environments, using container technology keeps the development and test environments and production environments version and dependency unified, ensuring that code executes in a highly unified environment.

(2 marks)

Q2

persistence options:
The Bolb ， s3

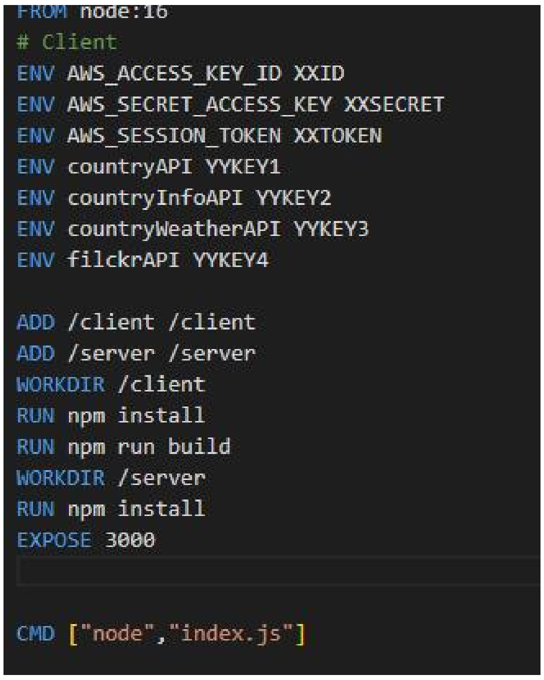
(2 marks)

References

Colley , derek . (2018, July 30). Importing Data from AWS DynamoDB into SQL Server 2017. Mssqltips. https://www.mssqltips.com/sqlservertip/5621/importing-data-from-aws-dynamodb-into-sql-server-2017/

Appendices

Dockerfile



```
FROM node:16
# Client
ENV AWS_ACCESS_KEY_ID XXID
ENV AWS_SECRET_ACCESS_KEY XXSECRET
ENV AWS_SESSION_TOKEN XXTOKEN
ENV countryAPI YYKEY1
ENV countryInfoAPI YYKEY2
ENV countryWeatherAPI YYKEY3
ENV filckrAPI YYKEY4

ADD /client /client
ADD /server /server
WORKDIR /client
RUN npm install
RUN npm run build
WORKDIR /server
RUN npm install
EXPOSE 3000


CMD ["node","index.js"]
```