

CAB230 Assignment 1, 2024

1. Introduction

This assignment requires you to develop a React-based web application to allow users to view and analyse data about volcanoes which we have exposed via a REST API. The assignment builds upon the lecture and practical material covered in the first half of CAB230. We will talk about the REST API in detail below.

The main aims of this assignment are to:

- Allow you to build a sophisticated client web application using modern approaches
- Provide experience in querying REST APIs and presenting the results on a web page
- Provide experience with modern web technologies such as React

2. The Dataset

The volcano dataset we are using is based on publicly available data collated by the Smithsonian Institution's Global Volcanism Program. The data describes volcanoes located around the world. There are 1,343 entries of the following form:

Column	Example data	Additional notes
ID	1	
Name	Abu	
Country	Japan	A volcano is associated with a country if it is within the boundaries of a country, on a shared boundary or area, in a remote territory, or within a maritime Exclusive Economic Zone
Region	Japan, Taiwan, Marianas	
Subregion	Honshu	
Last_Eruption	6850 BCE	
Summit	641	The summit of the volcano in meters (m)
Elevation	2103	The elevation of the volcano in feet (ft)
Latitude	34.5000	
Longitude	131.6000	
Population_5km	3597	The number of people living within a X km radius of the volcano
Population_10km	9594	
Population_30km	117805	
Population_100km	4071152	

It is important to appreciate that you will not work with this database directly when completing this assignment. Instead, all interactions should take place via our published REST API. You will be able to fetch the volcano data using HTTP GET operations. You will also need to handle a registration and login process from your React application to the API through appropriate HTTP POST requests. We will now consider the provided REST API in more detail.

3. The REST API

Accessing the documentation

The REST API is published and documented at: <http://4.237.58.241:3000/> The documentation on the index page was created using Swagger (<https://swagger.io/>). We will teach you more about Swagger in the coming weeks. For this assignment, you only need to read and interact with the documentation that we have created for you.

The Swagger Docs are your primary source for anything to do with the API during this assignment. They will be maintained more regularly than this specification, and they have executable examples. If there is any conflict in the information we provide, or if something is unclear, the Swagger Docs win.

The structure of the endpoints

The API endpoints are split into two types: data and authentication.

Data endpoints

HTTP Request	Route
GET	/countries
GET	/volcanoes
GET	/volcano/{id}

Authentication endpoints

HTTP Request	Route
POST	/user/register
POST	/user/login

All the data endpoints are publicly accessible. However, the `/volcano/{id}` endpoint provides additional information about the number of people living near the volcano if the user is authenticated. The best way to think about this is to assume some difference between “free” content and content which is only available to registered members of your website that have logged in.

Users wanting to access the members-only content must first interact with the authentication endpoints. New users may register to become members (`/user/register`) and

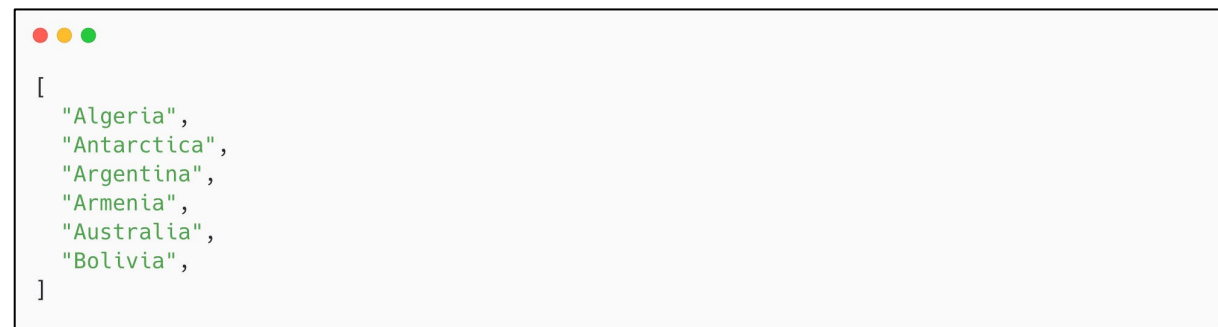
they can then login to the system (relying on */user/login*) to gain access to the authenticated information (accessible via */volcano/{id}*).

We will now consider each of the endpoints in turn.

Data endpoints

All data endpoints can be accessed via HTTP GET requests. You can discover the error conditions and precise usage in the Swagger docs. Here we are primarily concerned with the types of data that are available.

/countries

A screenshot of a terminal window with a light gray background and a dark border. At the top left, there are three colored circles (red, yellow, green) representing window control buttons. The terminal displays a JSON array of country names in a green monospace font. The array starts with an opening square bracket, followed by a list of six country names in quotes, each followed by a comma, and ends with a closing square bracket.

```
[  
  "Algeria",  
  "Antarctica",  
  "Argentina",  
  "Armenia",  
  "Australia",  
  "Bolivia",  
]
```

The countries endpoint returns a list of countries in the API that are associated with one or more volcanoes. The countries are listed in alphabetical order. In the screenshot above we have edited the results to show a manageable list of six countries. The real request yields 76 countries in total. This endpoint is primarily provided to help you query the */volcanoes* endpoint as it requires a country name as a query parameter.

/volcanoes

```
[
  {
    "id": 1,
    "name": "Abu",
    "country": "Japan",
    "region": "Japan, Taiwan, Marianas",
    "subregion": "Honshu"
  },
  {
    "id": 16,
    "name": "Aogashima",
    "country": "Japan",
    "region": "Japan, Taiwan, Marianas",
    "subregion": "Izu, Volcano, and Mariana Islands"
  },
]
```

The volcanoes endpoint has a mandatory query parameter: *country*. The endpoint returns a list of volcanoes that are associated with the queried country. In the above screenshot we have queried */volcanoes?country=Japan*. Once again, we have edited the results to show a manageable list of two records. In practice, this query returns 122 entries.

The volcanoes endpoint also has an optional query parameter: *populatedWithin*. This optional query parameter can be used to filter the list of volcanoes for a given country to include only those that have at least one person living within the provided radius. The only permitted values for this query parameter are: 5km, 10km, 30km or 100km. An example query using *populatedWithin* would be: */volcanoes?country=Japan&populatedWithin=10km*

/volcano/{id}

The final data endpoint, */volcano/{id}*, has a mandatory ID path parameter. This volcano ID can be obtained from the */volcanoes* endpoint. The */volcano/{id}* endpoint returns information about the queried volcano. In the screenshots below we have queried for the volcano with the ID of 1 (*/volcano/1*).

The data returned from this endpoint varies depending on whether the user is authenticated or not. Without authentication, the returned data for a queried volcano is:

```
{
  "name": "Abu",
  "country": "Japan",
  "region": "Japan, Taiwan, Marianas",
  "subregion": "Honshu",
  "last_eruption": "6850 BCE",
  "summit": 641,
  "elevation": 2103,
  "latitude": "34.5000",
  "longitude": "131.6000"
}
```

Once authenticated, the population density around the volcano (for a 5km, 10km, 30km and 100km radius) is also returned:

```
{
  "name": "Abu",
  "country": "Japan",
  "region": "Japan, Taiwan, Marianas",
  "subregion": "Honshu",
  "last_eruption": "6850 BCE",
  "summit": 641,
  "elevation": 2103,
  "latitude": "34.5000",
  "longitude": "131.6000",
  "population_5km": 3597,
  "population_10km": 9594,
  "population_30km": 117805,
  "population_100km": 4071152
}
```

Authentication endpoints

In Week 7 of CAB230 we will discuss authentication and JSON Web Tokens (JWTs). We will also provide a practical worksheet on client-side authentication that we anticipate will be of great assistance in completing this part of the assignment. If you are reading this specification prior to Week 7, then it is expected that most of this section won't make a lot of sense right now.

The authentication endpoints (*/user/register* and */user/login*) both need to be interacted with via HTTP POST requests. The body of these requests should include an email and password.

If a user is successfully registered, the returned response will be:

```
{
  "message": "User created"
}
```

If a user successfully logs in, the returned response will be:

```
{
  "token": "ajsonwebtoken",
  "token_type": "Bearer",
  "expires_in": 86400
}
```

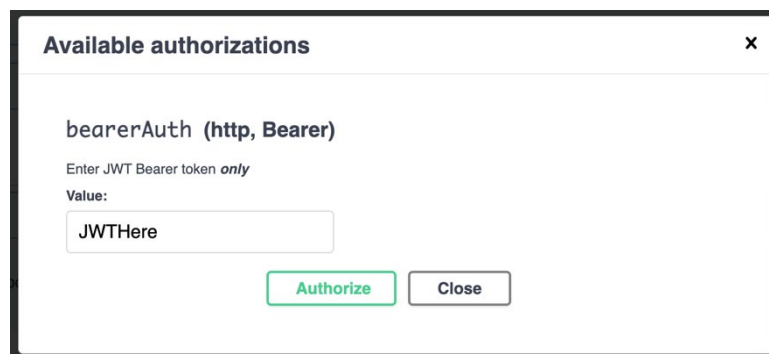
The key piece of information in the successful logged in response is the token. Your application needs to store this token and use it to prove that the user is authorised when accessing the authenticated route. Note that we have provided a fake token in the screenshot above. The real one will be much longer and far more random.

As mentioned earlier, we will show you how to use the authenticated routes with React in Week 7. However, it is also possible to interact with these routes via the Swagger docs. We will cover this process now.

First, register a new user. Then login with the same credentials. The response from the login route will be very similar to the screenshot above, but it should now include an actual token. At the top right of the Swagger page, you should see a green Authorize button:



Click on this button and you should now see the Authorisation dialogue:



Copy the token from your login response and paste it where I have “JWTHere” in the image. Click on the green Authorize button and close the dialogue. You should now see that the Authorize button lock is shut:



You are now able to use the authenticated route in Swagger.

For brevity we haven't included the possible error responses for the authenticated routes in this specification. However, they are all detailed in the Swagger docs.

4. Application Breakdown

This assignment requires you to make a lot of choices. These decisions are more obvious after you have created a few apps, but they can be hard if you are doing this for the first time. At the most basic level, you must develop a React application that allows the user to interact with all the available endpoints, but without the user ever being aware of the underlying calls to the API.

In this section we will give you some guidance on what we want to see. You should think of this assignment as a series of increasingly challenging steps, with each step allowing you the opportunity to achieve a higher grade level. You should think very carefully about how users will input things into your app, the components that you will use to display the API data, and how users will navigate between pages.

There is some latitude in all of this, and we have purposely left the rest of this section somewhat open ended. The screenshots that we show below are of an example application with almost no styling or consideration to design. We hope that it will serve as a very useful starting point, but it is not intended to be a model solution and nor do we wish for every student's assignment to look and function identically – your personal take is warmly encouraged.

If at any point you are wondering whether you are on the right track, consider whether your application utilises all the available endpoints and works with a range of components. If it does, you are likely heading in the right direction. Talk to us in the practicals if you are unsure. Towards the end of this specification, we will discuss the grade levels for this assignment in greater detail.

Landing Page

What will your users see when they first launch your app? Often there will be a hero image and a welcome message. The navigation choices available to the user should be clear. What can they access in your app and how? Your layout should be chosen to facilitate the information that is available at the endpoints. The screenshot below was thrown together quickly. As a design, it isn't amazing, but it is a useful starting point for working out where things should go.

[Home](#) [Volcano List](#) [Register](#) [Login](#)

Volcanoes of the World



Volcano List Page

The Volcano List page utilises data available from the */countries* and */volcanoes* endpoints. The purpose of this page is to allow the user to search for volcanoes in a specified country, and optionally filter the list of volcanoes to those that have people living within a specified radius. Well-suited components for this page are a form and table.

[Home](#) [Volcano List](#) [Register](#) [Login](#)

Country: Populated within:

Name	Region	Subregion
Abu	Japan, Taiwan, Marianas	Honshu
Aogashima	Japan, Taiwan, Marianas	Izu, Volcano, and Maria...
Adatarayama	Japan, Taiwan, Marianas	Honshu
Asamayama	Japan, Taiwan, Marianas	Honshu
Aira	Japan, Taiwan, Marianas	Ryukyu Islands and Kyu...
Akagisan	Japan, Taiwan, Marianas	Honshu
Asosan	Japan, Taiwan, Marianas	Ryukyu Islands and Kyu...
Akan	Japan, Taiwan, Marianas	Hokkaido
Ata	Japan, Taiwan, Marianas	Ryukyu Islands and Kyu...
Akita-Komagatake	Japan, Taiwan, Marianas	Honshu
Akita-Yakeyama	Japan, Taiwan, Marianas	Honshu

1 to 11 of 106 < < Page 1 of 10 > >

You will need to decide how the user will input the country name to search for. Can you provide a dropdown with the countries listed? Are there too many options to show in a dropdown? Will you just support free text input? If so, how will you handle an invalid country? Can you support auto complete suggestions for this field? Some of these ideas are easy, others are more challenging. Try to get something working in the first instance and experiment from there. Similar considerations apply when dealing with the populated within distance.

We will look favourably on assignments that use a sophisticated table component such as Ag Grid and implement pagination (some countries such as Japan have 122 recorded volcanoes), as well as client-side filtering and sorting.

You will also need to decide how the user will select an individual volcano. In our example we are utilising AG Grid's row click handler to navigate to an individual volcano page.


Individual Volcano Page

The individual volcano page utilises data from the `/volcano/{id}` endpoint. Of note on this page is the use of a map component to display the volcano's location and display a marker at its provided latitude and longitude.

[Home](#) [Volcano List](#) [Register](#) [Login](#)

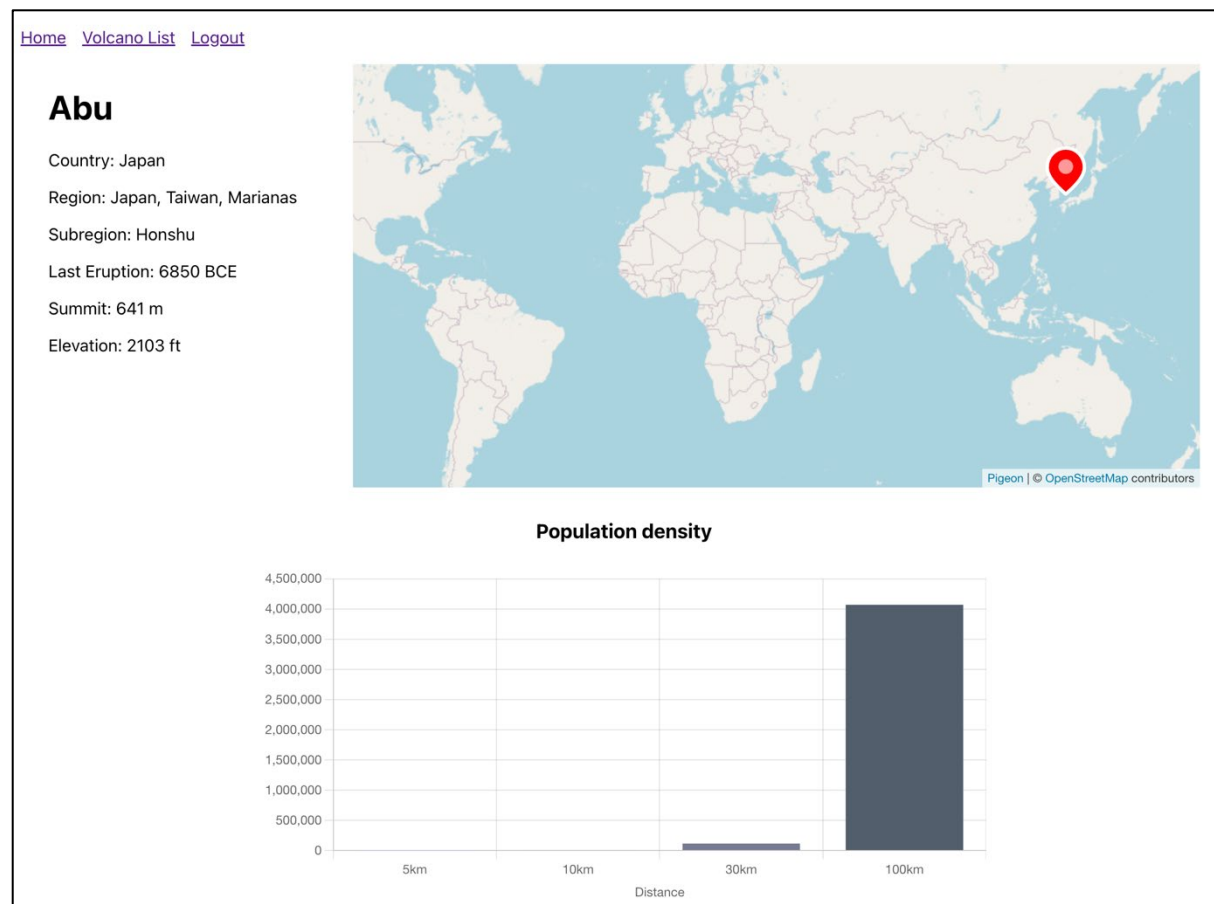
Abu

Country: Japan
Region: Japan, Taiwan, Marianas
Subregion: Honshu
Last Eruption: 6850 BCE
Summit: 641 m
Elevation: 2103 ft

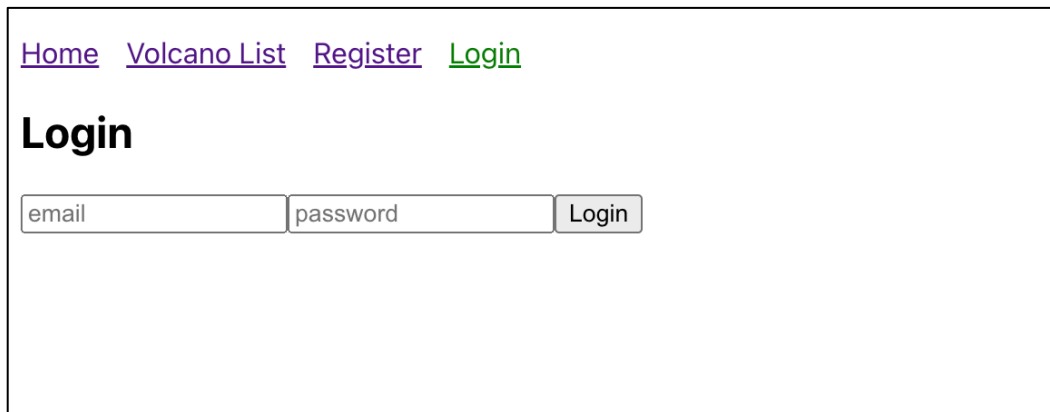


Pigeon | © OpenStreetMap contributors

If the user is authenticated, then the population density data will also be available to them. This data can be visualised in a bar chart, or similar, component. As the user is logged in, notice in the navigation bar that the user should now only see an option to log out.



Login and Register pages



[Home](#) [Volcano List](#) [Register](#) [Login](#)

Login

The login and register pages should utilise the two POST authentication endpoints: */user/login* and */user/register*. You can combine the login and register pages into one page should you wish. You will need to build a form component to allow the user to enter their email and password. You should also consider and appropriately handle any error cases - e.g., email is already in use, password is missing etc. See the Swagger docs for the full list of possible error responses.

5. Some Guidance on Website Design

CSS

We anticipate most students will use a component library that comes with some included styling, such as Reactstrap. Utilising a component library will allow you to design a clean and modern website without having to write much CSS. If you have prior experience with CSS, then you may wish to write most of the styling yourself – this is an acceptable approach too.

Routing

You should think about how you are going to handle each route in your client-side application. Have a look at the React Router examples in the lectures and worksheets. You may use a basic HTML menu as a last resort, but the use of React Router will attract much more credit.

Choice of Inputs/Widgets

Some of the endpoints need parameters and you will need to choose how to specify them. Have a look at the example screenshots above and their associated commentary. Are you going to use drop downs? Text boxes? Radio buttons? What makes sense for the choices you are making? Are you making consistent choices across the app? Draw some screens on paper or in a drawing tool before you start any programming.

Table Components

As discussed in the Application Breakdown section, the table component plays a crucial role in displaying the data from the server. Using a sophisticated table component like AG Grid means that you can offer rich functionality on the client-side (e.g., filtering, sorting, and pagination). Pay particular attention to the practical worksheet which walks you through this material.

We will not look favourably on apps which attempt to immediately load all 1,343 entries from the server when it is opened and from thereon handle everything client-side. You should strike a balance between client-side interactions and proper use of the API endpoints.

Mapping

We recommend using Pigeon Maps (<https://pigeon-maps.js.org/docs/>) to show a map of the volcano's location. Pigeon Maps is open source, easy to use, and designed specifically to work with React. There are, of course, other mapping alternatives such as Google Maps and Leaflet. We will not impose a hard rule on which mapping provider you must use. However, to put it bluntly:

QUT will not be held responsible and will not reimburse you for any fees that you incur should you sign up for a paid mapping service.

Pigeon Maps works very well for our requirements and is completely free.

Charting

To display the population density data, a bar chart or some other form of visualisation will be very well received. You may use standard chart libraries to produce these. We recommend the use of chartJS (<https://www.chartjs.org/>), especially via the widely used React wrapper which you can find here (<https://www.npmjs.com/package/react-chartjs-2>). d3 (<https://d3js.org/>) is also a popular choice, but it does have a steep learning curve and we therefore don't recommend using it unless you have prior experience.

6. The Report

We will expect a short report and user guide, generally running to 10 pages or so, including a lot of screenshots. Mostly this is just to help us better understand your application, but we will also require you to assess critically the quality of your UI design, and to analyse it from the perspective of accessibility, and the changes that would be needed for the application to be compliant. We will give more guidance on this in the report guide and template which sits alongside this specification on Blackboard. However, to make one thing clear, you can be as brutal and self-critical as you like in these sections without noticeably affecting your actual UI marks. In fact, this is very much encouraged.

The report will also help us get your thoughts on the process, the bits that worked and the bits that didn't, and the usability and correctness of the application. The report will include:

1. Introduction – telling us what was implemented and what wasn't, showing a few screenshots to illustrate the functionality.
2. Use of Endpoints. How you used the API endpoints and their relationship to the functionality described.
3. A list of any external components used, such as AG Grid.
4. Application Design and Usability. This section is a discussion of the choices you made in designing the app, and some frank assessments of its usability and the changes that would be needed to comply with accessibility requirements. This section attracts a lot of the marks allocated for the report and is noted separately in the CRA.
5. Technical Description of the Application. The architecture, test plan, results and technical limitations and compromises. We do not expect automated testing. We do expect you to verify that the application works for each of the use cases. Screen shots are your friend. This section is especially important if something doesn't actually work.
6. References
7. Appendix: brief user guide

The report should be saved as a PDF and submitted as report.pdf (see submission details below).

7. Grading

The marking for this assignment will be governed by the CRA rubric, and this will cover several aspects of the assignment process. These include:

1. The overall level of functionality successfully implemented
2. The usability of the application
3. The robustness of the application
4. Evidence of a professional approach to design
5. Evidence of a professional approach to development and code quality
6. The quality of the professional report – including your assessment of your UI design and analysis of changes needed for accessibility (see report section above).

Full details of this split will be found in the separate CRA document accompanying this specification on Blackboard. Here we are concerned only with the marks for functionality, application usability and robustness, and UI design. The precise marks awarded may be reduced because of features which are only partially implemented or error-prone or components which are poorly chosen and so on. As a guide, these are our expectations at each grade level:

- **[Grade of 4 level]:** A simple React app with limited styling which uses the data endpoints and presents the data cleanly. Some basic navigation between pages is required. User endpoints and the authenticated data query may not have been

implemented successfully. An attempt should be made at implementing a table component, but there may be limited use of pagination, and the user may be unable to filter or sort the data on the client side. A react-strap table component or other simple alternative would suffice here.

- **[Grade of 5 level]:** At this level we would expect successful implementation of the user registration and login endpoints, and the authenticated data endpoint. Table components should utilise the standard functionality provided by a component such as AG Grid, and there should not be excessive querying of the server. The design of the website should be clean and generally uncluttered, although some parts of the website may be clumsy or confusing to use. An attempt may have been made at implementing a map or chart component, but shortcomings are likely to be present.
- **[Grade of 6 and 7 level]:** Here the expectation is that you have exceeded the grade of 5 level in that all the basics are there and working smoothly. Navigation is handled using React Router, controlled forms are used for inputs and there is evidence of close alignment between your chosen components and the data they are displaying. The overall design of the website should be well thought out and it should be easy to use. We would also expect at this level for you to have used a map component to display the location of the volcano and a chart (or similar visualisation) to show the population density data. The split between the grade of 6 and grade of 7 will involve a trade-off between the features and the quality of the execution.

A reminder that the mark levels are based on *successful* implementation of the features mentioned. If they don't work or are substandard, then of course the grade level may be reduced.

8. Submission

See the Canvas assignment page for submission details.

9. Acknowledgements & Further Reading

The volcanoes dataset is collated by the Smithsonian Institution's Global Volcanism Program and is publicly available at: <https://volcano.si.edu/>. The relevant citation is:

Global Volcanism Program, 2013. Volcanoes of the World, v. 4.10.5 (27 Jan 2022). Venzke, E (ed.). Smithsonian Institution.

We include this here as an acknowledgement of the original source, and to provide you with the opportunity to view the dataset should you wish to. You may notice that the data we are exposing via our API is a reduced and slightly modified dataset of the original. We made these changes to better align the data to this assignment and its learning objectives. Your React application should only retrieve data from our provided API.

Please now take the time to read the assignment CRA and report template released alongside this specification