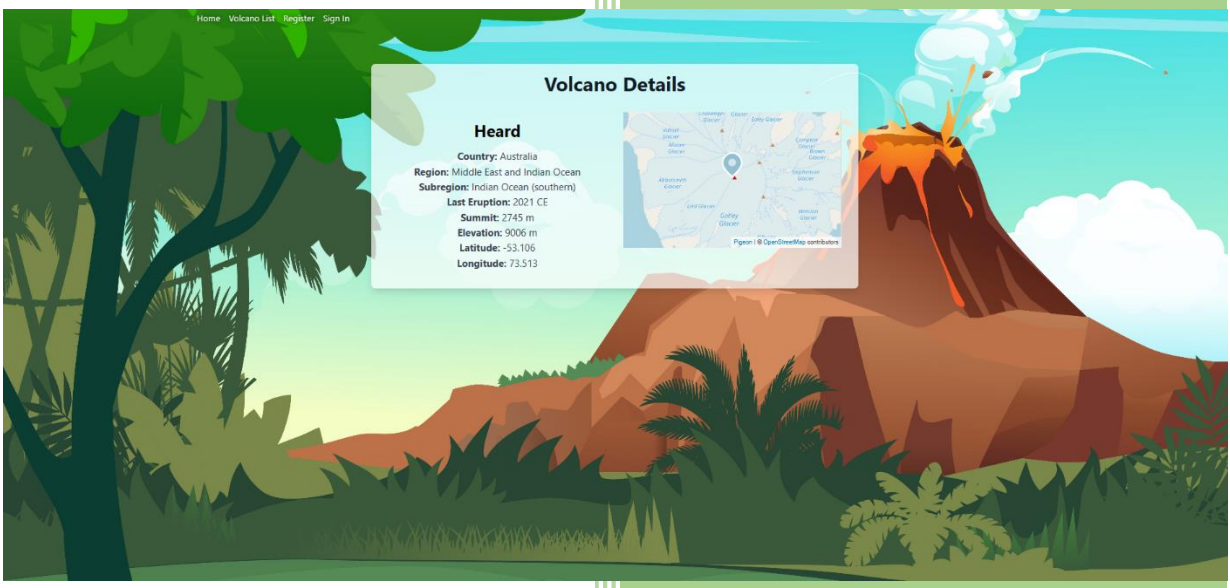
2024

CAB230 Assignment 2 Client Side



CAB230

Volcano API – Client Side Application

<Duc Minh Le>

<n11479116>

10/5/2024

Contents

Introduction

Purpose & description

The main objective of this project is to enable users to explore volcanoes interactively using the dataset from the API specified in this task through a REACT based user interface, for querying via a REST API. The projects interface allows users to view and examine in depth information about each volcano, such as its location, historical eruptions and the impact, on populations.

The main features are the advanced data filtering and sorting capabilities through AG Grid and the use of Pigeon Maps for visual mapping which represents the location of each volcano. For authenticated users, the application provides more data-appealing visualizations like the population density around the volcanoes that are designed as a bar chart using ChartJS. Also, the incorporation of the registration process in the login form helps to make the user interaction easy. Moreover, jwt-decode is used to manage tokens expiration using JWT for authentication. This is therefore, very secure and efficient. These features enrich the user experience by giving the user access to detailed volcanic data, that is, a combination of comprehensive functionality and user-friendly design.

Completeness and Limitations

The app is characterized by rich functionalities, such as using AG Grid to represent data advancedly, the React Router to give an easy and accessible navigation, and Pigeon Maps ChartJS to visualize your data in a dynamic manner. User registration and secure authentication are effectively managed using JWT. However, a limitation is noted in session management; specifically, user sessions are not automatically terminated upon a manual page refresh, and a second refresh is required to log out. This minor issue is recognized as a bug that is planned to be addressed to further enhance security and user experience.

Use of End Points

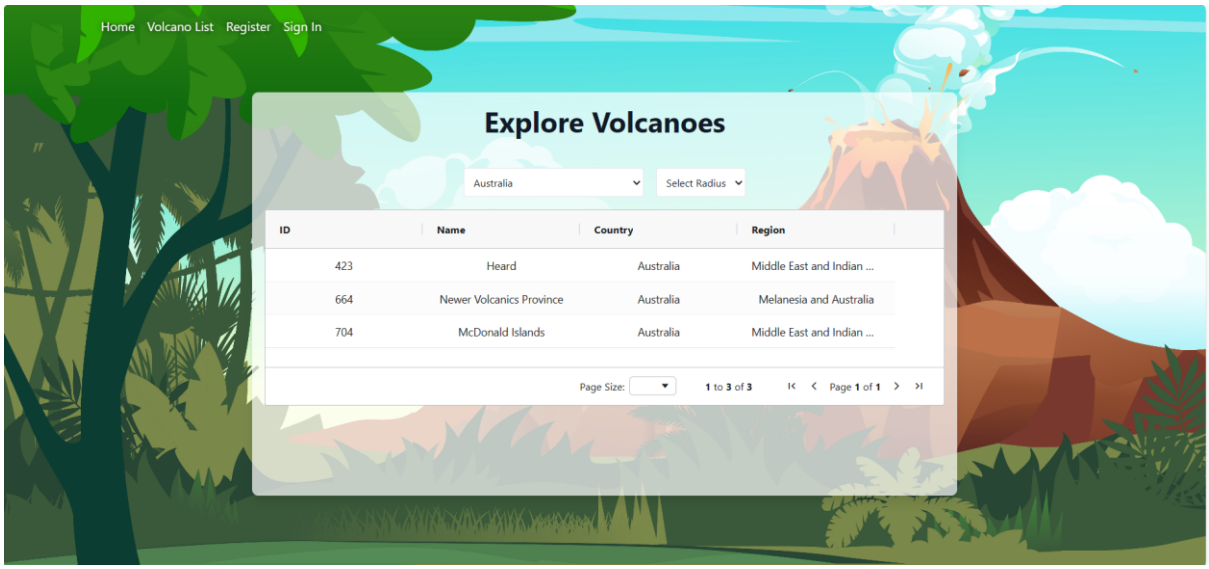
/countries

The default country is Australia.

```
export const fetchCountries = async () => {
    try {
        const response = await API('countries', 'GET');
        return response.data;
    } catch (error) {
        throw new Error('Failed to fetch countries');
    }
};
```

```
useEffect(() => {
    const fetchData = async () => {
        try {
            const countriesData = await fetchCountries();
            dispatch({ type: 'FETCH_COUNTRIES_SUCCESS', payload: countriesData });
            if (countriesData.length > 0) {
                setSelectedCountry(countriesData[4]);
            }
        } catch (error) {
            console.error('Error fetching countries:', error);
        }
    };

    fetchData();
}, [dispatch]);
```

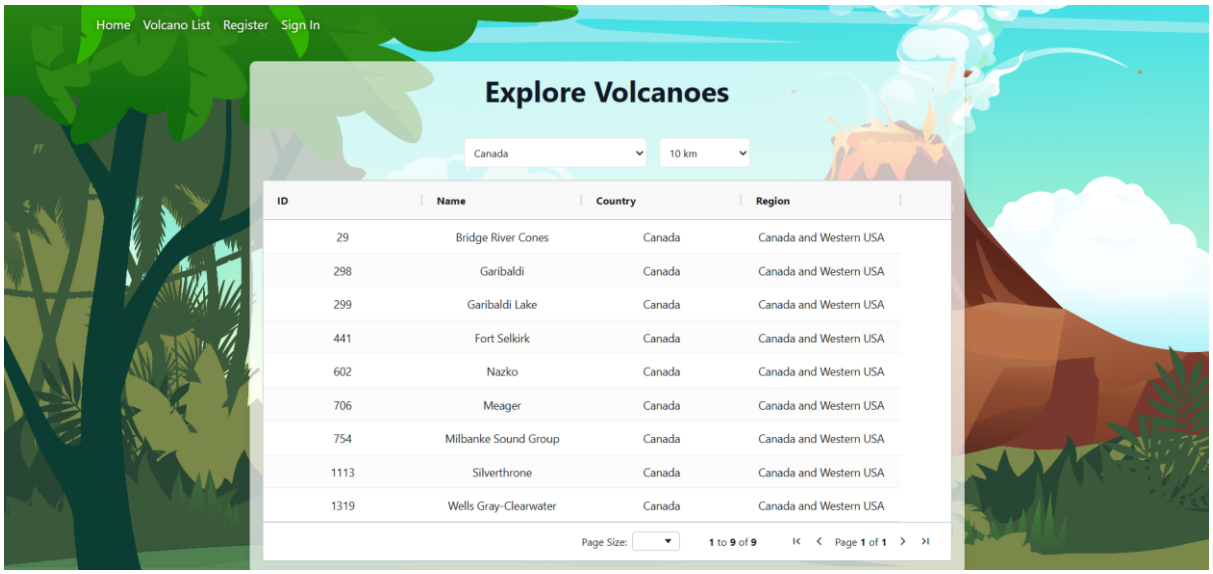


/volcanoes

```
export const fetchVolcanoes = (country, radius = "") => {
    return async (dispatch) => {
        dispatch({ type: FETCH_VOLCANOES_REQUEST });
        let endPoint = `volcanoes?country=${country}`;
        if (radius) {
            endPoint += `&populatedWithin=${radius}`;
        }

        try {
            const response = await API(endPoint, 'GET');
            const data = response.data;
            dispatch({ type: FETCH_VOLCANOES_SUCCESS, payload: data });
        } catch (error) {
            dispatch({ type: FETCH_VOLCANOES_FAILURE, payload: error.message || 'Failed to fetch volcanoes' });
        }
    };
};
```

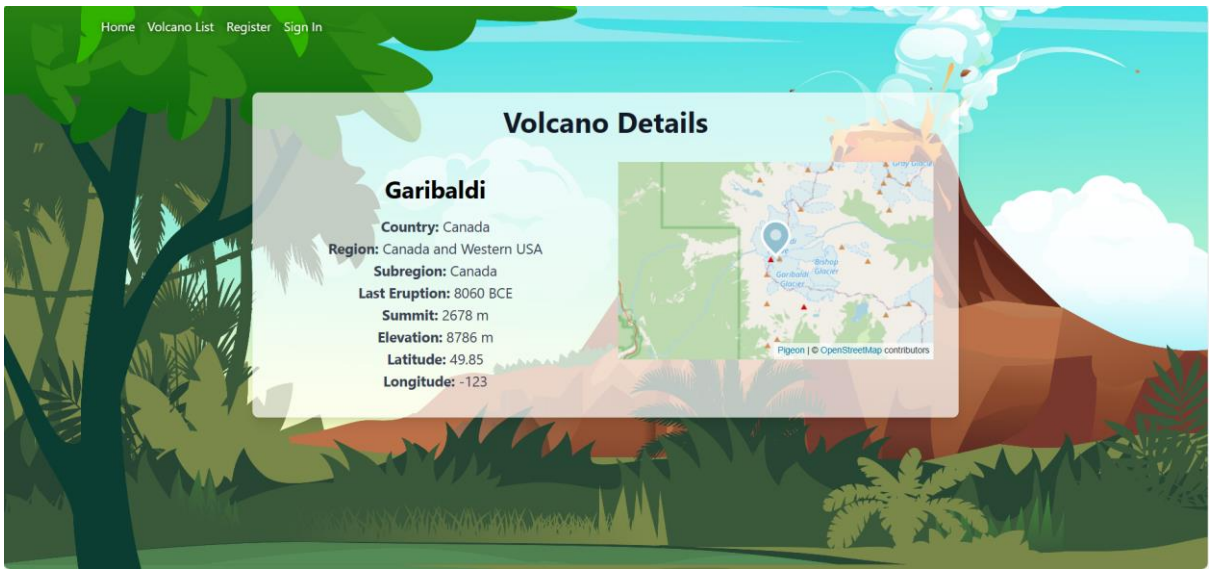
```
    useEffect(() => {
        if (selectedCountry) {
            dispatch(fetchVolcanoes(selectedCountry, selectedRadius));
        }
    }, [dispatch, selectedCountry, selectedRadius]);
```



/volcano/{id}

```
export const fetchVolcanoDetail = (id) => async (dispatch) => {
    try {
        const response = await API(`volcano/${id}`, 'GET');
        const data = response.data;
        dispatch({
            type: 'FETCH_VOLCANO_DETAIL_SUCCESS',
            payload: data
        });
    } catch (error) {
        dispatch({
            type: 'FETCH_VOLCANO_DETAIL_ERROR',
            payload: error.message
        });
    }
};
```

```
useEffect(() => {
    dispatch(fetchVolcanoDetail(id));
}, [dispatch, id]);
```

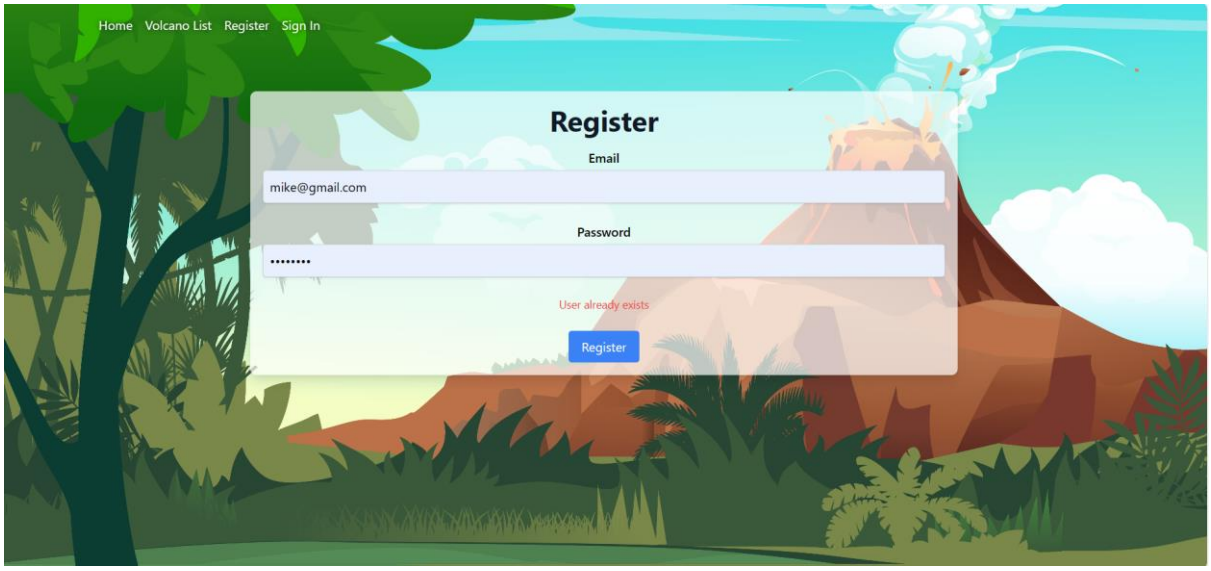


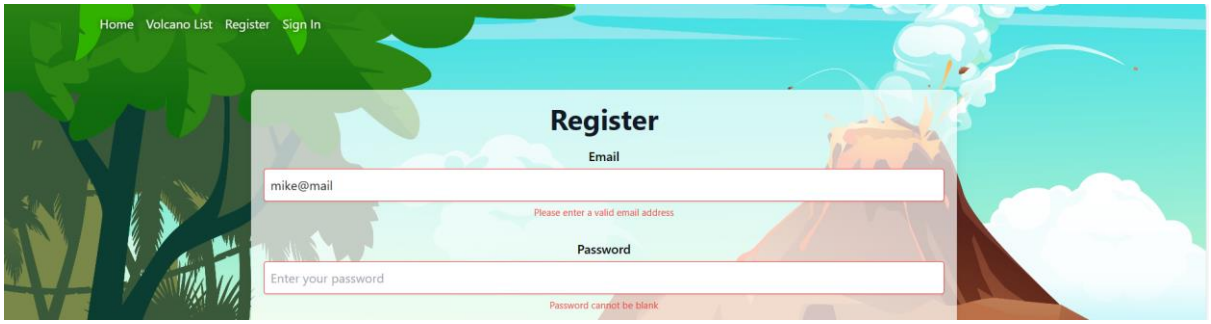
/user/register

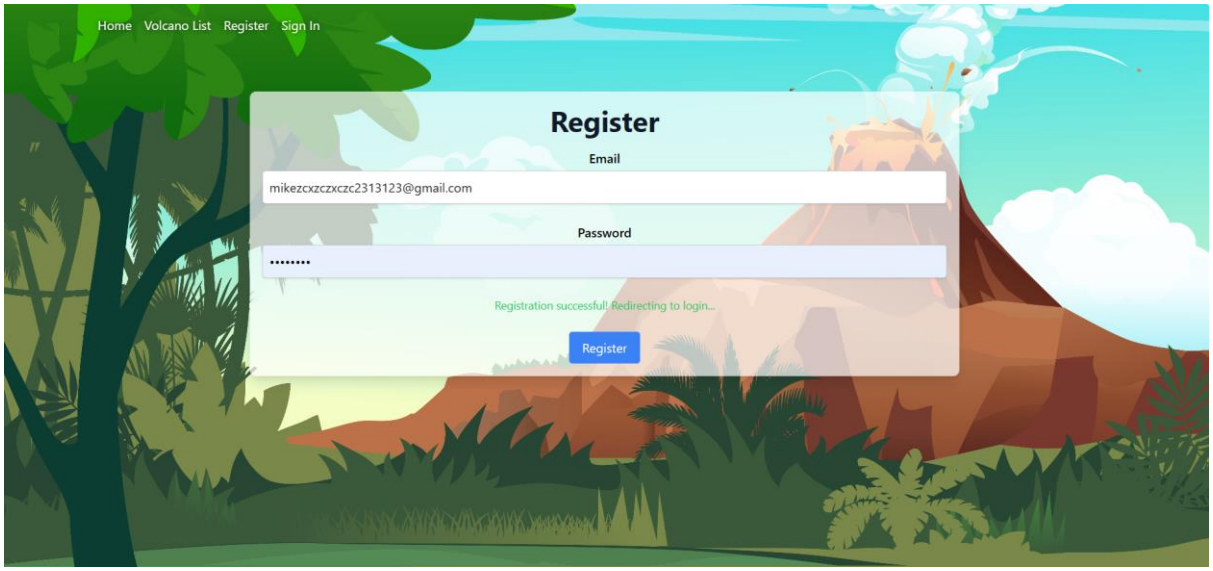
```
export const registerUser = (userData) => async (dispatch) => {
    try {
        const response = await API('user/register', 'POST', userData);
        const data = response.data;

        if (data.error) {
            dispatch({ type: 'REGISTER_FAIL', payload: data.message });
        } else {
            dispatch({ type: 'REGISTER_SUCCESS', payload: data.message });
        }
    } catch (error) {
        dispatch({
            type: 'REGISTER_FAIL',
            payload: error.response ? error.response.data.message : error.message
        });
    }
};
```

```
const handleSubmit = (e) => {
    e.preventDefault();
    dispatch(registerUser({ email, password }));
};
```







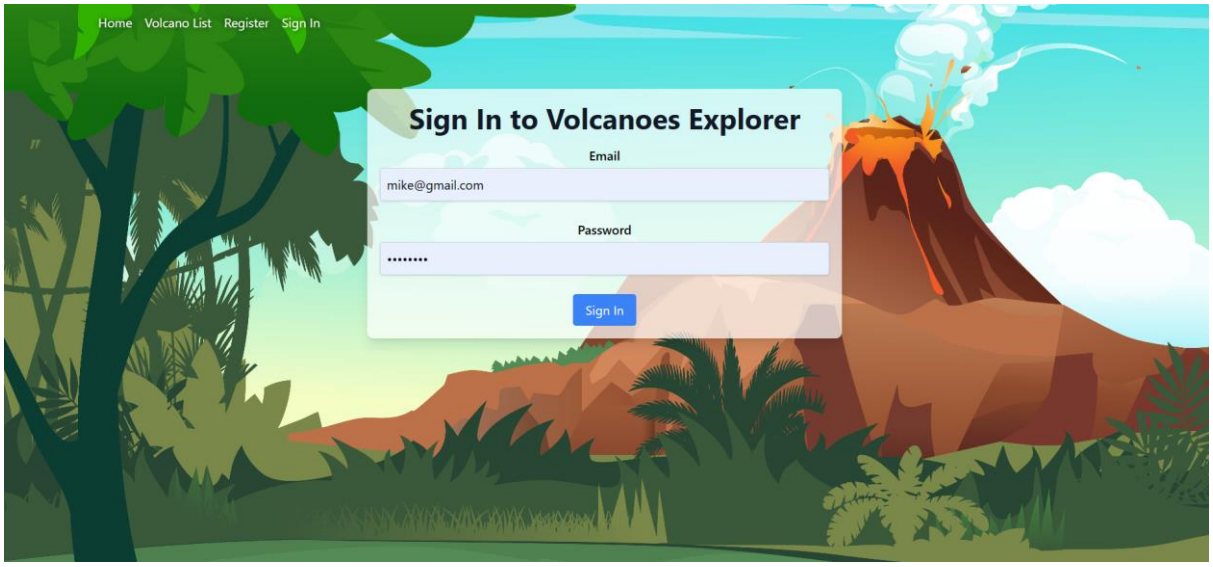
/user/login

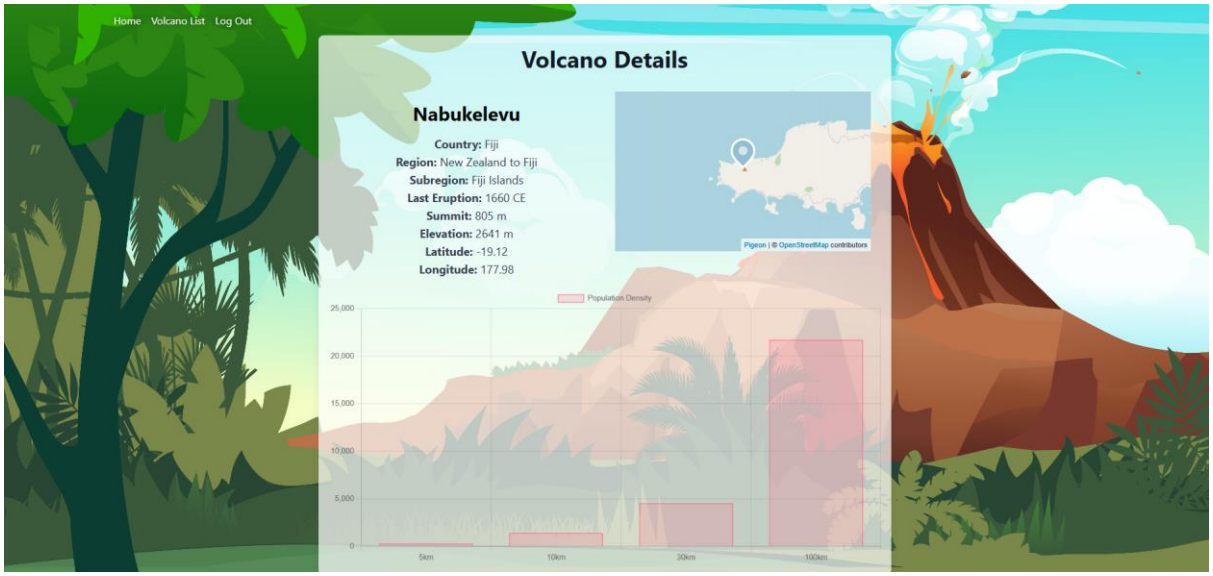
```
export const login = (email, password) => async (dispatch) => {
  try {
    const response = await API('user/login2', 'POST', { email, password });
    if (response.data && response.data.token) {
      localStorage.setItem('token', response.data.token);
      dispatch(loginUser(response.data.token));

      clearTimeout(logoutTimeout);
      const timeoutDuration = response.data.expires_in * 1000; // Convert seconds to milliseconds
      logoutTimeout = setTimeout(() => {
        console.log("Token expired. Logging out...");
        dispatch(logout());
      }, timeoutDuration);
    } else {
      throw new Error(response.data.message || "Login failed.");
    }
  } catch (error) {
    const errorMessage = error.response?.data?.message || error.message || "Login failed due to an unkno
    console.error('Login error:', errorMessage);
```

```
useEffect(() => {
    if (isAuthenticated) {
        navigate('/');
    }
}, [isAuthenticated, navigate]);

const handleLogin = (event) => {
    event.preventDefault();
    dispatch(login(email, password));
};
```





Modules Used

ag-grid-react

Module to provide fully-featured table components, including sorting and filtering.
[React Grid: Quick Start | AG Grid (ag-grid.com)]

axios

Promise based HTTP client for the browser and node.js.
[axios/axios: Promise based HTTP client for the browser and node.js (github.com)]

chart.js

Simple yet flexible JavaScript charting for designers & developers.
[Chart.js | Open source HTML5 Charts for your website (chartjs.org)]

jwt-decode

A small browser library that helps decoding JWTs token which are Base64Url encoded.
[jwt-decode - npm (npmjs.com)]

pigeon-maps

ReactJS maps without external dependencies.
[Demo | Pigeon Maps (pigeon-maps.js.org)]

react-redux

Official React bindings for Redux.
[React Redux | React Redux (react-redux.js.org)]

react-router-dom

DOM bindings for React Router.
[Home v6.23.1 | React Router]

redux-thunk

Middleware that allows you to write action creators that return a function instead of an action.
[reduxjs/redux-thunk: Thunk middleware for Redux (github.com)]

Application Design

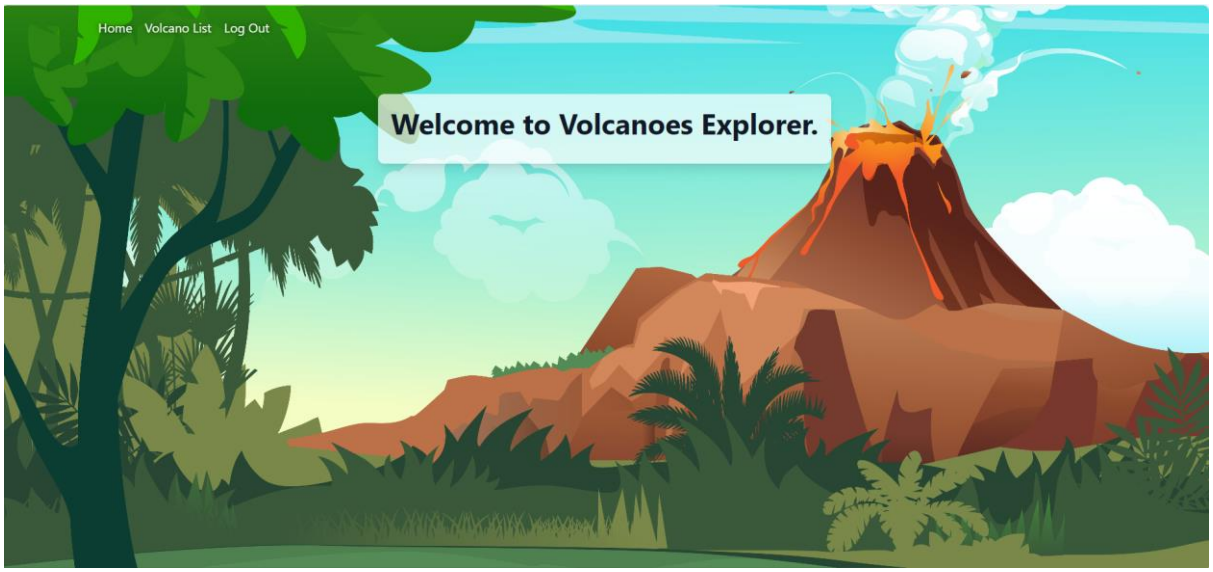
Navigation and Layout

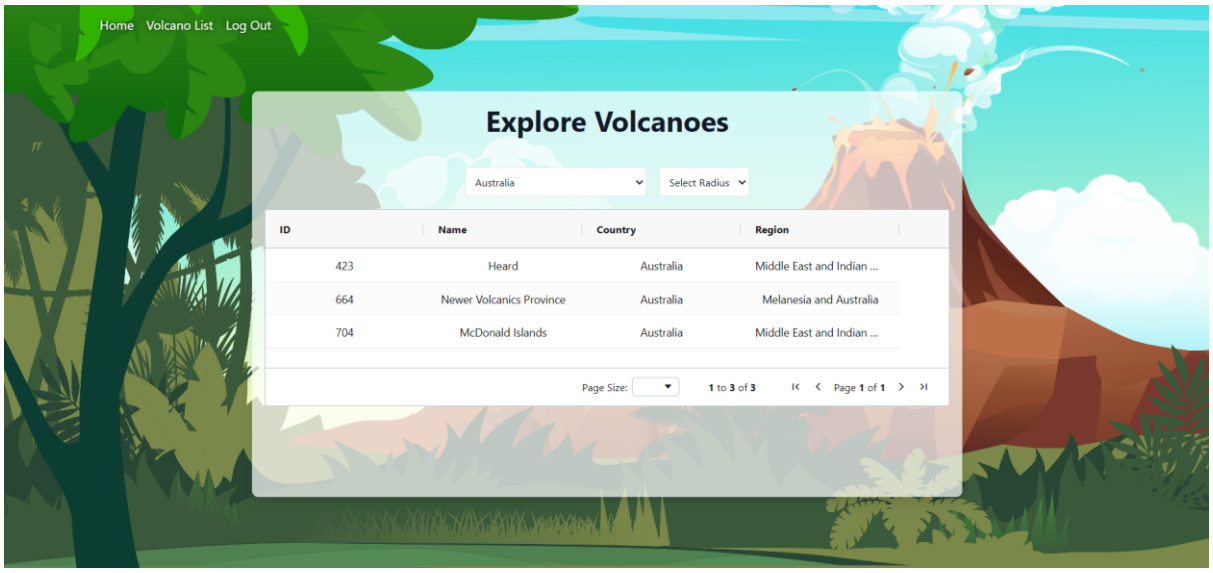
The "Volcanoes Explorer" application is structured for clarity and scalability, adhering to a modular architecture that facilitates maintenance and future expansion. The directory setup is as follows:

- axios/: Contains configurations essential for network requests:
    - API.js: Configures the Axios instance for API calls, centralizing the configuration to simplify modifications.
    - URL.js: Stores all API endpoints, ensuring that URL management is organized and easily accessible across the application.
- components/: This directory holds reusable React components, each in its own folder with its specific business logic and styling:
    - Component_Name/: A template directory for components, which includes:
    - Component_Name.jsx: The main component file.
    - module/: Contains Redux specific files like actions and reducers to manage the state related to the component.
    - action/actions.js: Defines Redux actions.
    - reducer/reducer.js: Houses the Redux reducer for the component.
- pages/: Comprises individual directories for each page of the application, encapsulating the page-specific components and logic:
    - Page_Name/: Includes the main React component (Page.jsx) that represents a distinct page within the application.
- redux/: Contains rootreducer.js, which combines all reducers from different modules to create a single root reducer, facilitating global state management across the application.

The top-level files include:

- App.js: The entry component that sets up the React Router and integrates the main layout with navigation components.
- index.js: The root JavaScript file that renders the React application and hooks it into the HTML.
- index.css: Provides global styles for the entire application.






```
const Header = () => {
    const isAuthenticated = useSelector(state => state.auth.isAuthenticated);
    const dispatch = useDispatch();
    const navigate = useNavigate();
```

```
const Home = () => {
  return (
    <div className="min-h-screen bg-fixed bg-cover bg-center" style={{ backgroundImage: `url(${heroImage})` }}>
      <Header />
      <div className="pt-16 flex flex-col items-center justify-center h-full">
        <div className="mt-12 text-center p-4 bg-white bg-opacity-70 rounded-lg shadow-lg">
          <h1 className="text-xl sm:text-4xl font-bold text-gray-900 mb-4">Welcome to Volcanoes Explorer.</h1>
        </div>
      </div>
    </div>
  );
};
```

Usability and Quality of Design

Display Organization and Layout:

- Pros: The display is well-organized with a logical flow that enhances user understanding and interaction. Tailwind CSS facilitates a clean and adaptable layout across various devices.
- Cons: Some drawbacks to note are that in sections the design might come across as a bit disorganized in information heavy parts such, as volcano specifics, which could potentially inundate users with too much data.

Navigation Clarity and Intuition:

- Pros: Navigation is constructed to be user-friendly, with logically named menus and regular positions of navigational elements, in line with the standard usability guidelines.
- Cons: For some users, the navigation between pages might be slightly confusing if not all actions are consistently highlighted or actions performed in similar ways are placed inconsistently in different viewers.

Consistency with User Expectations:

- Pros: It uses common design elements familiar to users of web applications, thus making this app readily comprehensible by to the first time user.
- Cons: The application may not have the expected interactive elements, for example, dynamic filters or clickable maps which are common in other modern web applications, this may lead to a mismatch in user expectations.

Visual Design Consistency:

- Pros: The visual design is similar across screens, using a harmonious color palette and uniform font choices which help to create a consistent user experience.
- Cons: Even though the colors and fonts are used consistently, the use of contrast and spacing to improve legibility and user's concentration may be another area to think of, specifically in the places with large amounts of information.

Usability and Design Compromises:

- Pros: The application implements the responsive design principles optimized for different types of device including desktops, tablets and mobile phones.
- Cons: The mobile responsiveness could be improved even more to enhance the touch interactions and readability on the smaller screens. Some navigation elements and data visualizations may not be scaled down as needed for mobile devices and therefore the overall user experience on such devices may not be up to the mark.

Improvements:

- Enhance the interactive elements like maps and charts to be more engaging and responsive to user inputs.
- Reevaluate layouts in information-dense areas to reduce cognitive load by introducing better spacing and segmenting of data.
- Improve mobile responsiveness by adjusting touch targets and layout shifts to ensure a smoother experience on handheld devices.

Accessibility

Provide a text equivalent for every non-text element:

- The application should ensure all images, charts, and maps have appropriate alt texts or descriptions. For example, the hero image in Home.jsx needs an alt attribute describing the image. The DetailVolcano.jsx map should have textual information about the location as an alternative to the visual map.

Ensure that all information conveyed with color is also available without color:

- In VolcanoDensityChart.jsx, the bar chart uses color to represent data. It's essential to ensure that these data representations are accessible through other means like labels or patterns to cater to users who are colorblind.

Organize documents so they may be read without style sheets:

- The documents should be organized semantically to ensure readability without CSS. This means using proper HTML elements in a logical structure. The application's use of React and JSX should prioritize semantic HTML where possible.

Ensure that text equivalents are updated when dynamic content changes:

- In VolcanoDensityChart.jsx, dynamic updates to the chart should be accompanied by updates to any text equivalents or descriptions for accessibility. Similarly, dynamic content in VolcanoTable.jsx should maintain updated aria-labels or descriptions.

Use the clearest and simplest language appropriate for a site's content:

- The text across the application should be simple and straightforward to ensure it is accessible to users with varying levels of comprehension and language proficiency.

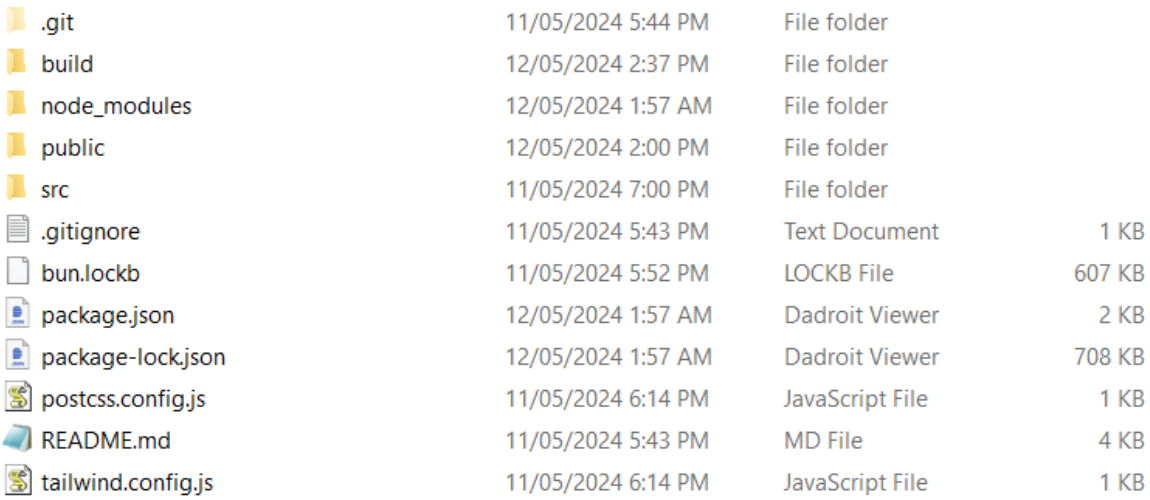
For tables, identify row and column headers:

- VolcanoTable.jsx uses AG Grid for displaying data. It is crucial to ensure that column headers are marked appropriately and distinct from data cells, possibly using <th> tags or corresponding ARIA roles.

Technical Description

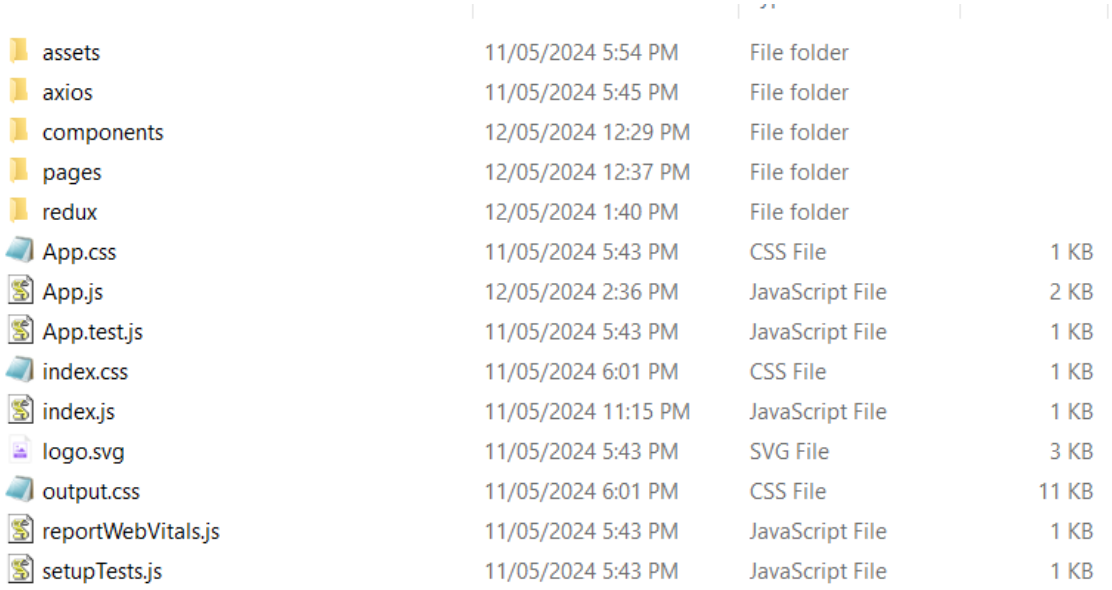
Architecture

The application is organized using a clean and efficient directory structure tailored for a React project, designed to enhance maintainability and scalability. It contains a public/ directory for the assets, a src/ folder with all the components and the source code stylized by Tailwind CSS for a clean and responsive layout and all the other configuration files including the package. json for managing dependencies. The project uses Bun as the JavaScript runtime and package manager, which is known for its excellent performance and compatibility, thus, the project is able to build quickly and manage dependencies efficiently. Regardless of the structured setup with tools like Bun and Tailwind CSS, leads to a strong infrastructure that is easy to use and update.



.git	11/05/2024 5:44 PM	File folder	
build	12/05/2024 2:37 PM	File folder	
node_modules	12/05/2024 1:57 AM	File folder	
public	12/05/2024 2:00 PM	File folder	
src	11/05/2024 7:00 PM	File folder	
.gitignore	11/05/2024 5:43 PM	Text Document	1 KB
bun.lockb	11/05/2024 5:52 PM	LOCKB File	607 KB
package.json	12/05/2024 1:57 AM	Dadroit Viewer	2 KB
package-lock.json	12/05/2024 1:57 AM	Dadroit Viewer	708 KB
postcss.config.js	11/05/2024 6:14 PM	JavaScript File	1 KB
README.md	11/05/2024 5:43 PM	MD File	4 KB
tailwind.config.js	11/05/2024 6:14 PM	JavaScript File	1 KB

The application uses a organized source code structure to improve how easy it is to maintain and scale; static files, like images and styles are stored in the assets/ folder for organization reusable user interface elements are located in components/ page specific elements in pages/ help, with navigating through the app and all files related to managing state with Redux are grouped together in redux/. This setup makes it clear who does what during development makes updates easier and helps the app grow efficiently.



assets	11/05/2024 5:54 PM	File folder	
axios	11/05/2024 5:45 PM	File folder	
components	12/05/2024 12:29 PM	File folder	
pages	12/05/2024 12:37 PM	File folder	
redux	12/05/2024 1:40 PM	File folder	
App.css	11/05/2024 5:43 PM	CSS File	1 KB
App.js	12/05/2024 2:36 PM	JavaScript File	2 KB
App.test.js	11/05/2024 5:43 PM	JavaScript File	1 KB
index.css	11/05/2024 6:01 PM	CSS File	1 KB
index.js	11/05/2024 11:15 PM	JavaScript File	1 KB
logo.svg	11/05/2024 5:43 PM	SVG File	3 KB
output.css	11/05/2024 6:01 PM	CSS File	11 KB
reportWebVitals.js	11/05/2024 5:43 PM	JavaScript File	1 KB
setupTests.js	11/05/2024 5:43 PM	JavaScript File	1 KB

Test plan

Appendix A:

Positive outcome cases:

Task	Description	Expected Outcome	Result	Screenshot (Appendix B)
Retrieve Countries	Trigger fetch from countries endpoint.	List of countries retrieved and displayed correctly.	PASS	01

Retrieve Population with Country	Select a country and trigger population data retrieval.	Population data relevant to selected country is displayed accurately on the interface.	PASS	02
Showing Map	Select a volcano to view detailed information.	Map centers on selected volcano with marker displayed correctly.	PASS	03
Showing Chart (Authorized User)	Select a volcano to view population density chart.	Chart displays population density data around the selected volcano accurately.	PASS	04
Registration	Attempt to register with valid details.	User is registered successfully and redirected to login page.	PASS	05
Login	Attempt to login with registered user credentials.	User is logged in successfully and session is started.	PASS	06
Log Out	User clicks log out button.	User is logged out and session is ended.	PASS	07
Handle Token Expired	Simulate expired token during session.	User is prompted to log out and the token will be removed from localStorage	PASS	08

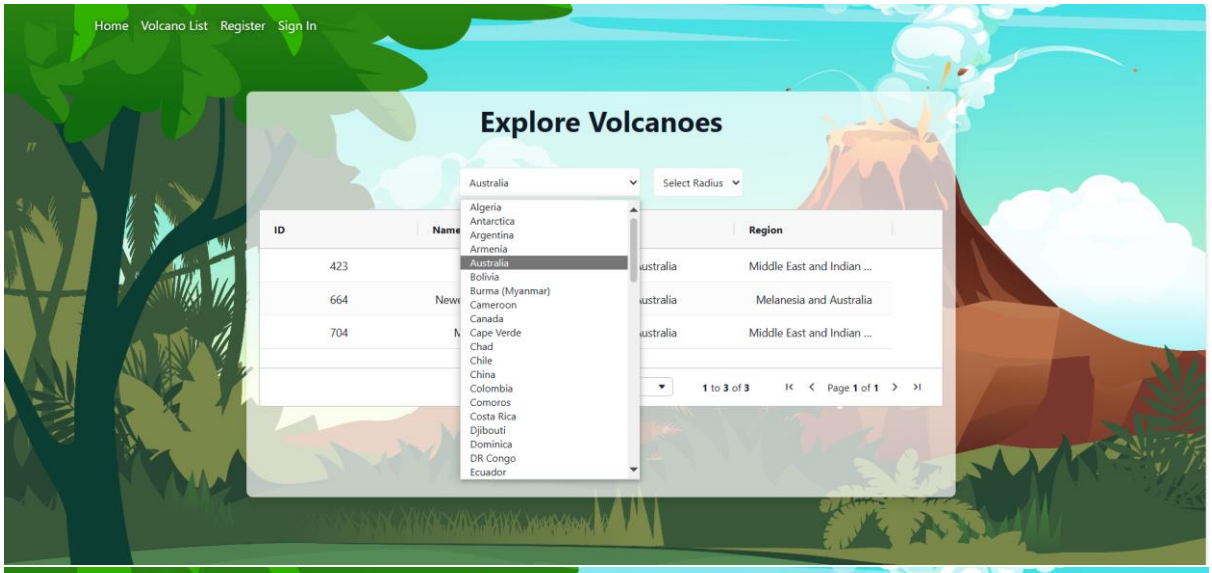
Negative Outcome Cases (Error Scenarios)

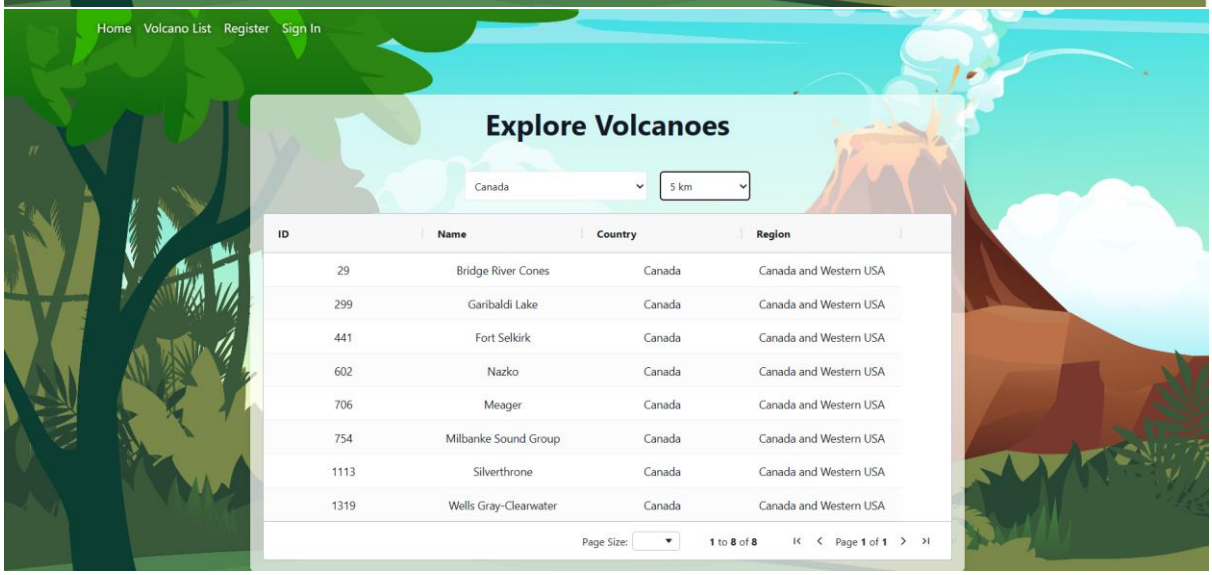
Task	Description	Expected Outcome	Result	Screenshot (Appendix B)
Registration (Error)	Attempt to register with already used email.	Error message about existing email shown, registration not completed.	PASS	09
Login (Error)	Attempt to login with incorrect credentials.	Error message displayed, user not logged in.	PASS	10
Handle Token Expired (Error)	Ignore token expiration and continue session.	Security breach prevented, user forcibly logged out and redirected to login.	PASS	11

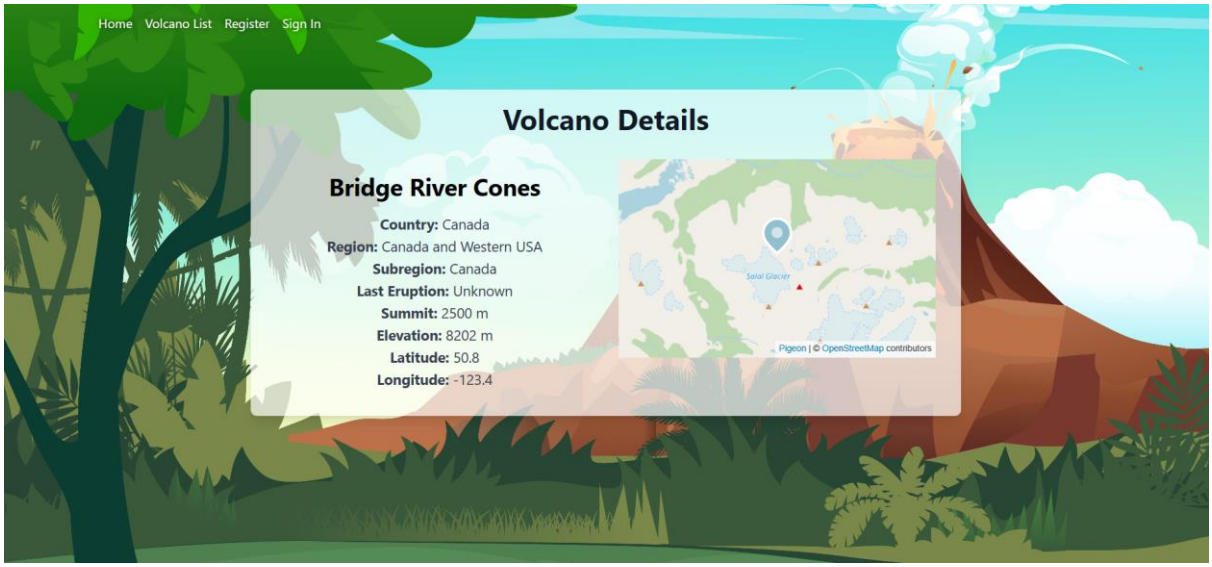
Edge Cases

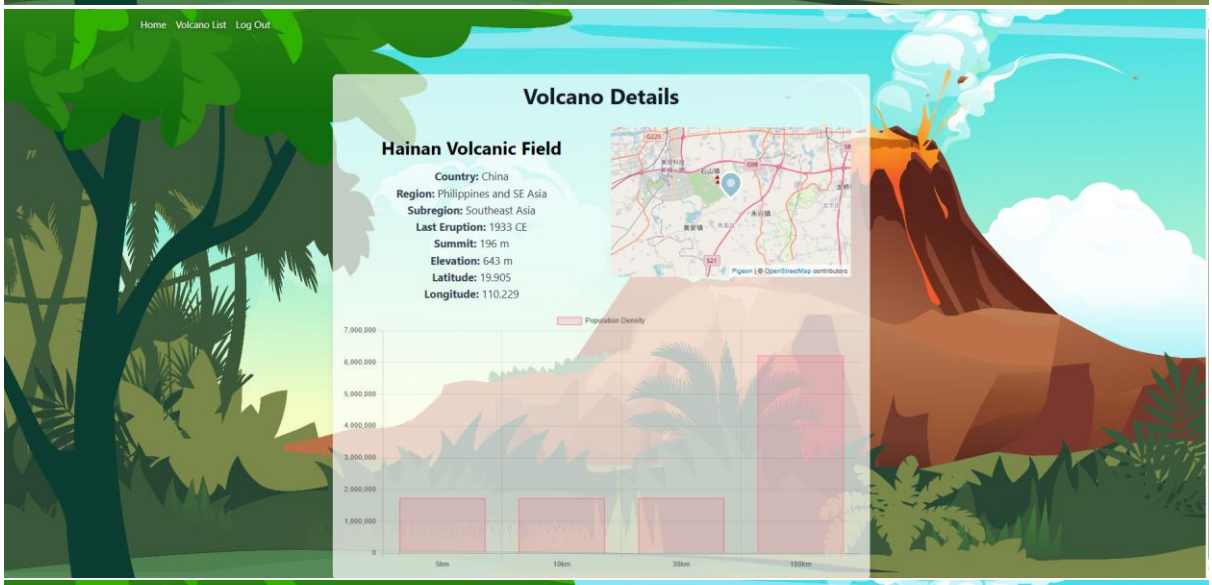
Task	Description	Expected Outcome	Result	Screenshot (Appendix B)
Login (Edge)	Attempt to login with boundary values for password.	Proper validation occurs, either logging in or showing error based on correctness.	PASS	12

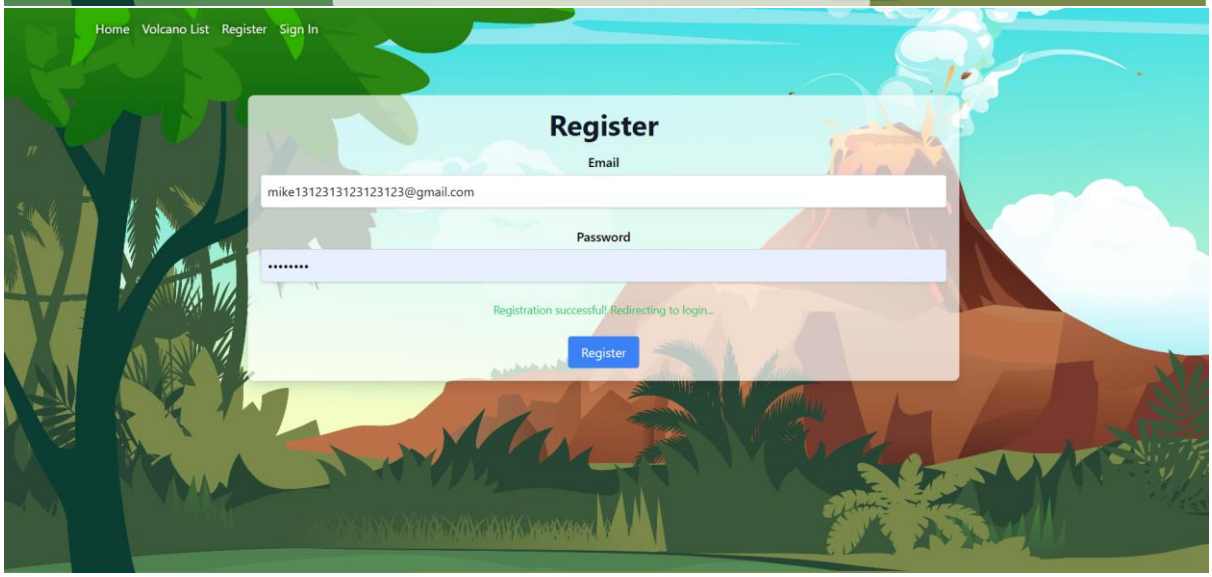
Appendix B

1. 

2. 

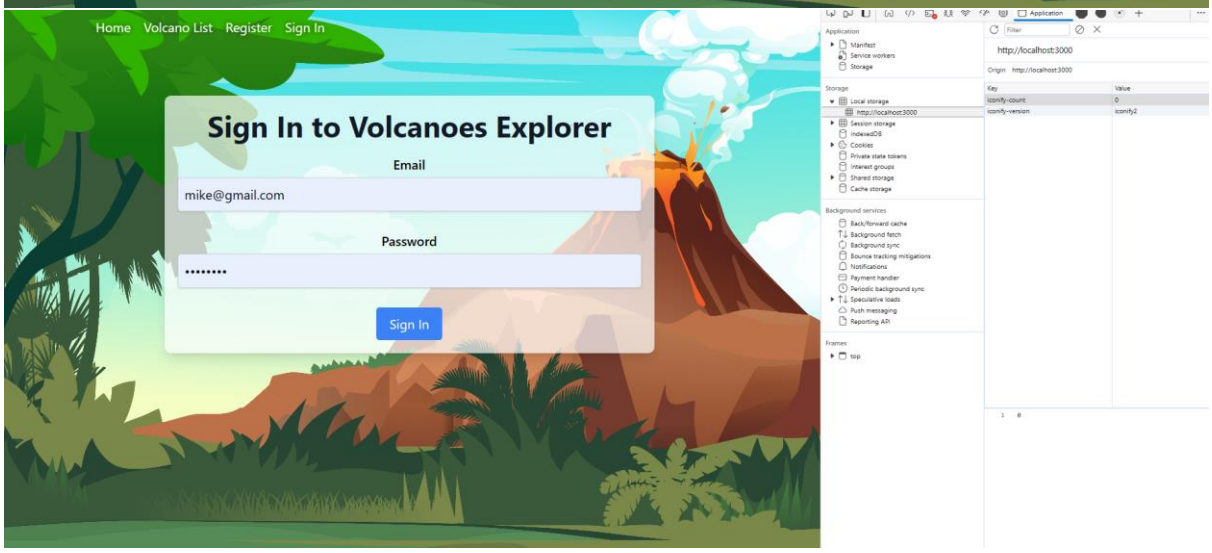
3. 
Home Volcano List Register Sign In
Volcano Details
Bridge River Cones
Country: Canada
Region: Canada and Western USA
Subregion: Canada
Last Eruption: Unknown
Summit: 2500 m
Elevation: 8202 m
Latitude: 50.8
Longitude: -123.4
Pigeon | © OpenStreetMap contributors

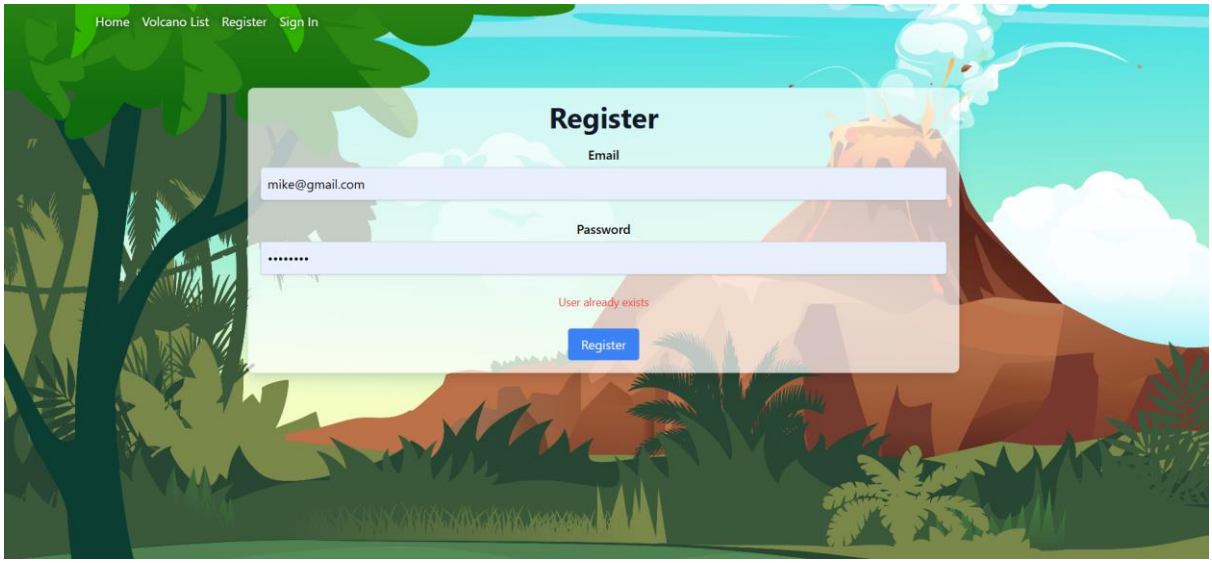
4. 
Home Volcano List Log Out
Volcano Details
Hainan Volcanic Field
Country: China
Region: Philippines and SE Asia
Subregion: Southeast Asia
Last Eruption: 1933 CE
Summit: 196 m
Elevation: 643 m
Latitude: 19.905
Longitude: 110.229
Pigeon | © OpenStreetMap contributors
Population Density
7,000,000
6,000,000
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
5km
10km
30km
100km

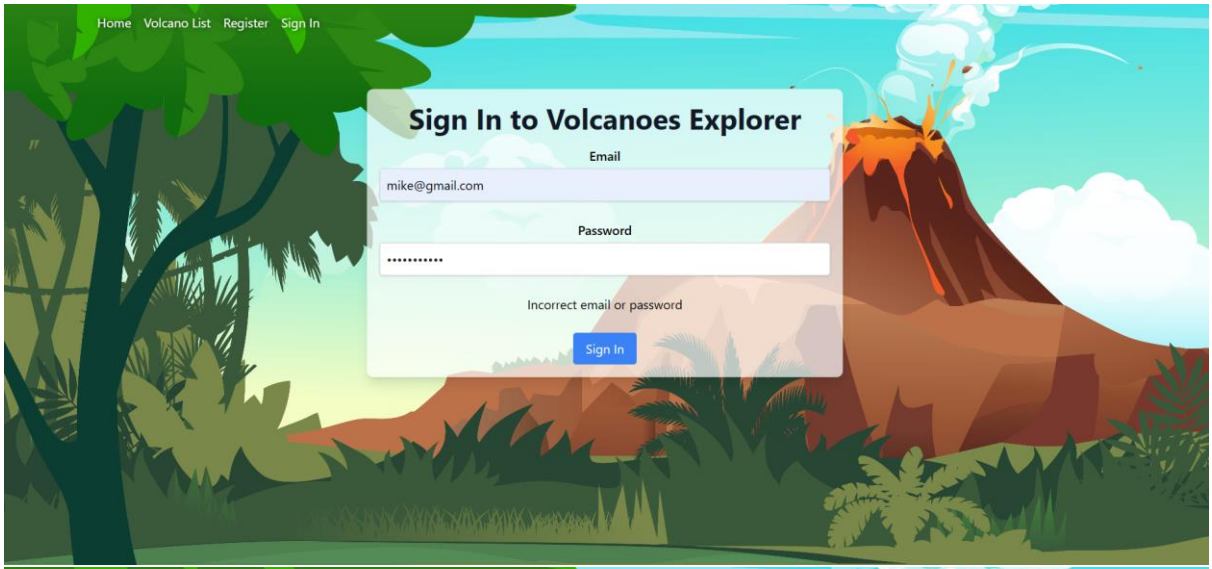
5. 
Home Volcano List Register Sign In
Register
Email
mike1312313123123123@gmail.com
Password
••••••••
Registration successful! Redirecting to login...
Register

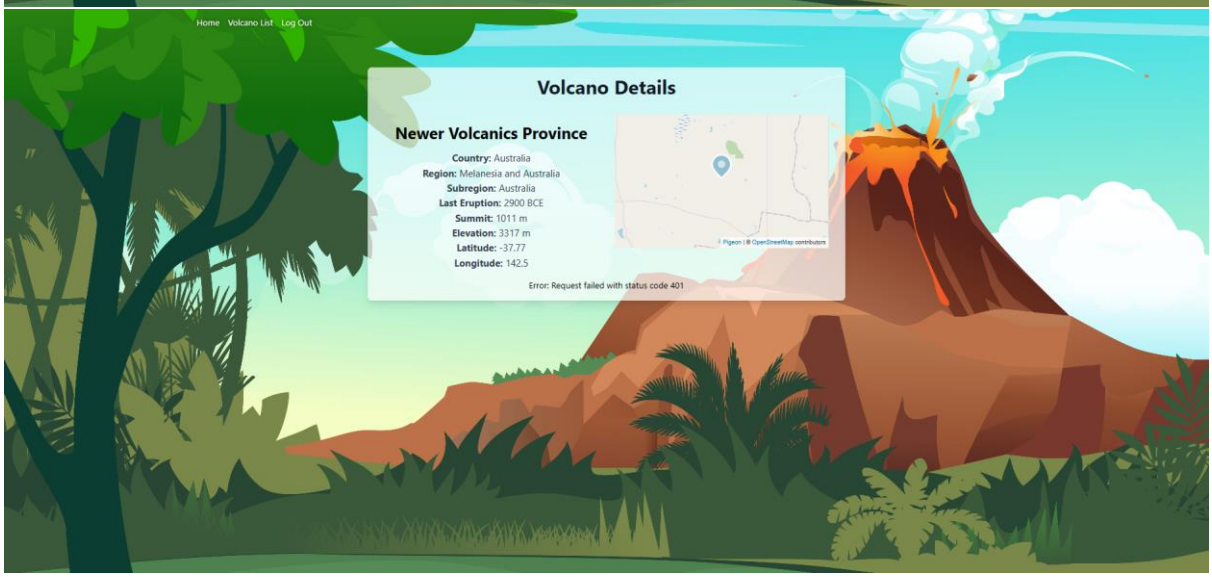
6. 
Home Volcano List Log Out
Welcome to Volcanoes Explorer.

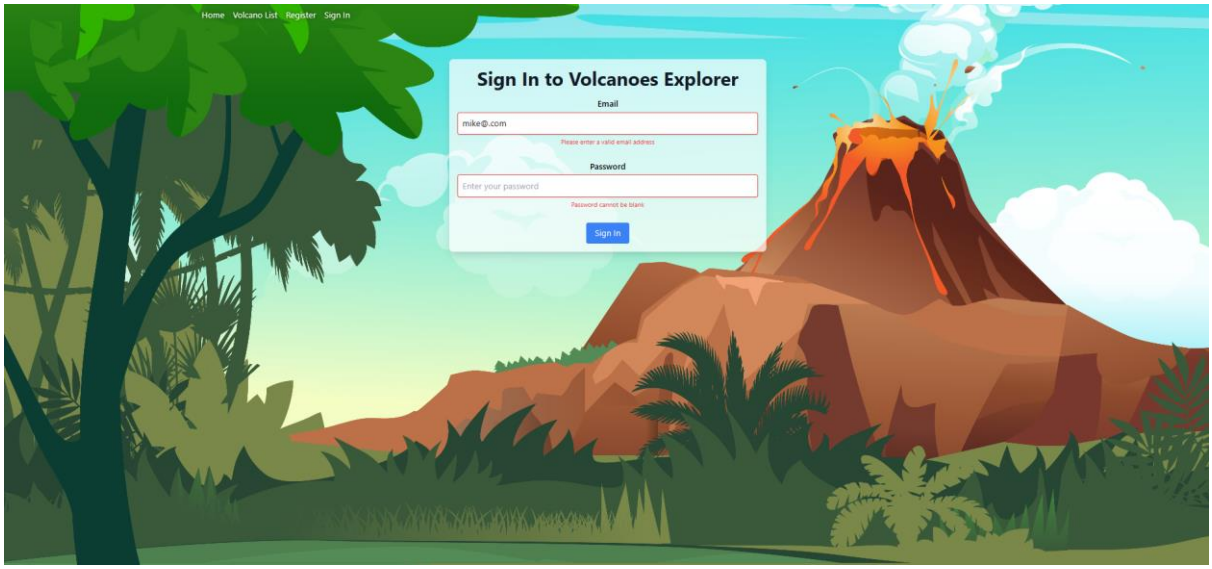
7. 
Home Volcano List Register Sign In
Welcome to Volcanoes Explorer.

8. 
Home Volcano List Register Sign In
Sign In to Volcanoes Explorer
Email
mike@gmail.com
Password
Sign In

9. 
Home Volcano List Register Sign In
Register
Email
mike@gmail.com
Password
User already exists
Register

10. 
Home Volcano List Register Sign In
Sign In to Volcanoes Explorer
Email
mike@gmail.com
Password
Incorrect email or password
Sign In

11. 
Home Volcano List Log Out
Volcano Details
Newer Volcanics Province
Country: Australia
Region: Melanesia and Australia
Subregion: Australia
Last Eruption: 2900 BCE
Summit: 1011 m
Elevation: 3317 m
Latitude: -37.77
Longitude: 142.5
Error: Request failed with status code 401

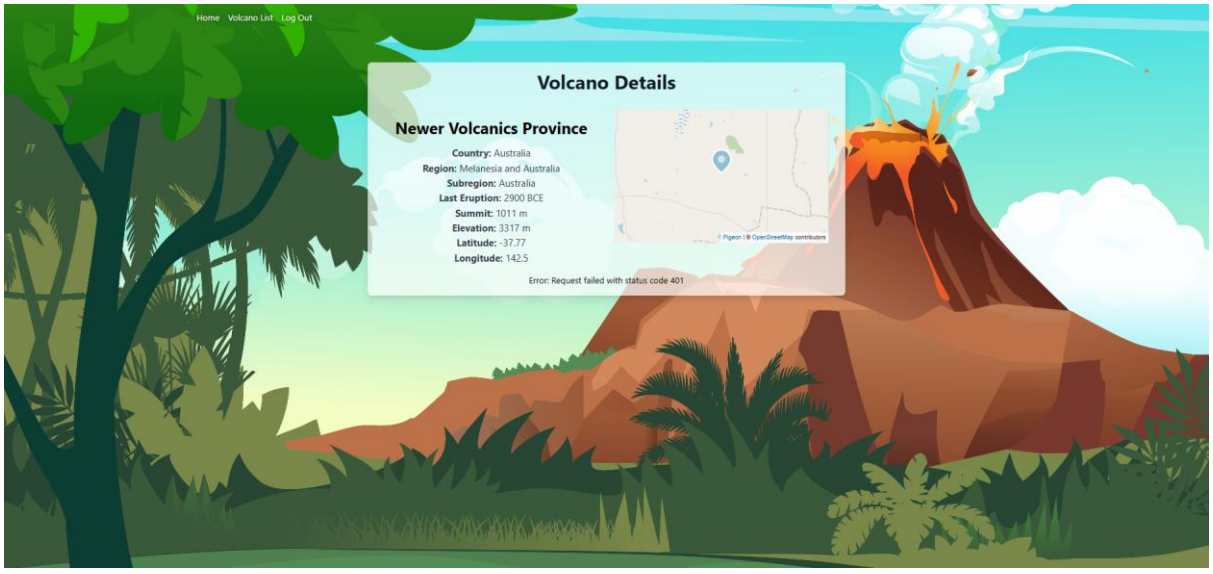
12. 

Difficulties / Exclusions / unresolved & persistent errors /

Handling the authentication tokens presented a major challenge, particularly in maintaining session integrity during manual browser refreshes:

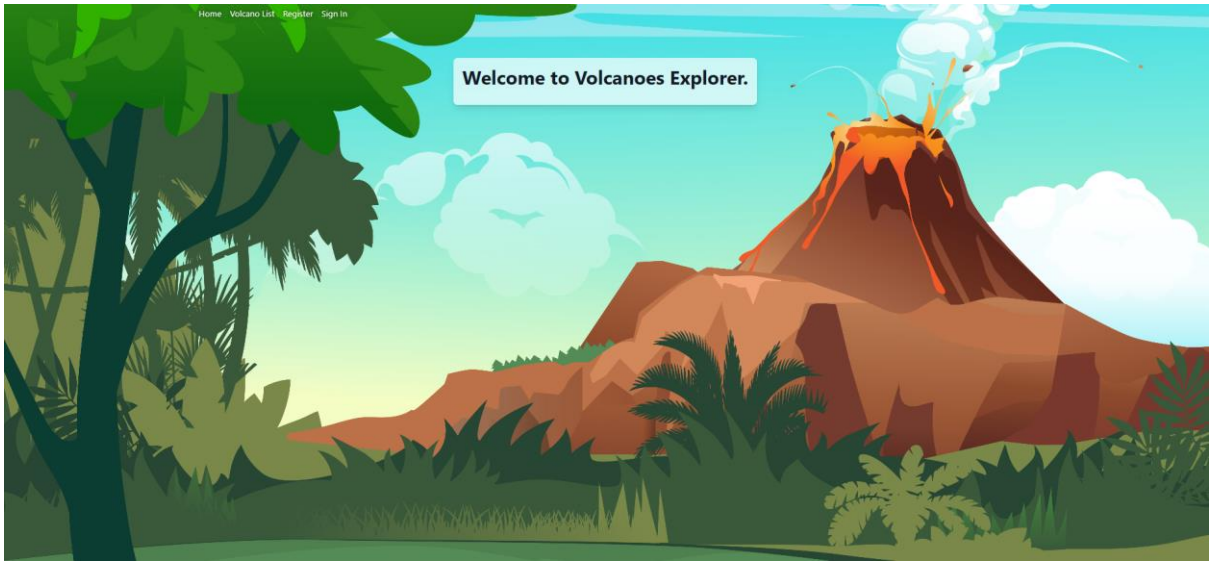
- Major Roadblock: The challenge was ensuring that authentication tokens (JWTs) were effectively managed.
- Technical Issue: The application faced difficulties with tokens not being immediately invalidated upon manual refresh, potentially allowing brief unauthorized access.

This token management issue remains partially unresolved.

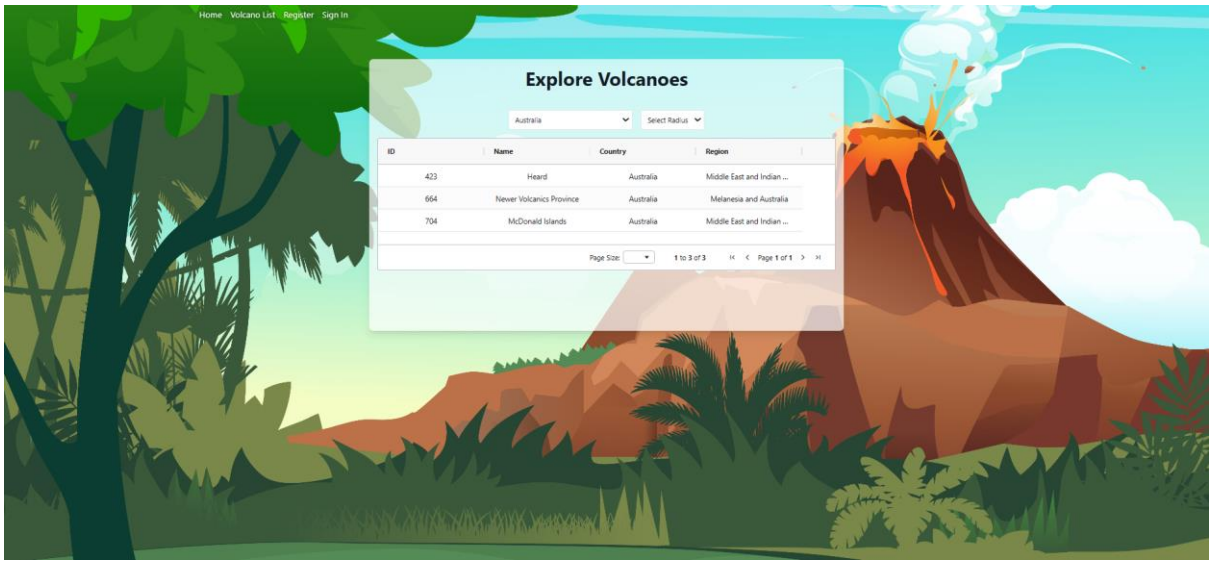


User guide

Home page:

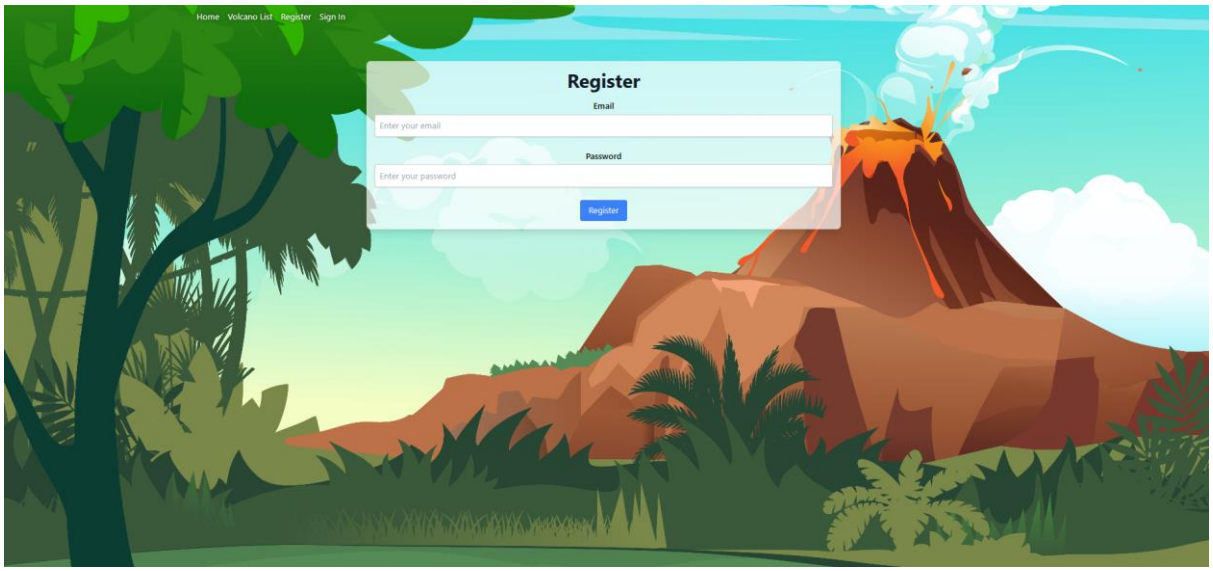


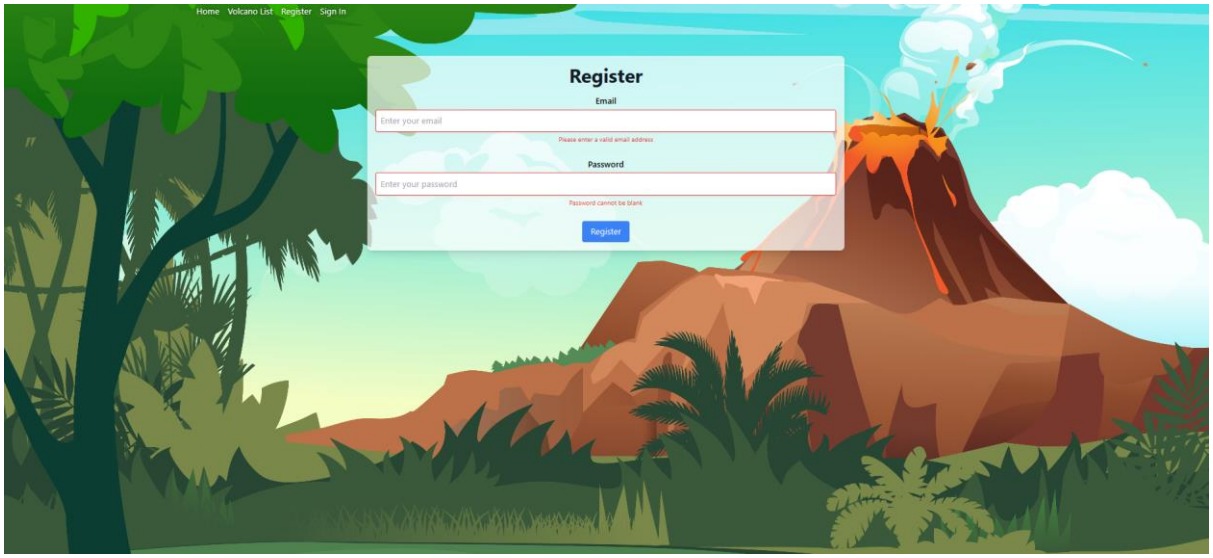
View general information from volcano table:



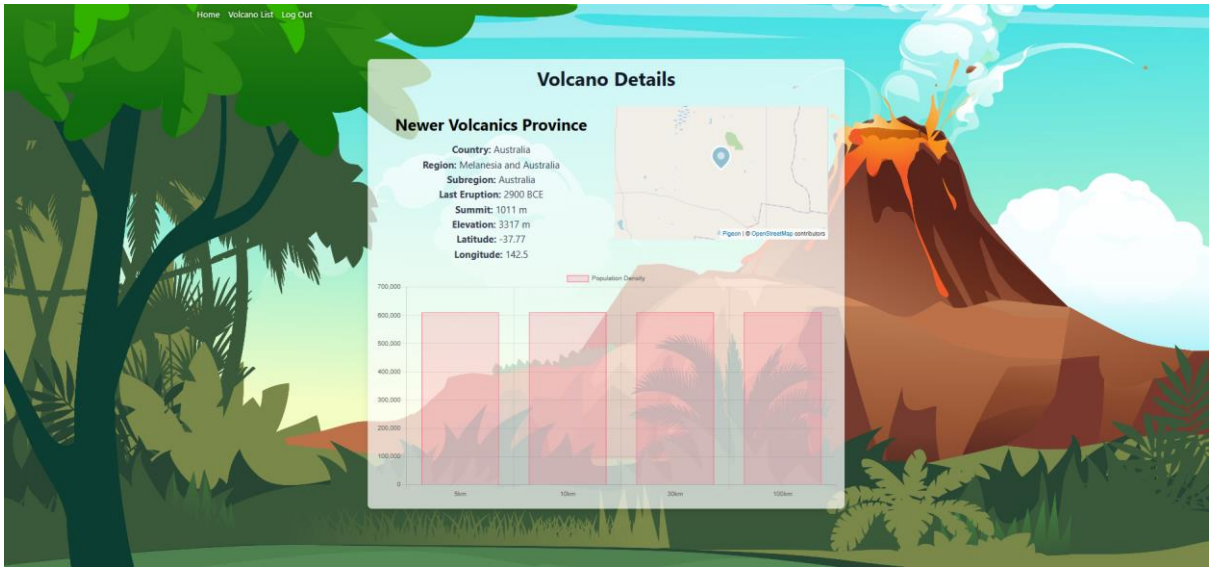
Click on the row to see details including the map and marker:

Registration process is required to be completed the form with valid email address and valid password:





Log in to view the details of volcano including the chart of population density:



References

Jarred Sumner. (2024). Bun - A fast all-in-one JavaScript runtime.

https://bun.sh

Adam Wathan et al. (2024). Tailwind CSS.

https://tailwindcss.com

Matt Zabriskie. (2021). Axios - Promise based HTTP client for the browser and node.js.
https://github.com/axios/axios

AG-Grid. (2024). React Grid: Quick Start. AG Grid.

https://www.ag-grid.com/react-grid/

Chart.js Contributors. (2024). Chart.js | Open source HTML5 Charts for your website.
https://www.chartjs.org

Auth0. (2024). jwt-decode - npm.

https://www.npmjs.com/package/jwt-decode

Pigeon Maps. (2024). Demo | Pigeon Maps.

https://pigeon-maps.js.org

Redux. (2024). React Redux.

https://react-redux.js.org

React Router. (2024). Home v6.23.1.

https://reactrouter.com

Redux. (2024). reduxjs/redux-thunk: Thunk middleware for Redux.

https://github.com/reduxjs/redux-thunk